IMPLEMENTASI RUMUS HAVERSINE DAN ALGORITMA DIJKSTRA DALAM PENCARIAN RUTE PERJALANAN MENGGUNAKAN KRL PADA APLIKASI BERBASIS ANDROID

Erdiansyah, Oman Komarudin, Iqbal Maulana
Program Studi Sistem Informasi S1, Fakultas Ilmu Komputer
Universitas Singaperbangsa Karawang. JL. HS. Ronggo Waluyo, Karawang, Jawa Barat, Indonesia.
erdiansyah.19076@student.unsika.ac.id

ABSTRAK

Kereta Rel Listrik (KRL) merupakan salah satu transportasi publik yang populer di Jabodetabek dan sekitarnya. Tercatat pada bulan Januari 2023 penumpang KRL mencapai 22.716.957 orang. Dimana terdapat 82 stasiun yang tersebar di jabodetabek dan sekitarnya. Sehingga calon penumpang perlu suatu solusi untuk menemukan stasiun terdekat berdasarkan lokasinya dan rute – rute mana saja yang dapat dilalui dalam perjalanan menggunakan transportasi KRL agar dapat melakukan perjalanan dengan lebih baik. Penelitian ini bertujuan untuk melakukan implementasi rumus haversine dan algorima Dijkstra dalam aplikasi Android. Proses pengembangan aplikasi dilakukan dengan metode *Extreme Programming* (XP) dengan memanfaatkan Kotlin sebagai bahasa pemrograman. Hasil pengujian rumus Haversine dan algoritma Dijkstra menunjukkan bahwa keduanya lolos pada kasus pengujian mencari rute KRL dari titik koordinat (-6.213, 106.791) menuju titik koordinat (-6.228, 106.853). Hasil pengujian alpha juga menunjukkan bahwa aplikasi sesuai dengan harapan berdasarkan beberapa kasus yang diuji dan hasil pengujian beta menunjukkan bahwa aplikasi mendapat rata rata nilai 4.1 dengan skala "baik" sehingga aplikasi dapat diterima dan digunakan pengguna.

Kata kunci: *KRL, Android, Haversine, Dijkstra, Extreme Programming.*

1. PENDAHULUAN

Transportasi adalah salah satu komponen yang menunjang keberlangsungan hidup sistem perkotaan dan sistem kemasyarakatan, terutama masyarakat perkotaan[1]. Saat ini keberadaan transportasi publik sudah menjadi bagian dari aktivitas sehari-hari masyarakat terutama kota-kota besar seperti Jakarta, Bogor, Depok, Tangerang, dan Bekasi (Jabodetabek). Salah satu transportasi yang populer adalah Kereta Rel Listrik (KRL). Dilansir dari portal berita Kompas.com, tercatat bahwa penumpang KRL pada bulan Januari 2023 saja mencapai 22.716.957 Orang [2]. Dikutip dari laporan tahunan PT Kereta Commuter Indonesia tahun 2021, terdapat 105 total stasiun KRL dimana 82 diantaranya merupakan stasiun aktif yang terdapat dalam rute KRL Jabodetabek[3]. Dengan banyaknya stasiun yang tersebar di jabodetabek dan sekitarnya calon penumpang perlu suatu solusi untuk menemukan stasiun terdekat berdasarkan lokasinya dan rute – rute mana saja yang dapat dilalui dalam perjalanan menggunakan transportasi KRL agar dapat melakukan perjalanan dengan lebih baik.

Rumus haversine merupakan salah satu persamaan yang dapat digunakan untuk menghitung jarak antara dua koordinat [4]. Dimana koordinat yang digunakan adalah posisi Latitude dan Longitude dari suatu lokasi di Bumi. Algoritma Dijkstra adalah salah satu metode yang dapat digunakan untuk mencari rute terdekat antara dua simpul [5]. Sehingga dengan menggunakan rumus haversine dan algoritma Dijkstra dapat digunakan untuk membantu calon penumpang KRL dalam melakukan perjalanan.

Dalam sebuah penelitian yang dilakukan untuk membandingkan perhitungan jarak koordinat antara metode euclidean dan metode haversine [4]. disimpulkan bahwa kedua metode memiliki hasil dan performa yang tidak jauh berbeda untuk menghitung jarak antar koordinat. Koordinat yang digunakan adalah posisi latitude dan longitude dari sebuah lokasi dan perhitungan dilakukan berdasarkan jarak antara latitude dan longitude lokasi 1 dan lokasi 2.

Penelitian lain yang dilakukan untuk mencari jarak terpendek menuju SPBU dengan membandingkan algoritma Dijkstra dan Bellman-Ford [5]. Penelitian ini menyimpulkan bahwa berdasarkan analisa kompleksitas waktu, diketahui bahwa algoritma Dijkstra membutuhkan waktu lebih sedikit dalam mengeksekusi pencarian rute dibandingkan algoritm Bellman-Ford.

Penelitian ini bertujuan untuk melakukan implementasi rumus haversine dan algorima Dijkstra dalam aplikasi Android agar dapat membantu calon penumpang KRL mengetahui stasiun terdekat dari sebuah lokasi dan rute terbaik serta rute alternatif yang dapat dilalui untuk mencapai stasiun yang dekat dengan titik tujuan.

