PENGEMBANGAN WEBSITE LICENSE MANAGEMENT DENGAN METODE TEST DRIVEN DEVELOPMENT

Disusun Oleh:

N a m a : Indra Taftazani Wisnuaji

NIM : 18523286

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2022

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PENGEMBANGAN WEBSITE LICENSE MANAGEMENT DENGAN METODE TEST DRIVEN DEVELOPMENT

TUGAS AKHIR JALUR MAGANG

Disusun Oleh:

N a m a : Indra Taftazani Wisnuaji

NIM : 18523286

Yogyakarta, 17 Oktober 2022

Pembimbing,

(Irving Vitra Paputungan, S.T., M.Sc., Ph.D.)

HALAMAN PENGESAHAN DOSEN PENGUJI

PENGEMBANGAN WEBSITE LICENSE MANAGEMENT DENGAN METODE TEST DRIVEN DEVELOPMENT

TUGAS AKHIR JALUR MAGANG

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 17 Oktober 2022

Tim Penguji

Irving Vitra Paputungan, S.T., M.Sc., Ph.D.

Anggota 1

Lizda Iswari, S.T., M.Sc.

Anggota 2

Affan Mahtarami, S.Kom., M.T

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama	:	Indra Taftazani Wisnuaji
NIM	:	18523286

Tugas akhir dengan judul:

PENGEMBANGAN WEBSITE LICENSE MANAGEMENT DENGAN METODE TEST DRIVEN DEVELOPMENT

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 17 Oktober 2022

(Indra Taftazani Wisnuaji)

HALAMAN PERSEMBAHAN

Memories carry our past, also made our present. Pain in life accompanies us, and becomes a meaningful existence. Thank you all for understanding my past, accepting my present, and believing on my future.

(Penulis)

Kalimat diatas beserta skripsi ini dipersembahkan kepada :

1. Orang tua yang senantiasa memberikan doa serta dukungan serta motivasi.
2. Bapak Irving Vitra Paputungan, S.T., M.Sc., Ph.D., selaku dosen pembimbing.
3. Kedua saudaraku yang selalu memberikan motivasi dan dukungannya.
4. Buat semua teman-temanku yang membantu berjalannya magang hingga penulisan skripsi ini.
5. Tukang servis laptop, karena jika tidak ada beliau mungkin skripsi ini akan tertunda lebih lama lagi, walaupun perbaikannya memakan waktu yang lumayan lama.

HALAMAN MOTO

The best way to get started is to quit talking and begin doing

(Walt Disney)

Perjalanan hidup ini penuh dengan cabang, setiap cabang ada karena setiap pilihan yang kita buat, dan merupakan keputusan kita lah di cabang mana kita berjalan saat ini. Setiap cabang yang kita pilih memiliki konsekuensinya sendiri-sendiri, dan semua konsekuensi yang kita terima merupakan tanggung jawab kita sendiri, maka janganlah kita lari dari tanggung jawab itu.

(Penulis)

KATA PENGANTAR

Assalamu'alaikum Wr. Wb.

Alhamdulillah dihaturkan kepada Allah Swt., yang telah melimpahkan rahmat dan taufiq serta hidayat-Nya sehingga dapat menyelesaikan laporan akhir magang di Ozeva Technology. Laporan ini disusun sebagai bukti telah melaksanakan kegiatan magang serta memaparkan informasi yang didapatkan selama melaksanakan penugasan di perusahaan tempat melaksanakan magang. Program magang di program studi Informatika Universitas Islam Indonesia merupakan salah satu jalur yang dapat ditempuh oleh mahasiswa Informatika untuk memperoleh gelar sarjana yang dilaksanakan selama 6 bulan. Program ini memberikan kesempatan untuk mahasiswa Informatika Universitas Islam Indonesia mendapatkan pengalaman di dunia kerja.

Terima kasih banyak kepada semua pihak yang telah membantu keberlangsungan proses magang yang dilaksanakan serta penulisan laporan akhir ini. Adapun pihak tersebut adalah :

1. Allah SWT yang telah memberikan kehidupan dan berkah nikmat yang banyak.
2. Orang tua yang senantiasa memberikan doa serta dukungan kepada penulis saat melaksanakan kegiatan magang.
3. Bapak Irving Vitra Paputungan, S.T., M.Sc., Ph.D., selaku dosen pembimbing.
4. Bapak Tan Gay O, selaku CEO Ozeva Technology yang telah membuka pintu kesempatan untuk melaksanakan kegiatan di Ozeva Technology.
5. Bapak Yustinus Hermawanto dan Ibu Eki Novita Putri selaku *supervisor* dan pembimbing selama melaksanakan magang di Ozeva Technology.
6. Teman-teman yang telah membantu selama magang hingga penyelesaian laporan ini.

Akhir kata, semoga laporan ini dapat bermanfaat bagi pembaca, mohon maaf jika terdapat kesalahan dan kekurangan dalam laporan yang telah ditulis. Sekian yang dapat disampaikan, terima kasih.

Wassalamu'alaikum Wr. Wb.

Yogyakarta, 17 Oktober 2022

(Indra Taftazani Wisnuaji)

SARI

Ozeva Technology merupakan perusahaan penyedia solusi IT yang berspesialisasi dalam pengembangan perangkat lunak, konsultasi dan pelatihan IT. Selama pelaksanaan magang di ozeva, mendapatkan kesempatan bekerja didalam tim penguji dan pengembangan *website*, rutinitas yang terjadi selama magang cukup sederhana, dimana setiap harinya dimulai dengan rapat sekitar pukul 08.00 pagi, dilanjutkan dengan melaksanakan penugasaan hingga istirahat siang, dan melanjutkan sisa pekerjaan yang ada setelahnya hingga jam kerja berakhir sekitar pukul 17.00. Pada proyek yang dipaparkan pada laporan ini, ozeva technology mengembangkan sebuah aplikasi manajemen yang diberi nama Whizeva *Application Client* yang di dalamnya terdapat beberapa aplikasi berbeda yang dapat digunakan setelah membeli lisensi per aplikasi tersebut. Dikarenakan jangka waktu berlangganan yang dapat berbeda diperlukan pengelolaan lisensi yang dapat mempermudah proses pembelian serta monitoring status lisensinya. Guna mendukung proses tersebut, dikembangkanlah solusi cepat berupa aplikasi berbasis *website* yang dapat digunakan segera setelah proses pengembangan selesai dilakukan. Karena merupakan solusi cepat, maka waktu pengembangan tidak terlalu lama, sehingga faktor yang harus diperhatikan dari pengembangan ini adalah waktu, struktur sistem, bebas dari *error*, *maintainable* dan mudah digunakan. Pengembangan *website* menerapkan *test driven development* sebagai metode pengembangan dan *MVC* sebagai pola arsitekturnya yang dinilai cocok dengan penggunaan *website* dimana terdapat 3 jenis pengguna berbeda yang mendukung pengembangannya menjadi modular. Dengan menjadi modular membuat pengembangan dan *maintainability* dari *website* mengalami peningkatan. Proses pengembangan *website* ini terdiri dari beberapa tahapan yaitu pembuatan *test case*, mengerjakan *working code*, *refactor code*, dan diakhiri dengan pengujian akhir berupa *black box testing* yang terdiri dari *functionality* dan *non functionality testing*. Hasil akhir dari pengembangan ini adalah sebuah aplikasi berbasis *website* yang digunakan untuk mengelola dan memantau lisensi aplikasi yang dimiliki penggunanya beserta *bugs report* yang dibuat setelah pengujian akhir dilakukan.

Kata kunci: *test driven development, website, license management, model-view-controller architecture, bugs report*, modular, *TDD, MVC*.

GLOSARIUM

Bugs report	dokumen yang memuat temuan-temuan pada saat dilakukannya pengujian.
MVC	merupakan salah satu pola arsitektur pada pengembangan sistem.
Refactoring	teknik yang digunakan untuk melakukan perubahan dari kode program yang sudah ada tanpa mengubah fungsi atau perilaku dari sistem.
TDD	merupakan metode pengembangan sistem.

DAFTAR ISI

HALAMAN JUDUL i
HALAMAN PENGESAHAN DOSEN PEMBIMBING ii
HALAMAN PENGESAHAN DOSEN PENGUJI iii
HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR iv
HALAMAN PERSEMBAHAN v
HALAMAN MOTO vi
KATA PENGANTAR vii
SARI viii
GLOSARIUM ix
DAFTAR ISI x
DAFTAR TABEL xii
DAFTAR GAMBAR xiii
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.1.1 Gambaran Umum Perusahaan 1
1.1.2 Model Bisnis Perusahaan 2
1.1.3 Proyek yang Dikerjakan 2
1.2 Ruang Lingkup 4
1.2.1 Proyek Pengembangan Ozeva *Sales* 4
1.2.2 Proyek Pengujian Whizeva *Application* 4
1.2.3 Proyek Pengembangan Ozeva *License Management* 5
1.3 Tujuan 5
1.4 Manfaat 6
1.5 Sistematika Penulisan 6
BAB II LANDASAN TEORI DAN TINJAUAN PUSTAKA 8
2.1 *Test Driven Development* 8
2.2 *Model-View-Controller Architecture* 9
2.3 *Unit Testing* 9
2.4 *Black Box Testing* 10
2.5 Tinjauan Pustaka 11
BAB III PELAKSANAAN MAGANG 16
3.1 Proyek *License Management* (Sistem Secara Global) 16
3.2 Manajemen Proyek 16

3.3 Aktivitas Umum dalam Proyek 18
3.3.1 Aktivitas Sebagai *Website Developer* 18
3.3.2 Aktivitas QA(Tester) 19
3.4 Timeline Pengembangan Website 20
3.5 Sebelum Siklus Test Driven Development 20
3.6 *Envisioning* 22
3.7 Siklus *Test Driven Development* 26
3.7.1 Priority Modeling dan Model Storming 26
3.7.2 Membuat *Unit Test Case* 26
3.7.3 Menuliskan Kode 28
3.7.4 Melakukan *Refactoring* 41
3.8 Pengujian Akhir 42
3.9 Hasil dan Pembahasan 45
3.9.1 Hasil 45
3.9.2 Pembahasan 47
BAB IV REFLEKSI PELAKSANAAN MAGANG 52
4.1 Relevansi Akademik 52
4.2 Pembelajaran Magang 54
BAB V PENUTUP 57
5.1 Kesimpulan 57
5.2 Saran 58
DAFTAR PUSTAKA 59
LAMPIRAN 61

DAFTAR TABEL

Tabel 3.1 Timeline Pengembangan *Website* 20
Tabel 3.2 Beberapa *test case* yang sudah dikembangkan 43
Tabel 3.3 Siklus *test driven development 1* 48
Tabel 3.4 Siklus *test driven development* 2 49
Tabel 3.5 Siklus *test driven development* 3 49

DAFTAR GAMBAR

Gambar 1.1 Tampak muka Gedung Ozeva Technology 1
Gambar 1.2 Lokasi Ozeva Technology pada Google Maps 2
Gambar 2.1 Siklus *test driven development* 8
Gambar 2.2 *MVC Architecture* 9
Gambar 3.1 *Use case diagram customer* 22
Gambar 3.2 *Activity Diagram* Login 23
Gambar 3.3 *Activity Diagram Order license* 24
Gambar 3.4 *Branch revision graph* TortoiseGit 25
Gambar 3.5 Mailtrap *Inbox* 26
Gambar 3.6 *Unit test case* 1 27
Gambar 3.7 *Unit test case* 2 27
Gambar 3.8 *Unit test case* 3 28
Gambar 3.9 Fungsi *store* yang dimiliki *customer* 29
Gambar 3.10 List pesanan lisensi 29
Gambar 3.11 Verifikasi akun baru 30
Gambar 3.12 Tampilan verifikasi akun (tampilan pada awal pengembangan) 30
Gambar 3.13 *Almost expire subscription* 31
Gambar 3.14 Lisensi yang hampir habis masa berlakunya 31
Gambar 3.15 Fungsi *index* 32
Gambar 3.16 Tampilan dari *index function* 32
Gambar 3.17 Fungsi *create* 33
Gambar 3.18 *Impersonating* 33
Gambar 3.19 *Leave impersonating* 33
Gambar 3.20 Tampilan sebelum *impersonating* 34
Gambar 3.21 Tampilan setelah *impersonating* 34
Gambar 3.22 CSS 35
Gambar 3.23 *View* pemilihan tipe lisensi 35
Gambar 3.24 *Observer* 36
Gambar 3.25 *Welcome email notification* 36
Gambar 3.26 *Middleware check role* 37
Gambar 3.27 *Middleware auth check* 37
Gambar 3.28 Penggunaan *authcheck* 37

Gambar 3.29 S*ubscription controller test*.....38
Gambar 3.30 *Login register controller test* 39
Gambar 3.31 *Welcome email notification*.....40
Gambar 3.32 *Reset password notification* 40
Gambar 3.33 *Commit* kode baru 41
Gambar 3.34 Penambahan kode karena fungsi lain.....42
Gambar 3.35 Beberapa *test case* login.....44
Gambar 3.36 *Test case payment* 44
Gambar 3.37 *Homepage* ozeva.....45
Gambar 3.38 Rincian informasi harga 45
Gambar 3.39 Pembelian lisensi.....46
Gambar 3.40 Halaman *customer id management* 46
Gambar 3.41 *Bugs report*.....47

BAB I
PENDAHULUAN

1.1 Latar Belakang

1.1.1 Gambaran Umum Perusahaan

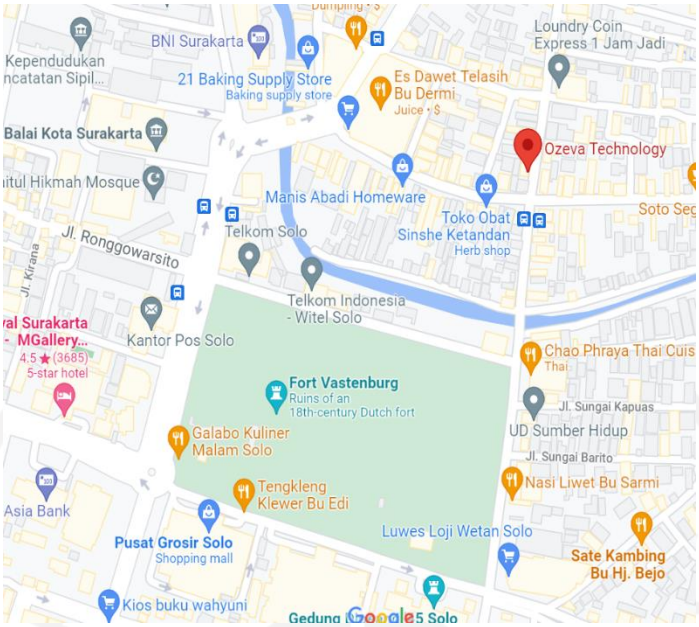
Ozeva Technology merupakan perusahaan penyedia solusi IT yang berspesialisasi dalam pengembangan perangkat lunak, konsultasi dan pelatihan IT. Sebagai perusahaan teknologi, Ozeva mengembangkan perangkat lunak yang dibutuhkan bisnis dan industry untuk memaksimalkan keuntungan dan meningkatkan efisiensi mereka dengan memberikan solusi yang tepat. Perangkat lunak tersebut harus memenuhi beberapa kriteria yaitu *user friendly*, *reliable*, *high performance*, mudah digunakan serta disesuaikan dengan kebutuhan bisnis perusahaan yang berbeda-beda (Ozeva Technology, 2021).

Dalam pelaksanaan pekerjaan sehari-hari, terdapat beberapa *core value* yang wajib untuk dipegang oleh pekerjanya, diantaranya :

a. *Result oriented.*
b. *Teamwork.*
c. *Integrity.*
d. *Accountability.*
e. *Continuous Learning.*

Ozeva Technology dikepalai oleh seorang CEO bernama bapak Tan Gay O, yang dibawahnya terdapat departemen *human resource*, *website developer*, dan *software developer*. Kantor Ozeva Technology beroperasi di gedung 3 lantai yang berada di tengah kota Solo, Jawa Tengah, tepatnya di jalan Kapten Mulyadi 52A, di dekat Benteng Vastenburg dan balai kota Solo seperti yang ditunjukkan pada gambar 1.1 dan 1.2.

Gambar 1.1 Tampak muka Gedung Ozeva Technology



Gambar 1.2 Lokasi Ozeva Technology pada Google Maps

1.1.2 Model Bisnis Perusahaan

Seperti yang telah dideskripsikan pada gambaran umum perusahaan di bagian 1.1.1, perusahaan Ozeva bergerak pada beberapa bidang yang digunakan untuk menjalankan bisnis mereka. Sebagai penyedia solusi IT, Ozeva menerima pembuatan aplikasi sesuai permintaan dari *customer*, aplikasi yang dimaksud dapat berupa aplikasi yang sudah dikembangkan oleh Ozeva sebelumnya, seperti Ozeva Sales, dan lainnya, atau berupa aplikasi baru yang disesuaikan dengan keperluan *customer*. Selain dari pembuatan aplikasi baru, Ozeva juga menawarkan pembelian aplikasi yang telah mereka kembangkan, dimana *customer* dapat membeli aplikasi beserta kustomisasinya, kustomisasi yang dilakukan akan dikenakan biaya lebih berdasarkan tipe aplikasi yang dibeli. Selain dengan pembuatan dan pembelian aplikasi yang sudah ada, Ozeva juga menawarkan *license subscription*, dimana *customer* melakukan pembayaran langganan perbulan untuk mendapatkan servis *maintenance* gratis, *training* serta installasi aplikasi, dalam hal *training* dan instalasi, seluruh *customer* akan mendapatkan *training* dan instalasi gratis hanya untuk instalasi pertama, akan tetapi jika *customer* tersebut berlangganan maka instalasi serta *training* selanjutnya tidak perlu melakukan pembayaran lagi.

1.1.3 Proyek yang Dikerjakan

Selama pelaksanaan magang dilakukan, terdapat beberapa proyek yang sedang berlangsung di Ozeva baik proyek yang terlibat maupun tidak terlibat, seperti :

Proyek Pengembangan Ozeva *Sales*

Proyek ini adalah sebuah pengembangan aplikasi PoS (*Point of Sale)* dalam *platform mobile* yang nantinya akan terkoneksi dengan aplikasi Whizeva, dimana Whizeva merupakan aplikasi *desktop* yang terdiri dari 5 aplikasi berbeda yaitu Whizeva *Sales*, Whizeva *Enterprise Resource Planning*, Whizeva *Payroll*, Whizeva *Restaurant*, dan Whizeva *Hotel*. Proyek ini dikembangkan dengan menggunakan metode *scrum* dan telah berjalan sejak sebelum pelaksanaan magang dan berakhir pada bulan Oktober. Pada proyek ini, tugas yang didapatkan adalah melakukan pengujian aplikasi baik fitur maupun koneksi data dari Whizeva dan Ozeva *Sales,* dimana pengujiannya dilaksanakan secara *black box testing*.

Proyek Pengujian Whizeva *Application*

Proyek ini merupakan sebuah proyek pengujian aplikasi Whizeva versi terbaru, pada proyek ini tugas yang diberikan adalah untuk melakukan pengujian pada fitur – fitur aplikasi Whizeva *Sales,* dan Whizeva *Enterprise Resource Planning,* serta pengujian pada tampilan baru Whizeva. Pengujian ini dilakukan secara *black box testing* dan berlangsung selama kurang lebih 1 bulan.

Proyek Pengembangan Ozeva *Farm*

Proyek ini adalah pengembangan aplikasi manajemen peternakan yang berbentuk *mobile application* yang sementara hanya ada versi *cloud,* aplikasi ini memiliki beberapa fitur yaitu pemberian pakan, pencatatan pengengeluaran dan penghasilan, pencatatan pengambilan telur, *packing,* pencatatan hewan ternak mati, pembelian dan penjualan ternak. Dengan menggunakan sistem versi *cloud,* akan memudahkan pengguna melakukan sinkronisasi data barang dan stok ke toko-toko online. Tidak ada keterlibatan dalam pengembangan ini dikarenakan pengembangan ini dimulai pada bulan februari dimana sudah diakhiri masa magang.

