BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Dalam pelaksanaan kerja magang di PT. Bank Index Selindo, posisi yang diambil yaitu *Programmer* atau *Application Developer*, yang berada di dalam Divisi IT dan Departemen Manajemen Bisnis. Lya Santi Rahayu, yang bekerja sebagai kepala bagian Pengembangan Strategis IT, menjadi pembimbing lapangan di kerja magang ini. Selain itu, pekerja Pengembang Aplikasi tetap di perusahaan juga mendampingi dalam proses kerja, yaitu Anju Arjanto pada bulan Agustus dan Fawwaz Saputra dari bulan September hingga November, yang bertugas memberikan pekerjaan magang untuk dilakukan dan diisi ke dalam *daily task*.

Mayoritas dari pekerjaan magang ini dikerjakan secara individu, namun terkadang pendamping kerja juga dipanggil untuk membantu ketika ada bagian dari tugas yang terasa sulit ataupun untuk melakukan *review* program.

3.2 Tugas yang Dilakukan

3.2.1 Tugas dan Metodologi

Tugas dari kerja magang ini adalah untuk membuat sebuah aplikasi sistem notifikasi/*alert* yang mengirim pesan ke akun Telegram user menggunakan bot, baik secara manual maupun saat ada email yang masuk. *Website* ini juga mampu mengirim pesan notifikasi ke email user yang sudah "*subscribe*" terhadap layanan bot Telegram. Website ini dibuat dengan bahasa pemrograman PHP dan *framework* Laravel. *Website* ini berfokus kepada sisi *back end*, dengan *front end*-nya diambil dari *template* Bootstrap di internet, khususnya dari Start Bootstrap.

Berikut ini merupakan *tools* yang digunakan untuk mendukung proyek ini:

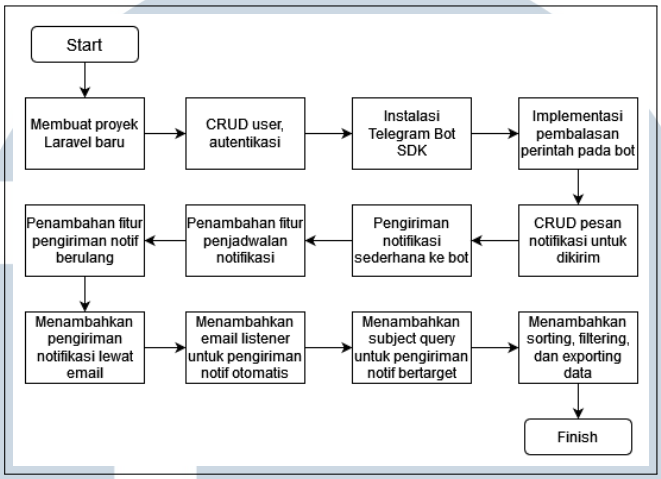
1. *Web browser* Firefox untuk menampilkan dan menguji coba *website* yang dibuat di localhost.
2. IDE Visual Studio Code sebagai tempat untuk menulis kode pemrograman.
3. XAMPP untuk menyediakan *tools* yang diperlukan untuk membuat *website* berbasis Laravel, yaitu PHP, Apache (sebagai metode *hosting* di localhost) dan MySQL (sebagai *database* utama).

4. *Library* PHP Composer untuk menginstal Laravel dan *library* lain yang digunakan di proyek seperti Telegram Bot SDK, Laravel Mailbox dan Laravel Excel.
5. Akun Telegram dan Telegram Bot sebagai salah satu metode untuk menerima notifikasi yang terkirim.
6. Ngrok untuk "mengekspos" host aplikasi sehingga dapat diakses secara publik serta menyediakan *webhook* supaya *website* dan Telegram Bot dapat terhubung untuk memberi respons sesuai perintah yang dikirim user.
7. Akun Mailtrap dan Mailgun untuk menyediakan inbox *testing* sehingga aplikasi web dapat mengirim email, serta melakukan *handling* terhadap email yang masuk di domain tertentu.

Berikut ini merupakan uraian singkat tentang tugas-tugas yang dikerjakan di proyek magang ini, khususnya proyek sistem *alert* [6]:

1. Membuat proyek Laravel baru untuk membuat *website* sistem *alert*.
2. Membuat menu autentikasi (login, *register*, *reset password*) dan CRUD (*create, read, update, delete*) dari user untuk admin.
3. Membuat akun Telegram, dan bot Telegram yang digunakan untuk mendapatkan notifikasi.
4. Menginstalasi *library* Telegram Bot SDK ke proyek Laravel untuk menghubungkan aplikasi web dengan bot Telegram yang dibuat.
5. Melakukan CRUD pesan yang akan dikirim ke notifikasi.
6. Membuat sistem pengiriman notifikasi ke Telegram dan Email (menggunakan *service* Mailgun) beserta dengan pengaturan penjadwalan dan pengiriman berulang (*recurring*) menggunakan Laravel Queue.
7. Membuat sistem dimana saat user mendapatkan email, maka akan mengirim notifikasi ke bot Telegram, menggunakan *listener* Mailgun. Selain itu, ketika subjek email memiliki kesamaan dengan *string* pada daftar *query*, maka notifikasi akan terkirim ke tujuan yang tertera pada daftar.
8. Membuat sistem *sorting* dan *filtering* untuk tabel *subscriber* dan *notification history*.

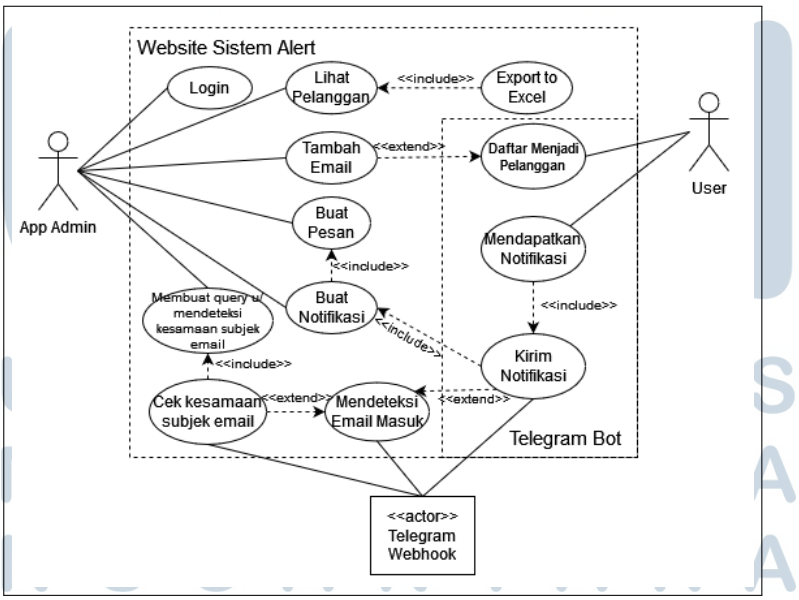
9. Melakukan *black box testing* sederhana dengan rekan-rekan kerja untuk menguji coba efektivitas aplikasi web dan bot Telegram.



