Implementasi REST API untuk Komunikasi Antara ReactJS dan NodeJS
(Studi Kasus : Modul Manajemen User Solusi247)

Rizqi Kartika Safitri
Jurusan Informatika Fakultas Teknologi Industri
Universitas Islam Indonesia
Yogyakarta
17523153@students.uii.ac.id

Hanson Prihantoro Putro
Jurusan Informatika Fakultas Teknologi Industri
Universitas Islam Indonesia
Yogyakarta
hanson@uii.ac.id

Abstract*— Penggunaan modul sangat membantu dalam pengembangan perangkat lunak. Seorang *developer* menggunakan modul untuk menghemat waktu dan efisiensi dalam pengembangan perangkat lunak. Pengembangan modul ini menggunakan *framework* ReactJS dan NodeJS. Meski keduanya berasal dari keluarga Javascript, tentu memiliki ranah yang berbeda yakni ReactJS biasa digunakan sebagai *frontend* (*client*), sedangkan NodeJS digunakan sebagai *backend* (*server*). Oleh karena itu keduanya sering dipadukan untuk mengembangkan perangkat lunak aplikasi. Masalah yang muncul dalam melakukan pertukaran data salah satunya, yaitu *client* tidak dapat mengakses *resource* secara langsung pada *server* karena *client* dan *server* merupakan dua bagian yang terpisah. Hal ini memerlukan komunikasi antara keduanya agar *client* dapat terhubung ke *server*. Dari permasalahan tersebut, dikembangkanlah API dengan arsitektur REST (*Representational State Transfer*) yang menjembatani dalam transfer data. Dalam implementasinya, REST API menggunakan sebuah alamat identitas untuk melakukan *request-response* dengan berbagai operasi data terdapat *method* GET, POST, DELETE. Keluaran yang dihasilkan dari REST dibuat dalam bentuk JSON (*JavaScript Object Notation*) sehingga lebih fleksibel penggunaannya. Hasil implementasi REST API dapat membantu meningkatkan efisiensi waktu dan efektifitas bandwidth dalam pengembangan aplikasi serta memudahkan pengembangan dalam hal komunikasi dan pertukaran data.

Keywords— Modul; REST API; ReactJS; NodeJS; JSON;

I. PENDAHULUAN

Manajemen *User* merupakan sebuah modul berbasis web yang berfungsi sebagai pengelola *user*. Seperti yang telah diketahui, penggunaan modul sangat membantu *developer* dalam pengembangan perangkat lunak. Pada umumnya, *developer* menggunakan modul untuk menghemat waktu dan efisiensi dalam pengembangan perangkat lunak. Konsep modul sendiri berasal dari paradigma pemrograman modular yang merupakan pendekatan untuk mengorganisir kode yang dibuat, sehingga setiap fungsi dapat melakukan tugas masing-masing secara spesifik dengan tujuan agar program mudah dipahami dan mudah digunakan kembali untuk pengembangan [1].

Modul Manajemen *User* akan dirilis ke berbagai proyek dan aplikasi yang dikembangkan oleh perusahaan, seperti ERP, POS, dan lain-lain. Berawal dari adanya kesamaan antara proyek yang satu dengan yang lainnya, maka dibuatlah sebuah modul yang fleksibel diterapkan diberbagai proyek perusahaan. Modul ini dikembangkan dengan *framework Javascript*, yaitu ReactJS dan NodeJS. Meskipun keduanya memiliki kesamaan dalam hal bahasa yang digunakan, ReactJS biasa digunakan dalam ranah *frontend* (*client side*) dan umumnya dipadukan dengan NodeJS sebagai *backend* (*server side*). Dalam melakukan pertukaran data, *client* tidak dapat mengakses *resource* secara langsung pada *database server* sehingga diperlukan komunikasi antara keduanya agar *client* dapat mengakses *resource* yang ada pada *server*. Hal ini juga untuk menghindari interaksi langsung ke *database*.

