ANALISIS DAN RANCANG SISTEM OTOMATIS *DATABASE SCHEMA MIGRATION* BERBASIS GITOPS UNTUK PROSES CI/CD PADA PERUSAHAAN *FINTECH*

Cristian Stevanus Oloan[1], Yusup Jauhari Shandi[2], Jenisa Felisa[3], Sudimanto[4]

STMIK LIKMI Bandung
Jl. Ir. H. Juanda No 96 Bandung 40132

[1]cristianstevanus09@gmail.com
[2]ujshandi@gmail.com

ABSTRAK

PT. XYZ adalah salah satu bursa aset Kripto di Indonesia yang menyediakan *platform* untuk dapat melakukan investasi terhadap aset Kripto yang sudah terpercaya dan sudah mendapatkan legalitas dari BAPPEBTI. Dalam proses bisnis yang dijalankan oleh PT. XYZ, *engineer* selaku *developer* aplikasi melakukan praktik *database schema migration* secara manual yang dilakukan cukup sering, yang setidaknya dalam kurun waktu satu minggu terdapat satu aktivitas untuk melakukan praktik tersebut. Hal tersebut perlu mendapatkan pertimbangan, dikarenakan PT. XYZ selaku perusahaan *FinTech* yang menyediakan jasa *financial* dalam bentuk teknologi. Terutama, praktik dalam mengelola *database* yang dilakukan secara manual harus dihindari. Dampak dari praktik tersebut tidak adanya *track record* atas *database schema migration* yang telah dilakukan, dan maka daripada itu tidak adanya juga data untuk dijadikan laporan, semisalnya ada audit.
Berdasarkan hal tersebut, direncanakan sistem aplikasi yang menerapkan metodologi GitOps, yang menjadikan Git sebagai sumber satu-satunya kebenaran terkait kondisi infrastruktur termasuk *database*, dan melakukan *trigger* secara otomatis menjalankan praktik *database schema migration* disaat adanya perubahan terkait *database schema* yang telah dilampirkan pada GitHub *repository*. Sistem aplikasi yang dirancang berdasarkan pemodelan dari *Unfiled Modeling Language* (UML) yaitu *use case diagram, activity diagram,* dan *sequence diagram* dengan pendekatan berbasis objek. Python dan Vue JS merupakan bahasa pemrograman yang digunakan dalam perancangan sistem ini, dan menghasilkan sistem aplikasi berbasis *website*. Dengan adanya sistem ini, dapat menjawab persoalan mengenai praktik yang perlu dipertimbangkan sebelumnya dengan lebih efisien dan tercatat segala aktivitas *database schema migration*.

Kata kunci : Sistem Informasi, Database Schema Migration, Prototype, CI/CD, Python, Automation, FinTech, Vue JS..

1. PENDAHULUAN

Pada era teknologi saat ini, tidak sedikit perusahaan yang memanfaatkan teknologi dalam proses bisnis. Salah satu jenis perusahaan yang menggunakan teknologi dalam proses bisnisnya yaitu perusahaan *Startup*. Fintech merupakan bagian dari Startup dikarenakan menggunakan teknologi dalam proses bisnis mereka, teknologi yang dihasilkan oleh perusahaan tersebut berupa sebuah aplikasi, baik dalam bentuk web ataupun mobile. Dalam

membuat suatu aplikasi, *Startup* pada umumnya menggunakan tahapan SDLC (*Software Development Lifecycle*) agar tercapai produk yang direncanakan.

PT. XYZ merupakan suatu perusahaan Fintech yang berfokus dalam aset kriptografi. Jasa yang mereka sediakan yaitu menyediakan *platform* aplikasi dalam bentuk web dan *mobile* untuk dapat melakukan investasi, membeli dan menjual aset kriptografi. Dalam proses *development* aplikasi, PT. XYZ menggunakan tipe SDLC yang bernama *Agile*, yang mengutamakan kecepatan dan fleksibilitas. Disisi lain, PT. XYZ menerapkan metodologi GitOps dalam menerapkan infrastruktur berbasis kode (*Infrastructure as Code*) dan menjadikan Git sebagai satu-satunya sumber dalam membuat, mengontrol atau bahkan menghapus suatu infrastructure environment. PT. XYZ mempunyai divisi technology dalam development application seperti contohnya yaitu *Backend Engineer* dan *Site Reliability Engineer.*

Dalam proses *development*, *Backend Engineer* mempunyai kebutuhan untuk merancang suatu database schema yang akan digunakan oleh aplikasi dalam proses *development*. Dalam proses eksekusi database schema migration, *Backend Engineer* tidak bisa melakukannya seorang diri saja, dikarenakan peran yang mempunyai akses terhadap *database* yaitu *Site Reliability Engineer*. Maka daripada itu, *Backend Engineer* perlu bekerja sama dengan *Site Reliability Engineer* dengan menggunakan *platform Slack* untuk berkomunikasi dan menyampaikan kebutuhan untuk *database schema migration* dengan cara Backend Engineer memberikan file script yang berkait dengan SQL DDL/DML untuk dieksekusi secara manual menggunakan platform PHPMyAdmin oleh *Site Reliability Engineer*. Bagian dari proses tersebut, akan dilakukan secara berulang-ulang saat adanya kebutuhan mengubah atau memperbaharui database schema. *Site Reliability Engineer* mengelola sejumlah database dengan total 7 database di setiap environment (terdapat 2 *environment* yaitu beta dan production), maka daripada itu jumlah keseluruhannya yaitu 14 database. Frekuensi adanya *database schema migration* dalam 4 bulan (April 2024 - Agustus 2024) yaitu sebanyak 35 kali database schema migration dengan rata-rata per bulannya yaitu 7x per bulan.