Gambar 3.1. Alur bagan metodologi pembuatan aplikasi sistem *alert* Telegram

3.2.2 Deskripsi Aplikasi

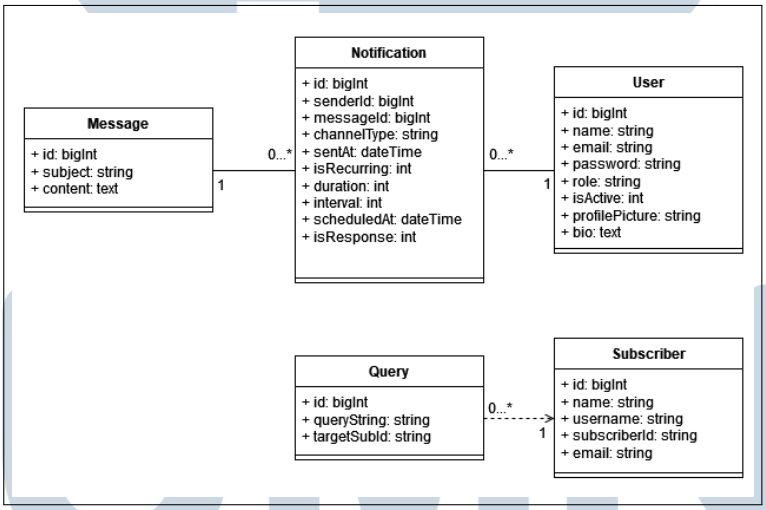
Website ini dibuat menggunakan pendekatan *object-oriented*. Berikut ini merupakan struktur aplikasi dalam bentuk diagram UML. Diagram UML yang dipakai yaitu *use case diagram, class diagram*, dan *activity diagram*. [7]



Gambar 3.2. *Use Case* Diagram untuk aplikasi sistem *alert* di Pt. Bank Index Selindo

Diagram *use case* adalah suatu diagram UML yang menggambarkan peran aktor dan interaksinya dengan aktor lain atau sistem. [8] Berdasarkan diagram

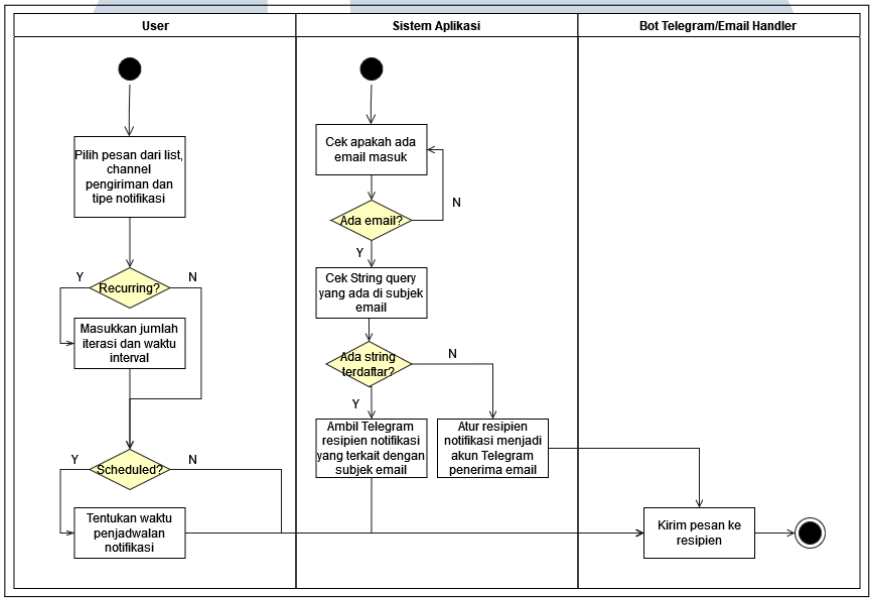
use case tersebut, ada 3 aktor yang memiliki peran penting pada *website* ini, yaitu admin aplikasi, webhook Telegram, dan user/*subscriber*. Admin aplikasi memiliki peran yang paling penting di aplikasi ini karena memiliki *use case* terbanyak di dalam diagram tersebut. Admin dapat melakukan login, melihat daftar *subscriber* serta menambahkan email atau mengekspor menjadi file Excel, membuat pesan, membuat notifikasi (yang bergantungan pada pembuatan pesan), dan membuat *string query* dalam pemeriksaan subjek email serta *subscriber* targetnya. Untuk Telegram Webhook, peran utamanya adalah mengirim notifikasi. Notifikasi dapat terkirim setelah admin membuat notifikasi, namun juga setelah mendeteksi adanya email masuk, yang mengirim isi email ke Telegram *subscriber*. Webhook juga dapat melakukan pengecekan subjek email berdasarkan *string*/target yang didefinisikan oleh admin untuk mengirim notifikasi ke user yang ditentukan. Peran user Telegram ada dua, yaitu berlangganan ke layanan notifikasi dan mendapatkan notifikasi, semuanya menggunakan bot Telegram.



Gambar 3.3. *Class Diagram* untuk aplikasi sistem *alert* di Pt. Bank Index Selindo

Class diagram menggambarkan hubungan antar entitas dalam suatu sistem. [9] Ada lima entitas utama yang dipakai di aplikasi ini, yaitu User, Subscriber, Message, Notification, dan Query. Di sini, beda User dan Subscriber adalah User merupakan pengguna yang login pada aplikasi ini untuk mengirim notifikasi, sedangkan Subscriber merupakan pengguna yang berlangganan pada bot Telegram untuk mendapatkan notifikasi. Ada dua relasi yang terbentuk antar *class*, yaitu asosiasi antara Notification dengan User dan Message, serta dependensi antara Query dengan Subscriber. Asosiasi antara Notification dengan User dan Message adalah one to many, dimana satu notifikasi memiliki satu User pengirim dan satu

pesan isi, sedangkan satu user bisa mengirim beberapa notifikasi dan satu pesan dapat dikirim menjadi beberapa notifikasi berbeda. Sedangkan untuk Query dan Subscriber, Query membutuhkan *class* Subscriber untuk dapat mengidentifikasi tujuan pesan Telegram yang terkirim ketika ada email masuk dengan subjek sesuai dengan *string* yang tersimpan, namun *class* Subscriber dapat berdiri dengan sendirinya.

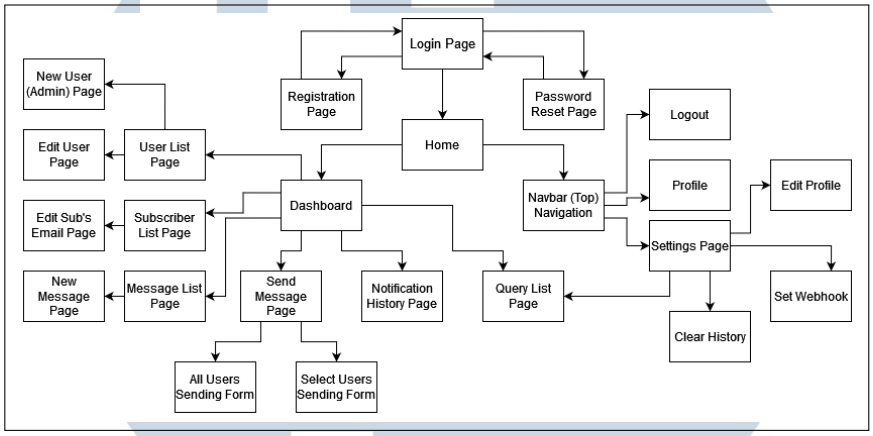


Gambar 3.4. *Activity Diagram* pengiriman notifikasi untuk aplikasi sistem *alert* di Pt. Bank Index Selindo

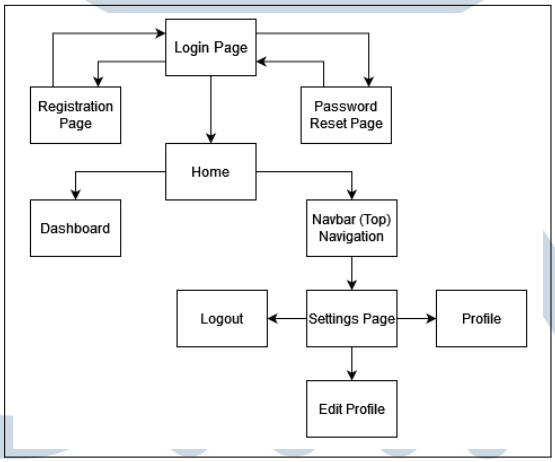
Activity diagram merupakan diagram untuk menunjukkan alur aktivitas suatu sistem, tanpa mengetahui kode atau tampilan layar. [10] Bagan tersebut merupakan *Activity Diagram* untuk proses pengiriman notifikasi. Sumber dari pengiriman notifikasi dapat berasal dari input user di halaman Kirim Notifikasi atau secara otomatis oleh sistem aplikasi. Untuk pengiriman manual, user dapat memilih pesan, *channel* pengiriman, dan tipe notifikasi. Kemudian, user bisa mengatur apakah notifikasi akan dikirim berulang atau tidak. Jika ya, maka user juga menginput jumlah pesan akan dikirim dan jarak antara pengiriman pesan, dalam menit. User juga bisa mengatur penjadwalan pesan. Jika pesan akan terjadwalkan, user menginput waktu dimana pesan nantinya akan dikirim. Setelah informasi selesai, notifikasi akan dikirim melalui Telegram atau email. Untuk pengiriman otomatis, sistem mengecek apabila ada email yang masuk di domain yang diberikan. Ketika ada, sistem akan mengecek subjek email, dan jika ada *string* di *database* yang tertera di subjek email, maka sistem akan mengirim notifikasi Telegram dengan isi email kepada akun tujuan yang berhubungan dengan

string tersebut. Namun, jika tidak, maka sistem akan mengirim notifikasi ke akun Telegram yang berhubungan dengan email tersebut, jika terdaftar.

