Implementasi Arsitektur *Microservice* pada Aplikasi Web Pengajaran Agama Islam Home Pesantren

Yuri Chandra Tri Putra[1], Thomas Adi Purnomo Sidi[2], Joseph Eric Samodra[3]
Program Studi Informatika, Fakultas Teknologi Industri, Universitas Atma Jaya Yogyakarta
Jl. Babarsari No. 43, Caturtunggal, Kab. Sleman, 55281, Daerah Istimewa Yogyakarta, Indonesia
Email: [1]yurichandra@gmail.com, [2]thomas.adi.ps@uajy.ac.id, [3]eric.samodra@uajy.ac.id

Abstrak. *Aplikasi web pengajaran agama Islam Home Pesantren adalah aplikasi berbasis web yang dapat digunakan untuk mendapatkan konten-konten pengajaran agama Islam. Dalam tahap pengembangan, aplikasi web Home Pesantren menggunakan arsitektur monolitik. Dengan arsitektur tersebut, seluruh struktur proyek ditempatkan dalam satu tempat yang sama. Selain itu, arsitektur monolitik yang diterapkan pada aplikasi web Home Pesantren memiliki kesulitan dalam skalabilitas, karena arsitektur tersebut mengharuskan keseluruhan proyek untuk ditingkatkan ketika kebutuhan skalabilitas menjadi tinggi. Tentu saja, penggunaan sumber daya akan menjadi tidak efisien. Penerapan arsitektur microservice diharapkan dapat meningkatkan modularitas serta kemudahan skalabilitas aplikasi. Penerapan arsitektur microservice akan didukung dengan HTTP dan event driven communication sebagai jembatan intercommunication antar microservice. Serta pengunaan tools docker swarm diharapkan dapat membantu penerapan microservice sehingga skalabilitas dapat dicapai dengan lebih mudah.*

Kata Kunci: *Microservice, arsitektur, aplikasi web, skalabilitas*

.

1. Pendahuluan

Codebase dapat terus bertumbuh dan menjadi besar ketika penambahan fitur terus terjadi secara terus-menerus. Pertumbuhan *codebase* yang terjadi secara terus-menerus [1], dapat membuat pengembang menghadapi kesulitan untuk mengetahui dimanakah letak perubahan akan dilakukan ketika *codebase* sudah bertumbuh besar. *Codebase* yang besar dapat terjadi ketika pengembangan perangkat lunak dilakukan dengan menggunakan arsitektur monolitik. Arsitektur monolitik membuat semua layanan berada pada satu *codebase* yang sama [2].

Home Pesantren (Hompes) merupakan aplikasi pengajaran agama Islam berbasis web yang menggunakan arsitektur *monolitik* pada tahap pengembangan. Seiring dengan pengembangan Hompes, *codebase* yang ada pada Hompes juga bertumbuh dengan pesat serta melahirkan *dependency* antar modul yang membuat pengembangan untuk fitur berikutnya menjadi sulit. Kesulitan lain yang dihadapi saat pengembangan Home Pesantren adalah waktu yang cukup lama dalam proses pengembangan. Waktu yang cukup lama ini mencakup proses pemahaman *codebase* yang sudah ada, sehingga produktivitas *delivery* produk akhir menjadi berkurang. Selain itu, aplikasi Home Pesantren yang dikembangkan dengan arsitektur monolitik akan sulit dan tidak efisien dalam hal sumber daya untuk *scale* jika pertumbuhan pengguna menjadi tinggi, ini disebabkan *scale* pada arsitektur *monolitik* mengharuskan keseluruhan sistem harus di-*scale* secara bersamaan, padahal hanya beberapa komponen atau bagian saja yang memerlukan *scale*.

Seiring dengan berkembangnya arsitektur perangkat lunak, kemunculan arsitektur *microservice* menjadi sebuah tren yang digunakan oleh pengembang dalam beberapa tahun terakhir. *Microservice* memberikan keleluasaan bagi pengembang untuk mengembangkan perangkat lunak dalam waktu yang cepat. Berkembangnya arsitektur *microservice* juga didukung oleh kurang handalnya arsitektur monolitik dalam menangani *system failure*. Karena dalam satu aplikasi memiliki satu *codebase* yang sama, merupakan hal yang pasti bahwa arsitektur monolitik juga menjadi *single point of failure* yang akan terjadi jika salah layanan terjadi malfungsi atau *error* [2]. Keunggulan arsitektur *microservice* lainnya adalah perubahan

yang terjadi pada satu layanan dalam frekuensi yang tinggi akan tetap membuat sistem secara keseluruhan bekerja dengan baik tanpa adanya perubahan yang signifikan [3]. Sementara itu arsitektur *microservice* memungkinkan untuk sistem dibangun dengan teknologi yang berbeda-beda. Tujuannya adalah untuk menyesuaikan teknologi dengan kebutuhan pada suatu layanan [4]. Keunggulan arsitektur *microservice* juga ada poin *scale.* Pada arsitektur *microservice, scale* dapat dilakukan pada komponen yang membutuhkan *scale* saja, tidak perlu sistem secara menyeluruh [1].

