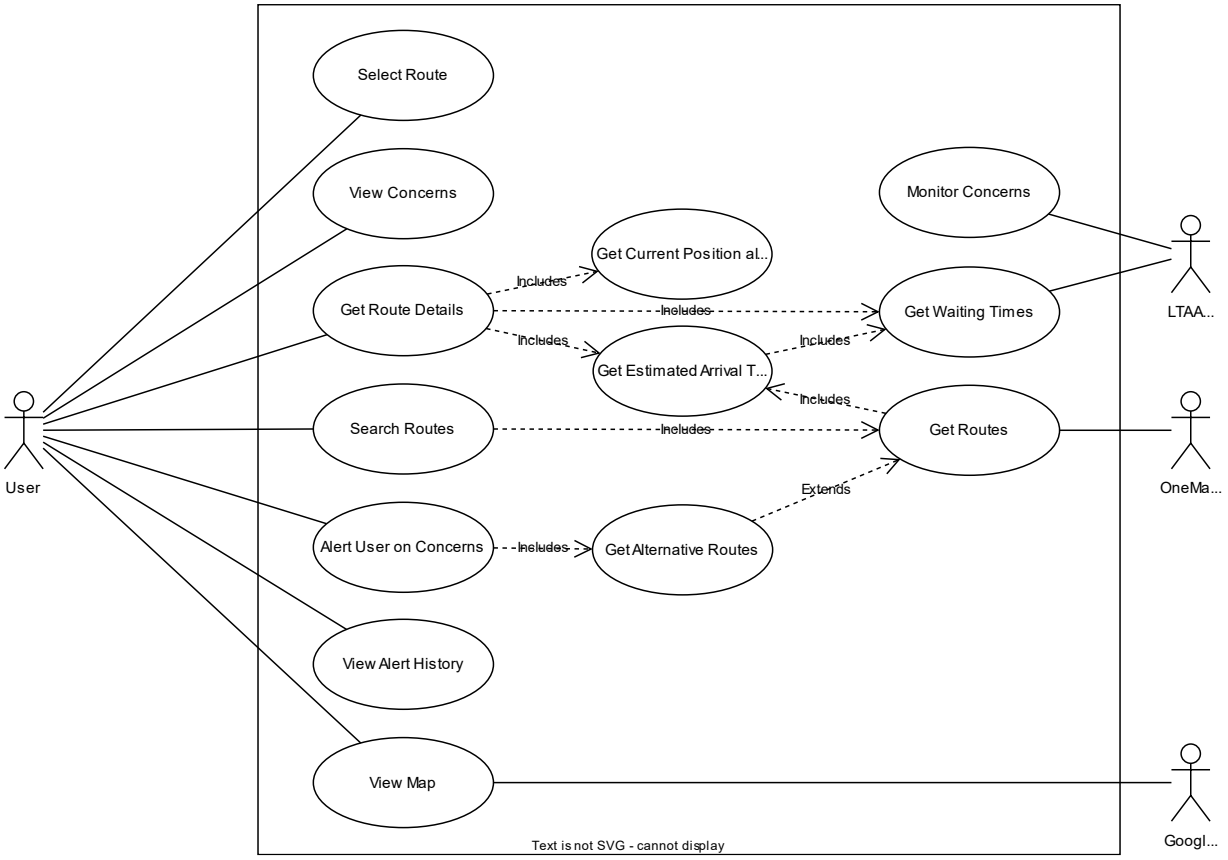
Use Case Diagram



Use Case Description

Use Case ID	1		
Use Case Name	Search Routes		
Created By	Yeo Kay Hong	*Updated By:*	
Created On	7th September 2023	*Updated On:*	
Description	Enables the user to search for possible routes from a starting location to a destination		
Actors	• User		
Preconditions	•		
Postconditions	• A list of routes is displayed to the user		
Priority	High		
Frequency of Use	High		
Flow of Events	1. The system requests for the starting point, destination, and transport type. 2. The user enters the starting point, destination, and transport type. 3. The system invokes "Get Routes" to retrieve possible routes. 4. The system displays possible routes to the user.		
Alternative Flows			
Exceptions	**EX-3A @3** **The starting point and/or destination are invalid** A message is displayed to tell the user that the input was invalid **EX-3B @3** **No routes found** A message is displayed to tell the user that no routes exist between the starting point and destination **EX-3C @3** **Get Routes not available** A message is displayed to tell the user that they are unable to search for directions now **EX-4A @4** **Routes do not have an accurate estimated arrival time** An indicator is displayed next to the estimated arrival time to explain the issue		
Includes	• Get Routes		
Special Requirement			

Assumption	• The system has an internet connection.
Notes & Issues	

Use Case ID	2		
Use Case Name	Get Routes		
Created By	Yeo Kay Hong	*Updated By:*	Yeo Kay Hong
Created On	7th September 2023	*Updated On:*	21st October 2023
Description	Retrieves possible routes from OneMap API.		
Actors	• OneMap API		
Preconditions	• A starting point has been provided. • A destination has been provided. • A transport type has been provided		
Postconditions	• Routes are returned.		
Priority	High		
Frequency of Use	High		
Flow of Events	1. The system sends a request to OneMap API. 2. OneMap API sends back possible routes. 3. The system checks each route against the concerns list and populates the routes with the concerns they are affected by. 4. The system invokes "Get Estimated Arrival Time" for each route and populates the routes with the data. 5. The system returns the list of routes.		
Alternative Flows			
Exceptions	**EX-1A** **OneMap API unavailable** **@1** An error is raised that routes cannot be retrieved now **EX-4A** **Estimated arrival time not accurate** **@4** Route is populated with arrival time anyway **@5** A flag is populated in the returned route to describe the issue		
Includes	• Get Estimated Arrival Time		
Special Requirement			
Assumption	• The OneMap API is responsive and working. • The system has an internet connection.		
Notes & Issues	Depends on data generated by "Monitor Concerns" use case		