Berdasarkan data jumlah frekuensi database schema migration dan jumlah database yang dikelola, tentunya dapat dikatakan tidak efisien dikarenakan proses tersebut dilakukan secara manual, tidak tercatat, dan bertolak belakang dengan prinsip yang telah ditetapkan oleh PT. XYZ, yang mana menerapkan metodologi GitOps dan SDLC Agile. Proses yang berlangsung berjalan dengan tidak efisien, dapat mempengaruhi kinerja dari *developer application (Backend Engineer). Database schema migration* yang dilakukan secara manual, menjadikan suatu batasan bagi *Site Reliability Engineer* untuk melacak *database schema migration* yang sudah terjadi di masa lampau.

Saat ini sudah ada beberapa alat yang tersedia untuk dapat melakukan *database schema migration,* namun ada beberapa hal keterbatasan yang dari alat-alat tersebut untuk memenuhi kebutuhan PT. XYZ, seperti halnya integrasi GitOps, notifikasi melewati Slack, dan juga pencatatan historis dari *database schema migration*. Berdasarkan latar belakang yang sudah dijelaskan sebelumnya, dilakukannya penelitian yang berjudul "**ANALISIS DAN RANCANG SISTEM OTOMATIS *DATABASE SCHEMA MIGRATION* BERBASIS GITOPS PADA PROSES CI/CD PADA PERUSAHAAN *FINTECH***" untuk mencapai dan menyesuaikan metodologi yang diterapkan pada perusahaan

2. STUDI LITERATUR

2.1. *Database Schema Migration*

Database schema migration merupakan suatu metode dalam melakukan pemindahan *database* dalam bentuk skema yang sama dari suatu perangkat ke perangkat lainnya. Disisi lain juga, *database schema migration* cukup membantu dalam membuat atau melakukan

perubahan terhadap suatu tabel di *database* [5]. Peran yang melakukan perubahan skema *database* pada umumnya dilakukan oleh *Database Administrator* atau disingkat DBA. DBA merupakan seseorang yang bertanggung jawab untuk membuat desain, kontrol dan juga mengelola administrasi *database*. Namun pada penelitian ini, peran DBA dirangkap oleh *Site Reliability Engineer* atau SRE, yang berarti pekerjaan DBA dilakukan oleh SRE.

Penerapan *database schema migration* sangat bermanfaat, selain bertujuan untuk mengelola perubahan *schema database* atau struktur data dengan cara terprogram, *database schema migration* memiliki manfaat antara lain :

a. Mencegah hilangnya data dalam proses perubahan struktur database.
b. Membantu *application developer* untuk merencanakan, memvalidasi dan menerapkan perubahan *database schema* secara aman.
c. Memungkinkan penyesuaian manual dalam proses, serta audit dan pengujian setiap perubahan.

Dalam penelitian dengan judul "*Database Schema Migration in Highly Available Service*" [4], dilampirkan bahwa proses *database schema migration* dilakukan di beberapa perusahaan yang menerapkan proses *development* aplikasi yang cepat dan *agile* atau metodologi *DevOps*, untuk mengurangi durasi dan biaya dalam *delivery* aplikasi hingga ke tahap *production*. Terlebih lagi perubahan *database schema* yang dilakukan secara manual dapat menunda proses pada *development cycle* yang dilakukan oleh *software developer* [4]. Dalam melakukan *database schema migration*, ada beberapa hal penting yang perlu diperhatikan antara lain sebagai berikut [6] :

a. Memastikan kelengkapan skema
b. Tabel dengan fungsi sistem sebagai klausa dengan nilai *DEFAULT* pada kolom.
c. Membuat akun pengguna beserta *role* pada *database*. Dengan menetapkan suatu akun dengan *role* tertentu dapat membantu dalam mengelola *object-level*.
d. Mempartisi tabel *database*. Dengan menerapkan partisi tabel, akan memudahkan dalam melakukan manajemen dan mengoptimalkan kinerja pada saat melakukan *query* pada *database*.

