

# Dampak Pemilihan Pustaka State Management Terhadap Efisiensi dan Kompleksitas Kode pada Aplikasi *React*. Studi Kasus Redux, Zustand, dan Signals

**Naila Nurul Faizah\***, Egin Sefiano Widodo, Najwa Chava Safiera, Muhammad Amrullah W, Delia Indrawati, Saifudin Yahya.  
Program Studi Teknik Informatika, Universitas Negeri Surabaya, Indonesia  
\*Email: faizahnaila421@gmail.com

| Info Artikel                                                                                                                                                                                                                                                                                                                | Abstrak                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Kata Kunci :</b><br/><i>React</i>, manajemen state; performa; redux; zustand; signals.</p> <p><b>Keywords :</b><br/><i>React</i>, state management; performance; redux; zustand; signals.</p> <p><b>Tanggal Artikel</b><br/>Dikirim : 27 November 2025<br/>Direvisi : 12 Juni 2026<br/>Diterima : 15 Oktober 2026</p> | <p>Pengembangan aplikasi <i>web</i> modern menggunakan <i>React</i> sering menghadapi tantangan dalam mengelola <i>state</i> aplikasi secara efisien, terutama seiring dengan meningkatnya kompleksitas. Pemilihan pustaka <i>state management</i> menjadi keputusan arsitektural yang penting karena berdampak langsung pada performa aplikasi dan pengalaman pengguna. Penelitian ini bertujuan untuk menganalisis dan membandingkan secara mendalam dampak performa dari tiga pustaka <i>state management</i> populer dalam ekosistem <i>React</i>, yaitu Redux, Zustand, dan Signals. Metode penelitian yang digunakan adalah eksperimen kuantitatif dengan membangun dua skenario aplikasi: aplikasi sederhana untuk mengukur <i>overhead</i> dasar dan aplikasi CRUD kompleks untuk mensimulasikan beban kerja dunia nyata. Evaluasi dilakukan berdasarkan empat metrik teknis, yaitu penggunaan memori (MB), penggunaan CPU (%), waktu <i>rendering</i> (ms), dan ukuran <i>bundle</i> (KB). Hasil pengujian menunjukkan bahwa Signals secara konsisten memberikan performa terbaik di semua metrik. Pada skenario CRUD, Signals mencatat penggunaan memori terendah (45,8 MB) dan waktu <i>rendering</i> tercepat (28,9 ms). Zustand menempati posisi kedua sebagai pustaka modern yang ringan dan efisien, sedangkan Redux menunjukkan beban performa tertinggi (65,2 MB memori; 45,3 ms <i>rendering</i>). Kesimpulannya, pemilihan pustaka <i>state management</i> memiliki dampak langsung dan terukur terhadap performa aplikasi <i>React</i>. Pustaka modern seperti Signals dan Zustand menawarkan alternatif yang lebih efisien dibandingkan Redux, terutama untuk aplikasi yang menekankan kecepatan, skalabilitas, dan efisiensi sumber daya.</p> <p><b>Abstract</b></p> <p><i>The development of modern web applications using React often faces challenges in efficiently managing application state, especially as complexity increases. Choosing the right state management library is an essential architectural decision because it directly affects application performance and user experience. This study aims to analyze and compare the performance impacts of three popular state management libraries in the React ecosystem: Redux, Zustand, and Signals. The research employs a quantitative experimental method by developing two application scenarios: a simple app to measure baseline overhead and a complex CRUD app to simulate real-world workloads. Evaluation was conducted based on four technical metrics: memory usage (MB), CPU usage (%), rendering time (ms), and bundle size (KB). The results show that Signals consistently delivers the best performance across all metrics. In the CRUD scenario, Signals recorded the lowest memory usage (45.8 MB) and fastest rendering time (28.9 ms). Zustand ranked second as a lightweight and efficient modern library, while Redux demonstrated the highest performance overhead (65.2 MB memory; 45.3 ms rendering). In conclusion, the choice of state management library has a direct and measurable impact on React application performance. Modern libraries such as Signals and Zustand provide more efficient alternatives to</i></p> |

---

*Redux, especially for applications that emphasize speed, scalability, and resource efficiency.*

---

## 1. PENDAHULUAN

Dalam ekosistem pengembangan aplikasi *front end* modern berbasis *React*, manajemen *state* merupakan aspek krusial yang secara langsung memengaruhi performa, skalabilitas, dan kemudahan pemeliharaan aplikasi [1]. Seiring dengan bertambahnya kompleksitas fitur, manajemen *state* yang efektif sangat dibutuhkan untuk menjamin konsistensi data di seluruh komponen, sekaligus meminimalkan render ulang antarmuka pengguna (UI) yang tidak perlu agar performa tetap optimal [1][2]. Tantangan utama bagi pengembang adalah menyeimbangkan antara struktur kode yang rapi dengan efisiensi eksekusi di sisi klien.

Selama bertahun-tahun, Redux telah menjadi standar industri dalam menjawab tantangan tersebut. Pustaka ini menawarkan *centralized store* dan aliran data searah (*unidirectional data flow*) yang terstruktur dan mudah di-debug [3][4]. Namun, dominasi Redux tidak lepas dari kritik. Redux sering dikritik karena kompleksitas dan kuantitas *boilerplate code* (seperti *actions* dan *reducers*) yang tinggi, yang sering dianggap berlebihan untuk proyek skala kecil hingga menengah [5]. Hal ini diperkuat oleh studi kasus terbaru yang menunjukkan adanya pola migrasi dari Redux menuju solusi lain yang dinilai lebih sederhana dan efisien, khususnya pada proyek dengan kebutuhan manajemen *state* yang tidak pelik[5].

