

Distributed Systems Engineering Submission Document DEAD (Final Project)

Each team member should enter his/her personal data below:

Team member 1	
Name:	Nenad Cikojevic
Student ID:	01549084
E-mail address:	a01549084@unet.univie.ac.at

Team member 2	
Name:	Nemanja Srdanovic
Student ID:	01576891
E-mail address:	a01576891@unet.univie.ac.at

Team member 3	
Name:	Adem Mehremic
Student ID:	01650669
E-mail address:	a01650669@unet.univie.ac.at

Team member 4	
Name:	Veljko Radunovic
Student ID:	01528243
E-mail address:	a01528243@unet.univie.ac.at

Birds-eye-view of the system:

MSCF is a framework that keeps entire communication and message-exchange logic “under the hood”. It is wrapped inside of a Java library which is imported by every MS. ConnectionHandler class is a bridge between MSCF and a MS. It starts connection after calling the connect() method. The network scanning and message exchange logic is held in udp_conn package. There is UDPServer class and ServerWorker which are responsible for receiving the DatagramPackets and making decision if received messages are going to be forwarded or processed (“destinatedMS” is property of Message class used to filter received Messages). It also sends on the same channel, simple acknowledgement “200 OK” String packed in DatagramPacket to signalize that message is received. It works in multi-threaded manner so the received DatagramPackets are handled simultaneously. If the Message is intended for this MS, MessageProcessor will call onMessage() method which activates Message processing. MessageProcessor is a functional interface implemented separately by each MS on its own way. Sending Messages are done by UDPClient/ClientWorker, also simultaneously. These two classes have also a task to pick IPaddresses and port numbers (semi randomly) for a given (white) list of IPs and given range of port numbers and find two MSs to send messages (next 10seconds) to. For this task each ClientWorker sends simple “Greeting” String packed in a DatagramPacket and waits for 200miliseconds. When acknowledgement is received from first two MSs, those IPs and ports are going to be held, for the rest of 10 seconds (previously given by Timer) to which the Messages are going to be sent to. Message sending works in a similar way- each is sent and socket waits for 600 miliseconds for an acknowledgement, if it is not received in that period the Message is sent again (with another waiting). In any case the Message is saved into the Database so if the Message with the same messageID(UUID) is received, it will be simply ignored.

MS1, when started generates “City map” which is basically composed as a chess board (15x15) of single tiles (ARSField class) of type STREET and GRASS, complete streets as List of single tiles which are placed in a ARS-Map which holds HashMap of tiles and list of Streets. It gets requests for DrivableFields and route from MS2; Map and specific area to block from MS4. The main purpose of this MS is to keep others updated about changing on the map, calculating the route for MS2, sending messages or statistical analysis as well as forwarding messages to other microservices.

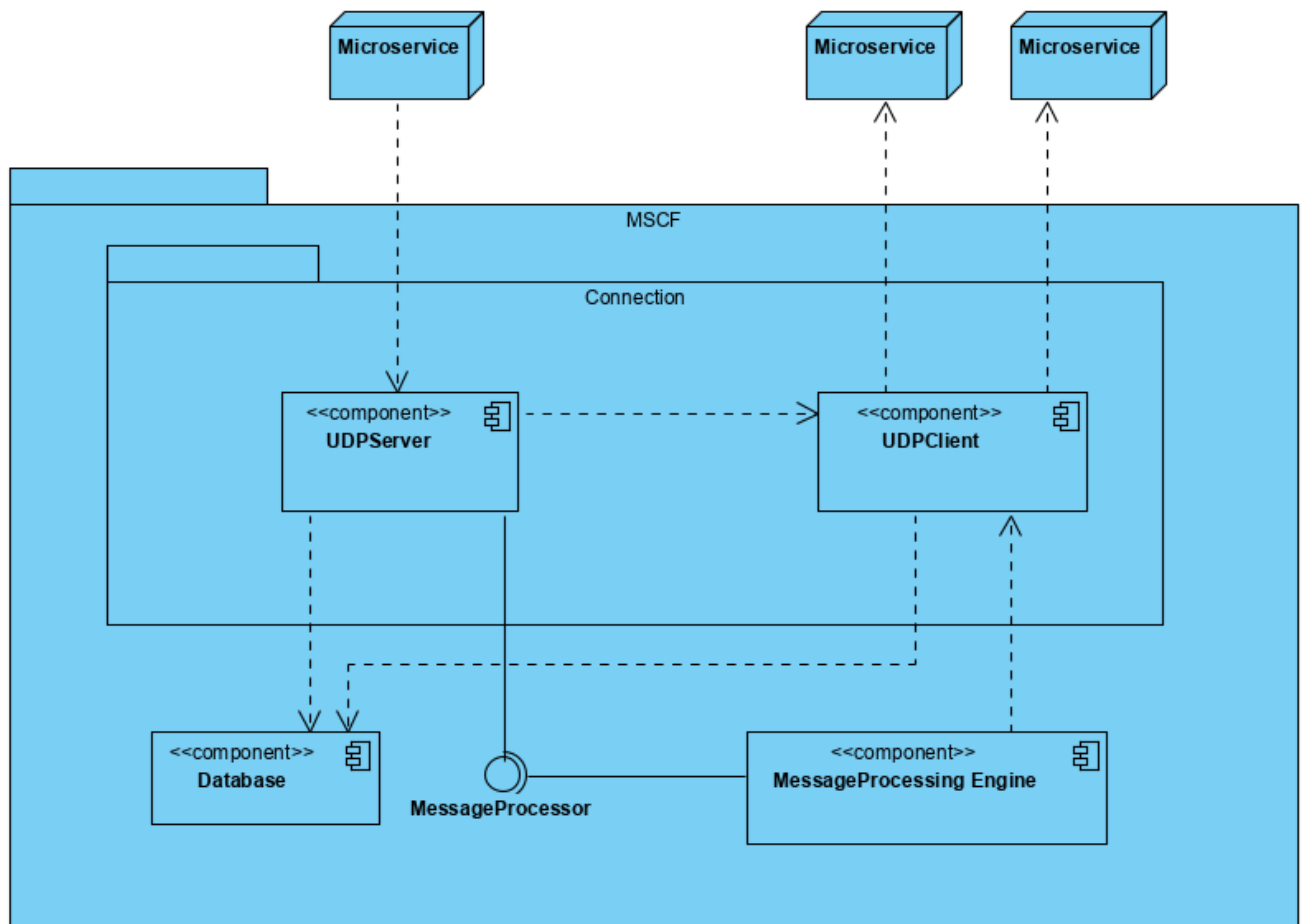
MS2 (Car Data Gateway) microservice represents a minimalistic service which simulates a selfdriving car. It simulates the traveling along a route provided by the Automatic Routing Service. For the route calculation it provides the ARS with a car object containing a current position and wanted destination. To be able to do that the microservice instantiates a car object when started, and sends a request to the ARS for getting the current drivable fields on the map. When it receives a response it choses randomly a position and destination from the provided list, informs the Road Management Service about its position and sends a request for a car route from the current position to the destination. After the car route has been calculated and received by the CDG it begins the travel and sends two messages after every hop on the next field in the route. The first message is another request to the ARS to get the car route (now with the updated position), and the second is the information sent to the RMS with its current position.

MS3 (Network Monitoring and Maintenance Service) microservice represents a graphical user interface for the representation of the messages that were exchanged between other microservices. The Spring Boot application generates a tomcat server on the localhost, and through thymeleaf, it transports java data to the web layer. The microservice is receiving the list of messages that are collected by others and put those in the database. Before presenting on them web, microservice is sorting those in the successfully received and dropped messages. The collected information in MS3 will be presented on the website that is based on a "mermaid" JS library.

MS4 (Road Management System)

MS4 is the unity over with the User can interact with the rest of the System through the use of the CLI. MS4 has a RMS controller which gets Data as Car Information and generates Data such as blocked areas. The Car is received and processed in the MessageController, which generates new threads (MessageWorker). The blocked Areas are created in the AreaController and in them the Street are blocked.

CLI has a CLI Application which runs a while loop and calls chosen methods from the CLIController. The CliController is the connection to the MS4.



Lessons learned:

1. What were the problems?

Figure out how synchronization practically works. A lot of tries and fails

To make sending messages for statistical processing

The difficulty with MS3 was with adjusting the spring boots beans. It was perplexed to connect all those packages in java to be "autowired". Hibernate MessagesRepository class for managing the H2 database was getting the null pointer as a given parameter whenever it was called outside that class even if the given parameter wasn't the null. We didn't have time to fix that problem.

2. Which problems could be solved, and which could not?

Proper integration test,

The already described problem with Hibernate was some kind of solved problem. We implemented SQLite like in all other microservices, and we have in some way bypass the problem with the hibernate.

3. Which decisions do you regret?

For not thinking of activating each worker after message is received, earlier

For the MS3 we definitely regret using mermaid as a JS library for data visualization. Problem was to display message parts in thymeleaf because all getters from messages were inserted in <div> and tags in HTML, and the mermaid framework can't load any HTML tags, a "mermaidJS" can only read the pure text. The solution at the end was to convert all messages in Java to StringBuilder to be able to parse them to the JS framework.

4. If you could start from scratch, what would you do differently?

- Figure out way for more rational thread worker usage
- Integration testing
- Time management

Team contributions

Nenad Cikojevic:

- MSCF
- MS1 (ARS)
- Unit and Integration Tests

Nemanja Srdanovic

- MSCF
- MS2 (CDG)
- Unit and Integration Tests

Veljko Radunovic

- MSCF
- MS3 (NMM)
- Unit and Integration Tests

Adem Mehremic

- MSCF
- MS4 (RMS)
- CLI
- Unit and Integration Tests

Discussion:

Not implemented:

- Error Handling: When errors are caught, they are just kind of “logged”, response is not sent back (could be implemented)
- Destination of a car is blocked -> Route calculation engine fails, NullPointerException follows (this case could have been handled)
- MS1 updates MS2 when area is blocked (route can be changed, but if a car happens to be on the couple field before the blocked one, it may not be properly updated)
- Secure that MS3 has received all statistic messages (several attempts have been done without success)
- Adding too much MS_IDs in visitedMS list- property of Message object used to be visualize hops on the way from start to destination of a message on MS3
- When a BlockArea or Unblock request is dropped, there will not be another sent unless the user makes one.

