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

Group 1

Get a Room!

Test Plan

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

Version history

Version	Date	Author	Description
0.1	01.11.2021	Vivian Lunnikivi	Add member names and student numbers, sketch DoD and general testing approach, testing roles and purpose of the document
0.2	03.11.2021	Sara Brentini	Special testing and added appendix B
0.3	03.11.2021	Mikko Pirhonen	Chapter *5 Adopted tools*
0.4	04.11.2021	Vivian Lunnikivi	Sections *1.1 Purpose and scope of the document*, *1.2 Product and environment*, and *4.5 Acceptance testing*. Sketch for section *1.3 Project constraints*. Added appendix A
0.5	05.11.2021	Vivian Lunnikivi	Chapters *3 Testing process* and *6 Open issues*, sections *1.3 Project constraints related to testing*, *4.1 Unit testing*, *4.2 Integration testing*, also fixed some minor issues
0.6	05.11.2021	Joona Prehti	Chapter *2 Quality assurance process* and section *4.3 System testing*
0.7	05.11.2021	Joonas Hiltunen	Add description of request performance testing in section *4.3 System testing*

Contents

1 Introduction

1.1 Purpose and scope of the document

Get a Room! *is a software project offered for implementation, by the technology company Vincit, in the project courses* COMP.SE.610 *and* COMP.SE.620 Software Engineering Project 1 & 2 *in fall 2021. The product produced in the project, is a mobile application targeted for the employees of the company to easily make ad hoc conference room reservations in the company's premises.*

This document lays out the plan for what parts of the *Get a Room!* product is tested, the planned methods for testing it and what is hoped to be achieved by testing it. The plan also lays out responsibilities, principles, and the schedule for the testing process and documentation of the results, guiding the implementation process.

Testing targets not only verifying the software itself but validating the product requirements. Techniques utilized in testing include automatized unit and hopefully, integration tests, which verify the correct working of the software, and formulate regression tests that enable developing with confidence during the project as well as after it's hand-over. Static style checks are integrated in the development workflow for keeping the code clean. Major efforts are put in usability testing for requirements validation, and it is done by both the customer and the project team. We want to verify the delivered product matches the customer needs.

To summarize the desired results, testing aims to improve code quality, product quality, and customer satisfaction, as well as assure software reliability, security, and platform compatibility. Additional expected benefits include improved communication amongst the team and offering support for whoever decides to continue development of the project after we hand it over. Testing provides a reference for what features the product has, how its functionalities have been verified and what are the already found constraints or improvement points that require attention.

1.2 Product and environment

The product is called *Get a Room!* It is a PWA application, which runs on a mobile phone browser, but behaves close to a native application. The main supported platforms are Windows with Edge and Mac/iOS with Safari since these are the most common platforms in use at Vincit.

The frontend application is built with React and the backend server is built with Node.js. The system also utilizes google APIs, and a more detailed description of the product and the system architecture can be found from the caption of the *Get a Room!* project plan, appendix A.

1.3 Project constraints related to testing

The most critical constraint concerning testing is the availability of resources. To ensure the product matches user needs and works properly even in erroneous situations, requires diverse set of testing techniques and simply, a lot of time. Therefore, towards the end of the project, more and more emphasis is put on testing and fixing issues revealed by testing.

From the technical point of view, testing environments, user data confidentiality and data integrity pose some restrictions to testing. For example, the customer must perform acceptance testing and ad hoc usability testing in the testing environment with test credentials instead of the production environment, where end users could use their personal google IDs for logging in. This harms evaluating the onboarding experience, and results from the fact that we cannot take the software to the production environment before it has been verified to work properly and secure enough without risking of losing data or exposing it for outsiders.

Another restriction related to user accounts is that we only have one test user account that is shared among everyone performing tests in the testing environment. This includes at least the development team and the customer as well during week 47. Multiple simultaneous users for the same test user account might cause conflicts, which we just must live with. Why there aren't more, is because users are not free. Sharing the account might also pose a minor risk for abuse. This risk, however, can be mitigated by changing the password after ad hoc testing.

Usability testing must also take place at Vincit's premises, for making the testing situation as similar to real use situations, as possible. The customer has also requested for a week's notice to ensure enough end users can attend the session and that the customer-provided testing gear is available for use during the test session. Constraints related to the usability testing session also include the confidentiality of their personal data might be temporarily endangered, since we must film the user's device screen during the session to evaluate the user experience. The participants will be notified before accepting to be test participants that the session will be recorded via microphone and mobile screen will be filmed with a document camera. This is to ensure that the test participants know what they are signing up for.