Menjawab tantangan terkait kompleksitas tersebut, pustaka yang lebih modern seperti Zustand hadir sebagai alternatif. Zustand memanfaatkan *React Hooks* untuk menyediakan API yang minimalis dan ringan, secara drastis mengurangi *boilerplate* [4][6]. Keunggulannya terletak pada *selective re-rendering* yang efisien dan ukuran *bundle* yang jauh lebih kecil dibandingkan Redux, sehingga menjanjikan performa yang lebih baik[2]. Meski menawarkan kemudahan, beberapa studi juga menekankan risiko *maintainability* dan kurangnya opini arsitektur yang ketat pada penggunaan Zustand di proyek jangka panjang, yang berpotensi menimbulkan inkonsistensi kode jika tidak dikelola dengan baik.

Selain pendekatan berbasis *hooks*, pendekatan yang lebih baru hadir melalui paradigma Signals [7]. Diadopsi dari kerangka kerja seperti SolidJS dan kini mulai diterapkan di ekosistem *React* [1][8][9], Signals mengimplementasikan *Reactive programming* secara murni [7]. Alih-alih bergantung pada mekanisme *Virtual DOM diffing* yang umum di *React*, Signals menciptakan koneksi langsung antara *state* dan komponen UI [8][9]. Mekanisme ini memungkinkan pembaruan yang sangat granular, di mana hanya komponen yang bergantung pada *state* yang berubah yang akan diperbarui [1]. Paradigma ini menjanjikan efisiensi sumber daya dan responsivitas aplikasi yang superior. *Benchmark* terbaru yang membandingkan Signals dengan pustaka lain menunjukkan keunggulan Signals dalam konsumsi memori, CPU, waktu *rendering* dan ukuran *bundle*, meskipun terdapat juga kritik atas *overhead* kognitif dan debugging yang lebih kompleks [9][10].

Saat ini, pengembang *React* dihadapkan pada tiga filosofi manajemen *state* yang fundamental berbeda: Redux yang terstruktur, Zustand yang minimalis, dan Signals yang reaktif granular. Beberapa penelitian terdahulu telah mengkaji pustaka-pustaka ini, baik secara arsitektural maupun performa. Sebagai contoh, studi yang dilakukan oleh Bachelor dkk. membandingkan performa antara Redux dan Zustand dalam aplikasi *React*, di mana temuan mereka menunjukkan bahwa pustaka modern yang lebih ringkas seperti Zustand mampu mereduksi kompleksitas *boilerplate* dibandingkan Redux [3]. Sejalan dengan hal tersebut, penelitian kuantitatif oleh N.S. dkk. juga mengevaluasi *developer experience* dan performa Redux serta Zustand, dengan menyoroti efisiensi pada pustaka yang berbasis *hooks*[4]. Sementara itu, dari sisi paradigma baru, riset oleh Balasubrahmanian menganalisis arsitektur berbasis *Signal-First* (seperti *Signals*) dan menyimpulkan bahwa mekanisme *Reactive programming* murni tersebut berpotensi besar dalam efisiensi performa pada sisi *front-end*[2]. Namun, berdasarkan tinjauan pada penelitian-penelitian sebelumnya, masih terdapat celah permasalahan (*research gap*). Mayoritas kajian ilmiah yang ada cenderung membandingkan performa Redux dan Zustand secara spesifik, atau membahas arsitektur Signals secara terisolasi tanpa komparasi performa langsung dengan pustaka pendahuluannya. Pemahaman mendalam dan komprehensif mengenai perbandingan Redux, Zustand, dan Signals secara simultan menggunakan metrik performa kuantitatif yang sama masih sangat terbatas [2][3][4].

Oleh karena itu, penelitian ini bertujuan melakukan analisis perbandingan kuantitatif secara langsung (*head-to-head*) untuk mengukur dampak performa dari Redux, Zustand, dan Signals dalam aplikasi *React*. Penelitian ini berfokus secara objektif pada empat metrik krusial: penggunaan memori, penggunaan CPU, waktu *rendering*, dan

ukuran *bundle*. Melalui analisis ini, diharapkan dapat mengisi celah literatur yang ada dan menyediakan data empiris yang valid untuk membantu pengembangan membuat keputusan teknis yang tepat dalam memilih pustaka *state management*.

## 2. METODE PENELITIAN

### 2.1 Tahapan Penelitian

Penelitian ini dirancang sebagai eksperimen kuantitatif yang bertujuan untuk mengukur dan membandingkan secara objektif dampak kinerja dari tiga Pustaka *state management* Redux, Zustand, dan Signals saat diterapkan pada aplikasi *React* [3].