Implemented:

- Peer Discovery – Microservices
- Routing and Forwarding
- Peers are at every moment connected at most to 2 other Peers
- Asynchronous communication – microservices receive/processed/send every message in a separate thread
- Persistent Message Storage – all messages that are created/forwarded/received are saved into a database
- Etc.

Interfaces & API:

MS1 API Specification (Automatic Routing Service):

Title	Send route for a given Car
Description and Use Case	<p>Calculate the route for the given car object received at udp server. When received the MS1 checks if the object already exists inside cache.</p> <p><u>Use case 1 - Received new car object</u></p> <ul style="list-style-type: none"> Object will be inserted in cache (List<Car>) The new car object will be put inside a List<Car> on the Field which has the same coordinates as the car position. Then the RouteService class will be triggered, so that a route from the car's position to the car's destination will be calculated. The calculated route will be put inside the car object as a list of fields, and the object sent to the destination (MS2) it came from, as a Message object. <p><u>Use case 2 - Received an existing car object</u></p> <ul style="list-style-type: none"> For the existing car object the position and destination will be set accordingly with the received car object. The car position will be adapted on the map accordingly with the received object. All other steps are exactly as in the use case 1.
Transport Protocol Details	<p>UDPClient, ResponseID- UUID of the received message</p> <p>UDPServer</p>
Request Body Example (Server)	<pre>{ "MessageId": "37d1dd22-314c-4cc4-aeba-a2c30a0e2211", "DestinatedMS": "MS1_ARS", "SourceIP": "10.101.101.13:3020", "Endpoint": "drivableFields" "Data": "Car", "VisitedMS": "MS2_CDG_3020", }</pre>

Request Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> Endpoint: Identifier for the destination microservice method.</p> <p> Data: Is the object which is send to the microservice that processes the request.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Sent (Response) Body Example (Client)	<pre>{ "MessageId": "529b370b-ea30-4553-97a0-c15332ce84c3", "DestinatedMS": "MS2_CDG", "SourceIP": "10.101.101.5:3022", "DestinationIP": "10.101.101.13:3020", "ResponseID": "37d1dd22-314c-4cc4-aeba-a2c30a0e2211", "Data": "Car", "VisitedMS": " MS1_ARS_3022 MS2_CDG_3020 ", }</pre>
Sent (Response) Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> DestinationIP: The IP address and Port of the microservice, which has send the request.</p> <p> ResponseID: Endpoint indentifier for the micreservice which has send the request. The responseID is the same as the messageID from the message the request was received.</p> <p> Data: The data which has to be processed at the destination.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Error Cases	<p> Position/Destination field invalid: The position or destination field type is not drivable or invalid (grass), so that the car route can not be calculated. Also it could be that the maximum amount of cars on a field is already reached.</p> <p> Invalid data type: Expected data type is Car in every other case exception is thrown.</p>
Notes & Remarks	The Field and Car class are further described in the MSCF Class Diagram and Description.

Title	Send drivable Fields
Description and Use Case	<p>To set the cars position and destination, the MS2 requests a List of all drivable Fields.</p> <p>Use case:</p> <ul style="list-style-type: none"> • Microservice receives a request message on the "drivableFields" endpoint • MapService engine returns a list of streets with drivable fields • List of streets is packed into map object and then wrapped into Message and sent back to MS2
Transport Protocol Details	<p>UDP, Client endpoint: /drivableFields</p> <p>UDP, Server endpoint:</p>
Request Body Example (Server)	<pre>{ "MessageId": "37d1dd22-314c-4cc4-aebe-a2c30a0e2211", "DestinatedMS": "MS1_ARS", "SourceIP": "10.101.101.13:3020", "Endpoint": "drivableFields" "Data": "Car", "VisitedMS": "MS2_CDG_3020", }</pre>
Request Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> Endpoint: Identifier for the destination microservice method.</p> <p> Data: Is the object which is send to the microservice that processes the request.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>

Send (Response) Body Example (Client)	<pre>{ "MessageId": "529b370b-ea30-4553-97a0-c15332ce84c3", "DestinatedMS": "MS2_CDG", "SourceIP": "10.101.101.5:3022", "DestinationIP": "10.101.101.13:3020", "ResponseID": "37d1dd22-314c-4cc4-aeba-a2c30a0e2211", "Data": "Map", "VisitedMS": " MS1_ARS_3022 MS2_CDG_3020 ", }</pre>
Response Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> DestinationIP: The IP address and Port of the microservice, which has send the request.</p> <p> ResponseID: Endpoint indentifier for the micreservice which has send the request. The responseID is the same as the messageID from the message the request was received.</p> <p> Data: The data which has to be processed at the destination.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>

Title	Send message statistic to Network Monitoring and Maintenance Service
Description and Use Case	<p>After starting the microservice a timer is activated which will trigger a function that sends the database statistic to the NMM every 4 seconds.</p> <p><u>Use case:</u></p> <ul style="list-style-type: none"> • The microservice is started on which a timer also starts, which triggers a function every 4 seconds. • That function is getting all the messages from a database copy and sends them to the NMM as a <code>ArrayList<Message></code> in a Message object. • After sending the message the Database copy is cleared, so that the same data is not send again.
Transport Protocol Details	<p>UDP, Client endpoint: / MS3_NMM</p> <p>UDP, Server endpoint: /</p>

Sent Body Example (Client)	<pre>{ "MessageId": " c2217af6-030a-41e1-9edd-e5000ec64dbc", "DestinatedMS": "MS3_RMS", "SourceIP": "10.101.101.5:3021", "Data": "ArraList<Message>", "VisitedMS": "MS1_ARS_3021", }</pre>
Sent Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> Data: Is the object which is send to the microservice that processes the request.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Response Body Example (Server)	<pre>{ "MessageId": "2cb93f97-3833-4ab1-974c-4be5149a05c8", "DestinatedMS": "MS2_CDG", "SourceIP": "10.101.101.9:3024", "DestinationIP": "10.101.101.5:3021", "ResponseID": " c2217af6-030a-41e1-9edd-e5000ec64dbc", "Data": "", "VisitedMS": " MS3_NMM_3024 MS1_ARS_3021 ", }</pre>
Response Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> DestinationIP: The IP address and Port of the microservice, which has send the request.</p> <p> ResponseID: Endpoint indentifier for the micreservice which has send the request. The responseID is the same as the messageID from the message the request was received.</p> <p> Data: The data which has to be processed at the destination.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Error Cases	<p> Invalid data type: Expected data type is Car in every other case exception is thrown.</p>
Notes & Remarks	<p>The Field class is further described in the MSCF Class Diagram and Description.</p>

Title	Map requested (list of all streets)
Description and Use Case	<p>The microservice MS4 sends a request to get a map object with a list of all available streets and their fields, so that the requester can work with those objects.</p> <p><u>Use case - Send a map request to MS1:</u></p> <ul style="list-style-type: none"> MS4 sends a request to MS1 to get a map from MS1 The MapService engine is triggered which returns a list of all street objects which are packed in map and passed to UDPClient to send it on network
Transport Protocol Details	<p>UDP, Client endpoint: ResponseID- UUID of the received message</p> <p>UDP, Server: /streets</p>
Request Body Example	<pre>{ "MessageId": "89sdf-kajf-71fv-iaj3", "SourceIP": "10.101.101.17:3022", "DestinatedMS": "MS1_ARS", "VisitedMS": " MS4_RMS_2021 " "endpoint": "streets" }</pre>
Request Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> SourceIP: The IP address of the microservice and the port on which the program is running.</p> <p> DestinatedMS: The unique Microservice ID of the microservice to whom the message was intended.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p> <p> endpoint: Endpoint on MS1 where the message will be received</p>
Send (Response) Body Example	<pre>{ "MessageId": "ksj4-er23-ng54-8d7g", "SourceIP": "10.101.101.5:3024", "DestinatedMS": "MS4_RMS", "VisitedMS": " MS1_ARS MS2_CDG MS4_RMS", "Object": "Map", }</pre>

Send (Response) Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> SourceIP: The IP address of the microservice and the port on which the program is running.</p> <p> DestinatedMS: The unique Microservice ID of the microservice to whom the message was intended.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p> <p> Object: The data which has to be processed at the destination.</p>
-------------------------------------	---

Title	Block area
Description and Use Case	<p>MS4 sends Street object in the message in order to block specific area (or the whole street)</p> <p>Use case:</p> <ul style="list-style-type: none"> • Street object is received, validated and if conditions are fulfilled MapService will find the street to block in map, change the is-Drivable to false • MS1 sends back a message with the same street back to MS4 • MS1 sends back to MS2s Updated list of all Streets containing Drivable Fields
Transport Protocol Details	<p>UDP, Client endpoint: ResponseID</p> <p>UDP, Server endpoint: /block</p>
Sent Body Example	<pre>{ "MessageId": "h4j1-ada5-jsg4-aj5d", "SourceIP": "10.101.101.17:3022", "DestinatedMS": "MS1_ARS", "VisitedMS": " MS4_RMS MS3_NMM", "endpoint": "block", "Object": "Street", }</pre>
Sent Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> SourceIP: The IP address of the microservice and the port on which the program is running.</p> <p> DestinatedMS: The unique Microservice ID of the microservice to whom the message was intended.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p> <p> endpoint: Endpoint on MS1 where the message will be received</p> <p> Object: Street with the blocked fields.</p>

Send (Response) Body Example	<pre>{ "MessageId": "h4j1-ada5-jsg4-aj5d", "SourceIP": "10.101.101.17:3022", "DestinatedMS": "MS4_RMS ", "VisitedMS": " MS1_RMS MS3_NMM MS4_RMS", "Object": "Street", }</pre>
Response Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> SourceIP: The IP address of the microservice and the port on which the program is running.</p> <p> DestinatedMS: The unique Microservice ID of the microservice to whom the message was intended.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p> <p> endpoint: Endpoint on MS1 where the message will be received</p> <p> Object: Street with the blocked fields.</p>
Error Cases	<p> Received Area(Street and the Fields) from CLI is invalid: the StreetName and the field do not exist</p> <p> The area that is supposed to be blocked, is a destination of a car</p>

