Implementasi Dan Manajemen State Pada Website Next.Js: Perbandingan Context Api Dan Redux Pada Website Mangaice

Wisnu Shena Arrafi[1], Ricky Eka Putra[2]

[1,2] Teknik Informatika,Fakultas Teknik, Universitas Negeri Surabaya
[1]wisnu.20057@mhs.unesa.ac.id
[2]rickyeka@unesa.ac.id

***Abstrak*—Pengelolaan state yang efisien menjadi tantangan dalam pengembangan website modern terutama ketika website memiiki tingkat kompleksitas yang tinggi. Pemillihan tools yang tepat dalam memanajemen state sering kali menimbulkan kebingungan bagi developer, mengingat masing-masing tools memiliki pendekatan yang berbedan dan keunggulan serta kekurangan masing-masing dalam hal performa, skalabilitas, dan memori. Contoh tools yang populer digunakan dalam manajemen state adalah Context API dan Redux. Oleh karena itu, penelitian ini mengimplementasikan dan melakukan perbandingan terhadap Context APi dan Redux pada website Mangaice yang dikembangkan dengan menggunakan framework Next.js. Perbandingan dilakukan berdasarkan parameter performa, skalabilitas, dan memori. Penelitian ini diharapkan dapat menjadikan panduan empiris dalam memilih state management tools yang sesuai dengan kebutuhan projek. Penelitian dilakukan dengan tahapan yang pertama adalah studi literatur, analisis kebutuhan, implementasi, dan pengujian. Pengujian dalam penelitian dilakukan dengan menjalankan skenario penggunaan yang mencangkup autentikasi, pencarian dan penambahan library, membaca manga dan penyimpanan progress baca, pengelolaan history baca, dan pengaturan preferensi pengguna. Dari hasil pengujian, didapatkan hasil bahwa Redux unggul 11,16% dalam performa, terutama dalam menampilkan konten utama dan rendering pada repeat view, sedangkan Context API lebih efisien dalam scripting dan painting. Dari segi skalabilitas, Redux lebih unggul 25,91%, dengan waktu respons lebih cepat, throughput lebih tinggi, dan stabilitas lebih baik dalam skenario dengan banyak pengguna. Dalam penggunaan memori, Redux lebih efisien dengan penghematan total sebesar 51,0% dibandingkan Context API setelah adanya interaksi pengguna. Pada akses awal, Redux juga lebih hemat sekitar 3,5%.**

***Kata Kunci*— Context API, Redux, Perbandingan State Management, Pengembangan Website.**

I. PENDAHULUAN

A. Latar Belakang

Web Development merupakan salah satu bidang IT yang mengalami perkembangan selama beberapa tahun terakhir. Website sekarang sudah bukan lagi sebagai media yang hanya sebatas menampilkan informasi. Namun, website sudah berevolusi menjadi aplikasi yang dapat memberikan layanan beragam. Kalau kita simpulkan secara singkat, trend perkembangan web meliputi beberapa area antara lain progressive web application, implementasi artificial intelligence, dan teknologi serverless[1]. Perubahan ini didorong oleh kebutuhan pengguna untuk mengakses informasi dan layanan secara real-time, interaktif, dan responsif [2]. Dalam pengembangan website, banyak framework yang telah dikembangkan untuk mempermudah dalam pengembangan. Masing-masing framework memiliki kelebihan dan kekurangan masing-masing yang dapat dijadikan acuan oleh developer dalam mengembangkan website. Salah satu framework yang populer adalah Next.js

Next.js merupakan framework React yang digunakan untuk membuat *full-stack web application*, dengan menggunakan React *component* untuk *interface* dan Next.js untuk *tools* tambahan dan optimalisasi[3]. Next.js sekarang telah menjadi pilihan framework populer bagi para developer dalam pengembangan website modern karena mendukung server-side-rendering (SSR) yang dapat meningkatkan performa SEO. Selain itu, Next.js juga menyediakan fitur seperti static site generation (SSG), automatic code splitting, dan API routes, yang membuat Next.js sangat fleksibel dan efisien.

Dalam sisi Frontend Development, salah satu aspek penting adalah state management. State management merujuk pada cara mengelola dan menyimpan informasi atau data dalam aplikasi[4]. State mengacu pada data yang merepresentasikan suatu kondisi/keadaan saat ini. State bersifat mutable, data dalam state dapat berubah-ubah tergantung dengan kondisi selama website berjalan. Dengan semakin kompleksnya website, kemampuan developer dalam memanajemen state menjadi sebuah keharusan. Oleh karena itu, state management yang efektif merupakan faktor penting agar website dapat menjalankan fungsinya dengan baik, responsive, dan penting untuk skalabilitas, *maintainability*, dan performa website. Sayangnya, implementasi *state management* tak terlepas dari beberapa problematika yang harus dihadapi. Contohnya seperti *prop drilling,* kesulitan sinkronisasi state antar-komponen, hingga kompleksitas *debugging*. Untuk mengatasi masalah tersebut, kini banyak berkembangnya *tools* yang mempermudah dalam manajemen state. Contoh *tools* yang populer yakni Context API dan Redux.

Context API merupakan library state management bawaan dari React, yang memungkinkan manajemen state dengan efektif dan sederhana tanpa memerlukan tambahan library. Context API berguna untuk aplikasi yang sederhana tanpa memerlukan kebutuhan state yang kompleks[5]. Context API merupakan solusi *native* yang tidak memerlukan *library* tambahan dan menjadi solusi pertama untuk mengatasi *prop*

drilling yang merupakan masalah fundamental dalam React. Disisi lain, Redux merupakan library state management independen yang banyak digunakan dalam pengembangan website. Redux telah menjadi standar industri dengan *track record* yang proven di aplikasi *production*. Redux menawarkan dan menyediakan arsitektur yang terstruktur dengan menggunakan konsep one-way data flow, mencangkup penggunaan store, middleware, reducer, dan action. Redux memiliki ekosistem yang kaya seperti Redux DevTools dan middleware.

Dalam pengembangan website, performa, terutama dalam hal *rendering* dan *re-rendering* komponen, dapat dipengaruhi oleh cara *state* dikelola. Pengelolaan *state* yang buruk dapat mempengaruhi performa website menjadi buruk, dengan dampak langsung pada *loading time*, skalabilitas, penggunaan memori, dan performa website secara keseluruhan. Disamping itu, penerapan dukungan untuk Single-Sign-On (SSO), sebagai fitur dalam aplikasi web modern, memerlukan integrasi yang mulus dengan sistem manajemen state guna menjaga kenyamanan dan keamanan pengguna. Pemilihan *tools* atau *library state management* yang tepat menjadi faktor kunci untuk memudahkan *developer* dalam mengimplementasikan sistem *state management* guna mendukung website agar dapat menjalankan fungsinya dengan baik dan memberikan kenyaman kepada *user* dalam menggunakan website tersebut.

Maka dari itu, peneliti mencoba melakukan penelitian terkait implementasi Context API dan Redux sebagai *state management tools*, dan melakukan perbandingan serta analisis dalam hal performa, skalabilitas, memori, dll. Penelitian ini dilakukan dengan mengimplementasikan Context API dan Redux pada website Mangaice yang dikembangkan dengan menggunakan framework Next.js. Website Mangaice, sebagai studi kasus penelitian, menawarkan kesempatan untuk mengeksplorasi dan membandingkan Context API dan Redux dalam konteks nyata. Dengan demikian, penelitian ini tidak hanya bertujuan untuk memberikan wawasan teoritis tetapi juga menyediakan bukti empiris mengenai kelebihan dan kekurangan masing-masing metode. Hasil dari penelitian ini diharapkan dapat memberikan wawasan kepada para developer dalam memilih metode state management yang tepat untuk website mereka.

