BAB 4
HASIL PENELITIAN

4.1 RINGKASAN HASIL PENELITIAN

Hasil dari penelitian ini berupa aplikasi *mobile* yang dirancang untuk mendiagnosis penyakit gigi berdasarkan input gejala dari pengguna. Proses diagnosis dalam aplikasi ini didukung oleh penerapan metode *Forward Chaining* dan *Certainty Factor*. Sistem dirancang untuk membantu pengguna dalam mengenali indikasi awal penyakit gigi secara mandiri, melalui proses input gejala yang kemudian diolah menggunakan basis pengetahuan yang diperoleh dari pakar. Hasil diagnosis disajikan dalam bentuk nama penyakit disertai tingkat keyakinan.

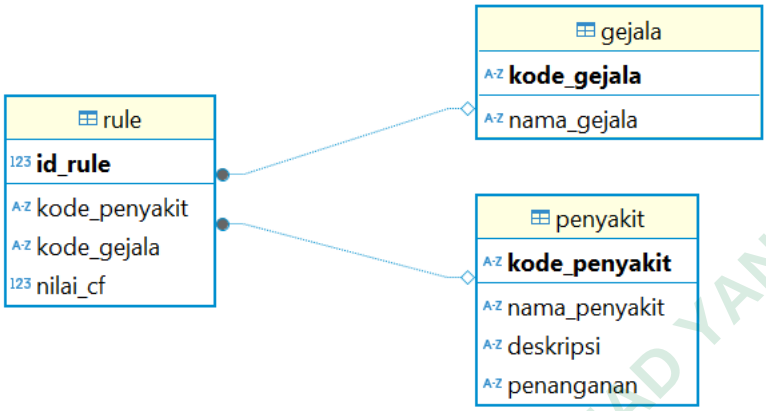
4.2 IMPLEMENTASI BASIS DATA

Implementasi basis data dalam sistem pakar diagnosis penyakit gigi ini dirancang untuk mendukung proses penyimpanan, pengolahan, dan pengambilan data secara efisien serta terstruktur. Sistem basis data disusun berdasarkan kebutuhan utama aplikasi, yaitu mengelola informasi tentang gejala, penyakit, serta relasi logis antar keduanya dalam bentuk aturan (*rule*) yang menjadi dasar proses inferensi. Terdapat tiga tabel inti yang digunakan dalam implementasi ini, yaitu tabel penyakit, gejala, dan *rule*.

Tabel penyakit menyimpan data mengenai berbagai jenis penyakit gigi yang dapat didiagnosis oleh sistem. Informasi yang dicatat mencakup kode penyakit sebagai *primary key*, nama penyakit, deskripsi penyakit, serta saran penanganan yang sesuai. Selanjutnya, tabel gejala mencatat daftar gejala yang mungkin dialami oleh pengguna. Setiap gejala diidentifikasi melalui kode gejala sebagai *primary key* dan dilengkapi dengan nama gejala yang menggambarkan kondisi klinis tertentu.

Tabel *rule* berfungsi sebagai jembatan antara tabel penyakit dan tabel gejala. Tabel ini menyimpan kombinasi antara kode gejala dan kode penyakit beserta nilai *Certainty Factor* (CF) yang menunjukkan tingkat keyakinan pakar terhadap hubungan gejala dan penyakit tersebut. Relasi antara tabel *rule* dengan penyakit dan

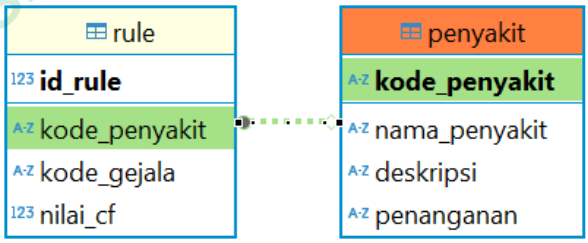
gejala dibangun menggunakan *foreign key*. Struktur relasi antar tabel basis data yang digunakan dalam sistem ini dapat dilihat pada Gambar 4.1.



Gambar 4.1 Implementasi Basis Data

4.2.1 Implementasi Tabel Penyakit

Tabel penyakit dirancang untuk menyimpan data mengenai jenis-jenis penyakit gigi. Setiap entri dalam tabel ini memiliki atribut berupa kode_penyakit sebagai *primary key*, nama_penyakit untuk menyebutkan nama penyakit secara spesifik, deskripsi sebagai penjelasan detail mengenai penyakit tersebut, dan penanganan untuk memberikan informasi mengenai saran tindakan medis atau solusi yang sesuai. Implementasi dari tabel penyakit dapat dilihat pada Gambar 4.2.

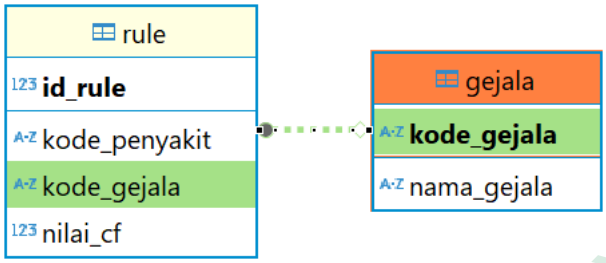


Gambar 4.2 Implementasi Tabel Penyakit

4.2.2 Implementasi Tabel Gejala

Fungsi dari tabel gejala adalah menyimpan daftar gejala yang dapat dipilih sesuai dengan apa yang dirasakan oleh pengguna. Struktur tabel ini mencakup dua atribut utama, yaitu kode_gejala sebagai *primary key* dan nama_gejala sebagai penjelasan gejala secara deskriptif. Tabel ini menjadi bagian penting dalam proses

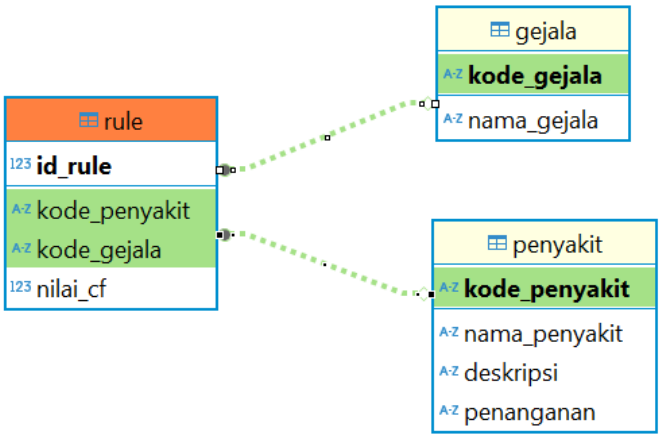
diagnosis karena input gejala dari pengguna akan dibandingkan dengan data yang ada dalam tabel ini untuk kemudian diproses lebih lanjut dalam mekanisme inferensi. Implementasi tabel gejala dapat dilihat pada Gambar 4.3.



