IMPLEMENTASI ARSITEKTUR MICROSERVICE PADA PEMBUATAN SURAT UNIT KEGIATAN MAHASISWA INFORMATIKA DAN KOMPUTER MENGGUNAKAN NODE.JS

Y.Yohakim Marwanta[1], Badiyanto[2]
[1,2] Teknik Informatika, STMIK Akakom, Yogyakarta
Email: yohakim@akakom.ac.id, badi@akakom.ac.id

Abstrak

Microservice adalah kumpulan proses independen dan kecil yang berkomunikasi antara satu dengan lainnya untuk membentuk aplikasi kompleks yang agnostik terhadap bahasa API apa pun. Servis-servis ini terdiri dari blok-blok kecil, terpisah, dan fokus pada tugas-tugas ringan untuk memfasilitasi metode modular dalam pembangunan sistem. Arsitektur bergaya microservice mulai menjadi standar dalam pembangunan sistem yang dinamis dan konstan berkembang.

REST API merupakan implementasi dari API (Application Programming Interface). REST (Representational State Transfer) adalah suatu arsitektur metode komunikasi yang menggunakkan protokol HTTP untuk pertukaran data. Dimana tujuannya adalah untuk menjadikan sistem yang memiliki performa yang baik, cepat dan mudah untuk dikembangkan (scale) terutama dalam pertukaran dan komunikasi data.

Pada penelitian ini peneliti mencoba menerapkan arsitektur Microservice pada aplikasi Pembuatan Surat Unit Kegiatan Mahasiswa Informatika dan Komputer dengan menggunakan Node.js sebagai sistem backend. Arsitektur ini dimanfaatkan untuk meningkatkan performa dan pengembangan (scale) sistem.

Penelitian ini menghasilkan sebuah sistem aplikasi yang lebih flexible baik dalam pengembangan atau perawatan karena penerapan arsitektur Microservice yang dapat memisahkan atau membagi suatu sistem yang besar menjadi sistem-sistem kecil yang disesuaikan dengan fitur dan fungsinya. Selain itu, arsitektur ini juga memisahkan antara Frontend dengan Backend dari sistem sehingga performa aplikasi menjadi lebih baik, cepat, dan mudah dikembangkan (scalable).

Kata Kunci : Arsitektur, Microservice, Backend, Frontend, REST API.

Abstract

A micro-service is a collection of small, independent processes that communicate with each other to form complex applications that are agnostic to any API language. These services consist of small, discrete, light-duty focused blocks to facilitate a modular method of system building. Micro-service-style architecture is starting to become the standard in the development of dynamic and constantly evolving systems.

REST API is an implementation of API (Application Programming Interface). REST (Representational State Transfer) is an architectural method of communication that uses the HTTP protocol for data exchange. Where the goal is to make a system that has good performance, is fast, and easy to scale, especially in data exchange and communication.

In this study, the researcher tried to apply the Micro-service architecture for Making Informatics and Computers Student Activity Unit Letters using Node.js as a back-end system. This architecture is used to improve system performance and scale.

This research produces an application system that is more flexible both in development and maintenance because of the application of a Micro-service architecture that can separate or divide a large system into small systems that are adapted to their features and functions. In addition, this architecture also separates the Front-end and Back-end from the system so that application performance is better, faster, and easily scalable.

Keywords: Architecture, Backend, Microservice, Frontend, REST API.

I. PENDAHULUAN

Unit Kegiatan Mahasiswa Informatika dan Komputer atau yang disingkat menjadi UKM IK adalah salah satu organisasi yang ada di lingkungan STMIK AKAKOM Yogyakarta dengan tujuan untuk ikut berperan menggali, meningkatkan dan mengembangkan kreatifitas di bidang keilmuan, penalaran ilmiah dan melaksanakan kegiatan

sosial di masyarakat. Dalam struktur organisasinya terdapat divisi sekretaris yang bertugas mencatat semua administrasi kegiatan salah satu diantaranya adalah mengelola surat. Menurut Kamus Besar Bahasa Indonesia (KBBI) sekretaris adalah orang (pegawai, anggota pengurus) yang diserahi pekeraan tulis-menulis, atau surat-menyurat, dan sebagainya; penulis; panitia;.

Sekretaris organisasi adalah mahasiswa aktif di STMIK AKAKOM Yogyakarta. Seseorang dikatakan sebagai mahasiswa aktif ditandai dengan lunasnya pembayaran Sumbangan Pembinaan Pendidikan (SPP) Tetap dan berhasil melakukan Kartu Rencana Studi (KRS) yang diketahui oleh akademisi kampus terkait mata kuliah dan jadwal yang diambil. Adanya jadwal mata kuliah yang harus dihadiri oleh Sekretaris merupakan kewajiban yang tidak bisa ditinggalkan, sehingga terjadi penundaan terhadap pembuatan surat-menyurat. Disisi lain, pembuatan surat dilakukan melalui perangkat laptop atau komputer dengan bantuan aplikasi Microsoft Office Word karena semua dokumen masih tersimpan secara offline pada penyimpanan laptop atau komputer pribadi.