Implementasi arsitektur *microservice* akan dilakukan pada aplikasi web Home Pesantren dengan menggunakan teknologi seperti Golang dan PHP pada bagian *server side* dan Javascript pada *client side.* Penggunaan teknologi pendukung lainnya adalah penggunaan Docker untuk mendukung virtualisasi pada tahap perangkat lunak dan membungkusnya dalam bentuk *container* [5]. Lalu, penggunaan teknologi pendukung lainnya seperti Docker Swarm untuk mendukung *clustering* beberapa *virtual machine* agar dapat distribusi *container microservice.* Komunikasi antar *microservice* juga didukung dengan penggunaan dua metode, yaitu *synchronous* dan *asynchronous.* Bentuk komunikasi dalam model *asynchronous* dilakukan dengan menggunakan *pub/sub* dengan model arsitektur *event-driven communication*. Lalu, untuk *synchronous* dilakukan dalam bentuk HTTP atau gRPC [6]. Harapan dari penelitian ini adalah aplikasi web Home Pesantren dapat mengimplementasikan arsitektur *microservice* serta implementasi komunikasi antar *microservice* berhasil dan dapat dilakukan.

2. Tinjauan Pustaka

Pada penelitian yang melibatkan arsitektur *microservice* dilakukan pada *refactoring* Sistem Manajemen Anggota AISINDO (Asosiasi Profesi Sistem Informasi Indonesia). Penelitian tersebut bertujuan untuk meningkatkan resiliensi untuk kebutuhan kualitas perangkat lunak. *Refactoring* dilakukan dengan mengubah arsitektur perangkat lunak serta penggunaan tools seperti Docker untuk pembangunan arsitektur *microservice*. Hasil dari penelitian ini dapat diamati melalui peningkatan aspek resiliensi sistem yang memampukan ketangguhan sistem ketika salah satu layanan mengalami gangguan. Penelitian ini mendukung penggunaan Eureka dan Zuul yang masing–masing merupakan *service discovery* dan *API gateway* yang memiliki fungsi untuk mencari layanan yang tersedia dan menjadi layanan untuk memetakan *request* kepada salah satu layanan [7].

Pada penelitian berikutnya adalah penerapan arsitektur *microservice* untuk membangun platform *open source* SmartCity. Tujuan penelitian tersebut adalah membuktikan bahwa dampak yang diberikan dengan pembangunan arsitektur *microservice* dapat berdampak pada *scalability* dan *evolveabilty* dari sistem yang dikembangkan. Hasil dari penelitian ini adalah memang benar bahwa arsitektur *microservice* dapat meningkatkan *scalability* dalam sebuah sistem atau perangkat lunak. Ini dibuktikan dengan *scaling* yang dilakukan dapat meningkatkan rata–rata *response time* dari perangkat lunak yang sebelumnya berkisar 1720 *milisecond* menjadi 320 *milisecond* [8].

Penelitian terakhir yang melibatkan penerapan arsitektur *microservice* adalah pengembangan *Student Information System (*RosarioSIS*)*, yang merupakan *open source legacy* yang sebelumnya menerapkan arsitektur monolitik. Tujuan utama dari penerapan arsitektur ini adalah pemanfaatan *single responsibility* dan proses *independent deployment*. Penerapan arsitektur *microservice* pada penelitian ini juga didukung dengan penggunaan Amazon Web Service (AWS) sebagai *cloud environment* untuk kebutuhan *deployment*. Hasil dari penelitian ini adalah arsitektur *microservice* mendukung penerapan *single responsibility* dan *independent deployment* dengan hasil penerapan tiap layanan di-*deploy* pada satu *instance* pada platform Amazon Web Service [9].

