Jurnal Teknik dan Sistem Komputer (JTIKOM)
Volume 6 No. 2, 28 Desember 2025
ISSN: 2723-6382

# Implementasi Docker Container Dalam Pembuatan Sistem Anggaran, Keuangan dan Akuntansi Pada Lingkungan Universitas

**Pratyaksa Ocsa Nugraha Saian[1], Ariya Dwika Cahyono[2], Sri Yulianto Joko Prasetyo[3]**
[123]Fakultas Teknologi Informasi, Universitas Kristen Satya Wacana
Jl. Dr. O. Notohamidjojo No.1 - 10, Salatiga 50715, Jawa Tengah, Indonesia
Email: [1]pratyaksa.ocsa@uksw.edu, [2]ariyadc@uksw.edu, [3]sri.yulianto@uksw.edu

***Abstract***

*Financial management and accounting are crucial aspects for higher education institutions that deal with complex administrative processes, including Satya Wacana Christian University in Salatiga. The financial application previously used at SWCU still had several limitations, such as difficulties in generating timely reports, duplicate transaction codes, and the inability to adapt to changes in policies or Standard Operating Procedures (SOPs). This study aims to develop a prototype of a financial and accounting management system based on microservices architecture utilizing Docker Container technology. The development method applied is Prototyping, which includes requirement analysis, system design, implementation, and evaluation of the prototype up to the third iteration. The results show that the adoption of microservices simplifies system management by dividing the application into eleven independent services. The implementation of Docker supports this architecture by providing separate containers for each service, thereby facilitating maintenance, development, and system scalability. The study concludes that a microservices-based approach using Docker can serve as an effective solution for building financial and accounting systems within universities.*

***Keywords****: Docker, Microservices, Financial System, Accounting System*

***Abstrak***

*Manajemen keuangan dan akuntansi merupakan aspek penting bagi perguruan tinggi yang memiliki kompleksitas administrasi tinggi, termasuk Universitas Kristen Satya Wacana Salatiga. Aplikasi keuangan yang sebelumnya digunakan di UKSW masih memiliki keterbatasan, seperti kesulitan menghasilkan laporan secara tepat waktu, terjadinya duplikasi kode transaksi, serta ketidakmampuan sistem mengikuti perubahan kebijakan atau SOP. Penelitian ini bertujuan untuk mengembangkan prototipe sistem manajemen keuangan dan akuntansi berbasis arsitektur microservices dengan memanfaatkan teknologi Docker Container. Metode pengembangan yang digunakan adalah Prototyping, dengan tahapan analisis kebutuhan, perancangan, pembangunan, serta evaluasi prototipe hingga iterasi ketiga. Hasil penelitian menunjukkan bahwa penerapan microservices mempermudah pengelolaan sistem melalui pembagian layanan menjadi sebelas service independen. Implementasi Docker terbukti mendukung penerapan arsitektur ini dengan menyediakan container terpisah bagi setiap service, sehingga memudahkan proses pemeliharaan, pengembangan, serta skalabilitas sistem. Penerapan microservices berbasis Docker dapat menjadi solusi efektif untuk membangun sistem keuangan dan akuntansi di lingkungan universitas.*

***Kata kunci****: Docker, Microservices, Sistem Keuangan, Sistem Akuntansi*

## 1. PENDAHULUAN

Manajemen Keuangan dan Akuntansi dalam sebuah perusahaan merupakan sesuatu yang penting dan harus diperhatikan demi kelancaran hidup dari perusahaan itu sendiri. Hal ini juga berlaku di lingkungan universitas, termasuk Universitas Kristen Satya Wacana (UKSW) sebagai salah satu perguruan tinggi besar di Salatiga. UKSW merupakan sebuah universitas swasta yang berada di Kota Salatiga. UKSW merupakan sebuah universitas yang cukup besar yang dapat terlihat dari adanya 15 fakultas dengan 63 program studi di dalamnya [1]. Oleh karena itu, sudah sewajarnya untuk UKSW membutuhkan juga proses manajemen keuangan dan akuntansi yang baik pula.

Saat ini, UKSW sudah menggunakan sebuah aplikasi untuk membantu itu semua. Namun, setelah dilakukan proses evaluasi dengan para pengguna dari aplikasi ini masih ditemukannya beberapa masalah. Sebagai contoh, sistem ini masih belum sesuai dengan harapan pengguna aplikasi dikarenakan adanya perubahan-perubahan kebijakan atau Standar Operasional Prosedur (SOP) yang tidak terakomodasi ke dalam sistem. Pencarian data-data transaksi keuangan juga masih sangat sulit untuk dicari maupun dilacak karena data yang tidak terorganisir dengan baik. Berdasarkan masalah tersebut, penelitian ini mengusulkan sebuah solusi untuk diterapkan. Dari beberapa penelitian sebelumnya [2], [3], [4] lebih banyak membahas penerapan *microservices* di dunia industri atau sekolah, namun belum membahas pada konteks sistem manajemen keuangan di lingkungan perguruan tinggi. Untuk itu dipilihlah sebuah konsep arsitektur sistem berupa *microservices* yang dirasa cocok untuk menjadi solusi bagi permasalahan tadi. Pada dasarnya, konsep *microservices* ini akan membagi sistem aplikasi yang besar menjadi beberapa *service* yang lebih kecil. Dengan begitu, proses *maintenance* dan *update* akan menjadi lebih mudah dilakukan [5].

Adanya teknologi Docker dipilih karena Docker akan menggunakan beberapa *container* untuk menjalankan sistemnya. *Container* inilah yang akan dipakai untuk setiap *services* yang berjalan dalam konsep *microservices* yang sudah disebutkan sebelumnya [6]. Penggunaan Docker ini juga akan membuat adanya pembagian tugas sesuai dengan *service* yang dibuat. Tidak hanya itu, dengan adanya Docker ini pengembangan dan pemeliharaan aplikasi juga dapat dilakukan bersamaan dengan digunakannya aplikasi [7]. Berdasarkan latar belakang masalah yang telah dikemukakan, rumusan masalah yang diangkat melalui penelitian ini adalah 1) Apakah penggunaan *microservices* dengan Docker dapat membantu pembuatan aplikasi sistem manajemen keuangan dan akuntansi universitas? 2) Apakah Docker dapat membantu untuk menerapkan konsep *microservices* ini?

Penelitian tentang penggunaan sistem informasi akuntansi pernah dilakukan sebelumnya. Penelitian yang dilakukan oleh Haria Saputri dkk menjelaskan bahwa penggunaan sistem informasi akuntansi membuat peningkatan yang signifikan terhadap kualitas laporan keuangan perusahaan di Jakarta Utara [8]. Begitu juga dengan penelitian yang dilakukan oleh Hafiz Riyadli dkk yang menjelaskan bahwa dengan dibangunnya sebuah sistem informasi keuangan dapat membantu perusahaan dalam mengelola administrasi keuangannya [9]. Kedua penelitian ini menunjukkan bahwa sebuah sistem informasi berperan penting bagi sebuah perusahaan dalam pengelolaan administrasi keuangan. Konsep ini yang mendasari yang coba akan diterapkan di dalam lingkungan universitas yang memiliki kompleksitas keuangan yang tinggi juga.

Di sisi lain, pengembangan sebuah sistem informasi modern menghadapi tantangan ketika kebutuhan untuk meningkatkan skala aplikasi juga ikut membesar. Arsitektur monolitik yang sering digunakan menemui kesulitan ketika ingin dikembangkan lebih lanjut. Oleh karena itu, muncul pendekatan untuk menggunakan *micorservices*. Penggunaan *microservices* pada awalnya memang mendapatkan tantangan terutama ketika banyaknya *services* yang akan saling berkomunikasi satu dengan yang lainnya. Selain itu, sisi keamanan dan juga *load balancing* juga perlu diperhatikan agar sistem

dapat berjalan dengan baik. Oleh karena itu, penelitian yang dilakukan oleh Ramaswamy Chandramouli telah menjelaskan dan memberikan panduan teknis untuk mengatasi hal tersebut [10].

