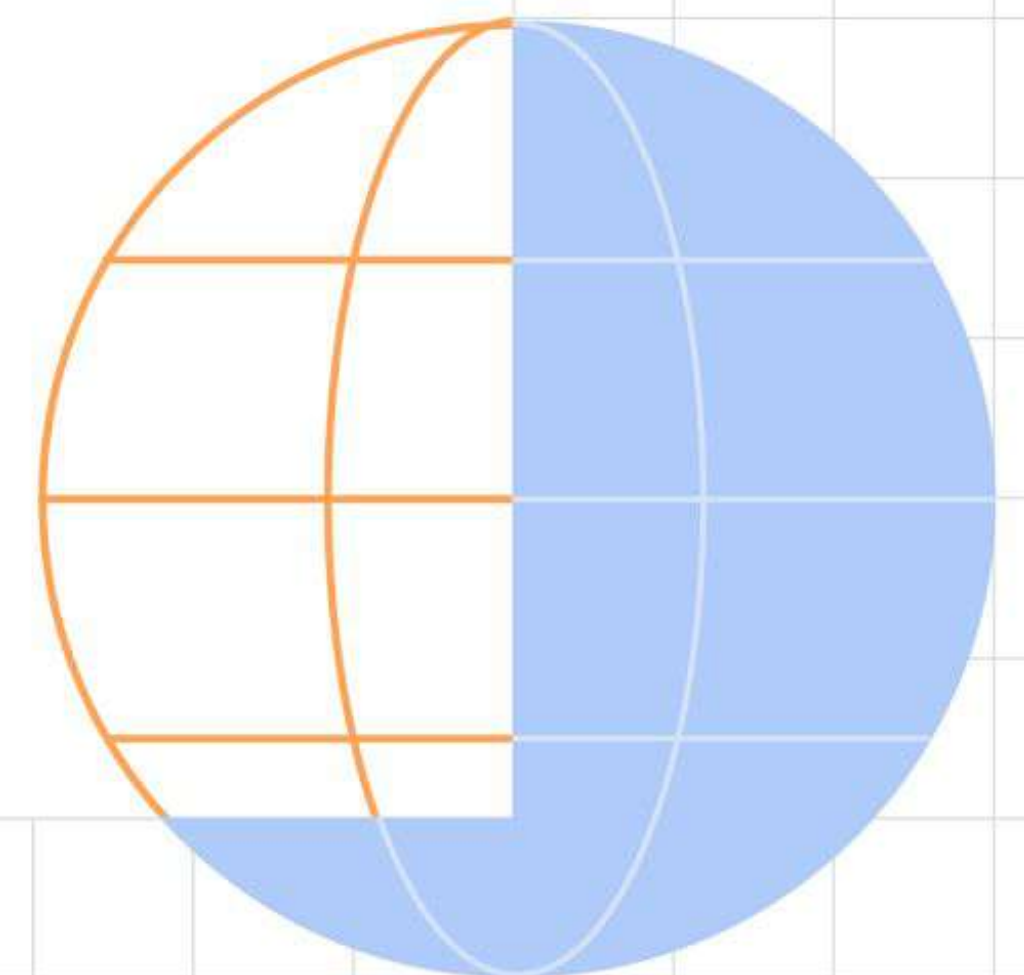


Implement Lighthouse-CI

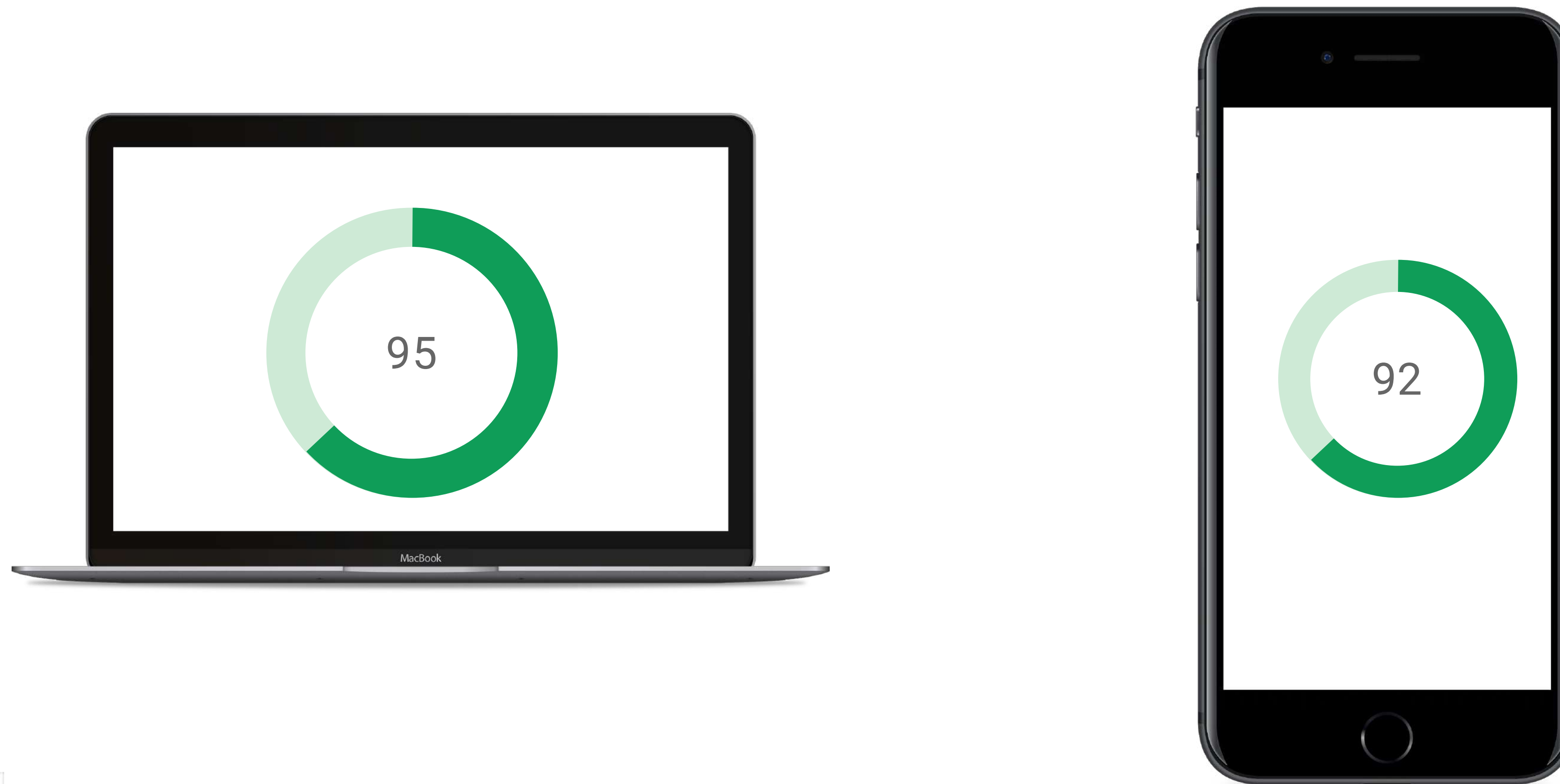
With your web development workflow



Ivan Kristianto
GDE Web and Performance
[@ivankrisdotcom](mailto:ivan@ivankristianto.com) / ivan@ivankristianto.com

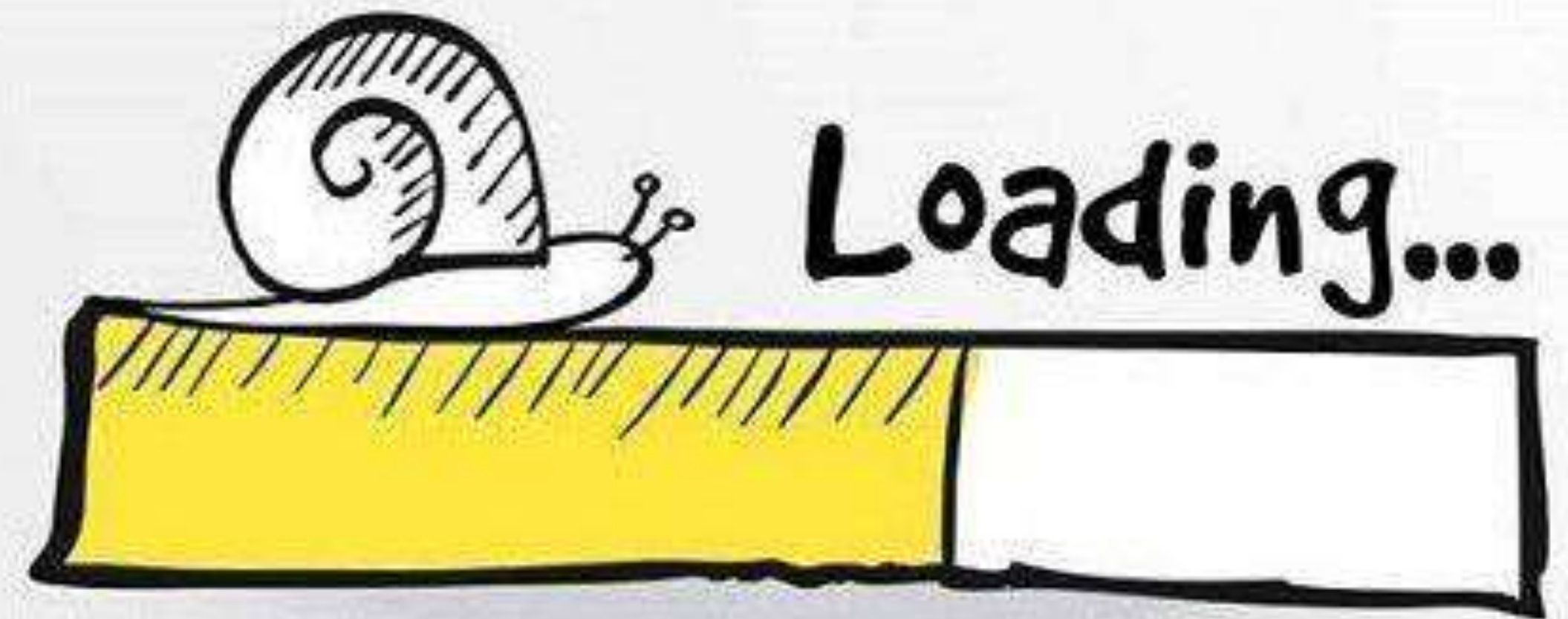


Performance is important!



We hate slow!

That's it!