Menurut [3] dalam bukunya yang berjudul "*High Performance MySQL, 4th Edition"*, untuk mengelola perubahan skema ada beberapa poin yang perlu dipertimbangkan:

a. Menetapkan rekan untuk meraih kesuksesan. Dalam hal ini yang mempunyai makna bahwa dalam melakukan perubahan skema tidak membuat *developer* bergantung pada seseorang atau tim yang mengelola *database*, akan tetapi mereka sendiri dapat melakukannya sendiri.
b. Mengintegrasikan manajemen skema dengan *continuous integration*. Dengan menetapkan premis bahwa peran yang mengelola skema yaitu *developer*, maka perlu ditentukan alur kerja yang berhubungan dengan mereka agar meningkatkan kinerja *developer*.
c. *Source control* (contohnya: *Git*) sebagai media yang menyimpan informasi terkait skema. Mengelola perubahan skema dalam jangka besar perlunya media yang mendukung dan melacak perubahan skema yang telah dilakukan.
d. Menghindari perubahan skema yang dilakukan pada lingkungan *production* yang menyebabkan *downtime* dengan cara menggunakan *native* DDL *statements* dan menggunakan *external tool* untuk menjalankan perubahan skema.

e. Mengimplementasikan perubahan skema pada alur CI/CD dapat menghilangkan *bottleneck* pada produktivitas *engineer*.

Saat ini, sudah ada beberapa alat *database schema migration* yang populer di kalangan perusahaan IT untuk dapat memudahkan dan mengelola *database schema migration*, seperti contohnya yaitu Liquibase dan Flyway. Kedua aplikasi tersebut dibentuk dari perusahaan yang berbeda dan juga mempunyai keunggulannya masing-masing. Perbandingan dari kedua alat tersebut disajikan dalam Tabel 1.

Tabel 1. Tools Database Schema Migration

	FLYWAY	**LIQUIBASE**
PERUSAHAAN	RedGate	Liquibase Inc.
DIPUBLIKASIKAN	2010	2006
ENGINE	Java + JVM	Java + JVM
EDITION	Community, Teams, and Enterprise	Open Source and Pro
***DATABASE* GITOPS**	Tidak tersedia (Kalau ingin mengimplementasikan perlu ada cara tambahan diluar dari fitur yang disediakan)	Tidak tersedia (Kalau ingin mengimplementasikan perlu ada cara tambahan diluar dari fitur yang disediakan)
PENCATATAN HISTORIS	Tersedia (Hanya terdapat pada *Enterprise Edition* atau diatasnya)	Tidak tersedia
DEVELOPER INTERFACE	CLI	CLI
SUPPORT DATABASE	Aurora MySQL, Aurora PostgreSQL, Azure Synapse, Clickhouse, CockroachDB, Databricks, DB2, Derby, Firebird, Google BigQuery, Google Cloud Spanner, H2, HSQLDB, Informix, MariaDB, MongoDB, MySQL, Oracle, Percona XtraDB Cluster, PostgreSQL, Redshift, SAP HANA	Amazon DocumentDB, Amazon DynamoDB, Amazon Redshift, AWS Aurora - MySQL, AWS Aurora for PostgreSQL, AWS RDS - MariaDB, AWS RDS - Microsoft SQL Server, AWS RDS - MySQL Server, AWS RDS - Oracle Database, AWS RDS - PostgreSQL, Azure Database for MySQL,
SUPPORT DATABASE	(Including SAP HANA Cloud), SingleStoreDB, Snowflake, SQLite, SQL Server, Sybase, ASE, TiDB, TimescaleDB, YugabyteDB	Azure Database - PostgreSQL - Flexible Server, Azure Database - PostgreSQL Single Server, EnterpriseDB, Google BigQuery Pro, Google BigQuery Open Source, Google Cloud SQL - MSSQL, Google Cloud SQL - PostgreSQL, IBM DB2 LUW, MariaDB Server, MariaDB SkySQL, Microsoft Azure SQL Database, Microsoft Azure SQL Managed Instance, Microsoft SQL Server, MongoDB Pro Extension Community and Enterprise Server, MongoDB Atlas Pro Extension, MySQL Server, Oracle Database, Oracle Autonomous Database, PostgreSQL, Snowflake
***SUPPORT* MYSQL V5.7**	Ya (Limitasi pengujian pada *Community Edition*)	Ya (Akan tetapi sudah tidak berlaku lagi pada MySQL Server dan AWS RDS)
KEMUDAHAN PENGGUNAAN	Ya (Mudah digunakan bagi pemula)	Tidak (Banyaknya fitur yang disediakan sehingga tidak cocok untuk pemula)

INTEGRASI NOTIFIKASI	Tidak	Tidak
SQL *CHECK VALIDATION*	Tidak	Tidak
TERDAPAT NAMA PENGEKSEKUSI *DATABASE SCHEMA MIGRATION*	Tidak	Ya

2.2. *Continuous Integration and Continuous Delivery/Deployment* (CI/CD)