MS2 API Specification (Car Data Gateway):

Title	Request drivable Fields
Description and Use Case	<p>To set the cars position and destination, the microservice requests a List of all drivable Fields.</p> <p>Use case:</p> <ul style="list-style-type: none"> • The microservice creates a new Car object and

	<p>initializes the function to set the objects position and destination. For this the microservice needs a List of all drivable fields.</p> <ul style="list-style-type: none"> • Sends a get request in form of a message where the object is a car object. • Receives a message with the Map object, from which it extracts all drivable fields and chooses random the position and destination, after which he triggers the function travelRoute.
Transport Protocol Details	<p>UDP, Client endpoint: /drivableFields</p> <p>UDP, Server endpoint: / message ResponseID</p>
Sent Body Example (Client)	<pre>{ "MessageId": "37d1dd22-314c-4cc4-aebe-a2c30a0e2211", "DestinatedMS": "MS1_ARS", "SourceIP": "10.101.101.13:3020", "Endpoint": "drivableFields" "Data": "Car", "VisitedMS": "MS2_CDG_3020", }</pre>
Sent Body Description	<p>I MessageId: Message identifier which is unique and automatically assigned.</p> <p>L DestinatedMS: Endpoint identifier for the destination microservice.</p> <p>L SourceIP: The IP address and Port of the microservice which generated the message.</p> <p>L Endpoint: Identifier for the destination microservice method.</p> <p>L Data: Is the object which is send to the microservice that processes the request.</p> <p>L VisitedMS: List of visited microservices till arrival at the destination.</p>
Response Body Example (Server)	<pre>{ "MessageId": "529b370b-ea30-4553-97a0-c15332ce84c3", "DestinatedMS": "MS2_CDG", "SourceIP": "10.101.101.5:3022", "DestinationIP": "10.101.101.13:3020", "ResponseID": "37d1dd22-314c-4cc4-aebe-a2c30a0e2211", "Data": "Map", "VisitedMS": " MS1_ARS_3022 MS2_CDG_3020 ", }</pre>

Response Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> DestinationIP: The IP address and Port of the microservice, which has send the request.</p> <p> ResponseID: Endpoint identifier for the microservice which has send the request. The responseID is the same as the messageID from the message the request was received.</p> <p> Data: The data which has to be processed at the destination.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Error Cases	<p> Invalid data type: Expected data type is Map in every other case exception is thrown.</p> <p> Not all fields in the received List are drivable in which case an exception is thrown.</p>
Notes & Remarks	The travel method is further described in the code implementation supd.

Title	Request car route
Description and Use Case	<p>To start the travel method (in order the car to move on the map) the CDG microservice sends a request to the ARS microservice to get a route depending on the position and destination of the car.</p> <p><u>Use case:</u></p>

	<ul style="list-style-type: none"> • After setting the position and destination of the new car object, the microservice requests a route from the position field to destination field. • This is done by putting a car object with the set fields into a message and sending it to the ARS microservice. • The microservice calculates the route and puts it inside the car object which is again put in a message and sent back to the requester. • After receiving the new car object with the route to the destination, the travel method for this object is triggered.
Transport Protocol Details	<p>UDP, Client endpoint: /carRoute</p> <p>UDP, Server endpoint: / message ResponseID</p>
Sent Body Example (Client)	<pre>{ "MessageId": "baleldb0-548e-458a-9187-fbf12c56f649", "DestinatedMS": "MS1_ARS", "SourceIP": "10.101.101.13:3020", "Endpoint": "carRoute" "Data": "Car", "VisitedMS": "MS2_CDG_3020", }</pre>
Sent Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> Endpoint: Identifier for the destination microservice method.</p> <p> Data: Is the object which is send to the microservice that processes the request.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Response Body Example (Server)	<pre>{ "MessageId": "01dc5c0b-7507-4cae-9d10-7bd7b5ee51fd", "DestinatedMS": "MS2_CDG", "SourceIP": "10.101.101.5:3022", "DestinationIP": "10.101.101.13:3020", "ResponseID": " baleldb0-548e-458a-9187-fbf12c56f649", "Data": "Car", "VisitedMS": " MS1_ARS_3022 MS2_CDG_3020 ", }</pre>

Response Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> DestinationIP: The IP address and Port of the microservice, which has send the request.</p> <p> ResponseID: Endpoint indentifier for the micreservice which has send the request. The responseID is the same as the messageID from the message the request was received.</p> <p> Data: The data which has to be processed at the destination.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Error Cases	<p> Invalid data type: Expected data type is Car in every other case exception is thrown.</p> <p> Invalid Field type: Inside the route (List<Street>) there's a Field with a type undrivable.</p>
Notes & Remarks	The Field class is further described in the MSCF Class Diagram and Description supd.

Title	Inform the Road Management Service about position
Description and Use Case	<p>After the car has set its position and destination it informs the RMS.</p> <p><u>Use case:</u></p> <ul style="list-style-type: none"> • The car has received from the ARS the drivable fields on the map and chosen a position and destination. • It informs the RMS about its position and destination and starts traveling, where it will after every next field inform the RMS again about its position and route it will travel on. • The RMS will receive a car object which will be registered if its not already in the system. • If in the system the RMS will update the data for the existing car object.
Transport Protocol Details	UDP, Client endpoint: / instanceof Car

Sent Body Example (Client)	<pre>{ "MessageId": " d1873fbe-76a0-4780-8342-5fc03f9a9994", "DestinatedMS": "MS4_RMS", "SourceIP": "10.101.101.13:3020", "Data": "Car", "VisitedMS": "MS2_CDG_3020", }</pre>
Sent Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> Data: Is the object which is send to the microservice that processes the request.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Error Cases	<p> The message is dropped in the network and the RMS has not received the message. This message will be resend in the next cycle.</p>
Notes & Remarks	<p>The Field class is further described in the MSCF Class Diagram and Description supd.</p>

Title	Send message statistic to Network Monitoring and Maintenance Service
Description and Use Case	<p>After starting the microservice a timer is activated which will trigger a function that sends the database statistic to the NMM every 4 seconds.</p> <p><u>Use case:</u></p>

	<ul style="list-style-type: none"> • The microservice is started on which a timer also starts, which triggers a function every 4 seconds. • That function is getting all the messages from a database copy and sends them to the NMM as a <code>ArrayList<Message></code> in a Message object. • After sending the message the Database copy is cleared, so that the same data is not send again.
Transport Protocol Details	UDP, Client endpoint: / MS3_NMM
Sent Body Example (Client)	<pre>{ "MessageId": " c2217af6-030a-41e1-9edd-e5000ec64dbc", "DestinatedMS": "MS3_RMS", "SourceIP": "10.101.101.13:3020", "Data": "ArraList<Message>", "VisitedMS": "MS2_CDG_3020", }</pre>
Sent Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> Data: Is the object which is send to the microservice that processes the request.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Error Cases	<p> The message has been dropped in the network. It will be resend after 4 seconds.</p>
Notes & Remarks	The Message class is further described in the MSCF Class Diagram and Description supd.

MS3 API Specification

(Network Monitoring and Maintenance Service):

Title	Collecting messages
Description and Use Case	<p>MS3 is a Microservice that will handle visualization of collected data/messages that are received from other microservices. Every microservice collect all messages that it has sent or forwarded and put those messages in a list of messages, and sent a new message to MS3 that message contains that list of messages as an object. MS3 goes through that object(list of messages) and inserts those into its own database. Before presenting those messages from the database on the web in the shape of sequence diagrams, MS3 sorts all messages into successfully received and dropped messages and present them in two different views on the web.</p> <p>Use case:</p> <ul style="list-style-type: none"> • Getting a list of messages from other microservices. Add all messages to the database. • Before presenting messages on the web, sorts all messages into successfully received and dropped messages. • Goes through the sorted list of messages, transforming all of them to String type so that can be read by JS library. • Send Spring boot "modelAndView" that contains a list of transformed messages as strings to the html file. • Goes through all list of transformed strings using thymeleaf, and append those to the JS mermaid <div> that presents string in the form of a diagram.
Transport Protocol Details	<p>UDP, Client endpoint: /drivableFields</p> <p>UDP, Server endpoint: / message ResponseID</p>
Sent Body Example (Client)	<pre>{ "MessageId": " d1873fbe-76a0-4780-8342-5fc03f9a9994", "DestinatedMS": "MS4_RMS", "SourceIP": "10.101.101.13:3020", "Data": "Car", "VisitedMS": "MS2_CDG_3020 MS3_NMM_3021 ", }</pre>

Sent Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> Data: Is the object which is send to the microservice that processes the request.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Response Body Example (Server)	<pre>{ "MessageId": "2ku93r97-2472-4ac1-887c-4sv5149i05c8", "DestinatedMS": "MS2_CDG", "SourceIP": "10.101.101.9:3024", "DestinationIP": "10.101.101.13:3020", "ResponseID": " c2217af6-030a-41e1-9edd-e5000ec64dbc", "Data": "ArrayList<Messages>", "VisitedMS": " MS3_NMM_3024 MS2_CDG_3020 ", }</pre>
Response Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> DestinationIP: The IP address and Port of the microservice, which has send the request.</p> <p> ResponseID: Endpoint indentifier for the micreservice which has send the request. The responseID is the same as the messageID from the message the request was received.</p> <p> Data: The data which has to be processed at the destination.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Error Cases	<p> Invalid data type: Expected data type is List<Messages> in every other case exception is thrown.</p>
Notes & Remarks	

MS4 API Specification (Road Management Service):

