Pengembangan *REST API* Untuk Aplikasi Pencarian Pekerjaan Sampingan Dengan Arsitektur *Microservices* Menggunakan Metode *Waterfall*

Aditya Alfarezy Damanik[1,*], Apriade Voutama[2]

[1,2] Fakultas Ilmu Komputer, Sistem Informasi, Universitas Singaperbangsa Karawang, Karawang, Indonesia
Email: [1,*]adityaalfarezyd@gmail.com, [2]apriade.voutama@staff.unsika.ac.id
*) Email Penulis Utama

Abstrak– Tingginya tingkat pengangguran dan menurunnya daya beli masyarakat mendorong individu untuk mencari pekerjaan sampingan sebagai sumber penghasilan tambahan. Namun, pencarian pekerjaan sampingan masih menghadapi tantangan utama, yaitu informasi yang tersebar di berbagai platform serta ketiadaan sistem terpusat yang terstruktur. Untuk mengatasi permasalahan ini, penelitian ini mengusulkan pengembangan SuruhAkuAja, sebuah aplikasi berbasis *REST API* dengan arsitektur *microservices* yang dirancang untuk menghubungkan pencari kerja dengan pemberi kerja secara lebih efisien. Penelitian ini menerapkan metode pengembangan perangkat lunak *Waterfall*, yang terdiri dari tahapan perencanaan sistem, desain arsitektur, implementasi, dan pengujian. Pada tahap perencanaan, sistem dimodelkan menggunakan *Unified Modeling Language* (UML) melalui *Use Case Diagram* untuk mengidentifikasi aktor dan interaksi dalam sistem serta *Class Diagram* untuk memodelkan hubungan antar entitas secara teknis, termasuk atribut, metode, dan keterkaitan objek. Pada tahap desain arsitektur, sistem dirancang menggunakan *microservices*, yang mencakup *API Gateway*, *Service Discovery*, serta komunikasi antar layanan menggunakan gRPC dan *publish-subscribe* dengan RabbitMQ. PostgreSQL digunakan dalam model *database-per-service* untuk memastikan skalabilitas dan fleksibilitas sistem. Implementasi dilakukan menggunakan bahasa pemrograman Golang dan PostgreSQL sebagai database utama untuk setiap *service*. Komunikasi antar *service* mengadopsi gRPC serta mekanisme *publish-subscribe* dengan RabbitMQ guna meningkatkan efisiensi dan keandalan sistem. Aplikasi ini juga terintegrasi dengan layanan pihak ketiga seperti Midtrans untuk transaksi pembayaran yang aman dan otomatis, serta Nominatim untuk reverse geocoding, yang dimana memungkinkan pencarian pekerjaan berbasis lokasi secara lebih akurat.. Dengan adanya SuruhAkuAja, pencari kerja dapat dengan mudah menemukan pekerjaan sampingan yang sesuai dengan keterampilan dan lokasi mereka, sementara pemberi kerja dapat mengelola lowongan lebih efektif, meningkatkan efisiensi perekrutan, memudahkan proses pencarian, serta memperluas akses terhadap tenaga kerja berkualitas. Aplikasi SuruhAkuAja diharapkan menjadi solusi inovatif dalam mendukung ekosistem ketenagakerjaan digital dan membantu mengurangi tingkat pengangguran di Indonesia. Tahap pengujian dilakukan menggunakan metode black-box testing yang berfokus pada validasi fungsionalitas utama layanan. Hasil pengujian dengan pendekatan Equivalence Partitioning menunjukkan bahwa seluruh dari 8 skenario pengujian REST API SuruhAkuAja berjalan sesuai dengan ekspektasi tanpa ditemukan kesalahan. Hasil pengujian performa REST API menggunakan Apache JMeter menunjukkan bahwa 85,52% request berhasil diproses, sementara 14,48% mengalami kegagalan.

Kata Kunci: *REST API*, *Microservices*, Golang, *Waterfall*, Pekerjaan Sampingan

Abstract– *The high unemployment rate and declining purchasing power have encouraged individuals to seek side jobs as an additional source of income. However, the process of finding side jobs still faces major challenges, particularly the dispersion of job information across various platforms and the absence of a structured, centralized system. To address this issue, this study proposes the development of SuruhAkuAja, an application based on REST API and microservices architecture designed to connect job seekers with employers more efficiently. The study adopts the Waterfall software development methodology, which includes the stages of system planning, architectural design, implementation, and testing. In the planning phase, the system is modeled using Unified Modeling Language (UML), specifically Use Case Diagrams to identify system actors and interactions, and Class Diagrams to technically model the relationships between entities, including their attributes, methods, and object associations. In the architectural design phase, the system is structured using microservices, incorporating components such as API Gateway, Service Discovery, and inter-service communication via gRPC and a publish-subscribe mechanism using RabbitMQ. PostgreSQL is implemented in a per-service database model to ensure system scalability and flexibility. The implementation is carried out using the Go programming language (Golang) and PostgreSQL as the primary database for each service. Inter-service communication adopts gRPC and publish-subscribe mechanisms via RabbitMQ to enhance system efficiency and reliability. The application is also integrated with third-party services such as Midtrans for secure and automated payment transactions, and Nominatim for reverse geocoding, which enables more accurate location-based job searches. With SuruhAkuAja, job seekers can easily find part-time jobs that match their skills and location, while employers can manage job postings more effectively, streamline recruitment processes, and expand access to qualified workers. The application is expected to serve as an innovative solution to support the digital employment ecosystem and help reduce unemployment in Indonesia. The testing phase was conducted using black-box testing, focusing on validating the core service functionalities. The results, obtained through the Equivalence Partitioning approach, showed that all 8 REST API test scenarios of SuruhAkuAja performed as expected without any errors. Performance testing of the REST API using Apache JMeter showed that 85.52% of requests were successfully processed, while 14.48% failed*.

Keywords: *REST API, Microservices, Golang, Waterfall, Side Jobs*

1. PENDAHULUAN

Indonesia berada di peringkat keempat sebagai negara dengan jumlah penduduk terbanyak di dunia. Berdasarkan laporan Hasil Sensus Penduduk (SP2020) yang dilakukan pada September 2020, populasi Indonesia tercatat mencapai 270,20 juta jiwa dengan laju pertumbuhan penduduk per tahun sebesar 1,25% pada kurun waktu 2010-2020. Jumlah populasi tersebut bertambah 32,56 juta jiwa dibandingkan Sensus Penduduk 2010 (SP2010). Dari banyaknya jumlah penduduk tersebut, dicatatkan bahwa 70,72% dari total populasi merupakan penduduk dengan usia produktif yakni dari umur 15-64 tahun, sehingga Indonesia dapat dikatakan sedang dalam masa bonus demografi [1].

Namun, di tengah jumlah penduduk yang besar, Indonesia juga mengalami tingkat pengangguran terbuka (TPT) yang cukup tinggi, yaitu sebesar 7,07% pada Agustus 2020 (BPS, 2020). Angka tersebut turut dipengaruhi oleh dampak pandemi Covid-19, yang semakin memperburuk kondisi ketenagakerjaan di Indonesia. Akibatnya, tingkat kesejahteraan dan daya beli masyarakat menurun, sementara persaingan dalam memperoleh pekerjaan menjadi semakin ketat. Dengan menurunnya daya beli masyarakat, hal ini secara otomatis menyebabkan para pelaku usaha dari yang kecil hingga besar melakukan efisiensi dengan melakukan pemutusan hubungan kerja (PHK) untuk menekan biaya operasional [2].

Dalam situasi ekonomi yang semakin tidak menentu, masyarakat Indonesia berupaya beradaptasi dengan mencari penghasilan tambahan melalui pekerjaan sampingan. Informasi mengenai pekerjaan sampingan ini umumnya diperoleh dari kerabat, media sosial, atau berbagai sumber lainnya. Namun, keterbatasan akses dan penyebaran informasi yang tersebar di berbagai tempat membuat pekerjaan sampingan sulit ditemukan.

Di era Revolusi Industri 4.0, kemajuan teknologi berkembang dengan pesat, mendorong munculnya berbagai inovasi yang efektif dalam menyelesaikan beragam permasalahan. Oleh karena itu, untuk mengatasi tantangan dalam penyebaran informasi mengenai pekerjaan sampingan, diperlukan sebuah aplikasi baru yang mampu menyediakan lowongan secara cepat, mudah, dan aman. Dengan demikian dikembangkanlah aplikasi yang bernama SuruhAkuAja, dengan adanya aplikasi ini, diharapkan para pekerja dapat lebih mudah menemukan peluang pekerjaan sampingan yang sesuai. Aplikasi ini dikembangkan dengan tujuan mengatasi permasalahan penyebaran informasi pekerjaan sampingan yang tidak terstruktur dan terbagi-bagi.