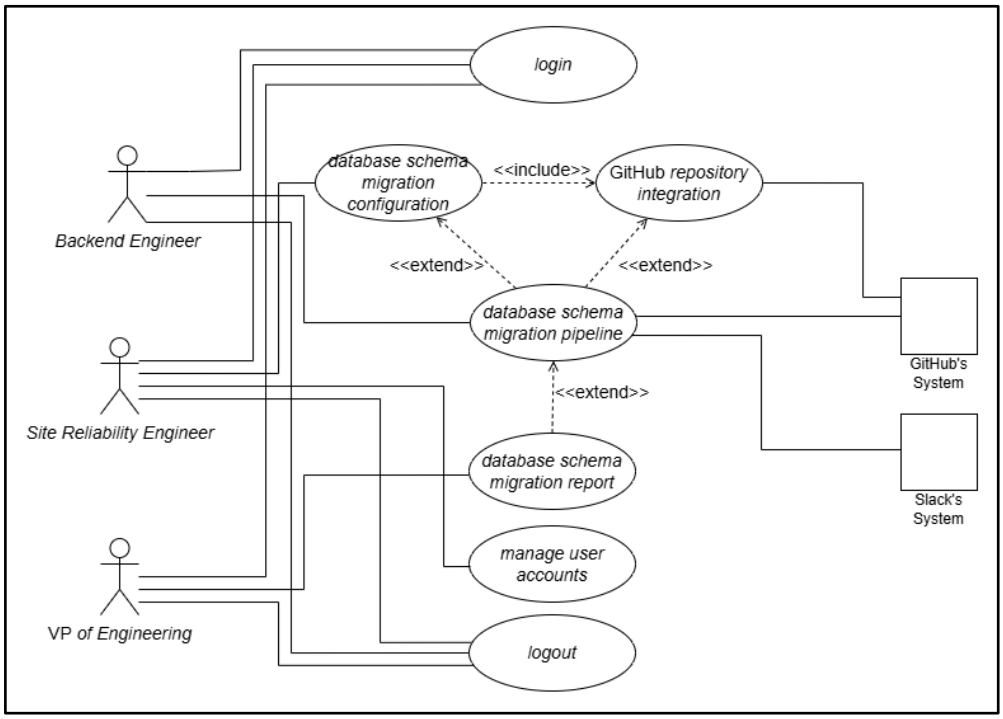
Beck memperkenalkan metodologi bernama *eXtreme Programming* dengan menghasilkan praktik dari *software engineering* yang bernama CI/CD. CI/CD merupakan bagian dari *software development* untuk menunjang otomatis proses dari rangkaian alur seperti kompilasi *software*, pembuatan *software*, pengujian *software*, validasi *software*, rilis *software* dan hal-hal lainnya [8]. CI/CD terbagi menjadi beberapa proses, antara lainnya adalah:

a. *Continuous Integration* (CI)
   CI diperkenalkan pada tahun 2006, akan tetapi tidak terlalu populer oleh kalangan praktisi, akan tetapi setelah hadirnya alat CI seperti Travis CI, implementasi CI meningkat dibanding sebelumnya [9]. Tahapan proses ini mengacu pada penggabungan dari *branch feature* ke *branch main* pada suatu *repository*, kemudian dilakukannya berbagai proses uji coba pada *code* [7]. Manfaat dari *continuous integration* adalah membantu proses dalam menemukan dan memperbaiki bug, meningkatkan kualitas *software*, menemukan dan memperbaiki commit yang rusak, meningkatkan *development workflow* [2].
b. *Continuous Delivery* (CD)
   Fase berikutnya setelah *continuous integration*, merupakan implementasi *code software* pada lingkungan infrastruktur *staging* dan sekaligus melakukan pengujian fungsi *software* tersebut [1]. Namun hal lain mengatakan bahwa fase ini berarti perubahan yang terjadi pada *software* oleh *developer* itu otomatis diuji oleh *bug* dan *upload* ke suatu *repository* seperti GitHub atau *container registry* [10].
c. *Continuous Deployment* (CD)
   Fase terakhir yang menerapkan alur proses yang dilakukan secara otomatis dengan menjalankan tahapan rilis *software* bahkan implementasi *software* pada lingkungan infrastruktur *production* [1]. Proses ini dapat membantu dalam menyelesaikan permasalahan pada saat tim operasional melakukan implementasi *software* atau *delivery software* pada lingkungan infrastruktur *production* dan proses yang dilakukan hanya membutuhkan waktu beberapa menit saja. Akan tetapi walau proses ini dilakukan secara otomatis, proses ini sebelumnya harus melewati fase yang telah teruji dan tervalidasi sebelum *software* dirilis [10]

3. ANALISIS DAN PERANCANGAN APLIKASI

3.1. *Use Case* Diagram

Pembahasan pertama terkait hal yang akan dirincikan prosedur pada PT. XYZ yaitu diagram konteks. Diagram konteks merupakan penggambaran dari aliran input dan output dari suatu program aplikasi. Diagram konteks memiliki relasi dengan entitas external, yang di mana entitas tersebut berperan untuk berinteraksi langsung dengan suatu sistem. Berikut diagram system use case dari tahapan prosedur kerja database schema migration yang dapat dilihat pada Gambar 1.

