ILP CW2 / CW3 / CW4 Specifications

01.11.2025

Background information (details to follow later in the document):

- **CW2** is designed as a logical continuation of CW1 by adding new endpoints and implementations. The focus is on evaluation which drones are available, dynamic allocation of deliveries to drones to routes and route-finding options.

  Marking is done using the auto-marker and you can receive 33 points max.

  Important: As marking is based on ilp_submission_image.tar (docker image), any missing or incorrect image will cause 8 points deduction of the CW2 mark. Should it be impossible to build your solution based on your submitted sources, 0 points will be awarded.
  To make sure that you can prove your progress and submission creation in case of dispute or need to demonstrate, please use github and maintain the COMMIT-history.

- **CW3** provides an opportunity for you to create anything you consider important as an addition to ILP. You are free to use AI, any language of choice and any environment.

  The marking will be based on a submitted document cw3_explanation.pdf in PDF format not exceeding 1,000 words and is done by 3 parallel markers (using the average mark) and 20 points are available

  Important: Without this PDF explanation and the submitted sources to build your solution CW3 cannot be marked and will be considered 0 points.

- **CW4** is the video presentation for CW3 where you showcase your solution with a short demo, some background and some implementation details.

  Video length is 5-7 minutes. Marking is done by 3 parallel markers (using the average mark) and 12 points are available. The rubric can be found later in this document

  Important: Without a CW3 submission CW4 cannot be marked and will be considered 0 points.

AI usage is allowed in full for CW3, and as a conceptual tool in CW4 (no content generation!). For CW2 no AI is allowed whatsoever.

CW2

Your CW2 tasks can be summarized as follows:

- Extend the Java-REST-Service built in CW1
  - Must be in Java
  - Implement the necessary endpoints for the POST and GET requests (see below)
  - JSON handling
  - You are only supposed to use 200 as return code as all requests are valid as such. Should a request not be completable, use the corresponding result JSON structures

- Place the service in a docker image
  -

- save the docker image in a file called **ilp_submission_image.tar**
  (it is in TAR format anyhow)

- **place** the file **ilp_submission_image.tar into your root directory of your solution**

  Your directory would look something like this:

  ilp_submission_2 *(your root directory)*
      ilp_submission_image.tar
      src (the Java sources…)
          main
              …
          …

- **Create a ZIP file of your solution directory**
  - Image
  - Sources
  - IntelliJ (or whatever IDE you are using) project files

- upload the ZIP as your submission in Learn

Important general information:
During implementation you can use the ILP-service endpoint
https://ilp-rest-2025-bvh6e9hschfagrgy.ukwest-01.azurewebsites.net/
As we will change this for auto-marking, your service must accept a different endpoint to be used in an environment variable **ILP_ENDPOINT**.

Normally, you would use a class annotated with @Configuration for that, which includes a method annotated with @Bean to return the current value as a String. You can retrieve the value of an environment variable using **System.*getenv*(...) - and provide a default value to use whilst no different value is set.**

Make sure you retrieve all necessary data for your calculations and decisions every time you start processing an endpoint. *During the call in the endpoint data won't change, yet it could between 2 calls.*

A **`medicine dispatch record`** (called MedDispatchRec in future) has the following JSON structure (actual data is just for demonstration purpose and will change):

```
{
"id": 123,
"date": "2025-12-22",
"time": "14:30",
"requirements": {
     "capacity": 0.75,
"cooling": false,
"heating": true,
"maxCost": 13.5
}
}
```

The only required elements are id and capacity. All others are optional and only used if present. Cooling / heating are either / or (you can either have cooling or heating) and if not present are false.

MaxCost is only considered when present and must include the initial and final cost pro rata for the flight. To ease the calculation, the total moves flown for all deliveries multiplied by the cost / move are cost basis as well and will be distributed pro rata, too.

If 3 deliveries are made for a flight with a total of 1,200 moves, then each delivery carries 1/3 of these costs.

