PERANCANGAN APLIKASI *SMART FENCE* MENGGUNAKAN *VISUAL STUDIO CODE* BERBASIS IOT

PROYEK AKHIR

Laporan akhir ini dibuat dan diajukan untuk memenuhi salah satu syarat kelulusan Sarjana Terapan/Diploma IV Politeknik Manufaktur Negeri Bangka Belitung

Diusulkan oleh:

Hasya Akhlaqah Harmie NIM : 1062011

Rahmattian Diza Dwi Putra NIM : 1062023

POLITEKNIK MANUFAKTUR NEGERI

BANGKA BELITUNG

TAHUN 2024

LEMBAR PENGESAHAN

PERANCANGAN APLIKASI *SMART FENCE* MENGGUNAKAN *VISUAL STUDIO CODE* BERBASIS IOT

Oleh:

Hasya Akhlaqah Harmie / 1062011

Rahmattian Diza Dwi Putra / 1062023

Laporan akhir ini telah disetujui dan disahkan sebagai salah satu syarat kelulusan Program Sarjana Terapan Politeknik Manufaktur Negeri Bangka Belitung

Menyetujui,

Pembimbing 1	Pembimbing 2
Ahmat Josi, S.Kom.,M.Kom.	Riki Afriansyah, S.T., M.T.
NIP. 198908202019031015	NIP.199004042019031013

Penguji 1	Penguji 2
Irwan, M.Sc., Ph.D.	Priestiani, S.P., M.P.
NIP. 197604182014041001	NIP. 199003032022032005

PERNYATAAN BUKAN PLAGIAT

Yang menandatangani di bawah ini:

Nama Mahasiswa	:	Hasya Akhlaqah Harmie	NIM	:	1062011
Nama Mahasiswa	:	Rahmattian Diza Dwi Putra	NIM	:	1062023

Dengan Judul : Perancangan Aplikasi *Smart Fence* Menggunakan *Visual Studio Code* Berbasis IoT

Kami menyatakan bahwa laporan akhir ini adalah hasil karya kami sendiri dan bukan merupakan tindakan plagiat. Pernyataan ini kami buat dengan sejujurnya, dan jika pada suatu waktu terbukti melanggar pernyataan ini, kami siap menerima sanksi yang diberlakukan.

Nama Mahasiswa — Sungailiat, 01 Januari 2024

Tanda tangan

1. Hasya Akhlaqah Harmie

……

2. Rahmattian Diza Dwi Putra

……

ABSTRAK

Perkembangan *Internet of Things* memudahkan integrasi antara perangkat dengan lingkungan dengan menfasilitasi jaringan kepada sistem Penelitian ini memusatkan pada sistem *monitoring* pagar *pin*tar berbasis IoT yang nantinya akan mampu melakukan *monitoring* serta pengendalian jarak jauh melalui aplikasi Android. Sistem juga dapat melakukan pengendalian menggunakan suara untuk, meningkatkan keterjangkauan dan kenyamanan pengguna pada situasi tertentu. Metodologi penelian berpusat pada metode kuantitatif dimana peneliti akan mengumpulkan data hasil dari sistem setelah dilakukan pengujian. Pengujian yang dimaksud memberikan hasil yang bervariasi. Hasil yang diperoleh dalam pengujian berupa; pada kecepatan sinyal sebesar rata-rata 46,74mb, rata-rata suara sebesar 75%, sistem operasi (tombol/suara) dengan rata-rata 9,71s dan keberhasilan 66,67% serta, pengujian aplikasi dengan rata-rata 2,13 s dan keberhasilan sebesar 71,43%. Berdasarkan hasil uji coba sistem, diketahui kinerja sistem dapat dikategorikan cukup baik. Sistem mendapatkan kendala terutama dalam implementasi sistem ke dalam Bahasa Inggris, dengan nilai yang menunjukkan gagalnya dalam pengujian. Penelitian ini diharapkan dapat meningkatkan efesiensi dalam pengendalian pagar berbasis IoT dan memiliki potensi pengembangan lebih lanjut untuk mendukung kemajuan teknologi IoT.

*Kata Kunci : Internet of Things (*IoT*); Keamanan Pagar; Pagar Pintar; Pengaturan Suara; Real-time.*

ABSTRACT

The development of the Internet of Things facilitates the integration of devices with the environment by providing networking capabilities to systems. This research focuses on an IoT*-based Smart Fence monitoring system that will be capable of remote monitoring and control through an Android application. The system also enables voice control, enhancing accessibility and user convenience in specific situations. The research methodology centers around quantitative methods, where the researcher collects data from the system after conducting tests. The tests yield varied results, including an average signal speed of 46.74 Mbps, an average voice recognition rate of 75%, an average system operation time (via buttons/voice) of 9.71 seconds with a success rate of 66.67%, and application Testing with an average time of 2.13 seconds and a success rate of 71.43%. Based on the system's trial results, it is observed that the system's performance can be categorized as quite good. The system encounters challenges, especially in implementing the system in English, with values indicating failures in Testing. This research is expected to improve the efficiency of* IoT*-based fence control and has the potential for further development to support the advancement of* IoT *technology.*

*Keywords: Internet of Things (*IoT*); Fence Security; Voice Control; Smart Fence; Real-time.*

KATA PENGANTAR

Assalamu'alaikum warahmatullahi wabarakatuh,

Puji syukur kami panjatkan kepada Allah SWT Yang Maha Pengasih lagi Maha Penyayang atas rahmat serta karuniaNya, Nabi Muhammad SAW sebagai pedoman pemberi petunjuk sehingga kami dapat menyelesaikan penelitian ini dengan judul "Perancangan Aplikasi *Smart Fence* Menggunakan *Visual Studio Code* Berbasis IoT". Penelitian ini dibuat dalam rangka sebagai salah satu persyaratan dalam kelulusan pada program studi D4 Teknologi Rekayasa Perangkat Lunak pada Politeknik Manufaktur Negeri Bangka Belitung. Penulis menyadari bahwa dalam proses penyusunan penelitian serta pembuatan sistem yang penulis lakukan tidak akan lepas dari pengetahuan, doa, dukungan, serta bantuan dari berbagai pihak. Oleh karena itu, penulis ingin menyampaikan rasa terima kasih kepada:

1. Allah SWT atas segala keberkahan serta nikmat yang telah diberikan.
2. Orang Tua, Saudara serta Keluarga penulis yang telah memberikan motivasi serta perhatian kepada penulis.
3. Bapak I Made Andik Setiawan, M.Eng, Ph.D selaku Direktur Politeknik Manufaktur Negeri Bangka Belitung.
4. Bapak Irwan Ramli, M.Sc, Ph.D selaku Wakil Direktur I Politeknik Manufaktur Negeri Bangka Belitung.
5. Bapak Muhammad Subhan, M.T selaku Wakil Direktur II Politeknik Manufaktur Negeri Bangka Belitung.
6. Bapak Eko Sulistyo, M.T selaku Wakil Direktur III Politeknik Manufaktur Negeri Bangka Belitung.
7. Bapak Zanu Saputra S.ST., M.Tr.T selaku Ka. Jurusan Teknik Elektro & Informatika Politeknik Manufaktur Negeri Bangka Belitung.
8. Bapak Ahmat Josi, M.Kom selaku Ka. Prodi D4 Teknologi Rekayasa Perangkat Lunak dan Dosen Pembimbing I serta Bapak Riki Afriansyah, M.T.

sebagai pembimbing II, atas dukungan, kritik dan saran dalam membangun sistem.

9. Teman-teman penulis yang mungkin tidak dapat penulis sebutkan satu-satu, terutama teman seperjuangan pada D4 Teknologi Rekayasa Perangkat Lunak baik dalam suka dan duka, selalu mendukung serta memberikan arahan kepada penulis.

Demikian laporan yang penulis buat, penulis mengucapkan terima kasih dan berharap semoga penelitian yang penulis buat dapat menambah wawasan untuk kita semua.

Wassalamu'alaikum Warahmatullahi Wabarakatuh

Sungailiat, 01 Januari 2024

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB I

PENDAHULUAN

1.1 Latar Belakang

Seiring berkembangnya zaman, manusia akan selalu mengalami peningkatan dalam aspek kehidupan termasuk dalam bidang teknologi yang dibuktikan dengan adanya berbagai macam teknologi yang saat ini tersedia. Teknologi tersebut digunakan untuk memenuhi unsur kebutuhan dalam hidup dan salah satunya adalah teknologi *Internet of Things*. *Internet of Things* atau yang biasanya disingkat IoT merupakan suatu konsep dengan maksud memanfaatkan konektivitas *internet* [1].

Sistem penerapan IoT sudah merajalela dan mempengaruhi berbagai aspek kehidupan masyarakat, bahkan tanpa kita sadari sekalipun. Salah satu penggunaan dari IoT adalah digunakan untuk mengendalikan alat dari jarak tertentu sekalipun kita tidak berada didekatnya. Hal ini dapat dimanfaatkan dalam menciptakan suatu sistem keamanan seperti keamanan dengan menggunakan pagar. Dengan adanya sistem pengamanan, kita dapat melakukan akses serta *monitoring* dari mana saja tanpa hambatan dnegan syarat memiliki dan menggunakan data *internet* sebagai sarana koneksi sistem dengan jaringan[2].

Sistem *monitoring* ini bermanfaat bagi masyarakat yang ingin meningkatkan keamanan ke tahapan lebih, dimana sebagian besar pagar masih menggunakan pagar manual. Penggunaan pagar tersebut dapat menjadi tidak efesien, terutama saat pemilik suatu tempat harus membuka serta menutup pagar dengan manual ketika sedang mengendarai kendaraannya. Dengan sistem pagar *pin*tar ini, pengguna dapat membuka dan *memonitoring* pagar melalui telepon genggam mereka dengan tombol *interface* aplikasi serta dengan fitur suara yang akan mempermudah penggunaan sistem guna mempersingkat waktu dengan sistem yang sudah automatis, meingkatkan keamanan serta *memonitoring* siapa saja yang

mengakses pagar. Perkembangan ini dapat memberikan pengaruh yang signifikan bagi kehidupan[3]. Sistem akan bekerja dengan cara memanfaatkan konektifitas IoT yang memungkinkan komunikasi dengan perangkat lain, melakukan transmisi data dan mendeteksi ketika ada seseorang yang membuka atau menutup pagar, serta sistem dapat mengikuti intruksi perintah yang diberikan untuk melakukan aksi yang sesuai.

1.2 Perumusan Masalah

Berdasarkan uraian diatas, dapat disimpulkan bahwa permasalahan yang dibahas dalam Proyek Akhir ini yaitu:

1. Bagaimana merancang aplikasi *Smart Fence* berbasis IoT yang nantinya akan diintegrasikan dengan sistem kontrol berbasis *Internet of Things*
2. Bagaimana memberikan sistem antarmuka pengguna yang intuitif serta mudah digunakan untuk mengontrol dan memantau pagar *pin*tar berbasis IoT
3. Bagaimana merealisasikan sistem *Smart Fence* dengan perangkat *Android* sehingga pengguna dapat memantau dan mengontrol pagar dari jarak jauh secara *real-time*
4. Bagaimana cara mengimplementasikan dan mengintegrasikan sistem dengan *Voice Control*, penggunaan tombol?

1.3 Batasan Masalah

Pada penelitian kali ini agar pembahasan tidak terlalu jauh dan keluar dari topik masalah utama penulis membuat beberapa batasan masalah antara lain :

1. Implementasi bahasa

   Pada penelitian kali ini penulis membatasi implementasi pada bahasa yang dapat dimengerti oleh *voice control*, *voice control* hanya dapat mendengarkan kalimat yang diucapkan oleh pengguna dalam Bahasa Indonesia, akan tetapi pengguna harus mengatur bahasa *default* pada *smartphone* mereka kedalam Bahasa Indonesia.

2. Pendeteksi suara

   Pada penelitian kali ini penulis membatasi pengenalan suara hanya pada ketika tombol *voice control* ditekan maka sistem akan mendeteksi apakah ada pengguna yang berbicara atau tidak.

3. *Monitoring* sistem

   Pada penelitian kali ini penulis membatasi parameter yang dapat dipantau oleh pengguna secara *real time* hanyalah status dari pagar itu, apakah pagar itu dalam kondisi tertutup atau terbuka.

1.4 Tujuan Proyek Akhir

Berdasarkan rumusan masalah yang telah dijelaskan tujuan penelitian ini adalah:

1. Merancang dan membangun aplikasi yang terintegrasi dengan *voice control* atau suara, serta tombol *visual* pada aplikasi untuk pagar *pin*tar berbasis IoT
2. Mengembangkan aplikasi *monitoring* untuk pagar berbasis IoT pada *Android* secara *real-time*

BAB II

DASAR TEORI

2.1 Penerapan *Internet of Things*

Internet of Things (IoT) adalah jaringan yang terdiri dari objek fisik yang terhubung ke *internet* dan berkomunikasi antara satu sama lain melalui sensor *pin*tar, perangkat elektronik, dan teknologi komunikasi. Tujuan utama IoT adalah untuk menghubungkan objek fisik ke dalam ekosistem digital, yang memungkinkan pertukaran data yang lebih efisien dan memungkinkan Objektif sehari-hari seperti infrastruktur kota, kendaraan, perangkat medis, dan peralatan rumah tangga dapat terhubung dan berinteraksi secara otonom dengan manusia dan objek lain. Penerapan *Internet of Things* yang dilakukan penulis berfokus kepada penelitian penelitian sebelumnya yang relevan dengan judul penelitian penulis. Berikut hasil dari penelitian-penelitian yang telah dilakukan sebelumnya:

Dalam penelitian yang berjudul *Smart Automatic Sliding Gate* Dengan Memanfaatkan Teknologi Berbasis *Internet of Things*(IoT)[4] membahas tentang Penerapan *Internet of Things* (IoT) dalam mengotomatiskan penggunaan *sliding gate* bertujuan untuk meningkatkan kemudahan dan efisiensi, khususnya ketika tidak ada pengawasan langsung oleh petugas. Penggunaan Aplikasi Ewelink yang terkoneksi dengan 2CH *Smart Switch Wifi* melalui *internet* memungkinkan pengendalian jarak jauh yang lebih efisien dibandingkan dengan penggunaan Remote RF konvensional. Selain itu, untuk memperbaiki fitur pagar, penelitian ini melibatkan penggunaan *magnetic limited switch* dan sensor anti jepit. Hasil uji coba menunjukkan bahwa integrasi IoT dalam kendali jarak jauh dapat menjadi solusi yang efektif dan praktis untuk mengotomatiskan operasi *sliding gate*.

Dalam penelitian yang berjudul Sistem Kontrol Otomatis Pagar Rumah Berbasis *Internet of Things*(IoT)[5] membahas tentang Pengembangan Sistem Otomatis

Pembuka dan Penutup Pagar menggunakan Teknologi *Internet of Things* (IoT) dengan Pengontrolan *Smartphone*. Jurnal ini membahas implementasi IoT untuk mengotomatiskan pembukaan dan penutupan pagar secara efektif dan efisien. Sistem menggunakan *NodeMCU* yang dikontrol melalui koneksi *internet* dengan aplikasi *Blynk* pada *smartphone* sebagai pengontrol. *Relay* digunakan sebagai *switch* untuk menggerakkan motor DC 12V 200rpm yang menggerakkan mekanisme pembuka dan penutup pagar. Motor servo digunakan untuk mengunci pagar setelah penutupan. Sistem juga mencakup informasi mengenai kecepatan motor, waktu yang dibutuhkan untuk membuka dan menutup pagar, dan spesifikasi tegangan yang diperlukan.

Dalam penelitan yang berjudul Sistem Pengontrol Pintu Pagar Dengan *Voice Control* berbasis Mikrokontroler[6] membahas tentang Pengembangan Prototipe Sistem Pengontrol *Pin*tu Pagar dengan *Voice Control* menggunakan *Google Assistant* dan Sistem Keamanan Terintegrasi. Jurnal ini membahas tentang pembuatan prototipe sistem pengontrol *pin*tu pagar yang dapat dioperasikan menggunakan *voice control* melalui *Google Assistant* pada *smartphone* berbasis *Android*. Prototipe ini melibatkan *NodeMCU* yang berkomunikasi dengan Arduino Nano untuk mengendalikan motor *pin*tu pagar. Selain itu, sistem keamanan juga diimplementasikan, di mana prototipe mengirimkan *trigger* alarm ke *NodeMCU* yang selanjutnya diteruskan ke akun *Gmail*. Tujuan dari prototipe ini adalah untuk memberikan solusi yang mudah digunakan bagi pengguna dalam membuka dan menutup *pin*tu pagar secara otomatis, sekaligus meningkatkan keamanan di rumah atau tempat tinggal.

Dalam penelitian yang berjudul Sistem *Monitoring* dan Kontrol Rumah Berbasis *Internet of Things* (IoT)[7] membahas tentang Pembangunan Sistem *Monitoring* Rumah Berbasis *Internet of Things* (IoT) dengan Penggunaan Sinyal RFID dan Pemantauan Gambar dan Video. Penelitian ini bertujuan untuk mengatasi kekhawatiran pemilik rumah terkait keamanan dan pemantauan barang berharga saat mereka berada di luar rumah. Sistem yang dikembangkan menggunakan konsep *smart home* berbasis IoT, yang melibatkan perangkat *monitoring*

Mikrokontroler dengan kemampuan membaca sinyal RFID serta mengambil foto dan video. Hasil dari sistem ini dapat diakses melalui web aplikasi yang terhubung dengan *REST API*. Evaluasi kualitas layanan menunjukkan performa yang baik dengan *delay*, *throughput*, *jitter*, dan *packet loss* yang sangat baik. Penelitian ini mencakup konsep-konsep kunci seperti *Internet of Things*, *Smart Home*, Mikrokontroler, dan *RESTful API*.