II. METODE PENELITIAN

Tujuan utama dari dilakukannya penelitian ini adalah untuk mengetahui perbandingan antara penggunaan Context API dengan Redux sebagai state management tools dalam ruang lingkup aplikasi berbasis React, yang dimana perbandingan ini diharapkan dapat dijadikan acuan untuk mempertimbangkan pemilihan dan penggunaan state management tools pada aplikasi web berbasis React. Oleh karena itu, jenis penelitian yang sesuai untuk penelitian ini adalah penelitian komparatif eksperimental. Jenis penelitian ini merupakan jenis penelitian yang bertujuan untuk membandingkan dua atau lebih kelompok dalam kondisi teratur dan terkontrol dengan tujuan untuk menemukan perbedaan di antara mereka berdasarkan suatu variabel tertentu. Penelitian ini melibatkan proses kontrol variabel yang dapat mempengaruhi hasil penelitian sehingga perbandingan antara metode menjadi adil. Dalam penelitian ini, peneliti mengimplementasikan masing-masing metode dalam kondisi yang sama, kemudian melakukan pengukuran kepada keduanya. Hasil penelitian dibandingkan guna mengetahui mana metode yang lebih efektif berdasarkan kriteria-kriteria yang telah ditentukan. Penelitian ini diharapkan dapat memberikan wawasan teoritis dan praktis yang dapat berkontribusi pada literatur yang ada, dan juga diharapkan dapat memberikan wawasan mengenai kelebihan dan kekurangan penggunaan masing-masing metode sehingga dapat menjadi acuan developer dalam memilih metode untuk state management untuk aplikasi website mereka.

Rancangan penelitian yang digunakan adalah dengan menggunakan metode pendekatan pengembangan untuk mengimplementasikan dan membandingkan kedua metode *state management* dalam website Mangaice. Dengan menggunakan rancangan ini, memungkinkan untuk dilakukan komparasi secara langsung dalam lingkungan pengembangan yang adil dan dalam kondisi operasional yang nyata. Dalam hal ini, website Mangaice perlu dikembangkan agar dapat mengimplementasikan kedua metode tersebut. Data/state dalam website akan dimanajemen dengan Context API dan Redux, sehingga dalam penelitian ini akan terdapat dua kondisi eksperimental, yakni website Mangaice yang menggunakan Context API dan website Mangaice yang menggunakan Redux. Implementasi kedua metode tersebut kemudian dilakukan observasi dan pengujian meliputi performa, skalabilitas, memori, dll. Parameter tersebut kemudian dianalisis dan dilakukan perbandingan parameter dari masing-masing metode pendekatan sehingga dapat diambil keunggulan dan kekurangan masing-masing metode. Dengan demikian, rancangan penelitian ini menyediakan bukti empiris mengenai perbandingan implementasi state management tools dapat mempengaruhi suatu kinerja aplikasi website, selain itu juga menyoroti implikasi nyata terhadap penggunaan masing-masing metode dalam aplikasi website.

Gbr 1. Rancangan Penelitian

Berdasarkan gambar 1, dapat diketahui tahapan penelitian yang diterapkan adalah sebagai berikut:

A. *Studi Literatur*

Penelitian menggunakan sumber informasi dari jurnal-jurnal maupun penelitian yang telah dilakukan dalam

rentang tahun 2019 - 2023. Sumber yang digunakan dalam mencari informasi adalah melalui Google Scholars, Mendeley, dan beberapa sumber jurnal lainnya. Point utama yang ditekankan dalam mencari sumber informasi adalah literatur terkait dengan topik *state management*, penggunaan context API, penggunaan Redux, dan pengembangan website modern menggunakan framework Next.js.

B. *Analisis Kebutuhan*

Pada tahap penelitian ini, dilakukan penentuan persyaratan dan kebutuhan untuk menunjang penelitian atau kebutuhan yang dimungkinkan perlu ada dalam proses pengembangan website Mangaice dan implementasi Context API atau Redux. Berikut merupakan persyaratan dan kebutuhan yang dipertimbangkan:

1. Framework dan Library

Next.js dan React.js digunakan sebagai framework utama dalam proses pengembangan website Mangaice. Framework Next.js digunakan sebagai kerangka kerja utama React yang mendukung server-side rendering (SSR) dan pembuatan aplikasi web statis. SSR memberikan keuntungan dalam hal performa website dan SEO dikarenakan halaman dapat di-render di server sebelum dikirimkan ke client. Disisi lain, framework React digunakan untuk pembangunan antarmuka pengguna pada website Mangaice, memanfaatkan komponen yang dinamis dan efisien untuk menciptakan pengalaman pengguna yang responsif dan interaktif.

2. State Management Tools.

State management tools digunakan dalam pengembangan sebagai alat untuk memanajemen *state* sekaligus sebagai variabel yang akan dilakukan perbandingan. Masing-masing state management tools akan diimplementasikan pada website Mangaice secara terpisah dan akan dilakukan analisis dari segi performa, memori, skalabilitas. *State management tools* yang digunakan adalah Context API dan Redux.

3. Testing Tools

Testing tools merupakan alat yang digunakan untuk mengukur parameter-parameter yang digunakan sebagai pembanding dari Context API dan Redux. Testing tools yang digunakan dalam penelitian ini antara lain:

a. Web Page Test: *Tools* ini memberikan analisis mengenai kinerja website termasuk waktu respon API, memori, dll.
b. JMeter: Tools yang digunakan untuk menguji beban dan kinerja website. JMeter dapat mensimulasikan banyak pengguna yang mengakses website secara bersamaan untuk mengukur kapasitas penanganan beban dan stabilitas sistem.
c. Google Web Dev Console: Serangkaian tools developer web yang langsung disertakan ke dalam browser Google Chrome. Tools ini memungkinkan developer dapat mendiagnosis dengan cepat sehingga dapat membantu mengembangkan website yang cepat dan lebih baik. Beberapa tools yang disediakan antara lain tab memory, tab console, tab performance, dan network, dll.

4. Fungsionalitas Website

Website Mangaice perlu memiliki fungsionalitas yang dapat membantu dalam membandingkan Context API dengan Redux secara maksimal sehingga fungsionalitas website diasumsikan harus dapat menyediakan sarana agar dapat menggunakan Context API dan Redux semaksimal mungkin. Fungsionalitas website tersebut antara lain sebagai berikut:

TABEL I
FUNGSIONALITAS WEBSITE MANGAICE

No	Fungsionalitas	Deskripsi
1	User Authentication & User Authorization	Implementasi sign in, sign up, dan logout
2	Dynamic Data Handling	Data dinamis seperti profil pengguna, daftar bacaan manga, daftar manga favorit, dll.
3	Stateful Component	Komponen yang memerlukan manajemen *state* seperti reading list, history, reading layout, dsb.
4	Search & Filter	Pencarian dan penyaringan lanjutan sesuai dengan keinginan pengguna

5. Skenario Interaksi Website

Dalam melakukan pengujian Context API dan Redux, perlu adanya suatu skenario percobaan interaksi website dengan pengguna guna memaksimalkan kinerja kedua state management tools tersebut dan untuk mendapatkan hasil perbandingan yang komprehensif antara Context API dan Redux. Peneliti mengembangkan serangkaian skenario yang mewakili interaksi pengguna dengan website secara menyeluruh. Skenario ini dirancang untuk menguji berbagai aspek manajemen state pada Mangaice, mulai dari autentikasi, manipulasi data, pencatatan riwayat, hingga kostumisasi pengaturan pengguna.