Website yang dibuat memiliki 24 total halaman dengan 19 halaman primer dan 5 halaman sekunder. [11] Berikut ini merupakan bagan dari daftar halaman yang ada pada *website* ini dalam bentuk *sitemap*:



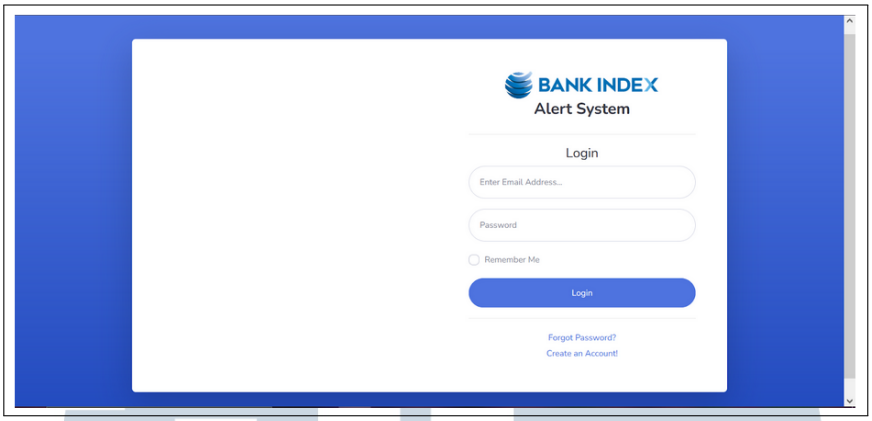
Gambar 3.5. *Sitemap* aplikasi sistem *alert* pada PT. Bank Index Selindo untuk admin



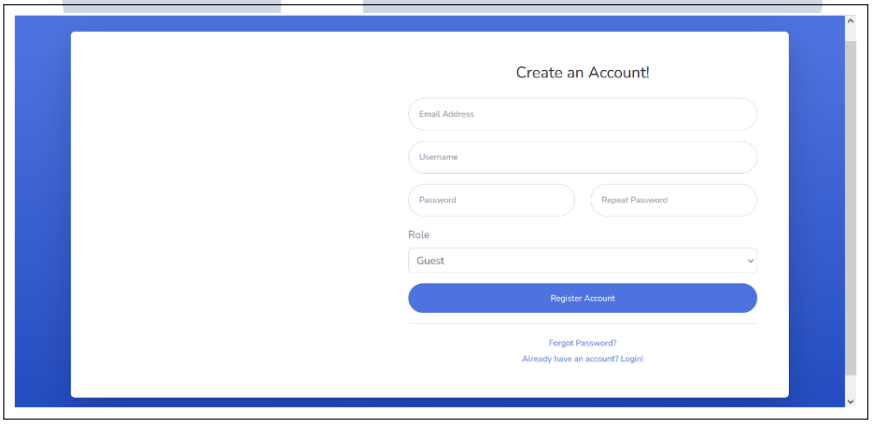
Gambar 3.6. *Sitemap* aplikasi sistem *alert* pada PT. Bank Index Selindo untuk *guest*

Sitemap merupakan diagram yang memperlihatkan arsitektur serta relasi antar halaman pada aplikasi. [12] [13] *Sitemap* ini menunjukkan relasi antara halaman yang dapat diakses melalui klik di *website* ini. Dua halaman utama pada *website* ini adalah halaman autentikasi (login/registrasi/*reset password*) dan halaman menu (*dashboard*/navbar).

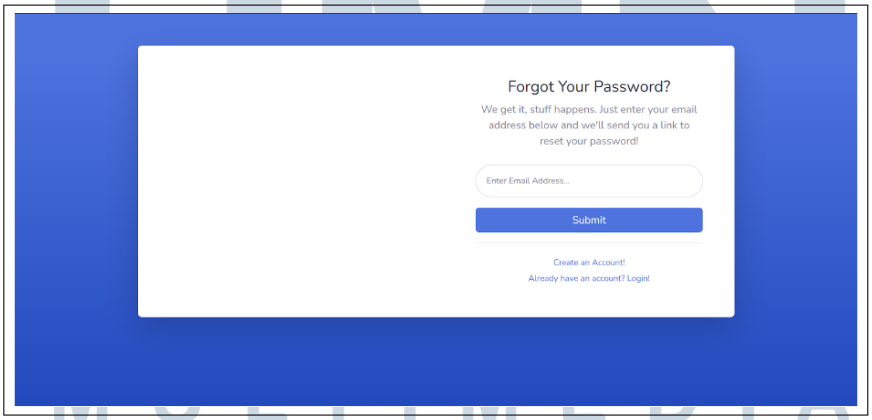
Berikut ini merupakan tampilan masing-masing halaman pada *website* yang dibuat, beserta beberapa potongan kode penting yang berhubungan.



Gambar 3.7. Tampilan halaman login pada aplikasi sistem *alert*



Gambar 3.8. Tampilan halaman registrasi pada aplikasi sistem *alert*

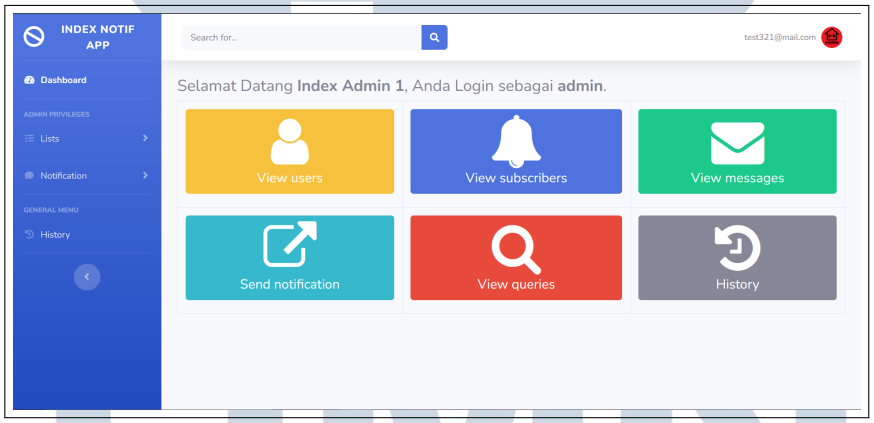


Gambar 3.9. Tampilan halaman *reset password* pada aplikasi sistem *alert*

Tampilan login pada aplikasi ini merupakan tampilan yang sederhana, dimana user diminta untuk memasukkan email dan *password*. [14] Ada opsi untuk registrasi akun, dimana user dapat menentukan email, *username*, *password*, dan peran (*guest*/admin) untuk akunnya di *website* ini, dimana hanya admin yang

dapat mengirim pesan. Sehingga, hanya PIC tertentu di Bank Index yang dapat menggunakan *website* ini. Ada juga opsi untuk melakukan *reset password* ketika lupa *password*.

