

Contents

1. Theory	2
1.1 Google Chrome approach	2
1.2 Custom algorithm	3
1.3 Cache.....	4
1. API Documentation	4
2. Testing.....	5
2.1 Testing Results	6
3. How to run?	6

1. Theory

I decided to implement two approaches. They are completely different and have its' own pros and cons. Therefore, one approach is to use GoogleChrome debugging tools API, where we have a great tool for tracking `CssRuleUsage`, another approach is to calculate rule usage by custom algorithm.

1.1 Google Chrome approach

The idea of this approach is to load url inside google chrome, and get `CssRuleUsage` results via GoogleChrome debugging tools API.

To optimize performance and avoid memory leaks, I decided to implement *ChromeSessionPool.cs* class. During application startup (*dotnet run* command), we are starting chrome process and creating specified count of sessions, in our case = 10. Chrome Session Pool is configurable via `appsettings.json`:

```
"ChromeSessionPool": {  
  "ChromeDebuggingPort": 9222,  
  "IsHeadlessMode": true,  
  "WaitForInitializing": false,  
  "IsPreInitializeChromeSessionPool": true,  
  
  "MaxSessionPoolCount": 10,  
  //In seconds  
  "CommandTimeout": 120,  
  "RequestTimeout": 120  
},
```

`ChromeSessionPool` has two important methods *GetInstance* and *ReleaseInstance*. Therefore, when we have request to calculate min css for URL, one instance of `ChromeSession` will be locked, and in the end of calculation instance will be released.

If we will have a count parallel requests which are more than `ChromeSessionPool` count, then requests will be waiting for available session (Timeout for waiting is 120 seconds, "`RequestTimeout`" config value), after timeout, server will return error '`There is no free process to handle your request. Try again later, please.`'.

"`WaitForInitializing`" config value indicates if we want to wait when all chrome sessions will be created before API start.

"`IsPreInitializeChromeSessionPool`" – used for unit tests, where we starting `TestServer`.

For taking CSS Rule Usage, we are sending several commands to Chrome like active `CssRuleUsage` tracking and for receiving result we are able to send only '`TakeCoverageDelta`' method (per Chrome specification <https://chromedevtools.github.io/devtools-protocol/tot/CSS#method-takeCoverageDelta>)

So we really don't know when chrome CSS analyzing was ended, therefore I have custom solution, which could be improved (I believe):

In loop, we are sending 'TakeCoverageDeltaCommand' when delta = 0 we are waiting for Delta 3 times with different sleep time (waitCount* 100 milliseconds), so it's 100ms 200ms 300ms. If delta still 0, it means that chrome analyzing was ended. If during waiting delta became > 0, waitCount is resting.

Increasing waitCount and Sleep time will make this approach more accurate, but slower. And vice-versa, decreasing these values will make the result less accurate, but faster.

Advantages	Disadvantages
Support of all CSS features since we use native browser tools.	Performance... We really don't know when chrome coverage tool ended analyze of CSS, therefore sometimes we are having unnecessary 'sleep'.
More correct result.	Dependency on chrome.exe ...
	Chrome sessions eat RAM memory
	Parallel executing limited by Chrome Sessions count.

1.2 Custom algorithm

For calculating minimal css for given URL:

1. Receive html content of given URL.
2. Query all html 'style' attributes
3. Query all css 'link' elements
4. Load all css links in parallel
5. All css are gathered in one container, then parsed via ExCSS lib.
6. For each CSS Rule, we are trying to find HTML element using CSS Rule selector. If HTML element was found, this means that CSS Rule is being used on page, otherwise -> skip this rule.
7. All used CSS Rules are converted back to string and minified via NUglify lib.

Advantages	Disadvantages
Performance.	Less accurate. Since not all CSS features are supported by libs which are used to parse CSS rules...
Parallel executing are limited on thread level (so count of parallel threads are limited by OS)	Less accurate, in case of dynamic logic to load CSS (via javascript or else)
No dependency on other processes.	Again, less accurate

1.3 Cache

InMemory cache was used to save calculated results. Cache time is configurable via appsettings.config