Use Case ID	3		
Use Case Name	Get Route Details		
Created By	Yeo Kay Hong	*Updated By:*	
Created On	7th September 2023	*Updated On:*	
Description	Provides detailed information about a route.		
Actors	• User		
Preconditions	• The user has clicked on a route.		
Postconditions	• The route details are displayed to the user		
Priority	High		
Frequency of Use	High		
Flow of Events	1. The user selects a specific route. 2. The system invokes "Get Current Position along Route" to get the user's current position along the route. 3. The system invokes "Get Estimated Arrival Time" for the activated route to get the user's arrival time based on the current position of the user along the route. 4. The system invokes "Get Waiting Times" to get the waiting times at each of the transfers along the route. 5. The system displays a detailed breakdown of the route's legs and stops. 6. The system displays the live estimated arrival time. 7. The system highlights the user's live current position along the route. 8. The system displays the live waiting times at each of the transfers along the route.		
Alternative Flows			
Exceptions	**EX-2A @2**	**Get Current Position Along Route cannot find a nearby stop** The user is likely between 2 stops, so it is assumed that the last known stop is the user's current position	
	EX-2B @2	**Get Current Position Along Route has a GPS error** It is assumed that the last known stop is the user's current position, however, the indicator is greyed with a message to convey that the data might not be accurate	
	EX-3A @6	**Live estimated arrival time not accurate** An indicator is displayed next to the estimated arrival time to explain the issue	
	EX-4A @8	**Live waiting times not available** An indicator is displayed to explain the issue	
Includes	• Get Current Position along Route		

	• Get Estimated Arrival Time • Get Waiting Times
Special Requirement	
Assumption	• The system has an internet connection.
Notes & Issues	

Use Case ID	4		
Use Case Name	Get Current Position Along Route		
Created By	Yeo Kay Hong	*Updated By:*	
Created On	7th September 2023	*Updated On:*	
Description	The system captures the user's current GPS location and determines their position relative to the active route.		
Actors	None		
Preconditions	• A route is provided.		
Postconditions	• Returns the id of the stop along route that the user is closest to.		
Priority	Medium		
Frequency of Use	High		
Flow of Events	1. The system accesses the GPS to get the user's current location. 2. The system determines the nearest stop/station to the user's location. 3. The system returns the id of the identified stop.		
Alternative Flows			
Exceptions	**EX-1A** **@1**	**GPS is weak/unavailable** The system raises an error stating that GPS is unavailable	
	EX-2A **@2**	**There are no stops within 50m of the user's location** The system returns a message that no nearby stop was found	
Includes	None		
Special Requirement	• The user's device needs to have a GPS module		
Assumption	• The user has given permission for the app to access GPS		
Notes & Issues			

Use Case ID	5		
Use Case Name	Get Estimated Arrival Time		
Created By	Yeo Kay Hong	*Updated By:*	Yeo Kay Hong
Created On	7th September 2023	*Updated On:*	21st October 2023
Description	Provides a time range for the estimated time of arrival (ETA) at the destination for a given route based on the travel time as well as the waiting times for each leg.		
Actors			
Preconditions	• A route is provided.		
Postconditions	• ETA is returned.		
Priority	Medium		
Frequency of Use	High		
Flow of Events	1. The system initializes the optimistic sum to 0, this is the sum of travel and waiting times assuming the user always boards the first available service upon arriving at the transfer. 2. The system initializes the pessimistic sum to 0, this is the sum of travel and waiting times assuming the user always boards the second available service upon arriving at the transfer. 3. The system adds the travel time of the first leg to both sums. 4. The system invokes "Get Waiting Time" to get the waiting time of the next leg's service. 5. The system filters out all waiting times that are less than the optimistic sum to find the waiting times that have not elapsed by the time the user arrives at the transfer stop. 6. The system then adds the earliest waiting time to the optimistic sum. 7. The system then filters out all waiting times that are less than the pessimistic sum to find the waiting times that have not elapsed by the time the user arrives at the transfer stop. 8. The system then adds the second earliest waiting time to the pessimistic sum. 9. Steps 3-8 are repeated until all legs have been traversed. 10. The system calculates estimated time of arrival from the optimistic and pessimistic sums. 11. The system returns the estimated time of arrival.		
Alternative Flows			
Exceptions	**EX-4A** **Get Waiting Times is unavailable** **@4** A waiting time array of five-minute intervals is used instead **@11** A message is returned at the end of the function to notify the caller of the degraded performance		

	EX-5A **There are no waiting times remaining after filtering** **@5** A waiting time array of five-minute intervals is used instead **@11** A message is returned at the end of the function to notify the caller of the degraded performance
Includes	• Get Waiting Times
Special Requirement	
Assumption	• The system has an internet connection. • The LTA API is responsive and working.
Notes & Issues	

Use Case ID	6		
Use Case Name	Get Waiting Times		
Created By	Yeo Kay Hong	*Updated By:*	Yeo Kay Hong
Created On	7th September 2023	*Updated On:*	21st October 2023
Description	Retrieves the waiting times for the given public transport service at a particular stop/station		
Actors	• LTA API		
Preconditions	• A stop and a service has been provided.		
Postconditions	• Waiting times for that service at that stop is returned.		
Priority	Medium		
Frequency of Use	High		
Flow of Events	1. The system sends a request to the LTA API to get the frequency schedule of the service at the specified stop. 2. The system will generate an array of estimated waiting times based on the frequency schedule. It does so by assuming the first waiting time is half of the service's frequency, and all subsequent waiting times are in increments of the service's frequency. This array is populated until an element's value is greater than 2 hours. 3. The system sends a request to the LTA API to get the actual waiting times for the service at the specified stop. 4. All elements in the estimated waiting times that are less than the greatest value in the actual waiting times is removed. 5. The system returns both the estimated waiting times, as well as the actual waiting times.		
Alternative Flows			
Exceptions	**EX-1A** **LTA API is not responsive** **@1** An error is raised stating that waiting times are unavailable		
Includes			
Special Requirement			
Assumption	• The system has an internet connection. • The LTA API is responsive and working.		
Notes & Issues			