Proyek Pengembangan Ozeva *License Management*

Proyek ini adalah pembangan aplikasi *website* yang dipergunakan untuk penjualan lisensi Whizeva *application* seperti Whizeva *Sales*, dan lain-lainnya. Selain itu, aplikasi ini juga dapat digunakan untuk memantau kapan waktu berlangganan habis dan harus dilakukan perpanjangan, serta sejarah pembelian lisensi aplikasi yang pengguna miliki. Pada *website* ini jjuga terdapat pengguna dengan sebutan *reseller*, atau pengguna yang sudah menjadi *partner*

perusahaan untuk menjual lisensi melalui mereka, dimana *reseller* dapat mendaftarkan pengguna lain yang bukan *reseller* sebagai anggota atau pelanggan tetap mereka. Proyek ini berlangsung selama kurang lebih 4 bulan, mulai dari oktober hingga awal februari, mendapatkan tanggung jawab sebagai *full stack developer*. Pengembangan aplikasi dilakukan dengan metode *test driven development* dan diakhiri dengan pengujian akhir dengan metode *black box testing*.

Proyek ini merupakan proyek yang digunakan sebagai topik skripsi ini, dikarenakan merupakan proyek pengembangan yang diikuti sejak awal pengembangan hingga berakhir, serta memiliki tugas yang lebih besar pada proyek ini dibandingkan dengan proyek sebelumnya.

1.2 Ruang Lingkup

1.2.1 Proyek Pengembangan Ozeva *Sales*

Pada proyek ini, tugas yang diberikan adalah sebagai *tester* untuk menguji fitur-fitur yang ada pada aplikasi, terdapat beberapa aktivitas yang dilakukan serta batasan kerja yang telah ditetapkan.

Aktivitas pada proyek pengembangan ini adalah

a. Melakukan pengujian fitur aplikasi menggunakan metode *black box testing*.
b. Membuat skenario dan *test case* pengujian aplikasi.
c. Membuat laporan akhir pengujian dan *bugs report*.
d. Membuat tampilan menu serta beberapa elemen tampilan seperti *icon button*.

Pada proyek kali ini terdapat batasan kerja yang dimiliki yaitu, pekerjaan hanya dilakukan di bagian pengujian dan laporan serta sedikit bagian pada tampilan dan pembuatan *icon* saja, tanpa ikut serta dalam penulisan kode untuk fungsi fitur.

1.2.2 Proyek Pengujian Whizeva *Application*

Pada proyek ini, tanggung jawab yang diterima adalah sebagai *tester* untuk menguji fitur-fitur yang ada pada aplikasi, terdapat beberapa aktivitas yang dilakukan serta batasan kerja yang telah ditetapkan.

Aktivitas pada proyek pengembangan ini adalah

a. Melakukan pengujian fitur aplikasi menggunakan metode *black box testing*.
b. Membuat skenario dan *test case* pengujian aplikasi.
c. Membuat laporan akhir pengujian dan *bugs report*.

Pada proyek kali ini terdapat batasan kerja yang dimiliki yaitu, hanya bekerja di bagian pengujian dan laporan, tanpa ikut serta dalam penulisan kode dan desain tampilan aplikasi.

1.2.3 Proyek Pengembangan Ozeva *License Management*

Pada proyek ini, tugas yang diemban adalah sebagai *fullstack developer* untuk pengembangan fitur-fitur yang ada pada aplikasi beserta tampilan aplikasi tersebut, selain pengembangan fitur, penugasan berlanjut hingga melakukan pengujian akhir aplikasi. Terdapat beberapa aktivitas yang dilakukan serta batasan kerja yang telah ditetapkan.

Aktivitas pada proyek pengembangan ini adalah

a. Membuat *use case* diagram dan activity diagram.
b. Menuliskan kode program.
c. Membuat tampilan aplikasi.
d. Membuat *unit test case.*
e. Melakukan *automation unit testing.*
f. Melakukan *refactoring code*.
g. Melakukan pengujian akhir dengan metode *black box testing*.
h. Menuliskan *bugs report.*

Terdapat beberapa batasan kerja yang diterima dalam pengerjaan proyek, batasan tersebut antara lain, tidak ikut bekerja dalam mendesain basis data, juga tidak ikut serta dalam *bugs fixing* dan *regression testing*, karena aktivitas tersebut dilakukan setelah masa magang berakhir.

1.3 Tujuan

Dari latar belakang dan ruang lingkup yang telah dijelaskan di atas, maka tujuan yang ingin di capai yang terbagi menjadi tujuan umum dan tujuan khusus.

a. Tujuan Umum

   Tujuan umum dari pengembangan ini adalah menciptakan aplikasi manajemen lisensi yang *reliable* dan dapat langsung digunakan dengan mudah oleh penggunanya.

b. Tujuan Khusus
   1. untuk mengetahui seberapa besar pengaruh metode *test driven development* dalam menciptakan aplikasi yang *reliable*.
   2. Untuk mengetahui apakah penggunaan *architecture pattern* memiliki pengaruh terhadap pengembangan menggunakan metode *test driven development*.

3. Untuk mengetahui seberapa pentingnya dilakukan pengujian akhir sebagai pendukung metode *test driven development.*

1.4 Manfaat

Berdasarkan tujuan pengembangan yang ingin dicapai, maka pengembangan ini diharapkan memperoleh manfaat sebagai berikut:

a. Manfaat Teoritis

   Hasil pengembangan ini diharapkan dapat memberikan manfaat bagi dunia pendidikan khususnya pada pengembangan teknologi manajemen, serta sebagai pijakan dan referensi serta bahan kajian pengembangan-pengembangan selanjutnya.

b. Manfaat Praktis

   1. Bagi pengguna diharapkan dapat memudahkan proses *monitoring* dan manajemen lisensi yang dimiliki oleh pengguna.
   2. Bagi peneliti lain diharapkan dapat bermanfaat sebagai sebagai cara memanfaatkan ilmu yang didapatkan serta menjadi penambah wawasan dan pengalaman langsung dalam proses pengembangan di dunia kerja.
   3. Bagi perusahaan diharapkan dapat bermanfaat dalam proses *monitoring* serta dapat membantu perkembangan aplikasi yang dimiliki.

1.5 Sistematika Penulisan

Sistematika penulisan skripsi berisi rincian urutan penulisan dari setiap bab serta bagian-bagian bab didalam skripsi, mulai dari bab I hingga bab V.

Bab I berisi rincian tentang pendahuluan yang merupakan bagian awal dari skripsi yang dituliskan dan terdiri dari :

a. Latar Belakang.
b. Ruang Lingkup.
c. Tujuan.
d. Manfaat.
e. Sistematika Penulisan.

Bab II berisi rincian tentang landasan teori dan tinjauan pustaka yang berperan penting sebagai landasan teoritik dalam melakukan pengembangan aplikasi dari skripsi yang dituliskan. Bab II terdiri dari :

a. Test Driven Development.

b. Model-View-Controller Architecture.

c. Unit Testing.

d. Black Box Testing.

e. Tinjauan Pustaka.

Bab III berisi rincian tentang Pelaksanaan magang, proses pengembangan aplikasi, adaptasi pengembangan dengan kondisi, hasil pengembangan dan pengujian akhir. Bab III terdiri dari :

a. Proyek License Management.
b. Manajemen Proyek.
c. Aktivitas Umum dalam Proyek.
d. Timeline Pengerjaan Proyek.
e. Sebelum Siklus *Test Driven Development.*
f. *Envisioning.*
g. Siklus *Test Driven Development.*
h. Pengujian Akhir.
i. Hasil dan pembahasan.

Bab IV berisi rincian tentang refleksi pelaksanaan magang berdasarkan pengalaman magang yang dilaksanakan selama kurang lebih 6 bulan. Bab IV terdiri dari :

a. Relevansi Akademik.
b. Pembelajaran magang.

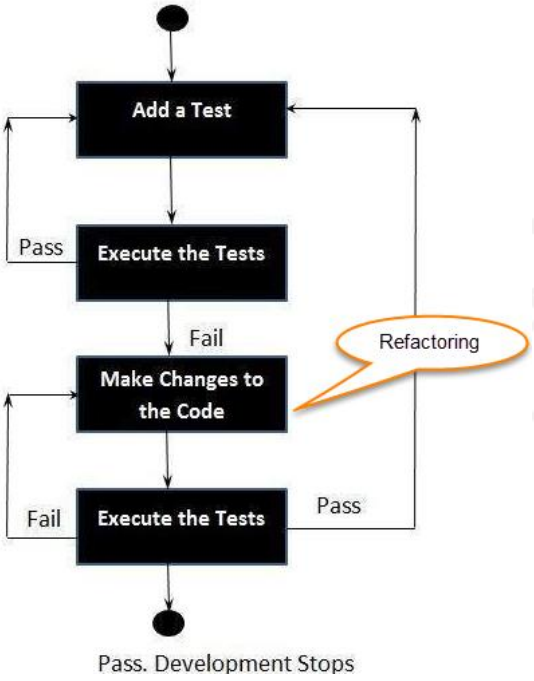
Bab V berisi rincian terhadap hasil temuan dari proses pengembangan yang dilakukan. Bab V terdiri dari :

a. Kesimpulan.
b. Saran.

BAB II
LANDASAN TEORI DAN TINJAUAN PUSTAKA

2.1 *Test Driven Development*

Test driven development atau yang sering disingkat sebagai *TDD* merupakan metode pengembangan aplikasi dimana *test cases* dikembangkan untuk dijadikan spesifikasi dasar dan validasi atas apa yang akan dilakukan oleh kode yang akan dituliskan (Hamilton, 2022). Secara sederhana *TDD*, dilakukan dengan mengembangkan *test cases* untuk setiap fungsionalitas yang ada terlebih dahulu dan dilakukan pengujian pertama, dan jika pengujian gagal baru lah pengembang menuliskan kode sederhana agar fungsi yang ada lulus dari pengujian, setelah lulus pengujian kode dapat dilakukan *refactoring* jika diperlukan. Hal ini diberlakukan untuk membuat kode yang tidak terlalu komplek atau rumit. Penggunaan metode *test driven development* dilakukan dengan bantuan beberapa metode pendukung seperti penggunaan *MVC architecture*, *unit testing*, dan *black box testing*. Metode tersebut terpilih untuk memenuhi *requirement* pengembangan *website*. Siklus *test driven development* ditunjukkan pada gambar 2.1.

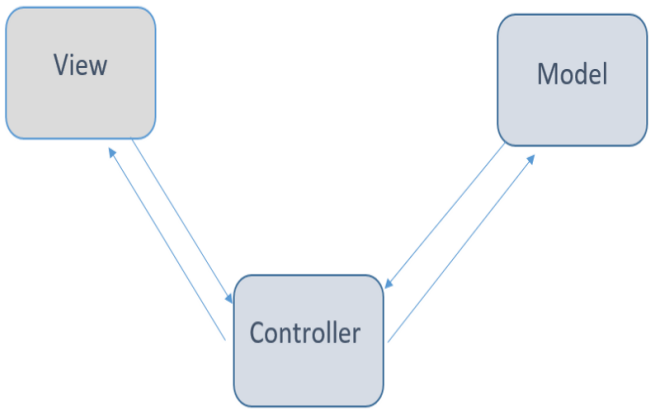


Gambar 2.1 Siklus *test driven development*

Sumber : (Hamilton, 2022)

2.2 *Model-View-Controller Architecture*

Model-view-controller atau yang biasa disebut sebagai *MVC* merupakan sebuah pola arsitektur yang membagi aplikasi ke dalam 3 komponen yaitu *model*, *view* dan *controller* yang masing-masing memiliki fungsi yang berbeda (Pop & Altar, 2013). *Model* merupakan bagian dari sistem yang bertanggung jawab atas semua tugas yang berhubungan dengan data. *View* bertanggung jawab atas semua elemen yang berhubungan dengan *user interface* baik itu tombol, *form* dan lainnya. *Controller* bertanggung jawab untuk menangani semua *event* yang terjadi saat website berjalan, baik *event* yang terjadi karena interaksi dengan pengguna aplikasi maupun yang terjadi karena proses website itu sendiri. *MVC* pada pengembangan ini digunakan sebagai pendukung penerapan *TDD* serta struktur utama dari *website*, dimana dengan mengunakaan *MVC*, pengembangannya akan memprioritaskan pada satu modul setiap siklus *TDD*-nya, sehingga pada saat masuk ke siklus selanjutnya pengembang tidak perlu merubah isi modul sebelumnya, serta membuat pengujian dapat dilakukan per modul yang telah dikembangkan yang akan memberikan efek kepada besarnya *maintainability* yang dimiliki oleh *website* yang dihasilkan. Gambar *MVC* ditunjukkan pada gambar 2.2.



Gambar 2.2 *MVC Architecture*

Sumber : (Svirca, 2020)

2.3 *Unit Testing*

Unit Testing merupakan salah satu jenis pengujian *software* dengan cara menguji tiap *unit* atau modul, pengujian ini dilakukan oleh tim pengembang karena diperlukannya detail *internal* dari kode yang dituliskan (Lewis, Veerapillai, & Dobbs, 2009). Tim pengembang

melakukan *unit testing* ini selama siklus *TDD* berlangsung. Pengujian dilakukan saat adanya *unit* yang baru dibuat maupun yang telah dilakukan *refactor*. Pengujian ini diberikan untuk memastikan bahwa setiap komponen dari kode bekerja sesuai yang diharapkan. *Unit testing* merupakan kegiatan utama serta memiliki efek yang paling besar dalam keberhasilan siklus pengembangan *TDD*. Dalam metode *TDD*, pembuatan *test case* dari *unit testing* diselesaikan terlebih dahulu sebelum memulai menuliskan program, karena proses pembuatan *unit testing* diselesaikan terlebih dahulu dan dijadikan suatu *stack* atau tumpukan fitur, terdapat kemungkinan pengembang berbeda yang membuat kode program dari *unit testing* yang telah dikembangkan, oleh karena itu peran *unit testing* tidak hanya digunakan untuk melakukan pengujian akan tetapi juga digunakan sebagai sarana komunikasi, dengan hanya melihat dari *test case unit testing* dan beberapa *requirement* tambahan pengembang lain dapat membuat kode program yang sesuai. Oleh sebab itu pengembang yang menggunakan *TDD* sebagai metode pengembangan diharuskan untuk menulis *test case* untuk *unit testing* yang jelas, karena jika terdapat kesulitan dalam membaca *unit testing* maka proses pengerjaan setiap siklusnya akan mengalami keterlambatan.

2.4 *Black Box Testing*

Black Box Testing adalah salah satu jenis pengujian *software* untuk memastikan bahwa setiap fungsionalitasnya bekerja sesuai dengan kebutuhan atau spesifikasi yang sudah ditentukan sebelumnya (Nidhra & Dondeti, 2012). Pengujian dilakukan setelah siklus *TDD* selesai baik secara keseluruhan maupun saat salah satu modul yang ada diuji dan berhasil. Pengujian *black box* yang dilakukan adalah pengujian fungsionalitas, pengujian non-fungsionalitas serta *regression testing*.

Pengujian fungsionalitas merupakan proses pengujian untuk mengetahui apakah fungsi yang dimiliki tiap-tiap fiturnya sudah berjalan dengan baik, *functional testing* yang dilakukan di ozeva technology digunakan untuk menguji sistem secara menyeluruh. Pengujian non-fungsionalitas adalah pengujian yang digunakan untuk menguji aspek non-fungsionalitas, diantaranya yang diujikan adalah kecepatan *load* pada laman *website*, tampilan layar pada kondisi *portrait* maupun *landscape* pada saat website dibuka melalui *mobile phone*. Sedangkan *regression testing*, merupakan pengujian baik pengujian fungsionalitas maupun non-fungsionalitas yang dilakukan setelah pengujian akhir dan *bugs fixing* selesai dilakukan.

Pada saat melakukan pengujian diterapkan metode *boundary value analysis* dimana pengujian dilakukan pada nilai domain batas yang dapat diterima oleh setiap data. Pengujian

ini dilakukan untuk memastikan *error* atau *bug* yang mungkin terlewat pada saat proses pengembangan, terutama pada bagian interaksi di *user interface* seperti *error messages* dan lain sebagainya. *Black box testing* merupakan metode pengujian kedua yang digunakan yang dilakukan diluar dari *TDD*, metode ini dinilai perlu untuk digunakan karena pada pengembangan menggunakan metode *TDD*, pengujian yang dilakukan hanya berdasarkan perspektif dari pengembang tanpa perspektif dari pengguna, maka dari itu diperlukan sebuah metode pengujian yang mengutamakan perspektif dari pengguna, hal ini dilakukan untuk mengetahui apakah *website* yang dihasilkan mudah untuk digunakan oleh pengguna. Meskipun tidak termasuk dalam proses *TDD*, penggunaan *black box testing* sangat penting untuk dilakukan karena merupakan metode untuk mengatasi kelemahan dari *TDD* yang terpusat kepada perspektif pengembang.

2.5 Tinjauan Pustaka

Ditengah pandemi seperti yang kita alami sekarang banyak individu yang tidak dapat atau lebih memilih melakukan sesuatu secara daring daripada bertemu langsung, sehingga terdapat masalah yang mengharuskan dikembangkannya *website license management* yang dapat digunakan dalam jangka panjang / *maintainability* yang tinggi. Karena masalah ini menjadi salah satu masalah yang harus segera terselesaikan, biaya yang tidak besar serta waktu pengembangan juga tidak terlalu panjang sehingga penerapan prinsip *agile* sangatlah membantu dalam pengembangannya, waktu pengembangan yang tidak terlalu lama dapat menyebabkan banyaknya terjadi *error* dan *bug* serta, adanya kode yang tidak terstruktur dengan baik, sehingga pengembangan menitik beratkan kepada waktu pengerjaan dan pengujian *error*.

Terdapat beberapa metode yang menjadi pertimbangan untuk menjadi metode dasar pengembangan *website* ini diantaranya : *continuous integration, scrum, extreme programming, acceptance test driven development,* dan *test driven development* dimana *test driven development* yang terpilih menjadi metode pengembangan yang disebabkan oleh beberapa perbandingan antara metode yang ada.

Metode *continuous integration* merupakan metode dimana pengembang menambahkan perubahan-perubahan kecil pada kode dan diikuti dengan *automated testing* secara terus menerus yang memungkinkan *bug* diperbaiki dengan sangat cepat, yang secara tidak langsung membantu dalam *continuous deployment* (Shahin, Babar, & Zhu, 2017). Akan tetapi *continuous integration* pada jangka panjang memerlukan biaya yang sangat besar sehingga

maintainability dari metode ini cukup rendah, dikarenakan penggunaan sejumlah *pipeline* terus menerus. Hal ini menjadi alasan utama tidak digunakannya *CI* karena *maintainability* dari metode ini cukup sulit untuk dipertahankan dalam jangka panjang.

Metode *scrum* merupakan metode yang sangat populer digunakan karena dapat melakukan pengerjaan dengan tempo yang cepat dan progres proyek dan kinerja masing – masing individu yang terlihat jelas dengan adanya *daily meeting* (Koi-Akrofi, Koi-Akrofi, & Matey, 2019), akan tetapi belajar dari proyek – proyek sebelumnya, terdapat kemungkinan adanya kesulitan melakukan *daily meeting*, dikarenakan terdapat beberapa masalah terkait koneksi masing-masing anggota tim, serta pada pengembangan kali ini, anggotanya berasal dari 2 departemen berbeda sehingga terdapat beberapa anggota yang mengalami kesulitan dalam melaksanakan *daily meeting* secara daring karena terdapat beberapa jadwal tim satu dan lainnya.

Metode *extreme programming* merupakan metode pengembangan yang memungkinkan tim kecil dan menengah sistem yang berkualitas tinggi dan memiliki *adaptability* yang tinggi terhadap perubahan kebutuhan perangkat (Fruhling & De Vreede, 2014). *Adaptability* ini diperoleh dari *feedback* yang diberikan terus – menerus dan diberikannya kemampuan kepada pengembang untuk menambahkan atau mengurangi spesifikasi jika terdapat perubahan kebutuhan. Tetapi pada penerapannya *extreme programming* memerlukan peran pengguna atau calon pengguna sejak awal pengembangan sistem, sehingga diperlukannya perwakilan calon pengguna yang harus ada pada proses pengembangan sistem, hal ini menjadi alasan utama mengapa *extreme programming* tidak cocok digunakan pada pengembangan kali ini, dikarenakan pandemi yang terjadi.