Dalam penelitian yang berjudul Perancangan Sistem *Pin*tu Gerbang Otomatis Berbasis *NodeMCU* Menggunakan *Sms Gateway*[8] membahas tentang Perancangan Sistem *Pin*tu Gerbang Otomatis Berbasis *NodeMCU* dengan Kontrol melalui SMS Gateway. Jurnal ini membahas permasalahan terkait ketidakberadaan alat pengendali *pin*tu gerbang otomatis, kurangnya sistem pengontrol melalui *smartphone* menggunakan SMS *gateway*, serta kurangnya efektivitas dan efisiensi waktu pengguna dalam membuka dan menutup *pin*tu gerbang. Metode pemecahan masalah melibatkan pengumpulan data melalui observasi, studi kepustakaan, studi dokumentasi, pembangunan prototipe, serta pengujian perancangan sistem dan alat. Hasil penelitian mencakup rancangan alat pengendali dan sistem pengontrol *pin*tu gerbang otomatis berbasis *NodeMCU* menggunakan SMS *gateway*, terutama untuk *pin*tu gerbang geser dengan motor servo sebagai penggerak. *Smartphone* digunakan melalui SMS gateway untuk mengontrol sistem, dan notifikasi disampaikan melalui Web kepada pengguna. Fungsi alat ini terdiri dari membuka dan menutup *pin*tu gerbang setelah menerima perintah SMS dari *smartphone*. Sistem memberikan balasan SMS kepada pengguna dan memberikan notifikasi Web. Penerapan *pin*tu gerbang otomatis diharapkan dapat mempermudah pengguna dan mengatasi masalah yang diidentifikasi.

Dalam penelitian yang berjudul Garasi Rumah Berbasis *Internet of Things* (IoT)[9] membahas tentang Pengembangan *Smart Garage* dengan Menggunakan *NodeMCU* untuk Meningkatkan Kemudahan Akses dan Penggunaan *Pin*tu Garasi. Jurnal ini membahas penemuan alat inovatif yang disebut *Smart Garage* yang dirancang untuk mempermudah penggunaan *pin*tu garasi sehari-hari. Prinsip kerja alat ini melibatkan konektivitas dengan *NodeMCU*, yang merupakan unit pengendali

mikro. Rangkaian motor *pin*tu menggunakan Servo yang terhubung dengan *NodeMCU* dan dikontrol melalui perintah yang dikirimkan melalui *smartphone* pengguna. Penelitian ini bertujuan untuk memberikan solusi bagi pengguna agar tidak perlu turun dari mobil saat ingin masuk atau keluar dari garasi. Sistem ini dirancang untuk merespon perintah yang diberikan melalui *smartphone*, memungkinkan pengguna untuk membuka dan menutup *pin*tu garasi tanpa harus secara manual mendorong atau menarik *pin*tu.

Dalam penelitian yang berjudul Inovasi Kendali *Pin*tu Berbasis *Internet of Thing* (IoT) Menggunakan Aplikasi *Raspbrry Pi-3* Dengan Proteksi Fitur *Image* (*Ionic*)[10] membahas tentang Pengembangan Sistem Keamanan Rumah dengan Pemantauan IP Camera Berbasis Web dan Kontrol *Pin*tu Pagar Jarak Jauh Menggunakan Raspberry Pi-3 dan *Smartphone* Android. Jurnal ini membahas cara meningkatkan keamanan rumah dengan memanfaatkan teknologi IP Camera berbasis web untuk memonitor situasi rumah, terutama saat penghuninya tidak berada di tempat. Metode ini kemudian dikembangkan dengan pembuatan aplikasi pada *Smartphone* untuk mengendalikan pembukaan/tutupan *pin*tu pagar dari jarak jauh. Penelitian ini mencakup pembuatan *prototype* pengendali *pin*tu pagar jarak jauh menggunakan Raspberry Pi-3 sebagai *controller* dan server, *Smartphone* berbasis *Android* dengan aplikasi Ionic untuk mengakses kamera dari jarak jauh, modem, jaringan *internet*, dan saklar bell *pin*tu sebagai sensor notifikasi. Hasil penelitian menunjukkan bahwa respons pergerakan *pin*tu pagar dapat bervariasi tergantung pada kecepatan *internet*, dengan waktu load video yang cukup cepat dalam jaringan 4G dan 3G.

Dari hasil kajian penilitan-penelitian yang telah dilakukan sebelumnya, dapat kami simpulkan bahwa terdapat beberapa akses dalam penggunaan pagar ada yang menggunakan perintah suara dan ada juga yang menggunakan perintah *remote* bahkan ada yang melakukan perintah melalui *sms*. Adapun perbedaan dari penelitian sebelumnya dengan penelitian yang akan kami lakukan ini, terdapat pembaruan dalam penelitian sistem. Pembaharuan sistem yang dimaksud akan mengalami beberapa penambahan fitur seperti *monitoring* pagar secara *real-time*.

Adapun parameter yang akan di*monitoring* adalah status pagar tersebut, terdapat 2 metode yang bisa digunakan dalam mengoperasikan pagar yaitu menggunakan tombol *visual* yang terdapat dalam aplikasi dan menggunakan perintah suara. Penambahan kedua fitur ini dilakukan agar sistem tidak hanya memberikan akses dengan menekan tombol, namun juga dapat mengoperasikan pagar dengan perintah suara secara real time. Sistem akan mendapat pembaruan dari segi tampilan *User Interface* (*UI*) terkini agar terlihat lebih menarik serta mudah gunakan oleh pengguna.

2.2 *Smart Fence*

Smart Fence, atau "Pagar *Pin*tar" merupakan sistem pagar *pin*tar yang memanfaatkan teknologi untuk meningkatkan kefektifan *user* sistem. Sebagian besar pengguna pagar *pin*tar menggunakan teknologi *Internet of Things* (IoT*)* dengan alasan penggunaan sistem tersebut memusahkan *user* dalam mengakses pagar, meningkatkan keamanan pada pagar, meningkatkan efesiensi dalam pengaplikasian pagar serta dapat memberikan informasi *real-time* kepada tempat tersebut tentang status pagar sesuai dengan kondisi terakhir dalam *monitoring*.
Smart Fence sendiri dapat menjadi solusi untuk mengurangi permasalahan dari isu-isu tindak kriminal seperti pencurian. Selain meningkatkan keamanan pada pagar, penerapan *Smart Fence* juga dapat meningkatkan kenyamanan dan juga kefisienan dalam penggunaan pagar, dimana pagar yang awalnya dioperasikan secara manual sekarang bisa di operasikan hanya menggunakan *smartphone* yang terkoneksi dengan *internet* dan menggunakan perintah suara.

2.3 *Visual Studio Code*

Visual Studio Code atau *VS Code* merupakan suatu *edit*or jode yang dikembangkan oleh Microsoft yang berfokuskan pada kecepatan, keringanan dan keterbukaan dalam memberikan pengalaman pengembangan yang tangkas dan responsif. *Edit*ior ini mendukung berbagai macam bahasa seperti Java, JavaScript, Python, C# dan lain-lain. Sistem *edit*or ini juga mendukung sistem operasi multiplatform, artinya tersedia juga untuk versi Linux, Mac, dan Windows. [13]

2.4 *Figma*

Figma merupakan salah satu platform yang digunakan untuk mendesaim atau membuat tambpilan website, mobile, desktop dan lain lain. *Figma* dapat digunakan dalam berbagai sistem operasi selama terhubung dengan jaringan *internet* dengan umumnya digunakan untuk meningkatkan UX/UI karena berfokuskan didalam penampilan dan pembuatan desain.[14]

2.5 *Kotlin*

Kotlin adalah bahasa pemrograman untuk android yang memiliki kaitan dengan sistem Object Oriented dan fungsional. Bahasa *kotlin* juga berbasiskan pada Java Virtual Machine yang sebelumnya dikembangkan oleh Jetbrains. *Kotlin* digunakan untuk mengembangkan aplikais berbasis web, desktop ataupu backend dan penggunaan bahasa ini dapat meringkas bahasa kode dengan Java, sehingga *kotlin* sangat mudah dipelajari. [15]

2.6 Mikrokontroler

Mikrokontroler adalah perangkat elektronik yang berfungsi sebagai otak atau pusat kontrol sistem elektronik. Ini terdiri dari memori, *CPU*, serta perangkat *input*/output yang terintegrasi dalam satu chip. Mereka memiliki kemampuan untuk mengontrol dan mengendalikan berbagai perangkat elektronik. Mikrokontroler memiliki komunitas pengembang yang luas dan antarmuka yang mudah digunakan. Sebaliknya, komponen tambahan yang digunakan, *ESP32 NodeMCU*, berfungsi sebagai pengirim dan penerima sinyal WiFi, yang memungkinkan *smartphone* dan perangkat *smart home* terhubung satu sama lain dari jarak jauh. Sistem ini akan tersambung ke jaringan *internet* dan menggunakan *smartphone* untuk melakukan *monitoring* dari jarak jauh[16]. Mikrokontroler yang digunakan dalam proyek ini memiliki kemampuan untuk membaca sensor dan melaksanakan perintah aplikasi secara berurutan. Ini adalah sistem pagar *pin*tar yang dapat terhubung ke jaringan dan menggunakan *smartphone* untuk melakukan pengaturan dan pengawasan dari jarak jauh[17].

2.7 Sistem *Monitoring*

Sistem yang dimaksudkan untuk mengumpulkan informasi yang relevan dengan melacak dan mengamati keadaan dan aktivitas suatu sistem atau lingkungan dikenal sebagai sistem *monitoring*. Sistem *monitoring* digunakan dalam proyek rancang bangun aplikasi *Smart Fence* menggunakan *Visual Studio Code* berbasis *Internet of Things* (IoT) untuk memantau dan mengawasi berbagai peralatan listrik di suatu tempat, seperti lampu, kipas, *pin*tu garasi, dan parameter suhu, kelembaban, deteksi api, dan gas, akan tetapi pada proyek kali ini kami membatasi fitur *monitoring* nya jadi nantinya sistem hanya mampu *memonitoring* status dari pagar tersebut apakah pagar tersebut sedang tertutup ataupun sedang terbuka. Aplikasi berbasis *Android* yang terintegrasi sistem nanti memungkinkan pengguna mengakses secara real-time informasi tentang pagar. Informasi ini ditampilkan pada aplikasi yang nantinya akan kami kembangkan yang terhubung dengan Mikrokontroler. Selain itu, sistem ini mengirimkan notifikasi kepada pengguna melalui aplikasi tersebut, sehingga pengguna dapat mengambil tindakan yang diperlukan jika terjadi situasi yang memerlukan perhatian.[18]

2.8 *Firebase*

Platform yang digunakan dalam pengembangan penelitian ini berupa *Firebase* yang dimana menyediakan layanan sistem untuk mengelolah, membangun serta mengoptimalkan aplikasi. *Firebase* memberikan pengguna *database* cloud yang terkoneksi secara *real-time* sehingga memungkinkan pengguna mendapatkan hasil data secara instan.

2.9 *Voice Control*

Voice control atau *Voice Regconition* mengacu pada penggunaan suara manusia sebagai *input* untuk mengontrol sistem atau perangkat menggunakan perintah suara. Dalam penelitian ini mengacu pada penggunaan suara manusia sebagai *input* untuk mengendalikan sebuah pagar secara keseluruhan. Ini memungkinkan pengguna menggunakan suara mereka untuk memberikan arahan, yang kemudian diterjemahkan dan dilaksanakan oleh sistem yang terhubung. Kontrol suara memanfaatkan kemajuan dalam teknologi IoT untuk membuat interaksi dengan

sistem *Smart Fence* lebih mudah dan mudah dipahami bagi pengguna[19]. Dengan sistem *Smart Fence* yang menggunakan kontrol suara, pengguna dapat menggunakan suara mereka untuk mengontrol atau mengendalikan pagar, yang mana dapat mempermudah *user* dalam mengoperasikan pagar. Dalam penelitian ini, fungsinya dibatasi hanya untuk pengendalian pagar. Ini dilakukan untuk membuatnya lebih aman dan untuk meningkatkan pengalaman pengguna, terutama bagi mereka yang tinggal sendiri. Pagar dapat dikendalikan dengan lebih mudah dan responsif, meningkatkan kenyamanan dan keamanan, dan memberikan pengalaman yang lebih baik bagi pengguna aplikasi[20].

2.10 Aplikasi Berbasis *Android*

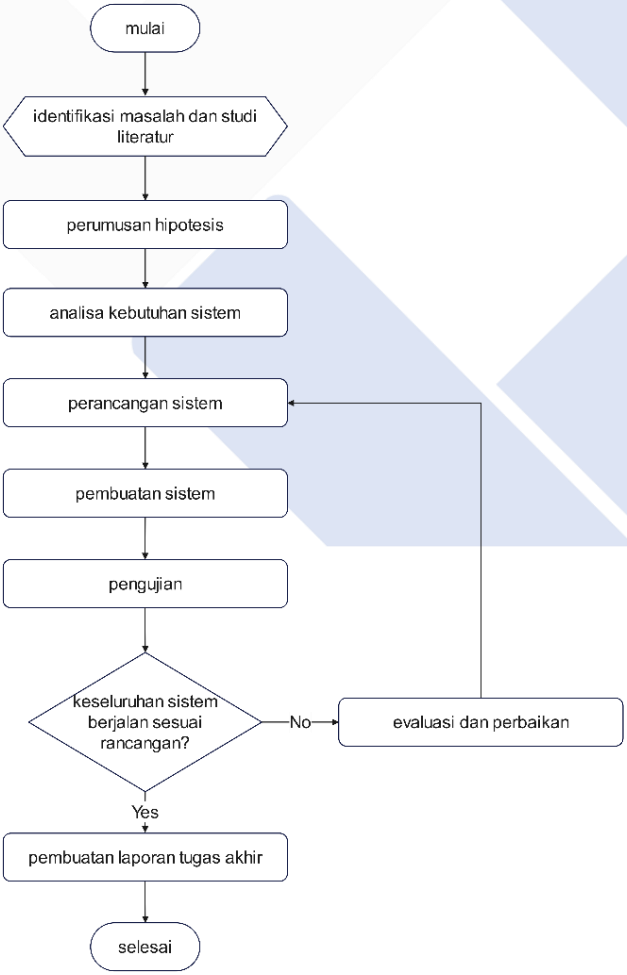
Aplikasi yang dibuat untuk digunakan pada sistem operasi *Android* disebut aplikasi berbasis *Android*. Aplikasi ini memanfaatkan fitur dan kemampuan yang disediakan oleh platform *Android*, seperti antarmuka pengguna yang mudah digunakan, dukungan untuk sensor dan perangkat keras, serta koneksi ke jaringan. Aplikasi berbasis *Android* dapat diinstal dan digunakan pada berbagai perangkat yang menjalankan sistem operasi *Android*, seperti *smartphone*, tablet, dan lainnya. Dengan aplikasi berbasis *Android*, pengguna dapat melakukan berbagai aktivitas dan tugas menggunakan perangkat *mobile* yang mereka miliki. Aplikasi ini memiliki banyak fungsi yang berguna, seperti hiburan, komunikasi, produktivitas, serta kontrol dan pengendalian perangkat elektronik. Aplikasi berbasis *Android* digunakan sebagai antarmuka untuk mengendalikan perangkat elektronik melalui Mikrokontroler dan modul jaringan seperti *bluetooth* atau *internet*. Dengan demikian, aplikasi berbasis *Android* memberikan kemudahan dan fleksibilitas bagi pengguna dalam mengoperasikan sistem dan perangkat elektronik secara jarak jauh.

BAB III

METODE PELAKSANAAN

3.1 Pendahuluan

Metode pelaksanaan merupakan metode serta tahapan yang akan digunakan untuk memberikan gambaran sistematika pelaksanaan penelitian dari awal hingga akhir dalam rangka menyelesaikan permasalahan yang diteliti. Adapun tahapan yang dilakukan dalam proses pembuatan penelitian dapat dilihat pada *flowchart* sebagai berikut:



Gambar 3. 1 Diagram Alir Pelaksanaan Proyek Akhir

3.1.1 Identifikasi Masalah

Tahapan identifikasi masalah dilakukan untuk mengetahui permasalahan yang diangkat pada penelitian ini. Adapun hasil dari identifikasi masalah dapat dilihat pada rumusan masalah di BAB I.

3.1.1.1 Metodologi Kuantitatif

Penelitian menggunakan metode kuantitatif yang melibatkan aksi simulasi pagar, yang nanatinya akan memberikan data hasil pengujian sebagai hasi serta bandingan untuk menganalisa tingkat efesiensi penggunaan pagar tersebut.

3.1.2 Studi Literatur

Tahapan studi literatur dilakukan untuk menemukan solusi yang dapat membantu memberikan menyelesaikan berdasarkan studi permasalahan sebelumnya. Adapun dalam studi literatur kali ini dilakukan dengan cara mencari, membaca dan menyimpulkan jurnal dan penelitian terdahulu yang valid untuk menemukan dasar teori yang baik untuk penelitian ini.

3.1.3 Perumusan Hipotesis

Dari tahapan studi literatur yang sudah dilakukan, didapatkan solusi dari masalah yang diangkat yaitu dengan mengembangkan *Smart Fence* berbasis IoT. Adapun hipotesis yang dibuat sebagai berikut:

a. Hipotesis mengenai masalah:
    a) Masalah yang disebabkan oleh banyaknya isu tindak kriminal yang telah terjadi berdampak pada cara masyarakat untuk melindungi rumah mereka
    b) Masalah yang disebabkan oleh banyaknya isu tindak kriminal berdampak pada teknologi keamanan.

b. Hipotesis mengenai solusi:
    a) Penerapan automisasi pada pagar dengan mengimplementasikan teknologi berbasis IoT dapat meminimalisir isu tindak kriminal terutama pencurian

b) Penerapan automisasi pada pagar dengan mengimplementasikan teknologi berbasis IoT dapat meningkatkan efesiensi dari segi waktu.

