



TEST REPORT FOR VUE.JS

CT60A4160 Ohjelmistotestauksen periaatteet
Arttu Korpela
Joona Haikonen

Document history

Version	Date	Creator	Latest updates
T1	2020-09-18	Erno Vanhala	Template is created
T2	2023-10-16	Arttu Korpela & Joona Haikonen	Vue.js testing & evaluation records

Table of content

Introduction	4
Points proposal	5
Testing strategy	6
Environment and test tools	7
Installing Vue.js on MacOS and Ubuntu	8
Existing tests	9
Written tests	10
Test cases	12
Functional tests	13
Non-functional test	15
Other tests	18
Bug report	21
Risks report & Conclusion	21
Vue.js upon extensive testing, demonstrates a high level of readiness for production use. The testing phase focused on assessing the framework's stability, performance, and usability. Vue.js proved to be a robust and reliable choice. Testing was carried out by covering various aspects of the framework.	21
Documentation stood out as very extensive and clearly articulated which benefits beginners as well as more experienced developers. Additionally, while the framework itself is stable it is essential to stay aware about potential issues with third-party plugins and libraries often used with Vue.js.	21
In conclusion Vue.js' stability, performance, and features align with industry standards, making it a dependable choice for web development projects. The overall state of the project is positive, with Vue.js passing various testing scenarios and emerging as a strong contender for building web applications.	21

Introduction

This course project focuses on the evaluation of Vue.js, which is a progressive JavaScript framework for building user interfaces and web applications of wide variety. Vue.js has gained popularity for its simplicity compared to its alternatives, flexibility and scalability.

In this project our objective is to conduct both functional and non-functional testing on Vue.js, the framework itself to assess its robustness, reliability and usability. Functional testing is done to ensure that the core features and functionality work as expected. On the other hand, non-functional tests will cover things such as documentation, ease of installation, load times and usability.

Points proposal

<Describe what tasks you have carried out and how many points you should get from each task>

Ominaisuus	Toteutus + Toteuttaja	Maksimipisteet
Hyvin kirjoitettu raportti		10
Ohjelmiston asennus on käyty läpi yhtenä testitapauksena	P-1 testitapaus käsittelee asennusta testitapauksena Joona Haikkonen & Arttu Korpela	2
Ohjelmiston asennus on suoritettu ainakin kolmeen erilaiseen laitteeseen (esim. Ubuntu, MacOS, Windows, Android, iOS... kolmea eri Windows-konetta ei lasketa eri laitteiksi)	Asennus suoritettu MacOS (Ventura) sekä Ubuntulla (20.04.6) Joona Haikonen	1
Ohjelmisto on testattu saavutettavuuden näkökulmasta; voiko sitä käyttää pelkästään näppäimistöllä? Ääniohjattuna? Osaako ruudunlukija lukea ohjelmiston käyttöliittymää?	Testattu ääniohjauksella. Saatu toimimaan, mutta käytettävyyden kannalta vaatisi Vuelle tarkoituksella tehdyn command konfigin. Joona Haikonen	3
Suunnittele 10 funktionaalista testitapausta (esim. "Käyttäjän tulee kyetä tallentamaan tiedosto") ja testaa ne	Yhteensä kymmenen testitiedostoa, joissa jokaisessa vähintään kaksi testia. Myös viisi testitiedostoa on muutettu cypress:llä toimivaksi. Arttu Korpela	4
Suunnittele 10 ei-funktionaalista testitapausta (esim. "Ohjelmiston tulee toimia nopeasti", miten määrittelet "nopeasti"? ja testaa ne	Yhteensä kymmenen eri testitapausta yleisimmistä usecaseista. Joona Haikonen	4
Projektin olemassa olevat (yksikkö)testit on ajettu, jotta ohjelman (ja testien) tila on saatu varmistettua	Kaikki 2744 yksikkötestiä ajettu ja läpäisty. Arttu Korpela	3

