Tampere University Unit of Computing Sciences
COMP.SE.610 Software Engineering Project 1
COMP.SE.620 Software Engineering Project 2

Student-ICT-PROJ “Get A Room!”, Vincit / 1

Get a Room!

Final Report

Jere Koski
Joona Prehti
Joonas Hiltunen
Martti Grönholm
Mikko Pirhonen
Sara Brentini
Vivian Lunnikivi

Version history

Version	Date	Author	Description
0.1	6.12.2021	Vivian Lunnikivi	Copy template, fix layout, and add team details.
0.2	11.12.2021	Vivian Lunnikivi	Write chapters 1 and 2, sketch section 3.2.
0.3	12.12.2021	Vivian Lunnikivi	Write sections 3.1 and 3.3.
0.4	13.12.2021	Vivian Lunnikivi	Add content to chapters 8, 9, and 10.
0.6	14.12.2021	Vivian Lunnikivi	Write sections 3.7, 5.1, chapters 6, 9 and 10.
0.7	15.12.2021	Joona Prehti	Write my part to sections 1.2, 3.2, 4.1 and 8.3
0.8	15.12.2021	Sara Brentini	Added text to 3.2, 5.3, 7.1, 7.2 and 8.3
0.9	15.12.2021	Joonas Hiltunen	Added text to 4.1 and 7.1
1.0	15.12.2021	Mikko Pirhonen	Add to 4.1
1.1	16.12.2021	Vivian Lunnikivi	Update figures, fix some todos.
1.2	17.12.2021	Joonas Hiltunen	Add text to 5.3

Table of contents

1 Introduction

1.1 Purpose of the report

This document reports the main outcomes of *Get a Room!* software project – a student project, offered for implementation on the Tampere University courses *COMP.SE.610* and *COMP.SE.620 Software Engineering project 1 & 2* by the technology company Vincit in the fall 2021. The project group consisted of seven students of Information Technology and was supported by a Product owner (PO), Agile coach, Tech coach and an IT Support person from the customer end.

Before the project, everyday meeting room reservations at Vincit were made via Google Calendar, by reserving calendar resources hosted in Vincit's deployment of Google workspace. Google Calendar had proven to be unnecessarily laborious to use, especially on mobile devices and for ad hoc reservations. Consequently, Vincit requested for a cross-functional team with web, cloud, and service-modelling skills to create a highly usable, mobile-friendly application that allows Vincit employees to easily make meeting room reservations on the fly.

To this end, the group designed and implemented a progressive web application (PWA) that integrates to the company's cloud environment, allowing users to log in with their pre-existing Google ID credentials. The application makes ad hoc reservations quick and easy, as employees can use both mobile and desktop devices to see available rooms in their preferred office location and make reservations with just a few clicks. As usability was considered a focus point in the project, the group also conducted a formal user experience (UX) testing session and used the received end user feedback to revise the application's UX.

The project was successful in creating a stable, highly usable MVP and handing over the project under customer maintenance. By the end of the project, the application was integrated with the company infra, and was taken into use at Vincit.

1.2 Product and environment

The product is called *Get a Room!* and it is targeted for Vincit employees for making ad hoc room reservations. "Ad hoc" refers to situations when an acute need for a room arises, e.g., when employees find themselves brainstorming during a coffee break, and then would wish to take the brainstorming session to a whiteboard. While *Get a Room!* has been designed primarily to serve this use case, it can be extended for reserving other resources in the future.

Office rooms from various Vincit office locations, and other reservable company resources such as cars and pop-up workspaces, are recorded in Vincit's Google Workspace instance, where they can be reserved by logged in users.

Get a room! Utilizes Vincit SSO that allows the app to delegate authentication issues to Vincit's own authorization server. Thus, users do not have to create new credentials to get started with the system. Instead, they can merely navigate to *getaroom.vincit.com* with a browser and log in with their existing credentials by the standard Google SSO method.

Most Vincit employees use mobile and desktop devices running either iOS with Safari, or Android with Chrome. Therefore, the goal was to provide support for at least these platforms. Full support was achieved by implementing the app as an PWA, which runs on

browsers, while utilizing native implementations. As an additional benefit, most other common platforms are automatically supported as well, even though official support is not promised.

The cloud environment is modelled in Figure 1.1. This same figure models both the staging and production environments, they are exact copies of each other. We wanted to make staging and production exact copies to minimize the risk for production problems. Due to security reasons, we don't have any admin credentials to the production Google Cloud environment, so checking logs and debugging problems there would be slow and time consuming.

The most important part of the cloud architecture is the Google Cloud Platform which consists of 2 Google Cloud Run services. Cloud Run services are serverless, container-based and fully scalable hosting environments. First Cloud Run service runs Nginx server and serves our React application as a static file. This Nginx service also redirects all traffic coming to path /api to the Node.js backend Cloud Run service. The Node.js Cloud Run is the lower one in the figure.

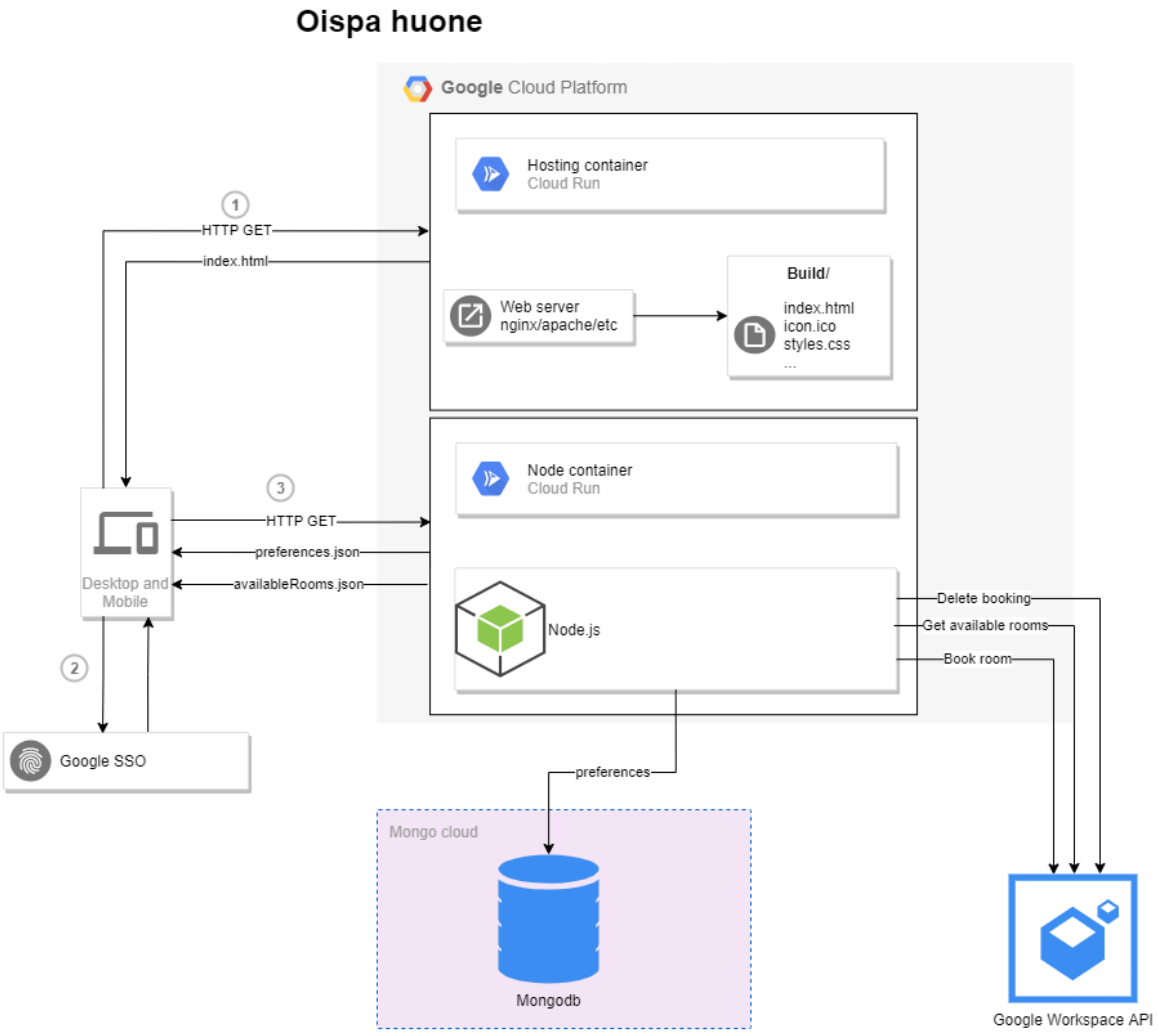

Figure 1.1: *Get a room!* high level concrete architecture.

