CICES (*Cyberpreneurship Innovative and Creative Exact and Social Science*)
DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**

Implementasi Arsitektur Mikroservis dan Orkestrasi Kubernetes dengan Paradigma DDD pada Website Freelancing

Hafi Ihza Farhana*[1], Retno Mumpuni[2], Fawwaz Ali Akbar[3]
[1,2,3]Program Studi Informatika, Fakultas Ilmu Komputer, UPN "Veteran" Jawa Timur
E-mail: [*1]**hafiihza2002@gmail.com**, [2]retnomumpuni.if@upnjatim.ac.id,
[3]fawwaz_ali.fik@upnjatim.ac.id

Abstrak

Perkembangan pesat era digital yang dipicu oleh COVID-19 telah mengubah cara masyarakat memperoleh penghasilan, dengan semakin banyaknya orang yang beralih ke pekerjaan freelance. M-Knows Consulting merespons perubahan ini dengan menciptakan platform freelance berbasis website, yang dirancang untuk menjadi wadah bagi para freelancer Indonesia dalam mengembangkan karir mereka. Mengingat kompleksitas sistem dan tingginya interaksi dari berbagai pengguna, diperlukan pengembangan yang lebih modular untuk meningkatkan efisiensi, terutama dalam menangani tantangan proyek yang beragam. Oleh karena itu, arsitektur mikroservis dipilih sebagai solusi yang lebih tepat dibandingkan arsitektur monolitik. Perancangan arsitektur mikroservis ini melibatkan beberapa langkah penting, termasuk penerapan paradigma Domain-Driven Design (DDD) dengan prinsip bounded context untuk memisahkan domain bisnis secara jelas dan mengimplementasikan pendekatan multi-database yang sesuai dengan kebutuhan spesifik setiap layanan. Proses deployment akan dilakukan menggunakan Kubernetes untuk mengelola workload dari setiap mikroservis serta memastikan skalabilitas dan keandalan sistem. Dengan pendekatan ini, diharapkan pengembangan platform freelance berbasis website dapat berjalan lebih efisien dan cepat, sehingga segera dapat menyediakan layanan yang optimal bagi para freelancer di Indonesia.

Kata Kunci*—Mikroservis, DDD, Kubernetes, Bounded Context*

Abstract

The rapid development of the digital era triggered by COVID-19 has changed the way people earn income, with more and more people turning to freelance work. M-Knows Consulting responded to this change by creating a website-based freelance platform, designed to be a place for Indonesian freelancers to develop their careers. Given the complexity of the system and the high interaction of various users, a more modular development is needed to increase efficiency, especially in handling diverse project challenges. Therefore, a microservice architecture was chosen as a more appropriate solution than a monolithic architecture. The design of this microservice architecture involves several important steps, including the application of the Domain-Driven Design (DDD) paradigm with the principle of bounded context to clearly separate business domains and implement a multi-database approach that suits the specific needs of each service. The deployment process will be carried out using Kubernetes to manage the workload of each microservice and ensure system scalability and reliability. With this approach, it is hoped that the development of a website-based freelance platform can run more efficiently and quickly, so that it can immediately provide optimal services for freelancers in Indonesia.

Keywords*—Microservices, DDD, Kubernetes, Bounded Context*

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**

1. PENDAHULUAN

Era digital terus berkembang pesat seiring dengan perubahan global saat ini. Pandemi Covid-19 dalam beberapa tahun lalu juga masih memberikan dampak terhadap sektor industri yang mengakibatkan banyak masyarakat kehilangan pekerjaan dan mendorong mereka untuk menggunakan teknologi informasi secara tiba-tiba. Pada Februari 2021, sebanyak 19,10 juta orang di Indonesia yang berusia kerja terkena dampak Covid-19, dengan rincian 1,62 juta pengangguran, 0,65 juta bukan angkatan kerja, 1,11 juta kehilangan pekerjaan sementara, dan 15,72 juta mengalami pengurangan waktu kerja[1]. Hal ini mendorong masyarakat untuk bekerja secara informal dengan memanfaatkan teknologi yang ada.

Fenomena ini dikenal sebagai *gig economy*, sistem kerja yang melibatkan kontrak jangka pendek antara pekerja lepas dengan pihak yang membutuhkan jasa mereka[2]. Pekerjaan *gig* banyak ditemukan di sektor digital seperti pengembangan web, desain digital, dan lainnya. Pada tahun 2019, di Indonesia, jumlah pekerja *gig* sebagai pekerjaan utama berkisar antara 430 ribu hingga 2,3 juta orang, sekitar 0,3% hingga 1,7% dari total angkatan kerja. Di sektor non-transportasi, terdapat sekitar 1,1 juta pekerja *gig* yang tidak memiliki *platform* tetap untuk menyalurkan jasanya[3]. Perusahaan M-Knows Consulting, yang berfokus pada konsultasi dan bisnis, berusaha mengembangkan model bisnis untuk media *gig economy* berupa *freelance marketplace* berbasis web untuk menyediakan *platform* bagi masyarakat Indonesia dalam mempertemukan penyedia jasa dan pihak yang membutuhkan jasa tersebut.

Sistem ini cukup kompleks seperti autentikasi, otorisasi, notifikasi, manajemen pengguna, *gig*, percakapan, pemesanan, dan ulasan. Mengelola pengembangan hingga fase produksi untuk *marketplace freelance* akan sangat menantang jika menggunakan arsitektur monolitik. Arsitektur monolitik menyatukan tampilan, logika, dan akses data dalam satu aplikasi[4]. Semua layanan dijalankan dalam satu server, sehingga menambah fitur menjadi sulit karena semua komponen saling bergantung.