Penerapan arsitektur *microservice* pada *platform* Medium dimulai akibat *bottleneck* yang terjadi pada sistem yang sudah berjalan. Beberapa tugas yang dijalankan di *platform* Medium memiliki beban komputasi serta ketergantungan I/O yang tinggi yang dianggap tidak sesuai dengan teknologi yang digunakan saat itu, yaitu Node.js. Hal berikutnya yang menjadi alasan mengapa Medium mengubah arsitektur menjadi *microservice* adalah membuat

pengembangan aplikasi menjadi lama akibat banyaknya *engineer* yang mengembangkan aplikasi dalam satu *codebase* yang sama sehingga menyebabkan *coupling* yang tinggi. Arsitektur *monolitik* juga tidak memberikan ruang bagi *engineer* di Medium untuk melakukan perubahan yang besar terhadap *codebase*. Karena *codebase* di-*deploy* menjadi satu kesatuan, ketika terjadi perubahan yang besar dan terjadi malfungsi pada perubahan yang dilakukan, maka dapat dipastikan sistem akan mengalami malfungsi secara keseluruhan. Arsitektur *microservice* memberikan keleluasaan bagi engineer di Medium untuk mengembangkan layanan yang tidak terikat dengan layanan lain. Penyebab lain dari perubahan arsitektur adalah kesulitan untuk melakukan *scale* terhadap sistem. Karena *codebase* menjadi satu, sulit untuk melakukan isolasi terhadap tugas atau pekerjaan kepada *resource* tertentu. Sehingga penggunaan *resource* menjadi tidak hemat karena tugas yang ringan juga menggunakan satu *resource* yang sama. Hal terakhir adalah keleluasaan yang diberikan arsitektur *microservice* untuk memilih teknologi tertentu dalam memproses tugas tertentu. Hasil penelitian dan penerapan yang dilakukan di Medium menghasilkan peningkatan produktifitas pengembang serta mendukung penggunaan teknologi yang berbeda secara lebih aman [10].

3. Metode Penelitian

Metode penelitian yang digunakan dalam mengembangkan aplikasi ini antara lain sebagai berikut: (1) Kajian Pustaka. Pada tahap ini, penulis mencari bahan pustaka untuk dijadikan referensi dalam pengembangan aplikasi serta menggali informasi terkait pengembangan aplikasi berbasis arsitektur *microservice*. (2) Analisis. Pada tahap ini, penulis menganalisis sistem yang sudah berjalan serta menentukan *boundary context* yang akan digunakan untuk memisahkan layanan dari arsitektur monolitik menjadi *microservice*. (3) Perancangan. Pada tahap ini, penulis merancang perangkat lunak dengan merancang basis data serta komponen komunikasi terkait arsitektur *microservice*. (4) Implementasi perangkat lunak. Pada tahap ini, penulis mengimplementasikan hasil dari tahap perancangan. (5) Pengujian perangkat lunak. Pada tahap ini pengujian akan dilakukan pada tiap *microservice* untuk memastikan bahwa setiap layanan memiliki fungsionalitas yang baik untuk membentuk kesatuan layanan.