2. TINJAUAN PUSTAKA

2.1. Android

Android adalah sistem operasi yang paling banyak digunakan pada *smartphone*, dengan lebih dari dua miliar pengguna perangkat dengan sistem operasi ini. Sejak dirilis pada tahun 2007, saat ini android telah banyak digunakan pada perangkat keras lainnya seperti tablet, televisi, jam tangan pintar, perangkat *Internet-of-Things* (IoT), dan beberapa lainnya [6].

Android adalah sistem operasi yang bersifat *open source* dengan komunitas yang masih berkembang dan beragam, dokumentasi yang baik, proses pengembangan dan distribusinya yang tanpa biaya,

banyak dukungan *API Library* yang memungkinkan untuk membuat aplikasi di dalamnya sesuai dengan keinginan pengembang aplikasinya dan memunculkan kesempatan untuk membuat aplikasi yang inovatif terlepas dari pengalaman pengembang aplikasinya [6].

2.2. Algoritma Dijkstra

Algoritma Dijkstra adalah sebuah algoritma untuk mencari jarak terpendek dari sebuah graph berarah yang ditemukan oleh Edsgar Dijkstra. Masalah pencarian jarak terpendek adalah masalah yang krusial karena terhubung dengan masalah optimasi yang relevan seperti jaringan routing, circuit design, pemetaan, dan game sehingga peneliti terus mempelajarinya [7].

Algoritma Dijkstra bekerja dengan menentukan bobot (jarak) dari simpul sumber menuju simpul terdekat, lalu mempertimbangkan bobot tersebut dengan bobot simpul lainnya dan akan dicari yang terkecil sampai akhirnya mencapai simpul tujuan dan didapatkan rute dengan jarak terpendek paling optimal, algoritma ini juga disebut sebagai algoritma greedy karena mengunjungi semua simpul sampai ditemukan jalur dengan bobot simpul terkecil [7].

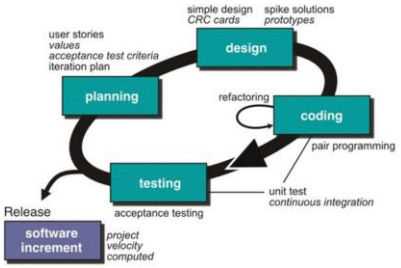
2.3. Rumus Haversine

Rumus atau Metode Haversine adalah persamaan yang digunakan untuk menghitung jarak antara dua titik di permukaan bumi. Rumus pada metode ini menggunakan latitude (garis bujur) dan longitude (garis lintang) dari kedua titik sebagai masukan untuk dicari jaraknya [8]. Rumus ini dapat menghitung jarak antara dua titik dengan asumsi bumi memiliki bulat sempurna sehingga akan mengabaikan ketinggian bukit atau gunung dan kedalaman lembah di permukaan bumi dan masih cukup akurat untuk sebagian besar perhitungan [4]. Berikut pada persamaan 1 yang digunakan pada rumus haversine.

$$Jarak = 2r.\arcsin\left(\sqrt{asin^2\left(\frac{lat_1 - lat_2}{2}\right) + \cos(lat_1).\cos(lat_2).sin^2\left(\frac{long_1 - long_2}{2}\right)}\right) (1)$$

3. METODE PENELITIAN

Metode yang dilakukan pada penelitian ini menggunakan pendekatan metode pengembangan aplikasi Extreme Programming (XP). Extreme Programming atau biasa disebut sebagai XP adalah salah satu model pengembangan perangkat lunak agile. Model ini diciptakan oleh Kent Beck pada tahun 1996, dengan mengenalkannya dalam sebuah buku dengan judul 'Extreme Programming Explained'. Metode ini cukup sederhana dan mudah untuk beradaptasi dengan proses pengembangan yang ambigu dan cepat berubah, sehingga cocok untuk tim berukuran kecil hingga sedang (2 - 10 orang) [9].



Gambar 1. Tahapan Metode Extreme Programming (XP)

Terdapat empat tahapan pada metode ini, diantaranya ialah:

3.1. Perencanaan (Planning)

Pada tahap ini akan dilakukan perencanaan pengembangan aplikasi seperti pengumpulan data, pembuatan user stories untuk mendefinisikan kebutuhan pengguna agar fitur yang akan dibuat nantinya dapat memenuhi kebutuhan tersebut, dan acceptance test criteria

3.2. Perancangan (Design)

Pada tahap ini dilakukan perancangan sistem dan perancangan antarmuka aplikasi android yang akan dibuat. Perancangan sistem dibuat dengan memanfaatkan UML Diagram (Unified Modelling Language Diagram). UML Diagram adalah cara berkomunikasi secara visual untuk membuat dokumentasi sebuah sistem, umumnya sering digunakan pada pengembangan sistem perangkat lunak[10]. Penelitian ini akan menggunakan Use Case Diagram dan Activity Diagram untuk menggambarkan interaksi antara aktor dengan sistem dan aktivitas yang dilakukan dalam sebuah skenario [11].

3.3. Pengkodean (Coding)

Pada tahap ini perancangan yang dilakukan sebelumnya akan diimplementasikan dalam bentuk kode program yang dapat berfungsi. Penelitian ini akan memanfaatkan bahasa pemrograman Kotlin untuk pengembangan aplikasi android dengan menggunakan Android Studio IDE sebagai alat pengembangan aplikasi.