Pengembangan aplikasi ini diharapkan dapat memberikan berbagai manfaat, baik bagi pekerja maupun pencari jasa. Dengan adanya aplikasi ini, informasi mengenai pekerjaan sampingan dapat diakses secara terpusat dan lebih mudah dijangkau oleh masyarakat, sehingga tidak lagi tersebar di berbagai sumber yang sulit ditemukan. Dengan lebih banyak orang mendapatkan pekerjaan sampingan, daya beli masyarakat meningkat, yang pada akhirnya berkontribusi pada pertumbuhan ekonomi lokal.

Desy Intan Pertamasari, dkk mampu membuat aplikasi pencarian kerja sampingan berbasis *Framework Flutter* yang digunakan untuk mempertemukan penyedia pekerjaan sampingan dengan pencari pekerjaan sampingan dalam satu wadah digital dalam rangka mendorong terciptanya gotong royong ekonomi di dalam kehidupan masyarakat. Dalam pengujian aplikasi, penelitian tersebut dibagi menjadi dua yakni *API Testing* dan *Usability Testing*. *API testing* dilakukan dengan melakukan *request* ke *web API* dengan menggunakan *tools Postman*. Sementara untuk *Usability Testing* dilakukan dengan menggunakan alat pengukuran bernama SUS (*System Usability Scale*). Penelitian ini memiliki kemiripan dengan penelitian yang diangkat namun terdapat perbedaan dalam cara implementasi serta fitur yang dikembangkan. Penelitian tersebut juga belum menjelaskan lebih lanjut mengenai implementasi pembayaran, pengambilan gaji untuk pekerja, dan juga konteks pekerjaan yang dilakukan [3]. Muhammad Avied Bachmid, dkk telah mengembangkan aplikasi bernama *Jobhere* yang bertujuan untuk proses rekrutmen untuk UMKM di Kota Malang. Sebelumnya proses rekrutmen masih menggunakan forum sosial media seperti *Facebook* sehingga sering mengalami kesulitan dan keterbatasan dalam penyebaran informasi lowongan kerja. Pengembangan aplikasi tersebut menggunakan model *Waterfall* sebab kebutuhan fungsional dari sistem sudah didefinsikan diawal pengembangan dan tidak adanya kebutuhan untuk merubah kebutuhan. Pengujian yang dilakukan terbagi menjadi dua, yakni pengujian fungsional dan usabilitas. Penelitian ini membatasi rekrutmen pekerjaan untuk UMKM di Kota Malang, selain itu tidak ada pencarian berbasis lokasi yang dekat dengan pengguna [4].

Penelitian akan berfokus pada pengembangan sebuah *REST API* dari aplikasi SuruhAkuAja. REST (*Representational State Transfer*) atau *RESTful* merupakan salah satu arsitektur yang popular digunakan untuk membangun *service* yang dapat diintegrasikan untuk mendukung interopabilitas [5]. Untuk integrasi antar *service*, *REST API* menggunakan *JSON*, sehingga memudahkan pengembang dalam menggunakannya [6]. *REST API* merupakan bagian dari pengembangan *Back-end* [7]. *Backend Development* adalah proses perancangan sistem yang berjalan di belakang layar sebuah *website* atau aplikasi dan tidak dapat diakses secara langsung oleh pengguna [8]. Untuk mengembangkan *REST API*, digunakan bahasa pemrograman Go atau Golang yang dikembangkan oleh *Google*. Bahasa ini terinspirasi dari konsep C dan C++, sehinga memiliki sintaks yang mirip [9]. Golang dapat digunakan oleh pengembang perangkat lunak untuk membangun aplikasi berbasis *web, cloud*, serta berbagai jenis aplikasi lainnya [10]. *REST API* dapat menggunakan berbagai jenis database untuk menyimpan data, salah satunya adalah *PostgreSQL*. *PostgreSQL* merupakan sistem manajemen basis data yang bersifat *open-source* dan *compatible* dengan berbagai platform [11]. Arsitektur REST API dapat

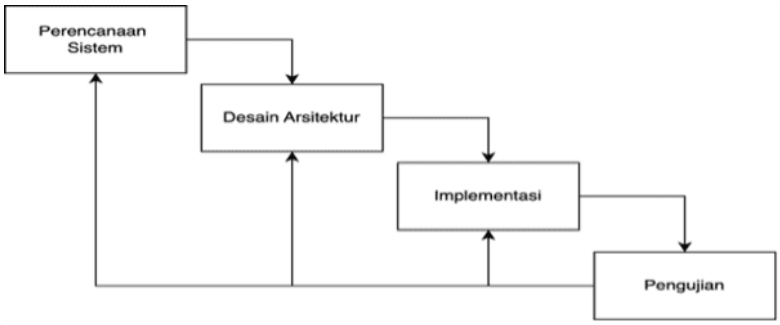
diimplementasikan dalam bentuk *Microservices*, yang membagi aplikasi menjadi *layanan-layanan* kecil yang saling terhubung, berbeda dengan pendekatan monolitik yang menyatukan seluruh komponen dalam satu kesatuan [12]. Arsitektur *microservices* memengaruhi keterhubungan antara aplikasi dan basis data. Dalam pendekatan ini, setiap layanan memiliki skema database yang terpisah dan dikelola secara independen [13]. Pengembangan *REST API* dilakukan dengan menerapkan model *Waterfall* sebagai pendekatan dalam *Software Development Life Cycle* (SDLC). Model *Waterfall* merupakan metode pengembangan sistem yang bersifat berurutan, di mana setiap tahap harus diselesaikan sepenuhnya sebelum berlanjut ke tahap berikutnya [14]. Selain itu, model *Waterfall* memiliki sifat *non-iteratif*, sehingga setiap tahap yang telah diselesaikan tidak dapat dikembalikan atau diulang ke tahap sebelumnya [15]. Metode *Waterfall* memiliki risiko seperti pemborosan waktu dan biaya, manajemen proyek yang kurang efektif, kesalahan dalam perancangan atau penentuan persyaratan di awal, serta penyimpangan dari proses yang telah ditetapkan secara berurutan [16].

Tujuan dari penelitian ini adalah untuk merancang dan membangun REST API dari aplikasi SuruhAkuAja dengan arsitektur *microservices* yang mengintegrasikan komponen-komponen penting seperti payment gateway, geolocation service, dan mekanisme komunikasi antar layanan (gRPC dan RabbitMQ) guna memfasilitasi penyediaan informasi pekerjaan sampingan secara terpusat, cepat, dan terstruktur. Dengan demikian, aplikasi ini diharapkan dapat menjadi solusi inovatif dalam membantu masyarakat mendapatkan pekerjaan tambahan secara mudah dan aman, serta mendukung upaya pengurangan tingkat pengangguran di Indonesia.

2. METODE PENELITIAN

2.1 Metode Pengembangan *Waterfall*

Proses rancang bangun *REST API* SuruhAkuAja mengadopsi model pengembangan *Waterfall* yang bersifat sistematis dimana terdapat beberapa tahap yang harus dilalui secara berurutan [17]. Metode *Waterfall* dipilih karena setiap tahapan dalam pengembangan telah direncanakan dengan baik sejak awal, sehingga perubahan atau revisi terhadap tahapan sebelumnya sangat jarang terjadi. Pendekatan ini sesuai dengan karakteristik proyek yang memiliki kebutuhan dan alur kerja yang sudah jelas serta tidak memerlukan iterasi berulang antar tahapan. Adapun beberapa tahap dalam pengembangan REST API tersebut telah digambarkan sesuai dengan urutan dalam diagram *Waterfall* yang ditampilkan pada Gambar 1.



Gambar 1. Diagram *Waterfall*

Berikut adalah penjabaran lebih rinci masing-masing tahapan yang telah digambarkan pada diagram *Waterfall*:

2.1.1 Perencanaan Sistem

Pada tahap ini sistem dirancang dan direncanakan yang dimana proses tersebut akan digambarkan dalam *Use Case Diagram* dan *Class Diagram*. *Use Case Diagram* digunakan untuk menggambarkan aktor yang terlibat serta kelakuannya terhadap sistem [18]. *Class diagram* adalah representasi visual dari struktur *database* yang akan dibuat, terdiri dari berbagai *class* yang digunakan untuk membangun suatu sistem [19]. Dalam pembuatan use case dan class diagram, digunakan sebuah tools bernama draw.io yang merupakan sebuah situs web yang dirancang khusus untuk membuat diagram seperti flowchart dan Unified Modelling Language secara daring. Seluruh fitur yang tersedia pada platform ini dapat diakses melalui browser yang mendukung HTML5 [20]. Untuk menyimpan file diagram, draw.io memungkinkan pengguna untuk menyimpannya pada Google Drive [21]. Diagram yang telah dibuat pada tahap ini menjadi dasar dalam menyusun desain arsitektur *microservices*, Draw.io yang terintegrasi baik dengan penyimpanan cloud, memudahkan pengguna dalam berkolaborasi karena diagram yang telah dibuat dapat diakses dan diperbarui oleh pengguna lain secara simultan. Diagram yang telah dibuat pada tahap ini menjadi dasar dalam menyusun desain arsitektur microservices, karena menggambarkan kebutuhan sistem, aktor, serta struktur data yang harus diakomodasi oleh layanan-layanan yang akan dibangun.