Begitu juga dengan penelitian yang dilakukan oleh Wirawan juga mendukung bahwa penggunaan Docker juga akan membantu jalannya sebuah aplikasi. Dalam penelitiannya, Wirawan memanfaatkan Docker Container untuk menjalankan aplikasi *website* cuti yang dibuatnya. Penerapan Docker terbukti membantu jalannya aplikasi serta proses pemeliharaan aplikasinya [11].

Berdasarkan penelitian terdahulu, dapat disimpulkan bahwa sistem informasi berkontribusi cukup signifikan terhadap pengelolaan keuangan sebuah perusahaan, sementara adanya teknologi *microservices* dan Docker memberikan solusi teknis untuk mendukung pengembangan sistem yang lebih kompleks dan mempermudah proses pengembangan selanjutnya. Oleh karena itu, penelitian ini akan menerapkan arsitektur *microservices* berbasis Docker pada sistem manajemen keuangan dan akuntansi di dalam lingkup universitas.

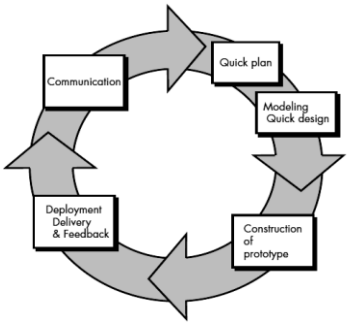
## 2. METODOLOGI PENELITIAN

Penelitian ini diawali dengan analisis kebutuhan melalui wawancara serta observasi langsung terhadap segala kegiatan transaksi keuangan dan penulisan jurnal yang terjadi di UKSW. Wawancara dilakukan terhadap para operator dari Direktorat Anggaran dan Keuangan (DAKU) serta unit-unit yang berinteraksi langsung dengan aplikasi yang sebelumnya sudah ada. Hasil analisis menunjukkan bahwa aplikasi sebelumnya memiliki beberapa kekurangan antara lain: 1) belum dapat menghasilkan laporan maupun penulisan jurnal secara tepat dan *realtime*, 2) masih seringnya terjadi kesalahan seperti duplikat kode transaksi dan kesalahan penulisan jurnal, dan 3) beberapa SOP yang mewajibkan DAKU/Unit untuk melakukan perubahan data yang tidak sesuai dengan kaidah pemakaian aplikasi (mengubah ke dalam *database* secara langsung) tanpa melalui aplikasi.

Setelah analisis kebutuhan, dilakukan pengumpulan bahan penelitian melalui studi literatur terkait sistem informasi keuangan, *microservices*, dan Docker. Selanjutnya dilakukan perancangan aplikasi menggunakan UML, Entity Relationship Diagram (ERD), serta rancangan arsitektur sistem berbasis *microservices*. Tahap berikutnya adalah pembangunan aplikasi berdasarkan rancangan yang telah dibuat. Implementasi dilakukan secara bertahap dan hasilnya dievaluasi untuk kemudian ditarik kesimpulan.

### Metode Pengembangan Sistem

Penelitian ini akan menggunakan Metode Prototyping, yaitu pendekatan yang menekankan pada pembangunan sistem secara iteratif dan berulang hingga memenuhi kebutuhan pengguna. Metode ini dipilih karena sesuai dengan konteks penelitian, di mana sistem harus mampu menyesuaikan diri dengan perubahan kebijakan maupun SOP yang sering terjadi di UKSW.

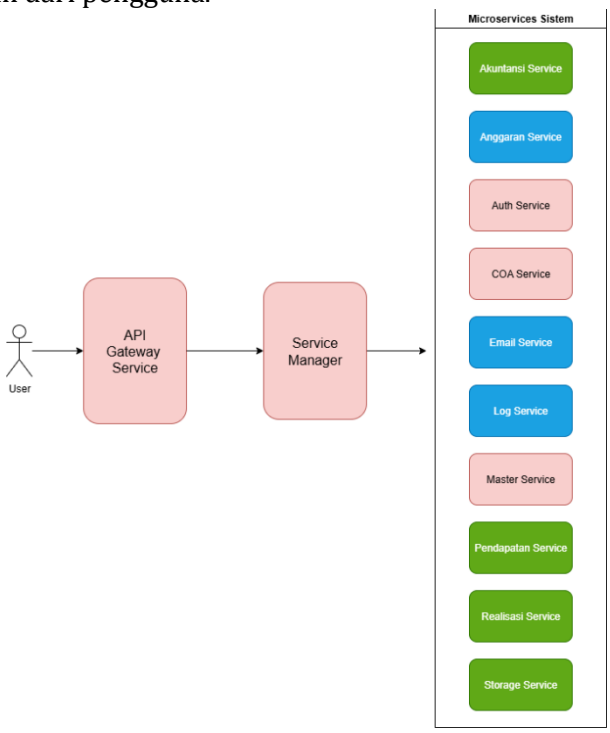



**Gambar 1. Metode Prototyping [12]**

Gambar 1 menunjukkan bagaimana tahapan-tahapan pengembangan aplikasi dengan Metode Prototyping. Metode Prototyping dalam penelitian ini terdiri dari beberapa tahapan. Tahap pertama adalah Quick Plan, yaitu identifikasi kebutuhan pengguna berdasarkan hasil wawancara dan observasi. Selanjutnya dilakukan Modeling Quick Design, berupa rancangan awal sistem meliputi ERD, arsitektur sistem, serta pembagian layanan *microservices*. Setelah itu masuk ke tahap Construction of Prototype, yakni pembangunan sistem berdasarkan desain awal. Pada penelitian ini dilakukan tiga kali iterasi prototipe karena pengembangan masih berlangsung. Tahap terakhir adalah Deployment, Delivery, and Feedback, di mana prototipe diuji coba dan dievaluasi oleh pengguna (operator DAKU serta unit-unit terkait). Masukan dan saran dari pengguna menjadi dasar untuk pengembangan prototipe selanjutnya.

## 3. HASIL DAN PEMBAHASAN

Bab ini akan menyajikan hasil penelitian yang berupa proses pengembangan prototipe sistem manajemen keuangan dan akuntansi menggunakan arsitektur *microservices* dengan Docker. Hasil yang dipaparkan mencakup gambaran umum arsitektur sistem, rincian layanan (*services*) yang dikembangkan, serta tahapan iterasi prototipe hingga iterasi ketiga. Selain penyajian hasil, bab ini juga membahas analisis terhadap kelebihan, kekurangan, serta evaluasi dari sistem yang telah dibangun berdasarkan masukan dari pengguna.




