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

Hive Group - 1

Geocaching.fi front-end

Test Report

Members:

439860, Matias Kaakinen
k56521, Harri Hietanen
151304859 , Andrey Essaulov
H283635, Juho Keto-Tokoi
150359915, Melany Mendoza Romero
K65167, Mikko Kaukonen
K99006, Nelli Leinonen

Version history

Ver-sion	Date	Author	Description
0.1	15.03.2023	Matias Kaakinen	First draft
0.2	30.03.2023	Matias Kaakinen	Part 1 started
0.3	14.04.2023	Matias Kaakinen	Parts 1-3 finished
0.4	28.04.2023	Matias Kaakinen	Part 4 started
1.0	05.05.2023	Matias Kaakinen	Part 4 + Appendix finished

Contents

1 Introduction

1.1 Purpose and scope of document

This document describes testing practices of the Geocaching.fi mobile website's development process. It shortly describes automated testing with the main focus on manual testing practices.

1.2 Product and environment

Geocaching.fi website's UI is created by using React with TypeScript and Express.js. The current implementation also includes a mock back-end which will be replaced by the customer's own implementation during release.

All testing is performed solely on the front-end.

1.3 Project constraints related to testing

A test account with premium access was needed for full access to the website.

As the final state of the back-end will remain out of the project's scope, all testing data must be mocked.

Hosted environment was needed for testing on an actual mobile device.

The mobile device itself was also required.

1.4 Definitions, abbreviations and acronyms

Use case	scenario that mimics users' behaviour
Test	testing of one requirement, functionality, or feature
Test case	everything required to one test
Test suite	collection of related tests

2 Testing process

Project used ESlint to guide coding style. ESlint is a tool that searches through ECMAScript/JavaScript source code to find predefined patterns of unwanted coding practice. It will then list any found cases in a report. ESlint configurations were maintained by the lead developer.

2.1 General approach

Testing was done in hierarchical layers described in the Table 1. Each layer was required to pass before moving to the one below it.

As of 5.5.2023, no Acceptance testing is yet to be scheduled by the customer.

Test layer	Approach
Unit	Done separately by each individual developer. Uses Jest and React Testing Library for testing React components. Use of TDD (Test Driven Development) method was encouraged.
Code quality	Done by the CI/CD pipeline. ESlint pattern check-ups are written by the lead developer and run automatically on each pull request.
Integration	Done by the CI/CD pipeline. Uses Jest and React Testing Library for testing inter-componential relationships. Tests were written by the development team, while the QA provided assistance in test design.
Peer review	Done manually by the development team. Review assures that the incoming changes are semantically sensible, maintainable, and properly documented.
Exploratory testing	Done manually by QA/Project manager/developers after each sprint. Involved roaming around with the UI and searching for defects, with the focus on fresh features. Any found defects where then added to the backlog as tasks.
Usability	Done manually by the QA manager and Project Manager during the Beta testing period. Involved both separate testing of individual functionalities and testing of a complete user session.

Acceptance	Defined by the development team in co-operation with the customer. Performed by the customer at the end of the project. Customer may involve an end-user in acceptance testing.

Table 1 - Test process hierarchy

Defects/Bugs are also classified in hierarchy, described in Table 2. A feature must have no major or middle level defects and no more than 2 minor level defects

Defect level	Definition
Major	Defect disables a use case, completely eliminating the UX
Middle	Defect disturbs a use case, moderately degrading the UX
Minor	Defect disturbs a use case, marginally degrading the UX

Table 2 – Defect level hierarchy

2.2 Testing roles

Test roles are defined in Table 3:

Role	Definition
QA manager	A person in charge of defining the Test Plan and writing the Test Report. Due to size of the project, also designed integration and unit tests for the developers and performed usability tests in co-operation with other personnel.
Project manager	A person in charge of the project. Knows the requirements the best. Performed exploratory testing during every sprint and helped QA in usability testing.
Lead developer	A person in charge of code quality and architecture. Configured the CI/CD pipeline. Defined the patterns by which the ESlint tool inspects incoming code changes. Also participated as a developer.
Developer	A person in charge of new features. Wrote and performed tests for their own features and inspected incoming changes in peer reviews.
Customer	A person that is not included in the project team and performs the Acceptance

	testing. May involve an End-user in the process.
End-user	A person that is not included in the project team and performs usability testing on their own device.

Table 3 – Testing roles

2.3 Test schedule

As the project used a CI/CD pipeline, no scheduling was needed for unit or integration testing, or code check-ups.