Title	Map requested (list of all streets)
Description and Use Case	<p>The microservice MS4 sends a request to get a map object with a list of all available streets and their fields, so that the requester can work with those objects.</p> <p><u>Use case - Send a map request to MS1:</u></p> <ul style="list-style-type: none"> • MS4 sends a request to MS1 to get a map from MS1 • After the response arrives MS4 will get the needed data from the updated Map • The data will be used to display all Streets and Fields on which it is possible for a Car to move
Transport Protocol Details	<p>UDP, Client endpoint: /streets</p> <p>UDP, Server:</p>
Sent Body Example	<pre>{ "MessageId": "89sdf-kajf-71fv-iaj3", "SourceIP": "10.101.101.17:3022", "DestinatedMS": "MS1_ARS", "VisitedMS": " MS4_RMS_2021 " "endpoint": "streets" }</pre>
Sent Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> SourceIP: The IP address of the microservice and the port on which the program is running.</p> <p> DestinatedMS: The unique Microservice ID of the microservice to whom the message was intended.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p> <p> endpoint: Endpoint on MS1 where the message will be received</p>
Response Body Example	<pre>{ "MessageId": "ksj4-er23-ng54-8d7g", "SourceIP": "10.101.101.5:3024", "DestinatedMS": "MS4_RMS", "VisitedMS": " MS1_ARS MS2_CDG MS4_RMS", "Object": "Map", }</pre>

Response Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> SourceIP: The IP address of the microservice and the port on which the program is running.</p> <p> DestinatedMS: The unique Microservice ID of the microservice to whom the message was intended.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p> <p> Object: The data which has to be processed at the destination.</p>
Error Cases	<p> Invalid object: If an Object is received does not match the Object that is expected, an Error message will be printed.</p>

Title	Car management
Description and Use Case	<p>Processing of Cars (CDG instances), which data will be displayed in the CLI</p> <p><u>Use case - Receive Car from MS2 instance:</u></p> <ul style="list-style-type: none"> • MS4 gets a message which has a Car in it and unpacks it. • The car data (carID, route, current location, etc...) will be processed for later uses.
Transport Protocol Details	UDP, Server: instance of Car
Received Body Example	<pre>{ "MessageId": "89sdf-kajf-71fv-iaj3", "SourceIP": "10.101.101.13:3062", "DestinatedMS": "MS4_RMS", "VisitedMS": " MS2_CDG_3022 MS4_RMS_3021" "endpoint": "Car" }</pre>

Received Body Description	<ul style="list-style-type: none"> MessageId: Message identifier which is unique and automatically assigned. SourceIP: The IP address of the microservice and the port on which the program is running. DestinatedMS: The unique Microservice ID of the microservice to whom the message was intended. VisitedMS: List of visited microservices till arrival at the destination. Object: The data which has to be processed at the destination.
Error Cases	<ul style="list-style-type: none"> Invalid object: If an Object is received does not match the Object that is expected, an Error message will printed.

Title	Block area
Description and Use Case	<p>After the user inserted the data about the Area that he wants to block in the CLI, the MS4 sends the post request to MS1 so that he can, if possible, execute the block action.</p> <p>Use case:</p> <ul style="list-style-type: none"> Creates and sends a Street object, in which the specifics Fields are set to notDrivable, in a message object MS1 sends back a message with the same street which is then put into a List for an overview over successfully blocked Areas
Transport Protocol Details	<p>UDP, Client endpoint: /block</p>
Sent Body Example	<pre>{ "MessageId": "h4j1-ada5-jsg4-aj5d", "SourceIP": "10.101.101.17:3022", "DestinatedMS": "MS1_ARS", "VisitedMS": " MS4_RMS MS3_NMM", "endpoint": "block", "Object": "Street", }</pre>
Sent Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> SourceIP: The IP address of the microservice and the port on which the program is running.</p> <p> DestinatedMS: The unique Microservice ID of the microservice to whom the message was intended.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p> <p> endpoint: Endpoint on MS1 where the message will be received</p> <p> Object: Street with the blocked fields.</p>
Response Body Example	<pre>{ "MessageId": "h4j1-ada5-jsg4-aj5d", "SourceIP": "10.101.101.17:3022", "DestinatedMS": "MS4_RMS ", "VisitedMS": " MS1_RMS MS3_NMM MS4_RMS", "endpoint": "block", "Object": "Street", }</pre>

Response Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> SourceIP: The IP address of the microservice and the port on which the program is running.</p> <p> DestinatedMS: The unique Microservice ID of the microservice to whom the message was intended.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p> <p> endpoint: Endpoint on MS1 where the message will be received</p> <p> Object: Street with the blocked fields.</p>
Error Cases	<p> Received Area (Street and the Fields) from CLI is invalid: the StreetName and the field do not exist</p> <p> The area that is supposed to be blocked, is a destination of a car</p>
Notes & Remarks	

Title	UnBlock area
Description and Use Case	<p>the user can see all successfully blocked street names in the cli. After he insets the name of the street that he wants to unblock, all fields in the street will be unblocked and the street will be sent to MS1. If the unblocking succeeds, MS4 gets a message from MS1 with the street, and removes it from the blocked area list</p> <p>Use case:</p> <ul style="list-style-type: none"> • Creates and sends a Street object, in which all Fields are set back to Drivable, in a message object • MS1 sends back a message with the same street which is then removed from a List of successfully blocked Areas
Transport Protocol Details	UDP, Server endpoint: /block

Sent Body Example	<pre>{ "MessageId": "hg41-apd3-j774-98cy", "SourceIP": "10.101.101.17:3022", "DestinatedMS": "MS1_ARS ", "VisitedMS": " MS4_RMS", "endpoint": "block", "Object": "Street", }</pre>
Sent Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> SourceIP: The IP address of the microservice and the port on which the program is running.</p> <p> DestinatedMS: The unique Microservice ID of the microservice to whom the message was intended.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p> <p> endpoint: Endpoint on MS1 where the message will be received</p> <p> Object: Street with the unblocked fields.</p>
Response Body Example	<pre>{ "MessageId": "h4j1-ada5-jsg4-aj5d", "SourceIP": "10.101.101.17:3022", "DestinatedMS": "MS1_ARS", "VisitedMS": " MS1_ARS_3023 MS2_NMM_3024 MS4_RMS_3021", "Object": "Street", }</pre>
Response Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> SourceIP: The IP address of the microservice and the port on which the program is running.</p> <p> DestinatedMS: The unique Microservice ID of the microservice to whom the message was intended.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p> <p> endpoint: Endpoint on MS1 where the message will be received</p> <p> Object: Street with the blocked fields.</p>
Error Cases	<p> Invalid object: If an Object is received that does not match the Object that is expected, an Error message will be printed.</p>
Notes & Remarks	

Title	Car Routes
Description and Use Case	<p>The user chooses the option 1 int the CLI after that all Cars and their current routes will be displayed</p> <p>Use case:</p> <ul style="list-style-type: none"> • The user chooses 1 in the CLI • All Cars and their routes as coordinates are displayed
Transport Protocol Details	REST, GET endpoint: /carRoutes
Response Body Example	<pre>{ "CarId": "3026", "Coordinates": [{ "x": "0", "y": "1", } { "x": "0", "y": "2" }] "CarId": "3021", "Coordinates": [{ "x": "7", "y": "1", } { "x": "8", "y": "12" }] }</pre>
Response Body Description	<p>! carId: ID of the car</p> <p>! Coordinates: Route will be presented over the coordinates of the fields that the car has to pass to get to the destination</p>
Error Cases	<p>There is no Car data yet</p> <p>MS4 is not online</p>

Title	Car Progress
Description and Use Case	<p>Displaying the number of travelled Field and number of to be traveled Field until current destination.</p> <p>Use case:</p> <ul style="list-style-type: none"> • The user chooses 2 in the CLI • All Car ids are displayed. • One Car ID can be inserted after which the numbers will be displayed
Transport Protocol Details	<p>REST, GET endpoint: /carIDs</p> <p>REST, GET endpoint: /progress/{id}</p>
Response Body Example	<pre>{ ["CarId": "3021", "CarId": "3022", "CarId": "3023",] } { "travelled ": "5" "stillToTravel": "6" }</pre>
Response Body Description	<p> carId: ID of the car</p> <p> travelled: number of travelled Field since car started traveling on this route</p> <p> stillToTravel: number of Field to the current destination</p>
Error Cases	<p>There is no Car data yet</p> <p>MS4 is not online</p>

Title	All Streets and Field Coordinates
Description and Use Case	<p>Displays all street names, number [from and to] which are needed to be inserted when blocking, and all field coordinates</p> <p>Use case:</p> <ul style="list-style-type: none"> •
Transport Protocol Details	REST, GET endpoint: /allStreets
Response Body Example	<pre>{ "streetName": "2nd_avenue", "Coordinates": [{ "x": "0", "y": "1", }, { "x": "0", "y": "2" }] "streetName": "währinger", "Coordinates": [{ "x": "7", "y": "1", }, { "x": "8", "y": "1" }] }</pre>
Response Body Description	<p> streetName: Name of the Street as a String</p> <p> Coordinates: Street will be presented over the coordinates of the fields that the car has to pass to get to the destination</p>
Error Cases	<p>There is no Map data yet</p> <p>MS4 is not online</p>

Title	Block Area
Description and Use Case	<p>The user can block part of a street that he wants</p> <p>Use case:</p> <ul style="list-style-type: none"> • The user choses the option to block in the CLI • All Streets, options to block, Fields of the streets as coordinate, Cars that are currently on the map and their Routes are shown in the console • The user enters a Street Name, FROM and TO fields as integers (2nd_Avenue-4-8) • If the Street with that name exists, the entered Field will be blocked, and the street sent to MS1 • If successfully blocked the street will be added into a blocked area list
Transport Protocol Details	REST, POST endpoint: /blockArea/{streetName}/{von}/{to}
POST Body Example	<pre>{ "streetName": "2nd_avenue", "von": "2" "to": "8" }</pre>
Sent Body Description	<p> streetName: Name of the Street as a String</p> <p> von: field number from which the blocking starts</p> <p> to: field number where the blocking ends</p>
Response Body Example	<pre>{ "response": "Was sent to be Blocked" }</pre>
Response Body Description	response: returns if the blocking was sent or not
Error Cases	<p>Street Name if wrongly entered or does not exist.</p> <p>From and To are smaller than 0 or bigger than the actual street size.</p>

