BAB 1

PENDAHULUAN

1.1 Latar Belakang

Pengumuman di jurusan Teknik Informatika UNPAR pada umumnya dilakukan oleh email. Pengumuman lewat email ini praktis karena tidak perlu menunggu sampai email sampai ke tujuan dan dijamin sampai ke tujuan. Selain itu, konten yang disampaikan melalui email fleksibel. Konten tidak harus hanya tulisan tapi dapat menambahkan lampiran, mengubah gaya tulisan dan lain-lain.

Namun, email kurang terorganisir dengan baik. Email yang masuk sering tercampur dengan email lain sehingga mahasiswa kesulitan mencari email yang penting. Dampaknya, pengumuman-pengumuman penting sering tidak terbaca secara tidak sengaja.

Pada skripsi ini, akan dibuat solusi masalah tadi dengan membuat suatu fitur. Fitur ini akan menangkap email-email pengumuman yang masuk ke sebuah email khusus untuk menangkap pengumuman. Pertama email yang masuk ke email khusus akan diperiksa pengirimnya. Apabila pengirim adalah email yang terdaftar sebagai email yang berhak melakukan pengumuman maka email tersebut adalah email pengumuman. Setelah itu email tersebut akan dibuatkan permanent link dan disisipkan pada basis data. Lalu, mahasiswa akan menerima permanent link tersebut melalui notifikasi dari akun Line@. Line@ adalah layanan dari Line Corporation yang memudahkan pemilik bisnis atau organisasi menyampaikan pesan kepada pengikutnya di aplikasi pengirim pesan Line.

Sistem ini akan dibangun sebagai fitur tambahan pada BlueTape, sebuah website milik jurusan teknik Informatika Unpar. Pembangunan fitur ini membutuhkan modifikasi BlueTape sehingga dapat dijalankan di Heroku dan menggunakan basis data PostgreSQL. Heroku adalah layanan yang memungkinkan pengembang membangun, menjalankan, dan mengoperasikan aplikasi di dalam internet. Selain itu, sistem ini membutuhkan beberapa fitur dari layanan surel GMail dan layanan pengirim pesan instan Line@.

1.2 Rumusan Masalah

- Bagaimana cara mengubah BlueTape agar dapat mendukung fitur kolektor pengumuman dengan bantuan Heroku dan PostgreSQL ?
- Bagaimana cara mengimplementasikan kolektor pengumuman pada BlueTape ?

1.3 Tujuan

- Melakukan perawatan pada BlueTape agar dapat mendukung fitur kolektor pengumuman dengan bantuan Heroku dan PostgreSQL
- Mengimplementasikan fitur kolektor pengumuman pada BlueTape

1.4 Batasan Masalah

Pada skripsi ini masalah dibatasi dengan batasan-batasan sebagai berikut :

- Sistem ini dibuat hanya untuk jurusan teknik informatika Unpar, mahasiswa dari jurusan lain tidak dapat menggunakan sistem ini

1.5 Metodologi

Metode penelitian pada skripsi ini sebagai berikut :

1. Melakukan studi literatur tentang GMail, Line, Heroku dan PostgreSQL
2. Memodifikasi BlueTape sehingga dapat menangkap email yang masuk ke email khusus
3. Memodifikasi BlueTape sehingga dapat melakukan push notification ke akun Line@
4. Memodifikasi BlueTape sehingga dapat berjalan di Heroku menggunakan PostgreSQL
5. Melakukan pengujian
6. Menulis dokumen skripsi

1.6 Sistematika Pembahasan

1. Bab 1 : Pendahuluan Bab ini membahas gambaran umum dari skripsi.
2. Bab 2 : Dasar Teori Bab ini membahas dasar teori yang mendukung pembuatan skripsi ini.
3. Bab 3 : Analisis Bab ini membahas analisis yang dilakukan terhadap masalah yang diusung dalam skripsi ini.
4. Bab 4 : Perancangan Bab ini membahas perancangan sistem yang dibangun pada skripsi ini.
5. Bab 5 : Implementasi dan Pengujian Bab ini membahas hasil implementasi yang dilakukan beserta pengujian sistem.
6. Bab 6 : Kesimpulan dan saran Bab ini berisi kesimpulan yang didapat dari penelitian dan saran oleh penulis kepada pembaca yang hendak melanjutkan penelitian ini.

BAB 2

LANDASAN TEORI

2.1 *BlueTape*

BlueTape adalah perangkat lunak yang berfungsi untuk membantu urusan-urusan *paper-based* di FTIS UNPAR menjadi *paperless*. Perangkat lunak ini berbasis web dengan memanfaatkan *CodeIgniter* dan *ZURB Foundation*. Saat ini aplikasi ini memiliki dua layanan, yaitu Transkrip *Request / Manage* dan Perubahan Kuliah *Request / Manage*. Layanan Transkrip *Request / Manage* memberikan layanan untuk melakukan permohonan serta pencetakan transkrip mahasiswa. Sedangkan layanan Perubahan Kuliah *Request / Manage* memberikan layanan untuk permohonan dan pencetakan perubahan jadwal kuliah oleh dosen. [1]

2.2 Heroku

Heroku adalah *platform cloud* yang memungkinkan pengembang untuk membangun, menjalankan, dan mengoperasikan perangkat lunak pada *cloud*. Heroku mendukung beberapa bahasa pemrograman, meliputi : Ruby, Node.js, Java, Python, Clojure, Scala, Go, dan PHP. [2]

2.2.1 Arsitektur Heroku

Heroku memungkinkan seorang pengembang untuk menyebarkan (*deploy*), menjalankan (*run*), dan mengelola (*manage*) perangkat lunak yang ditulis di dalam bahasa Ruby, Node.js, Java, Python, Clojure, Scala, Go, dan PHP. Heroku mendefinisikan perangkat lunak sebagai gabungan dari *source code* yang ditulis di dalam salah satu bahasa ini (dapat berupa *framework*), deskripsi dependensi yang dipakai, dan Procfile (jika diperlukan).

2.2.1.1 Dependensi

Pengembang perlu mendeskripsikan dependensi tambahan yang diperlukan agar perangkat lunak dapat dibangun dan dijalankan. Aturan penulisan deskripsi dependensi berbeda-beda untuk tiap bahasa. Contohnya pada bahasa Ruby deskripsi dependensi dituliskan dalam dokumen `Gemfile`, pada bahasa Python ditulis di dokumen `requirements.txt`, pada bahasa Node.js ditulis di dokumen `package.json`, pada bahasa Java ditulis di dokumen `pom.xml`, dan seterusnya.