Metode *acceptance test driven development* atau *ATDD* merupakan metode yang mirip dengan penerapan *test driven development*, yang membedakan *ATDD* dari metode *TDD* adalah, pada *ATDD* pengujian berfokus kepada *acceptance test* bukan berfokus kepada *unit testing*, sehingga pada praktiknya pengujiannya lebih menitik beratkan kepada *behavior interface* atau dengan kata lain, *ATDD* melakukan pengujian dengan perspektif pengguna bukan dari perspektif pengembang sedangkan pada *TDD* fokus pengujian menitik beratkan kepada *unit testing* (Zeba & Ahsan, 2017), akan tetapi dengan anggota tim yang berasal dari departemen *web developer* dan *QA*, pengujian dapat dilakukan dari 2 perspektif dengan menggabungkan *TDD* dan *black box testing*, sehingga penggunaan *TDD* dinilai lebih baik dengan pertimbangan anggota tim yang ikut serta dalam pengembangan proyeknya, serta dengan sifat *TDD* yang modular membuat *maintainability* dari sistem lebih baik. *Maintainability* menjadi alasan utama

terpilihnya metode pengembangan dengan mengetahui bahwa semua metode yang memegang prinsip *agile* memiliki waktu pengembangan yang relatif cepat dan menggunakan banyak pengujian, hal ini dimiliki oleh sebagian besar metode *agile*, akan tetapi terdapat faktor lain yaitu komposisi anggota tim dan keadaan pandemi yang dialami menjadi salah satu faktor yang harus dipertimbangkan dalam pemilihan metode yang diterapkan, faktor – faktor tersebut menjadi alasan utama mengapa penggunaan *TDD* terpilih sebagai metode pengembangan *website*.

Untuk mendukung perkembangan sistem dengan metode *TDD*, diperlukan kode yang terstruktur dengan baik, struktur modular sangat berperan besar dalam kesuksesan pengembangan dengan metode *TDD*, maka penerapan *MVC* sangat membantu, dimana *MVC* merupakan sebuah pola arsitektur yang membagi aplikasi ke dalam 3 komponen yaitu *model*, *view* dan *controller*, yang setiap bagiannya memiliki peran yang berbeda – beda (Pop & Altar, 2013). *Model* merupakan bagian dari sistem yang bertanggung jawab atas semua tugas yang berhubungan dengan data. *View* bertanggung jawab atas semua elemen yang berhubungan dengan *user interface* baik itu tombol, *form* dan lainnya. *Controller* bertanggung jawab untuk menangani semua *event* yang terjadi saat *website* berjalan, baik *event* yang terjadi karena interaksi dengan pengguna aplikasi maupun yang terjadi karena proses *website* itu sendiri.

Pada *TDD* pengembang melakukan *unit testing* untuk tiap fungsionalitas yang ada. *Unit testing* merupakan salah satu jenis pengujian *software* dengan cara menguji tiap *unit* atau modul. *Unit testing* buat diawal sebagai acuan penulisan *code*, serta sebagai alat uji *code* yang baru dituliskan ataupun *code* yang telah dilakukan *refactoring* sebelumnya (Lewis, Veerapillai, & Dobbs, 2009).

Selain waktu, struktur, dan bebas dari *error* atau *bug*, tentunya salah satu hal terpenting yaitu apakah *website* mudah untuk digunakan atau tidak, akan tetapi pada pengembangan *TDD* semua pengujian dilakukan berdasarkan perspektif pengembang, sehingga sangatlah penting untuk dilakukan *black box testing* karena *black box testing* lebih berpusat kepada perspektif pengguna. Pengujian ini dilakukan diluar siklus *TDD* dengan penerapan metode *boundary value analysis* atau *boundary testing* dimana penguji memasukkan data pada nilai batas data yang ada pada spesifikasi. Pengujian ini dilakukan untuk memastikan *error* atau *bug* yang mungkin terlewat pada saat proses pengembangan, terutama pada bagian interaksi di *user interface* seperti *error messages* dan lain sebagainya.

Terdapat sebuah penelitian yang telah mengkaji dampak dari penggunaan *TDD* sebagai metode dalam mengembangkan aplikasi, hasil dari penelitian tersebut mengindikasikan

terjadinya pengurangan *bug* dan *error pre-release* sebanyak 40% (Nagappan, Maximillien, Bhat, & Williams, 2008).

Pada Penelitian ini telah dibuktikan bahwa penggunaan *tool support* seperti IDE, *framework* dapat meningkatkan kemampuan *refactoring* dan kapabilitas dari *test driven development* (Erdogmus, Melnik, & Jeffries).

Pada jurnal ini dituliskan bahwa terdapat kesulitan dalam penerapan *test driven development* ketika mengkonversikan kebutuhan pengguna ke dalam *unit test*, serta masalah pada pengujian *test driven development* hanya menguji *back-end* saja tetapi tidak menguji *front-end* (Thohari & Amelia, 2018).

Pada jurnal ini, dipaparkan beberapa hal terkait penggunaan *test driven development* dan *MVC architecture,* dimana *developer* melakukan komunikasi melalui *unit test* yang dikembangkan, *test case* akan menjadi dasar dan merupakan kesepakatan dari seluruh pengembang yang ikut serta dalam pelaksanaan pengembangan, dimana dengan pendekatan tersebut terdapat beberapa kelebihan yang dapat diterima seperti kualitas produk mengalami peningkatan, mengurangi ambiguitas dalam *requirement* yang ada, dan segala perubahan yang terjadi akan tercatat dalam *unit test* (Anjaria, 2018).

Pada jurnal ini, dipaparkan bahwa penggunaan *test driven development* memberikan efek yang signifikan terhadap kualitas kode yang dikembangkan serta mengurangi biaya perawatan atau *maintenance cost* untuk *website* dikemudian hari, hal ini dapat terjadi karena *test driven development* mengembangkan *test case* sejak masa pengembangan dan berfokus kepada *subset* informasi yang lebih kecil (Yenduri & Perkins, 2006).

Dari tinjauan pustaka diatas dapat disimpulkan bahwa penggunaan *test driven development* dan *tool support* yang tepat dapat membantu peningkatan kualitas kode sehingga *website* menjadi lebih *maintainable*, meskipun terdapat kekurangan dari sisi pengujiannya, pada pengembangan kali ini masalah tersebut akan diatasi dengan penggunaan *black box testing* sehingga terdapat pengujian dari sisi pengguna, serta penggunaan *test case* sebagai alat komunikasi dapat menjadi salah satu solusi dalam sulitnya berkomunikasi pada masa pandemi yang dialami pada saat mengembangkan *website* ini. Oleh karena itu penggunaan *test driven development* cocok untuk digunakan dalam mengembangan *website* ini.

Dengan penggunaan metode – metode tersebut sudah mencakup hal – hal penting yang menjadi faktor penting dalam pengembangan ini. Pengembangan *website* memiliki target yaitu pengembangan *website* yang terstruktur, *maintainable*, bebas dari *bug* sehingga dapat cepat

dirilis setelah pengembangan selesai, dan yang paling penting adalah mudah digunakan oleh pengguna aplikasi.

BAB III
PELAKSANAAN MAGANG

3.1 Proyek *License Management* (Sistem Secara Global)

Pengembangan *license management* ini dilakukan oleh Ozeva untuk mempermudah penjualan, serta proses perpanjangan dari lisensi yang dimiliki oleh *customer*, serta sebagai bentuk adaptasi dari pandemi, dimana *customer* mengalami kesulitan untuk melakukan pembelian yang mengharuskan *customer* datang langsung ke kantor. Lisensi yang dijual melalui *website* adalah lisensi dari semua aplikasi yang telah dikembangkan oleh Ozeva, seperti Ozeva Sales, Ozeva Farm, Whizeva, dan lain-lain. Untuk mengawali pengembangan *website* secara keseluruhan dimulai lah proyek ini, dimana pada proyek ini pembuatannya difokuskan pada aplikasi Ozeva Sales. Proyek awal yang dijalankan ini, *website* dikembangkan untuk membantu manajemen lisensi aplikasi Ozeva Sales saja. Pada *website* yang dikembangkan terdapat beberapa fitur yang dijalankan oleh 3 jenis pengguna yang berbeda, yaitu *admin, reseller,* dan *customer. Admin* pada *website* ini adalah karyawan dari Ozeva sendiri, dimana mereka dapat menjalankan beberapa fitur seperti menambahkan atau mengangkat *customer* menjadi *reseller* resmi dari Ozeva, membantu atau mewakilkan *customer* dalam melakukan pembelian atau perpanjangan lisensi. *Reseller* memiliki akses terhadap beberapa fitur seperti mendaftarkan *customer* baru (jika *reseller* mendaftarkan *customer* menggunakan fitur ini, maka *customer* yang didaftarkan masuk kedalam *list customer* dibawah *reseller* tersebut), melakukan pembelian dan perpanjangan lisensi dari *customer* yang berada di *list customer* miliknya. *Customer* memiliki akses terhadap beberapa fitur terkait pembelian dan perpanjangan lisensi serta daftar lisensi yang dimiliki, dan daftar lisensi yang akan *expire*.

3.2 Manajemen Proyek

Pada pengembangan proyek ini, metodologi yang diterapkan dalam proses pengembangan sistem mengikuti siklus atau alur *test driven development*, setelah siklus selesai pengerjaan dilanjutkan oleh tim penguji untuk dilakukan *black box testing* yang terdiri dari serangkaian skenario pengujian yang dibagi menjadi *functionality testing* dan *non-functionality testing*. Pada dasarnya proses atau fase pengerjaan *test driven development* terdiri dari 3 step utama yaitu membuat *unit test case*, menuliskan kode, dan *refactoring* akan tetapi pada pengembangan proyek kali ini terdapat beberapa fase tambahan sebelum dimulainya *test driven*

development, fase-fase tambahan diberlakukan demi menjamin *scalability* dari *website* yang dihasilkan, beberapa fase masuk ke dalam perulangan *test driven development* dan beberapa lainnya tidak. Fase tambahan berikut adalah *envisioning*, *priority modeling*, dan *model storming*. Pada pengembangan proyek ini *database* menggunakan *database* yang sudah ada sehingga tidak terdapat proses pembuatan *database*, dan hanya membangun *requirement* disekitar *database* tersebut.

Untuk fase *envisioning* dilakukan diluar siklus *test driven development* dan hanya dilakukan diawal atau sebelum siklus *TDD* dilakukan. *Envisioning* merupakan fase dimana pengembang melakukan diskusi awal dimana pada fase ini akan ditentukan, pengguna yang akan menggunakan *website* ini, fitur-fitur yang akan dikembangkan, spesifikasi dari *website*, *flow* penggunaan *website*, model *user interface*, alat-alat pendukung pengembangan yang dibutuhkan, serta *architecture pattern* yang akan digunakan.

Priority modeling dan *model storming* dilakukan pada setiap siklus *test driven development*, fase ini biasanya dilakukan pada saat melakukan *meeting* pada hari pertama bekerja setiap minggunya. Pada fase ini, pengembang berdiskusi terkait jangka waktu tiap-tiap siklus *test driven development* yang dilakukan, menentukan fitur mana yang menjadi prioritas, melakukan *re-prioritize* jika diperlukan, menentukan bagaimana *requirement* yang sudah ditentukan diimplementasikan kepada fitur tersebut, menggambarkan *flow* penggunaan tiap fitur, mendiskusikan terkait masalah yang ditemukan pada pengerjaan minggu sebelumnya, serta cara penyelesaian masalah tersebut, selain hal tersebut pada pertemuan tersebut pengembang juga melaporkan progres pengerjaan fitur dan memperlihatkan hasil pada pengerjaan minggu sebelumnya, serta mendiskusikan apakah kode yang dikembangkan perlu dilakukan *refactoring*. Setiap kali siklus *test driven development* dinyatakan selesai, dilakukan pengujian *black box testing*.

Test driven development dilakukan setelah *priority modeling* dan *model storming* dilakukan, pada *test driven development* pengembang pertama-tama membuat *unit test case* sesuai dengan *flow* fitur yang sudah ditentukan pada saat *priority modeling* dan *model storming*. *Unit test case* yang sudah dibuat diujikan untuk memastikan bahwa pengujian tersebut gagal karena belum terdapat kode terkait fitur tersebut, setelah pengujian tersebut gagal maka pengembang dapat memulai menuliskan kode terkait fitur tersebut, kode yang dituliskan dibuat seminimal mungkin sampai *unit test case* yang dibuat lulus uji. Kemudian pengembang melakukan *refactoring* pada kode tersebut jika diperlukan.

Terdapat beberapa alasan perlu dilakukannya *refactoring* diantaranya:

a. Pengujian masih mendeteksi adanya *error*.
b. Penulisan kode yang terlalu rumit.
c. Terdapat kode yang berulang atau kode yang tidak diperlukan.
d. Kode yang menyebabkan *runtime* melewati standar yang ditentukan.

3.3 Aktivitas Umum dalam Proyek

3.3.1 Aktivitas Sebagai *Website Developer*

Pada bidang *web developer*, pihak ozeva technology menggunakan salah satu bentuk dari metode *agile* yaitu *test driven development* atau yang biasa disebut dengan *TDD*. Pada setiap proyeknya biasanya diawali dengan membuat *use case, activity*, dan *entity relationship diagram,* khususnya untuk fitur yang memiliki langkah-langkah penggunaan yang cukup panjang atau kompleks, beberapa fitur yang dapat digunakan oleh banyak jenis pengguna berbeda, tetapi ada beberapa kasus dimana pembuatan *use case*, dan *activity diagram* tidak perlu dibuat, seperti fitur-fitur yang sederhana atau yang sudah cukup jelas langkah penggunaannya seperti fitur *edit profile,* dan beberapa fitur-fitur sederhana lain. Setelah tahap desain awal selesai, dilanjutkan dengan membuat struktur basis data yang akan digunakan dalam proyek.

Setelah seluruh tahap desain selesai dan sudah di terima, pembuatan kode dapat dilakukan, pada *TDD*, penulisan kode dimulai dengan melakukan membuat *test case*, kemudian *developer* menuliskan beberapa kode terkait *test case*, kemudian dilakukan *refactoring* untuk merapikan, serta menyederhanakan kode yang sudah dibuat sebelumnya, langkah ini diulangi setiap menambahkan kode yang baru. Karena pada pengerjaannya *TDD* dilakukan pada saat *unit testing* maka skenario pengujian baik skenario gagal atau skenario berhasil dibuat untuk setiap *function* yang ada pada kode tersebut. *Framework* yang digunakan pada pengembang proyek adalah *framework* laravel, dan *unit testing* dilakukan menggunakan laravel *unit testing*.

Setelah dipastikan setiap pengujian berjalan dengan baik, hasil kode dikirimkan kepada supervisor menggunakan TortoiseGit untuk pengecekan dan konfirmasi hasil pekerjaan, jika kode belum di *approve* atau dianggap masih belum sesuai, maka supervisor akan memberikan arahan terhadap bagian - bagian yang masih belum di *approve* untuk dilakukan revisi, pekerjaan ini akan terus dilakukan sampai supervisor memberikan *approval* terhadap pekerjaan yang telah dilakukan.

Setelah diberikan *approval* dan kode diimplementasikan pada server utama, *developer* akan pergi ke *website* tersebut untuk melakukan pengecekan terhadap pengaruh penambahan elemen-elemen seperti kode, gambar dan lain hal untuk mengetahui apakah setelah ditambahkan elemen tersebut *website* masih cukup cepat dan handal, jika terdapat pengaruh yang cukup besar terhadap performa web, maka perlu dilakukan revisi lebih lanjut, dimana jika performa menurun setelah ditambahkan kode, maka kode perlu dibuat lebih sederhana lagi, sedangkan jika terjadi setelah ditambahkan gambar, *developer* cukup melakukan editing sederhana seperti melakukan kompresi terhadap gambar, dan mengubah format gambar.

3.3.2 Aktivitas QA(Tester)

Pada bidang *Quality Assurance(QA)* atau *tester*, penguji melakukan pengujian menggunakan metode *black box testing*, pengujian dilakukan secara manual dan dibagi menjadi 3 tipe pengujian yaitu *black box testing* yaitu *functional testing, non functional testing,* dan *regression testing* yang seluruh pengujiannya menerapkan teknik *boundary value analysis* atau yang lebih dikenal sebagai *boundary testing*.

a. *functional testing* adalah proses pengujian untuk mengetahui apakah fungsi yang dimiliki tiap-tiap fiturnya sudah berjalan dengan baik, *functional testing* yang dilakukan di ozeva technology digunakan untuk menguji sistem secara menyeluruh.
b. *non-functional testing* adalah proses pengujian yang dilakukan terhadap hal hal tambahan seperti pengaruh aplikasi jika dalam keadaan *portrait* atau *landscape*, beda besaran layar dan faktor tambahan lain. contoh kasus yang dialami : pada beberapa layar hp milik developer, di halaman tertentu pada saat dalam keadaan *landscape* terdapat tombol yang tidak terlihat karena kapasitas layar yang kecil, dan pada developer lain, tombol tetap terlihat.
c. *regression testing* adalah proses pengujian yang dilakukan jika terdapat versi baru pada aplikasi, pengujian ini dilakukan untuk mengetahui efek dari pembaruan aplikasi yang dilakukan. beberapa efek yang sudah pernah dialami selama melakukan pengujian di ozeva technology adalah terdapat penurunan performa aplikasi (aplikasi semakin lambat), terdapat beberapa fitur yang pada versi sebelumnya tidak memiliki *error*, setelah pembaruan terdapat *error*, baik dari aspek *functional* (fungsi tiap fitur) maupun *non functional*.

d. *Boundary value analysis* atau *boundary testing* adalah jenis pengujian yang menitikberatkan pada batas-batas domain data yang diperbolehkan. batas batas tersebut meliputi tipe data, jumlah minimal dan maksimal karakter, dll.

Pengujian dilakukan berdasarkan *test case* yang diberikan oleh supervisor, skenario yang sudah diujikan diberikan tanda sesuai kondisi yang didapatkan, jika berhasil diberi tanda centang hijau, terdapat error diberi tanda silang berwarna merah, dan apabila fungsi memang belum ada atau belum diimplementasikan diberikan tanda kotak merah yang semuanya disimpan dalam 1 file excel, jika pada saat pengujian terdapat skenario yang tidak tercantum dalam *test case* yang diberikan maka skenario tersebut ditambahkan sesuai dengan fitur yang dijalankan skenario tersebut. Dengan diterapkannya *boundary testing, functionality testing* dititik beratkan kepada batas-batas *domain* data yang di-*input*-kan.

3.4 Timeline Pengembangan Website

Pengembangan *license management* terbagi menjadi beberapa aktivitas yang berlangsung selama kurang lebih 3 bulan lamanya yang dijabarkan dalam tabel 3.1.

Tabel 3.1 Timeline Pengembangan *Website*

NO	Aktivitas	Durasi
1	*Envisioning* dan *TDD* 1 pengembangan modul *customer* sampai dengan *refactoring code*	1 Bulan
2	*TDD* 2 pengembangan modul *admin* sampai dengan *refactoring code*	2 Pekan
3	*TDD* 3 pengembangan modul *reseller* sampai dengan *refactoring code*	1 Bulan
4	Pengujian akhir / *black box testing*	1 Pekan
5	Perbaikan *error* dan *bug* yang ditemukan	1 pekan

3.5 Sebelum Siklus Test Driven Development

Terdapat faktor-faktor yang diutamakan pada saat melakukan pengembangan *website.* Faktor yang diutamakan dari pengembangan ini adalah :

a. Waktu pengembangan.
b. Struktur sistem yang mudah dimengerti.
c. *Maintainability* tinggi.
d. Bebas dari *error* dan *bugs.*
e. Mudah digunakan.

f. *Scalable.*

Melihat dari faktor-faktor tersebut diperlukan alat dan metode tambahan yang cocok dikombinasikan dengan metodologi *test driven development*, sehingga faktor-faktor tersebut dapat terpenuhi dengan baik. Dengan menerapkan *envisioning*, *scalability* dari proyek sudah mendapatkan peningkatan karena tujuan dari *envisioning* tersebut adalah meningkatkan *scalability* dari metodologi *test driven development.* Selain itu karena merupakan cabang dari *agile, test driven development* memiliki waktu pengembangan yang lebih cepat, dan karena merupakan metodologi yang mengutamakan pengujian, *error* dan *bugs* dapat diminimalisir sejak tahap pengembangan dilakukan.

Untuk membuat struktur lebih mudah dimengerti pengembang mengimplementasikan salah satu *architectural pattern* yaitu *model-view-controller architectural pattern* serta *framework* laravel. Dengan menerapkan *model-view-controller* dan laravel, pengembang dapat mengembangkan sistemnya secara modular, dengan masing-masing *role* yang ditetapkan dianggap sebagai satu modul. Penggunaan konsep ini juga digunakan untuk mengurangi perubahan kode lain saat salah satu kode ditambahkan atau dilakukan perubahan, kode yang dikembangkan secara modular dapat meningkatkan *maintainability* dari suatu sistem.