Usability testing was planned to be performed followingly:

1. Alpha testing: Usability testing of the first viable state of the product. Will involve the End-user.
2. Beta testing: Usability testing of the refined state of the product. Will involve the End-user.
3. Acceptance testing: Final testing of the finished product that has no open issues. Will involve the Customer. May involve the End-user if Customer sees fit.

No dates were scheduled in the plan as contact with an end-user and a proper hosting environment was needed to be made first.

In practice the Test Plan did not hold completely. No hosting environment was acquired until the later sprints of the project, and thus no End-user was included in the testing.

A separate Alpha testing period was not held, as conditions for Beta testing were met rather quickly. Undocumented exploratory testing was performed at the end of every sprint to find defects. Any finds were then listed as issues on GitHub backlog.

Beta testing period was held between 5.4.-1.5.2023 (including sprints 4-6), on the most recent implementations of the Geocaching.fi mobile website. Both QA and the Project manager were involved. Results are included in this document.

2.4 Test documentation

CI/CD pipeline stores reports of every test and code check-up included. They can all be reviewed on GitHub. A screenshot of a Test Pipeline run is provided at the end of this document (Picture 3).

A separate test diary was being kept by the QA manager during the Beta testing period. Results are integrated into this document.

3 Testing Tools

Unit/Integration testing uses React Testing Library as the main testing tool.

ESlint performs code check-ups.

Usability testing was done manually on an actual user device.

CoverAlls was planned to be included for test coverage analysis, but no subscription was done in practice and the tool was abandoned. The reason for the abandonment was the cost of the service.

Project had no need for keeping any statistics.

4 Test cases and results

4.1 Test results

Overall the project's test performance has been sufficient.All required front-end functionalities have passed both automated and manual testing. React Testing Library has provided an easy framework for scripting user-like behaviour. More specific tests can of course always be implemented to increase test coverage.

Project included no real end-to-end system testing, as the back-end is only mocked and any dependable system testing would require its complete integrity.

Manual usability testing was able to cover all the required use cases, because the project's scope was so small and the requirements well documented. When the UI is expanded by additional components in the future, more use cases must be invented. Some use cases included in the document already cover functionalities that were not part of the requirements but are included for future Quality Assurance.

4.2 Unit and Integration testing

Unit and integration testing was performed throughout the implementation process and no feature was accepted unless it passed all automated tests.

A total of 13 Test Suites, including 26 individual tests, was added to the Pipeline by the developers. Total Test Coverage of code lines in the latest Pull Request, provided by Jest reporting tool, was 72.7 %. The aimed test coverage, described in the Test Plan, was 80 %, which was not achieved.

The reason for not achieving the goal was two-fold:

- Improper definition: defining test coverage for a React project was more complicated than predicted. As React uses JSX, which consists of both functional code and HTML, it was hard to define what exactly needs to be covered.
- Lack of time: short duration of the project forced the developers to concentrate on implementing features instead of more specific tests.

All pipeline test reports are found on GitHub and snapshots of a single report are provided at the end of this document (Pictures 1 & 2).

4.3 Usability testing

In the context of this project, Usability tests were manual UI test that followed user-like scenarios. They were performed on each separate Page, including each functional component.

Some tests were linked, in which case a description of what has already been done is provided.

Tests used artificial data provided by the mocked back-end.

A total of 32 tests was performed, of which 7 were not part of the requirements or could not resolve properly without a complete back-end. From the 25 required tests, 6 failed. None of the failings were results of conflicting bugs, only being results of non-optimal implementations that should be taken account in the future development.

Both use cases and results are reported, with the reports following a Given/When/Then pattern:

1. Describe the setting (Given)
2. Describe the action (When)
3. Describe the expected outcome (Then)
4. Describe the result (Success/Failure)

4.4 Special testing

All new features were tested on all major browsers to assure compatibility.

Mobile friendliness was tested on an actual mobile device.

4.5 Acceptance testing

Acceptance testing has passed when customer has accepted the following:

- At least 90 % of all functionality requirements are completed.
- Implemented design is the same as the agreed one, or as close as possible.
- It is possible to connect the front-end implementation (provided by the development team) with the back-end implementation (provided by the customer), following the technical documentation.

No acceptance testing is performed by the customer as of 05.05.2023.