2.2.1.2 Tipe Proses (Proccess Type)

Di dalam perangkat lunak web, terdapat dua atau lebih titik masuk. Itu artinya ada dua atau lebih perintah untuk menjalankan perangkat lunak. Setiap titik masuk ini disebut tipe proses (`process type`). Tipe proses berbeda untuk tiap perangkat lunak.

Tipe proses dapat berupa web server dari perangkat lunak, berbagai tipe *worker process*, *singleton process* (contoh : clock), dan tugas yang harus dijalankan sebelum sebuah release disebar. Di antara

[1] https://github.com/ftisunpar/BlueTape
[2] https://www.heroku.com

beragam tipe proses tersebut, ada dua tipe proses spesial : tipe proses `web` dan `release`. Tipe proses `web` adalah satu-satunya tipe proses yang dapat menerima arus HTTP eksternal dari router Heroku. Jika sebuah perangkat lunak melibatkan web server, pengembang harus menyatakannya sebagai proses `web`. Tipe proses `release` adalah tipe proses yang digunakan untuk menyebutkan perintah yang dijalankan selama fase release.

Tipe proses dan dyno saling berhubungan. Tipe proses adalah prototipe yang menjadi tempat dimana dyno dibentuk. Semakin banyak dyno pada suatu tipe proses maka konkurensi untuk pekerjaan yang ditangani tipe proses tersebut akan meningkat. Semakin banyak tipe proses maka semakin beragam beban kerja.

Setiap tipe proses memiliki spesialisasinya sediri pada jenis pekerjaan tertentu. Misalnya, sebuah perangkat lunak memiliki dua jenis pekerja : pekerja untuk pekerjaan yang mendesak dan pekerja untuk pekerjaan yang berlangsung dengan lama. Dengan melakukan pembagian seperti itu, pekerjaan yang mendesak lebih cepat tertangani. Selain itu, penggunaan sumber daya lebih mudah dikontrol dan dikalkulasi.

Untuk mengatur berapa banyak dyno yang bekerja di suatu proses, perintah yang dibutuhkan adalah `heroku ps:scale` diikuti dengan daftar pasangan tipe proses dengan jumlah dyno yang ditugaskan untuk proses tersebut. Contoh :

```
heroku ps:scale web=2 worker=4 clock=1
```

Selain dapat mengatur jumlah dyno yang ditugaskan pada suatu pekerjaan, jangka waktu pekerjaan itu dilakukan juga dapat diatur. Caranya dengan menggunakan add-on Heroku Scheduler atau menggunakan tipe proses khusus untuk mengatur jadwal pekerjaan.

2.2.1.3 Procfile

Pengembang perlu memberitahu Heroku bagian perangkat lunak yang dapat dijalankan. Jika pengembang menggunakan framework yang sudah ada, Heroku dapat mencari tahu. Contohnya, di dalam Ruby pada Rails, biasanya `rails server`, di dalam Django adalah `python <app>/manage.py runserver` dan di dalam Node.js adalah `main` field di dalam `package.json`. Untuk perangkat lunak lain, pengembang mungkin perlu untuk secara eksplisit menyatakan apa yang harus dieksekusi. Caranya dengan menyertakan sebuah dokumen teks bernama Procfile.

Dokumen Procfile tidak memiliki ekstensi dokumen, seperti `.txt`, `.docx`, `.jpg` dan lain-lain. Apabila Procfile diberi ekstensi dokumen (contoh : Procfile.txt), maka Procfile tersebut tidak sah. Selain itu, Procfile harus diletakkan di direktori root. Jika diletakkan di tempat lain, Procfile tidak akan berfungsi sebagaimana mestinya.

Isi dari Procfile adalah satu atau lebih baris yang menyatakan sebuah tipe proses. Format penulisan tiap baris Procfile adalah :

```
<process type>: <command>
```

`<process type>` (tipe proses) adalah nama perintah yang mengandung huruf dan angka, seperti : `web`, `worker`, `urgentworker`, `clock`, dan lain-lain. Untuk perangkat lunak sederhana, pengembang cukup menuliskan tipe proses `web` saja. `<command>` adalah perintah yang setiap dyno dari tipe proses tersebut harus jalankan pada saat startup, seperti `rake jobs : work`. Contoh isi Procfile :

```
web: java -jar lib/foobar.jar \textdollar PORT
```

Pada contoh, `web` merupakan `<process type>`, sedangkan `web: java -jar lib/foobar.jar $PORT` adalah perintah yang harus dijalankan pada saat startup.

Heroku adalah polyglot platform : Heroku memungkinkan Anda membangun, menjalankan, dan mengatur perangkat lunak di cara yang serupa dalam segala bahasa pemrograman - menggunakan dependensi dan Procfile. Procfile menyingkap aspek arsitektur dari perangkat lunak dan arsitektur ini memungkinkan pengembang, contohnya, mengatur setiap bagian secara independen. Namun, membuat Procfile tidak wajib untuk sebagian besar bahasa pemrograman yang didukung Heroku. Heroku akan secara otomatis mendeteksi bahasa yang digunakan, dan membuat tipe proses `web` untuk menjalankan server perangkat lunak. Apabila perangkat lunak menggunakan `heroku.yml` sebagai build manifest, Procfile juga tidak diwajibkan. Perintah yang disebutkan di bagian `run` pada `heroku.yml` harus mengikuti format yang sama dengan format Procfile.

2.2.1.4 Deploy Perangkat Lunak

Heroku menggunakan Git sebagai sarana utama untuk melakukan deploy perangkat lunak. Deploy disini adalah proses penyebaran perangkat lunak dari satu lingkungan ke lingkungan lain, misalnya dari lingkungan mesin pengembang perangkat lunak ke lingkungan heroku. Sedangkan Git adalah sistem version control terdistribusi yang kuat yang digunakan banyak pengembang untuk mengelola dan menerjemahkan source code (https://git-scm.com/). Heroku juga menyediakan cara lain untuk melakukan deploy, seperti dengan menggunakan GitHub integration atau Heroku API.