4. Hasil dan Diskusi

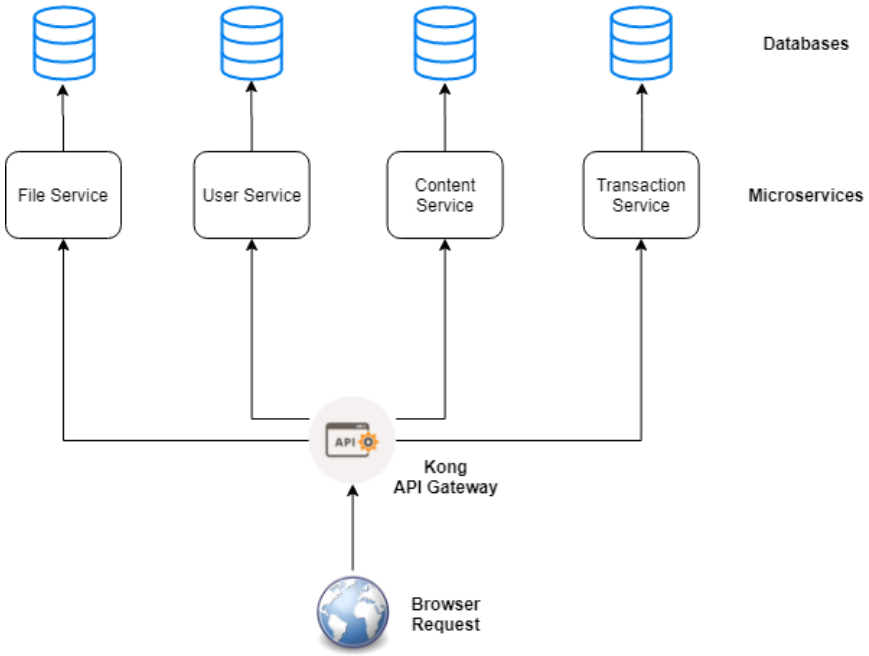
4.1. Analisis Sistem

Aplikasi web Home Pesantren dibangun dengan menggunakan arsitektur *microservice.* Penerapan arsitektur *microservice* didukung oleh dua jenis bahasa pemrograman pada tingkat *server side* yaitu Golang dan PHP. PostgreSQL digunakan sebagai *Database Management System* (DBMS) untuk penyimpanan data. Teknologi berikutnya yang digunakan adalah Kong. Kong adalah *API* Gateway untuk menjadi *entrypoint client* agar dapat mengakses *Application Programming Interface* (API). Penggunaan Kong juga dimanfaatkan sebagai tempat untuk mendaftarkan *microservice* beserta *routing* dari *microservice* yang terdaftar. Proses ini mendukung kebutuhan *client side* saat melakukan *request API*. Ketika *request* terjadi, Kong akan melakukan *reverse proxy* kepada *routes* yang telah terdaftar di dalam Kong *database.* Teknologi berikutnya yang digunakan adalah RabbitMQ. Penggunaan RabbitMQ untuk mendukung *event-based communication* sebagai salah satu bagian *asynchronous communication* untuk komunikasi antar *microservice.*

4.2. Arsitektur Sistem

Perancangan arsitektur pada aplikasi web Home Pesantren terdapat pada Gambar 1. Dengan pengembangan menggunakan arsitektur *microservice*, masing – masing *microservice* seperti *file service, user service, content service* dan *transaction service* memiliki basis datanya masing-masing untuk mendukung *modularity* serta menjamin *single responsibility* dari tiap *microservice.* Masing- masing *microservice* berfungsi sebagai tempat pengolahan data, serta akses basis data berdasarkan modul pengembangan. Sistem *microservice* dibagi menjadi empat bagian *microservice* kecil yang memiliki tugas dan tujuannya masing-masing. *Microservice*

user-service akan bertugas untuk melakukan pengolahan data terkait dengan *user*. *Microservice content-service* bertugas untuk melakukan pengolahan data terkait dengan konten buku, data penerbit, data penulis dan paket buku. *Microservice file-service* bertugas untuk mengolah data terkait *file*. Yang terakhir *microservice transaction-service* bertugas untuk mengolah data terkait dengan transaksi.



Gambar 1. Arsitektur Sistem

Dengan menggunakan *REST API (Representational State Transfer Application Programming Interface)* masing-masing *microservice* mengekspos *routes* yang dapat diakses oleh internal *microservice* lain atau *layer* lain di dalam sistem secara keseluruhan. Pembangunan *microservice* didukung oleh penggunaan bahasa pemrograman *Golang* dan *PHP* dengan framework *Lumen.* Selain menggunakan *REST API,* pembangunan sistem juga didukung oleh penggunaan *tools message broker* yang dalam pengembangan ini menggunakan RabbitMQ untuk mendukung komunikasi *asynchronous* antar *microservice.*

Komponen lain dalam pembangunan Home Pesantren adalah *API gateway*. *API gateway* digunakan sebagai penghubung antara *client request* dengan *microservice* yang telah dibangun. Penggunaan *API gateway* mendukung simplifikasi dan menjadikan *API gateway* sebagai satu-satunya *entrypoint* pada komunikasi antara *client* dan *microservice. API gateway* dalam pengembangan ini juga berfungsi untuk mendaftarkan *microservice* yang terdapat di dalam sistem, selanjutnya *API gateway* akan melakukan *reverse proxy* berdasarkan *routing matching* yang telah dilakukan saat pendaftaran *microservice* pada *API gateway.* Pada pembangunan aplikasi Home Pesantren, *API gateway* yang digunakan adalah *Kong API gateway.*

Komponen berikutnya adalah *browser request.* Komponen ini bertugas sebagai akses untuk *client* agar dapat mengakses sistem. Agar *client* dapat mengakses sistem, pembangunan aplikasi Home Pesantren ini menggunakan dua aplikasi yang dapat dikunjungi, yang pertama untuk pengguna dengan wewenang admin dan yang berikutnya untuk pengguna dengan wewenang penulis dan *customer.* Dua aplikasi tersebut nantinya akan melakukan *API request* ke *microservice* melalui *API gateway.* Pembangunan aplikasi web tersebut menggunakan *Vue.js* sebagai *framework* dengan arsitektur MVVC (*Model – View – View Controller).*