Pemilihan skenario didasarkan pada karakteristik aplikasi berbasis konten seperti Mangaice yang memerlukan pengelolaan state yang efisien untuk memastikan pengalaman pengguna yang baik. Dengan menggunakan skenario ini, peneliti bertujuan untuk mengukur performa, skalabilitas, dan penggunaan memori dari implementasi kedua state management tools tersebut. Skenario tersebut antara lain:

TABEL II
SKENARIO INTERAKSI WEBSITE MANGAICE

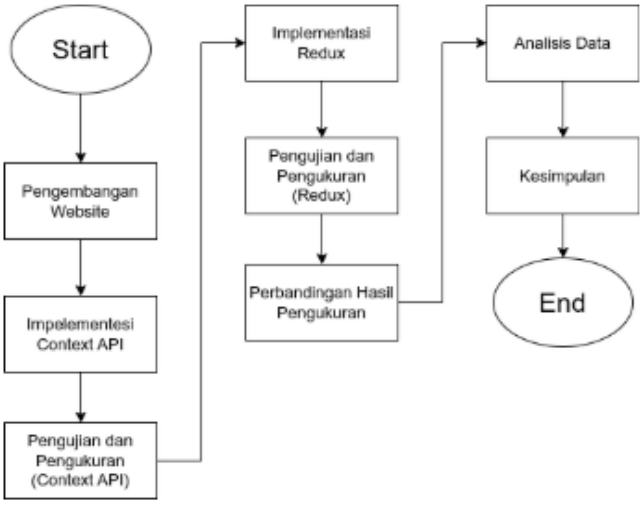
No	Skenario	Deskripsi

1.	Autentikasi (Register & Login)	Menguji kemampuan manajemen state dalam menangani data pengguna
2.	Pencarian dan penambahan ke Library	Menguji pengelolaan data dari API dan memanipulasi data koleksi pengguna
3.	Membaca manga	Menguji performa dalam menangani navigasi antar halaman dan menyimpan progress manga
4.	Pengelolaan History	Menyentuh aspek (CRUD) yang penting dalam pengujian manajemen state
5.	Pengaturan preferensi	Menguji persistensi perubahan config pada website dan bagaimana perubahan mempengaruhi komponen lain

C. *Implementasi*

Pada tahap ini, peneliti mulai melakukan implementasi Context API dan Redux dalam website Mangaice. Implementasi *tools* dilakukan dua kali dalam website Mangaice, yang pertama adalah implementasi Context API dan yang kedua adalah Redux. Variabel independen dalam penelitian ini adalah metode state management, yakni Context API dan Redux. Kemudian untuk variabel dependen yakni kinerja aplikasi, mencangkup performa, penggunaan memori, skalabilitas, arsitektur, konsep dasar, fleksibilitas, dan kemampuan kostumisasi kedua *tools* tersebut.

Secara sederhana, langkah-langkah implementasi dapat digambarkan pada gambar 2 berikut:



Gbr 2. Langkah-Langkah Implementasi

D. *Pengujian*

Pengujian dilakukan terhadap Context API dan Redux guna mengetahui perbandingan kedua *state management tools* tersebut dalam hal performa, skabilitas, dan memori. Oleh karena itu, setelah pengembangan website Mangaice selesai, selanjutnya akan dilakukan pengujian terhadap Context API dan Redux sebagai berikut:

1. Pengujian Performa

Pengujian performa dilakukan untuk mengukur efisiensi dan kecepatan dari masing-masing state management tools dalam berbagai kondisi operasional. Parameter yang diukur dalam pengujian ini antara lain:

TABEL III
PARAMETER PENGUJIAN PERFORMA

No	Parameter	Deskripsi
1.	Time to First Byte (TTFB)	Waktu yang diukur dari pengiriman permintaan pertama pengguna hingga saat byte pertama dari halaman diterima server. TTFB mempengaruhi waktu keseluruhan yang diperlukan untuk memuat halaman.
2.	First Contentful Paint (FCP)	Waktu saat pengguna mulai memuat halaman hingga saat konten pertama website muncul.
3.	Start Render	Waktu yang diukur dan pertama kali permintaan dikirimkan hingga sesuatu ditampilkan di layar.
4.	Cumulative Layout Shift (CLS)	Jumlah total dari pergeseran layout komponen yang tidak terduga.
5.	Largest Contentful Paint (LCP)	Waktu dari saat pengguna mengakses halaman hingga saat konten terbesar pada layar selesai dimuat.
6.	Total Blocking Time (TBT)	Merupakan total waktu halaman diblokir agar tidak merespon input dari pengguna. Jumlah ini dihitung dengan menambahkan bagian pemblokiran dari semua tugas berdurasi panjang antara FCP dan waktu untuk interaktif. TBT yang rendah

		menunjukan bahwa halaman dapat berinteraksi dengan halaman lebih cepat.
7.	Page Weight	Merupakan total dari semua sumber daya yang digunakan oleh halaman, termasuk gambar, script, dan stylesheet.
8.	Loading	Keseluruhan waktu yang dibutuhkan untuk memuat halaman sepenuhnya.
9.	Scripting	Waktu yang digunakan untuk menjalankan script javascript halaman.
10.	Rendering	Waktu yang digunakan untuk oleh browser untuk merender halaman.
11.	Painting	Waktu yang dibutuhkan browser untuk menggambar elemen visual ke layar pengguna.
12.	System	Mengukur penggunaan sumber daya sistem seperti CPU dan memori selama pemuatan halaman.

Adapun *tools* yang digunakan dalam pengujian adalah Google Web Dev Console dan Web Page Test.

2. Pengujian Skalabilitias

Pengujian skalabilitas mengevaluasi bagaimana kedua state management tools dapat menangani peningkatan jumlah data dan kompleksitas aplikasi website. Parameter yang diukur diantaranya yakni:

TABEL IV
PARAMETER PENGUJIAN SKALABILITAS

No	Parameter	Deskripsi
1.	Median Response Time	Merupakan waktu respons rata-rata dari semua respons yang diukur. Memberikan gambaran umum mengenai keseluruhan kinerja website.
2.	Throughput	Jumlah transaksi yang diselesaikan per detik. Membantu untuk mengidentifikasi kemampuan sistem dalam menangani beban.
3.	Success Rate	Proporsi respons yang berhasil dibandingkan dengan total respons. Menunjukan tingkat keandalan sistem dalam menangani respons.
4.	99th %tile Response	Waktu tanggap terbaik sistem yang dicapai oleh 99% dari semua *response*. Membantu mengidentifikasi kinerja terbaik sistem.
5.	Request	Jumlah parameter yang dikirimkan dan berhasil ditangani oleh website.
6.	Median Script Execution	Waktu rata-rata dari eksekusi semua script pada website.
7.	99th %tile Script Execution	Waktu eksekusi script terbaik yang dicapai oleh 99% dari semua script.

Adapun *tool* yang digunakan pada pengujian ini adalah JMeter.

3. Pengujian Memori

Pengujian memori digunakan untuk mengetahui berapa penggunaan memori yang diperlukan aplikasi ketika mengimplementasikan Context API atau Redux. Dengan mengetahui penggunaan memori, kita dapat mengetahui efisiensi dari masing-masing state management tools tersebut terutama untuk website dengan data intensif. Parameter yang dijadikan tolak ukur penggunaan memori Context API dan Redux adalah sebagai berikut:

TABEL V
PARAMETER PENGUJIAN MEMORI

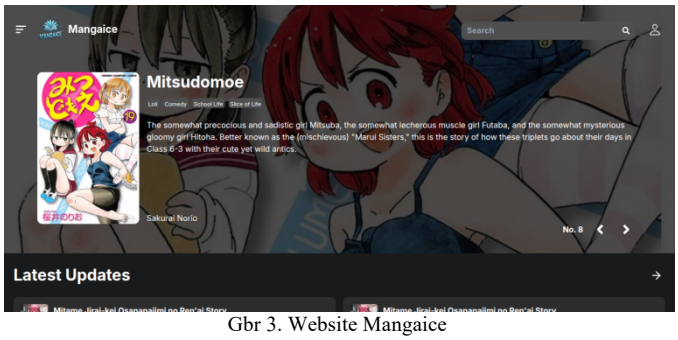
No	Parameter	Deskripsi
1.	Code	Penggunaan memori oleh kode Javascript dalam website. Kode ini termasuk fungsi dan objek yang dibuat oleh kode Javascript. Code ini akan menunjukan seberapa banyak memori yang digunakan website untuk logika, fungsi, dan algoritma website.
2.	String	Penggunaan memori yang digunakan oleh string pada

		website. String dapat menjadi salah satu penyebab pemborosan penggunaan memori, khususnya untuk aplikasi yang banyak menampilkan konten, seperti website untuk membaca manga. Memantau penggunaan memori untuk string dapat menjadi identifikasi apakah ada kebocoran atau inefisiensi dalam penggunaan string.
3.	JS Arrays	Penggunaan memori oleh array Javascript. Array sering digunakan untuk menyimpan dan mengolah data. Ukuran dan kompleksitas data yang disimpan pada array dapat mempengaruhi performa pada website.
4.	Typed Arrays	Penggunaan memori oleh Typed Arrays, yakni array yang digunakan untuk penyimpanan data biner mentah. Typed Arrays ini sering digunakan untuk aplikasi yang membutuhkan pemrosesan data intensif seperti grafik, audio, video, atau mengakses data mentah menggunakan websocket, dsb. Pengukuran ini untuk mengetahui apakah website dapat menangani data biner dengan efisien.
5.	System Object	Waktu yang digunakan untuk mengerjakan operasi sistem seperti garbage collection dan pengelolaan memori oleh browser.

Adapun *tool* yang digunakan pada pengujian ini adalah Google Web Dev Console.

III. HASIL DAN PEMBAHASAN

A. *Studi Kasus Web Mangaice*



Gbr 3. Website Mangaice

Mangaice merupakan website yang menyediakan layanan bacaan manga kepada pengguna. Dengan menggunakan website ini, pengguna dapat mencari beragam manga sesuai dengan keinginan pengguna. Menyediakan layanan pustaka dan pencarian lanjutan, memungkinkan pengguna mencari, menyimpan, dan membaca manga suka hati. Website ini dikembangkan dengan menggunakan framework Next.js dengan tailwind css untuk memberikan style website. Next.js sebagai framework utama memungkinkan rendering cepat baik dari sisi server maupun dari sisi client. Next.js juga menyediakan dinamic routing dan optimasi performa yang baik, terutama untuk website yang membutuhkan pengambilan data dinamis seperti Mangaice.

Website ini terdiri dari 7 halaman utama yakni halaman home, halaman library, halaman history, halaman profile, halaman advanced search, halaman recently updated, halaman latest updated, dan halaman random manga. Dalam segi state management, website Mangaice menggunakan dua pendekatan yang berbeda: Context API dan Redux. Dalam penelitian ini, peneliti mengembangkan website Mangaice untuk melakukan perbandingan pendekatan metode state management. Context API lebih banyak digunakan untuk pengelolaan state yang sederhana dan langsung, seperti pengaturan tema dan status login pengguna. Di sisi lain, Redux lebih banyak digunakan untuk aplikasi yang memerlukan pengelolaan state yang lebih kompleks. Oleh karena itu, peneliti ingin mengetahui, untuk website dengan level kompleksitas seperti Mangaice, apakah memerlukan library state management tools seperti Redux, atau cukup hanya dengan menggunakan tools bawaan dari React seperti Context API.

Adapun komponen-komponen yang terdapat pada website Mangaice adalah sebagai berikut:

TABEL VI
KOMPONEN WEBSITE MANGAICE

No	Komponen
1	Authentication dan User Management
2	User's Manga Library
3	Manga Search dan Filter
4	Notification dan Messages
5	Reading Progress dan History
6	Theme dan Setting Management

B. *Impelementasi Context API dan Redux*

Tahapan ini merupakan tahapan implementasi Context API dan Redux pada website Mangaice. Target peneliti dalam tahap ini adalah mengimplementasikan Context API dan Redux yang kemudian akan dilakukan pengujian dari beberapa parameter untuk menentukan state management *tools* mana yang tepat dalam pengelolaan *state* di website Mangaice.

1. Implementasi Context API

Context API merupakan tools yang dirancang untuk mempermudah bekerja dengan *state* dan menghapus keharusan untuk mengirimkan suatu properti dari komponen tinggi ke komponen rendah dari hirarki *tree* komponen[6]. Terdapat dua hal yang harus dibuat dalam implementasi Context API, yakni context dan provider. Project akan dibagi-bagi menjadi folder-folder sesuai dengan fitur/fungsi masing-masing. Oleh karena itu, terdapat dua folder khusus untuk Context API yakni folder context dan folder provider. Folder context digunakan untuk menyimpan berbagai context yang diperlukan website, dan folder provider menyimpan provider website.

a. Context

Menurut *official documentations* dari React, context merupakan komponen yang menyediakan cara untuk mengirimkan data ke komponen dalam *tree* tanpa secara manual meneruskan data tersebut kedalam properti komponen di setiap level komponen dalam *tree*. Context digunakan untuk mengirimkan data yang dianggap "global" dalam DOM Tree, seperti status autentikasi pengguna, tema, bahasa, dll. Untuk membuat context dapat menggunakan createContext API yang disediakan oleh React. API createContext membuat komponen context yang nantinya dapat kita modifikasi dalam komponen. Contoh penggunaan createContext sebagai berikut:

```
const someContext = createContext(defaultValue)
```

API createContext menerima satu parameter, yakni *default value*. *Value* ini merupakan nilai *default* jika tidak ada nilai yang didefinisikan di provider. Untuk Context API, *state* atau data yang akan dijadikan *state* global hanya dapat didefinisikan di komponen provider context. API createContext hanya mengembalikan objek context, yang dimana objek ini tidak memiliki informasi apapun. Hanya merepresentasikan context mana yang digunakan oleh suatu komponen. Melainkan menggunakan komponen provider untuk mendefinisikan *value* dari context dan menggunakan hooks useContext untuk menggunakan context tersebut.

Context API memungkinkan context dibuat dengan dua cara, yakni global context atau modular context. Global context merupakan context tunggal, artinya context ini dijadikan context untuk semua fitur/fungsi dari website. Misal terdapat fitur autentikasi, fitur history, fitur library, dll. maka fitur-fitur tersebut hanya menggunakan satu context. Cara kedua adalah modular context, dimana untuk setiap fitur dibuat satu context khusus untuk penyedia datanya. Jadi jika terdapat fitur autentikasi, fitur history, maka akan terdapat auth context atau history context. Jumlah context dalam aplikasi akan menyesuaikan banyaknya fitur-fitur yang tersedia dalam website.

Untuk implementasinya dalam website Mangaice, state dibuat menggunakan modular. Oleh karena itu terdapat banyak context, menyesuaikan jumlah fitur yang dikembangkan. Context yang terdapat dalam website Mangaice adalah sebagai berikut:

TABEL VII
CONTEXT PADA WEBSITE MANGAICE

No	Context	Deskripsi
1.	authContext	Context untuk *authentication* dan *authorization*
2.	filterContext	Context untuk filter dalam pencarian manga lanjutan
3.	historyContext	Context untuk daftar history pengguna
4.	libraryContext	Context untuk daftar pustaka pribadi pengguna
5.	readingProgressContext	Context untuk progress bacaan chapter manga
6.	themeContext	Context untuk tema dari website
7.	utilityContext	Context untuk *utility function* seperti navbar

		click, sidebar click, dll.

b. Provider

Provider merupakan komponen yang menyediakan state atau data global ke semua komponen dalam aplikasi atau komponen yang berada di bawah komponen tree tersebut. Provider bertindak sebagai "penyedia" yang membungkus komponen aplikasi sehingga komponen tersebut dapat dengan langsung mengakses context melalui provider. Seperti yang sudah dijelaskan sebelumnya, context hanya mengembalikan object context, tidak menyimpan informasi penting. Sedangkan provider, digunakan untuk mendefinisikan semua *value* atau *state* dan fungsi yang ingin dijadikan *state* global.