Pull request uusille (yksikkö)testeille (yksi riittää)	Vaikka yksikkötestit ovat hyviä, eivät ne noudata projektin tyyliohjetta palautukseen tai koodiin. Voin laittaa pull-requestin menemään, mutta sitä ei hyväksytä. Arttu Korpela	4
Automatisoi jotain testitapauksia projektissa esimerkiksi https://www.cypress.io/ :lla (vähintään 5 tapausta täytyy rakentaa)	Viisi ensimmäistä testitiedostoa on muutettu toimimaan Cypressillä. N.10 testitapausta. Arttu Korpela	5
Dokumentaation tarkistus: Onko se ajantasainen? Puuttuuko jotain?	Dokumentaatio on erittäin kattava ja tarjoaa hyvin tietoa kehittäjälle. Joona Haikonen	2
Työssä on käytetty projektin omia testaustyökaluja, sekä yleisesti hyödynnetty useaa työkalua testien toteuttamiseen.	Vitest ja Vue:n omat dev-työkalut ovat olleen funktionaalisen testauksen pohjana. Myös Cypress:iä on hyödynnetty testien toteutukseen. Arttu Korpela	3

Testing strategy

For functional testing we are mostly doing black box unit testing of the foundational features and behaviours of Vue.js. This is because the underlying code below even a simple feature can be very complex. We can also expect these features to work because they serve as the base for millions of websites. So, the testing strategy mainly involves around validation of these features with the expectation of them being correct every time.

Our approach to non-functional testing was mainly centered around usability, where we placed a strong emphasis on evaluating the framework from a user experience and development perspective. Additionally, we included a couple of performance tests to ensure the framework's responsiveness. Most of these tests were conducted as grey box testing, as we used the Vue.js documentation, which provided insights into the framework's inner workings.

The functional testing follows a similar template. A simple component is created and a spec.js test file to match the component. The test file has two tests: One to check a simple behaviour

and one to check a more complex scenario. All ten functional tests will be done using the Vue/test-utils and vitest. The first five test will also be repeated in cypress.

The non-functional tests encompass most common use cases and situations to ensure that the application built with Vue.js or Vue.js itself performs as expected.

Environment and test tools

The functional testing was done on Windows 11 using Visual Studio Code as an IDE to manage the project. Node.js and NPM were extensively used throughout the project, from installation to the actual running of the project and tests.

The installation process starts with the installation of Node.js. Node.js is an open-source cross-platform runtime environment written in JavaScript. It is built on Chrome's V8 JavaScript engine, which parses and executes the JavaScript code. We first used the installation tool to also install the NPM package manager.

NPM is a library and registry for JavaScript software packages. NPM also has command-line tools to help you install the different packages and manage their dependencies. We will use this installation tool shortly.

Now for the actual installation we will mostly use the Vue QuickStart guide:

<https://vuejs.org/guide/quick-start.html>

- First, we must use the installed NPM package manager to start a Vue project. Typing the following to our command line to start the project scaffolding tool.

```
> npm create vue@latest
```

- Next, we define our projects name and what supporting technologies we might want to add. We give the name to our project, add Vitest and Cypress for testing.
- Next, we head over to our projects folder and install our packages.

```
> cd <your-project-name>
> npm install
> npm run dev
```

- Now we have our first single page running and we can head over to it by copying the provided IP address.

For testing we are using the tools that we installed during the installation of Node.js. These include Vitest and Cypress that will be used to conduct our functional tests. Also, Vue's dev-tools were used (mostly just mount). The Vitest tools used were it, describe and expect. They were used to mostly confirm the functionality of Vue components. Cypress has a very similar syntax to vitest, so for learning purposes, a couple of the tests have been converted to cypress tests.

Why were these testing tools chosen? The reason is strong support embedded into the source libraries. The Vue has a clear testing ecosystem that is relatively easy to grasp and learn. From

not knowing what Vue is to being able to make unit test only took a couple of hours. This is the reason Vitest and Cypress were chosen as they are THE tools for Vue testing.