Arsitektur Microservice adalah kumpulan proses independen dan kecil yang berkomunikasi antara satu dengan lainnya untuk membentuk aplikasi kompleks yang agnostik terhadap bahasa API apa pun. Servis-servis ini terdiri dari blok-blok kecil, terpisah, dan fokus pada tugas-tugas ringan untuk memfasilitasi metode modular dalam pembangunan sistem. Arsitektur bergaya microservice mulai menjadi standar dalam pembangunan sistem yang dinamis dan konstan berkembang.

Dalam Arsitektur Microservice, komunikasi antar service atau yang biasa dikenal dengan network-call untuk transfer data dilakukan melalui beberapa cara yang salah satunya adalah Remote Procedure Invocation (RPI) agar service eksternal tidak langsung berkomunikasi dengan database. Salah satu teknologi yang digunakan berkaitan dengan framework yang akan digunakan yaitu ExpressJS. Framework ExpressJS didesain untuk membangun web aplikasi dan API, dengan ini aplikasi dapat melakukan transfer data melalui RPI yaitu menggunakan REST API (HTTP).

REST API merupakan implementasi dari API (Application Programming Interface). REST (Respresentation State Transfer) adalah suatu arsitektur metode komunikasi yang menggunakkan protokol HTTP untuk pertukaran data. Dimana tujuannya adalah untuk menjadikan sistem yang memiliki performa yang baik, cepat dan mudah untuk dikembangkan (scale) terutama dalam pertukaran dan komunikasi data. Aplikasi ini dibangun sebagai contoh pengimplementasian arsitektur microservice pada media pembuatan dan pengelolaan surat-menyurat yang ada didalam UKM IK. Aplikasi ini dapat digunakan oleh sekretaris organisasi untuk mengatasi keterlambatan dalam pembuatan surat atau perbaikan surat pada saat ia mengikuti perkuliahan. Dimana sekretaris akan menambahkan stuktur dasar surat kedalam aplikasi yang berbentuk berkas dokumen bertipe docx yang sudah diberikan variabel sebagai tanda data (untuk berkomunikasi dengan aplikasi) yang akan diubah secara dinamis melalui aplikasi. Implementasi Arsitektur Microservice dilakukan menggunakan Node.js sebagai sistem backend-nya. Dimana arsitektur ini membuat aplikasi menjadi lebih scaleable, secure, dan reliable, sehingga dalam proses transfer data menjadi lebih cepat dan aman.

II. METODE

1. Prosedur dan Pengumpulan Data

Metode yang digunakan dalam proses pengumpulan data sebagai berikut:

a. Studi Literatur
   Studi Literatur dilakukan dengan mencari bahan materi yang berhubungan dengan arsitektur *Microservices* terutama pada saat menggunakan teknologi Node.js. Hal ini dilakukan guna mempermudah proses implementasi sistem baik dalam perancangan, pengembangan ataupun uji coba sistem. Pencarian materi dilakukan melalui buku panduan, buku bacaan, journal, dan internet.
b. Dokumen
   Pengumpulan data melalui file dokumen yang dimiliki oleh Sekretaris UKM IK untuk mendapatkan riwayat data surat seperti nomor surat, prihal surat, tanggal surat dan lain sebagainya.

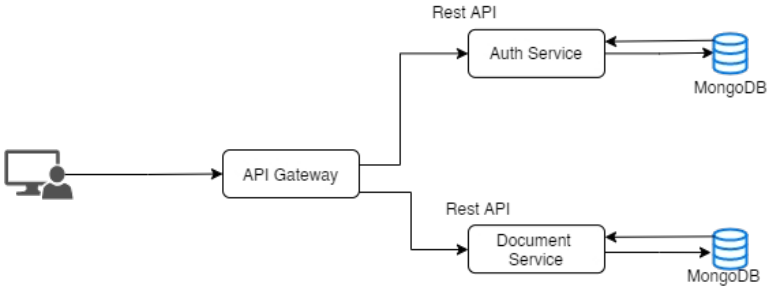
2. Analisis dan Perancangan Sistem

Pada sistem yang dibuat ini terdapat beberapa rancangan yang akan digunakan sebagai pedoman implementasi sistem ke dalam program aplikasi yang akan dibuat.