Projects deployment also uses Google Cloud Build API which is a service to build docker images on the cloud. The Cloud Build API is used through GitHub Actions to build up the

images which are then deployed to the Cloud Run Services. Deployment process is explained more thoroughly in file .github\workflows\ci.yml which can be found inside our GitHub repository.

The Node.js Cloud Run service connects to the Atlas MongoDB hosting service where users' preference data is saved. Preference data includes information about the users' favourite office location. Additionally, users Google authentication refresh token is saved in the MongoDB. Refresh token is a token which can be used to keep the user logged in the application more than one hour.

Node.js server has access to the Google Workspace where it can fetch room and building information and create and delete bookings. Node.js server is also able to add time to bookings and fetch rooms current reservations etc.

1.3 Definitions, abbreviations, and acronyms

CI/CD	*Continuous Integration and Continuous Development*. School of software engineering underlining the role of automation and testing.
CI/CD pipeline	An automated system running tests and deploying changes in a continuous matter, e.g., on every push to master branch.
IT	*Information technology*. Field of technology studying computer systems.
MMT	*Metrics Management Tool*. A logging tool utilized in the project.
MoSCoW	Requirement prioritization framework that categorizes features-to-implement in the product into must haves, should haves, could haves and won't haves.
MVP	*Minimum viable product*. The minimum implementation which can be accepted as a whole product. Contains only and all necessary features for the product.
PO	*Product owner*. Scrum role for person with the best vision and final authority over the product specifications.
PWA	*Progressive web application*, a type of web application that runs on browser, but utilizes native implementations of the features of the platform the app is running on.
Scrum	Agile software engineering framework.
UI	*User interface*. Describes the technical entity that allows an end user to interact with the developed system. Practically, this refers to what the user sees about the system on their device.
UX	*User experience*, Field of science that researches how it feels to use products. In the context of software projects, UX design aims to ensure business success by guiding the product implementation to deliver good experience of using the product.

Visdom	ITEA 4 research project for DevOps project visualizations. *Get a Room!* project participated in the study.

2 Project organisation

2.1 Group

Jere Koski, Junior Developer	Frontend development
Joona Prehti, Architect, Tech Specialist	Architecture, CI/CD, backend, and cloud development
Joonas Hiltunen, Junior Developer	Backend and cloud development
Martti Grönholm, UI Specialist	UI design and frontend development
Mikko Pirhonen, CI/CD Specialist	CI/CD and full-stack development
Sara Brentini, UX Specialist	Requirements, UX, and frontend development
Vivian Lunnikivi, Project Manager	Project management

2.2 Customer

Ari Metsähalme Tech Director	Tech Coach
Ari Kontiainen Tech Coach	Product Owner
Jonne Heiskanen IT Support	IT Support
Veli-Pekka Eloranta Cell Lead	Agile Coach

Special thanks also to the four anonymous Vincit employees that gave us their time and thoughts at the usability testing session. Your feedback was most welcome and helped us design and implement a truly working and likeable product. Our only regret is the little time we had to implement fixes!

2.3 Other stakeholders

Timo Poranen, University lecturer	Team coach
Outi Sievi-Korte	Researcher, observed sprint review meetings 2–4 as a part of Visdom project the project organization agreed to participate in.

3 Project implementation

3.1 Communication

The most prevalent communication channel turned out to be Telegram, where the team had three group chats: one for the whole group, one for UI/UX wizards, and for backend developers. Group chats and private messages were used continuously throughout the project. Continuous communication with the client was achieved via Slack, which turned out pertinent a channel towards the end of the project.

The take-away from the mere volume of messages passed through instant messaging services was, that there is a lot to discuss in a software project, and not just among the development team, but with the customer as well. It took us a while to get comfortable with sharing ideas and questions with the client, but taking the effort was worth it, and helped us considerably in building a better product.

Email was used for sending video conference invitations and keeping in touch with the team coach and Visdom researcher. Phone calls weren't needed after the first contact with the client. Google meet platform was used for biweekly online meetings with the client, aka. sprint reviews, and Teams meetings were used for group's inner remote meetings such as weekly meetings, sprint planning sessions, workshops, and debugging meetings.

On a more formal note, the team held weekly meetings on every Monday to give development team members an update on what has been done, what will happen next and what are the current open issues. Weeklies helped the team stay organized, focus efforts on most relevant issues, and stay in sync with each other. In addition to weeklies, the team held regular meetings that followed the two-week increment schedule.

Sprint reviews were the only regular meeting targeted at the whole project organization, and these were held in the beginning of each of the 7 sprints. In sprint review meetings the team presented sprint results to the customer, got feedback from the customer, and discussed the contents of the next iteration. Around mid-project the team figured out that there was a need for separate demo planning meets prior to sprint reviews to figure out how to present the results of the product increment, and demo planning sessions were held in the end of each sprint.

Another sprint scheduled meeting amongst development team was sprint retrospectives and sprint planning sessions, which were held around sprint reviews. In retrospective, the team evaluated team performance during the ending sprint, and in sprint planning, the contents for the upcoming sprint were decided. The timing of sprint planning swayed between pre-review and post-review, since we had to know a bit of the team members' schedules before the review meet but could not decide the exact contents before consulting with the client.

Sharing documents such as meeting notes, project deliverables, design documents and further reading link bank, was done via a OneDrive folder shared among the group members, holding project documents in one place. For publishing select documents for the customer, we created and shared a folder there for customers as well.

The requirements, requirements prioritization, and implementation work was managed with a Kanban board, built in Trello. All participants, including the project group members, the customer representatives, and the coach, had access to the board, making project progress and direction transparent among all the involved parties.

MMT, aka. *Metrics Management Tool* was used to log work hours, weekly reporting and monitoring hourly work allocations, project progress, and risks. MMT turned out to be useful especially with keeping track of the hours spent by team members, which in turn, helped us share work more evenly with the team. It is also nice to have real data of how much work a project of our extent requires – it helps with evaluating needed time for new projects.

3.2 Tools and technologies

While the pre-existing systems of Vincit, namely, Vincit Google Workspace, Vincit SSO and user devices dictated parts of the system implementation, the implementation technologies were mostly freely selected by the group. The reasoning for these parts, including user and room management as well as supported platforms, are documented in section *1.2: Product and environment*.

Get a Room! client application is a PWA-compatible React app that runs on a browser, user's mobile or desktop device, with a Node.js backend. Current supported browsers are Mac/iOS Safari and Windows/Android Chrome, since these are the most common environments used at Vincit. As the app is a PWA, it can be installed on users' mobile devices home screen which makes it behave very much like a native mobile application.

React was chosen as the frontend framework because its good compatibility with PWAs and because the team was somewhat familiar with it. Additionally React is one of the most used JavaScript frameworks so the internet is full of great libraries and tutorials on how to use React. React is also a popular framework at Vincit so choosing React will cut maintaining costs.

With the limited time resources we had, we had to save time when implementing the mobile application. PWA was an easy and fast way to create a mobile application from web application. Mostly everything went smoothly with the PWA, and it worked on Android and iOS. PWA turned out to be the right choice because we wouldn't ever have the time to implement a real mobile application for both Android and iOS. Probably not even with React Native.