Installing Vue.js on MacOS and Ubuntu

Installing Vue.js on Ubuntu

Step 1: Install Node.js and npm

- 1 - Open your terminal by pressing **Ctrl + Alt + T** or right click on desktop and choose terminal
- 2 - Update your package list to ensure you have the latest information about available packages:
`> sudo apt update`
- 3 - Install Node.js and npm using the following command:
`> sudo apt install nodejs npm`
- 4 - Verify the installation by checking the installed versions:
`> node -v npm -v`

Step 2: Install Vue CLI

- 1 - After installing Node.js and npm, you can install Vue CLI using npm:

```
> sudo npm install -g @vue/cli
```

This command installs the Vue CLI tool on your system.

- 2 - Verify the installation by checking the Vue CLI version:
`> vue --version`

Installing Vue.js on macOS

Step 1: Install Node.js and npm

- 1 - Open your terminal. You can find it in the utilities folder within the applications folder, or use Spotlight (Cmd + Space) to search "Terminal"
- 2 - Install Homebrew if you don't have it already. Homebrew is a package manager for macOS. It can be installed using the following command:
`> /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"`

- 3 - Once Homebrew is installed, use it to install Node.js and npm:
`> brew install node`

- 4 - Verify the installation by checking the installed versions:
`> node -v npm -v`

Step 2: Install Vue CLI

- 1 - After installing Node.js and npm, you can install Vue CLI using npm:

```
> npm install -g @vue/cli
```

This command installs the Vue CLI on your system.

- 2 - Verify the installation by checking the Vue CLI version:
`> vue --version`

Existing tests

Vue.js has an extensive library and different tools to test your own components and projects, but the program itself hundreds of unit-, integration- and E2E-tests. Being an open-source project with hundreds of contributors, the need for testing becomes critical as even a small change that isn't properly tested can affect millions of lives.

Below is a completed run of all the unit tests in the current main branch of Vue core:

```
Test Files 174 passed (174)
Tests 2744 passed (2744)
Start at 16:31:30
Duration 69.09s (transform 16.71s, setup 15.45s, collect 233.42s, tests 20.24s, environment 70.15s, prepare 83.06s)

PASS Waiting for file changes...
press h to show help, press q to quit
```

Written tests

Installation of the program is mandatory step to start the actual testing phase. The installation is made as the first test case.

P-1 Installation of Vue.js

- Description:
 - Test that the installation of Vue is clear and uncomplicated. Also test that the documentation is up to date on the installation.
- Steps:
 - Read documentation and seek the installation steps
 - Follow the provided steps for your OS
 - Confirm in the terminal that the latest Vue Cli has been installed
- Expected Outcome:
 - All OS's have Vue Cli installed without complications

ID	Tester	Feature being tested	Testing approach and type	Test result	Notes
P-1	J.H	Vue installation and the installation documentation	Black Box – Compatibility test	Pass	Windows, Ubuntu and MacOS installations succeeded, and the documentation was up to date

For functional test we are looking to validate the foundational features and behaviours of Vue.js. The test are mostly unit tests, but some dabble in the integration test territory. Below is list of ten features that we tested and short description of them.

F-1 Data Binding

- Description:
 - Test if Vue successfully binds data to the template.
- Steps:
 - Declare a reactive property in the component's data.
 - Use the property in the template.
 - Update the property and validate the change
- Expected Outcome: The template should reflect the initial and updated values of the property.

F-2 Computed properties

- Description:
 - Test if computed properties return the expected value based on reactive dependencies.
- Steps:
 - Declare a computed property that depends on one or more data properties.
 - Use the computed property in the template.

- Expected Outcome: The template should reflect the value of the computed property, and it should update when its dependencies change.

F-3 Watchers

- Description:
 - Test if watchers are triggered when the data, they're watching changes.
- Steps:
 - Declare a watcher for a data property.
 - Update the watched property.
- Expected Outcome:
 - The watcher's callback is invoked upon the data property's update.

F-4 Methods

- Description:
 - Test if Vue component methods can be executed and update component data.
- Steps:
 - Declare a method that modifies a data property.
 - Call this method.
- Expected Outcome:
 - The data property should reflect the changes made by the method.