a. Arsitektur Sistem

User melakukan akses melalui website yang terhubung ke API Gateway, API Gateway adalah suatu tempat antara pengguna dengan layanan (services) yang bertingak sebagai routing request dari permintaan (request) pengguna ke service. Dari API Gateway, akses menuju fitur-fitur kecil yang telah dipisahkan dapat diakses melalui satu alamat domain tanpa harus membuat request untuk masing-masing service secara

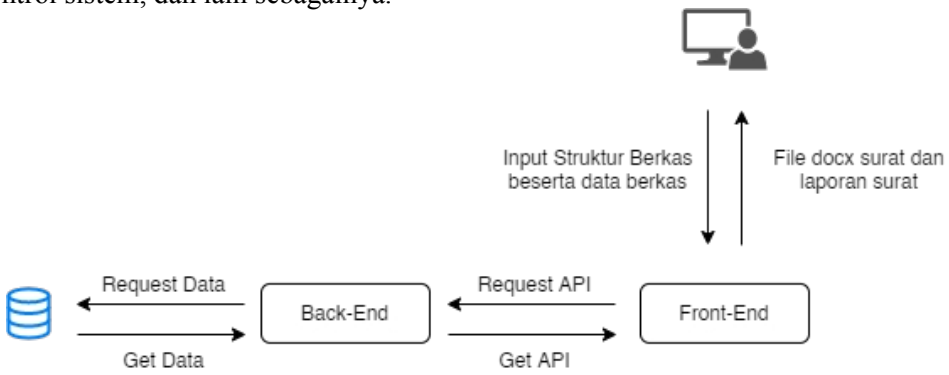
langsung, arsitektur dari sistem seperti terlihat pada gambar 2.1



Gambar 2.1 *Arsitektur Microservice*

Dalam pengembangan sistem aplikasi ini sepenuhnya menggunakan Bahasa pemrograman Javascript yang menggunakan Node.js sebagai compiler backend untuk Javascript. Adanya Node.js memungkinkan Javascript dapat bekerja sebagai backend sehingga mengubah fungsi bahasa pemrograman yang awalnya hanya untuk membuat website lebih interaktif (frontend) menjadi server (backend) juga seperti mengakses menyimpan, mengelola, membaca, memperbaharui, dan menghapus data yang tersimpan di dalam database.

Setiap service akan dibentuk dengan menggunakan Node.js bersama dengan ExpressJS sebagai frameworknya. ExpressJS menyediakan fitur atau fungsi yang dapat memudahkan programmer untuk membuat REST API dengan lebih mudah dan singkat dibandingkan membuat sendiri setiap routing, akses database, kontrol sistem, dan lain sebagainya.



Gambar 2.2 *Arsitektur Sistem pada Document Service*

Mengacu pada Gambar 2.1 maka Gambar 2.2 diatas menjelaskan secara singkat tentang alur penggunaan aplikasi secara sederhana jika sistem terlepas dari arsitektur Microservice. Pengguna akan mengakses sistem melalui *website* yang sudah terhubung dengan API *Gateway* dengan Axios. *Request* dilakukan melalui *frontend* ke *backend* melalui API *Gateway* untuk diteruskan ke masing-masing layanan agar dapat berkomunikasi dengan *database*.

Hal pertama yang dilakukan Sekretaris adalah menambahkan template surat kedalam sistem. Template yang dimasukkan tentu harus sudah diberikan penanda dan harus dengan format file *docx* agar aplikasi dapat berjalan dengan baik. Format penulisan penanda mengikuti kaedah penulisan yang ditentukan dari *library* yang digunakan yaitu docxtemplater.

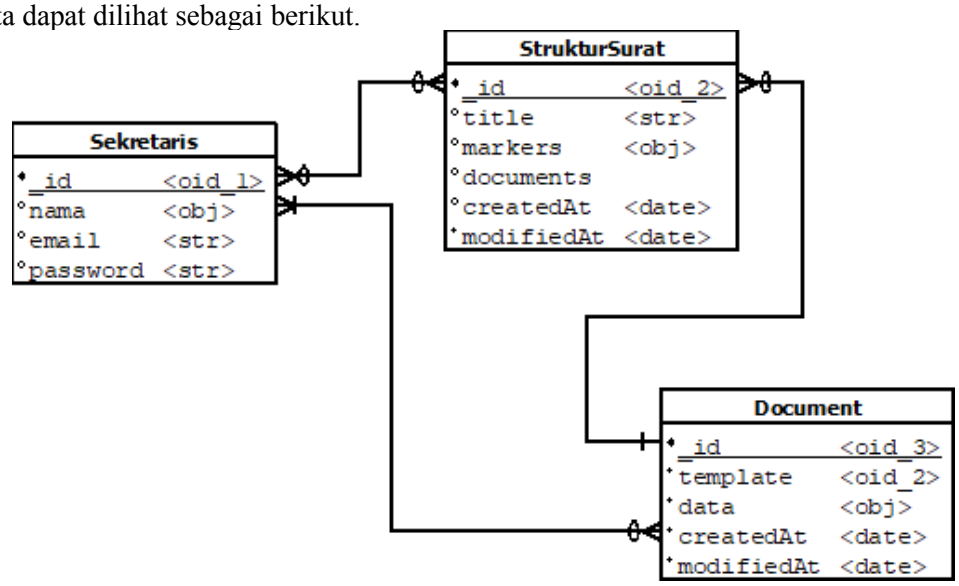
Struktur data (template) yang sudah disiapkan dan tersimpan didalam *database* nantinya akan digunakan oleh Sekretaris dalam pembuatan surat baru. Sekretaris meminta *request* template surat melalui *frontend* ke *backend* dan *request* tersebut diteruskan ke *database* oleh *backend*. Dari *database* akan memberikan respon berupa data struktur beserta tag-tag yang sudah ada dan disiapkan sebelumnya ke *backend* untuk diteruskan oleh *backend* menuju sisi front-end agar Sekretaris dapat memberikan nilai untuk suratnya.