Gambar 4.3 Implementasi Tabel Gejala

4.2.3 Implementasi Tabel Rule

Tabel *rule* merupakan inti dari proses inferensi dalam sistem pakar yang menggabungkan antara gejala dan penyakit berdasarkan aturan yang telah ditetapkan oleh pakar. Tabel ini memiliki atribut id_rule sebagai *primary key*, kode_penyakit dan kode_gejala sebagai *foreign key* yang terhubung dengan tabel penyakit dan gejala. Selain itu, terdapat atribut nilai_cf yang menunjukkan bobot tingkat keyakinan pakar terhadap hubungan antara gejala dan penyakit, sesuai dengan metode *Certainty Factor*. Tabel ini akan digunakan dalam proses perhitungan untuk menentukan kemungkinan penyakit berdasarkan gejala yang dipilih pengguna. Implementasi tabel *rule* dapat dilihat pada Gambar 4.4.



Gambar 4.4 Implementasi Tabel *Rule*

4.3 IMPLEMENTASI DESAIN ANTARMUKA

Antarmuka pengguna dirancang agar intuitif, responsif, dan mudah digunakan oleh pengguna dari berbagai kalangan. Desain antarmuka dikembangkan menggunakan pendekatan modern dan minimalis dengan memperhatikan konsistensi elemen visual dan navigasi aplikasi.

4.3.1 Implementasi Halaman Login

Halaman *login* berperan sebagai gerbang awal bagi pengguna untuk mengakses sistem aplikasi. Melalui halaman ini, pengguna harus memasukkan alamat email serta password yang telah dibuat sebelumnya untuk dapat melanjutkan akses ke dalam sistem. Tersedia ikon mata pada input password untuk mempermudah verifikasi karakter oleh pengguna. Selain itu, terdapat tombol navigasi menuju halaman pendaftaran bagi pengguna baru. Halaman *login* dapat dilihat pada Gambar 4.5.

Gambar 4.5 Implementasi Halaman *Login*

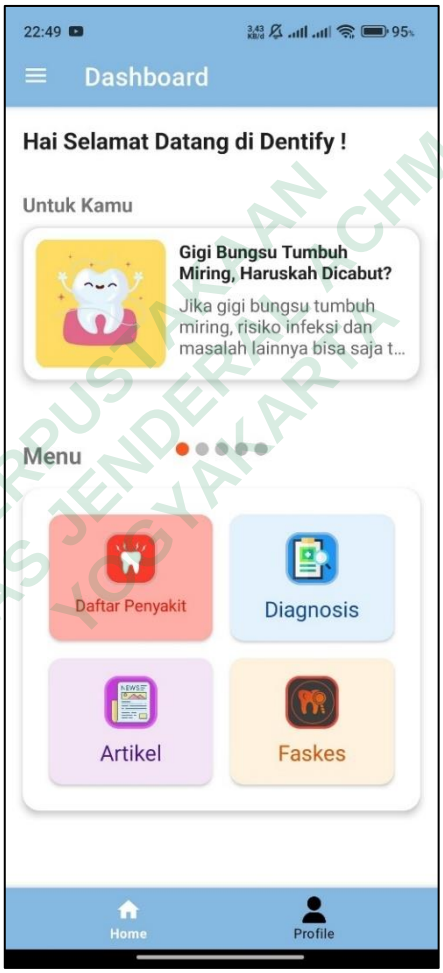
4.3.2 Implementasi Halaman Registrasi

Halaman registrasi memungkinkan pengguna baru untuk membuat akun dalam sistem. Pengguna diwajibkan mengisi tiga kolom input, yaitu email, password, dan konfirmasi ulang password. Untuk mendukung keamanan data, input password disembunyikan secara *default* dengan opsi untuk menampilkannya. Tombol daftar digunakan untuk mengirim data pendaftaran ke *Firebase Authentication*. Halaman registrasi ditampilkan pada Gambar 4.6.

Gambar 4.6 Implementasi Halaman Registrasi

4.3.3 Implementasi Halaman Dashboard

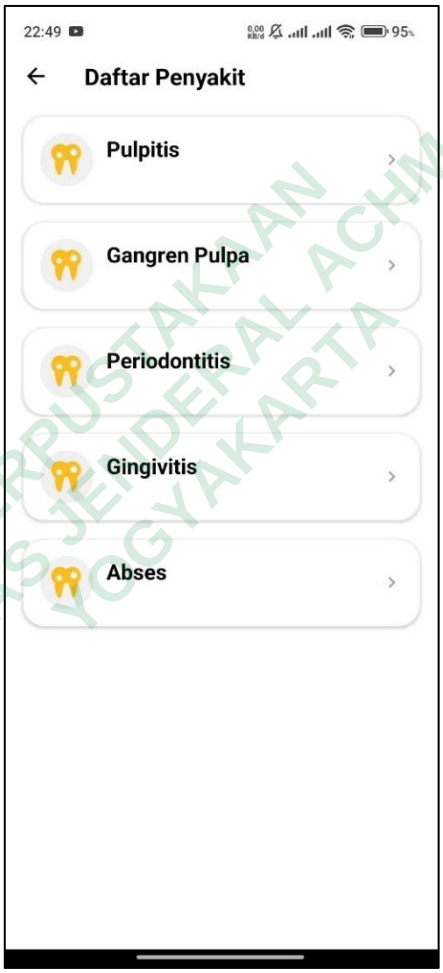
Dashboard merupakan halaman utama setelah pengguna berhasil masuk ke aplikasi. Halaman ini menyajikan sambutan personal, artikel pilihan terkait kesehatan gigi, serta menu navigasi utama yang terdiri dari empat fitur, daftar penyakit, diagnosis, artikel, dan faskes. Tata letak dirancang dengan warna berbeda untuk setiap ikon agar memudahkan identifikasi menu. Halaman *dashboard* dapat dilihat pada Gambar 4.7.