I am assigned to lead a project

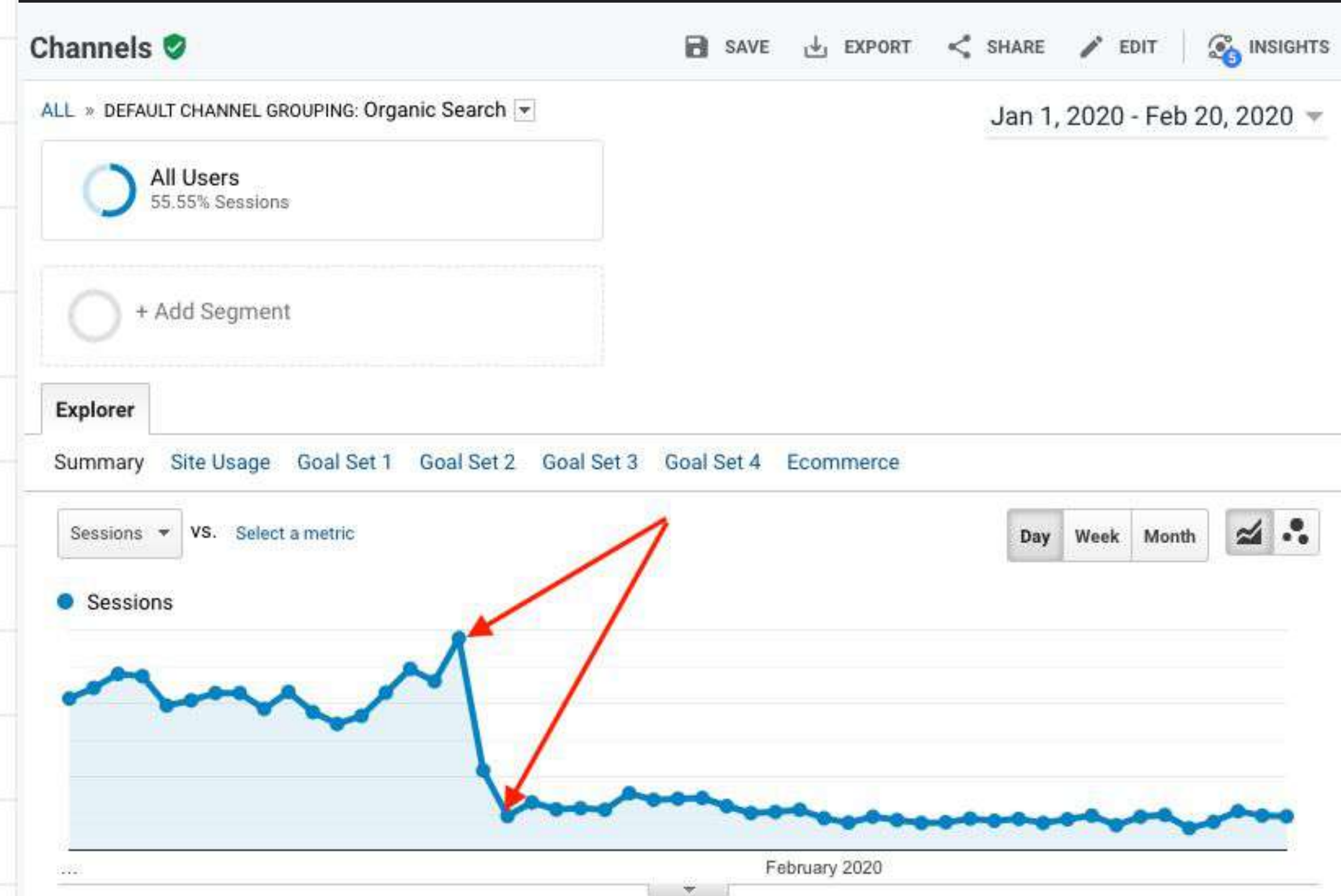
- Redesign 3 sites and rewrite it from scratch
- Fortunately, the requirement need to use WordPress with Gutenberg blocks
- The sites **should be fast**, for the end user and for the editor
- It has quite optimistic timeline
- It has features needs to be built on top of WordPress.

Core Issues

Traffic Drop = < \$\$\$

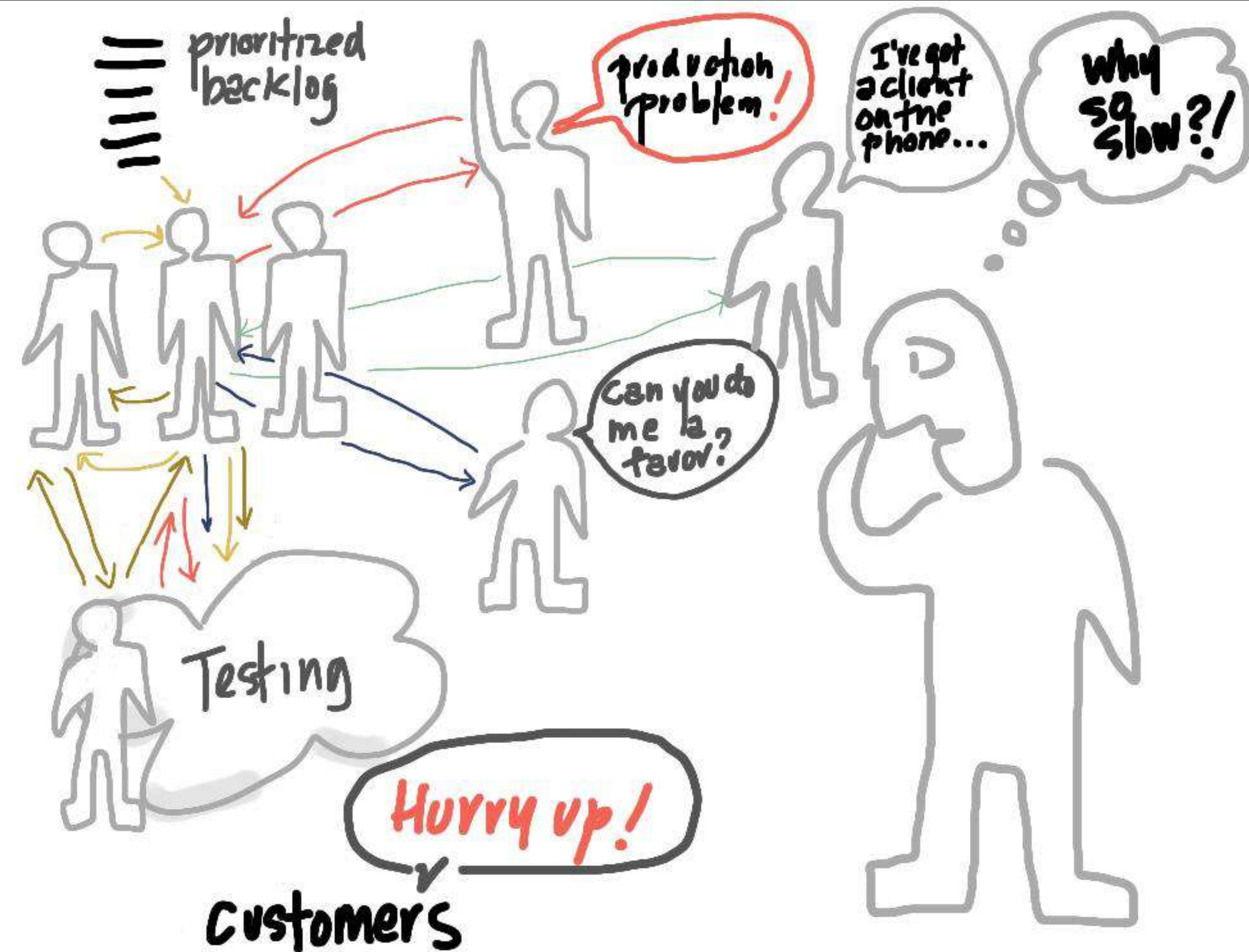
Traffic drop 30%.

And the site owner lose potential revenue each day!



Tangled by Workflow and Code Management

It takes ~15 days To do
minor change (ex: color,
font size, etc)



Performance Analysis

Current Performance

“This picture paints a 1000 words”

HOMETEST RESULTTEST HISTORYFORUMSDOCUMENTATIONABOUT

Web Page Performance Test for

Need help improving?

E

D

A

A

A

F

X

Security score

First Byte Time

Keep-alive Enabled

Compress Transfer

Compress Images

Cache static content

Effective use of CDN

SummaryDetailsPerformance ReviewContent BreakdownDomainsProcessing BreakdownScreenshotImage AnalysisRequest Map

Tester: MotoG4_20-192.168.1.120

Test runs: 3

Re-run the test

View JSON result

Raw page data - Raw object data

Export HTTP Archive (.har)

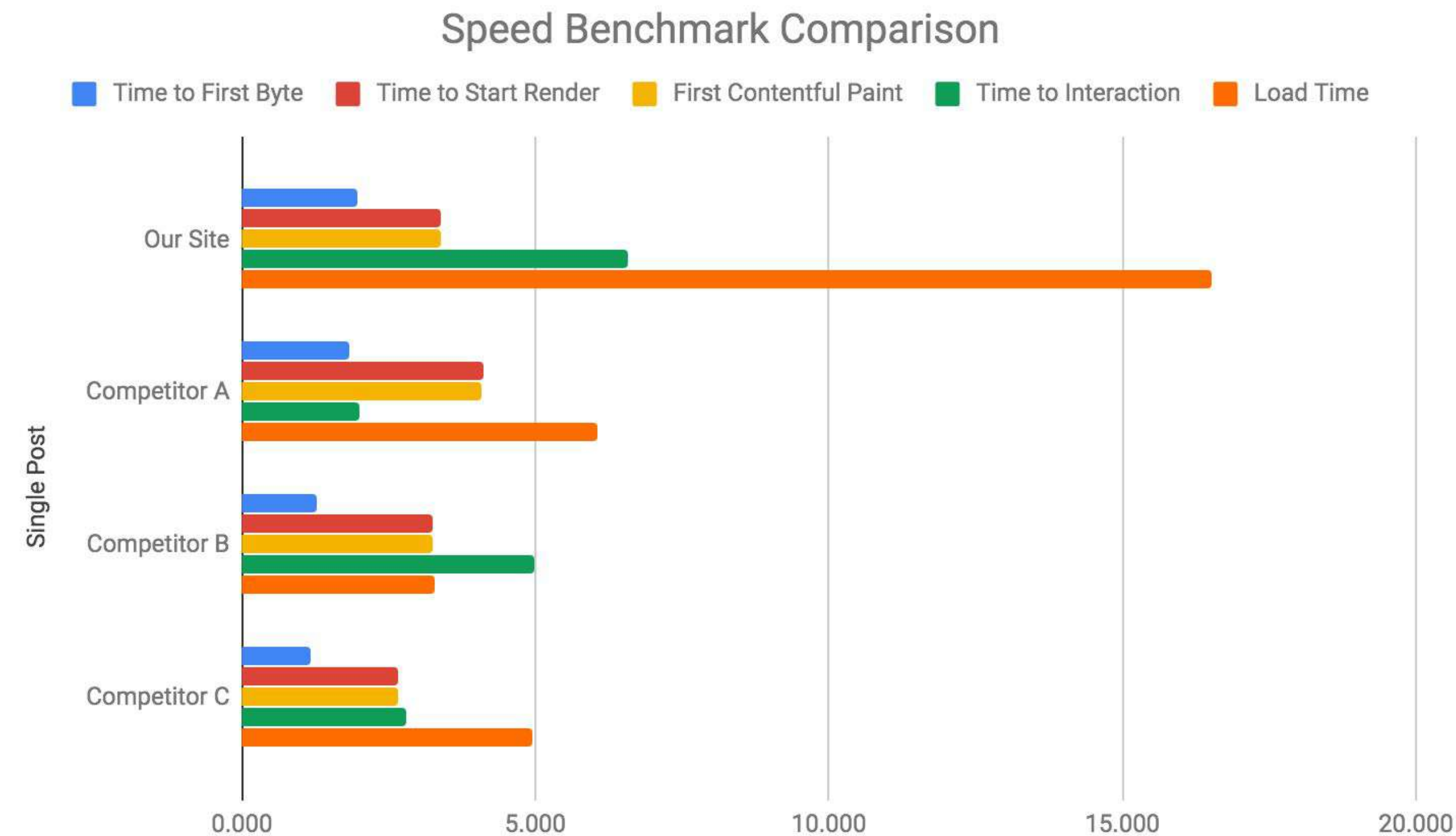
View Test Log

Performance Results (Median Run - SpeedIndex)

| | First Byte | Start Render | First Contentful Paint | Speed Index | Last Painted Hero | Web Vitals | | | Document Complete | | | Fully Loaded | | | |
|---------------------|------------|--------------|------------------------|-------------|-------------------|--------------------------|-------------------------|---------------------|-------------------|----------|----------|--------------|----------|----------|-----------|
| | | | | | | Largest Contentful Paint | Cumulative Layout Shift | Total Blocking Time | Time | Requests | Bytes In | Time | Requests | Bytes In | Cost |
| First View (Run 3) | 1.976s | 3.407s | 3.392s | 3.627s | 12.679s | 3.392s | 0.004 | ≥ 1.587s | 8.260s | 31 | 853 KB | 16.526s | 67 | 1,150 KB | \$\$\$- = |
| Repeat View (Run 3) | 2.575s | 4.615s | 4.615s | 4.719s | 9.894s | 4.615s | 0.004 | ≥ 1.223s | 6.304s | 7 | 298 KB | 11.737s | 27 | 509 KB | |