Title	Unblock Area
Description and Use Case	<p>The user can unblock a successfully blocked street</p> <p>Use case:</p> <ul style="list-style-type: none"> • The user choses the option to unblock in the CLI • All Streets names will be displayed that are successfully blocked • The user enters a Street Name • If the Street with that name exists, the Street with the unblocked Fields will be sent to MS1 • If successfully unblocked the street will be removed from the blocked area list
Transport Protocol Details	<p>REST, GET endpoint: /blockedStreets</p> <p>REST, POST endpoint: /unblock/{street}</p>
GET Body Example	<pre>{ "streetName": "2nd_avenue", "streetName": "3rd_avenue", "streetName": "Mariahilfer" }</pre>
POST Body Example	
Sent Body Description	streetName: Name of the Street as a String
POST Body Response Example	<pre>{ "response": "Was sent to be UnBlocked" }</pre>
POST Body Response Description	response: returns if the Unblocking was sent or not
Response Body Description	response: returns if the blocking was sent or not

Error Cases	Street Name entered that is not blocked
-------------	---

Title	Send message statistic to Network Monitoring and Maintenance Service
Description and Use Case	<p>After starting the microservice a timer is activated which will trigger a function that sends the database statistic to the NMM every 4 seconds.</p> <p><u>Use case:</u></p> <ul style="list-style-type: none"> • The microservice is started on which a timer also starts, which triggers a function every 4 seconds. • That function is getting all the messages from a database copy and sends them to the NMM as a <code>ArrayList<Message></code> in a Message object. • After sending the message the Database copy is cleared, so that the same data is not send again.
Transport Protocol Details	UDP, Client endpoint: / MS3_NMM
Sent Body Example (Client)	<pre>{ "MessageId": " c2217af6-030a-41e1-9edd-e5000ec64dbc", "DestinatedMS": "MS3_RMS", "SourceIP": "10.101.101.17:3027", "Data": "ArraList<Message>", "VisitedMS": "MS4_RMS_3027", }</pre>
Sent Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> Data: Is the object which is send to the microservice that processes the request.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>

Response Body Example (Server)	<pre>{ "MessageId": "2cb93f97-3833-4ab1-974c-4be5149a05c8", "DestinatedMS": "MS2_CDG", "SourceIP": "10.101.101.9:3024", "DestinationIP": "10.101.101.13:3020", "ResponseID": " c2217af6-030a-41e1-9edd-e5000ec64dbc", "Data": "", "VisitedMS": " MS3_NMM_3024 MS4_RMS_3020 ", }</pre>
Response Body Description	<p> MessageId: Message identifier which is unique and automatically assigned.</p> <p> DestinatedMS: Endpoint identifier for the destination microservice.</p> <p> SourceIP: The IP address and Port of the microservice which generated the message.</p> <p> DestinationIP: The IP address and Port of the microservice, which has send the request.</p> <p> ResponseID: Endpoint indentifier for the micreservice which has send the request. The responseID is the same as the messageID from the message the request was received.</p> <p> Data: The data which has to be processed at the destination.</p> <p> VisitedMS: List of visited microservices till arrival at the destination.</p>
Error Cases	<p> Invalid data type: Expected data type is Car in every other case exception is thrown.</p>
Notes & Remarks	<p>The Field class is further described in the MSCF Class Diagram and Description.</p>

How To:

MS1 ARS

Project should be imported as an existing Maven Project and "Run as Java Application". If the "DSE-VPN" is not connected Exception will be thrown. This app not intended for interaction with an end- user so there is no GUI and the interaction with other MSs can be examined via Console on IDE(Eclipse e.g.).

```

Sending statistics...
Sending statistics...
Setting drivable fields for MS2_CDG...
Setting drivable fields for MS2_CDG...
Setting drivable fields for MS2_CDG...
Setting drivable fields for MS2_CDG...
Setting car route to return for MS2_CDG...
Route Worker started...
+++++
x= 9 y=7:  REceived DESTINATIONCAR: 3029
+++++
x= 7 y=1:  REceived PositionCAR: 3029
+++++
Received route
+++++
ID: 3029 is registered now
*****
x= 9 y=7:  CAR IN REGISTER DESTINATION
x= 7 y=1:  CAR IN REGISTER POSITION
*****
My Route for car
*****
Current position for car : 3029 is x= 7 y=1
New route for car id 3029:
x= 7 y=2
x= 7 y=3
x= 7 y=4
x= 7 y=5
x= 7 y=6
x= 7 y=7
x= 8 y=7
x= 9 y=7
Reseting Connection...
Sending statistics...
Setting car route to return for MS2_CDG...
Route Worker started...

```

In order to conduct Junit tests just right click on the test source folder and run as Junit.

MS2 CDG

```

workspaceDSE - ms2/src/ms2/mainCDG.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer
ms2
  src
    controllers
    exceptions
    model
    ms2
    receiver
    statistic_sender
  test
  JRE System Library [JavaSE-11]
  JUnit 4
  Maven Dependencies
  bin
  library
  target
  pom.xml
ms3
mscf

mainCDG.java
10 public class mainCDG {

Console
CarController.java
<terminated> mainCDG [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (9 Jun 2020, 19:39:38)
Starting client
Server constructor on port 3022
Car 3022 setting position and destination...
# MESSAGE: 56cc1923-b9d2-4c40-9109-7aaf2347dc76 # ----- > ARS Fields requested.

Connected peers : [10.101.101.17:3022 | 10.101.101.5:3023]

# MESSAGE RESPONSE: 56cc1923-b9d2-4c40-9109-7aaf2347dc76 #----- > ARS Fields received.

##### Car 3022 position field::Sx= 8 y=7
##### Car 3022 destination field:Sx= 1 y=4

# MESSAGE: 19169262-ea55-454a-83e2-8196f7ac71f9 # ----- > ARS Route requested.
# MESSAGE: 65336b5e-df44-4ecd-9e96-2af05dd226f4 # ----- > NMM Statistic send.
# MESSAGE RESPONSE: 19169262-ea55-454a-83e2-8196f7ac71f9 #----- > New ARS route received.

##### Car3022 is now traveling #####

##### Estimated travel time: 90seconds #####

# MESSAGE: 84b41d5f-9519-48f1-8698-0ea46d254d87 #----- > RMS informed.
# MESSAGE: 228e3c45-076c-4704-807f-5864345473f1 # ----- > ARS Route requested.
# MESSAGE RESPONSE: 228e3c45-076c-4704-807f-5864345473f1 #----- > New ARS route received.