1.4 Definitions, abbreviations, and acronyms

CI/CD	Continuous integration, continuous delivery. School of software engineering underlining the role of automation and testing.
Core functionality	All technical functionality required to implement requirements labelled as "Must have".
DoD	Definition of done. Criteria for a feature to be considered completed.
Exploratory Testing	Testing technique involving no test cases but stories to follow.
Linting	Automated static code analysis used to flag programming errors, bugs, stylistic errors and suspicious constructs.
MVP	Minimum viable product. Minimum set of features completed for the product to be considered ready for release.
PR	Pull request. Request to review and integrate feature implementation as a part of the product code base.

QA	Quality Assurance. Activities related to verifying sufficient quality of the product.
Regression	When development of new features breaks existing functionality.
Scenario, use scenario	Description of how the software behaves in a use case.
Scrum	Agile software development framework.
UI	User interface. Interface between the system and the user of the system, allowing interaction between the parties.

2 Quality Assurance Process

2.1 Manual Quality Assurance Process

One of the most important ways to manually assure the quality is to use pull requests every time code is merged into development branch. This ensures that only reviewed code can get to our staging environment. Most pull requests are reviewed by at least two people. Importance of pull requests is highlighted because some of the technologies are quite new to most of the team. Usually all in the backend team reviews all pull requests related to backend and same for the frontend. Customer is involved in the quality assurance of usability through usability testing plan. More about this on *APPENDIX B: Usability testing plan*.

2.2 Automated Quality Assurance Process

When code is committed locally, prettier runs and formats the code according to the team's styles. Also on local commits, ESLint runs and does static code analysis. These tools also run on the CI/CD pipeline. Additionally, both frontend and backend tests run during the CI/CD pipeline. More about tools on chapter *5 Adopted Tools*.

3 Testing process

3.1 General approach

Unit tests are implemented for verifying correct working of core functionalities and preventing regression during development. Both frontend and backend are tested, and the regression tests run as a part of the deployment pipeline, providing test automation.

Alongside unit tests, we would love to implement automated integration tests for the core functionality as well to verify software component interactions. However, with the current timetable, it seems inevitable to pinch time from something, so we will likely take the responsibility of interactions testing into system testing phase and settle with manual testing instead of test automation. This has the downside that in further development, there is less assurances for developers about whether their edits introduce regression.

Nonetheless, we have planned and documented integration test cases to help possible future developers figure out our course. The planned cases are laid out in section *4.2 Integration tests,* and we will begin the implementation if there is time. Planned integration test cases also serve as a base for the system testing round during the QA sprint.

Another key point in testing is assessing the usability of the product. Usability testing is done both by the client and the development team and does not restrict only to general usability issues such as delays of font sizes but includes validation of product requirements to make sure we are building the right product. More about usability testing in section *4.4 Usability testing*.

The customer has not expressed need for formal acceptance testing, but the customer has expressed their wishes regarding acceptance. The customer will use their personal judgement on when and whether the product is usable, useful, and mature enough to be adopted into use.

3.2 Definition of done and goals

Our definition of done (DoD) contains five parts: implementation, checking the implementation against requirements, testing the implementation, review of the implementation and integration into the product. The steps and the corresponding quality assurance goals are discussed below.

Feature is implementation includes both the frontend and backend functionality related to the feature under development. This requires coordination between frontend and backend developers to synchronize efforts towards same features and synchronization is considered already in the sprint planning process.

Once the developer of the feature is satisfied with the implementation, they check that the implementation matches the requirements of the corresponding scenario description, stated in the requirements scenarios document. Scenario descriptions are considered the source of truth, to ensure common understanding of what should be implemented. Scenario descriptions also enable controlled management of possible updates in the requirements and collect an updated view on the requirements in one place.

Unit tests are implemented for the core functionality of the feature, both in frontend and the backend, to verify correct functionality both in happy and most likely negative cases. Small-grained testing provides traceability for possible bugs, improves code quality, and offers certainty over correct functionality. Automated regression testing gives confidence over that implementing new features and refactoring does not break existing code – thus, all unit tests run as a part of the integration pipeline automatically.

Code must pass style checks and the manual review process, achieved via code review done before accepting a pull request (PR). Code review is considered a good practice to improve code quality, especially in terms of understandability and finding bugs. PR process also enhances communication between developers and catches bugs and misunderstandings. Style checks, running as a part of the integration pipeline, are used to enhance collaboration and understandability of the code.

Finally, a feature must be integrated into the final product and be accessible in the production environment. If the client can’t test it, it is not considered part of the product.

Apart from product acceptance, there is no acceptance testing for individual features. Of course, there is a heavy weight in usability testing with end users, and the results from usability testing are used for updating requirements. However, implementations of improvements are considered “new features” and may not fit the project schedule in their entirety. They are then left for further development.