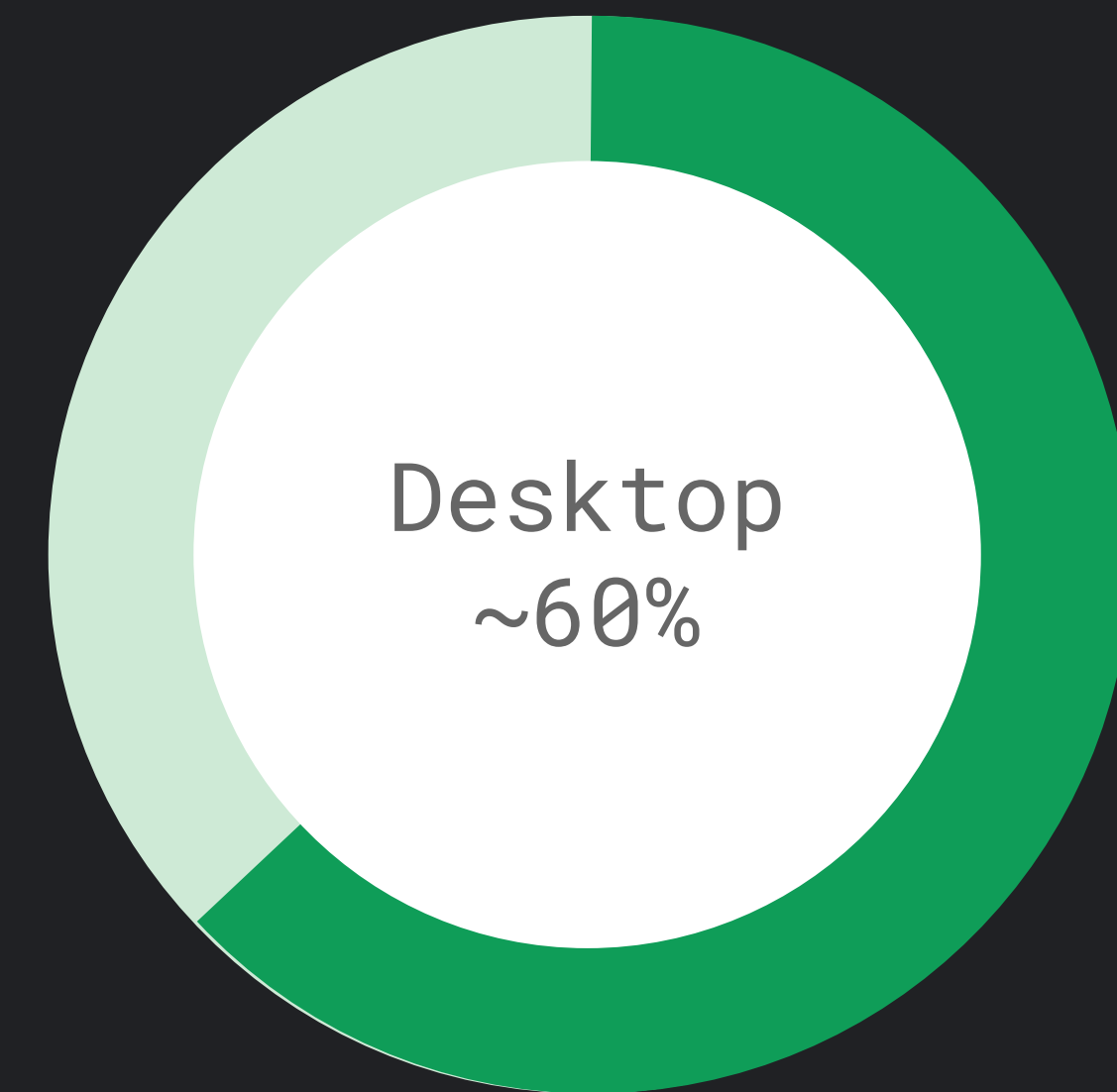
Comparison with Competitors

TTFB slowest,
FCP still slow,
And our **Load Time** is the worst.



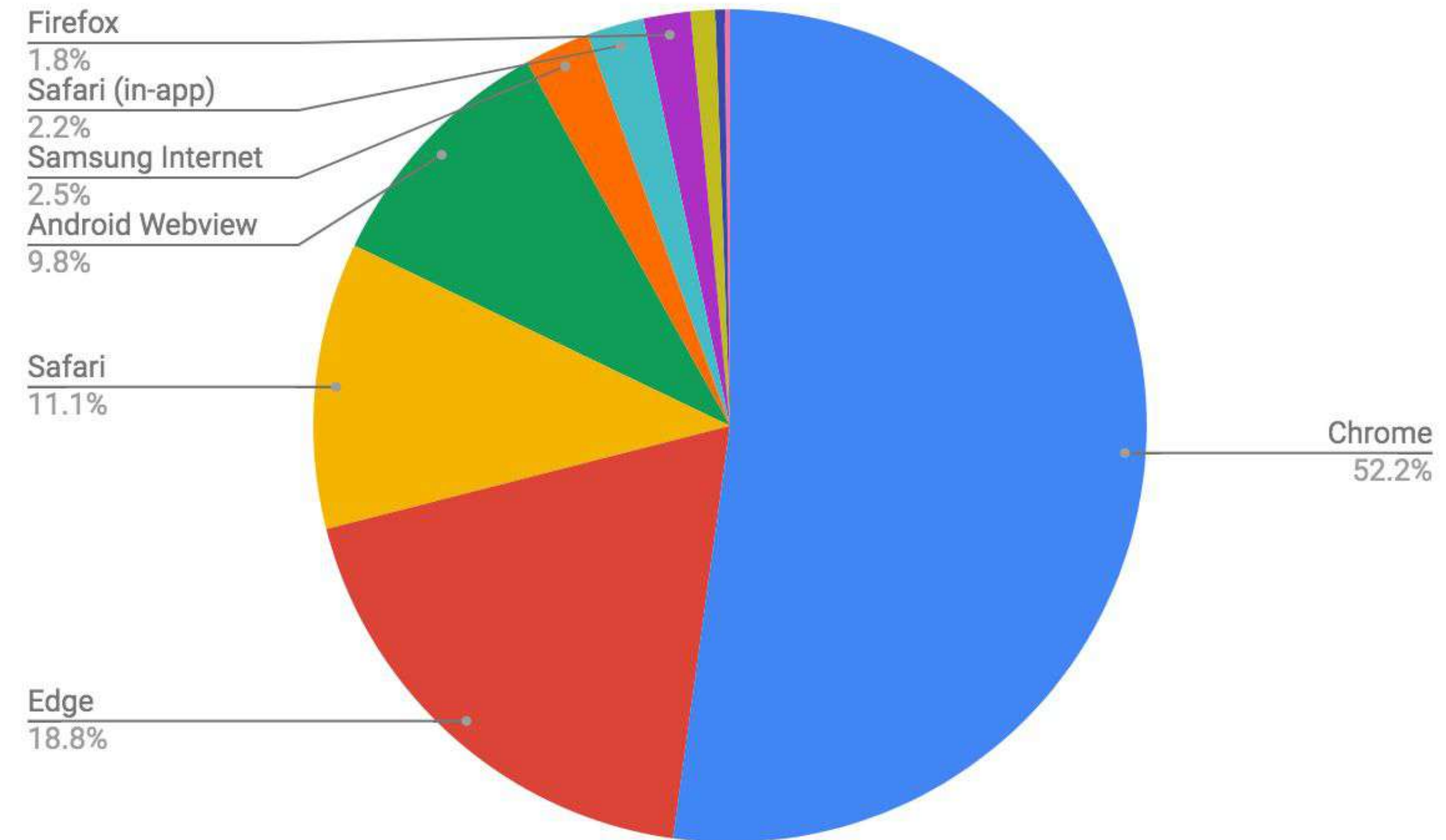
User Devices

From the last 6 months, 60% visitors. But it's declining since previous 6 month by 15% compare to mobile.



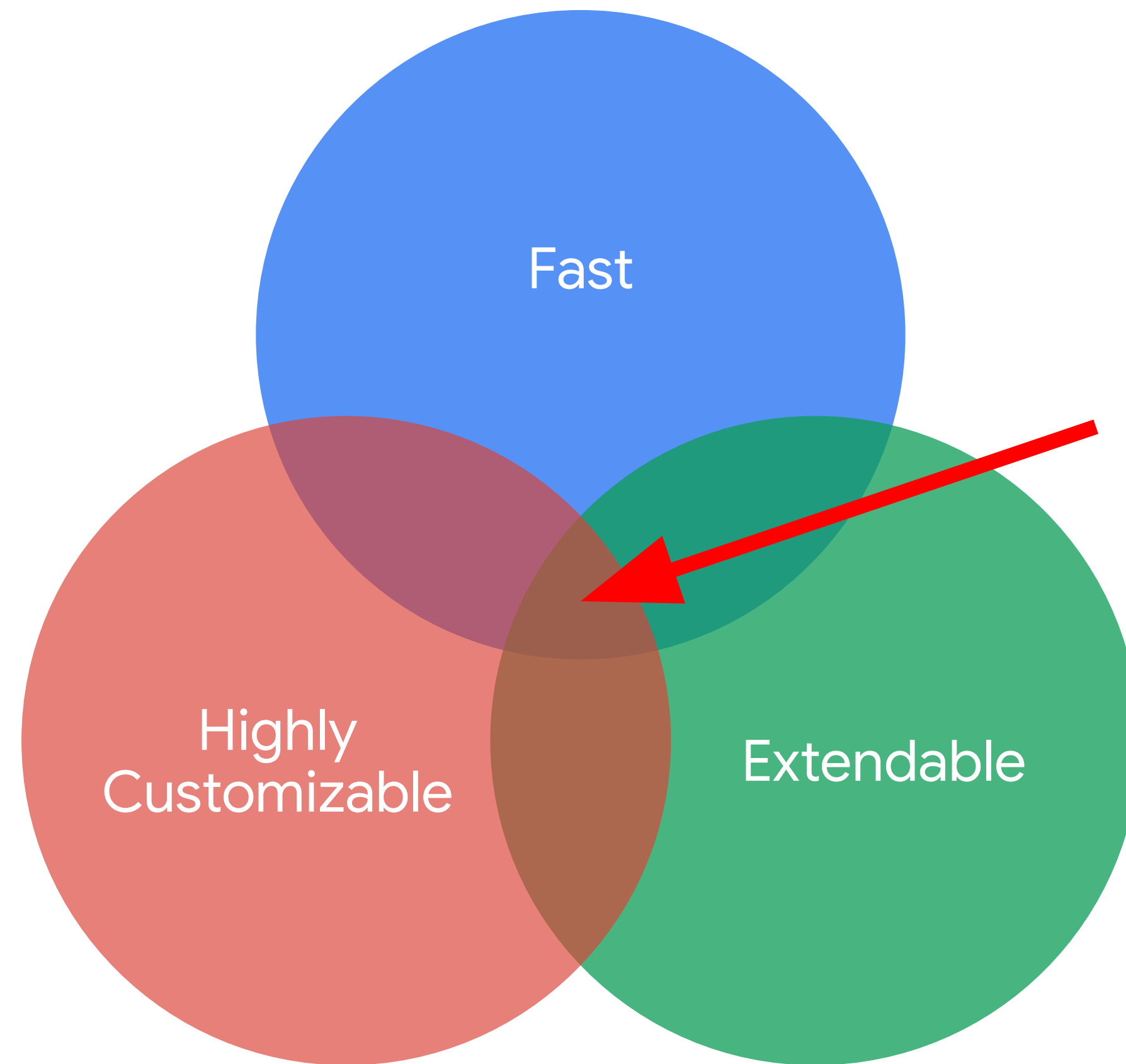
Browser Supports

From the last 6 months majority of users use Chrome by 52.2%, followed by Edge (18.8%) and Safari (11.1%).

