1. Pendahuluan

Arsitektur monolitik mendesain perangkat lunak hanya terdiri dari satu bagian besar dan berjalan pada satu *computational instance* [1]. Meskipun begitu, arsitektur monolitik memungkinkan untuk menjalankan lebih dari satu proses pada satu CPU namun semuanya berbagi satu sistem operasi dan perangkat keras yang sama. Hal ini dapat menyebabkan diantaranya kurangnya fleksibilitas dan skalabilitas [1]. Ketika akan melakukan peningkatan atau memperbaiki kerusakan suatu layanan, dalam arsitektur monolitik harus mematikan seluruh layanannya. Tidak hanya itu, *update* sekecil apapun sistem harus di re-*compile* dan *deploy* secara keselurahan, agar *update* bisa sampai kepengguna. Permasalahan tersebut yang dicoba untuk diatasi oleh Heartenly.com.

Heartenly.com merupakan salah satu aplikasi pencari jodoh asal Indonesia, yang saat ini mempunyai jumlah pengguna lebih dari 300.000 dan masih terus bertambah. Heartenly.com berencana untuk menambahkah layanan-layanan baru dan memperbaiki yang sudah ada agar lebih memanjakan penggunanya. Namun saat ini Heartenly.com masih menggunakan arsitektur monolitik. Seperti yang sudah dijelaskan sebelumnya, dalam monolitik harus mematikan seluruh layanan ketika akan menambah atau memperbaiki suatu layanan tertentu. Untuk itu, sebelum melakukan penambahan atau peningkatan layanan, Heartenly.com harus mempersiapkan arsitekturnya terlebih dahulu agar mudah untuk melakukan peningkatan atau perbaikan layanan. Arsitektur tersebut adalah arsitektur *Microservice* [1].

Arsitektur *microservice* mendekomposisi satu entitas besar atau layanan besar menjadi layanan – layanan yang kecil sehingga layanan tersebut independen namun secara *autonomous* masih saling bekerja sama [2]. Tidak ada ukuran yang pasti dalam menentukan seberapa kecil suatu layanan didekomposisi, tetapi faktor penetapan ukuran dapat dilihat dari keselarasan tim untuk menangani kompleksitas sistem dari hasil dekomposisi layanan [2]. Hasil dekomposisi layanan tersebut kemudian dihubungkan menggunakan arsitektur REST (Representational State Transfer), sebuah gaya pendekatan dalam membangun arsitektur perangkat lunak yang memisahkan antara *Client* dengan Server atau sebagai medium antara *Client* dengan *Service* [3]. Client dan Server berkomunikasi mengikuti standar protokol *Hypertext Transfer Protocol* (HTTP) [4]. REST dipilih selain karena Heartenly.com sudah menggunakanya, tetapi juga karena penyedia *Web* saat ini dianggap dapat menyediakan skalabilitas, keamanan dan *reliability* [4]. HTTP dapat diimplementasikan untuk berbagai macam kebutuhan seperti *transfer file, streaming*, dan *shared database* [5]. Kombinasi keunggulan *microservice* dan REST ini sejalan dengan kebutuhan Heartenly.com, yaitu dapat menambah dan/atau memperbaiki layanannya, secara independen, maka Heartenly.com perlu merubah arsitekturnya dari monoilitik ke *microservice.*

Berdasarkan permasalahan di atas dibutuhkan analisis dan desain arsitektur untuk mendekomposisi layanan-layanan Heartenly.com, sehingga dapat menangani permasalahan di atas. Dekomposisi yang dilakukan menggunakan pendekatan pendekatan *Domain-Driven Design* (DDD). DDD merupakan suatu metode untuk memodelkan kegiatan domain organisasi yang ada di dunia nyata [2]. Dalam metode DDD pengembang dan analis duduk bersama untuk memahami konsep domain, *policies*, dan logika [6]. Analisis desain ini dapat digunakan sebagai referensi untuk pengembangan Heartenly.com selanjutnya.

Rumusan Masalah

Berdasarkan pendahuluan yang telah di jelaskan sebelumnya, maka didapat beberapa rumusan masalah, yaitu :

1. Bagaimana mendesain arsitektur *Microservices* dengan REST pada Heartenly.com?
2. Bagaimana perbedaan kemudahan *upgrade* antara arsitektur Monolitik dengan *Microservice*?

Tujuan

Tujuan dari penelitian tugas akhir ini yaitu

1. Mendesain arsitektur *Microservice* dengan REST pada Heartenly.com
2. Membandingkan kemudahan *upgrade* arsitektur Monolitik dan *Microservice*