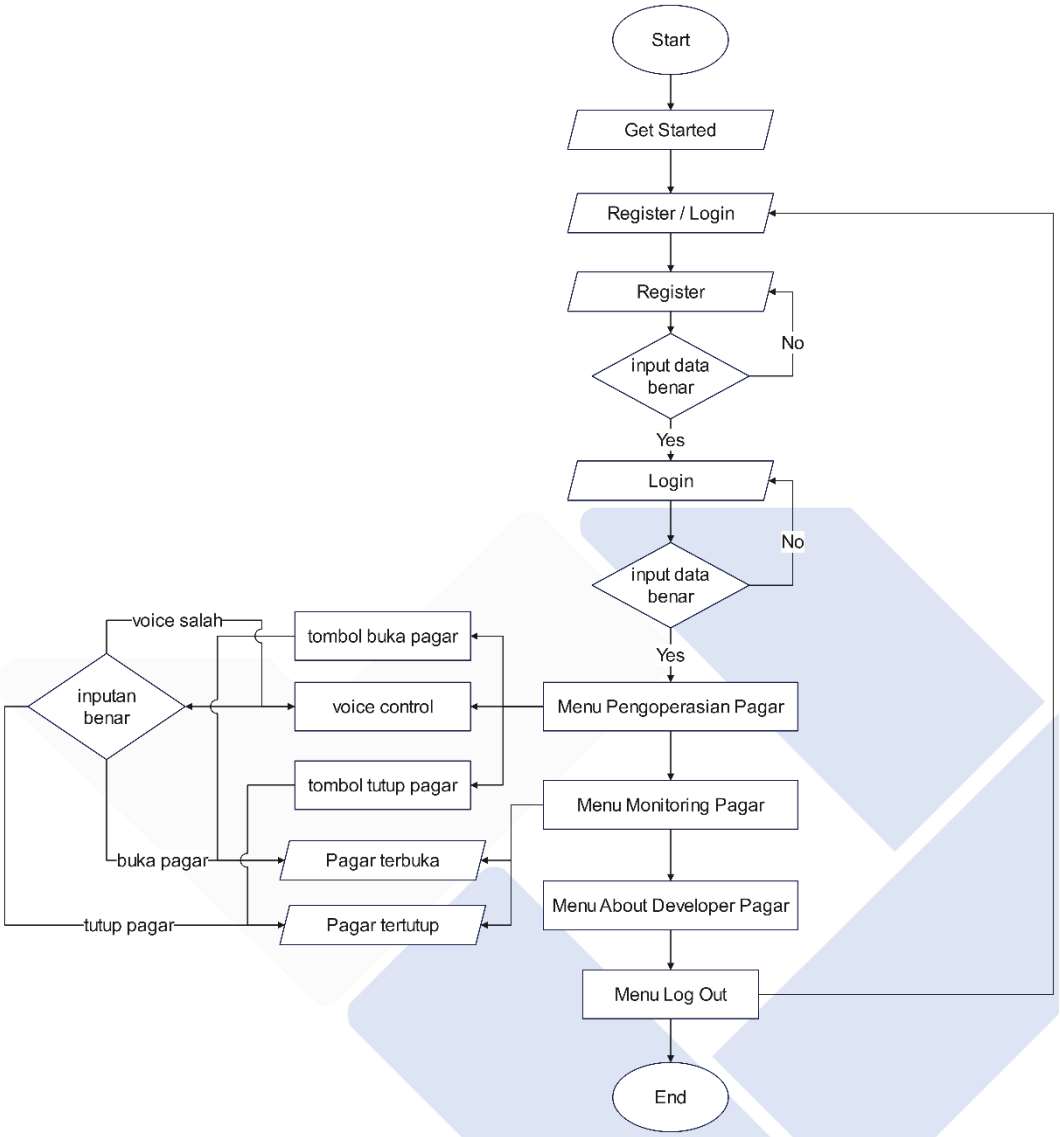
c. Hipotesis mengenai hasil:

a) Meningkatkan keamanan suatu tempat dengan memadukan teknologi berbasis IoT sehingga dapat diakses dari manapun.

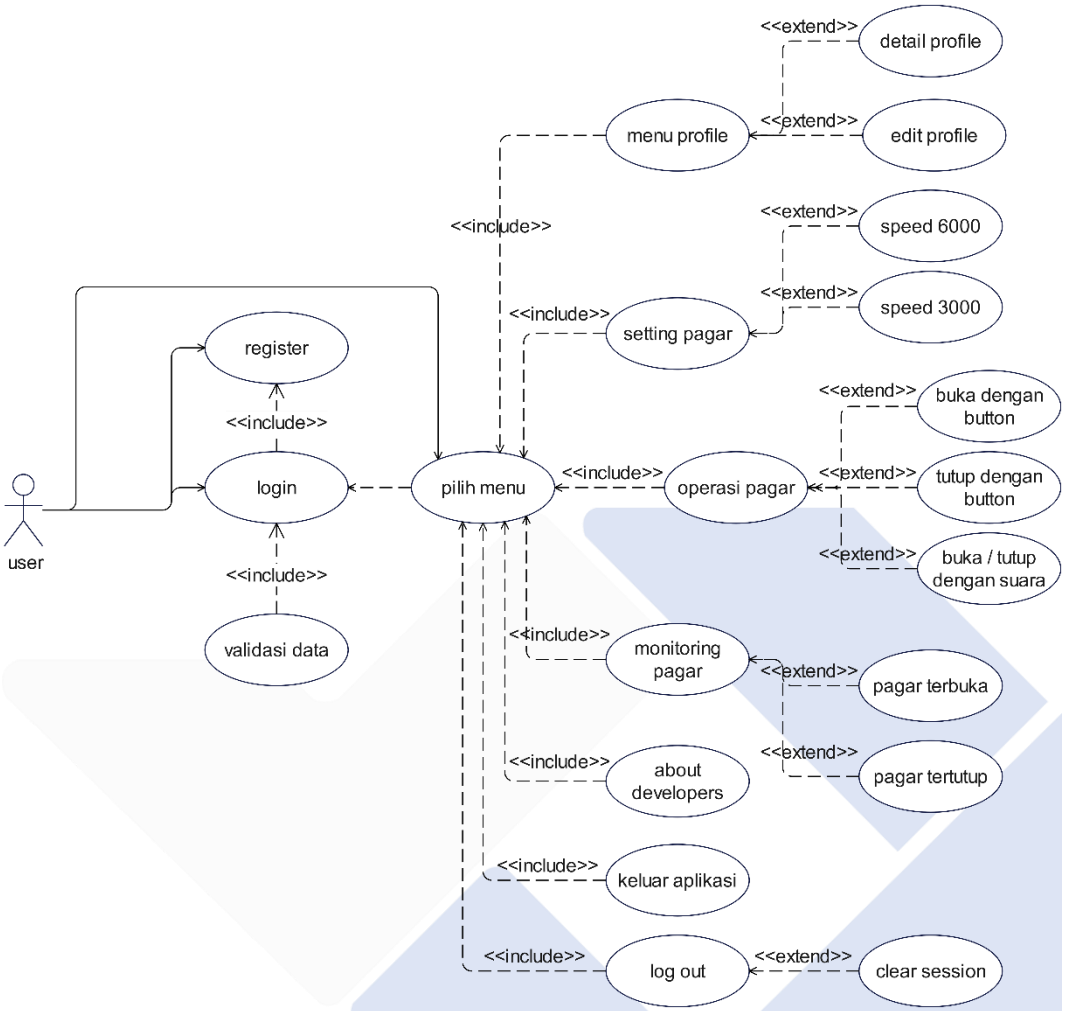
b) Menerapkan teknologi berbasis IoT pada pagar dapat meningkatkan efesiensi waktu yang dibutuhkan dalam mengoperasikan pagar dan juga dapat mempermudah masyarakat dalam mengoperasikan pagar itu sendiri.

3.2 Analisa Kebutuhan Sistem

Pada tahapan analisa kebutuhan sistem ini meliputi spesifikasi fungsional, perangkat keras dan juga perangkat lunak yang akan digunakan. Adapun analisa dari spesifikasi fungsional dari sistem *Smart Fence* ini yaitu mampu membuka pagar berdasarkan *input*an *button visual* dan juga *voice control* yang terdapat pada aplikasi berbasis *Android* dan juga sistem dapat melakukan *monitoring* terhadap status dari pagar tersebut. Pada sistem kali ini menggunakan beberapa perangkat keras diantara lain berupa yang terhubung dengan *NodeMCU ESP32, motor stepper, breadboard,* kabel *jumper, power supply* dan lain-lain. Analisa penggunaan perangkat lunak pada sistem ini berupa komunikasi 2 arah antara aplikasi berbasis *Android* dengan Mikrokontroler, akses *database* dan *output* status pagar yang akan ditampilkan pada menu *monitoring*.



Gambar 3. 2 Diagram *Flowchart* Sistem



Gambar 3. 3 Diagram *Use Case* Sistem

Dibawah ini merupakan *Use Case* skenario yang dibuat untuk menggambarkan lebih jelas aksi yang dilakukan oleh aktor dan reaksi sitem terhadap aksi yang dilakukan.

Nama *Use Case* : *Register*

Use Case Skenario

Tabel 3.1 Use Case Scenario Register

Aksi Aktor	Respon Sistem
Skenario Normal	
1. Memasukan *email*, *username* dan *password*	2. Validasi data yang di*input* 3. Masuk ke menu *login*

Skenario Alternatif	
1. Memasukan *email, username* dan *password*	2. Validasi data yang di*input*
	3. Menampilkan *pop up* data yang dimasukan sepertinya sudah terdaftar atau sudah digunakan
4. Memasukan *email, username* yang belum terdaftar atau belum digunakan	5. Validasi data yang di*input*
	6. Masuk ke menu *login*

Nama *Use Case* : *Login*

Use Case Skenario

Tabel 3 2 Use Case Scenario Login

Aksi Aktor	Respon Sistem
Skenario Normal	
1. Memasukan *username* dan *password*	2. Validasi data yang di*input*
	3. Masuk ke aplikasi *Smart* Pagar
Skenario Alternatif	
1. Memasukan *username* dan *password*	2. Validasi data yang di*input*
	3. Menampilkan *pop up* data yang dimasukan sepertinya belum terdaftar gunakan akun yang sudah terdaftar
4. Memasukan *username* dan *password* yang sudah terdaftar	5. Validasi data yang di*input*
	6. Masuk ke aplikasi *Smart* Pagar

Nama *Use Case* : Pilih menu; Menu *Profile*

Use Case Skenario

Tabel 3 3 Use Case Scenario Menu Profile

Aksi Aktor	Respon Sistem
Skenario Normal	
1. Memilih menu detail *profile*	2. Menampilkan halaman detail *profile*
3. Memilih menu *edit profile*	4. Menampilkan halaman *edit profile*
5. Memasukan *username* dan *password* yang baru	6. Validasi data yang di*input*
	7. Melakukan *update* data *profile user*, lalu mengarahkan *user* ke halaman *login*
Skenario Alternatif	
1. Memilih menu detail *profile*	2. Menampilkan halaman detail *profile*
3. Memilih menu *edit profile*	4. Menampilkan halaman *edit profile*
5. Memasukan *username* dan *password* yang baru	6. Validasi data yang di*input*
	7. Menampilkan *pop up* sepertinya data yang anda masukan sudah terdaftar atau digunakan orang lain
8. Memasukan *username* dan *password* baru yang benar	9. Validasi data yang di*input*
	10. Melakukan *update* data *profile user*, lalu mengarahkan *user* ke halaman *login*

Nama *Use Case* : Pilih menu; Menu *setting* pagar

Use Case skenario

Tabel 3 4 Use Case Scenario Setting Pagar

Aksi Aktor	Respon Sistem
Skenario Normal	

1. Memilih menu *setting* pagar	2. Menampilkan halaman *setting* pagar
3. Menekan *button speed* 3000	4. Mengirim data kecepatan pagar ke *database*
5. Menekan *button speed* 6000	6. Mengirim data kecepatan pagar ke *database*

Nama *Use Case* : Pilih menu; Menu Pengoperasian Pagar

Use Case skenario

Tabel 3 5 Use Case Scenario Pengoperasian Pagar

Aksi Aktor	Respon Sistem
Skenario Normal	
1. Memilih menu pengoperasian pagar	2. Menampilkan halaman pengoperasian pagar
3. Menekan *button* buka pagar	4. *Mengupdate* data yang ada di *database* dengan nilai “buka pagar”
5. Menekan *button* tutup pagar	6. *Mengupdate* data yang ada di *database* dengan nilai “tutup pagar”
7. Menekan *button voice control* dan mengucapkan kalimat “buka pagar”	8. *Mengupdate* data yang ada di *database* dengan nilai “buka pagar”
9. Menekan *button voice control* dan mengucapkan kalimat “tutup pagar”	10. *Mengupdate* data yang ada di *database* dengan nilai “tutup pagar”
Skenario Alternatif	
1. Menekan *button voice control* dan mengucapkan kalimat yang bukan buka pagar atau tutup pagar	2. Tidak merespon dan tidak melakukan *update* data ke *database*
3. Menekan *button voice control* dan mengucapkan kalimat “buka pagar”	4. *Mengupdate* data yang ada di *database* dengan nilai “buka pagar
5. Menekan *button voice control* dan mengucapkan kalimat “tutup pagar”	6. *Mengupdate* data yang ada di *database* dengan nilai “tutup pagar”

Nama *Use Case* : Pilih menu;*Monitoring* pagar

Use Case skenario

Tabel 3 6 Use Case Scenario Monitoring Pagar

Aksi Aktor	Respon Sistem
Skenario Normal	
1. Memilih menu *monitoring* pagar	2. Menampilkan halaman *monitoring* pagar dan akan menampilkan status pagar apakah terbuka atau tertutup

Nama *Use Case* : Pilih menu;*About developer*

Use Case skenario

Tabel 3 7 Use Case Scenario About Developer

Aksi Aktor	Respon Sistem
Skenario Normal	
1. Memilih menu *about developer*	2. Menampilkan halaman *about developer*

Nama *Use Case* : Pilih menu;*Logout*

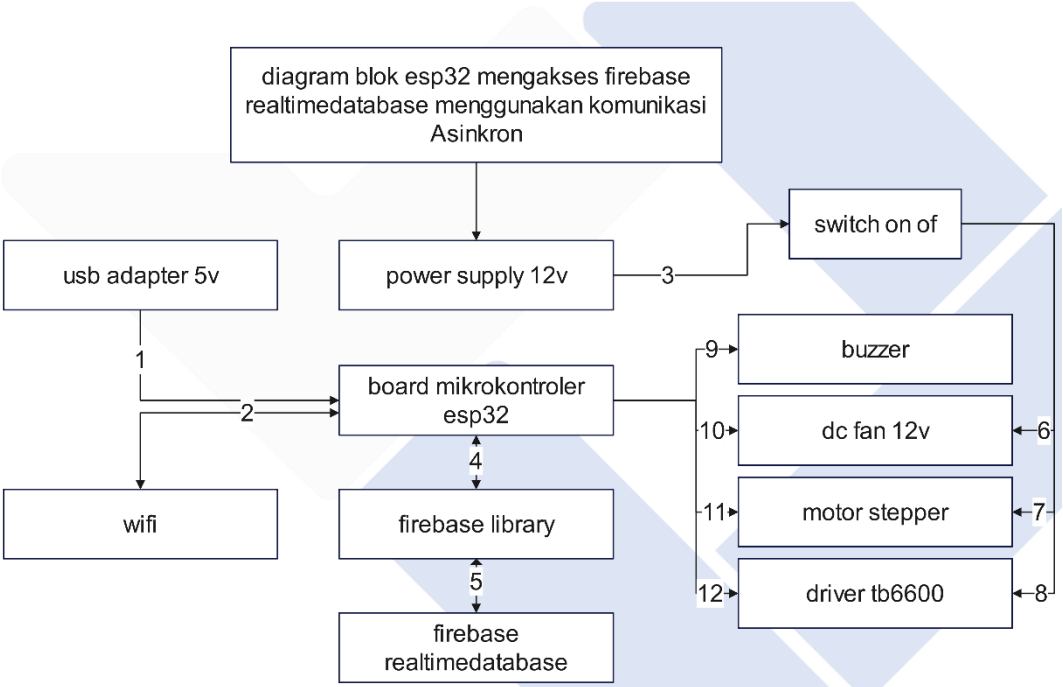
Use Case skenario

Tabel 3 8 Use Case Scenario Logout

Aksi Aktor	Respon Sistem
Skenario Normal	
1. Memilih menu *Logout*	2. Menampilkan *pop up* apakah anda yakin untuk *Logout?*
3. Memilih menu Tidak	4. Mengarahkan *user* ke halaman utama
5. Memilih menu Ya	6. Menghapus *session login user* dan mengarahkan *user* ke halaman *login.*

3.3 Pengembangan sistem *Smart Fence*

Pengembangan sistem *Smart Fence* ini dilakukan untuk melanjutkan dari hasil analisa yang telah dilakukan sebelumnya. Adapun proses perencanaannya meliputi rancangan desain arsitektur sistem, identefikasi komponen komponen yang digunakan, serta penentuan bahan dari konstruksi mekaniknya. Adapun pada tahapan pengembangan sistem ini akan dibagi menjadi beberapa perancangan antara lain, perancangan mekanik, kontrol, dan sistem aplikasi berbasis *Android*. Adapun desain dari keseluruhan rancangan dapat dilihat pada blok diagram berikut.



Gambar 3. 4 Blok Diagram Sistem *Smart Fence*

Dibawah ini merupakan deskripsi atau penjelasan dari diagram blok diatas untuk memudahkan pembaca dalam memahami alur dari interaksi antar blok *diagram*

Tabel 3 9 Deskripsi Interaksi antar Blok Diagram

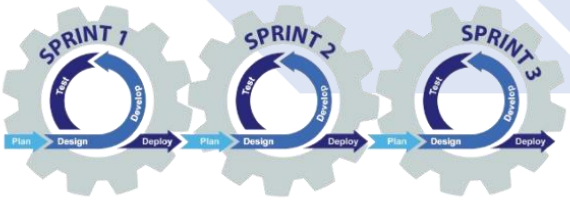
Langkah	Skenario penggunaan	*Input*	Output	Interaksi antar blok
1.	Suplai daya	Arus AC	DC 5V	USB memberikan suplai arus DC 5V sebagai daya nya agar Mikrokontroler dapat berfungsi

2.	Prantara komunikasi	-	-	Wi-fi berfungsi sebagai penghubung komunikasi antara *ESP32* dan server *firebase realtime database*
3.	*Switch on - off*	Arus DC 12V	Arus DC 12V	*Switch* digunakan untuk memutus daya yang di suplai ke komponen elektronik
4.	*Library* penunjang	-	-	Sebagai penunjang agar *ESP32* bisa melakukan komunikasi ke server *firebase realtime database*
5.	Mengambil data yang dikirmkan dari aplikasi	-	Data dari server *firebase realtime database*	*Database* digunakan untuk menampung semua data yang dikirimkan dari aplikasi yang nantinya akan diterima oleh *ESP32*
6.	Suplai daya	Arus AC	Arus DC 12V	Catu daya 12V memberikan suplai daya agar dc *fan* dapat berfungsi
7.	Suplai daya	Arus AC	Arus DC 12V	Catu daya 12V memberikan suplai daya agar motor *stepper* dapat berfungsi
8.	Suplai daya	Arus AC	Arus DC 12V	Catu daya 12V memberikan suplai daya agar *driver* tb6600 dapat berfungsi
9.	Tanda peringatan	Logika *high*	Suara “beep”	*Buzzer* digunakan untuk memberi tanda ketika *ESP32* sudah terkoneksi dengan Wi-fi maka *buzzer* akan berbunyi
10.	Pendingin	Arus DC 12V	Angin pendingin	Dc *fan* digunakan untuk mendinginkan berbagai komponen elektronik yang digunakan

11.	Penggerak	Logika *high* dan *low*	Motor pengggerak berputar searah jarum jam jika menerima logika *high* dan sebaliknya	Motor *stepper* digunakan sebagai penggerak yang nantinya akan menggerakan pagar untuk menutup ataupun membuka pagar
12.	Mengatur motor *stepper*	Logika *high* dan *low*	Sinyal yang digunakan untuk mengatur *mikrostep*	*Driver* digunakan untuk mengatur gerak dari motor seperti mengatur *mikrostep* yang menentukan berapa langkah yang dibutuhkan motor untuk melakukan 1 putaran.