3.3 Testing roles

Regarding the produced code, developers do all the testing. Developers have the responsibility to test their own code against scenarios manually and write unit tests for their code. Since every developer will test their own code, the frontend developers will test mainly frontend and backend developers mainly backend. Instead, integration and system test responsibilities are shared, so development team will coordinate to decide on who will implement or perform which test case. Developers also help in testing others’ code via the pull request and related code review process.

Test automation also has an important role in testing the software. The CI/CD responsible has taken care of building the integration pipeline. The pipeline includes style checks and regression tests for both frontend and backend.

The customer participates in the testing by performing ad hoc usability testing and acceptance testing in a scope they see appropriate. Usability testing validates requirements, at first in test environment, later hopefully in production environment and acceptance testing is required before deploying the product in the real production environment.

Last party involved in testing is the end users. End users participate in the ad hoc testing and the usability test session, providing invaluable opinions on how the product should work, look, and feel.

3.4 Test schedule

Table 1 presents the planned testing schedule for the project. As seen in the first row of table 1, testing has started before implementation of any code as requirements validation. The requirements were discussed, transcribed, and reviewed with the customer to ensure the team has a common understanding of the requirements with the client. Similar process was used for UI design, and architecture design has also been discussed with the client, although more informally.

Weeks	**What**
35–42 (31.8.–22.10.)	Requirements gathering and validation Requirements review meet with the client 6.9. UI design review meet with the client 22.9. Written scenarios 26.10.
39–50 (27.9.–17.12.)	Unit testing
47 (22.–26.11.)	Ad hoc testing and acceptance for MVP
48 (29.11.–3.12.)	Usability test session Plan fixes Integration test implementations if any
49 (6.–10.12.)	System testing and reporting Fixes
50 (13.–17.12.)	Last fixes Final reporting

Table 1: Testing schedule summary.

Some minor refinements have been made to the requirements along the way. Requirements were ultimately written also into formal scenario descriptions to allow assuring implementation matches agreed requirements.

As per the second row of table 1, continuous unit testing was meant to be begun from the beginning of implementation. Backend testing had a smooth start, but unit testing of the frontend was delayed to the second implementation sprint.

Next types of testing, ad hoc and a type of acceptance tests, succeeds the completion of the MVP. Acceptance testing in this case means, that the customer must approve taking the MVP product to the production environment is safe enough. Around the same time, the customer may also begin gathering feedback on the MVP product via ad hoc usability testing with the end users. Hopefully we'll get the acceptance, after which we may conduct our formal usability test session. The session should date in week 47, as laid out in table 1.

Integration tests, if any, will be implemented during weeks 47 and 48, when the MVP features have been published and the product is being tested for its usability.

In the QA sprint, the work related to testing will include performing system testing and reporting the results of all testing done in the project. There is also a time reservation for bug fixes and improving tests where needed.

3.5 Test documentation

Main information source of the results of testing will be a testing report, written in the end of our work in the project. The report will detail what was tested, how, and what was found during tests, as well as if we have decided on improvement points that we did not yet have time to fix in the product.

The usability testing results will be documented in a separate final report, attached as a part of general test report. This report will contain a brief introduction to what was done, what the original research questions were, and what the findings were to these questions. These results will be further discussed together with the development team and in a separate meeting with the client. These meetings will cover what the implications of the findings are to both the team and the client, what our recommendations are for future actions, and what possible follow up research can be conducted for this application.

Other guidelines for executing testing include principles laid out for regression testing, and system testing. For unit tests, this means heuristics detailed in section *4.1 Unit testing* and for integration tests, planned test cases are collected into section *4.2 Integration testing*. Planned system testing is detailed, in section *4.3 System testing*.

4 Test cases

4.1 Unit testing

Unit test cases should cover at least the core functionalities of every implemented feature. Core functionalities cover all technical implementation required by MVP features. MVP features can be found from table 2.

Main Functionality	**ID**	**Sub functionalities**
Login	A	Logging in by using Google SSO
User's preferences	B1 B2	User can set their office location User can edit their office location
Available room search	C1 C2 C3	The user can search for free rooms available at the office. The user can see how long each room is available The user can see what features each room has.
Room booking	D1 D2 D3	The user can book an available room The booking title has a default value The room can be booked with default times; 30 or 60 minutes

User's own reservations	E	The user can see their own current booking
Add time to current reservation	F	The user can add more time to their current booking
Delete current reservation	G	The user can delete their current booking

Table 2: *Get a Room!* MVP features.