Google Cloud was chosen as the hosting platform for the project because we knew that the client was also using it. The client gave us admin Google credentials that we were able to use to create Google Workspace and Google Cloud project for our staging environment. The client even exported to us their room and building information that we were able to import to the staging environment. It was great to use real room and building data in the staging environment because this way we were able to minimize the risk for production problems later.

Deployments to staging and later to production environment were handled through GitHub Actions. Merging to dev branch deployed to the staging environment and merging to main

branch deployed to the production environment. Autonomous deployments allowed fast fixes and modifications to the application.

3.3 Sprints

Get a Room! consisted of 7 two-week sprints: one for planning, five for implementation and one for quality assurance (QA). The requirements for the app were gathered and prioritized during sprint zero, and table 1 demonstrates the planned and implemented contents of each implementation sprint. No new features were implemented during the QA sprint – instead the sprint focused on stabilizing, testing, and documenting the software and implementing usability fixes.

Sprint	Planned content	Actualized content
Sprint 0 30.8.–24.9.	Grouping, contacting the client, and gathering requirements. Setting up development environments and working up a project plan.	As planned.
Sprint 1 27.9.–8.10.	First implementation sprint. Building a skeleton project and implementing first features. Getting to know the tools, finalizing MVP UI and UX design.	Mostly as planned. Additionally, a CI/CD pipeline was built, but unit tests did not make it to the sprint end.
Sprint 2 11.–22.10.	MVP feature implementations; UI and core functionality, test plan.	MVP on the way, otherwise as planned.
Sprint 3 25.10.–5.11.	End user mid-project testing, improvements design.	Still working with the MVP.
Sprint 4 8.–19.11.	Implementing the improved product design, extending the MVP.	Completion of the MVP, ad hoc usability testing, product stabilization.
Sprint 5 22.11.–3.12.	Final features.	Formal usability testing, usability improvements. Bug fixes.
QA sprint 6.–17.12.	Quality assurance; last end user tests, system tests and bug fixes, handing the project over to client. Project and tests reporting.	Last usability and bug fixes, project hand-over to customer, documentation, and reporting.

Table 3.1: Sprints with planned and completed contents.

As seen in table 1, the MVP implementation turned out to take a little longer than anticipated. The MoSCoW-prioritization that was implemented for the gathered requirements already in sprint zero, served as a handy tool for picking backlog items for each new sprint. Ultimately, the team managed to implement full MVP, test its usability with end users and implement several usability fixes. For the MVP, we completed 126 work items and the MVP contained all “Must have” features, and the prioritization did not change during the project. In addition, only supporting tasks were added to sprint backlog, so the team considers the requirements gathering and prioritization successful.

Although the group decided to record and prioritize all identified and customer-agreed requirements in product backlog, the purpose was not to cement the requirements or imple-

ment all of them during the fall, since it was apparent from the beginning that the requirements may change, and that the development resource was not going to suffice for implementing everything. Additively, we hoped that the formal usability testing session with end users would shed light on the priority of requirements.

As a result, we planned to follow agile practices and agree the next features in the beginning of each new sprint. This left us with an extensive set of 22 de-prioritized features still glooming in the backlog. As hoped, testing with end users gave us valuable feedback on the priority of the remaining features, as almost all improvement requests made by test persons had already been recorded in our requirements. Consequently, we consider the product backlog a record of ideas for further development and their priority has been recorded in the backlog with labels “Must have”, “Should have”, and “Could have”. Contrary to MoSCoW practices, we decided to not include “Wishes” in the backlog.

Sprint	Number of planned work items	Number of completed work items
Sprint 0	12	8
Sprint 1	21	5
Sprint 2	25	13
Sprint 3	28	17
Sprint 4	33	18
Sprint 5	38	27
QA sprint	19	19

Table 3.2: Sprints with planned and completed work items.

Table 2 displays the requirements in numeric format. A couple of interesting trends can be recognized despite possible errors in the data, resulting from manual insertion of work items. The number of planned work items seems to have grown steadily over the project. This is likely explained by the fact that we decided to split work items into smaller tickets. Another trend seems to be that we consistently over-estimated the amount of work we could complete during a sprint. Especially in the first half of the project, this trend can be explained with the team having issues with finding a united view on the definition of done, which resulted in confusion with when features are done, so we were left with leftover work for new sprints.

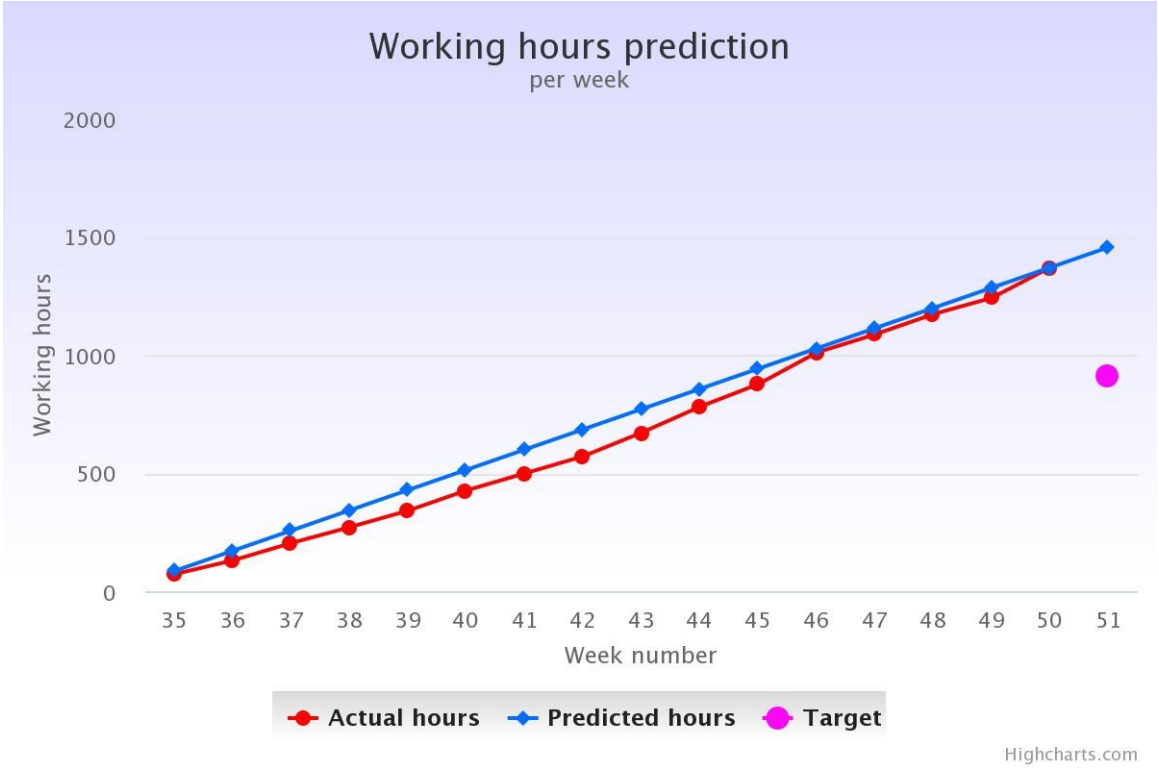

Figure 3.1: Predicted and actualized working hours as reported in MMT on Thursday Dec 16th, 2021.

Figure 3.1 shows the cumulation of working hours for the team. Ultimately, the team spent about 1400 hours on the project with an average of 87,5 hours per week, and 175 hours per sprint.

3.4 Deliverables and outcomes

The main deliverables from the project are the GitHub repository holding the source code for the system, and production deployment of the software in customer's environments. Source code is shipped with instructions on how to setup an appropriate instance of Google Cloud for the project and deploy the project into a production environment, and the instructions can be found from the project's README.md file. This means that while the customer has a running instance of the product in use already, they also have what's needed to set up the system in new environments.