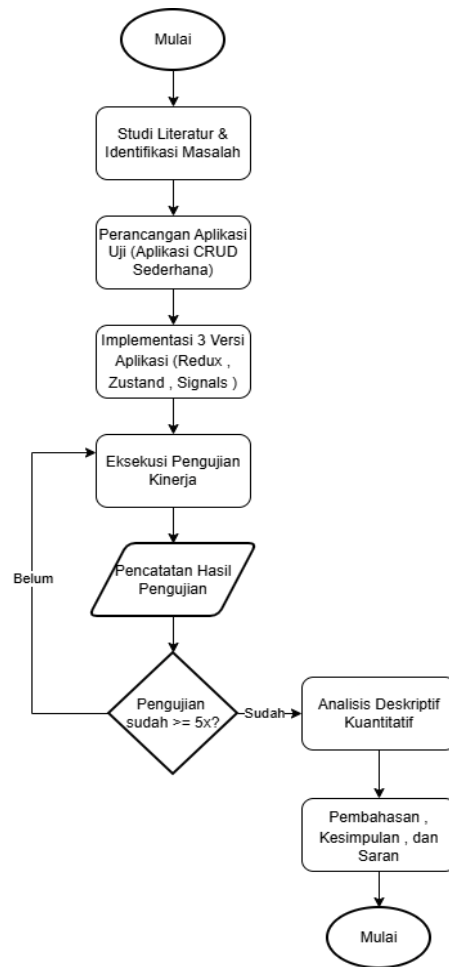
#### 2.1.1 Implementasi dan Pengujian

Tahapan penelitian ditunjukkan pada Gambar 1, dimulai dengan perancangan sebuah aplikasi *web CRUD* (*Create, Read, Update, Delete*) sederhana yang berfungsi sebagai objek penelitian [4]. Selanjutnya, dilakukan tahap implementasi di mana tiga versi identik dari aplikasi tersebut dibangun, dengan satu-satunya perbedaan adalah pustaka *state management* yang digunakan [6].

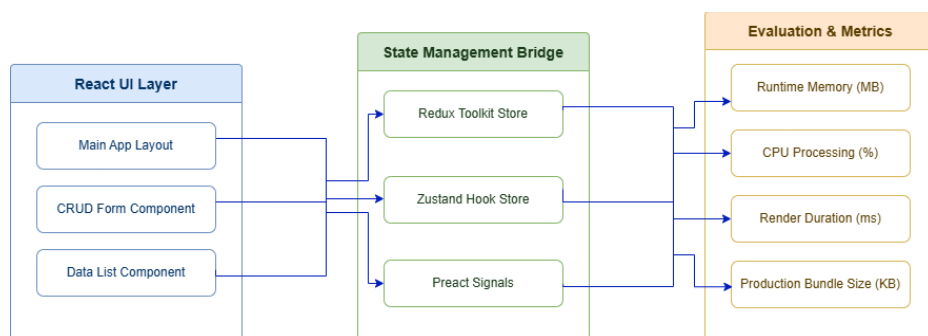
Setelah ketiga versi aplikasi siap, dilakukan tahap pengujian kinerja secara sistematis menggunakan instrument penelitian seperti *Chrome Developer Tools* dan *Lighthouse* untuk mengukur metrik-metrik kunci, termasuk penggunaan memori, penggunaan CPU, waktu *rendering*, dan ukuran *bundle* aplikasi. Untuk menjamin reliabilitas dan validitas hasil, setiap scenario pengujian diulang sebanyak lima kali. Dasar pemilihan metrik-metrik tersebut didasarkan pada karakteristik arsitektural *state management* dan dampaknya terhadap *user experience* (UX), dengan rincian sebagai berikut :

1. **Penggunaan Memori (*Memory Usage*):** Dipilih untuk mengevaluasi seberapa efisien struktur data dari masing-masing pustaka (*Redux, Zustand, Signals*) dalam mengalokasikan memori pada peramban. Penggunaan memori yang tinggi berpotensi menyebabkan *memory leak* yang dapat menurunkan stabilitas aplikasi pada perangkat dengan spesifikasi terbatas. Metrik ini sejalan dengan penelitian terdahulu yang mengukur efisiensi *runtime memory* pada arsitektur berbasis komponen [1][4].
2. **Penggunaan CPU (*CPU Usage*):** Dipilih untuk mengukur beban komputasi alur pembaruan data (*state dispatching/Reactivity*). Efisiensi pemrosesan CPU berpengaruh langsung pada kelancaran respons aplikasi, yang menjadi salah satu fokus utama dalam evaluasi performa *frontend* modern [4].
3. **Waktu Rendering (*Rendering Time*):** Dipilih karena *state management* bertugas mengatur kapan dan komponen mana saja dalam *React* yang harus digambar ulang (*re-render*). Pengukuran waktu *rendering* penting untuk mendeteksi adanya *unnecessary re-renders* yang dapat membuat antarmuka terasa lambat, sebagaimana dianalisis dalam studi perbandingan *state management* oleh Balasubramanian dan Bachelor [2][3].
4. **Ukuran Bundle Aplikasi (*Bundle Size*):** Dipilih untuk menilai dampak penambahan pustaka pihak ketiga terhadap ukuran akhir berkas JavaScript yang harus diunduh oleh pengguna. Ukuran *bundle* yang lebih kecil berkontribusi positif terhadap metrik *Initial Load Time* yang dioptimalkan oleh pustaka minimalis seperti Zustand dan Signals [1][5].

Data mentah yang terkumpul dari seluruh pengujian kemudian dicatat dan dianalisis menggunakan metode **analisis deskriptif kuantitatif**. Di mana nilai rata-rata dari setiap metrik dihitung dan dibandingkan untuk menarik kesimpulan mengenai pustaka mana yang paling efisien [3].