Unit testing a feature means implementing both happy and negative cases for it. Implementing positive and negative cases asserts the software works in an expected way both when everything works as they should, as well as when errors occur. Common errors that are likely to occur in *Get a room!* include at least connection errors to the server, timeouts for logins, and user related activities such as entering wrong account credentials, or the app being suddenly shut down. The software should behave graciously even when these errors occur and be able to recover from these error situations.

4.2 Integration testing

Integration test cases are inferred from the MVP requirements, listed in table 2, by including the most common error cases. Some overlaps in similar test cases have been removed to optimize testing efforts, but more optimizations may be performed by choosing the test implementation order. Table 3 lists integration test cases from the feature point-of-view, when error cases are considered, H for happy cases, N for negative. Emphasized cases are considered the most important ones.

Feature	Case ID	Test case
A	**A.P1** **A.P2** **A.N1**	**Login with Google SSO succeeds with correct credentials** **Login fails due to wrong credentials** **Login fails due to network error**
B1	**B1.P1** B1.N1	**User sets their office location on first login, succeeds** User sets their office location on first login, fails due to connection error
B2	**B2.P1** B2.N1	**User edits their office location, succeeds** User edits their office location, fails due to sudden shutdown
C1	**C1.H1** C1.H2 C1.H3 **C1.N1**	**Room listing shows the available rooms at the user's preferred office location** Room listing notifies the user when no rooms are available Room listing notifies the user of connection error when room data cannot be fetched from the server due to connection error **Erroneous room data is received**
C2	C2.H1	The user sees how long each room is available
C3	C3.H1	The user sees what features each room has
D1	**D1.H1** **D1.N1**	**The user succeeds at booking an available room** **User tries to book a room that another user succeeds to reserve just before. User gets a notification of failed reservation**
D2	D2.H1	User books a room without editing default settings. The booking's title has a default value
D3	D3.H1 **D3.N1**	Booking succeeds, when use books a room with either one of the default time booking buttons; 30 or 60 minutes. **Booking succeeds in the border case, where time goes below 30/60 minutes while booking request travels to server.**

E	E.H1 **E.H2** **E.H3**	The user can see their own current booking **The user does not see upcoming bookings, only current ones** **User sees their all current bookings if there are more than one**
F	**F.H1** **F.H2**	**The user can add more time to their current booking** **User cannot add more time than is left before next booking**
G	**G.H1** G.H2	**User does not see the deleted booking, when user succeeds at deleting their own current booking** User sees their booking if deletion fails due to connection error

Table 3: Planned integration test cases for MVP features.

Of course, as discussed in section *3.4 Testing schedule*, the time will probably not be enough for implementing these integration tests as programmed test cases, running as a part of integration pipeline. Table 3 can then work as a reference for manual system testing, so we will have the knowledge of the system correctness status at least at the moment of hand-over.

4.3 System testing

System testing will be implemented by manually testing all the features on all supported platforms. Platforms that will be tested are Windows Edge and Mac/iOS Safari. Also, PWA applications both on Android and iOS will be tested. Manual system testing is performed as exploratory testing, utilizing the integration test case list or use scenario descriptions, and any flaws will be reported. Every core functionality will be tested on all supported platforms. System testing assures that the various components of the application interact correctly as a whole on all supported browsers.

Client will implement ad hoc testing on their own offices. More about that in chapter *3.3 Testing roles.* As last notion there probably isn't going to be automated system testing because of time constraints.

Performance of the backend is tested by measuring the response times of the requests made with Postman. Multiple requests are made to the same endpoint, and the average response time is determined from the results. This kind of performance testing assures that some endpoints do not become too bloated.

4.4 Usability testing

Usability testing will be implemented to gather qualitative data on the usability of the application. It will be done when the MVP features have been completed and when the client has been able to start their ad hoc testing.

For the usability testing a set of research question have been provided that are used during the testing session and afterwards to analyze the data. The testing itself will take place in Hermia, Tampere, where we expect to test 5 Vincit workers with the application on mobile phone. The estimated time for each individual test will be around 20 minutes and during this time the participant will undertake a set of 5 tasks designed beforehand and answer questions if needed.

During the testing the participant will be recorded via a mic and a document camera to video capture the phone screen. The session will be held by a facilitator and second person who will be writing notes down during the testing session. Once the sessions are over, the collected data can be analyzed to hopefully understand the usability of the application better. The usability testing plan was validated by the client and the final plan can be viewed in Finnish at appendix B.

4.5 Acceptance testing

Although there is no formal acceptance testing agreed, the minimum requirements for taking the system into production environment have been established with the customer. Before delivery, the MVP features have been implemented as per our DoD, defined in section 3.2, and the customer has tested the product in the testing environment – the test environment being a copy of the production environment with the one difference that instead of real user accounts, the only user accounts in the test environment are our test user accounts.