To help possible future developers get started, we leave working test and deployment pipeline configuration in the repo, with an extensive set of unit tests in both backend and frontend to help prevent regression. We have also documented further development ideas, and a review on system stability and security, in the documentation that was handed over. Another notable entity from delivered documents is the result summary of the usability test session that can be used as a reference for evaluating further development ideas.

Some relevant metrics of the code base can be found from chapter *10. Statistics*.

3.5 Restrictions and limitations

Most notably, the speed of Google API serving the room reservation requests, has turned out to be relatively slow. Slowness restricts the responsiveness of the whole *Get a Room!* app, and there is not much the development team of *Get a Room!* can do about it.

On the requirements side, how to find rooms proved to be a popular development request for further development. However, the current data collected from the rooms appears to not have location information, so implementing the feature requires collecting new data. On a related note, arranging rooms in the main view of the app was another thing end users requested, and could be achieved, if location data was available for rooms.

Another notable restriction turned out to be development time, which resulted in excluding the one and only, originally MVP included feature that wasn't implemented during the project. This is requesting consent to save user data according to GDPR regulations.

3.6 Third party components, licenses and IPRs

Get a Room! software builds on several open-source libraries and frameworks such as React, Node.js, create-react-app, and axios. Licensing of the utilized OSS components allow use of the code to build and commerce new applications freely, with the precondition that original authors are accredited.

For the parts developed by the *Get a Room!* team, we follow a similar practice by licencing the software with a standard MIT licence, allowing copying, editing, extending, and using the code for any purposes, as long as original authors are accredited. This allows, for instance, Vincit to continue development of the product as they see fit.

What comes to Vincit's instance of Google Workspace, Google will keep on serving the service requests as long as Vincit pays the bills from the usage. Google does not have intellectual right to the software or data running in Googles cloud services but will bill from hosting the software and data.

3.7 GDPR, ePrivacy and related matters

Information security-wise user data is protected well-enough according to our understanding, gathered by careful design in the beginning of the project, and a security review performed by the team in the end of the project. Security review can be found from the testing report, section *4.4 Special testing*. Moreover, the customer performed their own review on the codebase before integrating the *Get a Room!* system with the production environment in the end of sprint 4.

However, the software does not currently fully conform to European General Data Protection Regulations (GDPR). The app handles and saves person data in the form of login credentials (Google ID) and office preferences, which technically makes the organization behind the app a registry owner. GDPR obligates registry holders to publish a registry report and offer users the ability to ask for a report on what data is saved of them and ask for its removal. Due to time restrictions, the group was unable to ensure the application conforms to GDPR, so the project team recommends the customer to consider this in further development.

4 Working hours

Despite group members did not give accurate numerical promises on the hours that they would be willing to spend in the project, we held a few motivation discussions during the project. More precisely, in the beginning of the project, around mid-project and once again, when members started to exceed the hours required by the course. The message was that

everyone was willing to go some amount beyond the minimum requirements, and most were okay with taking the hours that we would need for good results.

	Jere	Joona	Joonas	Martti	Mikko	Sara	Vivian	**Total**
Documentation	1	13,5	8,75	0,5	10,75	14,75	54,62	**70,55**
Requirements	0	1	0	1	3	13,25	10,65	**28,9**
Design	0,5	3	0	12	4,75	14	12,92	**47,17**
Implementation	66,25	153,75	117,25	81,05	84,75	35,25	0	**530,8**
Testing	1,5	7	0	0	0	15,83	9,02	**33,35**
Meetings	36,73	49,28	41,93	37,68	40,6	39,58	42,02	**244,67**
Studying	17,75	21,5	11	7,5	40,5	25,5	10,75	**134,25**
Other	18	9	8,25	18,4	5	27,1	65,67	**108,44**
Lectures	8,75	9,75	7	10,6	13,75	2,8	7,98	**53,63**
Total	**150,48**	**267,78**	**194,18**	**168,73**	**203,1**	**188,06**	**213,63**	**1385,96**

Table 4.1. Realized working hours in person-hours by person and task.

Differences in motivation and preconditions to spending time on the course resulted in a situation, where there would be some unbalance between group members. We did not go out of our way to maintain a balance either. Instead, we focused on the most relevant tasks that would produce value while them being tasks members could perform, and not give anyone more than they could do. How the hours divided in the end, can be seen in table 4.1. There are considerable differences, and we hope that this reflects on the amount of study credits team members receive from the course.

Week	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
Hours	72	130	204	271	341	426	498	571	670	780	876	1009	1086	1172	1241	1394

Table 4.2. Realized working hours (weekly totals / project).

Hours reported in tables 4.1 and 4.2 are approximations of the total realized hours, as reported in MMT on Friday Dec 17th. Charts illustrating the details of hours spent can be found from chapter *10 Statistics*, along with other numeric details of the project.

4.1 Personal contributions to project

Jere developed the React layout to the frontend and implemented the first version of the login and office selector to the frontend alongside other smaller UI components. Later in the project implemented front end tests with Martti and created poster about the project.

Martti Grönholm designed the user interface prototypes in the early phases of the project. Latter part of the project Martti implemented front-end alongside unit testing.

Vivian worked as the project manager, guiding the implementation work from the high level, trying to see ahead and recognize what needs to be done and when. She coordinated customer communication and spent time planning necessary process practices. She also held the ultimate responsibility over the team keeping up with deadlines and quality requirements, and a good deal of the content in the mandatory course deliverables such as project and test plans, as well as reports, came from Vivian's hands.

Joona was the teach lead on the project and responsible for deployments and Google Cloud environments. He also handled a great deal of tech related communication with the client and on the side some backend development and testing. Additionally, some time was spent on consulting team members on different technical problems relating to React, PWAs, development environments etc.

Sara collected, analyzed, and prioritized the requirements for the application. Her other main focus was on UX, and usability issues and she was the facilitator and organizer of the usability test session. She also supported Martti with his UI design and tried to point out any UX issues related to the design itself. Furthermore, she also implemented a couple components on the frontend with unit testing.

Joonas worked as a backend developer on the project, writing a large part of its main functionality. He also wrote the majority of the backends' unit tests and was the one responsible for figuring out how to use Google's APIs and services.

Mikko set up the development environment and its tools. A large part of this was the CI/CD pipeline, which Mikko initially set up (not the deployment part). He setup the simple database and did some basic backend functionalities. Later in the project transitioned more into frontend development.

5 Quality assurance

5.1 General description of testing

The testing approaches utilized in the project are detailed in our testing plan, and the results in our testing report. However, on the high level, testing was done throughout the project, focusing always on the work items on the team's table, and both customer, developers and end users participated in the testing activities.

First, testing targeted the requirements in the form of discussing and approving the collected requirements and their priority with the client. This work was orchestrated by our UX head. Once implementation work started, unit tests were written by developers to ensure correct functionality and prevent regression, roughly at the same time as the features were delivered. The unit tests are also integrated in out CI/CD pipeline, configured by our CI/CD specialist.

Related to feature releases, the customer got to perform acceptance testing towards the end of the project. The team agreed to post release notes to Slack on every time new features were merged into the staging environment, where the customer could go try out the feature and give feedback. On customer's acceptance of a feature, features were then released to the production version of the product.

Since the timetable for the project was rather tight, we could not implement integration or system tests in the form of automated tests, as we had hoped in the beginning. However, all features were tested manually, first by the developers, then by the client on our staging

environment before acceptance. Additional, informal manual system testing was done both by the developer team and the client in an exploratory form, to find possible bugs and required other fixes – many were found and fixed.

Finally, we conducted a formal usability testing session with four of our end users with different backgrounds to validate our requirements and find usability issues. End users were tasked with performing operations such as logging in and booking rooms, while using their own devices. Users were monitored and asked clarifying questions regarding usability as they performed the tasks, and test persons and device screens were recorded with camera to enable ensuring details such as how long operation take or what they do. The results were summarized in a summary document and processed into refined requirements and usability fixes, implemented in the last two sprints.