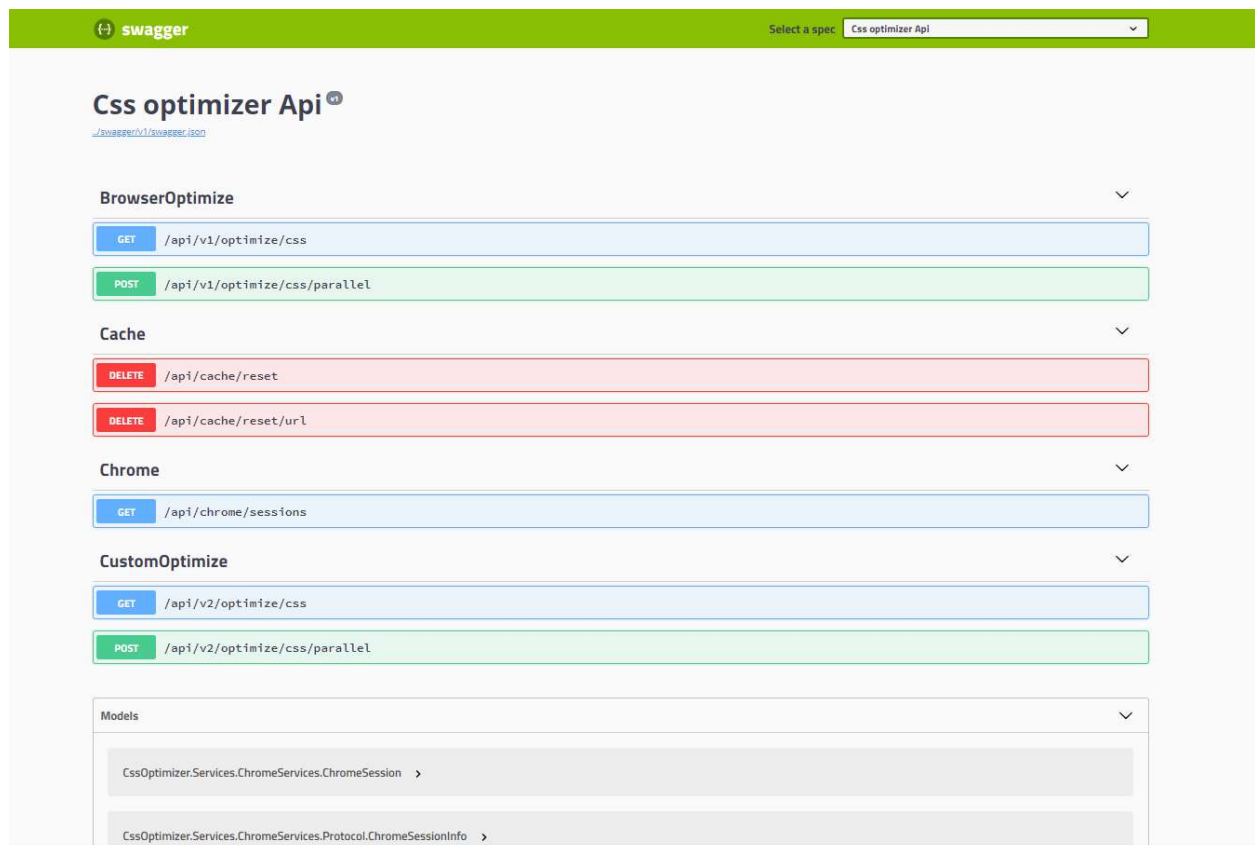
```
"Cache": {  
  //in seconds  
  "UrlCacheTime": 3600  
}
```

1. API Documentation

I am using Swagger. Go to this URL: <http://localhost:56587/swagger/>

/api/v1 – GoogleChrome approach

/api/v2 – CustomAlgorithm approach



The image shows the Swagger UI for the 'Css optimizer Api'. The top bar is green with the Swagger logo and a dropdown menu showing 'Select a spec' with 'Css optimizer Api' selected. The main title is 'Css optimizer Api' with a version 'v1' icon. Below the title, there are several expandable sections: 'BrowserOptimize', 'Cache', 'Chrome', 'CustomOptimize', and 'Models'. Each section contains a list of API endpoints with their HTTP methods and paths. The 'BrowserOptimize' section has two endpoints: GET /api/v1/optimize/css and POST /api/v1/optimize/css/parallel. The 'Cache' section has two endpoints: DELETE /api/cache/reset and DELETE /api/cache/reset/url. The 'Chrome' section has one endpoint: GET /api/chrome/sessions. The 'CustomOptimize' section has two endpoints: GET /api/v2/optimize/css and POST /api/v2/optimize/css/parallel. The 'Models' section is currently collapsed, showing a dropdown arrow.

swagger

Select a spec: **Css optimizer Api**

Css optimizer Api^{v1}

[Swagger UI](#)

BrowserOptimize

- GET /api/v1/optimize/css
- POST /api/v1/optimize/css/parallel

Cache

- DELETE /api/cache/reset
- DELETE /api/cache/reset/url

Chrome

- GET /api/chrome/sessions

CustomOptimize

- GET /api/v2/optimize/css
- POST /api/v2/optimize/css/parallel

Models

- CssOptimizer.Services.ChromeServices.ChromeSession >
- CssOptimizer.Services.ChromeServices.Protocol.ChromeSessionInfo >

2. Testing

CssOptimizer.Test – unit tests. For performance testing I wrote custom JavaScript file StressTests.js (which is inside CssOptimizer.Test library).