Berbeda dengan arsitektur monolitik, arsitektur mikroservis memungkinkan pembagian fitur menjadi beberapa layanan independen yang tidak saling tergantung. Jika salah satu layanan mengalami masalah, layanan lainnya tetap berfungsi[5]. Perancangan Arsitektur mikroservis biasanya menggunakan paradigma Domain Driven Design (DDD) untuk membagi kompleksitas dan mendefinisikan domain serta sub-domain yang diterjemahkan menjadi layanan mikro. Pada fase produksi, digunakan orkestrasi Kubernetes oleh Google pada 2014, untuk mengelola beban kerja, proses *deployment*, dan konfigurasi *container*[6]. Kubernetes berfungsi sebagai *platform* untuk secara otomatis mengelola *deployment*, *scaling*, dan pengelolaan aplikasi yang telah dilakukan proses kontainerisasi[7].

Dengan mengembangkan web *freelance marketplace*, dapat menyediakan *platform* yang membantu pekerja gig untuk menawarkan jasa mereka kepada konsumen yang membutuhkan. Ini memberikan media yang mempermudah mereka dalam menyalurkan layanan kepada pelanggan.

2. METODE PENELITIAN

Tahapan penelitian menjadi sebuah pedoman awal dalam proses pengembangan atau implementasi sebuah sistem pada web *freelancing* yang terurut dan konsisten.

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**

Gambar 1. Diagram Alur Penelitian

Pada gambar 1 menunjukan bahawa proses penelitian diawali dengan melakukan studi literatur dari berbagai sumber seperti artikel, jurnal, dan dokumentasi dari teknologi. Setelah itu, dilakukan analisis kebutuhan untuk melakukan identifikasi permasalahan yang akan diselesaikan, diikuti dengan diskusi untuk dalam menentukan teknologi yang akan digunakan dalam fase pengembangan. Selanjutnya, merupakan proses perancangan sistem yang mencakup desain dari domain bisnis berdasarkan ERD (Entity Relationship Diagram) yang dipecah menjadi beberapa subdomain bisnis menggunakan paradigma DDD (Domain-Driven Design) dengan prinsip bounded context. Perancangan ini meliputi desain basis data yang disesuaikan dengan domain bisnis, desain arsitektur mikroservis, dan desain gaya komunikasi antar mikroservis, serta desain UI/UX. Setelah sistem dirancang, dilakukan implementasi melalui proses coding dan deployment menggunakan Kubernetes versi lokal yang menggunakan *cluster* Minikube. Akhirnya, dilakukan pengujian untuk memastikan bahwa aplikasi *freelance* berbasis web berjalan sesuai dengan perancangan menggunakan Black Box.

2.1. Studi Literatur

Pada tahap ini, penulis melakukan proses pencarian dan pengumpulan informasi dan data, serta beberapa referensi atau informasi yang sesuai terhadap topik utama, yaitu arsitektur mikroservis, orkestrasi Kubernetes, dan perancangan arsitektur mikroservis berdasarkan paradigma DDD (Domain-Driven Design) dengan prinsip bounded context. Informasi ini diperoleh dari berbagai sumber, seperti jurnal dan artikel. Teori yang dikumpulkan kemudian diimplementasikan ke dalam sistem. Untuk memperjelas teori yang ada, dibuat diagram yang sesuai untuk membantu penulis dalam memahami informasi yang telah dikumpulkan.

2.2. Analisis Kebutuhan

2.2.1. Perangkat Lunak

Berikut merupakan kebutuhan perangkat lunak yang perlu dipersiapkan dalam implementasi sebuah sistem.

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**

Tabel 1. Kebutuhan Perangkat Lunak

No	**Kebutuhan Perangkat Lunak**	
1	Code Editor	Visual Studio Code v1.86.2
2	Runtime JavaScript	Node JS v18.17.1
3	Back-end Technology	Express JS v4.18.2, Express JS (TypeScript) v4.17.21
4	Front-end Technology	React JS v18.2.0, React JS (TypeScript) v18.2.58
5	Database & Cache	MongoDB, Redis (cache), PostgreSQL
6	Search Engine	Elasticsearch
7	Repository Manager	GitHub
8	Diagram Maker	Draw.io, DBdiagram.io
9	Virtual Machine	Hyper-V

2.2.2. Perangkat Keras

Berikut merupakan kebutuhan perangkat keras yang perlu dipersiapkan dalam implementasi sebuah sistem.

Tabel 2. Kebutuhan Perangkat Keras

No	**Kebutuhan Perangkat Keras**	
1	Processor	Intel I5-1035G1 CPU @ 1.00Ghz (8CPUs), ~ 1.2Ghz
2	Memory	20 GB DDR4 2667Mhz (Upgraded)
3	Storage	SSD 215GB, HDD 1TB
4	GPU	Intel(R) UHD Graphics