**Gambar 2. Gambaran Umum Micorservices dari Sistem**

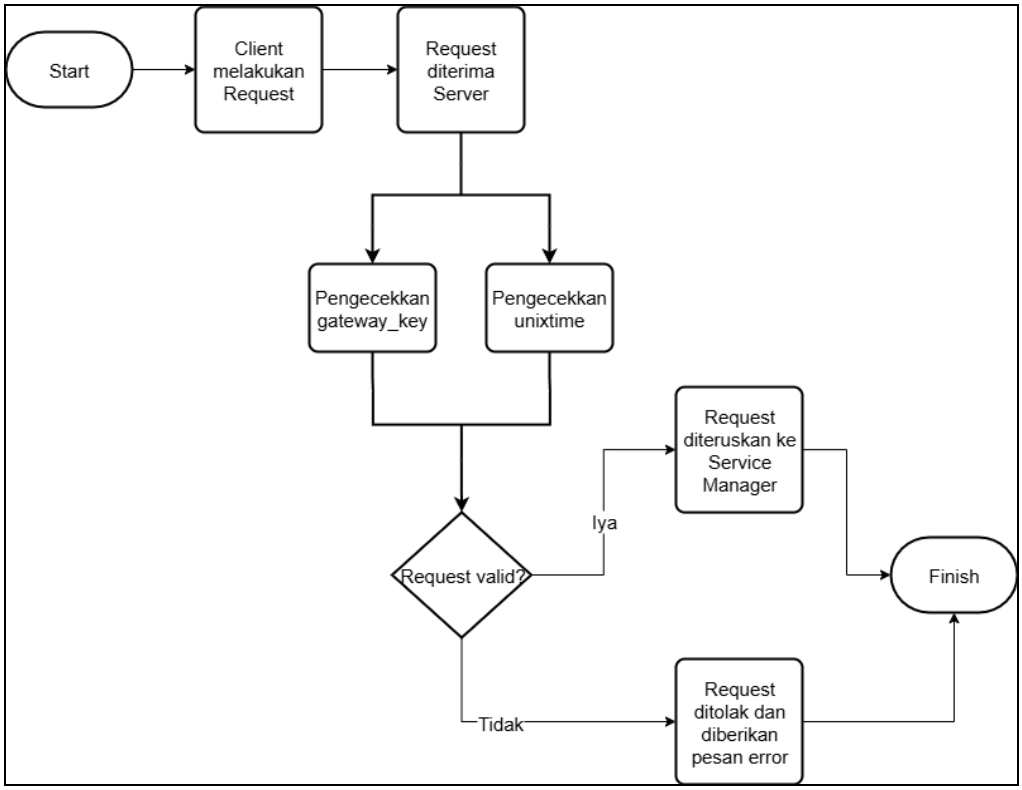
Seperti yang sudah dijelaskan sebelumnya, penelitian ini akan membahas proses pengembangan prototipe sampai iterasi ketiga saja. Dimana iterasi pertama prototipe sistem *backend* aplikasi akan berfokus kepada Api Gateway Service, Auth Service, COA Service, Master Service, dan Service Manager. Iterasi pertama dapat dilihat di Gambar 2 yang ditandai dengan warna merah. Iterasi kedua akan memperbaiki iterasi pertama dan juga menambahkan tiga *service* lagi yaitu Anggaran Service, Email Service, dan Log Service (pada Gambar 2 ditandai dengan warna biru). Terakhir, iterasi ketiga akan memperbaiki iterasi kedua dan juga menambahkan empat *service* lagi yaitu, Akuntansi Service,

Pendapatan Service, Realisasi Service, dan juga Storage Service (ditandai dengan warna hijau).

Di setiap *service* yang dibuat itu sendiri juga menerapkan konsep Model-Controller (tanpa View) [13] untuk memudahkan pengembangan sistem. Konsep ini sebenarnya merupakan sebuah konsep pemrograman yang sudah lazim digunakan dalam pengembangan sistem. Intinya, semua data yang ada di dalam *database* akan dibentuk menjadi sebuah "model" dan pengaksesan datanya semua akan dilakukan melalui *controller*. Secara garis besar, *service* yang dibangun ini akan memiliki fungsi dan kegunaannya masing-masing. Berikut ini adalah keterangan untuk setiap *service* yang dibangun:

### 3.1. API Gateway Service

API Gateway Service adalah sebuah *service* yang menjadi pintu gerbang untuk semua *request* dari *frontend* aplikasi. Dalam *service* ini tidak ada *database* yang digunakan dan hanya ada proses pengamanan untuk setiap *request* yang dilakukan dari *frontend*.




**Gambar 3. Flowchart Request yang terjadi dalam API Gateway Service**

Berdasarkan Gambar 3, dapat dilihat bagaimana setiap *request* yang diterima akan diperiksa terlebih dahulu apakah itu merupakan *request* yang dikenal dan valid atau tidak. Jika memang tidak valid, maka *request* tersebut akan langsung ditolak oleh sistem. Sebaliknya, jika *request* yang dilakukan memang *request* yang dikenal dan valid maka akan diproses sesuai dengan kebutuhannya.

Tahap pengecekan bahwa request yang dilakukan valid adalah dengan mencocokkan dua kunci, yaitu "gateway_key" dan "unixtime". Kedua kunci ini merupakan sebuah String unik yang akan dienkripsi dengan algoritma HMAC untuk kemudian hasilnya dicocokkan dengan *cipher text* yang telah tersimpan di dalam server. Jika keduanya cocok, maka *request* yang dilakukan merupakan *request* yang dikenal oleh sistem. Implementasi dari ini dapat dilihat pada Kode 1.

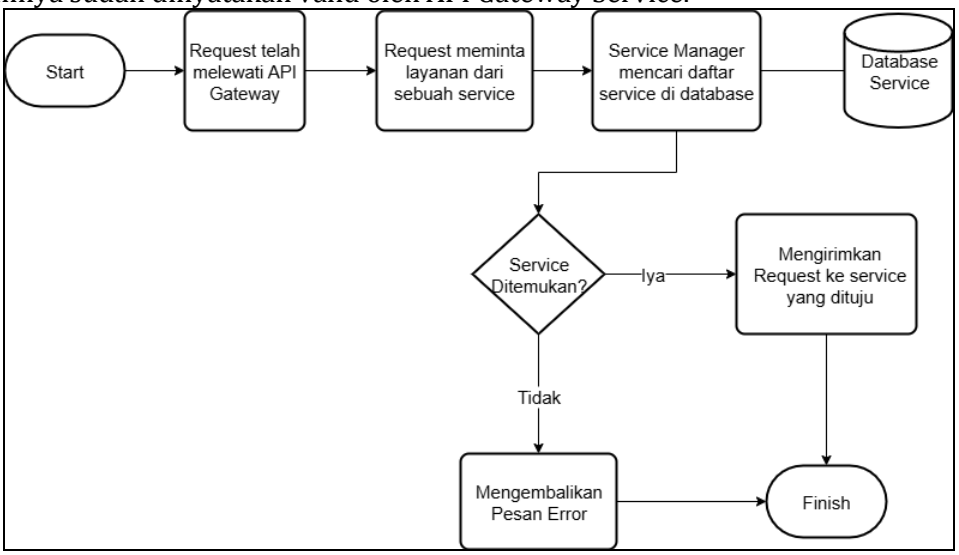
**Kode 1. Potongan Kode Program untuk Melihat Sebuah Request Valid atau Tidak**

```
if (req.headers["gateway-key"] && req.headers["unixtime"]) {
  gateway_key = req.headers["gateway-key"];
  unixtime = req.headers["unixtime"];

  if (gateway_key != `${process.env.DEV_GATEWAY}`) {
    var resultServer = encryptHMAC(unixtime, process.env.API_GATEWAY);
    if (resultServer != gateway_key) {
      return resError(res, 400, `ApiKey Tidak Valid`);
    }
  }
  next();
} else {
  return resError(res, 400, `Header Tidak Lengkap`);
}
```

### 3.2. Service Manager

Service Manager adalah sebuah *service* yang memiliki tugas khusus menyimpan daftar alamat serta menunjukkan *service* apa yang akan menangani setiap *request*. Cara kerjanya mirip dengan DNS Server dimana *service* ini akan menunjukkan alamat *service* apa yang terdaftar di dalam sistem. Sebuah *request* dapat masuk ke dalam *service* ini jika sebelumnya sudah dinyatakan valid oleh API Gateway Service.