Surat yang sudah diberikan nilai (data) dapat diunduh menjadi berkas dokumen (*.docx*) dan juga disimpan kedalam penyimpanan data atau database. Data ini difungsikan untuk memberikan kemudahan untuk sekretaris ketika ingin mengunduh kembali surat yang sudah dibuatnya.

b. Perancangan Basis Data

Penyimpanan data yang digunakan pada sistem ini adalah basis data NoSQL yaitu MongoDB. Model

penyimpanan data dapat dilihat sebagai berikut.



Gambar 2.3 *Rancangan Database*

```
// Bentuk JSON
{
  // Sekretaris
  sekretaris: { _id: <oid_1>,
    name: {fname: <str>, lname: <str>,},
    email: <str>, password: <str>,
  },
  // Struktur Surat (Template)
{
_id: <oid_2>,  title: <str>, markers: [{
isTable: <bool>,markerLabel: <str>,marker: <str>
}],
documents: <array>,
createdAt: <date>,
modifiedAt: <date>
  },
  // Document
  {
    _id: <oid_3>, title: <oid_1>,template: <obj>,
    data:
{
      markerOne: <str>,
      markerTwo: <arr>,
      ...
      markerN: <str>,
    }
  }
}
```

Gambar 2.4 *Model Basis Data MongoDB*

Dari struktur atau model basis data pada Gambar 3.6 atau struktur JSON diatas, terdapat satu *collections* yang akan menampung semua *document* didalam MongoDB. *Collections* dalam MongoDB mirip seperti tabel pada *Relational Database Management System* (RDBMS), pada sebuah *Collections* dapat memiliki *document* (*record/row* dalam RDBMS) yang berbeda-beda dengan *field* (kolom dalam RDBMS).

Dalam MongoDB, terdapat dua model konsep dasar untuk menghubungkan data yaitu *Embedded Data Model* dan *Normalized Data Model.*

MongoDB memiliki cara yang berbeda dengan basis data relasional dalam perancangan dan penanganan data. MongoDB memungkinkan untuk langsung menggabungkan beberapa table pada RDBMS menjadi satu *collection*, sehingga jika diperhatikan pada Gambar 3.6 ada hubungan *many-to-many* yang langsung terhubung dari dokumen satu dengan lainnya. Untuk menandakan relasinya ditandai dengan *field* _*id* yang bertugas sebagai *primary key* pada MongoDB, dari model basis data yang ada penggunaan _*id* menunjukkan *Normalized Data Model* (*Reference*).

Pada *field* data pada dokumen surat dan *field* marker pada document Template memperoleh data yang akan dinamis dimana itu akan berbeda antara surat satu dengan lainnya menyesuaikan dengan masing-masing kebutuhan yang ada pada surat yang dimasukkan.