For executing stress tests go to <http://localhost:56587/swagger/> in console execute JavaScript which is in StressTests.js file and just run `testRunner.run()` command. (I was able to inject this into swagger, but really don't have time.....)

Example of output:

```
> testRunner.run()
CUSTOM ALGORITHM:
[/v2/optimize/css] Calculated min CSS for https://oikabu.ru/ for 4.361 second(s).
[/v2/optimize/css] Calculated min CSS for https://www.privat24.ua/#login for 0.887 second(s).
[/v2/optimize/css] Calculated min CSS for https://mail.google.com/mail/u/0/#search/has33anouserlabels for 0.639 second(s).
[/v2/optimize/css] Calculated min CSS for https://www.freelancer.com for 1.516 second(s).
[/v2/optimize/css] Calculated min CSS for http://olx.ua for 0.454 second(s).
[/v2/optimize/css] Calculated min CSS for http://youtube.com for 10.165 second(s).
[/v2/optimize/css] Calculated min CSS for https://outlook.live.com/owa/ for 1.802 second(s).
[/v2/optimize/css] Calculated min CSS for http://staff-clothes.com for 8.431 second(s).
Average time is 3.5318749999999994 second(s).
[/v2/optimize/css/parallel] Calculated min CSS for https://oikabu.ru/,https://www.privat24.ua/#login,https://mail.google.com/mail/u/0/#search/has33anouserlabels,https://youtube.com,https://outlook.live.com/owa/,http://staff-clothes.com for 9.772 second(s).
GOOGLE CHROME Approach:
[/v1/optimize/css] Calculated min CSS for https://oikabu.ru/ for 9.301 second(s).
[/v1/optimize/css] Calculated min CSS for https://www.privat24.ua/#login for 5.895 second(s).
[/v1/optimize/css] Calculated min CSS for https://mail.google.com/mail/u/0/#search/has33anouserlabels for 1.891 second(s).
[/v1/optimize/css] Calculated min CSS for https://www.freelancer.com for 4.945 second(s).
[/v1/optimize/css] Calculated min CSS for http://olx.ua for 3.576 second(s).
[/v1/optimize/css] Calculated min CSS for http://youtube.com for 4.251 second(s).
[/v1/optimize/css] Calculated min CSS for https://outlook.live.com/owa/ for 2.01 second(s).
[/v1/optimize/css] Calculated min CSS for http://staff-clothes.com for 4.65 second(s).
Average time is 4.564875 second(s).
[/v1/optimize/css/parallel] Calculated min CSS for https://oikabu.ru/,https://www.privat24.ua/#login,https://mail.google.com/mail/u/0/#search/has33anouserlabels,https://youtube.com,https://outlook.live.com/owa/,http://staff-clothes.com for 13.342 second(s).
< undefined
>
```

If you want to monitor chrome session pool, visit this url <http://localhost:9222/>. Example:

Inspectable WebContents

[Сервис объявлений OLX: сайт частных объявлений в Украине - купля/продажа б/у товаров на OLX.ua](#)

[YouTube](#)

[Приват24 - Ваш живой Интернет-банк!](#)

[about:blank](#)

[Staff - бренд качественной молодежной одежды.](#)

[about:blank](#)

[Outlook.com - Microsoft free personal email](#)

[Hire Freelancers & Find Freelance Jobs Online - Freelancer](#)

[Staff - бренд качественной молодежной одежды.](#)

[Gmail](#)

[about:blank](#)

[Service Worker https://www.youtube.com/sw.js](#)

[Service Worker https://www.youtube.com/sw.js](#)

2.1 Testing Results

Custom algorithm:

Synchronous execution for 8 urls took 28.255 seconds (average is 3.53 seconds).

Parallel execution for 8 urls took 9.772 seconds (289% faster).

GoogleChrome approach:

Synchronous execution for 8 urls took 36.5 seconds (average is 4.56 seconds).