Gambar 4.7 Implementasi Halaman *Dashboard*

4.3.4 Implementasi Halaman Daftar Penyakit

Halaman daftar penyakit menampilkan seluruh jenis penyakit gigi yang telah tersimpan dalam basis data. Masing-masing penyakit disajikan dalam bentuk *list* item vertikal yang dapat diklik untuk membuka detail informasi lebih lanjut. Penggunaan ikon gigi dan teks tebal pada nama penyakit membantu pengguna mengenali isi daftar dengan cepat. Halaman daftar penyakit dapat dilihat pada Gambar 4.8.



Gambar 4.8 Implementasi Halaman Daftar Penyakit

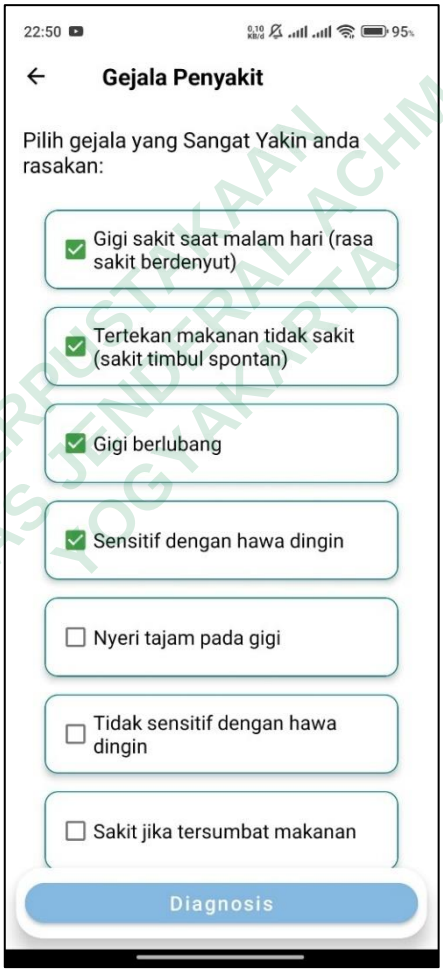
4.3.5 Implementasi Halaman Detail Penyakit

Halaman detail penyakit menyajikan informasi lengkap mengenai penyakit gigi tertentu. Bagian atas menampilkan nama penyakit dan gambar ilustrasi, diikuti dengan penjelasan penyakit serta informasi mengenai penanganan yang disarankan. Informasi ini ditampilkan dalam dua kotak terpisah untuk memudahkan pembacaan. Halaman detail penyakit dapat dilihat pada Gambar 4.9.

Gambar 4.9 Implementasi Halaman Detail Penyakit

4.3.6 Implementasi Halaman Diagnosis

Halaman diagnosis dirancang untuk memungkinkan pengguna melakukan pemilihan gejala yang dirasakan berdasarkan daftar gejala yang tersedia. Pengguna dapat memilih lebih dari satu gejala dengan menandai kotak centang (*checkbox*) yang tersedia pada setiap item gejala. Tampilan dirancang dalam bentuk *list* dengan desain minimalis dan responsif terhadap interaksi pengguna. Setelah satu atau lebih gejala dipilih, tombol diagnosis akan aktif dan dapat ditekan untuk melanjutkan proses analisis. Halaman diagnosis dapat dilihat pada Gambar 4.10.



Gambar 4.10 Implementasi Halaman Diagnosis

4.3.7 Implementasi Halaman Hasil Diagnosis

Tampilan hasil diagnosis menampilkan analisis berdasarkan gejala-gejala yang telah dipilih oleh pengguna sebelumnya. Informasi yang ditampilkan meliputi daftar gejala yang dipilih, hasil dari proses diagnosis berupa identifikasi nama penyakit gigi, serta tingkat persentase keyakinan dari sistem. Terdapat pula informasi metode inferensi yang digunakan. Desain halaman menggunakan tata letak berbasis *card* dan elemen visual yang mendukung keterbacaan informasi. Halaman hasil diagnosis dapat dilihat pada Gambar 4.11.

Gambar 4.11 Implementasi Halaman Hasil Diagnosis

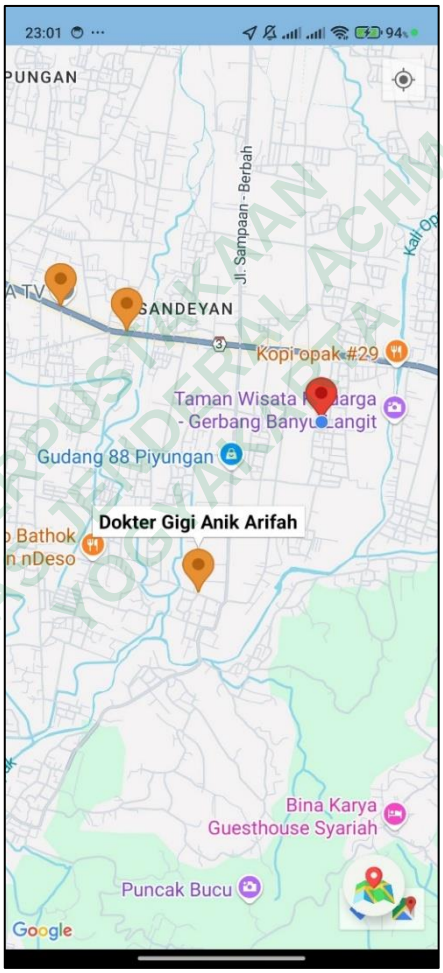
4.3.8 Implementasi Halaman Artikel

Halaman artikel menampilkan daftar artikel terkait kesehatan dalam format daftar vertikal. Setiap artikel ditampilkan dalam bentuk *card* yang memuat judul, ringkasan isi, tanggal publikasi, serta ikon sumber informasi. Pengguna dapat memilih salah satu artikel untuk membuka detail isi artikel tersebut. Halaman artikel dapat dilihat pada Gambar 4.12.