3.4 Metode Agile Scrum

Tahapan Agile Software Development merupakan sebuah metodologi pengembangan perangkat lunak yang didasarkan pada proses pembuatan yang dilakukan secara berulang berulang di mana sesuai dengan aturan serta solusi yang disepakati dikembangkan dengan kolaborasi antar tiap tim secara terorganisir dan terstruktur.[21]



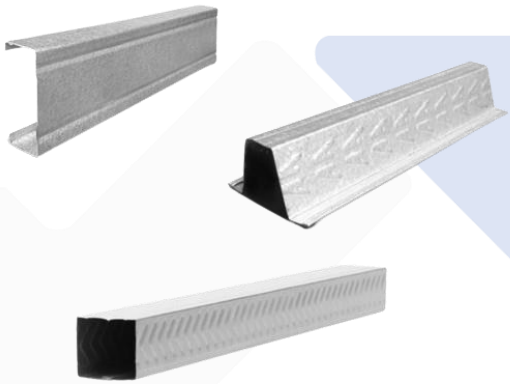
Gambar 3. 5 Diagram Metode Agile Scrum

3.5 Pengembangan Kontruksi Mekanik

Adapun pada pengembangan konstruksi mekanik pada sistem ini meliputi beberapa tahapan sebagai berikut:

3.5.1 Penentuan Ukuran dan Bahan Pada Konstruksi Mekanik

Dalam penelitian ini, pembuatan konstruksi mekanik pagar nantinya akan dibuat hanya sebagai prototype dalam penelitian ini. Pagar sendiri akan dibuat menggunakan bahan baja ringan, adapun beberapa jenis baja ringan yang digunakan adalah kanal c, reng dan juga hollow 4x4. sedangkan untuk kontrol sistem akan diletakan pada sebuah box yang dimensi ukurannya lebih kecil yaitu 17 cm x 6 cm. Penjelasan lebih lanjut akan disampaikan pada bab 4.



Gambar 3. 6 Kombinasi Penggunaan Baja Ringan

3.5.2 Peletakan Komponen Pada Konstruksi Mekanik

Adapun peletakan setiap komponen elektrik untuk penggerak pagar yang akan digunakan diletakan pada sebuah box sebagaimana sudah dijelaskan diatas. Adapun komponen – komponen yang akan diletakan didalam box tersebut antara lain, *NodeMCU ESP32*, motor *stepper*, kabel *jumper*, buzzer, driver tb6600 yang mana *NodeMCU ESP32* dan komponen lain akan di rakit pada sebuah pcb berlubang.

3.6 Perancangan Sistem

Tahap ini adalah tahapan rancangan dari sistem *Smart Fence* berbasis IoT menggunakan *Visual Studio Code*. Ketika *user* membuka aplikasi maka maka *user* diwajibkan melakukan *Register* jika belum punya akun, namun jika sudah pernah *Register user* hanya perlu *Login*, setelah melakukan *Login* maka *user* akan

diarahkan langsung ke page utama yang berisi page pengoperasian pagar, *monitoring* pagar, about *developer*, serta *Logout*. Pada menu pengoperasian pagar ketika *user* menekan button buka pagar maka sistem akan melakukan *update* data yang ada di *realtimedatabase* yang nantinya oleh Mikrokontroler data perubahan itu akan diterima dan akan membuka pagar, begitupun sebaliknya. Namun pada pengoperasian pagar menggunakan *voice control user* akan diminta untuk mengucapkan kalimat buka pagar, yang mana nanti sistem akan memproses dan melakukan konversi dari suara ke *text* lalu akan dilakukan *update* lagi data yang ada di *realtimedatabase* lalu Mikrokontroler akan kembali menerima data yang sudah di *update* akan tetapi pada Mikrokontroler akan melakukan pengecekan terlebih dahulu, jika kalimat yang diucapkan adalah buka pagar maka pagar akan terbuka, begitupun sebaliknya namun jika kalimat yang diucapkan tidak sama dengan buka atau tutup pagar maka tidak akan terbuka.

3.7 Teknik Analisis Data

Teknik analisis data pada penelitian kali ini akan melakukan pengujian pada jarak, *voice*, serta *monitoring* dengan mempertimbangkan rata-rata waktu respon dan n rata-rata waktu berhasil memiliki peran penting dalam memahami dan meningkatkan kinerja suatu sistem atau layanan. Analisis data jarak memungkinkan identifikasi pola atau tren dalam respons sistem terhadap variasi jarak, sementara analisis data suara membantu mengevaluasi kualitas komunikasi suara. Melalui *monitoring*, rata-rata waktu respon dan waktu berhasil memberikan informasi tentang performa sistem seiring waktu. Analisis data ini berguna untuk evaluasi kinerja, deteksi masalah jaringan atau audio, dan respons cepat. Beberapa contoh rumus yangakan digunakan berupa :

a. $$\boldsymbol{Rata - Rata\ Waktu\ Respon} = \frac{Total\ waktu\ respon}{Total\ pengujian}$$

b. $$\boldsymbol{Rata - Rata\ Kecepatan\ Transmisi} = \frac{Total\ waktutransmisi}{Total\ pengujian}$$

c. $$\boldsymbol{Persentase\ Berhasil} = \frac{Total\ keberhasilan}{Total\ pengujian}$$

BAB IV

PEMBAHASAN

4.1 Implementasi

Proses perancangan dari penelitian dimulai dengan melakukan perancangan sistem hardware sistem serta *software*, hingga melakukan pengujian sistem.

4.1.1 Pembuatan Perangkat Keras

Dalam bab ini, penulis akan membahas terkait proses implementasi serta perakitan rangkaian *hardware* projek dimulai dengan proses perancangan hingga pengujian alat serta aplikasi *Smart Fence* berbasis IoT.

4.1.2 Pembuatan Perangkat Lunak

Tahapan Pembuatan atau perancangan perangkat lunak akan membahas terkait desain yang akan digunakan serta mengimplementasikan sistem *Voice Control* serta *monitoring* pada aplikasi *Android*. Pemanfaatan bidang IoT pada kontrol suara dapat mempermudah interaksi dan memanfaatkan kemajuan teknologi secara maksimal sehingga sistem *Smart Fence* dapat dipahami dengan mudah bagi pengguna[22]. Pagar yang dapat dikendalikan dengan mudah akan meningkatkan kenyamanan dan memberikan pengalaman kepada pengguna aplikasi, terutama jika pagar bergerak secara *real-time*[23]. Berikut merupakan desain awal gambar rancangan antarmuka awal sistem.

4.1.3 *Pre-Sprint*

Pada tahap *pre-sprint* ini dilaksanakan kurang lebih selama 1 bulan, di tahap ini penulis melakukan research guna mengumpulkan *user story* yang relevan dengan kebutuhan pada sistem aplikasi *Smart Fence* atau *Smart* Pagar, dari semua *user story* yang sudah dikumpulkan nantinya akan dibuat *product backlog* yang terdiri dari *user story*, prioritas serta estimasi.

Prioritas dalam pembuatan aplikasi ini dibagi menjadi 3 yaitu, *High, Medium dan Low*. PrIoTitas *High* yang artinya prioritas dari fitur atau *function* itu harus dikerjakan terlebih dahulu dikarenakan merupakan fitur utama dari sistem. Prioritas *Medium* akan dikerjakan setelah *High* dikarenakan item yang memiliki prioritas *Medium* merupakan fitur yang nantinya akan menunjang fitur yang memiliki prioritas *High.* Sedangkan prioritas *Low* akan dikerjakan terakhir kali, dikarenakan item yang memiliki prioritas *Low* merupakan fitur yang tidak begitu penting sehingga tidak akan mengganggu alur kerja sistem.

Estimasi merupakan estimasi upaya yang diperlukan dalam menyelesaikan item tersebut, adapun estimasi akan dibagi menjadi beberapa tingkatan 1 sampai 5, yang mana estimasi dengan tingkat 1 sangat mudah untuk dikerjakan sedangkan estimasi dengan tingkat 5 sangat sulit untuk dikerjakan.

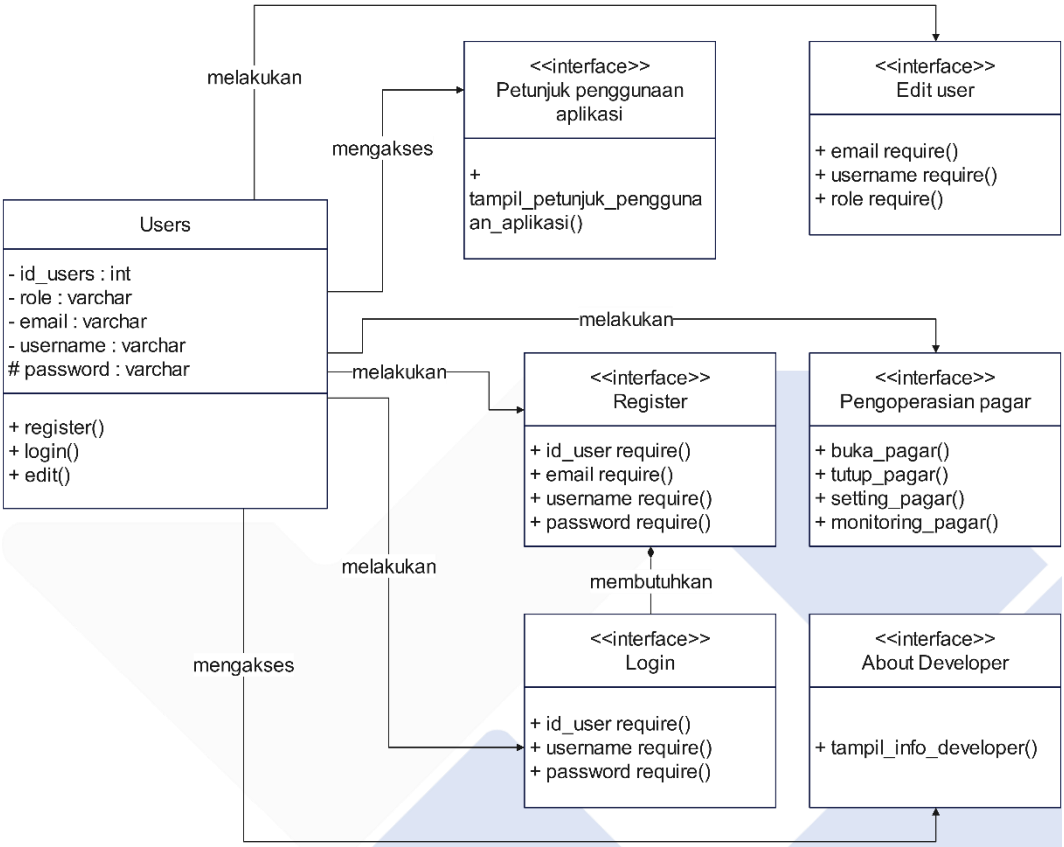
Product backlog yang dibuat ini merupakan versi pertama dan merupakan *product backlog* semi final dari sistem yang akan dibuat, yang kemungkinan nanti akan terus dilakukan perubahan seiring dengan *sprint* yang dilakukan, tabel dibawah merupakan tabel *product backlog final* yang telah disusun selama tahap pengembangan.

Tabel 4. 1 Product Backlog

No.	Sebagai	Ingin membuat	Agar	Prioritas	Estimasi	*Sprint*
1.	Konfigurasi *Visual Studio Code*		Memudahkan *developer* dalam membuat aplikasi	*High*	5	1
2.	Konfigurasi *android studio*		Memudahkan *developer* dalam melakuan *running* test aplikasi pada *emulator*	*High*	3	1
3.	Perancangan *database*			*High*	5	1
4.	*Developer*	Fitur petunjuk penggunaan aplikasi	Agar memudahkan pengguna dalam menggunakan aplikasi	*High*	3	1
5.	*Developer*	Fitur opsi *login*	Agar pengguna bisa memilih untuk melakukan *register* terlebih dahulu atau langsung *login* jika pengguna sudah memiliki akun	*High*	3	1
6.	*Developer*	Fitur *register*	Agar pengguna bisa mendaftarkan akun mereka ke aplikasi yang dibutuhkan untuk *register* antara lain *email*, *username* dan *password* pengguna	*High*	5	1
6.	*Developer*	Fitur *login*	Agar pengguna bisa *login* ke aplikasi dan juga agar hanya *user* yang sudah *register* saja yang bisa menggunakan aplikasi	*High*	5	1
7.	*Developer*	Fitur detail *profile*	Agar pengguna bisa melihat detail dari *profile* akun mereka seperti *email* yang digunakan pengguna dan *username* pengguna	*High*	4	2

8.	*Developer*	Fitur *edit profile*	Agar pengguna bisa meng*edit* atau mengubah informasi yang ada di *profile* mereka seperti *username* dan *password* pengguna	*High*	4	2
9.	*Developer*	Fitur pengoperasian pagar	Agar pengguna bisa mengoperasikan pagar mereka melalui aplikasi mereka, seperti membuka dan menutup pagar	*High*	5	2
10.	*Developer*	Fitur *monitoring* pagar	Agar pengguna bisa *memonitoring* status dari pagar mereka, apakah pagar mereka sedang terbuka ataupun sedang tertutup	*High*	3	2
11.	*Developer*	Fitur pengaturan pagar	Agar pengguna bisa mengatur pagar mereka, tapi hanya ada 1 parameter yang bisa diatur atau di setting oleh pengguna adalah kecepatan motor penggerak	*High*	3	2
12.	*Developer*	Fitur about *developer*	Agar pengguna bisa melihat info dari siapa yang mengembangkan aplikasi *Smart Fence* atau *Smart* Pagar yang mereka gunakan	*Low*	2	3
12.	*Developer*	Fungsi *Logout*	Agar pengguna bisa *Logout* atau mengeluarkan akun mereka dari aplikasi, tapi ketika pengguna melakukan *Logout* maka ketika masuk ke aplikasi lagi pengguna diwajibkan melakukan *login* terlebih dahulu	*High*	3	3

Dari rancangan *product backlog* diatas dirancanglah *class diagram* sesuai dengan *product backlog* yang sudah dibuat:



Gambar 4. 1 Class Diagram

4.1.4 *Sprint* Pertama

Sprint pertama ini berfokus membuat fitur – fitur utama alur dari sistem *Smart Fence* atau *smart* pagar yang dilengkapi *register* dan *login*. Hasil dari *sprint* pertama berupa aplikasi yang telah dapat melihat petunjuk penggunaan aplikasi, memilih opsi *login*, *register* dan juga *login*.

4.1.4.1 *Plan* dan *Design Sprint* Pertama

Pada tahapan ini, dibentuk sebuah *Sprint Backlog* untuk *Sprint* pertama yang diambil dari *Product Backlog* dan berfokus mengerjakan fitur yang memiliki prioritas *High* yang menjadi alur utama dari sistem *Smart Fence* atau *Smart* Pagar. Berikut merupakan *Sprint Backlog* iterasi pertama.