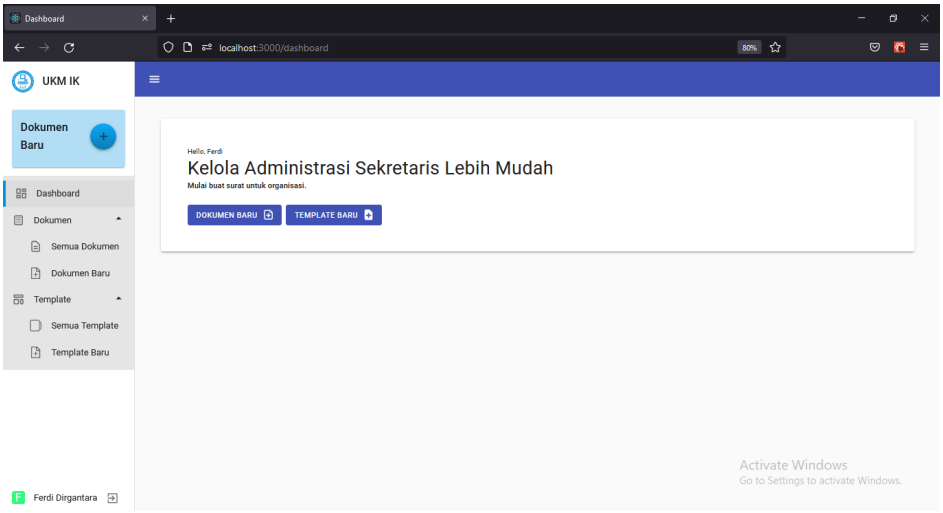
III. HASIL DAN PEMBAHASAN

Merujuk pada desain rancangan arsitektur sistem pada Gambar 2.1 komunikasi yang terjadi pada sisi *frontend* hanya melalui API Gateway saja. Dalam sistem pada saat dijalankan memang terasa tidak memiliki perbedaan sama sekali seperti aplikasi pada umumnya, namun itu hanya berlaku pada sisi *frontend* (pengguna) saja. Seperti yang terlihat pada potongan program sebelumnya terdapat REST API yang mengatur jalannya setiap permintaan pengguna kemasing-masing layanan. Misalkan pada Auth Service yang mengatur pengguna yang dapat mengakses sistem aplikasi seperti dituliskan pada Gambar 3.1 dan 3.3 terkait Login dan Register.

Penggunaan autentikasi tersebut terlihat sangat normal, namun pada hasil akhirnya disisi *backend* layanan ini memiliki database yang berbeda (terenkapsulasi) dengan database dokumen. Begitu pula dengan layanan dokumen yang terenkapsulasi bersama dengan databasenya tersendiri.

Pada saat pertama kali pengguna melakukan login kedalam aplikasi, hal pertama yang akan ditampilkan sistem adalah halaman dashboard. Perlu diketahui bahwa pengembangan sistem pada sisi *frontend* ini dibuat dengan menerapkan bentuk Single Page Application (SPA) menggunakan ReactJS.

SPA disini memberikan keuntungan dalam hal performa aplikasi saat digunakan. Permintaan terhadap aplikasi hanya dilakukan dalam satu kali request saja pada saat pertama kali aplikasi diakses, sehingga pada saat membuka alamat lainnya dalam sistem menjadi lebih cepat.



Gambar 3.1 *Tampilan Dashboard aplikasi*

Pada gambar 3.1 adalah tampilan awal setelah pengguna atau sekretaris melakukan login. Pengembangan aplikasi ini menerapkan pengembangan aplikasi SPA (*Single Page Application*) sehingga request antarmuka pengguna akan diminta satu kali saja keserver dan memberikan performa yang lebih baik untuk pengguna saat menggunakannya.

A. Template

Kunci utama dari aplikasi ini adalah pada sisi dokumennya agar penelitian terkait implementasi arsitektur microservice ini bisa berjalan dengan baik. Salah satunya adalah terkait template (struktur dasar) dari dokumennya.

Persiapan template tidak dilakukan langsung melalui sistem dalam aplikasi, melainkan harus melalui aplikasi pengolah kata seperti Microsoft Office Word terlebih dahulu. Penulisan markerpun harus disesuaikan dengan ketentuan yang dibutuhkan oleh *library* NodeJS yaitu docxtemplater.

Template yang diterima sistem melalui *form* yang dikirimkan menggunakan Axios akan dibaca dan disimpan ke collection template pada database MongoDB.