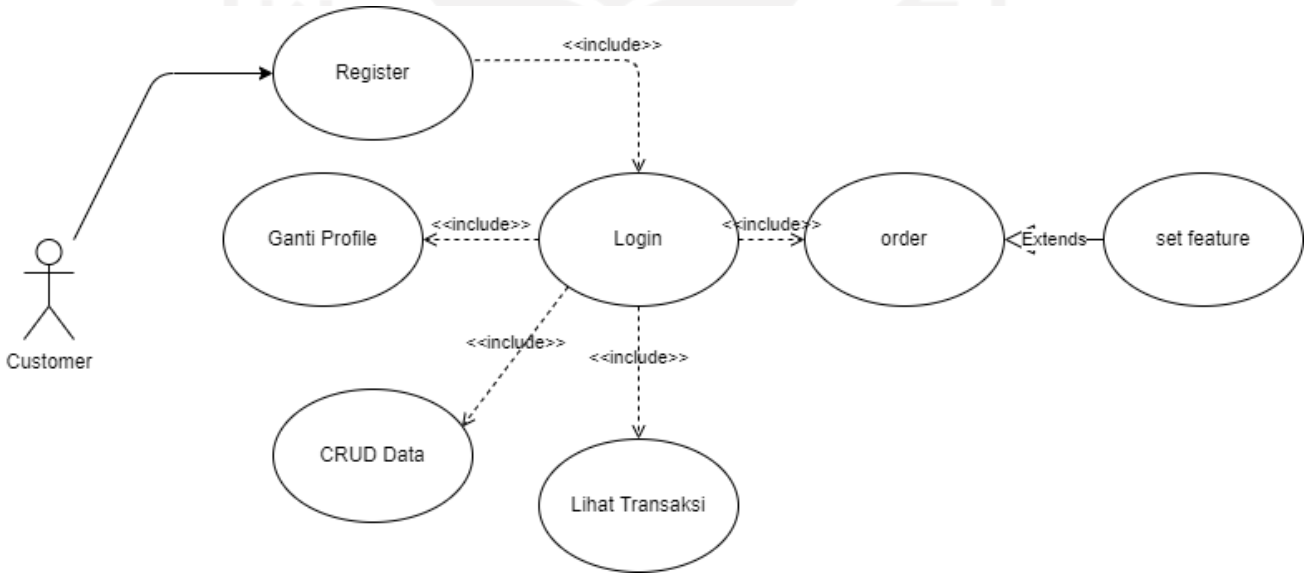
Selain karena laravel mengikuti konsep *model-view-controller*, laravel juga menerapkan *eloquent ORM* sehingga penulisan *query database* menjadi lebih sederhana, sehingga dapat mempercepat waktu pengembangan suatu fitur, laravel juga memiliki *unit test case* yang tersambung dengan kode utamanya pada saat membuat *project* baru, sehingga dapat langsung dituliskan saja *unit test case* yang sudah ditentukan pada folder khusus pengujian pada file proyek, laravel juga memiliki berbagai *package* baik yang sudah ada pada laravel nya sendiri maupun *package* luar yang dapat ditambahkan ke dalam *vendor* laravel dan langsung terkoneksi dengan *vendor* laravel lainnya. Dengan menerapkan metode *test driven development*, pengembang memfokuskan pengembangan kepada 1 fitur, tidak berpindah fokus hingga semua *unit* yang berhubungan dengan fitur tersebut selesai / lulus dalam *unit test*. Dengan perulangan seperti ini, kualitas kode akan terus mengalami peningkatan, membuat *error* dan *bug* mudah ditemukan, serta menjadikan kode yang ditulis lebih *reusable*. Karena berfokus kepada satu fitur per siklus, pengembangan akan secara tidak langsung menjadi modular, sehingga penerapan *TDD*, laravel dan *MVC* memiliki sinergi yang baik dikarenakan ketiganya membuat pengembangan sistem berbentuk modular.

Dengan menjadikan pengembangan menjadi modular, membuat kode menjadi lebih mudah dibaca, lebih mudah dilakukan pengujian dan lebih mudah dilakukan *refactoring*. Tiap

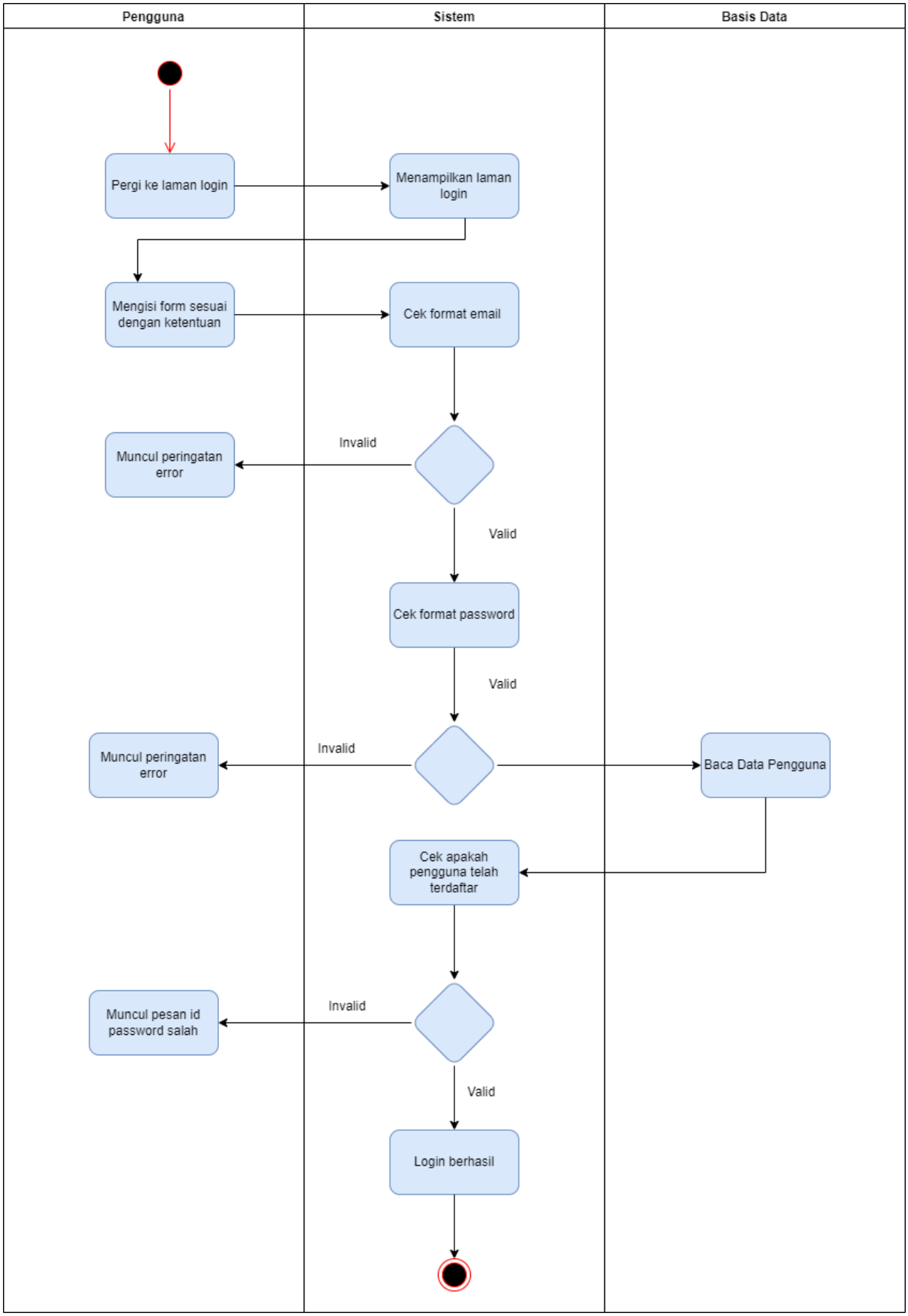
modul pada modular memiliki fungsi yang spesifik yang memperkecil tempat munculnya *bug* maupun *error*, serta meningkatkan *reuseability* karena kita dapat memanggil fungsi dari satu sumber yang ada di modul apapun.

3.6 *Envisioning*

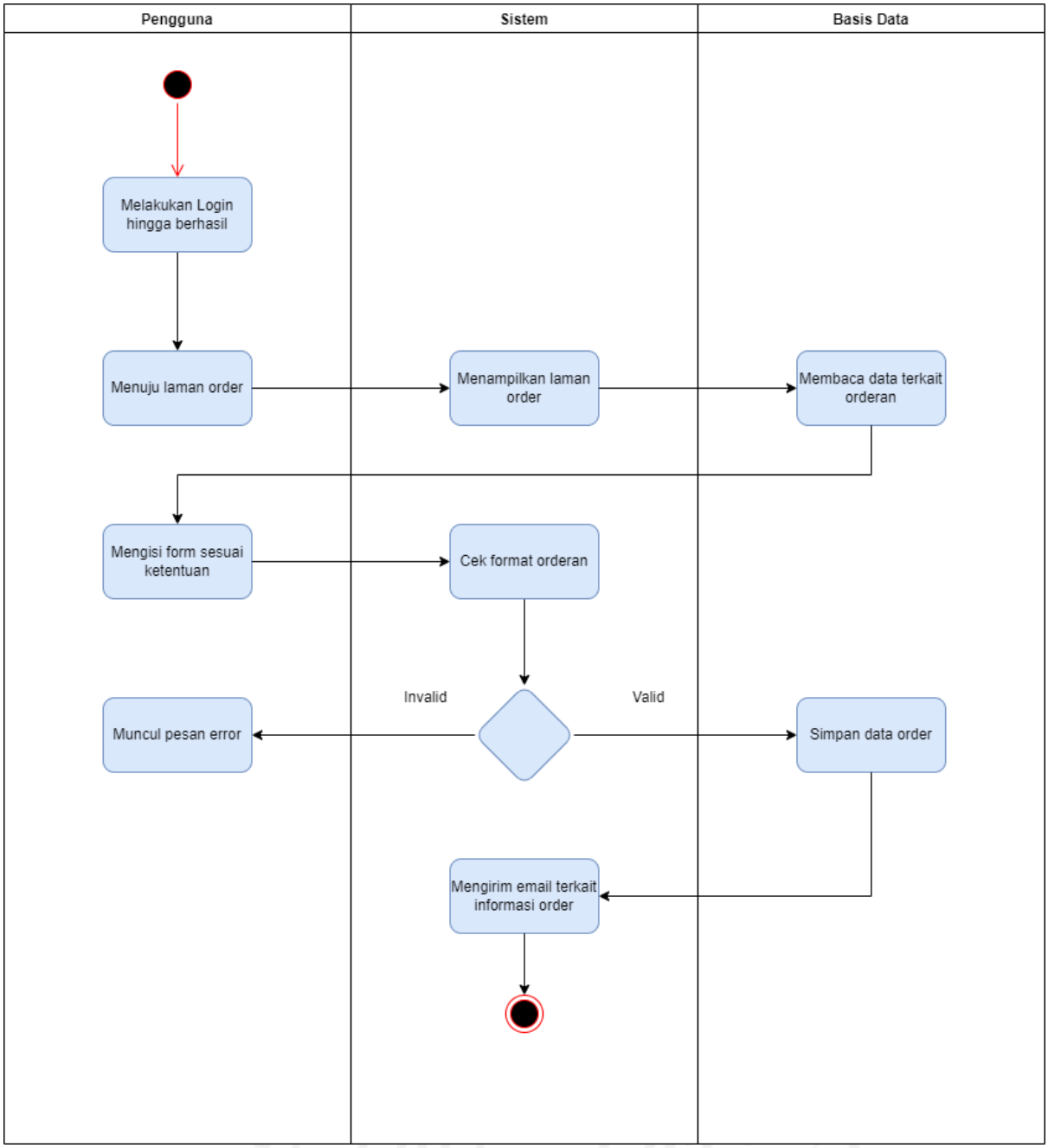
Pada saat melakukan *envisioning* tim pengembang melakukan prediksi atau perkiraan terkait *scope*, dan struktur dari sistem yang akan dikembangkan. Beberapa hal yang diprediksi dengan *envisioning* adalah calon pengguna *website*, fitur masing-masing pengguna, *flow* penggunaan *website*, pembuatan diagram bantu jika diperlukan, spesifikasi *error* seperti tidak dapat login untuk sementara jika terjadi kesalahan login sebanyak tiga kali. Pada saat diskusi telah selesai dilakukan, hasil dari diskusi yang diterima dikumpulkan dan menjadi *initial requirement*, dalam praktiknya *envisioning* masih dilakukan pada saat siklus *test driven development* berlangsung karena dapat terjadi kasus dimana penambahan *requirement* baru maupun penghapusan beberapa *initial requirement*. Diagram yang dibuat pada saat melakukan *envisioning* ditunjukkan pada gambar 3.1, 3.2 dan 3.3.



Gambar 3.1 *Use case diagram customer*



Gambar 3.2 *Activity Diagram* Login



Gambar 3.3 *Activity Diagram Order license*

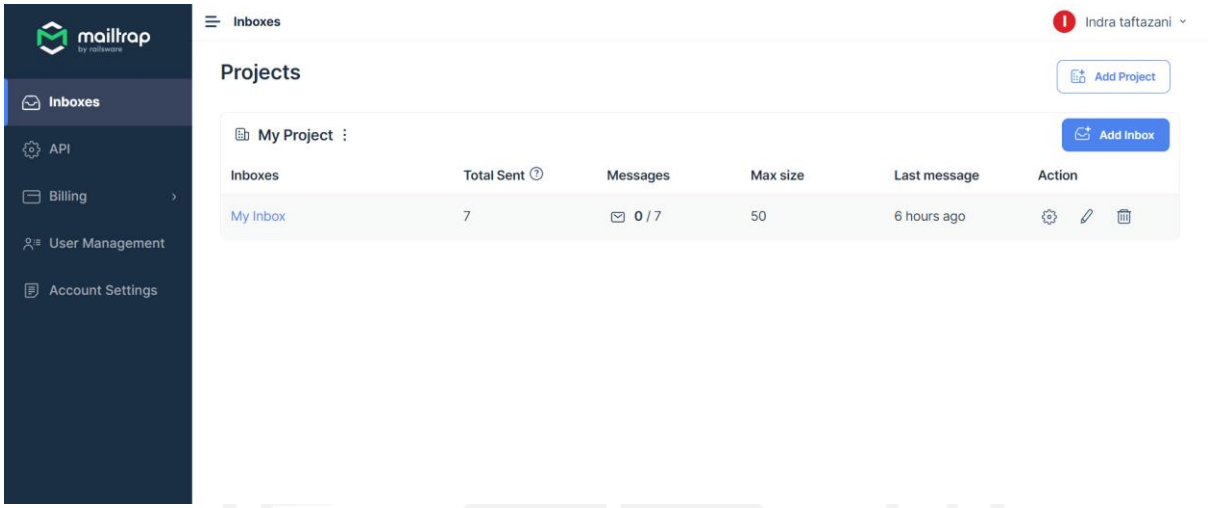
Pada fase *envisioning*, pengembang juga mendiskusikan terkait beberapa alat bantu tambahan yang dapat digunakan baik pada fase *test driven development* maupun pengujian akhir, beberapa alat bantu yang digunakan adalah TortoiseGit dan Mailtrap. TortoiseGit digunakan oleh pengembang untuk melakukan segala kegiatan *push and pull* kode baru, membuat *branch* kerja masing-masing, tortoiseGit dipergunakan karena pada saat pengembangan terdapat beberapa proyek kecil serta *bug fixing* yang terjadi sehingga

pengembang sering berpindah dari *branch* satu dan lainnya sesuai dengan proyek yang dikerjakan. Tampilan *branch* pada TortoiseGit ditunjukkan pada gambar 3.4.

Gambar 3.4 *Branch revision graph* TortoiseGit

Mailtrap merupakan sebuah *email sandbox service* yang digunakan untuk melakukan pengujian terhadap suatu fitur yang berhubungan dengan aktivitas kirim mengirim email. Servis ini memungkinkan kita untuk menguji apakah *email* yang kita kirim menuju penerima yang sesuai, apakah konten pesan sesuai dengan yang diatur dalam kode yang telah dibuat. Terdapat beberapa fitur dari pengembangan modul *customer*, *reseller*, dan *admin/staff* yang berhubungan dengan kirim mengirim *email*, dan setiap fitur tersebut akan

dilakukan pengujian dengan cara menyambungkan *email* yang dikirim ke mailtrap untuk ditampilkan. Penggunaan mailtrap yang sudah dilakukan ditunjukkan pada gambar 3.5.



Gambar 3.5 Mailtrap *Inbox*

3.7 Siklus *Test Driven Development*

3.7.1 Priority Modeling dan Model Storming

Priority modeling dan *model storming* dilakukan setiap pertemuan di awal minggu kerja atau setiap hari senin, pada tahap ini pengembang memberikan prioritas terhadap *requirement* yang sudah ada, menentukan pembagian tugas pengerjaan serta melakukan perkiraan durasi yang diperlukan, perkiraan durasi yang diperkirakan adalah perkiraan durasi yang paling lama, sehingga terdapat kemungkinan pengerjaan lebih cepat daripada durasi yang ditentukan. Selain melakukan pembagian tugas, pengembang juga berdiskusi terkait masalah yang terjadi selama menjalan siklus pada minggu sebelumnya, beberapa pengembang lain akan memberikan solusi cepat dari masalah tersebut, dan memberikan bantuan cepat kepada pengembang yang memiliki masalah tersebut atau bahkan memutuskan untuk mengambil alih bagian itu jika diperkirakan keterlambatan pengerjaan fitur tersebut dapat menghambat progres siklus pengembangan, dari hasil diskusi dapat dimungkinkan terjadi penambahan *requirement*, penghapusan *requirement*, dan pertukaran tugas yang sudah diberikan.

3.7.2 Membuat *Unit Test Case*

Pada tahap ini tim pengembang mulai mengembangkan *unit test case* sesuai dengan *requirement* yang sudah ditentukan pada saat *envisioning*. *Unit testing* dilakukan secara otomatis menggunakan laravel phpunit. Karena merupakan bagian dari *framework* laravel, *test*

case yang ada sudah langsung tersambung dengan kode yang ada di proyek yang sama, yang membuat kerja tim pengembang menjadi lebih cepat. *Unit test case* ditunjukkan pada gambar 3.6, 3.7, dan 3.8.

Gambar 3.6 *Unit test case* 1

Pada gambar 3.6 merupakan *unit test case* yang digunakan untuk menguji fungsi *store* dimana pengguna dengan *role customer* melakukan pemesanan lisensi dan data terkait pesanan tersebut akan tercatat di basis data.

```
/**
* @test
*/
public function it_can_edit_profile()
{
    $this->visitRoute('loginForm');
    $this->seePageIs('/');

    $this->submitForm('Login', [
        'email'    => 'adeliajf@gmail.com',
        'password' => 'adelia130301',
    ]);
    $this->seePageIs('/admin')
         ->visitRoute('profile')
         ->visit('/account/edit/13')
         ->type('Adelia Julia Febrianti', 'name')
         ->type('adeliajf@gmail.com','email')
         ->type('Jl Umbul Permai 19','alamat')
         ->type('081376458361','phone_number')
         ->type(653645, 'pin')
         ->press('Save')
         ->seeRouteIs('profile');
}
```

Gambar 3.7 *Unit test case* 2

Pada *unit test case 2* menunjukkan sebuah pengujian dengan skenario pengguna sedang melakukan perubahan atau pengisian data profil dari akun yang dimiliki, yang diakhiri dengan perubahan data dan laman akhir yang seharusnya muncul setelah proses selesai, dalam kasus ini halaman akhir setelah proses selesai adalah halaman *profile* dari akun tersebut.

```php
/**
* @test
*/
public function it_create_reseller()
{
    $randompin = mt_rand(100000,999999);
    $user = [
    'name' => 'Amat',
    'email' => 'amat@gmail.com',
    'password' => Hash::make('12345678'),
    'pin' => $randompin,
    'role' => 'reseller',
];
$response = $this->post(route('reseller.store'), $user);
$response->assertResponseStatus(302);
}

/**
* @test
*/
public function it__fail_create_reseller()
{
    $randompin = mt_rand(100000,999999);
    $user = [
    'name' => 'Amanah',
    'email' => 'amat1@gmail.com',
    'password' => Hash::make('1234567'),
    'pin' => $randompin,
    'role' => 'reseller',
];
$response = $this->post(route('reseller.store'), $user);
$response->assertResponseStatus(302);
}
```

Gambar 3.8 *Unit test case* 3

Pada *unit test case 3* berisi 2 *unit test case* yaitu pengguna berhasil mendaftarkan diri sebagai reseller dan pengguna gagal mendaftarkan diri, kondisi gagal dan berhasil didapatkan dari pengisian data *password* karena syarat minimal *password* terdiri dari 8 karakter.