Gambar 4.12 Implementasi Halaman Artikel

4.3.9 Implementasi Halaman Fasilitas Kesehatan

Halaman faskes menyajikan informasi lokasi fasilitas kesehatan gigi terdekat dalam bentuk peta interaktif. Titik lokasi ditandai dengan pin yang dapat ditekan untuk menampilkan informasi seperti nama fasilitas, alamat, dan jarak dari lokasi pengguna. Desain antarmuka menyesuaikan tampilan *mobile* agar memudahkan pengguna dalam menavigasi dan menemukan layanan kesehatan. Halaman faskes dapat dilihat pada Gambar 4.13.



Gambar 4.13 Implementasi Halaman Fasilitas Kesehatan

4.3.10 Implementasi Halaman Profile

Halaman *profile* menampilkan informasi pengguna dan menyediakan fitur untuk mengganti kata sandi. Formulir penggantian kata sandi terdiri atas dua kolom isian, yaitu password saat ini dan password baru, lengkap dengan ikon untuk melihat kata sandi. Tersedia tombol ubah password untuk menyimpan perubahan. Halaman *profile* dapat dilihat pada Gambar 4.14.

Gambar 4.14 Implementasi Halaman *Profile*

4.4 IMPLEMENTASI KODE SISTEM

Implementasi dilakukan menggunakan bahasa pemrograman kotlin untuk pengembangan aplikasi *mobile*. Setiap fitur utama dijelaskan melalui potongan kode dan disertai dengan uraian fungsionalitasnya. Bagian ini bertujuan untuk menjelaskan secara teknis bagaimana sistem dioperasikan, sesuai dengan kebutuhan fungsional yang telah dirumuskan pada tahap perancangan sebelumnya.

4.4.1 Kode Login

Kode *login* digunakan untuk menangani proses autentikasi pengguna menggunakan *Firebase Authentication*. Sistem akan memverifikasi input pengguna terlebih dahulu sebelum mengirimkannya ke server untuk proses autentikasi. Berikut merupakan cuplikan kode *login*:

```
binding.button.setOnClickListener {
    val email = binding.emailEt.text.toString().trim()
    val pass = binding.passET.text.toString().trim()
```

Bagian ini menangani aksi saat tombol *login* ditekan. Input dari kolom email dan password diambil, lalu dibersihkan dari spasi menggunakan fungsi *trim()* untuk mencegah kesalahan akibat karakter kosong di awal atau akhir input.

```
    if (email.isEmpty() || pass.isEmpty()) {
        Toast.makeText(
            this,
            "Email dan password tidak boleh kosong",
            Toast.LENGTH_SHORT
        ).show()
        return@setOnClickListener
    }
```

Setelah input diambil, sistem memeriksa apakah email atau password kosong. Jika salah satu kosong, proses *login* dihentikan dan pesan peringatan ditampilkan kepada pengguna menggunakan *Toast*.

```
    firebaseAuth.signInWithEmailAndPassword(email, pass)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                startActivity(
                    Intent(this, MainActivity::class.java)
                )
                finish()
            } else {
                Toast.makeText(
```

```
                    this,
                    "Login gagal: ${task.exception?.message}",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
        .addOnFailureListener {
            Toast.makeText(
                this,
                "Terjadi kesalahan: ${it.localizedMessage}",
                Toast.LENGTH_LONG
            ).show()
```

Jika input valid, sistem memproses autentikasi menggunakan *signInWithEmailAndPassword()* dari *Firebase*. Jika *login* berhasil, pengguna diarahkan ke halaman utama (*MainActivity*) dan aktivitas *login* ditutup. Namun jika gagal, pesan kesalahan ditampilkan. *addOnFailureListener* digunakan untuk menangani kesalahan umum seperti gangguan jaringan, dan akan menampilkan pesan tambahan kepada pengguna.

4.4.2 Kode Registrasi

Kode registrasi digunakan untuk membuat akun pengguna baru menggunakan *Firebase Authentication*. Sistem akan memverifikasi input terlebih dahulu sebelum mengirimkannya ke server. Berikut merupakan cuplikan kode registrasi:

```
binding.button.setOnClickListener {
    val email = binding.emailEt.text.toString().trim()
    val pass = binding.passET.text.toString()
    val confirmPass = binding.confirmPassEt.text.toString()

    if (email.isEmpty() || pass.isEmpty() || confirmPass.isEmpty()) {
        Toast.makeText(
            this,
            "Data tidak boleh kosong",
            Toast.LENGTH_SHORT
        ).show()
        return@setOnClickListener
    }

    if (pass != confirmPass) {
        Toast.makeText(
            this,
            "Password tidak sesuai",
            Toast.LENGTH_SHORT
        ).show()
        return@setOnClickListener
```

Bagian pertama kode ini mengambil input email, password, dan konfirmasi password dari pengguna. Selanjutnya dilakukan validasi, jika ada kolom yang kosong, atau jika password tidak sesuai dengan konfirmasi, maka proses registrasi dihentikan dan pengguna akan diberi peringatan melalui *Toast*.

```
    firebaseAuth.createUserWithEmailAndPassword(email, pass)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                startActivity(Intent(this, SignInActivity::class.java))
                finish()
            } else {
                Toast.makeText(
                    this,
                    "Registrasi gagal: ${task.exception?.message}",
                    Toast.LENGTH_LONG
                ).show()
            }
```

Jika data valid, sistem menjalankan proses pendaftaran akun melalui *createUserWithEmailAndPassword()*. Jika registrasi berhasil, pengguna langsung diarahkan ke halaman *login*. Jika gagal, sistem akan menampilkan pesan kesalahan dari *Firebase* untuk memberi tahu pengguna alasan kegagalan.

4.4.3 Kode Dashboard

Kode *dashboard* bertanggung jawab untuk mengatur navigasi pengguna dari halaman utama (*dashboard*) menuju fitur-fitur utama dalam aplikasi. Setiap menu dashboard diwakili oleh *CardView*, seperti menu penyakit, diagnosis, fasilitas kesehatan, dan artikel. Ketika pengguna memilih salah satu menu, sistem akan membuka halaman (*activity*) yang sesuai menggunakan *Intent*. Berikut merupakan cuplikan kode navigasi pada halaman *dashboard*:

```
mcPenyakit.setOnClickListener {
    startActivity(Intent(requireContext(), DaftarPenyakitActivity::class.java))
}

mcDiagnosis.setOnClickListener {
    startActivity(Intent(requireContext(), KonsultasiActivity::class.java))
}
```

Kode di atas menangani navigasi untuk dua menu utama. Saat pengguna menekan menu "Penyakit", sistem akan membuka halaman daftar penyakit.