```

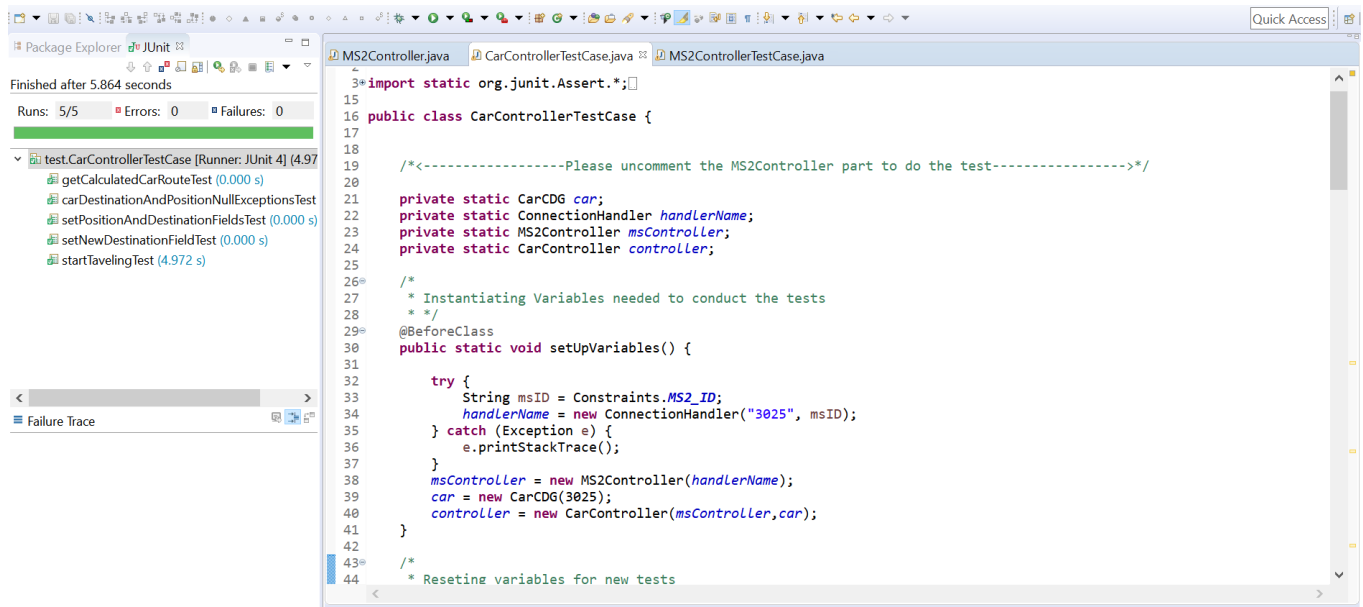
In order to test the CDG project it has to be imported as an existing Maven Project and Run as Java Application. To start multiple CDG at once no new port number has to be set, because the application chooses autonomous and randomly between port numbers from 3020 to 3030 and repeats the process till a free port has been found. Therefore, just Run the same app as much times as CDGs are needed but not more than ten times. If the "DSE-VPN" is not connected an Exception will be thrown. To avoid other errors or misbehaviours all CDG must have different port numbers to be differentiated by the UDP Server. This app not intended for interaction with an end-user so there is no GUI and the interaction with other MSs can be examined via Console on IDE (Eclipse e.g.).

```

Console
CarController.java
<terminated> mainCDG [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (9 Jun 2020, 19:41:53)
Connection constructor
Starting client
Server constructor on port 3023
<----- UDP Server Exception, Port in use ----->
Restarting process!
Connection constructor
Starting client
Server constructor on port 3023
<----- UDP Server Exception, Port in use ----->
Restarting process!
Connection constructor
Starting client
Server constructor on port 3022
Car 3022 setting position and destination...
# MESSAGE: 4616ed8c-51ee-4c59-b766-579640b1c15f # ----- > ARS Fields requested.
# MESSAGE: 40e671f4-c864-44fd-9634-034be8a8c585 # ----- > NMM Statistic send.
# MESSAGE: 9a61edd3-fd90-4a4b-9f8a-82bdfbf7df15 # ----- > NMM Statistic send.
Resetting Connection...
# MESSAGE: 82b31655-3c43-44e5-a15a-cbcb5dbb234 # ----- > NMM Statistic send.
# MESSAGE REQUEST: 4616ed8c-51ee-4c59-b766-579640b1c15f #----- > Timeout request resending.
# MESSAGE: 4aaf6f90-0957-4b17-b02e-513861786745 # ----- > ARS Fields requested.

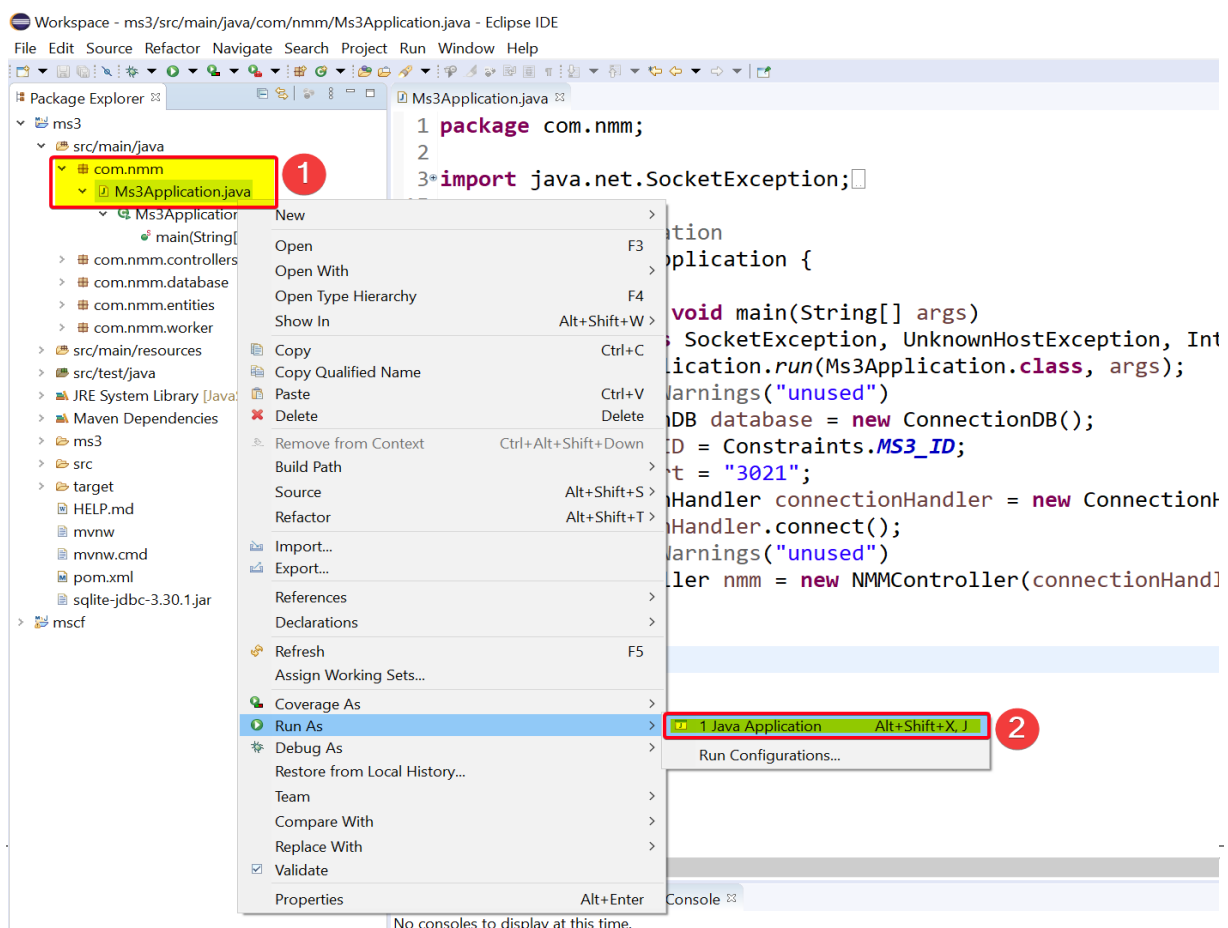
```

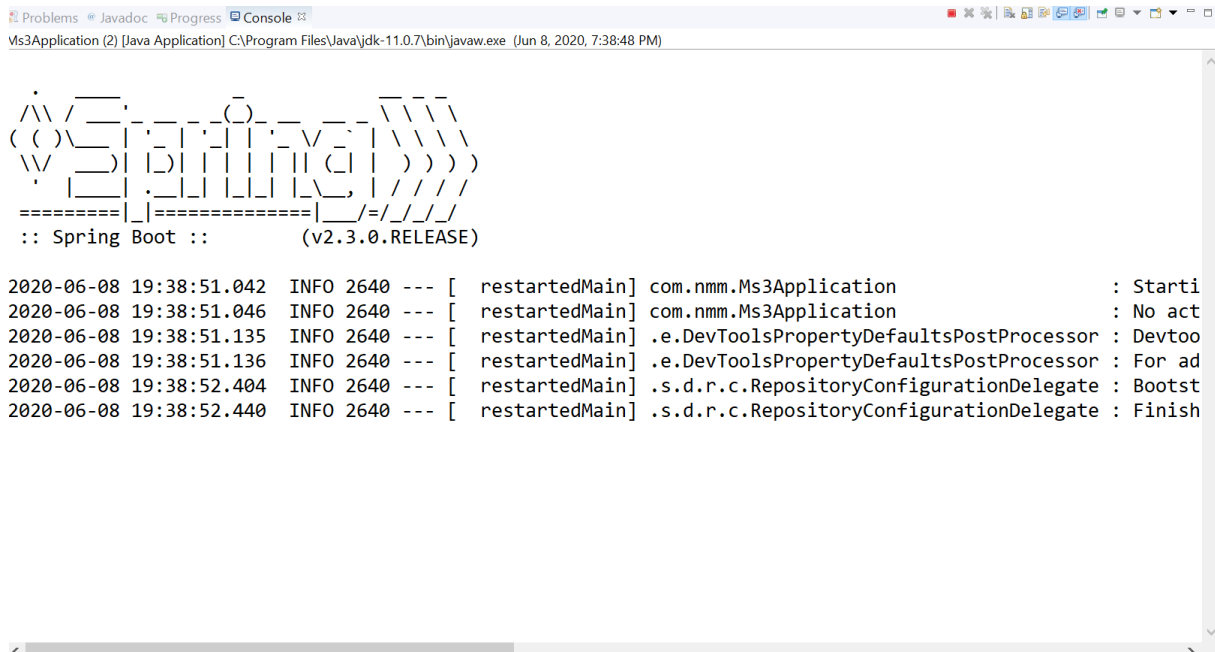
In order to conduct the JUnit tests in the test folder, specific parts in the MS2Controller (getDrivableFields and get carRoute method) have to be un-commented. Further description can be found in the MS2Controller class.



MS3 NMM

1.Run the application as a java application! It will start the Spring Boot application





```

:: Spring Boot :: (v2.3.0.RELEASE)

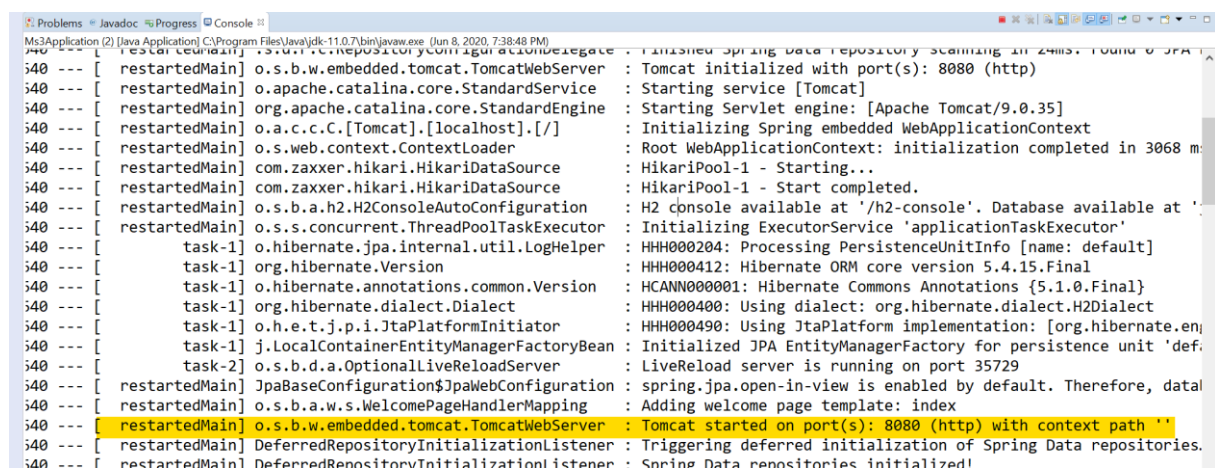
2020-06-08 19:38:51.042 INFO 2640 --- [ restartedMain] com.nmm.Ms3Application : Starti
2020-06-08 19:38:51.046 INFO 2640 --- [ restartedMain] com.nmm.Ms3Application : No act
2020-06-08 19:38:51.135 INFO 2640 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtoo
2020-06-08 19:38:51.136 INFO 2640 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For ad
2020-06-08 19:38:52.404 INFO 2640 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootst
2020-06-08 19:38:52.440 INFO 2640 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finish

```

2. Spring Boot will run tomcat web server on port 8080.

Open the web browser (Firefox, Chrome, Opera, Explorer).