Ketika perangkat lunak baru dibuat di Heroku, Heroku secara otomatis membuat Git remote baru. Heroku secara otomatis menamainya `Heroku`, namun pengembang dapat menggantinya. Git remote ini terhubung dengan perangkat lunak tersebut dan berfungsi sebagai alamat yang digunakan saat melakukan deploy. Sehingga apabila pengembang ingin melakukan deploy, maka pengembang hanya perlu melakukan `git push` diikuti dengan nama remote dan nama branch repository. Contoh :

```
$ git push heroku master
```

2.2.1.5 Membangun Perangkat Lunak

Ketika platform Heroku menerima source code perangkat lunak, heroku akan memulai pembangunan (build) berdasarkan source code. Mekanisme build biasanya tergantung bahasa pemrograman yang dipakai, tapi mengikuti pola yang sama. Mekanisme build biasanya mengambil dependensi yang ditentukan, dan menciptakan aset yang diperlukan. Contohnya, perangkat lunak Java mungkin mengambil dependensi library biner menggunakan Maven, mengompilasinya, lalu menghasilkan dokumen JAR untuk dieksekusi.

Source code untuk perangkat lunak, dependensi yang diraih, dan hasil dari fase build digabungkan ke dalam slug. Slug adalah gabungan dari source code, dependensi yang diambil, language runtime, dan hasil kompilasi atau keluaran yang dihasilkan oleh sistem build yang siap untuk dieksekusi. Slug ini adalah aspek dasar dari eksekusi perangkat lunak. Slug berisi perangkat lunak yang sudah dikompilasi, digabungkan, dan siap untuk dijalankan.

Di belakang proses kompilasi slug terdapat buildpack. Buildpack mengambil perangkat lunak, dependensi, dan language runtime dan kemudian menghasilkan slug. Buildpack bersifat open source, sehingga memungkinkan pengembang memperluas Heroku ke bahasa pemrograman lain dan framework.

2.2.1.6 Buildpack

Buildpack bertanggung jawab untuk mengubah source code menjadi slug, sehingga dyno dapat mengeksekusinya. Buildpack terdiri dari sekumpulan script yang ditulis dalam bahasa pemrograman

yang sama dengan source code. Script tersebut akan mengambil dependensi, mengeluarkan aset atau kode yang sudah dikompilasi, dan sebagainya. Keluaran ini akan digabungkan ke dalam slug oleh slug compiler.

Heroku akan mencari buildpack yang sesuai dan menggunakannya untuk mengompilasi perangkat lunak. Jika build sukses, buildpack yang sudah terdeteksi sesuai akan secara permanen diatur untuk push selanjutnya. Buildpack yang telah dimodifikasi dapat dipakai untuk mendukung bahasa atau framework yang tidak dapat di cakup oleh buildpack resmi.

Buildpack yang digunakan dapat diubah dengan mengatur nilai buildpack. Caranya dengan mengetikkan perintah ini pada command shell :

```
$ heroku buildpacks:set <nama buildpack>
```

<nama buildpack> adalah nama buildpack yang ingin dipakai, contoh : `heroku/php`. Buildpack baru akan digunakan saat push berikutnya.

Apabila pengembang ingin mengatur buildpack yang dipakai saat perangkat lunak pertama kali dibuat, maka dapat menggunakan perintah ini :

```
$ heroku create myapp --buildpack <nama buildpack>
```

<nama buildpack> adalah nama buildpack yang ingin dipakai.

Buildpack juga dapat secara eksplisit diatur di dalam app.json sehingga perangkat lunak yang dibuat menggunakan tombol Heroku dapat menggunakan buildpack yang telah dimodifikasi.

Apabila pengembang ingin menghilangkan buildpack dari perangkat lunak, pengembang dapat mengetikkan perintah :

```
$ heroku buildpacks:remove <nama buildpack>
```

<nama buildpack> adalah nama buildpack yang ingin dipakai. Hal yang harus diperhatikan adalah jika buildpack tidak ada, maka proses mendeteksi buildpack yang sesuai akan dijalankan lagi saat push berikutnya.

Selain dapat menggunakan buildpack yang telah dimodifikasi, terdapat third-party buildpack tersedia di Element marketplace atau menggunakan CLI. Pada CLI, ketikkan perintah :

```
$ heroku buildpacks:search <kata kunci>
```

<kata kunci> misalnya adalah nama bahasa pemrograman yang dipakai, misalnya `elixir`. Apabila ingin melihat informasi tentang buildpack yang didapat, maka ketikkan perintah :

```
$ heroku buildpacks:info <kata kunci>
```

<nama buildpack> adalah nama buildpack yang ingin dipakai. Dengan perintah tersebut, informasi seperti deskripsi, kategori, lisensi, source, dan informasi lainnya dapat dilihat.

Apabila pengembang ingin mengembalikan perangkat lunak ke buildpack awalnya, maka pengembang dapat mengetikkan perintah :

```
$ heroku buildpacks:clear
```

Biasanya buildpack yang dipakai oleh perangkat lunak hanya satu, tapi ada beberapa kasus buildpack yang dipakai tidak cukup hanya satu. Beberapa kasus tersebut adalah :

- Menjalankan buildpack untuk tiap bahasa pemrograman yang perangkat lunak gunakan. Contohnya, menjalankan JavaScript buildpack untuk aset dan buildpack Ruby untuk perangkat lunak.
- Menjalankan proses daemon seperti `pgbouncer` dengan perangkat lunak.
- Menarik dependensi sistem dengan apt.

Pengembang dapat menentukan buildpack-buildpack yang diperlukan perangkat lunak dan mengatur urutan eksekusinya dengan Heroku CLI. Untuk menambahkan buildpack yang ingin dipakai, pengembang dapat mengetikkan perintah :

```
$ heroku buildpacks:add --index <index> <nama buildpack>
```

<index> adalah urutan eksekusi buildpack, sedangkan <nama buildpack> adalah nama buildpack yang ingin dipakai.

Untuk mengubah buildpack yang dipakai pada urutan eksekusi tertentu, pengembang dapat mengetikkan perintah :

```
$ heroku buildpacks:set --index <index> <nama buildpack>
```

<index> adalah urutan eksekusi buildpack, sedangkan <nama buildpack> adalah nama buildpack yang ingin dipakai.

Untuk melihat daftar buildpack yang dipakai oleh perangkat lunak, ketikkan perintah :

```
$ heroku buildpacks
```

Buildpack terakhir pada daftar adalah buildpack yang digunakan untuk menentukan tipe proses dari perangkat lunak. Tipe proses lain yang disebutkan akan diabaikan.