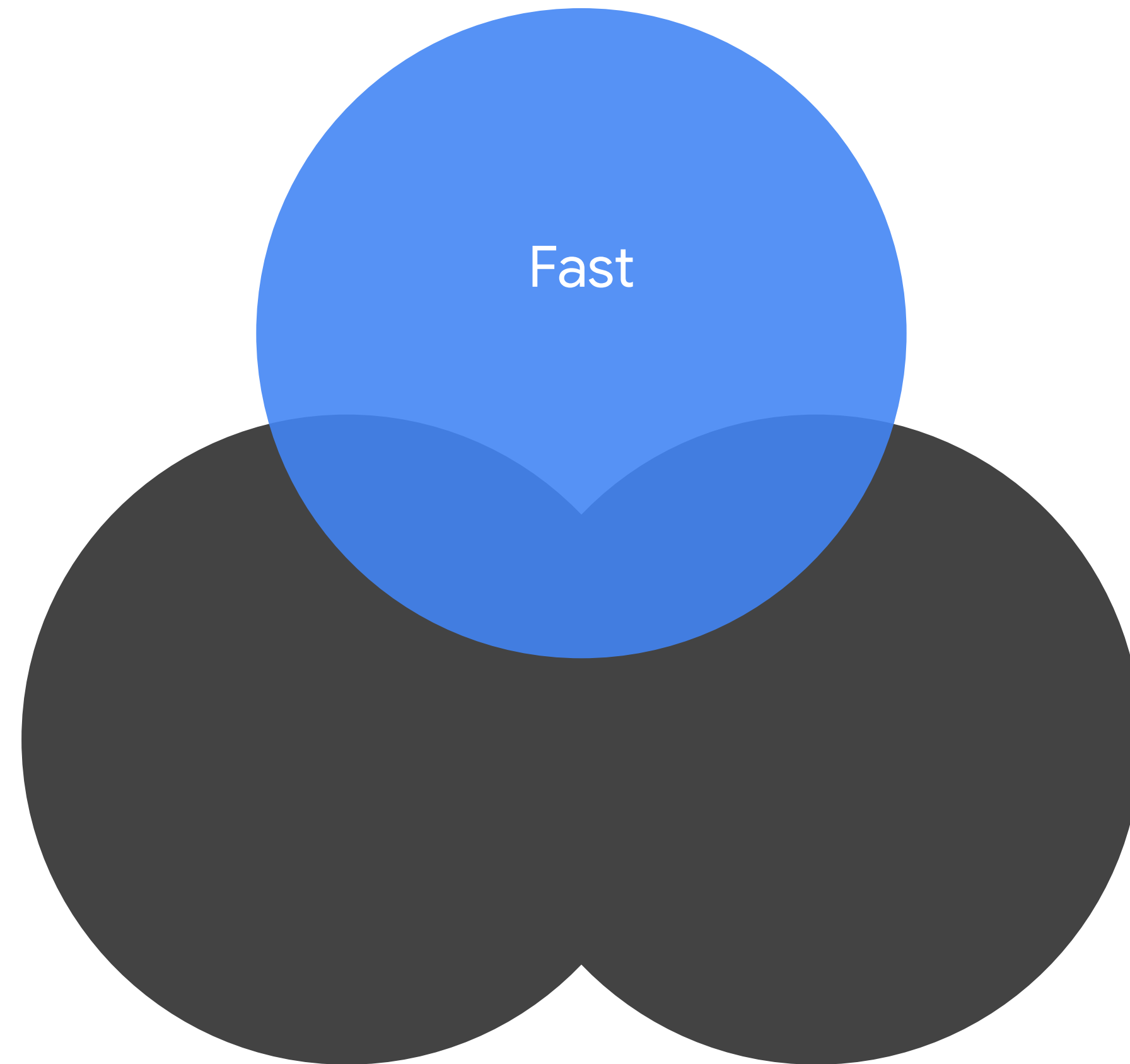


Target

Define our Goals



Define our Goals



Indicator

Fast

Defined our desired goals:

1. **Time To First Byte (TTFB)**. TTFB should occur within **0.5 second** from the user request.
2. **First Contentful Paint (FCP)**. FCP should occur within **1 second** of when the page first starts loading.
3. **Largest Contentful Paint (LCP)**. LCP should occur within **2.5 seconds** of when the page first starts loading.

<https://bit.ly/2ZDJdZu>

Performance Budget

Fast

Your budget:

599kb of 600kb used

Customise your budget

99kb

HTML

150kb

CSS

200kb

JavaScript

101kb

Images

0kb

Video

49kb

Fonts

<https://www.performancebudget.io>

Core Web Vitals

Fast

(Loading)

LCP

Largest Contentful Paint



(Interactivity)

FID

First Input Delay



(Visual Stability)

CLS

Cumulative Layout Shift



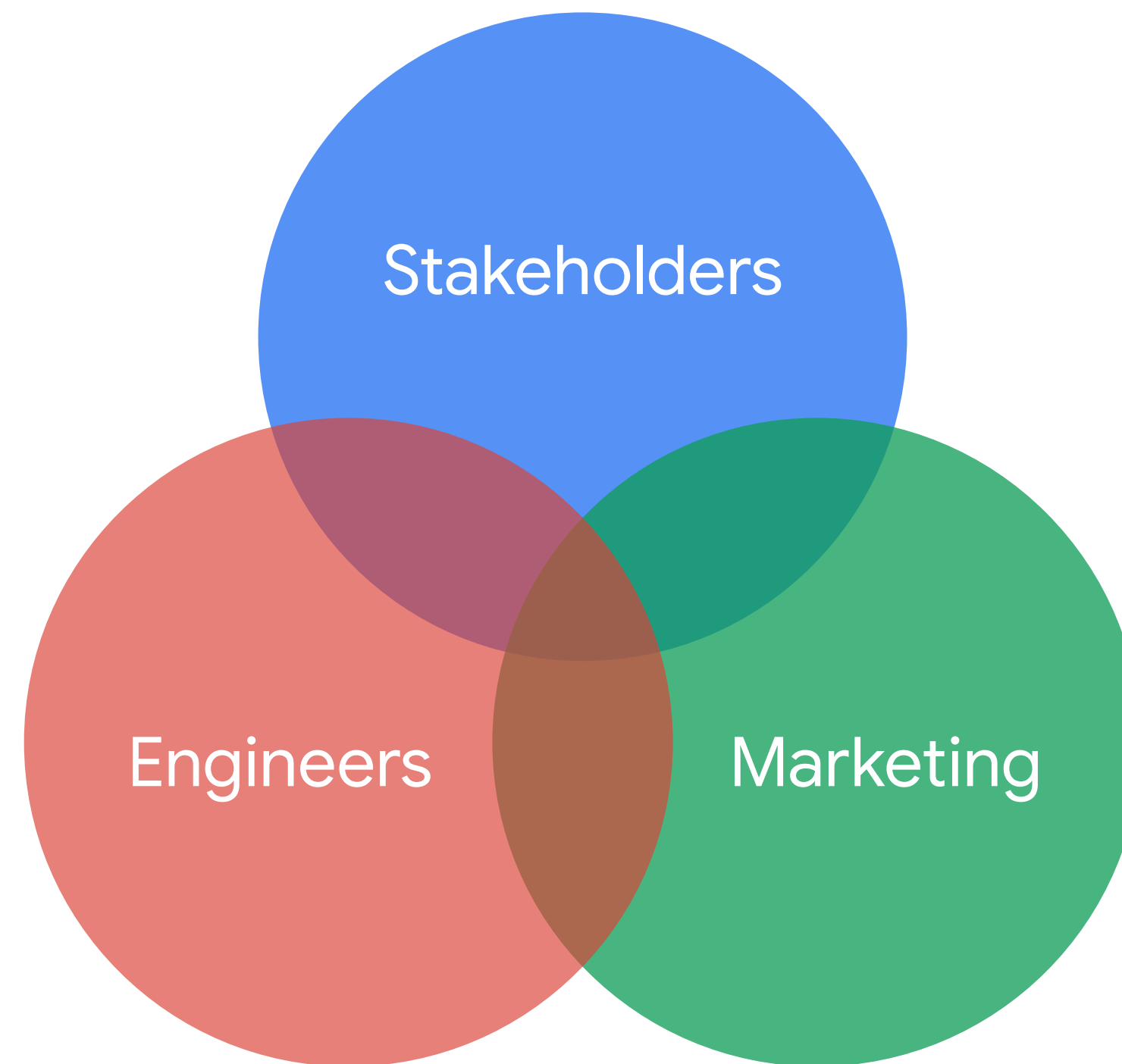
<https://bit.ly/2ZDJdZu> - Addy Osmani

Approach & Strategy

Communication

Communication

Get the message to all the stakeholders, engineers and marketing.



Desain UX & UI

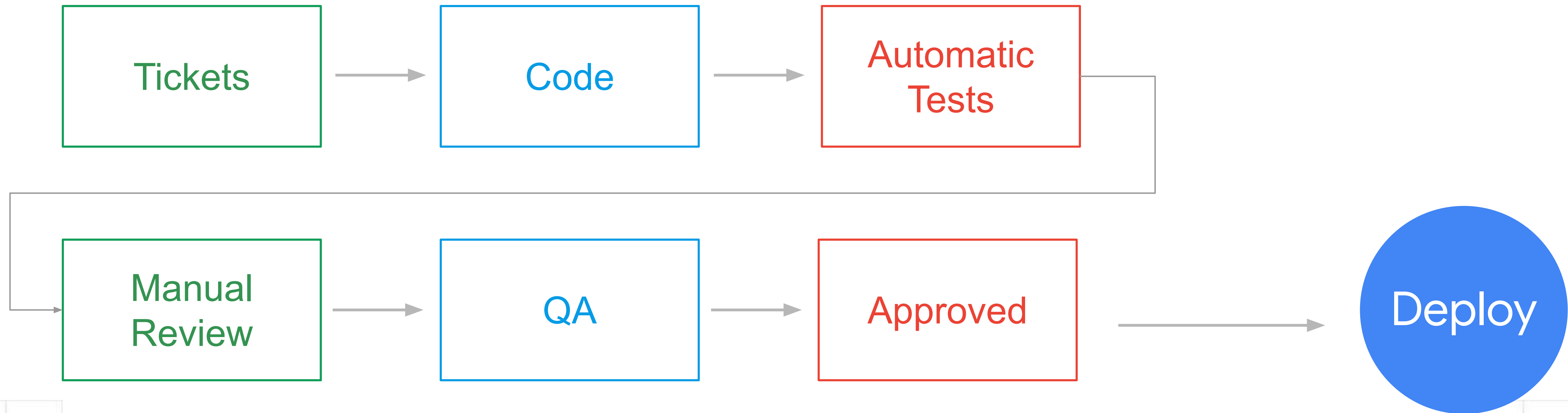
Design

Design UX & UI Approach

1. **Mobile first**, UX approach with mobile first could potentially minimize the complexity of the site on mobile view.
2. **Avoid hidden content**, ex: avoid the use of unnecessary sliders or tabbed content.
3. **Avoid element overflow and long content**

Development Workflow

Create documentation of the workflow



Images

A red circle containing the word "Images" in white text.

Requirement for images:

1. **LazyLoad**. Use native lazyLoad and fallback to JS lazyLoad.
2. **Responsive**. Images should use responsive sizes.
3. **Compressed**. Images should use gzip compression / brotli
4. **Use CDN**. Use 3rd party services for CDN.
5. **Support WebP** (*if browser support*).