Gambar 1. Diagram alur penelitian



Gambar 2. Diagram Blok Perancangan Sistem

## 2.2 Spesifikasi Lingkungan Pengembangan dan Konfigurasi Sistem

Untuk memastikan bahwa hasil pengujian kinerja bersifat objektif, valid, dan dapat direplikasi (*reproducible*), seluruh lingkungan pengembangan diseragamkan dengan spesifikasi teknologi dan konfigurasi sebagai berikut:

### 1. Spesifikasi Versi Pustaka (*Library Dependencies*):

Aplikasi studi kasus dibangun menggunakan ekosistem JavaScript modern dengan versi pustaka spesifik, yaitu :

React: Versi 19.1.1

Node Js: Versi 22.20.0

Rincian versi pustaka spesifik yang digunakan dalam lingkungan pengujian dapat dilihat pada Tabel 1.

**Tabel 1. Spesifikasi Versi Pustaka Pengembangan Aplikasi**

| Nama Pustaka / Dependensi | Versi ( <i>Version</i> ) | Fungsi dalam Pengujian                                         |
|---------------------------|--------------------------|----------------------------------------------------------------|
| Redux                     | ^2.9.0<br>^9.2.0         | Manajemen <i>state</i> terpusat ( <i>Slice &amp; Actions</i> ) |
| Zustand                   | ^3.3.1                   | Manajemen <i>state</i> berbasis <i>Custom Hooks</i>            |
| Signals                   | ^5.0.8                   | Manajemen <i>state</i> berbasis <i>Reactive Primitive</i>      |

## 2. Strategi *Rendering*

Aplikasi ini menggunakan strategi *Client-Side Rendering* (CSR) penuh. Pemilihan CSR didasarkan pada tujuan pengujian yang berfokus pada efisiensi manajemen *state* di sisi peramban (*browser runtime performance*). Dengan CSR, seluruh proses pembaruan komponen (*re-rendering*), manipulasi DOM, dan alokasi memori murni dibebankan pada perangkat klien, sehingga dampak performa dari masing-masing pustaka *state management* dapat diukur secara terisolasi tanpa dipengaruhi oleh optimasi dari sisi server (*Server-Side Rendering*).

## 3. Konfigurasi Alat Pembuat (*Build Tools/Bundler*):

Proyek ini dikonfigurasi menggunakan Vite sebagai *build tool* dan *bundler* utama. Konfigurasi *bundler* dibiarkan pada kondisi standar (*production build mode*) menggunakan perintah vite build yang memanfaatkan Rollup untuk melakukan *minification* dan *tree-shaking* secara optimal. Hal ini krusial untuk memastikan pengukuran metrik ukuran *bundle* (*Bundle Size*) pada masing-masing pustaka merepresentasikan kondisi riil pengiriman berkas ke lingkungan produksi.

# 3. HASIL DAN PEMBAHASAN

Penelitian ini menggunakan pendekatan eksperimental kuantitatif untuk mengukur dan membandingkan dampak performa secara objektif. Objek penelitian adalah tiga pustaka *state management* populer di ekosistem *React*: Redux, Zustand, dan Signals. Untuk memastikan hasil yang komprehensif, pengujian dilakukan pada dua skenario aplikasi yang berbeda, yaitu aplikasi sederhana (untuk mengukur *overhead* dasar) dan aplikasi CRUD kompleks (untuk mensimulasikan beban kerja aplikasi dunia nyata yang dinamis). Instrumen penelitian yang digunakan meliputi perangkat keras berupa laptop dengan prosesor Intel Core i7 dan RAM 16 GB, serta perangkat lunak Google Chrome dengan *Developer Tools* untuk analisis *runtime* dan *Webpack Bundle Analyzer* untuk analisis ukuran *bundle*. Prosedur pengumpulan data dilakukan secara sistematis: pertama, membangun enam aplikasi prototipe (tiga pustaka x dua skenario) dengan fungsionalitas identik; kedua, menjalankan serangkaian interaksi pengguna yang telah ditentukan pada setiap aplikasi; ketiga, mencatat data performa selama pengujian berlangsung. Setiap pengujian diulang sebanyak tiga kali untuk memastikan konsistensi data, dan nilai rata-ratanya diambil sebagai hasil akhir. Metrik performa (variabel) yang diukur dalam penelitian ini adalah Pengguna Memori (MB), Pengguna CPU(%), Waktu *Rendering* (ms).

## 3.1. Penerapan *State Management Redux, Zustand dan Signal*

Untuk mengevaluasi dampak dari setiap pustaka *state management*, langkah pertama adalah mengimplementasikan logika bisnis yang sama pada aplikasi CRUD (*Create, Read, Update, Delete*) sederhana. Fungsionalitas inti ini menambah, menghapus, dan mengedit pengguna diterapkan menggunakan tiga pendekatan berbeda, yang perbandingan kode dan paradigmanya dianalisis sebagai berikut:

### 3.1.1. Redux (Menggunakan Redux Toolkit)

Implementasi Redux, yang ditenagai oleh Redux Toolkit, menunjukkan pendekatan yang paling terstruktur. Logika *state* didefinisikan menggunakan *createSlice*, yang secara rapi mengelompokkan *initial state*, *reducers*, dan *action creators* ke dalam satu file. Seperti yang terlihat pada kode, *reducers* (misalnya *addUser*, *deleteUser*) ditulis dengan sintaks "mutatif" yang dimungkinkan oleh Immer.js (yang terintegrasi dalam Toolkit), seperti