Use Case ID	7		
Use Case Name	Monitor Concerns		
Created By	Yeo Kay Hong	*Updated By:*	
Created On	7th September 2023	*Updated On:*	
Description	Checks LTA APIs for events (such as train service disruptions of excessively crowded stations) that may negatively impact commute and keeps a record of all such active concerns across Singapore.		
Actors	• LTA API		
Preconditions			
Postconditions	• System-wide list of concerns is updated.		
Priority	High		
Frequency of Use	Continuous		
Flow of Events	1. The system queries the LTA API for specific issues every 5 minutes. 2. LTA API sends back events and concerns. 3. The system adds new concerns to the system-wide concerns list. 4. The system removes expired items from the concerns list.		
Alternative Flows			
Exceptions	**EX-2A** **LTA API is not responsive** **@2** The event is logged **@2** The system will try again in in next interval		
Includes			
Special Requirement			
Assumption	• The LTA API is responsive and working. • The system has an internet connection.		
Notes & Issues			

Use Case ID	8		
Use Case Name	View Concerns		
Created By	Yeo Kay Hong	*Updated By:*	Yeo Kay Hong
Created On	7th September 2023	*Updated On:*	21st October 2023
Description	Displays all the currently active concerns across Singapore.		
Actors	• User		
Preconditions			
Postconditions	• All concerns across Singapore are displayed. OR • A message is displayed to tell the user that there are no concerns now		
Priority	Low		
Frequency of Use	Low		
Flow of Events	1. User selects to view concerns. 2. System retrieves and displays the active concerns from system-wide list.		
Alternative Flows			
Exceptions			
Includes			
Special Requirement			
Assumption			
Notes & Issues	Depends on data generated by “Monitor Concerns” use case		

Use Case ID	9		
Use Case Name	Alert User on New Concern		
Created By	Yeo Kay Hong	*Updated By:*	Yeo Kay Hong
Created On	7th September 2023	*Updated On:*	21st October 2023
Description	When there is a new concern, the user should be alerted if it affects the active route.		
Actors	• User		
Preconditions	• There is an active route. • The active route is affected by the concern		
Postconditions	• The system displays a list of alternative routes to the user, allowing them to select a new route if desired.		
Priority	High		
Frequency of Use	Low		
Flow of Events	1. The system receives an event by "Monitor Concerns" that there is a new concern. 2. The system checks if the active route is affected by the concern. 3. A notification is sent to the user. 4. The details of the notification are logged to a local file. 5. The user clicks on the notification. 6. The app is opened. 7. The system invokes "Get Alternative Routes" to find a set of alternative routes. 8. The system displays the alternative routes to the user.		
Alternative Flows	**AF-3A** **The user dismisses the notification** **@3** The process is aborted		
Exceptions			
Includes	• Get Alternative Routes		
Special Requirement			
Assumption	• The user has given permission for the app to access notifications		
Notes & Issues	Depends on data generated by "Monitor Concerns" use case		

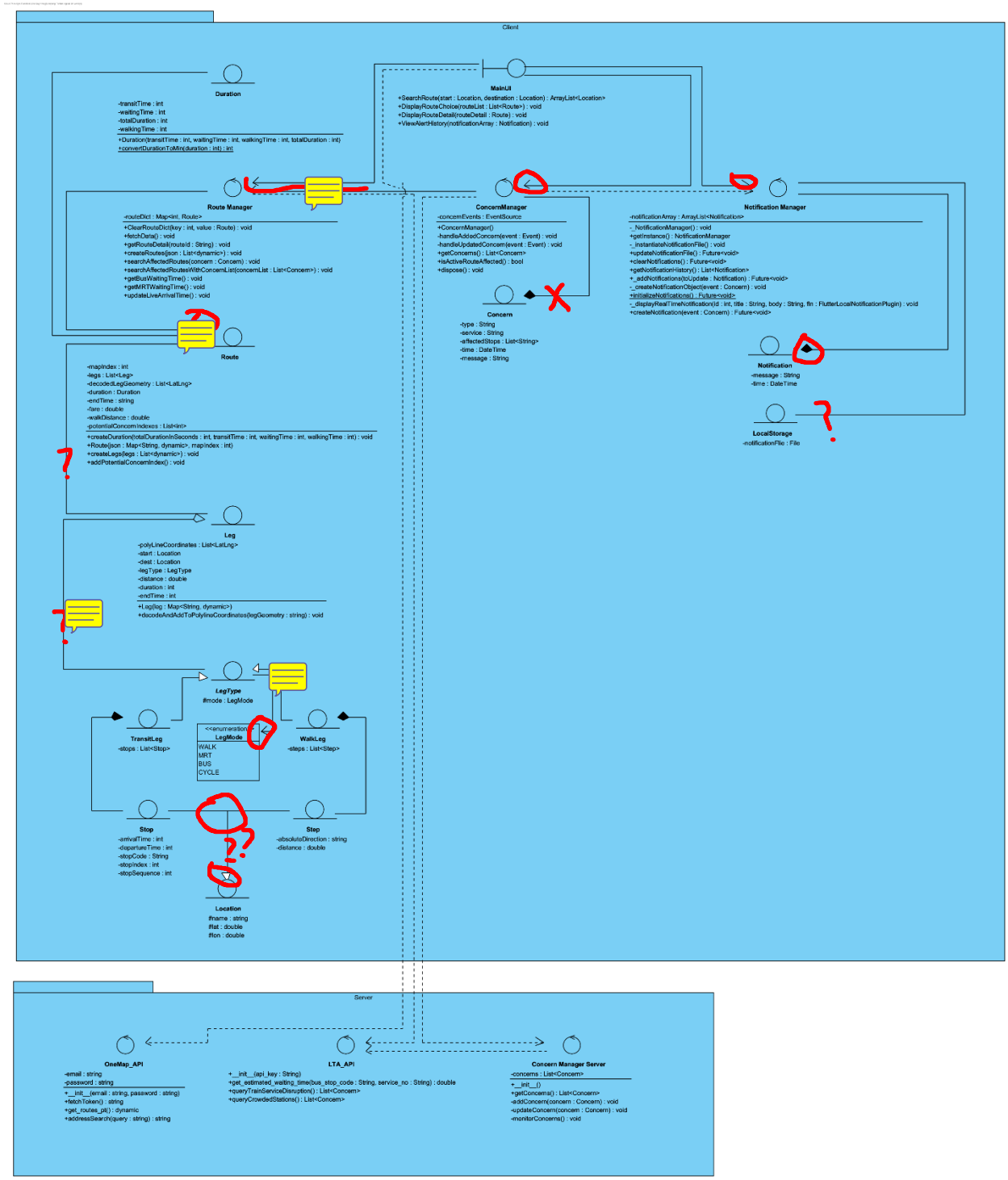
Use Case ID	10		
Use Case Name	Get Alternative Routes		
Created By	Yeo Kay Hong	*Updated By:*	Yeo Kay Hong
Created On	7th September 2023	*Updated On:*	21st October 2023
Description	Shows the alternative routes to avoid concerns.		
Actors	None		
Preconditions			
Postconditions	• The alternative routes are returned. **OR** • A message is displayed to tell the user that there are no suitable alternative routes from their current location		
Priority	High		
Frequency of Use	Low		
Flow of Events	1. The system accesses the GPS to get the user's current location. 2. The system searches for routes from the user's current location to the original destination. 3. The system filters out routes that are affected by concerns that imply that the specific route cannot be completed. 4. System returns the remaining routes.		
Alternative Flows			
Exceptions	**EX-1A @1**	**GPS is weak/unavailable** The system prompts the user to manually enter their current location as the start point of the search	
Extends	None		
Special Requirement	The user's device needs to have a GPS module		
Assumption	• The user has given permission for the app to access GPS		
Notes & Issues			