2.2.3. Perancangan Sistem

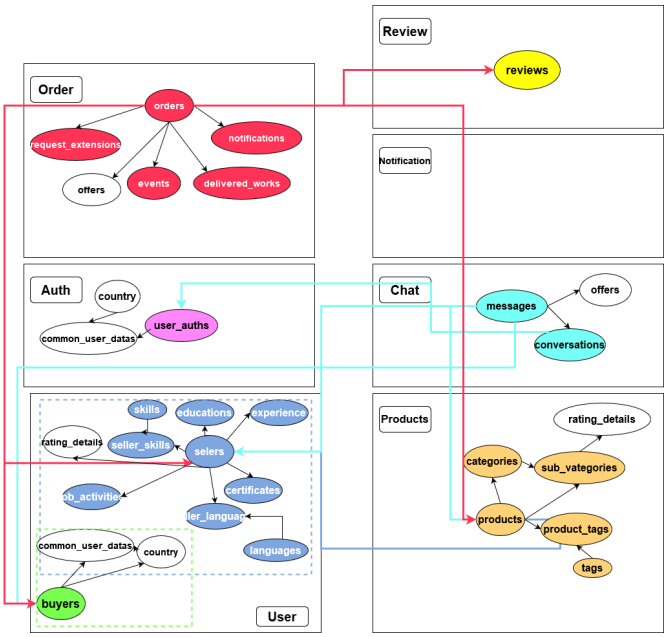
Pada tahap ini, proses pembuatan dan pengembangan sistem freelance berdasarkan kebutuhan yang dimiliki oleh M-Knows Consulting berbentuk ERD (Entity-Relationship Diagram). Namun, ERD tersebut dimodifikasi untuk meningkatkan kompleksitas sistem agar dapat direfleksikan ke dalam arsitektur mikroservis menjadi lebih ideal, mengingat arsitektur ini sangat cocok untuk aplikasi berskala besar. Penulis akan membagi ERD menjadi beberapa domain menggunakan paradigma DDD (Domain-Driven Design) dengan prinsip bounded context. Prinsip bounded context tidak bersifat absolut, karena penerapannya bergantung pada berbagai faktor, termasuk arsitektur aplikasi, fitur yang dibutuhkan, aspek keamanan, dan struktur organisasi dalam pengembangan perangkat lunak[8]. Lalu menentukan sub-domain dari domain bisnis yang telah ditentukan. Selain itu, penulis menerapkan pendekatan multi-*database* yang melibatkan lebih dari satu jenis basis data. Domain dan subdomain bisnis yang telah dipecah ini akan digunakan sebagai dasar dalam merancang arsitektur mikroservis.

2.2.4. Memecah Domain Dan Subdomain Bisnis

DDD (Domain-Driven Design) menjadi sebuah pendekatan yang menekankan pemahaman mendalam terhadap domain bisnis sebagai landasan dalam proses pembuatan dan pengembangan

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**

perangkat lunak. Pengembangan berbasis domain bisnis dapat diterapkan secara independen[9]. Paradigma ini bertujuan untuk mengurangi kompleksitas dalam pengembangan dengan memanfaatkan prinsip bounded context dalam memecah domain bisnis menjadi beberapa domain dan sub-domain yang lebih ideal. Namun, penerapan DDD tidak mengikuti langkah-langkah yang kaku atau baku, karena implementasinya perlu disesuaikan dengan kebutuhan bisnis, tim pengembang, dan kondisi spesifik lainnya.

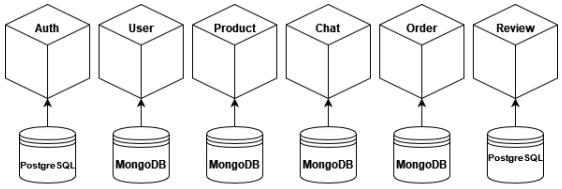


Gambar 2. Domain Setelah Menggunakan Paradigma DDD

Gambar 2 menunjukkan kondisi domain bisnis setelah menerapkan paradigma DDD dengan prinsip bounded context, entitas dibagi menjadi skema yang berbeda sebagai representasi dari domain dan subdomain. Setelah penulis memutuskan untuk menghubungkan entitas acuan ke dalam context (pembagian) yang ada, beberapa kolom mungkin mengalami perubahan, sehingga proses penerapannya sedikit berbeda dari entitas awal. Setiap context memiliki model data yang konsisten. Namun, jika telah menerapkan multi *database* yang menggunakan basis data NoSQL, relasi antar *context* tidak lagi diperlukan.

2.2.5. Menerapkan Multi Database

Setelah mengetahui model data sebagai domain bisnis, maka proses selanjutnya adalah menerapkan konsep multi *database* di dalam sistem. Setiap service memiliki *database* masing-masing, membuat tiap *service* jauh lebih aman dan dapat diskalakan, cocok untuk aplikasi berskala *enterprise*[10]. Konsep multi database ini mengacu pada penggunaan banyak *database* yang akan diimplementasikan pada tiap-tiap model data yang dibangun sebelumnya.



Gambar 3. Implementasi Multi Database

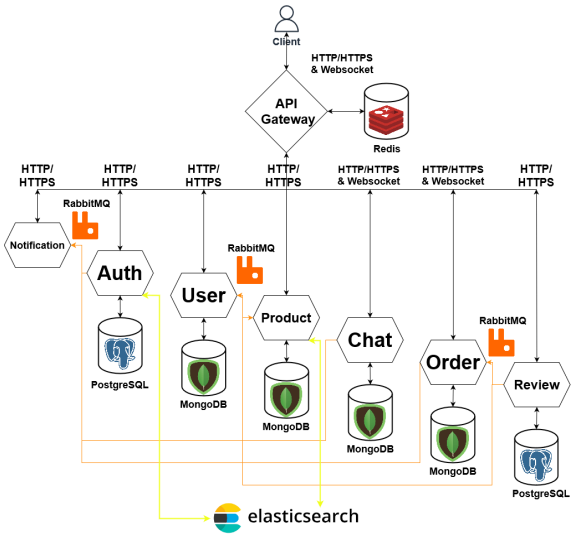
Gambar 3 menunjukan penerapan dari multi-*database*. Dalam implementasinya, model data *'auth'* dan *'review'* menggunakanan database PostgreSQL. Model data *'user'*, *'product'*,

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**

'chat', dan *'order'* menggunakan *database* MongoDB. Penerapan *database* MongoDB yang bertipe NoSQL untuk operasi seperti menambah, mengubah, dan menghapus data sering kali menawarkan performa yang lebih baik dibandingkan dengan menggunakan *database* SQL seperti PostgreSQL[11].