Untuk melihat daftar perintah yang dapat digunakan untuk mengatur buildpack, pengembang dapat mengetikkan perintah :

```
$ heroku help buildpacks
```

2.2.1.7 Dyno

Heroku mengeksekusi perangkat lunak dengan menjalankan perintah yang telah ditulis di dalam Procfile. Heroku akan menjalankan dyno yang telah dimuat dengan slug yang telah disiapkan. Dyno adalah wadah perangkat lunak berbasis Unix yang terisolasi, tervirtualisasi, dan menyediakan lingkungan yang dibutuhkan untuk menjalankan suatu perangkat lunak. Umumnya, jika perangkat

lunak di-deploy ke Heroku untuk pertama kali, Heroku akan menjalankan satu web dyno secara otomatis.

Setiap dyno termasuk dalam salah satu dari konfigurasi berikut :

- Web : Web dyno adalah dyno dari tipe proses "web" yang disebutkan di dalam Procfile. Web dyno adalah satu-satunya dyno yang dapat menerima arus HTTP dari router Heroku.
- Worker : Worker dyno adalah dyno dari tipe proses apapun selain "web" yang disebutkan di dalam Procfile. Worker dyno biasanya digunakan untuk pekerjaan di latar belakang, sistem antrian, dan pekerjaan yang memiliki jangka waktu.
- One-off : One-off dyno adalah dyno yang bersifat sementara yang dapat berjalan terpisah atau dengan masukan/keluaran dari terminal lokal.

. Dyno ini dapat digunakan untuk tugas yang bersifat administratif, seperti migrasi basis data dan sesi konsol. Dyno ini juga dapat digunakan untuk melakukan pekerjaan di latar belakang yang bersifat sesekali, seperti Heroku Scheduler.

Heroku menyediakan beberapa tipe dyno yang berbeda. Tiap dyno memiliki sifat yang unik dan kinerja yang berbeda. Untuk semua pengguna Heroku, tersedia pilihan dyno tipe Free, Hobby, Standard, dan Performance. Ada satu tipe dyno lagi, yaitu tipe Private. Dyno tipe Private hanya tersedia di Heroku Enterpise yang diperuntukkan untuk organisasi.

Dyno dapat diskala secara horizontal dan vertikal. Untuk menskala secara horizontal, tambahkan lebih banyak dyno. Contohnya, menambah web dyno agar dapat menangani arus yang lebih besar. Untuk menskala secara vertikal, gunakan dyno yang lebih besar. Dyno yang lebih besar berarti jumlah pemakaian memori RAM yang lebih besar. Jumlah RAM maksimal yang tersedia untuk perangkat lunak tergantung dari tipe dyno yang digunakan. Penskalaan secara horizontal dan vertikal ini adalah fitur dari dyno profesional, dan tidak tersedia untuk dyno tipe free dan hobby.

Perangkat lunak dengan banyak dyno yang berjalan akan memiliki resiko kegagalan yang lebih rendah daripada yang sedikit. Jika ada dyno yang hilang, perangkat lunak dapat terus memproses permintaan sementara dyno yang hilang diganti. Dyno yang hilang biasanya langsung dimulai ulang, tapi terkadang membutuhkan waktu yang lama.

Semua dyno terisolasi dari dyno lain untuk alasan keamanan. Walaupun dyno tipe Free, Hobby, dan Standard terisolasi, dyno mungkin berbagi komputasi dasar yang sama. Heroku memiliki teknik tersendiri untuk memastikan penggunaannya adil. Di sisi lain, dyno tipe Performance dan Private tidak berbagi komputasi dasar yang sama dengan dyno lain. Hal ini membuat dyno tipe Performance dan Private memiliki kinerja yang lebih stabil dibanding dengan dyno tipe Free, Hobby, dan Standard. Selain memiliki sumber daya komputasi yang dikhususkan untuknya, Private dyno juga memiliki jaringan virtual yang terisolasi.

Tiap dyno memiliki sistem dokumen ephemeral, dengan salinan kode dari hasil deploy terbaru. Saat masa hidup dyno, proses yang dijalankannya dapat menggunakan sistem dokumen ini sebagai tempat menulis sementara. Namun, dokumen yang ditulis tidak dapat dilihat oleh proses dari dyno lain dan dokumen yang ditulis akan dihapus saat dyno berhenti bekerja atau dimulai ulang.

Tiap dyno memiliki antarmuka jaringannya sendiri. Konfigurasi dari jaringan bergantung pada tipe Runtime yang dipakai. Ada dua tipe Runtime : Common Runtime dan Private Space Runtime.

Common Runtime menyediakan isolasi yang kuat dengan menghalangi tiap dyno dari dyno lainnya. Pada runtime ini, hanya web dyno yang dapat mendengarkan angka yang disebutkan di variabel `$PORT` dan melakukan inbound request. Tiap dyno pada runtime ini memiliki IP atau subnet sendiri. Semua tipe dyno dapat membuat outbound request ke layanan yang bekerja di tempat lain selama terkoneksi ke internet. Namun, alamat IP asal untuk permintaan ini tidak dapat dikontrol oleh pengguna.

Private Space Runtime menghubungkan dyno lewat jaringan virtual privat yang dikonfigurasi sebagai bagian dari suatu ruang. Pada runtime ini, dyno jenis apapun dapat mendengarkan `$PORT` dan menerima koneksi dari dyno lain pada jaringan privat. Runtime ini memungkinkan penggunaan

Trusted IP Ranges untuk mengontrol IP klien yang dapat berkomunikasi dengan perangkat lunak dalam Private Space. Outbound request ke layanan internet dilakukan melalui NAT gateway untuk memastikan koneksi asal berasal dari kumpulan alamat IP outbound yang stabil.

2.2.1.8 Dyno manager

Dyno manager adalah bagian dari Heroku yang bertanggungjawab untuk menjaga dyno tetap berjalan. Dyno manager melakukan pekerjaan seperti memastikan dyno didaur ulang setidaknya satu kali sehari atau setiap dyno manager mendeteksi kesalahan di dalam perangkat lunak yang berjalan. Daur ulang dyno ini berlangsung secara transparan dan otomatis secara teratur dan tercatat.

Perangkat lunak yang menggunakan dyno tipe Free akan masuk mode sleep (tidur) jika tidak ada arus HTTP selama jangka waktu 30 menit. Ketika perangkat lunak yang tidur menerima arus HTTP, maka perangkat lunak tersebut akan terbangun. Hal ini menyebabkan perangkat lunak lebih lambat beberapa detik dari perangkat lunak yang menggunakan dyno tipe lain. Dyno tipe lain tidak memiliki mode sleep, dan akan selalu terjaga.