Sementara itu, saat menu "Diagnosis" ditekan, sistem mengarahkan ke halaman konsultasi diagnosa.

```
cvFaskes.setOnClickListener {
    startActivity(Intent(requireContext(), FaskesActivity::class.java))
}

cvArtikel.setOnClickListener {
    startActivity(Intent(requireContext(), ArticleListActivity::class.java))
}

return view
```

Selanjutnya, kode ini juga mengatur navigasi ke halaman fasilitas kesehatan dan artikel. Menu "Faskes" akan menampilkan daftar lokasi fasilitas terdekat, sedangkan menu "Artikel" menampilkan kumpulan artikel terkait informasi kesehatan gigi. Semua navigasi dilakukan menggunakan *Intent*, dengan konteks yang berasal dari *fragment* melalui *requireContext()*.

4.4.4 Kode Daftar Penyakit

Kode daftar penyakit berfungsi untuk menampilkan daftar penyakit gigi yang tersimpan dalam database lokal SQLite. Data diambil menggunakan fungsi *getDaftarPenyakit()* dari *DatabaseHelper*, kemudian ditampilkan menggunakan komponen *RecyclerView*. Berikut merupakan cuplikan kode daftar penyakit:

```
private val listData: Unit
    get() {
        modelDaftarPenyakitList =
            databaseHelper!!.getDaftarPenyakit()

        if (modelDaftarPenyakitList.size == 0) {
            rvDaftarPenyakit!!.visibility = View.GONE
        } else {
            rvDaftarPenyakit!!.visibility = View.VISIBLE
            daftarPenyakitAdapter = DaftarPenyakitAdapter(
                this,
                modelDaftarPenyakitList
            )
            rvDaftarPenyakit!!.adapter = daftarPenyakitAdapter
        }
```

Kode di atas memuat data penyakit dari database dan menyimpannya dalam *modelDaftarPenyakitList*. Jika data kosong, *RecyclerView* disembunyikan. Jika data tersedia, *RecyclerView* ditampilkan dan diisi menggunakan adapter yang

sesuai dengan struktur data. Hal ini memastikan hanya data yang valid yang ditampilkan kepada pengguna.

4.4.5 Kode Detail Penyakit

Kode detail penyakit digunakan untuk menampilkan informasi detail dari penyakit yang dipilih pengguna. Data diambil dari database lokal berdasarkan kode_penyakit yang dikirim melalui *intent* dari halaman sebelumnya. Berikut merupakan cuplikan kode detail penyakit:

```
databaseHelper = DatabaseHelper(this)
if (databaseHelper!!.openDatabase()) {
    sqLiteDatabase = databaseHelper!!.readableDatabase
}

val strKodePenyakit = intent.getStringExtra("KODE_PENYAKIT")
val selectQuery = """
    SELECT nama_penyakit, deskripsi, penanganan
    FROM penyakit
    WHERE kode_penyakit = '$strKodePenyakit' """.trimIndent()

val cursor = sqLiteDatabase!!.rawQuery(selectQuery, null)
cursor.moveToFirst()
```

Bagian pertama kode ini membuka koneksi ke database SQLite dan mengambil nilai kode_penyakit dari *intent*. Nilai tersebut digunakan dalam *query* untuk mengambil informasi terkait nama penyakit, deskripsi, dan penanganannya.

```
toolbar = findViewById(R.id.toolbardetail)
tvTitle = findViewById(R.id.tvTitle)
tvPenjelasan = findViewById(R.id.tvPenjelasan)
tvPenanganan = findViewById(R.id.tvPenanganan)

tvTitle.text = cursor.getString(0)
tvPenjelasan.text = cursor.getString(1)
tvPenanganan.text = cursor.getString(2)
cursor.close()
```

Hasil *query* kemudian ditampilkan pada tampilan antarmuka aplikasi. Komponen tvTitle, tvPenjelasan, dan tvPenanganan digunakan untuk menampilkan nama penyakit, deskripsi, serta langkah penanganannya. Setelah data ditampilkan, cursor ditutup untuk menghindari kebocoran memori.

4.4.6 Kode Diagnosis

Kode diagnosis digunakan untuk menangani proses pemilihan gejala oleh pengguna sebelum sistem melakukan perhitungan diagnosis. Gejala ditampilkan dalam bentuk daftar menggunakan *RecyclerView* dan KonsultasiAdapter. Berikut merupakan cuplikan kode diagnosis:

```
rvGejalaPenyakit.layoutManager = LinearLayoutManager(this)
konsultasiAdapter = KonsultasiAdapter(
    this,
    modelKonsultasiArrayList
)
rvGejalaPenyakit.adapter = konsultasiAdapter
rvGejalaPenyakit.setHasFixedSize(true)
```

Bagian pertama kode mengatur tampilan daftar gejala menggunakan *RecyclerView* dengan *LinearLayoutManager*. Data gejala yang dimuat dari *modelKonsultasiArrayList* kemudian dikaitkan dengan KonsultasiAdapter agar dapat ditampilkan secara interaktif kepada pengguna.

```
btnHasilDiagnosa.setOnClickListener { v ->
    val gejalaTerpilih = StringBuffer()
    val gejalaList = modelKonsultasiArrayList

    for (i in gejalaList.indices) {
        val gejala = gejalaList[i]
        if (gejala.isSelected()) {
          gejalaTerpilih.append(gejala.getStrGejala()).append("#")
        }
    }
    if (gejalaTerpilih.toString().isEmpty()) {
        Toast.makeText(
            this@KonsultasiActivity,
            "Silakan pilih gejala dahulu!",
            Toast.LENGTH_SHORT
        ).show()
    } else {
        val intent = Intent(v.context, HasilDiagnosaActivity::class.java)
        intent.putExtra("HASIL", gejalaTerpilih.toString())
        startActivity(intent)
    }
}
```

Bagian kedua menangani pengambilan data gejala yang dipilih pengguna. Data gejala yang dicentang dikumpulkan ke dalam *StringBuffer* dan dipisahkan dengan tanda pagar (#). Jika tidak ada gejala yang dipilih, sistem akan menampilkan

peringatan. Jika ada, data dikirim melalui *Intent* ke HasilDiagnosaActivity untuk diproses lebih lanjut oleh sistem pakar.