5 Adopted Tools

The project uses a variety of tools to ensure quality in produced code and functionalities. The CI/CD pipeline is the backbone of this process, facilitating all the other tools and running them on every commit to the repository in GitHub [1]. The same tools are used for both the frontend application and the server application. This is possible because both subprojects' main technologies are supported by the tools. All the major quality assurance tools used in the project are listed in table 1.

Tool	Description
GitHub Actions [2]	Used to create and run the CI/CD pipeline, which runs the other quality assurance tools.
Prettier [3]	Used to create and enforce code style guidelines.
ESLint [4]	Used to statically analyse code and detect programming errors and bugs
Jest [5]	Used to design and run unit and integration tests.

Table 4: Used quality assurance tools

GitHub Actions is a tool for building CI/CD pipelines, developed by the GitHub team. This was a natural choice for creating the pipeline, since the source code is hosted on GitHub, and GitHub Actions was therefore easy to integrate into the repository.

The created pipeline is depicted in figure 1. In the first step of the pipeline, the source code in the repository is analysed for deviation from the style rules with Prettier. In the second stage, the code is linted with ESLint and the typescript files are transpiled into JavaScript for production. In the last quality assurance stage, unit and integrations are run with Jest.

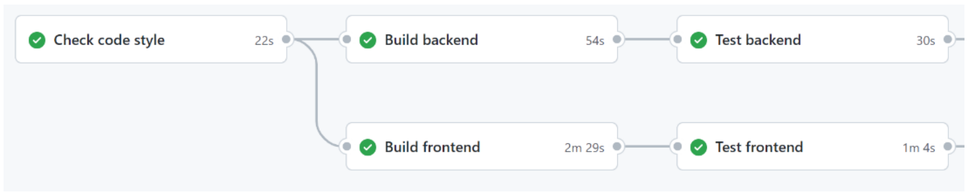

Figure 1: CI/CD pipeline

Prettier is a code formatter. It was introduced to achieve a uniform code style across the project, and to make it easier to enforce the code style agreed by the developers. The style rules were decided and configured at the start of the project. Prettier was also configured to run before each commit as a pre-commit hook in Git. This helps to ensure that all code pushed into the GitHub repository is properly formatted. As a fallback, the pipeline fails in the Check code style -phase if the style rules are not followed.

ESLint is a linting tool used for finding programming errors and bugs from source code through static analysis. Linting is especially useful in this project since the used languages and tools are not very familiar to a lot of the developers. ESLint plugins are utilized to achieve proper analysis for each part of the project. For example, the React application and Node.js server use the eslint-plugin-react and eslint-plugin-node plugins respectively.

Jest is a JavaScript testing framework. It was chosen because some developers already had experience with it and because Jest has good integration with React as both are developed by Facebook. React testing library is also used in conjunction with Jest to make it easier to test React components.

All tools are configured to not permit any errors, and to fail the CI pipeline immediately if errors are encountered in any phase. This way encountered potential bugs caused by failing tests or linting errors need to be fixed before the integration into the staging environment can be done. This strategy is adopted because permitting failing test cases or linting errors can be a path to ignoring linting errors completely or integrating broken functionality into the staging environment.

For static analysis and testing is performed on every push to the GitHub repository, reports are not saved anywhere. The emphasis on utilizing these tools is on detecting changing results caused by regressions or new code and reacting quickly to errors.

6 Open issues

Current open issues in the project include figuring out some timetables regarding testing, especially, when the customer gets to perform ad hoc and acceptance testing. These depend entirely on how long it takes to complete the implementation of the MVP, and while we are on a good speed with the implementation, surprises might arise. Another point to agree upon, is whether end users will use their own devices or if we can manage a loan test phone from Vincit. Finally, we should agree when to gather in a sauna evening with the project group. It has proven more difficult than expected.

7 References

[1] “GitHub” [Online]. Available: https://github.com. [Accessed 03.11. 2021].
[2] “GitHub Actions” [Online]. Available: https://github.com/features/actions. [Accessed 03.11 2021].
[3] “Prettier” [Online]. Available: https://prettier.io/. [Accessed 03.11. 2021].

[4] “ESLint” [Online]. Available: https://eslint.org/. [Accessed 03.11. 2021].
[5] “Jest” [Online]. Available: https://jestjs.io/. [Accessed 03.11. 2021].

APPENDIX A: Caption from *Get a Room!* Project Plan

From page 4/22:

1.2 Product and environment

The product's main goal is to make it easier for Vincit employees to quickly book a meeting room at the workplace once the need for a brainstorming session with colleagues arises. Additional features may include booking other calendar resources, such as flexible office desks or even company cars.