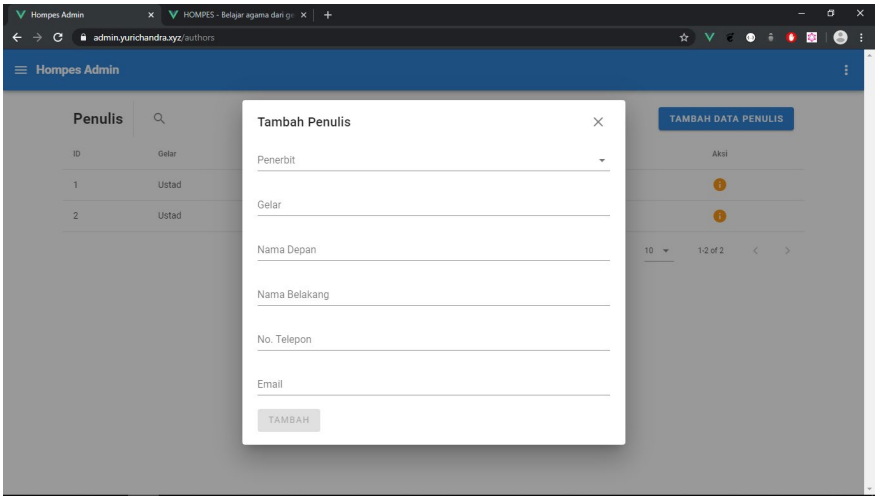
4.3. Implementasi Sistem
4.3.1. Docker Swarm

Gambar 2. Implementasi Docker Swarm

Gambar 2 adalah implementasi penerapan docker swarm dalam bentuk *visualizer* pada implementasi arsitektur *microservice*. *Visualizer* di atas menggambarkan *container* yang terdaftar. *Container* yang terdapat pada implementasi arsitektur di atas terdiri dari komponen *backend* serta layanan pendukung seperti halaman web, halaman admin, basis data, *API gateway*, halaman *API Gateway management,* dan *message broker* RabbitMQ. Penerapan docker swarm di atas menggunakan dua buah *Virtual Private Server* (VPS) atau dua buah *node* yang tugasnya dibagi menjadi dua yaitu *node* pertama sebagai *manager* dan *node* berikutnya sebagai *worker*. *Node* pertama yang merupakan *manager* akan menjadi pengatur untuk mengeksekusi perintah *docker service* yang bertujuan untuk mengatur *deployment* pada level *docker swarm,* membaca *logs* dan *serving container microservice*. Lalu *node* kedua yang merupakan *worker* akan bertugas sebagai *serving container microservice*.

4.3.2. Implementasi Tambah Data Penulis

Gambar 3 merupakan tampilan dari halaman admin untuk menambahkan data penulis. Pada proses penambahan data penulis, *client browser* akan melakukan *request* ke *microservice content-service* melalui *API gateway* untuk mendaftarkan atau menyimpan data penulis. Sistem akan melakukan validasi apakah semua data yang harus dimasukkan sudah valid. Lalu, ketika data penulis berhasil disimpan, proses selanjutnya adalah melakukan sinkronisasi data dengan *microservice user-service.* Dalam pengembangan, data penulis akan dibuatkan akun pengguna agar penulis dapat mengakses sistem dan melakukan salah satu fitur yaitu tanya – jawab. Sinkronisasi data dilakukan dengan melakukan *publish* data melalui *user-service-queue* yang memanfaatkan *tools* RabbitMQ. Lalu, nantinya *microservice user-service* akan menangkap, memproses dan menyimpan data yang telah dikirim dari *content-service*.



Gambar 3. Halaman Admin Tambah Data Penulis

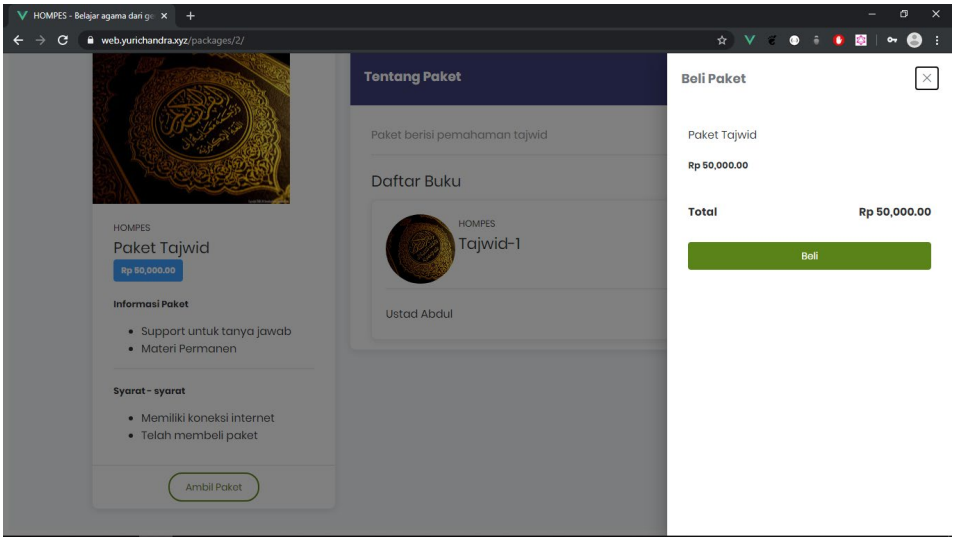
```php
public function handle(RegistrationService $registration_service)
{
    Amqp::consume('user_service_queue', function ($message, $resolver) use ($registration_service) {
        $payload = json_decode($message->body, true);
        var_dump($message->body);
        $data = [
            'email' => $payload['email'],
            'password' => app('hash')->make($payload['password']),
            'role_id' => Role::AUTHOR,
            'personal' => [
                'full_name' => $payload['personal']['full_name'],
                'phone_number' => $payload['personal']['phone_number'],
            ],
        ];

        $user = $registration_service->register($data);

        if (!is_null($user)) {
            $resolver->acknowledge($message);
            echo 'Message has been acknowledged';
        }
    });
}
```