2.2.6. Refleksi Ke Arsitektur Mikroservis

Arsitektur mikroservis mencerminkan domain bisnis dengan membagi model bisnis menjadi beberapa mikroservis yang independen yang masing-masing mikroservis menangani satu domain bisnis yang terisolasi tanpa bergantung pada sistem lainya. Sistem akan jauh lebih mudah untuk dikelola atau lebih fleksibel untuk dikembangkan pada saat atau kondisi tertentu.



Gambar 4. Arsitektur Mikroservis Pada Sistem Freelance Berbasis Web Dan Teknologi Pendukung Lainya

Sesuai pada gambar 4 terdapat 7 mikroservis utama yang akan diimplementasikan. Keenam mikroservis yang ada merupakan refleksi dari model data yang telah dirancang. Lalu, terdapat 1 tambahan untuk mikroservis utama yaitu *'notification'* yang berfungsi untuk mengirim pesan ke *email* pengguna. Selain itu, terdapat mikroservis api gateway yang berfungsi sebagai gerbang komunikasi antara sisi klien dengan mikroservis utama. Tidak hanya sebagai gerbang komunikasi, api gateway juga berfungsi untuk mengontrol alur sistem, mengautentikasi, dan mengotorisasi sistem.

Dalam komunikasinya dari sisi klien yang dihubungkan oleh api gateway ke mikroservis utama menggunakan beberapa jenis protokol yaitu HTTP/HTTPS untuk mengirim *request* dan menerima *response* data. Pengiriman menggunakan protokol HTTP biasanya terdiri dari data, status, dan tipe[12]. Lalu, terdapat mikroservis *'chat'* dan *'order'* yang juga menggunakan protokol WebSocket yang mengharapkan komunikasi *real time*. WebSocket menjamin komunikasi 2 arah dalam 1 koneksi[13]. Selain itu, penggunaan protokol WebSocket juga untuk mengurangi kebutuhan *bandwidth*. Dalam komunikasi antar mikroservis digunakan protokol AMQP dengan dengan *tool* yang digunakan adalah RabbitMQ. Terdapat 4 komponen dalam protokol AMQP yaitu *publisher*, *exchange*, *subscriber*, dan *exchange*[14].

Terdapat teknologi Redis yang nantinya akan menyimpan data sementara agar sistem tidak terus menerus untuk melakukan *query* ke dalam *database*. Redis dapat memperkuat memaksimalkan proses pengelolaan *query* atau transaksi[15]. Lalu, untuk mendapatkan data produk, menggunakan teknologi Elasticsearch karena sistem pencarian jauh lebih cepat dan canggih daripada *query* dari *database*. Elasticsearch tidak menggunakan konsep relasional, tetapi dokumen atau JSON[16].

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**

2.3. Implementasi Sistem

Tahap berikutnya adalah implementasi dari sistem yang telah dirancang sebelumnya. Sistem ini dibangun menggunakan bahasa pemrograman TypeScript yang menggunakan *framework* Express JS berbasis TypeScript sebagai back-end dan *framework* React JS berbasis TypeScript sebagai front-end. Dua *database* utama yang digunakan adalah MongoDB dan PostgreSQL. Selama fase pengembangan, Docker digunakan untuk menjalankan aplikasi dalam lingkungan yang terisolasi, karena program tidak akan dijalankan dalam satu rute yang sama. Pada fase produksi, Kubernetes versi lokal akan digunakan, dengan *node* berbasis Minikube dan *virtual machine* Hyper-V.

3. HASIL DAN PEMBAHASAN

2.4. Kebutuhan Menjalankan Sistem Kubernetes

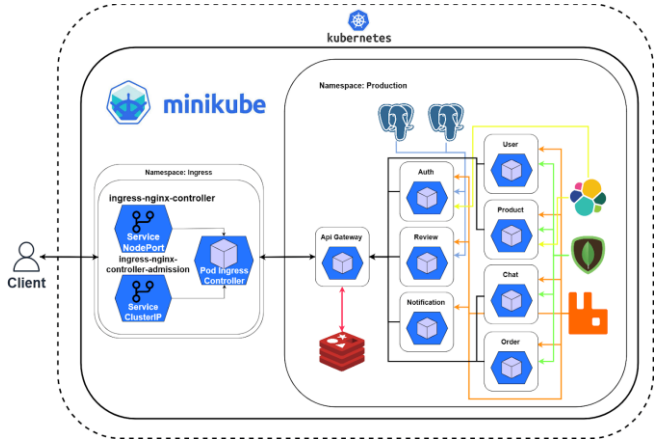
Proses orkestrasi sistem ke dalam Kubernetes memerlukan spesifikasi yang perlu disesuaikan.

Tabel 3. Perbandingan Algoritma A dan Algoritma B

No	**Kebutuhan Pengembangan Sistem Ke Dalam Kubernetes**	
1	Container Manager (Local)	Docker v4.16.3
2	Deploy Manager	Kubernetes v1.30.2
3	Cluster	Minikube v1.33.1
4	Virtual Machine	Hyper-V
5	Jumlah CPU	4
6	Jumlah Memory	9 GB

Tabel 3 menunjukkan kebutuhan sistem yang akan digunakan oleh Kubernetes. *Container manager* berfungsi sebagai pendukung proses *build* aplikasi tiap mikroservis, lalu di-*upload* ke dalam *container registry* (Docker Hub). *Cluster* untuk menjalankan orkestrasi dalam mengelola *image container* yang ada dalam Kubernetes. *Virtual machine* sebagai *emulator* dan mengisolasi aplikasi yang berjalan di dalam Kubernetes dari aplikasi dari luar VM. Jumlah CPU dan penyimpanan yang digunakan dibatasi hingga 4 CPU dan 9 GB.