All API-calls start with /api/v1

The REST-Service must provide the following endpoints:

1) All entry points as defined by CW1 in **the correct** form. **You might have to fix something depending on the feedback you will receive for CW1.**

2) **`Static Queries`**

This group contains fixed queries for fixed attributes.

a) `dronesWithCooling/{state} (GET)`

Return a [] of drone-ids which support cooling (state is true) or not (false)

Example:

`/api/v1/dronesWithCooling/true`

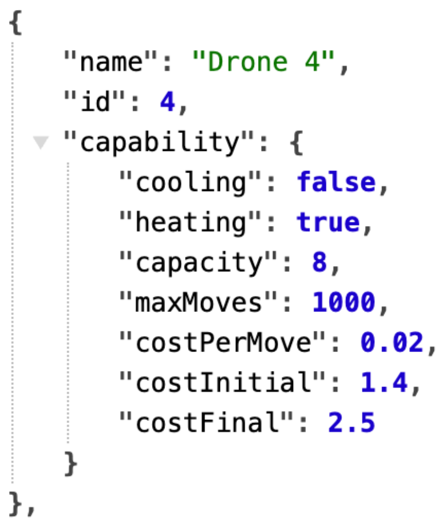
b) `droneDetails/id (GET)`

Return the JSON object for a single drone with the given id.

This is an exception to the 200 code only rule as if an invalid (or non-existent) id is passed in, a 404 is to be returned. In general, when using direct ids and make those available to the outside world, far more stringent checking is necessary and 404 becomes truly useful.

An example for id 4 could be:

`/api/v1/droneDetails/4`



```
{
    "name": "Drone 4",
    "id": 4,
    "capability": {
        "cooling": false,
        "heating": true,
        "capacity": 8,
        "maxMoves": 1000,
        "costPerMove": 0.02,
        "costInitial": 1.4,
        "costFinal": 2.5
    }
},
```

3) **`Dynamic Query`**

a) `queryAsPath/attribute-name/attribute-value` (GET)

You are supposed to return an [] of drone-ids which have their attribute with the given value **as path-variables in the URL**. In the example for Id #4 above, this could be:

```
/api/v1/queryAsPath/capacity/8
```

This way you can only query 1 attribute and compare with 1 value.
The comparison is equals (=)

You must take attribute-name and value as strings, find the corresponding JSON-element and compare accordingly.

b) `query (POST)`

In this POST request the incoming query-attribes are passed as a [] of query attributes (like the following example):

```
[
	{
		“attribute”: “capacity”,
		“operator”: “<”,
		“value”: “8”
	},
	{
		“attribute”: “cooling”,
		“operator”: “=”,
		“value”: “true”
	},
]
```

You are supposed to return an [] of drone-ids which have their attribute(s) with the given value(s) **according to the passed in POST-data**.

This way you can query as many attributes as you like (joined by AND) and compare them always with 1 value.

Everything is a string -> numerical values must be cast, and you must convert the operator accordingly as well. The operators are equals (=) for most cases, except numerical attributes, where any operator in the range (=, !=, <, >) can be applied.

You must take attribute-name and value as strings, find the corresponding JSON-element and compare accordingly.

4) **`Drone availability queries`** (POST)

You are always supposed to return a [] or drone ids, if there are any drones matching the requirements in the passed in [] of MedDispatchRec (see above).

The important thing is that dispatches are joined by AND. So, only if all records can be matched with 1 drone, this drone can be used and returned. In other cases (no drone to match the requirements) an empty vector is returned.

Consider date and time as well as drones only have limited availability on certain days of the week.

Assume the date to be a LocalDate and time a LocalTime.

You can have 0..n drones as a result

`queryAvailableDrones (POST)`

Passed in is a [] of MedDispatchRec which contains the necessary attributes

- **`calcDeliveryPath (POST)`**

Given a [] of valid MedDispatchRec find the best logical order how the available drones can deliver the dispatches without falling from the sky (running out of moves).

You must start at a Drone Service Point (which has the relevant drone(s) available) and fly as many dispatches as possible in one string, before returning to the same service point (and before running out of moves).

This call shall return a data structure like:

```
{
	“totalCost”: 1234.44,
	“totalMoves”: 12111,
	“dronePaths”: [
		{
			“droneId”: 4,
                        “deliveries": [
                               {
                                      “deliveryId”: 123,
                                      “flightPath”: [
                                             {
                                                    "lng": -3.1863580788986368,
                                                    "lat": 55.94468066708487
                                      },
                                             {"lng": -3.186359,"lat": 55.94468066708487},
                                             ...
                                      ]
                               }
                        ]
                 }
          ]
   }
```

“dronePaths” is a [] of drone paths structures. Each of these has a droneId (of the drone doing the job -> must be available) and a [] of deliveries. Each delivery has the id of the MedDispatchRecord it delivers and a flightPath as a [] of LngLat entries which are your moves. You start with the service point, do your first delivery, continue and terminate at the service point. Each flightPath is from the logical start (previous delivery or service point) until the next delivery (or service point).