4.6 Open issues

All known open issues (including bugs) are as follows:

- (Usability problem) All user-readable content is hard-coded, and thus the website does not support multi-lingual configurations. A solution would be to implement all content as variables, whose values are fetched based on the language in use.

- (Usability problem) The geocache information view is unfinished, lacking visual styles and some of the data.
- (Bug, minor) “Center map to the user” -button in the map page an upredictable delay, which is caused by *navigator.geolocation.getCurrentLocation()* function. This could potentially be fixed by changing some of the parameters.

5 APPENDIX A […Z]

Geocache.fi usability testing results

5.1 Main Page

5.1.1 Caches

- **Given**: A user that wants to read detailed information about a single cache
- **When**: The user clicks on the cache’s box
- **Then**: The user should be redirected to the cache’s information view
- **Result**: Passed

- **Given**: A user that wants to load more caches
- **When**: The user clicks on the ”LoadMoreCaches” button
- **Then**: More caches should be loaded into the View
- **Result**: Failed
- **Note:** Not part of the requirements

5.1.2 Community Meetings

- **Given**: A user that wants to read detailed information about a single community meeting
- **When**: The user clicks on the meeting box
- **Then**: A meeting information view should be rendered
- **Result**: Failed
- **Note:** Not part of the requirements

- **Given**: A user that wants to load more upcoming community meetings
- **When**: The user clicks on the ”LoadMoreMeetings” button
- **Then**: More meetings should be loaded into the View
- **Result**: Failed
- **Note:** Not part of the requirements

5.2 Authentication

5.2.1 General

- **Given**: A user that is not logged in and wants to open the Authentication overlay
- **When**: The user clicks the Navigation bar's ”User” icon
- **Then**: The “LogInOrRegister” button should be rendered
- **Result**: Passed

- **Given**: A user that is not logged in and has clicked the “User” icon
- **When**: The user clicks the “LogInOrRegister” button
- **Then**: The Authentication overlay should be rendered
- **Result**: Passed

- **Given**: A user that is logged in and wants to see their information
- **When**: The user clicks the Navigation bar's ”User” icon
- **Then**: A user information overlay should be rendered
- **Result**: Passed

- **Given**: A user that is logged in and wants to log out
- **When**: The user clicks the ”LogOut” button
- **Then**: Feedback from a successful log out should be rendered
- **Result**: Failed

5.2.2 Register

- **Given**: A user that wants to register and has opened the Authentication overlay
- **When**: The user clicks the ”ToRegistration” text
- **Then**: A register form should be rendered
- **Result**: Passed

- **Given**: A user that tries to register a reserved username
- **When**: The user types a register a reserved username and clicks the ”Register” button
- **Then**: Feedback from failed register, due to reserved username, should be rendered
- **Result**: Passed

- **Given**: A user that tries to register an email that is already in use
- **When**: The user types an email that is already in use and clicks the ”Register” button
- **Then**: Feedback from failed register, due to email already in use, should be rendered
- **Result**: Failed

- **Given**: A user that tries to type wrong confirmation password
- **When**: The user types wrong password to the password confirmation box
- **Then**: Feedback from non-matching passwords should be rendered
- **Result**: Passed

- **Given**: A user that tries to register an unreserved username, a new email and a confirmed password
- **When**: The user has typed correct information and clicks the ”Register” button
- **Then**: Feedback from a successful registration should be rendered
- **Result**: Failed

5.2.3 Log In

- **Given**: A user that tries to log in and has forgotten the password
- **When**: The user clicks the “PasswordForgotten” text
- **Then**: The password recovery form should be rendered
- **Result**: Failed
- **Note:** Not part of the requirements

- **Given**: A user that tries to log in with a wrong username-password combination
- **When**: The user types the wrong username-password combination and clicks the ”LogIn” button
- **Then**: Feedback from a failed log in attempt should be rendered
- **Result**: Passed
- **Note**: The feedback message is not descriptive.