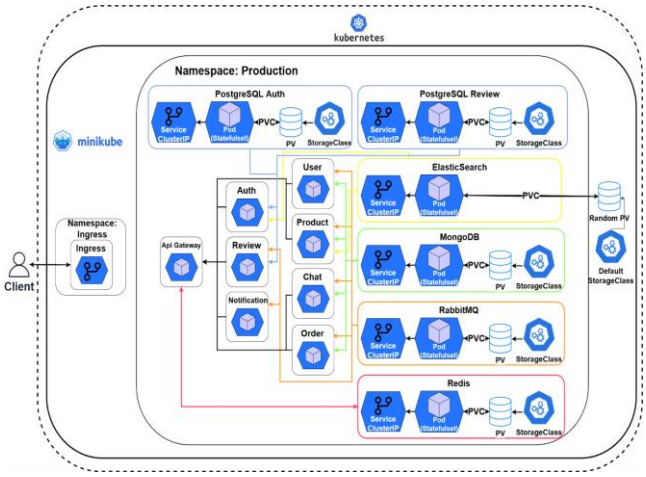
2.5. Arsitektur Di Dalam Cluster Kubernetes (Minikube)



Gambar 5. Arsitektur Di Dalam Cluster Minikube Bagian 1 (Ingress)

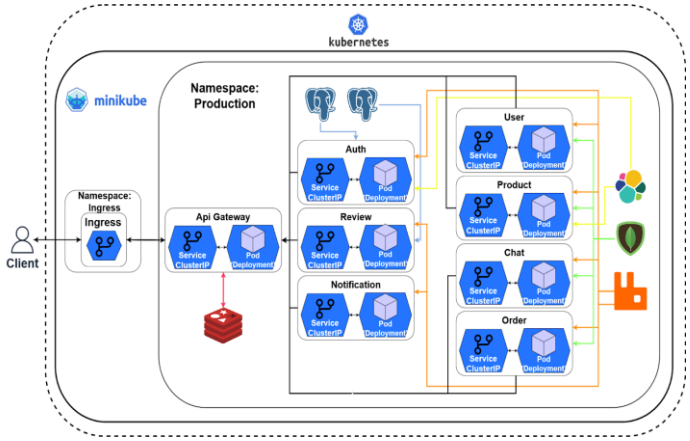
DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**

Arsitektur *cluster* Minikube dalam proyek ini menggunakan Ingress untuk mengelola komunikasi antara mikroservis dan lingkungan luar Kubernetes. Mikroservis api gateway dan mikroservis lainnya menggunakan Service bertipe 'ClusterIP', tapi hanya api gateway yang diakses melalui Ingress untuk berkomunikasi dengan dunia luar. Mikroservis lainnya mengandalkan trafik yang disalurkan melalui api gateway. 'ingress-nginx-controller' adalah komponen yang bertugas mengarahkan trafik dari klien ke api gateway, yang menggunakan tipe Service 'NodePort'.



Gambar 6. Arsitektur Di Dalam Cluster Minikube Bagian 2 (StatefulSet)

Selain mikroservis utama, terdapat *instance* tambahan seperti *database* dan *tools* pendukung lainya seperti Redis, MongoDB, PostgreSQL, RabbitMQ, dan Elasticsearch. Seluruh *instance* ini memiliki jenis pod bertipe 'statefulSet' karena akan menyimpan data yang bersifat persisten atau tetap. Terdapat beberapa komponen penting seperti storageClass yang berfungsi untuk memberikan aturan mengenai bagaimana data akan dimanajemen, PV (PersistentVolume) adalah tempat untuk menyimpan data, PVC (PersistentVolumeClaim) untuk mengambil data oleh sistem lainya terhadap PV dari *instance* yang ada, pod untuk menyimpan *container* yang di *pull* melalui *container registry* seperti Docker Hub, dan terdapat *service* sebagai langkah komunikasi dengan *instance* lainya yang bertipe 'clusterIp' karena hanya saling berkomunikasi di dalam *cluster* Kubernetes.



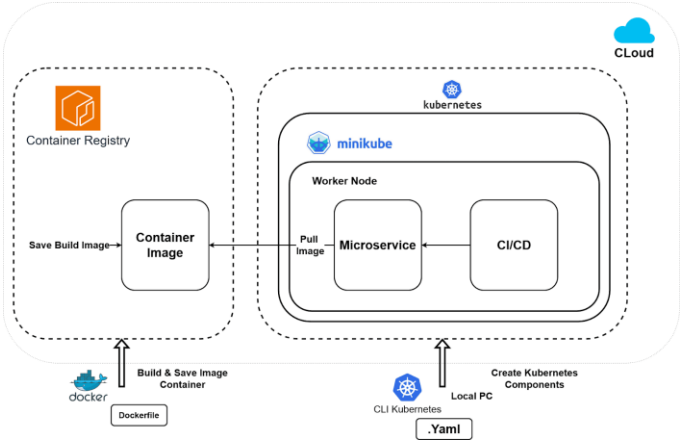
Gambar 7. Arsitektur Di Dalam Cluster Minikube Bagian 3 (Deployment)

Begitu juga dengan *instance* dari mikroservis utama yang menggunakan tipe pod 'deployment' karena tidak membawa data persisten atau tetap. Tipe 'deployment' tidak memerlukan pendefinisian komponen storageClass, PV (PersistentVolume), dan PVC (PersistentVolumeClaim).