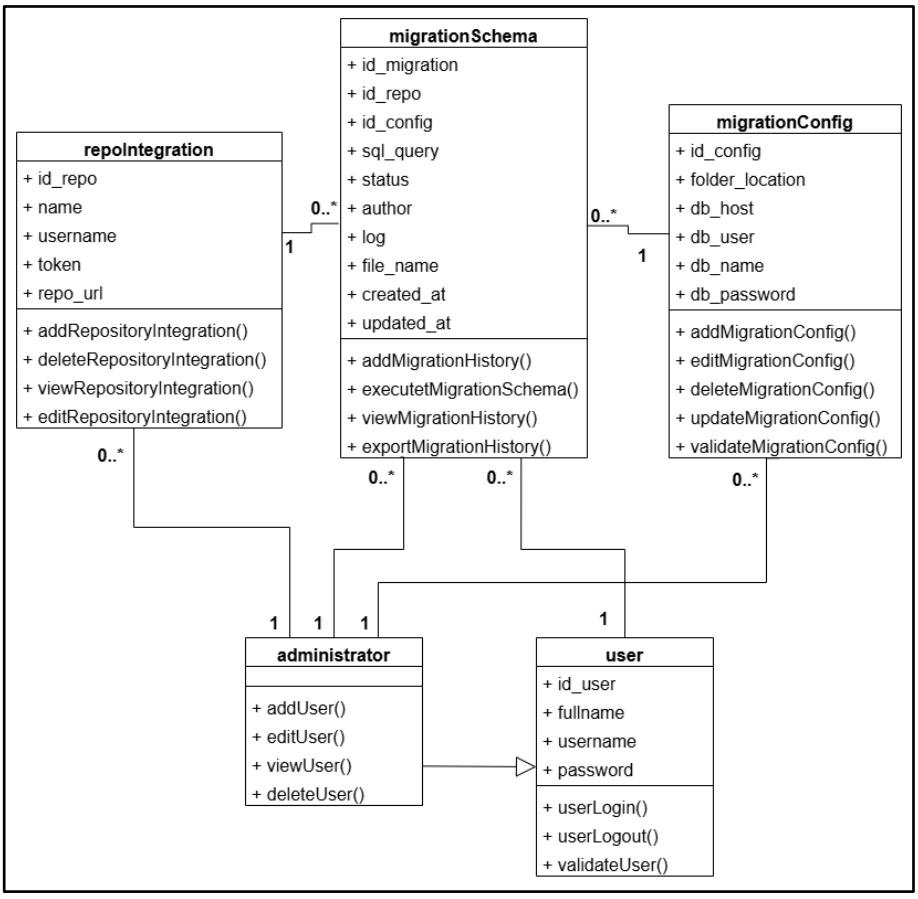


Gambar 1. System Use Case

3.2. *Class Diagram*

Class diagram menunjukkan rancangan keterhubungan antar *class*, dan setiap *class* tersebut juga dijelaskan kemampuan atau metode yang bisa dilakukan. Rancangan *class diagram* tersebut dapat dilihat pada Gambar 2. Berdasarkan pada Gambar 2, sistem memiliki 5 *class*, yaitu:

a. *Class* "user" digunakan untuk menyimpan data user yang tidak memiliki *role* sebagai administrator.
b. *Class* "administrator" merupakan turunan dari *parent class* "user". Yang membedakan *class* "administrator" dan "user" yaitu *class* "administrator" dapat melakukan fungsi seperti menambahkan, mengubah, dan menghapus *user*.
c. *Class* "repoIntegration" berfungsi untuk mengelola data *repository* dan membangun integrasi dengan GitHub *repository*.
d. *Class* "migrationConfig" berfungsi untuk mengelola *migration configuration* sebagai landasan untuk menjalankan *database schema migration*, dan dalam *class* ini disertai relasi data terhadap *class* "repoIntegration" yang membutuhkan data nama *repository*.
e. *Class* "migrationSchema" berfungsi untuk eksekusi *database schema migration* yang berlandaskan dari *state* yang telah ditambahkan pada *class* "repoIntegration" dan *class* "migrationConfig", dan menampung data historis *database schema migration*.



Gambar 2. Class Diagram

4. IMPLEMENTASI DAN PENGUJIAN APLIKASI

4.1. Pemilihan perangkat lunak

Menjalankan sistem aplikasi, perlu *software* (perangkat lunak) yang mendukung berjalannya sistem aplikasi. *Software* yang digunakan pada sistem aplikasi adalah :

a. Sistem Operasi Ubuntu Server versi 20.04
b. MySQL versi 8.0
c. Redis versi 1.89.0
d. GitHub
e. Slack versi 4.38.115
f. Docker versi 20.10.21
g. Docker Compose versi 2.13.0
h. Caddy versi 2.6.4
i. NPM versi 9.5.1

4.2. Arsitektur *Back-End* dan *Front-End*

a. *Back-End*

Back-end service pada penelitian ini menggunakan bahasa pemrograman Python 3.10.12 + Django 3.14.0 + Django REST Framework 5.0.2.. Fungsinya adalah:

1) Menangani permintaan REST API dari frontend
2) Menerima webhook dari GitHub
3) Memproses dan mengeksekusi file SQL migration

4) Menyimpan histori ke MySQL database
5) Mengirim notifikasi hasil ke Slack melalui webhook

b. *Front-End*

Front-end service pada penelitian ini menggunakan bahasa pemrograman Node JS + Vue 3.4.21. Fungsinya adalah:

1) UI dashboard pengguna
2) Form input konfigurasi GitHub & DB
3) Menampilkan data histori migrasi
4) Menggunakan Axios untuk komunikasi ke API backend

c. Interaksi *Front-End* dan *Back-End*:

1) Vue mengakses API Django melalui endpoint REST seperti:
   a) POST /api/repo (menambahkan konfigurasi repository)
   b) GET /api/history (menampilkan histori migrasi)
2) Response ditampilkan dalam UI Vue.