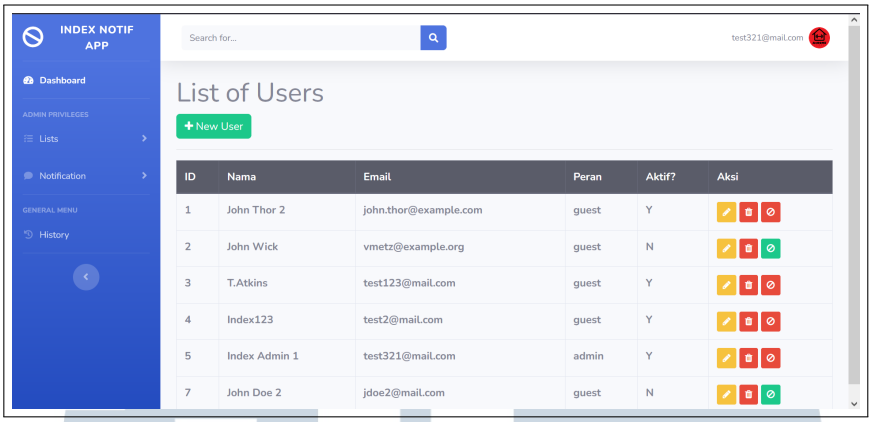
Mekanisme login dilakukan menggunakan kelas *facade* bawaan Laravel yaitu Auth. Untuk melakukan login, digunakan command *Auth:attempt* di proses login dalam LoginController yang berfungsi untuk mengecek kesamaan email dan *password* dengan data yang telah disimpan di database. Khususnya, sistem akan melakukan *hashing* terhadap input *password* dan mengecek dengan *password hash* yang berhubungan dengan alamat email yang telah diinput dan disimpan dalam *database*. Jika sukses, maka sistem akan mengambil user yang berhubungan dengan email tersebut menjadi objek user dalam *facade* Auth, dan user di sini dapat diakses di *controller* lainnya seperti HomeController untuk mengecek apakah user diblokir atau tidak. Namun, jika gagal, maka sistem me-*return* error dan user diarahkan untuk mencoba lagi.



Gambar 3.10. Tampilan halaman utama pada aplikasi sistem *alert*

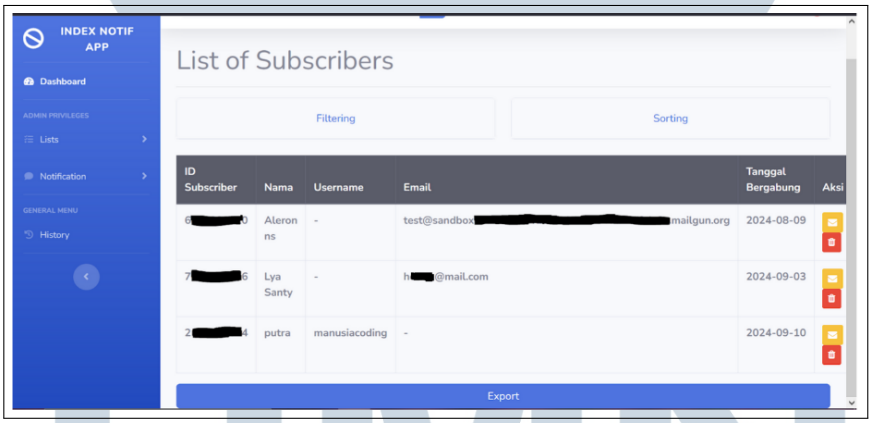
Setelah user login sebagai admin, akan disuguhkan tampilan *dashboard* seperti pada Gambar 3.10. Ada enam menu utama yang bisa diakses dari *dashboard*, dengan menu lainnya, seperti *settings*, juga dapat diakses dari navbar.

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.11. Tampilan halaman daftar user pada aplikasi sistem *alert*

Di menu *users*, admin dapat melihat daftar user yang terdaftar pada sistem ini, serta melakukan edit, penghapusan, atau blokir pada user tertentu. User yang terblokir tidak dapat mengakses fitur-fitur pada *website* ini.



Gambar 3.12. Tampilan halaman daftar *subscriber* pada aplikasi sistem *alert*

```
public function handle() {
    $enteredToken = $this->argument('sub_token');
    if($enteredToken == env('SUBSCRIBER_TOKEN')) {
      $fallbackUsername = $this->getUpdate()->getMessage()->from->
    username;
      $fallbackFirstname = $this->getUpdate()->getMessage()->from
    ->first_name;
      $fallbackLastname = $this->getUpdate()->getMessage()->from->
    last_name;
      $fallbackId = $this->getUpdate()->getMessage()->from->id;
      $username = $this->argument('username', $fallbackUsername);
      $fname = $this->argument('username', $fallbackFirstname);
      $lname = $this->argument('username', $fallbackLastname);
      $name = "{$fname} {$lname}";
```

```
12          $id = $this->argument('username', $fallbackId);
13          $this->replyWithMessage([
14              'text' => "Thanks for subscribing, {$username} aka {$name}
        (added to the database)",
15          ]);
16          $this->replyWithMessage([
17              'text' => "Your user ID: {$id}",
18          ]);
19          $sub = Subscriber::create([
20              'subscriber_id' => $id,
21              'username' => $username,
22              'name' => $name
23          ]);
24      } else {
25          $this->replyWithMessage([
26              'text' => "Sorry, the access token is incorrect! Please
        try again or contact IT",
27          ]);
28      }
29  }
```

Kode 3.1: Kode membuat subscriber baru di SubscribeCommand.php

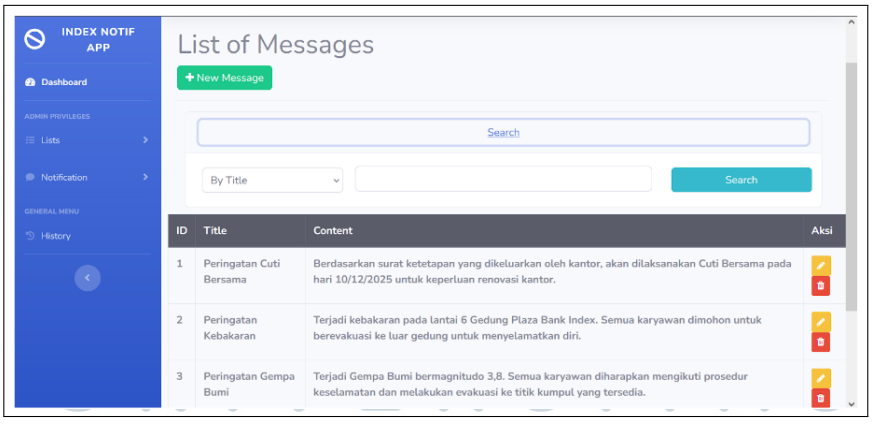
Di menu *subscribers*, admin dapat melihat dan mengelola daftar *subscriber*. User dapat melakukan *sorting subscriber* berdasarkan nama dan tanggal bergabung, serta mengekspor daftar *subscriber* menjadi *file* Excel. Secara *default*, *subscriber* memiliki ID Telegram yang digunakan untuk forwarding notifikasi, namun admin juga bisa menambahkan email untuk setiap *subscriber* sehingga notifikasi yang dikirim melalui email akan masuk ke inbox email masing-masing.

Kode 3.1 merupakan sebagian dari proses pembuatan *subscriber* baru melalui bot Telegram. Dalam proses pendaftaran *subscriber*, digunakan *webhook* untuk menghubungkan *website* dengan bot Telegram, menggunakan ID Bot Telegram yang disimpan di *file* .env dan opsinya diatur melalui WebhookController. Saat user mendapatkan link bot Telegram, secara otomatis sistem akan mengirim perintah "/start" dan muncul balasan dari bot untuk mengetik "/*subscribe*" diikuti dengan token *subscriber* yang disimpan di *file* .env. Jika token sesuai, maka program akan mengambil informasi dari user yaitu nama lengkap, *username*, dan ID user untuk disimpan di *database*. Pertama, informasi diambil dari pesan yang dikirim user, lalu disimpan di variabel *username, fname*, dan *lname*. Setelah itu, Telegram Bot mengirim pesan kepada user sebagai konfirmasi dan *Subscriber* baru di *database* dibuat dengan informasi yang didapatkan. [15]