Tabel 4.2 Sprint Backlog 1

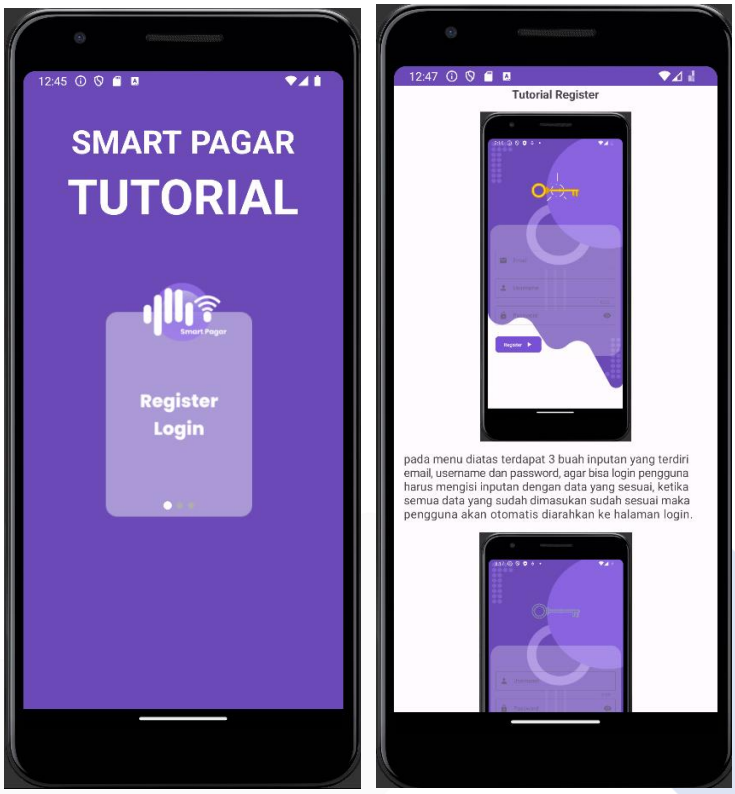
No.	*User Story*	*Task*	*Person in Change*	*Priority*	*estimation*
1.	Konfigurasi *Visual Studio Code*	*Download*, *install* serta melakukan beberapa konfigurasi pada *code editor*	Hasya Akhlaqah Harmie & Rahmattian Diza Dwi Putra	*High*	5
2	Konfigurasi *android studio*	*Download*, *install* serta melakukan beberapa konfigurasi pada *code editor*	Hasya Akhlaqah Harmie & Rahmattian Diza Dwi Putra	*High*	5
3.	Sebagai *developer* saya ingin membuat fitur petunjuk penggunaan aplikasi	Membuat *figma wireframe*, melakukan slicing ui kedalam kode xml *kotlin*, membuat *backend* menggunakan kode *kotlin*	Hasya Akhlaqah Harmie	*High*	3
4.	Sebagai *developer* saya ingin membuat fitur opsi *login*	Membuat *figma wireframe*, melakukan *slicing ui* kedalam kode xml *kotlin*, membuat backend menggunakan kode *kotlin*	Hasya Akhlaqah Harmie	*High*	3
5.	Sebagai *developer* saya ingin membuat fitur *register*	Membuat *figma wireframe*, melakukan *slicing ui* kedalam kode xml *kotlin*, membuat CRUD data *user register*, membuat *backend* menggunakan kode *kotlin* untuk mengirimkan data *user register* ke *database*	Rahmattian Diza Dwi Putra	*High*	5

6.	Sebagai *developer* saya ingin membuat fitur *login*	Membuat *figma wireframe*, melakukan *slicing ui* kedalam kode xml *kotlin*, membuat CRUD data *user login*, membuat *backend* menggunakan kode *kotlin* untuk membandingkan data *input user login* dengan data *user register* yang ada di *database*	Rahmattian Diza Dwi Putra	*High*	5

4.1.4.2 *Build* & *Test Sprint* Pertama

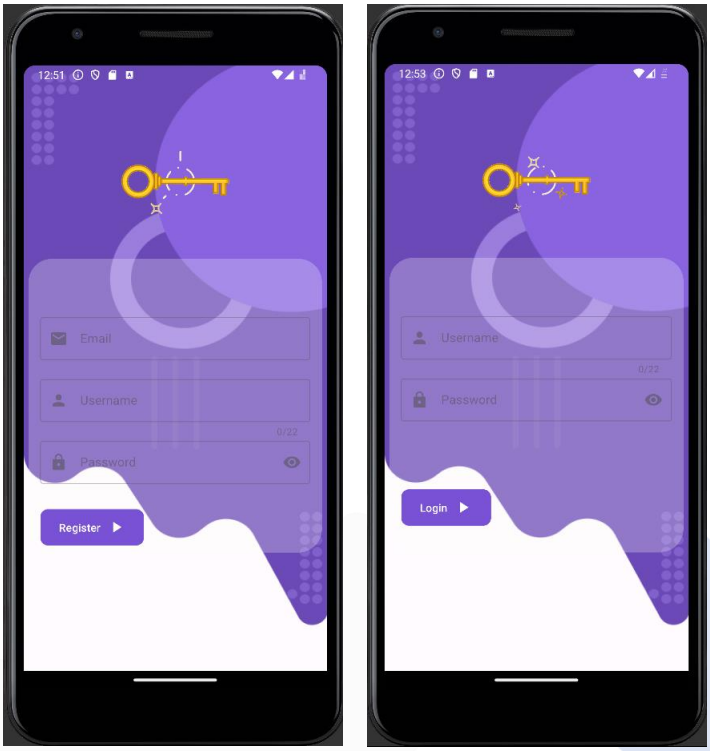
Pada tahapan *build* dari *sprint* pertama dilakukan dengan mengimplementasikan hasil dari *wireframe* dilakukan *slicing ui* kedalam kode xml, setelah selesai melakukan *slicing ui* dilanjutkan membuat kode untuk *backend*. Tahapan ini disertai dengan pengujian yang sudah dilakukan oleh *developer*. Berikut hasil *build* pada iterasi ini:

Gambar 4.2 *Tampilan Get Started*



Gambar 4.3 *Tampilan petunjuk penggunaan aplikasi*

Gambar 4.4 *Tampilan Opsi login*



Gambar 4.5 *Tampilan login dan register*

4.1.4.3 *Review* & *Retrospective Sprint* Pertama

Hasil *review* dan *Retrospective Sprint* yang dilakukan *developer* mendapatkan kesimpulan bahwa fitur – fitur yang sudah dibuat menggunakan *sprint backlog* pertama sudah berfungsi sesuai dengan harapan. Pada *sprint* pertama ini berhasil dibangun aplikasi yang sudah memiliki fitur petunjuk penggunaan aplikasi, *register* dan *login* serta fitur -fitur sederhana lainnya.

4.1.5 *Sprint* Kedua

sprint kedua ini berfokus membuat fitur – fitur utama alur dari sistem *Smart Fence* atau *smart* pagar yang sudah bisa melakukan pengoperasian pagar menggunakan aplikasi, fitur detail *profile*, *edit profile*, *monitoring* dan pengaturan pagar.

4.1.5.1 *Plan* dan *Design Sprint* kedua

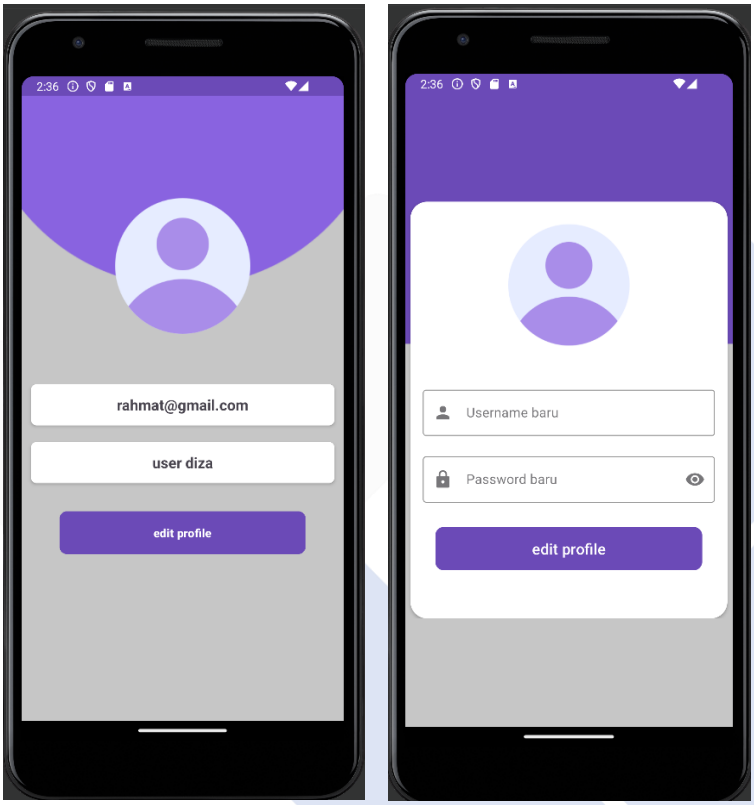
Pada tahapan ini, dibuat *sprint backlog* yang berfokus mengerjakan fitur fitur yang memiliki prioritas *High* dan *Medium* berikut *sprint backlog* iterasi kedua:

Tabel 4. 3 Sprint Backlog 2

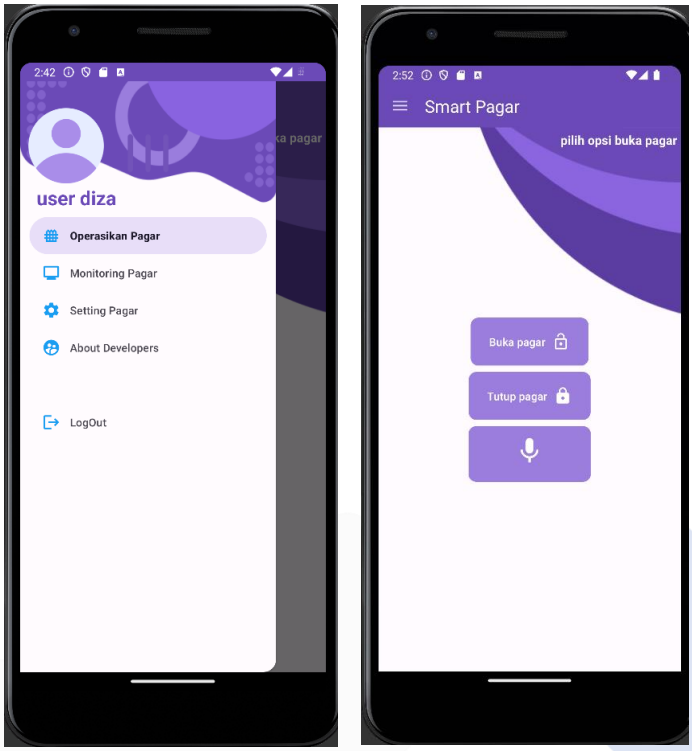
No.	*User Story*	*Task*	*Person in Change*	*Priority*	*estimation*
1.	Sebagai *developer* saya ingin membuat fitur *detail profile*	Membuat *figma wireframe*, melakukan slicing ui kedalam kode xml *kotlin*, membuat *backend* menggunakan kode *kotlin* untuk mengambil data *user* dari *database*	Hasya Akhlaqah Harmie & Rahmattian Diza Dwi Putra	*High*	4
2	Sebagai *developer* saya ingin menambah fitur *edit profile*	Membuat *figma wireframe*, melakukan slicing ui kedalam kode xml *kotlin*, membuat *backend* menggunakan kode *kotlin*	Hasya Akhlaqah Harmie & Rahmattian Diza Dwi Putra	*High*	4
3.	Sebagai *developer* saya ingin membuat fitur petunjuk penggunaan aplikasi pengoperasian pagar	Membuat *figma wireframe*, melakukan slicing ui kedalam kode xml *kotlin*, membuat *backend* menggunakan kode *kotlin*	Rahmattian Diza Dwi Putra	*High*	5
4.	Sebagai *developer* saya ingin membuat fitur opsi *login monitoring*	Membuat *figma wireframe*, melakukan *slicing ui* kedalam kode xml *kotlin*, membuat backend menggunakan kode *kotlin*	Rahmattian Diza Dwi Putra	*High*	3
5.	Sebagai *developer* saya ingin membuat fitur pengaturan pagar	Membuat *figma wireframe*, melakukan *slicing ui* kedalam kode xml *kotlin*, membuat backend menggunakan kode *kotlin*	Rahmattian Diza Dwi Putra	*High*	3

4.1.5.2 *Build & Test Sprint* Kedua

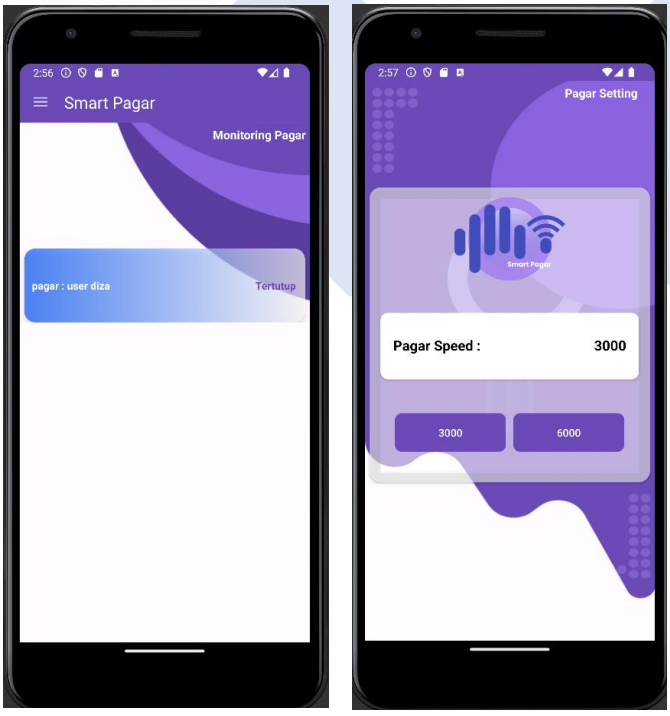
Pada tahapan *build* dari *sprint* kedua dilakukan dengan mengimplementasikan hasil dari *wireframe* dilakukan *slicing ui* kedalam kode xml, setelah selesai melakukan *slicing ui* dilanjutkan membuat kode untuk *backend*. Tahapan ini disertai dengan pengujian yang sudah dilakukan oleh *developer*. Berikut hasil *build* pada iterasi ini:



Gambar 4. 6 *Tampilan detail profile dan edit profile*



Gambar 4. 7 Tampilan *Side Navigation Panel dan* Menu Pengoperasian Pagar



Gambar 4. 8 Tampilan *Monitoring* dan Pengaturan Pagar

4.1.5.3 *Review* dan *Retrospective Sprint* Kedua

Hasil *review* dan *Retrospective Sprint* yang dilakukan *developer* mendapatkan kesimpulan bahwa fitur – fitur yang sudah dibuat menggunakan *sprint backlog* kedua sudah berfungsi sesuai dengan harapan. Pada *sprint* kedua ini berhasil dibangun aplikasi yang sudah memiliki fitur petunjuk penggunaan aplikasi, *register* dan *login,* mengoperasikan pagar, *monitoring* pagar dan pengaturan pagar serta fitur -fitur sederhana lainnya.

4.1.6 *Sprint* Ketiga

sprint ketiga ini berfokus membuat fitur – fitur utama serta tunjangan dari sistem *Smart Fence* atau *smart* pagar yang sudah bisa melakukan menampilkan menu *about developer* dan juga fitur untuk *Logout.*

4.1.6.1 *Plan* dan *Design Sprint* Ketiga

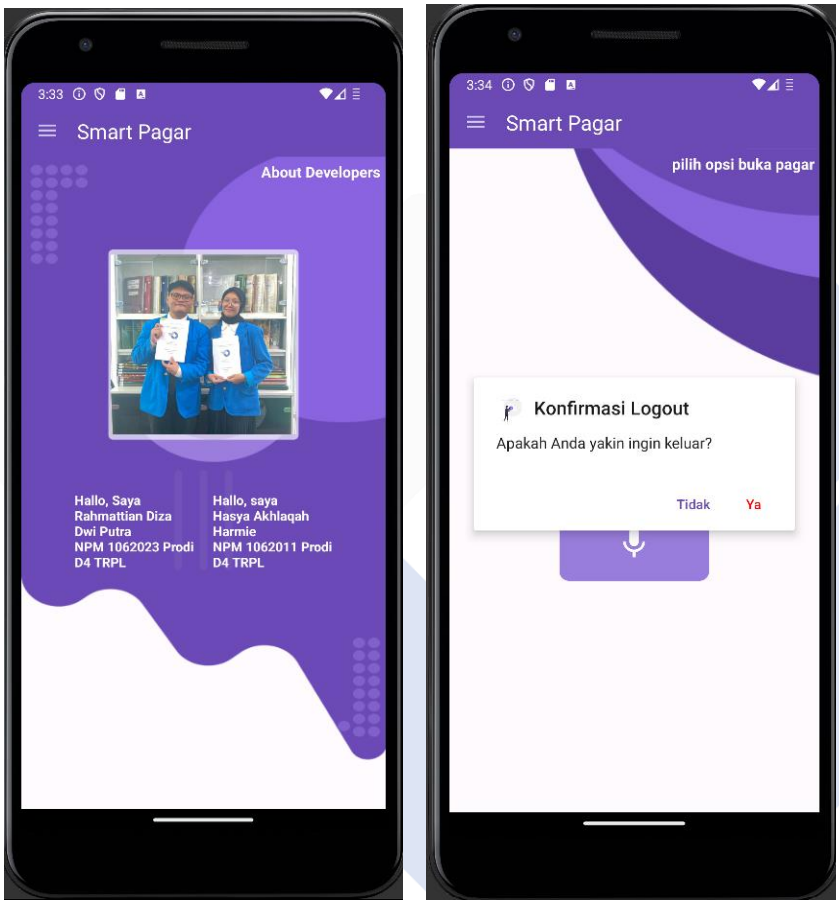
Pada tahapan ini, dibuat *sprint backlog* yang berfokus mengerjakan fitur fitur yang memiliki prioritas *High* dan *Medium* berikut *sprint backlog* iterasi ketiga:

Tabel 4. 4 Sprint Backlog 3

No.	*User Story*	*Task*	*Person in Change*	*Priority*	*estimation*
1.	Sebagai *developer* saya ingin membuat fitur *about developer*	Membuat *figma wireframe*, melakukan slicing ui kedalam kode xml *kotlin*, membuat *backend* menggunakan kode *kotlin* untuk mengambil data *user* dari *database*	Rahmattian Diza Dwi Putra	*High*	4
2	Sebagai *developer* saya ingin menambah fitur *Logout*	Membuat *figma wireframe*, melakukan slicing ui kedalam kode xml *kotlin*, membuat *backend* menggunakan kode *kotlin*	Rahmattian Diza Dwi Putra	*High*	4

4.1.6.2 *Build* dan *Test Sprint* Ketiga

Pada tahapan *build* dari *sprint* ketiga dilakukan dengan mengimplementasikan hasil dari *wireframe* dilakukan *slicing ui* kedalam kode xml, setelah selesai melakukan *slicing ui* dilanjutkan membuat kode untuk *backend*. Tahapan ini disertai dengan pengujian yang sudah dilakukan oleh *developer*. Berikut hasil *build* pada iterasi ini:



Gambar 4. 9 Tampilan About developer dan tampilan Logout

4.1.6.3 *Review* dan *Retrospective Sprint* Ketiga

Hasil *review* dan *Retrospective Sprint* yang dilakukan *developer* mendapatkan kesimpulan bahwa fitur – fitur yang sudah dibuat menggunakan *sprint backlog* kedua sudah berfungsi sesuai dengan harapan. Pada *sprint* ketiga ini berhasil dibangun aplikasi yang sudah memiliki fitur about *developer* dan *Logout* serta fitur -fitur sederhana lainnya.