Get a Room! client application is a PWA-compatible React app that runs on a browser, user’s mobile or desktop device. Current supported browsers are Windows Edge and Mac/iOS Safari, 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.

Authentication happens via Vincit SSO, which is based on Google SSO. After authentication, user gets a valid authentication token, which allows access to Vincit’s instance of Google’s Workspace API, providing resource management and access. The API is used to check for available rooms and make resource reservations.

The products domain name is "oispahuone.com". The React client is served to users from Nginx Cloud Run service, hosted on Google Cloud platform. There’s also a Node.js server hosted from Google Cloud Platform, and it is used to save client data to a MongoDB database and to make queries to the Google Workspace.

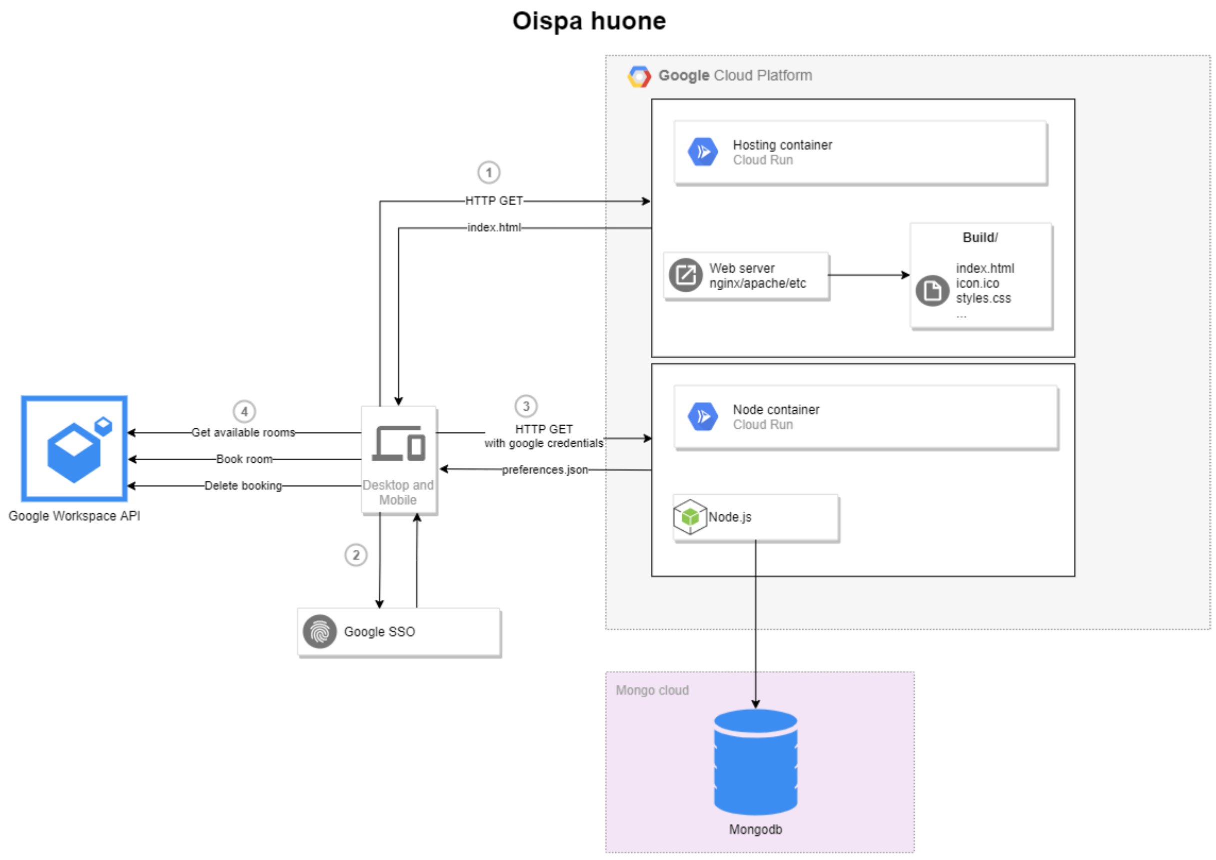

Figure 1: *Get a room!* high level concrete architecture.

Figure 1 presents the high-level component architecture with the planned technology stack. The end user downloads the app to their mobile or desktop device, signs in with their Google id via Vincit's SSO server, after which the application connects to Google Cloud Platform to fetch and save user preferences and reservable resources. As preferences are saved in cloud, settings are synced automatically between user's devices.