3.4. Pengujian (Testing)

Pada tahapan terakhir ini, akan dilakukan pengujian terhadap kode yang telah dibuat pada tahap sebelumnya. Pengujian akan dilakukan dengan pendekatan alpha testing dan beta testing. Alpha testing atau pengujian alpha adalah pengujian yang dilakukan oleh tim penguji internal pengembang itu sendiri ataupun pengguna yang dipilih untuk melakukan pengujian dengan instruksi yang jelas dari tim pengembang perangkat lunak[12]. Sedangkan beta testing adalah pengujian yang dilakukan oleh target pengguna perangkat lunak tanpa adanya pengawasan dari tim pengembang perangkat lunak dan hasil pengujiannya akan dilaporkan pada tim pengembang

[12]. Pengujian akan berfokus pada pengujian fitur dan fungsionalitas aplikasi.

4. HASIL DAN PEMBAHASAN

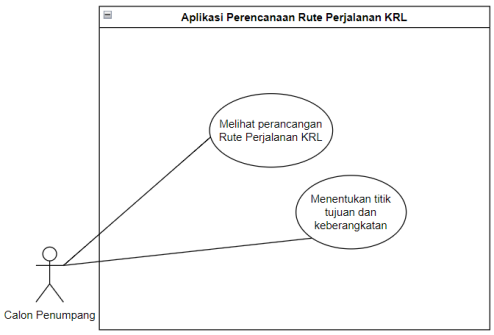
4.1. Perencanaan (Planning)

Kegiatan diawali dengan pengumpulan data dan pembuatan *user stories*. Pengumpulan data dilakukan untuk mengumpulkan kebutuhan sistem, salah satu diantaranya yakni data stasiun dengan kolom nama_stasiun, id_stasiun, type_stasiun, latitude, longitude. Didapatkan hasil berupa daftar 82 stasiun di jabodetabek beserta detailnya. Selanjutnya yakni pembuatan *user stories* yang dibuat berdasarkan kuesioner yang dibagikan kepada 50 responden. Berikut pada gambar 2 adalah *user stories* yang telah dirumuskan.

Gambar 2. Persona *user stories*

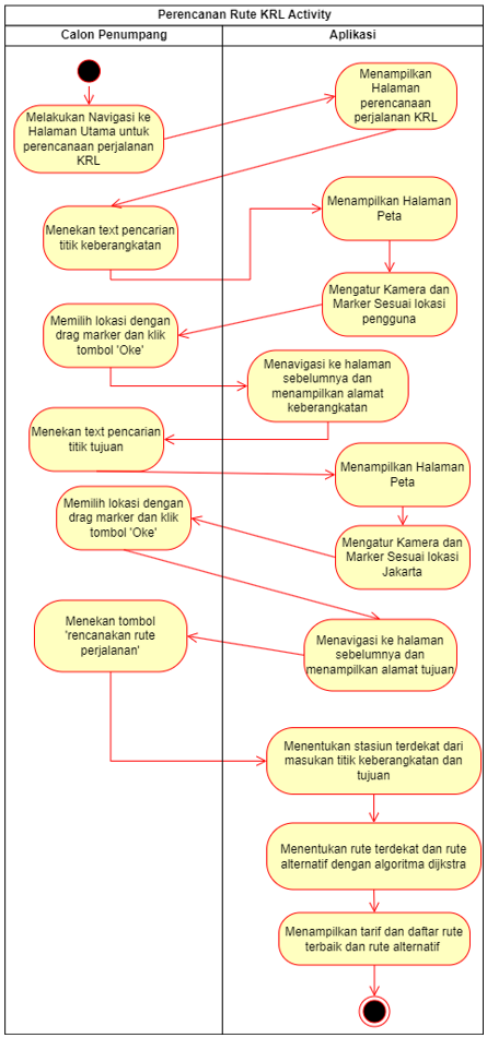
4.2. Perancangan (Design)

Setelah melalui tahap perencanaan, selanjutnya yakni tahap perancangan dimana pada tahap ini dibuat rancangan berdasarkan analisa kebutuhan pada tahap sebelumnya. Rancangan yang dibuat akan menjadi tolak ukur fitur apa saja yang akan disematkan pada aplikasi. Berikut adalah *use case* diagram pada gambar 3 yang menggambarkan interaksi antara pengguna (calon penumpang) dengan sistem.



Gambar 3. *Use case diagram* aplikasi perencanaan rute perjalanan KRL

Berdasarkan use case diagram pada gambar 2, dibuat activity diagram yang memberikan gambaran visual mengenai proses aktivitas yang dilakukan pada suatu skenario. Berikut pada gambar 4 adalah activity diagram.