Tabel 3.1. Penjelasan Kode 3.1 tentang pembuatan *subscriber* baru

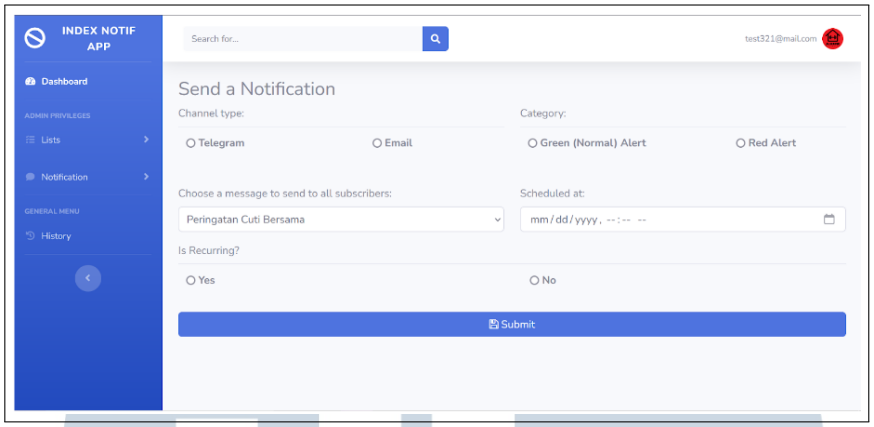
Baris	Deskripsi
3	Mendapatkan *subscriber token* dari input user
4	Memvalidasi *subscriber token* yang di-input, jika sesuai dengan token yang ada di *file* .env, maka jalankan baris 18-37
5-8	Mendapatkan *username*, nama, dan ID Telegram user dan menyimpannya ke variabel *fallback* (sementara)
9-13	Memindahkan nilai di setiap variabel *fallback* ke variabel tetap
14-19	Mengirim pesan ke Telegram user bahwa proses berlangganan berhasil dan data tersimpan di *database*
20-24	Membuat item *Subscriber* baru di *database* dengan informasi yang didapatkan

Mengekspor daftar *subscriber* menjadi file Excel dilakukan menggunakan *library* Laravel-Excel dari maatwebsite. Untuk dapat mengekspor, pertama dibuat sebuah *file* Export untuk model *Subscriber* yang melakukan *return* terhadap semua data, lalu buat sebuah fungsi *export* di *controller* yang memerintahkan untuk men-*download file* Excel dengan fungsi Excel::download(new SubsExport, 'subs.xlsx'). [16]

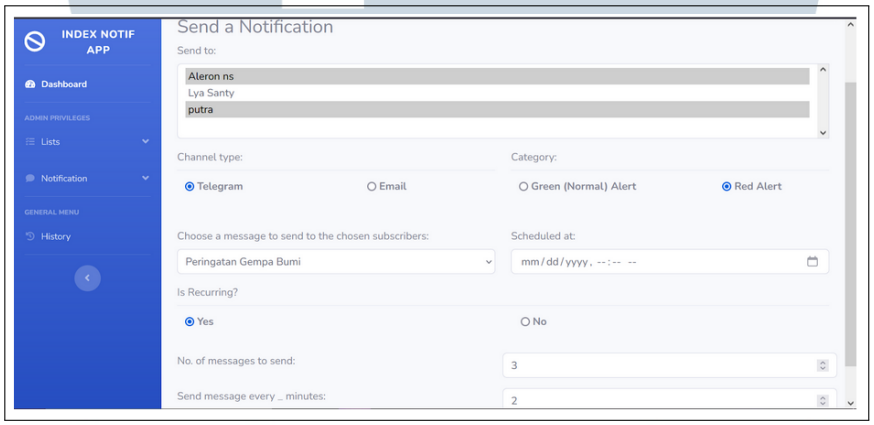


Gambar 3.13. Tampilan halaman daftar pesan pada aplikasi sistem *alert*

Di menu *messages*, admin dapat melihat daftar pesan dan membuat pesan baru. Admin juga dapat melakukan *search* untuk mencari pesan yang judul atau isinya mengandung *string* tertentu yang diketik di *search box*.



Gambar 3.14. Tampilan halaman pengiriman notifikasi untuk semua *subscriber* pada aplikasi sistem *alert*



Gambar 3.15. Tampilan halaman pengiriman notifikasi untuk *subscriber* tertentu dengan opsi *recurring* diaktifkan

```
public function newRecord(Request $req, $scheduled, $usr) {
  if($req->interval <= 1 || $req->interval == null || $req->recur
   == '0') {
    $iterations = 1;
  } else {
    $iterations = $req->interval;
  }
  $time = $scheduled;
  for($i = 0; $i < $iterations; $i++) {
   $notif = Notification::create([
   'sender_id' => Auth::user()->id,
   'channel_type' => $req->channel,
   'message_id' => $req->message,
   'category' => $req->category,
   'sent_at' => Carbon::now()->toDateTimeString(),
   'is_recurring' => $req->recur,
```

```
    'duration' => $req->duration,
    'interval' => $req->interval,
    'scheduled_at' => $time,
    'users_sent_to' => $usr,
  ]);
  if($req->duration != null) {
    if($scheduled == null) { $time = $time ?: Carbon::now(); }
      if($time != null) {
        $time->modify("+{$req->duration} minutes");
       }
    }
  }
}
```

Kode 3.2: Kode membuat notifikasi baru serta menyimpannya di database di MessageController.php

```
public function handle() {
  $not = Notification::with('message')->whereBetween('sent_at', ['
   2024-01-01 01:00:00', now()->startOfMinute()])->update(['
   is_response' => 0]);
  $notifs = Notification::with('message')->where('scheduled_at', '
   >=', Carbon::now()->toDateTimeString())->whereNull('is_response
   ')->get();
  foreach($notifs as $n) {
    $messtitle = $n->message->subject;
    $messcontent = $n->message->content;
    $messtype = $n->channel_type;
    $messsa = Carbon::parse($n->scheduled_at);
    $subs = $n->users_sent_to;
    SendScheduledNotificationJob::dispatch($messtitle,
   $messcontent, $messtype, $subs)->onQueue('default')->delay(
   $messsa);
  }
}
```

Kode 3.3: Kode pengiriman pesan yang dijadwalkan di SendScheduledMessage.php

Halaman ini merupakan salah satu halaman utama pada *website* ini, dimana admin dapat melakukan pengiriman notifikasi. Ada dua pilihan untuk target pengiriman, yaitu untuk seluruh *subscriber* dan untuk beberapa *subscriber* terpilih saja.

Halaman pengiriman notifikasi memiliki beberapa pilihan untuk pembuatan notifikasi. Ada opsi untuk mengirim ke Telegram atau email, opsi pemilihan pesan yang telah dibuat di menu *messages*, opsi untuk melakukan penjadwalan serta

memilih waktu penjadwalan, dan opsi untuk mengirim pesan secara berulang serta memilih jarak pengiriman notifikasi dalam menit dan jumlah notifikasi yang akan dikirim.

Kode 3.2 adalah potongan dari MessageController.php, *controller* Laravel yang digunakan untuk mengelola pembuatan dan pengiriman pesan dan notifikasi. Fungsi newRecord yang tersedia dibuat untuk membuat notifikasi baru di *database* berdasarkan data-data tentang scheduling dan *recurring* yang diinput oleh user. Jumlah iterasi yang diperlukan untuk memasukkan notifikasi ke *database* juga diatur berdasarkan input user di argumen "interval". Ketika user mengeset *recurring* menjadi *false* atau mengisi nilai dengan *null* atau kurang dari 1, maka iterasi yang dilakukan cukup 1. Kemudian, dimulai dengan waktu yang diberi oleh user, satu notifikasi dibuat dan ditambahkan di *database* sejumlah iterasi, dengan setiap iterasi menambahkan waktu sebanyak jumlah menit yang diberikan user di argumen "*duration*". Jika pesan tidak dijadwalkan namun *recurring* diset menjadi *True*, maka variabel $time yang tadinya *null* akan diatur menjadi waktu pesan dikirim, lalu proses penambahan waktu dilakukan untuk notifikasi kedua dan seterusnya.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