4.4.7 Kode Hasil Diagnosis

Kode hasil diagnosis terdiri dari dua bagian utama pencarian kandidat penyakit menggunakan metode *Forward Chaining*, dan perhitungan tingkat kepastian diagnosis menggunakan metode *Certainty Factor*. Berikut merupakan cuplikan kode hasil diagnosis:

```
val kandidat = mutableListOf<String>()
val cursorPenyakit = sqLiteDatabase!!.rawQuery(
    "SELECT DISTINCT kode_penyakit FROM rule", null
)

while (cursorPenyakit.moveToNext()) {
    val kodePenyakit = cursorPenyakit.getString(0)
    val cursorRule = sqLiteDatabase!!.rawQuery(
     """
       SELECT kode_gejala
       FROM rule
        WHERE kode_penyakit = '$kodePenyakit'
     """.trimIndent(), null
    )

    var cocok = false
    while (cursorRule.moveToNext()) {
        val kodeGejalaRule = cursorRule.getString(0)
        if (gejalaTerpilih.contains(kodeGejalaRule)) {
            cocok = true
            break
        }
    }
    if (cocok) kandidat.add(kodePenyakit)
    cursorRule.close()
```

Kode di atas memeriksa semua penyakit yang memiliki aturan (*rule*) di database. Untuk setiap penyakit, sistem akan membandingkan gejala yang dipilih pengguna dengan kode gejala dalam *rule*. Jika ada minimal satu kecocokan, penyakit tersebut dimasukkan ke dalam daftar kandidat diagnosis.

```
var cfGabungan = 0.0
val cursorRule = sqLiteDatabase!!.rawQuery(
    """
    SELECT nilai_cf
    FROM rule
    WHERE kode_penyakit = '$kodePenyakit'
    """, null
```

```
)
while (cursorRule.moveToNext()) {
    val cfExpert = cursorRule.getDouble(0)
    val cfUser = CF_USER
    val cfRule = cfExpert * cfUser

    cfGabungan = if (cfGabungan == 0.0) cfRule
    else cfGabungan + (cfRule * (1 - cfGabungan))
}
val hasilPersen = cfGabungan * 100
```

Bagian ini menghitung nilai *Certainty Factor* untuk satu penyakit berdasarkan semua rule yang cocok. Nilai CF dari pakar dikalikan dengan CF pengguna, lalu dikombinasikan menggunakan rumus CF gabungan. Nilai akhir dikonversi menjadi persentase yang menunjukkan tingkat keyakinan sistem terhadap hasil diagnosis.

4.4.8 Kode Artikel

Kode artikel bertanggung jawab untuk menampilkan daftar artikel kesehatan pada halaman artikel aplikasi. Artikel diambil secara *real-time* melalui API eksternal menggunakan *coroutine* agar tidak mengganggu *main thread*. Berikut merupakan cuplikan kode artikel:

```
private fun fetchArticles() {
    CoroutineScope(Dispatchers.IO).launch {
        val response = newsApiService
            .getArticles("Kesehatan", apiKey)

        if (!response.isSuccessful) return@launch
        val articles = response.body()?.articles ?: emptyList()
        withContext(Dispatchers.Main) {
            articlesAdapter = ArticlesAdapter(
                articles
            ) { article ->
                val intent = Intent(
                    this@ArticleListActivity,
                    ArticleDetailActivity::class.java
                )
                intent.putExtra("ARTICLE_URL", article.url)
                startActivity(intent)
            }
            recyclerView.adapter = articlesAdapter
```

Fungsi *fetchArticles()* menjalankan proses pengambilan data artikel secara asinkron menggunakan *coroutine* dengan *dispatcher IO*, yang ditujukan untuk operasi jaringan. Data artikel diambil melalui *newsApiService.getArticles()*, dan

jika berhasil, daftar artikel ditampilkan di antarmuka utama (UI) dengan *withContext(Dispatchers.Main)*. Artikel ditampilkan melalui *ArticlesAdapter*, dan ketika pengguna menekan salah satu item, sistem akan membuka *ArticleDetailActivity* dan menampilkan konten artikel tersebut melalui URL yang dikirim via *intent*.

4.4.9 Kode Fasilitas Kesehatan

Kode fasilitas kesehatan digunakan untuk menampilkan lokasi fasilitas kesehatan gigi pada peta. Terdapat dua fungsi utama, yaitu *addManualClinics()* untuk menambahkan marker klinik secara manual, dan *openGoogleMapsSearch()* untuk membuka pencarian lokasi dokter gigi menggunakan aplikasi Google Maps. Berikut merupakan cuplikan kode fasilitas kesehatan:

```
private fun addManualClinics() {
    val clinics = listOf(
        Triple("Dokter Gigi Anik Arifah", -7.8428388, 110.4399384)
    )

    for ((name, lat, lng) in clinics) {
        val latLng = LatLng(lat, lng)
        googleMap.addMarker(
            MarkerOptions()
                .position(latLng)
                .title(name)
                .icon(
                    BitmapDescriptorFactory.defaultMarker(
                        BitmapDescriptorFactory.HUE_ORANGE
```

Fungsi *addManualClinics()* digunakan untuk menampilkan titik-titik lokasi klinik gigi tertentu pada Google Maps. Data lokasi ditentukan secara manual melalui daftar *Triple*, yang berisi nama klinik serta koordinat lintang dan bujur. Setiap titik akan ditampilkan dengan marker berwarna oranye dan label sesuai nama klinik.

```
private fun openGoogleMapsSearch() {
    val latLng = userLatLng
    if (latLng != null) {
        val gmmIntentUri = Uri.parse(
            "geo:${latLng.latitude},${latLng.longitude}?q=dokter+gigi"
        )
        val mapIntent = Intent(Intent.ACTION_VIEW, gmmIntentUri)
        mapIntent.setPackage("com.google.android.apps.maps")
        startActivity(mapIntent)
```

```
    } else {
        Toast.makeText(
            this,
            "Lokasi belum tersedia",
            Toast.LENGTH_SHORT
        ).show()
```

Fungsi *openGoogleMapsSearch()* memungkinkan pengguna membuka aplikasi Google Maps dengan kata kunci pencarian "dokter gigi" berdasarkan lokasi mereka saat ini. Jika lokasi berhasil dideteksi (*userLatLng* tidak *null*), maka *intent* akan dikirim ke aplikasi Google Maps. Jika tidak, pengguna akan menerima notifikasi bahwa lokasi belum tersedia.