Gambar 4. *Activity diagram* aplikasi perencanaan rute perjalanan KRL

4.3. Implementasi Rumus Haversine

Rumus Haversine digunakan dalam pencarian stasiun terdekat dari lokasi pengguna. Implementasi dilakukan pada persamaan (1) kedalam bahasa pemrograman kotlin. Berikut adalah implementasi rumus haversine dalam bahasa Kotlin pada gambar 4.

```kotlin
fun haversineCalculateDistance(
    lat1: Double,
    lon1: Double,
    lat2: Double,
    lon2: Double
): Double {
    val R = 6372.8 // Earth Rad in kilometers
    val dLat = Math.toRadians(lat2 - lat1)
    val dLon = Math.toRadians(lon2 - lon1)
    val a = sin(dLat / 2).pow(2) +
            cos(Math.toRadians(lat1)) * cos(Math.toRadians(lat2)) *
            sin(dLon / 2).pow(2)
    val c = 2 * asin(sqrt(a))
    return R * c
}
fun getNearbyStation(context: Context, currentLocation: LatLng, type: String) : Station {
    val stations = Utils.getStation(context)
    var nearestStation: Station? = null
    var minDistance = Double.MAX_VALUE

    for (station in stations) {
        val distance = haversineCalculateDistance(
            currentLocation.latitude,
            currentLocation.longitude,
            station.latitude,
            station.longitude
        )
        if (distance < minDistance) {
            minDistance = distance
            nearestStation = station
        }
    }
    return nearestStation
}
```

Gambar 4. Implementasi rumus Haversine

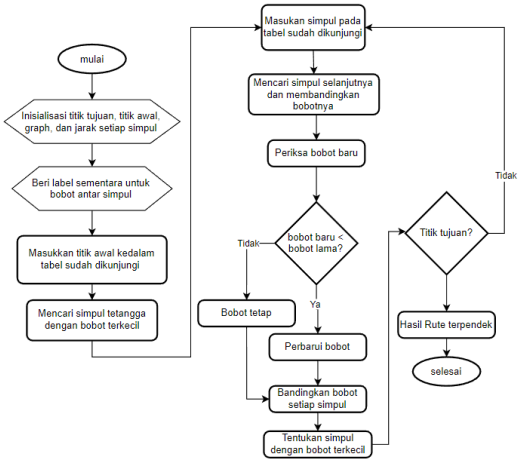
Rumus haversine dilibatkan dalam pencarian stasiun terdekat dengan cara melakukan looping pada daftar stasiun dan dicari jaraknya dengan lokasi pengguna. Selanjutnya jarak akan dibandingkan antar stasiun. Keluaran fungsi adalah stasiun terdekat dari lokasi pengguna.

4.4. Implementasi Algoritma Dijkstra

Algoritma Dijkstra digunakan dalam pencarian jalur terdekat pada rute KRL. penggambaran *graph* dapat terlihat secara visual pada gambar 4 berikut.

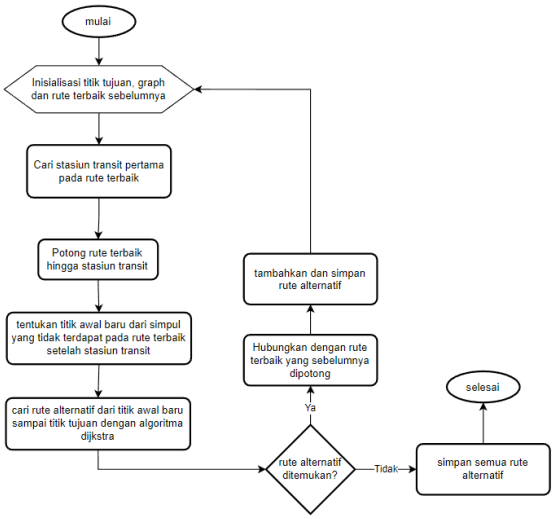
Gambar 4. Peta representasi rute KRL

Dengan pemisalan stasiun sebagai simpulnya dan jalur antar stasiun sebagai *edge* atau garis penghubung antar simpul-simpul. Maka secara konsep penggambaran algoritma Dijkstra secara garis besar dapat dilakukan melalui *flowchart* pada gambar 5 dibawah ini.



Gambar 5. flowchart algoritma dijkstra

Setelah diketahui rute terdekatnya menggunakan algoritma dijkstra, selanjutnya dicari rute alternatif dari titik awal menuju titik tujuan dengan rute yang berbeda daripada rute terdekat sebelumnya. Pencarian rute terdekat ini dilakukan hingga semua kemungkinan rute alternatif yang dapat dilalui habis. Pendekatan yang dilakukan pada penelitian ini yakni dengan mendapatkan stasiun transit atau simpul dengan lebih dari dua *edge*. Selanjutnya rute akan dipotong dari titik awal sampai stasiun transit tersebut dan dicari rute terbaik dengan mengecualikan rute terdekat yang telah dilakukan sebelumnya, setelah didapatkan rute terdekat berikutnya, maka rute awal yang sebelumnya dipotong, akan digabungkan kembali dan menjadi rute alternatif. Pencarian rute alternatif terus berulang hingga rute alternatif yang tersedia habis. Berikut pada gambar 6 adalah gambaran *flowchart* mengenai pencarian rute alternatif secara garis besar.



Gambar 6. flowchart Pencarian rute alternatif

Implementasi pencarian rute terdekat dan rute alternatifnya menggunakan algoritma Dijkstra pada bahasa pemrograman Kotlin dilakukan berdasarkan konsep algoritma Dijkstra pada gambar 5 dan pencarian rute alternatifnya pada gambar 6. Algoritma Dijkstra diimplementasikan dengan membuat sebuah fungsi dengan kembalian berupa *List* stasiun yang merupakan rute terdekat yang dilalui. Berikut pada gambar 7 adalah implementasi algoritma Dijkstra dalam bahasa pemrograman Kotlin.