Oleh karena itu, untuk mengatasi permasalahan tersebut dikembangkanlah API (*Application Programming Interface*) yang menjembatani transfer data. API memiliki berbagai jenis arsitektur salah satunya yaitu REST. REST (*Representational State Transfer*) merupakan seperangkat prinsip arsitektur yang melakukan transmisi data melalui antarmuka yang terstandarisasi seperti HTTP [2]. Untuk dapat mengakses *resource*, diperlukan URI (*Uniform Resource Identifier*) sebagai pengenal ketika melakukan *request* ke HTTP. Terdapat beberapa HTTP *method* seperti GET, PUT, DELETE, POST. Metode REST memiliki keunggulan yang dapat dipertimbangkan dalam hal kecepatan transfer data dan performa daripada metode SOAP [3]. Pada implementasinya, keluaran yang dihasilkan REST lebih fleksibel karena dapat berupa JSON atau XML sedangkan untuk SOAP hanya berupa XML [4].

Karya ilmiah ini bertujuan untuk menjelaskan bagaimana REST API digunakan sebagai jembatan komunikasi dalam pengembangan Modul Manajemen *User* yang lebih baik saat menggunakan ReactJS dan NodeJS. Adapun manfaat yang dihasilkan adalah penggunaan API akan mempermudah *developer* dalam kolaborasi antar sistem dan juga membantu dalam mengembangkan modul baru pada sistem yang bersangkutan [5]. Selain itu REST API juga memudahkan pengembangan dalam hal komunikasi dan pertukaran data.

Berdasarkan pertimbangan pada uraian di atas, pengembangan modul dengan metode REST API diharapkan dapat meningkatkan efisiensi dan efektivitas dalam pengembangan aplikasi. Developer tidak perlu mengulangi pembuatan fitur yang sudah ada, tetapi dapat menggunakan layanan yang sudah ada. Selain itu, REST API juga mampu meningkatkan fleksibilitas dalam pengembangan selanjutnya.

II. KAJIAN PUSTAKA

Dalam penulisan karya ilmiah ini, digali beberapa penelitian untuk mendapatkan informasi kasus yang serupa sebagai landasan penulisan. Dimulai dari sebuah penelitian [6] yang menjelaskan tentang pengembangan REST API pada Sistem Informasi Akademik. Dengan metode ini, komunikasi cukup melalui *web service* menggunakan API dan mendapatkan keluaran data berformat JSON sehingga meminimalisir akses langsung ke *database*. Kelebihan dari penelitian tersebut adalah penggunaan REST API untuk memudahkan integrasi dan pengembangan selanjutnya tanpa mengganggu kode utama program. Penerapan API ini dibangun dengan *framework* React Native yang digunakan untuk aplikasi berbasis *mobile*. Sama dengan penelitian sebelumnya, penelitian [7] yang menjelaskan tentang implementasi REST API pada aplikasi android, juga menggunakan format JSON untuk mendukung agar performa menjadi lebih baik. Hal ini dibuktikan bahwa transfer data lebih cepat dilakukan. Pada penelitian ini, digunakan Codeigniter untuk implementasi *web service*.

Dalam implementasi REST API, dapat tersedia berbagai macam *framework* selain yang telah digunakan pada dua penelitian yang telah dilakukan. Penelitian [8] menjelaskan tentang perancangan web service pada sebuah sistem informasi menggunakan NodeJS sebagai backend. Dipilihnya NodeJS sebagai bahasa pemrograman sisi *server* karena memiliki sifat *asynchronous* atau *non-blocking* dengan *event-driven*. Selain itu, penelitian [9] tentang pengembangan aplikasi *Geotagging*, menjelaskan bahwa sifat NodeJS tersebut dapat meminimalisir kemungkinan terjadinya *overhead* dan skalabilitas web *server* menjadi lebih optimal.

Jika dalam penelitian sebelumnya telah disebutkan penggunaan React Native, penelitian yang lain [10] menjelaskan tentang perancangan sebuah sistem manajemen menggunakan framework Codeigniter dan ReactJS. Meskipun React Native dan ReactJS berasal dari keluarga yang sama, keduanya memiliki basis yang berbeda yaitu berbasis mobile dan web. Penggunaan ReactJS di sini sebagai pendukung web *framework*. Kelebihan ReactJS di antaranya memiliki dokumentasi yang lengkap dan mudah digunakan untuk pengembangan aplikasi berbasis web, REST API, maupun membuat *web framework* yang kompleks.