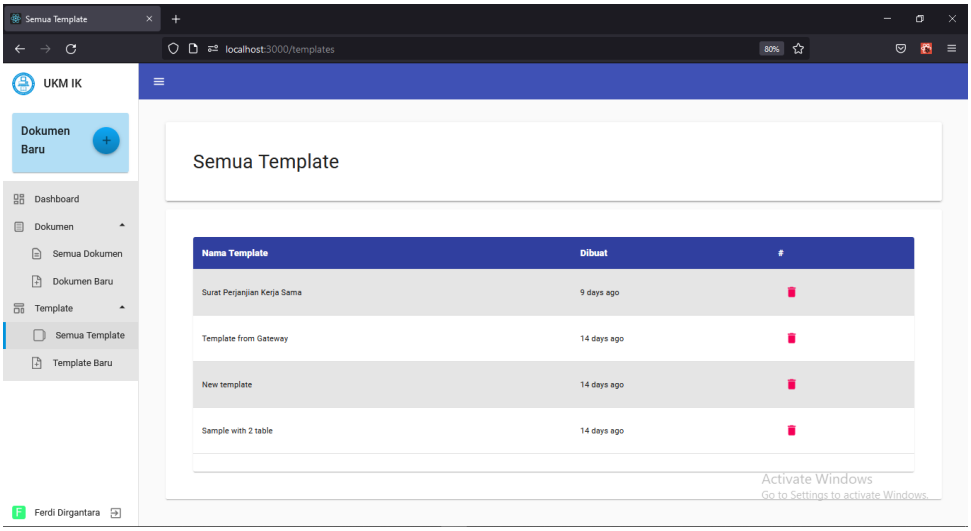
```
// Controller untuk membuat template baru
export const createTemplate = async (req, res) => {
  const markers = req.body.markers;
  const newTemplate = new Template({
    title: req.body.title,
    markers: JSON.parse(markers),
    file: req.file.filename
  })
  try {
    const data = await newTemplate.save()
    res.status(201).json(data)
  } catch (error) { console.log(error)}
}
//Route API create template
app.use("/api/template", TemplateRoute);
router.post("/", upload.single("file"), createTemplate);
// Akses api create template
const API = axios.create({ baseURL: 'http://localhost:9090' })
export const createTemplate = (newTemplate) => API.post(`/api/template`,
   newTemplate)
export const createTemplate = (newTemplate) => async (dispatch) => {
  try {
    const { data } = await api.createTemplate(newTemplate);
    dispatch({ type: "CREATE", payload: data });
  } catch (error) {console.log(error)}
}
```

Gambar 3.2 *REST API menambahkan template baru*

Gambar 3.3 *Contoh ketentuan template surat*

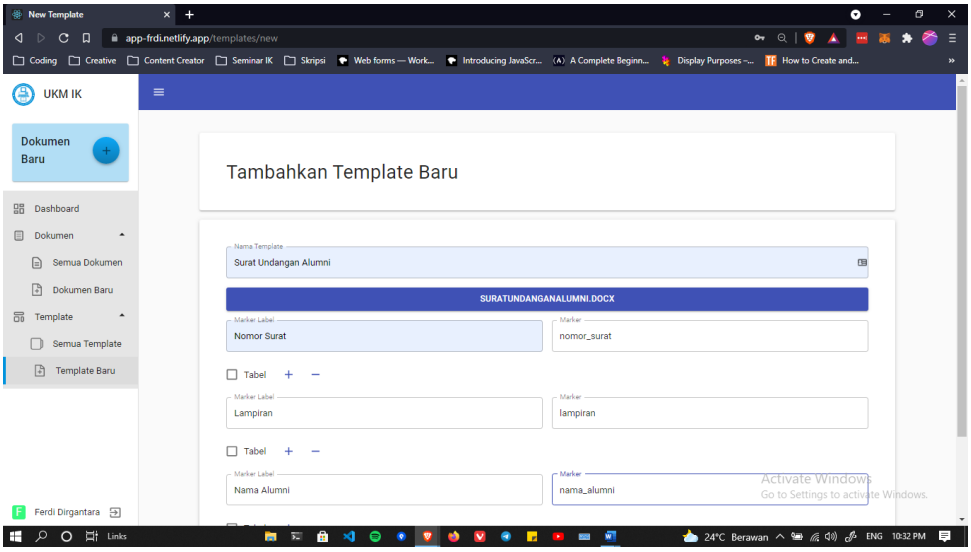
Ketentuan yang digunakan adalah dengan menuliskan nama marker yang diapit oleh kurung kurawal. Salah satu contoh marker yang tertulis pada Gambar 3.3 adalah marker untuk nomor surat yaitu **{nomor_surat}**. Begitupun dengan marker-marker lainnya, semuanya harus diapir oleh tanda kurung kurawal.

Nama marker dalam file juga harus memperhatikan case maupun spasi, apabila terjadi perbedaan antara file dengan data yang diinputkan melalui sistem seperti Gambar 3.5.



Gambar 3.4 *Tampilan Daftar Template*

Pada gambar 3.4 diatas merupakan memberikan infromasi kepada pengguna terkait daftar template yang sudah tersimpan didalam database. Template yang sudah tertera didalam daftar inilah yang nantinya dapat digunakan sebagai bahan dasar membuat dokumen (surat) baru pada saat penambahan dokumen baru.