Use Case ID	11		
Use Case Name	View Alert History		
Created By	Yeo Kay Hong	*Updated By:*	
Created On	7th September 2023	*Updated On:*	
Description	Displays the historical list of alerts sent to the user		
Actors	• User		
Preconditions			
Postconditions	• Historical list of alerts displayed to user **OR** • A message is displayed to tell the user that there have been no alerts		
Priority	Low		
Frequency of Use	Low		
Flow of Events	1. The user selects to view alert history. 2. The system retrieves the log of alerts from a local log file and displays it to the user.		
Alternative Flows			
Exceptions	**EX-1A @1**	**Alert log is not accessible** A message is displayed to tell the user that there has been an issue with retrieving the alerts	
Includes			
Special Requirement			
Assumption			
Notes & Issues			

Use Case ID	12		
Use Case Name	Display Map		
Created By	Yeo Kay Hong	*Updated By:*	
Created On	21st October 2023	*Updated On:*	
Description	Displays a map to the user		
Actors	• User • Google Maps API		
Preconditions			
Postconditions	• An interactive map is displayed to the user		
Priority	High		
Frequency of Use	High		
Flow of Events	1. The system queries Google Maps API for dynamic map 2. The returned map is displayed		
Alternative Flows	**AF-2A @2**	**If a route is selected** The geometry of the route will be superimposed on the map	
Exceptions	**EX-1A @1**	**Google Maps is not available** A message is displayed to tell the user that there has been an issue with displaying the map	
Includes			
Special Requirement			
Assumption			
Notes & Issues			

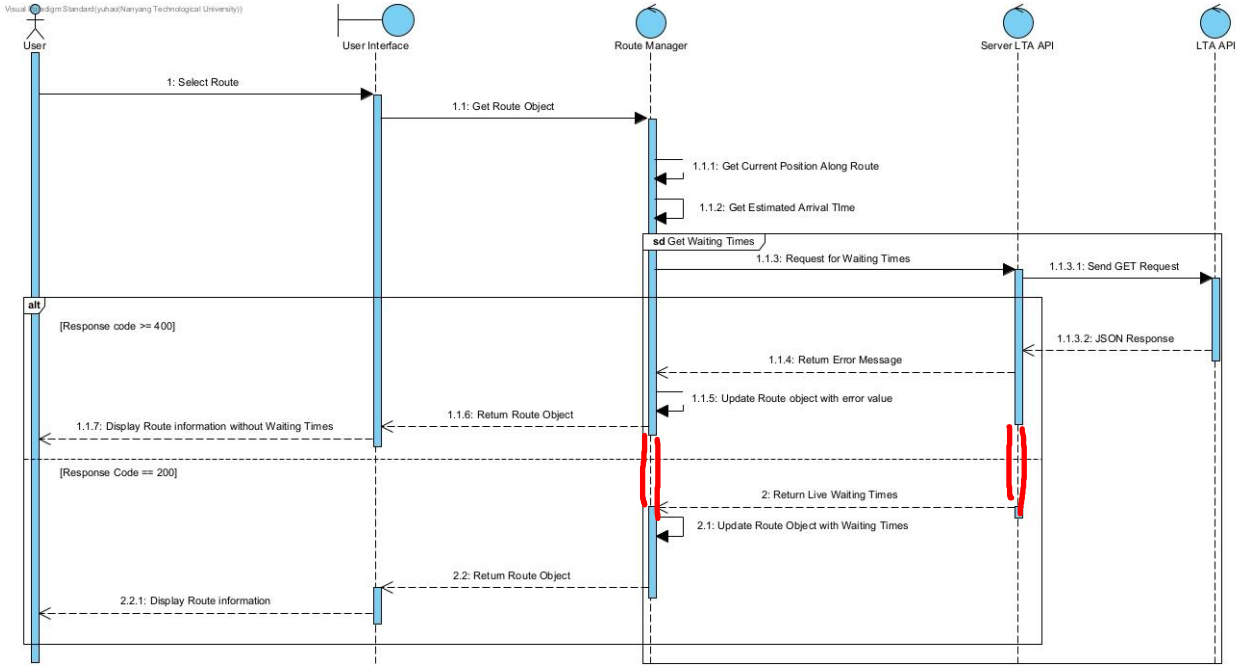
Use Case ID	13		
Use Case Name	Activate Route		
Created By	Yeo Kay Hong	*Updated By:*	
Created On	21st October 2023	*Updated On:*	
Description	Displays the historical list of alerts sent to the user		
Actors	• User		
Preconditions	• There are selectable routes		
Postconditions	• The route is activated		
Priority	High		
Frequency of Use	High		
Flow of Events	1. The user selects the route 2. The route is marked as active globally		
Alternative Flows			
Exceptions			
Includes			
Special Requirement			
Assumption			
Notes & Issues			

Class Diagram

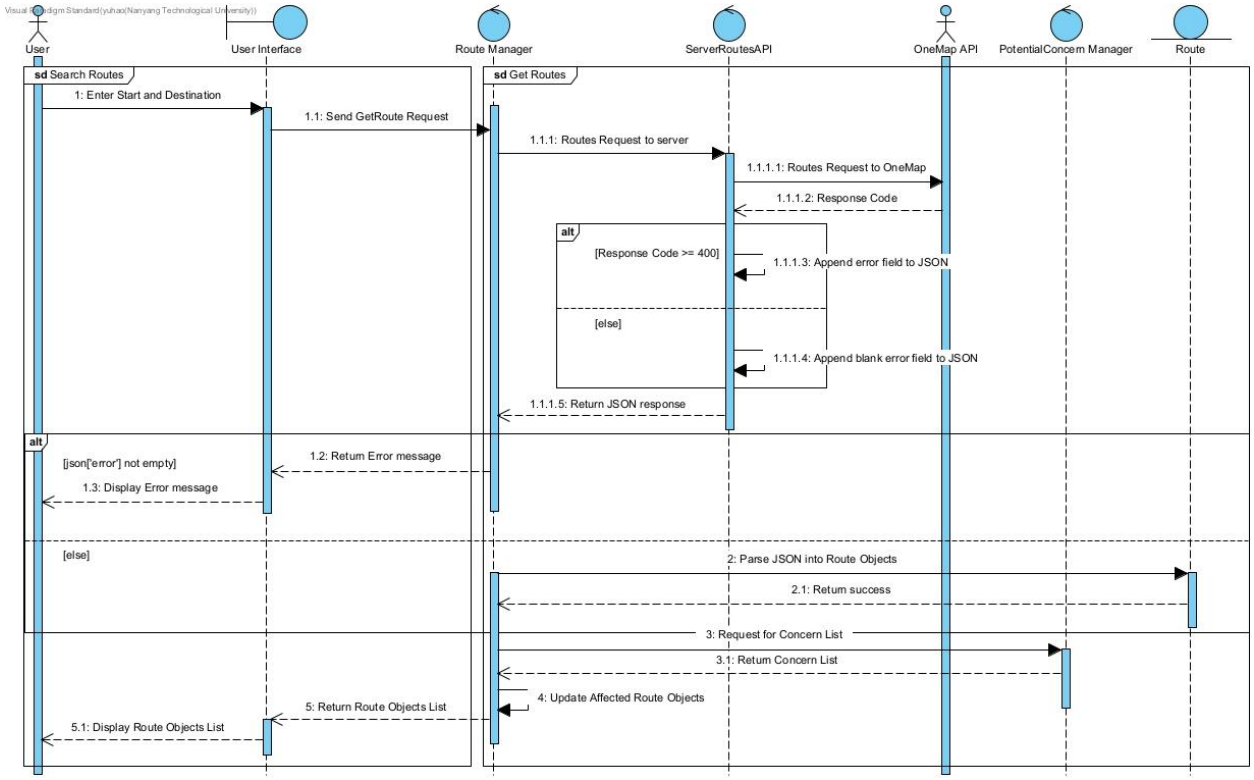


Sequence Diagrams

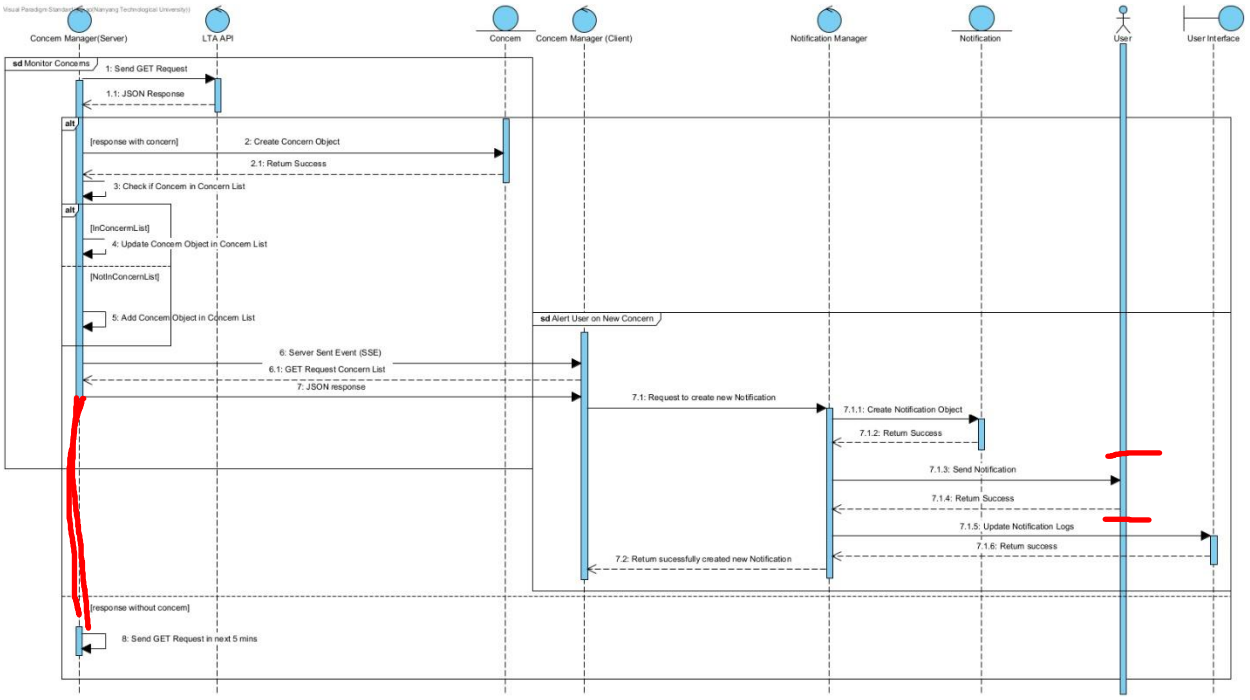
Search Routes + Get Route