2.2.1.9 Config vars

Suatu perangkat lunak selalu berjalan di lingkungan (environment) yang berbeda, meliputi setidaknya mesin yang digunakan untuk mengembangkan perangkat lunak dan Heroku. Suatu perangkat lunak yang bersifat open-source mungkin disebar ke lingkungan yang berbeda. Walaupun lingkungan-lingkungan tersebut mungkin menjalankan kode yang sama, lingkungan-lingkungan tersebut biasanya memiliki konfigurasi khusus untuk tiap lingkungan. Konfigurasi ini harus diletakkan pada environment variable, bukan di source code. Dengan menggunakan environment variable, konfigurasi dapat diubah secara terpisah. Selain itu, konfigurasi yang bersifat kredensial dapat terhindar dari tersimpan pada version control. Pada host tradisional atau saat bekerja secara lokal, environment variable sering diset di dalam dokumen `.bashrc`.

Heroku memungkinkan pengembang untuk menjalankan perangkat lunak dengan konfigurasi yang dapat diubah dengan mudah. Konfigurasi tersebut diletakkan di luar dari source code perangkat lunak. Konfigurasi dapat diubah secara independen tanpa harus mengubah source code. Konfigurasi tersebut disimpan di dalam config vars.

Untuk mengatur config vars ada tiga cara : dengan menggunakan Heroku CLI, dengan menggunakan Heroku Dashboard, dan dengan menggunakan Heroku Platform API. Apabila menggunakan Heroku Platform API, config var dapat diatur menggunakan HTTPS REST client sederhana dan data struktur JSON. Apabila menggunakan Heroku Dashboard, config var dapat dilihat, ditambah, dan dihapus dengan mudah menggunakan tombol yang ada . Sedangkan apabila menggunakan Heroku CLI, config var diatur menggunakan command shell. Berikut perintah-perintah untuk mengatur config var menggunakan Heroku CLI:

- Menampilkan seluruh config var beserta nilainya : `$heroku config`
- Menampilkan nilai dari config var tertentu : `$heroku config : get <config var>` dimana `config var` adalah nama config var
- Menambah config var : `$heroku config:set <config var> = <config value>` dimana `config var` adalah nama config var dan `config value` adalah nilai dari config var tersebut
- Menghapus config var : `$heroku config:unset <config var>` dimana `config var` adalah nama config var

Dalam mengatur config var, ada beberapa hal yang harus diperhatikan :

- Setiap config var ditambah atau dihapus, perangkat lunak akan dimulai ulang dan release baru akan dibuat.

- Jika perangkat lunak menggunakan add-on, biasanya add-on tersebut akan menambahkan satu atau lebih config var ke perangkat lunak. Nilai dari config var tersebut mungkin diperbarui oleh penyedia add-on kapan saja.
- Config var data (kombinasi dari semua kunci dan nilainya) tidak dapat melebihi 32kb per perangkat lunak
- Nama config var tidak boleh diawali dengan garis bawah dua kali (`__`).
- Nama config var tidak bisa diawali dengan `HEROKU_`, kecuali ditambahkan oleh platform Heroku sendiri.

2.2.1.10 Releases

Releases adalah buku besar yang mencatat setiap ada deploy baru. Deploy baru ini termasuk build perangkat lunak, perubahan di config vars, dan perubahan di daftar add-ons. Pengembang dapat melihat catatan releases ini dengan menggunakan perintah :

```
$ heroku releases
```

Isi dari releases adalah satu atau lebih baris dari release yang tiap barisnya memiliki format : `<versi deploy> Deploy <commit hash> <username pengembang> <waktu deploy>` Contoh isi releases :

```
== demoapp Releases
v103 Deploy 582fc95 jon@heroku.com 2013/01/31 12:15:35
v102 Deploy 990d916 jon@heroku.com 2013/01/31 12:01:12
```

Releases ini berguna saat pengembang ingin mengembalikan perangkat lunak ke deploy lama. Cara mengembalikan perangkat lunak ke deploy lama dengan mengetikkan perintah :

```
$ heroku releases:rollback <versi deploy>
```

Contoh :

```
$ heroku releases:rollback v102
```

2.2.1.11 Add-ons

Perangkat lunak biasanya memanfaatkan add-ons untuk menyediakan layanan penyokong seperti basis data, sistem antrean, layanan email, dan lainnya. Add-ons disediakan oleh Heroku atau pihak ketiga. Cara menambah addons melalui command shell :

```
$ heroku addons:create <nama addons>:<tipe addons>
```

Contoh :

```
$ heroku addons:create heroku-redis:hobby-dev
```

2.2.1.12 Log

Log adalah catatan setiap proses yang terjadi di perangkat lunak. Heroku menggunakan Logplex untuk menyampaikan log ini. Logplex akan secara otomatis menambahkan entri log baru dari semua dyno yang berjalan di perangkat lunak, dan juga komponen lain seperti router. Pengembang dapat memeriksa log dengan cara mengetikkan perintah :

```
$ heroku logs
```

Contoh isi log adalah :

```
2013-02-11T15:19:10+00:00 heroku[router]: at=info method=GET path=/
    ↪ articles/custom-domains host=mydemoapp.heroku.com fwd=74.58.173.188
    ↪  dyno=web.1 queue=0 wait=0ms connect=0ms service=1452ms status=200
    ↪ bytes=5783
2013-02-11T15:19:10+00:00 app[web.2]: Started GET "/" for 1.169.38.175 at
    ↪ 2013-02-11 15:19:10 +0000
2013-02-11T15:19:10+00:00 app[web.1]: Started GET "/" for 2.161.132.15 at
    ↪ 2013-02-11 15:20:10 +0000
```

2.2.1.13 Region

Perangkat lunak di dalam Heroku dapat disebarkan ke lokasi geografis yang berbeda. Lokasi yang tersedia untuk suatu perangkat lunak tergantung pada Runtime yang dipakai oleh perangkat lunak (Common Runtime atau Private Space). Untuk perangkat lunak yang memakai Common Runtime, pengembang perlu menyebutkan region perangkat lunak saat membuat perangkat lunak. Untuk perangkat lunak yang memakai Private Space, region diatur saat membuat Private Space. Apabila pengembang tidak menyebutkan region yang dipakai, maka region akan diisi secara otomatis sebagai `us` (apabila memakai Common Runtime) atau `virginia` (apabila memakai Private Spaces).