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**

Segala komunikasi dari mikroservis utama akan mengarah ke mikroservis api gateway karena akan menggantungkan komunikasinya melalui api gateway. Apabila ada mikroservis utama yang bermasalah itu tidak akan mengganggu jalanya sistem berjalan, tetapi tidak menutup kemungkinan apabila api gateway bermasalah.

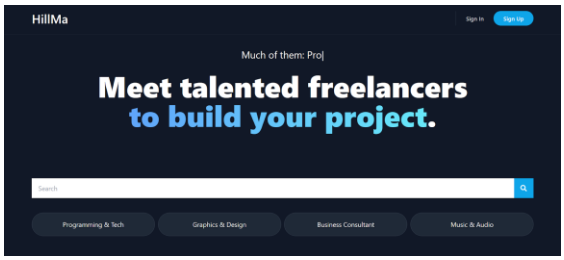
2.6. Alur Kubernetes Berinteraksi Dengan Container Registry



Gambar 8. Alur Kubernetes Dalam Berinteraksi Dengan Container Registry

Kubernetes memanfaatkan *container registry* seperti Docker Hub untuk mengambil *image container* yang diperlukan dalam menjalankan pod. Proses pertama kali yang dilakukan oleh Kubernetes adalah menunggu perintah untuk menjalankan komponen sesuai dengan konfigurasi pengembang di dalam file '.yaml'. Apabila terdapat perintah, maka akan dilakukan proses CI/CD atau Continuous Integration dan Continuous Delivery untuk memeriksa konfigurasi sudah sesuai atau tidak. Apabila sesuai, maka Kubernetes akan melakukan *pull* terhadap *image container* yang diambil dari *container registry*. *Image container* sendiri merupakan aplikasi yang telah di-*build* yang berisi kode aplikasi, *runtime environment*, *package*, *environment variables*, dan konfigurasi yang diperlukan.

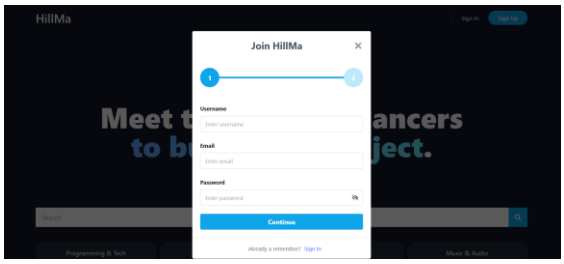
2.7. Implementasi Sistem Freelance Berbasis Website



Gambar 9. Halaman Index

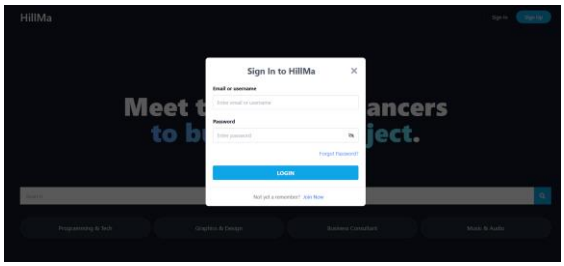
Pada gambar 9 merupakan halaman awal dari pengguna yang belum terautentikasi dan terotorisasi oleh sistem sehingga tidak bisa masuk ke halaman *home* dari pengguna yang terautentikasi dan terotorisasi.

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**



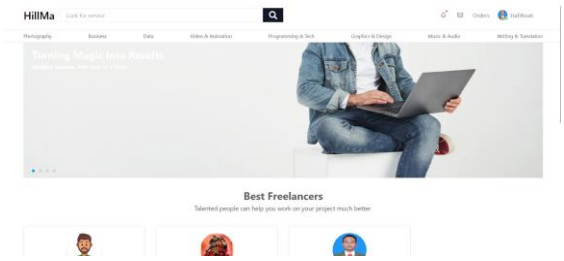
Gambar 10. Halaman Register

Pada gambar 10 merupakan halaman daftar untuk pengguna dengan mengisi beberapa *input* data.



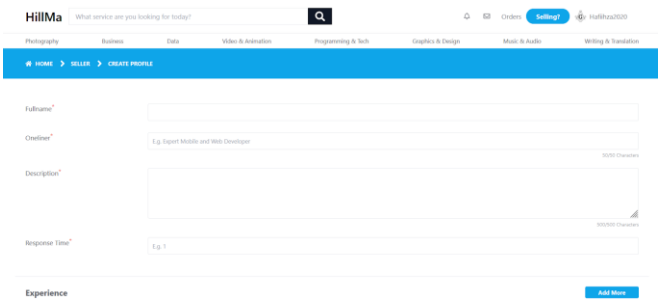
Gambar 11. Halaman Login

Pada gambar 11 merupakan halaman masuk sistem oleh pengguna dengan mengisi beberapa *input* data.



Gambar 12. Halaman Home

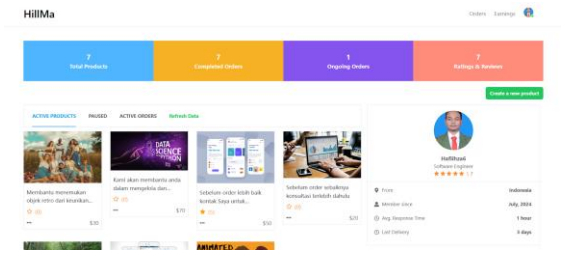
Pada gambar 12 merupakan halaman *home* dari pengguna yang telah mendaftar dan masuk ke dalam sistem. Sistem akan memeriksa pengguna yang telah terautentikasi dan terotorisasi. Apabila ada ketidaksesuaian, maka tidak akan bisa mengakses halaman *home* atau halaman lainya yang memerlukan *login* pengguna.