Go to <http://localhost:8080> or <http://127.0.0.1:8080> to start the web application.

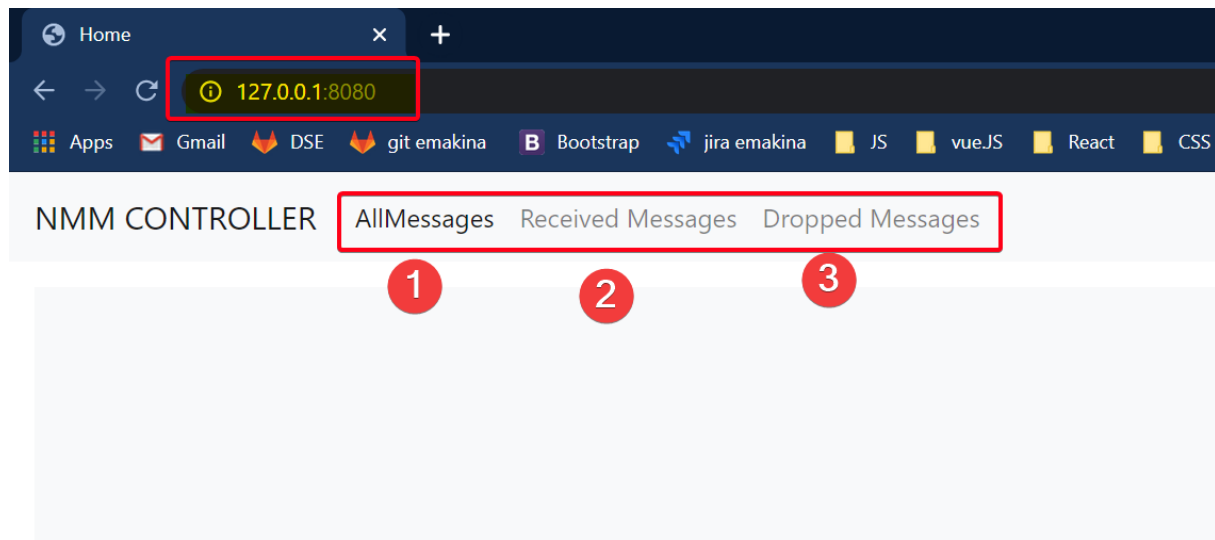


```

i40 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
i40 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
i40 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.35]
i40 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
i40 --- [ restartedMain] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 3068 ms
i40 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
i40 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
i40 --- [ restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at '
i40 --- [ restartedMain] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
i40 --- [ task-1] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
i40 --- [ task-1] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.4.15.Final
i40 --- [ task-1] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
i40 --- [ task-1] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
i40 --- [ task-1] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.en]
i40 --- [ task-1] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'def.
i40 --- [ task-2] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
i40 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, dataa
i40 --- [ restartedMain] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page template: index
i40 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
i40 --- [ restartedMain] DeferredRepositoryInitializationListener : Triggering deferred initialization of Spring Data repositories.
i40 --- [ restartedMain] DeferredRepositoryInitializationListener : Spring Data repositories initialized!

```

3. In the navbar are three links. All messages display list of all messages that are exchanged between microservices, received messages display all successfully received messages, and drop messages display all dropped messages.



NMM CONTROLLER **AllMessages** Received Messages Dropped Messages

ALL MESSAGES

MessageID: 0 Source: 10.101.101.5:3021 Destination: MS2_CDG Visited [MS1_CDG_3013 MS4_RMS_3013 MS2_CDG_3013]

MessageID: 1 Source: 10.101.101.5:3021 Destination: MS2_CDG Visited [MS1_CDG_3013 MS4_RMS_3013 MS7_RMS_3013 MS8_RMS_3013 MS2_CDG_3013]

MessageID: 2 Source: 10.101.101.5:3021 Destination: MS2_CDG Visited [MS2_CDG_3013 MS4_RMS_3013]

MessageID: 3 Source: 10.101.101.13:3021 Destination: MS1_ARS Visited [MS2_CDG_3021 MS1_ARS_3024]

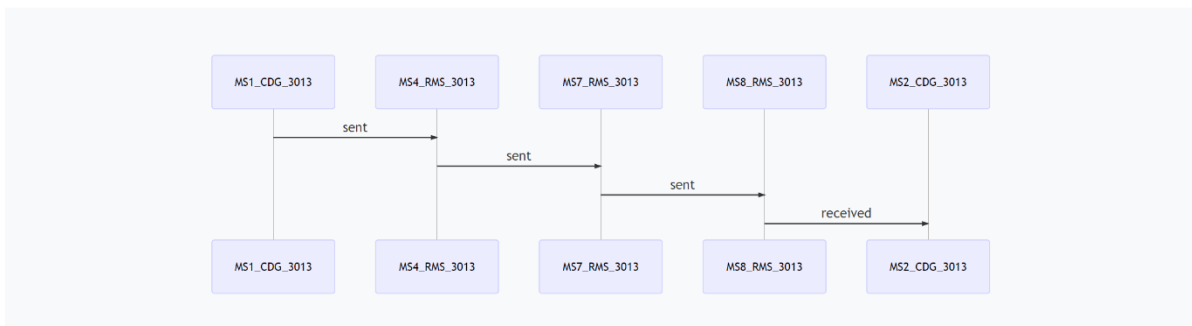
MessageID: 4 Source: 10.101.101.13:3021 Destination: MS1_ARS Visited [MS2_CDG_3021 MS4_RMS_3013 MS4_RMS_3013]

MessageID: 5 Source: 10.101.101.13:3021 Destination: MS1_ARS Visited [MS2_CDG_3021 MS2_CDG_3021 MS2_CDG_3021]

NMM CONTROLLER AllMessages **Received Messages** Dropped Messages

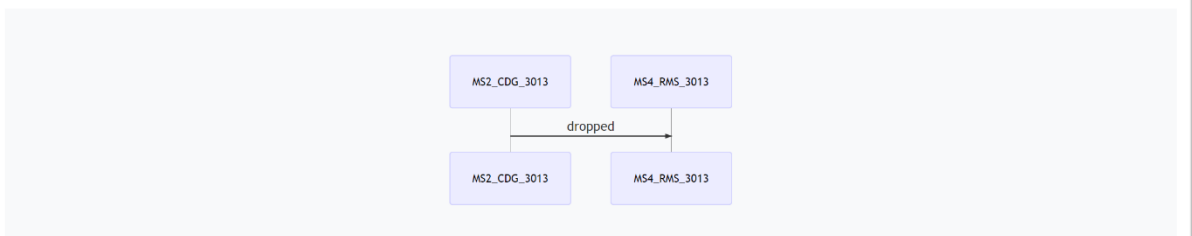
RECEIVED MESSAGES

MessageID: 1

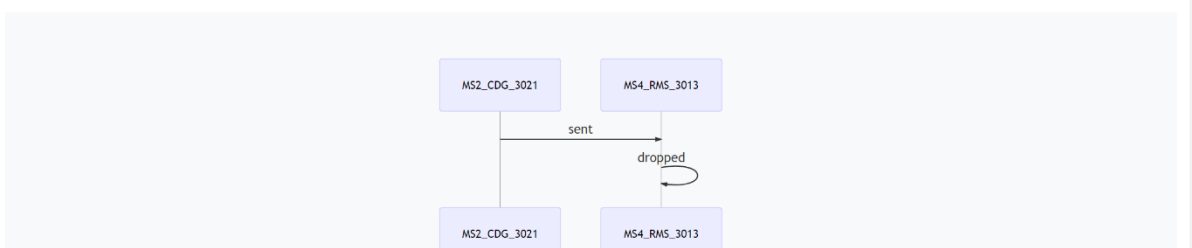
NMM CONTROLLER AllMessages Received Messages **Dropped Messages**

DROPPED MESSAGES

MessageID: 2



MessageID: 5



4. In the console, the user can follow how the messages are coming to the microservice, how many messages are in the list, and in which table in the database the messages are inserted. Classical debugging.

```

Ms3Application (2) [Java Application] C:\Program Files\Java\jdk-11.0.7\bin\javaw.exe (Jun 8, 2020, 7:38:48 PM)
*****
INSERT TO MS3 -> DIAGRAMS: Message [id=0, messageID=1, source=10.101.101.5:3021, destination=10.101.101.13
*****
ADD MESSAGE TO DIAGRAMS
*****
INSERT TO MS3 -> DIAGRAMS: Message [id=1, messageID=1, source=10.101.101.5:3021, destination=10.101.101.13
*****
ADD MESSAGE TO DIAGRAMS
*****
INSERT TO MS3 -> DIAGRAMS: Message [id=2, messageID=2, source=10.101.101.5:3021, destination=10.101.101.13
*****
ADD MESSAGE TO DIAGRAMS
*****
INSERT TO MS3 -> DIAGRAMS: Message [id=3, messageID=4, source=10.101.101.13:3021, destination=null, destir
*****
ADD MESSAGE TO DIAGRAMS
*****
INSERT TO MS3 -> DIAGRAMS: Message [id=4, messageID=5, source=10.101.101.13:3021, destination=null, destir
*****
ADD MESSAGE TO DIAGRAMS
*****
INSERT TO MS3 -> DIAGRAMS: Message [id=5, messageID=6, source=10.101.101.13:3021, destination=null, destir
*****
ADD MESSAGE TO DIAGRAMS

```

```

2020-06-08 20:52:05.094 INFO 14896 --- [ task-1] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons
2020-06-08 20:52:05.259 INFO 14896 --- [ task-1] org.hibernate.dialect.Dialect : HH0000400: Using dialect: org.
2020-06-08 20:52:05.560 INFO 14896 --- [ task-1] o.h.e.t.j.p.i.JtaPlatformInitiator : HH0000490: Using JtaPlatform i
2020-06-08 20:52:05.572 INFO 14896 --- [ task-1] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerF
2020-06-08 20:52:05.639 INFO 14896 --- [ task-2] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running c
2020-06-08 20:52:05.894 WARN 14896 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is ena
2020-06-08 20:52:06.219 INFO 14896 --- [ restartedMain] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page template:
2020-06-08 20:52:07.158 INFO 14896 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 808
2020-06-08 20:52:07.162 INFO 14896 --- [ restartedMain] DeferredRepositoryInitializationListener : Triggering deferred initializa
2020-06-08 20:52:07.164 INFO 14896 --- [ restartedMain] DeferredRepositoryInitializationListener : Spring Data repositories initi
2020-06-08 20:52:07.193 INFO 14896 --- [ restartedMain] com.nmm.Ms3Application : Started Ms3Application in 6.14

MESSAGE TABLE CREATED IN DATABASE
DIAGRAMS TABLE CREATED IN DATABASE
DROP MESSAGES TABLE CREATED IN DATABASE

TABLES CLEARD!

-----
Connection constructor
Starting client
Server constructor on port 3021
NMMController loading messages...
Resetting Connection...
Resetting Connection...
Resetting Connection...
Resetting Connection...
Resetting Connection...
***** Message size: 0
----- ON MESSAGE -----
GETING MESSAGE
RECEIVED MESSAGE: message ##fe59ab0d-16d0-4cfa-ad68-7566cb3a197f## source: 10.101.101.17:3022## destinedMS MS3_NMM## visitedMS:
LIST SIZE:##### 0
Resetting Connection...