5.2 Bug reporting

The total number of bugs is difficult to determine since at the beginning bugs weren't clearly distinguished from other Trello tickets. However, the ones that could be distinguished are listed below in Table 5.1.

Number	Bug description	Severity (High, Medium, Low)	State
1	Current booking only shown after page refresh	High	Fixed
2	Current booking shows time left as 0 minutes, when less than 1 minute is left.	Medium	Fixed
3	Available room listing runs into errors when the client uses the application	High	Fixed
4	When a current booking is deleted, the room doesn't move back to the available list of rooms	Medium	Fixed
5	Collapse elements shouldn't automatically close	Low	Fixed
6	If a room cannot be booked for 60 minutes, shouldn't the 60 minutes booking button be pressable	Low	Fixed
7	The user isn't informed if a new booking cannot be made due to a currently active booking.	Low	Fixed
8	+15 minutes button should be disabled if it cannot be used to add more time	Medium	Fixed
9	Collapse elements with no details can be found and should be removed	Low	Fixed
10	Current booking time left is calculated wrong	Medium	Fixed
11	Toggle button reverses on the main view.	Medium	Fixed
12	The current booking isn't removed if it ends, and current booking time left becomes negative	Medium	Fixed
13	The frontend crashes if connection is lost to the server	High	Not Fixed

14	The application doesn't handle a situation if there are no rooms available	Medium	Not Fixed

Table 5.1: Found bugs and their state.

Most of the found bugs were found by the development team, but some of the bugs were also found by the client that has been testing the application. All bugs found by the customer have been fixed, except for number 13 and 14 that haven't been fixed due to lack of time, but still need to be acknowledged.

5.3 Conclusions on product's quality

The product is currently in production and is being actively tested and used by the client. Any errors found by the client are reported to the development team and these errors are fixed as soon as possible. Currently there are no bug fix requests by the client, indicating that the product should be error-free enough for production. Similarly, since the product is in actual production use already this also indicates that the customer is satisfied with the product and its quality.

Regarding security, the product can be considered safe to use in production as there aren't any known vulnerabilities that could affect the product's security.

It should be noted that there are a couple bug tickets in the product backlog, but these were found by the team and haven't been mentioned by the client yet. These should be fixed when the product is being developed further. Since there is a chance that these bugs can lower the quality of the product.

The client put great importance on usability of the product, which lead to testing the MVP with a usability test session. The results of this test were used to improve the product even further, thus increasing its quality even more, especially on the usability side. All the fixes that seemed critical have been done and added to the production code. From the results were also collected some further development ideas, which are discussed in *7.2 Further development*.

Especially since usability is very important for this product, another usability test session could be done to see if the made fixes were what the users wanted and if these fixes made them more satisfied with the product. Additionally, integration tests are highly recommended since it will ensure that any further development on the product will not break the currently developed application.

6 Risks and problems

6.1 Foreseen risks

Table 6.1 lists risks that were recognized beforehand. Some of the risks did realize, some of them did not, but for all of them, the team was prepared for. Most identified risks we were able to prevent from realizing, such as inadequate user involvement, by making plans to include users – e.g., by usability testing session. Another risk that was avoided almost completely due to good preparation, was misunderstanding requirements amongst the team.

For this risk, our requirements head prepared a use scenario document detailing how features should work, and team members were instructed to build their implementations against the scenarios.

Risk	Impact	Probability	Realized
1 Getting a suitable project	High	None	no
2 Misidentified requirements	High	Medium	no
3.A Failure at development environment setup	Very high	Medium	no
3.B Not receiving test data for development	Medium	Medium	no
4 Unbalanced workloads among team members	Medium	High	yes
5 Inadequate user involvement	High	Medium	no
6 Changing requirements	Medium	Medium	yes / no
7 Safari support consumes too much effort	High	Medium	no
8 iOS PWA incompatibility	Low	Low	no
9 Misunderstanding requirements amongst team	Medium	Medium	no
10 Underestimating technical difficulties	High	High	yes
11 Unfamiliarity with the tech stack	Very high	High	yes
12 Unstated requirements	Medium	Medium	yes / no
13 Neglecting non-functional requirements	High	Medium	no
14 Uncertain project scope	High	Very high	no
15 To-be-determined requirements	Medium	High	no
16 Free tier services	Very high	Very low	no
17 All work is not submitted in Trello	High	Medium	no
18 Inadequate customer inclusion in project	High	Very high	no
19 Working hours run out before completion	High	High	yes

Table 6.1: Foreseen risks with updated impact and probability, captured from MMT on Dec 14th, 2021.

However, some of the risks could not be completely avoided, such as unbalanced workload among team members and the working hours running out before project completion. These required us to take mitigating actions such as prioritizing work items and having motivational conversations about what we were still willing to accomplish.

For the part of risks 6 and 12, we had no difficulties with in general since while there were some changes in requirements, handling them was controlled in the sense that they came in as testing results. The only surprise was in the project end, when customer requested transferring the code base along with the CI/CD pipeline from GitHub to Vincit's GitLab. Why this caught us off-guard, was that we had discussed hosting the project in GitHub already in the first client meeting, and the final weeks of the project were rather busy. However, we understood that it is important for the customer to host and administer services running against their internal systems from their own hosting instances, and thus, found middle ground in moving the sources under Vincit's GitHub organization.

What comes to risks 10 and 11, these realized in one package; we had difficulties in planning small enough sprints and implementation of the MVP took longer than expected. However, our prioritization saved us from getting into trouble by this underestimation, since our minimal goal was to implement MVP features, which we managed anyway.

6.2 Risks not foreseen

One pain point was realizing that getting an approvement to a feature and acceptance for its implementation are two different things. The confusion presented itself in the sprint reviews, when presenting the outcomes of previous sprint. The customer was rightly confused why we had marked features done, despite them not being integrated in the running demo. However, we found an agreement on how to process tickets in the future, and we believe the customer was ultimately happy with us listening to feedback, despite there being some bumps in the road at first.

Then there were some issues with finding a united view on definition-of-done (DoD). The management believed it self-evident that completing a feature included writing sufficient unit tests, and integrating the feature into the product, but developers had differing views, so there was some panic with getting work finished in the beginning. Luckily, we found a common chord on this issue as well, although not before the customer having to give feedback on that as well.

Release management was something we considered at some point in the beginning but excluded from the plans as we expected that there would not be need for it. Ultimately, we ran production and demo versions separately, and getting features accepted by the customer and merged into the production turned to be a bit of a hassle. Of course, with our scale of a project, we could manage the chaos with our architect casually taking the release management responsibility, but thinking ahead, there should be clear process for making product releases.

On the technical side of things, there were some issues regarding usability test persons' own, saved Google IDs interfering with test user credentials in the testing session, resulting in the user not being able to log in to the system with reasonable effort. We changed those two users to use the test operative person's personal devices for the duration of the test as an emergency solution, and similar problems are unlikely to occur in production, since there, people use their personal Google IDs instead of test credentials. We also ran out of both memory and battery with the camera during the testing session.

7 Not implemented in this project

7.1 Rejected ideas

The API documentation was originally accessible from an endpoint in the backend, using the third-party npm package swagger-ui-express [https://www.npmjs.com/package/swagger-ui-express]. Later it came clear that this approach was not very practical, and it was easier to check the API documentation from a static address that was always up. The package was consequently removed from the project. This feature would have been used only by future developers, so the removal of this package did not affect the final product at all.

The idea to implement an easier login method compared to Google SSO was rejected at the very beginning of the product development. Initially, a third-party npm package passport [https://www.npmjs.com/package/passport] was used for providing the authentication flow as an easier alternative to Google's own authentication library. Eventually, the passport proved to be missing some features that we required, and we ended up switching back to Google's library even though it was more difficult to implement. Ultimately, the users will be using their Google work accounts to use this application and it would be a lot better to not waste time on another login method.