2.1.2 Desain Arsitektur

Pada tahap ini, peneliti akan merancang serta menyusun desain arsitektur *microservices* yang akan diterapkan. Proses ini mencakup perencanaan komponen *services*, komunikasi antar *services*, serta penentuan infrastruktur yang tepat untuk memastikan sistem berjalan secara efisien dan terdistribusi. Desain arsitektur yang dihasilkan adalah diagram yang menggambarkan bagaimana setiap layanan dalam arsitektur *microservices* saling berinteraksi dan berbagi data, termasuk protokol komunikasi yang digunakan, *database* yang diadopsi, teknologi yang digunakan, serta alur data yang mengalir di dalam sistem. Dalam tahap desain arsitektur digunakan sebuah aplikasi diagram berbasis *website* yang bernama *Lucidchart*. *Lucidchart* dapat digunakan untuk membuat diagram, visualisasi alur kerja, dan pemetaan konsep secara kolaboratif dan *real-time [22]*. Tahapan desain arsitektur adalah tahapan yang krusial karena kesalahan dalam perancangan dapat berdampak pada kinerja dan skalabilitas sistem secara keseluruhan. Oleh karena itu, dalam tahap ini, peneliti memastikan bahwa setiap layanan memiliki tanggung jawab yang jelas, saling terisolasi, namun tetap dapat berkomunikasi secara optimal. Desain arsitektur yang dihasilkan pada tahap ini menjadi acuan utama dalam proses implementasi, karena setiap komponen layanan yang telah dirancang akan diubah menjadi kode program yang merepresentasikan fungsi masing-masing layanan dalam sistem.

2.1.3 Implementasi

Tahap implementasi merupakan proses pengkodean berdasarkan sistem yang telah dirancang serta arsitektur yang telah dibuat. Pada tahap ini, kode sumber dikelola dalam satu *repository* yang mencakup berbagai proyek atau layanan yang berbeda, dengan keterkaitan yang telah ditetapkan secara jelas. Penggunaan monorepo memiliki beberapa keuntungan seperti kemudahan dalam pengelolaan proyek, kode yang tersinkronisasi, manajemen dependensi yang mudah dan terpusat, dan kolaborasi atau aksebilitas yang baik. Proses implementasi dilakukan menggunakan bahasa pemrograman Golang. Bahasa pemrograman Golang dipilih karena memiliki beberapa keunggulan dari segi performa dan penggunaan sumber daya komputer seperti CPU dan memori. Selain itu bahasa pemrograman Golang memiliki fitur *concurrency*, yang memungkinkan eksekusi beberapa tugas secara bersamaan dalam satu waktu. Selain *concurrency* terdapat fitur Garbage Collector, yang membantu mengelola penggunaan memori secara efisien, sehingga mengurangi konsumsi memori saat aplikasi dijalankan [23]. Tahap implementasi yang dilakukan menjadi dasar bagi tahap pengujian berikutnya, di mana seluruh fitur yang telah dibangun akan divalidasi untuk memastikan bahwa aplikasi berjalan sesuai dengan harapan.

2.1.4 Pengujian

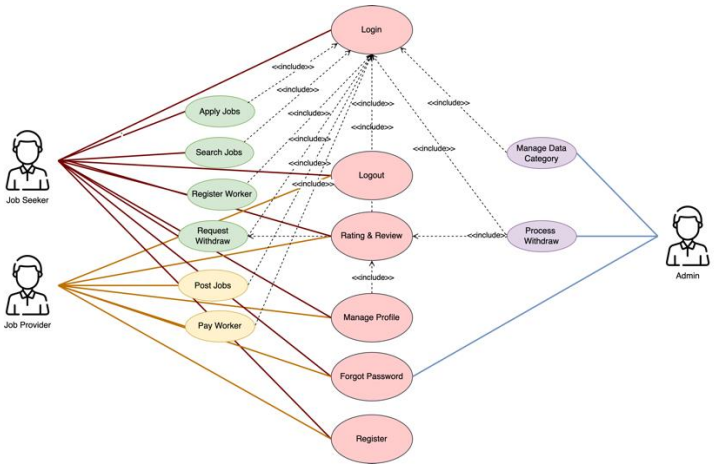
Tahap pengujian dilakukan oleh peneliti untuk memastikan bahwa aplikasi yang telah dikembangkan dapat bekerja sesuai dengan spesifikasi yang telah ditetapkan dan sesuai dengan alur bisnis yang direncanakan. Pada penelitian ini untuk menguji fungsionalitas aplikasi dilakukan sebuah metode *black box*. Pengujian *Black Box* merupakan metode pengujian perangkat lunak yang berfokus pada evaluasi fungsi eksternal dari perangkat lunak yang sedang dikembangkan, tanpa memperhatikan struktur internalnya [24]. Dalam pengujian *black box* dibuatkan beberapa *test case* yang dibuat untuk mendefinisikan ekspektasi hasil dari setiap fitur yang diuji. *Test case* ini mencakup berbagai skenario penggunaan, termasuk *input* valid dan *input* tidak valid, memastikan aplikasi dapat menangani berbagai kemungkinan interaksi pengguna. Setiap *test case* dirancang untuk menguji apakah sistem merespons dengan benar terhadap *input* yang diberikan, baik dalam keadaan normal maupun saat terjadi kesalahan. Jika hasil yang diperoleh sesuai dengan yang diharapkan, maka fitur tersebut dianggap berhasil memenuhi spesifikasinya. Namun, jika ditemukan ketidaksesuaian, maka akan dilakukan perbaikan dan pengujian ulang hingga fitur dapat berfungsi dengan baik. Proses pengujian ini berperan penting dalam memastikan bahwa aplikasi berfungsi sesuai dengan kebutuhan yang telah dirancang. Melalui tahapan pengujian, validitas dari setiap fitur yang dikembangkan dievaluasi agar sesuai dengan harapan. Pengujian ini menjadi langkah krusial untuk menilai apakah sistem mampu mendukung pencapaian tujuan utama penelitian. Dengan demikian, hasil dari tahap pengujian akan menjadi dasar untuk menilai keberhasilan solusi yang diusulkan dalam menjawab permasalahan yang telah diidentifikasi.

3. HASIL DAN PEMBAHASAN

3.1 Perencanaan Sistem

Terdapat beberapa hal yang dilakukan dalam tahap perencanaan sistem yang dimana salah satunya adalah memodelkan aplikasi dalam bentuk *Usecase* dan *Class Diagram* yang merupakan bagian dari *Unified Modelling Language* (UML) [25]. *Unified Modeling Language* (UML) adalah sebuah standar yang digunakan untuk

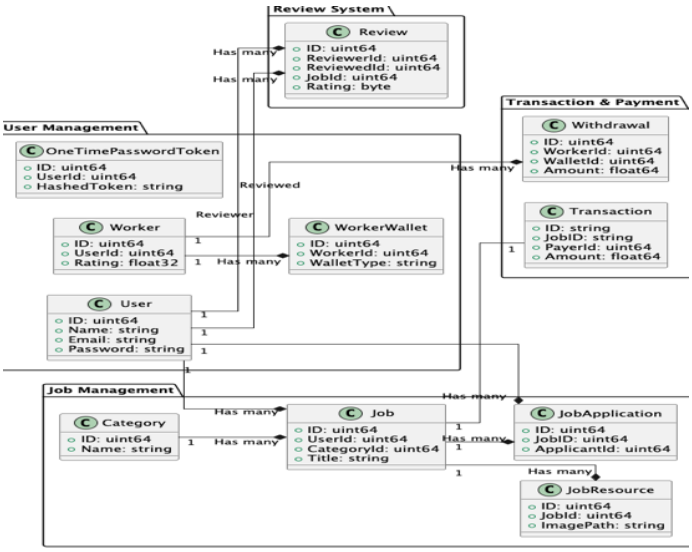
memodelkan, mendokumentasikan, serta merancang sistem perangkat lunak [26]. Pemodelan tersebut dilakukan untuk memberikan gambaran umum mengenai aplikasi yang akan dikembangkan agar memudahkan pemberian informasi tentang aplikasi. Berikut adalah pemodelan yang dilakukan:



Gambar 2. *Usecase Diagram* Aplikasi SuruhAkuAja

Gambar 2 di atas merupakan *Usecase Diagram* dari aplikasi SuruhAkuAja. Pada aplikasi, pengguna dikelompokkan menjadi beberapa aktor yakni, *Job Seeker*, *Job Provider* dan Admin. Semua aktor yang didefinisikan memiliki beberapa fitur dasar yang dapat diakses seperti *Login, Logout.* Aktor *Job Seeker* dan *Job Provider* dapat menggunakan beberapa fitur spesifik yang sesuai dengan kebutuhannya. Semua fitur spesifik tersebut dapat dilakukan ketika sudah berhasil melakukan *Login*. Aktor lain dalam diagram ialah Admin yang dimana membantu proses bisnis aplikasi.

