Optimisasi Kinerja Aplikasi Fitness Berbasis Next.js Melalui Penerapan Metode Caching Pada PT. Anugerah Wijaya Raga

Abdul Rahman[1], Agus Prihanto[2]

[1,2] Teknik Informatika, Fakultas Teknik, Universitas Negeri Surabaya
[1]abdul.20052@mhs.unesa.ac.id
[2]agusprihanto@unesa.ac.id

***Abstrak*— Perkembangan teknologi web modern mendorong penggunaan framework seperti Next.js dalam pengembangan aplikasi. PT. Anugerah Wijaya Raga (AWR) mengembangkan aplikasi GGL (Gak Gendut Lagi) untuk manajemen nutrisi dan kebugaran, namun menghadapi tantangan kinerja terkait pemanggilan API yang tidak efisien dan berpengaruh ke respons aplikasi. Penelitian ini bertujuan mengoptimalkan kecepatan waktu respons aplikasi GGL dengan metode caching menggunakan Tanstack Query. Hasil menunjukkan bahwa caching efektif mengoptimalkan waktu respons aplikasi. Sebelum optimasi, waktu login hingga ke dashboard adalah 6,70 detik, setelah optimasi menjadi 4,43 detik (33,8%). Halaman mealplan meningkat dari 6 detik menjadi 3,94 detik (34,3%), dan halaman measurement dari 2,93 detik menjadi 1,47 detik (49,8%). Efektivitas pemanggilan API juga meningkat tanpa adanya duplikasi pemanggilan. Penerapan caching dan prefetching meningkatkan responsivitas aplikasi dan pengalaman pengguna secara keseluruhan.**

Kata Kunci: Next.js, Caching, Tanstack Query, Optimisasi, Aplikasi Fitness

I. PENDAHULUAN

Dalam era digital saat ini, aplikasi web berbasis framework, khususnya yang menggunakan JavaScript, telah menjadi sangat populer dalam pengembangan website modern. Salah satu framework yang sering digunakan adalah Next.js, yang menawarkan kemudahan dalam pengembangan dengan pendekatan berbasis komponen dari React, memungkinkan pengembang untuk membangun aplikasi yang cepat dan responsif.

PT. Anugerah Wijaya Raga (AWR), sebuah perusahaan yang bergerak di bidang kebugaran dan kesehatan, telah mengembangkan aplikasi GGL (Gak Gendut Lagi) yang berfokus pada manajemen nutrisi, kebugaran, dan penurunan berat badan. Aplikasi ini menyediakan berbagai fitur yang dirancang untuk membantu pengguna mencapai tujuan kesehatan mereka, seperti pencatatan makanan, informasi nutrisi, dan pemantauan perkembangan penurunan berat badan. Namun, aplikasi GGL mempunyai tantangan terkait kinerja, karena seringkali fokus utama pengembang adalah pada fitur dan fungsionalitas, tanpa memperhatikan secara mendalam aspek-aspek kinerja aplikasi. Hal ini disebabkan oleh berbagai faktor, termasuk kurangnya pemahaman tentang pentingnya optimasi kinerja, tekanan waktu, atau prioritas pengembangan yang lain. Sebagai hasilnya, performa aplikasi seringkali terabaikan, padahal performa yang baik merupakan faktor kunci dalam menentukan keberhasilan sebuah aplikasi.

Aplikasi GGL mempunyai pemanggilan API yang tidak efisien. Misalnya, ketika pengguna mengunjungi halaman dashboard, aplikasi ini memanggil beberapa API. Jika pengguna berpindah ke halaman lain dan kembali lagi ke dashboard, aplikasi ini masih memanggil API yang sama meskipun responnya tidak berubah. Hal ini menimbulkan network request yang tidak perlu dan memperlambat respons aplikasi.

Penelitian terdahulu telah menunjukkan bahwa metode caching dapat secara signifikan meningkatkan kinerja aplikasi web. Misalnya, penelitian oleh Ahmad Nizar (2022) menunjukkan bahwa refaktorisasi state management dari react-redux ke react-query dapat meningkatkan performa aplikasi dengan mengurangi pemanggilan API yang tidak perlu. Selain itu, penelitian oleh Jia Wang (1999) dan Ivano Malavolta et al. (2020) juga mendukung bahwa metode caching dapat meningkatkan kecepatan load time halaman web.

Dengan memperhatikan tantangan aplikasi dan penelitian terdahulu, penelitian ini bertujuan untuk mengatasi masalah kinerja aplikasi fitness berbasis Next.js milik PT. Anugerah Wijaya Raga dengan mengimplementasikan metode caching. Dengan demikian, diharapkan aplikasi dapat menjadi responsif serta meningkatkan pengalaman pengguna secara keseluruhan.

II. METODOLOGI PENELITIAN

Penelitian ini memiliki beberapa tahapan, yaitu melakukan identifikasi pemanggilan API pada aplikasi, implementasi metode caching pada aplikasi, lalu pengujian performa aplikasi.

Gambar. 2.1 Tahapan Penelitian

A. Identifikasi pemanggilan API pada aplikasi.

Tahap pertama adalah mengidentifikasi titik-titik pada codebase aplikasi yang memerlukan implementasi caching. Identifikasi ini dilakukan dengan memeriksa bagian-bagian aplikasi yang sering memanggil API atau memuat data dari server.

B. Implementasi Metode Caching

Setelah bagian yang memerlukan caching teridentifikasi, langkah selanjutnya adalah menerapkan metode caching dengan menyimpan data yang sering diminta / di request oleh client, sehingga tidak perlu dipanggil ulang dari server setiap kali suatu request dilakukan.

C. Pengujian Performa Aplikasi

untuk membandingkan performa dengan melihat Response Time Aplikasi dan efektivitas pemanggilan API sebelum dan setelah implementasi metode caching. Berikut adalah langkah-langkah pengujian yang dilakukan:

1. Menjalankan Aplikasi Tanpa Implementasi Caching
2. Aplikasi dijalankan dalam kondisi normal tanpa implementasi caching.
3. Memantau waktu respons aplikasi menggunakan alat monitoring seperti Chrome DevTools.
4. Mencatat efektivitas pemanggilan API yang terjadi dalam skenario tertentu.
5. Menjalankan Aplikasi dengan Implementasi Caching
6. Aplikasi dijalankan setelah implementasi caching.
7. Kembali memantau performa aplikasi, efektivitas pemanggilan API, serta responsivitas aplikasi menggunakan alat monitoring yang sama

Setelah kedua pengujian selesai, hasil dari kedua pengujian tersebut dianalisis dan dibandingkan, sehingga hal ini dapat membantu dalam mengevaluasi efektivitas dari implementasi caching yang sudah diimplementasikan. Parameter hasil yang dilihat adalah Response Time, yaitu waktu yang dibutuhkan untuk memuat halaman atau komponen aplikasi, diukur menggunakan alat monitoring seperti Chrome DevTools. Hal ini akan menunjukkan apakah implementasi caching dapat meningkatkan responsivitas aplikasi, serta Efektivitas Pemanggilan API, yaitu jumlah request data yang terjadi dalam skenario tertentu, sebelum dan sesudah implementasi caching. Hal ini akan menunjukkan apakah implementasi caching dapat mengurangi jumlah pemanggilan API yang tidak perlu.

III. HASIL DAN PEMBAHASAN

Hasil dari penelitian ini adalah data pengukuran yang didapatkan dari Chrome Devtools sebelum dan sesudah mengimplementasikan metode caching.

A. Hasil Identifikasi Halaman Login

Pada tahap ini, peneliti melakukan identifikasi titik-titik pada codebase aplikasi yang memerlukan implementasi caching.

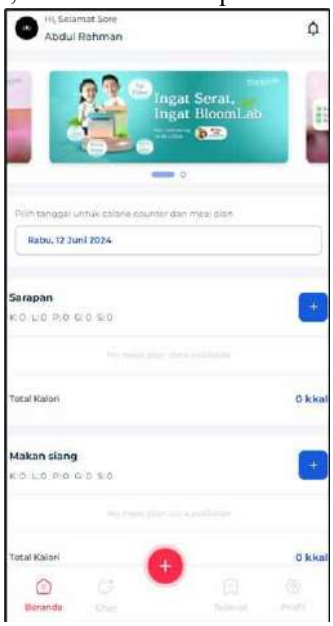
```
dispatch.user.getProfile().then(() => {
  router.push(`/${router.query.redirect}`)
})
```

Gambar. 3.1 Tahapan Penelitian

Pada gambar diatas, bisa dilihat bahwa adanya pemanggilan data profil menggunakan redux dan tanpa caching. Sehingga jika di aplikasi ingin mendapatkan data profil, maka API tersebut akan selalu dipanggil melalui useEffect.

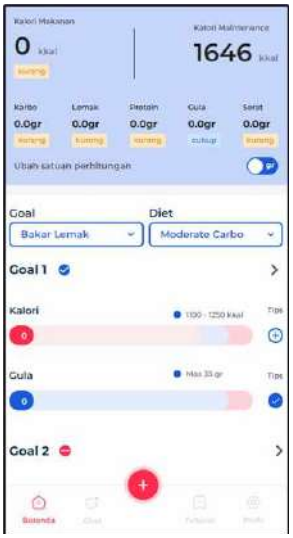
B. Hasil Identifikasi Halaman Dashboard

Pada halaman dashboard, terdapat beberapa komponen pada halaman tersebut, berikut adalah tampilan halaman dashboard:



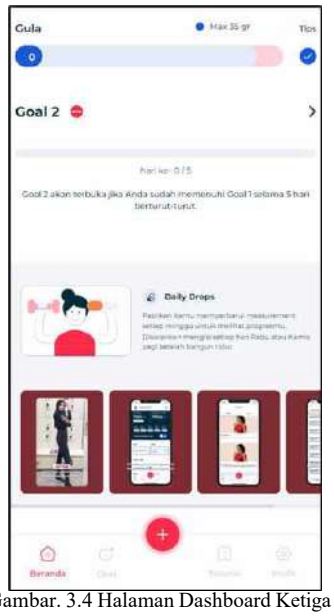
Gambar. 3.2 Halaman Dashboard Pertama

Halaman tersebut ada beberapa bagian, yaitu bagian paling atas yaitu nama user dan foto profil user, bagian pemilihan tanggal, dan bagian dashboard makanan yang terdiri dari “Sarapan”, “Makan Siang” dll. Setiap bagian tersebut memerlukan pemanggilan API yang dapat dilakukan caching, karena halaman tersebut adalah yang paling utama dalam aplikasi ini dan sering dikunjungi berkali – kali oleh user. Sehingga jika dilakukan caching, tidak diperlukan untuk memanggil data tersebut kembali.



Gambar. 3.3 Halaman Dashboard Kedua

Saat user melakukan scroll pada halaman dashboard, maka akan terlihat bagian nutrisi dan bagian goal, yang terdiri dari total kalori yang dikonsumsi oleh user, serta detail makronutrisi yang telah dikonsumsi, serta ada bagian goal yang terdiri dari suatu slider sesuai dengan tujuan dan tipe diet yang diinginkan oleh user. Bagian tersebut juga memerlukan pemanggilan API dan mempunyai potensi untuk diimplementasi caching.



Gambar. 3.4 Halaman Dashboard Ketiga

Pada bagian terakhir pada dashboard terdapat bagian Daily Drops yaitu yang memberikan tips kepada user, data tersebut disimpan pada aplikasi dan tidak diperlukan pemanggilan API. Lalu juga ada bagian stories yaitu terdiri dari video – video pendek yang memberikan tips kepada user, bagian tersebut memerlukan pemanggilan API, sehingga mempunyai potensi untuk dilakukan caching agar user tidak perlu melakukan pemanggilan data berulang saat bernavigasi ke bagian aplikasi yang lain lalu kembali ke dashboard.

C. Hasil Identifikasi Halaman Mealplan

Pada halaman Mealplan terdapat banner iklan yang didapatkan dari pemanggilan API, sehingga bisa dilakukan caching agar tidak terjadi layout shift dan pemanggilan API yang lebih efektif.