4.4.10 Kode Profile

Kode *profile* digunakan untuk mengelola fitur ubah password pada halaman *profile*. Sistem akan melakukan validasi input, proses verifikasi identitas (*reauthentication*), dan pembaruan password di *Firebase*. Berikut merupakan cuplikan kode untuk fitur ubah password:

```
val credential = EmailAuthProvider.getCredential(
    currentUser.email!!, currentPassword
)

currentUser.reauthenticate(credential)
    .addOnCompleteListener { authTask ->
        if (authTask.isSuccessful) {
            currentUser.updatePassword(newPassword)
                .addOnCompleteListener { updateTask ->
                    if (updateTask.isSuccessful) {
                        auth.signOut()
                        startActivity(
                         Intent(requireContext(), SignInActivity::class.java)
```

Kode di atas merupakan bagian inti dari proses ubah password. Pertama, sistem membuat objek *credential* berdasarkan email dan password lama pengguna. Objek ini digunakan untuk proses *reauthenticate*, guna memastikan bahwa pengguna yang sedang *login* adalah pemilik sah akun. Jika proses berhasil, sistem akan melanjutkan dengan updatePassword untuk mengganti password dengan yang baru. Setelah password diperbarui, pengguna akan otomatis keluar dari akun dan diarahkan ke halaman *login*.

4.5 PEMBAHASAN

Bagian pembahasan ditujukan untuk menguraikan bagaimana sistem pakar berbasis *mobile* diimplementasikan, serta menilai tingkat efektivitasnya dalam melakukan diagnosis penyakit gigi menggunakan pendekatan *Forward Chaining* dan *Certainty Factor*.

4.5.1 Identifikasi Penyakit Oleh Sistem

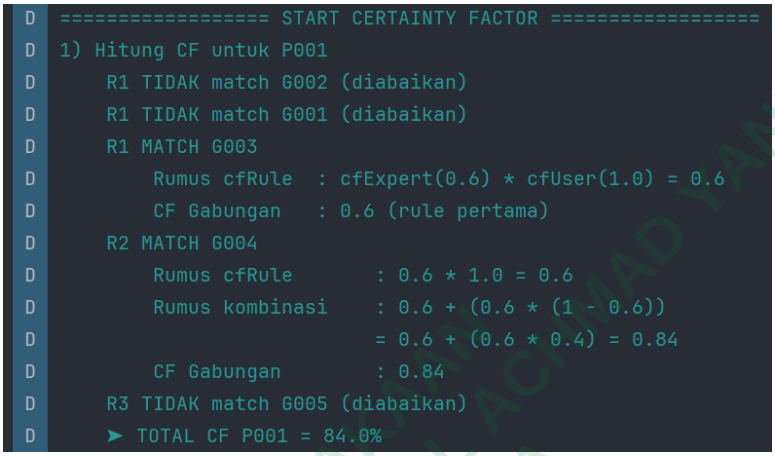
Sistem mampu mengidentifikasi penyakit berdasarkan data gejala yang dipilih oleh pengguna. Melalui penerapan metode *Forward Chaining*, sistem melakukan penelusuran terhadap basis aturan (*rule base*) guna menemukan kecocokan antara gejala input dan gejala dalam setiap aturan penyakit. Berdasarkan log sistem, sebagai contoh, untuk penyakit P001, gejala G003 dan G004 berhasil dicocokkan dengan input pengguna, meskipun tidak semua gejala terpenuhi. Hal ini menunjukkan bahwa sistem tidak hanya bergantung pada pencocokan penuh, tetapi juga mampu mengakomodasi kecocokan sebagian, sehingga memperluas cakupan kemungkinan penyakit yang relevan. Log sistem *Forward Chaining* dapat dilihat pada Gambar 4.15.


```
D ================== START FORWARD CHAINING ==================
D 1. Cek P001
D     ✗ G001
D     ✗ G002
D     ✓ G003
D     ✓ G004
D     ✗ G005
D     ➤ kandidat parsial: P001
D 2. Cek P002
D     ✓ G003
D     ✗ G006
D     ✗ G007
D     ✗ G008
D     ✗ G009
D     ➤ kandidat parsial: P002
```

Gambar 4.15 Log Sistem *Forward Chaining*

Metode *Certainty Factor* selanjutnya digunakan untuk menghitung tingkat kepastian terhadap hasil diagnosis. Nilai CF dihitung berdasarkan nilai keyakinan pakar (*expert*) dan keyakinan pengguna (*user*). Contohnya, untuk P001, dua gejala memiliki cf *expert* sebesar 0.6. Perhitungan gabungan CF menghasilkan nilai akhir

sebesar 84%, menunjukkan tingkat kepastian yang cukup tinggi terhadap hasil diagnosa tersebut. Dengan hasil ini, sistem terbukti mampu memberikan diagnosis yang tidak hanya informatif tetapi juga disertai tingkat keyakinan yang terukur, yang dapat meningkatkan kepercayaan pengguna. Log sistem *Certainty Factor* dapat dilihat pada Gambar 4.16.



Gambar 4.16 Log Sistem *Certainty Factor*

4.5.2 Penerapan Metode Inferensi Dalam Sistem

Integrasi metode inferensi *Forward Chaining* dan *Certainty Factor* dilakukan secara berurutan dan efisien dalam aplikasi. Pertama, metode *Forward Chaining* dimanfaatkan untuk menelusuri dan menyaring kemungkinan penyakit yang sesuai dengan gejala-gejala yang telah dipilih oleh pengguna. Penyakit-penyakit yang memenuhi sebagian atau seluruh syarat gejala dimasukkan ke dalam daftar kandidat. Berikutnya, metode *Certainty Factor* digunakan untuk menghitung tingkat kepastian diagnosis dari setiap kandidat penyakit yang telah disaring sebelumnya. Perhitungan ini dilakukan secara dinamis berdasarkan *rule* yang tersedia di database lokal SQLite. Sistem mencatat setiap langkah evaluasi dan perhitungan CF pada log untuk memastikan transparansi dan keterlacakan proses. Kombinasi kedua metode ini terbukti efektif dalam membangun sistem diagnosis, karena masing-masing metode memiliki peran yang saling melengkapi, *Forward Chaining* berfungsi sebagai filter awal dan *Certainty Factor* sebagai penguat keputusan diagnosis.