Dari penelitian yang ada, telah dilakukan pengembangan *Restful web service* untuk berbagai macam aplikasi baik berbasis mobile maupun web. Beberapa penelitian yang telah diuraikan menggunakan berbagai macam *framework* seperti Codeigniter, NodeJS, React Native maupun ReactJS. Terdapat dua penelitian, telah menggunakan NodeJS sebagai *server*, dan satu penelitian yang menggunakan ReactJS sebagai *frontend*. Namun belum ada penelitian yang melakukan penggabungan antara ReactJS sebagai *frontend* dan NodeJS sebagai *backend* pada sebuah *web service*. Oleh karena itu, pada penelitian ini akan dilakukan implementasi REST API sebagai jembatan komunikasi antara ReactJS sebagai *frontend* dan NodeJS sebagai *backend*.

III. METODOLOGI

Pada tahap ini dilakukan metodologi penerapan REST API dengan tahapan perancangan, implementasi dan pengujian. Berikut tahapan secara detail:

A. *Perancangan REST API*

Tahap perancangan merupakan tahap yang penting untuk dijadikan dasar dalam pengembangan yang dilakukan. Dalam perancangan API untuk Modul Manajemen *User* menggunakan arsitektur REST yang dibangun dengan basis web. Pada tahap ini akan dijelaskan bagaimana arsitektur sistem yang digunakan. Selain itu juga membahas tentang penggunaan URI sebagai alamat identitas untuk mengakses *resource*.

B. *Implementasi REST API*

Tahap ini menjelaskan bagaimana implementasi REST API yang akan dibangun dengan bahasa pemrograman *Javascript* pada sisi *client* maupun *server*, serta MySQL sebagai *database* untuk menampung *resource*. REST API akan dibangun secara terpisah antara *client* dan *server*. Selain itu, juga disimulasikan dengan penyimpanan lokal sehingga keduanya menggunakan alamat localhost namun dengan nomor port yang berbeda yakni 3000 untuk *server* dan 3001 untuk *client*.

C. *Pengujian API*

Terakhir adalah tahap pengujian yang berfungsi untuk memastikan kualitas dan keandalan API dapat berjalan dengan baik. Pengujian terhadap API dilakukan dengan *tools* Postman. Caranya dengan mengatur *method* dan memasukkan alamat API yang akan dipanggil. Pengujian juga diperlukan untuk mencegah berbagai kemungkinan kendala yang terjadi, sehingga dapat menimbulkan kerugian ke depannya.

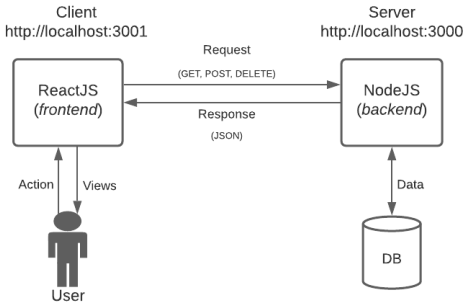
IV. HASIL DAN PEMBAHASAN

Pada bab ini, akan dibahas secara jelas hasil dari setiap tahapan yang sudah diuraikan pada bab sebelumnya.

A. *Perancangan REST API*

Sebagaimana yang telah dijelaskan pada metodologi, pada tahap ini akan dijelaskan hasil perancangan REST API yang telah dilakukan. REST API mempresentasikan interaksi antara *client* (ReactJS) dengan *server* (NodeJS) pada Modul Manajemen *User* seperti pada Gambar 1. Agar keduanya dapat saling berkomunikasi, maka diperlukan API sebagai penghubung antar aplikasi yang dibuat. Beberapa *method* yang digunakan antara lain:

1) *GET*, untuk membaca data dari *database*
2) *POST*, untuk membuat data baru dan memperbarui data pada *database*
3) *DELETE*, untuk menghapus data dari *database*



Gambar 1. Arsitektur Sistem

Berdasarkan rancangan arsitektur sistem pada Gambar 1, *client* melakukan HTTP *Request* dengan *method* GET, POST, DELETE ke *server* yang menyediakan *resource* melalui URI atau sering disebut dengan *endpoint*. Kemudian REST *Server* memberikan umpan balik berupa HTTP *Response* sesuai permintaan dari sisi *client*. *Server* akan mengembalikan *response* kepada *client* dengan format JSON. Keluaran yang dihasilkan berformat JSON tersebut akan diubah sedemikian rupa sehingga dapat dibaca dengan jelas pada sisi *client.*

B. Implementasi REST API

Tahap selanjutnya adalah implementasi REST API menggunakan NodeJS sebagai *server side* (REST *Server*) dan ReactJS sebagai *client side* (REST *Client*) sesuai pada Gambar 1. Modul Manajemen *User* ini memiliki operasi dasar CRUD (*Create, Read, Update, Delete*) dengan tambahan fitur lain seperti *pagination*, *sort*, *search*, dan *limit show*. Pentingnya penggunaan REST API selain sebagai penghubung, juga memungkinkan aplikasi yang berbeda untuk berkomunikasi dan berbagi data.

1) REST Server

Dalam arsitektur REST, terdapat beberapa *method* yang didukung oleh protokol HTTP. Namun, disini hanya akan menggunakan tiga *method*, antara lain yaitu GET, POST, dan DELETE. Tabel I berikut adalah daftar *endpoint* yang dikembangkan.

TABEL I. ENDPOINT API

Endpoint	Method
`/api/users`	`GET`
`/api/users`	`POST`
`/api/users/:id`	`GET`
`/api/users/:id`	`POST`
`/api/users/id`	`DELETE`

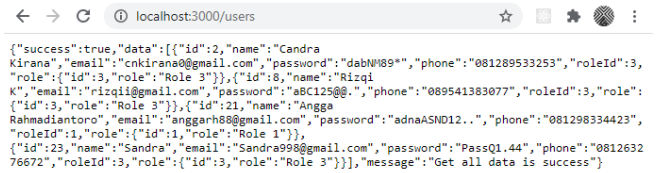
Untuk mengelola *request-response* sekaligus *routing* operasi HTTP, menggunakan *framework* Express yang merupakan web *framework* pada *server side*. Selanjutnya dibuat *controlle*r untuk menentukan data yang dikirimkan agar dapat diakses oleh pengguna. *Controller* yang dibuat mencakup seluruh *endpoint* API, sehingga efektif untuk pertukaran data. Gambar 2 merupakan salah satu contoh implementasi *endpoint* `/api/users` menggunakan *method* GET pada *controller*.

```
//GET ALL DATA FROM USER
controllers.list = async (req,res) => {
  try{
    const data = await User.findAll({
    include: [ Role ]
  });
  if(data){
    res.status(200).json({success:true,data:data, message:"Get all data is success"})
  }

  return res.status(404).json({message:"Data does not exists"});
  } catch (error) {
    return res.status(500).json({error: error.message})
  }
};
```

Gambar 2. Contoh *Method* GET pada *Controller*

Penggunaan *method* GET pada Gambar 2 untuk menampilkan seluruh data *user* yang dapat diakses menggunakan alamat tertentu sesuai dengan *endpoint* yang telah dibuat. Pada contoh tersebut digunakan alamat http://localhost:3000/api/users. Bagian pertama adalah nama domain. Dikarenakan dalam implementasi ini masih menggunakan *database* lokal, maka dituliskan "localhost". Angka "3000" merupakan nomor *port* dari *server*. Kemudian selanjutnya adalah direktori yang menandakan layanan API dapat diakses. Bagian terakhir memuat informasi yang ada dalam *controller*. Pada Gambar 3 diperlihatkan hasil dari pemanggilan alamat yang telah disebutkan di atas. Keluaran yang didapatkan berupa format JSON, kemudian akan diolah kembali agar mudah terbaca oleh *client.*