`state.users.push(newUser)`. Meskipun modernisasi dari Toolkit ini sangat mengurangi *boilerplate*, Redux tetap mempertahankan arsitektur yang jelas dengan memisahkan logika *state* (di dalam *slice*) dari pembarutannya (yang dipicu oleh *actions*).

```
import { createSlice } from '@reduxjs/toolkit';

const initialState = {
  users: [],
};

export const userSlice = createSlice({
  name: 'users',
  initialState,
  reducers: {
    addUser: (state, action) => {
      const newUser = { id: Date.now(), name: action.payload };
      state.users.push(newUser);
    },
    deleteUser: (state, action) => {
      state.users = state.users.filter(user => user.id !== action.payload);
    },
    editUser: (state, action) => {
      const { id, newName } = action.payload;
      const userToEdit = state.users.find(user => user.id === id);
      if (userToEdit) {
        userToEdit.name = newName;
      }
    },
  },
});

export const { addUser, deleteUser, editUser } = userSlice.actions;
export default userSlice.reducer;
```

### 3.1.2. Zustand

Zustand menghadirkan pendekatan yang jauh lebih minimalis dan berfokus pada *hooks*. Seluruh *store* dibuat menggunakan satu fungsi `create`. Berbeda dengan Redux, Zustand tidak memerlukan *actions* atau *reducers* yang terpisah. Sebaliknya, *state* (`users: []`) dan fungsi untuk memodifikasinya (`addUser`, `deleteUser`) didefinisikan bersama dalam satu objek. Pembaruan *state* dilakukan secara *immutable* melalui pemanggilan fungsi `set`, yang menerima *state* sebelumnya dan mengembalikan *state* baru. Pendekatan ini secara drastis mengurangi jumlah kode yang diperlukan dan menyederhanakan logika *state* menjadi satu *hook* kustom (`useUserStore`).

```
import { create } from 'zustand';

export const useUserStore = create((set) => ({
  users: [],
  addUser: (name) =>
    set((state) => ({
      users: [...state.users, { id: Date.now(), name }],
    })),
}));
```

```
    })),  
    deleteUser: (id) =>  
      set((state) => ({  
        users: state.users.filter((user) => user.id !== id),  
      })),  
    editUser: (id, newName) =>  
      set((state) => ({  
        users: state.users.map((user) =>  
          user.id === id ? { ...user, name: newName } : user  
        ),  
      })),  
  }));
```

### 3.1.3. Signals (Menggunakan @pReact/signals-React)

Signals memperkenalkan paradigma yang berbeda secara fundamental. Alih-alih mengelola *state* dalam sebuah *store* terpusat, Signals membungkus *state* individu dalam sebuah *primitive Reactive* (signal). Seperti yang ditunjukkan, *users* adalah sebuah *signal* yang nilai aktualnya diakses dan diperbarui melalui properti *.value*. Fungsi-fungsi seperti *addUser* dan *deleteUser* adalah fungsi JavaScript standar yang di-ekspor; fungsi-fungsi ini secara langsung memanipulasi nilai *signal* (*users.value = ...*) [11]. Ini adalah pendekatan reaktivitas yang paling granular, di mana pembaruan terjadi dengan langsung menetapkan nilai baru ke properti *.value*, yang kemudian secara otomatis memicu pembaruan pada komponen yang "mendengarkan" *signal* tersebut [11].

```
import { signal } from '@pReact/signals-React';  
export const users = signal([]);  
export const addUser = (name) => {  
  users.value = [...users.value, { id: Date.now(), name }];  
};  
export const deleteUser = (id) => {  
  users.value = users.value.filter((user) => user.id !== id);  
};  
export const editUser = (id, newName) => {  
  users.value = users.value.map((user) =>  
    user.id === id ? { ...user, name: newName } : user  
  );  
};
```

## 3.2 Pengujian Performa

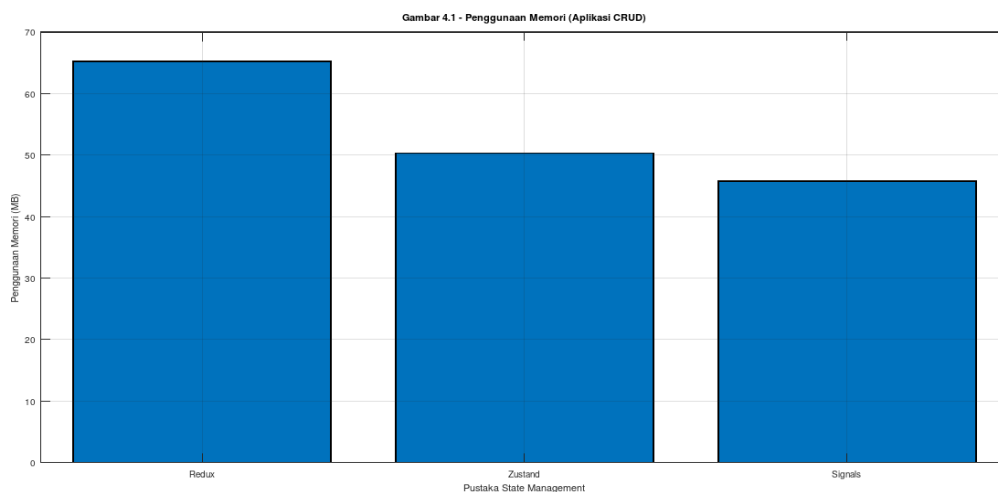
Pengujian dilakukan dengan menyimulasikan operasi pada kedua skenario aplikasi (sederhana dan CRUD). Data performa kuantitatif yang terkumpul dari hasil pengujian dirangkum secara lengkap pada Tabel 2.