4.3.Persiapan Penerapan Sistem

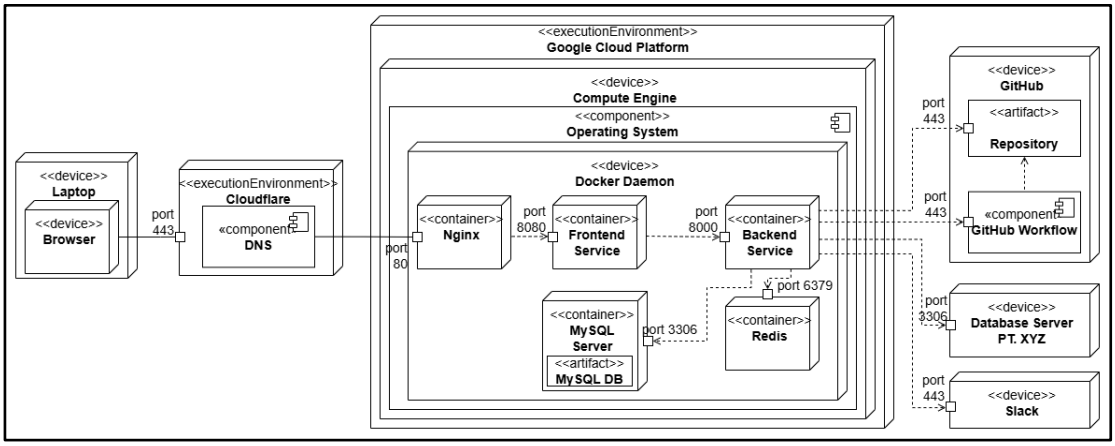
Sistem aplikasi yang telah dibangun, tidak semerta-merta dapat digunakan oleh pengguna. Namun ada aspek yang perlu dilakukan sebelum sistem aplikasi dapat digunakan semestinya. Aspek tersebut biasanya diperlukan dikarenakan sistem aplikasi mempunyai ketergantungan dengan komponen atau sistem lainnya. Hal ini cukup relevan dengan diagram komponen yang telah dijelaskan pada poin sebelumnya. Persiapan yang perlu dilakukan, antara lain yaitu:

a. *Virtual machine* dan OS (*Operating System*).
   Persiapan pertama yaitu menyiapkan *virtual machine* yang menjadi wadah sistem aplikasi dijalankan. *Virtual machine* yang digunakan yaitu menggunakan produk *compute engine* dari *Google Cloud Platform* (GCP). Tipe e2-medium merupakan spesifikasi yang digunakan oleh *compute engine* dengan 2vCPU core dan 4 GB RAM. Kemudian OS (*Operating System*) yang berjalan pada *virtual machine* tersebut yaitu Ubuntu Server 20.04.
b. Docker dan docker-compose.
   Setelah *virtual machine* selesai disiapkan, selanjutnya *install* docker dan docker-compose untuk sebagai *engine* yang menjalankan keseluruhan sistem aplikasi. Namun sebelum itu, perlu membuat *file* dari dalam bentuk format docker-compose untuk dapat menjalankan *multi-container* pada waktu yang bersamaan. Hal yang didefinisikan pada docker-compose adalah *service* dari sistem aplikasi, antara lain yaitu Redis, MySQL Server, Nginx, *frontend service*, dan *backend service*.
c. DNS
   Konfigurasi domain untuk sistem aplikasi, dengan menggunakan layanan DNS Cloudflare.
d. Integrasi notifikasi
   Persiapan selanjutnya, integrasi sistem aplikasi dengan platform kolaborasi Slack. Dengan membuat aplikasi pada Slack, aplikasi pada Slack akan memberikan notifikasi pesan dari hasil operasi *database schema migration* berdasarkan hasil aktifitas *database schema migration*.

e. GitHub *Workflow*
Membuat *script* pada GitHub Workflow pada suatu *repository* untuk membuat *pipeline* operasi validasi *script* SQL *file* dan *trigger webhook* operasi *database schema migration*.

4.4. *Deployment diagram*

Deployment diagram merupakan representasi dari aspek fisik yang berkaitan dengan sistem aplikasi. *Deployment diagram* pada sistem aplikasi di PT. XYZ melibatkan seperti Cloudflare, GitHub, *Database Server* pada PT. XYZ, dan Slack. Sistem aplikasi yang dijalankan pada *Google Compute Engine*. Untuk lebih lengkapnya dapat Gambar 3.



Gambar 3. Deployment Diagram

4.5. Pengujian sistem

Aplikasi yang telah dirancang dan sudah dibangun perlu melewati tahapan pengujian, agar aplikasi yang telah dibangun sesuai dengan perancangan di awal. Pengujian ini dilakukan secara manual dengan memastikan bahwa aplikasi berfungsi dengan spesifikasi yang telah ditentukan. Hasil pengujian tersebut dapat dilihat pada Tabel 2.