- **Given**: A user that tries to log in with a correct username-password combination
- **When**: The user types the correct account-password combination and clicks the ”LogIn” button
- **Then**: The user should be redirected to the Main Page
- **Result**: Passed

5.2.4 Traverse to Map Page

- **Given**: A user that wants to see the Map Page
- **When**: The user clicks on the “BurgerMenu” button
- **Then**: A menu overlay should open from the side
- **Result**: Passed

- **Given**: A user that wants to see the Map Page and has opened the menu overlay
- **When**: The user clicks on the “Caches” accordion
- **Then**: A “Map” option should be rendered
- **Result**: Passed
- **Note:** Not very intuitive traverse solution. The accordion title should be more describing, ie. “Views”

- **Given**: A user that wants to see the Map Page and can see the “Map” option
- **When**: The user clicks on the “Map” option
- **Then**: The user should be redirected to the Map Page
- **Result**: Passed

5.3 Map Page

5.3.1 Map Visualisation

- **Given**: A user that wants to change the Map Visualisation
- **When**: The user clicks on the “MapOption” button
- **Then**: A map option overlay should be rendered
- **Result**: Passed

- **Given**: A user that wants to change the Map Visualisation and can see the map option overlay
- **When**: The user clicks on a map visualisation option
- **Then**: The map should be re-rendered in different style
- **Result**: Failed
- **Note:** Not part of the requirements

5.3.2 Map Page Buttons

- **Given**: A user that wants to know the role of the “Filter” button
- **When**: The user hovers mouse over the “Filter” button
- **Then**: A textbox explaining the role of the button should be rendered
- **Result**: Failed

- **Given**: A user that wants to know the role of the “Map” button
- **When**: The user howers mouse over the “Map” button
- **Then**: A textbox explaining the role of the button should be rendered
- **Result**: Failed

- **Given**: A user that wants to know the role of the “Location” button
- **When**: The user howers mouse over the “Location” button
- **Then**: A textbox explaining the role of the button should be rendered
- **Result**: Failed

5.3.3 Cache Information

- **Given**: A user that wants to read information about a single cache
- **When**: The user clicks on the cache’s icon
- **Then**: A cache information pop-up window should be rendered
- **Result**: Passed

- **Given**: A user that wants to hide a cache information pop-up window
- **When**: The user clicks anywhere outside of the pop-up window
- **Then**: The pop-up window should disappear
- **Result**: Passed

- **Given**: A user that has clicked a cache’s icon and wants to read more specific information about the cache
- **When**: The user clicks on the “MoreInformation” button
- **Then**: The user should be redirected to the cache’s information view
- **Result**: Passed

- **Given**: A user that has been redirected to a cache’s information view and wants to return to the map view
- **When**: The user clicks on the “Go back” button on the browser
- **Then**: The user should be redirected to the Map View with the previously set filters in use
- **Result**: Failed
- **Note:** Not part of the requirements

5.3.4 Cache Visibility

- **Given**: A user that wants to see caches from a specific region

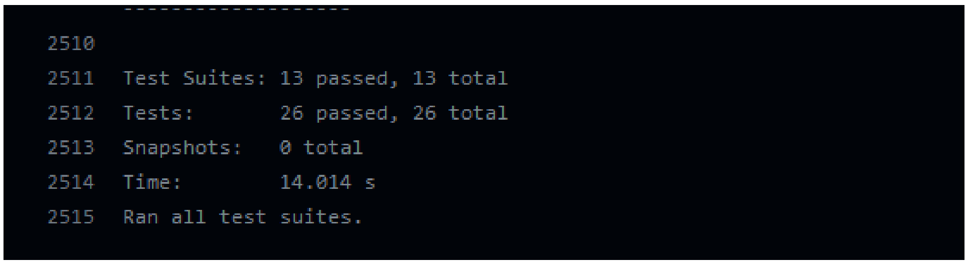
- **When**: The user has zoomed the map on the region and clicks on the ”Search-FromTheRegion” button
- **Then**: The map should only render caches belonging to that region
- **Result**: Passed

- **Given**: A user that wants to review the filters in use
- **When**: The user clicks on the Filter icon box
- **Then**: Filter overlay should appear from the left of the screen
- **Result**: Passed

- **Given**: A user that has opened the Filter overlay and wants to reset the filters in use
- **When**: The user clicks on the ”Reset” button
- **Then**: All filters should return to the default state
- **Result**: Passed