F-5 Lifecycle Hooks

- Description:
 - Test if lifecycle hooks like created, mounted, etc., are called at the appropriate times.
- Steps:
 - Insert logging or other side-effects into various lifecycle hooks.
 - Instantiate the component.
- Expected Outcome:
 - The side-effects or logs appear at the expected stages of the component's lifecycle.

F-6 Conditional Rendering:

- Description:
 - Test if v-if, v-else-if, and v-else directives correctly control template rendering.
- Steps:
 - Create a template with elements using these directives.
 - Toggle the conditions.
- Expected Outcome:
 - Elements change on the conditions provided.

F-7 List Rendering:

- Description:
 - Test if v-for correctly renders a list of items.
- Steps:
 - Bind a list of items to a template using the v-for directive.
 - Update the list programmatically.
- Expected Outcome:
 - The DOM reflects the correct list items, including any changes.

F-8 Event Handling:

- Description:
 - Test if v-on or @ directives bind events to handlers correctly.
- Steps:
 - Attach an event, like a click event, to a DOM element using @click.
 - Add modifiers like .stop etc.
 - Trigger the event.
- Expected Outcome:
 - The associated method or expression is invoked.

F-9 Dynamic Binding:

- Description:
 - Test if bindings like v-bind work correctly for properties like class and style.
- Steps:
 - Use v-bind to bind a reactive property to a DOM element's style.
 - Update the bound property.
- Expected Outcome:
 - The DOM element's style updates to reflect the new value.

F-10 Component Props:

- Description:
 - Test if parent components can pass data to child components using props.
- Steps:
 - Declare a child component that accepts a prop.
 - Use the child component in the parent and bind data to the prop.
 - Update the bound data in the parent.
- Expected Outcome:
 - The child component receives and reflects the prop data from the parent.

Test cases

Functional tests

ID	Tester	Feature being tested	Testing approach and type	Test result	Notes
F-1	A.K	Vue.js Data Binding Properties	Black Box – Unit test	Pass	Tested data binding on component initialization and reaction to data changes
F-2	A.K	Vue.js computed data properties	Black Box – Unit test	Pass	The computed data properties are correct and react to changes.
F-3	A.K	Vue.js watched properties react to changes.	Black Box – Unit test	Pass	Watched count property on call back updates the messages correctly
F-4	A.K	Vue.js component methods can be executed and update component data.	Black Box – Unit test	Pass	The button presses correlate with the count data implying the methods are called appropriately
F-5	A.K	Vue.js lifecycle hooks like created, mounted, etc., are called at the appropriate times.	Black Box – Unit test	Pass	OnMounted and OnUpdated update the message property correctly.
F-6	A.K	Vue.js conditional rendering with v-if, v-else-if, and v-else work as intended.	Black Box – Unit test	Pass	The count property is reflected correctly on the message implying the conditions work.
F-7	A.K	Vue.js list rendering works using v-for	Black Box – Unit test	Pass	The original array was iterated over correctly and changes to data reflected in the output.

F-8	A.K	Vue.js @click event handling works with its modifiers .stop and .prevent	Black Box – Unit test	Pass	The .stop ends propagation up the DOM tree. .prevent activated the method instead of following the rref
F-9	A.K	Vue.js dynamic binding using v-bind to set a style property of an element	Black Box – Unit test	Pass	The style elements changes correctly following a button press.
F-10	A.K	Vue.js test parent component data passes onto the child component	Black Box – Unit test	Pass	The childMessage updates on data change implying data flows correctly

Vitest results:

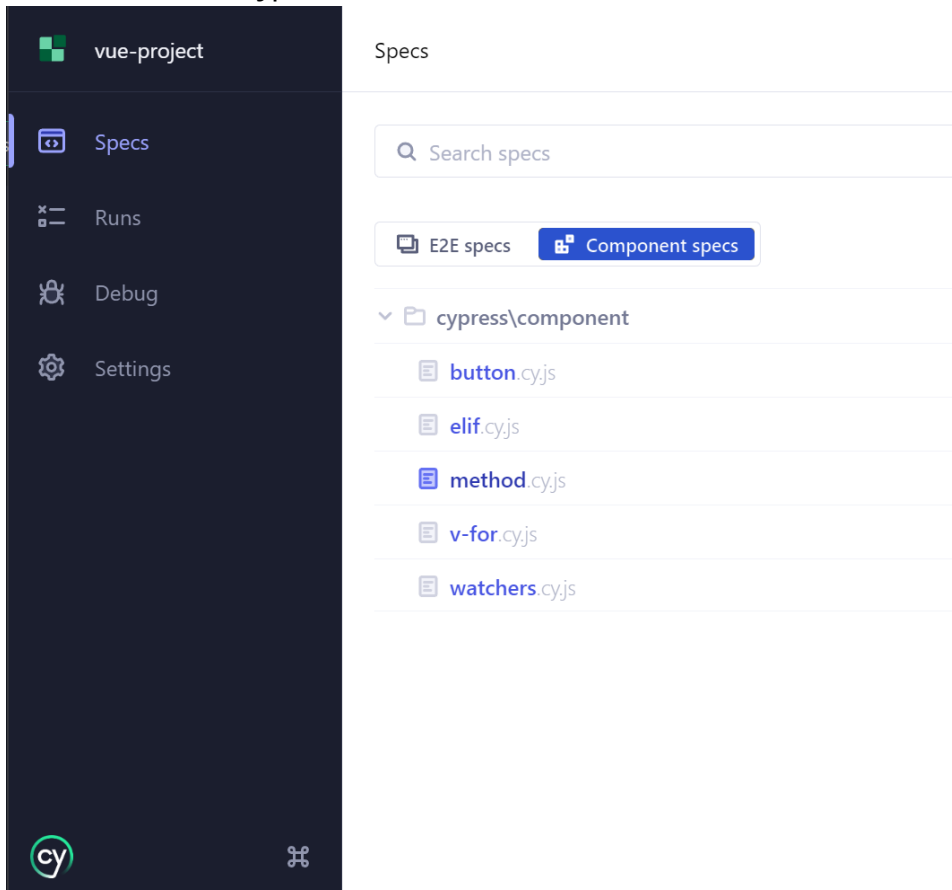
```
[Vue warn]: Maximum recursive updates exceeded. This means you have a reactive effect that is mutating
its own dependencies and thus recursively triggering itself. Possible sources include component template, render function, updated hook or watcher source function.

✓ src/components/__tests__/ButtonTest.spec.js (2)
✓ src/components/__tests__/DynamicTest.spec.js (2)
✓ src/components/__tests__/Elif.spec.js (2)
✓ src/components/__tests__/VforTest.spec.js (2)
✓ src/components/__tests__/HookTest.spec.js (2)
✓ src/components/__tests__/WatcherTest.spec.js (2)
✓ src/components/__tests__/MethodTest.spec.js (2)
✓ src/components/__tests__/InheritTest.spec.js (2)
✓ src/components/__tests__/HelloWorld.spec.js (2)
✓ src/components/__tests__/HelloWorld2.spec.js (2)

Test Files  10 passed (10)
Tests      20 passed (20)
Start at   15:57:44
Duration   6.51s (transform 1.02s, setup 5ms, collect 4.34s, tests 1.04s, environment 15.70s, prepare 4.99s)

PASS Waiting for file changes...
press h to show help, press q to quit
```

List of converted Cypress tests:



GitHub link to tests: <https://github.com/ArttuKorpela/Vue.JS-Testing/tree/7f2f3b5badaabb202c5f54a87f2dc7d058d602e7>
(These are the tests that would be added to the pull request)

Non-functional test

1 – Documentation:

ID	Tester	Testing approach & type	Result
N-1	JH	<Gray-box test> <Usability test>	Pass

- **Objective:** Evaluate the clarity and the extent of Vue.js documentation.
- **Steps:** Review the Vue.js documentation, focusing on the availability of clear examples, explanations, and references. Check if the documentation is easy to navigate and coherent.
- **Acceptance Criteria:** The documentation should be well structured, comprehensive and user-friendly.