A couple of ideas that were recognized during the usability test session were also rejected since it was apparent that these issues are very specific and would require tremendous work on the backend of the product. Thus, it was better to not waste time on a mundane task and instead focus on what really increases usability.

7.2 Further development

There is quite a long list of possible further development ideas. The list consists of requirements that the team didn't have time to complete due to lack of time and due to having a lower prioritization than must have.

- The user can switch the current booking to another room on fly
- The user can stop the current booking without deleting the whole booking from Google Calendar
- The user can book a room for the duration of how long it is free for
- The user can edit the booking title
- The user can view future free rooms
- The user can add favourite meeting rooms
- The user can filter the free meeting rooms with their favourite meeting rooms
- The user can book more than one meeting room
- The application fetches user Workspace settings and applies them
- The user can book a meeting room with a custom time
- The user can add other participants to the booking
- The application sends an invitation to other participants
- The user can edit their favourite meeting rooms

Furthermore, even more ideas were revealed during the usability test session. Some of the ideas supported the ideas found from the above list, however, there were also some new ideas that should be mentioned.

- Information on where each meeting room is located within the office
- Custom room order (alphabetical, capacity, physical placement of rooms)
- Some users should be able to change office preference from main view
- Better Vincit branding visually and textually
- The need for accessibility
- Need for a google play application
- User can book other resources than meeting rooms, e.g., company car
- User can fetch free rooms with criteria.
- User can book pop-up workstation
- Notification when current booking is running out of time

Additionally, the application works only on Vincit's Finland offices. But in the future, it could be expanded to work on Vincit's offices across the world. The Google Cloud staging environments admin credentials are given back to the client, so client can use it on further development and possible bug fixes. It is always good to have a staging environment because testing in production is not a good idea.

8 Lessons learnt

8.1 Team's strong points

To start with the positive, communication amongst team members and helping each other went brilliantly for the project. Another great thing was the team's professionalism in terms of attitude: doing things well, searching for improvement points and willingness to put the needed extra effort in. The team also had good technical skills and was effective in solving technical issues. Another thing to applaud for was preciseness: for most parts the team worked well coordinated, and especially requirement definitions made the needed work clear cut.

Of course, many a thing didn't go quite as well as hoped at first, but the customer gave us plenty of good tips and other feedback, which we welcomed, trying to improve for the next time. For this, the team received positive feedback.

8.2 Take-aways

While the inner working of the group was smooth from beginning to the end, we also had our difficulties. Our biggest pain points related to communication with the customer, even though the customer was, what the group deemed, exceptionally helpful and supportive. The difficulties can be explained simply by inexperience with customer interaction, which makes sense, since it is the subfield, the group collectively had least experience with. At the same time, better practices with customer interaction are probably one of the greatest take-aways for the team members.

To get into the details of the difficulties, for one, it took us a while to realize that customer is not really interested in what everyone did or will do in terms of work division, but instead, would like to process things from the feature point of view. More specifically, what are the completed features, and what features can the customer expect to see the next time.

Another pain point for us in the sprint reviews was how to present the demo. Ultimately, we decided that we should arrange feature tickets on top of the "developed in this sprint" list on Trello board, and demo the features one by one, while continuously asking for feedback from the customer. Then we could move the completed tickets to "Done" column with the customer when the feature was approved.

Regarding the presentation, the customer instructed us to always appoint the presentation responsibility to the group members, that implemented the feature. Not only this improves communication, but it also provides group members the opportunity to show pride in their work and get credit for it. All of this might sound self-evident, but we would not have realized these practices if it weren't for the customer. These tips, and several other, were presented to us by Vincit employees and for those learning experiences, we are profoundly grateful.

8.3 Learnings

One important learning was to realize the importance of communication and asking for help. For team performance it is important that everyone knows what other team members are doing at the moment. Also, although it is important to solve problems individually it's also important to know when to ask help because it is highly unmotivating to spend hours on something with zero progress. Usually after asking help, problems can be solved much faster, sometimes in minutes.

There was also considerable learning happening on the technical side of things. For instance, Google Cloud and Google Workspace APIs were completely new for everyone. Few members of the team were assigned a task to study these technologies and to transfer this knowledge to the rest of team. Many working hours were spent on studying them but undeniably this knowledge will be very valuable in the future. Also React was fairly new technology to most of the frontend team so quite some time was used studying it.

Luckily, the team had some members that had experience on delivering React projects, so they were able to consult the team on frontend problems. Additionally, plenty of knowledge on Node.js, Visual Studio Code and TypeScript development were transferred between team members. Besides mentioned, knowledge on things like architecting, deployments, Cloud environments, SSO and GitHub Actions were expanded.

Studying and planning the platform-compatibility was also a great learning experience. We spent quite some time studying if the PWA would be suitable technology for creating the Get a Room! mobile application.

Additionally, UX was new to everyone, and a lot of studying was required for that as well since the client put great importance on UX and usability. Some books and UX blogs were read. Some beginner knowledge from Human Technology Interaction courses was also implemented on this course. Furthermore, Usability testing was a new concept and required a lot of studying and reading about and the usability testing experience itself was completely new to the facilitator and observer.

9 Comments about the course

We found the course very useful, and we learned a lot. Document templates were helpful in terms of keeping track of the project. It was good that we were trusted with using our own judgement, while sufficient guidance was offered, especially in the terms of deliverables and document templates. If one could present a wish for additional content, some tips regarding how one could go about presenting done work and communicate contents of the upcoming sprint would be nice for helping future groups get a smoother start with sprint reviews. See section *8.2: Takeaways* for our notes on the topic.

In the future, it would be helpful if instructions on each deliverable and deadline could be gathered into one, apparent place. Now, information was scattered between lectures, emails, document templates, different slide sets and Moodle tabs, so finding all the necessary information was laborious, error-prone and one could never be quite certain whether something was forgotten or not.

10 Statistics

The project team consisted of seven student members, presented in figure 6.1. Senior members are presented in the first row and juniors on the second row for layout reasons. The customer was Vincit co. While not all team members knew each other before the project, communication worked quite as well as could be hoped for, and by the end of the project the group formed one entity, as opposed to few clicks.

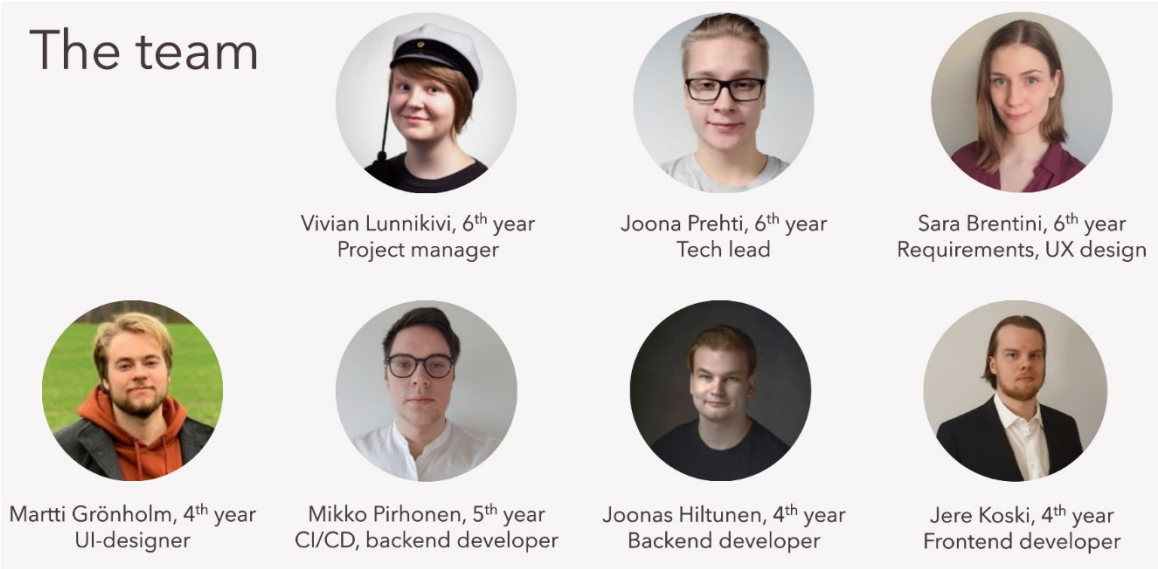

Figure 10.1: Development team members.