Untuk membuat provider, sesuai dokumentasi dari React, dapat menggunakan someContext.Provider. Komponen provider ini merupakan komponen React yang harus digunakan sebagai *wrapper.* Sehingga provider, dalam hirarki *tree* akan berfungsi sebagai komponen *parent* dan hanya komponen *children* dibawahnya saja yang dapat mengakses *state* yang disediakan. Berikut adalah contoh penggunaan provider:

```
export const App = ({children}:{children:React.ReactNode) => {
  const [theme, setTheme] = useState('light);

  return {
    <ThemeContext.Provider>
{children}
    <ThemeContext.Provider>
  }
}
```

Dalam implementasinya pada website Mangeice, dibuatkan komponen provider untuk masing-masing context. Sehingga jumlah komponen provider sesuai dengan jumlah context yang ada, yakni 7 (Tabel VII). Provider menerima prop value, yang berisikan data/*state*, atau fungsi yang ingin dibagikan. *Value* ini biasanya berisikan *state* global atau fungsi pengubah *state*. Dari contoh di atas, custom komponen provider dapat dibuat untuk memudahkan dalam modularisasi dan penggunaan ulang komponen. Komponen ini akan dibuat menjadi komponen parent yang dapat menerima komponen apa saja menjadi komponen childrennya. Komponen yang memerlukan provider hanya tinggal impor komponen ini sedangkan yang tidak membutuhkan maka tidak perlu melakukan impor komponen provider. Semua custom hooks dan custom komponen provider dibuat untuk menghindari penggunaan kode ulang, sehingga tiap komponen tidak perlu memanggil useContext dan Context.Provider secara manual.

2. Implementasi Redux

Pada tahapan ini, peneliti melakukan implementasi Redux yang nantinya akan digunakan sebagai salah satu tools dalam mengatur global state dalam website selain Context API. Proses implementasi ini merupakan bagian dari proses pengembangan website Mangaice. Untuk menggunakan Redux, perlu dilakukan instalasi terlebih dahulu. instalasi Redux pada website Mangaice dilakukan dengan menjalankan perintah berikut:

```
npm install redux react-redux @redux/toolkit
```

Setelah melakukan penginstalan, dalam project dibuatkan folder "redux" yang digunakan untuk menyimpan semua file atau folder terkait dengan Redux. Isi dari folder ini antara lain folder reduxProvider, folder slice yang menyimpan reducer, dan file store yang bertindak sebagai tempat penyimpanan semua state global dalam aplikasi. Redux memerlukan empat komponen dasar, yakni store, action, reducer, provider, dan dalam implementasi Redux pada website Mangaice menggunakan Redux Toolkit (RTK). Menggunakan RTK merupakan pendekatan yang disarankan oleh pengembang Redux. RTK menyediakan fungsi/*tools* yang mempermudah implementasi logika Redux[7].

a. Store

Store merupakan komponen dalam Redux yang digunakan sebagai tempat penyimpanan semua *state* dalam aplikasi. Store merupakan komponen guna mengimplementasikan konsep *single source of truth*[7]. Semua global state pada aplikasi disimpan pada *object tree* di dalam store tunggal. Untuk membuat store Redux, buat file store.ts dan menggunakan API createStore dari Redux Toolkit untuk inisialisasi store. API createStore merupakan metode standar untuk membuat Redux store. API ini menggunakan low-level Redux core createStore secara internal[8].

API configureStore menawarkan API yang sudah di-improve dan *pattern* penggunaan dibandingkan dengan createStore dengan menerima parameter seperti reducer, middleware, preload state, enhancers, dev tools, dan *type inference* yang lebih baik untuk bahasa Typescript (Redux Toolkit, n.d). Dalam Mangaice, berikut adalah contoh store untuk Redux:

```
export const store = configureStore({
reducer: rootReducer,
});

export const useAppDispatch: () => AppDispatch = useDispatch;
export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
```

b. Action dan Reducer

Action dan reducer merupakan komponen Redux penting yang digunakan untuk memanipulasi nilai dari suatu *state.* Menurut dokumentasi resmi dari Redux, action merupakan objek Javascript yang memiliki *type field,* atau dapat disimpulkan bahwa action merupakan objek *event* yang mendeskripsikan sesuatu yang terjadi pada aplikasi. Sedangkan reducer merupakan fungsi yang

melakukan suatu proses sesuai dengan action yang diberikan. Dalam implementasi Redux menggunakan RTK, action dan reducer dapat dibuat sekaligus dalam satu fungsi/API menggunakan API createSlice. Berikut adalah contoh dari slice pada website Mangaice:

```
const authSlice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    initializeAuth: (state) => {},
    logout: (state) => {},
    updateUser: (state, action: PayloadAction<{ newUsername?: string;
        newPassword?: string }>
    ) => {},
  },
  extraReducers: (builder) => {},
});

export const { initializeAuth, logout, updateUser } = authSlice.actions;
export default authSlice.reducer;
```

Sama halnya dengan Context API, slice dibuat berdasarkan fitur-fitur yang ada di website Mangaice. Setiap fitur memiliki slice-nya masing-masing. Secara keseluruhan, terdapat 8 slice pada website Mangaice sebagai berikut:

TABEL VIII
SLICE PADA WEBSITE MANGAICE

No	Slice	Deskripsi
1.	authSlice	Slice untuk *authentication* dan *authorization*
2.	filterSlice	Slice untuk filter dalam pencarian manga lanjutan
3.	historySlice	Slice untuk daftar history pengguna
4.	librarySlice	Slice untuk daftar pustaka pribadi pengguna
5.	readingProgressSlice	Slice untuk progress bacaan chapter manga
6.	themeSlice	Slice untuk tema dari website
7.	utilitySlice	Slice untuk *utility function* seperti navbar click, sidebar click, dll.

c. Provider

Sama seperti Context API, Redux menggunakan komponen provider untuk menyediakan *state* ke semua komponen. Bedanya adalah, provider dalam Redux bukan merupakan tempat yang mendefinisikan value dari state atau action function. State dan action didefinisikan pada slice, sedangkan provider hanya digunakan untuk menyediakan reducer, action, dan state yang sudah didefinisikan melalui store. Provider dibuat dengan menggunakan API provider dari library react-redux, dan digunakan sebagai komponen *wrapper* yang membungkus komponen *children* dibawahnya. Provider memiliki satu parameter yakni store. Parameter ini akan diberikan nilai berupa Redux store yang sudah diinisialisasi sebelumnya. Dalam Mangaice, contoh komponen Redux Provider adalah sebagai berikut:

```
import React from 'react'
import { Provider } from 'react-redux'
import { store } from '../store'

interface Props {
  children: React.ReactNode
}

const ReduxProvider = ({children}:Props) => {
  return (
    <Provider store={store}>
      {children}
    </Provider>
  )
}

export default ReduxProvider
```

C. *Pengujian*

Tahap ini merupakan tahapan pengujian terhadap Context API dan Redux mengenai parameter performa, skalabilitas, dan memori, untuk membandingan Context API dan Redux. Hasil dai pengujian, akan dibandingkan di analisis untuk mengetahui mana yang unggul dari semua parameter perbandingan tersebut dan mana *state management tools* yang cocok digunakan untuk website dengan tingkat kompleksitas seperti website Mangaice.