3.7.3 Menuliskan Kode

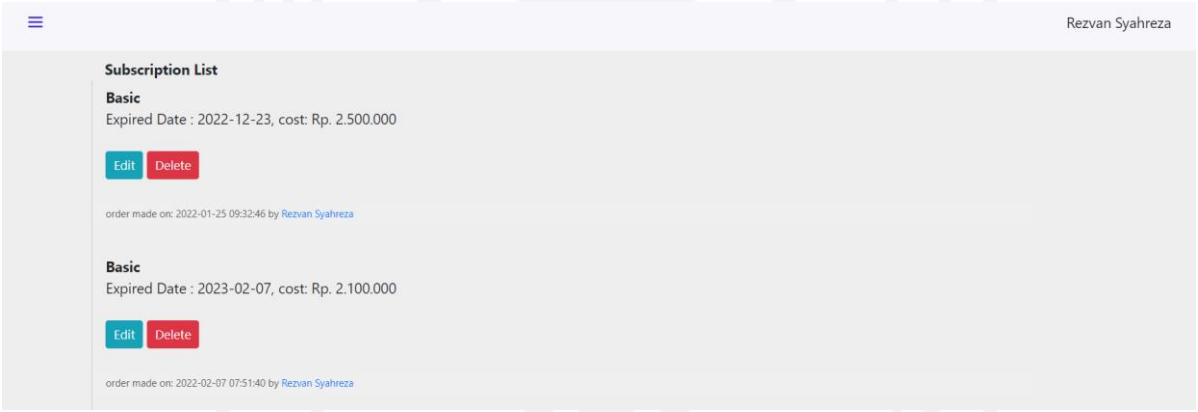
Pada fase ini pengembang mulai mengembangkan kode berdasarkan *unit test case* yang sudah dibuat, penyelesaian kode dilakukan berdasarkan pembagian tugas dan prioritas fungsi yang sudah ditentukan pada saat *priority modeling*. Penulisan kode awal dibuat seminimal mungkin untuk memastikan bahwa fungsi lulus pengujian dan hanya menambahkan kode baru saat kode awal dinyatakan gagal atau tidak lulus uji, hal ini dilakukan untuk meminimalisir adanya duplikasi kode. Pada dasarnya penambahan kode baru yang terjadi saat kode awal tidak lulus uji merupakan salah satu contoh dari kegiatan *refactoring*. Berikut merupakan kode fungsi dari fitur yang sudah dikembangkan ditunjukkan pada gambar 3.9, 3.11, dan 3.13.

```php
public function store(Request $request)
{
    $request -> validate([
        'subscription_id' => 'required',
        'user_id' => 'required',
        'chosen_feature' => 'required',
        'durasi' => 'required',
    ]);

    $current = Carbon::now();
    $expired = $current->addMonths($request->durasi);
    $history_id = DB::table('histories')->max('subscription_id');
    $price = DB::table('features')->where('feature_id',$request->chosen_feature)->value('feature_price');
    $extradev = $request->extra_device * 100000;
    if($request->durasi < 12){
        $total_price = $price * $request->durasi + $extradev * $request->durasi;
    }elseif ($request->durasi = 12) {
        $total_price = $price * 10 + $extradev * $request->durasi;
    }
    subscription::create([
        'subscription_id' => $request->subscription_id,
        'user_id' => $request->user_id,
        'feature_id' => $request->chosen_feature,
        'chosen_feature' => $request->chosen_feature,
        'total_price' => $total_price,
        'expire_date' => $expired,
        'extra_device' => $request->extra_device,
        'status' => 0,
    ]);

    return redirect()->route('customer');
}
```

Gambar 3.9 Fungsi *store* yang dimiliki *customer*



Gambar 3.10 List pesanan lisensi

Gambar 3.9 menunjukkan kode untuk melakukan penyimpanan data pesanan lisensi milik *customer* dengan status 0 (belum dilakukan pembayaran), pada fungsi ini sistem akan mendeteksi data yang masuk melalui *input* yang dilakukan oleh pengguna, terdapat 4 variabel yang memiliki status *required* yang menandakan 4 variabel tersebut wajib memiliki nilai, dengan menggunakan status *required* yang sudah disediakan oleh laravel, sistem akan langsung mengeluarkan notifikasi jika salah satu dari *required input* tidak diisi oleh pengguna. Hasil dari data yang dinputkan akan dimasukkan kedalam list pemesanan lisensi yang ditunjukkan pada gambar 3.10.

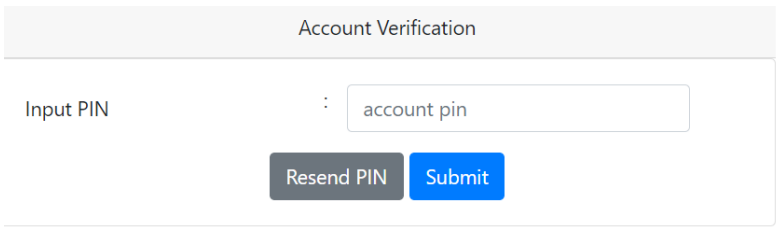
Dibawah ini merupakan pseudocode dari gambar 3.9

```
1.  // Menyimpan data pemesanan
2.
3.  //input
4.  history_id = 410 // id yang tersimpan pada history pemesanan
5.  price = 150000 // harga lisensi
6.  extradev = 2 * 100000 // 100000 = harga per device
7.  durasi = 1 // lama berlakunya lisensi
8.
9.  // info pengguna
10. subscription_id = 410 // id subscription (auto generate)
11. user_id = 5432 // id logged user
12. feature_id = 11 // tipe lisensi yang dibeli (basic, standard, erp)
13. extra_device = 2
14. status = 0
15.
16. if(durasi < 12)
17.     total_price = price * durasi + extradev * durasi
18. ElseIf(durasi = 12)
19.     total_price = price * 10 + extradev * durasi
20.     Endif
21.     output total_price
22.
23. subscription::create([datas]) // fungsi write dari Laravel (untuk menulis data baru ke db)
24. Return redirect()->route('customer') // mengarahkan ke homepage customer
```

```php
public function verifyAccount(Request $request)
{
    $this->validate($request, [
        'pin' => 'required|min:6',
    ]);

    if($request->pin != auth()->user()->pin)
    {
        return redirect()->back()->with('error', 'Wrong PIN Number');
    }
    else{
        $user = User::findOrFail(auth()->user()->user_id);
        $user->status = 1;
        $user->update();
        if(auth()->user()->role == 'reseller'){
            return redirect()->route('reseller');
        }
        elseif (auth()->user()->role == 'customer') {
            return redirect()->route('customer');
        }
        else{
            return redirect()->route('admin');
        }
    }
}
```

Gambar 3.11 Verifikasi akun baru



Gambar 3.12 Tampilan verifikasi akun (tampilan pada awal pengembangan)

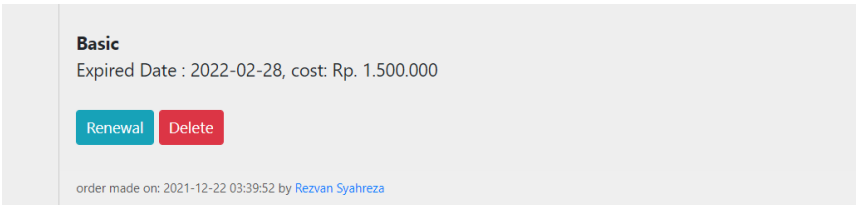
Dibawah ini merupakan pseudocode dari gambar 3.11

```
1.  // verifikasi akun
2.
3.  //info pengguna
4.  userpin = 123456
5.  userstatus = 0
6.  role = "reseller"
7.
8.  //input pengguna
9.
10. input pin
11.
12. if(pin != userpin)
13.     output "error pin anda salah" // notifikasi yang akan keluar di layar
14. else
15.     userstatus = 1
16.     output "akun telah berhasil di aktifkan" // notifikasi yang akan keluar di layar
17.     if(role == "reseller")
18.     return redirect()->route("reseller")
19.     else if(role == "customer")
20.     return redirect()->route("customer")
21.     else if(role == "admin")
22.     return redirect()->route("admin")
23. end if
```

Gambar 3.11 menunjukkan kode untuk melakukan verifikasi akun, verifikasi akun dilakukan pada akun yang baru terbuat, dimana verifikasi dilakukan dengan menggunakan nomor pin sebanyak 6 angka yang akan dikirimkan ke dalam email yang pengguna gunakan pada saat melakukan pendaftaran. Pin hanya dimasukkan sekali, pada saat pengguna pertama kali login menggunakan akun baru tersebut. Tampilan verifikasi ditunjukkan pada gambar 3.12.

```php
public function showAlmostExpireSubs()
{
    $current_date = Carbon::now();
    $maxDate = $current_date->addDays(14);
    $subscriptions = subscription::where('user_id', '=', auth()->user()->user_id)
        ->orderBy('expire_date', 'asc')
        ->whereDate('expire_date', '>=', Carbon::now())
        ->whereDate('expire_date', '<=', $maxDate)
        ->get();
    return view('customer', compact('subscriptions'));
}
```

Gambar 3.13 *Almost expire subscription*



Gambar 3.14 Lisensi yang hampir habis masa berlakunya

Gambar 3.13 menunjukkan sebuah fungsi yang berisikan kode yang akan memunculkan lisensi yang masa berlakunya kurang dari 14 hari atau sudah akan habis, dimana hasilnya akan di perlihatkan dalam bentuk *list* lisensi yang ditunjukkan pada gambar 3.14.

Karena pada pengerjaan proyeknya menggunakan *framework* laravel, koneksi kode pada proyek dengan data pada *database* menjadi lebih mudah dengan menerapkan laravel *active record* atau *eloquent object relational mapper*, dimana *controller* dapat mengambil data dengan memanggil *model* yang telah terkoneksi dengan *database*, terdapat beberapa aturan penamaan yang dapat memudahkan sambungan *model* dan *database* yaitu penamaan *model* menggunakan aturan bahasa inggris dimana jika model diberi nama *history* maka tabel di *database* diberi nama *histories*, jika penamaan menggunakan Bahasa lain atau singkatan dari nama tabel pada *database*, perlu diberikan inisialisasi bahwa *model* berhubungan dengan data tertentu di *database*. Pemanfaatan *active record* mempersingkat penulisan *query* sehingga membuat kode lebih mudah untuk dimengerti. Pada gambar 3.15 menunjukkan pemanfaatan *active record* sedangkan gambar 3.16 menunjukkan hasil tampilan yang didapat dari fungsi *index* pada gambar 3.15.

```php
public function index()
{
    $subscriptions = subscription::where('status','=','1')->get();
    return view('customer', compact('subscriptions'));
}
```

Gambar 3.15 Fungsi *index*

Gambar 3.16 Tampilan dari *index function*

```php
public function create()
{
    $subscriptions = subscription::all();
    $features = feature::all();
    return view('subscriptions.create', compact('subscriptions','features'));
}
```

Gambar 3.17 Fungsi *create*

Pada gambar 3.15 penulisan *query* lebih singkat karena *active record* menangani hubungan *model* dan *database* secara otomatis sehingga kita hanya perlu memanggil *model* yang dibutuhkan, seperti pada fungsi *index* sistem akan mengambil semua data pada tabel *subscriptions* di *database*, dengan ciri nilai status bernilai 1. Pada gambar 3.17 merupakan fungsi yang digunakan untuk mengambil data pada tabel *subscriptions* dan *features* yang nantinya akan digunakan sebagai isi dari variabel pada *form* pembelian lisensi. Penulisan kode yang sederhana membuat kode menjadi lebih mudah dibaca dan dimengerti sehingga *maintainability* mengalami peningkatan, penulisan kode seperti ini dapat dilakukan dengan menggunakan *framework* laravel.

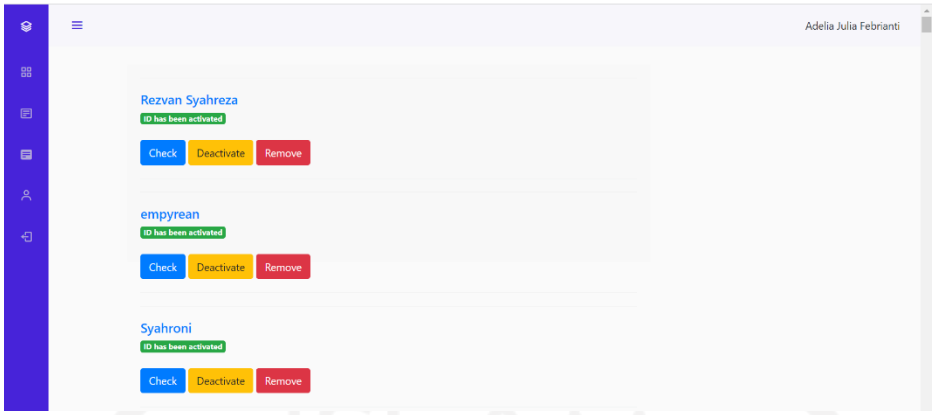
```php
public function impersonate(User $user, Request $request)
{
    auth()->user()->impersonate($user);
    if($user->role == 'reseller')
    {
        return redirect()->route('reseller');
    }
    else{
        return redirect()->route('customer');
    }
}
```

Gambar 3.18 *Impersonating*

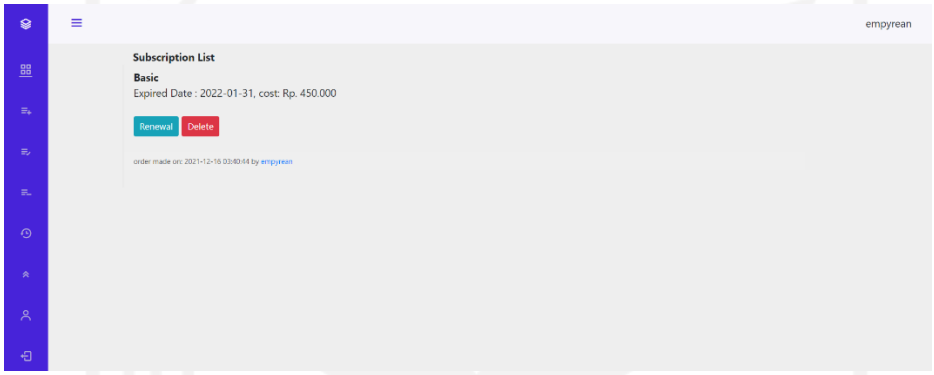
```php
//Checkout Customer
public function leaveImpersonate()
{
    auth()->user()->leaveImpersonation();
    if(auth()->user()->role == 'reseller'){
        return redirect()->route('reseller');
    }
    else{
        return redirect()->route('admin');
    }

}
```

Gambar 3.19 *Leave impersonating*



Gambar 3.20 Tampilan sebelum *impersonating*



Gambar 3.21 Tampilan setelah *impersonating*

Pada gambar 3.18 dan 3.19 menunjukkan fungsi *impersonate* atau *impersonating*, fungsi ini digunakan agar akun tertentu dapat melakukan aktivitas pada akun lain. Pada proyek ini, fungsi *impersonate* digunakan oleh pengguna dengan *role* sebagai *reseller* dan *admin*, dimana *reseller* ataupun *admin* dapat beraktivitas sebagai *customer* yang terdaftar sebagai pelanggan dari *reseller* atau *admin* tersebut. *Impersonate* merupakan salah satu laravel *package* yang dapat ditambahkan ke *folder vendor* pada *folder* proyek Laravel. Pada gambar 3.20 ditunjukkan suatu skenario dimana *admin* yang memiliki *username* Adelia ingin melakukan *impersonating* (dengan menekan tombol *check*) pada akun *customer* dengan *username* Empyrean, dimana *admin* akan mendapatkan akses kepada *customer* yang dipilih yang diperlihatkan pada gambar 3.21.

Selain pemrograman *back-end*, terdapat beberapa penugasan *front-end* yang diberikan kepada penulis, penugasan ini terdiri dari pembuatan *cascading style sheets* atau yang lebih dikenal dengan css dan pembuatan beberapa tampilan halaman *website*. Beberapa kode terkait *front-end* ditunjukkan pada gambar 3.22 dan 3.23.

Gambar 3.22 CSS

Gambar 3.22 menunjukkan beberapa kode CSS yang dipergunakan untuk menampilkan halaman pemilihan tipe lisensi.

```
<div class="container">
<div class="mt-5">
    @foreach($features as $f)
      <div class="d-style btn btn-brc-tp border-2 bgc-white btn-outline-blue btn-h-outline-blue btn-a-outline-blue w-100 my-2 py-3 shadow-sm">
        <div class="row align-items-center">
          <div class="col-12 col-md-6">
            <h4 class="pt-3 text-170 text-600 text-primary-d1 letter-spacing">
              {{$f->feature_name}}
            </h4>

            <div class="text-secondary-d1 text-120">
              <span class="ml-n15 align-text-bottom text-180">@currency($f -> feature_price)</span> / month
            </div>
          </div>
          <div class="col-12 col-md-4 text-center align-items-center" >
            <a href="{{ route('subscriptions.buySubs',$f->feature_id) }}" class="f-n-hover btn btn-info btn-raised px-4 py-25 w-75 text-600">Get Started</a>
          </div>
        </div>
      </div>
      @endforeach
</div>
```

Gambar 3.23 *View* pemilihan tipe lisensi

Gambar 3.23 menunjukkan kode yang terdapat pada blade.php yang merupakan bagian *view* atau halaman *website* akan ditampilkan, hasil tampilan dari gambar 3.23 dapat dilihat pada bagian hasil dan pembahasan di gambar 3.39. Terdapat beberapa penugasan *front-end* lainnya yang diterima diantaranya :

a. Pembuatan CSS *sidebar*.
b. Pembuatan dashboard *admin*, *reseller*, dan *customer*.
c. Pembuatan halaman *profile account* dan *edit profile*.

d. Pembuatan halaman *history* pembelian lisensi.

e. Pembuatan halaman pembelian lisensi.

Selain penugasan *front end* dan pembuatan fungsi-fungsi pada fitur *website*, juga memiliki tanggung jawab dalam menuliskan beberapa *middleware*, serta pembuatan *observer* serta beberapa notifikasi menggunakan email yaitu *password reset notification* dan *new user notification*. Laravel *observer* merupakan suatu fungsi yang aktif secara otomatis pada saat terjadi *event* khusus, pada pengembangan ini *observer* dikombinasikan dengan fungsi *email notification*, dimana *observer* akan mengirimkan *email notification* berisikan pin setiap terdapat pengguna baru yang terdaftar. Selain penggunaan *observer* yang dikombinasikan dengan *email notification*, terdapat fungsi lain yang menggunakan observer di dalam operasionalnya diantaranya adalah ketika pengguna gagal melakukan login sebanyak 3 kali, maka observer akan mengeluarkan google captcha, begitu pula saat pengguna ingin melakukan pemesanan dan pembayaran. Fungsi dari *observer* dan *email notification* ditunjukkan pada gambar 3.24 dan 3.25. Tampilan dari gambar 3.24 dan 3.25 ditunjukkan pada gambar 3.31.

```
class UserObserver
{
    /**
     * Handle the User "created" event.
     *
     * @param  \App\Models\User  $user
     * @return void
     */
    public function created(User $user)
    {
        $user->notify(new WelcomeEmailNotification($user));
    }
```

Gambar 3.24 *Observer*

```
public function toMail($notifiable)
{
    return (new MailMessage)
                ->line('This is your private pin for activate your account')
                ->line('pin : '.$this->user->pin)
                ->action('Go to website', url('/'))
                ->line(Lang::get('This PIN will expire in :count minutes.', ['count' => config('auth.passwords.'.config('auth.defaults.passwords').'.expire')]))
                ->line('Thank you for using our application!');
}
```

Gambar 3.25 *Welcome email notification*

Untuk penerapan *middleware*, terdapat 2 *middleware* yang telah dikembangkan, *Auth Check* dan *Check Role*, yang diperlihatkan pada gambar 3.26 dan 3.27.

```php
public function handle(Request $request, Closure $next)
{
    $roles = array_slice(func_get_args(), 2);

    foreach ($roles as $role) {
        $user = \Auth::user()->role;
        if( $user == $role){
            return $next($request);
        }
    }
return redirect()->back()->with('warning', 'anda tidak dapat mengakses konten ini');
}
```

Gambar 3.26 *Middleware check role*

```php
public function handle(Request $request, Closure $next)
{
    if (!session()->has('LoggedUser') && ($request->path() != '/' && $request->path() != 'login' && $request->path() != 'register/reseller' && $request->path() != '
        register/admin' && $request->path() != 'register/customer')) {
        return redirect('/')->with('error','Harus login terlebih dahulu');
    }
    elseif (session()->has('LoggedUser') && ($request->path() == '/' || $request->path() == 'login' || $request->path() == 'register/admin' || $request->path() == '
        register/customer' || $request->path() == 'register/reseller')) {
        return redirect()->back();
    }
    return $next($request)->header('Cache-Control','no-cache, no-store, max-age=0, must-revalidate')
                          ->header('Pragma','no-cache')
                          ->header('Expires','Sat 01 Jan 1990 00:00:00 GMT');
}
```

Gambar 3.27 *Middleware auth check*

Middleware check role digunakan sistem untuk mengarahkan pengguna ke *dashboard* sesuai dengan *role* masing-masing, karena pada pembuatan *dashboard*, *dashboard* hanya memiliki 1 *view* yang akan menampilkan 3 halaman dengan informasi berbeda sehingga *middleware* dikembangkan. Sedangkan *auth check* digunakan untuk mencegah pengguna yang sudah melakukan login tidak dapat kembali ke halaman login baik dengan mengetikkan url maupun menekan *back button* yang terdapat pada browser.