A delivery is indicated by having 2 identical records in the flightPath (so no move, just a hover). Each delivery must be indicated

So, if your sequence looks like:

1 drone: Service point -> delivery 1 -> delivery 2 -> Service Point of origin

Your data would be like:

1 dronePath, 2 deliveries with flightPath from Service Point -> delivery 1 and the second delivery from delivery 1 to delivery 2 to the service point.

Where your hover (deliver) in the first would be at the end, in the second it would be in the middle (as coming from delivery 1, delivering #2 and going back)

All the normal rules apply like (but not limited to):
only moving along the pre-defined degrees, no no-fly-areas, no cutting of corners, keep your step width, do not exceed the drone.

It might be necessary to use several drones to perform the delivery – and perhaps

from several drone service points.

5) **calcDeliveryPathAsGeoJson（POST）**

Similar to the previous request, just with the guarantee that all deliveries can be made with a single drone in a single delivery sequence.

The result is supposed to be a single. valid GeoJSON structure (https://geojson.io) - LineString might be a good start – which can be pasted in the online viewer and produces a visual result. So, no other data – just the GeoJSON structure.

Some helpful considerations:

- Point 1 is straightforward and just the fix from CW1
- Point 2 should be done first
- Point 3 can be complex and should only be attempted after point 4 is resolved as that one is needed a lot for #5 and #6
- Point 5 can be quite complex (there will be a simple and a complex scenario)
- Point 6 is an application of Point 5 and a re-mapping of the result
- You always return 200 (except the id call)
- All data is valid, yet might not be resolvable

The following should be considered when implementing the REST-service:

- Test your endpoints using a tool like Postman or curl. Plain Chrome / Firefox, etc. will do equally for the GET operations
- The filename for the docker image file must be exactly as defined as well as the location of it in the ZIP-file. Should you be in doubt, use copy & paste to get the name right

- **You can use any library and implementation detail you want if the final image is runnable in docker and uses Java**

Should you need help:

- See the literature links in the "Library" section in Learn. You should find most information there
- If you cannot find an answer to your question, please post it on Piazza, though try finding it yourself first, please (as we have only limited capacity)

Disclaimer: We will not be able to answer any question on piazza less than 3 days before the assignment deadline

So, please make sure you start the assignment in good time.

CW3

In CW3 you can implement whatever you want – if it is in the range of ILP. Use AI, any language you want and implement anything you consider useful.

You can extend the CW2 implementation or create something entirely new. Just submit everything you did and – please... - some instructions on how to build it.

The anticipated time for this task will be 20 – 24h, so, quite limited. Please, try not to get overboard by investing too much time – always consider the points and how much improvement those extra 10h will make.... We anticipate that solutions will be more a "proof of concept" (PoC) than complete implementations with all bells and whistles. Thus, do not try to create the richest, most complete implementations, but look for innovative ideas and use cases, things which are complex and can easily be adjusted, etc.

It could be (just examples – use your imagination):

- Flightpath visualizer integration
- MCP service to integrate ILP data into an LLM
- Order form(s) to provide medical dispatches and plan routes
- Drone availability and query system in GraphQL
- Drone / depot maintenance software
- Persistent data storage for ILP data
- RabbitMQ integration for drone real-time updates
- ...

Together with the submission of your work you have to submit a PDF document cw3_explanation.pdf which explains (based on the rubrics later in this document) why you did what for whom and which problem to you solve. In addition, you explain how you did it and what was your motivation for the tools used...

The document must not exceed 1,000 words maximum. For each 100 words excess, 10% will be deducted and if more than 1,500 words, 0 will be awarded as a mark.

CW4

In CW4 you will create a video showcasing the solution you built in CW3. The marking will be based on the rubrics later in this document and in principle you express in your own words what you built, why, for whom, how and with which tools.

The main part shall be the presentation of what you did as a demo.