Pada gambar 7 terlihat bahwa fungsi dijkstra() memiliki satu parameter tambahan yakni initialVisited. Parameter initialVisited dimaksudkan untuk pencarian rute alternatif, ketika pencarian rute terdekat utama parameter ini akan bernilai kosong dan akan diisi ketika pencarian rute alternatif dengan simpul atau stasiun setelah transit yang telah dilewati sebelumnya sehingga stasiun tersebut akan langsung ditandai sebagai visited sehingga tidak akan dilalui lagi. Implementasi pencarian rute alternatif dilakukan dengan membuat sebuah fungsi rekursif dimana fungsi tersebuh memanggil dirinya sendiri. Dengan fungsi ini, pencarian rute alternatif akan dilakukan selama masih terdapat rute alternatif yang tersedia, ketika rute alternatif sudah habis fungsi akan berhenti melakukan rekursif dan menyimpan hasil rute alternatif yang telah ditemukan. Berikut pada gambar 8 adalah implementasi pencarian rute alternatif.

```
suspend fun dijkstra(
    startStationId: String,
    endStationId: String,
    initialVisited: Map<StationBase, Boolean>,
): List<StationBase> {
    val getGraph = repository.getGraph()
    val graph = adjencyList(graph = getGraph.edges, nodes = getGraph.stations)
    val source = graph.keys.find { it.stationId == startStationId } ?: return emptyList()
    val destination = graph.keys.find { it.stationId == endStationId } ?: return emptyList()
    val distance = mutableMapOf<StationBase, Double>()
    val queue = PriorityQueue<Pair<StationBase, Double>>(compareBy { it.second })
    val previous = mutableListOf<Pair<StationBase, StationBase>>()
    val visited = mutableMapOf<StationBase, Boolean>()

    visited.putAll(initialVisited)

    for (node in graph.keys) {
        distance[node] = Double.MAX_VALUE
    }
    distance[source] = 0.0

    queue.offer(Pair(source, 0.0))
    visited[source] = true

    while (queue.isNotEmpty()) {
        val (current, currentDistance) = queue.poll()
        visited[current] = true
        for (edge in (graph[current] ?: emptyList())) {
            val visitedDestination = visited[edge.destination] ?: false
            if (!visitedDestination) {
                val newDistance = currentDistance + edge.distance
                if (newDistance < (distance[edge.destination] ?: Double.MAX_VALUE)) {
                    distance[edge.destination] = newDistance
                    previous.add(Pair(edge.destination, current))
                    queue.offer(Pair(edge.destination, newDistance))
                }
            }
        }
    }
    //if there is no path return emptylist
    if (distance[destination] == Double.MAX_VALUE) return emptyList()

    val path = mutableListOf<StationBase>()
    var current = destination
    while (current != source) {
        val sourceStationBase = previous.find {
            it.first.stationId == current.stationId
        }?.second
        if (sourceStationBase == null) {
            path.clear()
            break
        }
        path.add(current)
        current = sourceStationBase
    }
    path.add(source)
    path.reverse()
    if (path.last() != destination) return emptyList()
    return path
}
```

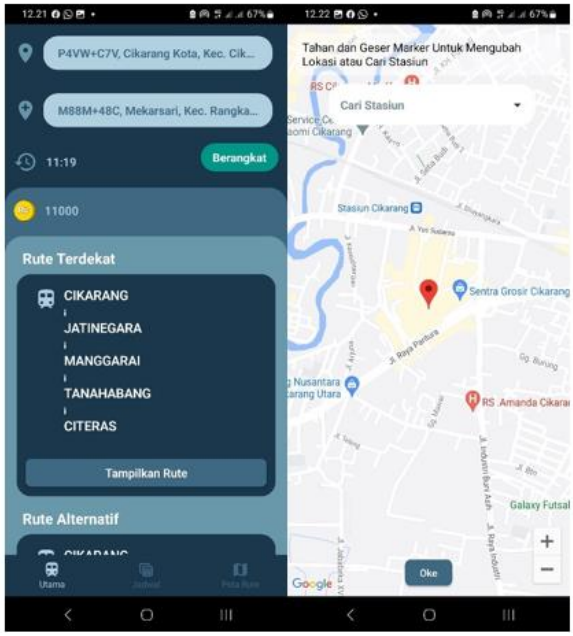
Gambar 7. Implementasi algoritma Dijkstra

```
private suspend fun alternativePath(
    headRoute: List<StationBase>,
    endStationId: String,
    bestRoute: List<StationBase>,
    visited: MutableMap<StationBase, Boolean>,
): List<List<StationBase>> {
    //get graph and destination
    val getGraph = repository.getGraph()
    val graph = adjencyList(graph = getGraph.edges, nodes = getGraph.stations)
    val destination = graph.keys.find { it.stationId == endStationId } ?: return emptyList()

    //get and add head route
    val headBest = mutableListOf<StationBase>()
    headBest.addAll(headRoute)

    //get and add visitedAfterTransit station
    val visitedAfterTransit = mutableMapOf<StationBase, Boolean>()
    visitedAfterTransit.putAll(visited)

    val firstTransit =
        bestRoute.firstOrNull { it.stationType == StationType.TRANSIT && it.stationId != destination.stationId }
    val firstTransitIndexInBestRoute = bestRoute.indexOf(firstTransit)
    val firstTransitAfterVisitedInBestRoute = bestRoute[firstTransitIndexInBestRoute + 1]
    visitedAfterTransit[firstTransitAfterVisitedInBestRoute] = true
    headBest.addAll(bestRoute.subList(0, firstTransitIndexInBestRoute + 1))
    val edgeOfTransit = graph[firstTransit]
    val initialStart = edgeOfTransit?.filter {
        it.destination.stationId != firstTransitAfterVisitedInBestRoute.stationId && it.destination !in bestRoute
    }
    for (i in initialStart ?: emptyList()) {
        val newInitialStart = i.destination

        val map = mutableMapOf<StationBase, Boolean>()

        map[firstTransitAfterVisitedInBestRoute] = true
        map.putAll(visitedAfterTransit)
        map[headBest.last()] = true
        val newBestRoute = dijkstra(
            startStationId = newInitialStart.stationId,
            endStationId = endStationId,
            initialVisited = map
        )
        if (newBestRoute.isNotEmpty()) {
            if (newBestRoute.last() == destination) {
                val newBest = headBest.plus(newBestRoute)
                alternative.add(newBest)
                map.putAll(visitedAfterTransit)
                map[newInitialStart] = true
                alternativePath(
                    headRoute = headBest,
                    endStationId = endStationId,
                    bestRoute = newBestRoute,
                    visited = map
                )
            } else {
                emptyList()
            }
        } else {
            emptyList()
        }
    }
    return emptyList()
}
```

Gambar 8. implementasi pencarian rute alternatif

4.5. Implementasi Aplikasi Android

Implementasi aplikasi android dibuat dengan menggunakan bahasa pemrograman Kotlin. Proses pengkodean aplikasi android dilakukan menggunakan Android Studio IDE yang merupakan IDE resmi untuk pengembangan aplikasi Android. Aplikasi android dibuat dengan melakukan integrasi dengan API Google Maps pada halaman Maps agar calon penumpang dapat dengan mudah menentukan titik keberangkatan dan titik tujuan melalui Peta dengan menggeser *marker* pada peta. Dari titik keberangkatan dan titik tujuan masukan calon penumpang, ditentukan stasiun terdekat dari keduanya dengan menggunakan rumus *haversine*. Stasiun yang telah didapatkan selanjutnya dilakukan pengolahan untuk pembuatan rute terdekat dan rute alternatifnya dengan algoritma Dijkstra. Hasil yang telah diperoleh dari pembuatan rute, selanjutnya ditampilkan pada halaman utama. Berikut adalah tampilan Antarmuka aplikasi perencanaan rute pada gambar 9.



(a) (b)

Gambar 9. (a) tampilan halaman utama (b) tampilan halaman penentuan titik

4.6. Pengujian Perencanaan Rute KRL

Setelah melakukan implementasi baik algoritma Dijkstra dalam pencarian rute terdekat dan rute alternatif. Selanjutnya yakni dilakukan pengujian terkait kesesuaian implementasi dengan konsep yang telah dibuat. Graph yang digunakan yakni sesuai dengan jalur KRL Jabodetabek pada gambar 4. Pengujian dilakukan dengan mencari stasiun terdekat dari titik keberangkatan dan titik tujuan yang dipilih secara acak dengan rumus Haversine. Berikut hasil pengujian pencarian stasiun terdekat pada tabel 1.

Tabel 1. Stasiun Terdekat dari titik keberangkatan dan tujuan

Pengujian rumus Haversine untuk mencari stasiun terdekat			
Jenis Titik	**(Lat, Long)**	**Stasiun Terdekat**	**Bobot Jarak (KM)**
Keberangkatan	(-6.213, 106.791)	Palmerah	0.96115
Tujuan	(-6.228, 106.853)	Tebet	0.539996

Dari hasil pada tabel 1 diketahui bahwa stasiun keberangkatan yakni Stasiun Palmerah dengan tujuan Stasiun Tebet. Berikut pada tabel 2 adalah rute terdekat dan alternatifnya dari Stasiun Palmerah menuju Stasiun Tebet dengan algoritma Dijkstra.

Tabel 2. Rute Stasiun Palmerah Menuju Stasiun Tebet

Pengujian pencarian rute terdekat dan rute alternatif dari stasiun Palmerah hingga stasiun Tebet		
Rute	**Jenis Rute**	**Bobot Total Rute (KM)**
Palmerah > Tanah Abang > Karet > Sudirman Baru > Sudirman > Manggarai > Tebet	Rute Terdekat	11.887
Palmerah > Tanah Abang > Duri > Angke > Kampung Bandan > Jakarta Kota > Jayakarta > Mangga Besar > Sawah Besar > Juanda > Gondangdia > Cikini > Manggarai > Tebet	Rute Alternatif	26.017
Palmerah > Tanah Abang > Duri > Angke > Kampung Bandan > Rajawali > Kemayoran > Pasar Senen > Gang Sentiong > Kramat > Pondok Jati > Jatinegara > Matraman > Manggarai > Tebet	Rute Alternatif	27.803

Pada hasil pengujian di tabel 2. Dapat terlihat bahwa algoritma Dijkstra untuk pencarian rute terdekat dan pencarian rute alternatif dapat dengan baik memberikan keluaran berupa tiga rute untuk mencapai stasiun Tebet dari stasiun Palmerah dengan bobot paling kecil pada rute terdekat yakni 11.887 KM dan bobot paling besar pada rute alternatif yakni 27.803. Sehingga rumus Haversine dan algoritma Dijkstra lolos untuk pencarian rute dari titik (-6.213, 106.791) menuju titik (-6.228, 106.853) dengan terdekat dari kedua titik yakni stasiun Palmerah dan stasiun Tebet.