1. Performa

Pengujian performa merupakan pengujian yang dilakukan untuk mengukur efisiensi dan kecepatan antara Context API dan Redux dalam berbagai kondisi operasional. Parameter yang diukur pada pengujian ini sudah dijelaskan pada Tabel III dan adapun *tools* yang digunakan dalam pengukuran adalah Web Page Test dan Google Web Dev Console. Berikut adalah hasil pengukuran performa dari Context API dan Redux:

TABEL IX
HASIL PENGUJIAN PERFORMA CONTEXT API DAN REDUX

No	Parameter	Context API		Redux	
1	Loading	10 ms		9 ms	
2	Scripting	6.823 ms		7.385 ms	
3	Rendering	8.017 ms		6.380 ms	
4	Painting	1.947 ms		2.273 ms	
5	System	6.002 ms		7.373 ms	
		First View	Repeat View	First View	Repeat View
6	Time to First	0,294 s	0,173 s	0,214 s	0,184 s

	Byte (TTFB)				
7	Start Render	0,700 s	0,300 s	0,500 s	0,300 s
8	First Contentful Paint (FCP)	0,727 s	0,433 s	0,516 s	0,324 s
9	Speed Index	7,477 s	8,607 s	8,155 s	13,627 s
10	Largest Contentful Paint (LCP)	10,069 s	29,733 s	13,845 s	0,936 s
11	Cumulative Layout Shift (CLS)	0	0	0	0
12	Total Blocking Time (TBT)	0 s	0,060 s	0,013 s	0,005 s
13	Page weight	17.241 kb	15.690 kb	15.767 kb	16.101 kb

2. Skalabilitas

Pengujian skalabilitas merupakan pengujian yang dilakukan untuk mengetahui apakah Context API dan Redux dapat menangani peningkatan jumlah data dan jumlah pengguna yang menggunakan website. Parameter yang diukur pada pengujian ini sudah dijelaskan pada Tabel IV dan adapun *tools* yang digunakan dalam pengukuran adalah JMeter. Terdapat keterbatasan dimana penelitian hanya terbatas pada penggunaan JMeter dengan mensimulasikan 5 pengguna virtual saja. Hasil dari pengujian skalabilitas Context API dan Redux adalah sebagai berikut:

TABEL X
HASIL PENGUJIAN SKALABILITAS CONTEXT API DAN REDUX

No	Parameter	Context API	Redux
1	Median respons Time	332 ms	269 ms
2	Throughput	38,0 Mbps	40,5 Mbps
3	Success Rate	100%	99,98%
4	99th %tile respons	5.237 ms	953 ms
5	Request	14 req/s	16 req/s
6	Median Script Execution	31.708 ms	19.303 ms
7	99th %tile Script Execution	34.227 ms	27.171 ms

3. Memori

Pengujian memori digunakan untuk mengetahui berapa banyak penggunaan memori pada masing-masing website yang mengimplementasikan Context API dan Redux. Pengujian ini menggunakan Chrome Dev Tools yang memiliki tools bawaan pada tab memory yang memungkinkan pencatatan parameter penggunaan memori pada saat website dijalankan (Tabel V). Pengukuran akan dilakukan dua kali, yang pertama adalah pengukuran saat website baru pertama kali diakses dan yang kedua adalah pengukuran yang dilakukan setelah website digunakan oleh pengguna dengan mengikuti skenario Tabel II. Berikut hasil dari pengujian memori:

TABEL XI
HASIL PENGUJIAN MEMORI CONTEXT API

Kategori	Memori Awal (Kb)	Memori Pasca Interaksi (Kb)	Selisih (Kb)	Persentase Perubahan
Code	8.200	9.907	+1.707	+20.82%
Strings	1.694	2.648	+954	+56,32%
JS Arrays	613	805	+192	+31,32%
Typed Arrays	2.866	6.636	+3.770	+131,55%
System Objects	660	1.057	+397	+60,15%
Total	22.611	30.902	+8.291	+36,67%

TABEL XII
HASIL PENGUJIAN MEMORI REDUX

Kategori	Memori Awal (Kb)	Memori Pasca Interaksi (Kb)	Selisih (Kb)	Persentase Perubahan
Code	8.674	4.747	-3.927	-45,28%
Strings	1.641	1.559	-82	-4,99%
JS Arrays	602	372	-230	-38,21%
Typed Arrays	2.491	2.595	+104	+4,18%
System Objects	386	814	+428	+110,88%
Total	21.824	15.152	-6.672	-30,57%

D. *Perbandingan*

Tahapan berikut adalah tahapan membandingan Context API dan Redux berdasarkan hasil dari pengujian ang sudah dilakukan dari tahapan sebelumnya. Berikut adalah hasil perbandingan Context API dan Redux:

1. Performa

Berdasarkan Tabel IX, dapat dibuatkan hasil pengukuran persentae perbedaan parameter pengujian performa antara Context API dan Redux sebagai berikut:

TABEL XIII
PERSENTASE PERBEDAAN PERFORMA CONTEXT API DAN REDUX

No	Parameter	Persentase
1.	Loading	Redux unggul 10%
2.	Scripting	Context API unggul 8,23%
3.	Rendering	Redux unggul 20,42%
4.	Painting	Context API unggul 16,75%
5.	System	Context API unggul

		22,85%
6.	Time to First Byte (TTFB)	Redux unggul 10,42%
7.	Start Render	Redux unggul 28,57%
8.	First Contentful Paint (FCP)	Redux unggul 27,10%
9.	Speed Index	Context API unggul 33,67%
10.	Largest Contentful Paint (LCP)	Redux unggul 29,77%
11.	Cumulative Layout Shift (CLS)	Tidak ada perbedaan (0%)
12.	Total Blocking Time (TBT)	Redux unggul 91,67%

Analisis:

Dari hasil pengukuran yang dilakukan menggunakan Google Web Console dan WebPageTest (Tabel IX), untuk implementasi Context API dan Redux pada sisi performa, didapatkan hasil dimana dari segi loading time, Redux sedikit lebih cepat dalam initial loading namun perbedaan tersebut minimal dan tidak terlalu signifikan. Context API memproses scripting lebih cepat dan efisien daripada Redux yang mungkin menunjukan bahwa Context API memiliki overhead yang lebih rendah dalam memproses fungsi-fungsi state management. Overhead merupakan biaya tambahan (dalam bentuk waktu, memori, atau resource lain) yang dilakukan untuk menjalankan fungsi atau proses diluar tugas utama. Untuk kasus Context API dan Redux, overhead mengacu pada resource yang dibutuhkan untuk pengelolaan state pada aplikasi. Context API biasanya memiliki overhead yang lebih rendah, hal ini dikarenakan Context API tidak memerlukan banyak konfigurasi dan dependensi tambahan. Dengan menggunakan Context API, pengaturan dan penggunaan state lebih sederhana sehingga lebih efisien dalam waktu eksekusi dan memori.

Disisi lain, Redux menambahkan lapisan abstraksi tambahan seperti actions, reducers, dan middleware yang berarti ada proses ekstra yang harus dikerjakan saat state diubah. Semua komponen yang menggunakan Redux membutuhkan overhead (resource tambahan) yang membuat Redux sedikit lebih lambat dan memakan banyak memori daripada Context API, terutama untuk aplikasi yang kecil atau tidak kompleks.

Dari segi rendering, Redux secara signifikan lebih cepat daripada Context API, mungkin dikarenakan Redux *unidirectional data flow* dan *immutable updates* memungkinkan untuk optimasi lebih baik. Context API menyelesaikan tahapan painting lebih cepat yang berarti elemen-elemen visual ditampilkan lebih cepat setelah rendering komponen selesai. Namun perbedaan ini relatif kecil dibandingkan dengan keunggulan Redux pada sisi rendering. Selain itu Context API lebih hemat dalam menggunakan resource sistem yang dapat membantu performa pada perangkat yang memiliki keterbatasan sumber daya. Redux memiliki waktu TFTB, FCP, dan LCP yang lebih baik terutama pada repeat view, artinya konten halaman muncul lebih cepat dan lebih responsif. Redux lebih efisien dalam proses rendering sehingga halaman dapat terlihat lebih cepat oleh pengguna setelah scripting. Redux memiliki TBT yang lebih rendah pada semua view, menunjukan bahwa Redux tidak terlalu memblokir thread utama, memberikan pengalaman yang lebih lancar pada pengguna.