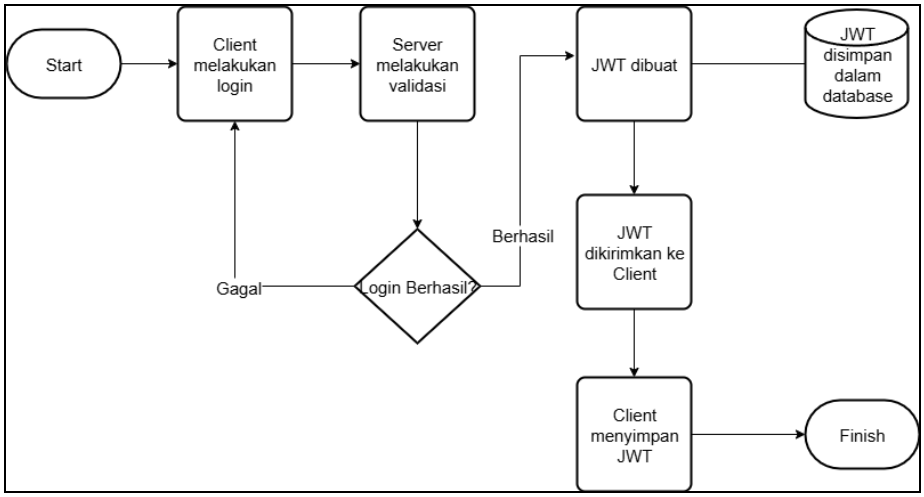

**Gambar 4. Flowchart Request yang terjadi dalam Service Manager**

Gambar 4 menjelaskan bagaimana alur sebuah *request* masuk ke dalam Service Manager dimana *request* tadi akan dicarikan alamat *service* mana yang benar-benar dibutuhkan. Database pada Service Manager hanya terdiri dua buah tabel yang menyimpan data "services" dan "service_path". Data "services" merupakan daftar *service* yang tersedia dalam sistem dan "service_path" adalah daftar alamat dari *service* yang sudah terdaftar.

### 3.3. Auth Service

Auth Service merupakan service yang berisi tentang segala sesuatu yang berkaitan dengan proses autentikasi. Fitur-fitur seperti *login, logout, forget password, reset password*, dan pendaftaran akun semuanya dilakukan pada *service* ini.

Di dalam service ini hanya ada satu tabel Users yang digunakan untuk menyimpan data pemakai dari sistem ini. Salah satu kolom digunakan untuk menyimpan *token* yang merupakan *token* hasil dari implementasi JSON Web Token (JWT). Secara detail, implementasi JWT ini dapat dilihat di Gambar 5.

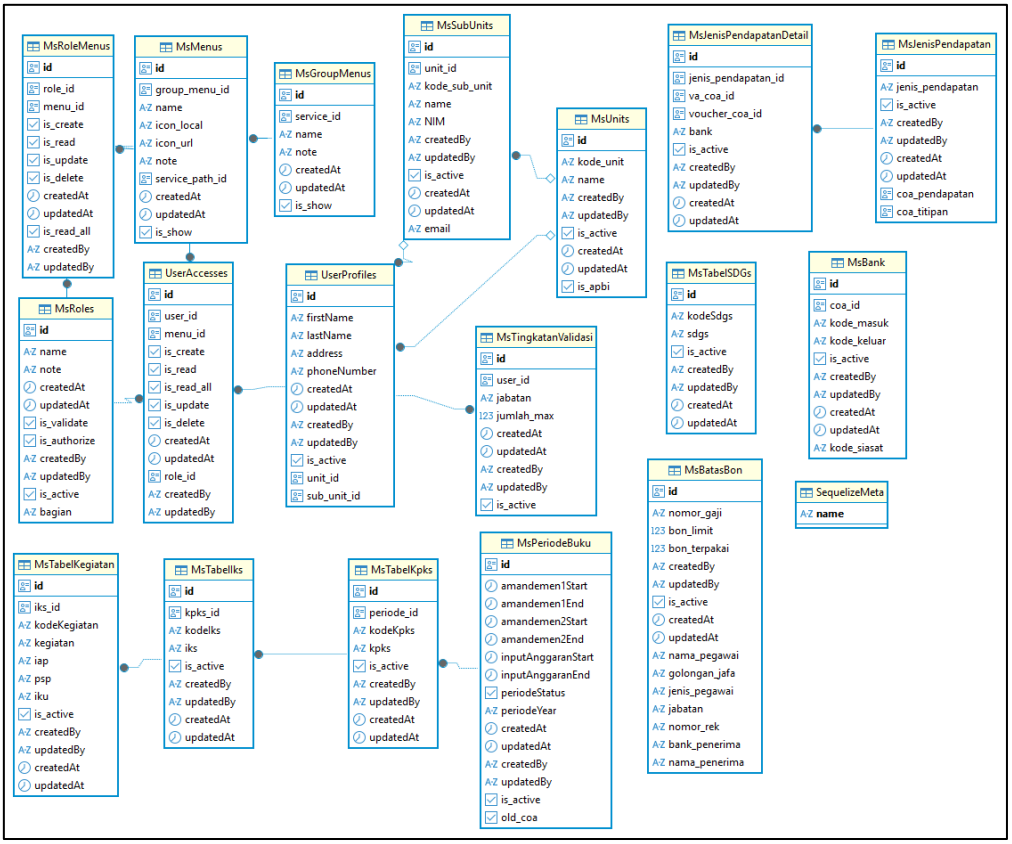



**Gambar 5. Implementasi JWT Token**

Gambar 5 menjelaskan tentang tahapan bagaimana sebuah JWT ini dibuat. Pembuatan JWT dilakukan agar sistem dapat mengenali user tanpa harus meminta user untuk melakukan *login* setiap kali menggunakan aplikasi. Untuk meningkatkan keamanan sistem, JWT ini juga akan menerapkan *expiry date* setiap jamnya.

### 3.4. Master Service

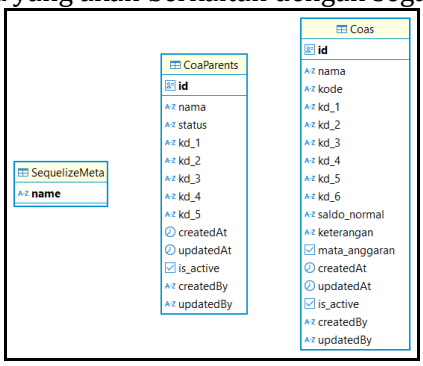
Master Service adalah service yang digunakan untuk melayani hal-hal yang berkaitan dengan master data yang diperlukan oleh sistem. Master data yang dimaksud adalah User, Roles, Bank, Batas Bon, Menu, Jenis Pendapatan, Sub Unit, Anggaran, dan Periode Buku.




**Gambar 6. ERD Master Service**

### 3.5. COA Service

COA Service adalah *service* yang akan melayani segala permintaan tentang akun-akun mata anggaran yang digunakan di dalam pencatatan akuntansi. Secara umum, *service* ini yang bertanggung jawab untuk menyimpan serta menyediakan layanan terkait akun-akun mata anggaran apa saja yang akan berkaitan dengan segala transaksi yang terjadi.




**Gambar 7. ERD dari COA Service**

Setiap *service* yang telah selesai dibuat akan dimasukkan ke dalam Docker Container. Pembuatan Docker Container sendiri cukup dengan membuat dua buah berkas konfigurasi yaitu Dockerfile dan docker-compose.yml. Kedua berkas ini akan memberikan perintah kepada Docker untuk bagaimana menjalankan *container* yang akan dibuat. Berikut ini adalah potongan kode konfigurasi yang dibuat:

**Kode 2. Potongan Kode Dalam docker-compose.yml**

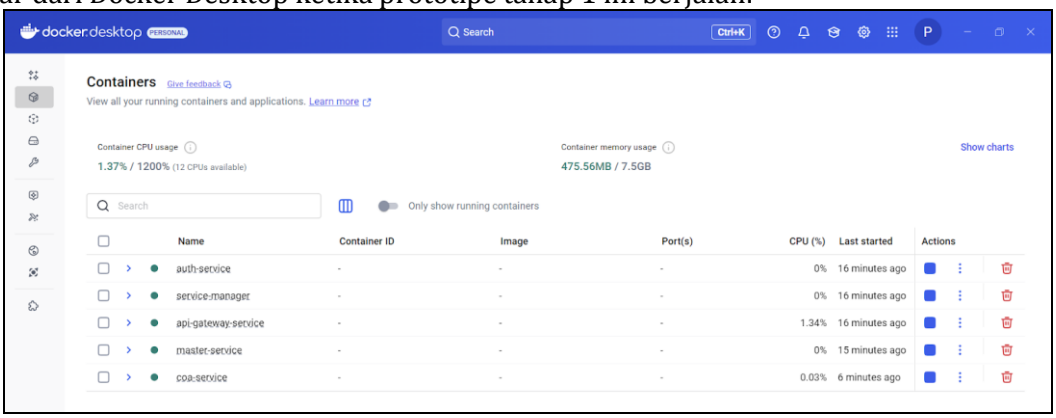
```
services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
      target: ${NODE_ENV}
    image: "${CONTAINER_NAME}:1.0"
    container_name: ${CONTAINER_NAME}
    environment:
      - NODE_ENV=${NODE_ENV}
    depends_on:
      - db
    volumes:
      - ./:/home/node/app
    networks:
      - siangkasa-net
    restart: always
    user: root
    command: sh -c 'npm install && npx sequelize-cli db:migrate && if [ "$NODE_ENV" =
"development" ]; then npm run start; else pm2-runtime bin/www; fi'
  db:
    image: postgres:16
    container_name: ${DB_DATABASE}
    ports:
      - "${DB_PORT_PUBLIC}:${DB_PORT}" # port public untuk remote db
    environment:
      POSTGRES_USER: ${DB_USERNAME}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_DATABASE}
      PGPORT: ${DB_PORT}
      TZ: Asia/Jakarta
    networks:
      - siangkasa-net
    volumes:
      - ./postgres-data:/var/lib/postgresql/data
    restart: always
networks:
  siangkasa-net:
    external: true
```

**Kode 3. Potongan Kode dalam Dockerfile**

```
FROM node:20.13-alpine as base
RUN mkdir -p /home/node/app/node_modules && chown -R node:node /home/node/app
WORKDIR /home/node/app
COPY . .
RUN npm install
FROM base as development
RUN npm install -g nodemon
USER node
COPY --chown=node:node . .
FROM base as production
RUN npm install -g pm2
USER node
COPY --chown=node:node . .
CMD ["pm2-runtime", "bin/www"]
```

Kode 2 dan Kode 3 adalah proses implementasi dari *service* yang sudah dibuat akan dimasukkan ke dalam Docker Container. Kode 2 adalah konfigurasi yang digunakan ketika Docker Container berjalan. Di dalam kode dapat dilihat bahwa *database* apa yang akan digunakan, *username & password* untuk mengakses *database, port* berapa yang digunakan, dan juga jaringan apa yang akan digunakan. Kode 3 adalah perintah-perintah yang dieksekusi ketika Docker Container akan dijalankan. Berhubung *service* yang digunakan menggunakan NodeJS, maka Docker Container yang dibuat juga akan menjalankan perintah-perintah untuk menjalankan *service* dengan perintah-perintah yang biasa digunakan oleh NodeJS.

Jika kedua kode program tersebut berhasil dieksekusi, maka terbentuklah lima buah Docker Container yang akan berjalan secara paralel. Berikut ini adalah tangkapan layar dari Docker Desktop ketika prototipe tahap 1 ini berjalan:




**Gambar 8. Tampilan Docker Container ketika Prototipe 1 Berjalan**

Setelah selesai melakukan pembuatan sistemnya, dilakukanlah pengujian prototipe pertama dengan menggunakan Black Box Testing. Black Box Testing sendiri adalah sebuah pengujian yang dilakukan oleh pengembang aplikasinya untuk menguji apakah sistem yang telah dibuat ini benar-benar berjalan sesuai dengan harapan.

**Tabel 1. Hasil Pengujian Prototipe Tahap 1**

| Skenario Pengujian | Hasil yang Diharapkan | Hasil Pengujian |
|---|---|---|
| **API Gateway Service** | | |
| Melakukan *request* ke dalam sistem tanpa menyertakan kunci apapun | *Request* ditolak | *Request* ditolak |
| Melakukan *request* ke dalam sistem dengan menyertakan kunci "*gateway_key*" dan "*unixtime*" yang salah | *Request* ditolak | *Request* ditolak |
| Melakukan *request* ke dalam sistem dengan | *Request* diterima | *Request* diterima |

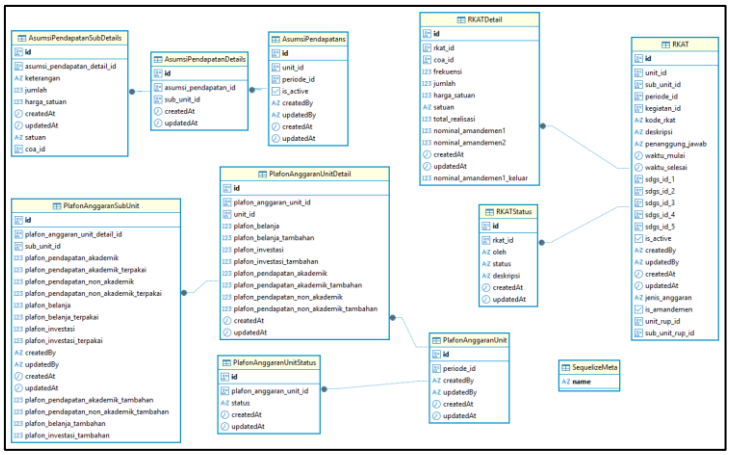
| | | |
|---|---|---|
| menyertakan kunci "*gateway_key*" dan "*unixtime*" yang benar | dan diteruskan | dan diteruskan |
| **Service Manager** | | |
| Melakukan *request* dengan alamat yang tidak terdaftar | Gagal | Gagal |
| Melakukan *request* dengan alamat yang sudah terdaftar | Sistem meneruskan *request* ke *service* yang dituju | Sistem meneruskan *request* ke *service* yang dituju |
| **Auth Service** | | |
| Melakukan login dengan akun yang belum terdaftar | Gagal | Gagal |
| Melakukan login dengan akun yang sudah terdaftar tetapi dengan kombinasi username dan password yang salah | Gagal | Gagal |
| Melakukan login dengan akun yang sudah terdaftar tetapi dengan kombinasi username dan password yang benar | Berhasil | Berhasil |
| Jika login berhasil, JWT berhasil dibuat | Sistem menyimpan *token* ke *database* | Sistem menyimpan *token* ke database |
| **Master Service** | | |
| Create, Read, Update, Delete (CRUD) Data User | Berhasil | Berhasil |
| CRUD Data Roles | Berhasil | Berhasil |
| CRUD Data Bank | Berhasil | Berhasil |
| CRUD Data Batas Bon | Berhasil | Berhasil |
| CRUD Data Menu | Berhasil | Berhasil |
| CRUD Data Jenis Pendapatan | Berhasil | Berhasil |
| CRUD Data Sub Unit | Berhasil | Berhasil |
| CRUD Data Periode Buku | Berhasil | Berhasil |
| **COA Service** | | |
| Membuat Kode Akun baru | Berhasil | Berhasil |
| Mengubah Kode Akun yang sudah ada | Berhasil | Berhasil |
| Menonaktifkan Kode Akun yang sudah tidak terpakai | Berhasil | Berhasil |
| Membaca Kode Akun yang telah tersimpan dalam database | Berhasil | Berhasil |

Berdasarkan Tabel 1 didapatkan bahwa fungsi-fungsi yang dibuat pada prototipe tahap 1 sudah memenuhi target yang diinginkan. Selanjutnya pembuatan sistem akan dilanjutkan ke tahap kedua. Prototipe yang kedua adalah menambah tiga buah *service* baru, yaitu Anggaran Service, Email Service dan Log Service.

### 3.6. Anggaran Service

Anggaran Service adalah *service* yang digunakan untuk menyediakan segala hal yang berkaitan dengan kegiatan penganggaran dari universitas, mulai dari penyusunan Rencana Kerja dan Anggaran Tahunan (RKAT) Unit-Unit, Validasi RKAT, Penetapan Pagu Anggaran, dan Daftar Anggaran yang dapat digunakan Unit nantinya.