Untuk memeriksa region yang tersedia di Heroku, pengembang dapat mengetikkan perintah :

```
$ heroku regions
```

Pengembang juga dapat memeriksa daftar region yang tersedia berdasarkan Runtime yang dipakai :

```
$ heroku regions <runtime>
```

<runtime> disini adalah `-common` untuk melihat daftar untuk Common Runtime dan `-private` untuk melihat daftar untuk Private Spaces.

Untuk mengatur region perangkat lunak, ketikkan perintah :

```
$ heroku create --region <id region>
```

<id region> disini diisi dengan id region yang ingin dipakai, contoh : `eu`.Id region bisa dilihat dengan memeriksa daftar region yang tersedia.

Untuk memeriksa region yang dipakai oleh perangkat lunak, pengembang dapat mengetikkan perintah :

```
$ heroku info
```

2.2.1.14 Stack

Stack adalah operating system image yang dikelola dan dipelihara oleh Heroku. Stack biasanya berlandaskan distribusi dari Linux yang ada, seperti Ubuntu. Pengembang dapat menentukan stack yang dipakai, dan buildpack akan mengubah source code menjadi paket yang dapat dieksekusi dengan stack tersebut. Saat skripsi ini dibuat, Heroku menyediakan tiga stack : Cedar-14, Heroku-16, dan Heroku-18. Cedar-14 berbasis Ubuntu 14.04 dan didukung sampai bulan April tahun 2019. Heroku-16 berbasis Ubuntu 16.04 dan didukung sampai bulan April tahun 2021. Heroku-18 berbasis Ubuntu 18.04 dan didukung sampai bulan April tahun 2023. Semua buildpack dari Heroku dapat bekerja dengan ketiga stack tersebut, namun buildpack yang merupakan hasil modifikasi belum tentu dapat bekerja dengan semua stack.

Untuk melihat stack yang dipakai oleh perangkat lunak, pengembang dapat mengetikkan perintah :

```
$ heroku stack
```

Untuk mengganti stack yang dipakai, pengembang dapat mengetikkan perintah :

```
$ heroku stack:set <stack>
```

<stack> disini dapat diisi dengan `cedar-14`, `heroku-16`, atau `heroku-18`.

2.2.2 Command Line

Pengembang perlu memasang heroku Command-Line Interface (CLI) agar dapat mengetikkan perintah-perintah `heroku` di command shell. Pengembang dapat mengikuti petunjuk unduhan pada `https://devcenter.heroku.com/articles/heroku-cli` untuk memasang heroku CLI sesuai sistem operasi yang ia pakai. Pengembang dapat memastikan heroku CLI sudah terpasang dengan mengetikkan perintah :

```
$ heroku -version
```

Setelah memasang heroku CLI, hal pertama yang harus dilakukan adalah masuk ke akun heroku. Caranya dengan mengetikkan perintah :

```
$ heroku login
```

Untuk membuat perangkat lunak yang baru di heroku, pengembang perlu mengetikkan perintah :

```
$ heroku create
```

2.2.3 Deployment

Sebagian besar proses deploy ke Heroku menggunakan Git, tapi pengembang juga dapat melakukannya dengan :

- Docker
- GitHub
- Tombol `Deploy` di situs Heroku
- WAR deployment

2.2.3.1 Deploy Menggunakan Git

Untuk melakukan deploy menggunakan Git, pengembang harus sudah memasang Git. Pengembang dapat mengikuti petunjuk unduhan pada https://git-scm.com. Sebelum pengembang dapat melakukan deploy dengan Git, pengembang perlu menginisialisasi git. Berikut perintah-perintah yang harus dijalankan :

```
$ git init
$ git add .
$ git commit -m "Commit message"
```

Setelahnya, pengembang dapat membuat perangkat lunak Heroku. Setiap perangkat lunak Heroku dibuat, maka `git remote` secara otomatis juga dibuat. Pengembang dapat memeriksanya dengan mengetikkan perintah :

```
$ git remote // Untuk daftar nama remote saja
$ git remote -v // Untuk informasi yang lebih detail
```

Untuk mengubah nama remote, pengembang dapat mengetikkan perintah :

```
$ git remote rename <nama lama> <nama baru>
```

<nama lama> adalah nama remote yang ingin diganti. <nama baru> adalah nama baru untuk remote tersebut.

Untuk melakukan deploy, pengembang dapat mengetikkan perintah :

```
$ git push <nama remote> <nama branch>
```

<nama remote> adalah nama remote dari tujuan deploy. Bila pengembang tidak mengubah nama remote, nama remotenya adalah `heroku`. <nama branch> adalah nama cabang dari tujuan deploy. Heroku secara otomatis membuat satu cabang bernama `master`.

2.2.3.2 Deploy Menggunakan Docker

Untuk melakukan deploy menggunakan Docker, pengembang harus sudah memasang Docker dan telah masuk ke akun Heroku (`heroku login`). Setelah itu, pengembang harus mengikuti langkah-langkah ini :

- Masuk ke Container Registry

```
$ heroku container:login
```

- Clone source code contoh dari Alpine

```
$ git clone https://github.com/heroku/alpinehelloworld.git
```

- Membuat perangkat lunak Heroku baru

```
$ heroku create
```

- Membangun image dan melakukan deploy ke Container Registry

```
$ heroku container:push web
```

- Melepaskan image ke perangkat lunak

```
$ heroku container:release web
```

- Membuka perangkat lunak

```
$ heroku open
```

2.2.3.3 Deploy Menggunakan GitHub

Deploy dengan cara ini membuat Heroku dapat dengan otomatis melakukan deploy ke GitHub apabila build berhasil. Pengembang perlu mengaktifkan GitHub integration terlebih dahulu sebelum dapat melakukan deploy. Setelah itu, pengembang harus melakukan otentikasi dengan akun GitHub. Otentikasi ini hanya perlu dilakukan satu kali per satu akun Heroku. Setelah itu, pengembang dapat memilih repository yang ingin disambungkan dengan perangkat lunak Heroku.

Ada dua cara untuk melakukan deploy, yaitu secara manual dan secara otomatis. Untuk cara manual, pengembang melakukan deploy dari GitHub. Untuk cara otomatis, pengembang harus mengaktifkan "Automatic deploys from GitHub".

2.2.3.4 Deploy Langsung di situs Heroku

Tombol "Deploy to Heroku" memungkinkan pengguna untuk melakukan deploy perangkat lunak tanpa meninggalkan situs Heroku dan hampir tidak memerlukan konfigurasi. Penggunaan tombol ini ideal untuk pelanggan, pemelihara proyek yang open-source.