APPENDIX B: Usability testing plan

Tarkoitus, tavoite ja päämäärä

Tämä dokumentti on suunnitelma sovelluksen ’Get A Room!’ käytettävyystestaukselle.

Varmistetaan, että tuote vastaa asiakkaan vaatimaa käytettävyystasoa. Varmistetaan, että käyttäjät pitävät tuotetta opittavana, ovat tyytyväisiä siihen, pitävät sitä tehokkaana ja että käyttäjälle tulee vastaan vähän virheitä, kun sovellusta käytetään.

Kerätään dataa, jolla voidaan parantaa sovelluksen käytettävyyttä kokonaisvaltaisesti.

Tutkimuskysymykset

Mitä tutkimuskysymyksiä pyritään vastaamaan tämän testauksen yhteydessä.

- Tuottaako sovelluksen asentaminen ja siihen kirjautuminen ongelmia (onboarding process)?
- Onko Päänäkymä ymmärrettävä?
- Kuinka nopeasti osallistuja pystyy varaamaan itsellensä huoneen?
- Kuinka nopeasti osallistuja pystyy muokkaamaan omaa varausta (lisäaika + poistaminen)?
- Löytyykö selkeitä käytettävyysongelmia? (selkeitä käytettävyysongelmia = ei ole helposti opittava, ei ole tehokas, vastaan tulee paljon virhetilanteita)
- Tuleeko osallistujalta kysymyksiä testauksen yhteydessä?
- Mitä mieltä osallistuja on sovelluksesta?
- Jos osallistuja vertailee sovellusta google kalenteriin, miltä käytettävyys tuntui siihen verrattuna?

Osallistujat

Testattavat henkilöt ja heidän roolinsa yrityksessä.

- Vincit työntekijät 3–5 henkilöä, jotka eivät ole nähneet sovellusta aiemmin.
- Mahdollisimman erilaisia rooleja (esim. Solun vetäjät, People (HR), Asiakkuusihminen, Myyntihenkilö, Designer, Devaajia)

Testimetodi

Metodi:
Kyseessä on käytettävyystesti, jossa tarkkaillaan osallistujan navigointia sovelluksessa ’Get A Room!’, ennalta määriteltyjen tehtävien kanssa. Tarkkailun yhteydessä kerätään testidataa, joka kertoo, miten hyvin osallistuja pystyi suoriutumaan tehtävistä. Myös kvalitatiivista dataa

kerätään, joka kertoo osallistujan käyttökokemuksesta kokonaisuudessa. Tarkkailuun osallistuu fasilitaattori ja tarkkailija, joka kerää muistiinpanoja.

Kesto:
~20 min

Ennen testiä (2min):

- Esittelyt
- Tämän istunto nauhoittaa ja kuvataan, onhan se ok?

Intro (2min):

- Mikä sovellus kyseessä ja mitä varten
- Onko osallistuja osallistunut ennen käytettävyystestaukseen?
- Fasilitaattorin rooli + Tarkkailijan rooli
- Mitä tapahtuu loppusession aikana
- ”Ajattele ääneen” metodin läpikäynti

Taustahaastattelu (1min):

- Käytätkö Google Kalenteria varaamaan neukkareita?
    - Miten usein?
    - Puhelimella vai tietokoneella?

Tehtävät (10min):

- Osallistuja suorittaa tehtävät, jotka on annettu tehtävälistaosiossa.

Tehtävien jälkeen (5min):

- Mahdollisia jatkokysymyksiä
- Mahdollisia kysymyksiä liittyen tiettyyn käytettävyysongelmaan, jonka osallistuja/fasilitaattori huomasi