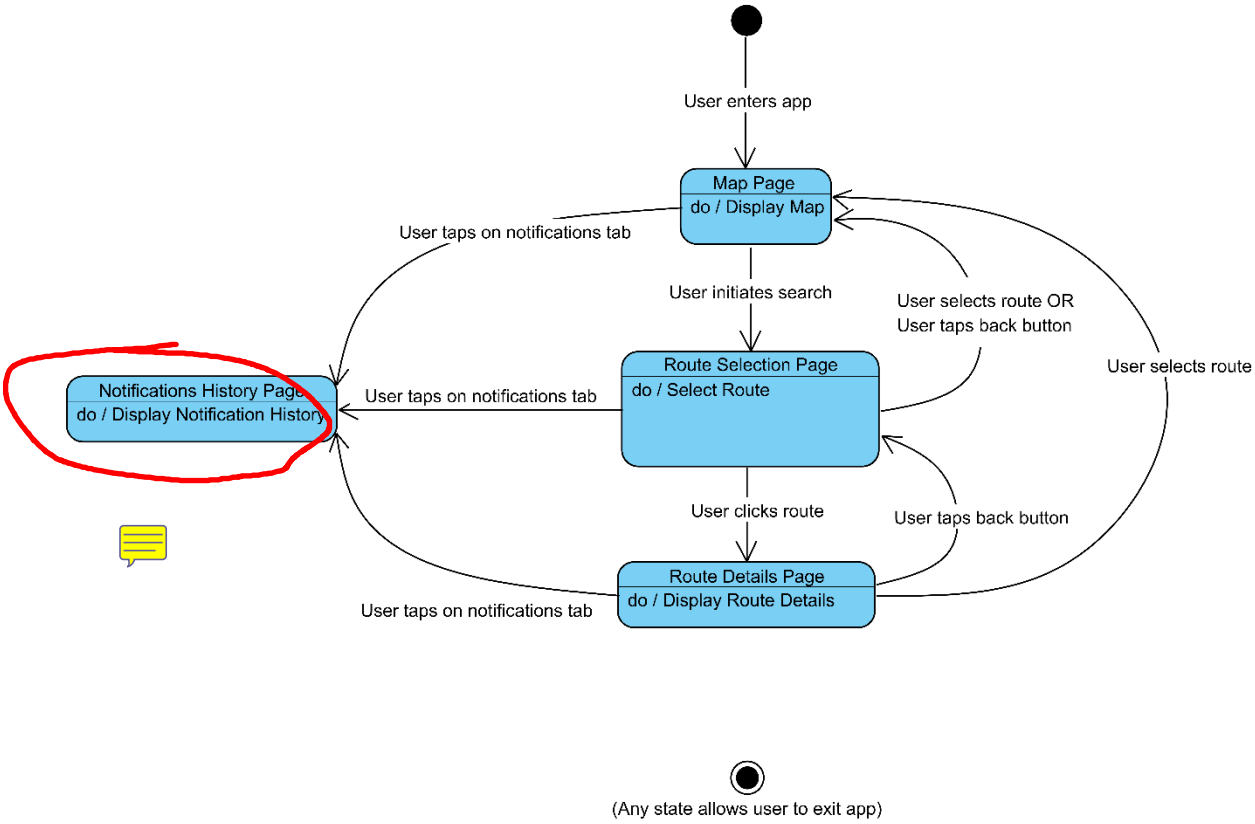
Get Route Details + Get Waiting Times



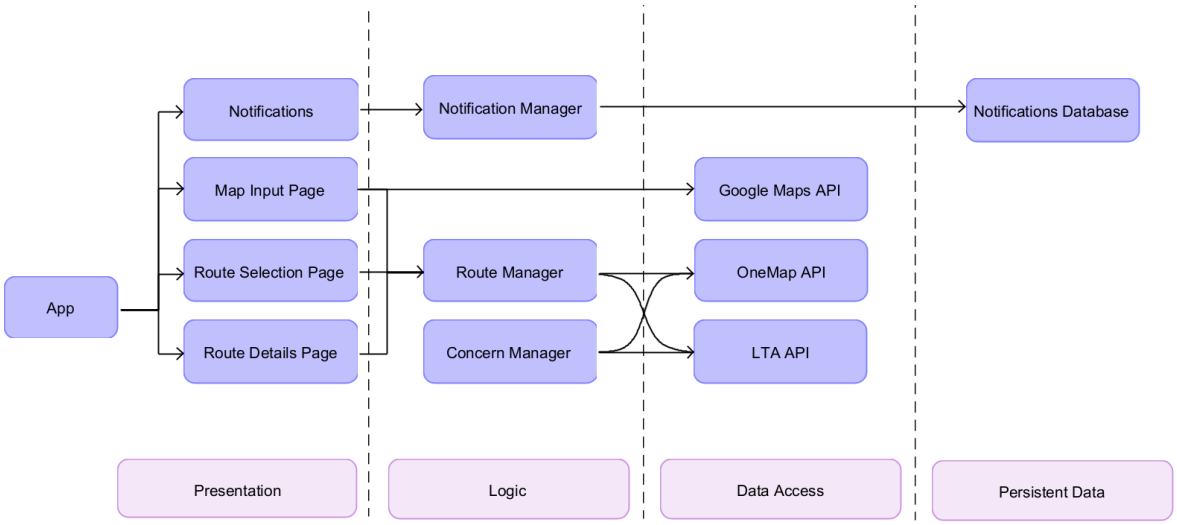
Monitor Concerns + Alert User on New Concern