Berdasarkan Tabel XIII, Context API memiliki keunggulan antara lain adalah Context API memiliki *system overhead* yang lebih rendah dan *speed index* yang lebih rendah daripada Redux. Selain itu, Context API juga melakukan *painting* lebih cepat daripada Redux. Disisi lain Redux lebih unggul dari Context API sekitar 11,16% dalam hal performa, dihitung dari rata-rata persentase parameter performa Context API dan Redux. Hal tersebut ditunjukan dari keunggulan Redux dalam rendering, TTFB, FCP, dan page weight. Secara keseluruhan, Redux memberikan pengalaman kecepatan yang lebih baik dalam menampilkan konten utama dan rendering terutama pada repeat view. Jika fokus utama adalah membuat konten utama terlihat lebih cepat, maka Redux lebih unggul pada aspek kecepatan. Sedangkan untuk efisiensi dan kestabilan, Context API lebih unggul dalam efisiensi sistem, scripting, dan painting. Jika prioritas adalah efisiensi penggunaan resource dan kestabilan, maka Context API lebih baik.

2. Skalabilitas

Berdasarkan Tabel X, dapat dibuatkan hasil pengukuran persentae perbedaan parameter pengujian skalabilitas antara Context API dan Redux sebagai berikut:

TABEL XIV
PERSENTASE PERBEDAAN SKALABILITAS CONTEXT API DAN REDUX

No	Parameter	Persentase Perbedaan
1.	Median respons Time	Redux unggul 18,98%
2.	Throughput	Redux unggul 6,68%
3.	Success Rate	Context API unggul 0,02%
4.	99th %tile respons	Redux unggul 81,80%
5.	Request	Redux unggul 14,29%
6.	Median Script Execution	Redux unggul 39,14%
7.	99th %tile Script Execution	Redux unggul 20,61%

Analisis:

Berdasarkan hasil pengujian yang dilakukan menggunakan JMeter terhadap Context API dan Redux (Tabel X), Redux menunjukan waktu respons yang lebih cepat dengan rata-rata 269 ms dibandingkan dengan Context API dengan rata-rata 332 ms yang menunjukan bahwa Redux memiliki *response time* yang lebih konsisten dan optimal. Dalam skenario beban ringan (5 pengguna) Redux lebih responsif dalam menangani permintaan dikarenakan mekanisme *subscription* selektif (useSelector) yang menghindari *re-render* komponen yang tidak perlu. Redux lebih konsisten dan stabil bahkan dalam permintaan yang

berat, diindikasikan dengan waktu *response* pada persentil ke-99 yang lebih baik yaitu 953 ms. Waktu ini lebih cepat dibandingkan dengan Context API yang mendapatkan waktu 5.237 ms. Redux mampu menangani respon dan menjaga performa bahkan pada *peak load condition.*

Dalam segi throughput, Redux memiliki throughput yang lebih tinggi dengan nilai 40,5 Mbps dibandingkan dengan Context API yang memperoleh nilai throughput sebesar 38,0 Mbps. Redux mampu menangani lebih banyak data per detik, cocok untuk website dengan lalu lintas data yang tinggi (e.g., Mangaice yang menangani banyak pengguna). Selain itu, Redux juga memiliki jumlah request per detik yang lebih banyak yakni 16 request dibandingkan Context API yang hanya 14 request. Oleh karena itu, Redux mampu menangani aliran data yang lebih besar dan lebih banyak permintaan secara simultan daripada Context API, selain itu juga menunjukan bahwa Redux mampu untuk men-*handle* pengguna yang lebih banyak. Redux menunjukkan stabilitas yang lebih baik dalam waktu eksekusi script dengan memiliki waktu eksekusi yang lebih rendah secara keseluruhan. Dari segi *median script execution*, Redux lebih cepat untuk meningkatkan efisiensi *javascript execution* dan pada 99th persen *script execution* Redux dapat menjaga performa *script execution* yang baik.

Redux menunjukan kemampuan performa yang superior dibandingkan dengan Context API yang berkaitan dengan *load handling*. Hal ini ditunjukan dari nilai *throughput* yang lebih tinggi (40,5 Mbps vs 38,0 Mbps), dapat menerima *request* yang lebih banyak dibandingkan Context API (16 vs 14 *request*) dan waktu respon yang baik dibawah *load*. Perbedaan 99th persen *response time* yang signifikan menunjukan Redux lebih stabil dalam kondisi *high load*. Context API mengalami degradasi performa pada saat *peak usage*. Redux juga memiliki kemampuan eksekusi *script* yang superior, mengindikasikan bahwa Redux lebih optimal dalam men-*update state*, memiliki operasi *batching* yang lebih baik dan memiliki *middleware pipeline* yang lebih efisien. *Throughout* Redux yang cepat digabung dengan waktu eksekusi yang cepat menunjukkan penggunaan sumber daya yang efisien, memungkinkan untuk *serve* lebih banyak *user* dengan infrastruktur yang sama.

Dari hasil Tabel XIV, secara keseluruhan Redux memiliki rata-rata kemampuan skalabilitas yang baik sekitar 25,91% dibandingkan dengan Context API. Redux menunjukan kinerja yang lebih baik dalam hal waktu *response*, *throughput*, dan stabilitas dalam skenario dengan beban berat. Dengan hal tersebut, mengindikasikan bahwa Redux lebih mampu mengelola dan menjaga performa website ketika menangani banyak pengguna secara bersamaan. Sedangkan Context API hanya unggul di *success rate*, tetapi mungkin karena tidak adanya kompleksitas *middleware* seperti pada Redux.

3. Memori

Berdasarkan Tabel XI dan Tabel XII, dapat dibuatkan hasil pengukuran persentae perbedaan parameter pengujian memori antara Context API dan Redux sebagai berikut:

TABEL XV
PERSENTASE PERBEDAAN MEMORI CONTEXT API DAN REDUX SAAT WEBSITE PERTAMA KALI DIAKSES

Kategori	Context API (Kb)	Redux (Kb)	Selisih (Kb)	Persentase Perubahan
Code	8.200	8.674	+474	+5,8%
Strings	1.694	1.641	-53	-3,1%
JS Arrays	613	602	-11	-1,8%
Typed Arrays	2.866	2.491	-375	-13,1%
System Objects	660	386	-247	-41,5%
Total	22.611	21.824	-787	-3,5%

TABEL XVI
PERSENTASE PERBEDAAN MEMORI CONTEXT API DAN REDUX PASCA INTERAKSI

Kategori	Context API (Kb)	Redux (Kb)	Selisih (Kb)	Persentase Selisih
Code	9.907	4.747	-5.160	-52,1%
Strings	2.648	1.559	-1.089	-41,1%
JS Arrays	805	372	-433	-53,8%
Typed Arrays	6.636	2.595	-4.041	-60,9%
System Objects	1.057	814	-243	-23,0%
Total	30.902	15.152	-15.750	-51,0%

Analisis:

