Bălan,Mihai-Stefan M.Ş.

Bms.balan@gmail.com

Contents

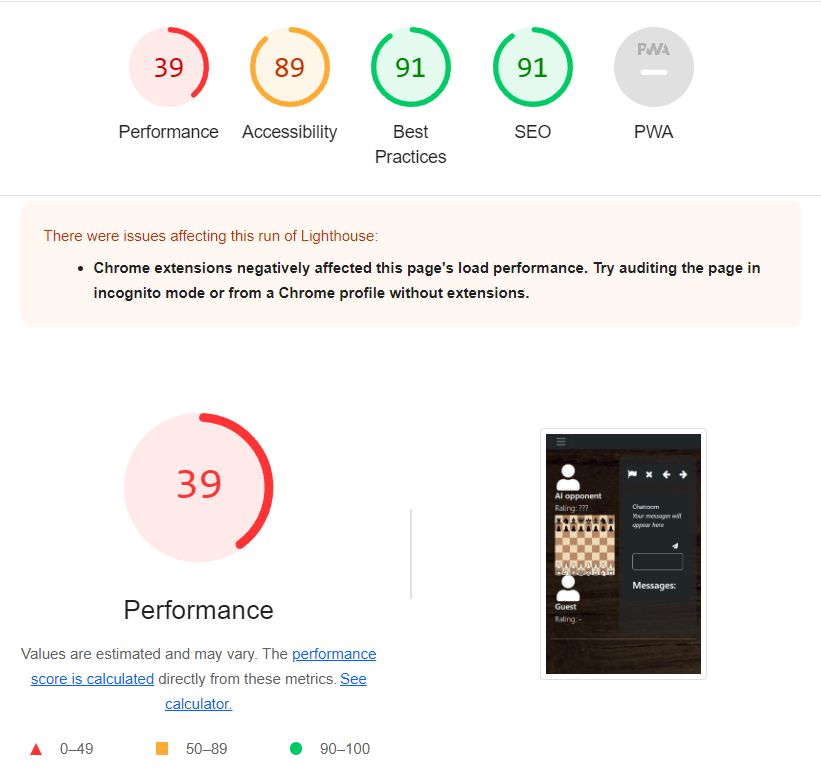
Document discussing the performance, accessibility and code practices of the application ChessNow

App Metrics report

ChessNow Project

# Front-End application

* Home page



**Key takeaways:**

1. The page loads everything at once.
2. The first contentful paint is heavily slowed down by the chessboard, which is composed of a total of 33 images. 32 out of those need to render user interactions as well.
3. The speed at which the page shifts the layout is good, the responsiveness does not need attention.
4. Decent accessibility, could improve on this aspect.
5. Good code quality

**Possible actions:**

1. Change the rendering priority of the components.
2. Compress chess images, reducing quality
3. Convert chess images to webp file format.

* Login page

A screenshot of a website

Description automatically generated

**Key takeaways:**

1. The page loads both the register and login form at once.
2. Almost perfect accessibility, no action needed
3. Code quality very good, no action needed

**Possible actions:**

Change the rendering priority of the components based on the page from which the user accessed the login and register page

* **Profile page**

**A screenshot of a computer

Description automatically generated**

**Key takeaways:**

1. The page loads the form first, then the match history list, this is a good render priority.
2. Unapplied CSS classes take up memory, in this case 43KiB.
3. Almost perfect accessibility, no action needed
4. Code quality very good, no action needed

**Possible actions:**

Optimize the backend requests to reduce the amount of times the database is called.

Note: this optimization has already been addressed

* **Social page**

**A screenshot of a computer

Description automatically generated**

**Key takeaways:**

1. The page loads every element of the list component at once. This takes a lot of time as it needs to wait for the objects from the backend.
2. If the search bar is empty, all users should be displayed. On the first render, no user is present in the search container. This is good for the speed of the application, as the same list is not rendered twice, unless the user actually wants it
3. Perfect accessibility, no action needed
4. Code quality very good, no action needed

**Possible actions:**

Add dynamic rendering of the list (one element at once instead of the entire list at once)

Note: this feature is out of scope.

**Conclusion**

Performance was not addressed during the development and hence the scores are low.

Achieving a good performance score was out of scope for this semester, so the focus was on accessibility and code quality (see tab ‘Best Practices’). Running the reports repeatedly it is apparent that the lighthouse evaluation is not entire objective. This means that the results may vary because of different devices used, testing on mobile or desktop, browser extensions, internet speed and server load (the server being my personal laptop in this case).

Some overall actions that could be done for every single page is adding priority to the rendering, compressing the images, minifying all the code in ‘Staging’ and upgrading the backend response speed by optimizing flows and/or changing the way the database is used.

While the performance of the application was unsatisfactory, every single page had some aspects addressing it. When actually using the application, the slow speed is not as obvious and not urgent to fix. The Jmeter performance load shows that over 1000 users using the app at once does not cause a significant performance drop for the backend. The database remains the only issue.