10.1 Code metrics

From the technical side of things, table 6.1 presents the different technologies and tools utilized in the implementation of the *Get a Room!* software. Management and communication tools are excluded from the table since these have been discussed already in section *3.1 Communication*.

Common	Backend-specific	Frontend-specific	Other
TypeScript	Node.js	React	Git
Axios	Express	MaterialUI	GitHub
Luxon	Google Workspace	create-react-app	GitHub Actions
Jest	Google Cloud		ESLint
npm	Vincit/Google SSO		

Table 6.1: Implementation tools and technologies by components.

During the project, the system was deployed to Google Cloud 110 times to both staging and production environments, making a total of 220 deployments. The deployments along hosting the app in Google Cloud cost 15,86 € worth of free Google credits.

Figure 10.2 presents some key numbers of the project's dev branch, as reported by cloc tool, made by author AlDanial. The statistics are from Monday Dec 13th after which only a few new fixes were implemented, so the numbers are very close to final ones.

Most notably, we did not write 56 thousand lines of code manually during the project despite that being the code line count. Out of this, about 48 thousand lines are plain generated JSON, likely package.lock files. Instead, the amount of TypeScript lines reflects most of the work that were done by the team. Considering the file types, it is quite safe to estimate that the group wrote roughly about 7 thousand code lines and roughly a hundred files by hand.

```
     137 text files.
     134 unique files.
      16 files ignored.

github.com/AlDanial/cloc v 1.82  T=1.58 s (77.2 files/s, 36210.2 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JSON                            10              0              0          48078
TypeScript                      97           1122            405           6817
YAML                             1             49             14            229
XML                              1              0              0            171
HTML                             2              2              4             81
Markdown                         3             59              0             76
SVG                              3              0              1             42
JavaScript                       1              7              4             40
Dockerfile                       2             12             11             16
CSS                              1              1              0             12
Bourne Shell                     1              1              0              3
-------------------------------------------------------------------------------
SUM:                           122           1253            439          55565
-------------------------------------------------------------------------------
```

Figure 10.2: Lines of code by the cloc tool from author AlDanial on GitHub. https://github.com/AlDanial/cloc

On the topic of code lines, it is notable that while we cannot give precise number for how much of the code is reused. Both our frontend and backend utilize several external libraries. In the world of modern web development, it is common to have tons of dependencies and that is the case on our project as well. To give an idea about the size of the frontend for people that understand React, our frontend consists of 23 separate React component. On the backend, we have, e.g., 16 controller files, 2 model files and 7 API endpoint definition files. So, the app is rather compact, still.

Our product backlog in Trello holds 21 unimplemented work items. These are mainly feature tickets with *Should have* and *Could have* priority. By the latter half of QA sprint, implemented work item count was 115.

10.2 Figures from MMT

All figures in this section have been captured from MMT on Thursday Dec 14th, 2021. As this is but the beginning of the last week on the project, these figures do not represent final working hours, but give a close approximation of the final hours.

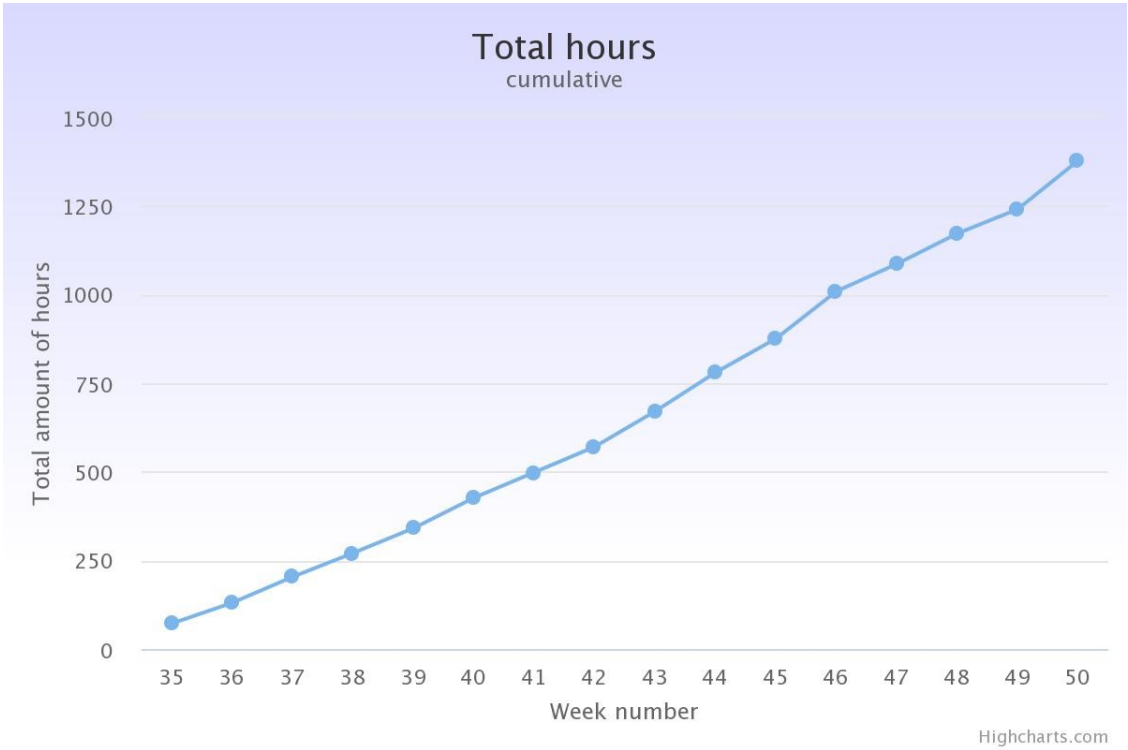

Figure 10.3: Total cumulative hours team logged in MMT.

Figure 10.3 illustrates the cumulation of the work hours spent with the project. The growth of cumulative hours is quite steady, although a jump in hours can be spotted from week 46. Week 46 was the latter half of implementation sprint 4, in which we had decided to complete the MVP implementation in order to have enough time for stabilizing the software and do usability testing, as well as implement usability fixes. We ended up exceeding the estimated time usage for the course by one fourth.