Tabel 2. Pengujian Sistem

ITEM UJI	KASUS UJI	HASIL YANG DIHARAPKAN	HASIL UJI
Login	Input *username* dan *password* yang valid	Berhasil *login* dan menampilkan halaman *home*	Sesuai
	Salah satu dari *username* dan *password* tidak valid	Menampilkan pesan “Gagal *login*, silahkan coba lagi”	Sesuai
	Salah satu dari *username* atau *password* tidak diisi	Menampilkan pesan “Gagal *login*, silahkan coba lagi”	Sesuai
Repository Integration	Input *name, username, token, branch dan repository* URL yang valid	Berhasil menambahkan *repository integration*	Sesuai
	Salah satu dari *name, username, token, branch dan repository* URL tidak diisi	Menampilkan pesan “Gagal menambahkan *repository integration*, silahkan coba lagi”	Sesuai

	Token tidak sesuai	Menampilkan pesan "Token GitHub tidak valid"	Sesuai
	Nama *branch* tidak sesuai	Menampilkan pesan "*Branch* tidak tersedia"	Sesuai
	Repository URL tidak sesuai	Menampilkan pesan "*Repository* tidak tersedia"	Sesuai
	Username tidak sesuai	Menampilkan pesan "GitHub *username* tidak tersedia"	Sesuai
	Nama *repository integration* duplikat	Menampilkan pesan "Nama *repository integration* sudah digunakan"	Sesuai
Migration Config	Input *folder location, database host, database username, database name, database password* dan repo yang sesuai	Menampilkan pesan "Berhasil menambahkan *migration config*"	Sesuai
	Salah satu dari *folder location, database host, database username, database name, database password* dan repo tidak diisi	Menampilkan pesan "Gagal menambahkan *migration config*, silahkan coba lagi"	Sesuai
	Folder location tidak sesuai	Menampilkan pesan "*Folder location* tidak tersedia pada *repository*"	Sesuai
	Database host tidak dapat dijangkau oleh aplikasi atau tidak valid	Menampilkan pesan "Gagal koneksi ke *database*"	Sesuai
	Database username tidak sesuai atau tidak ada	Menampilkan pesan "Gagal koneksi ke *database*"	Sesuai
	Database name tidak sesuai atau tidak tersedia	Menampilkan pesan "*Database name* tidak tersedia"	Sesuai
	Database password tidak sesuai	Menampilkan pesan "Gagal koneksi ke *database*"	Sesuai
Migration	*File* SQL yang ditambahkan ke *repository* tidak valid	Mengirimkan pesan "SQL script tidak valid" ke *channel* Slack	Sesuai
	Gagal melakukan *database schema migration*	Mengirimkan pesan berupa detail terkait *database schema migration* ke *channel* Slack	Sesuai
	Berhasil melakukan *database schema migration*	Mengirimkan pesan berupa detail terkait *database schema migration* ke *channel* Slack	Sesuai

	Ekspor historis *database schema migration* berdasarkan interval yang dibutuhkan	Mengunduh *file report* historis berupa PDF	
Users	Input *fullname, username, password* dan *role* yang valid	Berhasil menambahkan *user*	Sesuai
	Salah satu *fullname, username, password* tidak diisi	Menampilkan pesan "Gagal menambahkan *user*, silahkan coba lagi"	Sesuai
	Tidak memilih satu satu *role* antara *admin* dan *viewer*.	Menampilkan pesan "Belum memilih *role*"	Sesuai

Dari hasil uji yang telah disajikan pada Tabel 2, dilanjutkan membandingkan sistem aplikasi yang telah dibuat dengan aplikasi *database schema migration* yang tersedia seperti Flyway dan Liquibase yang dijabarkan dalam Tabel 3.

Tabel 3. Perbandingan Tools Database Schema Migration

KRITERIA UJI	**FLYWAY**	**LIQUIBASE**	**SISTEM APLIKASI**
Engine	Java + JVM	Java + JVM	Python
Edition	Community, Teams, and Enterprise	Open Source and Pro	-
KRITERIA UJI	**FLYWAY**	**LIQUIBASE**	**SISTEM APLIKASI**
***Database* GitOps**	Tidak tersedia (Kalau ingin mengimplementasikan perlu ada cara tambahan diluar dari fitur yang disediakan)	Tidak tersedia (Kalau ingin mengimplementasikan perlu ada cara tambahan diluar dari fitur yang disediakan)	Tersedia
Pencatatan Historis	Tersedia (Hanya terdapat pada *Enterprise Edition* atau diatasnya)	Tidak tersedia	Tersedia
Developer Interface	CLI	CLI	GUI
***Support Database* MySQL**	Ya	Ya	Ya

