

თბილისის თავისუფალი უნივერსიტეტი

მათემატიკის, კომპიუტერული მეცნიერების და ინჟინერიის სკოლა (MACS[E])

კომპიუტერული მეცნიერებისა და მათემატიკის პროგრამა

მარიამ ჯავახიშვილი

ნინო გოგობერიშვილი

ლიზი ექსეულიძე

საბაკალავრო პროექტი

“TURBO”

ხელმძღვანელი: ალექსანდრე ხოხიაშვილი

თბილისი

2024

ანოტაცია

ვებ დეველოპმენტი უსწრაფესად მზარდი მიმართულებაა. ამ სფეროში ერთ-ერთი მნიშვნელოვანი ადგილი უჭირავს Chrome Extension დეველოპმენტს, რომელიც დღითიდღე უფრო და უფრო დიდი პოპულარობით სარგებლობს. შესაბამისად, დგას მნიშვნელოვანი საკითხი: როგორ შეიძლება გავზომოთ ამა თუ იმ extension-ის პერფორმანსი, მივცეთ დეველოპერებს საშუალება, შეაფასონ და გააუმჯობესონ თავიანთი პროდუქტი ამ კუთხით? სწორედ ეს ამოცანა დაისვა GSoC 2024-ზე Chromium-ის მიერ, რაც ამ პროექტის მთავარ იდეად იქცა. აპლიკაცია მიზნად ისახავს წვლილი შეიტანოს ექსთენშენ დეველოპერების მუშაობის პროცესის გაუმჯობესებაში, კერძოდ ექსთენშენის პერფორმანსის გაზომვასა და ვიზუალიზაციაში.

ვებ აპლიკაციებზე დამოკიდებულება დღევანდელ რეალობაში ძალიან დიდია. შესაბამისად მოთხოვნა უფრო სწრაფ, მომხმარებელზე მორგებულ ბრაუზინგ გამოცდილებაზე კრიტიკულია. თანამედროვე ვებ იუზერები ელიან მინიმალურ შეფერხებას და რესურსების ეფექტურ მენეჯმენტს, რაც პერფორმანსის გაუმჯობესებას განსაკუთრებულად საყურადღებო ასპექტად აქცევს. შესაბამისად ამ პროექტის რელევანტურობა თვალსაჩინოა.

ჩვენი პროექტის მიზანიც გახლავთ, შევქმნათ პრეფორმანსის ანალიზის თული, რომელიც მოერგება ექსთენშენ კოდის ჭრილს. პროექტის მიმდინარეობა მოიცავდა რამდენიმე მნიშვნელოვან ეტაპს: განგვესაზღვრა და შეგვესწავლა მეტრიკები, რომლის მონიტორინგიც იქნებოდა საბოლოო პროდუქტის ნაწილი, ექსთენშენის იუზერ-ინტერფეისის შერჩევა და უშუალოდ ძირითადი ფუნქციონალის იმპლემენტაცია.

ამ პროექტის ფარგლებში უპირველესად მოგვიწია Javascript და ექსთენშენის დეველოპმენტის საფუძვლების შესწავლა. რომლის გამოყენებითაც გუნდის თითოეულმა წევრმა შექმნა სატესტო ექსთენშენი. მოგვიწია Chrome DevTools პროტოკოლის გაცნობა, ასევე devtools frontend-ის გაანალიზება. თუ როგორაა ეს ელემენტები ერთმანეთში ინტეგრირებული, რათა შერჩეულიყო ოპტიმალური გზა რელევანტური ფუნქციონალის იმპლემენტაციისთვის.

საბოლოოდ, პროექტის ფარგლებში შეიქმნა ექსთენშენი რომელსაც აქვს რამდენიმე ძირითადი ფუნქციონალი:

1)Extension code coverage(კოდის დაფარვის ანალიზი)

2)Network usage monitoring(ქსელური მოხმარების მონიტორინგი)

3)კოდის შესრულების რეალურ დროში მონიტორინგი profiling

4) ინფორმაციის ვიზუალიზაცია Flame Graph საშუალებით.

ეს ყოველივე ინტეგრირებული იქნა Chrome-ის არქიტექტურაში DevTools პანელის სახით. გაიტესტა როგორც მანუალურად ასევე იუნიტ ტესტების საშუალებით. მომხმარებელს საშუალება ექნება ექსთენშენის pop up საშუალებით მონიშნოს მის ბრაუზერში დამატებული ექსთენშენებიდან ერთ-ერთი და შესაბამის ფუნქციონალისთვის განკუთვნილ ღილაკზე დაჭერისას დაიწყოს პერფორმანსის გაზომვა. დასტოპების შემთხვევაში კი მიიღოს შესაბამისი შედეგი, რომელიც შეუძლია გამოიყენოს ექსთენშენის კოდის ე.წ. bottleneck-ის აღმოსაჩენად, coverage-ის სანახავად თუ სხვა.

Annotation

Web development is the fastest growing field. One of the important places in this area is occupied by Chrome Extension development, which is gaining more and more popularity every day. Therefore, there is an important question: how can we measure the performance of this or that extension, allowing developers to evaluate and improve their product in this regard? It was this task that was set at GSoC 2024 by Chromium, which became the main idea of ​​this project. The application aims to contribute to the improvement of the work process of extension developers, namely the measurement and visualisation of extension performance.

Dependence on web applications is very high in today's reality. Consequently, the demand for a faster, more user-friendly browsing experience is critical. Modern web users expect minimal latency and efficient resource management, making performance improvements a particularly important aspect. Therefore, the relevance of this project is obvious.

The goal of our project is to create a performance analysis tool that fits the extension code slot. The progress of the project included several important stages: we defined and studied the metrics that would be monitored as part of the final product, the selection of the user-interface of the extension and the implementation of the main functionality directly.

Within this project, we first had to learn the basics of Javascript and extension development. using which each team member created a test extension. We had to learn the Chrome DevTools protocol as well as analyze the devtools frontend. How these elements are integrated together to select the optimal way to implement the relevant functionality.

Finally, within the framework of the project, an extension was created that has several basic functionalities:

1) Extension code coverage (code coverage analysis)

2) Network usage monitoring

3) real-time monitoring of code performance profiling

4) Visualisation of information through Flame Graph.

All of this has been integrated into Chrome's architecture in the form of the DevTools panel. Tested both manually and through unit tests. The user will be able to mark one of the extensions added to his browser through the extension pop-up and start measuring the performance by pressing the button for the corresponding functionality. In the case of desktops, get the appropriate result, which can be used by the so-called extension code. To detect a bottleneck, to view coverage or other.

სამადლობელი

გვსურს, მადლობა გადავუხადოთ ყველას, ვინც წვლილი შეიტანა პროექტის განვითარებაში. განსაკუთრებული მადლობა ჩვენი პროექტის ხელმძღვანელს ალექსანდრე ხოხიაშვილს მენტრობისა და მხარდაჭერისთვის, საჭირო მასალების მოწოდებისთვის. მადლობა გუნდის თითოეულ წევრს. თავისუფალი უნივერსიტეტის ლექტორებს, რომელთაც მოგვცეს საშუალება მიგვეღო პროექტისთვის საჭირო ცოდნა და შეგვეძინა ახალი უნარები, განსაკუთრებით იმ საგნებიდან, როგორებიცაა: Design Patterns (პროგრამული უზრუნველყოფის არქიტექტურა) და პროგრამული უზრუნველყოფის ინჟინერია. ძალიან ვაფასებთ იმ ადამიანების თანადგომას, რომლებიც მთელი ამ ხნის განმავლობაში გვაძლევდნენ მოტივაციას.

სარჩევი

1. შესავალი
2. ძირითადი ნაწილი
   1. პროექტის სფერო
   2. ტექნიკური ამოცანა
   3. პოტენციური განვითარება
   4. პროექტის ტექნიკური მახასიათებლები
      1. Code Coverage
      2. Flame Graph
      3. Network Monitoring
3. შედეგები
4. დასკვნა
5. გამოყენებული ლიტერატურა
6. დანართი

სურათები

სურათი 1. DevTools client interacting with Chromium Layers

სურათები 2 და 3. *kExtensionsOnChromeURLS* flag

სურათი 4. Logo mock ups

სურათი 5. Draft front-end

სურათი 6. შერჩეული ფერები ფრონტ-ენდისთვის

სურათი 7.ქრომის ექსთენშენი აგებულება და კომუნიკაცია რესურებს შორის

სურათი 8.Color Picker

სურათი 9. Შეგროველი მონაცემების mock data

სურათი 10.საჭირო permissionების ერთ-ერთი მაგალითი

სურათი 11. coverage-ით მირებული შედეგი და სტატისტკური მაჩვენებლი

სურათი 12. Coverage-ით ფაილის ანალიზი bottleneckების აღმოსაჩენად

სურათი 13. Coverage-ით მიღებული მონაცემები სატესტო ექსთენშენზე

სურათი 14 Torrent scannerის coverage სტატისტიკა

სურათი 15. profiler შედეგი

სურათი 16. ტრანსფორმირებული დატა

სურათი 17.ექსთენშენის Pop up. მომხმარებელმა უნდა აირჩიოს ის ექსთენშენი რომლის დებაგირებაც სურს

სურათი 18. მომხმარებელი ირჩევს საკუთარი ექსთენშენების ჩამონათვალიდან ერთერთს.

სურათი 19. მომხმარებელს შეუძლია ექსთენშენის ID ნახვა pop up-ზე.

სურათი 20. პანელის საწყისი ფორმა

სურათი 21. პანელი ერთ-ერთი ფუნქციონალის გაშვების პროცესში

სურათი 22. Coverage შედეგები

სურათი 23. სკრიპტის გახსნა/დაჰაილაითებული ნაწილები დაფარვის მიხედვით

სურათი 24. ფლეიმგრაფის დაზუმების შედეგი

სურათი 25. ტაბის შემთხვევაში ექსთენშენის მიერ გაშვებული ფუნქციები ვიზუალურად

სურათი 26. Network ცხრილი

სურათი 27. Network Waterfall Chart Hover / Closeup

სურათი 28. Network requestLargeResource closeup

სურათი 29. ინფორმაცია ვერ მოიძებნა

სურათი 30. იუზერის ექსთენშენი არ ეშვება

შესავალი

პროექტის თემის შერჩევისას განსაკუთრებული ყურადღება მივაქციეთ პროექტის შედეგის გამოყენებადობას. დაინტერესება გამოიწვია იმანც, თუ რამდენად ცოდნისმომცემი იქნებოდა პროექტის დაგეგმვა და განხორციელება. შევჯერდით GSoC 2024-ზე Chromium-ის მიერ წარმოდგენილ ერთ-ერთ იდეაზე: “Chrome Extension Performance Analysis Tool”. გადაწყვეტილებაზე მნიშვნელოვანი გავლენა იქონია იმ ცოდნამ რომელსაც ამ პროექტის შედეგად მივიღებდით საბოლოო პროდუქტთან ერთად.

პროექტის ფარგლებში შეგვეძლებოდა გვესწავლა ჩვენთვის უცნობი, მაგრამ მსოფლიოში ერთ-ერთი წამყვანი პროგრამული ენა Javascript, ასევე, გავცნობოდით ექსთენშენ დეველოპმენტის სპეციფიკას, ასევე საშუალება მოგვეცემოდა გავცნობოდით Chrome devtools API-ს დოკუმენტაციას და ჩვენი ფუნქციონალის შესაბამისად შერჩეული მიდგომა გადაგვეხედა და გაგვეუმჯობესებინა.

ვებ დეველოპერებს აქვთ წვდომა ინსტრუმენტების ფართო სპექტრზე, რათა თვალყური ადევნონ ვებ აპლიკაციების ეფექტურობას და ოპტიმიზაცია გაუკეთონ საკუთარ პროგრამებს. ექსთენშენ დეველოპერებისათვის მსგავსი მხარდაჭერა საკმაოდ მწირია, თუმცაღა მათი წვლილი მომხმარებლის გამოცდილებაში ისეთივე კრიტიკულია როგორც თავად ვებ აპლიკაციის. ექსთენშენი ქრომის უნიკალურ და თავისზე მორგებულ გამოცდილებას სთავაზობს მილიონობით მომხმარებელს - მათ საშუალებას აძლევს, ბრაუზერი თავიანთ პირად მოთხოვნილებებსა და სპეციფიკურ საჭიროებებს მოარგონ.

შესაბამისად, ეს პროექტი გათვლილია ყველა იმ დამწყები თუ გამოცდილი დეველოპერისთვის, ვისაც სურს შექმნას საკუთარი ექსთენშენი და გაიგოს, ჩვენს მიერ მოწოდებული user-friendly ვიზუალით და ინსტრუმენთებით, თავისი ექსთენშენის პერფორმანსი როგორც კოდის ოპტიმიზაციის ასევე მომხმარებელთათვის მაქსიმალურად კომფორტული პროდუქტის შექმნის მიზნით.

საწყის ეტაპზე მთავარი იყო შედეგისთვის შესაბამისი მიდგომის მოძიება და გადარჩევა. საუკუთესო მიდგომა აღმოჩნდა პროექტის ექსთენშენის სახით შექმნა, რადგან საბოლოო პროდუქტი იქნებოდა როგორც ბრაუზერში ინტეგრირებული, ასევე მომხმარებელთათვის ხელმისაწვდომი.