**Tabel 2. Perbandingan performa runtime pada skenario aplikasi CRUD**

| Pustaka | Memori Awal (MB)<br>± SD | Scripting (ms)  | Waktu <i>Rendering</i><br>(ms) | Ukuran <i>Bundle</i><br>(KB) |
|---------|--------------------------|-----------------|--------------------------------|------------------------------|
| Redux   | 28,26 ± 10,89            | 1093,60 ± 85,98 | 254,40 ± 21,03                 | 18                           |

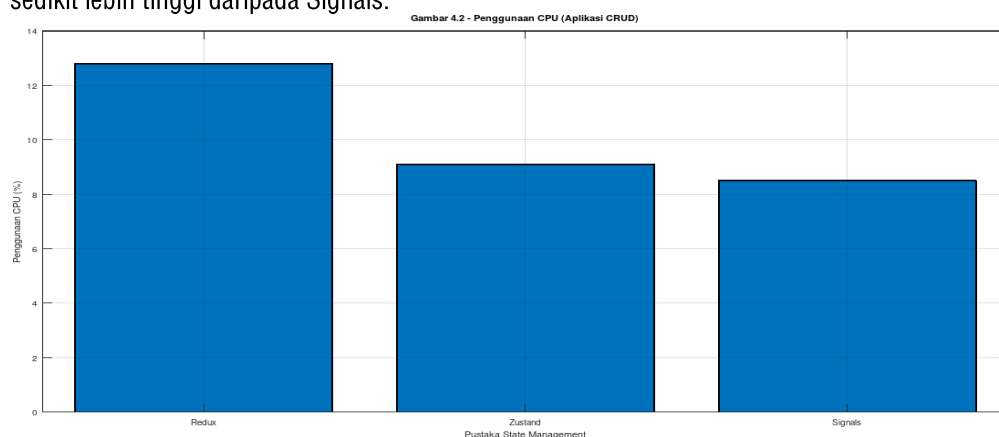
|         |               |                  |                |   |
|---------|---------------|------------------|----------------|---|
| Zustand | 25,34 ± 9,73  | 1214,60 ± 202,81 | 234,40 ± 35,92 | 8 |
| Signals | 31,20 ± 10,56 | 1041,40 ± 132,47 | 241,20 ± 31,45 | 5 |

Berdasarkan data pada Gambar 3, hasil pengujian penggunaan memori menunjukkan perbedaan efisiensi yang jelas di antara ketiga pustaka yang diuji. **Signals** tercatat sebagai yang **paling efisien**, dengan konsumsi memori terendah, yaitu hanya **45,8 MB**. Di sisi lain, **Redux** menunjukkan penggunaan memori **tertinggi** dalam pengujian ini, dengan konsumsi sebesar **65,2 MB**. **Zustand** berada di posisi tengah, membuktikan efisiensi yang lebih baik daripada Redux dengan penggunaan memori sebesar **50,3 MB**, namun belum seefisien Signals.



**Gambar 3. Grafik Perbandingan Penggunaan Memori: Diagram batang yang membandingkan penggunaan memori (MB) untuk Redux, Zustand, dan Signals pada aplikasi sederhana.**

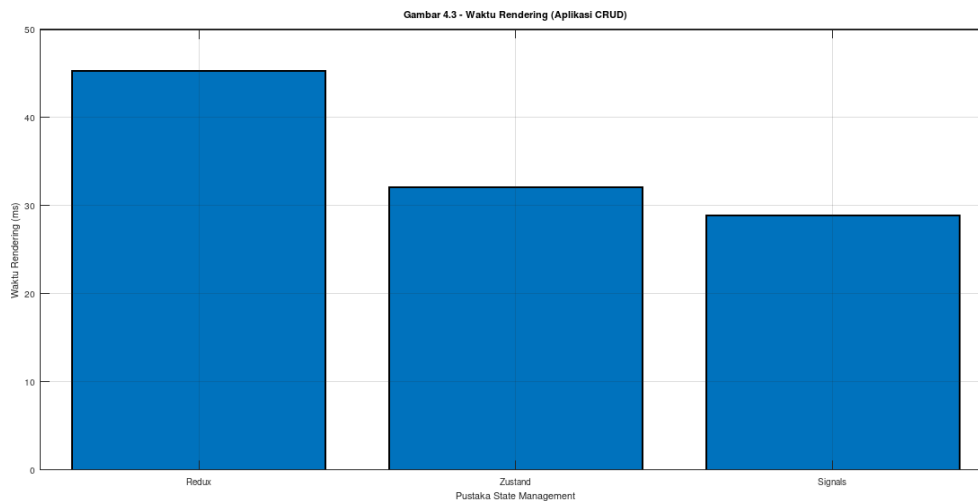
Berdasarkan data pada Gambar 4, grafik perbandingan penggunaan CPU menunjukkan tren yang serupa dengan efisiensi memori. **Signals** terbukti sebagai pustaka yang **paling ringan** beban pemrosesannya, dengan penggunaan CPU terendah, yaitu **8,5%**. Sebaliknya, **Redux** tercatat sebagai yang **paling intensif** terhadap CPU, dengan penggunaan sebesar **12,8%**. **Zustand** kembali menempati **posisi tengah**, dengan penggunaan CPU sebesar **9,1%**, yang menunjukkan beban kerja lebih ringan secara signifikan dibandingkan Redux tetapi masih sedikit lebih tinggi daripada Signals.