4.1.7 Implementasi Hardware

Penggunaan bahan untuk pembuatan *hardware* berdasarkan rancangan dan kebutuhan sistem dalam membangun alat pengendalian pagar. Spesifikasi *hardware* yang akan digunakan dapat diurai dengan keterangan sebagai berikut:

1. Modul

a. *Power Supply* 10A/12V

Modul ini berfungsi sebagai penyedia sumber daya listrik dengan tegangan *output* sebesar 12 volt.

b. Motor *Stepper Nema* 23

Motor ini memberikan presisis pergerakan dengan standar *NEMA* 23 dan sehingga dapat memindahkan motor sesuai dengan jumlah langkah yang diterimanya.

c. Kabel *Jumper MFMM*

Konektor berbentuk kabel dengan dua bagian, *yaitu Male to Female* serta *Male to Male.* Kabel ini digunakan untuk menyambungkan komponen perangkat kedalam *pin* atau port yang diminati.

d. Timah Solder

Logam yang digunakan untuk menyambungkan suatu komponen elektronik dengan menggunakan proses *soldering*. Umumnya digunakan ketika digunakan untuk memperbaiki atau menyambungkan perangkat elektronik.

e. TB6600 Motor *Drive*

Modul ini mengendalikan motor (*stepper motor driver*) dalam aplikasi elektronika serta mekatronika.

f. *Solder* 936

Modul digunakan dalam proses penyambungan alat dengan melelehkan solder pada permukaan logam yang akan dihubungkan.

g. *Breadboard*

Sebuah papan sirkuit yang digunakan untuk menyusun dan menyambungkan komponen elektronik tanpa *solder*.

h. *NodeMCU* Sp32

   Modul *wifi* untuk *Internet of Things* yang kerap dipakai untuk pemrosesan dan komunikasi *wifi* terintegrasi.

i. Baja Ringan kanal c

j. Baja Ringan reng

k. Baja Ringan *hollow* 4x4

l. Gear

m. Rantai Sepeda

n. PCB berlubang 18x7

   Sirkuit yang digunakan untuk memasang komponen elektronik serta konektor.

o. Switch Button

2. Penunjang

a. Kabel USB

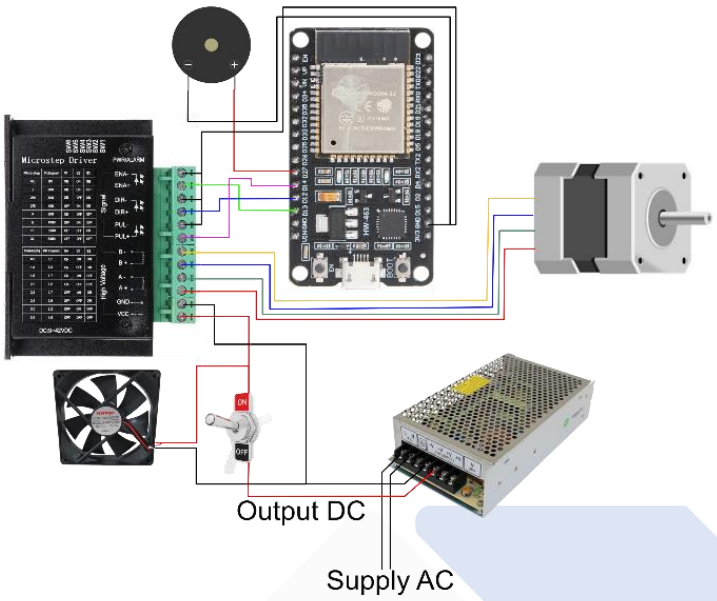
b. *Handphone Android*

c. Female DC Power Supply Connector

konektor daya DC yang dirancang untuk menerima plug atau konektor berjenis DC yang sesuai. Konektor ini sering digunakan dalam berbagai aplikasi elektronik dan perangkat listrik untuk menyediakan daya DC.

d. Jack DC Baut *Female*

Pada perancangan perangkat *hardware* dapat kita lihat skema komponen yang digunakan pada gambar dibawah ini beserta tampilan prototipenya,



Gambar 4. 10 Skema Rangkaian Sistem Smart Fence

Tabel 4. 5 Konfigurasi Motor dan Driver

Driver tb6600	*Motor stepper nema* 23
A+	Kabel merah motor
A-	Kabel hijau motor
B+	Kabel biru motor
B-	Kabel kuning motor

Tabel 4. 6 Konfigurasi Catu Daya dan Driver

Driver tb6600	Catu daya
Vcc	V+
Gnd	V-

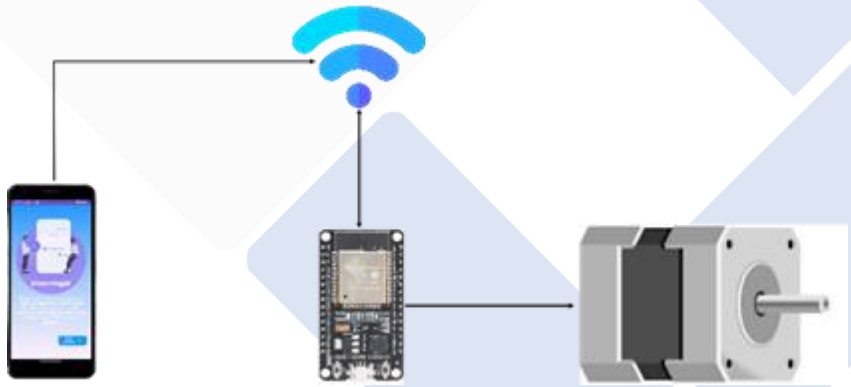
Tabel 4. 7 Konfigurasi Driver TB6600 dan ESP32

Driver tb6600	*ESP32*
Pul+	*Pin* 14 Gpio *ESP32*

Dir+	*Pin* 12 Gpio *ESP32*
En+	*Pin* 13 Gpio *ESP32*
Pul-	*Pin* Gnd *ESP32*
Dir-	*Pin* Gnd *ESP32*
En-	*Pin* Gnd *ESP32*

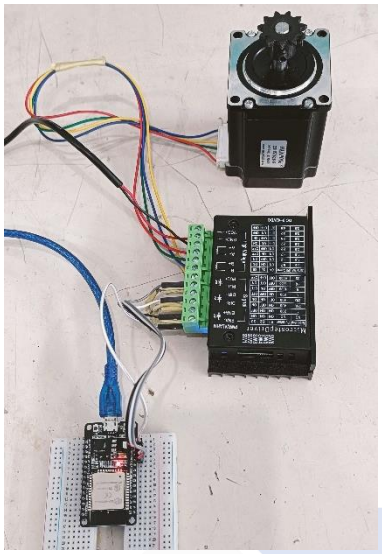
Tabel 4. 8 Konfigurasi Buzzer dan ESP32

Buzzer	*ESP32*
Vcc	*Pin* 27 Gpio *ESP32*
Gnd	*Pin* Gnd *ESP32*



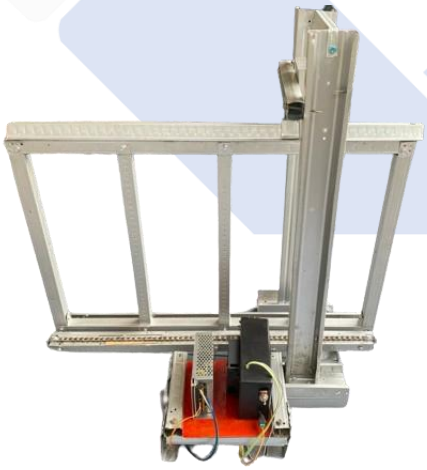
Gambar 4. 11 Skema Rancang Sistem Smart Fence

Gambar diatas merupakan dari skema sistem yang kami buat, dimana ketika pengguna mengakses aplikasi maka sistem akan berkomunikasi dengan *ESP32* menggunakan komunikasi *asynchronous real-time communication*. Yang mana hal itu menyebabkan aplikasi dan *ESP32* dapat terus melakukan operasi atau *get* data tanpa menunggu tanggapan dari *server* firebase, atau dalam bahasa mudahnya tidak perlu menunggu respon dari sistem dalam melakukan *update* data.



Gambar 4.12 Prototipe Mesin Pengendali Sistem *Smart Fence*

Gambar di atas adalah hasil dari prototipe motor penggerak yang akan digunakan dalam penelitian ini. Komponen yang terlihat pada gambar tersebut mencakup *Breadboard*, *Driver* tb6600, kabel *jumper*, Mikrokontroler *ESP32*, dan motor *stepper nema*23. Prototipe ini dibuat selama tahap pengembangan untuk memudahkan implementasi skema dari prototipe tersebut ke dalam PCB setelah motor penggerak dianggap berfungsi dengan baik sesuai kebutuhan.



Gambar 4.13 Prototipe Smart Fence

Gambar di atas menunjukkan hasil dari prototipe *Smart Fence* atau *Smart* Pagar yang kami buat. Komponen yang terdapat pada gambar tersebut melibatkan replika pagar yang dibuat menggunakan baja ringan dengan kombinasi jenis baja ringan

kanal, *hollow*, dan reng. Selain itu, terdapat satu catu daya 12V yang digunakan untuk mensuplai daya ke motor penggerak, serta satu unit motor penggerak yang dilengkapi dengan motor *stepper* yang telah terintegrasi dengan Mikrokontroler *ESP32*.

4.2 Pengujian Sistem

Pada bagian ini, sistem akan melakukan serangkaian pengujian atau *Testing* untuk melihat fungsional sistem dari nilai yang didapatkan dari pengujian jarak hingga *monitoring* pada *database.*

4.2.1 Pengujian Jarak dan Sinyal

Pengujian jarak membantu mengidentifikasi sejauh mana suatu sistem mampu menjaga koneksi dan kinerja pada jarak yang berbeda. Ini penting untuk sistem yang melibatkan komunikasi jarak jauh atau pengguna yang terletak di lokasi yang berbeda.

Tabel 4.9 Pengujian Jarak dan Sinyal

No.	Jarak	*Provider* Kartu	*Delay*	Kekuatan Sinyal	keterangan
1	1m	Wifi	01,35 s	20,43 mb	Buka Pagar
2	1m	Wifi	13,41 s	19,44 mb	Tutup Pagar
3	10 m	Wifi	18,29 s	10.99 mb	Buka Pagar
4	10 m	Wifi	05,38 s	17.46 mb	Tutup Pagar
5	7km	Indosat	03,48 s	69,47 mb	Buka Pagar
6	7km	Indosat	03,66 s	64,66 mb	Tutup pagar
7	7km	Indosat	02,89 s	89.48 mb	Buka pagar
8	7km	Indosat	03,84 s	76.88 mb	Tutup pagar

Dari data yang disapatkan maka *user* dapat menghitung hasil rata – rata respon serta tingkat keberhasilan yang dapat dituliskan sebagai berikut:

a. $\boldsymbol{Rata - Rata\ Waktu\ Respon} = \frac{Total\ waktu\ respon}{Total\ pengujian}$

Total waktu respon

$= (1{,}35 + 13{,}41 + 18{,}29 + 5{,}38 + 3{,}48 + 3{,}66 + 2{,}89 + 3{,}84)$

Banyaknya berhasil $= 8$

Rata-rata waktu respon $= 6{,}16$ s

b. $\boldsymbol{Rata - Rata\ Kecepatan\ Transmisi} = \frac{Total\ waktutransmisi}{Total\ pengujian}$

Rata-Rata Kecepatan Transmisi

$= \frac{(20{,}43 + 19{,}44 + 15{,}99 + 17{,}46 + 69{,}47 + 64{,}66 + 89{,}48 + 76{,}88)}{8} Mbps$

$= 49{,}50$ Mbps

c. $\boldsymbol{Persentase\ Berhasil\ Buka} = \frac{Total\ keberhasilan\ buka\ pagar}{Total\ pengujian\ buka\ berhasil}$

$= \left(\frac{3}{4}\right) x100\% = 75\%$

d. $\boldsymbol{Persentase\ Berhasil\ Tutup} = \frac{Total\ keberhasilan\ tutup\ pagar}{Total\ pengujian\ tutup\ berhasil}$

$= \left(\frac{3}{4}\right) x100\% = 75\%$

4.2.2 Pengujian *Voice Control*

Pengenenalan suara atau *Voice Control* pada sistem *Smart Fence* ini melibatkan kemampuan sistem dalam memahami dan mengenali perintah yang diucapkan oleh *user*. Pengujian ini melibatkan beberapa kata yang berfungsi sebagai *trigger* atau pemicu dalam menjalankan perintah.

Tabel 4.10 Pengujian *Voice Control*/Suara

No.	Kalimat yang diucapkan	Hasil pengenalan dari sistem	Waktu	Hasil
1	Buka	Buka	03,48 s	Berhasil
2	Buka Pagar	Buka Pagar	13,41 s	Berhasil
3	Tutup	Tutup	03,66 s	Berhasil
4	Tutup Pagar	Tutup Pagar	18,29 s	Berhasil
5	Open	-	-	Gagal
6	Close	-	-	Gagal

a. $\boldsymbol{Rata - Rata\ Waktu\ Respon} = \frac{Total\ waktu\ respon}{Total\ pengujian}$

Total waktu respon

$= (3{,}48 + 13{,}41 + 3{,}66 + 18{,}29)$

Banyaknya berhasil $= 4$

Rata-rata waktu respon $= 9,71$ s

b. $\boldsymbol{Persentase\ Berhasil} = \frac{Total\ keberhasilan}{Total\ pengujian}$

$= \left(\frac{4}{6}\right) x100\% = 66,67\%$

4.2.3 *Monitoring*

Monitoring memberikan gambaran terkait seberapa cepat sistem merespon terkait permintaan pengguna dan akan keberhasilan dalam menjalankan perintah yang diminta oleh pengguna.

Tabel 4.11 Pengujian *Monitoring* Sistem

No.	Opsi Operasi	Perintah *User*	Respon sistem	Waktu Respon
1	*Button* Aplikasi	Buka	Berhasil	03,48 s
2	*Button* Aplikasi	Tutup	Berhasil	03,66 s
3	*Button Mesin*	Klik (hidup/mati)	Berhasil	00,37 s
4	Suara (Indonesia)	Buka	Berhasil	03,48 s
5	Suara (Indonesia)	Tutup	Berhasil	03,66 s
6	Suara (Inggris)	*Open*	Gagal	-
7	Suara (Inggris)	*Close*	Gagal	-

a. $\boldsymbol{Rata - Rata\ Waktu\ Respon} = \frac{Total\ waktu\ respon}{Total\ pengujian\ berhasil}$

Total waktu respon

$= (3,48 + 3,66 + 0,37 + 3,48 + 3,66)$

Banyaknya berhasil $= 5$

Rata-rata waktu respon = 2,13 s

b. $\boldsymbol{Presentase\ Keberhasilan} = \frac{Jumlah\ Keberhasilan}{Total\ Pengujian} x\ 100\%$

$= \left(\frac{5}{7}\right) x100\% = 71{,}43\%$

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Penelitian "Perancangan Aplikasi *Smart Fence* Menggunakan *Visual Studio Code* Berbasis IoT" berhasil mengimplementasikan sistem keamanan pagar *pin*tar berbasis IoT dengan baik. Aplikasi *Smart Fence* dapat diintegrasikan dengan sistem kontrol Mikrokontroler, termasuk *NodeMCU*, *ESP32*, dan motor *stepper*. Analisis menunjukkan komunikasi dua arah antara aplikasi *Android* dan Mikrokontroler, memungkinkan pengendalian dan pemantauan pagar dari jarak 1 *meter* hingga 7 kilometer. Rata-rata waktu respons sistem adalah 6,16 detik, dengan kecepatan transmisi 49,50 Mbps. Sistem memerlukan koneksi *internet*, dan kendala pada salah satu sistem dapat memperlambat proses pembukaan pagar. Pengujian Voice Control mencapai tingkat keberhasilan 66,67%, sedangkan pembukaan dan penutupan *Smart Fence* memiliki tingkat keberhasilan 71,43% hingga 75%, menunjukkan bahwa sistem berjalan sesuai dengan ekspektasi.

5.2 Saran

Dalam pembuatan serta uji coba sistem *Smart Fence* ini, penulis menyadari akan beberapa keterbatasan berdasarkan hasil dari pengembangan aplikasi ini. Penulis berharap agar sistem dapat dikembangkan lebih lanjut dengan didasarkan dari beberapa aspek pengembangan berikut :

a. Implementasi Bahasa Sistem

   Sistem saat ini hanya mampu menggunakan bahasa Indonesia sebagai mahasa *default* dari sistem. Diharapkan implementasi bahasa akan bervariasi kedepannya.

b. Pengenalan Suara

Sistem mampu mendeteksi kalimat atau kata dari suara *user*, namun tidak dapat membedakan atau mengambil karakteristik suara dari berbagai macam *user*.

c. Pengembangan *Database*

Database pada sistem aplikasi belum dikembangkan secara menyeluruh oleh peneliti dan diharapkan sistem mampu memberikan *monitoring* penuh yang dimana dapat menampilkan berbagai fitur lainnya, tidak hanya melihat kondisi terkunci pagar.

DAFTAR PUSTAKA

[1] Y. Efendi, "*INTERNET OF THINGS* (IOT) SISTEM PENGENDALIAN LAMPU MENGGUNAKAN RASPBERRY PI BERBASIS MOBILE," *Jurnal Ilmiah Ilmu Komputer*, vol. 4, no. 1, 2018, [Online]. Available: http://ejournal.fikom-unasman.ac.id

[2] S. Arafat, M. Kom, and Kom, "SISTEM PENGAMANAN *PIN*TU RUMAH BERBASIS *Internet of Things* (IoT) Dengan *ESP32*," Oktober-Desember, 2016.

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "*Internet of Things* (IoT): A Vision, Architectural Elements, and Future Directions." [Online]. Available: www.buyya.com

[4] A. Prihanto and A. Prapanca, "*Smart* Automatic Sliding Gate Dengan Memanfaatkan Teknologi Berbasis *Internet of Things* (IoT)."

[5] C. Widiasari, P. Abram Sianipar, M. Diono, P. Caltex Riau, and T. Rekayasa Jaringan Telekomunikasi, "Sistem Kontrol Otomatis Pagar Rumah Berbasis *Internet of Things* (IoT)," 2022. [Online]. Available: https://jurnal.pcr.ac.id/index.php/elementer

[6] A. Ihsan Fadhilla, A. Zain, S. Handani, and P. studi Teknik Elektro Sekolah Tinggi Teknologi Bontang, "Sistem Pengontrol *Pin*tu Pagar Dengan Voice Control Berbasis Mikrokontroler," vol. 16, no. 2, pp. 43–49.

[7] A. B. Aldiansyah, M. Hakimah, and D. T. Tukadi, "Sistem *Monitoring* dan Kontrol Rumah Berbasis *Internet of Things* (IoT)."

[8] G. Mahalisa, M. Dedy Rosyadi, and M. Arsyad Al Banjari, "PERANCANGAN SISTEM *PIN*TU GERBANG OTOMATIS BERBASIS *NODEMCU* MENGGUNAKAN SMS GATEWAY."

[9] N. Adi Santoso, “Exact Papers in Compilation GARASI RUMAH BERBASIS *INTERNET OF THINGS*.”