პროექტი ექსთენშენისთვის განკუთვნილ პროტოკოლს არის მორგებული. ასევე, ფრონტენდის და ბექენდის მხარეებს შორის კომუნიკაცია შეტყობინებების(messaging) გავლით მყარდება. თითოეული მხარე ფუნქციონალის მიხედვით იშლება ლოგიკურ ერთეულებად, მაგალითად, ბექენდში ანალიტიკური ინფორმაცია მიიღება debugger API-ს შესაბამის domain-თან კავშირებით.

გარკვეული შეზღუდვები შექმნა ექსთენშენის საშუალებით ბრაუზერის low-level ინფორმაციაზე წვდომისას. ვინაიდან ექსთენშენები უსაფრთხოების მიზნით მკაცრად არის გაკონტროლებული უშუალოდ დეველოპმენტის პროცესში შეგვექმნა შეზღუდვა დებაგინგისთვის საჭირო ფერმიშენებთან დაკავშირებით. ასევე, ექსთენშენისთვის განკუთვნილ devtools API-ს მეთოდების დოკუმენტაცია საკმაოდ მწირია, ან არ არსებობს გამოყენების მაგალითები, ან არსებული მეთოდები კონკრეტულ ექსთენშენის ვერსიისათვის მოძველებულია და ჩამნაცვლებელიც არ ჰყავს. საჭირო ფლეგების თუ სხვა დეტალების შესახებ მომხმარებელს შეუძლია დეტალური ინფორმაცია უშუალოდ პროექტის რეპოზიტორიაში იხილოს.

პროექტის სფერო

ქრომის ექსთენშენებს ჯამურად საკმაოდ დიდი მომხმარებელთა ბაზა აქვს. Google Chrome, 2023 წლისთვის 3 მილიარდზე მეტი მომხმარებელი ჰყავდა(Dean, 2024). 2021 წლის მონაცემებით არსებობს 130,445(“Counting Chrome Extensions – Chrome Web Store Statistics,” 2024b) ქრომის ექსთენშენი, რომელთა შორისაც არის ისეთი წარმატებული ექსთენშენები როგორიცა AdBlocker, Grammarly და სხვა.

ქრომის ექსთენშენები იქმნება HTML, CSS, JavaScript-ის საშუალებით, რითიც უფრო ხელმისაწვდომი ხდება ვებ დეველოპერებისათვის, რომლებიც უკვე მუშაობენ ამ ენებზე. თუმცა, ექსთენშენის დეველოპმენტს სჭირდება თავის ცოდნა Chrome Extension API, რომელიც შექმნილია ქრომის ბრაუზერსა და მის კომპონენტებთან ინტერაქციისათვის.

ქრომის ექსთენშენები არის ქრომის ეკოსისტემის განუყოფელი ნაწილი, რომელიც სთავაზობს იუზერეებს ფართო სპექტრის ფუნქციონალს. მათი ფართო გამოყენება ხაზს უსვამს მათ მნიშვნელობას ვებ განვითარებისა და მომხმარებლის გამოცდილების ფართო კონტექსტში.

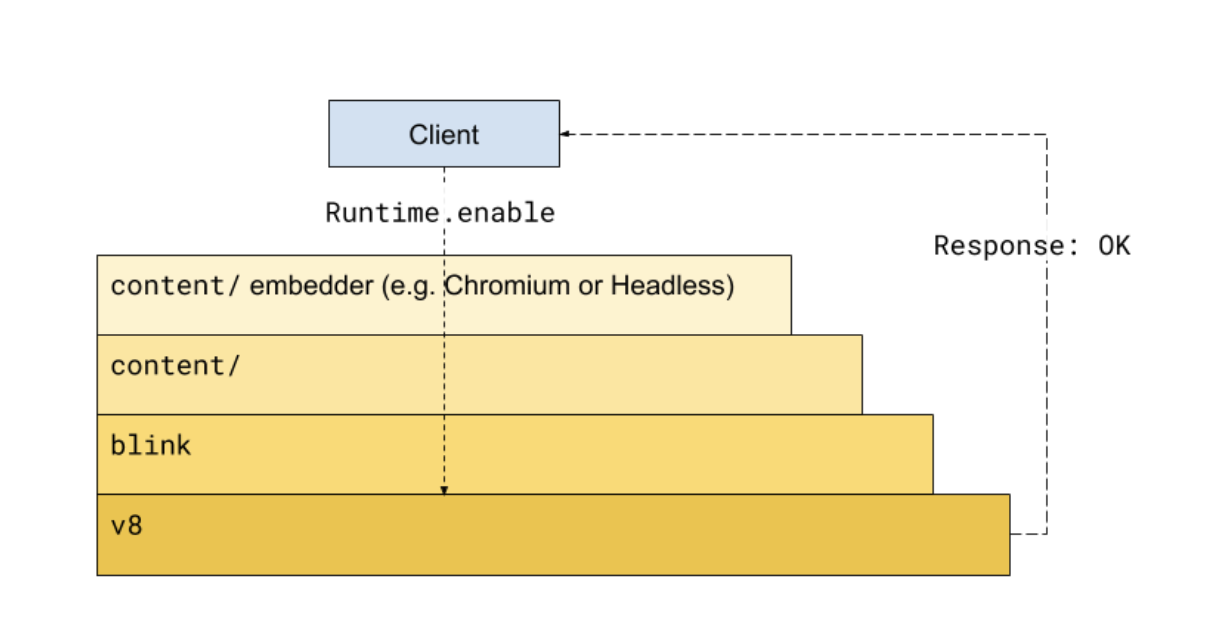
ერთ-ერთი მნიშვნელოვანი გამოწვევა ამ მიმართულებით არის სკრიპტების პერფორმანსის განსაზღვრა, როგორც background სკრიპტების ასევე content სკრიპტების. ასევე ამ შედეგების წამოჩენა user-friendly ვიზუალის საშუალებით, გადატვირთული ელემენტების გარეშე, გარდამტეხია. როგორც დეველოპერს ხშირ შემთხვევაში გჭირდება სწრაფად დაადეტექტო პრობლემა და ეფექტურად აღმოფხვრა პერფორმანსის ნაკლოვანება.

ექსთენშენ დეველოპმენტის განვთარებაში მნიშვნელოვანი წვლილი იქნებოდა უშუალოდ დეველოპერებისთვის კომფორტის შექმნა რომ უფრო მაღალი ხარისხი პროდუქტი შესათავაზონ მომხმარებელს. რასაც პირველ რიგში მათი პროდუქტის პერფორმანსი განსაზღვრავს.

აქამდე არსებული ხელსაწყოები საკმაოდ ბევრი ნაკლით გამოირჩევა და არასტაბილურია. კოდის კრიტიკული ნაწილის დეტექცია თავის მხრივ აუმჯობესებს ეფექტურობას, პოტენციურად უფრო სასურველს ხდის პროდუქტს და ამყარებს მომხმარებლის გადაწყვეტილებას არჩევანი სწორედ ყველაზე კომფორტულ, შეყოვნებებისგან დაცულ ექსთენშენს მიანიჭოს.

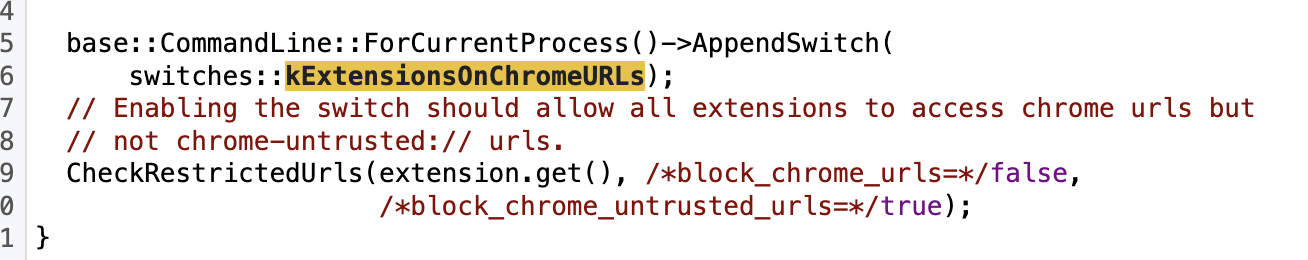
ტექნიკური ამოცანა:

პირველივე ამოცანა რასაც პროექტის დაგეგმვისას გადავაწყდით იყო მისი არქიტექტურის და სტრუქტურის განსაზღვრა, რაც მოიცავდა შემდეგ კითხვებს: რა სახით უნდა ყოფილიყო აპლიკაცია მომხმარებლებისთვის მიწოდებული? რა ტექნოლოგიები დასჭირდებოდა? Შესაბამისად წამოიჭრა შემდეგი იდეები: Chromium-ის Open Source კოდშივე გაგვეკეთებინა ჩვენი პროექტის ინტეგრაცია და პირდაპირ ჩაშენებული ყოფილიყო ისეთი სახით როგორც ამჟამად Chrome Dev Tools-ის პანელია, ყოფილიყო ცალკეული აპლიკაცია (მაგ.: Python) ან მიგვეცა ისევ Chrome Extensionის სახე.

აღსანიშნავია რომ ქრომის არქიტექტურა შედგება შემდეგი ფენებისგან: chrome/, headless/, content/, blink/, v8/. გასააზრებელი იყო javascript execution პროცესი ბრაუზერში და მასში ჩართული layer-ები, მეთოდი, თუ როგორ უნდა შეგვეძლო შესაბამის ფენასთან კომუნიკაციის დამყარება ჩვენი პროექტის კოდის იმპლემენტაციისთვის, მოგვიწია ყოველივე ვარიანტის ანალიზი და გადარჩევა, შევისწავლეთ როგორც DevTools არქიტექტურის UI, ისე მისი პროტოკოლი და ბრაუზერთან ინტერაქციის ლოგიკა .  


Სურათი 1. DevTools client interacting with Chromium Layers (*Contributing to Chrome DevTools Protocol*, n.d.)

ერთ-ერთი ყველაზე საჭირო ინფორმაცია რაც Chromium-ის source კოდის ანალიზმა მოგვცა იყო საჭირო permission flag-ების შესწავლა: ქრომის გარემოში არსებული ექსთენშენის აპლიკაციის დებაგირებისა და დატას შეგროვებისთვის. მაგალითად, თავდაპირველად პრობლემა იქმნებოდა ჩვენი extension მხრიდან target extension-ის script-ზე დებაგერის მიბმისას. Manifest v3 დოკუმენტაციაში დასახელებული flag მოძველებული აღმოჩნდა, პრობლემის გადაჭრა შევძელით ქრომიუმის source კოდის ანალიზით, როდესაც აღმოვაჩინეთ რომ კონკრეტული flag მოწმდებოდა permissions ტესტებში:





სურათები 2 და 3. *kExtensionsOnChromeURLS* flag *(Chromium Project, n.d)*

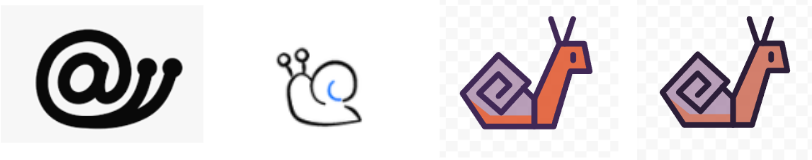
აღნიშნული flag-ის დასეტვით წარმატებით შევძელით დებაგერის სხვა extension-ის worker-ზე მიბმა. აგრეთვე ქრომიუმის კოდის ანალიზი გამოგვადგა ზოგადად DevTools Protocol API -ების გამოყენების მაგალითების საპოვნელად. დაგვჭირდა დოკუმენტაციის ანალიზი იმის დასადგენად, რომ extension-ებისთვის DevTools პროტოკოლი ხელმისაწვდომია chrome.debugger API გავლით.

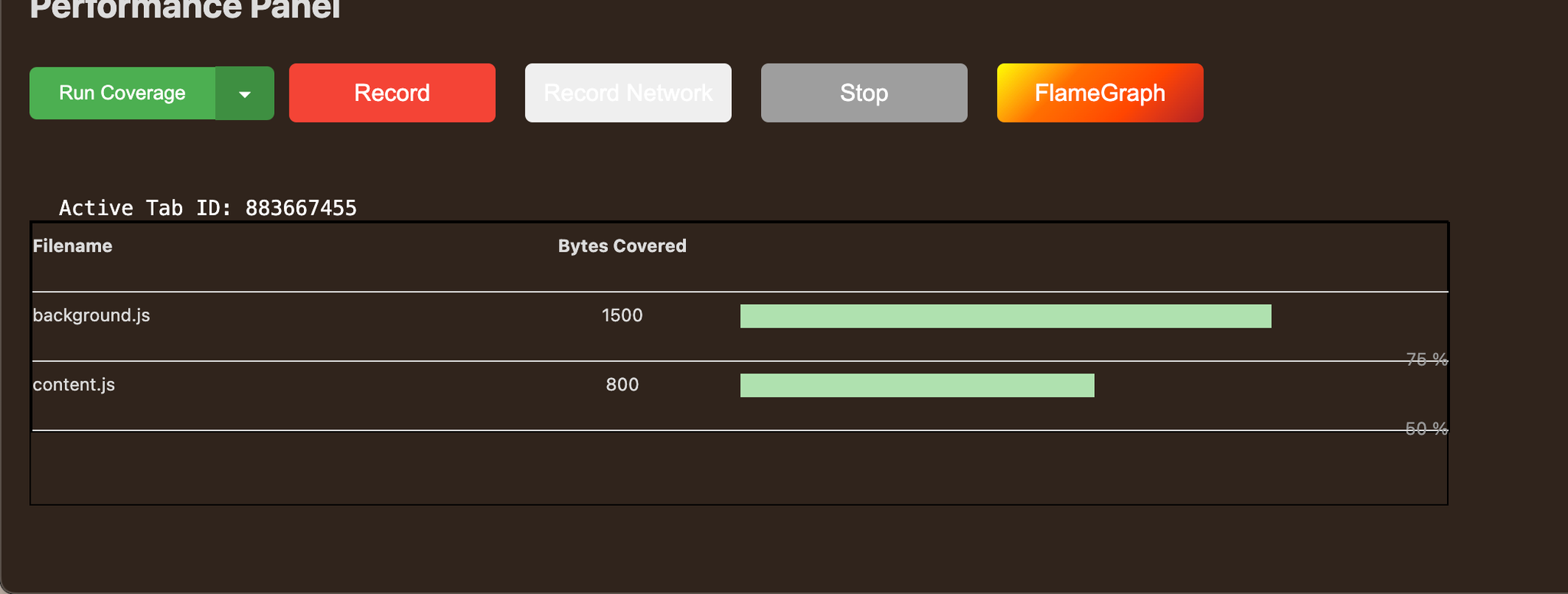
საბოლოოდ გადავწყვიტეთ, რომ ჩვენი პროექტისთვისაც ექსთენშენის სახე მიგვეცა ორი მიზეზის გამო: როგორც პროდუქტი ბევრად უფრო მარტივი გამოსაყენებლი იქნებოდა მომხმარებლებისთვის და ჩვენთვისაც საინტერესო გამოწვევა იქნებოდა.

შემდეგი ამოცანა რისი გადაჭრაც მოგვიწია იყო რა ტიპის ექსთენშენი იქნებოდა და რა ელემენტების გამოყენება შეგვეძლო. ქრომის კოდთან ინტეგრაციის ალტერნატივად developer tools-ის პანელში ექსთენშენის გაშვება იქცა და pop up ელემენტს განსხვავებული ფუნქციონალი მიენიჭა. Გადავწყდით პრობლემებებს, რომელიც ექსთენშენის დეველოპმენტთან იყო დაკავშირებული, როგორიცაა: deprecated manifest protocols, დოკუმენტაციის ან ბიბლიოთეკების ნაკლებობა, ფაილებს შორის კომუნიკაციის დამყარება, ქრომის მეხსიერების სწორი გამოყენება და კომპილაციის განსხვავებული ტექნიკები.

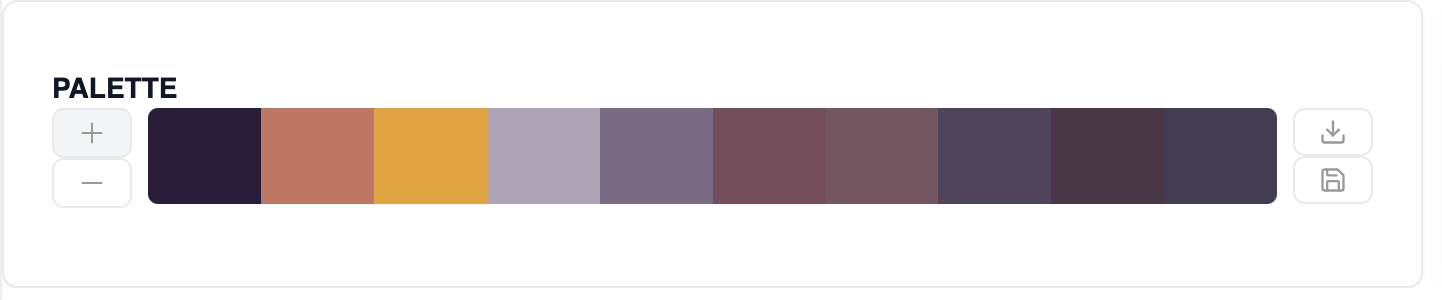
გარკვეული დრო დაიხარჯა ასევე UI/UXის ჩამოყალიბებასა და დახატვაზეც. მიუხედავად იმისა რომ პროექტი front-end-ს გადამწყვეტი პრიორიტეტი არაა, მაინც არსებობდა ელემენტები და კომპონენტები რისი გათვალისწინებაც საჭირო აღმოჩნდა. Საბოლოო პროდუქტის შესაქმელად აუცილებლად ჩავთვალეთ მომხმარებლისთვის გაგვემარტივებინა აპლიკაციის მოხმარება და მისი ფუნქციონალის ბოლომდე აღქმა.

სურათი 4. Logo mock ups:

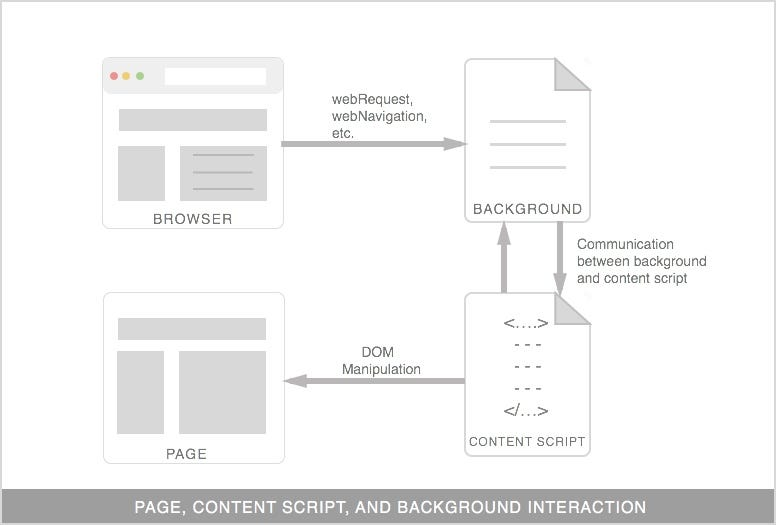


სურათი 5. Draft front-end:

სურათი 6. შერჩეული ფერები ფრონტ-ენდისთვის:



აღსანიშნავია ასევე შეზღუდვები რომელიც ექსთენშენის დატას შეგროვებას ეხებოდა. აპლიკაცია შედგება ორი ფუნქციური ნაწილისგან: service worker-ს და content script-ებს, რომლებიც ერთმანეთის დამოუკიდებლად ეშვება. Შესაბამისად მოგვიწია მოგვეფიქრებინა, რა სახით შეიძლებოდა ორივე კოდის მონაცემების მოპოვება: ერთდროულად ან განცალკევებით. Პრობლემის გადაჭრის გზა აღმოჩნდა ახალი ფუნქციონალის დამატება, რომელიც მომხმარებელს საშუალებას აძლევს თავისით აირჩიოს სასურველი კომპონენტი.

სურათი 7.

პოტენციური განვითარება

ამ დროისთვის კოდი გათვლილია რომ იყოს open source და review ფაზაშია ოფიციალური chrome extension storeზე განსათავსებლად, რადგანაც ეს ხანგრძლივი პროცესია და მოითხოვს ექსთენშენის შეფასებას, მისი permissionების შემოწმებასა და მხოლოდ შემდგომ დადასტურებას. თავდაპირველად ხელმისაწვდომი იქნება ნებისმიერი იუზერისთვის, რათა შევძლოთ უფრო მეტი მომხმარებლისგან უკუკავშირის მიღება და სტატისტიკური შეფასება: რას ექნება ყველაზე ხშირი გამოყენება და მოთხოვნა. Ამის შემდგომ შესაძლებელი გახდება subscription tierების შემოღება რაც მოიცავს, ზოგიერთი feature-ის მხოლოდ გამომწერი მომხმარებლებისთვის მიწოდებას, ულიმიტოდ გამოყენების უფლებას და სხვა ბენეფიტებს. Მოხდება ასევე frontის დახვეწა და შესაძლოა თავად კომპანიებისთვის შეთავაზება, გაყიდვა, რომელთაც აქვთ სხვადასხვა კომერციული ექსთენშენი. Ამის გათვალისწინებით გაკეთდეს company workspace-ებიც და სხვა feature-ებიც, რასაც მხოლოდ subscription-ის საშუალებით მოიხმარენ.

ტექნიკური მხარე

ამ ამოცანების გადასაჭრელად დაგვჭირდა რამდენიმე ძირითადი ტექნოლოგია. პირველ რიგში Chrome DevTools Protocol, რომელიც უზრუნვეყოფს წვდომას ბრაუზერის low-level დონეებზე და JavaScript-ის პერფორმანსის პროფაილინგის ტექნიკებს პროექტის ყველა ფუნქციონალისთვის. საყურადღებოა ისიც, რომ ამ პროტოკოლის მეთოდები და დოკუმენტაცია გათვლილია თავად ქრომიუმის შიდა მოხმარებისთვის და არ არის ადაპტირებული ფართო მასშტაბებზე გამოსაყენებლად. დამატებით დაგვჭირდა მოვრგებულიყავით ექსთენშენის პროტოკოლის Manifest v3 წესებს, რომელიც აწესებს შეზღუდვებს იუზერების უსაფრთხოებისთვის. ასევე კონკრეტულ ფუნქციონალთან მუშაობისას გამოვიყენეთ ბიბლიოთეკები ვიზუალიზაციისათვის.

პროექტის კვლევის ნაწილში გუნდის თითოეულმა წევრმა შექმნა სატესტო ექსთენშენები, რომელიც მოიაზრებდა background ფონის/Content-ის კავშირების, ინფორმაციის შენახვის და სხვა სპეციფიკის გაცნობას, და შესაძლო შეზღუდვების ამოცნობას ძირითადი პროექტისთვის.

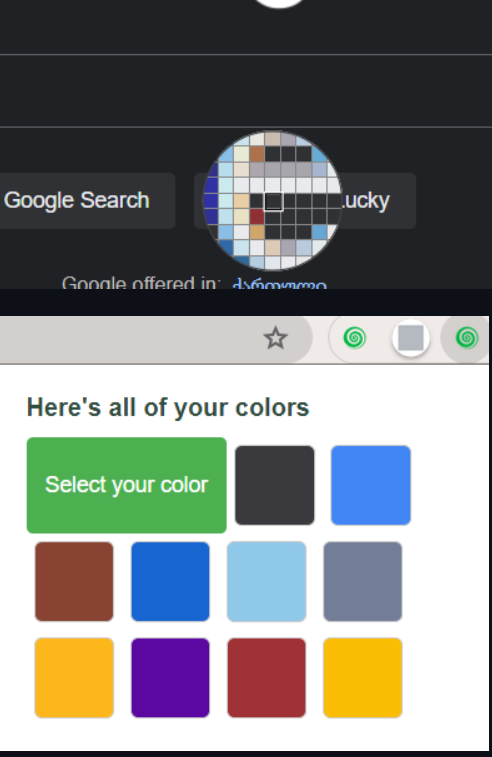
**Request Blocker** - ექსთენშენი, რომელიც ბლოკავს image რესურსებზე გაგზავნილ request-ებს ბრაუზერში. მისი შექმნის მოტივაცია იყო იმის გამოკვლევა, თუ როგორი მექანიზმით შეიძლება ჰქონდეს წვდომა ექსთენშენებს ბრაუზერში გაგზავნილ ქსელურ request-ებზე, რაც გამოსადეგი იქნებოდა შემდგომ პროექტში network მონიტორინგის იმპლემენტაციისთვის. ასევე მოტივაცია იყო იმის მარტივ დონეზე გააზრება, თუ რა პრინციპით მუშაობს, მაგალითად, ცნობილი ექსთენშენი Adblocker. აღმოჩნდა, რომ chrome.declarativeNetRequest API გამოყენებით მარტივია რექვესთებზე გარკვეული ruleset-ების დაყენება და მათი დაბლოკვა/მოდიფიცირება. (მუშაობდა ლიზი ექსეულიძე)

**Bookmarker** - ექსთენშენს აქვს საბაზისო ფრონტის ნაწილი, რომელიც ჩნდება pop upის სახით მას შემდეგ რაც იუზერი გააქტიურებს მას. იგი საშუალებას იძლევა შეინახოს სასურველი web-pageები ექსთენშენის კონტექტსში, რომელიც თავისთავად იყენებს chromeის მეხსიერებას ანუ local storageს. ასევე აქვს მოდიფიკაციის ფუნქციონალიც. Საჭირო წვდომები: tabs

(მუშაობდა ნინო გოგობერიშვილი)

**Color Picker**-ექსთენშენი საშალებას გაძლევს ამოირჩიო და დაიმახსოვრო სასურველი ფერი ვებ-გვერდიდა. შენი არჩეული ფერი დაკლიკების შემდეგ ემათება ექსთენშენის popup-ში მეტი მარტივად წვდომისთვის, ინახება იმ ფერში რა ფერიც მონიშნე, შესაბამისად ინახება თავად ფერის ჰექს კოდი დეველოპმენტის პროცესში საჭირო იყო წვდომები - storage, tabs, host\_permissions. (მუშაობდა მარიამ ჯავახიშვილი)

სურათი 8.Color Picker



**Test Profiler** - ექსთენშენი შეიქმნა ერთგვარ პირველად ნიმუშად პროექტის backend ლოგიკის იმპლემენტაციისთვის, კვლევის პროცესში აქ იტესტებოდა Debugger API-ის სხვადასხვა დომეინზე წვდომა და მათი გამოყენება, საჭირო მონაცემების შეგროვება და მათი ანალიზი, რის შედეგადაც დადგინდა რომ შესაბამისი permission-ებით ექსთენშენის კონტექსში ხელმისაწვდომი იყო chrome.debugger.Profiler / chrome.debugger.Network დომეინები, რომლებიც საკმარის ინფორმაციას იძლეოდა პროექტის ძირითადი ფუნქციონალის იმპლემენტაციისთვის, საბოლოო მიდგომად სწორედ ამაზე დაყრდნობით შეირჩა პროექტის extension ფორმით შექმნა. (მუშაობდა ლიზი ექსეულიძე)

Code coverage

Coverage-ის ფუნქციონალი გულისხმობს კოდის ბაიტების რაოდენობას დათვლას რომელიც მონიშნულ დროის მონაკვეთში დაიფარა. Როგორც სხვა დანარჩენი, ესეც გაყოფილია ორ ნაწილად: Tabის მონიტორინგისთვის (სადაც content-scripts ეშვება) და თავად ექსთენშენის backgroundსთვის, ანუ service workერის მონაცემების შეგროვება. Back-endში გამოყენებულია chrome.profilerის ბიბლიოთეკა, კეთდება მონაცემთა შეგროვება გაშვების მომენტიდან (startPreciseCoverage, takePreciseCoverage), რომელსაც შემდგომ ანალიზს, ფილტრაციასა და სხვადასხვა ტრანსფორმაციას ვუკეთებთ. შეგროვებულ დატას აქვს შემდეგი სახე (იხ. სურათი 9).

სურათი 9.

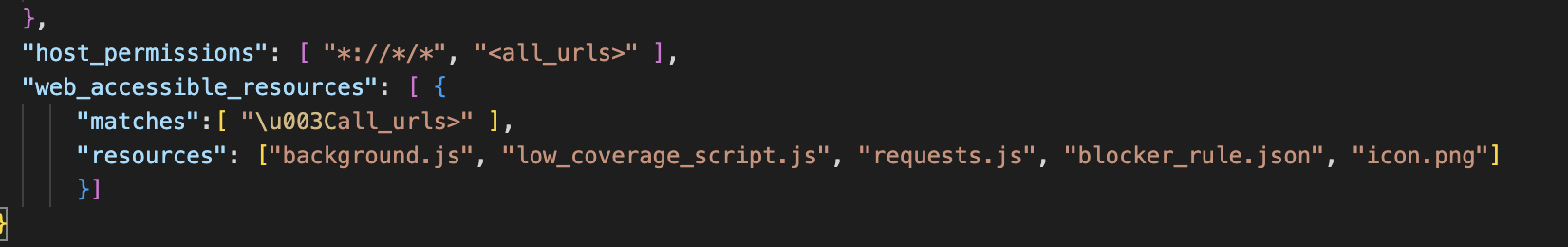


Urlის მიხედვით იფილტრება მხოლოდ ექსთენშენები და მას აქვს შემდეგი სახლის სტრუქტურა: “chrome-extension://<extension-id>/<file-name>”

Აღსანიშნავია, რომ ხშირ შემთხვევაში მიღებული დატას rangeები არ იყო თანაუკვეთი და გვქონდა მასივი სხვადასხვა შუალედებით, რომელთა გაფილტვრაც და უნიკალური ბაიტების რაოდენობის დათვლაც გახდა საჭირო, რაც ალგორითმული ამოცანაზე დავიდა და ამოიხსნა.

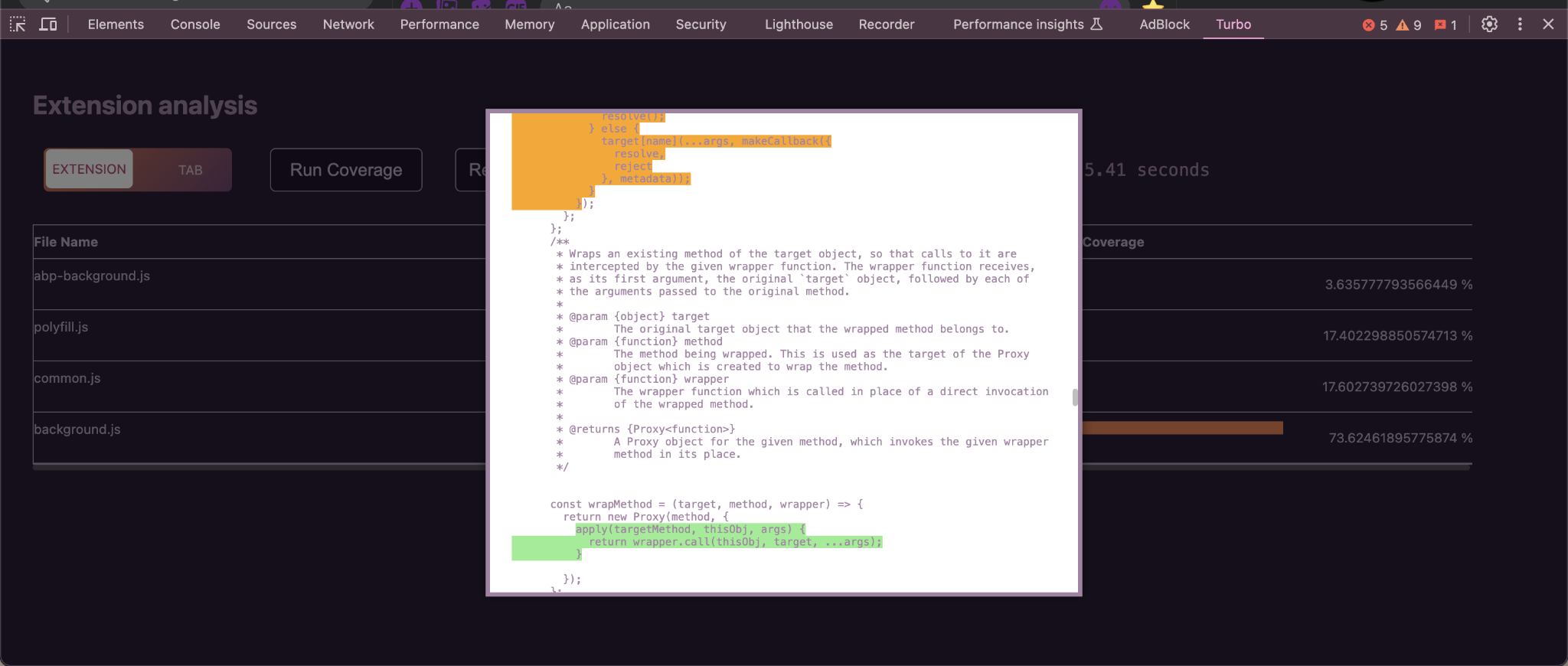
ერთ-ერთი სირთულე რასაც გადავაწყდით იყო, სწორი permissionების მოთხოვნა იუზერისგან რათა შეგვძლებოდა სრულფასოვანი მონაცემების შეგროვება ექსთენშენისთვის. Საჭირო გახდა რომ გადაგვეხედა და წაგვეკითხა არსებული ექსთენშენების permissionებიც და მათი ანალიზის საფუძველზე გვეპოვა რაზე შევძლებდით ჩვენც დებაგირებას. კვლევის შედეგად კი ავაწყვეთ manifest.json და ექსთენშენი, რომელიც ყველა მოთხოვნას აკმაყოფილებს (იხ. სურათი 10).

სურათი 10.

Ააღსანიშნავი დაკვირვებაა, რომ content-scriptებზე სხვანაირი მიდგომა იყო საჭირო, რადგანაც ისინი პირდაპირ არიან injected ქრომის ან ვებ გვერდის კოდში, და შესაბამისად მონაცემთა ფილტრაციის დროს გავითვალისწინეთ ეს შემთხვევაც, რომ მართლა გაეშვა კოდი თუ მხოლოდ injection მოხდა მაგრამ არანაირი ცვლილება, შედეგი არ ყოფილა.

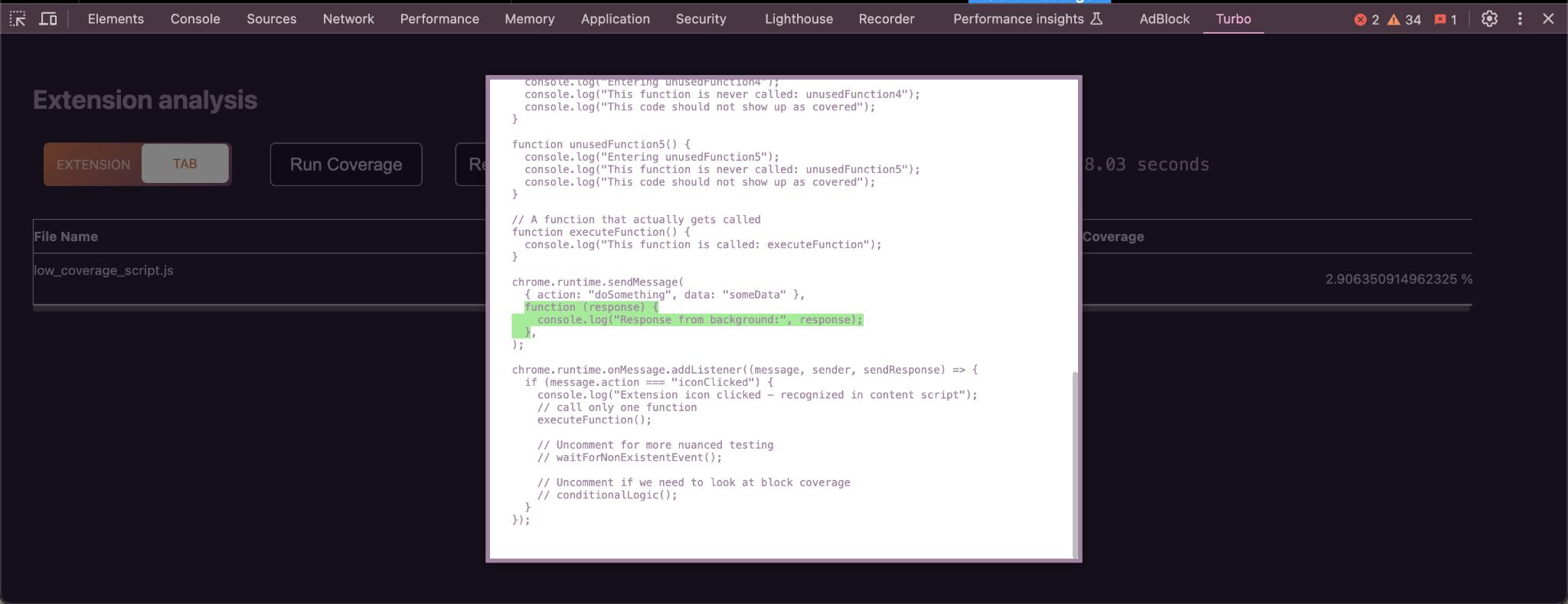
Ამის შემდგომ ეძახება მთლიანი ექსთენშენის კოდის წაკითხვის ფუნქცია, რათა დაითვალოს პროცენტული რაოდენობა დაფარვის და მთლიანი ფაილის. Ასევე მოინიშნება და ის ბაიტები, ფუნქციები და ხაზები, რომლებიც გაეშვა.

Ასევე ვითვლით თუ ერთი და იმავე კოდის ნაწილმა თუ რამდენჯერმე იმუშავა და დგება მისი სტატისტიკაც: რომელი ბაიტები იყო ყველაზე ხშირად და რომელი ყველაზე ნაკლებად. Ვიზუალურად მას იუზერი შემდეგი სახით ხედავს (იხ. სურათი 11). Იმის მიხედვით თუ რამდენად ხშირი იყო ერთი და იმავეს გამოძახება highlightის ფერი მუქდება და წითლდება, როგორც ყველაზე hot target. ~

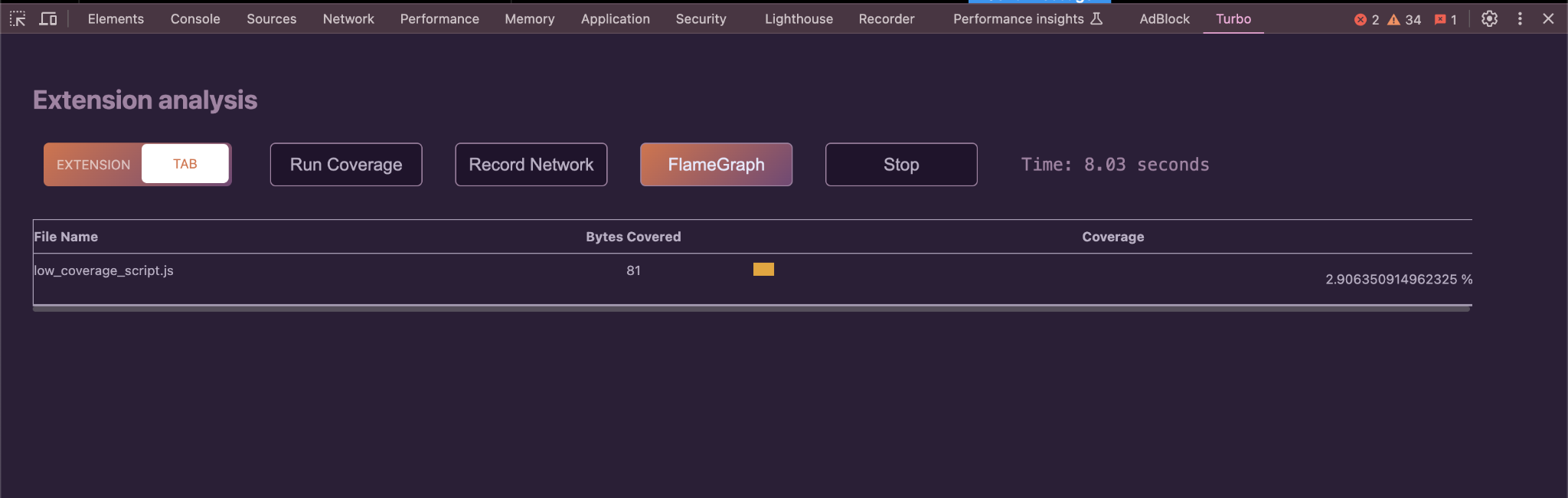
სურათი 11.

Ფრონტის ნაწილში გარე ბიბლიოთეკებიდან გამოყენებულია: progressbar.js, რომელიც იუზერისთვის უფრო მარტივად აღმად და ლამაზ ანიმაციას ქმნის საბოლოო დათვლილი დაფარვის პროცენტული რაოდენობისთვის. Მომხმარებელი ხედავს, ბაიტების რაოდენობას, პროცენტულ მახასიათებელს, ფაილის სახელს, ანიმაციას და ასევე მთელ კოდში მონიშნულ ნაწილებს.

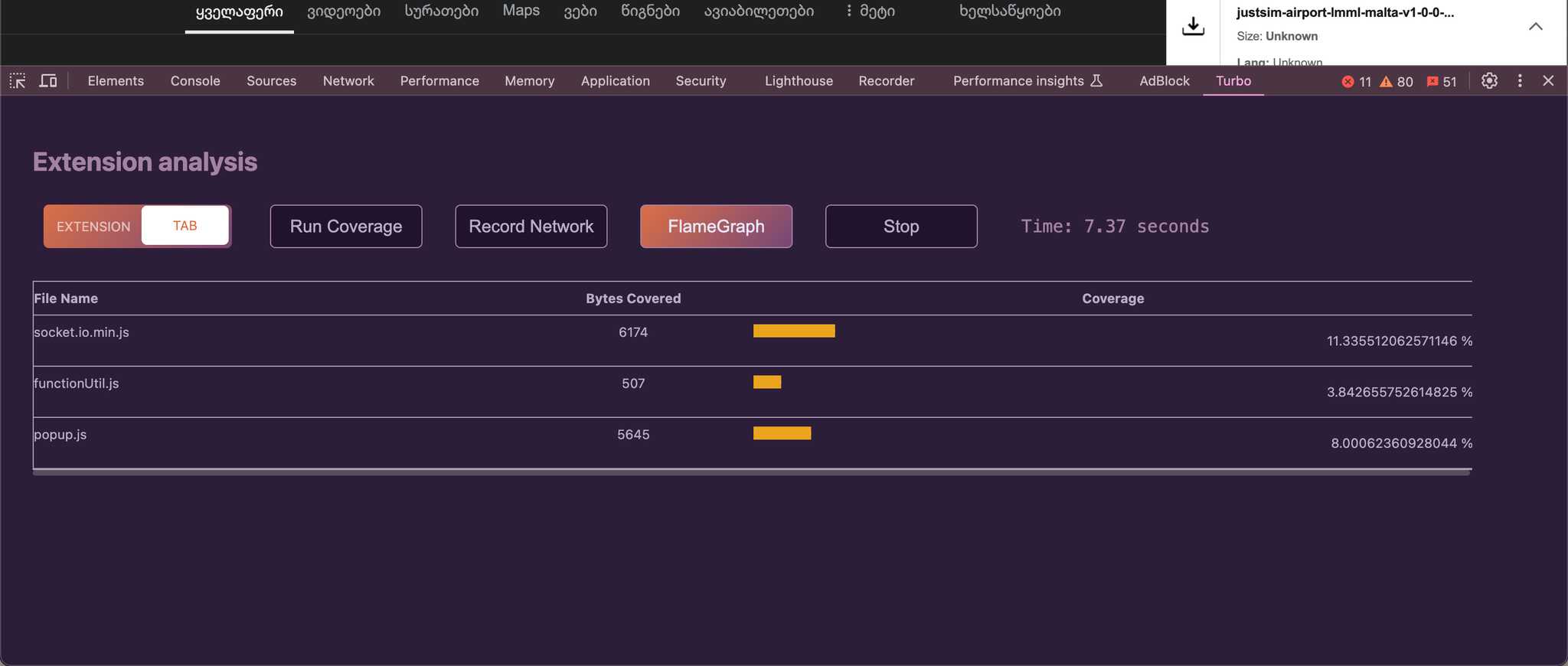
ტესტირების პროცესში ჩავატარეთ პატარა სიმულაცია ჩვენივე ექსთენშენზე რომ გვენახა რამდენად სწორად იმუშავებდა და თუ შეძლებდა არსებული პრობლემების აღმოჩენას coverage ფუნქციონალი. Შერჩეულ ექსთენშენზე ვნახეთ ძალიან მწირი კოდის დაფარვის მაჩვენებელი, ისევე როგორც ფუნქციათა გაშვების სტატისტიკა აქედან გამომდინარე იუზერს შეუძლია გააკეთოს ლოგიკური დასკვნები, რომ მის კოდში არსებობს unused, uncalled ან deprecated ფუნქციები.

სურათი 12.

სურათი 13.

აღსანიშნავია, რომ ასევე დაიტესტა რეალურად არსებული ექსთენშენებიც და მოხდა მათი კოდის ანალიზი როგორებიცაა: adblocker, torrent scanner. (იხ. სურათი 14)

სურათი 14

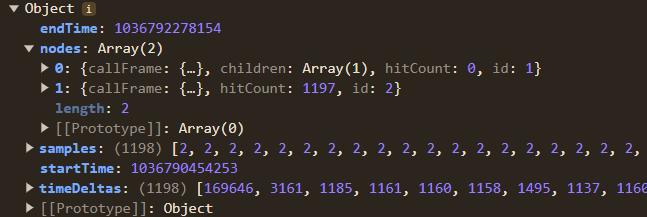


FlameGraph

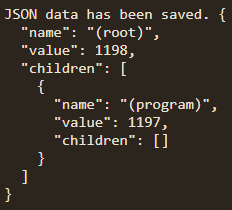
Flamegraph წარმოადგენს აპლიკაციის call stack-ის ვიზუალურ რეპრეზენტაციას. მისის საშუალებით შეგიძლია ამოიცნო კოდის ის ნაწილი რომელსაც ოპტიმიზაცია ესაჭიროება. თითოეული entry გრაფში ასახავს call stack ში ფუნქციას(ფუნქციის სახელის არარსებობის შემთხვევაში მომხმარებელი ხედავს url-ს). ჰორიზონტალური ღერძი წარმოადგენს რაოდენობას თუ რამდენჯერ მოხვდა ესა თუ ის ფუნქცია პროფაილინგის განმავლობაში call stack-ის თავზე. ხოლო ვერტიკალური ღერძი წარმოადგენს გამოძახების სტეკის სიღრმეს ამ დროში. როგორც დანარჩენი ფუნქციონალის, FlameGraph-ის მონიტორინგი შესაძლებალია Tab-ისათვის და თავად ექსთენშენის service worker-ისათვის.

Back-end-ში გამოყენებულია chrome.profiler-ის ბიბლიოთეკა, კეთდება მონაცემთა შეგროვება გაშვების მომენტიდან, რომელსაც შემდგომ ანალიზს, ფილტრაციასა და სხვადასხვა ტრანსფორმაციას ვუკეთებთ. შეგროვებული მონაცემებს აქვს შემდეგი სახე(იხ.სურათი 15). ფუნქციონალისთვის საინტერესო იყო node array, რომელშიც შეგვიძლია ვიპოვოთ ფუნქციის სახელი url(რომელზეც Tab-ის ნაწილში იფილტრება დატა და იღებს “chrome-extension://<extension-id>/<file-name>” სტრუქტურას). საჭირო იყო გამოგვეყენებინა ჩვენთვის კარგად ნაცნობი რეკურსია, რომ ხის საშუალებით წარმოგვედგინა შედეგები. ტრანსფორმირებული შედეგს აქვს შემდეგი სახე(იხ.სურათი 16), სადაც value გამოითვლება hitcount ების საშუალებით. Sampling წარმოადგენს snapshots პერიოდულად, რომლის შედეგადაც ვიღებთ node.id array. hitcount წარმოადგენს samples-ში რამდენჯერ შეგვხვდა ესა თუ ის ფუნქცია.

სურათი 15. profiler შედეგი



სურათი 16. ტრანსფორმირებული დატა



Flamegraphs გააჩნია როგორც სტატიკური ასევე დინამიური ვიზუალიზაცია. სტატიკურ ვიზუალიზაციაში იგულისხმება რომ ჩანს ყველაზე თვალსაჩინო ფრეიმები, თუმცა მცირე ზომის entries არ არის გამოკვეთილი. დინამიური ვიზუალიზაცია მომხმარებელს საშუალებას აძლევს ინტერაქტიულად გაეცნოს მიღებულ ინფორმაციას:

* როდესაც მაუსი დგას კონკრეტულ ფრეიმზე ჩანს ფუნქციის სახელი და მნიშვნელობა, ასევე ახლდება დეტალების ზოლი.
* ფრეიმზე დაჭერისას ახლოვდება ფრეიმი ჰორიზონტალურად, დიდი ზომის ფუნქციის სახელები უფრო სრულად ჩანს.
* “Reset Zoom” - გრაფს საწყის ვიზუალზე აბრუნებს.
* “Search”ასევე მომხმარებელს შეუძლია მოძებნოს კონკრეტული ფუნქცია, დეტალებში ჩანს რა ნაწილია მონიშნული როგორც სემპლების რაოდენობით, ისე პროცენტულად.

Tab-ის არჩევის შემთხვევაში flamegraph აჩვენებს თაბის ნაწილს და აჰაილთებს pop up-ში მონიშნული ექსთენშენის ნაწილს(იხ.სურათი 25).

ფრონტის ნაწილში გარე ბიბლიოთეკიდან გამოყენებულია d3-flame-graph(Spiermar, n.d.).

ტესტირების პროცესში ჩავატარეთ სიმულაცია ჩვენივე ექსთენშენზე რომ გვენახა რამდენად სწორად იმუშავებდა და თუ შეძლებდა არსებული პრობლემების აღმოჩენას flamegraph ფუნქციონალი. შერჩეულ ექსთენშენში ხელოვნურად იყო ჩასმული განსაზღვრეული დროით loop. ლოგიკურად ამ ფუნქციის მნიშვნელობა უნდა ყოფილიყო დიდი სხვა ფუნქციებთან შედარებით(იხ.სურათი 25) . ეს ტესტირება აჩვენებს რომ მომხმარებელს შეუძლია flamegraph-ის საშუალებით დაადეტექტოს bottlenecks თავის კოდში, ანუ ნახოს რა მეთოდი ეშვება დიდი დროით და ასევე დააკვირდეს ფუნქციათა იერარქიას რომ ადვილად იპოვოს ეს მეთოდი კოდში.

Network Monitoring

Network მონიტორინგის ფუნქციონალი საშუალებას აძლევს მომხმარებელს, დააკვირდეს დროის მოცემულ პერიოდში ექსთენშენის მიერ შესრულებულ ქსელურ მოთხოვნებს. Როგორც ყველა სხვა feature-ს შემთხვევაში, აქაც შესაძლებელია არჩევა, მონიტორინგი მოხდება service worker -ისა თუ tab (content script) კონტექსტში.

შეგროვებული მონაცემების ვიზუალიზაცია ხდება ცხრილის სახით (იხ. Სურათი 26), რომელშიც წარმოდგენილია მისამართი (url) რომელზეც განხორციელდა მოთხოვნა, მეთოდი (GET, POST..), ტიპი (fetch, script…) პასუხის სტატუსი, მოთხოვნის ავტორი (initiator) - შესაბამისი ფუნქციის სახელი ან სკრიპტის მისამართი, ასევე დაბრუნებული მონაცემების ზომა კილობაიტებში და latency მაჩვენებელი - დროის შუალედი მოთხოვნის გაგზავნასა და response მიღებას შორის. უკანასკნელ სვეტში კი წარმოჩენილია debugger-ის მიერ დაბრუნებული timing ინფორმაციის ვიზუალიზაცია ჩარტის სახით, რომელშიც თვალსაჩინოა ქსელური მოთხოვნის სხვადასხვა ეტაპი: Proxy, SSL negotiation, DNS Lookup, Send, Receive Headers და ა.შ. ყველა ეტაპი წარმოჩენილია სხვადასხვა ფერის bar-ით და hover-ის დროს მომხმარებელს შეუძლია წაიკითხოს ეტაპი და დროის შუალედი, რომელსაც შეესაბამება კონრეტული bar (იხ. სურათი 27).

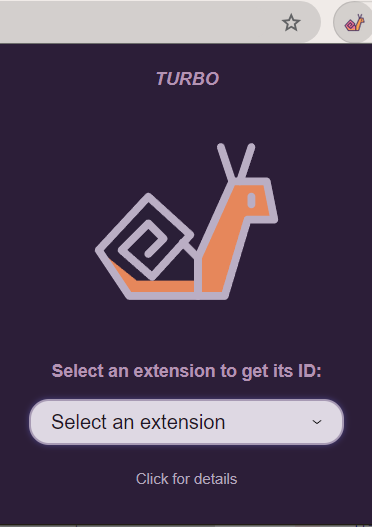
აღნიშნული ფუნქციონალი ეყრდნობა DevTools პროტოკოლის Network domain-ს. კონკრეტულად, ჩვენი პანელი ლისენერებით უსმენს შემდეგ event-ებს:

* *Network.requestWillBeSent* - თრიგერდება ზუსტად http request-ის გაშვებამდე მომენტში. თან ახლავს request-ის უნიკალური იდენტიფიკატორი და ისეთი ინფორმაცია, როგორიცაა url, timestamp, initiator რაც საბოლოო შედეგში გამოიყენება.
* *Network.responseReceived* - თრიგერდება მაშინ, როდესაც იწყება response-ს მიღება. Მასაც თან ახლავს იდენტიფიკატორი, რომლითაც ვაკავშირებთ წინა event-ში შენახულ ინფორმაციასთან, ამ ეტაპზე უკვე ვუმატებთ field-ებს status, type, ასევე ვინახავთ timing ობიექტს ჩარტის ასაგებად.
* *Network.dataReceived* - თრიგერდება მაშინ, როდესაც ქსელში მიღებულია მონაცემების გარკვეული chunk. თან ახლავს იდენტიფიკატორი, timestamp და მიღებული მონაცემის ზომა, სწორედ ამ ზომას ვუმატებთ აქამდე შენახულ ინფორმაციას რათა დავთვალოთ შესაბამისი request-ისთვის მთლიანი size. ასევე ვიმახსოვრებთ დროს რათა გამოვთვალოთ receiving დროის მაჩვენებელი (*dataReceived.timestamp - responseReceived.timestamp),* ამ უკანასკნელის ვიზუალიზაციას ვახდენთ მხოლოდ timing ობიექტის არარსებობის შემთხვევაში, ვინაიდან ის უფრო ნაკლებად ზუსტი და ინფორმაციულია.

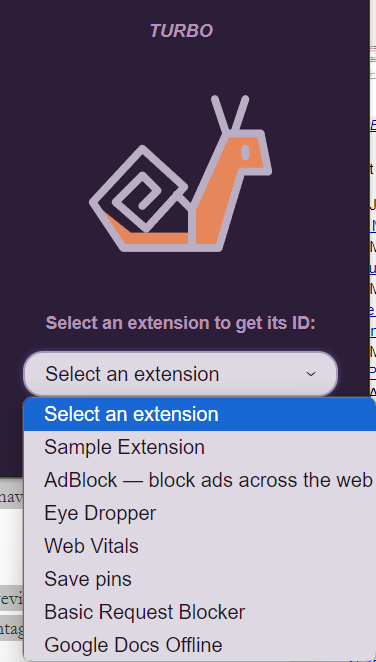
ამ ფუნქციონალის იმპლემენტაციის დროს ძირითადი გამოწვევა გახლდათ Network დომეინში გარკვევა, სწორი event-ების შერჩევა და დროის მაჩვენებლების ინტერპრეტაცია, ვინაიდან ეს API ფაქტობრივად არ არის დოკუმენტირებული და მისი გაგება საჭიროებდა გუგლის დეველოპერების დისკუსიებისა თუ source კოდის ფრაგმენტების გარჩევასა, აგრეთვე საკმაო trial and error-ს. მაგალითად, არაა ცალსახა რომ *Network.responseReceived* ივენთი თრიგერდება არა მაშინ, როდესაც response მთლიანად გახლავთ მიღებული, არამედ მაშინ, როდესაც იწყება header-ების პირველი ბაიტების მიღება, ხოლო body-ს მიღების მონიტორინგისთვის საჭიროა დამატებით *Network.dataReceived* ივენთზე მოსმენა. ასევე გარკვეული ველები დაბრუნებულ ობიექტებში არის ექსპერიმენტული - მაგალითად, timing ობიექტის ველების უმეტესობა. ამიტომ ამ ობიექტის ვიზუალიზაცია ყოველთვის ინტუიციურ შედეგს არ იძლევა. Მიუხედავად ამისა, საკვანძო მაჩვენებლები - latency და size პერფორმანსის გაზომვისთვის საკმაოდ ინფორმაციულია.

შედეგები

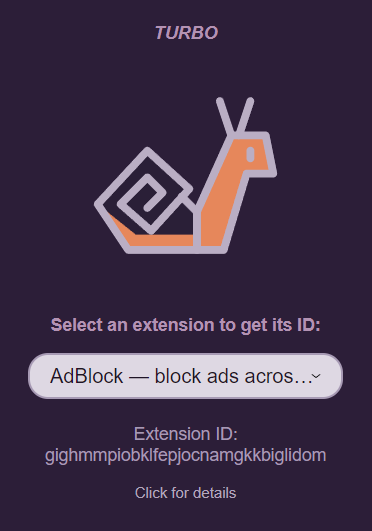
სურათი 17.ექსთენშენის Pop up. მომხმარებელმა უნდა აირჩიოს ის ექსთენშენი რომლის დებაგირებაც სურს.



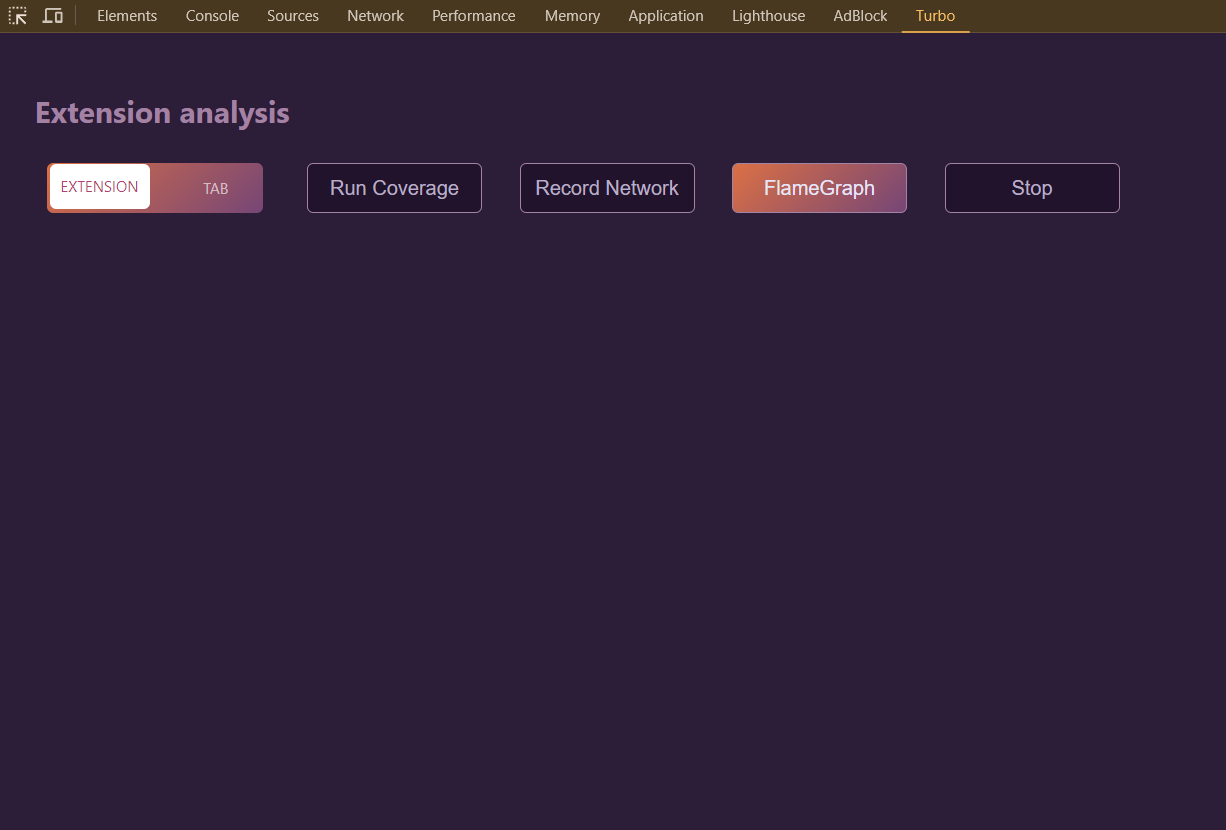
სურათი 18. მომხმარებელი ირჩევს საკუთარი ექსთენშენების ჩამონათვალიდან ერთერთს.



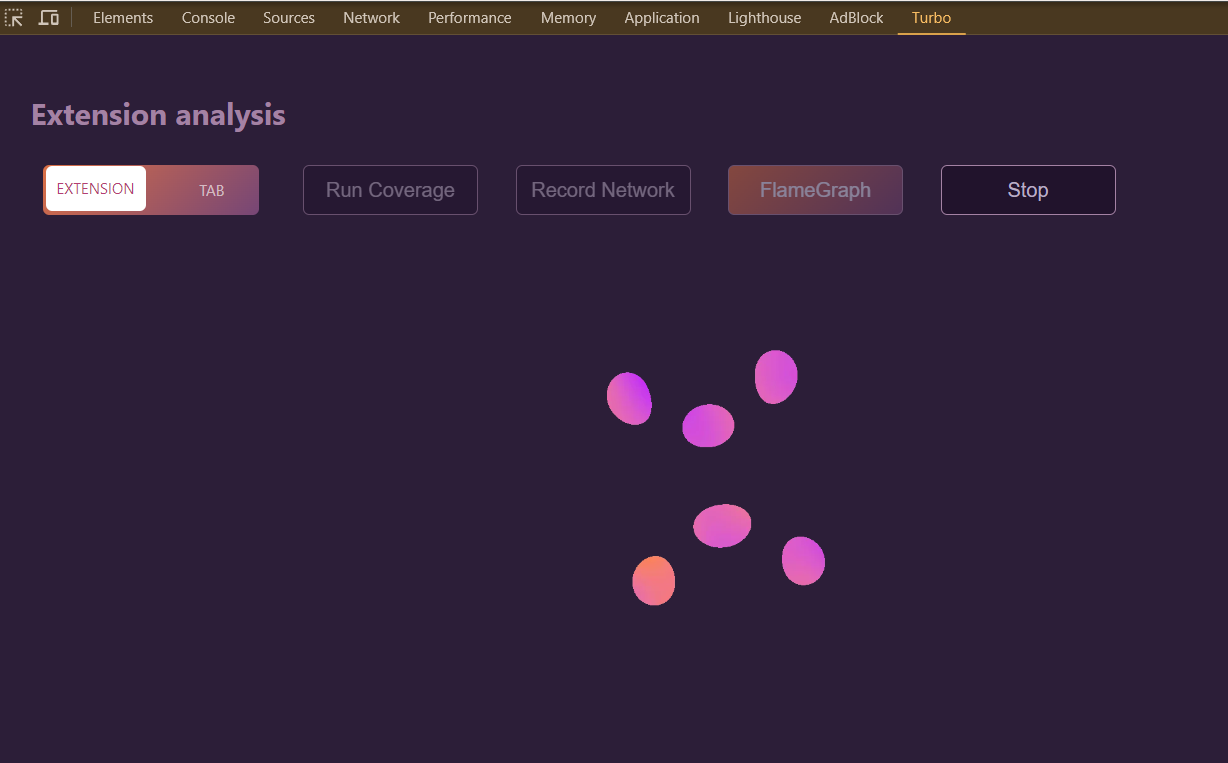
სურათი 19. მომხმარებელს შეუძლია ექსთენშენის ID ნახვა pop up-ზე. დეტალური ინფორმაცია გამოჩნდება “Click for details” ლინკზე დაჭერისას



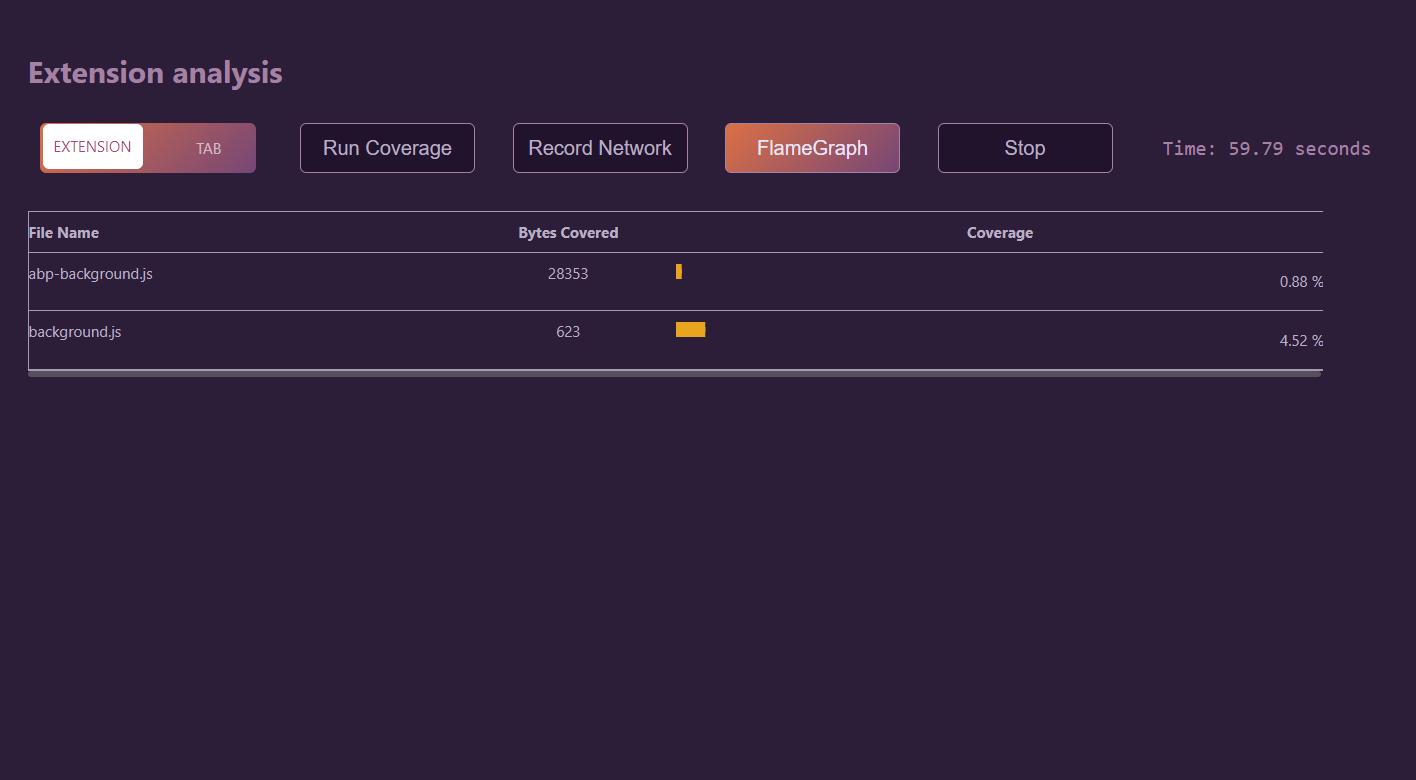
სურათი 20. პანელის საწყისი ფორმა



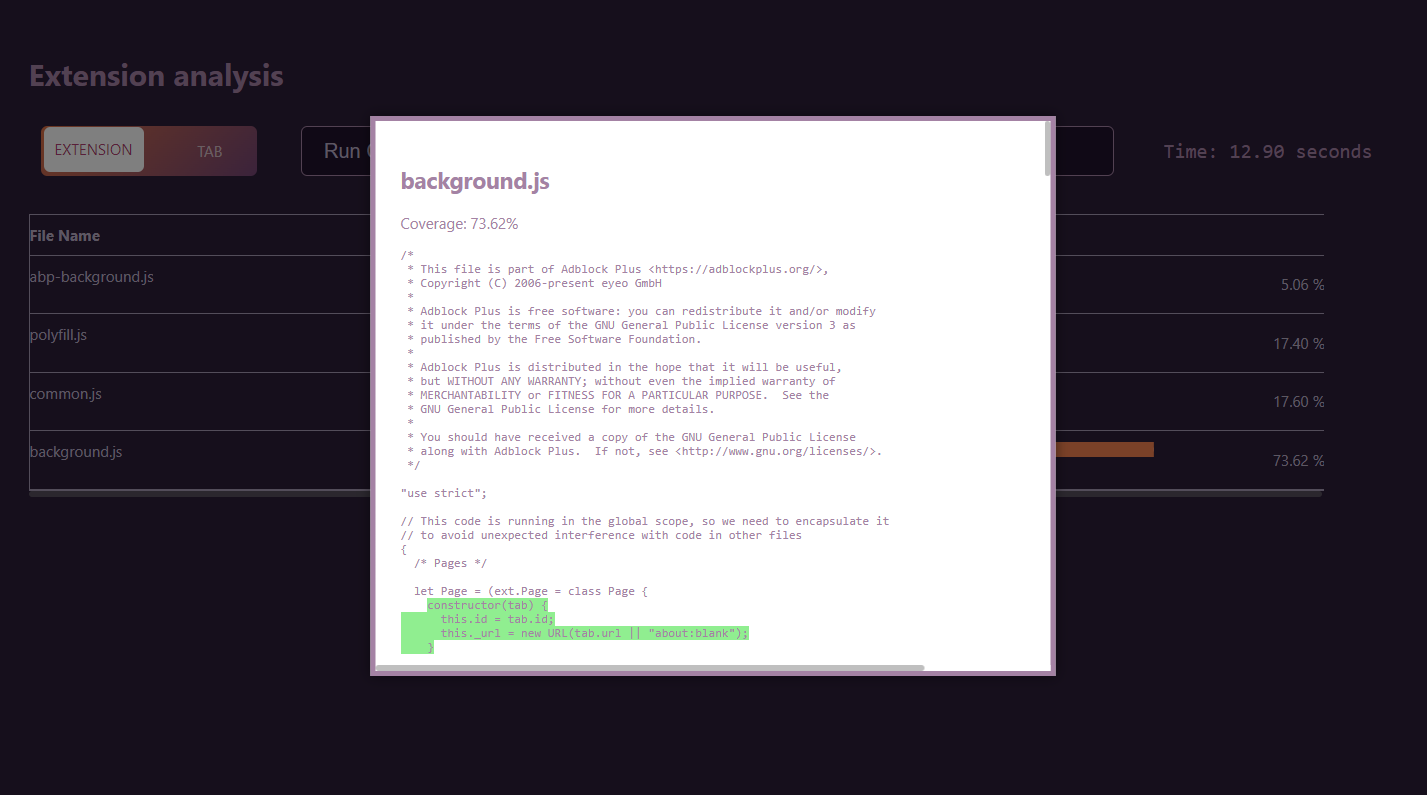
სურათი 21. პანელი ერთ-ერთი ფუნქციონალის გაშვების პროცესში



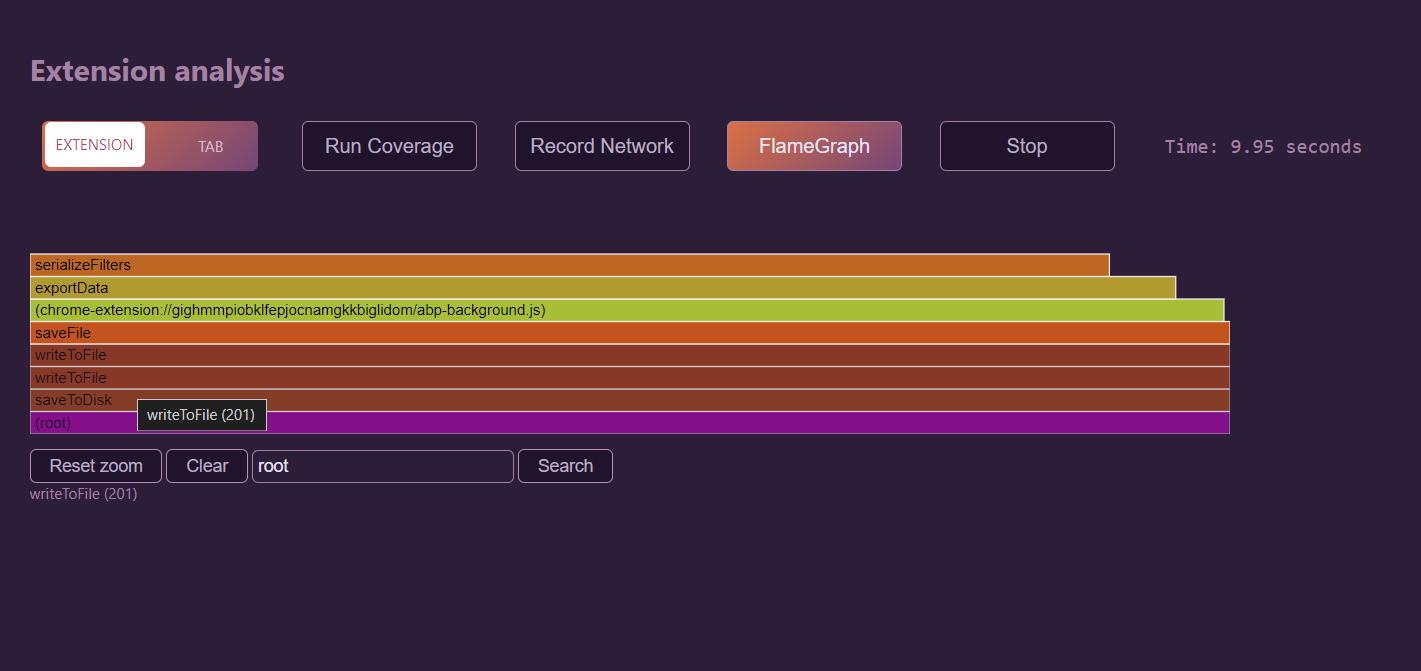
სურათი 22. Coverage შედეგები



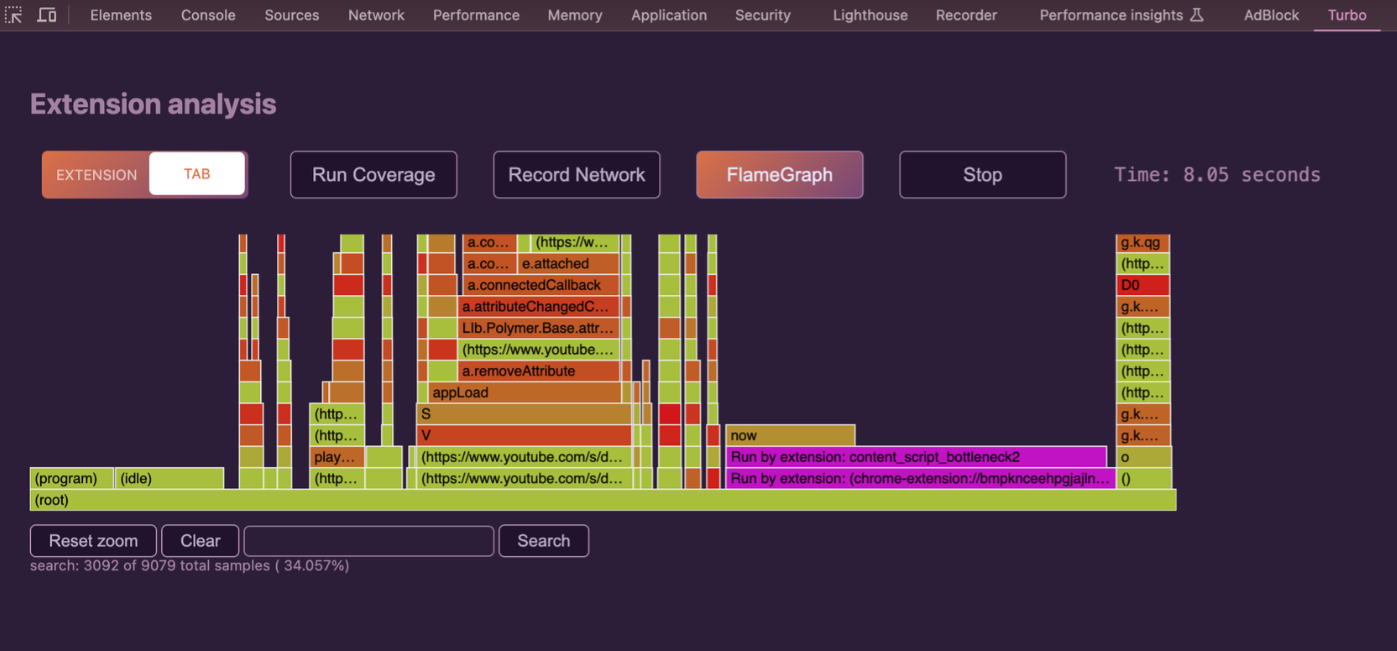
სურათი 23. სკრიპტის გახსნა/დაჰაილაითებული ნაწილები დაფარვის მიხედვით



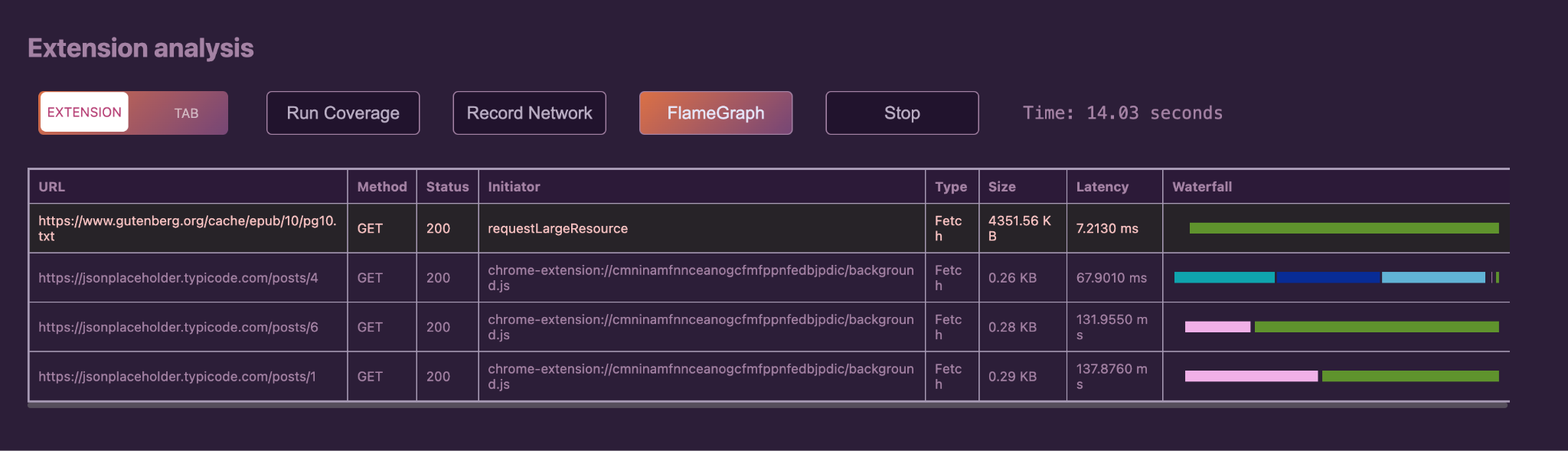
სურათი 24. ფლეიმგრაფის დაზუმების შედეგი



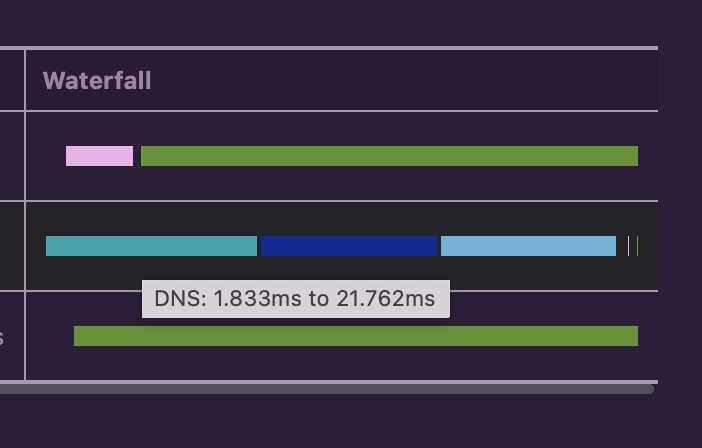
სურათი 25. ტაბის შემთხვევაში ექსთენშენის მიერ გაშვებული ფუნქციები ვიზუალურად



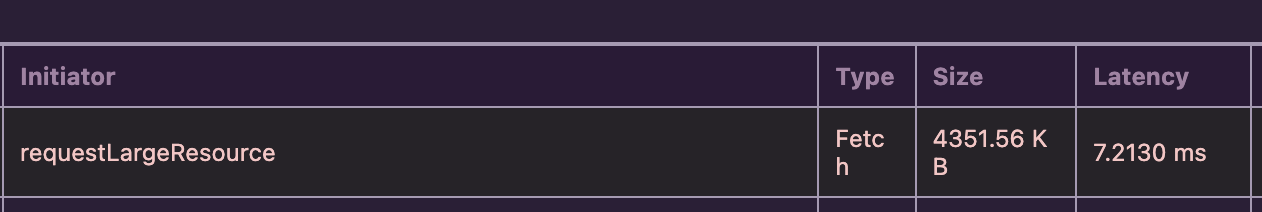
სურათი 26. Network ცხრილი



სურათი 27. Network Waterfall Chart Hover / Closeup

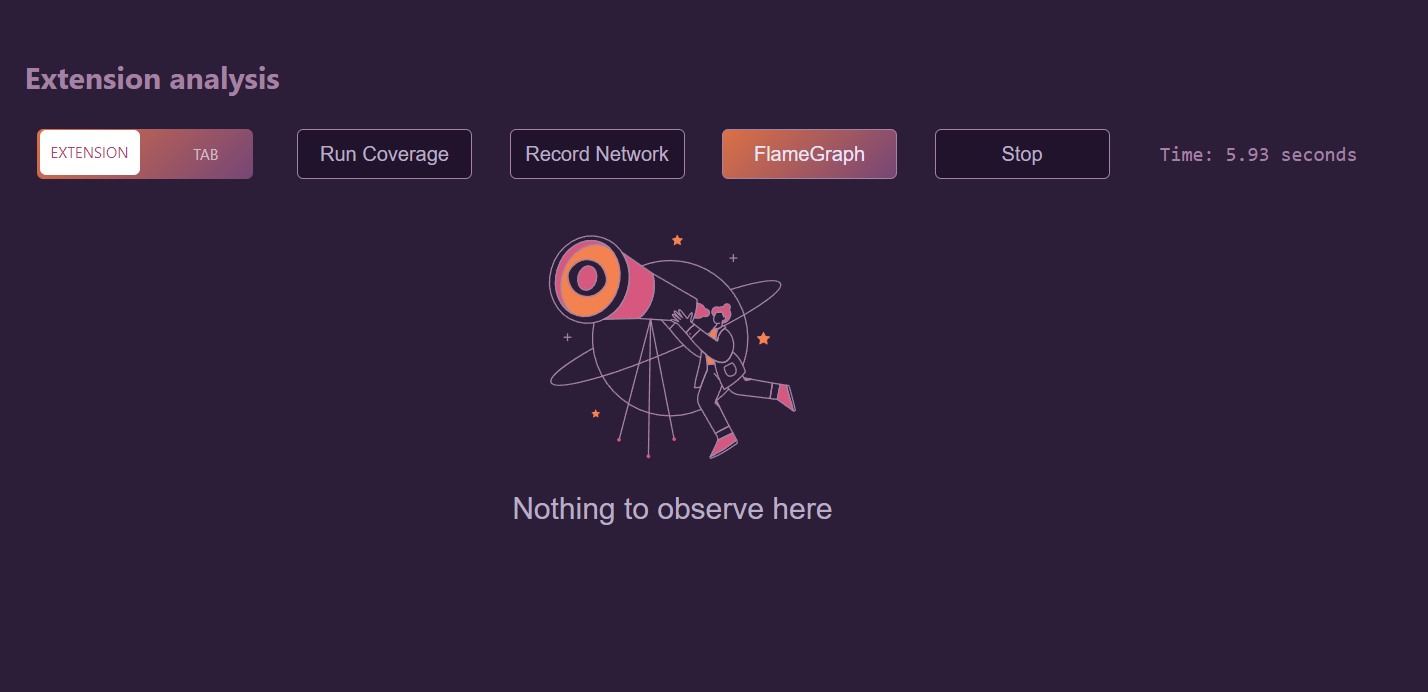


სურათი 28. Network requestLargeResource closeup

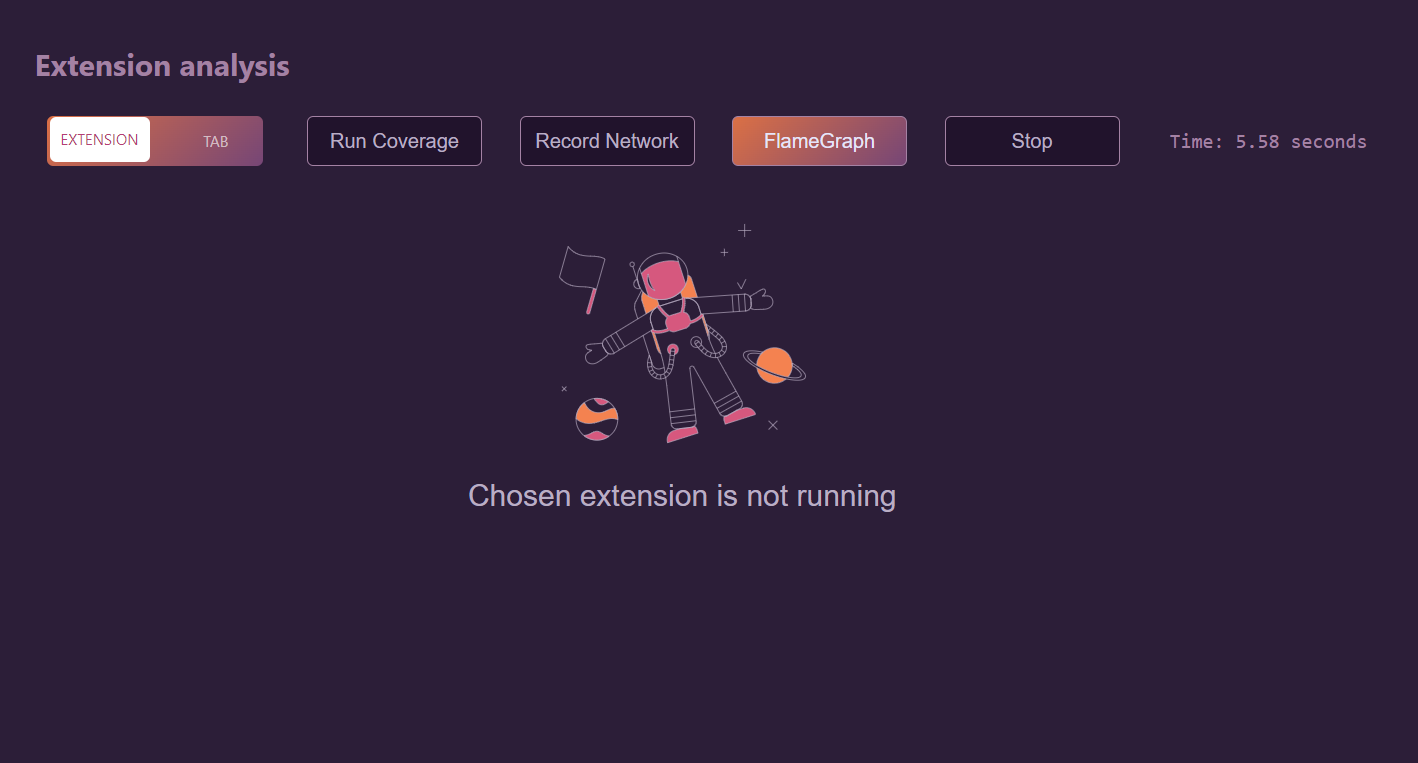


Სურათზე ვხედავთ network feature შედეგს Სატესტო ექსთენშენისთვის, რომელშიც ეშვება *requestLargeResource()* მეთოდი როგორც პირობითი bottleneck და აკეთებს request-ს 4.3 Mb ზომის რესურსზე. Როგორც ვხედავთ გამოტანილი მონაცემებით აღნიშნული bottleneck-ის აღმოჩენა მარტივადაა შესაძლებელი.

სურათი 29. ინფორმაცია ვერ მოიძებნა



სურათი30. იუზერის ექსთენშენი არ ეშვება



# დასკვნა

პროექტის შედეგად გადაიჭრა ყველა ძირითადი ტექნიკური ამოცანა. ექსთენშენის ვიზუალი არის ინტუიტიური და მომხმარებელს საშუალებას აძლევს გამოიყენოს როგორც DevTool-ისათვის მახასიეთებელი features, ასევე ექსთენშენის მოქნილობა და ვიზუალი. გარდა ვიზუალური უპირატესობისა, დავფარეთ ის სამი პერფორმანსის ანალიზის ხელსაწყო რაც გეგმის ნაწილი იყო და გავტესტეთ როგორც საკუთარ ასევე ყველასათვის ცნობილ ექსთენშენებზე (მაგალითად AdBlocker).

პროექტის წერის განმავლობაში მივიღეთ დიდი გამოცდილება. შევისწავლეთ ჯავასკრიპტის ენა, რომელთანაც უნივერსიტეტის სხვა კურსებში შეხება არ გვქონდა. chrome-ის გამოყენებით სასურველი ინფორმაციის შენახვა და გამოყენება. DevTools პანელის ჩაშენება html/css ის საშუალებით მომხმარებლისთვის მაქსიმალურად კომფორტული ვიზუალური მხარის შერჩევა და იმპლემენტაცია.

არანაკლებ მნიშვნელოვანია ინტერპერსონალური უნარებიც: პროექტის არასტანდარტულობამ განაპირობა ბევრი ისეთი საინტერესო ინფორმაციის მოძიება რომელიც აქამდე ჩვენთვის უცნობი იყო. რაც აუცილებელს ხდიდა მოცემულ ვადებში გაგვენაწილებინა საქმე გუნდის ყველა წევრზე. დიდმა პროექტმა საშუალება მოგვცა გაგვეუმჯობესებინა დროის მენეჯმენტი. ვყოფილიყავით დეტალებზე ორიენტირებულები და მაქსიმალურად არგუმენტირებულად გადაგვეჭრა წამოჭრილი პრობლემები.

# გამოყენებული ლიტერატურა

[1] Dean, B. (2024, March 14). *Google Chrome Statistics*. Backlinko. <https://backlinko.com/chrome-users>

[2] Counting Chrome Extensions – Chrome Web Store statistics. (2024, April 24). *DebugBear*. <https://debugbear.com/blog/counting-chrome-extensions>

[3] *Contributing to Chrome DevTools Protocol*. (n.d.). Google Docs. <https://docs.google.com/document/d/1c-COD2kaK__5iMM5SEx-PzNA7HFmgttcYfOHHX0HaOM/edit#heading=h.e6mz7k1mw34a>

[4] *Chromium Project.* (n.d.). *Search for kExtensionsOnChromeURLs*. Chromium Code Search. <https://source.chromium.org/search?q=kExtensionsOnChromeURLs&sq=&ss=chromium%2Fchromium%2Fsrc>

[5] Spiermar. (n.d.). *GitHub - spiermar/d3-flame-graph: A D3.js plugin that produces flame graphs from hierarchical data.* GitHub. <https://github.com/spiermar/d3-flame-graph>

# დანართი

პროექტის კოდი შეგიძლიათ იხილოთ github-ზე, ლინკზე:

<https://github.com/MJava2002/Chrome-extension-performance-tools>