***Support* MySQL v5.7**	Ya (Tetapi ada limitasi pengujian pada *Community Edition*)	Ya (Akan tetapi sudah tidak berlaku lagi pada MySQL Server dan AWS RDS)	Ya
Kemudahan Penggunaan	Ya (Mudah digunakan bagi pemula)	Tidak (Banyaknya fitur yang disediakan sehingga tidak cocok untuk pemula)	Ya
Integrasi notifikasi Slack	Tidak	Tidak	Ya
SQL *Check Validation*	Tidak	Tidak	Ya
Terdapat nama pengeksekusi *database schema migration*	Tidak	Ya	Ya

Sistem aplikasi *database schema migration* yang dibentuk jika dibandingkan dengan aplikasi lain secara keandalan dan fitur yang disediakan akan kalah. Faktor yang dipengaruhi dikarenakan kedua aplikasi dari Flyway dan Liquibase dibentuk sudah sejak tahun awal tahun 2000. Dapat dikatakan kedua aplikasi tersebut sudah mempunyai usia yang lebih matang jika dibandingkan dengan sistem aplikasi dari PT. XYZ. Selain daripada itu, kedua aplikasi tersebut tidak hanya kompatibel dengan *database server* MySQL, namun sudah banyak *database server* yang kompatibel dengan kedua aplikasi tersebut.

Namun kebutuhan utama dari PT. XYZ untuk menjalankan *database schema migration* secara otomatis, integrasi dengan platform untuk mengirimkan notifikasi, menerapkan GitOps, dan mempunyai data historis dari aktifitas *database schema migration*. Hal itu semuanya dapat dipenuhi oleh sistem aplikasi yang telah dibuat pada PT. XYZ. Adapun dari kedua aplikasi yang sudah ada, hanya dapat memenuhi beberapa kebutuhan dari PT. XYZ dan itu pun perlu dengan tambahan biaya.

Membandingkan ketiga aplikasi tersebut dalam konteks fitur yang dibutuhkan, sistem aplikasi yang telah dibuat pada PT. XYZ dapat disesuaikan dengan kebutuhan yang diperlukan suatu saat nanti atau lebih kompleks dari kebutuhan saat ini, sedangkan dengan kedua sistem aplikasi yang sudah ada tidak dapat dikembangkan secara *internal* dan perlu menyesuaikan dengan fitur yang kedua aplikasi tersebut sediakan.

5. KESIMPULAN

Analisa dan perancangan sistem aplikasi *database schema migration* pada PT. XYZ berlandaskan adanya proses kerja dalam melakukan *database schema migration* yang dilakukan secara manual sehingga tidak adanya pencatatan historis terkait *database schema migration*, dan SQL DDL/DML untuk menjalankan *database schema migration* tidak

dicantumkan pada suatu *GitHub repository*. Oleh karena itu direncanakan dan dibentuknya sistem aplikasi yang dijalankan secara otomatis untuk menjawab persoalan tersebut.

Kesimpulan atas hasil analisis ini antara lain yaitu perancangan sistem aplikasi *database schema migration* yang telah dibangun dapat mengotomatiskan proses *database schema migration* melalui integrasi dengan alur CI/CD, serta menyediakan fitur pelaporan historis yang detail. Selain daripada itu, sistem aplikasi ini menerapkan GitOps dalam konteks *database schema migration* dapat meningkatkan efisiensi, reliabilitas, dan transparansi dalam pengelolaan database. Kemudian sistem aplikasi ini juga membuat laporan historis dari proses database schema migration yang telah terjadi dan dilaporkan secara berkala kepada *Vice President of Technology*.

DAFTAR PUSTAKA

[1] Abiola, O. B., & Olufemi, O. G. (2023). An Enhanced CICD Pipeline: A DevSecOps Approach. *International Journal of Computer Applications*, 6.
[2] Arachchi, I. P. (2018). Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. 6.
[3] Botros, S., & Tinley, J. (2021). *High Performance MySQL: Proven Strategies for Operating at Scale 4th Edition.* New York: O'Reilly Media .
[4] Fernandez, J. B. (2021). "*Database Schema Migration in Highly Available Service*". KTH Vetenskap Och Konst. Stockholm, Sweden.
[5] Giri, G., I.G.D., D., & Wibawa, I. (2022). Rancang Skema Database dan Implementasi Database Migration Pada Aplikasi Peminjaman Ruangan. 10.
[6] Laszewski, T., & Nauduri, P. (2012). *Migration to The Cloud Oracle Client/Server Modernization.* Waltham: Elsevier Inc.
[7] Loubser, N. (2021). *Software Engineering for Absolute Beginners.* London: Apress. Arachchi, I. P. (2018). Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. 6.
[8] Mazrae, P. R., Mens, T., Golzadeh, M., & Decan, A. (2023). On the usage, co-usage and migration of CI/CD tools. *Empirical Software Engineering*, 45. Dipetik March 6, 2024, dari https://link.springer.com/content/pdf/10.1007/s10664-022-10285-5.pdf
[9] Rahman, A., Agrawal, A., Krishna, R., & Sobran, A. (2018). Characterizing The Influence of Continuous Integration. Empirical Results from 250+ Open Source and Proprietary Projects. 7.
[10] RedHat. (2023, December 12). *RedHat*. Dipetik March 8, 2024, dari What is CI/CD?: https://www.redhat.com/en/topics/devops/what-is-ci-cd