**Gambar 4. Grafik Perbandingan Penggunaan CPU: Diagram batang untuk metrik penggunaan CPU (%).**

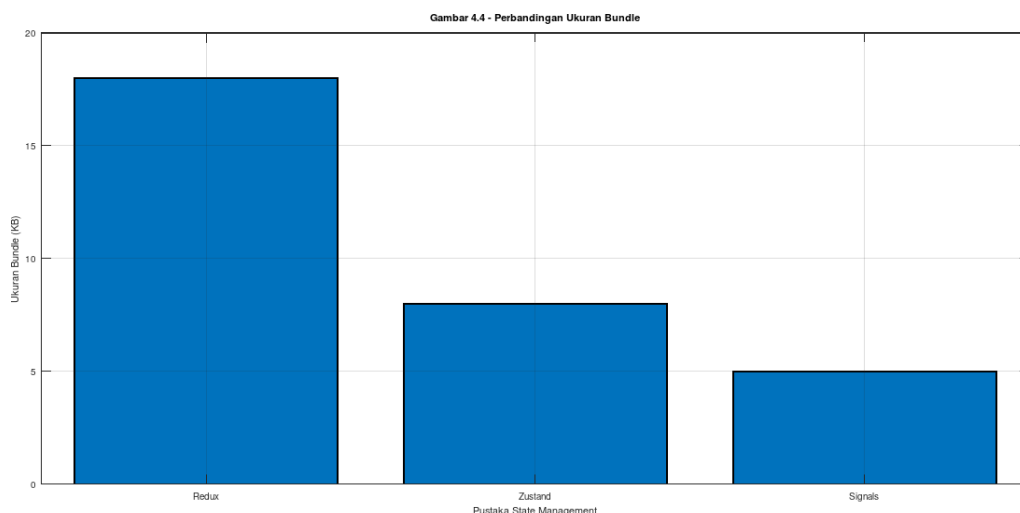
Data pada Gambar 5, menunjukkan perbandingan langsung waktu yang dibutuhkan oleh setiap pustaka untuk me-render pembaruan pada *User Interface* (UI), yang diukur dalam milidetik (ms). Hasilnya secara jelas mengidentifikasi **Signals** sebagai yang **paling cepat**, dengan waktu *rendering* terendah yaitu **28,9 ms**.

Sebaliknya, **Redux** tercatat sebagai yang **paling lambat** dalam skenario pengujian ini, membutuhkan waktu **45,3 ms** untuk menyelesaikan render. **Zustand** kembali menempati posisi tengah dengan waktu *rendering* **32,1 ms**, menunjukkan performa yang jauh lebih cepat daripada Redux tetapi sedikit lebih lambat dibandingkan dengan pendekatan reaktivitas granular dari Signals



Gambar 5. Grafik Perbandingan Waktu *Rendering*: Diagram batang untuk ukuran *rendering* (ms).

Gambar 6, menyajikan perbandingan ukuran *bundle* (dalam KB), yang mengukur seberapa besar ukuran file JavaScript yang ditambahkan oleh setiap pustaka ke dalam aplikasi akhir. Metrik ini sangat penting karena berdampak langsung pada waktu muat awal aplikasi. Hasil pengujian menunjukkan bahwa **Signals** adalah yang **paling ringan**, dengan ukuran *bundle* yang sangat kecil, yaitu hanya **5 KB**. Sebaliknya, **Redux** adalah yang **terbesar** di antara ketiganya, dengan ukuran **18 KB**. **Zustand** menempati posisi tengah dengan ukuran **8 KB**, yang membuktikan ukurannya jauh lebih kecil dibandingkan Redux, namun sedikit lebih besar dari Signals



Gambar 6. Grafik Perbandingan Ukuran *Bundle*: Diagram batang untuk ukuran bundle (KB).

### 3.3 Analisis Korelasi dan Interpretasi Integrasi Performa

Berdasarkan hasil pengujian kuantitatif pada Tabel 1, ditemukan adanya korelasi linear dan hubungan sebab-akibat yang kuat antar-metrik performa yang dipengaruhi oleh karakteristik arsitektur masing-masing pustaka.

Hubungan paling nyata terlihat antara metrik Penggunaan CPU (%) dan Waktu *Rendering* (ms). Pustaka *Signals* mencatatkan beban CPU terendah (8,5%) yang berbanding lurus dengan waktu respons *rendering* paling cepat (28,9 ms). Hal ini membuktikan keunggulan arsitektur *fine-grained Reactivity* pada *Signals* yang memperbarui node DOM secara langsung lewat properti `.value` tanpa memicu proses komputasi ulang (*Virtual DOM diffing*) standar *React*, sehingga mampu memangkas beban kerja CPU dan mempercepat respons antarmuka. Sebaliknya, *Redux* menunjukkan akumulasi beban tertinggi (CPU 12,8% dan render 45,3 ms) karena alur *action* dan *reducer* konvensionalnya memicu kalkulasi ulang dalam skala komponen yang lebih luas.