[10] Y. Praharto, “Intuisi Teknik dan Seni 40 | I t e k s V o l 1 2 N o 2 Inovasi Kendali *Pin*tu Berbasis *Internet* of Thing (IoT) Menggunakan Aplikasi Raspbrry Pi-3 Dengan Proteksi Fitur Image (Ionic) *Internet* of Thing (IoT) Based Door Control Innovation Using Raspbrry Pi-3 Application With Image Feature Protection (Ionic),” vol. 12, no. 2, 2020.

[11] D. P. P. R. K. Indra Aditya Fajar1, “Perancangan *Smart* Home(*Smart* Room) Menggunakan Mikrokontroler Arduino Uno”.

[12] D. P. P. R. K. Indra Aditya Fajar1, “Perancangan *Smart* Home(*Smart* Room) Menggunakan Mikrokontroler Arduino Uno”.

[13] Ummy Gusti Salamah, S.ST,. MIT. (2021). Tutorial *Visual Studio Code*. Media Sains Indonesia. Bandung (ISBN : 978-623-6068-74-8)

[14] Suparman, M., Rosada, M., Lutfi, M., Kamaliya, P., Sabaniah, F., Alfaro, I., & Rosdiana, M. (2023). Mengenal aplikasi *figma* untuk membuat content menjadi lebih interaktif di era society 5.0. *Abdi Jurnal Publikasi*, *1*(6), 552-555.]

[15] Sibarani, N. S., Munawar, G., & Wisnuadhi, B. (2018, July). Analisis performa aplikasi android pada bahasa pemrograman java dan *kotlin*. In *Prosiding Industrial Research Workshop and National Seminar* (Vol. 9, pp. 319-324).]

[16] S. Sawidin *et al.*, “Prosiding The 12 th Industrial Research Workshop and National Seminar Bandung,” 2021. [Online]. Available: www.arduino.cc

[17] S. Venkatraman, A. Overmars, and M. Thong, “*Smart* home automation—use cases of a secure and integrated voice-control system,” *Systems*, vol. 9, no. 4, Dec. 2021, doi: 10.3390/systems9040077.

[18] A. Ali, A. Taifour Ali, E. B. M Eltayeb, and E. Altigani Ahmed Abusail, "*Voice Control* Based *Smart* Home Control System Model-free fractional-order sliding mode control for an active vehicle suspension system View project *Voice Control* Based *Smart* Home Control System," vol. 6, no. 4, pp. 1–05, 2017, [Online]. Available: www.ijeijournal.com

[19] N. Hikmah, A. Suradika, and R. A. Ahmad Gunadi, "Metode Agile Untuk Meningkatkan Kreativitas Guru Melalui Berbagi Pengetahuan (Knowledge Sharing) (Studi Kasus: Sdn Cipulir 03 Kebayoran Lama, Jakarta)," Instruksional, vol. 3, no. 1, p. 30, Oct. 2021, doi: 10.24853/instruksional.3.1.30-39.

[20] S. Venkatraman, A. Overmars, and M. Thong, "*Smart* home automation—use cases of a secure and integrated voice-control system," Systems, vol. 9, no. 4, Dec. 2021, doi: 10.3390/systems9040077.

[21] A. Ali, A. Taifour Ali, E. B. M Eltayeb, and E. Altigani Ahmed Abusail, "*Voice Control* Based *Smart* Home Control System Model-free fractional-order sliding mode control for an active vehicle suspension system View project *Voice Control* Based *Smart* Home Control System," vol. 6, no. 4, pp. 1–05, 2017, [Online]. Available: www.ijeijournal.com

[22] Setiawan, Y., Tanudjaja, H., & Octaviani, S. (2018). Penggunaan *Internet of Things* (IoT) untuk Pemantauan dan Pengendalian Sistem Hidroponik. TESLA: Jurnal Teknik Elektro, 20(2), 175-182.

[23] AGUS, R. M. (2021). IMPLEMENTASI SISTEM KEAMANAN KENDALI PAGAR GESER OTOMATIS PADA RUMAH BERBASIS PLC (PROGRAMMABLE LOGIC CONTROLLER) DENGAN KOMUNIKASI FIREBASE MENGGUNAKAN APLIKASI ANDROID (Doctoral dissertation, Universitas Islam Negeri Sultan Syarif Kasim Riau).

LAMPIRAN

Lampiran 1: Daftar Riwayat Hidup

Data Pribadi

Nama	: Hasya Akhlaqah Harmie
NIM	: 1062011
Tempat & Tanggal Lahir	: Belinyu, 21 Desember 2001
Alamat	: Jln. Rempuding No. 25, RT.8, Parit Padang, Sungailiat, Kab. Bangka, Bangka Belitung
No. Hp	: 085766955967
Email	: hasyaharmie@gmail.com
Jenis Kelamin	: Perempuan
Agama	: Islam

Riwayat Pendidikan

a. SD Muhammadiyah Sungailiat :2008-2014
b. SMPN2 Sungailiat :2014-2017
c. SMAN 1 Sungailiat :2017-2020

Sungailiat, 01 Januari 2024

Hasya Akhlaqah Harmie

Data Pribadi

Nama	: Rahmattian Diza Dwi Putra
NIM	: 1062023
Tempat & Tanggal Lahir	: Toboali, 24 Oktober 2002
Alamat	: Jln. Amd Prumnas Guru Toboali
No. Hp	: 081280791622
Email	: rahmatiandiza24@gmail.com
Jenis Kelamin	: Laki-laki
Agama	: Islam

Riwayat Pendidikan

a. SDN 16 Toboali : 2008-2014
b. SMPN 5 Toboali : 2014-2017
c. SMKN 1 Toboali : 2017-2020

Sungailiat, 01 Januari 2024

Rahmattian Diza Dwi Putra

Lampiran 2: Program

Koding get started activity

```
package com.example.smartpagar
import android.content.Context
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.widget.Button
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.ValueEventListener

class GetStartedActivity : AppCompatActivity {
    private lateinit var firebaseDatabase: FirebaseDatabase
    private lateinit var databaseReference: DatabaseReference

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.get_started_activity)

        databaseReference =
FirebaseDatabase.getInstance.reference.child("users")
        val getDataUserLogin = getSharedPreferences("PrefsUserLogin",
Context.MODE_PRIVATE)
        val userLogin = getDataUserLogin.getString("key_userLogin", "")
        var role = ""
        val getStartedButton = findViewById<Button>(R.id.getStartedButton)

        // Listen for changes in the "nama_node" and update UI accordingly
        databaseReference.orderByChild("username").equalTo(userLogin)
            .addListenerForSingleValueEvent(object :
                ValueEventListener {
                override fun onDataChange(dataSnapshot: DataSnapshot) {
                    // Check if there are any matching results
                    if (dataSnapshot.exists) {
                        // Iterate through the dataSnapshot to check each result
                        for (snapshot in dataSnapshot.children) {
                            role =
```

```
snapshot.child("role").getValue(String::class.java).toString

                    // Only process the first result (if there are multiple matches)
                    break
                }

                // Navigasi setelah mendapatkan nilai dari Firebase
                navigateToNextActivity(userLogin, role)
            } else {
                // Handle the case when no matching data is found
                Log.d("Firebase", "No matching data found for username: $userLogin")

                // Navigasi jika tidak ada data yang cocok
                navigateToNextActivity(userLogin, role)
            }
        }

        override fun onCancelled(databaseError: DatabaseError) {
            // Handle errors if they occur
            // For example, log the error
            Log.e("Firebase", "Error: ${databaseError.message}")

            // Navigasi jika terjadi kesalahan
            navigateToNextActivity(userLogin, role)
        }
    })

    // Remove navigasi langsung ke aktivitas lain dari sini
    getStartedButton.setOnClickListener{
        val intent = Intent(this, LoginOptionActivity::class.java)
        startActivity(intent)
        finish
    }
}

private fun navigateToNextActivity(userLogin: String?, role: String) {
    if (userLogin != null) {
        if (userLogin.isNotEmpty && role == "super") {
            val intent = Intent(this, SuperUserMainActivity::class.java)
            startActivity(intent)
            finish
        } else if (userLogin.isNotEmpty && role == "normal") {
```

```
            val intent = Intent(this, MainActivity::class.java)
            startActivity(intent)
            finish
        }
    }
    else if(userLogin == null){
        val intent = Intent(this, GetStartedActivity::class.java)
        startActivity(intent)
    }
  }
}
```

koding *login* option activity

```
package com.example.smartpagar
import android.content.Intent
import android.os.Build
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.Button

class LoginOptionActivity : AppCompatActivity {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.login_option_activity)

      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP)
{
         window.decorView.systemUiVisibility =
            View.SYSTEM_UI_FLAG_LAYOUT_STABLE or
View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
         window.statUSBarColor = android.graphics.Color.TRANSPARENT
      }

      val btnGoToLoginActivity: Button = findViewById(R.id.login_opsi)
      val btnGoToSignActivity: Button = findViewById(R.id.register_opsi)

      btnGoToLoginActivity.setOnClickListener {
         val intent = Intent(this, LoginActivity::class.java)
```

```
            startActivity(intent)
        }

        btnGoToSignActivity.setOnClickListener {
            val intent = Intent(this, SigninActivity::class.java)
            startActivity(intent)
        }
    }
}
```

koding sign in activity

```
class SigninActivity : AppCompatActivity {
    private lateinit var binding: SigninActivityBinding
    private lateinit var firebaseDatabase: FirebaseDatabase
    private lateinit var databaseReference: DatabaseReference
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.signin_activity)
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP)
{
            window.decorView.systemUiVisibility =
                View.SYSTEM_UI_FLAG_LAYOUT_STABLE or
View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
            window.statUSBarColor = android.graphics.Color.TRANSPARENT
        }
        binding = SigninActivityBinding.inflate(layoutInflater)
        setContentView(binding.root)
        firebaseDatabase = FirebaseDatabase.getInstance
        databaseReference = firebaseDatabase.reference.child("users")
        //request fokus
        val emailFokus = findViewById<EditText>(R.id.inputSigninemailEdit)
        val userFokus = findViewById<EditText>(R.id.inputSigninUserEdit)
        val passwordFokus =
findViewById<EditText>(R.id.inputSigninPasswordEdit)
        binding.signButton.setOnClickListener{
            val inputSignemailEdit = binding.inputSigninemailEdit.text.toString
            val inputSignUserEdit = binding.inputSigninUserEdit.text.toString
            val inputSignPasswordEdit =
binding.inputSigninPasswordEdit.text.toString

            if(inputSignemailEdit.isNotEmpty && inputSignUserEdit.isNotEmpty
```

```
&& inputSignPasswordEdit.isNotEmpty)
        {
            val sharedPreferences = getSharedPreferences("PrefsUserSignIn",
Context.MODE_PRIVATE)
            val editor = sharedPreferences.edit
            editor.putString("key_userSignIn", inputSignUserEdit)
            editor.apply
            signupUser(inputSignemailEdit, inputSignUserEdit,
inputSignPasswordEdit)
        }
        else if (inputSignemailEdit.isNullOrEmpty)
        {
            emailFokus.error = "wajib diisi!"
            emailFokus.requestFocus
        }
        else if (inputSignUserEdit.isNullOrEmpty)
        {
            userFokus.error = "wajib diisi!"
            userFokus.requestFocus
        }
        else if (inputSignPasswordEdit.isNullOrEmpty)
        {
            passwordFokus.error = "wajib diisi!"
            passwordFokus.requestFocus
        }
    })

    showGif

  }
  private fun signupUser(email: String, username: String, password: String) {
    val hashedPassword = PasswordUtils.hashPassword(password)

databaseReference.orderByChild("username").equalTo(username).addListenerF
orSingleValueEvent(object : ValueEventListener {
        override fun onDataChange(dataSnapshot: DataSnapshot) {
            if (!dataSnapshot.exists) {
                val id = databaseReference.push.key
                val userData = UserData(id, email, username, hashedPassword) //
Gunakan hashedPassword
                databaseReference.child(id!!).setValue(userData)
                Toast.makeText(this@SigninActivity, "signup berhasil",
Toast.LENGTH_SHORT).show
```

```
                startActivity(Intent(this@SigninActivity,
LoginActivity::class.java))
                finish
            } else {
                val sharedPreferences = getSharedPreferences("PrefsUserSignIn",
Context.MODE_PRIVATE)
                val editor = sharedPreferences.edit
                editor.clear
                editor.apply
                showPupUpAlert
            }
        }

        override fun onCancelled(databaseError: DatabaseError) {
            Toast.makeText(this@SigninActivity, "Database Error:
${databaseError.message}", Toast.LENGTH_SHORT).show
        }
    })
  }

  fun showGif
  {
    val imgView: ImageView = findViewById(R.id.keyLoginImg)
    Glide.with(this).load(R.drawable.key_icon).into(imgView)
  }
  private  fun showPupUpAlert
  {
    val builder = AlertDialog.Builder(this)
    builder.setIcon(R.drawable.error_popup)
    builder.setTitle("Warning!!!")
    builder.setMessage("Opssss, Sepertinya username sudah terdaftar")
    builder.setPositiveButton("oke") {
        dialogInterface: DialogInterface, _: Int -> dialogInterface.dismiss
    }
    val dialog: AlertDialog = builder.create
    dialog.show
  }
}
```

koding *login* activity

```
package com.example.smartpagar

import android.content.Context
import android.content.DialogInterface
import android.content.Intent
import android.graphics.drawable.AnimationDrawable
import android.os.Build
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.EditText
import android.widget.ImageView
import android.widget.RelativeLayout
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.constraintlayout.widget.ConstraintLayout
import com.bumptech.glide.Glide
import com.example.smartpagar.databinding.LoginActivityBinding
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.ValueEventListener

class LoginActivity : AppCompatActivity {

    private lateinit var binding: LoginActivityBinding
    private lateinit var firebaseDatabase: FirebaseDatabase
    private lateinit var databaseReference: DatabaseReference

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = LoginActivityBinding.inflate(layoutInflater)
        setContentView(binding.root)

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP)
{
            window.decorView.systemUiVisibility =
                View.SYSTEM_UI_FLAG_LAYOUT_STABLE or
View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
            window.statUSBarColor = android.graphics.Color.TRANSPARENT
        }
```

```
        firebaseDatabase = FirebaseDatabase.getInstance
        databaseReference = firebaseDatabase.reference.child("users")

        //request fokus
        val usernameFokus =
findViewById<EditText>(R.id.inputLoginUserNameEdit)
        val passwordFokus =
findViewById<EditText>(R.id.inputLoginUserPasswordEdit)

        binding.loginButton.setOnClickListener{
            val inputLoginUserEdit =
binding.inputLoginUserNameEdit.text.toString
            val inputLoginPasswordEdit =
binding.inputLoginUserPasswordEdit.text.toString
            if(inputLoginUserEdit.isNullOrEmpty)
            {
                usernameFokus.error = "wajib diisi!"
                usernameFokus.requestFocus

            }
            else if (inputLoginPasswordEdit.isNullOrEmpty)
            {
                passwordFokus.error = "wajib diisi!"
                passwordFokus.requestFocus
            }
            else if(inputLoginUserEdit.isNotEmpty &&
inputLoginPasswordEdit.isNotEmpty)
            {
                val sharedPreferences = getSharedPreferences("PrefsUserLogin",
Context.MODE_PRIVATE)
                val editor = sharedPreferences.edit
                val hashedPassword =
PasswordUtils.hashPassword(inputLoginPasswordEdit)
                editor.putString("key_userLogin", inputLoginUserEdit)
                editor.apply
                loginUser(inputLoginUserEdit, hashedPassword)
            }
        }
        showGif
    }

    private fun loginUser(username: String, password: String)
    {
```

```
databaseReference.orderByChild("username").equalTo(username).addListenerF
orSingleValueEvent(object :ValueEventListener {
        override fun onDataChange(dataSnapshot: DataSnapshot) {
            if(dataSnapshot.exists)
            {
                for (userSnapshot in dataSnapshot.children)
                {
                    val userData = userSnapshot.getValue(UserData::class.java)
                    if (userData != null && userData.password == password &&
userData.role == "normal")
                    {
                        Toast.makeText(this@LoginActivity, "login berhasil",
Toast.LENGTH_SHORT).show
                        startActivity(Intent(this@LoginActivity,
MainActivity::class.java))
                        finish
                        return
                    }
                    else if (userData != null && userData.password == password
&& userData.role == "super")
                    {
                        Toast.makeText(this@LoginActivity, "login berhasil",
Toast.LENGTH_SHORT).show
                        startActivity(Intent(this@LoginActivity,
SuperUserMainActivity::class.java))
                        finish
                        return
                    }
                }
            }
//              Toast.makeText(this@LoginActivity, "login gagal username /
password salah", Toast.LENGTH_SHORT).show
            val sharedPreferences = getSharedPreferences("PrefsUserLogin",
Context.MODE_PRIVATE)
            val editor = sharedPreferences.edit
            editor.clear
            editor.apply
            showPupUpAlert
        }

        override fun onCancelled(databaseError: DatabaseError) {
            Toast.makeText(this@LoginActivity, "Database Error:
${databaseError.message}", Toast.LENGTH_SHORT).show
```

```
            }
        })
    }

    fun showGif
    {
        val imgView: ImageView = findViewById(R.id.keyLoginImg)
        Glide.with(this).load(R.drawable.key_icon).into(imgView)
    }
    private  fun showPupUpAlert
    {
        val builder = AlertDialog.Builder(this)
        builder.setIcon(R.drawable.error_popup)
        builder.setTitle("Warning!!!")
        builder.setMessage("Opssss, Sepertinya username / password yang kamu
masukan salah!")
        builder.setPositiveButton("oke") {
            dialoginterface: Dialoginterface, _: Int -> dialoginterface.dismiss
        }
        val dialog: AlertDialog = builder.create
        dialog.show
    }
}
```