```

CLI and MS4 RMS – Road Management System

After the CLI is started a server on the port 8081 runs, and in the console appear 5 options.

```

CliApplication [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (09.06.2020, 13:20:30)

:: Spring Boot :: (v2.2.6.RELEASE)

2020-06-09 13:20:32.108 INFO 12724 --- [main] com.dse.cli.CliApplication : Starting CliApplication on Adem with PI
2020-06-09 13:20:32.116 INFO 12724 --- [main] com.dse.cli.CliApplication : No active profile set, falling back to
2020-06-09 13:20:34.764 INFO 12724 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8081 (
2020-06-09 13:20:34.795 INFO 12724 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-06-09 13:20:34.795 INFO 12724 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat
2020-06-09 13:20:34.972 INFO 12724 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicat
2020-06-09 13:20:34.973 INFO 12724 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initializat
2020-06-09 13:20:35.273 INFO 12724 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicati
2020-06-09 13:20:35.671 INFO 12724 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8081 (http)
2020-06-09 13:20:35.683 INFO 12724 --- [main] com.dse.cli.CliApplication : Started CliApplication in 4.397 seconds

Enter an Option
1 - Display all Car and their Routes
2 - Display Travel Progress of a car
3 - Block Area
4 - Un-Block Area
0 - Exit

Type an option: 1

```

Option 1 – If “1” is inserted, in the console will be displayed the carId and his route, which is represented as coordinated of fields, from his current position to his destination, and can chose again from the first option list.

```

Task List  Javadoc  Declaration  Console  Coverage
CliApplication [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (09.06.2020, 13:20:30)

2020-06-09 13:20:34.972 INFO 12724 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2020-06-09 13:20:34.973 INFO 12724 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization complete
2020-06-09 13:20:35.273 INFO 12724 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecuter
2020-06-09 13:20:35.671 INFO 12724 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8081 (http) with context
2020-06-09 13:20:35.683 INFO 12724 --- [main] com.dse.cli.CliApplication : Started CliApplication in 4.397 seconds (JVM running

Enter an Option
1 - Display all Car and their Routes
2 - Display Travel Progress of a car
3 - Block Area
4 - Un-Block Area
0 - Exit

Type an option: 1
Car: 3024
Route: [{x=13, y=11}, {x=13, y=12}]
Car: 3021
Route: [{x=11, y=3}, {x=10, y=3}, {x=9, y=3}, {x=8, y=3}, {x=7, y=3}, {x=6, y=3}, {x=5, y=3}, {x=4, y=3}, {x=3, y=3}, {x=2, y=3}, {x=1, y=3}, {x=1, y=2}.
Car: 3022
Route: [{x=7, y=2}, {x=7, y=3}, {x=7, y=4}, {x=7, y=5}, {x=7, y=6}, {x=7, y=7}, {x=8, y=7}, {x=9, y=7}]
Car: 3023
Route: [{x=9, y=7}, {x=8, y=7}, {x=7, y=7}, {x=6, y=7}, {x=5, y=7}, {x=4, y=7}, {x=3, y=7}, {x=2, y=7}, {x=1, y=7}, {x=1, y=6}, {x=1, y=5}, {x=1, y=4}, {x=1, y=3}
Enter an Option
1 - Display all Car and their Routes
2 - Display Travel Progress of a car
3 - Block Area
4 - Un-Block Area
0 - Exit

Type an option:

```

If the map data from ms1 to ms4 still has not arrived or ms4 is offline, and error message will be shown.

```

Enter an Option
1 - Display all Car and their Routes
2 - Display Travel Progress of a car
3 - Block Area
4 - Un-Block Area
0 - Exit

Type an option: 1
|
MS4 is Offline or Map has not arrived!

```

Option 2 – If “2” is inserted there will be shown all CarIDs, their number of traveled Fields from the start of the current route and the number of Fields it still has to the current Detination.

```

Enter an Option
1 - Display all Car and their Routes
2 - Display Travel Progress of a car
3 - Block Area
4 - Un-Block Area
0 - Exit

Type an option: 2
Car: 3027 Has traveled: 2 and has: 6 Fields to travel on current Route!
Car: 3020 Has traveled: 2 and has: 4 Fields to travel on current Route!

```

If ms4 is offline, and error message will be shown.

```

Enter an Option
1 - Display all Car and their Routes
2 - Display Travel Progress of a car
3 - Block Area
4 - Un-Block Area
0 - Exit

Type an option: 2

There are NO Cars!

```

Option 3 – Block Area displays all Streets and their Field as Coordinates, and all the cars and car Routes and option 1.

It is bit trickie to follow the car Routes and the Fields if it is wantes to block a fiel of on which the Cars will arrive.

The user needs to insert a Street Name and two numbers (Example: Pine_street-2-5) with a minus inbetween “-”. A POST request is sent to MS4 and afterwars to MS1. If the values FROM and TO are larger than the Street size or smaller than 0 an Error will be thrown.

```

Street Name: Durham_Road, Enter between FROM 0 TO 8
Field Number: 0 Coordinate: {x=5, y=3}| Field Number: 1 Coordinate: {x=5, y=4}| Field Number: 2 Coordinate: {x=5, y=5}|
Field Number: 3 Coordinate: {x=5, y=6}| Field Number: 4 Coordinate: {x=5, y=7}| Field Number: 5 Coordinate: {x=5, y=8}|
Field Number: 6 Coordinate: {x=5, y=9}| Field Number: 7 Coordinate: {x=5, y=10}

Car: 3024 Route: [{x=1, y=3}, {x=1, y=2}, {x=1, y=1}, {x=2, y=1}, {x=3, y=1}, {x=4, y=1}]
Car: 3026 Route: [{x=9, y=9}, {x=9, y=10}, {x=9, y=11}, {x=10, y=11}, {x=11, y=11}, {x=11, y=12}, {x=11, y=13}, {x=12, y=13}, {x=13, y=13}]
Car: 3021 Route: [{x=7, y=12}, {x=7, y=13}]
-----
Insert a STREET NAME the FROM and TO fields as NUMBERS with - between
EXAMPLE 2nd_Avenue-4-6
Break - To go back
Type an option:
-----
Dogwood_Lane-10-11
Area was sent to be blocked
Insert a STREET NAME the FROM and TO fields as NUMBERS with - between
EXAMPLE 2nd_Avenue-4-6
Break - To go back
Type an option:
-----

```

Option 4 – Unblock Area will show the names of all the unblocked Areas. The user can enter a name which will send a unblocked Street to ms1. If the unblocking succeeds the street will be removed from the blocked areas.

```

Enter an Option
1 - Display all Car and their Routes
2 - Display Travel Progress of a car
3 - Block Area
4 - Un-Block Area
0 - Exit

Type an option: 4
#### Successfully Blocked Streets ####

Street Name: Berkshire_Drive
-----

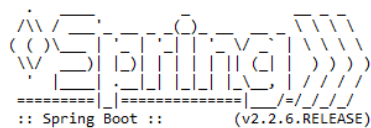
Insert a Street Name that should be Un-Blocked
Break - To go back
Type an option:
Berkshire_Drive

```

MS4 – Road Management system

After running the program as java app the program will be available on UDP server on port 3022, and a Tomcat server on port 8084 for the CLI to be available.

Ms4Application [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (09.06.2020, 14:17:35)



```

:: Spring Boot :: (v2.2.6.RELEASE)

2020-06-09 14:17:37.973 INFO 2424 --- [main] com.dse.ms4.Ms4Application : Starting Ms4Application on Adem with PID
2020-06-09 14:17:37.982 INFO 2424 --- [main] com.dse.ms4.Ms4Application : No active profile set, falling back to default
2020-06-09 14:17:40.116 INFO 2424 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8084 (l
2020-06-09 14:17:40.131 INFO 2424 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-06-09 14:17:40.131 INFO 2424 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat
2020-06-09 14:17:40.274 INFO 2424 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplica
2020-06-09 14:17:40.274 INFO 2424 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initializat

Connection constructor
Starting client
Server constructor on port 3022
Resetting Connection...
2020-06-09 14:17:41.592 INFO 2424 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicati
IPADDRESS: 10.101.101.5:3021
IPADDRESS: 10.101.101.13:3026
2020-06-09 14:17:42.899 INFO 2424 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8084 (http
2020-06-09 14:17:43.073 INFO 2424 --- [main] com.dse.ms4.Ms4Application : Started Ms4Application in 6.234 seconds
  
```

With a right click and Run as JUnit test the tests for MS4 will start. There could be Error with other Microservices because the the Test create additional Databases because there was an Error that the Port is already in use.