2 – Installation Process:

ID	Tester	Testing approach & type	Result
----	--------	-------------------------	--------

N-2	JH	<Gray-box test> <Usability test>	Pass
-----	----	-------------------------------------	------

- **Objective:** Assess the ease or difficulty of installing Vue.js.
- **Steps:** Install Vue.js following the official installation instructions provided in the documentation. Record the time and any challenges encountered during the installation process.
- **Acceptance Criteria:** The installation process should be straightforward and well-documented, with minimal complications.
- **Notes:** No complications and installation took only a couple minutes with Node.js already installed.

3 – Official “Tutorial Project”, quick start

ID	Tester	Testing approach & type	Result
N-3	JH	<Gray-box test> <Usability test>	Pass

- **Objective:** Asses the steps and difficulty of creating your first Vue.js project, following the official instructions.
- **Steps:** Project creation following the provided official instructions. Evaluate the clarity of the instructions and test if the project works.
- **Acceptance Criteria:** The tutorial project should be easy to follow for beginners and the resulting project should function as expected.

4 – Adding First Component

ID	Tester	Testing approach & type	Result
N-4	JH	<Gray-box test> <Usability test>	Pass

- **Objective:** Asses the simplicity and time taken to add your first own component to the tutorial project
- **Steps:** Adding a simple component to the tutorial project, following the documentation’s examples.
- **Acceptance Criteria:** Adding single basic element should be quick and not too hard even for beginners.

5 – Load Time

ID	Tester	Testing approach & type	Result
N-5	JH	<Black-box test>	Pass

		<Performance test>	
--	--	--------------------	--

- **Objective:** Asses the time it takes for a Vue.js project to load in a browser.
- **Steps:** Check the site load time when launching a Vue.js project.
- **Acceptance Criteria:** Depending on the size of the project, load times will vary marginally but it should be expected to load in a reasonable time < 5s
- **Notes:** Quick start project used for this test, which is almost instantaneous.

6 – Resource Usage:

ID	Tester	Testing approach & type	Result
N-6	JH	<Black-box test> <Performance test>	Pass

- **Objective:** Evaluate the memory usage of a Vue.js project.
- **Steps:** Check used memory of the Vue.js project browser tab from the task manager or from the browser itself.
- **Acceptance Criteria:** The Vue.js project memory usage should remain within reasonable limits per browser tab. (0-500mb)
- **Notes:** Quick start project used for this test.

7 – Overall Usability

ID	Tester	Testing approach & type	Result
N-7	JH	<Gray-box test>	Pass

- **Objective:** Asses the overall usability of Vue.js
- **Steps:** Read parts of the documentation and check the examples provided on Vue.js official website.
- **Acceptance Criteria:** The Vue.js framework should be able to perform common tasks and have the usual features expected of this type of framework.

8 – Cross-browser

ID	Tester	Testing approach & type	Result
N-8	JH	<Black-box test>	Pass

		<Compatibility test>	
--	--	----------------------	--

- **Objective:** Validating the consistency of the app built with Vue.js in different browsers
- **Steps:** Tested the example project and examples provided in the documentation with different browsers such as Firefox, Chrome and Edge
- **Acceptance Criteria:** The page should be identical between the browsers e.g.; colours & layout are the same.

9 – Cross-OS

ID	Tester	Testing approach & type	Result
N-9	JH	<Black-box test> <Compatibility test>	Pass

- **Objective:** Validating the consistency of the app built with Vue.js in different operating systems
- **Steps:** Tested the example project provided in the documentation with different operating systems: MacOS & Ubuntu (Safari, Firefox)
- **Acceptance Criteria:** The page should be identical between the browsers e.g.; colours & layout are the same.

