BAB IV
HASIL DAN PEMBAHASAN

4.1 Implementasi *Balanced K-Means* dan *Traveling Salesman Problem*

Aplikasi Pembuat *Itinerary* Wisata berbasis Android ini dibangun dengan menerapkan solusi *Traveling Salesman Problem* dan *Balanced K-Means Clustering*. Aplikasi menerima masukan dari pengguna yang berupa tanggal mulai berlibur, tanggal selesai berlibur, satu lokasi penginapan dan destinasi tujuan wisata yang akan dilakukan proses *clustering* berdasarkan jumlah hari yang sudah ditentukan pengguna. Hasil *clustering* menggunakan metode *Balanced K-Means* akan masuk ke proses penyelesaian TSP. Penyelesaian TSP akan dilakukan sebanyak jumlah hari berlibur dengan menggunakan algoritma *Held-Karp* yang secara kompleksifitas lebih baik dibandingkan *brute force*.

4.2 Batasan Implementasi

Aplikasi yang dibuat didistribusikan kedalam dua aplikasi, yaitu sisi *client* dan sisi *server*. Aplikasi dipisah menjadi dua dengan pertimbangan performansi aplikasi Android agar lebih stabil. Hal ini dikarenakan setiap ponsel Android memiliki spesifikasi *hardware* yang bervariasi, sehingga akan mempengaruhi durasi *runtime* algoritma yang diimplementasikan. Maka dari itu dibangun sebuah aplikasi *server* yang digunakan untuk melakukan komputasi algoritma berdasarkan masukan dari aplikasi *client* dan mengembalikkan nilai hasil komputasi algoritma ke aplikasi *client.*

Dalam implementasi ini, penelitian ini ditetapkan batasan berupa asumsi-asumsi sebagai berikut:

1. Pengguna telah menetapkan hari awal dan hari terakhir berlibur.
2. Jumlah destinasi wisata harus lebih besar atau samadengan dengan jumlah hari berlibur.
3. Lokasi awal berada dalam wilayah Daerah Istimewa Yogyakarta.
4. Lokasi awal ditetapkan tepat satu lokasi dalam pembuatan *itinerary.*
5. Destinasi wisata disediakan oleh data dari Google Maps API.
6. Menimbang kenyamanan dan durasi wisata pengguna, jumlah destinasi yang dapat dikunjungi setiap harinya sebanyak lima destinasi wisata. Pembatasan ini dilakukan karena diasumsikan dalam satu kali kunjungan, pengguna menghabiskan waktu dua jam.

7. Estimasi durasi perjalanan setiap harinya berdasarkan waktu antar rute setiap lokasi dan kunjungan lokasi.
8. Setiap kunjungan lokasi diasumsikan menghabiskan waktu dua jam.

4.3 Implementasi Aplikasi Pembuat *Itinerary Wisata*

Aplikasi yang dibuat didistribusikan ke dalam dua sisi, yaitu sisi *client* dan sisi *server*. Aplikasi dipisah menjadi dua dengan pertimbangan performansi aplikasi Android agar lebih stabil. Hal ini dikarenakan setiap ponsel Android memiliki spesifikasi *hardware* yang bervariasi, sehingga akan mempengaruhi waktu *runtime* algoritma yang diimplementasikan. Maka dari itu dibangun sebuah aplikasi *server* yang digunakan untuk melakukan komputasi algoritma berdasarkan masukkan dari aplikasi *client* dan mengembalikkan nilai hasil komputasi algoritma ke aplikasi *client.*

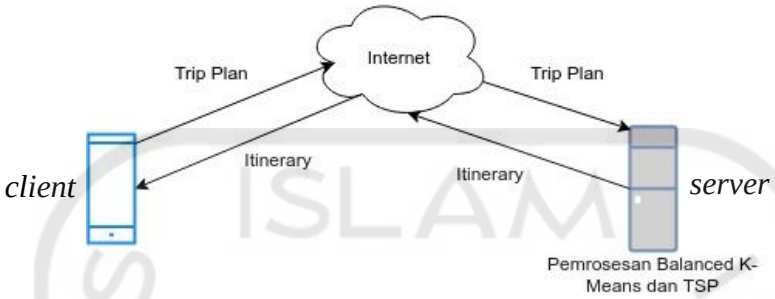
4.3.1 Implementasi Rest API

Aplikasi *server* dibuat untuk melakukan komputasi algoritma yang diterapkan pada aplikasi pembuat *itinerary* wisata. Hal ini merupakan solusi untuk menghindari ketidakstabilan performa ponsel dikarenakan setiap ponsel Android memiliki spesifikasi yang bervariasi. Penulis membangun aplikasi *server* ini berupa Rest API dengan Bahasa pemrograman Java dan menggunakan *json* dalam melakukan pertukaran data dengan aplikasi sisi *client*.

Rest API merupakan salah satu implementasi dari API yang menggunakan *protocol* HTTP untuk melakukan pertukaran data yang dapat berupa XML atau *json*. Fenomena pembuatan aplikasi dengan mendistribusikan kedalam dua sisi (*server* dan *client*) makin marak dalam dunia industry teknologi, karena diyakini sistem dapat memiliki performa yang lebih baik, lebih mudah untuk mengembangkan aplikasi dan menyelesaikan masalah interoperabilitas. Penulis membangun aplikasi *server* menggunakan *framework* Java Spring Boot dan Redis yang digunakan untuk menyimpan *cache* hasil data jarak yang didapat dari Google Maps API. Pemakaian *cache* digunakan untuk meminimalisir waktu *runtime* algoritma, dikarenakan dalam mengambil data dari Google Maps API menghabiskan banyak waktu pemrosesan. Sehingga hal ini dapat membuat waktu *runtime* benar-benar digunakan untuk melalukan operasi *Balanced K-Means* dan *Traveling Salesman Problem*.

Komunikasi yang digunakan antara *server* dan *client* aplikasi ini menggunakan format *json*. *Json* merupakan singkatan dari *JavaScript Object Notation* yang memiliki kemampuan

ringan dalam pertukaran data, mudah dibaca dan ditulis oleh manusia maupun oleh komputer. *Json* merupakan format teks yang tidak bergantung dengan bahasa pemrograman tertentu dan tersedia dalam berbagai bahasa pemrograman seperti C, C++, C#, PHP, Java, Python, Ruby meskipun format teks ini turunan dari Javascript.



Gambar 4.1 Ilustrasi Komunikasi *Client-Server*

Alur komunikasi antara aplikasi Android (*client*) dan *server* adalah aplikasi Android mengirimkan sebuah data *json* yang formatnya telah ditetapkan oleh aplikasi *server*. Format *json* yang telah ditetapkan adalah sebagai berikut.

```
{
   "destinations":[{
      "address":"string",
      "id":"string",
      "latLong":{
         "latitude":"double",
         "longitude":"double"
      },
      "name":"string",
      "photoReference":"string"
   }],
   "duration":"integer",
   "origin":{
      "address":"string",
      "id":"string",
      "latLong":{
         "latitude":"double",
         "longitude":"double"
      },
      "name":"string",
      "photoReference":"string"
   },
   "startDate":"date",
   "endDate":"date"
}
```

Gambar 4.2 Format *json* untuk masukkan ke server

Server menerima *json* dari aplikasi Android kemudian merubahnya kedalam bentuk objek dan dilakukan proses *clustering* dan *traveling salesman problem*. Hasil proses tersebut

akan dikembalikan ke pengirim (aplikasi Android) dalam bentuk *json* dan ditampilkan akan sesuai rancangan aplikasi Android. Format *json* yang membawa hasil pemrosesan dan akan diterima oleh aplikasi Android sebagai berikut.

```
{
   "destinations":[{
      "address":"string",
      "id":"string",
      "latLong":{
         "latitude":"double",
         "longitude":"double"
      },
      "name":"string",
      "photoReference":"string"
   }],
   "duration":"integer",
   "origin":{
      "address":"string",
      "id":"string",
      "latLong":{
         "latitude":"double",
         "longitude":"double"
      },
      "name":"string",
      "photoReference":"string"
   },
   "startDate":"date",
   "endDate":"date",
   "itineraryDetails":[
      {
         "day":"integer",
         "routes":[
            "origin":{
               "address":"string",
               "id":"string",
               "latLong":{
                  "latitude":"double",
                  "longitude":"double"
               },
               "name":"string",
               "photoReference":"string"

            },
            "destination":{
               "address":"string",
               "id":"string",
               "latLong":{
                  "latitude":"double",
                  "longitude":"double"
               },
               "name":"string",
               "photoReference":"string"
            },
            "tripDistance":"string",
            "tripDuration":"string"
         ]
      }
   ]
```

```
}
```

Gambar 4.3 Format *json* hasil dari pemrosesan pembuatan *itinerary*

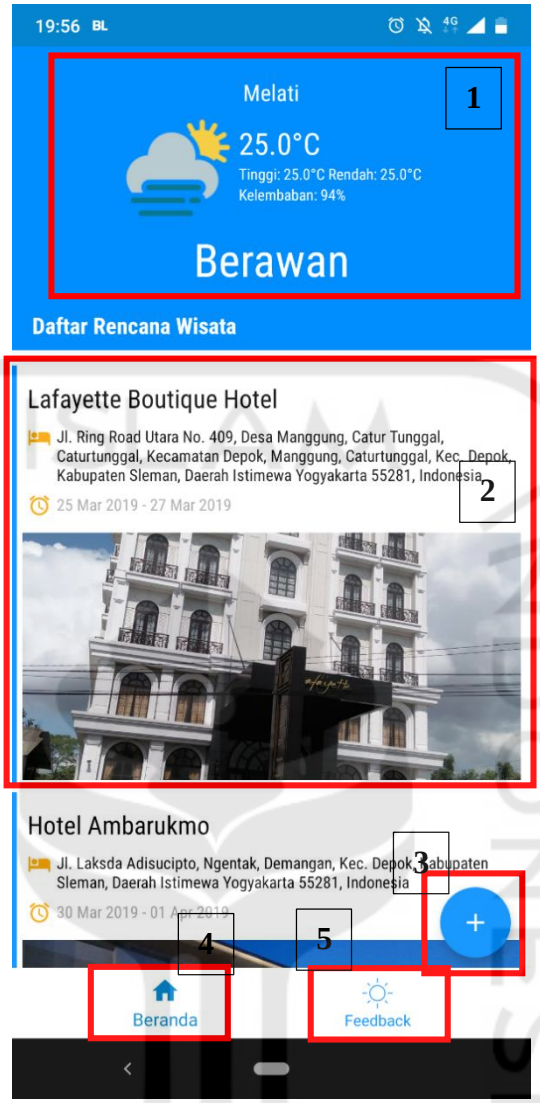
Dalam pengembangan aplikasi ini, sisi *client* akan mengirimkan data *Trip Plan* dan sisi *server* akan mengembalikan berupa data *Itinerary* seperti diilustrasikan pada Gambar 4.1. Sedangkan data *Trip Plan* dan *Itinerary* direpresentasikan sebagai *json* di atas.

4.3.2 Implementasi Aplikasi Android

a. ***Home Activity***

Activity Beranda merupakan halaman yang pertama kali dibuka oleh pengguna ketika memakai aplikasi pembuat *itinerary* ini. *Activity* ini menampilkan informasi cuaca pada lokasi pengguna dan daftar *itinerary* yang telah dibuat pengguna, serta beberapa tombol seperti yang terlihat pada Gambar 4.5.

Gambar 4.4 Antarmuka Awal *Home Activity* saat Pertama kali aplikasi dibuka



Gambar 4.5 Hasil Implementasi Antarmuka *Home Activity*

Bagian yang berada dalam kotak merah merupakan bagian yang dapat di pilih/di-klik. Setiap bagian memiliki fungsi yang berbeda-beda yang penulis jelaskan sebagai berikut:

1. Informasi Cuaca

Informasi cuaca disediakan oleh aplikasi dengan mengambil data cuaca dari api.openweathermap.com. Bagian ini menampilan cuaca saat ini di lokasi pengguna. Untuk mendapatkan informasi ini, aplikasi akan mendeteksi lokasi pengguna saat ini, kemudian mengambil data dari penyedia layanan cuaca berdasarkan *latitude* dan *logitude* lokasi pengguna. Informasi yang ditampilkan diantaranya lokasi saat ini, suhu, kelembabkan dan cuaca.

2. Itinerary Wisata

Bagian ini menampilkan daftar *itinerary* yang pernah dibuat oleh pengguna. *Itinerary* yang diolah oleh *server* kemudian akan disimpan oleh ponsel pengguna secara lokal dan ditampilkan dalam antarmuka seperti pada Gambar 4.11. Masing-masing daftar *itinerary* menampilkan nama lokasi awal, alamat lokasi awal, durasi berlibur dan foto lokasi awal. Selain durasi berlibur, data yang ditampilkan diambil dari Google Maps API. Setiap daftar dapat di-klik untuk melihat detail perjalanan wisata yang dibuat dan di *swipe* untuk menghapus data *itinerary* tersebut.

3. Tombol untuk Membuat Itinerary Wisata

Tombol ini akan memberikan aksi untuk menuju *activity* membuat *itinerary* baru.

4. Menu Beranda

Tombol menu ini, digunakan untuk membuka *activity* Beranda yang menampilkan halaman seperti pada Gambar 4.5.

5. Menu *Feedback*

Tombol menu ini, digunakan untuk membuka *activity Feedback* yang berisi halaman untuk mengisi kuesioner pengujian fungsionalitas.

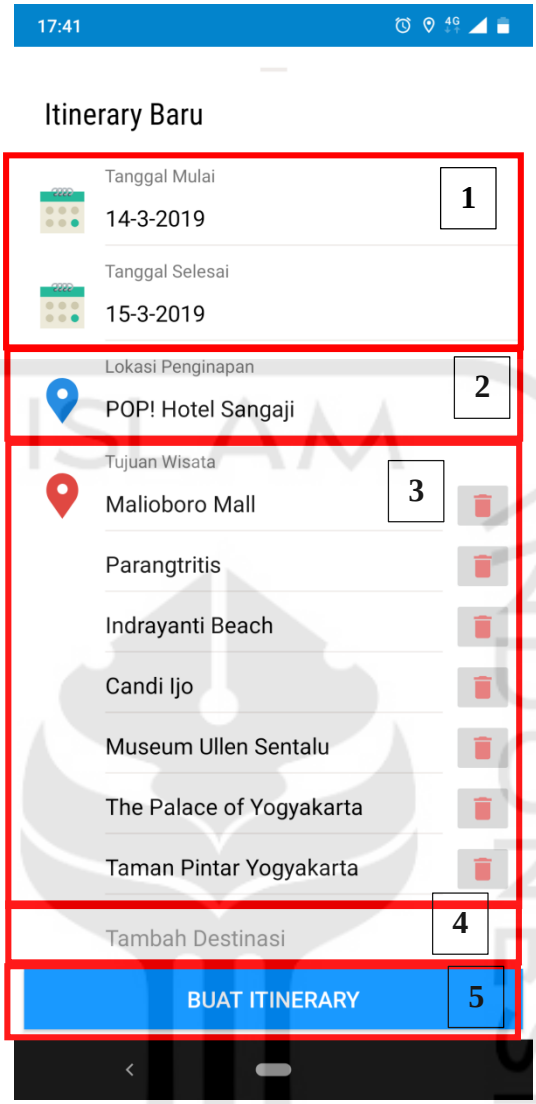
b. ***Itinerary Activity***

Activity ini akan tampil apabila pengguna menekan tombol Membuat Itinerary Wisata di activity Beranda. Activity ini menampilkan sebuah form untuk membuat itinerary wisata dan visualisasi *maps* yang diambil dari Google Maps API. Visualisasi *maps* disediakan agar pengguna dapat melihat lokasi awal dan destinasi wisata yang telah ia pilih.

Tampilan *activity* ini ditampilkan pada Gambar 4.6 dan penjelasan setiap bagian sebagai berikut:

1. Durasi berlibur

Pengguna memasukkan tanggal awal dan tanggal akhir berlibur. Tanggal yang dimaksud adalah tanggal dimulainya perjalanan wisata dimulai dari lokasi awal hingga urutan terakhir destinasi wisata yang telah direncanakan. Tanggal yang dimasukkan akan diubah menjadi durasi waktu (dalam hari) yang dijadikan *cluster* untuk menyelesaiakn permasalahan. Apabila pengguna memilih tanggal awal dan tanggal akhir yang sama, maka sistem akan mendeteksi, pengguna memiliki durasi libur satu hari.



Gambar 4.6 Hasil Implementasi *Itinerary Activity*

2. Lokasi Awal

Pengguna memilih lokasi awal untuk memulai perjalanan wisata dengan mengetikkan lokasi yang dikehendaki. Untuk memudahkan pencarian, aplikasi menyediakan fitur *autocomplete* saat pengguna mengetikkan lokasi seperti pada Gambar 4.7. Lokasi yang dipilih pengguna dapat berupa hotel, wisma, penginapan, atau tempat lainnya namun oleh data yang disediakan Google Maps API. Lokasi awal dapat diubah pengguna apabila terdapat kesalahan sebelum menekan tombol Buat Itinerary Wisata.

Gambar 4.7 Hasil Implementasi Halaman Pencarian Lokasi

3. Daftar Destinasi Wisata

Daftar destinasi wisata ini menampilkan destinasi yang dipilih oleh pengguna. Setiap destinasi dapat diubah dengan menekan daftar wisata yang ingin diubah, dan dihapus dengan menekan tombol dengan *icon* sampah pada sebelah kanan nama destinasi wisata.

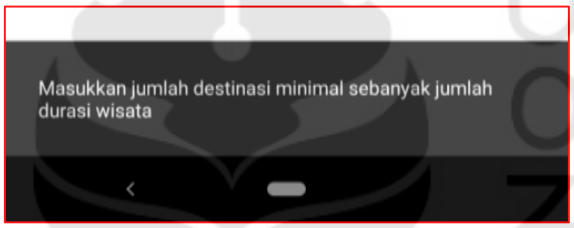
4. Tombol Tambah Destinasi

Tombol ini digunakan untuk menambah destinasi wisata yang akan dikunjungi. Apabila tombol ini ditekan oleh pengguna, maka akan muncul *activity* untuk pencarian lokasi yang akan dijadikan destinasi oleh pengguna. Lokasi yang tersedia untuk dipilih untuk destinasi wisata terbatas oleh data dari Google Maps API.

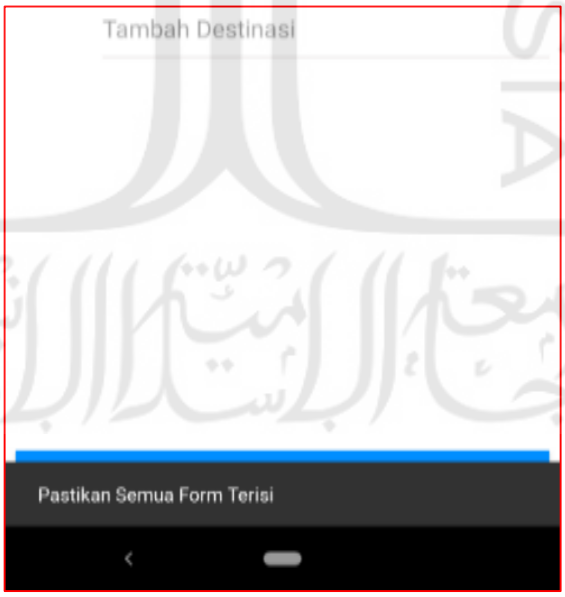
5. Tombol Buat Itinerary Wisata

Tombol Buat Itinerary Wisata akan mengirimkan data yang telah dimasukkan ke *server* untuk dilakukan proses pembuatan *itinerary*. Saat aplikasi menunggu respon dari server, aplikasi akan menampilkan animasi *progress bar/loading screen.* Tombol ini juga digunakan untuk mengecek apakah masukkan pengguna sudah valid atau belum, seperti pengecekan jumlah destinasi lebih dari atau sama dengan jumlah hari berlibur. Apabila kondisi tersebut tidak terpenuhi, akan muncul pesan *error* seperti Gambar 4.8. Adapun pesan *error* pada Gambar 4.9 akan muncul ketika pengguna tidak lengkap dalam mengisikan *form* pembuatan itinerary.

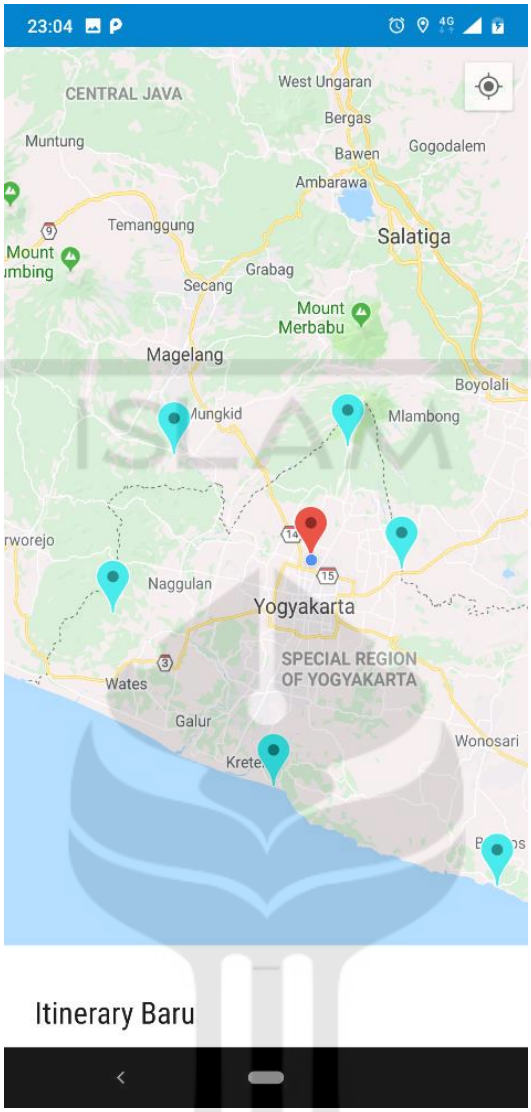
Adapun visualisasi *maps* setelah pengguna memasukkan lokasi awal ataupun destinasi wisata ditampilkan seperti pada Gambar 4.10.



Gambar 4.8 Notifikasi Pengguna kurang tepat dalam mengisi form



Gambar 4.9 Notifikasi Ketika *form* tidak lengkap



Gambar 4.10 Visualisasi *maps* pemilihan lokasi

c. ***Summary Activity***

Activity ini akan tampil ketika aplikasi berhasil membuat itinerary atau pengguna mengaksesnya dari daftar itinerary di Gambar 4.4. Pada *activity* ini berisi beberapa bagian sebagai berikut:

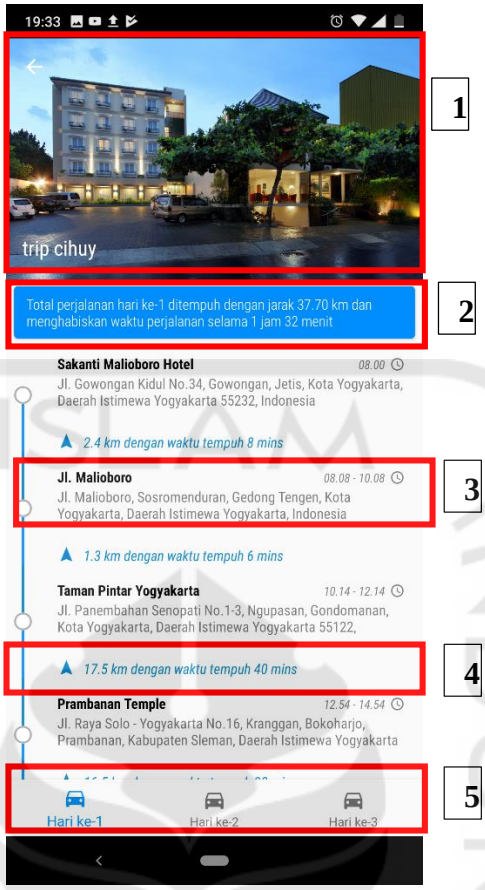
1. Foto dan Nama Lokasi awal atau lokasi penginapan yang dipilih pengguna.
2. Informasi total jarak perjalanan wisata pengguna.
3. Informasi total durasi waktu yang dihabiskan dalam menempuh seluruh destinasi wisata selama hari berlibur.
4. Tanggal mulai dan tanggal selesai berwisata.
5. Daftar destinasi wisata yang telah dipilih.
6. Tombol untuk membuka halaman *Detail Itinerary*.

Gambar 4.11 Implementasi Antarmuka *Summary Activity*

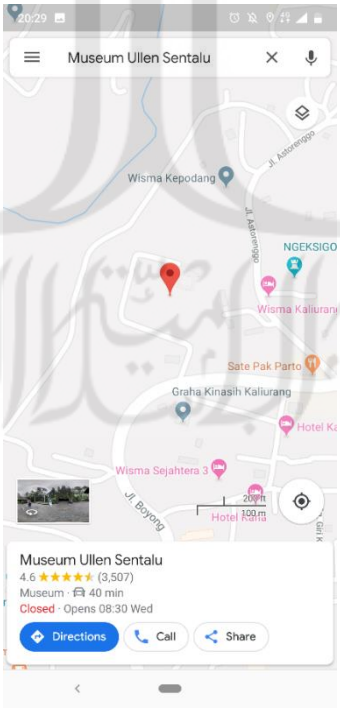
d. ***Detail Itinerary Activity***

Activity ini akan tampil ketika pengguna mengaksesnya dengan menekan tombol yang ada pada *Summary Activity*. Pada *activity* ini berisi beberapa bagian sebagai berikut:

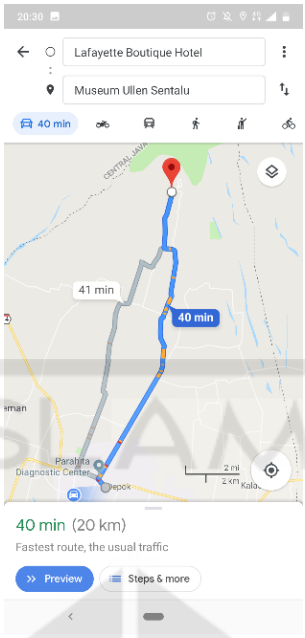
1. Foto Lokasi Awal dan *Nama itinerary.*
2. Detail tentang jarak total dan waktu tempuh perjalanan setiap harinya.
3. Nama dan alamat lokasi yang dipilih oleh pengguna. Apabila pengguna menekan pada nama atau alamat lokasi, aplikasi akan membuka Google Maps yang memberikan informasi mengenai lokasi yang dipilih Gambar 4.13.
4. Durasi waktu perjalanan dan jarak yang harus ditempuh antar lokasi. Apabila pengguna menekan pada bagian ini, aplikasi akan membuka Google Maps dan menjalankan navigasi dari lokasi awal dan lokasi akhir seperti pada Gambar 4.14.
5. Bagian kelima merupakan menu untuk melihat rekomendasi rute perjalanan setiap harinya.



Gambar 4.12 Implementasi *Detail Iitnerary Activity*

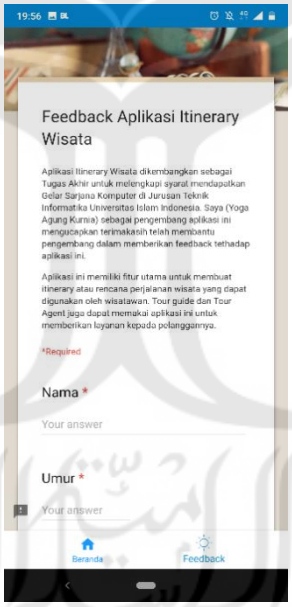


Gambar 4.13 Tampilan Informasi Lokasi pada Aplikasi Google Maps



Gambar 4.14 Tampilan Perjalanan pada Aplikasi Google Maps

e. ***Feedback Fragment***

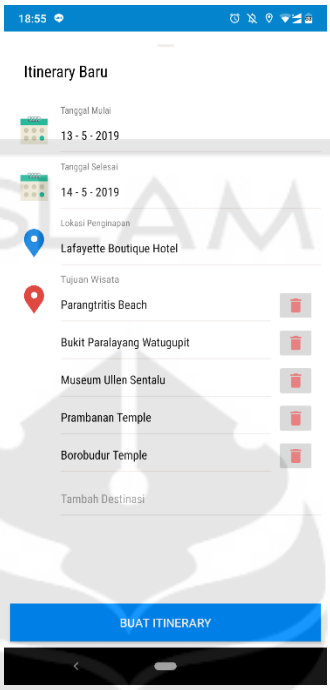


Gambar 4.15 Tampilan *Feedback Fragment*

Fragment ini akan digunakan pengguna untuk mengisi kuesioner *feedback* penggunaan aplikasi Itinerary Wisata yang bertujuan menguji fungsionalitas. Hasil pengujian ini digunakan untuk melihat fungsionalitas aplikasi ini kepada pengguna.

4.3.3 Skenario

Bagian skenario dibuat untuk memberikan gambaran proses kerja aplikasi pembuatan *itinerary* wisata agar dapat lebih dipahami. Berikut contoh masukan pada sistem dan gambaran proses terhadap masukan yang diberikan:



Gambar 4.16 Masukan pengguna

Dari masukan pengguna seperti pada Gambar 4.16, maka diketahui data masukan pengguna sebagai berikut:

1. Tanggal Mulai
   - 13 Mei 2019
2. Tanggal Selesai
   - 14 Mei 2019
3. Lokasi Awal
   - Hotel Lafayette Boutique (-7.7592524, 110.3852267)
4. Titik Lokasi Destinasi Wisata
   - Pantai Parangtritis (-8.010057, 110.313009)
   - Bukit Paralayang (-8.02685509, 110.3459122)
   - Museum Ullen Sentalu (-7.5978656,110.4233959)
   - Candi Prambanan (-7.751919, 110.492006)
   - Candi Borobudur (-7.607874, 110.203751)

Dari masukan pengguna pada aplikasi *itinerary* di atas, langkah awal proses adalah menentukan kelompok titik destinasi wisata menggunakan metode *balanced k-means* dimana jumlah hari sebagai *k*. Kemudian langkah selanjutnya, mencari rute dengan jarak terpendek antar titik wisata pada setiap *cluster*. Jarak antar titik wisata didapatkan dari data Google Maps. Didapatkan dari masukan pengguna di atas, daftar destinasi serta latitude dan longitude setiap lokasi tersebut sebagai berikut:

Tabel 4.1 Koordinat titik lokasi wisata

No.	Nama Tempat	Latitude	Longitude
1	Hotel Lafayette Boutique	-7.7592524	110.3852267
2	Pantai Parangtritis	-8.010057	110.313009
3	Bukit Paralayang	-8.02685509	110.3459122
4	Museum Ullen Sentalu	-7.5978656	110.4233959
5	Prambanan	-7.751919	110.492006
6	Candi Borobudur	-7.607874	110.203751

Proses *Clustering*

Iterasi Ke-1

Proses yang akan dijalankan pertama kali menggunakan *Balanced K-Means* untuk melakukan *clustering* berdasarkan lama hari yang dimasukkan oleh pengguna.

1. Jumlah *cluster* adalah 2.
2. *Centroid* awal setiap *cluster.*

   Centroid awal ditentukan dengan mengambil *k* data dari destinasi untuk dijadikan *centroid* setiap *clusternya* secara acak. Pada kasus ini yang dipilih dari Tabel 1 pada nomor 5 dan nomor 6. Di bawah merupakan nilai centroid pada iterasi ke-1:

   Tabel 4.2 Koordinat titik destinasi wisata

No.	*Centroid*	Latitude	Longitude
1	C1	-7.751919	110.492006
2	C2	-7.607874	110.203751

   Penentuan *centroid* baru dilakukan setelah seluruh data telah masuk kedalam *cluster*. Perhitungan *centroid* ditentukan lewat perhitungan rata-rata latitude dan longitude setiap data pada *cluster*. Apabila *centroid* baru sama dengan *centroid* sebelumnya, maka penentuan anggota *cluster* selesai.
3. Menghitung jarak setiap data terhadap *centroid*

   Perhitungan jarak menggunakan rumus *Euclidean distance* yang memiliki persamaan

sebagai berikut:

$$d_{ij} = \sqrt{\sum_{k=1}^{n}(x_{ik} - x_{jk})^2} \quad (1)$$

Keterangan:

d = Jarak

n = Jumlah data

j = Paramenter iterasi data

i = Paramenter iterasi terhadap *n*

c = Nilai *centroid*

x = Data

k = *Cluster*

Menghitung jarak setiap data terhadap *C1* (-7.751919, 110.492006):

A = (-8.010057, 110.313009)

$= \sqrt{(-7.751919 - (-8.010057))^2 + (110.492006 - 110.313009)^2}$

= 0.314126

B = (-8.02685509, 110.3459122)

$= \sqrt{(-7.751919 - (-8.02685509))^2 + (110.492006 - 110.3459122)^2}$

= 0.311341

C = (-7.5978656, 110.4233959)

$= \sqrt{(-7.751919 - (-7.5978656))^2 + (110.492006 - 110.4233959)^2}$

= 0.168641

D = (-7.751919, 110.492006)

$= \sqrt{(-7.751919 - (-7.751919))^2 + (110.492006 - 110.492006)^2}$

= 0

E = (-7.607874, 110.203751)

$= \sqrt{(-7.751919 - (-7.607874))^2 + (110.492006 - 110.203751)^2}$

= 0.322242

Menghitung jarak setiap data terhadap *C2* (-7.607874, 110.203751):

A = (-8.010057, 110.313009)

$= \sqrt{(-7.607874 - (-8.010057))^2 + (110.203751 - 110.313009)^2}$

= 0.416759

B = (-8.02685509, 110.3459122)

$= \sqrt{(-7.607874 - (-8.02685509))^2 + (110.203751 - 110.3459122)^2}$

= 0.442442

C = (-7.5978656, 110.4233959)

$= \sqrt{((-7.607874 - (-7.5978656))^2 + (110.203751 - 110.4233959)^2}$

= 0.219873

D = (-7.751919, 110.492006)

$= \sqrt{(-7.607874 - (-7.751919))^2 + (110.203751 - 110.492006)^2}$

= 0.322242

E = (-7.607874, 110.203751)

$= \sqrt{(-7.607874 - (-7.607874))^2 + (110.203751 - 110.203751)^2}$

= 0

Dari hasil perhitungan jarak dari *centroid* ke setiap destinasi, didapatkan hasil seperti pada tabel di bawah:

Tabel 4.3 Hasil perhitungan jarak antara *centroid*

No.	Nama Tempat	Centroid 1	Centroid 2
1	Pantai Parangtritis	0.314126	0.416759
2	Bukit Paralayang	0.311341	0.442442
3	Museum Ullen Sentalu	0.168641	0.219873
4	Prambanan	0	0.322242
5	Candi Borobudur	0.322242	0

4. Mengelompokan data perhitungan ke dalam *cluster*

Dalam mengelompokan data kedalam *cluster,* akan dilakukan operasi *Hungarian Algorithm* agar jumlah anggota pada *cluster* dapat seimbang. Syarat melakukan *hungarian algorithm* adalah ukuran baris dan kolom harus sama. Pada kasus ini,

untuk melakukan *Hungarian algorithm*, lokasi akan diijadikan sebagai baris, dan *cluster* sebagai kolom. Karena jumlah kolom dan baris tidak seimbang, maka jumlah kolom akan ditentukan dari hasil perkalian 2 (jumlah *cluster*) hingga jumlah kolom lebih dari jumlah baris (5), sehingga ukuran tabel menjadi 5 (baris) x 6 (kolom). Algoritma belum bisa dilakukan karena jumlah baris kurang dari jumlah kolom, sehingga akan ditambahkan baris data palsu yang memiliki nilai 0 pada baris paling akhir tabel.

Tabel 4.4 Koordinat nilai minimum setiap baris

	C1	C2	C1	C2	C1	C2	Minim
1	0	0.102633	0	0.102633	0	0.102633	0.314126
2	0	0.131101	0	0.131101	0	0.131101	0.311341
3	0	0.051232	0	0.051232	0	0.051232	0.168641
4	0	0.322242	0	0.322242	0	0.322242	0
5	0.322242	0	0.322242	0	0.322242	0	0
6	0	0	0	0	0	0	0

Tabel 4.5 Mencari nilai minimum setiap kolom

	C1	C2	C1	C2	C1	C2
1	0	0.102633	0	0.102633	0	0.102633
2	0	0.131101	0	0.131101	0	0.131101
3	0	0.051232	0	0.051232	0	0.051232
4	0	0.322242	0	0.322242	0	0.322242
5	0.322242	0	0.322242	0	0.322242	0
6	0	0	0	0	0	0
Minim	0	0	0	0	0	0

Langkah pertama dalam operasi algoritma ini, adalah menentukan nilai minimum setiap baris tabel, kemudian lakukan pengurangan nilai kolom, pada setiap baris dengan nilai minimum yang sudah ditemukan. Hasil pengurangan terdapat pada tabel berikut:

Tabel 4.6 Hasil pengurangan dan penandaan angka 0

	C1	C2	C1	C2	C1	C2
1	0	0.102633	0	0.102633	0	0.102633
2	0	0.131101	0	0.131101	0	0.131101
3	0	0.051232	0	0.051232	0	0.051232
4	0	0.322242	0	0.322242	0	0.322242
5	0.322242	0	0.322242	0	0.322242	0
6	0	0	0	0	0	0

Langkah selanjutnya adalah menandai nilai 0 pada setiap baris atau kolom tepat satu kali dan tandai baris atau kolom tegak lurus dengan nilai 0 yang ditandai (arsir warna biru muda), yang manandakan baris tersebut memiliki hubungan dengan kolom tersebut. Pada table di atas terdapat satu baris (baris 4) yang tidak memiliki angka 0, maka langkah berikutnya adalah mencari nilai terkecil dari data tidak diarsir yang mana pada table di atas memiliki latar putih.

Tabel 4.7 Hasil pencarian nilai minimum data tidak terarsir

	C1	C2	C1	C2	C1	C2
1	0	0.102633	0	0.102633	0	0.102633
2	0	0.131101	0	0.131101	0	0.131101
3	0	0.051232	0	0.051232	0	0.051232
4	0	0.322242	0	0.322242	0	0.322242
5	0.322242	0	0.322242	0	0.322242	0
6	0	0	0	0	0	0

Diketahui nilai terkecil pada bagian tidak terarsir adalah 0.51232 yang mana pada table di atas diberikan warna latar oranye. Kemudian, gunakan nilai terkecil (0.51232) untuk mengurangi nilai pada setiap data yang memiliki latar putih, dan tambahkan pada nilai yang diarsir dua kali (warna hijau).

Tabel 4.8 Hasil operasi penambahan dan pengurangan

	C1	C2	C1	C2	C1	C2
1	0	0.102633	0	0.102633	0	0.051401
2	0	0.131101	0	0.131101	0	0.07986
3	0	0.051232	0	0.051232	0	0
4	0	0.322242	0	0.322242	0	0.27101
5	0.373474	0.051232	0.373474	0.051232	0.373474	0
6	0.051232	0.051232	0.051232	0.051232	0.051232	0

Lakukan pengurangan nilai minimum setiap kolom dan baris agar mendapatkan minimal satu 0 pada setiap baris dan kolom tersebut.

Tabel 4.9 Mencari nilai minimum setiap baris

	C1	C2	C1	C2	C1	C2	Minim
1	0	0.102633	0	0.102633	0	0.051401	0
2	0	0.131101	0	0.131101	0	0.07986	0
3	0	0.051232	0	0.051232	0	0	0
4	0	0.322242	0	0.322242	0	0.27101	0

5	0.373474	0.051232	0.373474	0.051232	0.373474	0	0
6	0.051232	0.051232	0.051232	0.051232	0.051232	0	0

Tabel 4.10 Mencari nilai minimum setiap kolom

	C1	**C2**	**C1**	**C2**	**C1**	**C2**
1	0	0.05140	0	0.05140	0	0.051401
2	0	0.07986	0	0.07986	0	0.07986
3	0	0	0	0	0	0
4	0	0.27101	0	0.27101	0	0.27101
5	0.373474	0	0.373474	0	0.373474	0
6	0.051232	0	0.051232	0	0.051232	0
Minim	0	0.051232	0	0.051232	0	0

Nilai minimum setiap baris dan nilai kolom tabel, pada setiap baris dengan nilai minimum yang sudah ditemukan. Hasil pengurangan terdapat pada tabel berikut:

Tabel 4.11 Hasil pengurangan dan penandaan angka 0

	C1	**C2**	**C1**	**C2**	**C1**	**C2**
1	0	0.05140	0	0.05140	0	0.051401
2	0	0.07986	0	0.07986	0	0.07986
3	0	0	0	0	0	0
4	0	0.27101	0	0.27101	0	0.27101
5	0.373474	0	0.373474	0	0.373474	0
6	0.051232	0	0.051232	0	0.051232	0

Setiap baris dan kolom memiliki nilai minimum 0, sehingga tidak ada yang berubah dari setiap baris tabel di atas. Proses *Hungarian algorithm* selesai ketika semua baris memiliki pasangan dengan kolom, seperti pada Tabel 5. Langkah selanjutnya adalah menghitung *centroid* baru, apabila ada data pada salah satu *cluster* berpindah, maka akan dilakukan perhitungan jarak dengan *Euclidean distance* dan melakukan operasi *hungarian algorithm* hingga tidak ada data yang berpindah *cluster*.

Menghitung *centroid* 1 dilakukan dengan mengambil nilai rata rata *latitude* dan *longitude* dari data *cluster* 1.

$$Latitude = \frac{(-8.010057+(-8.02685509))+(-7.751919)}{3} = -7.92961$$

$$Longitude = \frac{(110.313009+110.3459122+110.492006)}{3} = 110.3836$$

Menghitung *centroid* 2 dilakukan dengan mengambil nilai rata rata *latitude* dan *longitude* dari data *cluster* 2.

Latitude = $\frac{(-7.5978656+(-7.751919)+(-7.607874))}{2}$ = -7.60286

Longitude = $\frac{(110.4233959+110.203751)}{2}$ = 110.3135

Berikut merupakan hasil akhir untuk *clustering* dalam pembuatan *itinerary* wisata iterasi 1:

Tabel 4.12 Hasil Akhir *Clustering* Iterasi ke-1

No.	Nama Tempat	Hari ke-1	Hari ke-2
1	Pantai Parangtritis	x	
2	Bukit Paralayang	x	
3	Museum Ullen Sentalu		x
4	Prambanan	x	
5	Candi Borobudur		x

Iterasi 2

1. Menghitung jarak setiap data terhadap *centroid*

   Menghitung jarak setiap data terhadap *C1* (-7. 92961, 110. 3836):

   A = (-8.010057, 110.313009)

   = $\sqrt{(-7.92961-(-8.010057))^2+(110.3836-110.313009)^2}$

   = 0.107027

   B = (-8.02685509, 110.3459122)

   = $\sqrt{(-7.92961-(-8.02685509))^2+(110.3836-110.3459122)^2}$

   = 0.104293

   C = (-7.5978656, 110.4233959)

   = $\sqrt{(-7.92961-(-7.5978656))^2+(110.3836-110.4233959)^2}$

   = 0.334123

   D = (-7.751919, 110.492006)

   = $\sqrt{(-7.92961-(-7.751919))^2+(110.3836-110.492006)^2}$

   = 0.208149

   E = (-7.607874, 110.203751)

   = $\sqrt{(-7.92961-(-7.607874))^2+(110.3836-110.203751)^2}$

   = 0.368592

Menghitung jarak setiap data terhadap *C2* (-7.60286, 110. 3135):

A = (-8.010057, 110.313009)

$= \sqrt{(-7.60286 - (-8.010057))^2 + (110.3135 - 110.313009)^2}$

= 0.407197

B = (-8.02685509, 110.3459122)

$= \sqrt{(-7.60286 - (-8.02685509))^2 + (110.3135 - 110.3459122)^2}$

= 0.425232

C = (-7.5978656, 110.4233959)

$= \sqrt{((-7.60286 - (-7.5978656))^2 + (110.3135 - 110.4233959)^2}$

= 0.110009

D = (-7.751919, 110.492006)

$= \sqrt{(-7.60286 - (-7.751919))^2 + (110.3135 - 110.492006)^2}$

= 0.232557

E = (-7.607874, 110.203751)

$= \sqrt{(-7.60286 - (-7.607874))^2 + (110.3135 - 110.203751)^2}$

= 0.109863

Dari hasil perhitungan jarak dari *centroid* ke setiap destinasi, didapatkan hasil seperti pada tabel di bawah:

Tabel 4.13 Hasil perhitungan jarak antara *centroid*

No.	Nama Tempat	Centroid 1	Centroid 2
1	Pantai Parangtritis	0.107027	0.407197
2	Bukit Paralayang	0.104293	0.425232
3	Museum Ullen Sentalu	0.334123	0.110009
4	Prambanan	0.208149	0.232557
5	Candi Borobudur	0.368592	0.109863

2. Mengelompokan data perhitungan ke dalam *cluster*

Tabel 4.14 Mencari nilai minimum setiap baris

	C1	C2	C1	C2	C1	C2	Minim
1	0	0.30017	0	0.30017	0	0.30017	0.107027
2	0	0.320939	0	0.320939	0	0.320939	0.104293
3	0.224114	0	0.224114	0	0.224114	0	0.110009
4	0	0.024408	0	0.024408	0	0.024408	0.208149
5	0.258729	0	0.258729	0	0.258729	0	0.109863
6	0	0	0	0	0	0	0

Tabel 4.15 Mencari nilai minimum setiap kolom

	C1	C2	C1	C2	C1	C2
1	0	0.30017	0	0.30017	0	0.30017
2	0	0.320939	0	0.320939	0	0.320939
3	0.224114	0	0.224114	0	0.224114	0
4	0	0.024408	0	0.024408	0	0.024408
5	0.258729	0	0.258729	0	0.258729	0
6	0	0	0	0	0	0
Minim	0	0	0	0	0	0

Menentukan nilai minimum setiap baris tabel, kemudian lakukan pengurangan nilai kolom, pada setiap baris dengan nilai minimum yang sudah ditemukan. Hasil pengurangan terdapat pada tabel berikut:

Tabel 4.16 Hasil pengurangan dan penandaan angka 0

	C1	C2	C1	C2	C1	C2
1	0	0.30017	0	0.30017	0	0.30017
2	0	0.320939	0	0.320939	0	0.320939
3	0.224114	0	0.224114	0	0.224114	0
4	0	0.024408	0	0.024408	0	0.024408
5	0.258729	0	0.258729	0	0.258729	0
6	0	0	0	0	0	0

Setiap baris dan kolom memiliki nilai minimum 0, sehingga tidak ada yang berubah dari setiap baris tabel di atas. Proses *Hungarian algorithm* selesai ketika semua baris memiliki pasangan dengan kolom, seperti pada Tabel 16. Langkah selanjutnya adalah menghitung *centroid* baru, apabila ada data pada salah satu *cluster* berpindah, maka akan

dilakukan perhitungan jarak dengan *Euclidean distance* dan melakukan operasi *hungarian algorithm* hingga tidak ada data yang berpindah *cluster*.

Menghitung *centroid* 1 dilakukan dengan mengambil nilai rata rata *latitude* dan *longitude* dari data *cluster* 1.

$$Latitude = \frac{(-8.010057+(-8.02685509))+(-7.751919)}{3} = -7.92961$$

$$Longitude = \frac{(110.313009+110.3459122+110.492006)}{3} = 110.3836$$

Menghitung *centroid* 2 dilakukan dengan mengambil nilai rata rata *latitude* dan *longitude* dari data *cluster* 2.

$$Latitude = \frac{(-7.5978656+(-7.607874))}{2} = -7.60286$$

$$Longitude = \frac{(110.4233959+110.203751)}{2} = 110.3135$$

Berikut merupakan hasil akhir untuk *clustering* dalam pembuatan *itinerary* wisata iterasi 2:

Tabel 4.17 Hasil Akhir *Clustering* Iterasi ke-1

No.	Nama Tempat	Hari ke-1	Hari ke-2
1	Pantai Parangtritis	x	
2	Bukit Paralayang	x	
3	Museum Ullen Sentalu		x
4	Prambanan	x	
5	Candi Borobudur		x

Tidak ada perubahan data antara *cluster* pada iterasi ini dengan iterasi sebelumnya, sehingga proses *clustering* selesai.

Proses *Traveling Salesman Problem*

Untuk menyelesaikan proses TSP, pertama aplikasi akan meminta data jarak masing-masing titik wisata dari Google Maps API. Kemudian akan dicari permutasi untuk mendapatkan rute terbaik menggunakan algoritma *Held-Karp*. Berikut proses TSP pada masing-masing hari untuk studi kasus pembuatan *itinerary* ini:

1. Hari ke-1

Tabel 4.18 Data jarak antar lokasi hari ke-1

No.	Lokasi Awal	Lokasi Tujuan	Jarak
1	Hotel Lafayette	Pantai Parangtritis	35 KM
2	Hotel Lafayette	Bukit Paralayang	36 KM
3	Hotel Lafayette	Prambanan	16 KM
4	Pantai Parangtritis	Bukit Paralayang	2,3 KM
5	Pantai Parangtritis	Hotel Lafayette	35 KM
6	Pantai Parangtritis	Prambanan	44 KM
7	Bukit Paralayang	Pantai Parangtritis	2,3 KM
8	Bukit Paralayang	Hotel Lafayette	38 KM
8	Bukit Paralayang	Prambanan	47 KM
9	Prambanan	Parangtritis	44 KM
10	Prambanan	Bukit Paralayang	47 KM
11	Prambanan	Hotel Lafayette	16 KM

Langkah pertama untuk menyelesaikan TSP dengan memodelkan data jarak tabel di atas kedalam *adjacency matrix* seperti pada tabel di bawah.

Tabel 4.19 *Adjacency Matrix* data jarak hari ke-1

Nama Lokasi	H. Lafayette	P. Parangtritis	B. Paralayang	Prambanan
H. Lafayette	0	35	36	16
P. Parangtritis	35	0	2,3	44
B. Paralayang	38	2,3	0	47
Prambanan	16	44	47	0

Langkah selanjutnya mencari rute dengan menelusuri kemungkinan rute dari titik akhir hingga titik awal. Titik awal dan titik akhir disimbolkan dengan Φ.

- Φ, 1 = 35
- Φ, 2 = 38
- Φ, 3 = 16
- 2, {1} = 35 + 2,3
  = 38,3
- 3, {1} = 35 + 44

= 79

- 1, {2} = 38 + 2,3
  = 40,3
- 3, {2} = 38 + 47
  = 85
- 1, {3} = 16 + 44
  = 60
- 2, {3} = 16 + 47
  = 63
- 3, {1,2} = $\min((3,2,1,\Phi),(3,1,2,\Phi))$
  = $\min((47+38,3),(44+40,3))$
  = $\min((85,3),(84,3))$
  = 84,3
- 2, {1,3} = $\min((2,1,3,\Phi),(2,3,1,\Phi))$
  = $\min((2,3+60),(47+79))$
  = $\min((62,3),(126))$
  = 62,3
- 1, {2,3} = $\min((1,2,3,\Phi),(1,3,2,\Phi))$
  = $\min((2,3+63),(44+85))$
  = $\min((65,3),(129))$
  = 65,3
- Φ, {1,2,3} = $\min((\Phi+\{3,1,2\}),(\Phi+\{2,1,3\}),(\Phi+\{1,2,3\}))$
  = $\min((16+84,3),(38+62,3),(35+65,3))$
  = $\min((100,3),(100,3),100,3)$
  = 100,3

Berdasarkan operasi penyelesaian *Traveling Salesman Problem* di atas, total jarak yang ditempuh adalah 100,3 km, dengan 3 alternatif urutan kunjungan sebagai berikut:

- Hotel Lafayette – Prambanan – Pantai Parangtritis – Bukit Paralayang – Hotel Lafayette
- Hotel Lafayette – Prambanan – Bukit Paralayang – Pantai Parangtritis – Hotel Lafayette

- Hotel Lafayette – Bukit Paralayang – Pantai Parangtritis – Prambanan – Hotel Lafayette

Karena memiliki lebih dari satu solusi, penulis memilih salah satu urutan rute dari alternatif solusi yang diberikan untuk dijadikan hasil akhir pencarian urutan.

Hasil urutan rute untuk *itinerary* hari ke-1:

a. Hotel Lafayette menuju Prambanan = 16 KM
b. Prambanan menuju Pantai Parangtritis = 44 KM
c. Pantai Parangtritis menuju Bukit Paralayang = 2,3 KM
d. Bukit Paralayang menuju Hotel Lafayette = 38 KM

Total jarak perjalanan hari ke-1 = 100,3 KM

2. Hari ke-2

Tabel 4.20 Data jarak antar lokasi hari ke-2

No.	Lokasi Awal	Lokasi Tujuan	Jarak
1	Hotel Lafayette	Museum Ullen Sentalu	21 KM
2	Hotel Lafayette	Candi Borobudur	38 KM
4	Museum Ullen Sentalu	Hotel Lafayette	21 KM
5	Museum Ullen Sentalu	Candi Borobudur	39 KM
7	Candi Borobudur	Museum Ullen Sentalu	39 KM
8	Candi Borobudur	Hotel Lafayette	38 KM

Kemudian memodelkan data jarak tabel di atas kedalam *adjacency matrix* seperti pada tabel di bawah.

Tabel 4.21 *Adjacency Matrix* data jarak hari ke-2

Nama Lokasi	H. Lafayette	Ullen Sentalu	C. Borobudur
H. Lafayette	0	21	38
Ullen Sentalu	21	0	39
C. Borobudur	38	39	0

Langkah selanjutnya mencari rute dengan menelusuri kemungkinan rute dari titik akhir hingga titik awal. Titik awal dan titik akhir disimbolkan dengan Φ.

- Φ, 1 = 21
- Φ, 2 = 38
- 2, {1} = 21 + 39
  = 60
- 1, {2} = 38 + 39

$= 77$

- $\Phi, \{1,2\}$ $= \min((\Phi + \{2,1\}), (\Phi + \{1,2\}))$

  $= \min((38 + 60), (21 + 77))$

  $= \min((98), (98))$

  $= 98$

Berdasarkan operasi penyelesaian *Traveling Salesman Problem* di atas, total jarak yang ditempuh adalah 98 km, dengan 2 alternatif urutan kunjungan sebagai berikut:

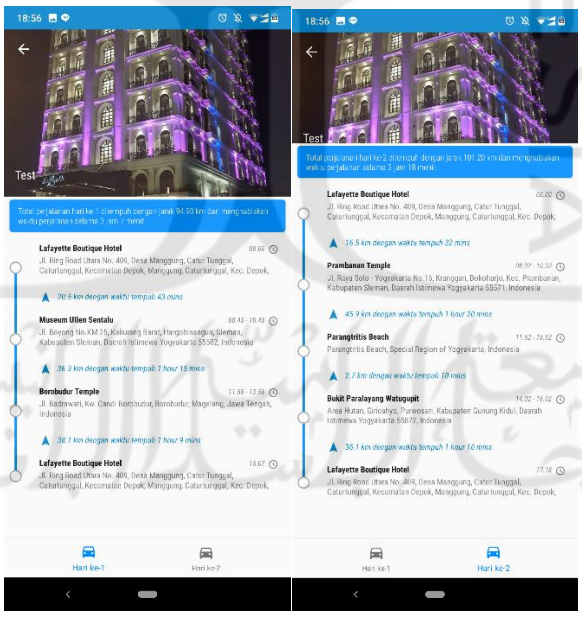
- Hotel Lafayette – Ullen Sentalu – Candi Borobudur – Hotel Lafayette
- Hotel Lafayette – Candi Borobudur – Ullen Sentalu – Hotel Lafayette

Karena memiliki lebih dari satu solusi, penulis memilih salah satu urutan rute dari alternatif solusi yang diberikan untuk dijadikan hasil akhir pencarian urutan.

Hasil Permutasi untuk *itinerary* hari ke-2:

a. Hotel Lafayette menuju Ullen Sentalu = 21 KM
b. Ullen Sentalu menuju Candi Borobudur = 39 KM
c. Candi Borobudur menuju Hotel Lafayette = 38 KM

Total jarak perjalanan hari ke-1 = 98 KM



Gambar 4.17 Hasil akhir *itinerary* yang dibuat oleh sistem

Hasil akhir yang dibuat oleh sistem, memiliki hasil yang sama dengan penjelasan skenario proses *clustering* dan TSP di atas. Perbedaan pada penempatan hari dapat diabaikan,

hal ini dikarenakan pemilihan *centroid* awal pada sistem dan penjelasan skenario proses berbeda.

4.4 Pengujian

4.4.1 Hasil Pengujian Fungsionalitas

Pengujian fungsionalitas dilakukan menggunakan beberapa ponsel Android yaitu Xiaomi Mi4 (P1), Nokia 6.1 Plus (P2) dan Samsung Galaxy S9 (P3). Ponsel P1 memiliki spesifikasi *processor* Snapdragon 801 keluaran tahun 2014 dengan RAM 2GB, ponsel P2 memiliki spesifikasi *processor* Snapdragon 636 keluaran tahun 2017 dengan RAM 4GB, dan ponsel P3 memiliki spesifikasi *processor* Exynos 9810 keluaran tahun 2018 dengan RAM 4GB. Ketika ponsel tersebut masing-masing memiliki spesfikasi *low-end, mid-end* dan *high-end* pada saat penelitian ini dibuat. Berikut hasil dari pengujian fungsionalitas yang telah dilakukan penulis masukan ke dalam Tabel 4.22.

Tabel 4.22 Pengujian Fungsionalitas

No	Fungsionalitas	Aktifitas	Hasil yang diharapkan	Hasil		
				P1	P2	P3
1.	Buat *itinerary* wisata	Memilih tombol “Buat Itinerary” berbentuk lingkaran pada pojok kanan bawah *Home Activity*. Swipe up pada halaman *Home Activity*. Menekan tombol “Pilih tanggal mulai” dan “Pilih tanggal mulai” kemudian memilih tanggal yang tersedia. Menekan tombol “Pilih tanggal mulai” atau “Pilih tanggal mulai” dan memilih tanggal yang tersedia. Menekan tombol “Pilih lokasi menginap”, aplikasi akan mengarah ke *Place Search Activity*¸ kemudian memasukkan nama lokasi. Menekan tombol “Pilih lokasi menginap”, aplikasi akan mengarah ke *Place Search Activity*¸ kemudian memasukkan nama lokasi. Menekan tombol “Buat Itinerary Wisata”.	Aplikasi dapat membuat rekomendasi *itinerary* wisata dengan solusi yang optimal.	Berhasil	Berhasil	Berhasil
2.	Deteksi lokasi pengguna	Mengaktifkan izin akses lokasi ponsel pengguna	Aplikasi dapat menampilkan	Berhasil	Berhasil	Berhasil

			penanda lokasi pengguna saat ini pada maps.			
3.	Lihat ringkasan *itinerary* yang pernah dibuat	Memilih daftar *itinerary* yang pernah dibuat pada *Home Activity*	Aplikasi dapat menampilkan lokasi penginapan, ringkasan durasi jarak dan waktu yang harus ditempuh dan daftar lokasi wisata	Berhasil	Berhasil	Berhasil
4.	Lihat detail *itinerary* yang pernah dibuat	Menekan tombol Lihat Itinerary Wisata pada *activity Summary*	Aplikasi dapat menampilkan *timeline* perjalanan wisata yang pernah atau akan dilakukan.	Berhasil	Berhasil	Berhasil
5.	Lihat informasi tempat wisata	Memilih daftar *itinerary* yang pernah dibuat pada *Home Activity*, kemudian memilih lokasi yang akan dilihat informasinya pada halaman *Itinerary Activity*.	Aplikasi dapat membuka aplikasi Google Maps untuk mendapatkan informasi tentang lokasi yang dipilih.	Berhasil	Berhasil	Berhasil
6.	Lihat informasi perjalanan antara lokasi	Memilih daftar *itinerary* yang pernah dibuat pada *Home Activity*, kemudian memilih perjalanan antar lokasi yang akan dilihat informasinya pada halaman *Itinerary Activity*.	Aplikasi dapat membuka aplikasi Google Maps untuk menjalankan navigasi perjalanan yang dipilih.	Berhasil	Berhasil	Berhasil

Berdasarkan pada hasil tabel di atas, ketiga ponsel berhasil melewati seluruh poin uji fungsionalitas, namun pada saat pengujian terdapat beberapa keterangan sebagai berikut.

1. Pada poin uji nomor 1, P1 mengalami *delay* dalam menampilkan visualisasi *maps* ketika memasukkan data *tripplan*. Untuk ponsel P2 dan P3 tidak ada kendala pada poin uji fungsionalitas nomor 1.
2. Pada poin uji nomor 5 dan nomor 6, P1 tidak dapat membuka aplikasi Google Maps dikarenakan ponsel tidak memiliki aplikasi tersebut. Namun pengguna diarahkan ke *browser* menuju halaman *maps.google.com* dan menampilkan informasi yang sesuai dengan hasil yang diharapkan pada poin uji nomor 5 dan nomor 6.

Kesimpulan yang dapat diambil pada hasil pengujian fungsionalitas adalah aplikasi dapat berjalan dengan baik menggunakan ponsel Android yang memiliki spesifikasi rendah, sedang dan tinggi, serta memiliki kompabilitas yang cukup luas terhadap spesifikasi ponsel Android yang bervariasi.

4.4.2 Hasil Pengujian UAT (*User Acceptance Test*)

Pengujian UAT diberikan kepada para responden yang sering berlibur, *tour agent* dan *tour agent*. Para responden akan diminta untuk menjalankan aplikasi Itinerary Wisata, membuat rencana *itinerary* dan mencoba fitur-fitur yang tersedia tanpa skenario khusus. Kemudian responden diminta untuk mengisikan *feedback* berupa Google Form dan mengisikan nilai sesuai dengan penilaian responden. Pengujian *User Acceptance Tests* ini dibagi menjadi dua yaitu UAT terhadap UI/UX aplikasi dan UAT terhadap fitur dan fungsionalitas aplikasi. Daftar responden yang berpartisipasi dalam mengisikan *feedback* dapat dilihat pada Lampiran 2. Berikut merupakan tabel hasil pengujian UAT.

Tabel 4.23 Hasil Pengujian Aplikasi kepada Pengguna (UI/UX)

No	Pertanyaan	Frekuensi Jawaban					Total Nilai
		STS	TS	N	S	SS	
1.	Apakah aplikasi memiliki tata letak, menu, tombol yang mudah dipahami?			7	15	7	116
2.	Apakah aplikasi dapat membuka Google Maps untuk menampilkan informasi lokasi yang dipilih?			1	7	21	136
3.	Apakah aplikasi dapat membuka Google Maps untuk menampilkan navigasi perjalanan yang dipilih?			2	6	21	135
4.	Apakah aplikasi dapat menampilkan informasi jarak dan waktu tempuh antar lokasi dengan akurat?			3	13	13	126
5.	Apakah keseluruhan fitur aplikasi dapat berjalan dengan baik?		1	4	15	9	119

Nilai maksimum $= nilai\ SS \times jumlah\ pertanyaan \times total\ responden$

$= 5 \times 5 \times 29$

$= 725$

Nilai kuesioner $= 116 + 136 + 135 + 126 + 119$

$= 632$

Nilai akhir $= \frac{632}{725} \times 100\% = 87{,}17\%$

Tabel 4.24 Hasil Pengujian Aplikasi kepada Pengguna (Terhadap Fitur dan Fungsionalitas Aplikasi)

No	Pertanyaan	Frekuensi Jawaban					Total Nilai
		STS	TS	N	S	SS	
1.	Apakah aplikasi dapat mendeteksi lokasi ponsel dan cuaca dengan akurat?			2	16	11	125
2.	Apakah aplikasi dapat memberikan lokasi penginapan yang dipilih dengan akurat?			3	10	16	129
3.	Apakah aplikasi dapat memberikan lokasi wisata yang dipilih dengan akurat?		1	2	9	17	129
4.	Apakah aplikasi dapat digunakan untuk membuat *itinerary* wisata?			2	14	13	123
6.	Apakah aplikasi dapat memberikan rute perjalanan paling pendek dan cepat untuk dilalui?			7	17	5	114

Nilai maksimum $= nilai\ SS \times jumlah\ pertanyaan \times total\ responden$

$= 5 \times 5 \times 29$

$= 725$

Nilai kuesioner $= 125 + 129 + 129 + 123 + 114$

$= 620$

Nilai akhir $= \frac{620}{725} \times 100\% = 85{,}51\%$

Kesimpulan dari pengujian *User Acceptance Test* ini, aplikasi memiliki nilai yang baik dan dapat memberikan kepuasan bagi responden dengan nilai presentase untuk UI/UX sebesar 87,17% dan presentase Fitur dan Fungsionalitas sebesar 85,51%. Pemberian predikat nilai baik ini berdasarkan kategori yang telah di tentukan dan dapat dilihat di bagian Lampiran 1.

4.4.3 Hasil Analisis Waktu Pemrosesan

Pada pengujian analisis waktu pemrosesan dilakukan pengan cara membandingkan waktu pemrosesan penyelesaian TSP menggunakan algoritma *Held-Karp* dan *Brute Force*. Hal ini dilakukan untuk mengetahui bagaimana performa algoritma yang digunakan penulis (*Held-Karp*) dengan algoritma lain. Masing-masing pengujian dilakukan sebanyak 4 kali dan mengambil nilai rata-rata yang dimasukkan ke dalam tabel perbandingan**.** Pengujian dilakukan 4 kali dikarenakan hasil waktu pemrosesan dapat berubah-ubah sesuai dengan koneksi internet yang digunakan untuk mengambil data jarak dari Google Maps API

Dari Tabel 4.25 menghasilkan waktu pemrosesan (*runtime*) yang bervariasi tergantung perbandingan antara jumlah hari dan destinasi yang dituju. Detail waktu yang dihasilkan setiap skenario percobaan algoritma dapat di lihat pada Lampiran 3.

Tabel 4.25 Pengujian Waktu Pemrosesan Algoritma

No	Jumlah Hari	Jumlah Destinasi	*Held-Karp*	*Brute Force*
1	1 hari	1	0,246 detik	0,323 detik
2	1 hari	2	0,321 detik	0,343 detik
3	1 hari	3	0,378 detik	0,492 detik
4	1 hari	5	0,656 detik	0,822 detik
5	1 hari	6	0,971 detik	1,155 detik
6	1 hari	7	1,147 detik	1,221 detik
7	1 hari	11	1,381 detik	4,954 detik
8	2 hari	2	0,564 detik	0,894 detik
9	2 hari	5	1,749 detik	1,901 detik
10	2 hari	6	2,681 detik	2,846 detik
11	2 hari	7	3,087 detik	3,146 detik
12	2 hari	13	6,590 detik	7,756 detik
13	3 hari	3	1,444 detik	1,454 detik
14	3 hari	7	3,618 detik	3,634 detik
15	3 hari	10	6,938 detik	7,411 detik

Dilihat dari hasil pengujian pada Tabel 4.25 pada nomor pengujian 1 sampai 7 diketahui *Held-Karp* selalu menghasilkan waktu yang lebih cepat (7) dibandingkan *Brute Force* dalam menyelesaikan pengujian penelitian. Hal ini membuktikan teori tentang pendekatan *Held-Karp* di halaman 9. laporan penelitian ini. Metode *Brute Force* memiliki peningkatan *runtime* yang cukup besar ketika jumlah destinasi 11 lokasi, sedangkan metode

yang dipakai dalam menyelesaikan studi kasus pada penelitian ini cukup stabil meski mengalami peningkatan *runtime* setiap penambahan destinasi. Pada nomor uji 8 hingga 15 memiliki jumlah lebih dari satu hari, waktu yang dibutuhkan lebih banyak dibandingkan dengan satu hari, hal ini dikarenakan program menyelesaikan permasalahan *clustering* terlebih dahulu menggunakan algoritma *Balanced K-Means* kemudian menyelesaikan masalah TSP sebanyak jumlah hari. Waktu *runtime* pengujian ini akan berbeda dengan *runtime* aplikasi, hal itu disebabkan karena penulis mengatur *cache* pada aplikasi, sehingga waktu *runtime* pada aplikasi jauh lebih cepat dibanding pengujian analisis waktu pemrosesan ini.