Gambar 3.5 *Tampilan Tambah Template*

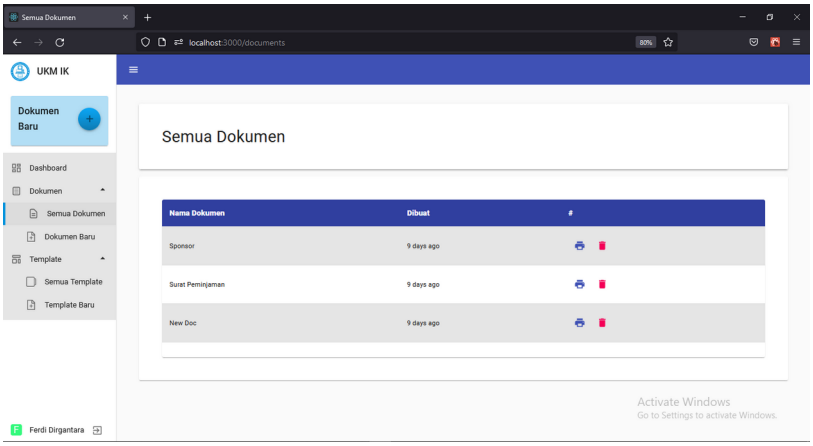
B. Dokumen

Bagian kedua dari fungsi sistem utama aplikasi yaitu pengelolaan dokumen surat milik UKM IK. Pembuatan dokumen ini menjadi satu bagian dengan template sehingga database yang digunakanpun cukup satu database saja. Namun, antara template dengan dokumen memliki collection yang berbeda untuk masing-masing bagian karena bentuk dan kebutuhan yang digunakan berbeda satu sama lain. Setelah sistem memiliki template dokumennya, maka selanjutnya adalah pembuatan dokumen surat itu sendiri. API ini akan menyimpan data-data yang dibutuhkan untuk membuat surat baru seperti yang dijelaskan pada bab sebelumnya.

```
// Controller GET Documents
export const getDocs = async (req, res) => {
  try {
    const documents = await Document.find().populate("template", "file docu-
ments").sort({ createdAt: -1 });
    res.status(200).json(documents);
  } catch (error) {
```

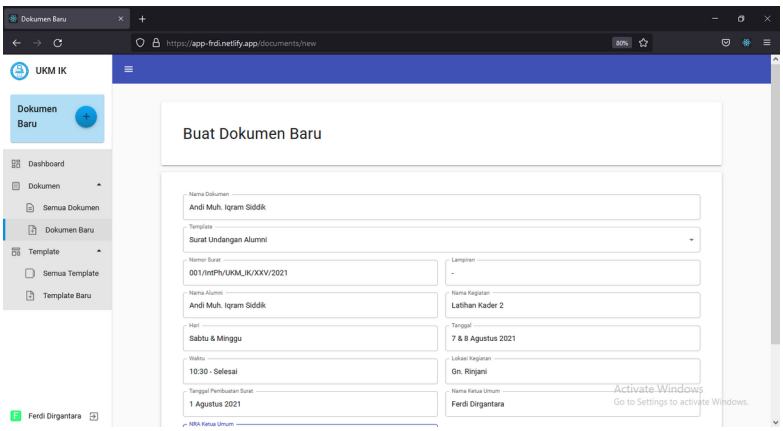
```
    console.log(error);
  }
};
// Route API GET Documents
app.use("/api/document", DocumentRoute);
router.get("/", getDocs);
// Akses API dengan Axios
const API = axios.create({ baseURL: 'http://localhost:9090' })
export const getDocs = () => API.get(`/api/document`)
export const getDocs = () => async (dispatch) => {
  try {
    const { data } = await api.getDocs();
    dispatch({ type: "GET_DOCS", payload: data });
  } catch (error) {
    console.log(error)
  }
};
```

Gambar 4.1 *REST API GET Documents*



Gambar 3.2 *Tampilan daftar dokumen yang tersimpan*

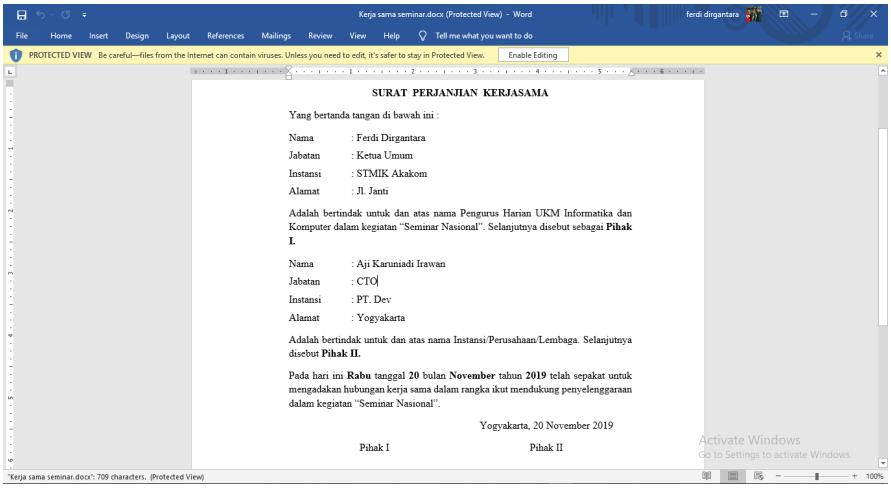
Data terkait dokumen yang tersimpan didalam database bisa diakses melalui REST API GET *document* dan dimanfaatkan dalam bentuk tabel seperti yang terlihat pada Gambar 3.6. Namun, sebelum dapat mengakses data tentunya harus ditambahkan terlebih dahulu data yang ingin diambil melalui form tambah dokumen baru seperti yang ditampilkan pada gambar 3.6.