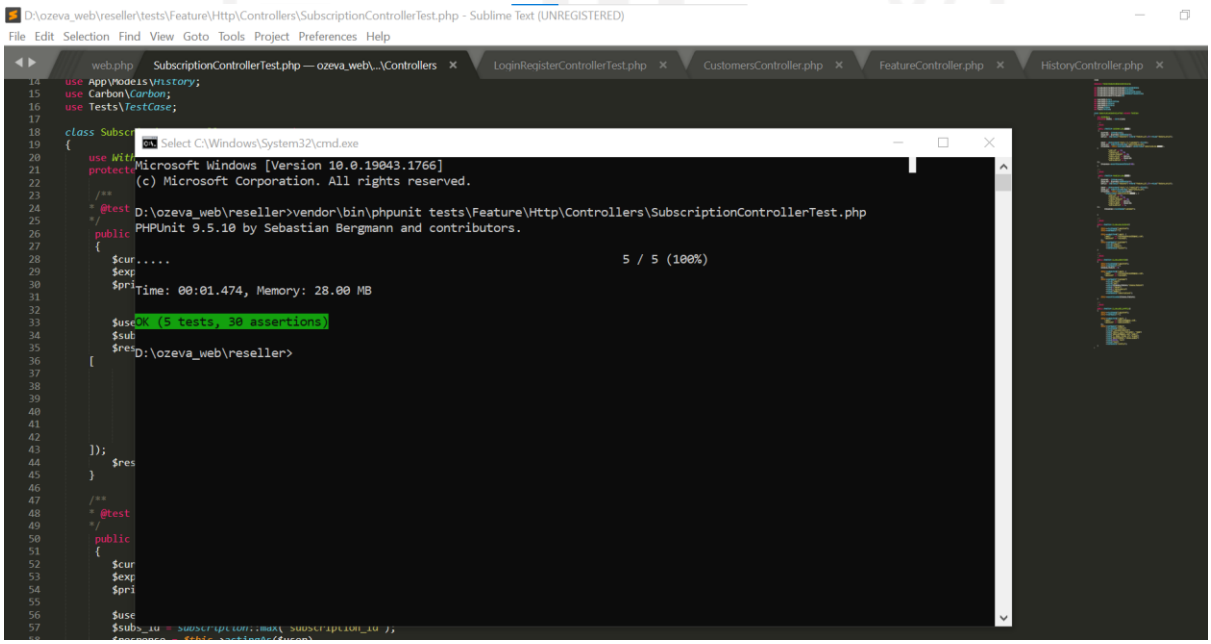
```php
Route::group(['middleware'=>['AuthCheck']], function(){
    Route::get('/', [LoginController::class, 'showLoginForm'])->name('loginForm');
    Route::get('/register/admin', [RegisterController::class,'showAdminRegisterForm'])->name('register.admin');
    Route::get('/register/customer', [RegisterController::class,'showCustomerRegisterForm'])->name('register.customer');
    Route::get('/reseller/register/customer', [RegisterController::class,'showCustomerRegisterForm'])->middleware(['checkRole:reseller,admin'])->name('reseller.customer');
    Route::get('/register/reseller', [RegisterController::class,'showResellerRegisterForm'])->name('register.reseller');
    Route::get('/admin', function () { return view('admin'); })->middleware(['checkRole:admin'])->name('admin');
    Route::get('/customer', function () { return view('customer'); })->middleware(['checkRole:customer'])->name('customer');
    Route::get('/reseller', function () { return view('reseller'); })->middleware(['checkRole:reseller'])->name('reseller');
```

Gambar 3.28 Penggunaan *authcheck*

Dibawah ini merupakan pseudocode dari gambar 3.28

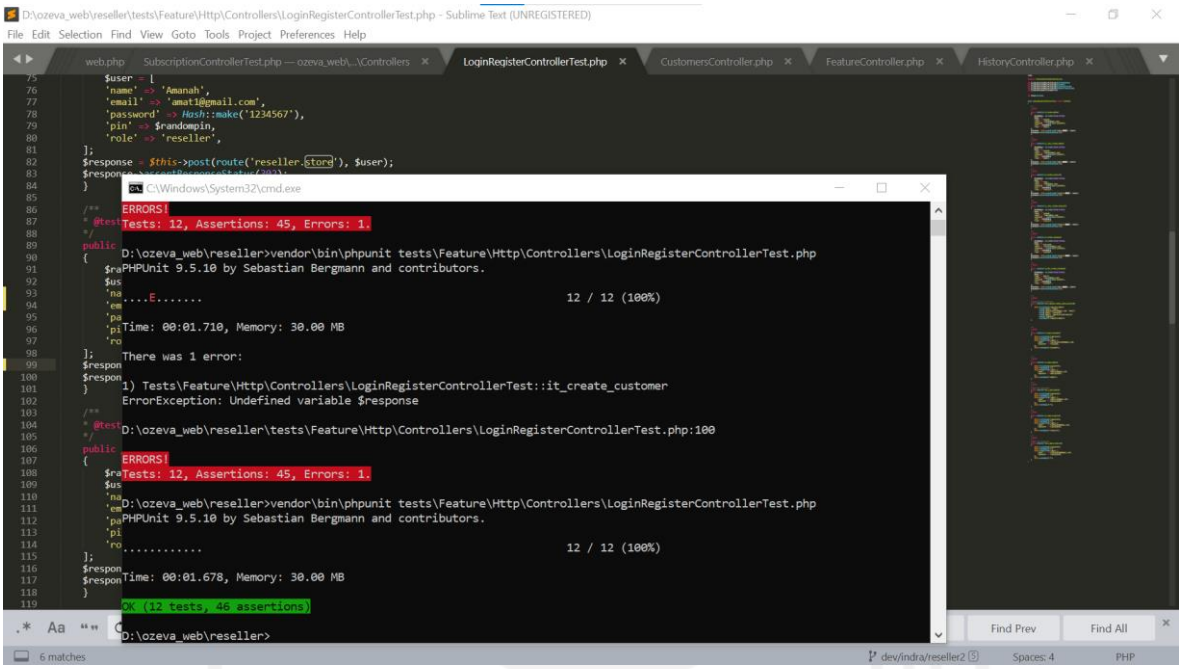
```
1.  Route::group(['middleware'=>['AuthCheck']], function(){
2.      Route::get('/', [loginController::class,'showLoginForm'])->name('loginForm');
3.      Route::get('/register/admin',
    [RegisterController::class,'showAdminregisterForm'])->name('loginForm');
4.      Route::get('/register/customer',
    [RegisterController::class,'showCustomerRegisterForm'])->name('loginForm');
5.      Route::get('/reseller/register/customer',
    [RegisterController::class,'showCustomerRegisterForm'])->name('loginForm');
6.      Route::get('/register/reseller',
    [RegisterController::class,'showResellerRegisterForm'])->name('loginForm');
7.      Route::get('/admin',       function()       {return       view('admin');       })-
    >middleware(['checkRole:admin'])->name('admin');
8.      Route::get('/customer',     function()     {return     view('customer');     })-
    >middleware(['checkRole:customer'])->name('customer');
9.      Route::get('/reseller',     function()     {return     view('reseller');     })-
    >middleware(['checkRole:reseller'])->name('reseller');
10. })
```

Penggunaan *middleware* dilakukan seperti pada gambar 3.28, dimana terdapat beberapa fitur yang menerima efek dari penggunaan *middleware.* Setelah penulisan kode selesai, *test case* yang telah dikembangkan dijalankan kembali untuk menguji apakah seluruh *unit test case* yang dikembangkan telah lulus uji, jika *unit test case* lulus pengujian maka pengembang mengirimkan kode yang sudah diujikan ke GIT *branch* menggunakan TortoiseGit, jika masih ditemukan *error* maka *refactoring* dilakukan. Hasil pengujian *unit test case* ditunjukkan pada gambar 3.29 dan 3.30.



Gambar 3.29 S*ubscription controller test*

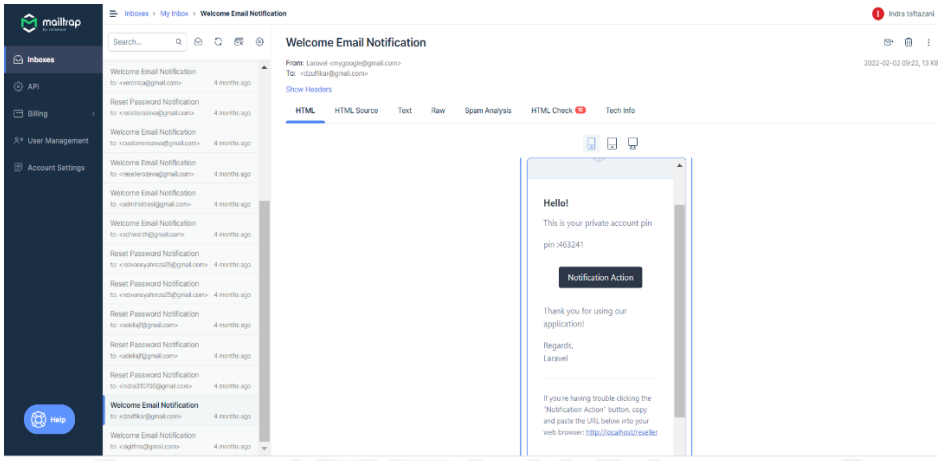
Pada gambar 3.29, diperlihatkan bahwa pengujian pada *subscription controller test case* berhasil diuji tanpa mendapatkan *error* maupun *bug*, dimana terdapat 5 *unit test case* yang berisikan 30 pernyataan yang secara keseluruhan sudah lulus uji yang ditunjukkan dengan indikator warna hijau, sehingga tidak perlu dilakukan *refactoring*.



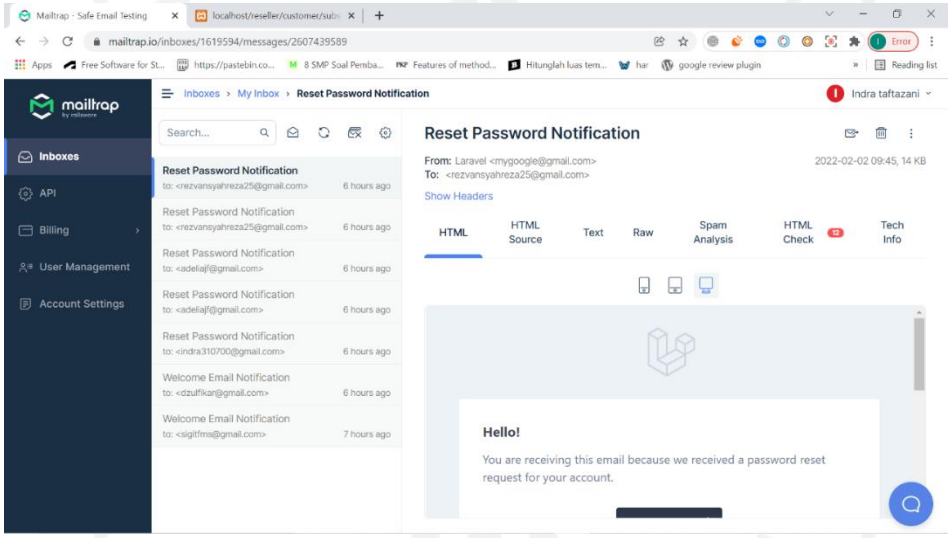
Gambar 3.30 *Login register controller test*

Pada gambar 3.30, pengujian berisikan 12 *test case* berbeda yang memiliki 46 pernyataan didalamnya, akan tetapi pada saat melakukan pengujian ditemukan *error* pada *it_create_customer test case* yang ditunjukkan dengan indikator berwarna merah, sehingga perlu dilakukan *refactoring* hingga pengujian berhasil secara keseluruhan tanpa ditemukan *error* dan *bug*.

Terdapat beberapa pengujian yang tidak dilakukan menggunakan laravel *unit test case*, melainkan menggunakan mailtrap, beberapa diantaranya adalah pengujian terhadap pengiriman pin pengguna baru dan pengiriman alamat reset password yang dikirimkan melalui email ke akun pengguna. Penggunaan mailtrap ditunjukkan pada gambar 3.31 dan 3.32.

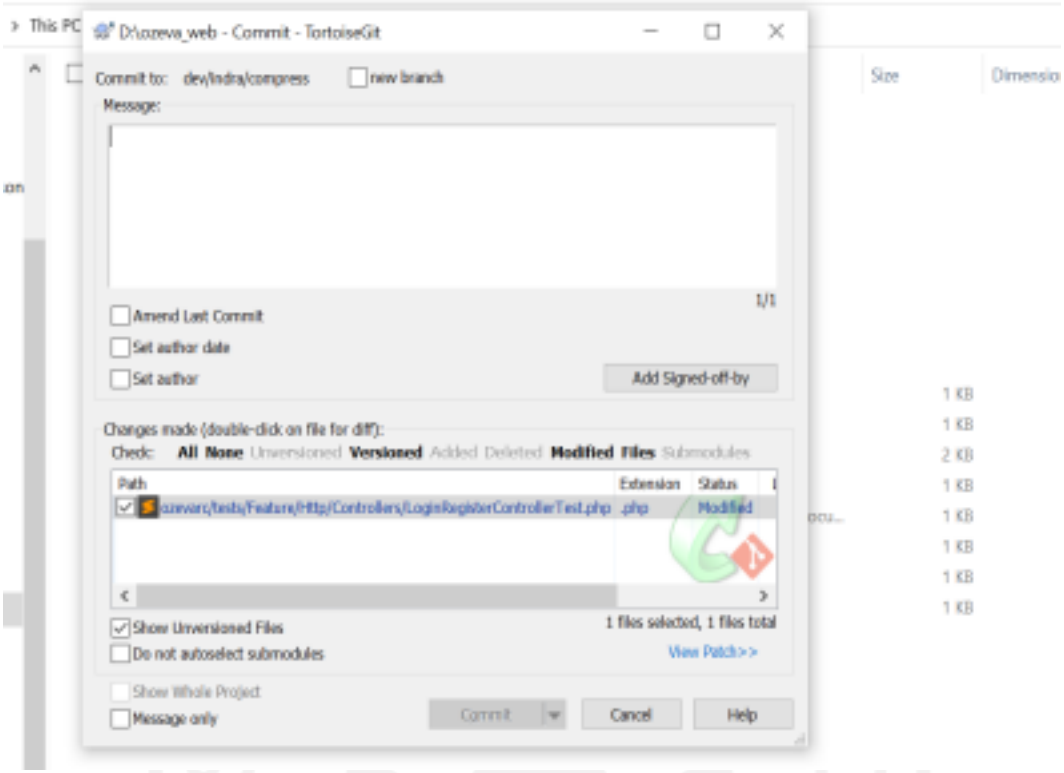


Gambar 3.31 *Welcome email notification*



Gambar 3.32 *Reset password notification*

Setelah seluruh *test case* berhasil diujikan, pengembang melakukan *push* dan *commit* perubahan data ke *project branch.* Jika *push* dan *commit* data dilakukan sebelum hari rabu, terdapat kemungkinan dimana supervisor memberikan *approval* atau permintaan *refactoring* pada minggu yang sama tanpa harus menunggu permintaan pada saat rapat mingguan pada hari senin. *TortoiseGit commit* ditunjukkan pada gambar 3.33.



Gambar 3.33 *Commit* kode baru

3.7.4 Melakukan *Refactoring*

Pada tahap *refactoring*, tim pengembang akan berdiskusi terkait kode yang sudah dikembangkan olehnya pada pekerjaan minggu lalu dengan anggota tim lain, untuk mengetahui apakah kode yang dituliskan sudah sesuai standar yang ada atau belum, pengembang lain dapat memberi saran kepada pengembang lainnya jika kode yang dibuat dapat dibuat semakin sederhana, atau mengganti beberapa fungsi agar kecepatan yang didapat semakin cepat, dan jika *refactoring* dilakukan, pengujian dan diskusi terkait hasil *refactoring* dilakukan lagi pada pertemuan berikutnya. Selain dari *refactoring* terdapat kondisi dimana kode yang sudah diujikan perlu ditambahkan atau diubah, diantaranya terdapat fungsi baru yang berhubungan dengan kode tersebut, sehingga terjadi penambahan kode, saat hal ini terjadi pengujian dilakukan kembali pada kode yang mengalami perubahan. Penambahan kode ditunjukkan pada gambar 3.34.

```php
public function store(Request $request)
{
    $request -> validate([
        'subscription_id' => 'required',
        'user_id' => 'required',
        'chosen_feature' => 'required',
        'durasi' => 'required',
    ]);

    $current = Carbon::now();
    $expired = $current->addMonths($request->durasi);
    $history_id = DB::table('histories')->max('subscription_id');
    $price = DB::table('features')->where('feature_id',$request->chosen_feature)->value('feature_price');
    $extradev = $request->extra_device * 100000;
    if($request->durasi < 12){
        $total_price = $price * $request->durasi + $extradev * $request->durasi;
    }elseif ($request->durasi = 12) {
        $total_price = $price * 10 + $extradev * $request->durasi;
    }
    subscription::create([
        'subscription_id' => $request->subscription_id,
        'user_id' => $request->user_id,
        'feature_id' => $request->chosen_feature,
        'chosen_feature' => $request->chosen_feature,
        'total_price' => $total_price,
        'expire_date' => $expired,
        'extra_device' => $request->extra_device,
        'status' => 0,
    ]);
    if(!session('impersonated_by')){
        $user = User::findOrFail($request->user_id);
        $user->point = $user->point + 10;
        $user->update();
    }
    else{
        $customer = User::findOrFail($request->user_id);
        $user = User::findOrFail($customer->created_by);
        $user->point = $user->point + 10;
        $user->update();
    }
    return redirect()->route('customer');
}
```

Gambar 3.34 Penambahan kode karena fungsi lain

Pada gambar 3.34, terjadi penambahan kode dikarenakan terdapat fungsi *impersonate*, perubahan ini terjadi untuk mengisi fitur *reseller ranking*, dimana pada saat *customer* melakukan pembelian lisensi melalui *reseller*, *reseller* mendapatkan bonus *ranking point*.

3.8 Pengujian Akhir

Pada praktiknya pengembangan *website* metode *test driven development*, mengurangi kemungkinan *bug* pada *output* yang dihasilkan akan tetapi ada kemungkinan ditemukannya kasus *error* dan *bug*, oleh karena penting dilakukannya pengujian akhir atau evaluasi. Evaluasi dilakukan menggunakan metode *black box testing*.

Black box testing perlu dilakukan untuk memastikan apakah setiap fungsionalitas dari fitur bekerja sesuai spesifikasi yang diinginkan. Selain itu, karena pengembangan berdasarkan *test driven development* melakukan pengujian dari sudut pandang pengembang, maka diperlukan serangkaian pengujian yang memperhatikan sistem dari perspektif pengguna *website*, *black box testing* melakukan pengujian kepada keseluruhan sistem berbeda dengan test driven development yang fokus pengujiannya dilakukan per *unit* dan modul. Sehingga *black box testing* sangat cocok dilakukan sebagai evaluasi akhir dimana pengujiannya mengisi

hal–hal yang tidak dicakup pada saat melakukan pengujian menggunakan metode *test driven development.*

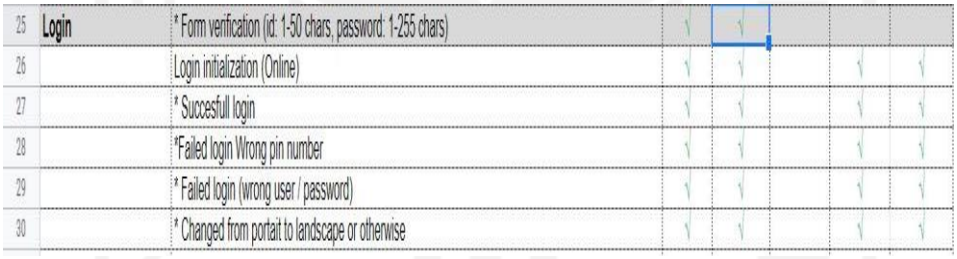
Black box testing yang dilakukan adalah *functionality testing* dan *non-functionality testing* dengan total 120 *test case*, jika *test case* sudah disetujui penguji akan menjalankan pengujian akhir. Pengujian dilakukan oleh 5 orang penguji yang terdiri dari 3 orang dari tim penguji dan 2 orang calon pengguna *website*. Pengujian dilakukan pada *platform desktop* dan *mobile* dengan *test case* yang sama, yang membedakan keduanya adalah pada pengujian *website* di *platform mobile* terdapat pengujian tambahan yaitu setiap *test case* dilakukan dalam 2 posisi layar berbeda *portrait* dan *landscape*, pengujian juga dilakukan dengan beberapa *smartphone* dengan ukuran layar yang berbeda-beda, dan pada pengujian di *platform desktop,* pengujian dilakukan menggunakan *3* browser berbeda yaitu *google chrome, mozilla*, dan *safari.* Hal ini dilakukan untuk memastikan kemudahan akses *website* dan kerapian tampilan w*ebsite*. Semua *error* dan *bug* yang ditemukan akan didokumentasikan ke dalam sebuah dokumen excel, yang Bernama *bugs report.* Setelah pengujian akhir dilakukan, pengembang akan melakukan *bug fixing* kepada semua *bug* yang dituliskan pada *bugs report* tersebut, dan dilanjutkan dengan melakukan *regression testing*. *Regression testing* digunakan untuk mencari tau dampak yang didapat dari penambahan kode setelah dilakukan *bug fixing*, jika terjadi penurunan performa pada *website*, maka fitur-fitur yang performanya mengalami penurunan perlu untuk di *refactoring* kembali. Beberapa *test case* dipaparkan pada tabel 3.2.

Tabel 3.2 Beberapa *test case* yang sudah dikembangkan

NO	***Test Case***	**Hasil yang diharapkan**	**Hasil yang diperoleh**
1	Langsung menuju form pembelian tanpa memilih aplikasi atau jenis lisensinya	Laman akan ke redirect ke halaman yang sebelumnya terbuka	Berhasil
2	Tidak mengisi durasi lisensi	Sistem akan menampilkan peringatan “this field is required ”	Berhasil
3	Mengisi durasi lisensi dengan angka 0 atau karakter	Sistem akan menampilkan peringatan “this field is must be fill with valid number”	Sistem tetap menyimpan data pembelian lisensi (Gagal)
4	Langsung menuju form pembelian tanpa melakukan login terlebih dahulu	Laman akan ke redirect ke halaman login, dan sistem akan menampilkan peringatan “Pengguna diharuskan melakukan login terlebih dahulu”	Berhasil
5	Melakukan Pembelian menggunakan sistem *check-in customer* oleh *reseller*	Pembelian berhasil dan email akan terkirim ke email customer yang digunakan dimana didalamnya terdapat informasi lisensi yang dibeli, serta jumlah dan nomor tagihan.	Berhasil

6	Menekan tombol navbar dalam kondisi portrait dan landscape di pada perangkat mobile	Navbar terbuka dan tombol terlihat dengan jelas	Pada saat membuka dalam kondisi portrait terdapat beberapa kalimat di navbar yang terpotong (gagal)
7	Admin atau Reseller keluar dari mode Check-IN customer	Kembali ke dashboard Admin atau Reseller sesuai dengan role masing-masing	Sistem menyatakan bahwa id telah terlogout dari sistem (Gagal)
8	Kesalahan login sebanyak 3 kali	Sistem akan mengeluarkan captcha pada usaha login selanjutnya	Berhasil
9	Melakukan reset pin dengan memasukkan angka pin sebanyak 5 digit	Sistem akan mengeluarkan peringatan “pin must be fill with 6 valid number”	Berhasil
10	Melakukan *password reset*	Sistem akan mengirimkan email konfirmasi terkait permintaan *reset password* serta link *reset password*	Berhasil

Selain melalui tabel 3.2, *test case* pengujian akhir juga ditunjukkan pada gambar 3.35 dan 3.36.



Gambar 3.35 Beberapa *test case* login

Gambar 3.36 *Test case payment*

3.9 Hasil dan Pembahasan

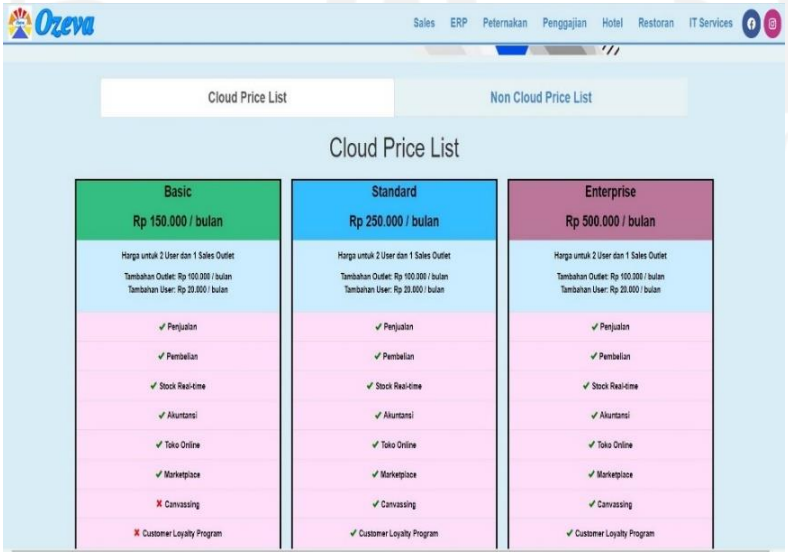
3.9.1 Hasil

Terdapat beberapa hasil pengembangan yang akan dipaparkan pada bagian ini seperti :

Homepage Ozeva Technology

Pada bagian *homepage* terdapat informasi terkait aplikasi yang diperjualkan lisensinya serta rincian fitur-fitur yang diterima, harga berlangganan serta cara pembelian dan instalasi aplikasinya. Tampilan *homepage* ditunjukkan pada gambar 3.37 dan 3.38.

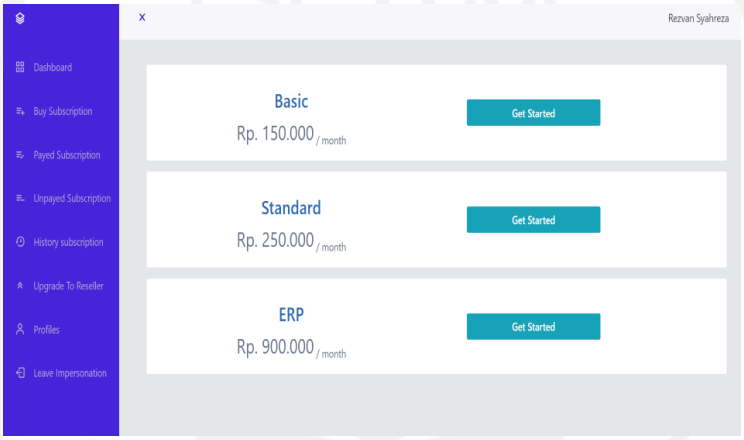
Gambar 3.37 *Homepage* ozeva



Gambar 3.38 Rincian informasi harga

Halaman Pembelian Lisensi

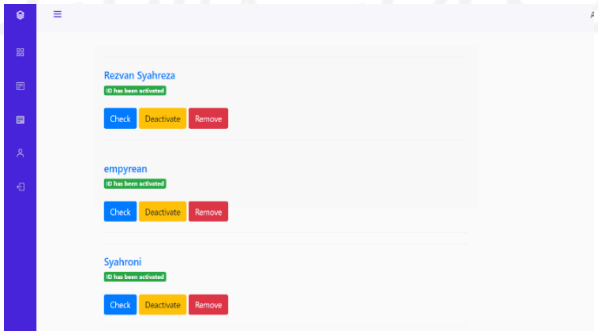
Pada saat pengguna melakukan pembelian lisensi, pengguna akan diarahkan untuk memilih license aplikasi yang akan dibeli serta jenis lisensinya yang terbagi menjadi beberapa kategori berdasarkan aplikasi yang dibeli, setelah selesai menentukan aplikasi dan jenis lisensinya, pengguna diarahkan ke laman pembelian dimana pengguna mengisi form untuk menentukan berapa lama pengguna ingin lisensi tersebut diaktifkan. Halaman pembelian lisensi ditunjukkan pada gambar 3.39.



Gambar 3.39 Pembelian lisensi

Halaman *Customer ID Management*

Terdapat beberapa kondisi unik dimana pengguna (*customer*) dapat meminta bantuan kepada pengguna khusus (pengguna dengan *role admin* serta *reseller*) untuk melakukan pembelian lisensi untuk dirinya. Maka dari itu *admin* dan *reseller* mempunyai kemampuan untuk masuk ke dalam akun *customer*. Selain masuk sebagai akun *customer*, *admin* dan *reseller* juga dapat menonaktifkan akun-akun *customer* yang sudah tidak aktif atau sudah tidak membeli lisensi lagi. Halaman *customer id management* ditunjukkan pada gambar 3.40.



Gambar 3.40 Halaman *customer id management*

Bugs Report

Setelah selesai dilakukannya pengujian akhir, temuan-temuan selama pengujian di tuliskan kedalam sebuah dokumen yang diberi nama *bugs report*. Jika *bugs report* sudah selesai didokumentasikan oleh tim penguji, dokumen akan dikirimkan kepada tim pengembang untuk dilakukan *bugs fixing*. Dokumen *bugs report* selain sebagai daftar fitur yang akan diperbaiki juga memiliki fungsi lain yaitu sebagai alat komunikasi dimana pada bagian *bugs report* terdapat kolom *fix result*, dimana jika terdapat sebuah *bug* selesai diperbaiki, anggota tim pengembang dapat menuliskan status '*done*' kedalam *fix result* yang menyatakan bahwa *bugs* sudah diperbaiki dan pengembang lain dapat mengambil *bug* yang lain. *Bugs report* ditunjukkan pada gambar 3.41.

Features	Scenario	GUI	Platform (M / D)	Result	Date	Fix Result	Remark
Pembelian lisensi	Mengisi durasi lisensi dengan angka 0 atau karakter	NO	M & D	X	02/02/2022		Data tetap tersimpan dalam database
Navbar	Menekan tombol Navbar pada keadaan portrait dan landscape	YES	M	X	03/02/2022		Pada saat membuka dalam kondisi portrait terdapat beberapa kalimat di navbar yang terpotong
Check-In Customer	Admin atau Reseller keluar dari mode Check-IN customer	NO	M & D	X	03/02/2022		ID terlogout dan admin harus melakukan login kembali
Register	Melakukan input nama > 50 karakter	NO	M & D	X	03/02/2022		Data tetap tersimpan meskipun nama melebihi 50 karakter
Edit Pemesanan Lisensi	Melakukan perubahan durasi lisensi	NO	M & D	X	05/02/2022		Perubahan data tersimpan tetapi penambahan harga tidak terubah
Profile	Membuka profil customer dalam keadaan check-in oleh reseller	NO	M & D	X	05/02/2022		Reseller berhasil masuk kedalam profile customer yang seharusnya tidak bisa

Gambar 3.41 *Bugs report*

3.9.2 Pembahasan

Kondisi Saat Pengembangan Berlangsung

Proyek berlangsung ditengah pandemi covid-19 sehingga terdapat beberapa kendala yang terjadi selama proyek berjalan, untuk mengatasi kendala tersebut dilakukan beberapa adaptasi yang dilakukan. Pada siklus *test driven development* 1 kondisi disekitar lokasi kantor dinyatakan sebagai zona merah sehingga pekerjaan terpaksa berlangsung secara *online*, namun pada praktiknya terdapat beberapa kendala yang dirasakan salah satunya berupa masalah koneksi, karena koneksi yang tidak terlalu bagus, terdapat beberapa informasi yang tidak jelas karena suara yang terputus-putus dan faktor lainnya, sehingga pengerjaannya melebihi batas waktu awal yang sudah ditentukan, akan tetapi terdapat beberapa hal yang tidak dapat dilakukan secara *online* sehingga karyawan kantor setidaknya harus pergi ke kantor setidaknya satu kali dalam seminggu, hal ini dikarenakan kantor yang menggunakan *tortoiseGit*, dimana karyawan perlu mengakses *local network* perusahaan untuk melakukan *updating*. Pada siklus

test driven development 2, kondisi pandemi di daerah kantor sudah mulai menurun sehingga kegiatan magang dapat dilakukan secara *full offline* dikantor dengan syarat karyawan sudah melakukan vaksin sebanyak 2 kali. Masalah pada *test driven development* 1 kembali terjadi pada siklus ke 3, karena sudah menjadi zona merah pengembangan dilakukan secara *online*, sebagai antisipasi masalah yang terjadi pada *test driven development* 1, durasi tim melaksanakan pertemuan diperpanjang sehingga pemaparan informasi serta diskusi yang dilakukan lebih jelas.

Siklus Test Driven Development

Seperti informasi yang terlihat pada tabel 1, pelaksanaan siklus *test driven development* terbagi menjadi 3 sesuai dengan modul pengguna.Pada siklus *test driven development* 1, pengembangan berfokus kepada keseluruhan fitur yang berhubungan dengan modul *customer* baik fitur umum (fitur yang digunakan oleh ke 3 modul) ataupun fitur khusus.

Fitur yang dihasilkan pada siklus *TDD* 1 ditunjukkan pada tabel 3.3.

Tabel 3.3 Siklus *test driven development 1*

NO	Fitur	Penjelasan
1	*Login & Sign In*	Fitur umum untuk melakukan login dan mendaftarkan akun
2	*Reset password*	Fitur umum untuk melakukan pengubahan password
3	*Customer Dashboard*	Fitur ini merupakan halaman awal yang muncul setelah login sebagai *customer* yang menampilkan lisensi yang dimiliki serta sisa waktu sebelum masa berlaku lisensi habis
4	Pembelian dan pembaruan lisensi aplikasi	Fitur untuk melakukan pembelian lisensi baru dan pembaruan lisensi yang sudah ada
5	*Profile*	Fitur untuk menambahkan profil pengguna
6	*Upgrade to reseller*	Fitur untuk mengajukan permintaan ke admin untuk mengganti *role* menjadi *reseller*
7	Riwayat pembelian lisensi	Fitur untuk melihat riwayat pembelian maupun perpanjangan lisensi yang dimiliki

Pada siklus *test driven development* 2, pengembangan berfokus kepada keseluruhan fitur yang berhubungan dengan modul *admin.* Fitur yang dihasilkan pada siklus *test driven development* 2 ditunjukkan pada tabel 3.4.

Tabel 3.4 Siklus *test driven development* 2

NO	Fitur	Penjelasan
1	*Approved Reseller*	Fitur untuk mengubah role *customer* menjadi reseller
2	*Admin Dashboard*	Fitur ini merupakan halaman awal yang muncul setelah login sebagai *admin* yang menampilkan keaktifan *reseller* yang terdaftar
3	*Deactivate reseller*	Fitur untuk menonaktifkan akun *reseller* yang sudah tidak aktif atau tidak memiliki progress
4	*Check-in as customer*	Fitur ini dimiliki oleh modul *admin* dan *reseller*, untuk masuk sebagai *customer* dan memproses pembelian sesuai permintaan dari *customer* yang memiliki akun tersebut (*reseller* hanya dapat melakukan check-in untuk customer yang didaftarkan oleh *reseller* tersebut)

Pada siklus *test driven development* 3, pengembangan berfokus kepada keseluruhan fitur yang berhubungan dengan modul *reseller*. Fitur yang dihasilkan pada siklus *test driven development* 3 ditunjukkan pada tabel 3.5.

Tabel 3.5 Siklus *test driven development* 3

NO	Fitur	Penjelasan
1	*Create customer account*	Fitur *reseller* untuk menambahkan *customer* baru
2	*Reseller Dashboard*	Fitur ini merupakan halaman awal yang muncul setelah login sebagai *reseller* yang menampilkan *ranking reseller point*, bonus yang diterima *reseller*, dan *list customer* yang dimiliki oleh *reseller*

Setiap pengujian aplikasi dilakukan secara otomatis dengan menggunakan PHP *unit test* yang terintegrasi dengan *framework* laravel, tetapi terdapat beberapa fitur yang perlu dilakukan pengujian untuk kedua kalinya dikarenakan sangat berhubungan dengan *user interface* atau fitur yang berhubungan dengan pengiriman email. Diantaranya memastikan bahwa captcha muncul setelah 3 kali kesalahan login, peringatan yang muncul saat terjadi kesalahan, dan informasi yang disampaikan di dalam email yang dikirimkan.

Terdapat beberapa adaptasi yang berbeda pada penerapan *test driven development* dikarenakan anggota tim dan kondisi pandemi, seperti diskusi dan laporan yang biasanya dilakukan 1 minggu sebanyak 2 kali menjadi hanya 1 kali pertemuan dengan durasi lebih lama dikarenakan tim terdiri dari departemen yang berbeda, serta pertemuan sering terganggu karena

beberapa kendala seperti koneksi membuat tim harus menentukan jadwal khusus untuk melakukan diskusi secara offline.

Pengujian Akhir

Setelah dilakukannya evaluasi akhir dengan menerapkan *black box testing* untuk menguji fungsional dan non-fungsional sistem, semua skenario *test case* diuji oleh 5 orang *tester* yang terdiri dari 3 orang tim penguji dan 2 orang calon pengguna *webstie,* pengujian berlangsung tanpa kesulitan yang cukup signifikan dikarenakan alur penggunaan aplikasi yang cukup sederhana membuat pengguna tidak terlalu sulit untuk menggunakannya, sistem dinyatakan layak untuk dilakukan *deployment* setelah dilakukan semua pengujian sesuai *test case* yang telah dibuat, dimana dari total 120 *test* ditemukan 6 *test case* dengan indikasi *error* atau sekitar 95%. Dengan tingkat fungsional yang lulus uji mencapai angka 95% menandakan bahwa *usability* sistem yang tinggi, serta alur penggunaan aplikasi yang simpel membuat pengguna tidak terlalu sulit untuk menggunakannya, walaupun terdapat beberapa *bug* yang terjadi diantaranya pada bagian *interface* terdapat tombol yang tertimpa atau hilang pada saat dibuka melalui perangkat mobile ataupun peringatan yang berbeda dari *error* yang terjadi. Setelah pengujian berakhir, pengembang melakukan *bug fixing* terhadap *error* yang ditemukan di 6 *test case* tersebut, dan di akhiri dengan dilakukannya *regression testing*.

Hasil Temuan

Terdapat beberapa temuan yang ditemukan selama proyek berlangsung, khususnya pada bagian pengujian aplikasi, beberapa *error* yang ditemukan selama pengujian baik pengujian pada saat *test driven development* dan pengujian akhir adalah sebagai berikut.

Pada saat melakukan *test driven development, error* sering terjadi dikarenakan kesalahan penulisan kode yang tidak sesuai dengan *requirement* yang dituliskan pada *unit test case*, *error* sering terjadi karena kesalahan penulisan batas data input yang dilakukan berbeda dari *requirement* serta *redirect page* yang salah, dimana terkadang salah satu *role* pengguna tidak dapat mengakses halaman tersebut akan tetapi kode memerintahkan pengguna untuk mengakses laman tersebut sehingga terjadi perulangan terus menerus yang menyebabkan *error* yang berulang.

Pada saat melakukan pengujian akhir terdapat beberapa *error* yang terjadi, *error* ini dapat terjadi karena pada saat melakukan *test driven development* semua pengujian dilakukan untuk menguji *back-end* saja. *Error* yang ditemukan pada saat pengujian akhir terdapat beberapa jenis

error yang berbeda baik pada *back-end* maupun *front-end*, pada *back-end* terdapat *bug* dimana karakter yang seharusnya tidak dapat menjadi *input* dapat dimasukkan dan data berhasil tersimpan, akan tetapi nilai dari *input* tersebut tercatat sebagai nilai 0. *Error* lainnya disebabkan karena faktor eksternal yaitu keadaan *portrait* atau *landscape*, dengan beda besaran layar dan faktor tambahan lain. Contoh kasus yang dialami yaitu pada beberapa layar hp milik developer, di halaman tertentu pada saat dalam keadaan *landscape* terdapat tombol yang tidak terlihat karena kapasitas layar yang kecil, dan pada developer lain, tombol tetap terlihat.

Selain pada masalah pengujian akhir terdapat beberapa temuan terkait penggunaan *test driven development*, pada minggu pertama siklus dilakukan terdapat perbedaan kecepatan kerja anggota yang sudah terbiasa atau sering berhadapan dengan penulisan *unit test case* dan anggota tim yang belum terbiasa dalam menuliskan *unit test case*, akan tetapi perbedaan waktu ini berangsur-angsur hilang setelah proyek sudah berjalan cukup lama. Terdapat juga beberapa masalah yang disebabkan oleh kode yang terlalu rumit atau panjang ataupun penggunaan suatu fungsi yang dapat membuat kinerja *website* menjadi lambat sehingga perlu dilakukannya *refactoring* kode. Hal ini dapat terjadi karena sedikitnya fungsi yang diketahui sehingga tidak dapat menggunakan beberapa fungsi alternatif lain yang memberikan efek yang mirip tetapi dengan kecepatan yang berbeda, masalah ini ditemukan pada awal mulai proyek, dan berangsur-angsur hilang karena pengalaman yang dirasakan selama proyek berlangsung.

BAB IV
REFLEKSI PELAKSANAAN MAGANG

4.1 Relevansi Akademik

Terdapat beberapa perbedaan proses dalam penerapan *test driven development* yang dilakukan pada pengembangan *website* berdasarkan teori dan pelaksanaan pengembangan di lapangan. Perbedaan ini merupakan bentuk dari adaptasi yang dilakukan karena beberapa kondisi yang dialami selama perkembangan, kondisi yang menyebabkan perlunya dilakukan adaptasi adalah sebagai berikut :

a. Wabah Covid-19.
b. Jumlah anggota tim pengembang dan tester.
c. Faktor penting yang diutamakan dalam pengembangan.

Dalam praktiknya perusahaan lokasi dilakukannya magang biasanya mengembangkan aplikasi menggunakan metode *scrum* dan *extreme programming* namun karena beberapa kondisi diatas, perusahaan memutuskan untuk mengganti metode dengan metode *agile* lainnya, yang pertimbangannya dijelaskan pada kajian pustaka yang tercantum pada bab 2, selain karena beberapa alasan tersebut, terpilihnya *test driven development* dikarenakan *test driven development* merupakan pecahan dari *extreme programming*, atau merupakan bagian dari langkah *extreme programming*, meskipun begitu *test driven development* merupakan metode *agile* tersendiri yang termasuk ke dalam *agile testing methodologies*.

Dalam penerapannya *test driven development* yang dilakukan memiliki perbedaan dengan teorinya dimana pada teorinya *test driven development* dilakukan dengan 3 langkah yaitu pembuatan *unit test*, penulisan kode, dan *refactoring*, sedangkan pada penerapan dilapangan berdasarkan kondisi yang dialami perlu dilakukannya adaptasi, yaitu penambahan 3 langkah yang mengawali siklus *test driven development* yaitu *envisioning, priority modeling* serta *model storming* dan dilakukannya pengujian tambahan sebagai evaluasi akhir, dilakukannya langkah tambahan ini terjadi karena beberapa tujuan yang ingin dicapai diantaranya :

a. *Test driven development* memiliki kemampuan *scalability* yang kecil, sehingga proses *envisioning, priority modeling*, serta *model storming* digunakan untuk menambah *scalability* dari metode tersebut.

b. Karena kondisi pandemi yang berlangsung selama masa pengembangan website komunikasi memiliki beberapa kendala sehingga diperlukan pembagian yang jelas terkait penyelesaian elemen-elemen yang terdapat pada *website*, dengan dilakukannya *priority modeling* pengembang dapat melakukan pembagian tugas dengan cukup jelas, selain itu karena tidak semua tugas dibagikan secara langsung, terdapat beberapa tugas tambahan yang harus dilakukan oleh pengembang yang disatukan dalam sebuah *stack* penugasan yang dapat diambil oleh pengembang berdasarkan *priority* yang dimiliki oleh tugas tersebut.
c. Dengan penambahan 3 langkah tersebut juga memudahkan kerja sama tim pengembang, karena pada saat *envisioning*, *requirement* yang ada pada *unit test* sudah dipaparkan, dan *flow* berjalannya fitur sudah ditentukan, sehingga jika di tengah berjalannya pengembangan *website* salah satu anggota tim pengembang ada yang absen, anggota lain dapat dengan mudah melanjutkan kode yang telah dikembangkan, yang secara tidak langsung meningkatkan kerja sama anggota tim, dan menjadi sarana *knowledge sharing* antara pengembang satu dan yang lain.
d. Karena *test driven development* hanya menguji fungsionalitas saja, masalah-masalah dari sudut pengguna tidak dilakukan pengujian pada siklus *test driven development*, masalah tersebut adalah tampilan dari *website* itu sendiri, dan alur penggunaan dan pengalamatan dari awal mulai melakukan login hingga langkah terakhir pembayaran pesanan. Berangkat dari masalah tersebut, tim pengembang memutuskan untuk melakukan pengujian akhir, dimana pengujiannya dilakukan berdasarkan perspektif pengguna, pengujian tersebut dilakukan oleh tim penguji dengan menggunakan *black box testing*.

Selain adaptasi dengan penambahan langkah-langkah yang dilakukan, terdapat adaptasi yang dilakukan karena kebiasaan dari tim pengembang perusahaan, di perusahaan pelaksanaan magang, pengembangan aplikasi selalu menggunakan *model-view-controller architectural pattern*, sehingga pada pengembangan kali ini desain awal pada pengembangan juga dilakukan dalam bentuk *MVC*, pada pengembangan ini hal itu dapat dilakukan dengan cara menjadikan setiap *role* yang ada menjadi sebuah modul tersendiri, dimana *stack* penugasan yang dikembangkan dibedakan setiap modulnya. Oleh karena itu penggunaan *framework* laravel sangat membantu dalam pengembangan ini, selain karena laravel menerapkan *MVC*, laravel memiliki *package-package* yang dapat mempermudah dan membuat kode yang dituliskan mudah untuk dipahami. Serta dengan menggunakan *unit test* yang ada pada laravel *package*

kerusakan atau *error* yang diterima pada saat melakukan pengujian baik sebelum atau sesudah *refactoring* lebih mudah terdeteksi, hal ini sangat membantu tim pengembang dalam melakukan *refactoring* karena tim pengembang tidak mengalami kesulitan dalam mendeteksi fungsi mana yang mengalami *error*.

Adaptasi diatas dilakukan tim pengembang untuk memenuhi faktor-faktor utama dari pengembangan yang diinginkan, adaptasi tersebut dipilih dianggap sesuai karena beberapa alasan yaitu:

a. Kebiasaan anggota tim pengembang.
b. Ukuran tim pengembang.
c. Pandemi covid-19.

Maka dari itu, adaptasi yang dilakukan bukan berarti adaptasi yang paling baik, berbeda tim atau perusahaan dapat menyebabkan *website* yang sama dengan kondisi yang sama menggunakan metode yang berbeda.

4.2 Pembelajaran Magang

Selama melakukan magang di Ozeva Technology selama kurang lebih 6 bulan terdapat banyak pembelajaran yang diperoleh penulis. Manfaat yang didapatkan selama melakukan magang antara lain adalah:

a. Mengerti kapan suatu kode dapat dimasukkan dalam satu fungsi atau kapan suatu kode tersebut di pisah dan dipanggil di fungsi lain, hal ini diperlukan karena berpengaruh terhadap beberapa hal yaitu *running time*, kerapian dan kemudahan dipahami, serta pengurangan duplikasi, kode yang dipisah menjadi dapat digunakan atau dipanggil fungsi lain yang hanya memerlukan kode tersebut.
b. Belajar berfikir efektif pada saat melakukan *model storming*, dikarenakan durasi diskusi yang tidak terlalu lama, dan banyaknya fitur yang dibicarakan, serta saran terkait penggunaan fungsi 1 dan lainnya, dimana terdapat 2 sampai 3 fungsi yang berbeda dapat digunakan untuk penyelesaian suatu masalah dan pengembang harus dengan cepat dan tepat memutuskan fungsi mana yang lebih cocok digunakan untuk suatu fitur, dengan mempertimbangkan beberapa kondisi yaitu *running time* dari kode dan koneksi kode tersebut dengan kode lainnya.
c. Dengan penggunaan *test driven development*, pengembang menjadi lebih familiar dengan bagaimana seharusnya *unit test* dituliskan, karena pada saat penulisan *unit test* seluruh pengembang harus memikirkan sebuah *test case* yang dapat mencakup

semua kemungkinan yang akan terjadi baik dari segi *input* yang dimasukkan, *domain data* yang dapat diterima, notifikasi yang akan dikeluarkan oleh sistem, dan memastikan bahwa sistem mengarahkan ke halaman selanjutnya dengan benar.

d. *Test driven development* melatih pengembang melatih pengguna untuk menuliskan kode seminimal mungkin sehingga kode yang dituliskan lebih mudah dipahami oleh pengembang lain, hal ini dapat terjadi karena penulisan kode dikembangkan berdasarkan *unit test* yang sudah dikembangkan terlebih dahulu, dimana *unit* test menjadi dasar dan merupakan kesepakatan dari tim pengembang.
e. *Test driven development* secara tidak langsung melatih pengembang dalam melakukan *refactoring*, karena pada saat melakukan *refactoring*, terdapat kemungkinan terjadinya kerusakan pada kode, tetapi dengan menjalankan serangkaian pengujian, akan muncul peringatan jika ditemukan kerusakan yang ada, dan titik dimana kerusakan terdeteksi sehingga tim pengembang menjadi lebih nyaman dalam melakukan *refactoring*.

Selain mendapatkan manfaat, juga ditemukan hambatan dan tantangan yang dirasakan selama melaksanakan magang antara lain :

a. Karena pengembangan menggunakan *test driven development*, maka pembuatan *unit test case* sangat penting, akan tetapi dengan *requirement* yang sama, pengembang satu dan lainnya dapat membuat *unit test case* yang berbeda. Terkadang penulisan *unit test* memerlukan waktu yang lama, karena *unit test* harus dikembangkan sesederhana mungkin karena jika pada *unit test* sudah terlalu rumit maka kode yang diproduksi akan menjadi lebih rumit dan susah dibaca. Oleh karena itu diperlukan kemampuan interpretasi yang baik atas *requirement* dan pengertian terhadap fungsi dan *command* yang tersedia agar *unit test* yang dibuat dalam bentuk yang sederhana.
b. Sering melakukan penulisan kode yang dianggap tidak memenuhi standar atau terlalu kompleks sehingga perlu dilakukan revisi.
c. Penulisan *test case* terkadang membutuhkan waktu yang lama dikarenakan cakupan dari *unit test*, serta terkadang diperlukan sebuah dokumentasi yang harus dimiliki oleh *test case*, seperti beberapa *requirement* yang dituliskan pada saat *envisioning*. Terlebih jika pengembang tidak terbiasa dalam menuliskan *unit test*, hal ini akan menghambat siklus *test driven development* karena siklus dimulai dari penulisan *unit test*, dan jika *unit test* tidak benar atau tidak akurat maka akan sering terjadi perubahan kode yang dituliskan pada fitur di *website*.

d. Terkadang *unit test* yang dituliskan tidak memberikan pengembang gambaran dari proyek, karena terkadang *test* dikembangkan tidak sesuai dengan proses bisnis awal, sehingga tidak dapat mengkonversikan kebutuhan pengguna ke dalam *unit test*. Hal ini juga merupakan alasan mengapa penulisan *unit test* membutuhkan waktu yang lama.
e. Untuk melakukan pengujian pada *back-end* terbilang cepat karena pengembang hanya perlu menjalankan pengujian yang dikumpulkan pada *file* yang sama tetapi pengujian tersebut hanya mencakup pengujian *back-end*, sehingga diperlukan waktu dan tenaga tambahan untuk melakukan pengujian terhadap *front-end* pada *website*.

Dari semua pembelajaran yang di terima, telah mengembangkan kepercayaan diri atas kemampuan yang dimiliki, juga menjadi lebih percaya dan menjadi lebih yakin untuk dapat beradaptasi dengan dunia kerja yang akan dihadapi setelah lulus dari perkuliahan di Universitas Islam Indonesia nanti.

BAB V
PENUTUP

5.1 Kesimpulan

Setelah dilakukan pengembangan *website* hingga mencapai pengujian akhir dapat disimpulkan bahwa penggunaan metode *test driven development* dapat meminimalisir adanya *bug* dan *error* yang dibuktikan dengan data dari evaluasi akhir yang menyatakan bahwa sekitar 95% fungsionalitas dinyatakan lulus pengujian. Meskipun terdapat beberapa adaptasi yang harus dilakukan karena pandemi yang terjadi, efek dari pandemi tidak terlalu berpengaruh dikarenakan dalam penerapan kolaborasi antara anggota tim. Anggota tim lain dapat melakukan perubahan kepada kode anggota tim lain dengan aman karena *unit testing* yang dilakukan akan memberi informasi jika perubahan tersebut memberikan efek yang tidak diperkirakan sebelumnya.

Efisiensi dan efektivitas dari *test driven development* dapat mengalami peningkatan maupun penurunan berdasarkan dari pengalaman anggota pengembang yang bertanggungjawab selama proyek berjalan, dikarenakan beberapa faktor seperti seberapa luas pengetahuan pengembang dalam menggunakan beberapa fungsi serta mengetahui efek fungsi tersebut dalam berjalannya aplikasi, karena aplikasi yang nyaman digunakan tidak hanya dilihat dari susah atau tidaknya aplikasi digunakan tetapi juga kecepatan proses aplikasi tersebut.

Penggunaan *MVC* dan *framework* laravel memiliki pengaruh yang cukup bagus dalam pengembangan menggunakan *test driven development*, dimana penggunaan metode dan *framework* tersebut memudahkan pengembang lain membaca kode yang sudah dituliskan, lokasi dari kode yang ingin dicari, sehingga *maintainability* dari sistem mengalami peningkatan, karena pengembangan menggunakan *MVC* berbentuk modular, ketika pengembang ingin menuliskan atau mencari kode terkait modul tertentu pengembang hanya perlu untuk mencari file terkait modul tersebut, karena fungsi-fungsi terkait *model, view* dan *controller* dipisahkan berdasarkan modul yang berhubungan dengan fungsi tersebut.

Penggunaan *black box testing* untuk menguji *front-end* dari aplikasi memiliki pengaruh yang sangat besar untuk menutupi kekurangan dari *test driven development*, dimana pada saat menguji tampilan pada *website* ditemukan masalah tampilan pada saat menggunakan mode

landscape dan portrait menggunakan perangkat *mobile,* hal ini menjadi bukti bahwa *black box testing* dapat menjadi metode pendukung dalam menjalankan *test driven development.*

Penggunaan *black box testing* untuk menguji dari perspektif pengguna berjalan dengan baik yang dibuktikan dengan tidak adanya kesulitan yang cukup signifikan pada saat pengujian ke tim tester maupun beberapa calon pengguna.

5.2 Saran

Berdasarkan pengalaman yang didapatkan pada saat melaksanakan pengembangan di lapangan mengenai *website license management*, pada bagian ini terdapat beberapa saran yang sekiranya dapat digunakan untuk pengembangan berikutnya.

Jika waktu pengembangan *website* yang akan dilakukan cukup panjang pengembangan berikutnya dapat menggunakan metode dari cabang *agile* yang lain, karena perbedaan pada waktu pengerjaan dan faktor lain seperti keuangan, dan lain sebagainya, membuat faktor yang diutamakan dalam *website* yang dikembangkan akan berbeda, sehingga terdapat kemungkinan metode *agile* lain lebih cocok digunakan.

Jika pengembangan memiliki waktu pengerjaan yang tidak terlalu panjang dan pengembang memutuskan untuk menggunakan *test driven development,* pada pengembangan berikutnya, perlu dipastikan terlebih dahulu bahwa anggota pengembang sudah terbiasa dalam menuliskan *unit test case* serta waktu yang digunakan untuk melakukan *envisioning* dapat dilakukan dalam waktu yang lebih lama sehingga *initial requirement* yang dihasilkan lebih jelas, hal ini akan secara langsung mempengaruhi seberapa besar kemampuan *scalability* yang dimiliki oleh *website* yang akan dikembangkan, serta pembuatan *test case* untuk *black box* dibuat lebih rinci untuk mencakup beberapa aspek yang mungkin tidak tertutupi dalam proyek ini. Anggota tim pengembang sebaiknya mengetahui beberapa alternatif fungsi yang dapat digunakan untuk beberapa skenario berbeda baik fungsi yang memerlukan *framework* jika pengembangan menggunakan *framework* tertentu maupun fungsi yang tidak memerlukan *framework*.

DAFTAR PUSTAKA

Anjaria, D. M. (2018). Communication in MVC Teams: A Tes tDriven Approach. *Indian Journal of Computer Science and Engineering (IJCSE)*.

Erdogmus, H., Melnik, G., & Jeffries, R. (n.d.). Test-Driven Development.

Fruhling, A., & De Vreede, G.-J. (2014). Field Experiences with eXtreme Programming:Developing an Emergency Response System. *Journal of Management Information Systems*.

Hamilton, T. (2022). *Guru99*. Retrieved from What is Test Driven Development (TDD)? Tutorial with Example: https://www.guru99.com/test-driven-development.html

Koi-Akrofi, G. Y., Koi-Akrofi, J., & Matey, H. A. (2019). Understanding The Characteristics, Benefits and Challenges of Agile IT Project Management: a Literature Based Perspective. *International Journal of Software Engineering & Applications (IJSEA)*.

Lewis, W. E., Veerapillai, G., & Dobbs, D. (2009). *Software Testing and Continuous Quality Improvement.* New York: Auerbach Publications.

Nagappan, N., Maximillien, E. M., Bhat, T., & Williams, L. (2008). Realizing quality improvement through test driven. *Springer Science + Business Media*.

Nidhra, S., & Dondeti, J. (2012). Black Box and White Box Testing Techniques – A Literature Review. *International Journal of Embedded Systems and Applications (IJESA)*.

Ozeva Technology. (2021). Retrieved from Ozeva: https://www.ozeva.com/in/about

Pop, D.-P., & Altar, A. (2013). Designing an MVC Model for Rapid Web Application Development. *24th DAAAM International Symposium on Intelligent Manufacturing and Automation*.

Shahin, M., Babar, M. A., & Zhu, L. (2017). 1 Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices.

Svirca, Z. (2020, May 29). *Everything you need to know about MVC architecture*. Retrieved from towardsdatascience: https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1

Thohari, A. N., & Amelia, A. E. (2018). Implementasi Test Driven Development Dalam Pengembangan Aplikasi Berbasis Web. *Jurnal Sistem Informasi dan Teknologi*.

Yenduri, S., & Perkins, A. L. (2006). Impact of Using Test-Driven Development: A Case Study. *Proceedings of the International Conference on Software Engineering Research*

and Practice & Conference on Programming Languages and Compilers, SERP 2006. Las Vegas, Nevada.

Zeba, K., & Ahsan, M. N. (2017). Evaluating the Effectiveness of Test Driven Development: Advantages and Pitfalls. *International Journal of Applied Engineering Research*.

LAMPIRAN

Lampiran tidak perlu diberi nomor halaman. Dokumen apa saja yang dimasukkan dalam lampiran cukup diberi judul dengan kata ‘LAMPIRAN’ yang dilanjutkan dengan huruf abjad besar untuk penomoran. Cukup judul ‘LAMPIRAN’ saja yang dimasukkan dalam daftar isi. Judul-judul lampiran, seperti Lampiran A, Lampiran B dan seterusnya, tidak perlu dimasukkan dalam daftar isi.