Dialog Map


User enters app
Map Page
do / Display Map
User taps on notifications tab
User initiates search
User selects route OR
User taps back button
User selects route
Notifications History Page
do / Display Notification History
User taps on notifications tab
Route Selection Page
do / Select Route
User clicks route
User taps back button
Route Details Page
do / Display Route Details
User taps on notifications tab
(Any state allows user to exit app)

System Architecture



Application Skeleton

Draft Implementation of Concern Manager (Client)

```
class ConcernManager {
      final NotificationManager _notificationManager = GetIt.instance<NotificationManager>();
  ConcernManager() {
    final html.EventSource concernEvents = http.EventSource(Uri.parse("/concerns/events"));
    concernEvents.addEventListener("added", (html.Event event) {
      _handleAddedConcern(event)
    });
    concernEvents.addEventListener("updated", (html.Event event) {
      _handleUpdatedConcern(event);
    });
  }

  void _handleAddedConcern(html.Event event) {
    if (!isActiveRouteAffected()) {
      _notificationManager.createNotification(event);
    }
  }

  void _handleUpdatedConcern(html.Event event) {
    if (!isActiveRouteAffected()) {
      _notificationManager.createNotification(event);
      }
    }

  List<Concern> getConcerns() {
    http.Response response = await http.get(Uri.parse("/concerns"));
    if (response.statusCode != 200) {
      throw Exception("Failed to get concerns");
    }
    List<Concern> concerns = [];
    for (var concern in response.body) {
      concerns.add(Concern(
        type: concern["type"],
        service: concern["service"],
        affectedStops: concern["affectedStops"],
        time: concern["time"],
        message: concern["message"],
      ));
    }

    return concerns;
  }

  bool isActiveRouteAffected() {
    return false;
  }

  void dispose() {
    this.concernEvents.close();
  }
}
```

Draft Implementation of Notification Manager

```
class NotificationManager {
  List<Notification> _notifications = [];
  static final NotificationManager _instance = NotificationManager._();
  EventBus get eventBus => GetIt.instance<EventBus>();

  NotificationManager._() {
    _instantiateNotificationFile();
  }

  factory NotificationManager.getInstance() {
    return _instance;
  }

  void _instantiateNotificationFile() async {
    final input = await rootBundle.loadString("assets/NotificationList.csv");
    final fields = const CsvToListConverter().convert(input);

    List<Notification> newNotifications = fields.map((field) {
      return Notification(
        time: DateTime.parse(field[0]),
        message: field[1],
      );
    }).toList();
    _notifications.addAll(newNotifications);
  }

  Future<void> updateNotificationFile() async {
    final appDocumentsDirectory =
        await path_provider.getApplicationDocumentsDirectory();
    final csvFilePath = '${appDocumentsDirectory.path}/NotificationList.csv';
    final File file = File(csvFilePath);
    if (_notifications.isEmpty) {
      final defaultCsvData = [
        ['Timestamp', 'Message']
      ];
      await file.writeAsString(
          const ListToCsvConverter().convert(defaultCsvData),
          mode: FileMode.write);
    } else {
      final List<List<dynamic>> csvData = _notifications
          .map((notification) => [
                notification.time.toIso8601String(),
                '"${notification.message}"',
              ])
          .toList();
      try {
        await file.writeAsString(const ListToCsvConverter().convert(csvData),
            mode: FileMode.write,
            flush: true);
        print('CSV file updated successfully.');
      } catch (e) {
        print('Error updating CSV file: $e');
      }
    }
  }

  Future<void> clearNotifications() async {
    _notifications.clear();
    await updateNotificationFile();
  }
```

Draft Implementation of Route Manager

```
class RouteManager {
  final Map<int, r.Route> _routeDict = {};
  EventBus get eventBus => GetIt.instance<EventBus>();

  RouteManager() {
    eventBus.on<RouteEvent>().listen((event) async {
      Map<String, dynamic> json =
          await fetchData(event.origin, event.dest, event.routeType);
      debugPrint(
          "Received: ${event.origin}, ${event.dest}, ${event.routeType}");
      createRoutes(json['plan']['itineraries']);
    });
  }

  // for debugging purposes
  Future<Map<String, dynamic>> fetchData(
      String start, String end, String routeType) async {
    // get Current Date and Time
    DateTime now = DateTime.now();
    String formattedDate = DateFormat('MM-dd-yyyy').format(now);
    String formattedTime = DateFormat('HH:mm:ss').format(now);
    debugPrint("Date: ${formattedDate}Time: $formattedTime");
    Map<String, dynamic> json = await RoutesAPI.getRoutes(
        start: start,
        end: end,
        routeType: routeType,
        date: formattedDate,
        time: formattedTime);
    debugPrint("Data Retrieved: $json");
    return json;
  }

  r.Route? getRouteDetail(String routeId) {
    if (_routeDict.containsKey(routeId)) {
      return _routeDict[routeId];
    }
    throw 'Route not found!';
  }

  // search affected Routes and update them
  void searchAffectedRoutes(Concern concern) {}

  // search affected Routes with entire concernList
  void searchAffectedRoutesWithConcernList(List<Concern> concernList) {}

  // get Bus Waiting Time
  void getBusWaitingTime() {}

  // get MRT Waiting Time
  void getMRTWaitingTime() {}

  // update Route Arrival Time inclusive of live Waiting Time
  void updateLiveArrivalTime() {}

  // create Route Object and add to dictionary, json should be of json['itineraries]
  void createRoutes(List<dynamic> json) {
    int counter = 1;
    for (var route in json) {
      r.Route newRoute = r.Route(json: route, mapIndex: counter);
      _routeDict[counter] = newRoute;
      counter++;
    }
    debugPrint("Routes: $_routeDict");
  }
}
```