10 – Cross-OS

ID	Tester	Testing approach & type	Result
N-10	JH	<Black-box test> <Compatibility test>	Pass

- **Objective:** Validating the consistency of the app built with Vue.js in different operating systems
- **Steps:** Tested the example project provided in the documentation with different operating systems: MacOS & Ubuntu (Safari, Firefox)
- **Acceptance Criteria:** The page should be identical between the browsers e.g.; colours & layout are the same.

Other tests

ID	Tester	Feature being tested	Testing approach and type	Test result	Notes
O-1	JH	Usage of voice assistant	<Black-box test> <Usability test>	Fail	Not very practical
O-2	JH	Completeness of documentation	<Gray-box test> <Usability test>	Pass	

O-1 – Voice Assistant for Vue.js Project in VSCode

Description: Asses how voice assistant works with Vue project
(<https://marketplace.visualstudio.com/items?itemName=b4rtaz.voice-assistant>)

Steps:

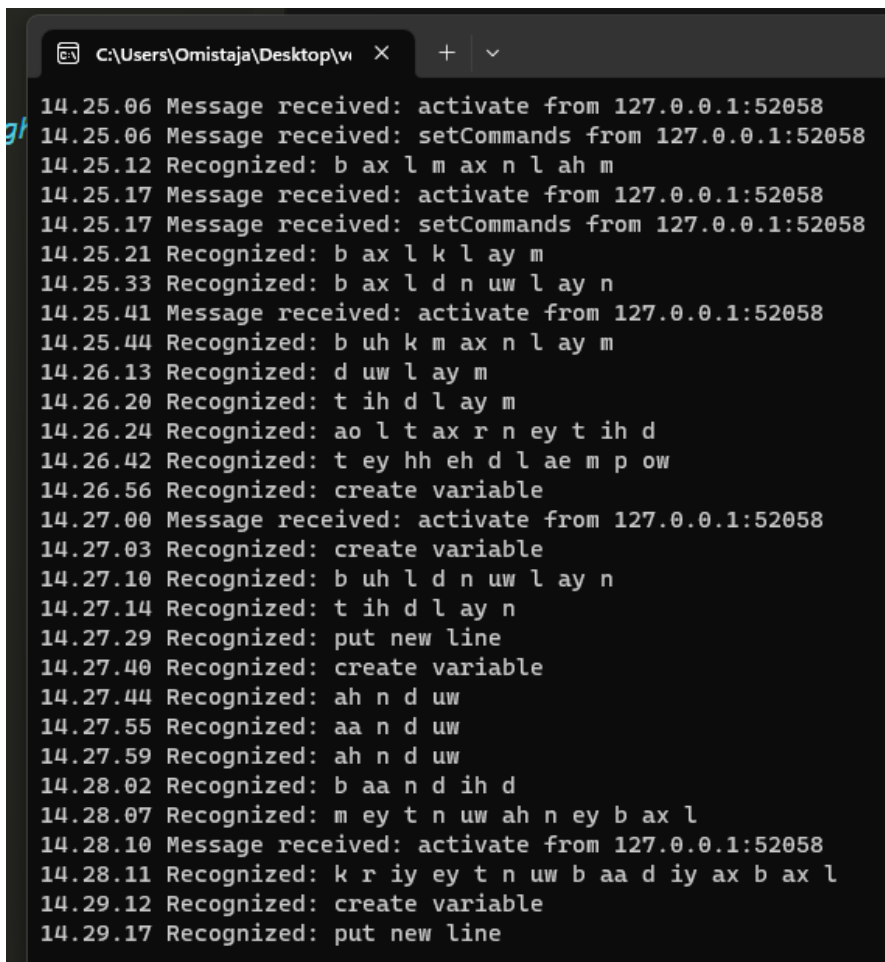
- Install Voice Assistant from the link
- Install the .NET voice recognition server provided in the link
- Run the server
- Add voice-assistant.json to the root directory of the project (Only HTML & TypeScript available, .json file can be edited though)
- Use your voice to active the specified commands in the voice-assistant.json file

Expected Outcome:

- Voice assistant should recognise the simplest commands almost 100% of the time

Actual Outcome:

- Using the voice assistant is quite burdensome and it does not recognise the commands too well (Picture 2.)