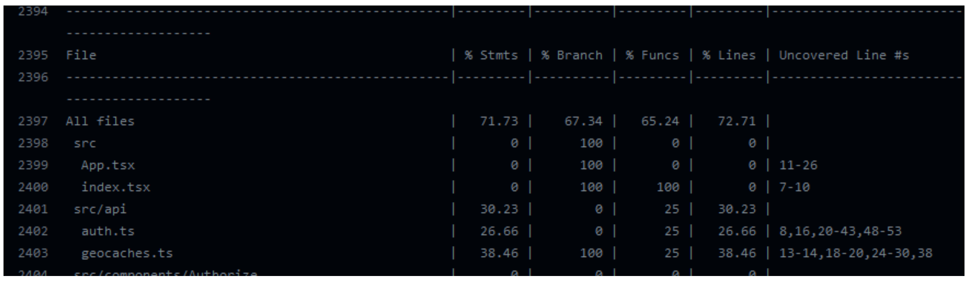
- **Given**: A user that has opened the Filter overlay and wants to view unfound caches
- **When**: The user clicks on the “OwnOption” accordion, chooses “Unfound” from the dropdown and clicks on the “UseFilters” button
- **Then**: The map should only render caches that have the “Unfound” trait
- **Result**: Failed
- **Note:** Result is not due to an improper implementation, but lack of functionality in the mocked back-end.

Geocache.fi automated testing



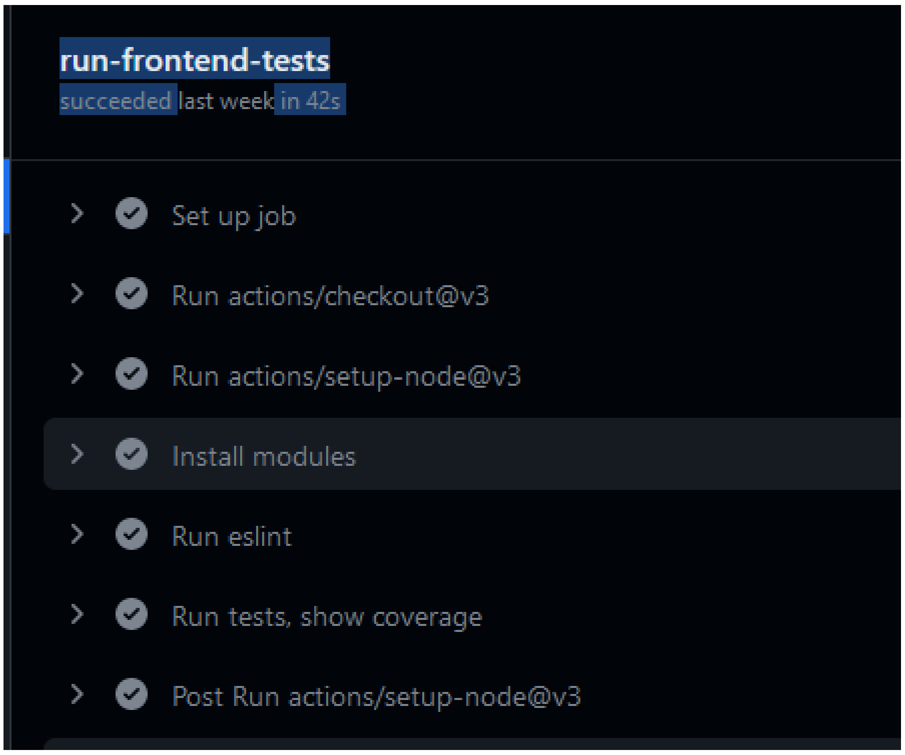
```
--------------------

Test Suites: 13 passed, 13 total
Tests:       26 passed, 26 total
Snapshots:   0 total
Time:        14.014 s
Ran all test suites.
```

Picture 1 - An example Test Run Report



```
------------------------------------------------|---------|----------|---------|---------|------------------------
------------------
File                                            | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
------------------------------------------------|---------|----------|---------|---------|------------------------
------------------
All files                                       |   71.73 |    67.34 |   65.24 |   72.71 |
 src                                            |       0 |      100 |       0 |       0 |
  App.tsx                                       |       0 |      100 |       0 |       0 | 11-26
  index.tsx                                     |       0 |      100 |     100 |       0 | 7-10
 src/api                                        |   30.23 |        0 |      25 |   30.23 |
  auth.ts                                       |   26.66 |        0 |      25 |   26.66 | 8,16,20-43,48-53
  geocaches.ts                                  |   38.46 |      100 |      25 |   38.46 | 13-14,18-20,24-30,38
 src/components/Authorize                       |       0 |        0 |       0 |       0 |
```

Picture 2 - An example Test Coverage report



run-frontend-tests
succeeded last week in 42s

- Set up job
- Run actions/checkout@v3
- Run actions/setup-node@v3
- Install modules
- Run eslint
- Run tests, show coverage
- Post Run actions/setup-node@v3

Picture 3 - An example Test Pipeline run