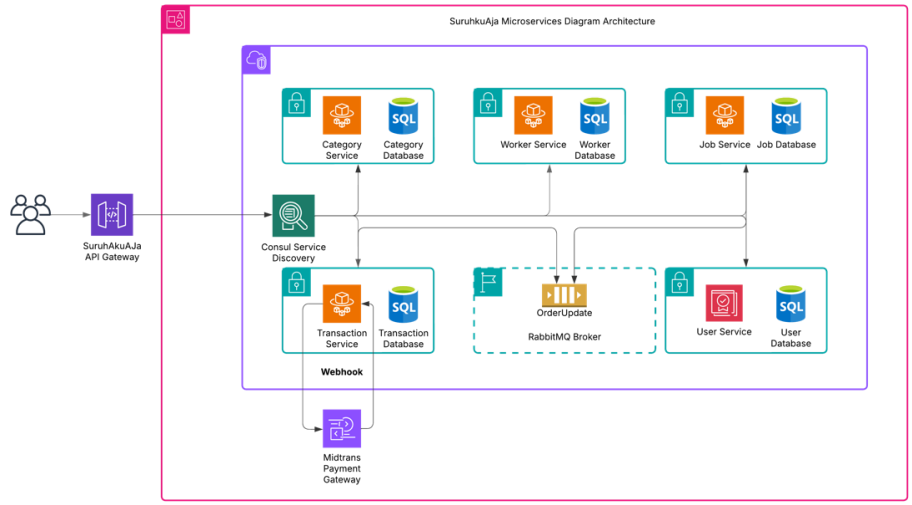
Class Diagram menggambarkan mengenai hubungan antar entitas dan atribut yang menjadi bagian dari proses bisnis SuruhAkuAja. Terhitung terdapat 11 entitas yang termuat dalam *class diagram* yang masing-masing memiliki atribut uniknya sendiri yang sesuai dengan kebutuhan. Diagram di atas terbagi menjadi beberapa *domain* utama yakni Manajemen Pengguna, Manajemen Pekerjaan, Sistem *Review*, dan Transaksi & Pembayaran.



Gambar 3. *Class Diagram* Aplikasi SuruhAkuAja

3.2 Desain Arsitektur

Perencanaan arsitektur yang matang sangat krusial dalam pengembangan aplikasi berbasis *microservices*, karena menentukan arah pengembangan serta keterkaitan antar komponen *services*. Gambar di bawah memperlihatkan diagram arsitektur *microservices* SuruhAkuAja yang terdiri dari berbagai *service* dan teknologi yang saling terintegrasi dan berkomunikasi satu sama lain.



Gambar 4. Diagram Arsitektur *Microservices*

Diagram diatas merupakan desain arsitektur berbasis *microservices* untuk aplikasi SuruhAkuAja. Arsitektur ini mengadopsi *API Gateway*, *Service Discovery*, *Publish-Subscribe Database-per-Service*, dan integrasi dengan *Payment Gateway* untuk transaksi pembayaran. Dalam diagram arsitektur terdapat beberapa komponen penting seperti *API Gateway* dan *Service Discovery*. API Gateway bertindak sebagai pintu masuk bagi pengguna dari internet sedangkan *Service Discovery* berguna untuk mendeteksi dan mengelola service yang ada dalam sistem yang memungkinkan komunikasi antar services tanpa harus menentukan alamat IP secara statis. Arsitektur *microservices* memiliki beberapa *services* utama yang berjalan secara independen, yakni sebagai berikut:

1. *User Service*: Bertanggung jawab atas pengelolaan pengguna, baik mereka yang mencari pekerja maupun yang ingin menawarkan jasanya. Layanan ini menangani autentikasi dan penyimpanan informasi pengguna.
2. *Category Service*: *Service* ini berperan dalam mengelompokkan pekerjaan ke dalam kategori-kategori tertentu sehingga pengguna dapat dengan mudah menemukan *service* yang mereka butuhkan.
3. *Worker Service*: *Service* yang khusus mengelola data pekerja, mencatat keterampilan, pengalaman, serta *rating* yang diberikan oleh para pemberi kerja.
4. *Job Service*: *Service* yang bertugas mengelola seluruh proses terkait pekerjaan, mulai dari penciptaan lowongan oleh pemberi kerja, pemilihan pekerja, hingga penyelesaian pekerjaan. Status pekerjaan, baik yang masih menunggu, sedang berlangsung, maupun telah selesai, akan selalu diperbarui di dalam *service* ini.
5. *Transaction Service*: *Service* tersebut sangat krusial dimana berperan sebagai layanan yang menangani pembayaran antara pemberi kerja dan pekerja. Layanan ini terintegrasi langsung dengan Midtrans *Payment Gateway* untuk memproses pembayaran, memastikan transaksi berlangsung dengan aman, dan mencatat setiap perubahan saldo di dalam sistem.

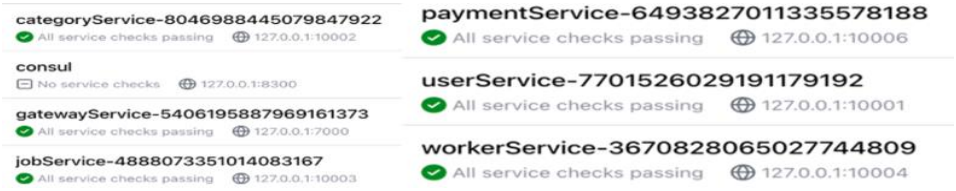
Keseluruhan sistem ini dirancang agar dapat berkembang secara fleksibel dan efisien. Dengan penggunaan *service discovery* seperti Consul serta komunikasi berbasis *publish-subscribe* melalui RabbitMQ, setiap *service* dapat berjalan secara mandiri tanpa bergantung langsung pada *service* lain. Jika suatu service mengalami gangguan, sistem tetap dapat beroperasi tanpa mengalami *downtime* yang signifikan.

3.3 Implementasi

Setelah dilaksanakannya perencanaan serta perancangan terhadap aplikasi SuruhAkuAja maka tahap selanjutnya adalah melakukan pengimplementasian dengan mengembangkan aplikasi SuruhAkuAja berdasarkan rancangan yang telah dibuat. Berikut adalah hasil dari pengembangan aplikasi SuruhAkuAja dengan arsitektur *microservices*:

3.3.1 *Service Discovery*

Dalam aplikasi berasitektur *microservices*, setiap *services* awalnya tidak akan mengetahui letak atau lokasi dari *services* lain. Sehingga diperlukan sebuah pengatur atau tempat yang menyimpan semua lokasi dari *services*, dengan itu ketika suatu *service* ingin berkomunikasi dengan *service* lain, maka *service* tersebut perlu menanyakan terlebih dahulu letak dari *service* yang dituju pada *service discovery*. Berikut adalah gambar yang menunjukkan daftar *service* yang telah ditemukan atau diketahui oleh *service discovery*:



Gambar 5. Daftar *Service* pada Consul *Service Registry*

Pada gambar 5, ditunjukkan *service-service* yang telah dikenali oleh Consul *Service Discovery*. Untuk memberi tahu kepada Consul *Service Discovery* bahwa terdapat *service* yang baru hidup atau mati, hal tersebut dapat dilakukan pada *entrypoint* dari masing-masing *service*. Pada saat *service* berjalan, *service* didaftarkan ke Consul dengan mengirimkan *request*, sedangkan jika service mati secara *graceful*, maka *service* akan dikeluarkan dari Consul dengan mengirimkan *request deregistrasi* ke Consul.