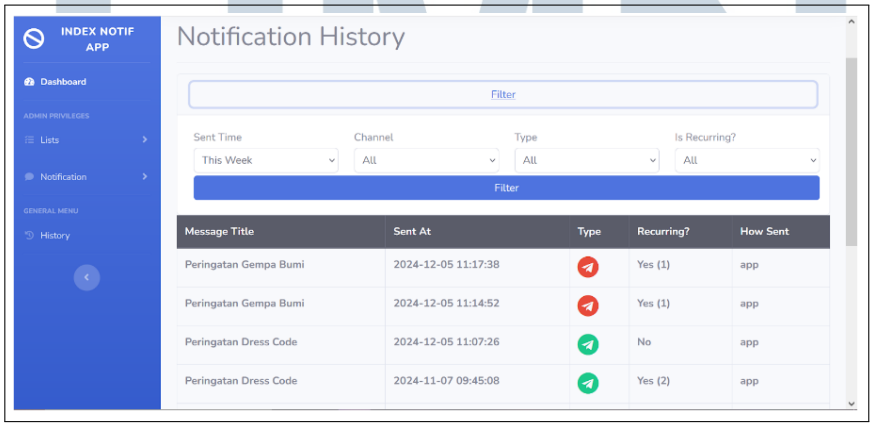
Tabel 3.2. Penjelasan Kode 3.2 tentang pembuatan notifikasi yang dijadwalkan

Baris	Deskripsi
2-6	Mengecek nilai input di *field recurring* atau interval (*messages to send*). Jika nilai interval kurang dari atau sama dengan 1 atau dikosongkan, ataupun user mengklik "No" pada opsi "*Is Recurring*", maka atur nilai terasi menjadi 1, dan jika tidak, atur nilai iterasi menjadi nilai *field* interval.
7	Membuat variabel $time yang menyimpan nilai waktu pesan akan terjadwalkan ($scheduled). Jika pesan tidak dijadwalkan, maka nilai variabel ini *null*.
8-28	Mengulang sebanyak iterasi untuk membuat notifikasi baru di *database*, dengan setiap nilai variabel $time bertambah menit sesuai dengan nilai *duration* (*send every _ minutes*).
9-20	Membuat objek notifikasi baru di *database*
21-26	Jika nilai input $duration tidak *null*, maka lakukan penambahan waktu sesuai iterasi dan nilai $duration yang diberikan user
22	Jika nilai variabel $time awal adalah *null* (tidak dijadwalkan) tetapi pesan *recurring*, maka atur nilai variabel menjadi waktu sekarang
23-24	Jika nilai variabel $time tidak *null* (diatur di baris 158 untuk *scheduled* dan 174 untuk non-*scheduled*) maka ubah nilai $time dengan menambahkan menit sesuai input user di *field duration*

Untuk dapat mengirim notifikasi yang telah dijadwalkan di *database*, digunakan Job dan Command yang dijalankan ketika perintah "queue:listen" dijalankan di *database*. [17] Kode 3.3 merupakan potongan dari SendScheduledMessage.php, perintah untuk mengirim notifikasi yang telah dijadwalkan. Pertama, status "isResponse" semua notifikasi di *database* kecuali yang akan dimasukkan ke queue diubah menjadi 0. Kemudian, semua notifikasi di *database* di *queue*, yang belum direspons oleh user, dimasukkan ke sebuah *Collection*. Kemudian, untuk setiap notifikasi di *Collection*, akan dikirim menggunakan perintah dispatch() ketika waktu sesuai dengan waktu pada atribut scheduled_at yang tertera di objek notifikasi, menggunakan delay() untuk menentukan waktu pengiriman notifikasi.

Tabel 3.3. Penjelasan Kode 3.3 tentang pengiriman notifikasi yang terjadwalkan

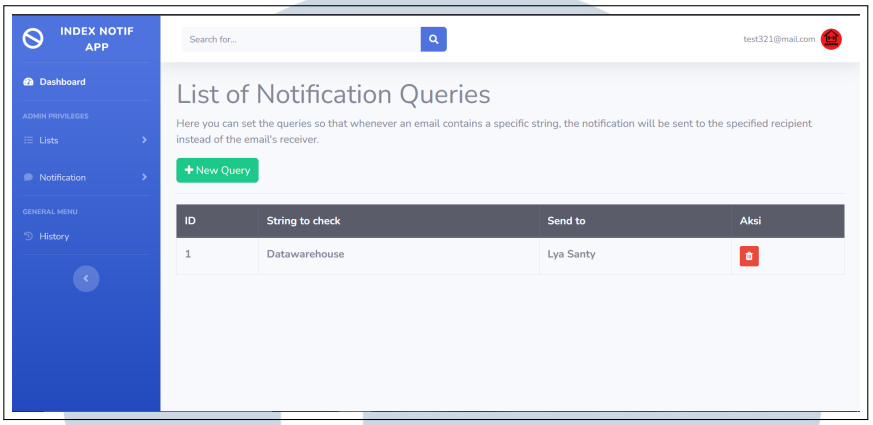
Baris	Deskripsi
2-3	Mendapatkan daftar notifikasi dengan isi pesan (menggunakan Eloquent *relationships*)
2	Mengubah nilai kolom is_response dari semua notifikasi yang dikirim sebelum waktu pengiriman notifikasi menjadi 0. Hal ini untuk memastikan bahwa hanya notifikasi yang dikirim pada waktu tersebut yang masuk ke *queue*.
3	Mendapatkan semua notifikasi dengan nilai is_response *null*, daftar notifikasi ini akan masuk ke *queue*.
4-11	Untuk setiap notifikasi di daftar notifikasi (variabel $notifs), isi kolom penting akan diambil untuk dipindahkan ke Job.
5-9	Mengambil nilai judul dan konten pesan, tipe *channel* pengiriman (Telegram/email), waktu pengiriman notifikasi, dan jika ada, daftar *subscriber* yang akan menerima notifikasi.
8	Melakukan *parsing* waktu pada nilai scheduled_at dengan Carbon sehingga mengubah nilai *String* menjadi DateTime.
10	Membuat Job baru di *queue* dengan melakukan *passing* setiap nilai variabel yang didapatkan dan mengirim notifikasi pada waktu yang tertera setelah *parsing*.



Gambar 3.16. Tampilan halaman *history* notifikasi apda aplikasi sistem *alert*

Di halaman ini, admin dapat melihat dan mengelola notifikasi yang sudah pernah terkirim sebelumnya. Notifikasi juga dapat di-*filter* berdasarkan waktu pengiriman, mode pengiriman (Telegram/email), kategori (merah/hijau), dan status

pengulangan (apakah dikirim berulang atau tidak). Halaman ini menggunakan teknik paginasi agar hanya 10 notifikasi saja yang ditampilkan per bagian.



Gambar 3.17. Tampilan halaman daftar *query* pada aplikasi sistem *alert*

```php
public function handlemailgun(Request $request) {
  $email = InboundEmail::fromMessage($request->get('body-mime'));
  $queries = EmailQuery::select('*')->get();
  $queryAvailable = false;
  $subject = $email->subject();
  $content = $email->text();
  $recip = $email->to();
  foreach($queries as $q) {
    if(str_contains($subject, $q->querystring)) {
      $queryAvailable = true;
      break;
    }
  }
  foreach($recip as $r) {
    if($queryAvailable) {
      $user = Subscriber::select('*')->where('subscriber_id', '=', $q->targetsubid)->first();
    } else {
      $user = Subscriber::select('*')->where('email', '=', $r)->first();
    }
    if($user != null) {
      //send to telegram
      $response = Telegram::sendMessage([
        'parse_mode' => 'HTML',
        'chat_id' => $user->subscriber_id,
        'text' => '<b>'.$subject.'</b>'."\n".$content
      ]);
```

```
27          }
28      }
29      return "test";
30  }
```