Gambar 4. Code untuk *Listen Event* pada *User Service Queue*

Gambar 4 adalah potongan *code* pada *microservice user-service* untuk memproses data yang dikirim ke *queue* dari *microservice content-service.* Proses di atas dilakukan di *background* sistem dengan memanfaatkan *task scheduler* dari *framework* Lumen. Dengan memanfaatkan *library* AMQP, *background task* pada *user-service* menunggu pesan pada *user_service_queue*. Jika terdapat pesan pada *queue* tersebut, *microservice user-service* akan memproses data dengan melakukan *decoding* data agar dapat diproses pada tahap berikutnya. Lalu data diekstrak sesuai dengan *field* yang dibutuhkan untuk melakukan registrasi pengguna. Ketika data selesai diproses, *microservice user-service* akan mengirimkan sinyal *acknowledge* untuk memberitahu RabbitMQ bahwa pesan sudah diterima dan sudah diproses. Tujuan dari *acknowledge* adalah memberi tahu bahwa pesan sudah diterima agar tidak terdapat duplikasi pesan pada *user_service_queue*.

4.3.3. Pembelian Paket Buku

Gambar 5 adalah antarmuka halaman pembelian paket buku yang dapat digunakan oleh pengguna dengan wewenang *customer*. Ketika pengguna menekan tombol beli, maka akan dilakukan *API call* untuk mengirim data paket buku yang dibeli beserta harga pembelian ke *microservice transaction-service* melalui *API gateway.*



Gambar 5. Halaman Pembelian Paket Buku

```
// Create :nodoc
func (handler *TransactionHandler) Create(writer http.ResponseWriter, request *http.Request) {
	var transactionRequest requests.TransactionRequest

	err := render.Bind(request, &transactionRequest)
	if err != nil {
		render.Render(writer, request, responses.CreateUnprocessableEntityResponse(err.Error()))
		return
	}

	userIDCtx := request.Context().Value(shared.UserID)
	userID := userIDCtx.(int)
	now := time.Now()

	transaction, err := handler.service.Create(
		uint(userID),
		transactionRequest.PackageID,
		transactionRequest.Amount,
		now,
		now,
		now,
	)

	writer.WriteHeader(http.StatusCreated)
	err = render.Render(writer, request, responses.CreateItemTransactionResponse(transaction))
	if err != nil {
		render.Render(writer, request, responses.CreateInternalServerErrorResponse(err.Error()))
		return
	}
}
```

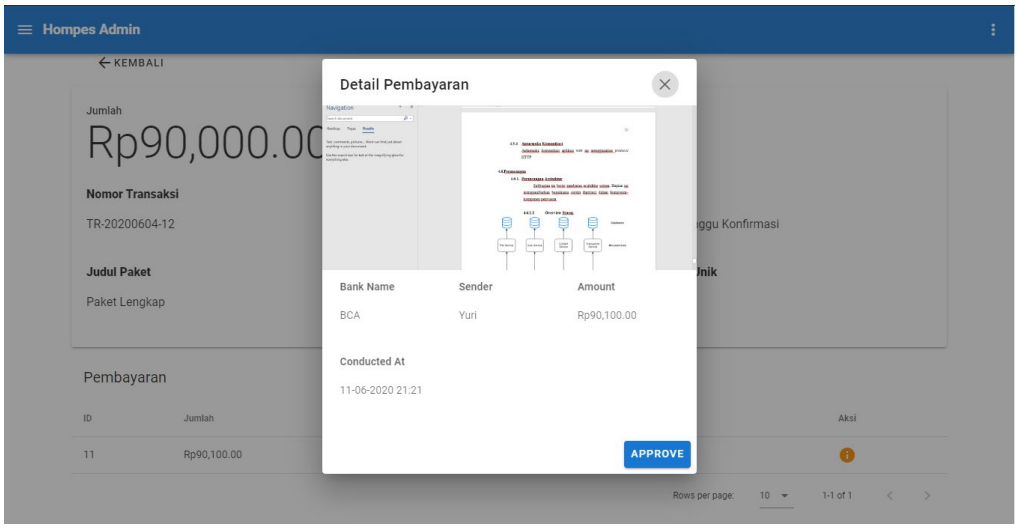
Gambar 6. Potongan Code Pembelian Paket Buku