3.3.2 *Inter-Service Communication* dengan gRPC

Arsitektur *microservices* pada aplikasi SuruhAkuAja menerapkan *database-per-service* yang dimana data-data yang ada terpecah belah dalam beberapa *database*, sehingga diperlukan komunikasi antar *services* untuk mewujudkan integritas data yang baik. Dalam komunikasi antar *services* tersebut, terdapat beberapa cara, salah satunya adalah menggunakan gRPC. gRPC merupakan *framework open-source* dengan performa tinggi yang memungkinkan komunikasi antar aplikasi dan layanan [27]. gRPC menggunakan *protocol buffer* untuk mendefinisikan struktur data serta metode pertukaran data [28].

```
syntax = "proto3";
import "google/protobuf/empty.proto";
import "user.proto";
option go_package = "github.com/alfarezyyd/go-takemikazuchi-microservices/common/genproto/worker";
message CreateWorkerRequest {
  UserJwtClaim UserJwtClaim = 1;
  string EmergencyPhoneNumber = 2;
  string Location = 3;
  string WalletType = 4;
  string AccountNumber = 5;
  string AccountName = 6;
  string BankName = 7;
}
service WorkerService {
  rpc Create(CreateWorkerRequest) returns (google.protobuf.Empty);
}
```

Gambar 6. *Protobuf* dari *Category Proto*

Pada gambar 6 ditampilkan *protobuf* yang mendefinisikan struktur data dan metode dari *Category Service*. Dapat dilihat bahwa deklarasi *protobuf* hampir mirip dengan deklarasi variabel dan *function* dalam bahasa pemrograman yang umum.

3.3.3 *API Gateway*

Dalam arsitektur, *API Gateway* berperan sebagai gerbang yang dimana akan menerima semua *request* yang dikirim oleh *client*. *API Gateway* dalam aplikasi SuruhAkuAja berperan untuk menerima *request HTTP* dan meneruskannya ke *service* terkait dalam bentuk gRPC. Selain itu, *API Gateway* juga bertugas untuk mentransformasikan *response* yang sebelumnya berbentuk gRPC menjadi bentuk HTTP kembali.

```
ginEngine := gin.Default()
    ginEngine.Use(gin.Recovery())
    ginEngine.Use(exception.Interceptor())
    rootRouterGroup := ginEngine.Group("/")
    userHandler := handler.NewUserHandler(consulServiceRegistry)
    categoryHandler := handler.NewCategoryHandler(consulServiceRegistry)
    jobHandler := handler.NewJobHandler(consulServiceRegistry)
    workerHandler := handler.NewWorkerHandler(consulServiceRegistry)
    authenticationRoutes := routes.NewAuthenticationRoutes(rootRouterGroup, userHandler)
    jobApplicationHandler := handler.NewJobApplicationHandler(consulServiceRegistry)
    protectedRoutes := routes.NewProtectedRoutes(rootRouterGroup, categoryHandler, jobHandler,
workerHandler, jobApplicationHandler, viperConfig)
    authenticationRoutes.Setup()
    protectedRoutes.Setup()
    ginError := ginEngine.Run(":8080")
    if ginError != nil {
        panic(ginError)
    }
```

Gambar 7. Inisialisasi *HTTP Server*

Pada gambar 7 ditampilkan sebuah kode yang merupakan implementasi *server web* menggunakan Gin dalam bahasa Go. Server dikonfigurasi dengan *middleware* untuk menangani *error* dan *logging*, serta memiliki beberapa *handler* yang mengelola entitas seperti pengguna, kategori, pekerjaan, dan pekerja. Selain itu, pada *API Gateaway* juga sebuah token JWT dicek validitasnya, jika tidak valid maka *request* tidak akan diteruskan ke *service* terkait. Setelah semua konfigurasi selesai, *server* dijalankan pada *port* 8080. Jika *client* ingin mengirimkan *request* ke aplikasi maka *client* hanya perlu kirim ke port 8080, tidak ke masing-masing *service*.

3.3.4 *Publish-Subscribe Communication*

Arsitektur *microservices* SuruhAkuAja tidak hanya menggunakan gRPC untuk komunikasi antar layanan, tetapi juga memanfaatkan metode *Publish-Subscribe*. Pendekatan ini digunakan dalam proses pembayaran oleh pembuat pekerja sampingan. Saat pembayaran berhasil dan Midtrans mengirimkan *request* ke *API* yang telah disiapkan oleh SuruhAkuAja, data pemesanan akan dikirim ke RabbitMQ dengan topik *OrderUpdate*.

Gambar 8. *Message* pada *Topic OrderUpdate*

Gambar 8 menunjukkan *message* yang dikirim melalui *RabbitMQ* dengan protokol AMQP. Pesan ini dikirim menuju topic *OrderUpdate*, yang memberikan informasi mengenai pembaruan status suatu pesanan. *Message* pada *topic* tersebut berisi *payload* dengan data dalam bentuk JSON. Dengan menggunakan mekanisme *publish-subscribe*, service lain yang memerlukan informasi pembaruan status *order* dapat merespon secara otomatis. Pendekatan ini lebih efisien karena service yang mempublikasikan *message* tidak perlu mengirimkan data secara manual ke masing-masing *service* yang membutuhkannya, melainkan cukup mengirimkan satu *message* yang dapat dikonsumsi oleh berbagai *service* sesuai kebutuhan.

3.3.5 Integrasi dengan Layanan *Third-Party Reverse Geocoding*

Dalam aplikasi SuruhAkuAja, pengguna yang ingin memposting sebuah pekerjaan diperlukan untuk mengirimkan lokasi pekerjaan tersebut dalam bentuk koordinat yang direpresentasikan dalam longitude dan latitude. Kedua informasi koordinat tersebut nantinya akan ditranslasi menjadi sebuah alamat yang mudah dibaca dengan teknik *Reverse Geocoding*. *Reverse Geocoding* merupakan proses konversi titik koordinat suatu lokasi menjadi alamat atau nama tempat yang dapat dikenali [29]. Untuk melakukan *Reverse Geocoding*, aplikasi SuruhAkuAja terintegrasi dengan layanan *third-party* yang bernama Nominatim. Untuk terintegrasi, aplikasi SuruhAkuAja perlu melakukan *HTTP call* ke *REST API* Nominatim.

```
func (userAddressServiceImpl *UserAddressServiceImpl) Create(ctx context.Context, createUserAddressDto *dto.CreateUserAddressDto) uint64 {
    reverseResponse, err := userAddressServiceImpl.nominatimHttpClient.HTTPClient.Get(fmt.Sprintf("%s/reverse?lat=%f&lon=%f&format=json",
*userAddressServiceImpl.nominatimHttpClient.BaseURL, createUserAddressDto.Latitude, createUserAddressDto.Longitude))
    helper.CheckErrorOperation(err, exception.NewClientError(http.StatusBadRequest, exception.ErrBadRequest, errors.New("failed call
reverse geocoding")))
    responseBody, readErr := io.ReadAll(reverseResponse.Body)
    helper.CheckErrorOperation(readErr, exception.NewClientError(http.StatusBadRequest, exception.ErrBadRequest, errors.New("failed call
reverse geocoding")))
    var userLocation dto.UserLocation
    jsonErr := json.Unmarshal(responseBody, &userLocation)
    helper.CheckErrorOperation(jsonErr, exception.NewClientError(http.StatusBadRequest, exception.ErrBadRequest, errors.New("failed call
reverse geocoding")))
    userAddress := mapper.MapLocationToUserAddress(userLocation, createUserAddressDto.UserId)
    err = userAddressServiceImpl.dbConnection.Transaction(func(gormTransaction *gorm.DB) error {
        userAddressServiceImpl.userAddressRepository.Store(gormTransaction, &userAddress)
        return nil
    })
    helper.CheckErrorOperation(err, exception.ParseGormError(err))
    return userAddress.ID
}
```

Gambar 9. Penggunaan *API* Nominatim

Gambar di atas menampilkan kode yang melakukan *HTTP call* ke *REST API* milik Nominatim. *HTTP call* yang dilakukan menggunakan method *GET* dengan *endpoint* "reverse?lat=${x}&lon=${y}&format=json". Pada *endpoint* tersebut, aplikasi SuruhAkuAja wajib mengirimkan data *latitude* dan *longitude* dalam *endpoint*. Setelah

berhasil melakukan *HTTP call*, *response* yang dikembalikan berbentuk *JSON*. Agar program dapat memproses data lebih jauh maka perlu dikonversi terlebih dahulu dari *JSON* menjadi *struct* yang telah didefinisikan.