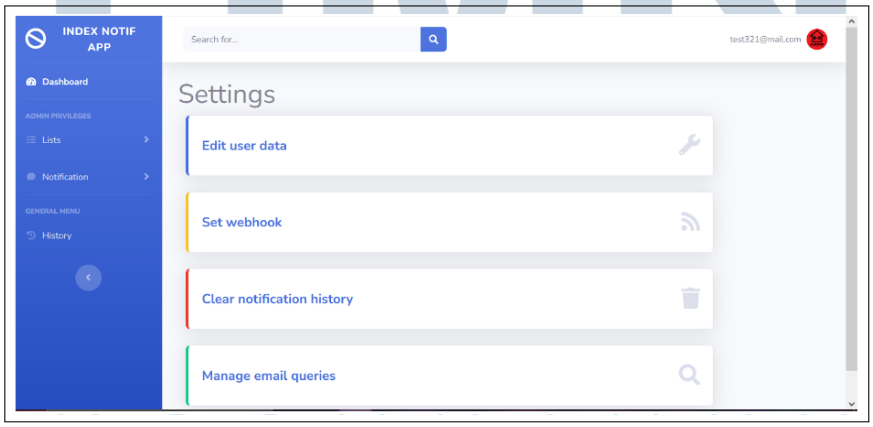
Kode 3.4: Kode deteksi subjek email dan pengiriman otomatis ke Telegram di NotifEmailController.php

Di halaman ini, admin dapat menambahkan pasangan *string query* dan target *subscriber* untuk mekanisme *email subject matching* secara otomatis. Pada Kode 3.4, isi dari email masuk diambil menggunakan fungsi fromMessage pada *class* bawaan library laravel-mailbox yaitu InboundEmail. [18] Kemudian, daftar email *query* diambil dan subjek, isi, dan tujuan email dimasukkan ke variabel berbeda. Kemudian dicek, apabila ada *string* di daftar *query* yang sesuai dengan subjek dari email. Setelah itu, untuk semua penerima email, program akan mengambil *subscriber* tertentu. Jika ada *string* yang sesuai, maka tujuannya adalah tujuan yang ada pada daftar *query* yang berhubungan dengan *string* tersebut. Jika tidak, maka kirim ke akun Telegram dari penerima email tersebut. Terakhir, pesan dikirim ke Telegram user yang telah ditentukan.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

Tabel 3.4. Penjelasan Kode 3.4 tentang pengiriman pesan Telegram setelah menerima email

Baris	Deskripsi
2	Mendapatkan *body* email (MIME) dari email yang didapatkan menggunakan *class* InboundEmail dari laravel-mailbox
3	Mendapatkan setiap pasangan string dan akun Telegram yang ada pada *database*
5-7	Menyimpan subjek email, isi email, dan penerima email ke variabel masing-masing
8-13	Untuk setiap *string query*, jika ada *string query* yang ditemukan di dalam subjek email, maka atur variabel queryAvailable menjadi *true*. Jika tidak ada *string* yang sesuai, maka atur variabel tersebut menjadi *false*.
14-28	Mengirim notifikasi ke setiap *subscriber* yang tertera pada variabel $recip
15-19	Jika nilai queryAvailable *true*, maka akun Telegram tujuan adalah akun Telegram yang sesuai dengan *string query* yang ditemukan. Jika nilai variabel tersebut *false*, maka kirim ke akun Telegram yang berhubungan dengan alamat email penerima yang diatur di menu *Subscriber*.
20-27	Jika ada user dengan akun Telegram sesuai dengan email tujuan, maka sistem akan mengirim notifikasi ke akun Telegram user tersebut dengan judul dan isi diatur menjadi subjek dan isi email.

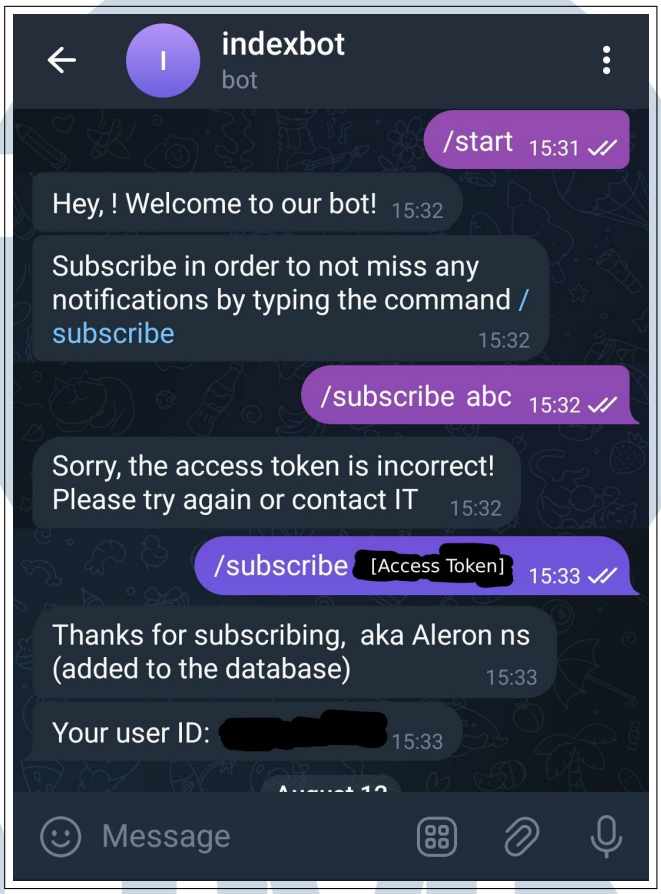


Gambar 3.18. Tampilan halaman daftar *settings* pada aplikasi sistem *alert*

Halaman *settings* dapat diakses dari navbar dengan mengklik email user atau foto profil di navbar. Di sini ada empat pengaturan, yaitu untuk mengedit profil

user, mengatur *webhook* untuk pengiriman pesan, menghapus *history* notifikasi, dan mengelola *query* email (juga bisa diakses dari *dashboard*).

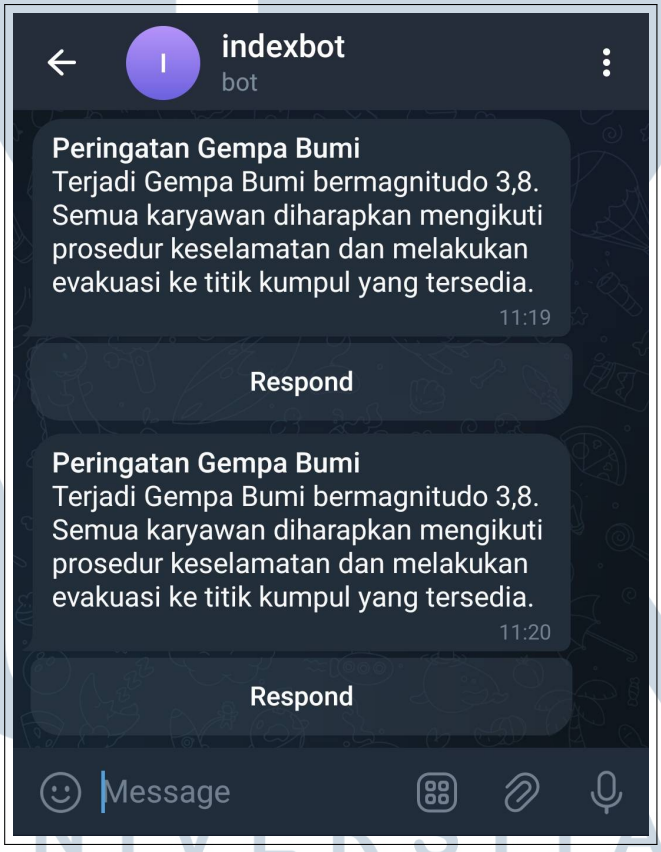
Berikut merupakan hasil dari pesan yang terkirim di chat Telegram bot user.



Gambar 3.19. Tampilan pesan ketika user melakukan *start* dan *subscribe*

Untuk dapat menerima notifikasi Telegram sistem *alert* ini, user perlu mendapatkan *link* untuk *chat* bot Telegram bernama Index Bot. Ketika membuka *chat* bot Telegram untuk pertama kalinya, maka akan terkirim secara otomatis perintah "/start" yang meminta user untuk berlanggan menggunakan perintah "/*subscribe*" diikuti dengan token akses yang benar. Jika token akses salah, maka sistem merespon dengan pesan "Akses token anda salah", sedangkan jika token akses benar, alias sama dengan token yang disimpan di *file* .env, maka sistem mengingatkan user bahwa user tersebut berhasil didaftarkan menjadi pelanggan di sistem *alert* dan dimasukkan ke *database*.