koding main activity

```
package com.example.smartpagar

import android.content.ClipData.Item
import android.content.Context
import android.content.Dialoginterface
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.MenuItem
import android.widget.Toast
import androidx.appcompat.app.ActionBarDrawerToggle
//import android.widget.Toolbar
import androidx.drawerlayout.widget.DrawerLayout
import androidx.appcompat.widget.Toolbar
import androidx.core.view.GravityCompat
```

```
import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentTransaction
import com.google.android.material.navigation.NavigationView
import android.content.Intent
import android.os.Handler
import android.widget.ImageView
import android.widget.TextView
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatDelegate
import androidx.core.content.ContextCompat
import com.google.android.material.switchmaterial.SwitchMaterial
import kotlin.system.exitProcess

class MainActivity : AppCompatActivity(),
NavigationView.OnNavigationItemSelectedListener {
    private lateinit var drawerLayout: DrawerLayout
    private lateinit var btntoggle: SwitchMaterial
    private var doubleBackToExitPressedOnce = false
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val getDataUserSignin = getSharedPreferences("PrefsUserSignIn",
Context.MODE_PRIVATE)
        val nameUserSignIn = getDataUserSignin.getString("key_userSignIn", "")
        val getDataUserLogin = getSharedPreferences("PrefsUserLogin",
Context.MODE_PRIVATE)
        val nameUserLogin = getDataUserLogin.getString("key_userLogin", "")
        drawerLayout = findViewById<DrawerLayout>(R.id.drawer_layout)
        val toolbar = findViewById<Toolbar>(R.id.toolbar)
        setSupportActionBar(toolbar)
        val navigationView = findViewById<NavigationView>(R.id.nav_view)
        val headerView = navigationView.getHeaderView(0)
        val navUser = headerView.findViewById<TextView>(R.id.nav_user)
        val profileUser =
headerView.findViewById<ImageView>(R.id.profile_user)
        navigationView.setNavigationItemSelectedListener(this)

        if (getDataUserSignin == getDataUserLogin)
        {
            navUser.text = nameUserSignIn
        }
        else if (getDataUserSignin != getDataUserLogin)
        {
```

```
            navUser.text = nameUserLogin
        }

        profileUser.setOnClickListener{
            val intent = Intent(this, ProfileDetailActivity::class.java)
            startActivity(intent)
        }

        val toogle = ActionBarDrawerToggle(this, drawerLayout, toolbar,
R.string.open_nav, R.string.close_nav)
        drawerLayout.addDrawerListener(toogle)
        toogle.syncState

        if (savedInstanceState == null){
            replacementFragment(PagarFragment)
            navigationView.setCheckeditem(R.id.nav_pagar)
        }
    }
    override fun onNavigationItemSelected(item: MenuItem): Boolean {
        when(item.itemId){
            R.id.nav_pagar -> replacementFragment(PagarFragment)
            R.id.nav_monitoring -> replacementFragment(MonitoringFragmen)
            R.id.nav_developer -> replacementFragment(DeveloperFragmen)
            R.id.nav_Logout -> LogoutAlert
        }
        drawerLayout.closeDrawer(GravityCompat.START)
        return true
    }
    private fun LogoutAlert {
        val builder = AlertDialog.Builder(this)
        builder.setIcon(R.drawable.error_popup)
        builder.setTitle("Konfirmasi Logout")
        builder.setMessage("Apakah Anda yakin ingin keluar?")

        // Tombol "Ya"
        builder.setPositiveButton("Ya") { _, _ ->
            // Aksi yang akan diambil jika "Ya" ditekan, misalnya, melakukan
Logout
            Logout
        }

        // Tombol "Tidak"
        builder.setNegativeButton("Tidak") { dialogInterface, _ ->
```

```
            dialogInterface.dismiss
        }

        val dialog: AlertDialog = builder.create
        dialog.setOnShowListener {
            dialog.getButton(AlertDialog.BUTTON_POSITIVE)?.setTextColor(
                ContextCompat.getColor(this, R.color.red)
            )

            dialog.getButton(AlertDialog.BUTTON_NEGATIVE)?.setTextColor(
                ContextCompat.getColor(this, R.color.purple)
            )
        }

        dialog.show
    }
    private fun Logout {
        // Navigasi ke halaman login tanpa menggunakan NavController
        val loginIntent = Intent(this, LoginOptionActivity::class.java)
        loginIntent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK
        startActivity(loginIntent)
        val getDataUserLogin = getSharedPreferences("PrefsUserLogin",
Context.MODE_PRIVATE)

        val editorLogin = getDataUserLogin.edit
        editorLogin.clear
        editorLogin.apply

        finish
    }
    private fun replacementFragment(fragment: Fragment){
        val transaction: FragmentTransaction =
supportFragmentManager.beginTransaction
        transaction.replace(R.id.frame_container, fragment)
        transaction.commit
    }
    override fun onBackPressed {
        if (drawerLayout.isDrawerOpen(GravityCompat.START)){
            // Jika drawer terbuka, tutup drawer
            drawerLayout.closeDrawer(GravityCompat.START)
        } else {
            // Jika drawer tertutup
```

```
        if (doubleBackToExitPressedOnce) {
            // Jika tombol kembali ditekan dua kali, keluar dari aplikasi
            finishAffinity
            super.onBackPressed
        } else {
            // Tampilkan pesan atau lakukan tindakan yang sesuai
            Toast.makeText(this, "Tekan sekali lagi untuk kembali",
Toast.LENGTH_SHORT).show

            // Setel variabel doubleBackToExitPressedOnce menjadi true
            doubleBackToExitPressedOnce = true

            // Set timer untuk mereset status doubleBackToExitPressedOnce
setelah batas waktu tertentu
            Handler.postDelayed({
                doubleBackToExitPressedOnce = false
            }, 2000)
        }
    }
  }
}
```

koding pagar fragmen

```
package com.example.smartpagar

import android.app.Activity
import android.app.AlertDialog
import android.app.Dialog
import android.content.ActivityNotFoundException
import android.content.Intent
import android.graphics.Color
import android.graphics.drawable.ColorDrawable
import android.os.Bundle
import android.provider.ContactsContract.CommonDataKinds.Email
import android.speech.RecognitionListener
import android.speech.RecognizerIntent
import android.speech.SpeechRecognizer
import android.util.Log
import android.view.LayoutInflater
```

```
import android.view.View
import android.view.ViewGroup
import android.view.Window
import android.widget.Button
import android.widget.PopupWindow
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatDelegate
import androidx.core.app.ActivityCompat.recreate
import androidx.fragment.app.Fragment
import com.google.android.material.switchmaterial.SwitchMaterial
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase
import java.util.*

class PagarFragment : Fragment {

    private lateinit var speechRecognizer: SpeechRecognizer
    private lateinit var recognizerIntent: Intent
    private lateinit var resultTextView: TextView
    private lateinit var database: DatabaseReference

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.pagar_fragmen, container, false)

        resultTextView = view.findViewById(R.id.output)
        val startButton: Button = view.findViewById(R.id.voiceFenceButton)


        speechRecognizer =
SpeechRecognizer.createSpeechRecognizer(requireContext)
        speechRecognizer.setRecognitionListener(SpeechRecognitionListener)

        recognizerIntent =
Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH)
        recognizerIntent.putExtra(
            RecognizerIntent.EXTRA_LANGUAGE_MODEL,
            RecognizerIntent.LANGUAGE_MODEL_FREE_FORM
        )
        recognizerIntent.putExtra(
```

```
        RecognizerIntent.EXTRA_LANGUAGE,
        Locale.getDefault
    )

    startButton.setOnClickListener {
        startSpeechToText
    }

    database = FirebaseDatabase.getInstance.reference

    val id = "smartpagar"
    val pagaropsiValue = "buka pagar"
    val pagaropsiValue2 = "tutup pagar"

    val userReference = database.child("pagar").child(id)

    val bukapagar = mapOf("pagaropsi" to pagaropsiValue)
    val tutuppagar = mapOf("pagaropsi" to pagaropsiValue2)


    // Mendapatkan referensi ke tombol
    val bukaButton = view.findViewById<Button>(R.id.openFenceButton)
    val tutupButton = view.findViewById<Button>(R.id.closeFenceButton)

    // Menambahkan fungsi penanganan klik pada tombol
    bukaButton.setOnClickListener {
        // Melakukan update data
        userReference.updateChildren(bukapagar)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    // Data berhasil diupdate
                    // Tambahkan logika atau tindakan lain yang diinginkan
                    // Misalnya, menampilkan pesan sukses
                    Toast.makeText(requireContext, "pagar terbuka",
Toast.LENGTH_SHORT).show
                } else {
                    // Gagal mengupdate data
                    // Tambahkan penanganan kesalahan
                    // Misalnya, menampilkan pesan gagal
                    Toast.makeText(requireContext, "oops ada kesalahan",
Toast.LENGTH_SHORT)
                        .show
                }
```

```
            }
        }

        tutupButton.setOnClickListener {
            // Melakukan update data
            userReference.updateChildren(tutuppagar)
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        // Data berhasil diupdate
                        // Tambahkan logika atau tindakan lain yang diinginkan
                        // Misalnya, menampilkan pesan sukses
                        Toast.makeText(requireContext, "pagar tertutup",
Toast.LENGTH_SHORT)
                            .show
                    } else {
                        // Gagal mengupdate data
                        // Tambahkan penanganan kesalahan
                        // Misalnya, menampilkan pesan gagal
                        Toast.makeText(requireContext, "oops ada kesalahan",
Toast.LENGTH_SHORT)
                            .show
                    }
                }
        }

        return view
    }

    private fun startSpeechToText {
        try {
            startActivityForResult(recognizerIntent, 100)
        } catch (e: ActivityNotFoundException) {
            // Handle the exception where speech recognition is not available on the
device
            e.printStackTrace
        }
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == 100 && resultCode == Activity.RESULT_OK) {
```

```
        val result =
data?.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS)
        result?.let {
            val text = it[0]
//              resultTextView.text = text

            val id = "smartpagar"
            val pagaropsiValue = text

            val userReference = database.child("pagar").child(id)

            val bukapagar = mapOf("pagaropsi" to pagaropsiValue)

            if (text == "buka pagar"){
                userReference.updateChildren(bukapagar)
                    .addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            // Data berhasil diupdate
                            // Tambahkan logika atau tindakan lain yang diinginkan
                            // Misalnya, menampilkan pesan sukses
                            Toast.makeText(requireContext, "pagar terbuka",
Toast.LENGTH_SHORT).show
                        } else {
                            // Gagal mengupdate data
                            // Tambahkan penanganan kesalahan
                            // Misalnya, menampilkan pesan gagal
                            Toast.makeText(requireContext, "oops ada kesalahan",
Toast.LENGTH_SHORT)
                                .show
                        }
                    }
            }
            if (text == "tutup pagar"){
                userReference.updateChildren(bukapagar)
                    .addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            // Data berhasil diupdate
                            // Tambahkan logika atau tindakan lain yang diinginkan
                            // Misalnya, menampilkan pesan sukses
                            Toast.makeText(requireContext, "pagar tertutup",
Toast.LENGTH_SHORT).show
                        } else {
                            // Gagal mengupdate data
```

```
                        // Tambahkan penanganan kesalahan
                        // Misalnya, menampilkan pesan gagal
                        Toast.makeText(requireContext, "oops ada kesalahan",
Toast.LENGTH_SHORT)
                            .show
                    }
                }
            }
        }

    }
  }


  private inner class SpeechRecognitionListener : RecognitionListener {
    override fun onReadyForSpeech(params: Bundle?) {}

    override fun onBeginningOfSpeech {}

    override fun onRmsChanged(rmsdB: Float) {}

    override fun onBufferReceived(buffer: ByteArray?) {}

    override fun onEndOfSpeech {}

    override fun onError(error: Int) {}

    override fun onResults(results: Bundle?) {}

    override fun onPartialResults(partialResults: Bundle?) {
    }


    override fun onEvent(eventType: Int, params: Bundle?) {}
  }
}
```

kode *monitoring* activity

```
package com.example.smartpagar

import android.content.Context
import android.os.Bundle
import android.util.Log
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.ValueEventListener

class MonitoringFragmen : Fragment {
    private lateinit var databaseReference: DatabaseReference
    private lateinit var statusPagar: TextView

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        val view = inflater.inflate(R.layout.monitoring_fragment, container, false)

        // Temukan TextView di dalam layout
        val nameTextView = view.findViewById<TextView>(R.id.owner_pagar)
        statusPagar = view.findViewById(R.id.status_pagar)

        // Mendapatkan nilai dari SharedPreferences
        val getNameFromUserSignIn = getDataUserSignIn
        val getNameFromUserLogin = getDataUserLogin

        // Set nilai TextView
        if (getNameFromUserSignIn == getNameFromUserLogin) {
            nameTextView.text = "pagar : $getNameFromUserSignIn"
        } else if (getNameFromUserSignIn != getNameFromUserLogin) {
            nameTextView.text = "pagar : $getNameFromUserLogin"
        }

        return view
```

```
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // Initialize DatabaseReference
        databaseReference = FirebaseDatabase.getInstance.reference

        // Listen for changes in the "nama_node" and update UI accordingly

databaseReference.child("pagar").child("smartpagar").child("pagaropsi").addV
alueEventListener(object : ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                // Get the value from dataSnapshot
                val valuePagar: String? = dataSnapshot.getValue(String::class.java)

                // Do something with the value, for example, display it in a TextView
                if (valuePagar == "buka pagar") {
                    statusPagar.text = "Terbuka"
                } else if (valuePagar == "tutup pagar") {
                    statusPagar.text = "Tertutup"
                }
            }

            override fun onCancelled(databaseError: DatabaseError) {
                // Handle errors if they occur
                // For example, log the error
                Log.e("Firebase", "Error: ${databaseError.message}")
            }
        })
    }

    private fun getDataUserSignIn: String {
        val sharedPreferences =
requireActivity.getSharedPreferences("PrefsUserSignIn",
Context.MODE_PRIVATE)
        // Mengambil nilai dari SharedPreferences
        return sharedPreferences.getString("key_userSignIn", "") ?: ""
    }
    private fun getDataUserLogin: String {
        val sharedPreferences =
requireActivity.getSharedPreferences("PrefsUserLogin",
Context.MODE_PRIVATE)
```

```
        // Mengambil nilai dari SharedPreferences
        return sharedPreferences.getString("key_user_Login", "") ?: ""
    }
}
```

koding profile *edit*

```
package com.example.smartpagar

import android.content.Context
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.widget.Button
import android.widget.EditText
import androidx.appcompat.app.AlertDialog
import androidx.core.content.ContextCompat
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.ValueEventListener

class ProfileEditActivity : AppCompatActivity {
    private lateinit var database: DatabaseReference
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.profile_edit_activity)
        database = FirebaseDatabase.getInstance.reference.child("users")
        val btnEditProfile = findViewById<Button>(R.id.btn_edit_profile)
        //getdatafromuser


        btnEditProfile.setOnClickListener{
            fokusListener
        }
```

```
    }
    private fun editProfileData {
        val getDataUserLogin = getSharedPreferences("PrefsUserLogin",
Context.MODE_PRIVATE)
        val userLogin = getDataUserLogin.getString("key_userLogin", "")
        // Gantilah "path_to_your_node" dengan path ke node yang ingin Anda
edit

        // Gantilah "username_to_update" dengan username yang ingin Anda
update

        // Mendapatkan referensi ke path yang diinginkan



database.orderByChild("username").equalTo(userLogin).addListenerForSingle
ValueEvent(object :
            ValueEventListener {
            val fieldUsername =
findViewById<EditText>(R.id.edit_user_edit).text.toString
            val fieldPassword =
findViewById<EditText>(R.id.edit_password_edit).text.toString
            val hashedPassword = PasswordUtils.hashPassword(fieldPassword)
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                for (snapshot in dataSnapshot.children) {
                    // Update data di dalam loop jika ditemukan
                    val key = snapshot.key
                    val updatedData = hashMapOf(
                        "username" to fieldUsername, // Ganti dengan nama baru yang
diinginkan
                        "password" to hashedPassword // Ganti dengan email baru yang
diinginkan
                        // Tambahkan field lain jika diperlukan
                    )

                    // Melakukan update ke database
                    key?.let {
                        database.child(it).updateChildren(updatedData as Map<String,
Any>)
                            .addOnSuccessListener {
                                Logout
                            }
                            .addOnFailureListener {
```

```
                        // Aksi yang diambil jika update gagal
                        // Contoh: Menampilkan pesan error
//                        showToast("Gagal mengupdate profil")
                    }
                }
            }
        }

        override fun onCancelled(databaseError: DatabaseError) {
            // Handle errors if they occur
            // For example, log the error
            Log.e("Firebase", "Error: ${databaseError.message}")
        }
    })
}
private fun LogoutAlert {
    val builder = AlertDialog.Builder(this)
    builder.setIcon(R.drawable.error_popup)
    builder.setTitle("Konfirmasi Edit Profile")
    builder.setMessage("Setelah edit profile anda harus login kembali")

    // Tombol "Ya"
    builder.setPositiveButton("Ya") { _, _ ->
        // Aksi yang akan diambil jika "Ya" ditekan, misalnya, melakukan
Logout
        editProfileData
    }

    // Tombol "Tidak"
    builder.setNegativeButton("Tidak") { dialoginterface, _ ->
        dialoginterface.dismiss
    }

    // Set text color secara eksplisit untuk memastikan konsistensi tampilan
    val dialog: AlertDialog = builder.create
    dialog.setOnShowListener {
        dialog.getButton(AlertDialog.BUTTON_POSITIVE)?.setTextColor(
            ContextCompat.getColor(this, R.color.red)
        )

        dialog.getButton(AlertDialog.BUTTON_NEGATIVE)?.setTextColor(
            ContextCompat.getColor(this, R.color.purple)
        )
```

```
        }

        dialog.show
    }
    private fun Logout {

        // Navigasi ke halaman login tanpa menggunakan NavController
        val loginIntent = Intent(this, LoginOptionActivity::class.java)
        loginIntent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK
        startActivity(loginIntent)
        val getDataUserLogin = getSharedPreferences("PrefsUserLogin",
Context.MODE_PRIVATE)

        val editorLogin = getDataUserLogin.edit
        editorLogin.clear
        editorLogin.apply

        finish
    }
    private fun fokusListener{
        //get value
        val valueUsername =
findViewById<EditText>(R.id.edit_user_edit).text.toString
        val valuePassword =
findViewById<EditText>(R.id.edit_password_edit).text.toString
        //set fokus
        val usernameFokusListener =
findViewById<EditText>(R.id.edit_user_edit)
        val passwordFokusListener =
findViewById<EditText>(R.id.edit_password_edit)
        if(valueUsername.isNullOrEmpty)
        {
            usernameFokusListener.error = "wajib diisi"
            usernameFokusListener.requestFocus
        }
        else if(valuePassword.isNullOrEmpty)
        {
            passwordFokusListener.error = "wajib diisi"
            passwordFokusListener.requestFocus
        }
        else if(valueUsername.isNotEmpty && valuePassword.isNotEmpty)
        {
```

```
            LogoutAlert
        }
    }
}
```

Lampiran 3: Gambar