Gambar. 3.5 Halaman Mealplan

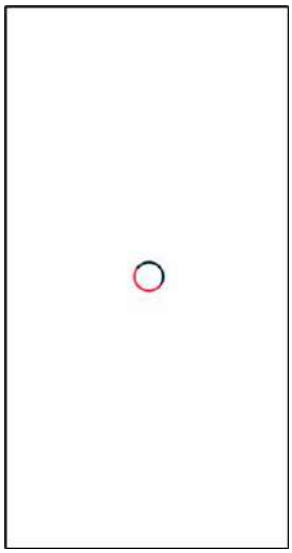
Pada halaman Mealplan, saat memuat halaman tersebut terdapat sedikit jeda saat banner iklan tersebut memuat, dan menyebabkan layout shift ketika banner iklan tersebut selesai memuat.

Gambar. 3.5 Banner Iklan Halaman Mealplan

Setelah banner iklan tersebut selesai dimuat terjadi pergeseran komponen pada halaman tersebut yang menyebabkan pengalaman pengguna yang kurang optimal, karena kemungkinan terjadi misclick atau aksi yang tidak disengaja. Sehingga disini bisa diidentifikasikan bahwa bisa dilakukan caching dengan melakukan prefetching banner iklan sebelum user bernavigasi ke halaman mealplan.

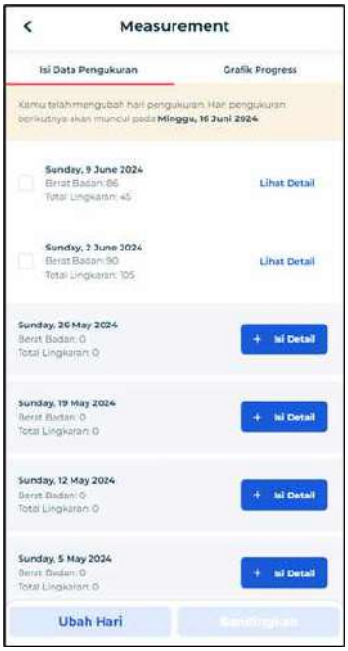
D. Hasil Identifikasi Halaman Measurement

Pada aplikasi ini juga terdapat halaman measurement, yaitu bagian aplikasi untuk mencatat dan mengukur berat badan.



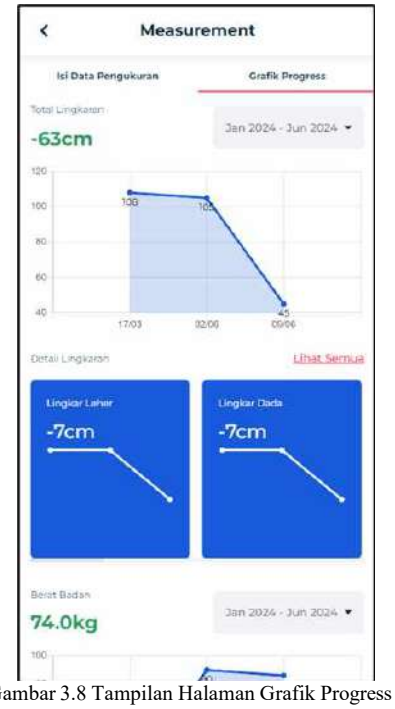
Gambar 3.6 Tampilan Loading pada Halaman Measurement

Saat user bernavigasi ke halaman measurement, terdapat suatu loading spinner yang muncul untuk menunggu data measurement tersebut sudah selesai dilakukan pemanggilan API.



Gambar 3.7 Tampilan Halaman Data Pengukuran

Setelah selesai dimuat, di halaman ini terdapat 2 bagian yaitu data pengukuran dan grafik progress. Pada halaman data pengukuran terdapat bagian yang melihatkan daftar data pengukuran berat badan user yang sudah pernah dimasukkan ke aplikasi sebelumnya.



Gambar 3.8 Tampilan Halaman Grafik Progress

Pada bagian grafik progress, user bisa melihat grafik dari data berat badan dan pengukuran lingkar badan dengan detail.

Setelah di-identifikasi, pada halaman measurement juga dapat dilakukan optimasi dengan implementasi caching dan pre-fetching untuk mengurangi load time awal dan menyimpan data tersebut apabila user bernavigasi ke aplikasi lalu kembali ke halaman tersebut agar tidak melakukan pemanggilan API dan terjadi loading kembali.

E. Implementasi Caching pada Halaman Login

Pada halaman ini, dilakukan refaktorisasi kode untuk mengimplementasi cache data user saat login.

```
queryClient
  .fetchQuery({
    queryKey: ['profile'],
    queryFn: GetProfileApi,
    staleTime: 5 * 60 * 1000,
  })
  .then((res) => {
    let user = res.data.data
    dispatch.user.setUser(user)
    router.push(`/${router.query.redirect}`)
  })
```

Gambar 3.9 Tampilan Kode Caching untuk Profil

Fungsi tersebut memanggil API untuk mendapatkan data user menggunakan Tanstack Query, data user tersebut digunakan pada keseluruhan aplikasi, sehingga sangat penting untuk dilakukan cache.

Jika data profil tersebut dibutuhkan, kode yang diperlukan adalah pemanggilan query yang sudah ditetapkan pada useGetProfile()

```
export const useGetProfile = () => {
  const dispatch = useDispatch()
  const query = useQuery({
    queryKey: ['profile'],
    queryFn: () => GetProfileApi(),
    select: (data) => data.data.data,
    staleTime: 5 * 60 * 1000,
    onSuccess: (data) => {
      dispatch.user.setUser(data)
    },
  })

  const user = query.data

  return {
    ...query,
    user,
  }
}
```

Gambar 3.10 tampilan kode untuk mendapatkan data profil

Jika suatu halaman membutuhkan data profil dan menggunakan kode tersebut, maka user tidak perlu memanggil API profil lagi karena sudah disimpan pada cache.

F. Implementasi Caching pada Halaman Dashboard

Peneliti juga mengimplementasikan Tanstack Query pada Halaman Dashboard agar data tersebut disimpan pada cache.

```
queryClient.prefetchQuery({
  queryKey: ['ads-list'],
  queryFn: GetAdsList,
  staleTime: 1000 * 60 * 60 * 6,
})
queryClient.prefetchQuery({
  queryKey: ['ads-popup'],
  queryFn: GetAdsPopup,
  staleTime: 1000 * 60 * 60 * 6,
})
queryClient.prefetchQuery({
  queryKey: ['dashboard', date],
  queryFn: () => GetDashboardApi(date),
  staleTime: 5 * 60 * 1000,
})
```

Gambar 3.11 tampilan kode untuk caching dashboard

Pada kode tersebut, juga dilakukan prefetching untuk API ads dan juga dashboard, yang akan digunakan pada halaman dashboard, sehingga seluruh data sudah siap digunakan pada aplikasi saat user bernavigasi ke halaman tersebut, dan mengurangi waktu layout shift pada halaman tersebut.

G. Implementasi caching pada Halaman Mealplan

Pada halaman mealplan, juga dilakukan implementasi caching menggunakan Tanstack Query.

```
queryClient.prefetchQuery({
  queryKey: ['ads-sponsored'],
  queryFn: GetAdsSponsored,
  staleTime: 1000 * 60 * 60 * 6,
})
```

Gambar 3.12 tampilan kode untuk caching banner iklan pada mealplan

Pada kode tersebut, juga dilakukan pre-fetching API ads-sponsored, yaitu API untuk mendapatkan gambar banner iklan yang akan dimunculkan di bawah search bar, sehingga data tersebut sudah ada sebelum user bernavigasi, dan tidak terjadi layout shift.

H. Implementasi caching pada Halaman Measurement

Dilakukan juga implementasi caching dan pre-fetching pada halaman measurement untuk mengurangi waktu loading awal seperti yang dilihat pada tahap identifikasi.

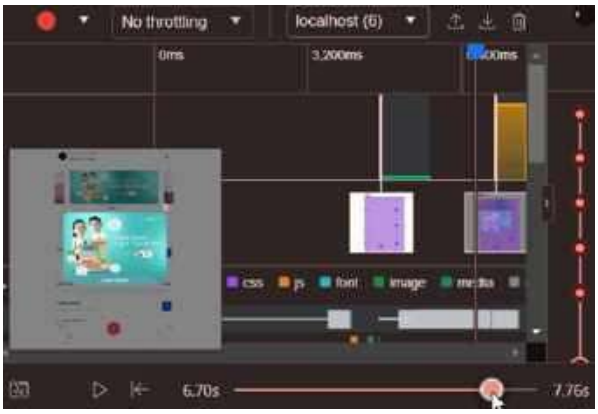
```
queryClient.prefetchQuery(['measurements'],
    () => fetchMeasurements(1),
    { staleTime: 1000 * 60 * 5 })
```

Gambar 3.13 tampilan kode untuk caching halaman measurement

Pada kode tersebut, dilakukan query kepada API measurements dan juga pre-fetching dengan waktu cache yang di set selama 5 menit.

I. Pengujian Response Time Halaman Dashboard

Peneliti sudah melakukan pengujian pada aplikasi yang didapatkan menggunakan Performance Tab pada Chrome DevTools untuk mengukur kecepatan aplikasi



Gambar 3.14 Pengujian Dashboard sebelum di-caching dengan Chrome Devtools

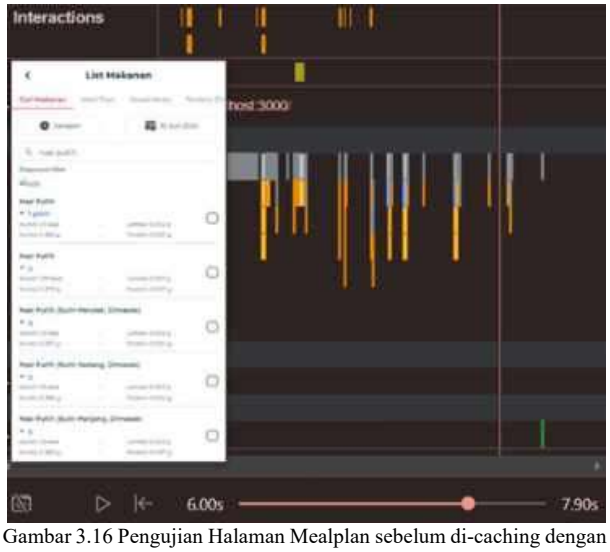
Gambar diatas menunjukkan timeline aplikasi dan pada aplikasi sebelum dilakukan caching, waktu yang dibutuhkan semenjak user melakukan login dan masuk ke dashboard pertama kali, membutuhkan 6.70 detik untuk seluruh data didapatkan dan me-render semua komponen pada halaman tersebut, ini dikarenakan data tersebut baru di-fetching saat di halaman tersebut, sehingga komponen tidak langsung di-render oleh aplikasi dan menyebabkan layout shift.

Gambar 3.15 Pengujian Dashboard setelah di-caching dengan Chrome Devtools

Setelah dilakukan implementasi caching, waktu yang dibutuhkan semenjak user melakukan login, dan masuk ke dashboard pertama kali, membutuhkan waktu 4.43 detik. Lebih cepat 33.8% dibandingkan sebelumnya, karena telah dilakukan cache dan prefetching, sehingga saat user ke halaman dashboard, seluruh komponen tersebut me-render secara instan, tanpa adanya layout shift.

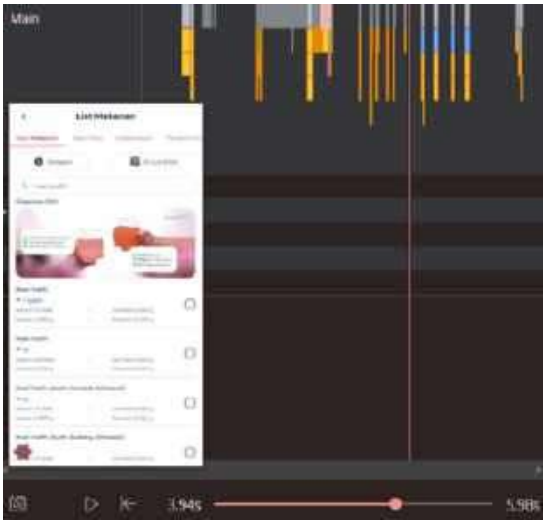
J. Pengujian Response Time Halaman Mealplan

Dilakukan juga pengujian Response Time aplikasi pada halaman mealplan khusus nya pada bagian banner iklan.



Gambar 3.16 Pengujian Halaman Mealplan sebelum di-caching dengan Chrome Devtools

Pada aplikasi sebelum di *caching*, bagian pada banner iklan menunjukan gambar yang tidak dimuat, sehingga akan menggeserkan komponen dibawahnya, dan akan memuat setelah 6 detik yang dihitung dari navigasi dashboard hingga halaman mealplan.



Gambar 3.17 Pengujian Halaman Mealplan sebelum di-caching dengan Chrome Devtools

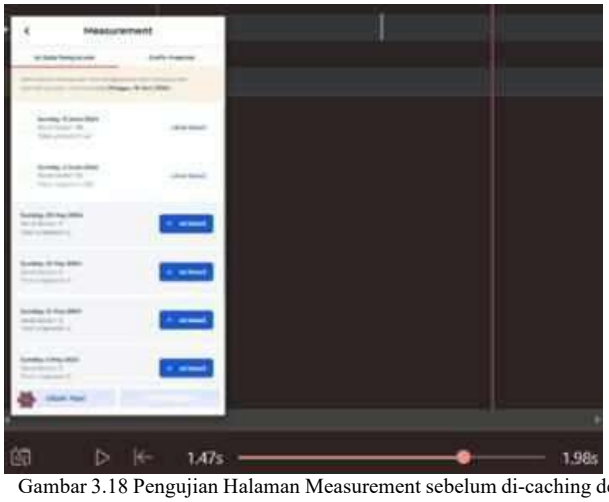
Pada aplikasi setelah dilakukan caching, banner iklan tersebut sudah memuat secara instan tanpa ada placeholder gambar yang akan menggeser komponen dibawah, dan itu sudah dimuat selama 3.94 detik interaksi user dari dashboard hingga ke halaman mealplan. Lebih cepat 34.3% dari sebelum dilakukan implementasi caching.

K. Pengujian Response Time Halaman Measurement

Pada halaman measurement, juga dilakukan pengujian sebelum dan sesudah implementasi caching. Sebelum dilakukan caching, terdapat state loading yang cukup lama sebelum halaman measurement dapat dimuat.

Gambar 3.18 Pengujian Halaman Measurement sebelum di-caching dengan Chrome Devtools

Berdasarkan pengujian dengan Chrome Devtools, halaman measurement sebelum di-caching menunjukkan sedikit jeda sebelum memuat halaman tersebut, pada waktu 2.93 detik yang dihitung sejak user bernavigasi ke halaman tersebut, masih terlihat bahwa halaman tersebut menunjukkan state loading.

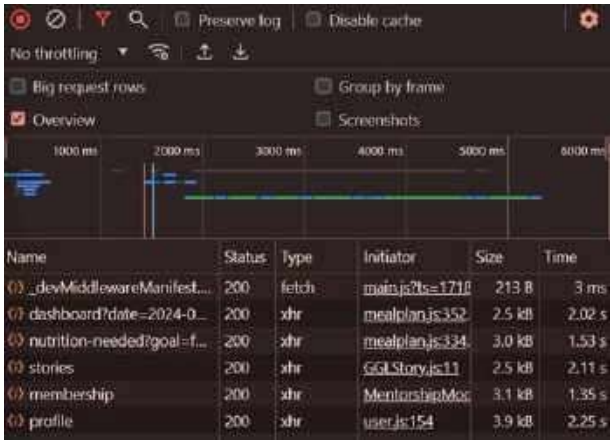


Gambar 3.18 Pengujian Halaman Measurement sebelum di-caching dengan Chrome Devtools

Setelah di—implementasikan caching, halaman tersebut dapat memuat data measurement secara instan, yaitu hanya 1.47 detik yang dihitung semenjak user bernavigasi ke halaman tersebut. Sehingga lebih cepat sebesar 49.8% dari sebelum diimplementasikan caching.

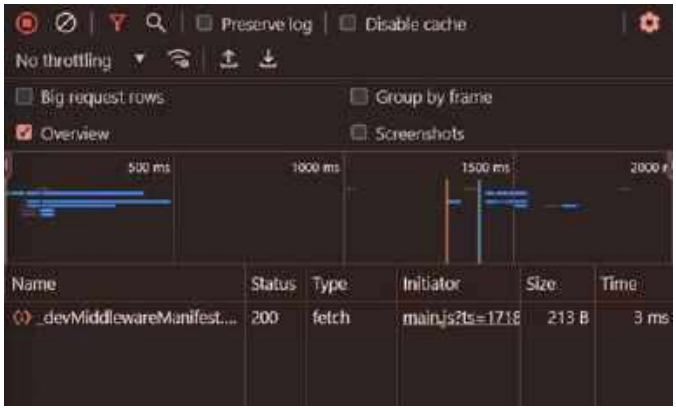
L. Pengujian Efektivitas Pemanggilan API

Peneliti juga melakukan pengujian pada efektivitas pemanggilan API, yaitu melihat apakah ada pemanggilan API yang duplikat dalam aplikasi yang sebenarnya bisa dilakukan caching pada aplikasi dengan menggunakan Network Tab pada Chrome Devtools. Skenario yang dilakukan adalah melakukan reload page pada bagian dashboard aplikasi.



Gambar 3.19 Bagian Network Tab Chrome Devtools pada aplikasi sebelum di-caching

Pada aplikasi sebelum di implementasi caching, saat melakukan reload page, maka seluruh API nya tetap terpanggil kembali, dan membutuhkan waktu total 9.29 detik. Walaupun sebelumnya saat memuat aplikasi pertama kali, seluruh data tersebut sudah di panggil, sehingga sebenarnya bisa dilakukan caching disini untuk meningkatkan efektivitas pemanggilan API.



Gambar 3.20 Bagian Network Tab Chrome Devtools pada aplikasi sebelum di-caching

Pada aplikasi setelah implementasi caching, saat halaman tersebut di reload, tidak ada pemanggilan API lagi, karena seluruh data yang sebelumnya dimuat sudah disimpan pada cache, sehingga tidak diperlukan pemanggilan API lagi. Ini juga mempunyai hasil yang sama dengan halaman lainnya yang sudah di-implementasi caching yaitu halaman mealplan dan halaman measurement.

M. Analisis Hasil

Peneliti melakukan analisa hasil dari implementasi caching pada aplikasi fitness berbasis Next.js milik PT. Anugerah Wijaya Raga. Peneliti menganalisis response time dalam aplikasi yang dipengaruhi oleh pemanggilan API yang tidak efektif karena tidak adanya caching, dan setelah dilakukan implementasi, terdapat perkembangan performa yang signifikan, sehingga meningkatkan responsivitas aplikasi dan pengalaman pengguna.

IV. KESIMPULAN

Berdasarkan hasil penelitian, implementasi, pengujian, dan pembahasan yang telah dilakukan, maka dapat ditarik suatu kesimpulan sebagai berikut:

1. Implementasi metode caching dalam penelitian ini terbukti efektif dalam mengoptimalkan waktu respons aplikasi fitness milik PT. Anugerah Wijaya Raga. Sebelum optimasi, waktu yang dibutuhkan untuk login hingga ke halaman dashboard adalah 6,70 detik. Setelah optimasi, kecepatan waktu respons aplikasi meningkat menjadi 4,43 detik (33,8%). Pengujian pada halaman mealplan menunjukkan peningkatan kecepatan dari 6 detik menjadi 3,94 detik (34,3%), dan pada halaman measurement dari 2,93 detik menjadi 1,47 detik (49,8%). Selain itu, efektivitas pemanggilan API juga meningkat, dengan tidak adanya pemanggilan API yang duplikat setelah implementasi caching.
2. Penerapan caching dan prefetching menggunakan Tanstack Query meningkatkan responsivitas aplikasi, yang berkontribusi pada pengalaman pengguna yang lebih baik. Pengguna dapat merasakan peningkatan kecepatan akses data dan interaksi yang lebih lancar pada aplikasi. Hal ini

terjadi karena komponen pada aplikasi tersebut me-render secara langsung karena data yang dibutuhkan telah disimpan pada cache.

REFERENSI

[1]. Adam Hanif Putra Hadi, Implementasi Progressive Web Application Dengan Teknologi Service Worker Pada Web Artikel Kebugaran Jasmani, 2022.
[2]. Ahmad Nizar, Mobile Web Search Results Refactor (Traveloka Xperience Product), 2022.
[3]. Jia Wang, A Survey of Web Caching Schemes for the Internet, 1999.
[4]. I. Malavolta, K. Chinnappan, L. Jasmontas, S. Gupta, dan K. A. K. Soltany, "Evaluating the impact of caching on the energy consumption and performance of progressive web apps," 2020.
[5]. P. Evergreen, Selecting a state management strategy for modern web frontend applications, 2023.