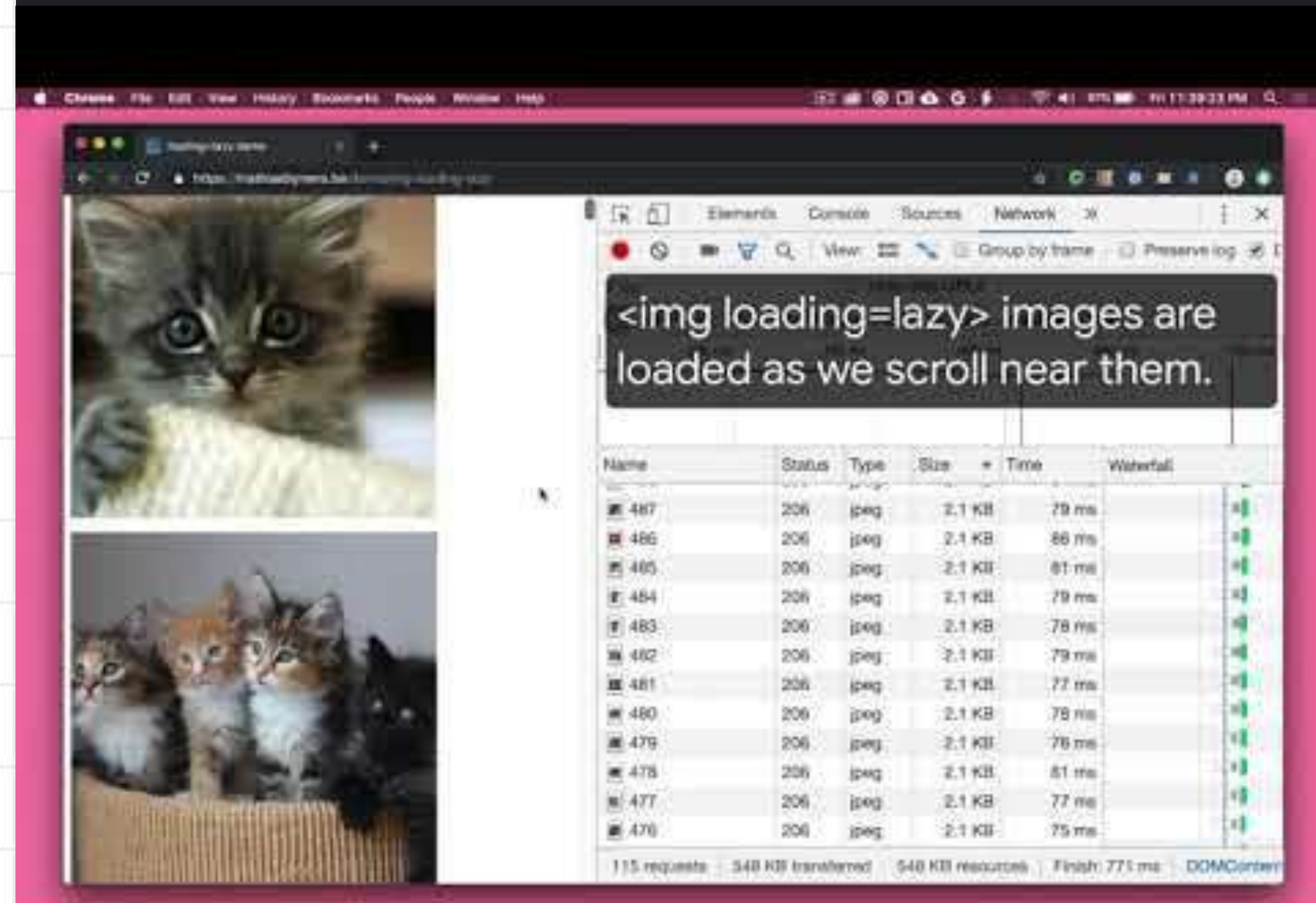
LazyLoad

Images

Most of the images should lazyLoad.

More:

<https://web.dev/native-lazy-loading/>



```
<!-- Let's load this in-viewport image normally -->


<!-- Let's lazy-load the rest of these images -->




<script>
  if ('loading' in HTMLImageElement.prototype) {
    const images = document.querySelectorAll("img.lazyload");
    images.forEach(img => {
      img.src = img.dataset.src;
    });
  } else {
    // Dynamically import the LazySizes library
    let script = document.createElement("script");
    script.async = true;
    script.src =
      "https://cdnjs.cloudflare.com/ajax/libs/lazysizes/4.1.8/lazysizes.min.js";
    document.body.appendChild(script);
  }
</script>
```

LazyLoad

Images

Ship as native since
WordPress 5.5

WordPress Plugin:

<https://wordpress.org/plugins/native-lazyload/>



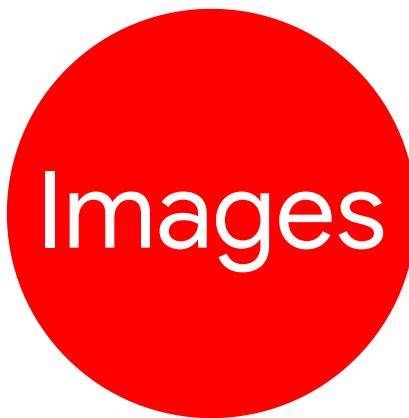
Native Lazyload

By Google



Download

Responsive



Most images should
implement `image srcset`.

WordPress support this
natively

```

```


CSS



Requirements:

1. **Visual Stability**, All images should have dimension attributes (width & height).
2. **Code Splitting**, separated by viewports.
3. **BEM (Block Element Modifiers)**, for *code reusability*.
4. **Best Practices**, membuat dokumentasi yang akan diterjemahkan menjadi *linter rules*

Visual Stability

CSS

All lazy load elements,
should have dimension
attributes (width & height)

Code Split

CSS

CSS File di pisah berdasarkan media queries:

1. small.css
2. medium.css
3. large.css

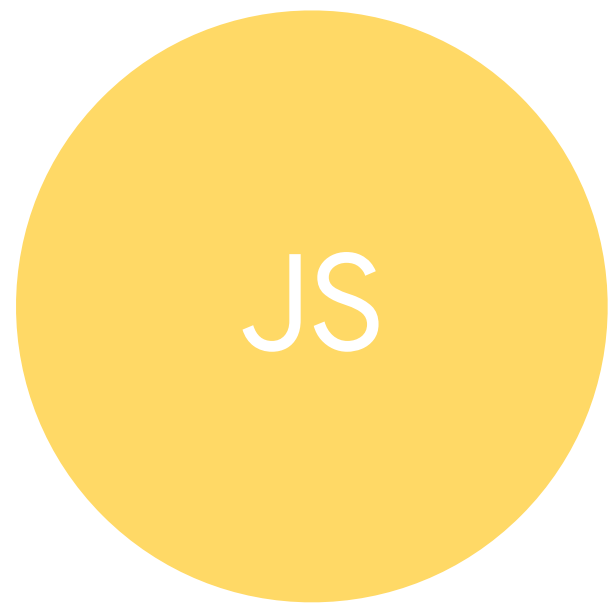
Code splitting dilakukan secara otomatis menggunakan webpack

```
<link rel="stylesheet"
      media="screen and (min-width: 300px)"
      href="small.css">
```

```
<link rel="stylesheet"
      media="screen and (min-width: 768px)"
      href="medium.css">
```

```
<link rel="stylesheet"
      media="screen and (min-width: 1200px)"
      href="large.css">
```

JavaScript



Our Approach in high level:

1. **Code Splitting**, and total size during first page load is max 160 kb
2. **No JQuery**, delete the use of jQuery on frontend
3. **Vanilla JS**, and only execute js when needed.
4. **Async & Defer**, all JS codes executed async and defer after document load
5. **Best Practices**, automate this with tests, linters rules.

Code Split

JS

JS files has *multiple entries* and used when only needed.

Max size per file is **80 Kb**, and will throw warning when exceed.

```
// webpack config > entries
```

```
entries: {
```

```
    // JS files.
```

```
    admin: './assets/js/admin/admin.js',
```

```
    blocks: './assets/js/blocks/blocks.js',
```

```
    frontend: './assets/js/frontend/frontend.js',
```

```
    styleguide:
```

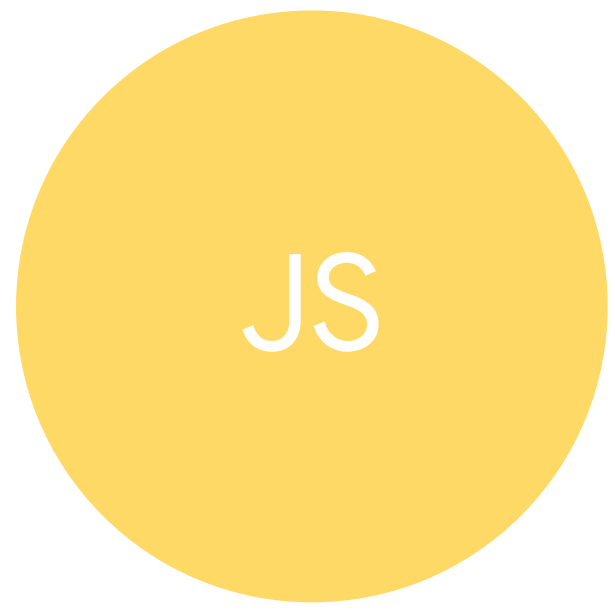
```
    './assets/js/styleguide/styleguide.js',
```

```
      'blocks-editor':
```

```
      './includes/blocks/blocks-editor.js',
```

```
}
```

3rd Party JavaScripts



Our approach

1. **Limit 3rd party *scripts***, always refer to our performance budget.
2. **Test & Monitoring**, do test and monitoring for this 3rd party JS
3. Use when needed, most of the times, not every page need it.
4. All 3rd party scripts, load after **`window.onload`** event

3rd Parties

JS

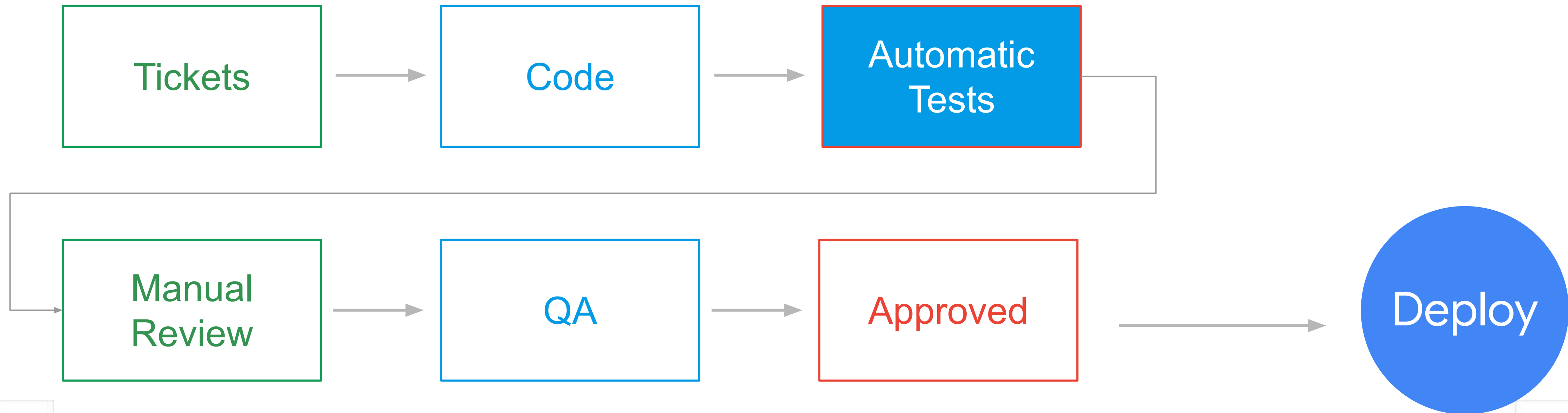
1. Refer back **Performance Budget**
2. From this list our exception is “OneTrust” need high priority.

```
{  
  "Google Tag Manager",  
  "Google Analytics",  
  "Lucky Orange",  
  "Adoric",  
  "Facebook Pixel",  
  "Survey Monkey",  
  "Aimtell",  
  "HotJar",  
  "Instana",  
  "OneTrust",  
}
```