Sebelum dapat melakukan deploy dengan cara ini, perangkat lunak harus memiliki dokumen `app.json` yang sah di direktori root, dan source code perangkat lunak harus berada di repository GitHub.

`app.json` adalah dokumen berisi deskripsi perangkat lunak web. Isinya dapat berupa environment variable, add-ons, dan informasi lain yang diperlukan untuk menjalankan perangkat lunak pada Heroku. Heroku tidak mewajibkan pengembang menuliskan informasi tertentu, tapi Heroku merekomendasikan untuk setidaknya menuliskan nama perangkat lunak(`name`), deskripsi perangkat lunak (`description`), dan logo perangkat lunak (`logo`). Berikut contoh isi dari `app.json` :

```
{
  "name": "Node.js Sample",
  "description": "A barebones Node.js app using Express 4",
  "repository": "https://github.com/heroku/node-js-sample",
  "logo": "https://node-js-sample.herokuapp.com/node.png",
  "keywords": ["node", "express", "static"]
}
```

2.2.4 Basis Data dan Manajemen Data

Heroku menyediakan tiga layanan data untuk semua pelanggan :

- Heroku Postgres
- Heroku Redis
- Apache Kafka

Heroku juga menyediakan pilihan lain untuk pelanggan Heroku Enterprise, yaitu Heroku Connect. Selain itu, Heroku juga memungkinkan penggunaan layanan data dari pihak ketiga. Layanan data dari pihak ketiga ini tersedia sebagai add-ons.

2.2.4.1 Heroku Postgres

Heroku Postgres adalah basis data SQL yang disediakan secara langsung oleh Heroku. Heroku Postgres dapat diakses oleh bahasa apapun dengan PostgreSQL driver. Heroku secara otomatis menambahkan add-ons Heroku Postgres setiap perangkat lunak dibuat, sehingga pengembang

tidak perlu menambahkannya secara manual. Namun, pengembang dapat menambahkannya secara manual, dengan mengetikkan perintah :

```
$ heroku addons:create heroku-postgresql:<PLAN_NAME>
```

<PLAN_NAME> adalah tipe Heroku Postgres yang ingin dipakai. Heroku secara otomatis menggunakan Heroku Postgres tipe `hobby-dev`.

Heroku Postgres memiliki lima tipe Heroku Postgres :

- Hobby Tier : untuk perangkat lunak dengan toleransi gagal bekerja sampai 4 jam per bulan.
- Standard Tier : untuk perangkat lunak dengan toleransi gagal bekerja sampai 1 jam per bulan.
- Premium Tier : untuk perangkat lunak dengan toleransi gagal bekerja sampai 15 menit per bulan.
- Private Tier : untuk pengguna Heroku Enterprise, memiliki toleransi gagal bekerja sampai 15 menit per bulan.
- Shield Tier : untuk pengguna Heroku Enterprise yang menginginkan basis data yang compliance-capable, memiliki toleransi gagal bekerja sampai 15 menit per bulan.

Selain tipe Hobby, basis data memiliki fitur fork, follow, rollback, dan Disk Encryption. Hanya tipe Hobby yang gratis.

Pengembang juga dapat menambahkan versi yang ingin dipakai dengan cara menambahkan `-version` di belakang perintah tersebut, contoh :

```
$ heroku addons:create heroku-postgresql:<PLAN_NAME--version=9.5
```

Secara otomatis, Heroku menggunakan versi paling baru dari Heroku Postgres. Saat skripsi ini ditulis, versi terbaru adalah versi 10.

Setelah dipasang, Heroku akan secara otomatis menambahkan config var `DATABASE_URL` ke perangkat lunak. Apabila Heroku Postgres yang dipakai ada lebih dari satu, nama config var akan menjadi `HEROKU_POSTGRESQL_<COLOR>_URL` dengan <COLOR> adalah nama warna yang dihasilkan secara acak. Contoh : `HEROKU_POSTGRESQL_<BLUE>_URL`.

Apabila pengembang menggunakan lebih dari satu basis data, pengembang dapat mengatur basis data utama. Basis data utama dapat diatur dengan perintah :

```
$ heroku pg:promote <database_url>
```

<PLAN_NAME> adalah tipe Heroku Redis yang ingin dipakai. Heroku Redis memiliki dua tipe : Hobby Dev dan Premium. Hobby Dev gratis, sedangkan Premium berbayar. Perbedaannya terletak pada jumlah memori dan batas koneksi yang dapat dibuat.

Heroku Redis memiliki kelebihan sebagai berikut :

- Memiliki analisa performa yang dapat membantu menemukan masalah basis data dengan mudah
- Heroku dapat diskala sesuai kebutuhan memori dan koneksi.

2.2.4.2 Heroku Redis

Heroku Redis adalah basis data berbasis key-value store yang bersifat in-memory. Heroku dijalankan oleh Heroku dan dikelola sebagai add-on Heroku Redis dapat diakses oleh bahasa apapun dengan Redis driver. Cara memasang add-on Heroku Redis :

```
$ heroku addons:create heroku-redis: <PLAN_NAME>
```

2.2.4.3 Apache Kafka

Apache Kafka adalah salah satu add-on di Heroku yang disediakan oleh Kafka yang berintegrasi penuh dengan Heroku. Apache Kafka dideskripsikan Kafka dideskripsikan oleh Heroku sebagai add-on yang memungkinkan pengembang mendistribusikan perangkat lunak yang dapat menangani jutaan event dan miliaran transaksi. Kafka didesain untuk memindahkan ephemeral data yang sangat besar dengan reliabilitas yang tinggi dan toleran akan kerusakan.

Pengembang harus memasang Python 2.7, node 8.x, .NET Framework, dan Visual C++ Build Tools terlebih dahulu sebelum memasang Apache Kafka. Setelah itu, pengembang mengetikkan perintah :

```
$ heroku plugins:install heroku-kafka
```

2.3 IMAP

IMAP (Internet Message Access Protocol) adalah metode untuk mengakses pesan elektronik yang disimpan di sebuah mail server.

2.3.1 PHP IMAP

Ekstensi ini dapat digunakan apabila c-client library sudah terpasang. Library ini dapat ditemukan di https://www.washington.edu/imap/. Dokumen IMAP tidak boleh diletakkan langsung ke dalam direktori system, karena dapat memicu konflik. Sebaiknya membuat direktori baru di dalam direktori system, lalu memasukkan dokumen IMAP ke dalamnya. Contoh : `/usr/local/imap-2000b`. Di dalam direktori baru tamabahkan direktori lagi bernama `lib/` dan `include/`. Semua dokumen dengan ekstensi `.c` dimasukkan ke direktori `lib/`. Saat IMAP dikompilasi, dokumen bernama `c-client.a` akan terbentuk. Dokumen tersebut juga diletakkan di direktori `lib/`.