Gambar 3. Keluaran Data Format JSON

2) REST Client

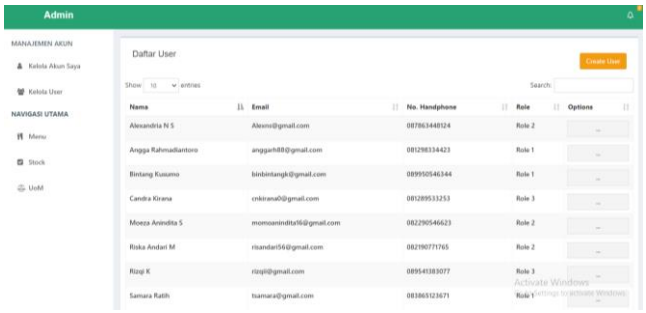
Pada *client side* diimplementasikan dengan ReactJS, yang menjadi salah satu *framework* pengembangan berbasis web. Agar *client* dapat melakukan operasi HTTP ke *endpoint* API, perlu menggunakan bantuan *library*. Saat ini yang digunakan adalah *library* Axios yang bersifat *open source*. Penggunaannya dengan menambahkan kode berikut: `import axios from 'axios'`. Gambar 4 merupakan salah satu contoh baris kode saat *client* melakukan HTTP *request* untuk menampilkan seluruh data *user* yang tersimpan dalam *database* menggunakan *method* GET.

```
class readUser extends Component {
  /**...
  constructor(props){
    super(props);
    this.state = {
      listUser:[]
    }
    this.child = React.createRef();
  }
  /**...
  componentDidMount(){
    this.loadUser()
    const script = document.createElement("script");
      script.src = 'js/content.js';
      script.async = true;
      document.body.appendChild(script);
  }
  /**...
  loadUser(){
    const url = Config.baseUrl+"/api/users"
    return axios.get(url)
    .then(res=>{
      if (res.data.success) {
        const data = res.data.data
        this.setState({listUser:data})
      }
      else {
        alert("Error web service")
      }
    })
    .catch(error=>{
      alert("Error server "+error)
    })
  }
```

Gambar 4. Kode Untuk *Request Method* GET

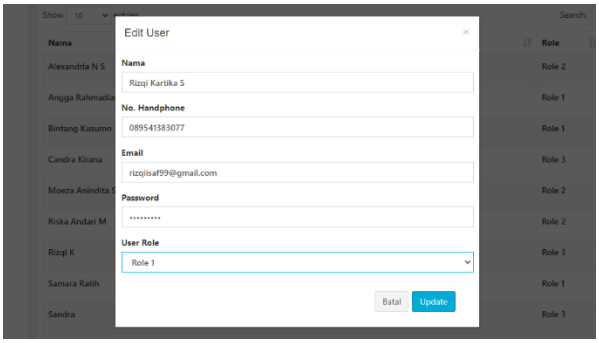
Pada bagian *constructor*, mendefinisikan *state* `listUser` yang bertipe array ([]) sebagai tempat penampungan data dengan format JSON yang berhasil dimuat oleh Axios. Kemudian, dalam *method* `componentDidMount()`, dilakukan eksekusi *method* `loadUser()`. Dalam *method* tersebut, Axios akan menarik data dari alamat API, kemudian meletakkan data pada *constanta* `data` dan memperbarui pada `listUser:data`.

Terlihat pada Gambar 5 merupakan tampilan dari halaman awal Modul Manajemen *User*. Halaman ini berguna untuk mengelola seluruh data pengguna melalui fitur CRUD yang disediakan, seperti: membuat pengguna baru, edit dan update data pengguna, serta hapus data pengguna. Selain fitur CRUD, pada halaman ini juga terdapat fitur pencarian, *pagination*, *limit show* dan *sort*.



Gambar 5. Halaman Awal Modul Manajemen *User*

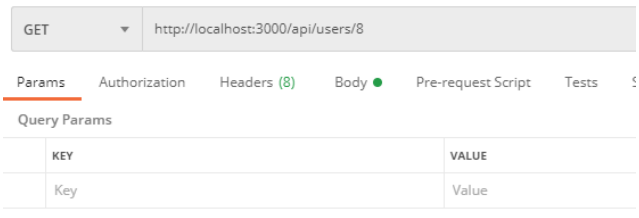
Contoh lain pada Gambar 6 merupakan tampilan halaman edit data pengguna pada Modul Manajemen *User*. Halaman ini berisi nama, nomor *handphone*, *email*, *password* dan *role*. Untuk menuju halaman ini hanya dengan memilih Edit pada bagian *Options* yang terdapat pada halaman awal Modul Manajemen *User*.