Parallel execution for 8 urls took 13.342 seconds.

Need to understand that this time depends on count of CSS files on site. Youtube has a lot of CSS ☺

For custom algorithm:

[/v2/optimize/css] Calculated min CSS for <http://youtube.com> for 10.165 second(s).

For GoogleChrome:

[/v1/optimize/css] Calculated min CSS for <http://youtube.com> for 4.251 second(s).

Sometimes chrome is better. But for small sites, custom algorithm is always faster.

3. How to run?

Cd to `CssOptimizer.Api.csproj` file, and do `'dotnet run'` command.

Example output (please ignore warning, it's because I'm using package which is not support .net core):

```
D:\da\CssOptimizer\CssOptimizer>dotnet run
D:\da\CssOptimizer\CssOptimizer.Services\CssOptimizer.Services.csproj : warning NU1701: Package 'ExCSS 2.0.6' was restored using '.NETFramework,Version=v4.6.1' instead of the project target framework '.NETCoreApp,Version=v2.0'. This package may not be fully compatible with your project. [D:\da\CssOptimizer\CssOptimizer\CssOptimizer.Api.csproj]
D:\da\CssOptimizer\CssOptimizer\CssOptimizer.Api.csproj : warning NU1701: Package 'ExCSS 2.0.6' was restored using '.NETFramework,Version=v4.6.1' instead of the project target framework '.NETCoreApp,Version=v2.0'. This package may not be fully compatible with your project.
D:\da\CssOptimizer\CssOptimizer\CssOptimizer.Api.csproj : warning NU1701: Package 'ExCSS 2.0.6' was restored using '.NETFramework,Version=v4.6.1' instead of the project target framework '.NETCoreApp,Version=v2.0'. This package may not be fully compatible with your project.
D:\da\CssOptimizer\CssOptimizer.Services\CssOptimizer.Services.csproj : warning NU1701: Package 'ExCSS 2.0.6' was restored using '.NETFramework,Version=v4.6.1' instead of the project target framework '.NETCoreApp,Version=v2.0'. This package may not be fully compatible with your project.
Using launch settings from D:\da\CssOptimizer\CssOptimizer\Properties\launchSettings.json...
Hosting environment: Development
Content root path: D:\da\CssOptimizer\CssOptimizer
Now listening on: http://localhost:56588
Application started. Press Ctrl+C to shut down.
```

For running tests, cd to `CssOptimizer.Tests.csproj` file, and do `'dotnet test'` command.

Example output:

```
Starting test execution, please wait...
[xUnit.net 00:00:02.1998003] Discovering: CssOptimizer.Tests
[xUnit.net 00:00:02.4370423] Discovered:  CssOptimizer.Tests
[xUnit.net 00:00:02.4482489] Starting:    CssOptimizer.Tests
[xUnit.net 00:00:03.4034127]     CssOptimizer.Tests.BrowserOptimizeTests.BrowserOptimizeUrl [SKIP]
[xUnit.net 00:00:03.4039753]         Require more work to properly dispose chrome session pool
[xUnit.net 00:00:03.4235219]     CssOptimizer.Tests.BrowserOptimizeTests.BrowserOptimizeUrlInParallel [SKIP]
[xUnit.net 00:00:03.4236937]         Require more work to properly dispose chrome session pool
[xUnit.net 00:00:03.4242582]     CssOptimizer.Tests.BrowserOptimizeTests.BrowserOptimizeInvalidUrl [SKIP]
[xUnit.net 00:00:03.4244039]         Require more work to properly dispose chrome session pool
Skipped  CssOptimizer.Tests.BrowserOptimizeTests.BrowserOptimizeUrl
Standard Output Messages:
    Require more work to properly dispose chrome session pool
Skipped  CssOptimizer.Tests.BrowserOptimizeTests.BrowserOptimizeUrlInParallel
Standard Output Messages:
    Require more work to properly dispose chrome session pool
Skipped  CssOptimizer.Tests.BrowserOptimizeTests.BrowserOptimizeInvalidUrl
Standard Output Messages:
    Require more work to properly dispose chrome session pool
[xUnit.net 00:00:22.0729127] Finished:    CssOptimizer.Tests

Total tests: 20. Passed: 17. Failed: 0. Skipped: 3.
Test Run Successful.
Test execution time: 23,5833 Seconds
```

I was able to upgrade ExCSS package to version which supports .NET core, but there was one issue, I decided to not spend a lot of time. Therefore, sorry... you will be able to run it only on windows.