Gambar 3.6 *Tampilan tambah dokumen baru*

Karena setiap dokumen memiliki kebutuhan dan jumlah marker yang berbeda-beda, maka *form* masukan datanya pun akan berubah-ubah mengikuti data yang dibutuhkan untuk masing-masing dokumen.

Data-data yang akan dibuat untuk suatu dokumen nantinya akan tersimpan kedalam database agar bisa digunakan kembali apabila ingin mencetak ulang dokumennya. Selain itu, setelah pengguna menekan tombol tambahkan, secara otomatis pengguna juga akan langsung diarahkan untuk menyimpan file tersebut kedalam penyimpanan lokal pengguna.



Gambar 4.3 *Tampilan output file setelah diditambahkan dan diunduh*

Jika data yang dimasukkan pada saat pembuatan stuktur dan juga pada saat menambahkan data, maka hasil akhir akan sesuai dengan apa yang dimasukkan. Jika tidak terdapat hasil undefined pada marker yang telah ditentukan maka dapat dikatakan bahwa semua marker dan data berhasil tersimpan.

IV. SIMPULAN DAN SARAN

Berdasarkan perancangan dan implementasi dari BAB sebelumnya, maka penelitian ini dapat diambil kesimpulan sebagai berikut :

1. Penerapan arsitektur microservice memberikan kemudaan pada saat pengembangan aplikasi kedepannya jika diperlukan penambahan fitur yang mampu meningkatkan performa aplikasi. Hal ini terjadi karena pengembang hanya perlu berfokus pada satu struktur sistem yang kecil tanpa harus memperhatikan keterkaitannya dengan fitur lainnya.
2. Service-service yang sudah dibuat juga dapat digunakan pada aplikasi lainnya tanpa harus mengganggu aplikasi utama karena mereka berdiri sendiri untuk setiap layanannya. Selain itu, data yang tersimpan juga akan menjadi lebih sedikit pada aplikasi lainnya karena tidak perlu menyimpan data layanan kedalam sistem utamanya.
3. Performance aplikasi menjadi lebih cepat karena menerapkan SPA (Single Page Application) pada aplikasi sehingga proses loading menjadi baik dan hanya melakukan satu kali request saja ke server untuk tampilannya.

Dari uraian pembahasan pada bab sebelumnya, sistem aplikasi ini masih mempunyai kekurangan. Adapun saran yang berguna untuk pengembangan aplikasi ini sebagai berikut :

1. Memperbaiki fitur *Authentication* untuk pengguna yang berfungsi untuk menandai siapa yang menambahkan template dan membuat dokumen baru. Serta bisa diberikan *role based permission* agar bisa membatasi akses tertentu.
2. Memperbaiki *user interface* aplikasi agar data tabel bisa diakses lebih mudah saat diakses melalui perangkat seluler.

PUSTAKA

[1] Kiddy.2018.*Kenapa API Harus Microservices?*.https://medium.com/@kiddy.xyz/pengertian- microservices-dan-kenapa-restful-api-harus-dibuat-seperti-itu-7326f217042d. 23 November 2020, 20:00 WIB.

[2] Putra, Andre. 2020. *Restfull Api Untuk Menampilkan List Berita Menggunakan Arsitektur Microservices (Studi Kasus : Portal Berita)*. Yogyakarta : STMIK AKAKOM Yogyakarta.

[3] Purnama, Heri. 2016. *Aplikasi Pengelolaan Skripsi di STMIK AKAKOM Menggunakan Arsitektur*

Microservice. Yogyakarta : STMIK AKAKOM Yogyakarta.

[4] Qamarudin, Ahmad. 2018. *Implementasi Arsitektur Microservice Menggunan Restpul API Untuk Portal Akakdemik PP AL-MUNAWWIR.* Yogyakarta : STMIK AKAKOM Yogyakarta.

[5] Yudana. 2019. *Pengertian dan Konsep Restful Api Programming.* https://www.yudana.id/pengertian-dan-konsep-restful-api-programming/. 05 April 2020, 20:00 PM.