Gambar 13. Halaman Menjadi Penjual

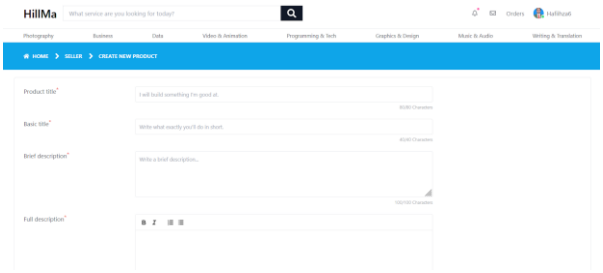
Pada gambar 13 merupakan halaman untuk menjadi penjual karena secara *default* pengguna akan menjadi pembeli secara otomatis. Sehingga diperlukan proses pendaftaran menjadi penjual, jika pengguna ingin menjualkan produknya.

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**



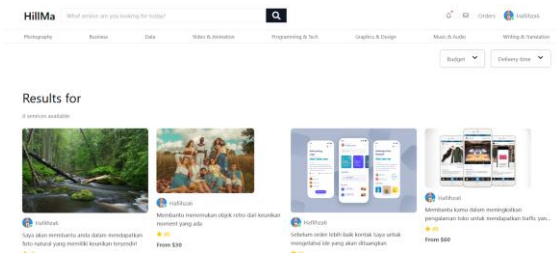
Gambar 14. Halaman Dasbor Penjual

Pada gambar 14 merupakan halaman dasbor penjual yang berisi informasi penjual dan produk yang dijual ke publik.



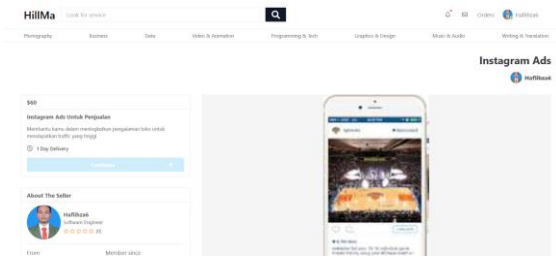
Gambar 15. Halaman Tambah Dan Ubah Produk

Pada gambar 15 merupakan halaman untuk menambah sekaligus mengubah produk oleh penjual yang memiliki produk tersebut.



Gambar 16. Halaman Cari Produk

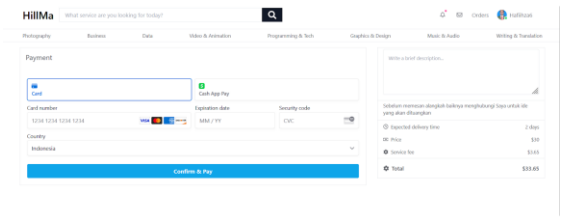
Pada gambar 16 merupakan halaman pencarian produk berdasarkan kata kunci pencarian atau kategori.



Gambar 17. Halaman Detail Produk

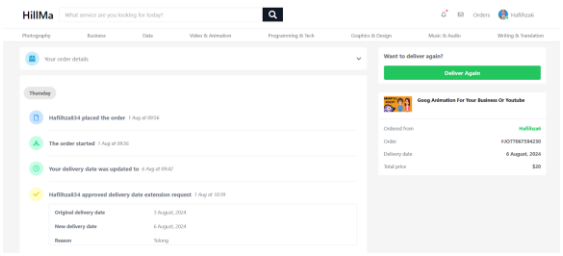
Pada gambar 17 merupakan detail dari produk. Berisi informasi data produk, penjual, penilaian, dan produk lainya yang disesuaikan dengan kemiripan data dari produk yang dipilih.

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**



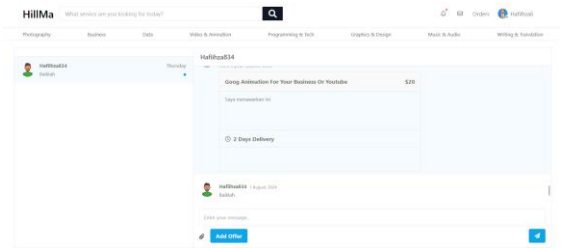
Gambar 18. Halaman Pembayaran

Pada gambar 18 merupakan halaman pembayaran terhadap pembeli yang akan melakukan pemesanan produk terhadap produk milik penjual.



Gambar 19. Halaman Detail Order

Pada gambar 19 merupakan halaman detail order atau pemesanan yang telah dipesan oleh pembeli sebelumnya.



Gambar 20. Halaman Pembayaran

Pada gambar 20 merupakan halaman percakapan antara pembeli dengan penjual. Pembeli dapat meminta kustom harga terhadap penjual dan penjual akan memberikan penawaran harga yang sesuai dengan keinginan pembeli.

4. KESIMPULAN

Penerapan arsitektur mikroservis sebagai refleksi dari model data yang dibangun berdasarkan paradigma DDD (Domain-Driven Design) telah berjalan sesuai harapan. Terdapat enam domain bisnis yang diidentifikasi dari model data, yaitu *auth*, *user*, *product*, *chat*, *order*, dan *review*, serta satu mikroservis tambahan, yaitu mikroservis *notification*, yang bertugas mengirim pesan melalui email. Selain itu, mikroservis api gateway berfungsi sebagai penghubung komunikasi antara klien dan mikroservis utama. Perancangan dan penerapan arsitektur mikroservis dapat disesuaikan dengan kondisi proyek yang ada dengan menerapkan paradigma DDD (Domain-Driven Design) dan prinsip bounded context. Dengan penerapan DDD dan bounded context, proyek ini dipermudah dalam perancangan arsitektur mikroservis dan implementasi multi-*database*. Penerapan Kubernetes sebagai manajemen *workload* berhasil dilakukan dengan menyesuaikan kondisi proyek, yang implementasinya secara keseluruhan dijelaskan pada bab 3 menggunakan *cluster* Minikube. Proses *deploy* mikroservis ke Kubernetes

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**

memerlukan *container registry* yang menyimpan *image container*, yang kemudian di-*pull* ke dalam Kubernetes versi lokal.