4.7. Pengujian Aplikasi Andorid

Pengujian aplikasi dilakukan untuk mengetahui apakah aplikasi berjalan sesuai dengan keinginan dan minim *bug*. Jadi, apabila terdapat kesalahan atau bug pada sistem dapat segera ditangani dan meningkatkan kualitas kode yang ditulis. Sehingga aplikasi yang di-*release* nantinya akan minim bug dan dapat digunakan dengan baik. Pengujian dilakukan dengan pendekatan *alpha testing* dan *beta testing* dimana fungsionalitas aplikasi diuji oleh pengembang aplikasi itu sendiri dalam lingkungan terkontrol dan selanjut diuji oleh para *beta tester* yang nantinya akan memberikan *feedback* mengenai aplikasi yang mereka coba.

Berikut adalah prosedur untuk melakukan pengujian pada fitur cek jadwal dan penampil peta rute KRL:

a. Menyiapkan sebuah perangkat *mobile* atau *emulator* dengan sistem operasi Android minimal versi 7 Nougat.
b. Meng-*install* aplikasi perencanaan perjalanan KRL.
c. Melakukan pengujian terhadap aplikasi.
d. Mendokumentasikan hasil pengujian

Berikut adalah hasil pengujian dengan beberapa kasus yang dilakukan dalam pengujian aplikasi pada tabel 1.

Tabel 3. Pengujian alpha perencanaan rute KRL

***Alpha Testing* Perencanaan Rute KRL**			
Kasus	**Harapan**	**Hasil**	**Kesimpulan**
Pengguna menentukan titik keberangka-tan dan tujuan melalui halaman Maps	Alamat dan lokasi titik *marker* disimpan dan ditampilkan	Alamat sesuai dan berhasil ditampilkan pada *textfield* di halaman utama	Berhasil
Pengguna menentukan titik keberangkatan dan tujuan melalui halaman Maps dengan memilih stasiun	Lokasi dan alamat stasiun disimpan dan ditampilkan	Ketika pengguna klik salah satu stasiun marker langsung berpindah pada lokasi stasiun ditampilkan pada *textfield*	Berhasil
Menampilkan daftar rute terbaik dan rute alternatif	Halaman menampilkan animasi *loading* ketika status *loading*	Halaman segera menampilkan animasi *loading* ketika tombol "Berangkat" ditekan	Berhasil
Menampilkan daftar rute terbaik dan rute alternatif	Halaman menampilkan pesan error ketika data tidak berhasil didapatkan	Halaman menampilkan pesan error sesuai dengan masalah yang terjadi ketika gagal mendapatkan data	Berhasil
	Halaman menampilkan *Toast* ketika salah satu dari masukan titik keberangkatan, titik tujuan atau waktu keberangkatan belum dimasukkan pengguna	Halaman berhasil menampilkan *Toast* ketika salah satu masukan pengguna kosong atau tidak diisi	Berhasil
	Halaman menampilkan daftar rute terbaik dan rute alternatif serta tarif saat data berhasil didapatkan	Halaman berhasil menampilkan daftar rute dan tarif setelah animasi *loading* ketika data berhasil didapatkan	Berhasil

Berdasarkan hasil pengujian alpha yang dilakukan pada beberapa kasus seperti pada tabel 1. Didapatkan hasil pada semua kasus memenuhi harapan dengan kesimpulan 'Berhasil'. Hal ini menunjukkan bahwa rumus Haversine dapat menentukan stasiun terdekat dengan baik dan algoritma Dijkstra dapat menunjukkan hasil rute terbaik dan rute alternatif KRL yang dapat dilalui pengguna. Jadi, pengguna dapat dengan baik memanfaatkan fitur aplikasi yang berjalan sesuai dengan harapan untuk melakukan perencanaan rute KRL dari titik keberangkatan dan titik tujuan yang ditentukan pengguna.

Selanjutnya pada pengujian beta dilakukan pengujian dengan 20 *beta tester* yang telah bersedia mengikuti *beta* testing dengan mencoba fitur pada aplikasi perencanaan rute perjalanan KRL. Penilaian aplikasi diukur dalam skala likert dimana nilai minimal yakni nilai 1 menunjukkan hasil sangat tidak baik dan nilai maksimal yakni nilai 5 menunjukkan hasil sangat baik. Pengguna dapat menilai dua fitur disini yakni fitur perencanaan rute dan fitur pencarian titik tujuan dan keberangkatan. Berikut hasil feedback oleh *beta tester* pada tabel 2.

Tabel 4. Pengujian beta perencanaan rute KRL

Beta Testing			
Kasus	**Nilai Penerimaan**	**Jumlah Responden**	**Rata-Rata Nilai**
Fitur pencarian titik keberangkatan dan tujuan	1	0	4.2
	2	0	
	3	3	
	4	10	
	5	7	
Fitur perencanaan rute dan penampil rute	1	0	4
	2	1	
	3	4	
	4	9	
	5	6	

Berdasarkan hasil *beta testing,* didapatkan hasil berupa nilai penerimaan pada setiap kasus penggunaan fitur. Didapatkan rata rata nilai Baik, untuk setiap fitur dengan nilai penerimaan tertinggi pada fitur pencarian titik. Hasil *beta testing* menunjukkan rata rata nilai penerimaan oleh pengguna yakni 4.1 menunjukkan bahwa kualitas penerimaan aplikasi mendapatkan skala "Baik" dan dapat diterima oleh pengguna.