Automation

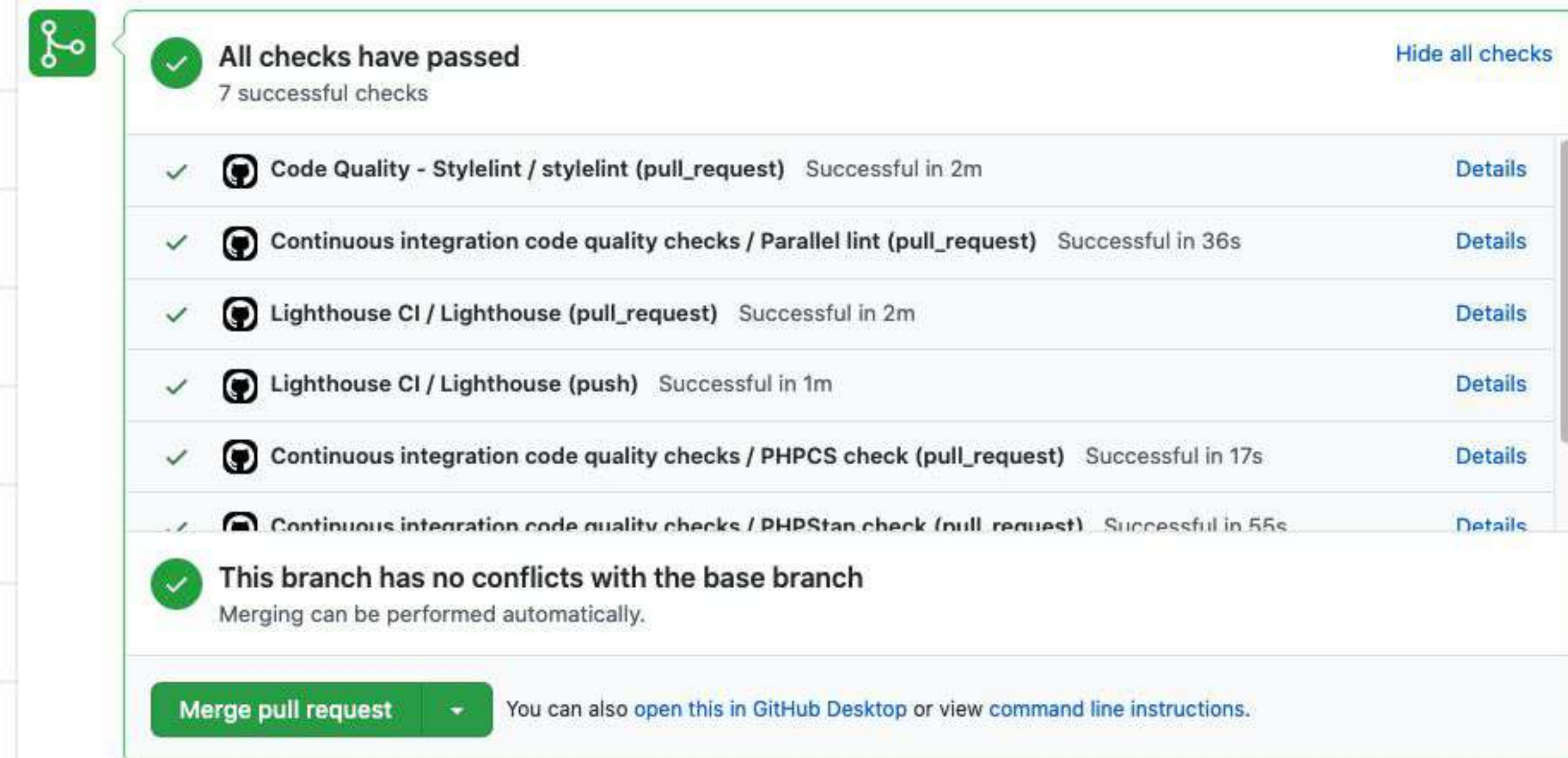
Development Workflow

Implement the automated tests



Automated Tests

1. Run automated tests and build with CI/CD
2. Run Lighthouse-CI for automated performance tests



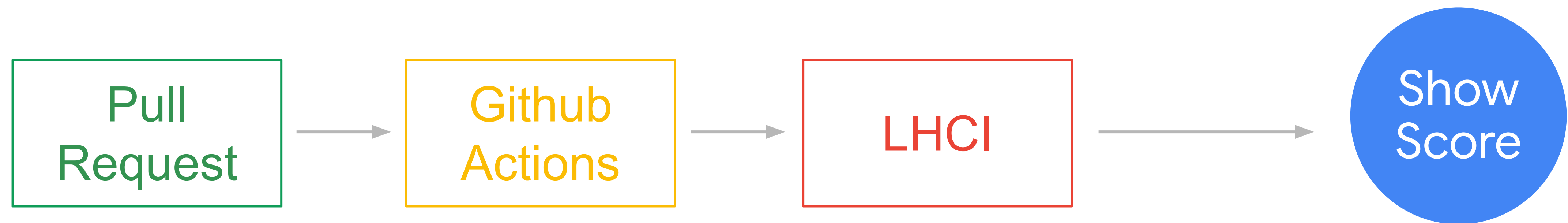
The screenshot displays a GitHub Actions workflow status for a pull request. At the top, a green checkmark icon and the text "All checks have passed" are shown, along with "7 successful checks" and a "Hide all checks" link. Below this, a list of checks is displayed, each with a green checkmark, a GitHub Actions icon, the check name, the status, and a "Details" link. The checks are:

- Code Quality - Stylelint / stylelint (pull_request) Successful in 2m
- Continuous integration code quality checks / Parallel lint (pull_request) Successful in 36s
- Lighthouse CI / Lighthouse (pull_request) Successful in 2m
- Lighthouse CI / Lighthouse (push) Successful in 1m
- Continuous integration code quality checks / PHPCS check (pull_request) Successful in 17s
- Continuous integration code quality checks / PHPStan check (pull_request) Successful in 55s

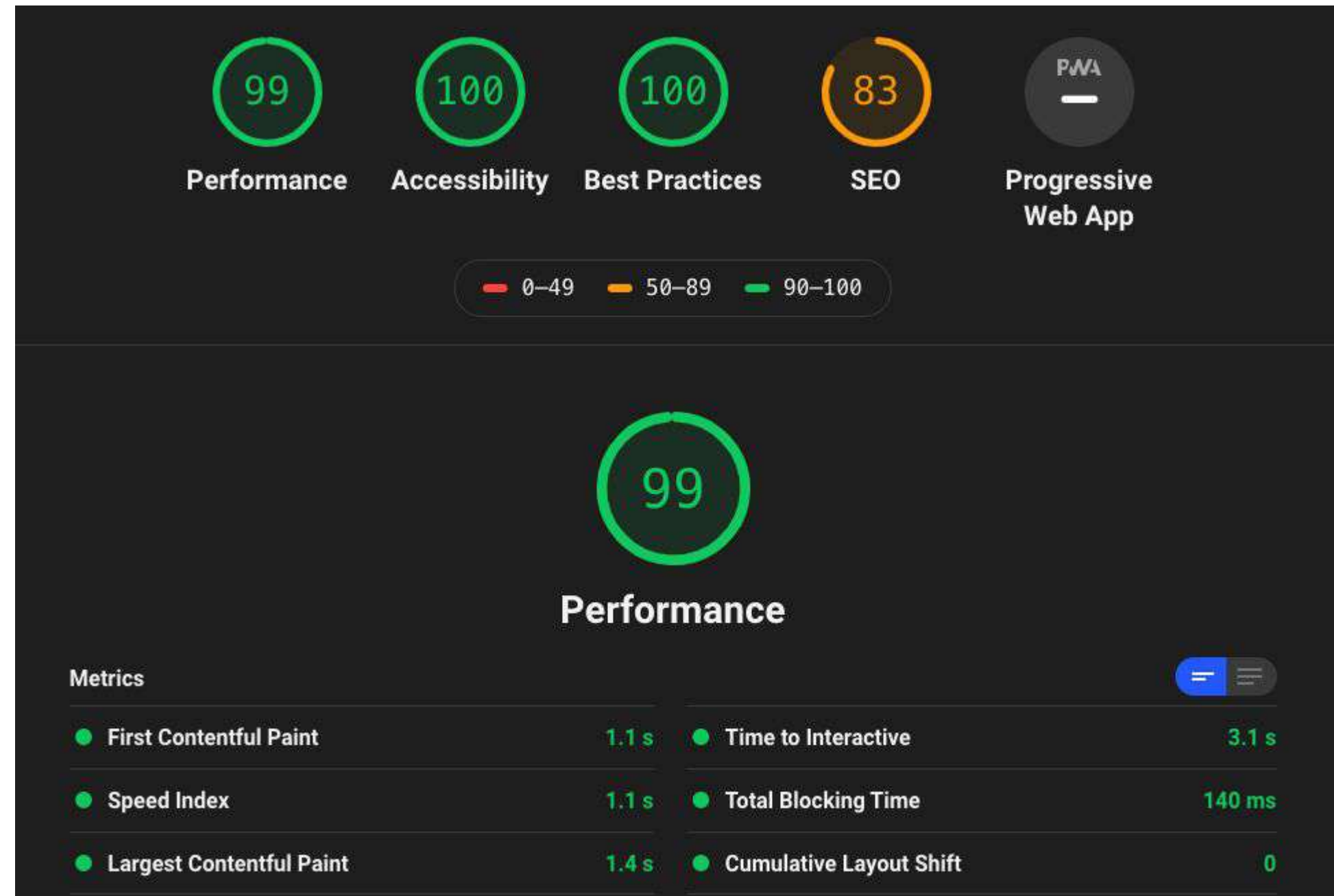
Below the list, a green checkmark icon and the text "This branch has no conflicts with the base branch" are shown, along with "Merging can be performed automatically." At the bottom, a green button labeled "Merge pull request" is visible, followed by a link to "You can also open this in GitHub Desktop or view command line instructions."