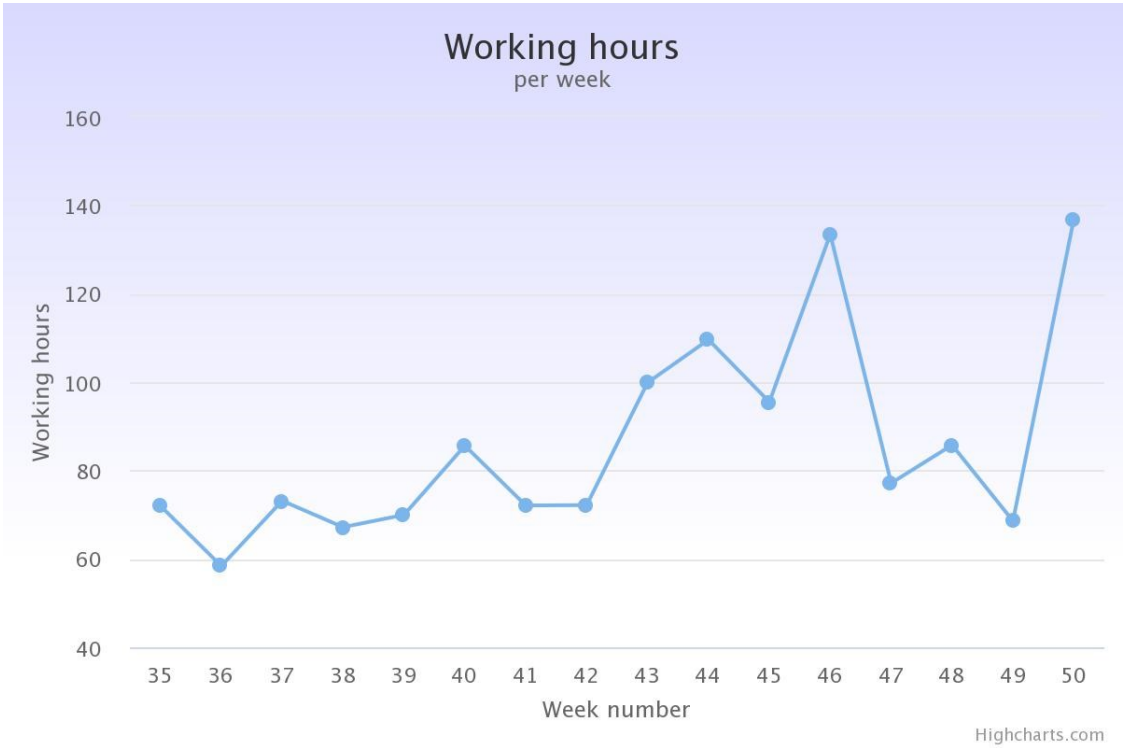

Figure 10.4: Total weekly hours team logged in MMT.

The spike in hours is especially well demonstrated in figure 10.4, showing hours per each week. Another trend seen in figure 10.4 is that once the implementation work got into full speed, team has been spending a lot of hours, but after MVP completion, there situation normalized.

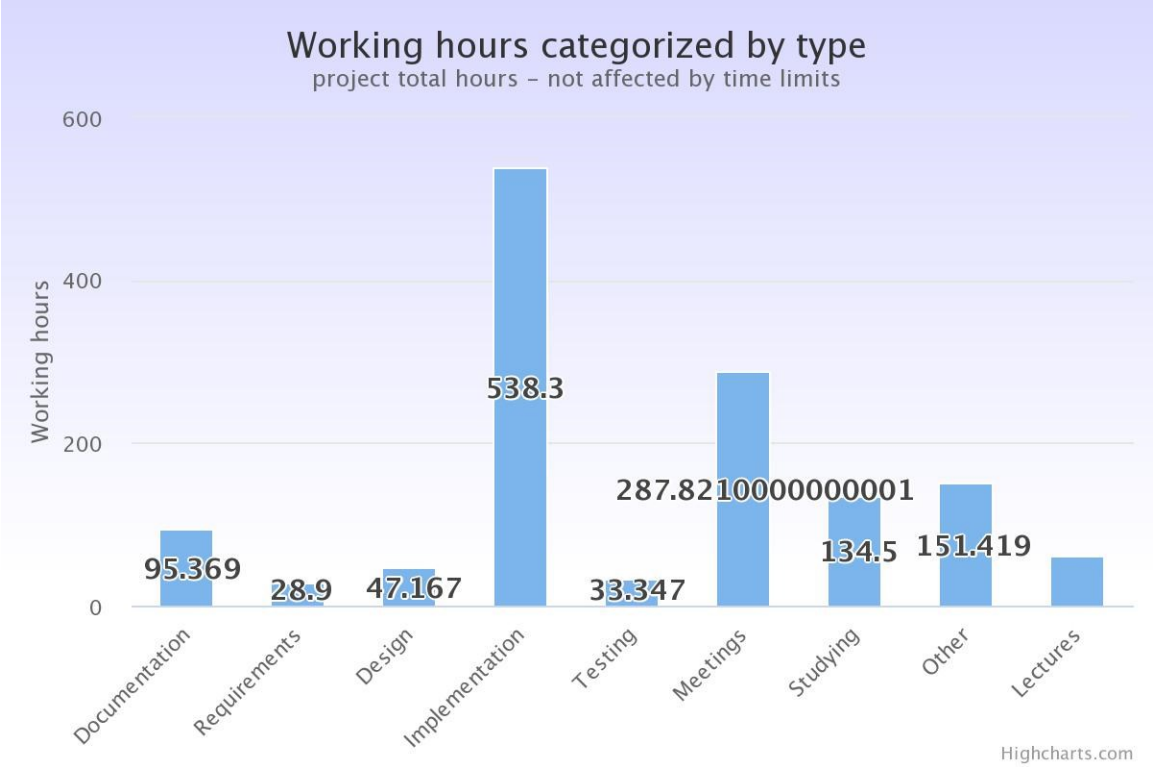

Figure 10.5: hours by work category.

Figure 10.5 categorizes work hours by the labels they were logged with in MMT. Not seen in the figure 10.5, category *Lectures* has 60.63 hours. The relative porpotions are better illustrated in figure 10.6 that shows that the two main things team spent time in was either implementing the product, or discussing how to implement it in meetings. Considerable efforts have been made both in studying, and in design and documenting.

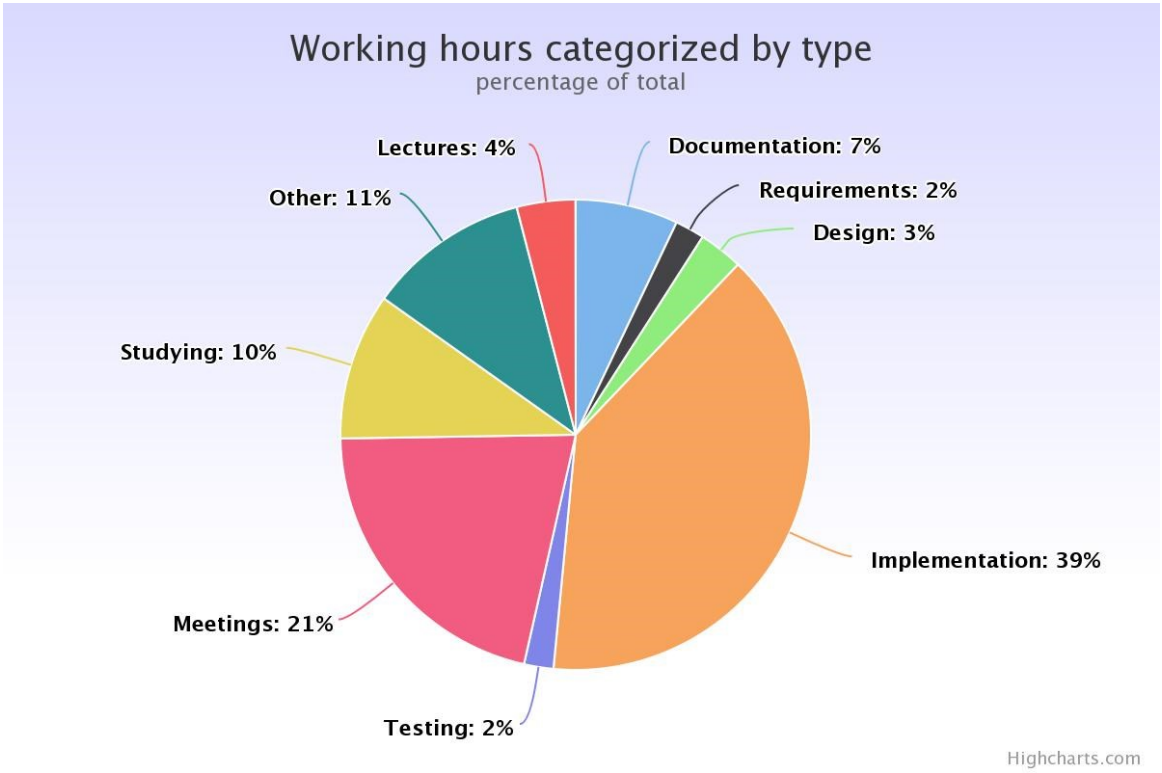

Figure 10.6: hours by work category, in relative illustration.