Tehtävälista

Tehtävät mitä osallistuja suorittaa testin aikana

Tehtävä	**Aloitustila**	**Milloin tehtävä on suoritettu onnistuneesti?**	**Huomioitavaa**
Pre - Onboarding process -Asentaminen ja kirjautuminen, asetukisien laittaminen	Sovellus avataan ensimmäistä kertaa.	Sovellus asennetaan, kirjaudutaan sisälle ja asetetaan oma toimisto.	- iOS vs. Android - Onko kirjautuminen vaivatonta? - Onko oman toimiston asettaminen helppoa?
1. Selvitä löytyykö toimistollasi yhtään neukkaria, jotka ovat vähintään 60 minuuttia vapaana.	Päänäkymä, jossa vapaat huoneet, ei varattua huonetta.	Osallistuja avasi keskustelun löytyikö huone, joka on vähintään 60min vapaana.	- Miten nopeasti osallistuja löytää huoneita, jotka ovat ainakin 60 minuuttia vapaana?
2. Varaa itsellesi neukkari, jonne mahtuu ainakin 4 ihmistä ja jossa sijaitsee televisio. Varaa huone itsellesi 30 minuutiksi.	Päänäkymä, jossa vapaat huoneet, ei varattua huonetta	Varaus on onnistunut, kun se ilmestyy Päänäkymään: 'Your Booking' otsikon alle	- Miten nopeasti osallistuja pystyy varaamaan huoneen? - Monellako painalluksella osallistuja pystyi varaamaan huoneen?
3. Haluat lisätä juuri varaamasi neukkariin lisäaikaa. Lisää huoneeseen 30 minuuttia lisäaikaa.	Päänäkymä, jossa varattu huone, vapaat huoneet ei varattavissa.	Kun varausta on jäljellä 59/60minuuttia.	- Ymmärtääkö osallistuja, että lisäaikaa voi laittaa vain 15min kerrallaan? - Nopeus (ajallisesti)
4. Varaamasi huone onkin väärä huone, ja haluat varata huoneen, jossa on sohva. Suorita varaus uudelle huoneelle 30 minuutiksi.	Päänäkymä, jossa varattu huone, vapaat huoneet ei varattavissa.	Vanha varaus poistetaan onnistuneesti ja uusi varaus tehdään samalla tavalla kuin kohdassa 2.	- Oliko huoneen poistaminen intuitiivista? - Oliko uuden varauksen tekeminen nopeampaa kuin ensimmäisellä kerralla? - Nopeus (ajallisesti) - Helppous (painallusta)

Testiympäristö & laitteisto

Testiympäristö

- Vincitin toimisto Hermiassa. Neukkari, jossa vähintään yksi pöytä.

Laitteet

- Mobiilipuhelin
- Nauhuri (Mikki)
- Sekuntikello
- Fasilitaattorin/Tarkkailijan läppäri – huone, jossa pistorasia.
- Dokumenttikamera

Fasilitaattori

Fasilitaattori: Sara Brentini

Fasilitaattori istuu huoneessa osallistujan kanssa session aikana. Hän aloittaa session, pitää lyhyen taustahaastattelun ja tämän jälkeen fasilitaattori esittelee tehtävät tarpeen mukaan. Testauksen aikana fasilitaattori voi kysyä jatkokysymyksiä selventämään osallistujan käyttäytymistä, odotuksia, ajatuksia ja kommentteja.

Kerättävä testidata

Mitä dataa kerätään ja miten sitä kerätään

Tutkimuskysymys	Kerättävä data
- Tuottaako sovelluksen asentaminen ja kirjautuminen ongelmia – onboarding process?	- Mahdollisten virheiden määrä - Aika miten nopeasti osallistuja kirjautuu ja pääsee päänäkymälle.
- Onko Päänäkymä ymmärrettävä	- Onko aakkosjärjestys looginen - Voiko toimiston asetuksen vaihtaa vaivattomasti
- Miten nopeasti osallistuja pystyy varaamaan itsellensä huoneen?	- Ajallisesti - Kuinka monta painallusta osallistuja tekee, jotta varauksen saa tehtyä - Tuliko virhetilanteita - Herättääkö 30|60 min varausaika kysymyksiä tai huomioita?

- Miten nopeasti osallistuja pystyy muokkaamaan omaa varausta (lisäaika + poistaminen)?	- Ymmärtääkö osallistuja, miten poistetaan varaus? - Ymmärtääkö osallistuja, miten lisätään lisäaikaa - Herättääkö 15min lisäaikaa kysymyksiä?
- Löytyykö selkeitä käytettävyysongelmia?	- Pysähdysten määrä, jossa osallistujan pitää miettiä miten pääsee tehtävässä eteenpäin. - Virhetilanteiden määrä kokonaisuudessa - Ajallinen ero, kuinka kauan meni ensimmäisen ja toisen varauksen aikana (onko opittava) - Pystyykö osallistuja toteuttamaan varauksen hänen haluamansa tavalla (mahdollisuus kerätä kehitysideoita)
- Tuleeko osallistujalta kysymyksiä testauksen yhteydessä?	- Kysymyksien määrä ja tyyppi
- Mitä mieltä osallistuja on sovelluksesta?	- Osallistujan avoimia mielipiteitä - Helppous kokonaisuudessa - Opittavuus kokonaisuudessa - Helppous asentaa ja kirjautua - Ovatko käytetyt termit sovelluksessa ymmärrettäviä
- Jos osallistuja vertailee sovellusta google kalenteriin, miltä käytettävyys tuntui siihen verrattuna?	- Osallistujan avoimia mielipiteitä - Preferenssi Get a Room! Vs. Google Calendar – miksi?