Sementara itu, korelasi antara Ukuran *Bundle* (KB) dan Penggunaan Memori (MB) juga memperlihatkan dampak struktural yang signifikan. Pustaka modern yang dirancang minimalis seperti *Signals* (5 KB) dan *Zustand* (8 KB) memiliki ukuran berkas JavaScript akhir yang jauh lebih kecil dibandingkan *Redux* (18 KB). Ukuran *bundle* yang ringkas ini terbukti berbanding lurus dengan efisiensi alokasi memori *runtime* di dalam peramban, di mana *Signals* hanya mengonsumsi memori sebesar 45,8 MB dan *Zustand* sebesar 50,3 MB, sedangkan *Redux* menyita memori paling tinggi hingga 65,2 MB. Secara keseluruhan, analisis interpretasi gabungan ini membuktikan bahwa efisiensi arsitektur internal pada pustaka *Zustand* dan *Signals* memberikan keuntungan performa yang komprehensif dalam menghemat sumber daya komputasi di sisi klien.

#### 4. KESIMPULAN

Berdasarkan analisis hasil penelitian dan pembahasan, dapat ditarik kesimpulan bahwa pemilihan pustaka *state management* memiliki dampak yang langsung, signifikan, dan terukur terhadap performa aplikasi *React*. Dalam pengujian pada skenario aplikasi sederhana maupun aplikasi CRUD kompleks, pustaka *Signals* secara konsisten menunjukkan performa paling unggul di semua metrik yang diuji (penggunaan memori, penggunaan CPU, waktu *rendering*, dan ukuran *bundle*). Hal ini membuktikan bahwa arsitektur *fine-grained Reactivity* (reaktivitas granular) sangat efisien untuk menangani pembaruan *state* yang dinamis. *Zustand* terbukti menjadi alternatif modern yang sangat kompetitif, menawarkan keseimbangan yang baik antara kesederhanaan sintaks, ukuran *bundle* yang kecil, dan performa yang tinggi, secara signifikan mengungguli *Redux*. *Redux*, meskipun memiliki ekosistem yang matang, menunjukkan beban performa tertinggi dalam skenario pengujian, terutama pada aplikasi CRUD yang interaktif. Bagi pengembang, disarankan untuk mempertimbangkan *Signals* atau *Zustand* untuk proyek *React* baru yang memprioritaskan performa tinggi, efisiensi sumber daya, dan waktu muat cepat. Untuk penelitian selanjutnya, disarankan agar dilakukan pengujian serupa dengan melibatkan lebih banyak pustaka *state management* (misalnya *Jotai*, *MobX*) [12], menguji pada aplikasi dengan skala yang jauh lebih besar, serta melakukan pengujian pada berbagai perangkat pengguna akhir untuk mendapatkan data performa di dunia nyata.

#### DAFTAR PUSTAKA

- [1] V. Veeri, "State Management in React: Redux vs Zustand - A Comprehensive Guide," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 2024. doi: 10.32628/CSEIT24106172.
- [2] G. N. C. Aluri and H. K. Mupparapu, "Redux State Management Patterns for Large-Scale React.js Enterprise Applications: An Empirical Analysis," *Int. J. Emerg. Res. Eng. Technol.*, 2024, doi: 10.63282/3050-922X.IJERET-V5I2P120.
- [3] A. S. Bachelor dan B. I. Technology, "Comparing State Management Between Redux And Zustand In *React*," 2023. [Daring]. Tersedia pada: <https://www.theseus.fi/handle/10024/816299>
- [4] N. S., S. C., S., dan D. S., "Performance and Developer Experience Comparison of Redux, Zustand, and Context API in *React* Applications," *Int. J. Sci. Technol.*, vol. 16, no. 2, hal. 1–9, 2025, doi: 10.71097/ijst.v16.i2.5026.
- [5] R. Yeh, "Zustand vs Redux: Making Sense of *React* State Management," 7 November 2025. [Daring]. Tersedia pada: <https://www.wisp.blog/blog/zustand-vs-redux-making-sense-of-React-state-management>
- [6] Poimandres, "Introduction," 7 November 2025. [Daring]. Tersedia pada: <https://zustand.docs.pmnd.rs/getting-started/introduction>
- [7] M. Hagemeister dan J. Miller, "Introducing Signals," 7 November 2025. [Daring]. Tersedia pada:

- <https://pReactjs.com/blog/introducing-signals/>
- [8] E. Bainomugisha, A. Lombide Carreton, T. Van Cutsem, S. Mostinckx, dan W. De Meuter, "A Survey on Reactive Programming," *ACM Comput. Surv.*, vol. 45, 2012, doi: 10.1145/0000000.0000000.
  - [9] S. Ulili, "Zustand vs. Redux Toolkit vs. Jotai." [Daring]. Tersedia pada: <https://betterstack.com/community/guides/scaling-nodejs/zustand-vs-redux-toolkit-vs-jotai/>
  - [10] S. Shubham, "Signals and Fine-grained *Reactivity*: The Next Wave in State Management," 7 November 2025. [Daring]. Tersedia pada: <https://medium.com/@shubhamsinghsengar8/signals-and-fine-grained-Reactivity-the-next-wave-in-state-management-3c32e3762fa6>
  - [11] PReact, "Signals," 7 November 2025. [Daring]. Tersedia pada: <https://pReactjs.com/guide/v10/signals/>
  - [12] M. Weststrate, "MobX: Simple, Scalable State Management," 7 November 2025. [Daring]. Tersedia pada: <https://mobx.js.org/README.html>