Gambar 3.20. Tampilan pesan biasa di Telegram



Gambar 3.21. Tampilan pesan yang *recurring* di Telegram

Setelah admin mengirim pesan, maka pesan tersebut muncul di *chat* Telegram bot user. Gambar 3.20 menunjukkan tampilan pesan biasa, sedangkan Gambar 3.21 menunjukkan pesan yang dikirim secara berulang atau *recurring*. Bedanya, pesan yang dikirim secara *recurring* dapat direspon oleh user, dan jika direspon dengan memencet tombol ”Respond”, maka sistem akan berhenti mengirim pesan secara berulang ke *chat* user.

3.3 Uraian Pelaksanaan Magang

Berikut ini merupakan tabel yang menjabarkan tugas-tugas penting yang dilakukan saat proses kerja magang ini.

Tabel 3.5. Waktu Pelaksanaan Magang Perusahaan

No	Deskripsi Tugas	Tanggal Mulai	Tanggal Selesai
1	*Onboarding*		
	Pengenalan struktur organisasi dan budaya kerja di kantor	05/08/2024	05/08/2024
2	Aplikasi tahap 1: Pengiriman notifikasi ke Telegram		
	CRUD user serta autentikasi	06/08/2024	07/08/2024
	Implementasi Telegram Bot SDK untuk membalas pesan *command* dari user	07/08/2024	08/08/2024
	Implementasi Telegram Bot SDK untuk mengirim pesan ke subscriber	09/08/2024	13/08/2024
	Implementasi *scheduling* untuk mengirim notifikasi berdasarkan waktu tertentu	13/08/2024	16/08/2024
3	Aplikasi tahap 2: Pengiriman notifikasi ke Email		
	Penambahan pengiriman email dengan Mailtrap	19/08/2024	03/09/2024
	Menambahkan fitur *recurring notification* (pengiriman notifikasi berulang) ke pengiriman Telegram	26/08/2024	29/08/2024
	Penambahan "*listener*" untuk mengirim notifikasi ke Telegram saat mendapatkan email baru menggunakan Mailgun	04/09/2024	17/09/2024
	Penambahan *filtering* dan *sorting*/pengurutan dan *export* Excel untuk tabel tertentu (*notification history*, *subscriber*)	18/09/2024	27/09/2024

Lanjut pada halaman berikutnya

Tabel 3.5 Waktu Pelaksanaan Magang Perusahaan (lanjutan)

No	Deskripsi Tugas	Tanggal Mulai	Tanggal Selesai
4	Pembuatan aplikasi CRUD Nest.js sebagai tugas tambahan		
	Implementasi CRUD sederhana (termasuk upload file) di proyek Nest.js + Front end di proyek React	03/10/2024	08/10/2024
	Pembuatan autentikasi user dengan Nest.js	09/10/2024	16/10/2024
	Menambahkan fitur pengiriman messages oleh user pada proyek Nest/React	17/10/2024	25/10/2024
5	Tahap-tahap akhir pengembangan aplikasi Laravel		
	Memberi perubahan perubahan minor dan bug fix pada *website* sistem *alert* seperti pengiriman notifikasi *recurring* yang tidak dijadwalkan	04/11/2024	15/11/2024
	Melakukan testing *black box* dengan rekan kerja	06/11/2024	14/11/2024

Setelah aplikasi sistem *alert* dibangun, dilakukan tahapan testing *black box*. [19] *Black box testing* adalah metode pengujian tanpa harus melihat kode aplikasi. Metode pengujian ini berfokus pada fungsionalitas fitur-fitur pada suatu sistem, sehingga hanya menguji struktur luar dari program. [20] Tahap pengujian ini berguna untuk menguji fitur dan fungsi dari aplikasi sistem *alert* yang ini. Hasil dari testing *black box* dapat dilihat pada tabel berikut:

UNIVERSITAS MULTIMEDIA NUSANTARA

Tabel 3.6. Hasil Pengujian *Black Box*

No	Deskripsi Aksi	Hasil yang Diharapkan	Hasil Pengujian	Kesimpulan
1	Login - Input email dan *password*, lalu klik tombol login	Berhasil login	Sukses	Valid
2	Buka Daftar User - Klik tombol *User List* di *dashboard*	Berhasil menampilkan daftar user	Sukses	Valid
3	Buka Daftar *Subscriber* - Klik tombol *Subscriber List* di *dashboard*	Berhasil menampilkan daftar *subscriber*	Sukses	Valid
4	Filter *Subscriber*	Berhasil melakukan *sorting subscriber*	Sukses	Valid
5	Buka Daftar Pesan - Klik tombol *Message List* di *dashboard*	Berhasil menampilkan daftar pesan	Sukses	Valid
6	Cari Pesan (search)	Berhasil menampilkan daftar pesan dengan *string* tertentu	Sukses, namun disarankan juga menambahkan *filter by title*	Valid
7	Buka *History* - Klik tombol *History* di *dashboard*	Berhasil menampilkan *history* notifikasi	Sukses	Valid

Lanjut pada halaman berikutnya

Tabel 3.6 Hasil Pengujian *Black Box* (lanjutan)

No	Deskripsi Aksi	Hasil yang Diharapkan	Hasil Pengujian	Kesimpulan
8	Filter *History*	Berhasil melakukan filtering terhadap notifikasi	Sukses, namun terjadi *error* saat klik *link* paginasi setelah difilter. Dilakukan *bug fix*, dan sukses	Valid
9	Buka Daftar *Query* - Klik tombol *Query List* di *dashboard*	Berhasil menampilkan daftar *query*	Sukses	Valid
10	Kirim notifikasi (simpel) - Buat sebuah notifikasi	Berhasil mengirim notifikasi sederhana	Sukses	Valid
11	Kirim notifikasi (*recurring*) - Buat sebuah notifikasi dengan pengaturan *recurring*, tetapi tidak dijadwalkan	Berhasil mengirim notifikasi yang berulang	Awalnya kurang sukses, lalu dilakukan *bug fix* untuk memperbaiki masalah, kemudian sukses	Valid
12	Kirim notifikasi (*scheduled* + *recurring*) - Buat sebuah notifikasi dengan pengaturan *recurring* dan dijadwalkan	Berhasil mengirim notifikasi yang berulang dan terjadwalkan	Sukses	Valid

Lanjut pada halaman berikutnya

Tabel 3.6 Hasil Pengujian *Black Box* (lanjutan)

No	Deskripsi Aksi	Hasil yang Diharapkan	Hasil Pengujian	Kesimpulan
13	Buka *settings* - Klik tombol *Settings* di navbar	Berhasil membuka halaman *settings*	Sukses	Valid
14	Logout - Klik tombol logout di navbar	Berhasil logout	Sukses	Valid

Setelah pelaksanaan tahapan testing dan melakukan *bug fix* terhadap pengiriman notifikasi *recurring* yang tidak dijadwalkan (pesan hanya akan terkirim sekali, daripada beberapa kali seperti yang telah ditentukan user) dan paginasi pasca *filtering history* notifikasi (ketika mengklik tombol *next*, akan muncul *error*), dapat dikatakan bahwa aplikasi ini sudah memiliki fungsionalitas yang lengkap dan bekerja.

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala

Dalam pengerjaan proyek ini, ada beberapa kendala yang dialami, yaitu sebagai berikut:

1. Terdapat beberapa hal-hal baru mengenai pemrograman Laravel yang belum pernah diajari di masa kuliah, seperti penggunaan Telegram Bot, *Job Queuing/Scheduling*, dan penggunaan Mailgun.
2. Kesulitan untuk memikirkan tugas baru atau fitur yang dapat ditambahkan ke *website* ketika tugas tertentu sudah selesai.

3.4.2 Solusi

Saat mengerjakan proyek ini, berikut ini adalah solusi yang diberikan terhadap kendala-kendala tersebut:

1. Membaca dokumentasi dari Laravel atau pengembang *library* dukungan serta permasalahan dan solusi atau tutorial dari situs lain seperti Stack Overflow dan YouTube tentang masalah atau topik yang terkait.
2. Bertanya kepada pendamping kerja tentang tugas baru yang dapat dikerjakan untuk menambahkan pengetahuan.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA