

LAPORAN KERJA PRAKTIK

PENGEMBANGAN SISTEM ANTRIAN REAL-TIME UNTUK MODUL VIDEO CALL PADA PT. MECHLAB TEKNOLOGI INDONESIA



Disusun oleh:

Baskoro Adi Wicaksono

16/395387/TK/44679

PROGRAM STUDI TEKNOLOGI INFORMASI

DEPARTEMEN TEKNIK ELEKTRO DAN TEKNOLOGI INFORMASI

FAKULTAS TEKNIK UNIVERSITAS GADJAH MADA

YOGYAKARTA

2020

HALAMAN PENGESAHAN

PENGEMBANGAN SISTEM ANTRIAN REAL-TIME UNTUK MODUL VIDEO CALL PADA PT. MECHLAB TEKNOLOGI INDONESIA

LAPORAN KERJA PRAKTIK

Diajukan Sebagai Salah Satu Syarat untuk Memperoleh
Gelar Sarjana Teknik Program S-1
Pada Departemen Teknik Elektro dan Teknologi Informasi Fakultas Teknik
Universitas Gadjah Mada

Disusun oleh:

BASKORO ADI WICAKSONO

16/395387/TK/44679

Telah disetujui dan disahkan

Pada 18 Agustus 2020

Dosen Pembimbing Kerja Praktik

Ir. Marcus Nurtiantara Aji, M.T.

NIP. 196404241995121001

SURAT PERINTAH KERJA PRAKTIK



UNIVERSITAS GADJAH MADA
FAKULTAS TEKNIK
DEPARTEMEN TEKNIK ELEKTRO DAN TEKNOLOGI INFORMASI

SURAT PERINTAH KERJA PRAKTIK
No. 2811/H1.17/TKE/KM/2019

Diperintahkan kepada,

Nama : Baskoro Adi Wicaksono
NIM : 16/395387/TK/44679
Tanggal Lahir / Umur : 03 November 1998 / 21 tahun
Status : Mahasiswa Departemen Teknik Elektro dan Teknologi Informasi
Fakultas Teknik Universitas Gadjah Mada
Program Studi : Teknologi Informasi
Alamat Sementara : Pogung Kidul RT. 01 RW. 49 no 22, Sinduadi, Mlati, Sleman
Alamat Asal : Kenteng RT. 02 RW 02, Ngadirojo Kidul, Ngadirojo, Wonogiri
No. HP : 085826664521
Tujuan KP : PT Mechlab Teknologi Indonesia
Mulai Tanggal : 07 November 2019
Sampai Tanggal : 07 Desember 2019 (tidak bersamaan UTS dan UAS)
untuk melaksanakan Kerja Praktik sesuai dengan ketentuan di atas.

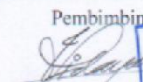

Yogyakarta, 29 November 2019

Mengetahui,
Sekretaris Departemen


Hanung Adi Nugroho, S.T., M.E., Ph.D., IPM.
NIP-197802242002121001

Tugas Kerja Praktik tersebut telah diselesaikan
Pada tanggal : 7 Desember 2019

Pembimbing


Didang Hidayat.....


Tembusan :

1. Ir. Marcus Nurtiantara Aji, M.T.
Dosen Pembimbing Departemen Teknik Elektro dan Teknologi Informasi FT UGM.
2. Mahasiswa yang bersangkutan.

KATA PENGANTAR

Puji dan syukur penulis panjatkan atas ke hadirat Allah SWT atas segala limpahan rahmat dan hidayah-Nya sampai saat ini, sehingga penulis dapat menyelesaikan kerja praktik di PT. Mechlab Teknologi Indonesia serta mampu menyelesaikan laporan kerja praktik dengan judul “Pengembangan Sistem Antrian *Real-Time* untuk Modul *Video Call* pada PT. Mechlab Teknologi Indonesia” dengan baik dan lancar. Laporan ini disusun sebagai hasil akhir kerja praktik yang telah dilaksanakan mulai tanggal 7 November 2019 sampai 7 Desember 2019.

Dalam penyusunan laporan ini, penulis menyadari sepenuhnya bahwa selesainya laporan Kerja Praktik ini tidak terlepas dari dukungan, semangat, serta bimbingan dari berbagai pihak, baik bersifat moral maupun materiil. Oleh karena itu, penulis ingin menyampaikan ucapan terima kasih antara lain kepada:

1. Bapak Sarijya S.T., M.T., Ph.D. selaku Ketua Departemen Teknik Elektro dan Teknologi Informasi Fakultas Teknik Universitas Gadjah Mada
2. Bapak Ir. Marcus Nurtiantara Aji, M.T. selaku Dosen Pembimbing Kerja Praktik.
3. Bapak Dadang Hidayat selaku CEO PT. Mechlab Teknologi Indonesia

Dalam penyusunan Laporan Kerja Praktik (KP) ini masih terdapat beberapa kekurangan di dalamnya, oleh karena itu saran dan kritik yang bersifat membangun dari semua pihak sangat diharapkan, tidak lupa harapan penulis semoga Laporan Kerja Praktik ini dapat bermanfaat bagi pembaca serta dapat menambah ilmu pengetahuan.

Yogyakarta, 18 Agustus 2020

Penulis

DAFTAR ISI

LAPORAN KERJA PRAKTIK.....	i
HALAMAN PENGESAHAN.....	ii
SURAT PERINTAH KERJA PRAKTIK.....	iii
KATA PENGANTAR.....	iv
DAFTAR ISI.....	v
DAFTAR GAMBAR.....	vii
DAFTAR TABEL.....	viii
DAFTAR LAMPIRAN.....	viii
BAB 1.....	1
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	2
1.3 Batasan Masalah.....	2
1.4 Tujuan.....	2
1.5 Waktu, Tempat, dan Kegiatan Kerja Praktik.....	2
1.6 Metodologi Pelaksanaan.....	2
1.7 Sistematika Penulisan.....	2
BAB 2.....	4
2.1 Profil Perusahaan.....	4
2.2 Struktur Perusahaan PT.....	4
2.3 Budaya Perusahaan.....	5
2.3.1 Tempat Kerja.....	6
2.3.2 Jam Kerja.....	7
2.3.3 Pakaian Kerja.....	7
BAB 3.....	8
3.1 InfinId.....	8
3.2 REST API.....	9
3.3 <i>Database</i>	10
3.4 Node.js.....	11
3.5 Express.....	11
3.6 Socket.io.....	12
3.7 JSON <i>Web Token</i>	13

3.8 Knex.js.....	14
BAB 4.....	15
4.1 Alat dan Bahan Kerja Praktik.....	15
4.2 Alur Kerja Praktik.....	15
BAB 5.....	16
5.1 Arsitektur Sistem.....	16
5.2 Skema <i>Database</i>	16
5.3 Struktur Sistem.....	17
5.4 <i>Helper</i>	18
5.4.1 <i>Helper JSON Web Token</i>	18
5.4.2 <i>Helper Verifikasi Status Login</i>	18
5.5 Model.....	18
5.5.1 <i>Index.js</i>	19
5.5.2 Bank Model.....	19
5.5.3 Agent Model.....	19
5.5.4 Antrian Model.....	20
5.6 <i>Controller</i>	20
5.6.1 Bank <i>Controller</i>	20
5.6.2 Agent <i>Controller</i>	20
5.6.3 Antrian <i>Controller</i>	21
5.7 Hasil Pengujian tiap API.....	22
BAB 6.....	28
6.1 Kesimpulan.....	28
6.2 Saran.....	28
DAFTAR PUSTAKA.....	29
LAMPIRAN.....	30

DAFTAR GAMBAR

Gambar 2.1 Logo Mechatronic Laboratory.....	4
Gambar 2.2 Struktur Perusahaan Mechatronic Laboratory.....	4
Gambar 2.3 Ruangan Innovative Academy Hub.....	6
Gambar 2.4 Lokasi Innovative Academy Hub.....	6
Gambar 5.1 Arsitektur Sistem Antrian InfinId.....	16
Gambar 5.2 Skema <i>Database</i> Sistem Antrian InfinId.....	16
Gambar 5.3 Struktur <i>Project</i> Sistem Antrian InfinId.....	17
Gambar 5.4 Pengujian API Register Bank Menggunakan Postman.....	22
Gambar 5.5 Pengujian API Register Agent Menggunakan Postman.....	22
Gambar 5.6 Pengujian API Login Agent Menggunakan Postman.....	23
Gambar 5.7 Pengujian API Refresh Token Menggunakan Postman.....	23
Gambar 5.8 Pengujian API Register Antrian Menggunakan Postman.....	24
Gambar 5.9 Pengujian API Register Antrian Menggunakan Google Chrome.....	24
Gambar 5.10 Pengujian API Next Antrian Menggunakan Postman.....	25
Gambar 5.11 Pengujian API Next Antrian Menggunakan Google Chrome.....	25
Gambar 5.12 Pengujian API End Antrian Menggunakan Postman.....	26
Gambar 5.13 Pengujian API End Antrian Menggunakan Google Chrome.....	26
Gambar 5.14 Pengujian API Count Antrian Menggunakan Postman.....	27

DAFTAR TABEL

Tabel 4.1 Alur Kerja Praktik.....	15
Tabel 5.1 <i>Source Code</i> Pengaturan Koneksi <i>Database</i> dengan Knex.js.....	19

DAFTAR LAMPIRAN

Lampiran L.1 Surat keterangan magang di PT. Mechlab Teknologi Indonesia.....	30
Lampiran L.2 Foto penulis beserta tim Mechatronic Laboratory.....	31
Lampiran L.3 Suasana ruang kerja saat istirahat makan siang.....	31
Lampiran L.4 Foto penulis beserta tim developer, CEO beserta pihak investor.....	31
Lampiran L.5 Kueri SQL Skema <i>Database</i> Sistem Antrian InfinId.....	32
Lampiran L.6 <i>Source Code Helper JSON Web Token</i>	33
Lampiran L.7 <i>Source Code Helper Verifikasi Status Login</i>	34
Lampiran L.8 <i>Source Code Bank Model</i>	35
Lampiran L.9 <i>Source Code Agent Model</i>	36
Lampiran L.10 <i>Source Code Antrian Model</i>	37
Lampiran L.11 <i>Source Code Bank Controller</i>	39
Lampiran L.12 <i>Source Code Agent Controller</i>	40
Lampiran L.13 <i>Source Code Antrian Controller</i>	42

BAB I PENDAHULUAN

1.1 Latar Belakang

PT. Mechlab Teknologi Indonesia atau yang biasa disebut dengan Mechatronic Laboratory merupakan *start-up* besutan Innovative Academy UGM yang berfokus pada pengembangan *financial technology*. Saat ini Mechatronic Laboratory sedang mengembangkan sebuah produk bernama InfinId. InfinId merupakan sebuah produk yang berfokus pada proses eKYC (*electronic know-your-customer*) berbasis *mobile app*.

Pada dasarnya setiap bank wajib melakukan prinsip KYC (*Know Your Customer Principles*) yang telah diwajibkan oleh Bank Indonesia, seperti yang telah dituangkan dalam peraturan BI Nomor 3/10/PBI/2001 tentang Penerapan Prinsip Mengenal Pelanggan (*Know Your Customer Principles*). Sesuai dengan pasal 1 ayat 2, pada dasarnya prinsip ini diterapkan bank untuk mengetahui identitas nasabah, memantau kegiatan transaksi nasabah termasuk pelaporan transaksi yang mencurigakan. Salah satu transaksi yang harus menerapkan prinsip KYC adalah pembukaan rekening baru.

Pengembangan InfinId didasari pada permasalahan yang dialami pelanggan pada prinsip KYC saat pembukaan rekening pada kebanyakan bank saat ini. Ada 2 fokus utama dibalik dikembangkannya produk InfinId. Pertama, proses pembukaan rekening baru sangat menyita waktu pelanggan. Pelanggan diharuskan datang dan mengantri di kantor cabang dari bank untuk membuka rekening baru. Seringkali antrian menjadi sangat panjang dan pelanggan tidak diperkenankan meninggalkan tempat saat menunggu antrian. Kedua, prosedur yang kompleks dimana pengumpulan data seringkali bersifat duplikasi. Contoh yang paling sering ditemui adalah pelanggan diminta menyerahkan berkas *fotocopy* KTP, namun juga diharuskan mengisi formulir yang isinya tentang data kependudukan. Proses penyerahan data ini juga hanya bisa dilakukan di *costumer service* setelah pelanggan menerima giliran antrian.

InfinId dirancang untuk mengenali pelanggan dengan beberapa modul yang terdiri dari *one-time password*, identifikasi tanda pengenalan dengan *optical character recognition*, deteksi *liveness*, *face recognition* dan *video call*. Modul *video call* akan digunakan validasi calon pelanggan oleh *customer service* dengan menanyakan pertanyaan yang bersifat *confidential* yang tidak bisa diverifikasi oleh fitur lainnya. Dalam pengembangan fitur ini, diperlukan sistem antrian secara *real-time* untuk menentukan giliran pelanggan yang akan dihubungi *customer service* melalui *video call*.

1.2 Rumusan Masalah

1. Modul *video call* memerlukan sistem antrian untuk menentukan giliran pelanggan yang akan dipanggil.
2. Antrian pelanggan dapat dilayani oleh lebih dari satu *customer service* sehingga diperlukan mekanisme agar tidak terjadi duplikasi proses pemanggilan pelanggan.
3. Pelanggan diharapkan dapat mengetahui saat telah menerima giliran antrian secara *real-time* sehingga tidak memperlambat proses pemanggilan.

1.3 Batasan Masalah

Pada kesempatan kerja praktik kali ini, penulis lebih berfokus ke perancangan awal dari sistem antrian secara *real-time* yang akan digunakan oleh modul *video call*. Batasan pada sistem antrian yang akan dibuat antara lain adalah antrian harus bisa mengakomodasi beberapa *customer service* yang akan melayani pelanggan. Selain itu pelanggan harus mengetahui secara *real-time* saat pelanggan menerima panggilan dari *customer service*.

1.4 Tujuan

Membuat rancangan sistem antrian secara *real-time* untuk modul *video call* dengan mempertimbangkan jumlah *customer service* yang akan melayani pelanggan.

1.5 Waktu dan Tempat Pelaksanaan Kerja Praktik

Kerja Praktik dilaksanakan selama 1 bulan dari tanggal 7 November 2019 sampai dengan 7 Desember 2019 di PT. Mechlab Teknologi Indonesia, yang memiliki *workshop* di Innovative Academy HUB, Jl. Bulaksumur H6, Universitas Gadjah Mada.

1.6 Metodologi Pelaksanaan

Metode yang digunakan adalah studi literatur, studi keperluan dan spesifikasi sistem antrian, diskusi dan konsultasi dengan pembimbing, serta pengembangan sistem antrian untuk modul *video call* InfinId.

1.7 Sistematika Penulisan

Laporan Kerja Praktik ini terdiri atas 5 bab, yaitu sebagai berikut:

1. Bab I Pendahuluan

Bab ini berisi tentang latar belakang, rumusan masalah, batasan, tujuan, waktu dan tempat pelaksanaan, metodologi pelaksanaan, serta sistematika penulisan laporan kerja praktik.

2. BAB II Profil Perusahaan

Bab ini berisi tentang uraian profil singkat PT. Mechlab Teknologi Indonesia, termasuk struktur perusahaan dan budaya kerjanya.

3. Bab III Tinjauan Pustaka dan Dasar Teori

Bab ini akan menjelaskan teori-teori terkait dengan permasalahan yang akan dibahas secara ringkas dan jelas.

4. Bab IV Metode Kerja Praktik

Bab ini menjelaskan mengenai alat dan bahan yang diperlukan dalam pelaksanaan Kerja Praktik, dan alur pelaksanaan Kerja Praktik.

5. Bab V Hasil Pembahasan

Bab ini berisi penjelasan secara rinci mengenai hasil pekerjaan yang dilaksanakan selama Kerja Praktik.

6. Bab VI Penutup

Bab ini berisi tentang kesimpulan dari Kerja Praktik yang telah dilaksanakan dan saran yang diharapkan dapat berguna untuk pengembangan selanjutnya.

BAB II PROFIL PERUSAHAAN

2.1 Profil Perusahaan Mechatronic Laboratoy

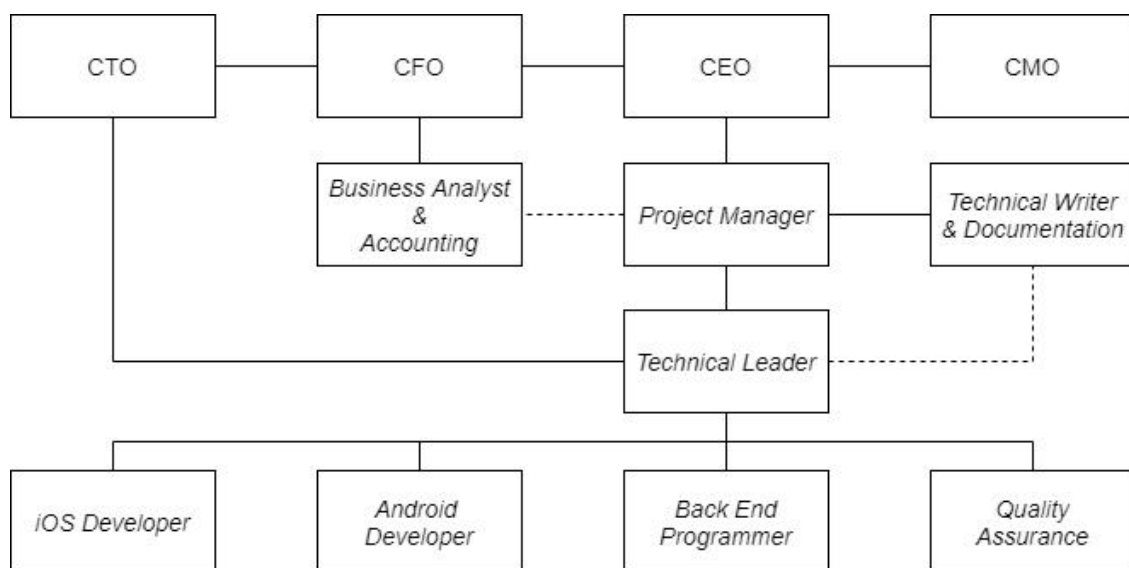
PT. Mechlab Teknologi Indonesia atau yang biasa disebut Mechatronic Laboratoy adalah perusahaan yang bergerak dibidang *financial technology*. Sekarang Mechatronic Laboratoy sedang bekerjasama dengan beberapa lembaga keuangan untuk mengembangkan program yang berfokus pada eKYC (*electronic know-your-customer*) berbasis *mobile apps*.



Gambar 2.1 Logo Mechatronic Laboratory

PT. Teknologi Mechlab Indonesia merupakan *start-up* besutan Innovative Academy UGM. Innovative Academy dikelola oleh Direktorat Pengembangan Usaha dan Inkubasi berkolaborasi dengan mitra-mitra dari pihak industri. *Start-up* yang dibesut oleh Innovative Academy melibatkan peserta dari berbagai bidang ilmu seperti sains, teknologi, dan bisnis. Tiap kelompok *start-up* akan diseleksi, dipandu, dan diarahkan untuk mengembangkan kultur baru dalam mengelola bisnis pemula berbasis teknologi digital. Selanjutnya, tiap kelompok diperkenalkan dengan para pelaku dalam ekosistem pengembangan bisnis, seperti modal ventura, investor pribadi, maupun unit kegiatan akselerator lainnya.

2.2 Struktur Perusahaan



Gambar 2.2 Struktur Perusahaan Mechatronic Laboratory

Pada dasarnya Mechatronic Laboratory memiliki 2 tingkatan pada struktur perusahaannya yaitu *C-level* dan *Product Team* yang susunannya dapat dilihat pada gambar 2.2. Terdapat 4 *role* yang terdapat pada *C-level* yaitu CTO, CFO, CMO dan CEO. *Chief Technology Officer* (CTO) adalah eksekutif yang bertanggung jawab atas kebutuhan teknologi organisasi serta penelitian dan pengembangan (R&D). *Chief Financial Officer* (CFO) adalah eksekutif yang bertanggung jawab untuk mengelola tindakan keuangan dari perusahaan. *Chief Marketing Officer* (CMO) adalah eksekutif yang bertanggung jawab atas aktivitas dalam perusahaan yang berkaitan dengan komunikasi dan penawaran bagi mitra bisnis. *Chief Executive Officer* (CEO) adalah eksekutif dengan peringkat tertinggi pada perusahaan, yang bertanggung jawab dalam membuat keputusan final terkait jalannya perusahaan serta bertindak sebagai titik komunikasi utama antara eksekutif pada *C-level*.

Pada tingkatan *Product Team*, *Product Manager* bertanggung jawab sebagai aktor utama untuk melakukan pengelolaan dari produk yang dikembangkan oleh perusahaan mulai dari proses perencanaan, pengembangan, hingga operasional produk. *Product Manager* bertanggung jawab langsung kepada CEO selaku eksekutif pada *C-level*. Dalam menjalankan tugasnya *Product Manager* berkoordinasi dengan *Business Analyst & Accounting* yang bertanggung jawab dalam menilai proses, menentukan persyaratan, dan memberikan rekomendasi serta laporan berbasis data terkait kebijakan perusahaan kepada CFO. Selain itu *Product Manager* bekerja sama dengan *Technical Writer & Documentation* yang bertanggung jawab dalam mendokumentasikan hasil kerja dari tim pengembang.

Tim pengembang dipimpin oleh seorang *Technical Leader* yang bertanggung jawab dalam memimpin pengembangan maupun *maintenance* dari produk. *Technical Leader* juga bertanggung jawab dalam melaporkan perkembangan dari produk kepada *Product Manager*. Tim pengembang dibagi menjadi 4 divisi sesuai dengan perannya masing-masing dalam pengembangan produk yang terdiri dari *iOS Developer*, *Android Developer*, *Back End Programmer* dan *Quality Assurance*.

2.3 Budaya Perusahaan

Sebagai perusahaan yang masih berada di level *start-up* hubungan antar karyawan di Mechatronic Laboratory bersifat fleksibel terlepas dari hierarki perusahaan. Setiap karyawan memiliki kesempatan yang banyak untuk bertukar ide dengan seluruh tim. Untuk peraturan mengenai tempat, waktu dan pakaian kerja juga mengikuti *trend start-up* pada umumnya, berikut merupakan penjelasannya.

2.3.1 Tempat Kerja

Mechatronic Laboratory menggunakan Innovative Academy Hub, *co-working space* yang dikelola oleh Innovative Academy UGM, sebagai tempat kerjanya. *Co-working space* ini dibangun untuk memperkuat ekosistem inovasi di lingkungan kampus. Innovative Academy Hub menyediakan ruang kerja yang nyaman, bergaya anak muda, dan dilengkapi dengan koneksi internet berkecepatan tinggi.



Gambar 2.3 Ruang Innovative Academy Hub

Innovative Academy Hub masih berada pada lingkungan kampus Universitas Gadjah Mada. Lokasi dari Innovative Academy Hub berada di barat gerbang utama kampus Universitas Gadjah Mada. Untuk alamat lengkapnya adalah di Jl. Bulaksumur H6, Universitas Gadjah Mada, Blimbing Sari, Caturtunggal, Sleman, Yogyakarta 55284.



Gambar 2.4 Lokasi Innovative Academy Hub

Selain Mechatronic Laboratory, 11 *perusahaan start-up* lainnya yang memanfaatkan Innovative Academy Hub sebagai tempat kerja antara lain Wemary, Galanggo, Pasienia, Pijar, Iwak, Muncak, Villageria, Bantu Ternak, Adsiconic, Majapahit, dan Calty Farm.

2.3.2 Jam Kerja

Jam Kerja untuk seorang *developer* di Mechatronic Laboratory adalah 40 jam setiap minggunya. Untuk pengaturan jadwal kerja diserahkan sepenuhnya kepada *Developer*. *Developer* dapat memilih waktu antara hari senin sampai minggu pada pukul 08.00 sampai 18.00 untuk memenuhi jam kerja selama 40 jam/minggu. Terdapat minimal 1 hari dalam seminggu dimana seluruh tim akan melakukan *meeting*, untuk melaporkan perkembangan dari produk yang dikerjakan dan rencana pengerjaan selanjutnya.

2.3.3 Pakaian Kerja

Mechatronic Laboratory tidak memiliki aturan pakaian kerja yang tidak terlalu mengikat. Perusahaan tidak terlalu mementingkan pakaian yang dipakai oleh karyawan selama pakaian dipakai masih dapat terbilang sopan dan rapi. Untuk alas kaki perusahaan masih mewajibkan karyawan untuk memakai sepatu, namun tidak ada aturan spesifik mengenai jenis sepatu yang akan dipakai. Contoh pakaian yang boleh dipakai saat jam kerja antara lain, kemeja, *sweater* atau jaket, celana denim, maupun sepatu *sneakers*.

BAB III TINJAUAN PUSTAKA DAN DASAR TEORI

3.1 InfinId

Mechatronic Laboratory saat ini sedang mengembangkan sebuah produk yang berfokus pada proses eKYC (*electronic know-your-customer*) berbasis *mobile apps* bernama InfinId. Pengembangan InfinId didasari pada permasalahan yang dialami pelanggan pada prinsip KYC saat pembukaan rekening pada kebanyakan bank saat ini. Salah satu transaksi yang harus menerapkan prinsip KYC adalah pembukaan rekening baru. Untuk mengenali pelanggan InfinId dirancang dengan beberapa modul yang terdiri dari *one-time password*, *optical character recognition*, *liveness*, *face recognition* dan *video call*. Modul *video call* akan menggantikan proses tatap muka langsung dengan *customer service*.

Video call diatur oleh Bank Indonesia melalui Peraturan Bank Indonesia Nomor 19/10/PBI/2017 tentang Penerapan Anti Pencucian Uang Dan Pencegahan Pendanaan Terorisme Bagi Penyelenggara Jasa Sistem Pembayaran Selain Bank Dan Penyelenggara Kegiatan Usaha Penukaran Valuta Asing Bukan Bank. Menurut pasal 20 perbankan diharuskan melakukan verifikasi terhadap identitas pengguna jasa dengan melakukan pemeriksaan kesesuaian terhadap:

- a. dokumen identitas yang diterbitkan instansi pemerintah;
- b. data dan informasi kependudukan yang ditatausahakan instansi pemerintah; dan/atau
- c. data biometrik atau data elektronik

Selengkapnya pada pasal 21 ayat 1 menyebutkan bahwa proses verifikasi yang dimaksud dalam pasal 20 dapat dilakukan dengan cara pertemuan langsung atau penggunaan cara lain. Hal ini dijelaskan lebih lanjut pada bagian penjelasan pasal demi pasal yang menyatakan pertemuan langsung dapat dilakukan melalui tatap muka secara langsung atau melalui sarana teknologi misalnya *video call*.

Pada dasarnya, *video call* akan menggantikan proses tatap muka langsung antara *customer service* dengan pelanggan. Karena itu skema dari sistem yang akan dikembangkan juga akan menyerupai skema pelayanan yang dijalankan oleh bank melalui tatap muka langsung. Pelanggan akan melakukan pendaftaran dan melakukan verifikasi data dengan *modul-modul* InfinId melalui *one-time password*, identifikasi tanda pengenalan dengan *optical character recognition*, deteksi *liveness*, dan *face recognition*. Setelah tahapan tersebut dilalui

maka pelanggan akan mendapat jatah antrian untuk melakukan *video call* dengan *customer service*. Pelanggan yang lebih dahulu melakukan verifikasi akan dilayani terlebih dahulu. antrian akan dapat dilayani oleh satu atau lebih *customer service*.

3.2 REST API

Application Programming Interface (API) adalah kumpulan fungsi perangkat lunak yang menyediakan serangkaian prosedur yang logis dan konsisten untuk pihak tertentu [1]. API menyediakan data yang dibutuhkan oleh komponen aplikasi dalam format yang telah disepakati sebelumnya semisal JSON atau XML. Dengan menggunakan API suatu aplikasi dapat mengakses data atau layanan yang disediakan oleh aplikasi lain tanpa harus mengetahui objek atau prosedur yang digunakan oleh aplikasi tersebut. API memberikan abstraksi tingkat tinggi yang memudahkan pengembangan program serta mendukung desain aplikasi yang terdistribusi.

REST adalah akronim dari *Representational State Transfer*. REST adalah gaya arsitektur perangkat lunak yang menjelaskan pedoman untuk membuat layanan *web* yang terukur. REST mencakup kumpulan aturan terkoordinasi yang dapat diterapkan pada desain komponen sistem terdistribusi untuk mencapai desain arsitektur yang lebih terpelihara [2]. REST bukan merupakan protokol atau standar melainkan suatu gaya arsitektur yang menyediakan pedoman perancangan API. Seperti gaya arsitektur lainnya, REST memiliki 6 batasan panduan yang harus dipenuhi agar suatu API dapat disebut mengaplikasikan prinsip REST secara penuh, atau biasa disebut RESTful. 6 Prinsip dari REST adalah *client-server*, *stateless*, *cacheable*, *uniform interface*, *layered system* dan *Code on demand*. Biasanya REST API memanfaatkan metode dari protokol HTTP.

Hypertext Transfer Protocol (HTTP) adalah sebuah protokol lapisan-aplikasi untuk mentransmisi dokumen yang dirancang untuk komunikasi antara *web browser* dan *web server*. Metode HTTP yang umum digunakan pada berbagai operasi adalah GET, POST, PUT dan DELETE. Untuk kegunaan masing-masing metode adalah sebagai berikut:

- a. GET: digunakan untuk menerima data dalam format yang sudah ditentukan.
- b. POST: digunakan untuk membuat atau mengirimkan data baru.
- c. PUT: digunakan untuk mengubah atau memperbarui data yang sudah ada.
- d. DELETE: digunakan untuk menghapus sebuah data.

Setelah permintaan dilakukan oleh *client*, maka *server* akan merespons permintaan dengan Kode respons HTTP yang menunjukkan apakah permintaan HTTP tertentu berhasil diselesaikan. Respons dikelompokkan dalam lima kelas: respons informasi, respons sukses, pengalihan, kesalahan *client*, dan kesalahan *server*.

3.3 Database

Database pada dasarnya dapat didefinisikan sebagai sekumpulan data terkait yang disimpan dengan cara terorganisir sehingga memungkinkan informasi untuk diambil secara logis sesuai kebutuhan [3]. Salah satu model *database* yang banyak digunakan saat ini adalah model relasional. Model relasional adalah model *database* yang mengatur data dalam bentuk relasi yang diwakili oleh suatu tabel. Suatu objek yang disimpan dalam database diwakili oleh baris, dan atribut yang membangun objek tersebut diwakili oleh kolom. Dalam satu tabel semua objek akan memiliki atribut tetap dan sama dengan nilai yang bervariasi. Misalkan tabel pegawai akan memiliki atribut 'id', 'nama', dan 'jabatan'. Tiap objek pegawai yang diwakili oleh baris akan memiliki atribut tersebut. Nama atribut tersebut akan menjadi nama kolom dari tabel. Setiap tabel akan memiliki aturan tertentu sehingga data antar tabel akan membentuk skema yang logis.

Database dengan model relasional harus memenuhi kaidah transaksi yang disebut ACID. ACID adalah singkatan dari empat properti yaitu *atomicity*, *consistency*, *isolation*, dan *durability*. Penjelasan dari properti ACID adalah sebagai berikut [4]:

1. *Atomicity*: Semua operasi dalam satu transaksi harus berhasil, atau transaksi akan dianggap gagal.
2. *Consistency*: *Database* akan berada dalam kondisi yang konsisten ketika transaksi dimulai dan berakhir.
3. *Isolation*: Transaksi akan dilakukan dengan seolah-olah transaksi tersebut adalah satu-satunya operasi yang dilakukan pada *database*
4. *Durability*: Setelah transaksi selesai, operasi tidak akan bisa dibatalkan.

Database dengan model relasional sebagian besar menggunakan *Standard Query Language* (SQL). SQL adalah bahasa yang dirancang pada awal 1970-an yang sangat cocok, untuk melakukan *query database*. Berikut merupakan contoh *query* untuk memasukkan data ke dalam *database* MySQL dengan bahasa SQL. Misal pengembang akan memasukkan data

tabel pegawai, di mana tabel tersebut memiliki atribut untuk id pegawai, nama, dan jabatan maka *query* yang digunakan untuk melakukannya adalah:

```
INSERT INTO pegawai VALUES ('15-1001', 'Smith', 'Marketing Supervisor');
```

3.4 Node.js

Dilansir dari *website* resminya, Node.js adalah sebuah JavaScript *runtime* yang dibangun dengan JavaScript *engine* V8 dari Chrome [5]. Node.js awalnya ditulis oleh Ryan Dahl pada tahun 2009. Saat awal dirilis Node.js hanya mendukung sistem operasi Linux dan Mac OS X. Pengembangan dan pemeliharaan Node.js dipimpin oleh Dahl dan kemudian disponsori oleh Joyent. JavaScript sendiri merupakan bahasa pemrograman sisi *client* yang paling populer. Pengembangan aplikasi berbasis *web* pasti tidak terhindarkan dari penggunaan JavaScript. Dengan berjalannya JavaScript di sisi *server* melalui Node.js berikut merupakan keuntungan yang dapat diperoleh oleh pengembang:

1. Pengembang hanya memakai satu bahasa untuk mengembangkan aplikasi yang bersifat *client-server* sehingga mengurangi *learning curve* untuk mempelajari bahasa *server* yang lain.

2. *Sharing* kode antara *client* dan *server* atau biasa dikenal dengan *code reuse*. JavaScript secara *native* mendukung JSON yang merupakan standar transfer data yang banyak dipakai saat ini. Hal ini akan memudahkan pemrosesan data dari pihak ketiga dengan menggunakan Node.js.

3. Node.js memakai JavaScript engine V8 yang selalu mengikuti perkembangan standar ECMAScript, sehingga setiap web browser akan mendukung serangkaian fitur dari Node.js.

3.5 Express

Express adalah kerangka kerja aplikasi *web* Node.js yang minimal dan fleksibel yang menyediakan serangkaian fitur yang kuat untuk mengembangkan aplikasi berbasis *web* [6]. Express didirikan oleh TJ Holowaychuk. Menurut repositori GitHub, rilis pertama Express adalah pada 22 Mei 2010 dengan Versi 0.12. Pada Juni 2014, hak pengelolaan Express diperoleh oleh StrongLoop. StrongLoop lalu diakuisisi oleh IBM pada September 2015, dan pada Januari 2016, IBM mengumumkan akan menempatkan Express di bawah pengawasan Node.js Foundation.

Express memfasilitasi pengembangan aplikasi *web* Node.js dengan minimal dan fleksibel. Berikut ini adalah beberapa fitur inti dari kerangka kerja Express:

1. Memungkinkan pengembang untuk menggunakan *middlewares* untuk menanggapi HTTP *request*.
2. Menentukan *routing* yang digunakan untuk API berdasarkan metode HTTP dan URL.
3. Memungkinkan untuk memproses laman HTML secara dinamis berdasarkan pada argumen yang dikirim pada *template*.

Middleware Express dapat digunakan untuk menambahkan fitur *cookie*, sesi, serta mengatur parameter dari HTTP *request*. Metode HTTP yang sering digunakan untuk menyusun API dan dapat diatur oleh Express mencakup GET, POST, PUT dan DELETE. Pemrosesan laman HTML oleh Express mencakup pengaturan mengenai *template engine* yang digunakan, konfigurasi tempat file template disimpan, dan template yang digunakan untuk memberikan respons dari HTTP *request*. Express mendukung penggunaan mekanisme basis data apa pun yang didukung oleh Node.js baik itu relasional maupun non-relasional.

3.6 Socket.io

Socket.io adalah *library* JavaScript untuk membangun aplikasi berbasis *web* yang memungkinkan komunikasi dua arah bersifat *real-time* antara *client* dan *server* [7]. Socket.io memungkinkan komunikasi *client-server* yang bersifat *real-time*, *bidirectional* dan *event-based*. Socket.io memiliki dua bagian, yaitu *library* sisi *client* yang berjalan di *web browser*, dan *library* sisi *server* untuk Node.js. Kedua komponen memiliki API yang identik yaitu [8]:

1. *Event emitter*, fitur yang akan mengirim data/pesan dengan topik tertentu.
2. *Event listener*, fitur yang berfungsi sebagai penerima data/pesan dari suatu topik, sehingga antara *client* dengan *server* dapat menerima pesan dari kedua pihak.
3. *Broadcast*, fitur untuk mengirim data/pesan seperti *event emitter*, namun *broadcast* mengirim ke semua alamat penerima yang aktif kecuali pengirimnya sendiri.

Fitur lain dari Socket.io yang sering digunakan adalah *namespaces* dan *room*. Berikut merupakan penjelasannya:

1. *Namespaces*, merupakan fitur yang memungkinkan pengembang untuk menetapkan *endpoint* atau jalur koneksi yang berbeda. fitur ini berguna untuk meminimalkan jumlah koneksi TCP dan memisahkan fungsionalitas aplikasi dengan pemisahan saluran komunikasi. *Namespaces* dibuat di sisi *server*, *client* dapat bergabung dengan mengirimkan permintaan ke *server*. *Namespace* default pada koneksi Socket.io adalah '/' atau *base url* dari koneksi Socket.io

2. *Rooms*, merupakan fitur untuk membuat membagi *namespace* menjadi beberapa *channel* yang disebut sebagai *room*. Pada setiap *namespace*, *room* dapat dibuat oleh setiap *client* untuk diikuti ataupun ditinggalkan oleh *client* yang terhubung dengan *namespace* yang sama.

3.7 JSON Web Token

JSON Web Token (JWT) adalah standar terbuka (RFC 7519) yang mendefinisikan cara yang ringkas untuk mentransmisikan informasi antar pihak secara aman sebagai objek JSON [9]. Informasi yang ditransmisikan menggunakan JWT akan menggunakan *digital signature* menggunakan algoritma HMAC atau pasangan kunci publik-pribadi.

JWT memiliki bentuk token yang telah memiliki *digital signature*. Token yang memiliki *digital signature* dapat memverifikasi integritas klaim yang terkandung di dalamnya. JWT biasanya digunakan untuk skenario sebagai berikut:

1. Otorisasi: Ini adalah skenario paling umum untuk menggunakan JWT. Setelah pengguna masuk, setiap permintaan berikutnya yang menyertakan JWT, memungkinkan pengguna untuk mengakses rute, layanan, dan sumber daya yang diizinkan dengan token yang memiliki *digital signature*.

2. Pertukaran Informasi: JSON Web Token adalah cara yang baik untuk mentransmisikan informasi antar pihak secara aman. Karena JWT memiliki *digital signature*, penerima dapat mengetahui pengirim dari token. Selain itu, karena *digital signature* disusun menggunakan *header* dan *payload*, penerima juga dapat memverifikasi bahwa informasi dalam token belum rusak. Dalam bentuknya yang ringkas, JSON Web Token terdiri dari tiga bagian yang dipisahkan oleh titik (.), yaitu *Header*, *Payload*, dan *Signature*.

3.8 Knex.js

Knex.js adalah *query builder* untuk bahasa SQL untuk *database* berbasis PostgreSQL, MSSQL, MySQL, MariaDB, SQLite3, Oracle, dan Amazon Redshift yang dirancang agar fleksibel, dan portabel [10]. Fitur-fiturnya mencakup *query and schema builders*, *transaction support*, *connection pooling* dan standarisasi respons antara sistem *database management system* yang berbeda. Berikut merupakan contoh sederhana *query builder* Knex.js untuk mengambil data dari suatu tabel untuk nilai tertentu di kolom tertentu:

```
knex('users').where('id', 1)
```

query builder tersebut akan diubah oleh Knex.js menjadi *query* SQL seperti berikut

```
select * from `users` where `id` = 1
```

Keunggulan Knex.js dari penulisan *query* SQL secara *native* adalah Knex.js memungkinkan penggunaan sintaks JavaScript untuk membuat *query*. Hal ini mengingat penulisan *query* SQL secara *native* dapat menjadi sangat sulit untuk dipahami dan dikelola.

BAB IV METODE KERJA PRAKTIK

4.1 Alat dan Bahan Kerja Praktik

Alat dan Bahan yang digunakan selama pelaksanaan kerja praktik adalah sebagai berikut:

1. Laptop dengan spesifikasi sistem operasi Windows 10 Home, RAM 8GB, dan *processor* Intel core i7 7500u.
2. *Software* Visual Studio Code untuk *code editor*.
3. *Software* Postman yang digunakan untuk menguji *endpoint* yang dibuat.
4. *Web browser* Google Chrome untuk menguji koneksi dari Socket.io.

4.2 Alur Kerja Praktik

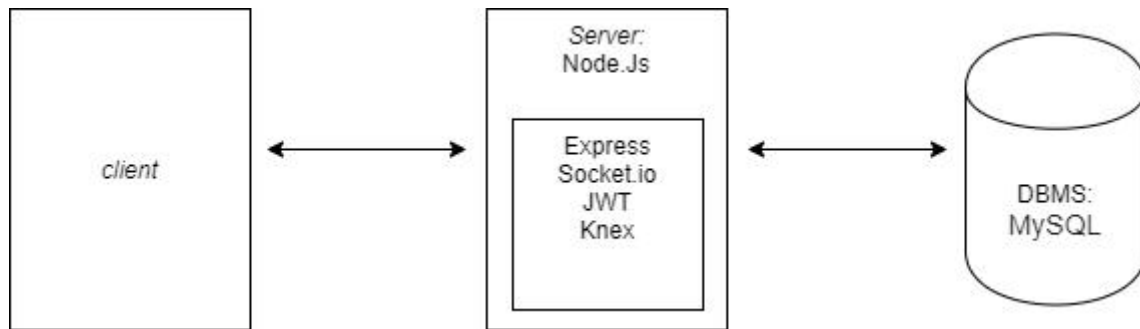
Tabel 4.1 Alur Kerja Praktik

Minggu	Pekerjaan
1	Mempelajari NodeJs beserta <i>library</i> yang digunakan (Knex.js, Socket.io, Express dan JWT). Mengatur <i>environmet</i> pengembangan dari Visual Studio Code dan <i>Git-repository manager</i> dengan GitLab.
2	Membuat skema awal dari <i>database</i> yang akan digunakan oleh sistem dan struktur dari sistem yang akan dikerjakan. Mulai mencoba menggunakan Express dan Knex.js untuk membuat REST API.
3	Mulai menerapkan JSON <i>Web Token</i> untuk otorisasi dari <i>request</i> REST API dan menggunakan Socket.io untuk membuat respons yang bersifat <i>real-time</i> dari komunikasi <i>client-server</i> .
4	Pengujian dari sistem yang dibuat beserta perbaikan dari skema, struktur maupun fungsionalitas dari sistem.

BAB V HASIL DAN PEMBAHASAN

5.1 Arsitektur Sistem

Sistem antrian yang dikembangkan oleh penulis berbasis *client-server* dengan gambaran seperti pada Gambar 5.1

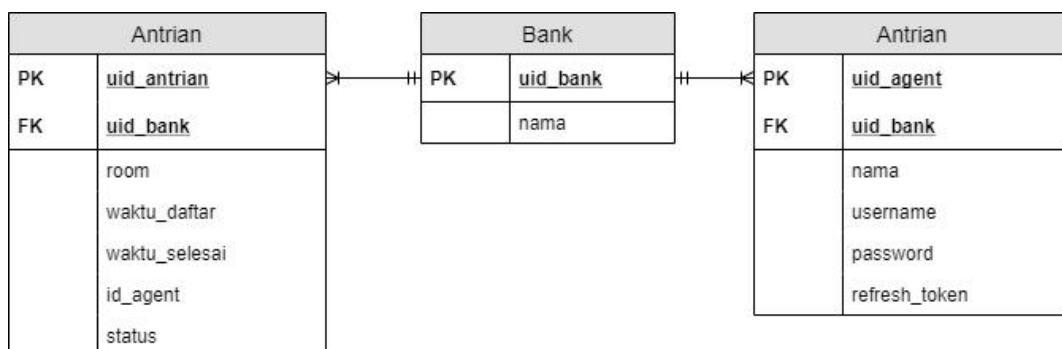


Gambar 5.1 Arsitektur Sistem Antrian InfinId

Pada Gambar 5.1 di atas sistem informasi akan ditulis dengan Node.js dengan memanfaatkan *library* Express, Socket.io, JWT dan Knex.js, serta untuk penyimpanan data akan dikelola dengan MySQL.

5.2 Skema Database

Terdapat 3 tabel yang akan digunakan oleh sistem antrian. Tabel bank berisi *unique id* dari tiap bank dan nama dari bank tersebut. Tabel agent berisi data *customer service* dari bank yang bersangkutan yang mencakup *unique id* dari *customer service*, nama, *username*, *password*, dan *refresh token* yang akan digunakan untuk memperbarui JWT. Tabel antrian berisi data antrian pelanggan dari bank bersangkutan yang mencakup *unique id* dari antrian, *room video call*, waktu daftar antrian, waktu antrian selesai dilayani, *unique id* dari *customer service* yang melayani, dan status antrian. Berikut merupakan skema dari ketiga tabel tersebut:

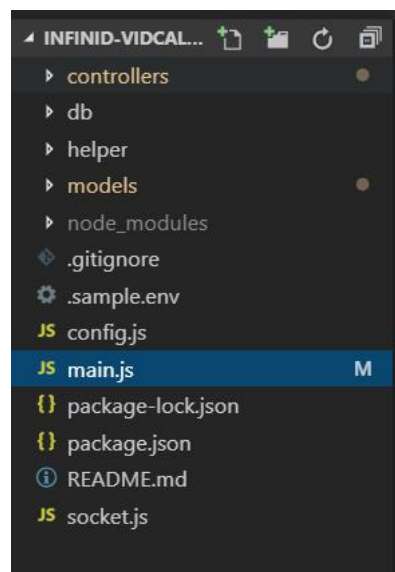


Gambar 5.2 Skema Database Sistem Antrian InfinId

Skema tersebut akan disimpan ke dalam *database* sebelum sistem dijalankan. Untuk itu skema ditulis dalam *query* SQL standar yang dapat diimpor oleh *server database*. *Database* yang digunakan oleh antrian dinamai dengan *antrianvidcall*. Untuk *query* SQL dari tiap tabel dapat dilihat pada lampiran L5.

5.3 Struktur Sistem

Sistem antrian yang dikembangkan oleh penulis memisahkan antara logika dari operasi *database* dan pemrosesan dari *request* API. Models akan berisi fungsi yang digunakan sistem untuk mengambil, menyisipkan, dan memperbarui informasi dalam *database*. Sedangkan *controller* berfungsi sebagai pengatur proses *request* dari API. Berikut merupakan struktur dari sistem yang dikembangkan penulis.



Gambar 5.3 Struktur *Project* Sistem Antrian InfinId

Bagian lain dari sistem mencakup *helper*, *db* dan *node_modules*. Folder *helper* berisi fungsi yang sering digunakan *controller* pada setiap *endpoint* API. Ada 2 *helper* yang digunakan oleh sistem yaitu untuk verifikasi status *login* dan operasi *JSON Web Token*. Folder *db* berisi skema *database* yang akan digunakan oleh sistem. Folder *node_modules* berisi *library* yang diunduh dari *node package manager*.

Untuk *file* berekstensi JavaScript di luar folder yang telah disebutkan adalah *main.js*, *config.js* dan *socket.js*. *Main.js* adalah *entry point* dari sistem dimana *main.js* adalah *file* pertama yang akan dieksekusi saat menjalankan sistem. *Config.js* berisi pengaturan *database* yang akan digunakan oleh sistem dan *secret-key* yang digunakan oleh *JWT*. *Socket.js* berisi inisialisasi penggunaan *library* *Socket.io* yang akan digunakan oleh sistem.

5.4 Helper

Ada 2 *helper* yang digunakan oleh sistem yaitu, verifikasi status *login* dan operasi JSON *Web Token*. Berikut merupakan pembahasan dari setiap *file helper*.

5.4.1 Helper JSON Web Token

Terdapat 4 fungsi yang ada pada *helper JSON Web Token*, yaitu *createAccessToken*, *createRefreshToken*, *verifyAccessToken*, dan *verifyRefreshToken*. Fungsi *createAccessToken*, *createRefreshToken*, berperan dalam membuat *signature* yang merupakan gabungan dari *header* yang disandikan, *payload* yang disandikan, *secret-key* yang ditentukan pembuat *signature*, dan terakhir algoritma yang ditentukan dalam *header*. Adapun *secret-key* yang digunakan untuk membuat *signature* disimpan dalam *file config.js*. Untuk masa berlaku *access token* diatur menjadi 15 menit, sedangkan untuk *refresh token* diatur menjadi 30 hari. Hal ini berarti setelah melakukan *login*, pengguna akan mendapatkan *access token* untuk memverifikasi status *login* yang berlaku selama 15 menit, beserta *refresh token* yang digunakan untuk memperbarui *access token*. Setelah 15 menit berlalu, pengguna harus melakukan *request* kepada sistem untuk membuat *access token* yang baru. Jika *refresh token* yang dimiliki pengguna masih berlaku, maka sistem akan membuat *access token* yang baru. Sebaliknya, jika *refresh token* yang dimiliki pengguna sudah tidak berlaku, pengguna akan diminta melakukan *login* ulang untuk mendapat *access token* dan *refresh token* yang baru. Untuk pengecekan apakah *access token* dan *refresh token* masih berlaku menggunakan fungsi *verifyAccessToken*, dan *verifyRefreshToken*. Untuk *source code* dari *helper JSON web token* dapat dilihat pada lampiran L.6.

5.4.2 Helper Verifikasi Status Login

Helper verifikasi status *login* memiliki fungsi *access* yang digunakan untuk mengecek masa berlaku *access token* yang terdapat pada *request header*. Jika token masih berlaku, fungsi akan mengembalikan data *payload* dari token yang berisi *unique id* dari *customer service* beserta waktu kadaluwarsa dari token. Untuk *source code* dari *helper verifikasi status login* dapat dilihat pada lampiran L.7.

5.5 Model

Folder *models* akan berisi *file* yang digunakan sistem untuk melakukan operasi pada tiap tabel *database*. Terdapat 3 *file* model yang mewakili 3 tabel yang ada pada *database*. Untuk

pengaturan koneksi *database* disimpan pada *file* `index.js`. Berikut merupakan pembahasan dari masing-masing *file*.

5.5.1 Index.js

Tabel 5.1 *Source Code* Pengaturan Koneksi *Database* dengan Knex.js

Index.js
<pre>const knex = require('knex'); const db = require('./../config').DATABASE; var database = knex({ client: 'mysql', connection: db, pool: { min: 0, max: 7 } }); module.exports = { database };</pre>

File ini berisi konfigurasi dari koneksi *database* yang akan digunakan pada tiap *file* model. Koneksi akan diatur oleh *library* Knex.js, dengan jenis *database* MySQL dengan pengaturan pooling maksimal 7 koneksi. Untuk konfigurasi nama, *username*, *password*, *host*, dan *port* dari *database* disimpan pada *file* `config.js`.

5.5.2 Bank Model

Bank model memiliki fungsi insert yang digunakan untuk mendaftarkan bank yang dapat menggunakan sistem antrian. Parameter yang dibutuhkan untuk mendaftarkan bank adalah nama bank. Untuk *source code* dari bank model dapat dilihat pada lampiran L.8.

5.5.3 Agent Model

Terdapat 4 fungsi yang ada pada agent model, yaitu insert, getAccount, setRefreshToken, dan getRefreshToken. Insert digunakan untuk menambah *customer service* dari bank yang dapat mengakses antrian. Parameter yang dibutuhkan saat menambah *customer service* adalah *username*, nama, *password*, dan id bank. *Password* dari *customer service* disimpan dengan enkripsi hash dengan menggunakan *library* bcrypt. Fungsi getAccount akan mengambil data *customer service* dengan *username* tertentu. Fungsi setRefreshToken, dan getRefreshToken masing-masing digunakan untuk menyimpan dan mengambil data *refresh token* dari tabel agent dengan parameter *unique id* dari *customer service*. Untuk *source code* dari agent model dapat dilihat pada [lampiran L.9](#).

5.5.4 Antrian Model

Terdapat 5 fungsi yang ada pada antrian model, yaitu, *insertQueue*, *callQueue*, *endQueue*, *getAgent*, dan *count*. Fungsi *insertQueue* digunakan untuk menambah antrian dengan parameter *unique id* dari bank beserta antrian. Saat antrian didaftarkan, waktu pelanggan mendaftar juga di rekam. Fungsi *getAgent* digunakan untuk mengecek apakah ada data *customer service* dengan *unique id* tertentu yang sedang ada dalam panggilan.

Fungsi *callQueue* digunakan untuk mengambil antrian selanjutnya yang belum dipanggil berdasarkan waktu pendaftaran dan mengubah statusnya menjadi sedang dipanggil. Transaksi *database* yang terjadi adalah, pertama pengambilan data antrian dari bank tertentu lalu pengubahan status antrian.

Fungsi *endQueue* digunakan untuk mengakhiri panggilan. Transaksi *database* yang terjadi adalah, pertama pengecekan status *customer service* yang akan mengakhiri pelayanan berdasarkan *unique id*. Jika *Customer service* tersebut memiliki status sedang melakukan panggilan, maka status antrian yang sedang dilayani oleh *customer service* tersebut akan diganti menjadi sudah dipanggil. Fungsi *count* digunakan untuk menghitung banyak antrian yang masih belum dilayani. *Source code* dari antrian model terdapat pada lampiran L.10.

5.6 Controller

Folder *Controller* akan berisi *file* yang berfungsi sebagai pengatur proses *request* dari API. Ada 3 *file* yang mewakili 3 objek terkait sistem antrian yaitu bank, *customer service* (diwakili oleh agent *controller*) dan antrian. Validasi *request body* dari API akan dilakukan dengan menggunakan *library* JOI. Berikut merupakan pembahasan dari masing-masing *file*.

5.6.1 Bank Controller

Bank *controller* memiliki API register dengan metode HTTP POST yang digunakan untuk mendaftarkan bank yang dapat menggunakan sistem antrian. Parameter pada *request body* yang dibutuhkan untuk mendaftarkan bank adalah nama bank. Untuk *source code* dari bank *controller* dapat dilihat pada lampiran L.11.

5.6.2 Agent Controller

Agent *controller* memiliki 3 API, yaitu register, login dan *refresh_token*. API register memiliki metode HTTP POST yang digunakan untuk mendaftarkan *customer service* dari bank. Parameter pada *request body* yang dibutuhkan untuk mendaftarkan *customer service* adalah *username*, nama, *password*, dan *id_bank*. Pertama sistem akan mengecek apakah

username yang akan didaftarkan tersedia dengan fungsi *getAccount* dari agent model. Jika *username* tersedia maka data *customer service* akan dimasukkan ke dalam *database* melalui fungsi *insert*.

API login memiliki metode HTTP POST yang digunakan untuk mendapatkan *access token* dan *refresh token*. Parameter pada *request body* yang dibutuhkan untuk login adalah *username* dan *password*. Pertama sistem akan mengecek apakah *username* pada *request body* ada pada *database* dengan fungsi *getAccount* dari agent model. Jika *username* memang ada maka *password* pada *request body* akan dibandingkan dengan *password* pada *database*. Jika keduanya cocok, maka *customer service* akan mendapatkan *access token* dan *refresh token* yang di buat melalui *JSON Web Token helper*. *Refresh token* yang didapat akan disimpan dalam *cookie*.

API *refresh_token* memiliki metode HTTP POST yang digunakan untuk mendapatkan *access token* baru melalui *refresh token*. Pertama *Refresh token* dalam *cookie* akan dicek masa berlakunya. Jika *Refresh token* masih berlaku maka *customer service* akan mendapatkan *access token* baru yang di buat melalui *JSON Web Token helper*. Untuk *source code* dari agent *controller* dapat dilihat pada lampiran L.12.

5.6.3 Antrian Controller

Antrian controller memiliki 4 API, yaitu *register*, *nextAntrian*, *endAntrian* dan *count*. API pada Antrian controller memiliki 2 skema respons, yaitu melalui respons HTTP *request* dan melalui koneksi *socket*. Saat kondisi antrian mengalami perubahan, *customer service* akan memperoleh informasi pergantian kondisi antrian melalui koneksi *socket*

API *register* memiliki metode HTTP POST yang digunakan untuk mendaftarkan antrian dari bank. Parameter pada *request body* yang dibutuhkan untuk mendaftarkan antrian adalah *id_antrian*, dan *id_bank*. Jika antrian berhasil didaftarkan maka, *customer service* akan memperoleh informasi pergantian kondisi antrian melalui koneksi *socket*. Untuk API *nextAntrian*, *endAntrian* dan *count*, *request* diharuskan memiliki *header token* untuk memverifikasi *customer service* yang melakukan pelayanan pada antrian.

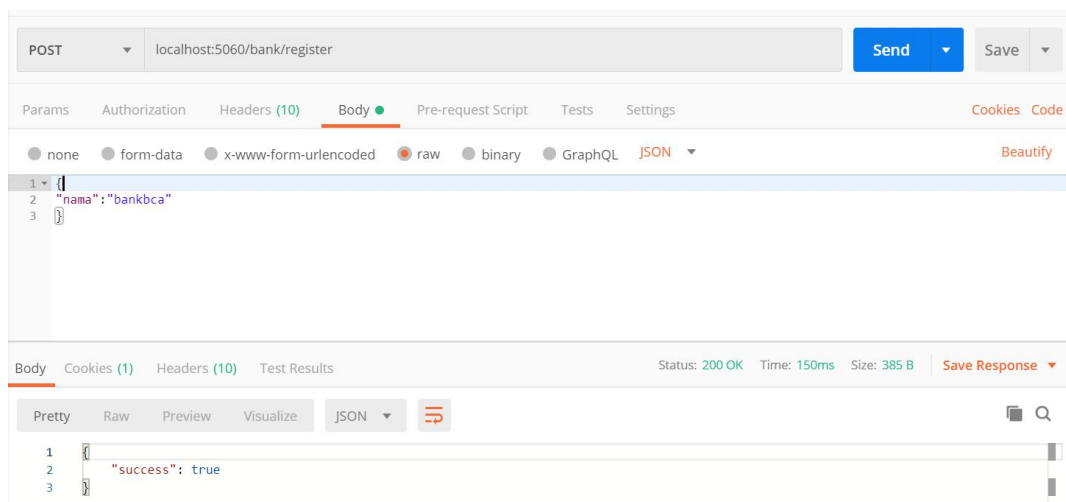
API *nextAntrian* memiliki metode HTTP POST yang digunakan untuk mengambil antrian selanjutnya. Parameter pada *request body* yang dibutuhkan untuk memanggil antrian adalah *id_bank*, *room*, dan *room_password*. Ada 2 tahap pengecekan sebelum *customer service* dapat memanggil antrian. Pertama *customer service* tidak boleh sedang dalam panggilan, dan kedua antrian harus masih ada. Jika kedua tahap tersebut berhasil dilewati,

maka data status antrian pada *database* akan diganti dari menunggu menjadi sedang dipanggil dan pelanggan akan diberitahu melalui koneksi *socket* mengenai *room* dan *room password* dari *video call*.

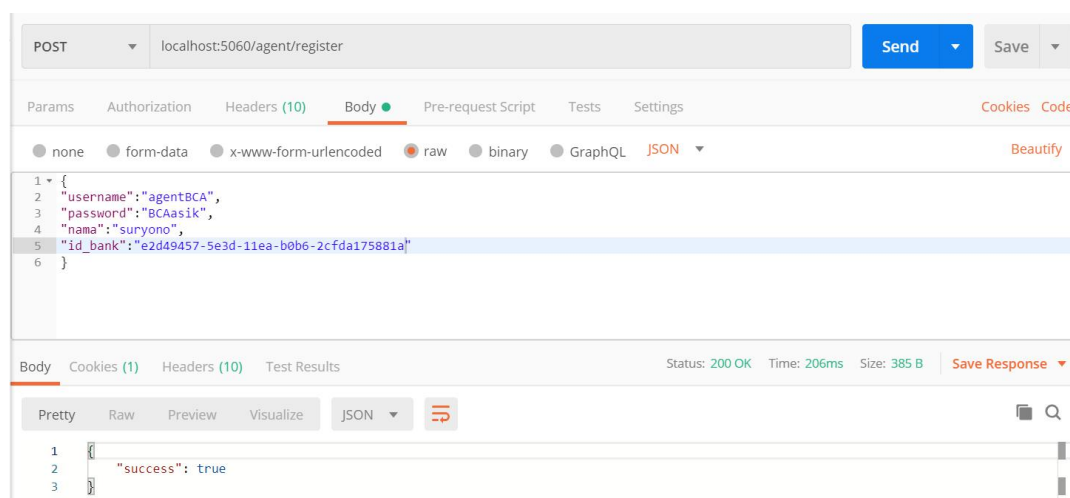
API endAntrian memiliki metode HTTP POST yang digunakan untuk mengakhiri *video call*. Jika *video call* berhasil diakhiri, maka pelanggan akan diberitahu melalui koneksi *socket*. API count memiliki metode HTTP GET yang digunakan untuk mendapatkan jumlah antrian yang belum dilayani dari suatu bank. Parameter yang dibutuhkan untuk mengetahui jumlah antrian yang belum dilayani adalah *unique id* dari bank. Untuk *source code* dari antrian *controller* dapat dilihat pada lampiran L.12.

5.7 Hasil Pengujian Tiap API

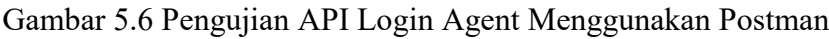
Berikut merupakan *screenshot* hasil pengujian API menggunakan Postman dan Google Chrome

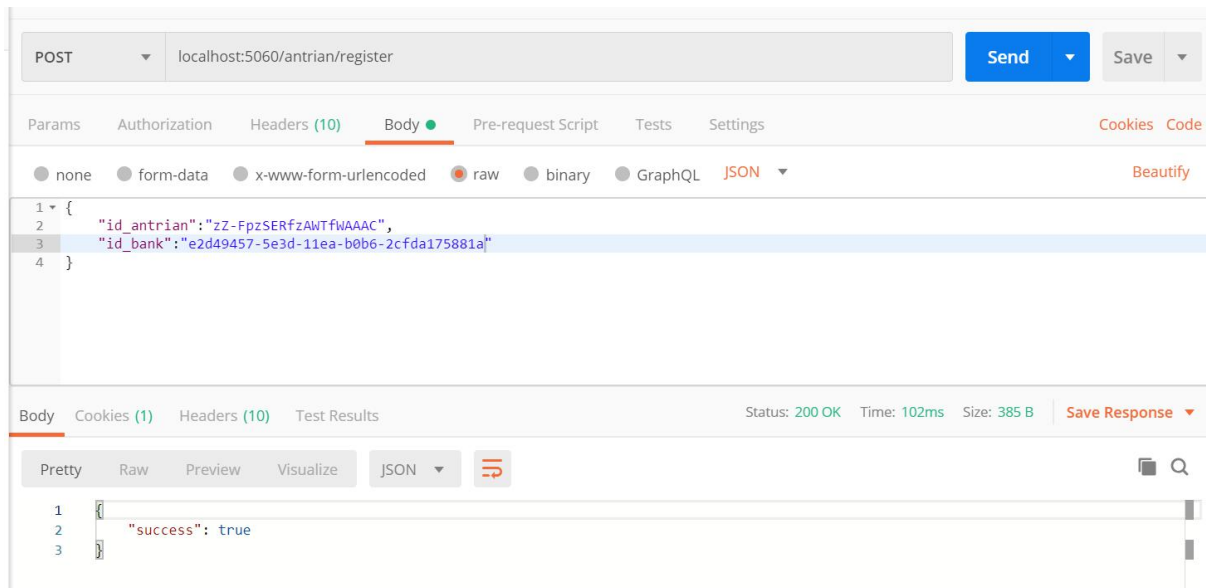


Gambar 5.4 Pengujian API Register Bank Menggunakan Postman

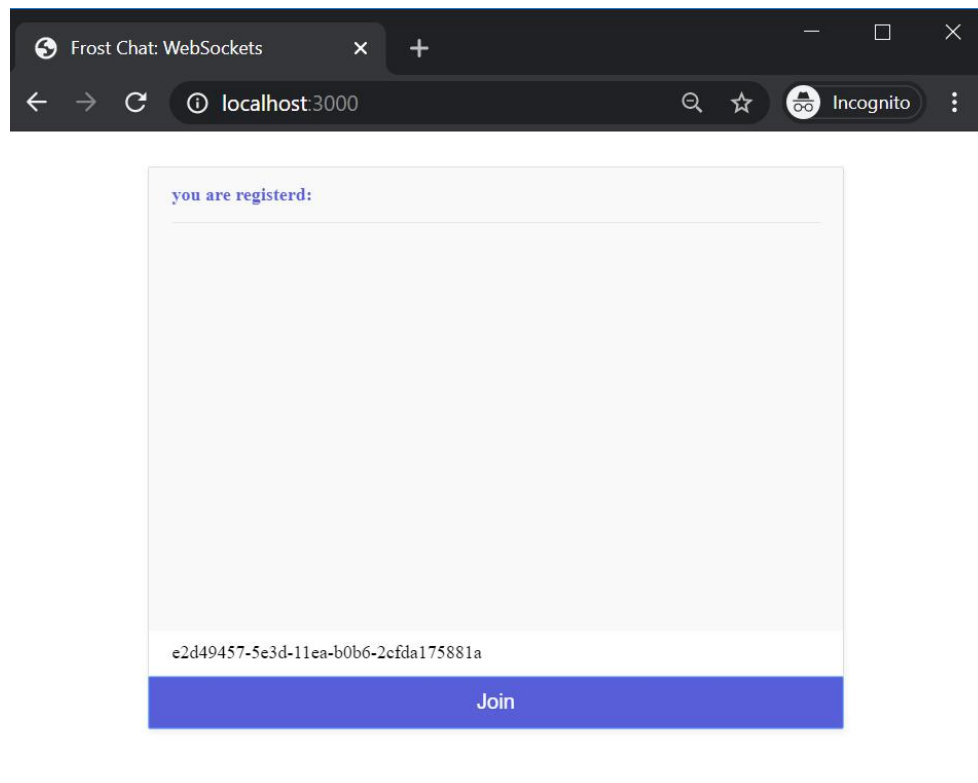


Gambar 5.5 Pengujian API Register Agent Menggunakan Postman

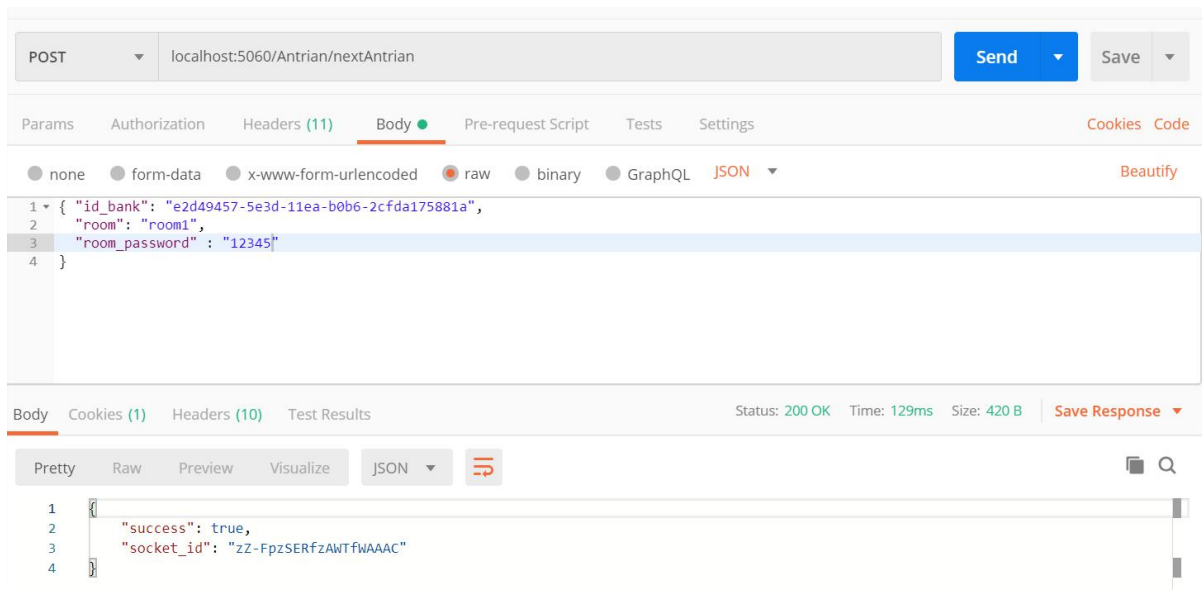




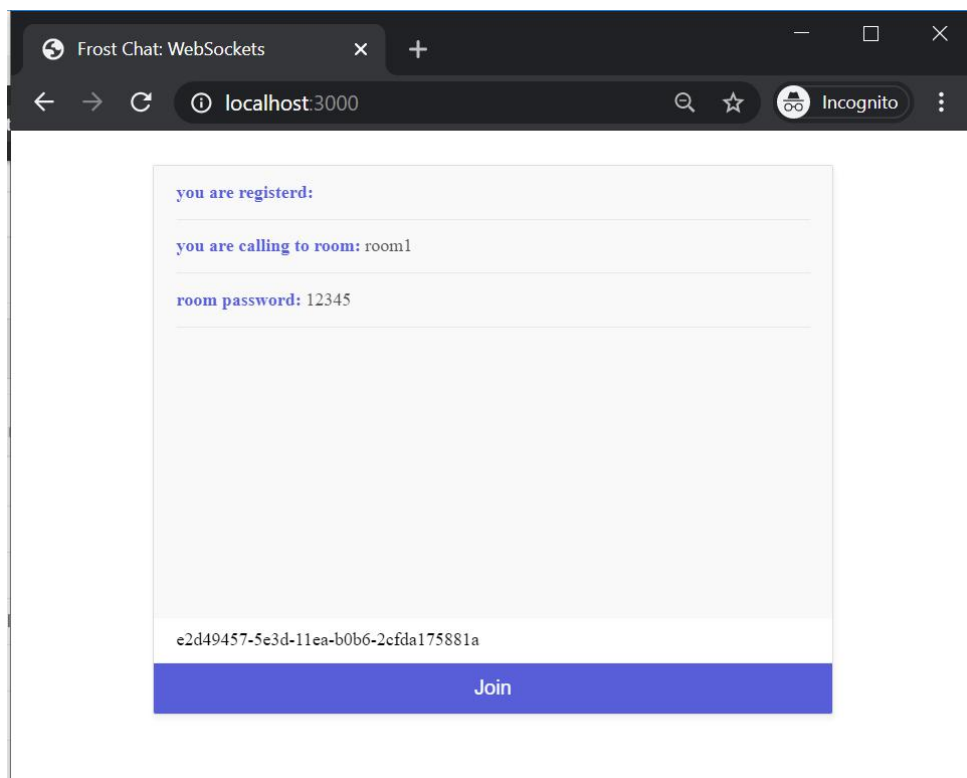
Gambar 5.8 Pengujian API Register Antrian Menggunakan Postman



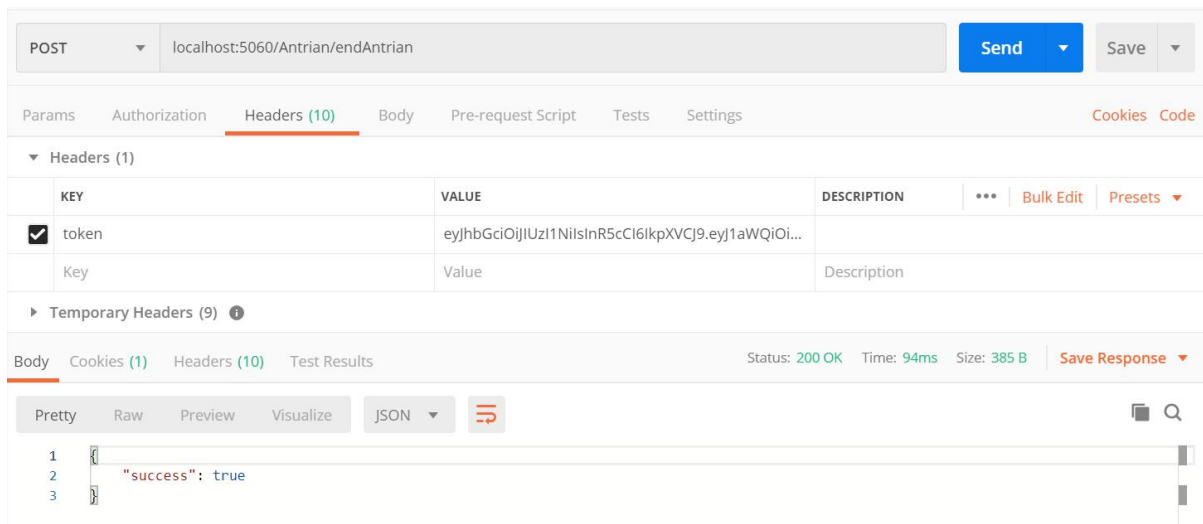
Gambar 5.9 Pengujian API Register Antrian Menggunakan Google Chrome



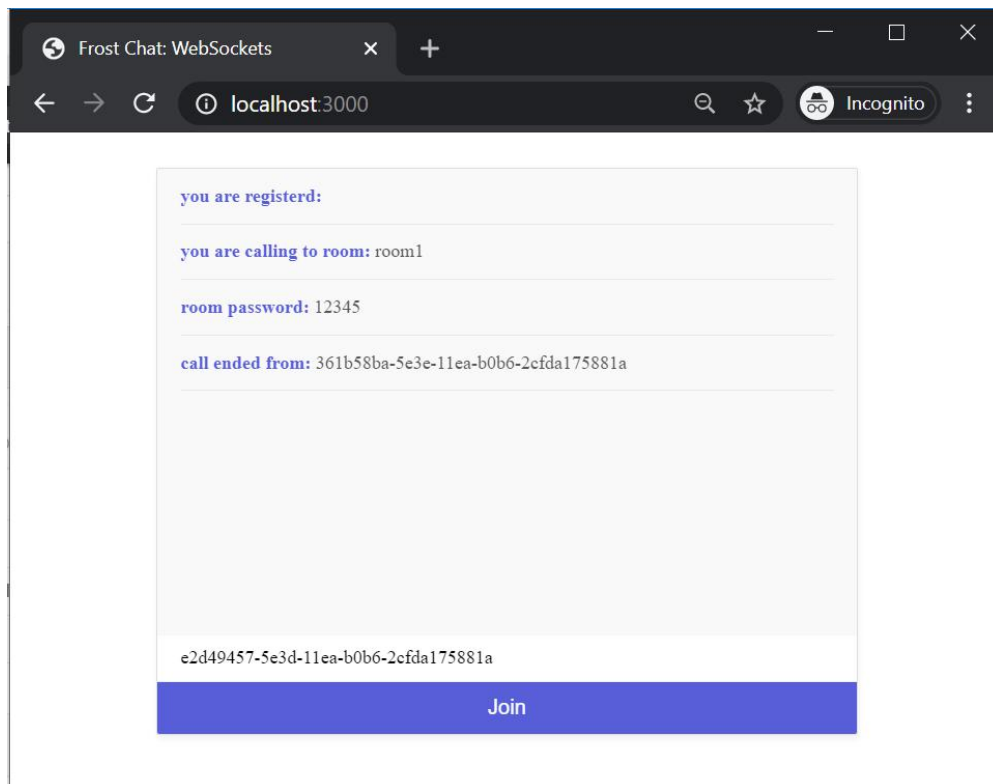
Gambar 5.10 Pengujian API Next Antrian Menggunakan Postman



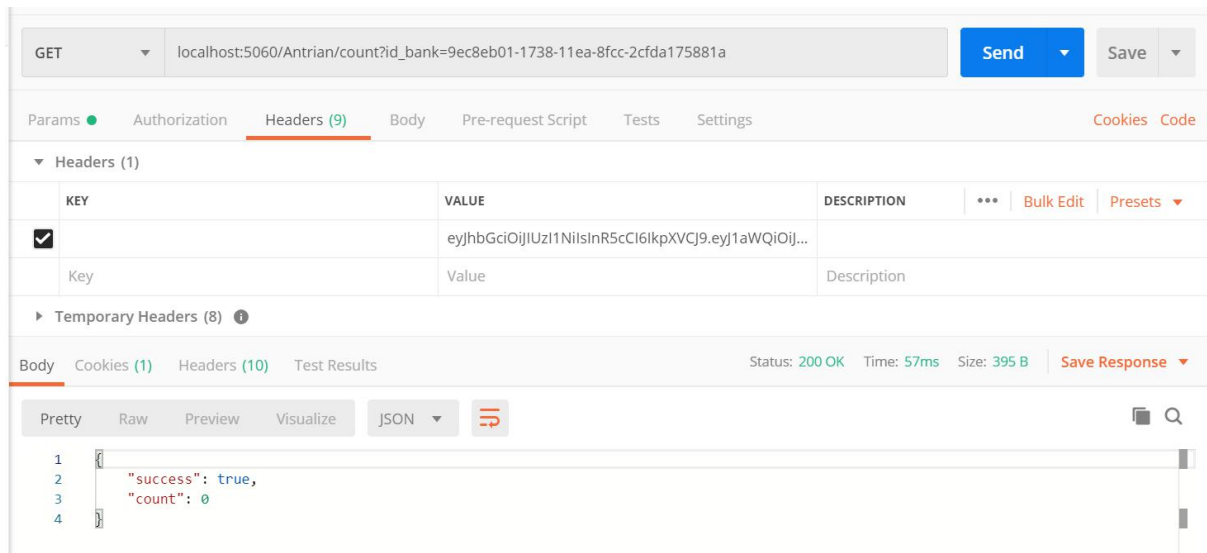
Gambar 5.11 Pengujian API Next Antrian Menggunakan Google Chrome



Gambar 5.12 Pengujian API End Antrian Menggunakan Postman



Gambar 5.13 Pengujian API End Antrian Menggunakan Google Chrome



Gambar 5.14 Pengujian API Count Antrian Menggunakan Postman

BAB VI KESIMPULAN DAN SARAN

6.1 Kesimpulan

1. Modul *video call* pada produk InfinId membutuhkan sistem antrian untuk mempermudah pelayanan *customer service*.
2. Sistem antrian dapat berjalan secara *real-time* dengan menggunakan *library* Socket.io dari Node.js

6.2 Saran

1. Diperlukan tes integrasi lebih lanjut dari sistem antrian dengan modul *video call* maupun modul lainnya
2. Sistem antrian masih perlu dilengkapi dari beberapa aspek, seperti data bank, maupun API yang memberikan informasi mengenai posisi antrian pelanggan

DAFTAR PUSTAKA

- [1] T. Perelmuter, "Understanding RESTful APIs and documenting them with Swagger", 2018.
- [2] V. Kumari, "Web Services Protocol: SOAP vs REST", *International Journal of Advanced Research in Computer Engineering & Technology*, vol. 4, no. 5, p. 2467, 2015.
- [3] D. Morley and C. Parker, *Understanding Computers: Today and Tomorrow*, Comprehensive, 15th ed. Boston: Cengage Learning, 2015.
- [4] N. Jatana, S. Puri, M. Ahuja, I. Kathuria and D. Gosain, "A Survey and Comparison of Relational and Non-Relational Database", *International Journal of Engineering Research & Technology (IJERT)*, vol. 1, no. 6, p. 1, 2012.
- [5] "Node.js", Node.js, 2020. [Online]. Available: <https://nodejs.org/en/>. [Accessed: 03- Mar- 2020].
- [6] "Node.js - Express Framework - Tutorialspoint", Tutorialspoint.com. [Online]. Available: https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm. [Accessed: 03- Mar- 2020].
- [7] "Socket.IO Tutorial - Tutorialspoint", Tutorialspoint.com. [Online]. Available: <https://www.tutorialspoint.com/socket.io/index.htm>. [Accessed: 01- Mar- 2020].
- [8] A. Afian, "Berkenalan dengan Socket.io", Medium, 2019. [Online]. Available: https://medium.com/@afifafian_/berkenalan-dengan-socket-io-1b9db2d983f6. [Accessed: 03- Mar- 2020].
- [9] "JWT.IO - JSON Web Tokens Introduction", Jwt.io. [Online]. Available: <https://jwt.io/introduction/>. [Accessed: 03- Mar- 2020].
- [10] "Knex.js - A SQL Query Builder for Javascript", Knexjs.org. [Online]. Available: <http://knexjs.org/>. [Accessed: 02- Mar- 2020].

LAMPIRAN

Lampiran L.1 Surat keterangan magang di PT. Mechlab Teknologi Indonesia

MechLab
MECHLAB TEKNOLOGI INDONESIA

Asem Kranji St K-7, Bulaksumur
Sleman, Yogyakarta 55284

cs@mechlabb.com
mechlabb.com

Yogyakarta, 6 November 2019

No : 009/I/XI/19
Lampiran : -
Perihal : **Persetujuan Kerja Praktik**

Kepada Yth
Ketua Departemen Teknik Elektro dan Teknologi Informasi
Fakultas Teknik
Universitas Gadjah Mada
Di tempat

Menunjuk surat Bapak No : 2568/H1.17/TKE/KM/2019 tanggal 4 November 2019 sesuai dengan pokok surat, dengan ini kami sampaikan jadwal pelaksanaan Kerja Praktik di PT Mechlab Teknologi Indonesia sebagaimana nama tersebut dibawah ini :

NO	NAMA	NIM	JURUSAN
1	Baskoro Adi Wicaksono	16/395837/TK/44679	S1 – Teknologi Informasi

Yang sudah dapat kami setuju dengan keterangan sebagai berikut :


Mulai tanggal : 7 November 2019
s.d/ tanggal : 7 Desember 2019
Posisi : Back End Programmer (Web- based)


Dengan persyaratan sebagai berikut :

1. Mahasiswa yang bersangkutan bersedia mengikuti peraturan- peraturan yang berlaku di perusahaan
2. Jam Kerja Praktik dimulai pukul 08.00 sampai dengan 17.00 WIB Senin s/d Jumat
3. Selama melaksanakan Kerja Praktik di PT. MechLab Teknologi Indonesia wajib menggunakan pakaian bebas rapi, bersepatu dan menggunakan id card yang telah diberikan oleh perusahaan.
4. Setelah akhir Kerja Praktik diwajibkan menyerahkan laporan dalam bentuk *soft file*.

Demikian kami sampaikan dan atas kerjasamanya kami ucapkan terima kasih

PT. MECHLAB TEKNOLOGI INDONESIA
Chief Executive Officer


Dadang Hidayat



Lampiran L.2 Foto penulis beserta tim Mechatronic Laboratory



Lampiran L.3 Suasana ruang kerja saat istirahat makan siang



Lampiran L.4 Foto penulis beserta tim *developer*, CEO beserta pihak investor



Lampiran L.5 Kueri SQL Skema Database Sistem Antrian InfinId

```
CREATE DATABASE antrianvidcall;

USE antrianvidcall;

DROP TABLE IF EXISTS `bank`;

CREATE TABLE `bank` (
  `uid_bank` varchar(64) NOT NULL DEFAULT UUID(), `nama` varchar(128) NOT NULL,
  PRIMARY KEY (`uid_bank`)) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `agent`;

CREATE TABLE `agent` (
  `uid_agent` varchar(64) NOT NULL DEFAULT UUID(), `nama` varchar(64) NOT NULL,
  `username` varchar(128) NOT NULL, `password` text NOT NULL, `refresh_token` text,
  `id_bank` varchar(64) NOT NULL, PRIMARY KEY (`uid_agent`), UNIQUE KEY
  `username` (`username`), CONSTRAINT `fk_agent_bank` FOREIGN KEY (`id_bank`)
  REFERENCES `bank` (`uid_bank`)) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `antrian`;

CREATE TABLE `antrian` ( `uid_antrian` varchar(64) NOT NULL,
  `waktu_daftar` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  `id_bank` varchar(64) NOT NULL, `room` varchar(64),
  `waktu_selesai` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  `id_agent` varchar(64), `status` ENUM('waiting','on_call','served') DEFAULT 'waiting',
  PRIMARY KEY (`uid_antrian`), CONSTRAINT `fk_antrian_bank` FOREIGN KEY
  (`id_bank`) REFERENCES `bank` (`uid_bank`))ENGINE=InnoDB DEFAULT
  CHARSET=latin1;
```

Lampiran L.6 Source Code Helper JSON Web Token

jwtHelper.js

```
const jwt = require("jsonwebtoken");
const USER_AUTH = require('../config').USER_AUTH_SECRET;
const REFRESH_AUTH = require('../config').REFRESH_AUTH_SECRET
function createAccessToken(payload){
return jwt.sign(payload,USER_AUTH,{
expiresIn: '15 minutes',
})
}
function createRefreshToken(payload){
return jwt.sign(payload, REFRESH_AUTH,{
expiresIn:'30 days',
})
}
function verifyAccessToken(token){
return jwt.verify(token, USER_AUTH)
}
function verifyRefreshToken(token){
return jwt.verify(token, REFRESH_AUTH)
}
module.exports = {createAccessToken, createRefreshToken, verifyAccessToken,
verifyRefreshToken }
```

Lampiran L.7 Source Code Helper Verifikasi Status Login

LoginRequired.js

```
const USER_AUTH = require('../config').USER_AUTH_SECRET;

const jwt = require('jsonwebtoken');

const jwtHelper = require('../helper/jwtHelper');

async function access(req, res, next) {
  var token = req.headers.token;

  try{
    var decodedAuth = jwt.verify(token, USER_AUTH);
    var exp = (decodedAuth.exp - (new Date().getTime()/1000)) / 60;
    req.agent = {"uid":decodedAuth.uid, "exp":exp}
    next();
  } catch(error){
    res.status(401).send({
      success:false,
      error:error,
    })
  }
}

module.exports={access};
```

Lampiran L.8 *Source Code* Bank Model

BankModel.js

```
const database = require('./index').database;
const TABLE_NAME = 'bank';

async function insert(nama){
  var account = {
    nama: nama
  };
  try{
    var result = await database(TABLE_NAME).insert(account);
  } catch(error){
    return Promise.reject(error);
  }
  return Promise.resolve(result);
}
module.exports = { insert};
```

Lampiran L.9 Source Code Agent Model

AgentModel.js

```
const database = require('./index').database;

const bcrypt = require('bcrypt');

const TABLE_NAME = 'agent';

async function insert(username, nama, password, id_bank){
  var hashPassword = await bcrypt.hash(password, 10);
  var account = { username: username, nama: nama, password: hashPassword,
  id_bank:id_bank };
  try{ var result = await database(TABLE_NAME).insert(account); }
  catch(error){ return Promise.reject(error); } return Promise.resolve(result);}

async function getAccount(username){
  try{ var result = await database(TABLE_NAME).where({ username: username }); }
  catch(error){ return Promise.reject(error); } return Promise.resolve(result[0]); }

async function setRefreshToken(uid, refreshToken){
  var update = { refresh_token: refreshToken };
  try{ var result = await database(TABLE_NAME).where('uid_agent', uid).update(update);}
  catch(error){ return Promise.reject(error); } return Promise.resolve(result); }

async function getRefreshToken(uid){
  try{ var result = await database(TABLE_NAME).where('uid_agent',
  uid).select('refresh_token'); }
  catch(error){ return Promise.reject(error); } return Promise.resolve(result[0]); }
module.exports = { insert, getAccount, setRefreshToken, getRefreshToken };
```

Lampiran L.10 Source Code Antrian Model

AntrianModel.js

```
const database = require('./index').database;
const TABLE_NAME = 'antrian';

async function insertQueue(id_antrian, id_bank){
var antrian = { uid_antrian:id_antrian, id_bank: id_bank, waktu_daftar:database.fn.now() };
try{ var result = await database(TABLE_NAME).insert(antrian); }
catch(error){ return Promise.reject(error);} return Promise.resolve(result); }

async function getAgent(id_agent){
try{ var result = await database(TABLE_NAME).where({ id_agent: id_agent,
status:'on_call'}); }
catch(error){ return Promise.reject(error); } return Promise.resolve(result[0]); }

async function callQueue(id_bank, id_agent, room){
return database.transaction(async function (t) {
return await database(TABLE_NAME) .transacting(t)
.where({ id_bank: id_bank, status:'waiting' }).orderBy('waktu_daftar').limit(1)
.then(async function (resp) {
if(resp.length){
var update = await database(TABLE_NAME) .transacting(t)
.where({ uid_antrian: resp[0].uid_antrian, status:'waiting' })
.update({status: 'on_call', id_agent:id_agent, room:room })
if(!update) { return update; } else{ return resp; }}})
.then(t.commit).catch(t.rollback))
.then(function(response) { return Promise.resolve(response); })
.catch(function (error) { return Promise.reject(error); }); }
}
```

Lanjutan Lampiran L.10 *Source Code* Antrian Model

```
async function endQueue(id_agent){
return database.transaction(async function (t) { return await database(TABLE_NAME)
.transacting(t) .where({ id_agent: id_agent, status:'on_call' }) .then(async function (resp) {
if(resp.length){
var update = await database(TABLE_NAME)
.transacting(t)
.where({ uid_antrian: resp[0].uid_antrian, status:'on_call' })
.update({status: 'served', waktu_selesai:database.fn.now()})
if(!update)
{ return update; }
else{ return resp; } } })
.then(t.commit)
.catch(t.rollback))
.then(function(response) { return Promise.resolve(response); })
.catch(function (error) { return Promise.reject(error); });})

async function count(id_bank){
try{ var result = database(TABLE_NAME).where({ id_bank: id_bank,
status:'waiting' }).count('* as count'); }
catch(error) { return Promise.reject(error); }
return Promise.resolve(result); }

module.exports = { insertQueue, callQueue, endQueue, getAgent, count };
```

Lampiran L.11 Source Code Bank Controller

BankController.js

```
const Bank = require('express').Router();
const Joi = require('joi');
const BankAccount = require('../models/BankModel');
Bank.post('/register', function(req, res, next){
  const Schema = Joi.object().keys({
    nama: Joi.string().required()
  });
  Joi.validate(req.body, Schema).then(result => {
    next();
  }).catch(error => {
    res.status(400).send({
      success: false,
      error: error.message
    });
  });
}, async function(req, res, next){
  try{
    result = await BankAccount.insert(req.body.nama);
    res.send({
      success: true,
    });
  } catch(error){
    console.log(error);
    return res.status(400).send({ success: false, error: error.message });
  }
});
module.exports = Bank;
```


Lampiran L.12 Source Code Agent Controller

AgentController.js

```
const Agent = require('express').Router();
const Joi = require('joi');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const AgentAccount = require("../models/AgentModel")
const USER_AUTH = require('../config').USER_AUTH_SECRET;
const REFRESH_AUTH = require('../config').REFRESH_AUTH_SECRET;
const jwtHelper = require('../helper/jwtHelper');

Agent.post('/register', function (req, res, next) {
  const Schema = Joi.object().keys({ username: Joi.string().trim().required(),
  nama: Joi.string().required(), password: Joi.string().required(),
  id_bank: Joi.string().trim().required(), });
  Joi.validate(req.body, Schema).then(result => { next(); })
  .catch(error => { res.send({ success: false, error: error.message }); }); },
  async function (req, res, next) {
    try { var account = await AgentAccount.getAccount(req.body.username);
    if (account != null) { throw new Error('username_used'); }
    result = await AgentAccount.insert(req.body.username, req.body.nama, req.body.password,
    req.body.id_bank);
    res.send({ success: true, }); }
    catch (error) { console.log(error); return res.send({ success: false, error: error.message }); }
  });

  Agent.post('/login', function (req, res, next) {
    const Schema = Joi.object().keys({
    username: Joi.string().required(),
```

Lanjutan Lampiran L.12. *Source Code Agent Controller*

```
password: Joi.string().required());
Joi.validate(req.body, Schema).then(result => {
  next();})
.catch(error => { res.send({ success: false, error: error.message }); }); },
async function (req, res, next) {
  try { var account = await AgentAccount.getAccount(req.body.username);
  } catch (error) { next(error); }
  if (!account) { return res.send({ success: false, error: 'username_not_valid' }) }
  if (!bcrypt.compareSync(req.body.password, account.password)) {
    return res.send({ success: false, error: 'password_not_match' }); }
  var payload = { uid: account.uid_agent };
  var refreshToken = jwt.sign(payload, REFRESH_AUTH, { expiresIn: '30 days' })
  var accessToken = jwt.sign(payload, USER_AUTH, { expiresIn: '15 minutes' });
  res.cookie('refresh_token', refreshToken, { maxAge: 1000 * 60 * 60 * 24 * 30,
    httpOnly: true })
  res.send({ success: true, accessToken: accessToken, refreshToken: refreshToken }); });

Agent.post('/refresh-token', function(req, res, next) {
  let refresh_token = req.cookies.refresh_token; console.log(refresh_token)
  try { let decoded = jwtHelper.verifyRefreshToken(refresh_token);
  let payload = { uid: decoded.uid, }
  let new_access_token = jwtHelper.createAccessToken(payload);
  res.send({ success: true, accessToken: new_access_token })
  } catch (error) { res.status(401).send({ success: false, error: error }) } })
module.exports = Agent;
```

Lampiran L.13 Source Code Antrian Controller

AntrianController.js

```
const Antrian = require('express').Router();

const JOI = require('joi');

const loginRequired = require('../helper/LoginRequired');
const ListAntrian = require('../models/AntrianModel');

function init(io){
  Antrian.post('/register', function(req, res, next){
    const Schema = JOI.object().keys({
      id_bank: JOI.string().trim().required(), id_antrian: JOI.string().trim().required() });
    JOI.validate(req.body, Schema).then(result => { next(); })
      .catch(error => { res.status(400).send({ success: false, error: error.message }); });
  }, async function(req, res){
    try{ await ListAntrian.insertQueue(req.body.id_antrian, req.body.id_bank);
      io.to('room.'+req.body.id_bank).emit('queue.change'); res.send({ success: true, });
    }catch(error){ console.log(error);
    return res.status(400).send({ success: false, error: error.message }); } });

  Antrian.use(loginRequired.access);
  Antrian.post('/nextAntrian', function(req, res, next){
    const Schema = JOI.object().keys({ id_bank: JOI.string().required().trim(),
      room: JOI.string().required(), room_password: JOI.string().required() });
    JOI.validate(req.body, Schema).then(result => { next(); })
      .catch(error => { res.status(400).send({ success: false, error: error.message }); });
  }, async function(req, res, next){
    try{ var result = await ListAntrian.getAgent(req.agent.uid);
      if(result){ return res.status(400).send({ success: false, error: 'sedang dalam panggilan'}) }
      next(); } catch(error){ next(error);
    return res.status(400).send({ success: false, error: error.message}) } },
```

Lanjutan Lampiran L.13. *Source Code Antrian Controller*

```
async function(req, res, next){
  try{var result= await ListAntrian.callQueue(req.body.id_bank, req.agent.uid, req.body.room);
  if(!result){ return res.status(400)
  .send({ success: false, error: 'antrian habis/gagal memanggil'}) }
  console.log("AGENT IS CALLING ", req.agent.uid);
  io.to('room.'+req.body.id_bank).emit('queue.change');
  io.to(result[0].uid_antrian).emit('call.start',{
  room:{ name:req.body.room, password: req.body.room_password } });
  res.send({success: true, socket_id:result[0].uid_antrian });
  }catch(error){ return res.status(400).send({ success: false, error: error.message}) } })

  Antrian.post('/endAntrian', async function(req, res){
  try{ var result= await ListAntrian.endQueue(req.agent.uid);
  if(!result){ Return res.status(400)
  .send({ success: false, error: 'tidak sedang memanggil/gagal menutup'}) }
  io.to(result[0].uid_antrian).emit('call.end', req.agent.uid);
  res.send({ success: true }); }catch(error){
  return res.status(400).send({ success: false, error: error.message}) } })

  Antrian.get('/count', function(req, res, next){
  const Schema = Joi.object().keys({ id_bank: Joi.string().required().trim(), });
  Joi.validate(req.query, Schema).then(result => {next();
  }).catch(error => { res.status(400).send({ success: false, error: error.message }); });
  }, async function(req,res,next){ try{ var count = await ListAntrian.count(req.query.id_bank)
  res.send({ success:true, count:count[0].count }) }
  catch(error){ res.status(400).send({ success: false, error: error.message }); } })
  return Antrian }
  module.exports = init;
```