Dari Tabel XV tersebut dapat diperoleh bahwa Redux memiliki total penggunaan memori yang lebih rendah (21.824 Kb) yakni 3,5% lebih hemat atau dengan perbedaaan 787 Kb dibandingkan dengan Context API (22.611 Kb). Untuk code, Context API menggunakan memori yang sedikit lebih rendah daripada Redux, yakni 8.200 Kb dibandingkan dengan 8.674 Kb. Namun, selisihnya tidak terlalu signifikan, yakni hanya 5,8%. Hal ini karena Redux memerlukan library tambahan seperti redux, react-redux, dan middleware yang dapat meningkatkan ukuran *bundle* Javascript. Dari segi *bundle size*, Context API dapat menjadi pilihan jika memprioritaskan ukuran *bundle* kecil. Hal serupa juga terjadi pada JS Array, Redux lebih hemat dalam penggunaan memori terkait dengan JS Array dibandingkan dengan Context API walaupun hanya berbeda 11 Kb atau 1,8%. Redux menggunakan normalisasi state (misal dengan createEntityAdapter di Redux Toolkit) untuk menghindari duplikasi data pada array. Context API mungkin menyimpan data mentah dalam array tanpa optimasi, terutama jika state tidak dipecah menjadi beberapa context.
Redux dalam penggunaan memori string lebih hemat 3.1% dibandingkan Context API. Redux menggunakan struktur actions dan tipe yang konsisten sehingga mengurangi duplikasi string, Context API mungkin menyimpan lebih banyak string dinamis (misal string error atau key context). Redux lebih efisien dalam penggunaan memori untuk Typed

Array dan System Objects (lebih hemat 13,1% dan lebih hemat 41,5%). Hal ini menunjukan bahwa Redux memiliki struktur yang lebih optimal untuk data array dan objek sistem. Redux mungkin melakukan optimasi penyimpanan data (seperti caching untuk gambar cover manga) melalui middleware seperti RTK Query. Dari segi *system object,* Context API memuat lebih banyak object internal karena setiap Context.Provider dan komponen yang menggunakan useContext memicu subscription terpisah, sedangkan Redux menggunakan satu store terpisah dengan mekanisme selektif (useSelector), mengurangi jumlah object internal. Secara keseluruhan, Redux lebih efisien dari segi memori dengan penggunaan yang lebih rendah dibandingkan dengan Context API.

Dari Tabel XVI dapat diperoleh bahwa setelah interaksi, Redux menunjukan penggunaan memori yang lebih rendah dibandingkan dengan Context API. Redux dapat menghemat memori sebesar 51,0% lebih rendah dari Context API atau 15.750 Kb. Untuk code, Redux dapat mengurangi penggunaan memori sebesar 52,1% dari Context API. Hal ini dapat menunjukan bahwa manajemen state dengan Redux lebih optimal dalam penggunaan memori saat aplikasi berinteraksi dengan pengguna dan berjalan dengan dinamis.

Redux juga dapat menangani manipulasi data array dengan lebih hemat memori dibandingkan dengan Context API. Redux secara signifikan lebih efisien dalam mengelola Typed Arrays (hemat 60,9%) dan JS Arrays (hemat 53,8%) dibandingkan dengan Context API. Meskipun penggunaan memori untuk System Object meningkat pasca interaksi, Redux masih lebih hemat 23,0% dibandingkan dengan Context API.

Redux secara keseluruhan lebih efisien dari segi penggunaan memori baik pada saat website pertama kali diakses maupun setelah adanya interaksi dengan pengguna. Saat pertama kali diakses, Redux menunjukan penggunaan memori yang lebih rendah daripada Context API (3,5% lebih efisien). Setelah adanya interaksi dengan pengguna, Redux secara signifikan lebih efisien dengan penghematan total memori sebesar 51,0% dibandingkan dengan Context API. Peningkatan memori yang paling besar terlihat pada Context API untuk Typed Arrays dan Code yang mengindikasikan bahwa Context API mengalami pembengkakan memori saat data tambahan ditransfer dan diakses di berbagai komponen. Peningkatan memori yang besar ini mungkin disebabkan karena pembaruan data pada banyak komponen yang saling bergantung, sehingga mengakibatkan lebih banyak penggunaan Type Arrays dan Code.

Redux tampaknya lebih efisien dalam penggunaan memori baik pada saat akses awal maupun setelah terjadi interaksi dengan total memori lebih rendah secara signifikan daripada Context API. Redux menggunakan store terpusat yang memungkinkan data dikelola secara konsisten, mengurangi overhead dari pembuatan objek atau referensi secara berlebihan. Sedangkan Context API memiliki keterbatasan untuk skenario yang membutuhkan data global di banyak komponen, yang dapat mengakibatkan duplikasi referensi atau tambahan *re-rendering* pada elemen yang tidak optimal.

Jika memori merupakan salah satu faktor krisis, terutama untuk aplikasi dengan tingkat interaksi pengguna yang tinggi, Redux lebih disarankan karena kemampuannya yang lebih baik dalam mengoptimalkan konsumsi memori. Selain itu, Redux juga lebih efisien dalam mengelola struktur data yang lebih kompleks selama interaksi aplikasi dengan pengguna.

IV. KESIMPULAN

Berdasarkan hasil penelitian dan pembahasan mengenai penelitian perbandingan Context API dan Redux,maka dapat diambil kesimpulan bahwa penelitian ini berhasil mengimplementasikan Context API dan Redux, website dapat ber alan dengan baik,dan masing-masing *state management tools* dapat memanajemen state dengan baik. Secara performa, Redux unggul dalam performa *rendering* dan *responsiveness* secara keseluruhan, hal ini dibuktikan dengan unggulnya Redux dalam hal *rendering* (20,42% lebih cepat), TTFB (27,21% lebih cepat), start render (28,57% lebih cepat), FCP (29,02% lebih cepat), dan page weight (8,56% lebih ringan). Redux lebih cocok digunakan di website Mangaice. Redux memiliki kecepatan menampilkan konten lebih baik yang dapat meningkatkan pengalaman pengguna pada aplikasi interaktif dan sering diakses.

Redux memiliki struktur pengelolaan state yang lebih terorganisir. Hal ini akan membantu mempertahankan performa yang stabil seiring dengan bertambahnya data atau fitur pada Mangaice. Secara keseluruhan, Redux unggul 11,16% dibandingkan Context API dalam hal performa. Dari hasil pengujian skalabilitas website, Redux menjadi pilihan yang baik dalam implementasi pada website Mangaice untuk memastikan performa website yang baik di bawah beban pengguna yang tinggi atau ketika skalabilitas website menjadi pertimbangan. Dari pengukuran skalabilitas dengan parameter yang sudah disebutkan, Redux unggul sekitar 25,91% dari Context API. Terakhir, Redux unggul dalam penggunaan memori yang menunjukan adanya pengoptimalisasi pada saat website berinteraksi dengan pengguna. Redux 51,0% lebih efisien untuk total penggunaan memori dibandingkan dengan Context API.

REFERENSI

[1] A I Dzhangarov, K K Pakhaev, & N V Potapova. (2021). Modern Web Application Development Technologies. IOP Conf. Series: Materials Science and Engineering. DOI: 10.1088/1757-899X/1155/1/012100

[2] Lazuardy, Mohammad Fariz Syah & Anggraini, Dyah. (2022). Modern Front End Web Architectures with React.Js and Next.Js. International Research Journal of Advanced Engineering and Science, 7(1). pp. 132-141, 2022.

[3] Vercel. (n,d.). Next.js Official Documentation. Vercel. https://nextjs.org/docs. Diakses pada 12 Oktober 2024.

[4] Dewanto, Rafli. (2023). Model State Management di Ekosistem React. Medium. https://medium.com/@dewantorafli/model-state-management-di-ekosistem-react-10e9f80bc994. Diakses pada 10 November 2024.

[5] Le, T. (2021). Comparison Of State Management Solutions Between Context API And Redux Hook In ReactJS. Metropolia University of Applied Sciences.
[6] Sanja Brekalo, Klaudio Pap, & Florijan Kos. (2025). Characteristic and Comparison of Global State Management Tools in React Applications. Journal For Printing Science and Graphic Communication.
[7] Redux. (n,d.). Redux Official Documentation. Diakses pada 12 Oktober 2024, dari https://redux.js.org/introduction/getting-started.
[8] Redux Toolkit. (n,d). Redux Toolkit Official Documentation. Diakses pada 12 Oktober 2024, dari https://redux-toolkit.js.org/introduction/getting-started