Lighthouse CI

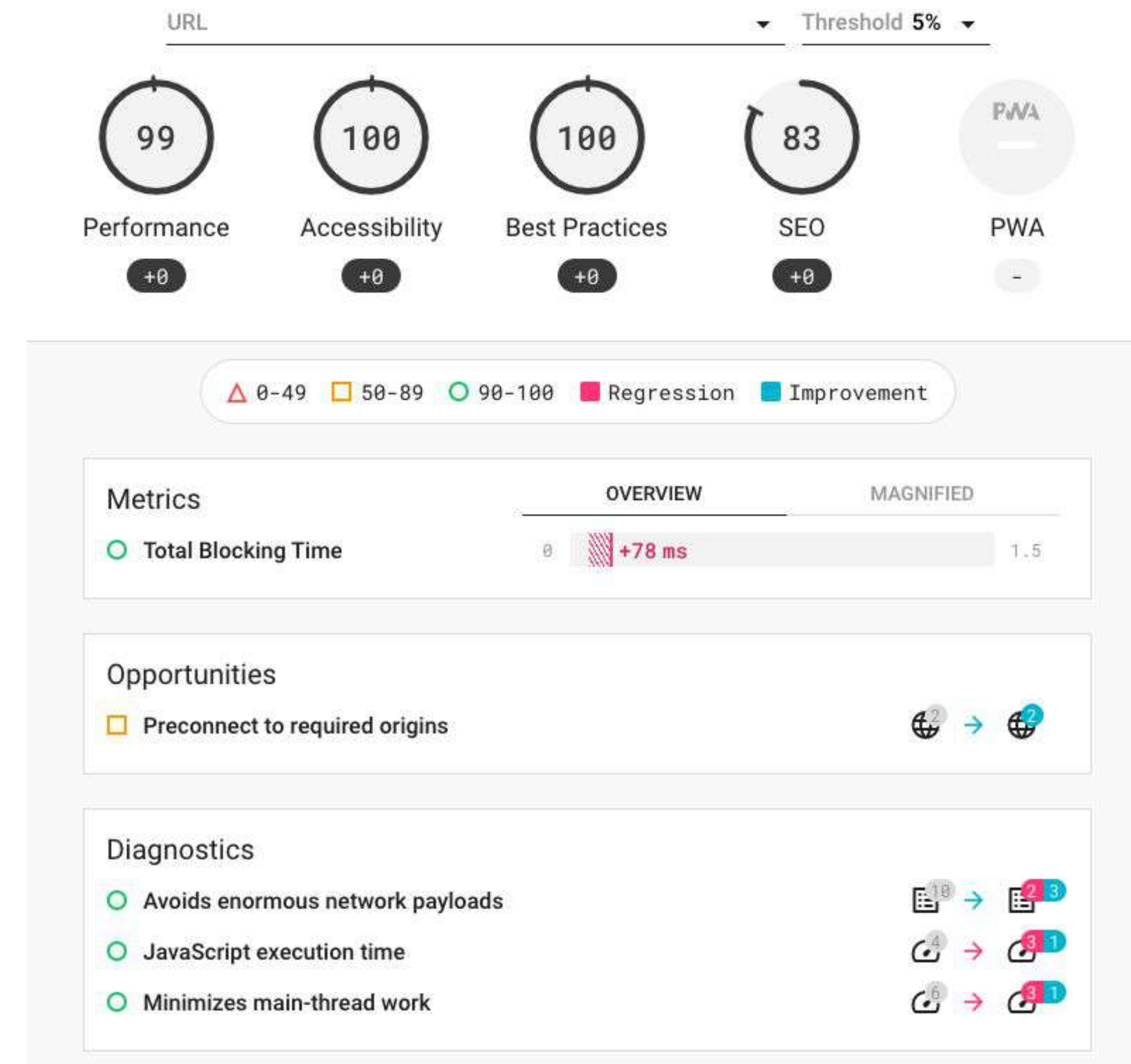
Run LHCI on Every *Pull Request*



Lighthouse CI Server



<https://github.com/GoogleChrome/lighthouse-ci>

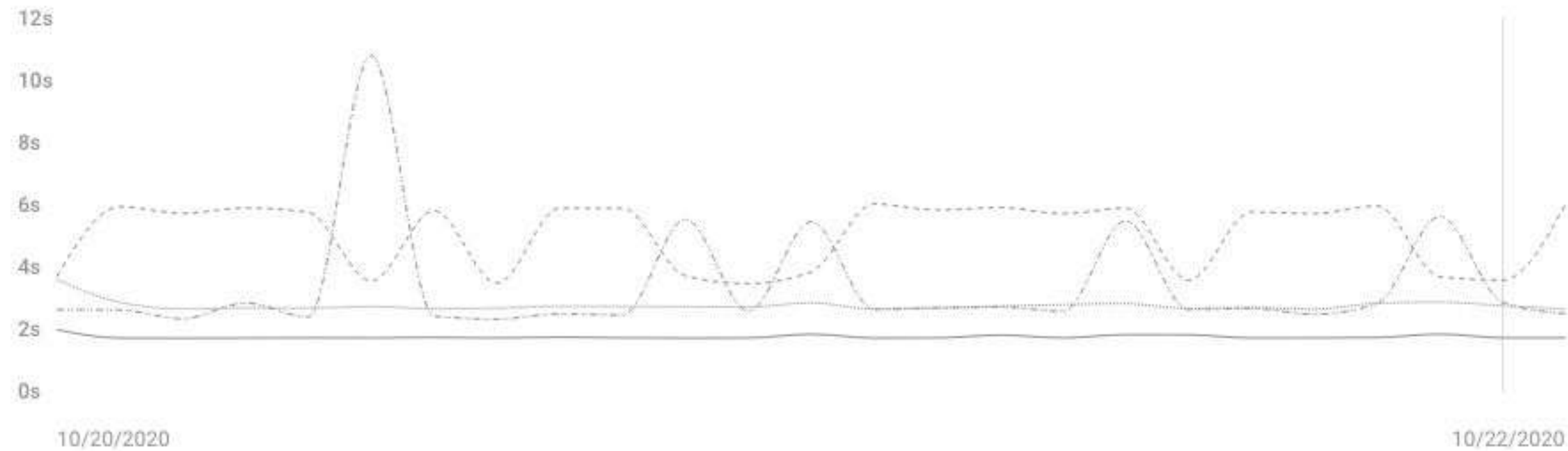
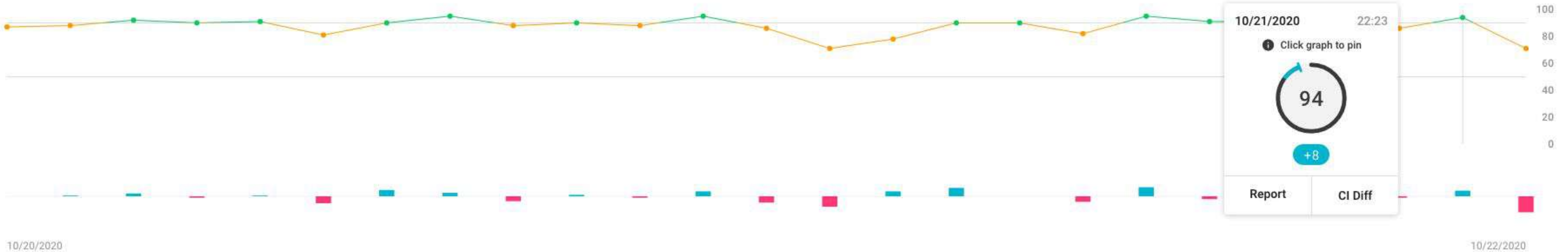


Performance

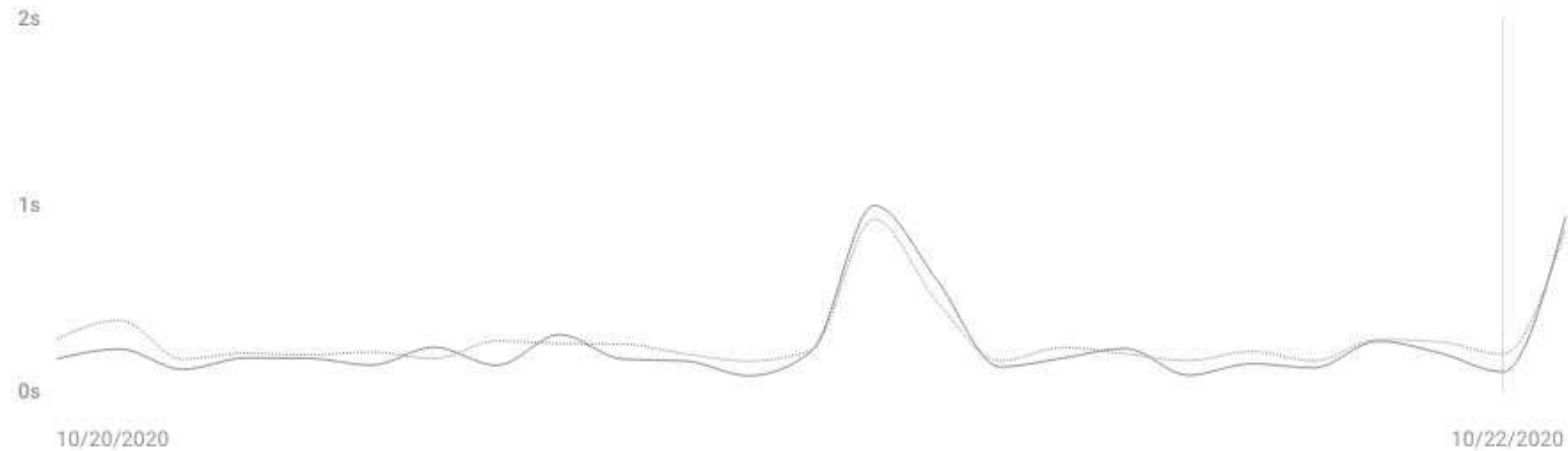
Timeline Distribution

25 50 100 MAX

Overview



- FCP First Contentful Paint
- LCP Largest Contentful Paint
- TTI Time to Interactive
- SI Speed Index

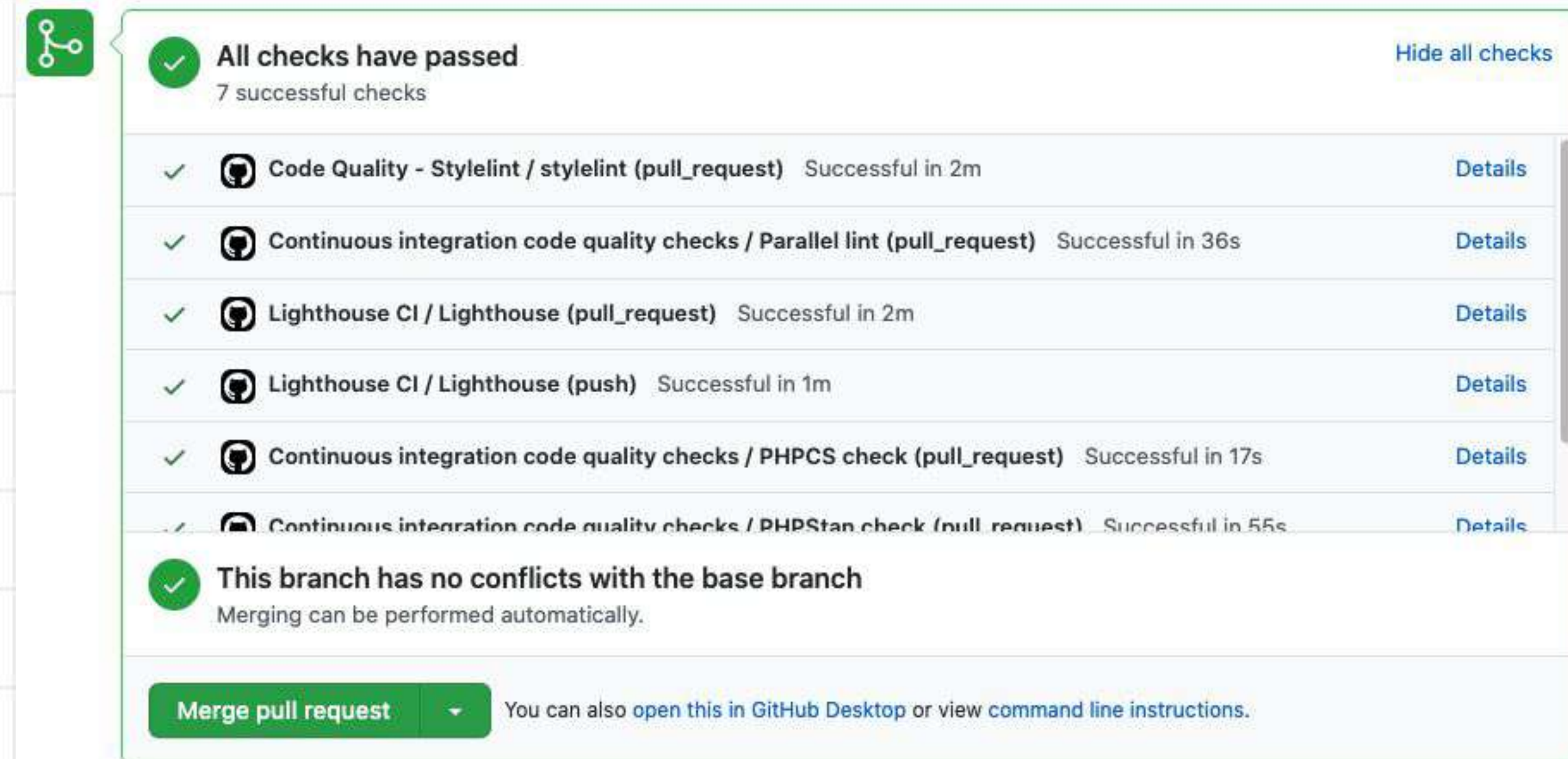


- TBT Total Blocking Time
- FID Max Potential FID

Implement LHCI

Options

1. Temporary Public Storage
2. Private CI Server:
 - a. Docker
 - b. Firebase
 - c. Heroku
 - d. etc.



The screenshot displays a GitHub Actions workflow status for a pull request. At the top, a green checkmark icon and the text "All checks have passed" are shown, along with "7 successful checks" and a "Hide all checks" link. Below this, a list of checks is displayed, each with a green checkmark, a GitHub Actions icon, the check name, the status, and a "Details" link. The checks are:

- Code Quality - Stylelint / stylelint (pull_request) Successful in 2m
- Continuous integration code quality checks / Parallel lint (pull_request) Successful in 36s
- Lighthouse CI / Lighthouse (pull_request) Successful in 2m
- Lighthouse CI / Lighthouse (push) Successful in 1m
- Continuous integration code quality checks / PHPCS check (pull_request) Successful in 17s
- Continuous integration code quality checks / PHPStan check (pull_request) Successful in 55s

Below the list, a green checkmark icon and the text "This branch has no conflicts with the base branch" are shown, along with "Merging can be performed automatically." At the bottom, a green button labeled "Merge pull request" is visible, followed by a dropdown arrow and the text "You can also open this in GitHub Desktop or view command line instructions."

Heroku



Setup LHCI on Heroku

1. Install Heroku CLI
2. Run the scripts from CLI
3. Done

```
> LHCI="unique-app-name" \  
&& heroku create $LHCI \  
&& git clone https://git.heroku.com/$LHCI.git \  
lhciapp \  
&& cd lhciapp \  
&& heroku addons:create heroku-postgresql:hobby-dev  
  
> curl \  
https://raw.githubusercontent.com/GoogleChrome/lighthouse-ci/master/docs/recipes/heroku-server/package \  
.json > package.json  
  
> curl \  
https://raw.githubusercontent.com/GoogleChrome/lighthouse-ci/master/docs/recipes/heroku-server/server. \  
js > server.js  
  
> git add --all && git commit -am "Initialize lhci" \  
&& git push origin master  
  
> heroku ps:scale web=1 && heroku open
```

LHCI Server



Welcome to Lighthouse CI!

Run `lhci wizard` to setup your first project.

Create LHCI Project

1. Install LHCI CLI
2. Run lhci wizard
3. Store the credentials safely

```
> npm i -g @lhci/cli  
> lhci wizard
```

```
? Which wizard do you want to run? New-project
```

```
? What is the URL of your LHCI server?
```

```
https://lhci-server-webfest.herokuapp.com/
```

```
? What would you like to name the project?
```

```
Twentytwentyone
```

```
? Where is the project's code hosted?
```

```
https://github.com/ivankristianto/twentytwentyone/
```

```
? What branch is considered the repo's trunk or  
main branch? trunk
```

```
Created project twentytwentyone  
(5bad793e-da68-45b2-a045-dbea42ce53bf)!
```

```
Use build token
```

```
f9c5a509-c539-4aa6-8ccb-f38a7c17ffab to add data.
```

```
Use admin token
```

```
UW4lyEHH7oiX58Cx2Jnuqh1wUkcITITVQc6RxFyp to manage  
data. KEEP THIS SECRET!
```

Lighthouse CI GitHub App

<https://github.com/apps/lighthouse-ci>



GitHub App

Lighthouse CI

Configure

Manage your installation settings.

Developer

 [patrickhulce](#)

 [Website](#)

Lighthouse CI is provided by a third-party and is governed by separate terms of service, privacy policy, and support documentation.

 [Report abuse](#)

Lighthouse CI posts the results of your [Lighthouse](#) runs in CI to PRs as separate status checks.

All checks have passed

3 successful checks



continuous-integration/travis-ci/push — The Trav...

[Details](#)



lhci/url/404.html — Performance: 97, Accessibility:...

[Details](#)



lhci/url/index.html — Performance: 96, Accessibili...

[Details](#)

Configure Repo Secrets

 LHCI_GITHUB_APP_TOKEN

 LHCI_SERVER_BASE_URL

 LHCI_TOKEN

GitHub Actions

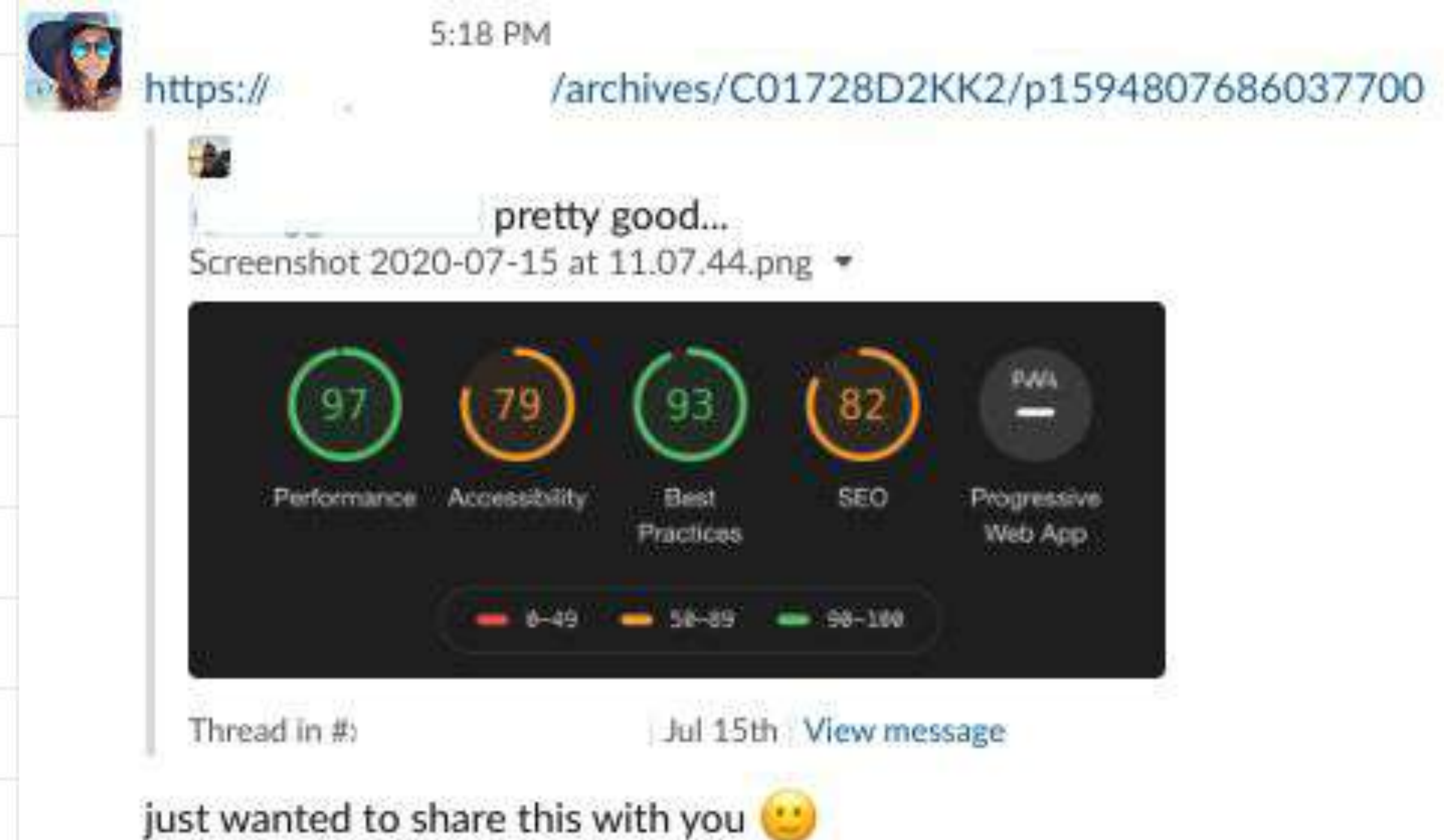
```
name: Lighthouse CI
on: [push,pull_request]
jobs:
  lhci:
    name: Lighthouse
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Use Node.js 12.x
        uses: actions/setup-node@v1
        with:
          node-version: 12.x
      - name: Install @lhci/cli and @wordpress/env
        run: |
          npm install -g @lhci/cli@0.5.x @wordpress/env
      - name: Run local server on port 8888
        run: |
          wp-env start
      - name: Run Lighthouse CI
        run: |
          lhci collect --url=http://localhost:8888 --numberOfRuns=1 --silent
      - name: Upload Artifact to Lighthouse CI Server
        run: |
          lhci upload
    env:
      LHCI_GITHUB_APP_TOKEN: ${secrets.LHCI_GITHUB_APP_TOKEN}
      LHCI_SERVER_BASE_URL: ${secrets.LHCI_SERVER_BASE_URL}
      LHCI_TOKEN: ${secrets.LHCI_TOKEN}
      - name: Assert Artifact
        run: |
          lhci assert
```


Let's See How It Works!

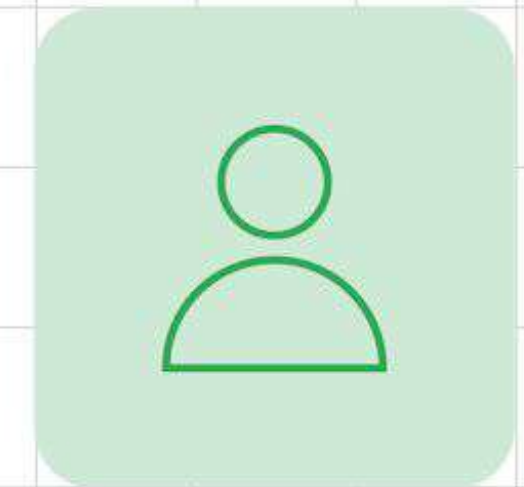
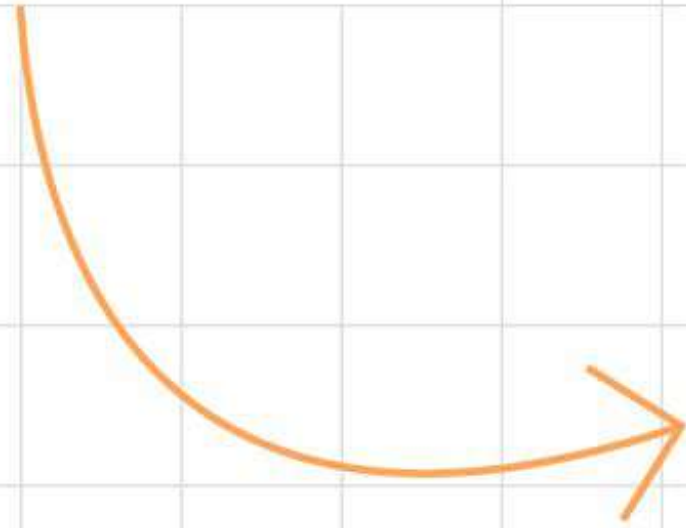
Result

Performance First Development,
make the performance as the
highest priority in all decisions
made, and implement it as part of
the workflow.

A good web performance have a lot
positive outcomes.



Google Developers



Thank You!



Ivan Kristianto
GDE Web and Performance
[@ivankrisdotcom / ivan@ivankristianto.com](mailto:ivan@ivankristianto.com)