Gambar 9 menjelaskan tentang ERD dari Anggaran Service dimana fungsi utama dari *database* yang dibuat adalah penyimpanan daftar RKAT dari seluruh unit terkait. Daftar RKAT ini yang nantinya akan digunakan sebagai dasar dari proses pencairan/realisasi dari anggaran. Selain itu, selama proses pembuatan Anggaran, user dengan *roles* tertentu juga dapat terus memantau proses pembuatan anggaran dari tahap pengajuan sampai disahkannya setiap kegiatan yang diajukan oleh unit-unit.




**Gambar 9. ERD Anggaran Service**

### 3.7. Email Service

Email Service adalah *service* yang bertugas untuk memberikan notifikasi berupa email kepada pengguna nantinya. Salah satu fitur yang akan dibahas adalah melakukan Reset Password. Fitur ini adalah fitur yang akan digunakan User untuk melakukan pengaturan ulang *password* yang akan dikirimkan melalui email ketika User lupa dengan *password* yang mereka gunakan. Pada *service* ini tidak ada *database* yang digunakan karena *service* ini hanya bertugas untuk melakukan perintah pengiriman email berdasarkan data-data yang didapatkan dari Auth Service.

### 3.8. Log Service

Log Service adalah *service* yang bertugas untuk menuliskan log atau semua hal yang terjadi di dalam *service* yang sedang berjalan. Sama seperti Email Service sebelumnya, *service* ini juga tidak menggunakan database untuk menyimpan data. Semua log yang dibuat akan disimpan ke dalam sebuah file.

Setelah semua *service* sudah siap, maka kode pembuatan Docker Container (seperti yang tercantum dalam Kode 2 dan Kode 3) dijalankan. Jika tidak ada *error*, maka saat ini sistem telah mempunyai delapan *service* yang berjalan bersamaan.

Tidak berbeda jauh dengan prototipe di tahap 1, di tahap kedua ini sistem juga diuji dengan metode pengujian Black Box.

**Tabel 2. Hasil Pengujian Prototipe Tahap 2**

| Skenario Pengujian | Hasil yang Diharapkan | Hasil Pengujian |
|---|---|---|
| **Anggaran Service** | | |
| Sub Unit dapat mengajukan kegiatan | Berhasil | Berhasil |
| Unit memvalidasi semua kegiatan yang diajukan Sub-Unit | Berhasil | Berhasil |
| Pimpinan Universitas dapat memvalidasi semua RKAT Unit | Berhasil | Berhasil |
| Pimpinan Universitas dapat mengubah RKAT Unit | Berhasil | Berhasil |
| Pimpinan Universitas dapat menolak RKAT Unit | Berhasil | Berhasil |
| Pimpinan Universitas menentukan Pagu Anggaran setiap Unit | Berhasil | Berhasil |
| RKAT dapat diambil datanya melalui *service* lain | Berhasil | Berhasil |
| Detail dari setiap RKAT dapat diambil datanya melalui *service* lain | Berhasil | Berhasil |
| **Email Service** | | |
| Sistem menolak jika user memasukkan alamat email yang tidak valid | Berhasil | Berhasil |
| Sistem mengirimkan email kepada user jika melakukan *reset password* | Berhasil | Berhasil |

| **Log Service** | | |
|---|---|---|
| Sistem mencatat log dengan benar | Berhasil | Berhasil |
| Sistem menyimpan log ke dalam sistem | Berhasil | Berhasil |

Berdasarkan Tabel 2, didapatkan bahwa pada Prototipe tahap 2 ini sudah berjalan sesuai dengan hasil yang diharapkan. Terlihat bahwa fungsi-fungsi sudah berjalan sesuai dengan fungsinya.

Selanjutnya, dibuatlah prototipe tahap 3, dengan menambah lagi *service* yaitu Akuntansi Service, Pendapatan Service, Realisasi Service, dan Storage Service. Prototipe tahap 3 ini merupakan kumpulan *service* yang berisi tentang inti dari sistem secara keseluruhan. Prototipe tahap 1 dan tahap 2 dapat dikatakan sebagai *service* yang bersifat umum atau hampir di semua aplikasi akan dibuat *service* itu, sedangkan di prototipe tahap 3 ini lebih spesifik ke tujuan khusus dari sistem yang akan dibuat. Detail dari setiap *service* yang dibangun pada prototipe tahap 3 ini dapat dijelaskan sebagai berikut:

### 3.9. Akuntansi Service

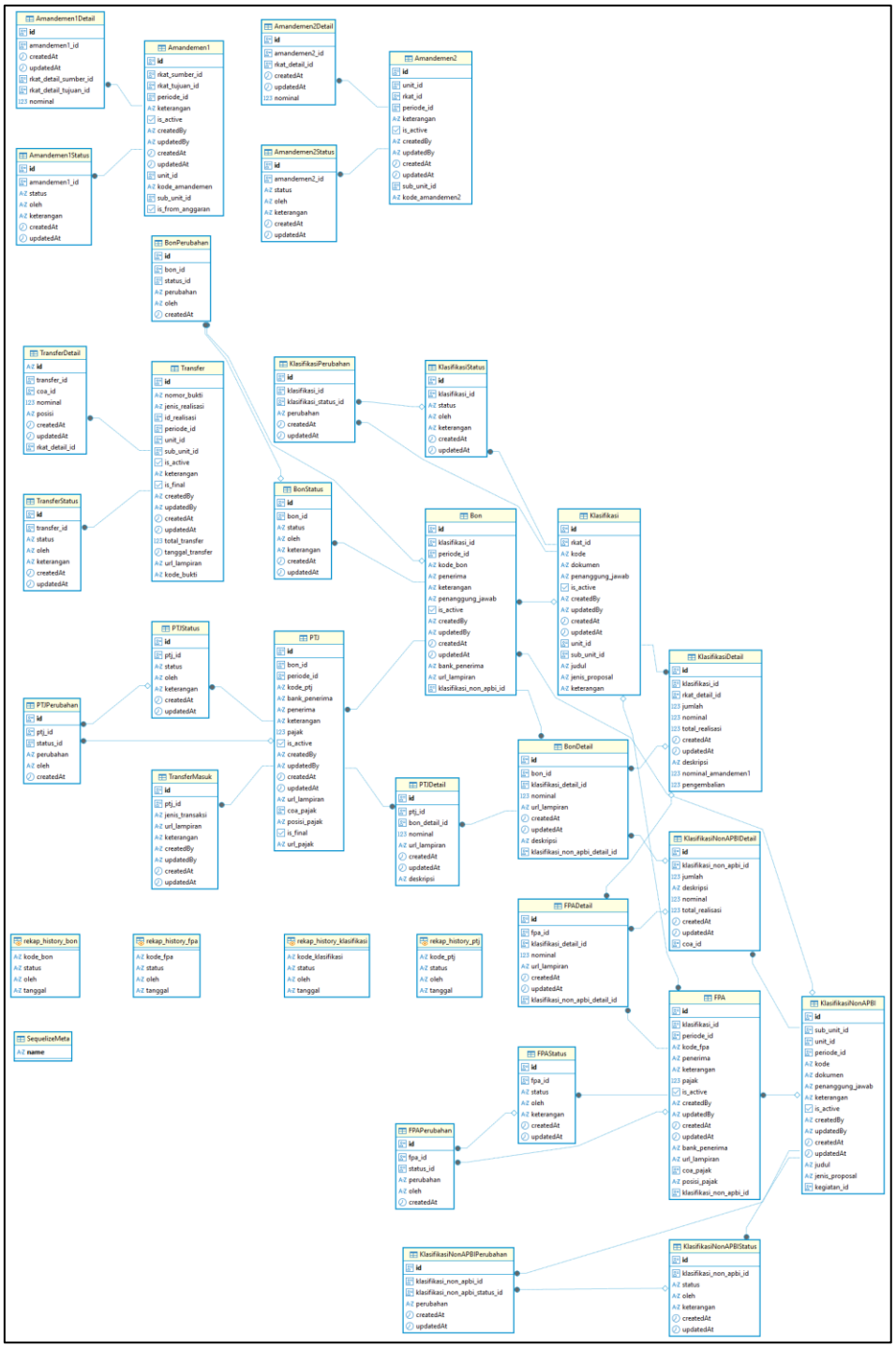
Akuntansi Service adalah *service* yang mencatat segala hal yang berkaitan dengan akuntansi. Dari setiap transaksi yang terjadi, sistem akan mencatat akun apa yang tergolong kategori Debit dan juga pasangannya yang tergolong kategori Kredit. Semuanya itu didasarkan pada dokumen Pedoman Akuntansi yang menjadi dasar di dalam setiap transaksi di universitas. Akuntansi Service juga membutuhkan koneksi dengan *service* lainnya untuk mengakses data-data yang dibutuhkan dari COA atau Master Data.