4.6 PENGUJIAN

Proses pengujian sistem dilakukan melalui pendekatan *Black Box Testing*, yaitu menguji fungsionalitas sistem berdasarkan masukan dan keluaran tanpa melihat struktur internal kode program. Pengujian ini bertujuan untuk memastikan setiap fitur utama dalam aplikasi berjalan sesuai dengan yang diharapkan. Tabel 4.1 menyajikan hasil pengujian dari beberapa fitur utama sistem.

Tabel 4.1 Pengujian *Black Box*

No	Fitur	Langkah Pengujian	Hasil yang Diharapkan	Hasil
1	*Login*	Mengisi email dan password yang valid	Berhasil *login* dan diarahkan ke *dashboard*	Berhasil
2	*Login*	Mengisi email atau password yang salah	Muncul pesan error: "Periksa kembali email dan password anda"	Berhasil
3	Registrasi	Mengisi email, password, dan konfirmasi password yang sesuai	Akun berhasil dibuat dan pengguna diarahkan ke halaman *dashboard*	Berhasil
4	Registrasi	Password dan konfirmasi password tidak cocok	Muncul peringatan bahwa password tidak cocok	Berhasil
5	*Dashboard*	Berhasil *login* dan masuk ke *dashboard*	Menampilkan menu: Daftar Penyakit, Diagnosis, Artikel, dan Faskes	Berhasil
6	Daftar Penyakit	Mengakses menu Daftar Penyakit	Menampilkan daftar penyakit sesuai data di database	Berhasil
7	Detail Penyakit	Memilih salah satu penyakit dari daftar	Menampilkan detail: nama penyakit, penjelasan, dan penanganan	Berhasil
8	Diagnosis Penyakit	Memilih beberapa gejala dan melakukan proses diagnosis	Menampilkan hasil diagnosis berupa nama penyakit dan persentase keyakinan	Berhasil

No	Fitur	Langkah Pengujian	Hasil yang Diharapkan	Hasil
9	Menu Artikel	Mengakses konten artikel pada *dashboard*	Menampilkan isi artikel terkait kesehatan	Berhasil
10	Menu Fasilitas Kesehatan	Membuka menu faskes	Menampilkan peta lokasi fasilitas kesehatan gigi terdekat	Berhasil
11	Ubah Password (berhasil)	Memasukkan password lama dan password baru yang sesuai	Password berhasil diubah, lalu pengguna diarahkan ke halaman *login* ulang	Berhasil
12	Ubah Password (gagal)	Memasukkan password saat ini yang salah	Muncul pesan kesalahan bahwa password saat ini tidak sesuai	Berhasil

Berdasarkan hasil pengujian menggunakan metode *Black Box*, seluruh fitur utama dalam aplikasi menunjukkan perilaku yang sesuai dengan yang diharapkan. Setiap skenario pengujian, baik dalam kondisi input valid maupun tidak valid, memberikan output yang tepat. Hal ini menunjukkan bahwa sistem telah mampu menangani proses *login*, registrasi, navigasi menu, diagnosis penyakit, serta pengelolaan akun pengguna dengan baik.

Setelah dilakukan pengujian fungsional melalui metode *black-box* untuk memastikan seluruh fitur sistem berjalan sebagaimana mestinya, tahap selanjutnya adalah melakukan evaluasi terhadap keakuratan hasil diagnosis sistem pakar. Evaluasi ini dilakukan dengan melibatkan pakar untuk membandingkan hasil diagnosis sistem terhadap analisis manual yang diberikan oleh pakar berdasarkan gejala yang sama. Hasil pengujian dapat dilihat pada Tabel 4.2.

Tabel 4.2 Pengujian Pakar

No	Gejala	Diagnosis Sistem	Diagnosis Pakar	Keterangan
1	G001, G002, G004	P001	P001	TP
2	G005, G010	P003	P003	TP
3	G002, G012, G017	P004	-	FP
4	G015, G017	P005	P005	TP

5	G004, G009	P002	-	FP
6	G012, G013	P004	P004	TP
7	G006, G008, G009	P002	P002	TP
8	G004	P001	P001	TP
9	G001, G011, G018	P003	-	FP
10	G016, G017, G018	P005	P005	TP

Kategori Evaluasi:

1. *True Positive* (TP): Sistem dan pakar sama-sama mendiagnosis penyakit yang benar.
2. *False Positive* (FP): Sistem memberikan hasil diagnosis, padahal pakar tidak menyimpulkan apa pun (belum bisa mendiagnosis).
3. *False Negative* (FN): Sistem tidak menemukan penyakit, padahal pakar memberikan diagnosis.
4. *True Negative* (TN): Sistem dan pakar sama-sama menyatakan tidak ada penyakit (jika applicable).

Ringkasan:

Keterangan	Jumlah
True Positive (TP)	7
False Positive (FP)	3
True Negative (TN)	0
False Negative (FN)	0

Rumus akurasi:

$$\text{Akurasi } = \frac{TP + TN}{TP + FP + FN + TN} \times 100\ \%$$

$$\text{Akurasi } = \frac{7 + 0}{7 + 3 + 0 + 0} \times 100\ \% \ = \frac{7}{10} \times \ 100\ \% \ = 70\ \%$$

Berdasarkan pengujian yang dilakukan terhadap 10 data uji dan divalidasi oleh pakar, sistem pakar berhasil memberikan diagnosis yang sesuai sebanyak 7

kasus dari total 10 kasus. Dengan hasil perhitungan tingkat akurasi sistem adalah sebesar 70 %, ini menunjukkan bahwa integrasi metode *Forward Chaining* dan *Certainty Factor* dalam sistem dapat memberikan hasil diagnosis yang cukup sesuai dengan pendapat pakar pada sebagian besar kasus.