5. KESIMPULAN DAN SARAN

Penelitian ini menghasilkan produk berupa aplikasi Android yang dapat digunakan oleh pengguna untuk merencanakan rute perjalanan KRL berdasarkan titik lokasi keberangkatan dan tujuan yang dipilih pengguna. Implementasi rumus Haversine dapat dengan baik menentukan stasiun terdekat dengan titik lokasi yang dipilih pengguna dan rute terbaik dan rute alternatif dapat diperoleh melalui implementasi algoritma Dijkstra. Sehingga pengguna dapat dengan mudah memilih rute yang ingin dilalui dari stasiun awal dekat keberangkatan dan stasiun akhir dekat dengan tujuan. Proses pengembangan aplikasi android dilakukan dengan metode Extreme Programming melalui tahapan perencanaan, perancangan, pengkodean, dan pengujian. Dimana pengembangan aplikasi android dilakukan dengan bahasa pemrograman Kotlin menggunakan Android Studio IDE. Hasil pengujian rumus Haversine dan algoritma Dijkstra menunjukkan bahwa keduanya lolos pada kasus pengujian mencari rute KRL dari titik (-6.213, 106.791) menuju titik (-6.228, 106.853) dengan rute KRL yang ditempuh yakni rute Stasiun Palmerah hingga Stasiun Tebet. Hasil pengujian alpha juga menunjukkan bahwa aplikasi sesuai dengan harapan berdasarkan beberapa kasus yang diuji sehingga aplikasi dapat digunakan pengguna dan hasil pengujian beta menunjukkan nilai 4.1 yang merupakan nilai yang masuk dalam skala "baik" dan dapat diterima oleh pengguna.

DAFTAR PUSTAKA

[1] S. Aminah, "Transportasi Publik dan Aksesibilitas Masyarakat Perkotaan," *Jurnal Teknik Sipil Universitas Bandar Lampung*, vol. 9, no. 1, pp. 1142–1155, 2018.

[2] A. S. A. Maharani, "Penumpang Commuterline Jabodetabek Tembus 22 Juta Orang," *Kompas*, Feb. 13, 2023. https://www.kompas.com/properti/read/2023/02/13/070000821/penumpang-commuterline-jabodetabek-tembus-22-juta-orang (accessed Apr. 07, 2023).

[3] PT Kereta Commuter Indonesia, "Annual Report 2021," 2021. Accessed: Aug. 14, 2023. [Online]. Available: https://commuterline.id/files/download/annual_report/Annual%20Report%202021.pdf

[4] C. A. Pamungkas, "APLIKASI PENGHITUNG JARAK KOORDINAT BERDASARKAN LATITUDE DAN LONGITUDE DENGAN METODE EUCLIDEAN DISTANCE DAN METODE HAVERSINE," *Jurnal Informa : Jurnal Penelitian dan Pengabdian Masyarakat*, vol. 5, no. 2, pp. 8–13, Aug. 2019, doi: 10.46808/INFORMA.V5I2.74.

[5] A. Serdano, M. Zarlis, and D. Hartama, "Perbandingan Algoritma Dijkstra dan Bellman-Ford Dalam Pencarian Jarak Terpendek Pada SPBU," *Seminar Nasional Sains & Teknologi Informasi (SENSASI)*, pp. 259–264, 2019.

[6] Reto. Meier and Ian. Lake, *Professional Android*. Canada: John Wiley & Sons, Incorporated, 2018. Accessed: Mar. 23, 2023. [Online]. Available: https://books.google.com/books/about/Professional_Android.html?id=Xu5qDwAAQBAJ

[7] S. S. Putro, D. R. Anamisa, and F. A. Mufarroha, *Algoritma Pemrograman*. Malang: Media Nusa Creative, 2019.

[8] R. Palupi, D. A. Yulianna, and S. S. Winarsih, "Analisa Perbandingan Rumus Haversine Dan Rumus Euclidean Berbasis Sistem Informasi

Geografis Menggunakan Metode Independent Sample t-Test," *JITU : Journal Informatic Technology And Communication*, vol. 5, no. 1, pp. 40–47, Jul. 2021, doi: 10.36596/jitu.v5i1.494.

[9] A. Akhtar, B. Bakhtawar, and S. Akhtar, "EXTREME PROGRAMMING VS SCRUM: A COMPARISON OF AGILE MODELS," *International Journal of Technology, Innovation and Management (IJTIM)*, vol. 2, no. 2, Oct. 2022, doi: 10.54489/ijtim.v2i2.77.

[10] H. Koç, A. M. Erdoğan, Y. Barjakly, and S. Peker, "UML Diagrams in Software Engineering Research: A Systematic Literature Review," MDPI AG, Mar. 2021, p. 13. doi: 10.3390/proceedings2021074013.

[11] Martin. Fowler, *UML Distilled- A Brief Guide to the Standard Object Modeling Language (3rd Edition)*. Boston: Pearson Education, Inc., 2003.

[12] R. S. Pressman, *Software Engineering: A Practitioner's Approach (5th Ed.)*, 5th ed. McGraw Hill, 2001