### 3.10. Pendapatan Service

Pendapatan Service adalah *service* yang bertanggung jawab terhadap pencatatan transaksi yang tergolong sebagai pendapatan universitas. Saat ini, UKSW menggunakan sistem lainnya untuk menangani transaksi-transaksi tersebut, seperti contohnya pembayaran uang pendaftaran dan uang kuliah mahasiswa, pembayaran uang wisuda mahasiswa, dana sponsor, hibah-hibah dari institusi eksternal, dan sebagainya. Semua transaksi itu nanti akan bermuara dan dicatat ke dalam sistem ini.

### 3.11. Realisasi Service

Realisasi Service adalah *service* yang bertanggung jawab terhadap proses realisasi anggaran yang sudah tersimpan. Sampai saat penelitian ini dilakukan, *service* ini merupakan *service* yang paling "sibuk" karena memiliki beban kerja yang cukup tinggi dengan tabel dalam *database* yang paling banyak jika dibandingkan dengan *service* lainnya. Untuk ERD dari Realisasi Service dapat dilihat pada Gambar 10.




**Gambar 10. ERD Realisasi Service**

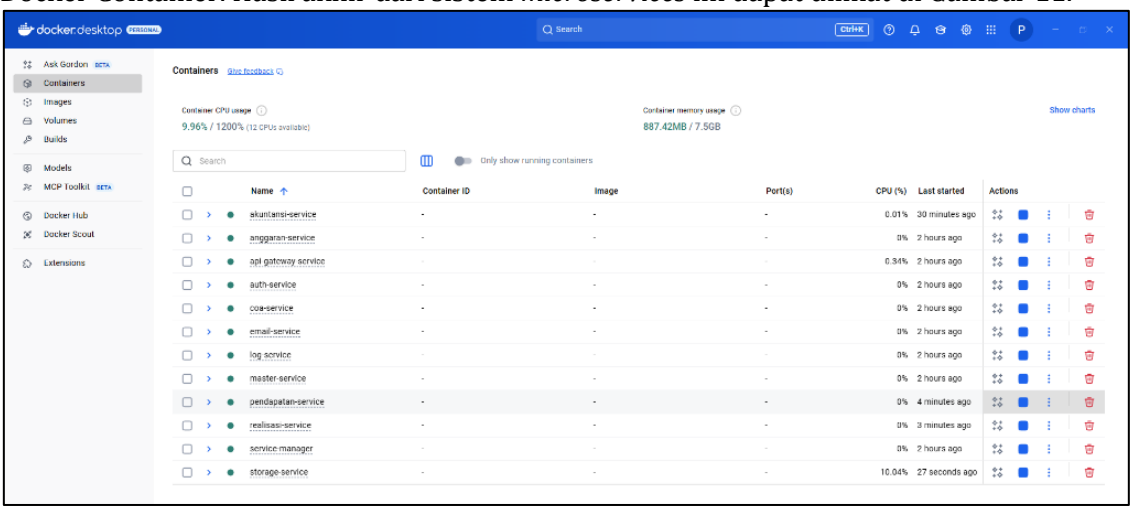
Proses realisasi anggaran di UKSW dapat dilakukan dalam tiga jenis, yaitu pengajuan Bon diikuti dengan pertanggungjawabannya (PTJ) dan juga Formulir Pencairan Anggaran (FPA). Bon merupakan proses pencairan yang digunakan oleh Unit untuk "meminta" terlebih dahulu dana yang akan direalisasikan, setelah selesai digunakan Unit nantinya wajib melakukan pertanggungjawaban atas Bon yang diminta. Sedangkan realisasi FPA berbalikan dengan Bon-PTJ dimana proses realisasi akan dilakukan setelah Unit menyelesaikan kegiatannya terlebih dahulu. Ketiga proses ini sudah masuk ke dalam sistem yang dibangun.

Selain itu, selama proses realisasi anggaran juga ada validasi berjenjang yang digunakan oleh universitas untuk melakukan kontrol terhadap kegiatan-kegiatan yang berjalan. Validasi ini dilakukan oleh Pimpinan Unit, Tim Validasi & Verifikasi Dokumen (VVD) DAKU, Pimpinan DAKU, dan juga Pimpinan Universitas. Jika pada proses realisasi anggaran ditemukan kesalahan dan perlu ada revisi maka salah satu dari tim validator akan memberikan notifikasi kepada Unit yang mengajukan. Tim validator juga berhak menolak pengajuan realisasi anggaran jika memang dirasa kurang sesuai dengan aturan yang berlaku. Semua proses realisasi anggaran beserta statusnya akan dicatat ke dalam sistem.

### 3.12. Storage Service

Storage Service adalah *service* yang digunakan untuk menangani proses unduh maupun unggah dokumen-dokumen pendukung selama proses realisasi anggaran. Setiap dokumen yang diunggah ke dalam sistem nanti akan diubah terlebih dahulu namanya, kemudian disimpan ke dalam sebuah folder khusus. Nama baru dan alamat tempat penyimpanan akan disimpan ke dalam *database* agar memudahkan proses pencarian data nantinya.

Ketika semua *service* sudah siap untuk dijalankan, maka sama seperti di prototipe tahap-tahap sebelumnya, semua *service* di tahap 3 ini juga akan dimasukkan ke dalam Docker Container. Hasil akhir dari sistem *microservices* ini dapat dilihat di Gambar 11.




**Gambar 11. Prototipe Tahap 3 Ketika Sudah Dijalankan**

Terakhir, prototipe tahap 3 juga diuji dengan menggunakan metode Black Box Testing.

**Tabel 3. Hasil Pengujian Prototipe Tahap 3**

| Skenario Pengujian | Hasil yang Diharapkan | Hasil Pengujian |
|---|---|---|
| **Akuntansi Service** | | |
| Pencatatan Debit & Kredit dalam transaksi Bon | Berhasil | Berhasil |
| Pencatatan Debit & Kredit dalam transaksi FPA | Berhasil | Berhasil |
| Pencatatan Debit & Kredit dalam transaksi PTJ | Berhasil | Berhasil |
| **Pendapatan Service** | | |
| Pencatatan pendapatan dari pembayaran uang kuliah mahasiswa | Berhasil | Berhasil |
| Pencatatan pendapatan dari pembayaran uang pendaftaran mahasiswa | Berhasil | Berhasil |
| Pencatatan pendapatan dari pembayaran uang wisuda mahasiswa | Berhasil | Berhasil |