Gambar 6 adalah potongan *code* untuk implementasi transaksi pembelian paket buku. Proses bertama adalah melakukan *binding* data dari *request object* yang terkirim melalui *API request.* Ketika terjadi data yang tidak valid, maka sistem akan mengeluarkan *warning unprocessable entity* dengan *HTTP status* 422 yang artinya data tidak dapat diproses ke tahap berikutnya. Lalu, proses berikutnya adalah ekstraksi data *user_id* yang tersimpan di *access token* saat melakukan *request* dengan metode *bearer authentication.* Lalu, data diproses kembali melalui *service layer* untuk disimpan ke basis data. Lalu, tahap berikutnya sistem mengembalikan *response* data transaksi yang baru saja disimpan dengan *http status* 201 yang artinya *created*.

4.3.4. Konfirmasi Pembayaran Paket Buku

Gambar 7 adalah implementasi dari konfirmasi pembayaran transaksi yang dilakukan pada halaman admin. Ketika pengguna dengan wewenang admin melakukan konfirmasi pembayaran, *client browser* akan melakukan *request* ke *microservice transaction-service* melalui API

gateway untuk menyimpan data transaksi pembayaran.



Gambar 7. Halaman Konfirmasi Pembayaran

```
// Approve :nodoc
func (service *ConfirmationService) Approve(id uint) (models.TransactionConfirmation, error) {
        err := service.approvalRepo.Approve(id)
        if err != nil {
                return models.TransactionConfirmation{}, err
        }

        err = service.uniqueCodeService.ReturnBack()
        if err != nil {
                return models.TransactionConfirmation{}, err
        }

        confirmation, err := service.Find(id)
        if err != nil {
                return models.TransactionConfirmation{}, err
        }

        service.proceedTransactionToQueue(confirmation.Transaction)
        return confirmation, nil
}
```

Gambar 8. Potongan Code Konfirmasi Pembayaran

Gambar 8 merupakan potongan *code* konfirmasi transaksi pembayaran. Pada potongan *code* di atas dilakukan pada *service layer* yang bertanggung jawab untuk memproses *bussiness logic*. Pada bagian *code* di atas, *approvalRepo* adalah *repository layer* yang memiliki fungsi untuk interaksi ke basis data. Pada proses *approve,* basis data akan melakukan perubahan pada dua tabel yaitu *transaction_confirmation* dan *transaction.* Data yang akan diubah adalah status transaksi, status pembayaran dan *field is_approved* pada tabel *transaction_confirmation.* Lalu proses berikutnya adalah mengembalikan *unique_code* yang digunakan sebagai identifikasi unik sebuah pembayaran ke dalam *database cache* yang menggunakan Redis. Proses berikutnya adalah mengirimkan data ke *microservice content-service* untuk menyimpan data *user_id* dan *package_id* ke dalam database.

Gambar 9 adalah potongan *code* untuk mengirim data ke *content_service_queue* dengan bantuan *library* AMQP, *queue* terlebih dahulu di-*declare* untuk menyatakan nama *queue* yang akan digunakan dalam proses ini. Lalu, dibentuklah data dalam tipe *map* yang menyimpan *key value* berupa *user_id* dan *package_ids*. Lalu data diubah kedalam bentuk JSON *(Javascript Object Notation)* untuk proses pengiriman data, berikutnya data dikirim ke *queue name* yang

sudah di-*declare* sebelumnya.

```
func (service *ConfirmationService) proceedTransactionToQueue(transaction models.Transaction) (bool, error) {
	ch, err := service.amqpConn.Channel()
	if err != nil {
		return false, err
	}

	queue, err := ch.QueueDeclare(
		"content_service_queue",
		true,
		false,
		false,
		false,
		nil,
	)
	if err != nil {
		return false, err
	}

	message := map[string]interface{}{
		"user_id":     transaction.UserID,
		"package_ids": []uint{transaction.Item.PackageID},
	}

	marshalledMsg, err := json.Marshal(message)
	if err != nil {
		return false, err
	}

	err = ch.Publish(
		"",
		queue.Name,
		false,
		false,
		amqp.Publishing{
			ContentType: "text/plain",
			Body:        marshalledMsg,
		},
	)
	return true, nil
}
```