A screenshot of a web browser window showing a log of voice recognition messages. The browser's address bar shows the file path 'C:\Users\Omistaja\Desktop\w'. The log consists of a series of timestamped messages. Some messages are 'Message received' from '127.0.0.1:52058' and others are 'Recognized' commands. The commands include 'activate', 'setCommands', 'create variable', and 'put new line'. The text is displayed in a monospaced font on a dark background.

```
14.25.06 Message received: activate from 127.0.0.1:52058
14.25.06 Message received: setCommands from 127.0.0.1:52058
14.25.12 Recognized: b ax l m ax n l ah m
14.25.17 Message received: activate from 127.0.0.1:52058
14.25.17 Message received: setCommands from 127.0.0.1:52058
14.25.21 Recognized: b ax l k l ay m
14.25.33 Recognized: b ax l d n uw l ay n
14.25.41 Message received: activate from 127.0.0.1:52058
14.25.44 Recognized: b uh k m ax n l ay m
14.26.13 Recognized: d uw l ay m
14.26.20 Recognized: t ih d l ay m
14.26.24 Recognized: ao l t ax r n ey t ih d
14.26.42 Recognized: t ey hh eh d l ae m p ow
14.26.56 Recognized: create variable
14.27.00 Message received: activate from 127.0.0.1:52058
14.27.03 Recognized: create variable
14.27.10 Recognized: b uh l d n uw l ay n
14.27.14 Recognized: t ih d l ay n
14.27.29 Recognized: put new line
14.27.40 Recognized: create variable
14.27.44 Recognized: ah n d uw
14.27.55 Recognized: aa n d uw
14.27.59 Recognized: ah n d uw
14.28.02 Recognized: b aa n d ih d
14.28.07 Recognized: m ey t n uw ah n ey b ax l
14.28.10 Message received: activate from 127.0.0.1:52058
14.28.11 Recognized: k r iy ey t n uw b aa d iy ax b ax l
14.29.12 Recognized: create variable
14.29.17 Recognized: put new line
```

Picture 2. Voice recognition server log

O-2 – Vue.js Documentation

Description: Asses how usefull and extensive the official Vue.js' documentation is (<https://vuejs.org/>)

Steps:

- Navigate to the provided link
- Explore "docs" and its sub-tabs such as "Guide", "Glossary" and "Examples"

Outcome:

- The tabs "Guide," "Glossary," and "Examples" in Vue.js documentation collectively provide extensive and invaluable resources for developers, including comprehensive guides, a glossary of key terms, and interactive code testing in the browser.

Bug report

Given Vue.js' status as a highly popular and widely adopted JavaScript framework, our perspective has evolved to prioritize different things than bugs. Rather than actively seeking out bugs in Vue.js, we thought that the framework benefits from a robust and diligent development team, an engaged open-source community and a rigorous testing process. This collective effort contributes to a framework that is known for its reliability, stability and well-maintained codebase.

Nevertheless, it is essential to understand that while Vue.js has a reputation for being relatively bug-free, it is not entirely immune to issues. As with any software, bugs may surface under certain conditions or when integrating Vue.js with other libraries, frameworks or custom code. Instead of trying to find bugs in Vue.js itself, our focus shifted more towards testing the usability of Vue.js and ensuring that the core features work.

Risks report & Conclusion

Vue.js upon extensive testing, demonstrates a high level of readiness for production use. The testing phase focused on assessing the framework's stability, performance, and usability. Vue.js proved to be a robust and reliable choice. Testing was carried out by covering various aspects of the framework.

Documentation stood out as very extensive and clearly articulated which benefits beginners as well as more experienced developers. Additionally, while the framework itself is stable it is essential to stay aware about potential issues with third-party plugins and libraries often used with Vue.js.

In conclusion Vue.js' stability, performance, and features align with industry standards, making it a dependable choice for web development projects. The overall state of the project is positive, with Vue.js passing various testing scenarios and emerging as a strong contender for building web applications.