| | | |
|---|---|---|
| Pencatatan pendapatan dari hibah-hibah eksternal | Berhasil | Berhasil |
| Pencatatan pendapatan dari sponsor | Berhasil | Berhasil |
| **Realisasi Service** | | |
| Unit mengajukan Bon | Berhasil | Berhasil |
| Unit melakukan PTJ | Berhasil | Berhasil |
| Unit mengajukan FPA | Berhasil | Berhasil |
| Pimpinan Unit melakukan validasi Bon | Berhasil | Berhasil |
| Pimpinan Unit meminta revisi Bon | Berhasil | Berhasil |
| Pimpinan Unit menolak pengajuan Bon | Berhasil | Berhasil |
| VVD melakukan validasi Bon | Berhasil | Berhasil |
| VVD meminta revisi Bon | Berhasil | Berhasil |
| VVD menolak pengajuan Bon | Berhasil | Berhasil |
| Pimpinan DAKU melakukan validasi Bon | Berhasil | Berhasil |
| Pimpinan DAKU meminta revisi Bon | Berhasil | Berhasil |
| Pimpinan DAKU menolak pengajuan Bon | Berhasil | Berhasil |
| Pimpinan Universitas melakukan validasi Bon | Berhasil | Berhasil |
| Pimpinan Universitas meminta revisi Bon | Berhasil | Berhasil |
| Pimpinan Universitas menolak pengajuan Bon | Berhasil | Berhasil |
| Pimpinan Unit melakukan validasi PTJ | Berhasil | Berhasil |
| Pimpinan Unit meminta revisi PTJ | Berhasil | Berhasil |
| Pimpinan Unit menolak pengajuan PTJ | Berhasil | Berhasil |
| VVD melakukan validasi PTJ | Berhasil | Berhasil |
| VVD meminta revisi PTJ | Berhasil | Berhasil |
| VVD menolak pengajuan PTJ | Berhasil | Berhasil |
| Pimpinan DAKU melakukan validasi PTJ | Berhasil | Berhasil |
| Pimpinan DAKU meminta revisi PTJ | Berhasil | Berhasil |
| Pimpinan DAKU menolak pengajuan PTJ | Berhasil | Berhasil |
| Pimpinan Universitas melakukan validasi PTJ | Berhasil | Berhasil |
| Pimpinan Universitas meminta revisi PTJ | Berhasil | Berhasil |
| Pimpinan Universitas menolak pengajuan PTJ | Berhasil | Berhasil |
| Pimpinan Unit melakukan validasi FPA | Berhasil | Berhasil |
| Pimpinan Unit meminta revisi FPA | Berhasil | Berhasil |
| Pimpinan Unit menolak pengajuan FPA | Berhasil | Berhasil |
| VVD melakukan validasi FPA | Berhasil | Berhasil |
| VVD meminta revisi FPA | Berhasil | Berhasil |
| VVD menolak pengajuan FPA | Berhasil | Berhasil |
| Pimpinan DAKU melakukan validasi FPA | Berhasil | Berhasil |
| Pimpinan DAKU meminta revisi FPA | Berhasil | Berhasil |
| Pimpinan DAKU menolak pengajuan FPA | Berhasil | Berhasil |
| Pimpinan Universitas melakukan validasi FPA | Berhasil | Berhasil |
| Pimpinan Universitas meminta revisi FPA | Berhasil | Berhasil |
| Pimpinan Universitas menolak pengajuan FPA | Berhasil | Berhasil |
| **Storage Service** | | |
| User dapat melakukan unggah data | Berhasil | Berhasil |
| User dapat melakukan unduh data | Berhasil | Berhasil |

Dari Tabel 1, Tabel 2, dan Tabel 3 menunjukkan bahwa sistem sudah berjalan sesuai dengan harapan. Sistem yang menerapkan konsep *microservices* dengan Docker Container dapat berjalan sesuai dengan harapan.

## 4. SIMPULAN

Dari pembahasan yang sudah dijabarkan, dapat diambil kesimpulan bahwa penerapan *microservices* memang benar-benar membantu pembuatan aplikasi keuangan

dan akuntansi dalam universitas. Memanfaatkan Docker juga menjadi pilihan yang tepat untuk menerapkan konsep *microservices* ini karena setiap service yang dibangun akan dapat berjalan secara independen dalam Docker Container yang telah disiapkan.

Saran pengembangan sistem selanjutnya adalah dibuatnya sebuah *frontend* aplikasi yang dapat mengakses sistem *backend* aplikasi yang sudah dibuat ini. Selain itu, jika dalam perkembangannya aplikasi membutuhkan *service* lain, maka jumlah *service* dapat ditambahkan juga.

## DAFTAR PUSTAKA

[1] *Universitas Kristen Satya Wacana*, Website Page, 28 Agustus 2025. Diakses: 28 Agustus 2025. [Daring]. Tersedia pada: https://uksw.edu/

[2] U. Syarif dan Pizaini, "Penerapan Event-Driven Microservices Pada Aplikasi Layanan Penerimaan Peserta Didik Baru," *JIPI J. Ilm. Penelit. Dan Pembelajaran Inform.*, vol. 7, no. 3, hlm. 745–756, Agu 2022, doi: 10.29100/jipi.v7i3.3067.

[3] D. H. Tinambunan, A. Baehaqi, R. P. Avrianto, dan R. E. Indrajit, "Microgen Implementation For Building Online Learning Management System with Microservices and GraphQL Generator Approach," *J. Tek. Inform. JUTIF*, vol. 4, no. 4, hlm. 967–976, Agu 2023, doi: https://doi.org/10.52436/1.jutif.2023.4.4.1313.

[4] Muhammad Rizki Alamsyah, H. Endah Wahanani, dan M. Idhom, "Penerapan Docker Untuk Membangun Infrastruktur Private Cloud Storage Berbasis IaaS," *J. Inform. Dan Sist. Inf.*, vol. 2, no. 2, hlm. 122–131, Jul 2021, doi: 10.33005/jifosi.v2i2.350.

[5] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, First Edition. Beijing Sebastopol, CA: O'Reilly Media, 2015.

[6] V. R. Gudelli, "Containerization Technologies: ECR and Docker for Microservices Architecture," *Int. J. Innov. Res. Eng. Multidiscip. Phys. Sci.*, vol. 11, no. 3, hlm. 1–10, 2023.

[7] F. Eyvazov, T. E. Ali, F. I. Ali, dan A. D. Zoltan, "Beyond Containers: Orchestrating Microservices with Minikube, Kubernetes, Docker, and Compose for Seamless Deployment and Scalability," dalam *2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, Noida, India: IEEE, Mar 2024, hlm. 1–6. doi: 10.1109/ICRITO61523.2024.10522382.

[8] Haria Saputri, Ujang Kusnaedi, dan Yandi Asmana, "Pengaruh Sistem Informasi Akuntansi Terhadap Kualitas Laporan Keuangan Perusahaan Jasa di Jakarta Utara," Mei 2023, doi: 10.5281/ZENODO.7932454.

[9] H. Riyadli, A. Arliyana, dan F. E. Saputra, "Rancang Bangun Sistem Informasi Keuangan Berbasis WEB," *J. Sains Komput. Dan Teknol. Inf.*, vol. 3, no. 1, hlm. 98–103, Nov 2020, doi: 10.33084/jsakti.v3i1.1770.

[10] R. Chandramouli dan Z. Butcher, "Building Secure Microservices-Based Applications Using Service-Mesh Architecture," National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-204A, Mei 2020. doi: 10.6028/NIST.SP.800-204a.

[11] A. A. Wirawan, "Penerapan Prototype Aplikasi Web E-Cuti Menggunakan Teknologi Docker Container (Studi Kasus: Universitas Teknologi Digital Indonesia)," Thesis, Universitas Teknologi Digital Indonesia, Yogyakarta, 2022. Diakses: 22 Agustus 2025. [Daring]. Tersedia pada: https://eprints.utdi.ac.id/9524/

[12] R. S. Pressman dan B. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed. McGraw Hill.

[13] M. Fowler, *Patterns of enterprise application architecture*, Nineteenth printing. dalam The Addison-Wesley Signature Series. Boston San Francisco New York Toronto Montreal London Munich Paris Madrid Capetown: Addison-Wesley, 2013.