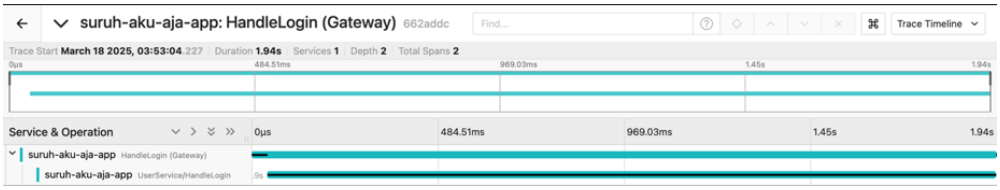
3.3.6 *Tracing* dengan *OpenTelemetry*

Pada sebuah aplikasi berasitektur *microservices*, *request* yang diterima oleh aplikasi bisa diproses oleh satu atau lebih *service* yang ada. Hal ini menyulitkan proses *debugging* karena sumber permasalahan dapat tersebar di berbagai *service*. Selain itu, karena layanan yang terpisah, mengukur performa aplikasi secara mendetail menjadi lebih sulit, terutama dalam mengidentifikasi proses mana yang menyebabkan aplikasi berjalan lambat. Oleh karena itu, diperlukan sebuah alat yang dapat melakukan *tracing* performa dari *request* serta mengidentifikasi layanan mana saja yang memproses *request* tersebut.

```
{
    "receivers": {"otlp": {"protocols": {"http": {"endpoint": "0.0.0.0:4318"}}},
    "exporters": {"logging": {"loglevel": "debug"}, "jaeger": {"endpoint": "http://jaeger:14268/api/traces", "tls": {"insecure": true}}},
    "service": {"pipelines": {"traces": {"receivers": ["otlp"], "exporters": ["logging", "jaeger"]}}}}
}
```

Gambar 10. Konfigurasi *OpenTelemetry*

Pada gambar 10 terdapat konfigurasi dari OpenTelemetry Collector, yang digunakan untuk mengumpulkan, memproses, dan mengekspor data telemetry seperti tracing, logging, dan metrik. OpenTelemetry Collector bertindak sebagai perantara yang menangani data observabilitas sebelum dikirim ke tujuan akhir seperti Jaeger atau sistem logging lainnya. Dalam konfigurasi tersebut, terdapat beberapa komponen, yakni *receivers*, *exporters*, *dan service*. *Receiver* bertugas untuk menerima data *telemetry*. Pada konfigurasi, terdapat *receiver* OTLP (*OpenTelemetry Protocol*) melalui protokol HTTP dengan *endpoint* 0.0.0.0:4318 yang berarti data *trace* dari aplikasi dikirimkan menggunakan OTLP melalui HTTP. Kemudian, terdapat bagian "*exporters*" yang bertanggung jawab untuk mengirimkan data yang diterima ke berbagai tujuan yang dimana dikonfigurasikan tujuannya ke *log* dengan *level debug* dan ke Jaeger melalui *endpoint* http://jaeger:14268/*api/traces*. Bagian terakhir adalah *service*, yang mendefinisikan *pipeline* pemrosesan untuk *tracing*. Pipeline ini menghubungkan *receiver* dengan *exporter*, di mana data yang diterima dari OTLP akan diproses dan dikirimkan ke dua tujuan sekaligus, yakni *logging* dan Jaeger.



Gambar 11. Visualisasi Data *Tracing* di Jaeger

Gambar ini menunjukkan *trace* dari proses *login*, dengan dua *span* utama yang merepresentasikan operasi tertentu dalam aplikasi, yaitu pemrosesan di *Gateway* dan *UserService*. *Trace* ini berdurasi 1,94 detik, dimulai di *Gateway* (*HandleLogin*) selama sekitar 484,5 ms, sebelum diteruskan ke *UserService* (*HandleLogin*) hingga selesai. Visualisasi ini membantu memahami bagaimana *request login* diproses dan mengidentifikasi potensi latensi di tiap tahap.

3.4 Pengujian

Tahap pengujian diawali dengan menyusun skenario *test case* berdasarkan hasil yang diharapkan dari sistem. Selanjutnya, beberapa *payload* dalam bentuk *JSON* akan dikirim untuk mengevaluasi apakah *response* aplikasi sesuai dengan ekspektasi atau tidak. Jika terdapat hasil yang tidak sesuai, maka perbaikan dapat dilakukan untuk memastikan sistem berfungsi dengan baik. Pengiriman *payload JSON* dibantu dengan menggunakan *tools HTTP client* yang bernama *Postman*.

3.4.1 Skenario *Test Case*

Agar proses pengujian memiliki hasil akhir yang pasti dan tetap, maka diperlukan pembuatan *test case* yang memuat hal yang diujikan beserta hasil yang diharapkan. Pada tabel 1 didefinisikan *test case* dan hasil yang diharapkan atau yang seharusnya muncul pada aplikasi SuruhAkuAja. Secara garis besar, Tabel 1 berisi hal yang

perlu diperiksa untuk menguji apakah sudah sesuai dengan kebutuhan yang diinginkan atau masih perlu dilakukan revisi. Berikut beberapa rancangan *test case* yang akan diberikan kode P01-n untuk mewakili setiap pengujian.

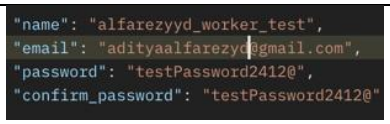
Tabel 1. Tabel *Test Case*

Kode	Test Case	Hasil yang Diharapkan
P01	Melakukan *login* dengan mengisi semua *field* dengan benar	Menerima *request login* dan mengembalikan *response* berupa *JSON* yang berisi token JWT.
P02	Melakukan registrasi dengan mengisi semua *field* dengan benar	Menerima *request* registrasi dan mengembalikan *response status true*.
P03	Melakukan pembuatan data kategori dengan mengirimkan *payload* yang valid	Menerima *request* pembuatan data kategori dan mengembalikan *response status true*.
P04	Melakukan pembuatan data *job* dengan mengirimkan *payload* yang valid	Menerima *request* pembuatan data job dan mengembalikan *response status true*.
P05	Melakukan pendaftaran untuk menjadi *worker* dengan mengirimkan *payload* yang valid	Menerima *request* registrasi *worker* dan mengembalikan *response status true*.
P06	Melakukan pembayaran terhadap *worker* ketika selesai bekerja	Mengalihkan pengguna ke halaman pembayaran Midtrans *Payment Gateway*
P07	Membuat *review* ketika pekerjaan telah selesai dengan mengirimkan *payload* yang valid	Menerima *request* pembuatan data *review* dan mengembalikan *response status true*.
P08	Melakukan penarikan saldo yang dimiliki oleh *worker*	Menerima *request* penarikan saldo dan mengembalikan *response status true*.

3.4.2 Pengujian Test Case

Setelah semua *test case* telah direncanakan dan didefinisikan maka selanjutnya melakukan pengujian tersebut sesuai dengan instrumen *test case*. Pengujian tersebut dilakukan menggunakan data uji sebanyak n dengan kode yang digunakan merupakan P01-P10. Dari pengujian tersebut, hasilnya akan diberikan kode T01-n. Dari hasil pengujian tersebut dapat ditarik kesimpulan apakah aplikasi sudah berjalan sesuai dengan yang diharapkan.

Tabel 2. Tabel Pengujian *Test Case*

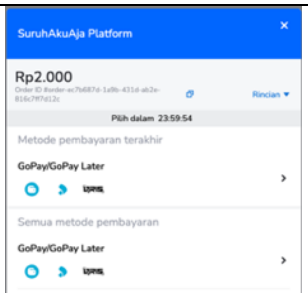
Kode	*Test Case*	Hasil yang Diharapkan	Hasil Pengujian
T01	1. Atur *endpoint* menuju /*authentication*/*login* 2. Atur *HTTP method* menjadi *POST* 3. Atur *body* dengan pilihan *raw* 4. Isi *payload* yang valid dengan akun yang terdaftar	Menerima *request login* dan mengembalikan *response* berupa *JSON* yang berisi token JWT.	 **Gambar 12.** *Response Login* Berhasil
T02	1. Atur *endpoint* menuju /*authentication*/*register* 2. Atur *HTTP method* menjadi *POST* 3. Atur *body* dengan pilihan *raw* 4. Kirim *payload* valid	Menerima *request* registrasi dan mengembalikan *response status true*.	 **Gambar 13.** *Response Register* Berhasil
T03	1. Atur *endpoint* menuju /*category* 2. Atur *HTTP method* menjadi *POST* 3. Atur *body* dengan pilihan *raw* 4. Kirim *payload* valid	Menerima *request* pembuatan data kategori dan mengembalikan *response status true*.	 **Gambar 14.** *Response* Pembuatan Kategori Berhasil

T04	1. Atur *endpoint* menuju */jobs* 2. Atur *HTTP method* menjadi *POST* 3. Atur *body* dengan pilihan *raw* 4. Kirim *payload* valid	Menerima *request* pembuatan data *job* dan mengembalikan *response status true*.	{"status": true, "message": "Create job success", "data": null}

Gambar 15. *Response* Pembuatan *Job* Berhasil

T05	1. Atur *endpoint* menuju */workers* 2. Atur *HTTP method* menjadi *POST* 3. Atur *body* dengan pilihan *raw* 4. Kirim *payload* valid	Menerima *request* registrasi *worker* dan mengembalikan *response status true*.	{"status": true, "message": "Register for worker success", "data": null}

Gambar 16. *Response* Pendaftaran *Worker* Berhasil

T06	1. Atur *endpoint* menuju */transactions* 2. Atur *HTTP method* menjadi *POST* 3. Atur *body* dengan pilihan *raw* 4. Kirim *payload* valid	Mengalihkan pengguna ke halaman pembayaran Midtrans *Payment Gateway*	

Gambar 17. Halaman Pembayaran Midtrans

T07	1. Atur *endpoint* menuju */reviews* 2. Atur *HTTP method* menjadi *POST* 3. Atur *body* dengan pilihan *raw* 4. Kirim *payload* valid	Menerima *request* pembuatan data *review* dan mengembalikan *response status true*.	{"status": true, "message": "Review has been created", "data": null}

Gambar 18. *Response* Pembuatan *Review* Berhasil

T08	1. Atur *endpoint* menuju */withdrawals* 2. Atur *HTTP method* menjadi *POST* 3. Atur *body* dengan pilihan *raw* 4. Kirim *payload* valid	Menerima *request* penarikan saldo dan mengembalikan *response status true*.	{"status": true, "message": "Create withdrawal success", "data": null}

Gambar 19. *Response* Pembuatan *Withdrawal* Berhasil

3.4.3 Hasil Pengujian

Berdasarkan hasil pengujian *test case*, diperoleh hasil uji seperti yang terlihat pada tabel 3 dan dapat dikemukakan hasilnya sebagai berikut:

Tabel 3. Hasil Pengujian

No	Test Case	Hasil yang Diharapkan	Kesimpulan
1	P01	T01	Sesuai
2	P02	T02	Sesuai
3	P03	T03	Sesuai
4	P04	T04	Sesuai
5	P05	T05	Sesuai
6	P06	T06	Sesuai
7	P07	T07	Sesuai
8	P08	T08	Sesuai

Hasil pengujian pada Tabel 3 yang telah dilakukan menggunakan metode Equivalence Partitions dengan membuat instrumen *test case* yang berjumlah 8 pengujian dari REST API SuruhAkuAja diperoleh 8 pengujian sesuai dan 0 pengujian yang belum sesuai.

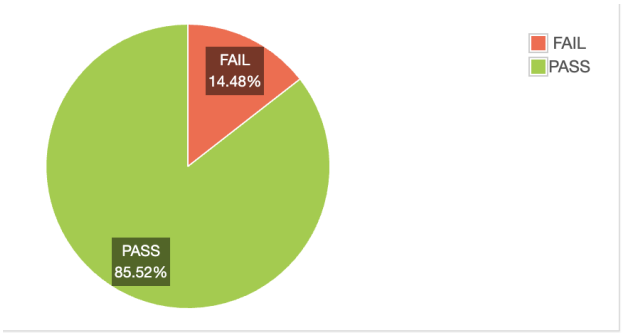
3.4.4 Pengujian Performa dengan *Load Testing*

Hal yang penting dalam mengukur kualitas REST API adalah dengan melakukan pengujian performa yang dimana salah satu metodenya adalah *Load Testing*. Pengujian tersebut dapat dibantu dengan *tools* yang bernama Apache JMeter [30]. Hasil pengujian performa dari beberapa *endpoint* tersebut ditampilkan dalam tabel berikut:

Tabel 4. Hasil Pengujian Performa

No	*Endpoint*	*Method*	*Concurrent User*	*Average Response Time (ms)*	*Throughput (req/sec)*	*Error Rate (%)*
1	/authentication/login	POST	75	8706.05	7,52	0
2	/authentication/register	POST	75	10625,76	1,79	76
3	/public/jobs	GET	75	2548,52	5,60	0
4	/categories	GET	75	1413,53	8,93	0
5	/categories	POST	75	1089,45	2,90	0
6	/job-applications	POST	75	3221,41	2,79	1,33
7	/transactions	POST	75	11752,77	2,65	24

Dari detail tabel yang disajikan di atas, persentase *request* berhasil dan tidak berhasil dapat disajikan dalam bentuk *pie chart* seperti berikut:



Gambar 20. Persentasi *Request* Berhasil dan Tidak Berhasil

Dalam pie chart tersebut terdapat hampir 14,48% request yang dinyatakan gagal, dimana request menuju endpoint register memiliki error rate paling tinggi. Informasi detail mengenai error dapat disajikan pada tabel berikut:

Tabel 5. Informasi Detail *Error*

No	Tipe *Error*	Jumlah *Error*	Persentase *Error* (%)	Persentase dari Semua Sampel (%)
1	504/Gateway Timeout	57	75	10,86
2	400/Bad Request	18	23,86	3,43
3	500/Internal Server Error	1	1,32	0,19

Berdasarkan hasil pengujian performa yang telah dilakukan menggunakan Apache JMeter, secara umum REST API telah memberikan performa yang cukup baik, dengan persentase keberhasilan (*request* berhasil) sebesar 85,52%, dan 14,48% *request* mengalami kegagalan. Waktu respons rata-rata juga menunjukkan adanya variasi yang cukup signifikan, dengan nilai tertinggi pada endpoint /transactions (11752,77 ms) dan /authentication/register (10625,76 ms), yang menunjukkan bahwa kedua endpoint ini memerlukan perhatian lebih dari sisi optimasi performa backend atau kapasitas server.

4. KESIMPULAN

Penelitian ini membahas pengembangan *REST API* untuk aplikasi pencarian pekerjaan sampingan dengan menggunakan arsitektur *microservices* dan metode *Waterfall*. Latar belakang penelitian ini didasarkan pada tingginya tingkat pengangguran di Indonesia serta kebutuhan akan platform yang dapat mempermudah masyarakat dalam mencari pekerjaan sampingan.

Melalui pendekatan pengembangan perangkat lunak berbasis metode Waterfall, seluruh tahapan yang mencakup perencanaan sistem, perancangan arsitektur, implementasi, hingga pengujian telah berhasil dilalui dan diuji dengan hasil yang sesuai spesifikasi. Hasil pengujian menunjukkan bahwa semua fitur utama, termasuk proses login, registrasi, pengelolaan kategori dan pekerjaan, transaksi pembayaran, penilaian (review), hingga penarikan saldo telah berjalan sesuai dengan ekspektasi sistem. Hal ini membuktikan bahwa sistem mampu mendukung proses bisnis dengan baik serta dapat digunakan secara fungsional.

Dari sisi teknis, arsitektur *microservices* yang diterapkan terbukti mendukung fleksibilitas, skalabilitas, dan efisiensi layanan. Komunikasi antar service menggunakan gRPC dan RabbitMQ telah berhasil mendukung interaksi antar komponen secara optimal. Selain itu, sistem telah terintegrasi dengan layanan pihak ketiga seperti Midtrans untuk pembayaran dan Nominatim untuk pencarian lokasi. Proses pemantauan sistem juga ditingkatkan dengan integrasi OpenTelemetry untuk pelacakan performa layanan.

Secara keseluruhan, dapat disimpulkan bahwa sistem ini efektif dalam menjawab rumusan masalah dan tujuan penelitian, yaitu menciptakan platform digital yang mampu mengatasi kendala dalam pencarian pekerjaan sampingan yang tersebar dan tidak terstruktur. Aplikasi SuruhAkuAja terbukti mampu menyediakan solusi terpusat yang aman, mudah digunakan, serta mampu menghubungkan pencari kerja dan pemberi kerja secara langsung berdasarkan kebutuhan, keterampilan, dan lokasi.

Untuk pengembangan selanjutnya, penulis menyarankan agar sistem ini diuji langsung oleh pengguna akhir agar diperoleh masukan nyata terkait kebutuhan pengguna atau diintegrasikan dengan antarmuka *website*. Fitur tambahan seperti rekomendasi pekerjaan berdasarkan keterampilan, sistem penjadwalan otomatis, dan notifikasi pekerjaan yang relevan juga dapat dipertimbangkan untuk meningkatkan pengalaman pengguna. Selain itu, diperlukan pengujian performa dalam skala besar guna memastikan kestabilan arsitektur *microservices* ketika menangani banyak pengguna secara bersamaan.

REFERENCES

[1] Badan Pusat Statistik (BPS), "Hasil Sensus Penduduk 2020," Jan 2021. Diakses: 15 Maret 2025. [Daring]. Tersedia pada: https://www.bps.go.id/id/pressrelease/2021/01/21/1854/hasil-sensus-penduduk--sp2020--pada-september-2020-mencatat-jumlah-penduduk-sebesar-270-20-juta-jiwa-.html

[2] "Keadaan Ketenagakerjaan Indonesia Agustus 2020," Nov 2020. Diakses: 15 Maret 2025. [Daring]. Tersedia pada: https://www.bps.go.id/id/pressrelease/2020/11/05/1673/-revisi-per-18-02-2021--agustus-2020-tingkat-pengangguran-terbuka--tpt--sebesar-7-07-persen.html

[3] D. Intan Permatasari, M. Udin Harun Al Rasyid, Y. Rizki Nusantoko, P. Elektronika Negeri Surabaya, dan P. Korespondensi, "APLIKASI PENCARI KERJA SAMPINGAN BERBASIS FRAMEWORK FLUTTER," *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK)*, vol. 9, no. 4, hlm. 669–674, Agu 2022, doi: 10.25126/jtiik.202295758.

[4] M. Avied Bachmid, M. Tri Ananta, dan K. C. Brata, "Pengembangan Aplikasi Pencarian dan Penawaran Kerja Paruh Waktu untuk Usaha Mikro, Kecil, dan Menengah (UMKM) berbasis Progressive Web App (Studi Kasus Kota Malang)," 2022. [Daring]. Tersedia pada: http://j-ptiik.ub.ac.id

[5] D. Prasetyawan dan P. D. Rahmanto, "Pengembangan Sistem Seleksi Proposal Penelitian Berbasis Web Service Menggunakan REST API," *JTIM : Jurnal Teknologi Informasi dan Multimedia*, vol. 6, no. 3, hlm. 283–295, Sep 2024, doi: 10.35746/jtim.v6i3.585.

[6] P. Satya Saputra dan L. Putu Ary Sri Tjahyanti, "PEMANFAATAN TEKNOLOGI INFORMASI MENGGUNAKAN WEB API DI MASA PANDEMI COVID-19," 2022.

[7] H. Fery Herdiyatmoko, "Desain Sistem Backend Berbasis REST API Menggunakan Framework Laravel 7," *SKANIKA: Sistem Komputer dan Teknik Informatika*, vol. 5, no. 2, hlm. 136–144, 2022, doi: https://doi.org/10.36080/skanika.v5i2.2947.

[8] M. N. Esa dan A. Voutama, "RANCANG BANGUN REST API APLIKASI PENCATATAN KEUANGAN BERBASIS WEB MENGGUNAKAN PENERAPAN MONGOOSE," *JATI (Jurnal Mahasiswa Teknik Informatika)*, vol. 8, no. 3, hlm. 4048–4054, Jun 2024, doi: https://doi.org/10.36040/jati.v8i3.9815.

[9] N. A. Prayogo, "Belajar Golang - Dasar Pemrograman Golang." Diakses: 15 Maret 2025. [Daring]. Tersedia pada: https://dasarpemrogramangolang.novalagung.com/1-berkenalan-dengan-golang.html

[10] "The Go Programming Language." Diakses: 15 Maret 2025. [Daring]. Tersedia pada: https://go.dev/

[11] "PostgreSQL: About." Diakses: 15 Maret 2025. [Daring]. Tersedia pada: https://www.postgresql.org/about/

[12] A. Sinambela dan F. Farady Coastera, "IMPLEMENTASI ARSITEKTUR MICROSERVICES PADA RANCANG BANGUN APLIKASI MARKETPLACE BERBASIS WEB," 2021. [Daring]. Tersedia pada: http://ejournal.unib.ac.id/index.php/rekursif/1

[13] S. Atmojo, R. Utami, S. Dewi, dan N. Widhiyanta, "Implementasi Sistem-informasi Desa Berbasis Arsitektur Microservices," *SMATIKA JURNAL*, vol. 12, no. 01, hlm. 55–66, Jun 2022, doi: 10.32664/smatika.v12i01.658.

[14] V. Adi Kurniyanti dan D. Murdiani, "Perbandingan Model Waterfall Dengan Prototype Pada Pengembangan System Informasi Berbasis Website," *Jurnal Syntax Fusion*, vol. 2, no. 08, hlm. 669–675, Agu 2022, doi: 10.54543/fusion.v2i08.210.

[15] R. Susanto dan A. D. Andriana, "PERBANDINGAN MODEL WATERFALL DAN PROTOTYPING UNTUK PENGEMBANGAN SISTEM INFORMASI."

[16] R. W. Saputra *dkk.*, "ANALISIS RESIKO PENGGUNAAN METODE WATERFALL DAN PROTOTYPING DALAM PENGEMBANGAN WEBSITE," 2024. doi: https://doi.org/10.36040/jati.v8i4.9943.

[17] A. Nurseptaji, "IMPLEMENTASI METODE WATERFALL PADA PERANCANGAN SISTEM INFORMASI PERPUSTAKAAN," *Jurnal Dialektika Informatika (Detika)*, vol. 1, no. 2, hlm. 49–57, Mei 2021, doi: 10.24176/detika.v1i2.6101.

[18] P. Studi Manajemen Informatika, N. Hotdiana Simanullang, dan A. Wardah Bilah Siregar, "SISTEM INFORMASI PEMESANAN MENU MAKANAN PADA RM SEDEP ROSO RANTAUPRAPAT BERBASIS WEB," 2021. doi: https://doi.org/10.36987/josdim.v1i1.2175.

[19] Y. Primadasa, H. Juliansa, S. Informasi, S. Bina Nusantara Jaya Lubuklinggau, J. Yos Sudarso No, dan A. Kelurahaan Jawa Kanan Kota Lubuklinggau, "Rancang Bangun Sistem E-Discussion Untuk Mahasiswa Kota Lubuklinggau Designing An E-Discussion System For Students Of Lubuklinggau City," *Cogito Smart Journal* |, vol. 6, no. 2, 2020, doi: https://doi.org/10.31154/cogito.v6i2.262.310-322.

[20] S. Rizal dan S. A. Saputra, "PERANCANGAN UI/UX DESIGN PADA APLIKASI JASA FREELANCER BERBASIS ANDROID MENGGUNAKAN METODE USER CENTERED DESIGN," *Jurnal Ilmiah MATRIK*, vol. 25, no. 1, Apr 2023, doi: https://doi.org/10.33557/jurnalmatrik.v25i1.2279.

[21] E. Suharyanto, M. Kom, S. Program, dan I. Sistem, "PERANCANGAN APLIKASI PENGENALAN BUDAYA NUSANTARA BERBASIS ANDROID DENGAN METODE RAD," *Jurnal Ilmu Komputer JIK*, hlm. 2022.

[22] "Diagramming, Data Visualization and Real-Time Collaboration | Lucidchart." Diakses: 16 Maret 2025. [Daring]. Tersedia pada: https://www.lucidchart.com/pages/product

[23] A. Putri Yulandi, "Analisis Performa Backend Framework: Studi Komparasi Framework Golang dan Node.js," *Jurnal Riset Sistem Informasi Dan Teknik Informatika (JURASIK)*, vol. 8, no. 1, hlm. 155–168, Feb 2023, doi: http://dx.doi.org/10.30645/jurasik.v8i1.551.

[24] E. Novalia dan A. Voutama, "Black Box Testing dengan Teknik Equivalence Partitions Pada Aplikasi Android M-Magazine Mading Sekolah," 2022. doi: https://doi.org/10.30865/klik.v4i4.1603.

[25] Elis dan A. Voutama, "PEMANFAATAN UML (UNIFIED MODELING LANGUAGE) DALAM PERENCANAAN SISTEM PENYEWAAN BAJU ADAT BERBASIS WEBSITE," *Jurnal Informatika, Manajemen dan Komputer*, vol. 14, no. 2, hlm. 26–35, Des 2022, doi: http://dx.doi.org/10.36723/juri.v14i2.445.

[26] C. Ayu Binangkit, A. Voutama, dan N. Heryana, "PEMANFAATAN UML (UNIFIED MODELING LANGUAGE) DALAM PERENCANAAN SISTEM PENGELOLAAN SEWA ALAT MUSIK BERBASIS WEBSITE," 2023. doi: https://doi.org/10.36040/jati.v7i2.6858.

[27] M. Niswar, R. A. Safruddin, A. Bustamin, dan I. Aswad, "Performance evaluation of microservices communication with REST, GraphQL, and gRPC," *International Journal of Electronics and Telecommunications*, vol. 70, no. 2, hlm. 429–436, Jun 2024, doi: 10.24425/ijet.2024.149562.

[28] M. Avatara dan R. Tan, "Implementasi Framework Gin dan gRPC pada Pengembangan Back-end Web," 2024.

[29] E. Sulistyo, M. I. Bayu P, F. Andini, dan I. Dwisaputra, "Implementasi Metode Reverse Geocoding pada Aplikasi Tracking Posisi," *Emitor: Jurnal Teknik Elektro*, vol. 1, no. 1, hlm. 44–49, Mar 2023, doi: 10.23917/emitor.v1i1.21464.

[30] C. Putri Agustika dan W. S. Saputra, "PENGUJIAN APLIKASI GREENWALLET DENGAN METODE LOAD TESTING DAN APACHE JMETER," *Jurnal Informatika dan Sistem Informasi (JIFoSI)*, vol. 2, no. 2, hlm. 190–195, 2021.