Draft Implementation of Map Page

```
class MapInputPage extends StatelessWidget {
  // implement the function callbacks for address search
  EventBus get eventBus => GetIt.instance<EventBus>();
  void handleOriginChange(String origin) {
    debugPrint("Origin selected: $origin");
  }

  void handleDestinationChange(String destination) {
    debugPrint("Destination selected: $destination");
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Map Page')),
      body: Column(
        children: [
          // temporarily substitute values in for testing
          Flexible(
            flex: 2,
            child: MapWidget(
              source: LatLng(1.320981, 103.84415),
              dest: LatLng(1.31875833025, 103.846554958),
              route: r.Route.placeholder(),
            ),
          ),
          Flexible(
            flex: 1,
            child: AddressSearchWidget(
                onOriginChanged: handleOriginChange,
                onDestinationChanged: handleDestinationChange),
          ),
          ElevatedButton(
            onPressed: () {
              // _showRouteOptions(context);
              // for actual implementation of fetching data, this will be done on Route Manager, this event should notify the Route Manager.
              eventBus.fire(RouteEvent(
                  "1.320981,103.84415", "1.318758,103.846554", "pt"));
            },
            child: const Text('Search Routes'),
          )
        ],
      ),
    );
  }
```

Draft Implementation of Backend Server

```python
from flask import Flask, request, jsonify
import json
from lta_api import LtaApi
from routes_api import ServerRoutesAPI
from ConcernManager import ConcernManager  # Assuming that the ConcernManager class is already defined

app = Flask(__name__)

# Load configuration file
with open('config.json', 'r') as config_file:
    config = json.load(config_file)

# Create instances of LtaApi and OneMapAPI
lta_api = LtaApi(config['lta_api_key'])
one_map_api = ServerRoutesAPI(config['oneMapEmail'], config['oneMapPassword'])

# Create instances of ConcernManager
concern_manager = ConcernManager()

# Example: Obtain estimated bus waiting time using bus stop code and service number instead of a route object
@app.route('/get_estimated_waiting_time', methods=['GET'])
def get_estimated_waiting_time():
    # Get bus stop code and service number from the request parameters
    bus_stop_code = request.args.get('bus_stop_code')
    service_no = request.args.get('service_no', "")

    try:
        # Call the method of the LtaApi instance to get the estimated waiting time
        estimated_waiting_time = lta_api.get_estimated_waiting_time(bus_stop_code, service_no)
        return jsonify({"estimated_waiting_time": estimated_waiting_time})
    except Exception as e:
        return jsonify({"error": str(e)})

# (To be completed) Call methods from the OneMap API
@app.route('/get_routes_pt', methods=['GET'])
def get_routes_pt():
    # Get parameters from the request
    start = request.args.get('start')
    end = request.args.get('end')
    routeType = request.args.get('routeType')
    date = request.args.get('date')
    time = request.args.get('time')
    mode = request.args.get('mode')
    maxWalkDistance = request.args.get('maxWalkDistance', "1000")
    numItineraries = request.args.get('numItineraries', "3")

    # Get the access token from OneMapAPI
    access_token = one_map_api.fetch_token()

    # Call the get_routes_pt method from OneMapAPI
    routes = one_map_api.get_routes_pt(
        access_token, start, end, routeType, date, time, mode, maxWalkDistance, numItineraries
    )

    return routes

# Assuming that the code for adding potential concern is defined
@app.route('/add_potential_concern', methods=['POST'])
def add_potential_concern():
    # Get the content of potential concern from the request
    content = request.json.get('content')

    # Create a Notification object
    notification = Notification(content)

    # Call the add_potential_concern method of the ConcernManager instance
    concern_manager.add_potential_concern(notification)

    return jsonify({"message": "Potential concern added successfully"})

# Add other app routes here and call methods from RouteManager and ConcernManager

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Draft Implementation of class to query LTA API

```python
import requests
from datetime import datetime

class LtaApi:
    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "http://datamall2.mytransport.sg/ltaodataservice"

    def get_estimated_waiting_time(self, bus_stop_code, service_no=""):
        url = f"{self.base_url}/BusArrivalv2?BusStopCode={bus_stop_code}&ServiceNo={service_no}"
        headers = {
            'AccountKey': self.api_key,
        }

        try:
            response = requests.get(url, headers=headers)

            if response.status_code == 200:
                data = response.json()
                services = data.get('Services', [])

                if services:
                    first_bus = services[0]
                    estimated_arrival = first_bus['NextBus']['EstimatedArrival']

                    # Parse estimatedArrival string and convert it to minutes
                    estimated_arrival_time = datetime.strptime(estimated_arrival, "%Y-%m-%dT%H:%M:%S%z")
                    now = datetime.now(estimated_arrival_time.tzinfo)
                    waiting_time = estimated_arrival_time - now

                    waiting_minutes = waiting_time.total_seconds() / 60.0
                    return waiting_minutes
                else:
                    raise Exception('No bus services available at the moment.')
            else:
                raise Exception('Failed to fetch estimated waiting time.')
        except Exception as e:
            raise Exception(f'Failed to get estimated waiting time: {e}')
```