5. SARAN

Dalam penelitian ini terdapat beberapa kekurangan dan untuk lebih mengoptimalkan penelitian ini terdapat beberapa saran sebagai berikut: a)Melakukan integrasi Kubernetes versi lokal ke dalam Kubernetes versi *cloud* seperti AWS, GCP, dan lainya; b)Melakukan proses *unit testing* atau *integration testing* terhadap system; c)Memecah mikroservis user menjadi 2 mikroservis lainya seperti pembeli dan penjual.

DAFTAR PUSTAKA

[1] Ryansyah, M. and Tambunan, K., 2021, DAMPAK COVID-19 TERHADAP TINGKAT PENGANGGURAN DI INDONESIA, *TRIANGLE*, No.2 Vol.4, hal 486-491.

[2] Masakazu, K., Sisdianto, E., Suwardika, G., and Nugroho, D.S., 2023, Peran Digital Freelancer Marketplace dan Media Sosial Terhadap Perkembangan Gig Economy Worker, *Jurnal Pendidikan Ekonomi Undiksha*, No.1, Vol.15, hal 214-225.

[3] Permana, M.Y., Sisdianto, E., Izzati, N.R., and Askar, M.W., 2023, Measuring the Gig Economy in Indonesia: Typology, characteristics, and distribution, *SSRN*, hal 1-22, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4349942.

[4] Anwar, M.D. and Kautsar, I.A., 2024, Arsitektur Perangkat Lunak Berbasis Layanan Mikro pada Sistem Manajemen Informasi Kantin, *Journal Of Innovation Research And Knowledge*, No.2, Vol.1, hal 1-13.

[5] Siagian, N., Tamba, T.E., Situmorang, H.H., and Samosir, H.S., 2021, APLIKASI APOTEK BERBASIS WEB MENGGUNAKAN ARSITEKTUR MICROSERVICES (STUDI KASUS APOTEK GLEN, KAB.TOBA), *Journal Of Applied Technology And Informatics*, No.2, Vol.1, hal 22-28.

[6] Adiwijaya, A.P., 2022, IMPLEMENTASI MULTI DATABASE SEKMA PADA INFRASTRUKTUR BERBASIS MICROSERVICE, *Jurnal Teknik Dan Science*, No.3, Vol.1, hal 54-58.

[7] Nadaf, S.R., and Krishnappa, H.K., 2022, Kubernetes In Microservices, *Journal of Advanced Science and Computer Applications*, No.1, Vol.2, hal 7-18.

[8] Stafford, G., 2022, Monolith to Microservices: Refactoring Relational Databases, https://programmaticponderings.com/2022/04/14/monolith-to-microservices-refactoring-relational-databases%EF%BF%BC, diakses pada 14 April 2022

[9] Söylemez, M., Tekinerdogan, B., and Tarhan, A.K., 2022, Feature-Driven Characterization of Microservice Architectures: A Survey of the State of the Practice, *Applied Science*, No.9, Vol.12, hal 1-20.

[10] Foged, F., 2021, Multi-tenancy considerations of microservice designs, https://rasmusfoged.medium.com/multi-tenancy-considerations-of-microservice-designs-82f3c2285a44, diakses pada 28 Februari 2021

[11] Naufal, N., Nurkhodijah, S., Anugrah, G.A., Pratama, A., Rabbani, M.I., Dilla, F.A., Anggraeni, T.N. et al., 2022, ANALISA PERBANDINGAN KINERJA RESPONSE TIME QUERY MYSQL DAN MONGODB, *Jurnal Jitek*, No.2, Vol.2, hal 158-166.

DOI: 10.33050/cices.v11i1.3496 ISSN: **2356-5209**
pp. 10~23 *Online* ISSN: **2655-3058**

[12] Kusuma, I.G.N.A., 2021, PERANCANGAN SIMPLE STATELESS AUTENTIKASI DAN OTORISASI LAYANAN REST-API BERBASIS PROTOKOL HTTP, *Jurnal Manajemen informatika & Sistem Informasi*, No.1, Vol.4, hal 78-87.

[13] Sena, I.G.W., Pattiasina, T.J., Basatha, R., and Reinaldo, N.G., 2024, Perancangan dan Pembuatan Website Kuis Daring dengan Menggunakan Websocket Communication Protocol. *Jurnal Konstelasi*, No.1, Vol.4, hal 99-113.

[14] Yunandar, R.T. and Fahmi, M, 2022, IMPLEMENTASI MESSAGING SYSTEM BERBASIS PROTOKOL AMQP UNTUK SISTEM CONTROL NETWORK BERBASIS ANDROID, *Jurnal Akrab Juara*, No.2, Vol.7, hal 300-310.

[15] Ramadhan, I.N. and Saraswati, G.W., 2024, Penerapan DatabaseRedis Sebagai Optimalisasi Pemrosesan Kueri Data Pengguna Aplikasi SIRESMA Berbasis Laravel, *Technomedia Journal*, No.3, Vol.8, hal 394-406.

[16] Mysiuk, I., 2023, Designing a Data Warehouse for Collected Data About User Activity in Social Networks Using Elasticsearch, *International Electronic Scientific Journal*, No.7, Vol.9, hal 4002-4005.