Gambar 6. Halaman Edit User pada Modul Manajemen User

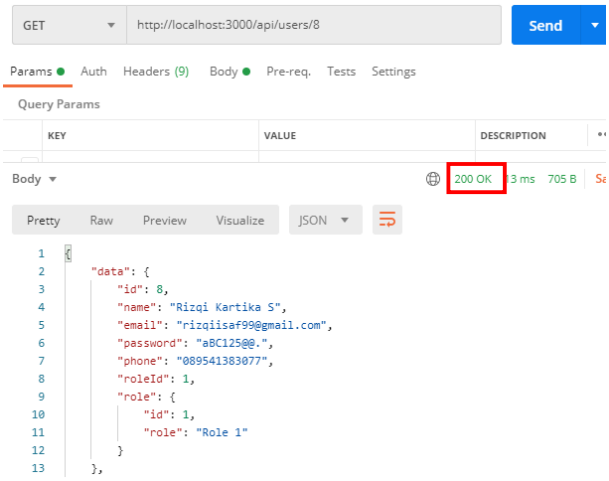
3) Pengujian API

Tahap terakhir yang perlu dilakukan adalah melakukan pengujian. Pengujian akan menentukan kelayakan REST API yang telah dibuat sehingga dapat digunakan secara optimal untuk Modul Manajemen *User*. Adapun cara menguji API dengan Postman adalah dengan mengatur *method* sesuai yang akan diuji dan memasukkan *endpoint* seperti pada Gambar 6.



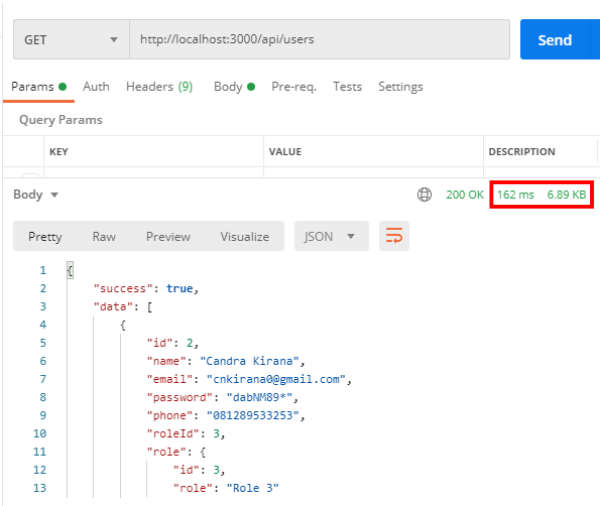
Gambar 7. Pengujian Fungsionalitas API pada Postman

Pengujian Gambar 7 untuk mengambil data pengguna sesuai *id* melalui *endpoint* `/api/users/:id`. Setelah mengirim *request*, kemudian REST *server* memberi *response* bahwa *request* berhasil yang ditandai dengan status *code* 200. Detail hasil terlihat pada Gambar 8 yang berformat JSON.



Gambar 8. *Response* API `/api/users/:id`

Kemudian, contoh pengujian yang lain pada Gambar 9 menunjukkan pengujian untuk mengambil seluruh data user yang terdapat pada *server*. Pengujian ini dilakukan dengan *request* data ke *endpoint* `/api/users`, lalu *server* memberikan *response* dengan waktu yang singkat sekitar 162 milidetik untuk menampilkan 25 data *user*.



Gambar 9. *Response* API `/api/users`

Berdasarkan pada pengujian yang telah dilakukan, penggunaan API relatif efisien karena tidak boros dalam penggunaan bandwidth dan tidak memakan waktu yang lama saat pertukaran data. Untuk lebih jelasnya, dapat dilihat pada Tabel II yang menunjukkan seluruh daftar API yang diuji beserta hasilnya.

TABEL II. DAFTAR PENGUJIAN API

Endpoint	Fitur	Hasil	Waktu (ms)	Bandwidth (byte)
`/api/users`	Menampilkan seluruh data pengguna	Sukses	162 ms	6890 byte

`/api/users`	Membuat pengguna baru	Sukses	74 ms	560 byte
`/api/users/:id`	Mengambil data pengguna berdasarkan id	Sukses	13 ms	705 byte
`/api/users/:id`	Memperbarui data pengguna berdasarkan id	Sukses	14 ms	656 byte
`/api/users/id`	Menghapus data pengguna berdasarkan id	Sukses	48 ms	554 byte

Hasil dan pembahasan REST API sudah sesuai dengan apa yang dijelaskan pada tahap perencanaan hingga pengujian. Penggunaan REST API mampu memudahkan *client* untuk mengambil *resource* ke basisdata *server* melalui operasi HTTP. Namun, pada saat ini belum ada implementasi terkait keamanan penggunaan REST API dan penambahan fitur *login* untuk otorisasi. Hal tersebut dkhawatirkan akan menyebabkan kerusakan atau hilangnya data pada penyimpanan.