Gambar 9. Potongan Code Mengirim Data ke content_service_queue

```
public function handle(PackageService $package_service)
{
    Amqp::consume('content_service_queue', function ($message, $resolver) use ($package_service) {
        $data = json_decode($message->body, true);

        $is_store = $package_service->storeUserPackage(
            $data['user_id'],
            $data['package_ids']
        );

        if ($is_store) {
            $resolver->acknowledge($message);
            echo 'Message has been acknowledged';
        }
    });
}
```

Gambar 10. Potongan Code *Listening Event* pada content_service_queue

Gambar 10 merupakan potongan *code* pada *microservice content-service* untuk memproses data dari *content_service_queue.* Dengan memanfaatkan *library AMQP, background task* pada *microservice content-service* akan menunggu pesan dari *queue content_service_queue.* Lalu ketika ada pesan tersedia, pesan akan diubah / *decode* untuk kebutuhan proses data ditahap berikutnya. Lalu *storeUserPackage* akan melakukan proses penyimpanan data di basis data dan akan mengembalikan nilai *Boolean*. Jika berhasil disimpan maka akan masuk proses berikutnya yaitu *acknowledge* yang akan memberikan sinyal kepada *queue* bahwa pesan sudah diterima dan diproses.

5. Kesimpulan dan Saran

Berdasarkan hasil perancangan dan implementasi yang telah dilakukan, maka dapat diambil kesimpulan sebagai berikut: (1) Pemisahan sistem dengan arsitektur *monolitik* ke arsitektur *microservice* dilakukan dengan menentukan *boundary context. Boundary context* bertujuan untuk memberikan batasan dari *domain model* tertentu. Dalam kasus Home Pesantren, terdapat empat *boundary context* yang dapat ditentukan, yaitu *user, transaction*, *content*, dan *file*. Selanjutnya, penerapan dapat dilakukan dengan melakukan pengkodean sistem dengan menerapkan basis data yang terpisah antar *microservice.* (2) Penerapan komunikasi antar *microservice* sehingga dapat menjadi sebuah kesatuan adalah dengan menerapkan dua teknik komunikasi yang berbeda, yaitu REST API untuk mendukung *synchronous communication* dan RabbitMQ untuk mendukung *asynchronous communication*.

Berikut adalah saran untuk pengembangan di tahap selanjutnya adalah (1) penggunaan layanan seperti Grafana untuk melakukan *monitoring* terhadap *microservices*. Dengan penggunaan *tools* seperti Grafana, kita dapat melakukan *monitoring* seperti berapa banyak *API request* yang dilakukan *client* ke *backend.* (2) Penggunaan *node* dapat ditambahkan pada pengembangan berikutnya, agar penempatan atau distribusi *container* pada *microservice* semakin baik. (3) Komunikasi *synchronous* antar *microservice* dapat ditingkatkan dengan menerapkan gRPC *(Remote Procedure Call),* untuk mendukung performa yang lebih baik dibandingkan REST API.

Referensi

[1] S. Newman, *Building Microservices*, First. California: O`Reilly, 2015.

[2] M. Villamizar *et al.*, “Evaluating the Monolitikic and the Microservice architecture pattern to deploy web applications in the cloud,” *2015 10th Colomb. Comput. Conf. 10CCC 2015*, no. Jul. 2018, pp. 583–590, 2015.

[3] R. V. O’Connor, P. Elger, and P. M. Clarke, “Continuous software engineering—a Microservices architecture perspective,” *J. Softw. Evol. Process*, vol. 29, no. 11, Nov. 2017.

[4] Microsoft Corporation, “Building Microservices on Azure,” pp. 1–5, 2019.

[5] P. Software and S. Units, “What is a container ? a standardized unit of software package software into standardized units for everywhere : Linux ,” pp. 1–5, 2020.

[6] C. Synchronous and D. Delivery, “Designing interservice communication for Microservices,” pp. 1–9, 2020.

[7] H. Suryotrisongko, “Arsitektur Microservice untuk resiliensi sistem informasi,” *Sisfo*, vol. 06, no. 02, pp. 231–246, 2017.

[8] A. D. M. Del Esposte, F. Kon, F. M. Costa, and N. Lago, “InterSCity: a scalable Microservice-based open source platform for smart cities,” *SMARTGREENS 2017 - Proc. 6th Int. Conf. Smart Cities Green ICT Syst.*, no. Smartgreens, pp. 35–46, Apr. 2017.

[9] S. Habibullah, X. Liu, Z. Tan, Y. Zhang, and Q. Liu, “Reviving legacy enterprise systems with Micro service-based architecture with in cloud environments,” pp. 173–186, 2019.

[10] Xiao Ma, “Microservice architecture at Medium,” pp. 1–14, 2019.