Setelah itu, kompilasi PHP dengan –with-imap[=DIR]. DIR disini adalah tempat c-client. Contoh : `with-imap=/usr/local/imap-2000b`. Pengguna sistem operasi Windows mungkin harus mengaktifkan `php_imap.dll`.

IMAP tidak didukung pada sistem operasi Windows yang versinya lebih lama dari Windows 2000. Hal ini karena IMAP menggunakan fungsi enkripsi agar koneksi lewat SSL ke mail server aktif.

Di dalam sistem operasi Ubuntu, pemasangan PHP IMAP bisa dilakukan dengan mudah.

```
// Pasang libc-client-dev
$ sudo apt-get install libc-client-dev

// Pasang PHP<versi> imap:
// sudo apt-get install php<versi>-imap
```

```
// Contoh :
sudo apt-get install php5-imap
```

2.4 Line@

Line@ adalah layanan oleh LINE yang didesain khusus untuk bisnis atau organisasi. Line@ menyediakan berbagai fitur untuk mempromosikan suatu perusahaan, merek, atau produk dalam cara yang baru dan dengan jangkauan yang luas. Salah satu fitur tersebut adalah fitur "Message Broadcasts". Fitur ini memungkinkan pengguna mengirimkan pesan melalui perangkat lunak mobile LINE@ atau melalui perangkat lunak komputer LINE@ Manager dan menyebarkannya ke pelanggan dan fans yang telah menjadikan akun pengguna sebagai teman. LINE@ menawarkan beberapa fitur bawaan yang bisa digunakan di pesan, seperti kupon dan survei. Fitur "1-on-1 chat" memungkinkan pengguna membalas secara langsung pesan yang dikirimkan pelanggan dan fans yang menjadikan pengguna teman. Fitur "Timeline Posts" memungkinkan pengguna mengirimkan postingan di linimasa. Postingan tersebut bisa diberi "like" atau dikomentari sehingga dapat memaksimalkan potensi linimasi sebagai media pemasaran.[3]

[3]https://at.line.me/en/

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

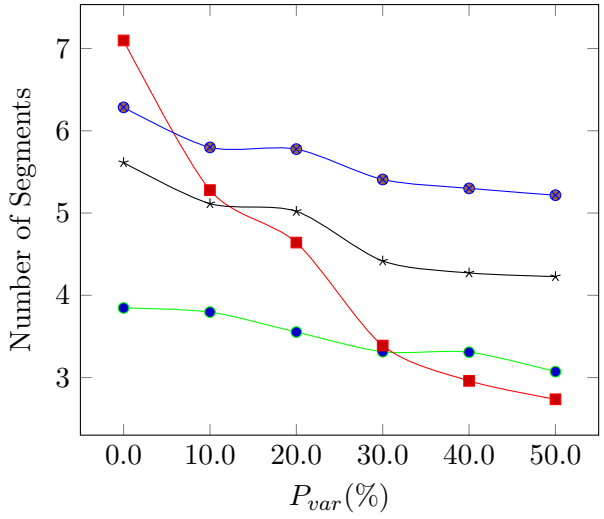
Listing A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

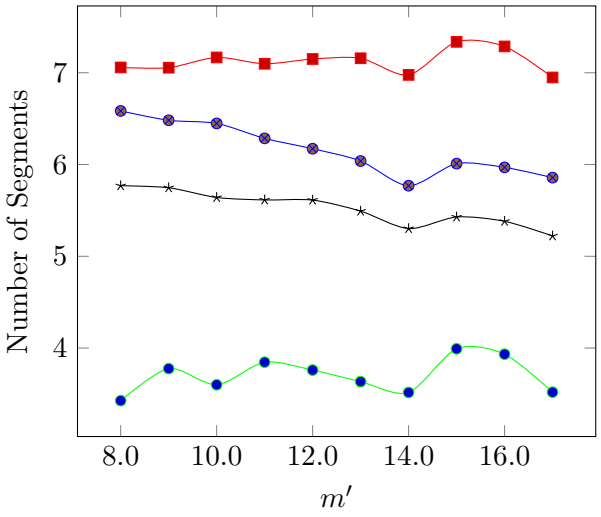
LAMPIRAN B

HASIL EKSPERIMEN

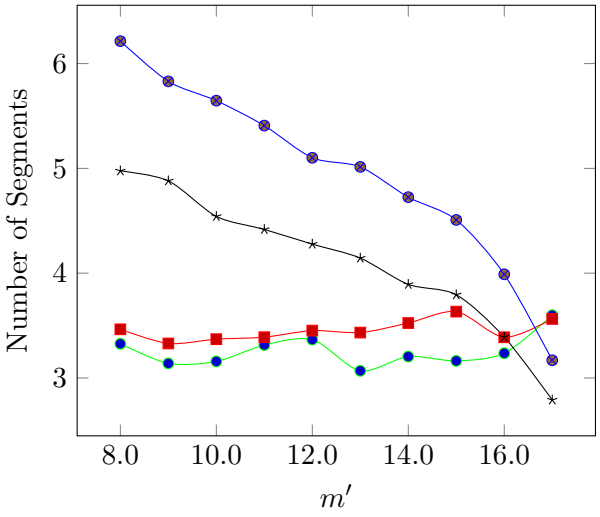
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



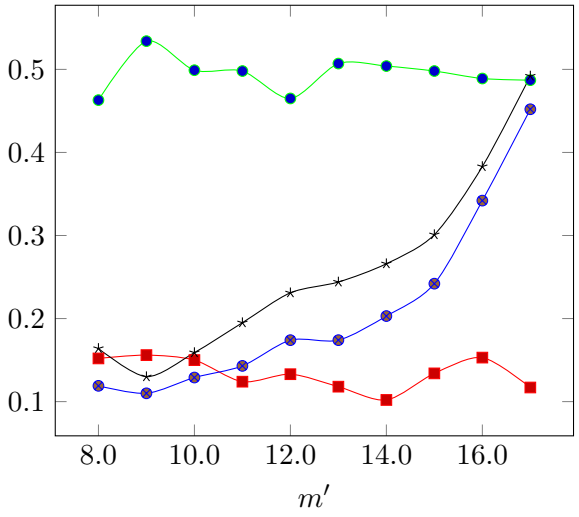
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4