V. KESIMPULAN DAN SARAN

Berdasarkan hasil yang didapat dari tahap perencanaan, implementasi dan pengujian, dapat disimpulkan bahwa penggunaan REST API dalam pengembangan Modul Manajemen *User* mampu bekerja secara efisien karena tidak boros dalam penggunaan bandwidth dan hemat waktu dalam proses pertukaran data. Selain itu juga efektif untuk berkomunikasi melalui seluruh *endpoint* API yang telah dibuat. Kombinasi antara *framework* ReactJS dan NodeJS melengkapi layanan yang dibutuhkan untuk *frontend* dan *backend*.

Dengan adanya REST API, *client* dapat mengakses *resource* pada *server* melalui operasi HTTP dengan *method* GET, POST, dan DELETE. Selain sebagai jembatan komunikasi, REST API juga membantu *developer* dalam kolaborasi antar sistem dan membantu pengembangan modul terkait agar menjadi lebih baik. Penggunaan format JSON sebagai hasil *output* dipilih karena merupakan teks sederhana dan lebih mudah diolah.

Adapun saran guna meningkatkan hasil penelitian selanjutnya antara lain, diperlukan penelitian lanjut untuk mengetahui risiko keamanan dari penggunaan API dan memberikan authorization untuk mencegah pencurian data. Perlu pengembangan fitur tambahan lainnya seperti *login* agar modul menjadi lebih baik.

DAFTAR PUSTAKA

[1] S. Seydnejad, Modular Programming with JavaScript, Birmingham: Packt Publishing Ltd, 2016.

[2] S. Dhingra, "REST vs. SOAP: Choosing the best web service," TechTarget, 07 November 2016. [Online]. Available: https://searchapparchitecture.techtarget.com/tip/REST-vs-SOAP-Choosing-the-best-web-service. [Accessed 26 October 2020].

[3] A. Dudhe and S. S. Sherekar, "Performance Analysis of SOAP and RESTful Mobile Web," *Second National Conference on Recent Trends in Information Security*, January 2014.

[4] K. Wagh and T. Ravindra, "A Comparative study of SOAP vs REST web services provisioning techniques," *Journal of Information Engineering and Applications*, vol. 02, July 2012.

[5] B. A. Pranata, "Perancangan Application Programming Interface (API) Berbasis Web Menggunakan Gaya Arsitektur Representational State Transfer (REST) Untuk Pengembangan Sistem Informasi Administrasi Pasien Klinik Perawatan Kulit," *Universitas Lampung*, 2017.

[6] D. Wijonarko and B. W. R. Mulya, "Pengembangan Antarmuka Pemrograman Aplikasi," *Smatika*, vol. 08, Oktober 2018.

[7] A. Rulloh, D. E. Mahmudah and H. Kabetta, "Implementasi REST API pada Aplikasi Panduan Kepaskibraan Berbasis Android," *Jurnal Teknikom*, vol. 01, 2017.

[8] A. Firdaus, S. Widodo, A. Sutrisman, S. G. F. Nasution and R. Mardiana, "Rancang Bangun Sistem Informasi Perpustakaan Menggunakan Web Service Pada Jurusan Teknik Komputer Polsri," *Jurnal Informanika*, vol. 05, 2019.

[9] N. R. Kusworo, A. Arwan and A. A. Soebroto, "Pengembangan Aplikasi Geotagging Pelaporan Keluhan Masyarakat pada Dinas Perhubungan Kota Mojokerto menggunakan Restful Web Services berbasis Mobile," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 03, 2019.

[10] M. Wali and L. Ahmad, "Perancangan Access Open Journal System (AOJS) dengan menggunakan Framework Codeigniter dan ReactJs," *Jurnal JTIK (Jurnal Teknologi Informasi dan Komunikasi)*, 2018.