Dynamic Bus Scheduling

TESTING INSTRUCTIONS

Directories:

- **src:** Contains the coding files of the System Database, Look Ahead, Route Generator, OSM Parser, Data Simulator (Travel Requests Simulator and Traffic Data Simulator), and Traffic Data Parser components.
- **resources/osm_files:** Used in order to store the input OpenStreetMap files.
- **CityPulseIntegration:** Contains coding files for connecting to the CityPulse Data Pulse, retrieving traffic data events, and storing them to the System Database.
- **tests:** Contains the testing files of the application.

MongoDB – Running: Before initializing the components of the system and running the tests, MongoDB should be running. Detailed instructions of how to start or stop MongoDB could be found following the official documentation link:

- *https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-ubuntu/*

System Parameters – Initialization: Before running the tests, some system parameters, such as the host and port of the System Database and Route Generator for example, should be initialized. These parameters are available at the **src/common/variables.py** file.

Route Generator – Initialization: Before running the Route Generator, some of its parameters, such as the number of gunicorn workers and the number of requests which are kept in the backlog when every worker is busy, should be initialized. These parameters are available at the **src/route_generator/route_generator_server_settings.py** file.

Route Generator – Running: Before running the tests, the Route Generator should be running. This can be done using the following command, which starts **gunicorn** using the **route_generator_server.py** as main file and the **route_generator_server_settings.py** as configuration file.

```
dynamic-bus-scheduling/src/route_generator$
    gunicorn -c route_generator_server_settings.py route_generator_server
```

Testing: Testing files can be found at the **tests/** directory. There are testing files for the whole application (**application_test.py**) and for individual components such as the Look Ahead (**look_ahead_test.py**), System Database (**mongodb_database_connection_test.py**), OSM Parser (**osm_parser_test.py**), Route Generator (**route_generator_test.py**), Travel Requests Simulator (**travel_requests_simulator_test.py**), Traffic Data Simulator (**traffic_data_simulator_test.py**), and Traffic Data Parser (**traffic_data_parser_test.py**).

In order to run the **application_test.py** file, the following command should be used:

```
dynamic-bus-scheduling/tests$ python application_test.py
```

The **application_test** offers the following options:

1. **(mongodb_database) clear_collections:** User can delete the documents of one or more collections of the System Database.
2. **(osm_parser) test_parse_osm_file:** The OSM Parser retrieves the geospatial data, which correspond to addresses, bus stops, edges, nodes, points, and ways, of the provided OpenStreetMap file.
3. **(osm_parser) test_populate_all_collections:** The OSM Parser stores the retrieved geospatial data to the corresponding collections of the System Database.
4. **(mongodb_database) print_collections:** User can print some of the documents of the aforementioned collections of the System Database. The number of the printed documents is selected by the user.
5. **(route_generator) test_get_route_between_two_bus_stops:** The Route Generator is tested, by retrieving the less time-consuming route which connects two bus stops. The names of the testing bus stops are initialized through the **testing_bus_stop_names** variable of the **src/common/variables.py** file. The Route generator corresponds the provided names to the corresponding bus stops which are stored at the System Database.
6. **(route_generator) test_get_route_between_multiple_bus_stops:** The Route Generator is tested, by retrieving the less time-consuming route between multiple bus stops.
7. **(route_generator) test_get_waypoints_between_two_bus_stops:** The Route Generator is tested, by retrieving all the possible waypoints which connect two bus stops. Waypoints correspond to lists of edges.
8. **(route_generator) test_get_waypoints_between_multiple_bus_stops:** The Route Generator is tested, by retrieving all the possible waypoints which connect multiple bus stops.
9. **(look_ahead_handler) test_generate_bus_line:** The Look Ahead is tested, by generating a new bus line document and store it at the corresponding collection of the System Database. Moreover, the waypoints which connect the bus stops of the bus line are retrieved by the Route Generator and stored at the System Database, so as to be used in the future by the Traffic Data Generator while generating traffic density values for the bus line.
10. **(mongodb_database) print_bus_line_documents:** User can print the generated bus line document.

11. **(mongodb_database) print_detailed_bus_stop_waypoints_documents:** User can print the stored bus stop waypoints documents.

12. **(travel_requests_simulator) test_generate_travel_request_documents:** The Travel Requests Simulator is tested, by generating a number of travel requests for a 24-hour period and store them to the corresponding collection of the System Database. User can set the number, the id of the bus line, and the initial datetime value of the generated travel requests.

13. **(mongodb_database) print_travel_request_documents:** User can print some of the generated travel request documents.

14. **(look_ahead_handler) test_generate_timetables_for_bus_line:** The Look Ahead is tested, by generating timetables of a bus line and store them at the corresponding collection of the System Database.

15. **(mongodb_database) print_timetable_documents:** User can print the generated timetable documents.

16. **(traffic_data_simulator) test_generate_traffic_data_between_multiple_stops:** Traffic Data Simulator is tested, by updating the traffic density values of the edges which connect the testing bus stops.

17. **(mongodb_database) print_traffic_density_documents:** User can print the updated traffic density documents.

18. **(look_ahead_handler) test_update_timetables_of_bus_line:** The Look Ahead is tested, by sending a request to the Route Generate in order to identify the less time-consuming route between the bus stops of the testing bus line, taking into consideration the current values of traffic density, and updating the corresponding timetables.

19. **(look_ahead_handler) start_timetables_generator_process:** A new process is spawned, responsible for generating new timetables periodically. The frequency and total duration of the operation is provided as input by the user, by setting the values of the **look_ahead_timetables_generator_timeout** and **look_ahead_timetables_generator_max_operation_timeout** variables, at the **src/common/variables.py** file.

20. **(look_ahead_handler) terminate_timetables_generator_process:** The timetables generator process is terminated.

21. **(look_ahead_handler) start_timetables_updater_process:** A new process is spawned, responsible for updating the existing timetables periodically. The frequency and total duration of the operation is provided as input by the user, by setting the values of the **look_ahead_timetables_updater_timeout** and **look_ahead_timetables_updater_max_operation_timeout** variables, at the **src/common/variables.py** file.

22. **(look_ahead_handler) terminate_timetables_updater_process:** The timetables updater process is terminated.

23. **(travel_requests_simulator) start_travel_requests_generator_process:** A new process is spawned, responsible for generating new travel requests periodically. The frequency and total duration of the operation is provided as input by the user, by setting the values of the **travel_requests_generator_timeout** and **travel_requests_generator_max_operation_timeout** variables, at the **src/common/variables.py** file.

24. **(travel_requests_simulator) terminate_travel_requests_generator_process:** The travel requests generator process is terminated.

25. **(traffic_data_simulator) start_traffic_data_generator_process:** A new process is spawned, responsible for updating the traffic density values of the edges periodically. The frequency and total duration of the operation is provided as input by the user, by setting the values of the **traffic_data_ generator_timeout** and **traffic_data_generator_max_operation_timeout** variables, at the **src/common/variables.py** file.

26. **(traffic_data_simulator) terminate_traffic_data_generator_process:** The traffic data generator process is terminated.