Your video is 5-7 minutes long with 10% off for each minute over and 0 as mark for videos exceeding 10 minutes.

Some important instructions for the video:

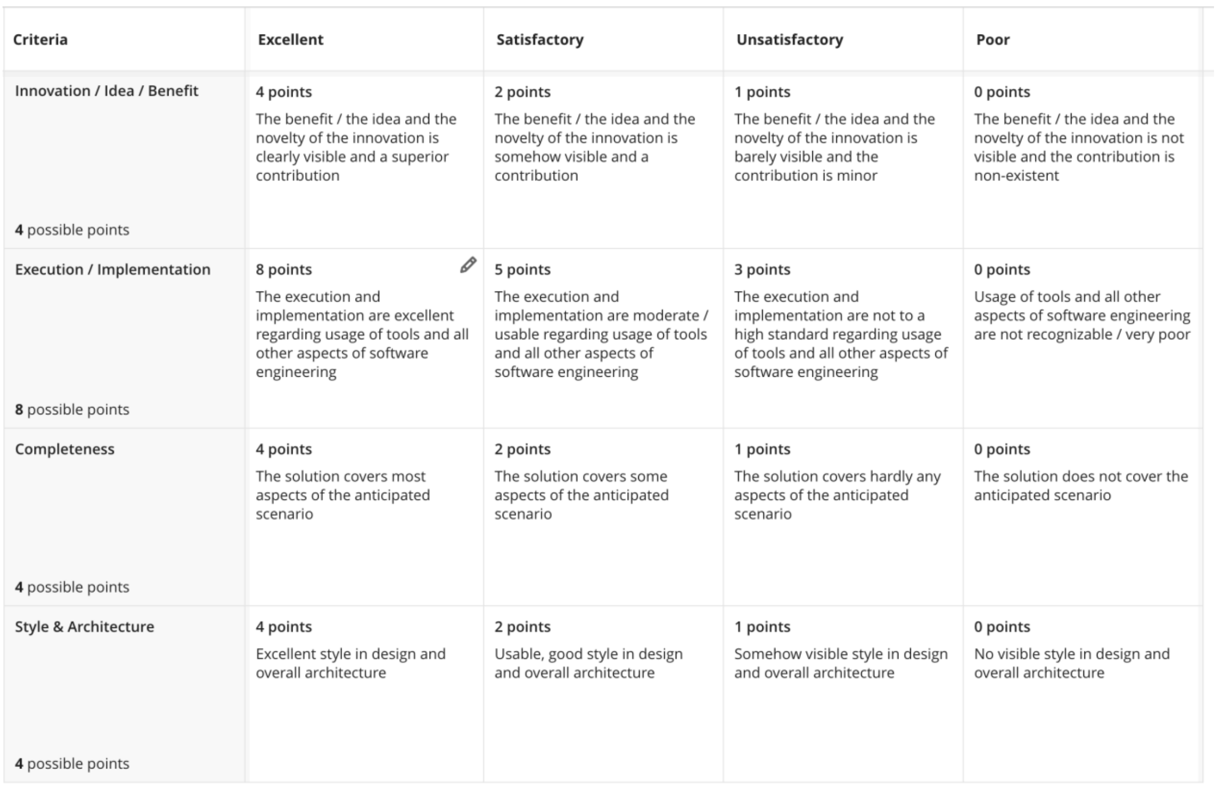
- **MP4-format (MOV or AVI only in exceptional cases)**
- In the very beginning your face must be visible and next to it your student id. Say your student id for us
- Then use your left hand, spread your fingers and move before your face left to right and back
- Same with right hand
- It is important that your face is visible through your fingers

Rubrics

CW2 (33 points)

Endpoint	***Comment***	***Points***
Re-running existing tests	Incorrect call	2
Group 2 (static queries)	2a	2
	2b	2
Group 3 (dynamic queries)	3a	3
	3b	3
Drone availability query	with availability	3 (1 +2)
	with no availability	4 (2 + 2)
calcDeliveryPath	Simple case	4
	Complex case	7
calcDeliveryPathAsGeoJSON	Correct call	3
Total		33

CW3 (20 points)



Criteria	**Excellent**	**Satisfactory**	**Unsatisfactory**	**Poor**
Innovation / Idea / Benefit **4** possible points	**4 points** The benefit / the idea and the novelty of the innovation is clearly visible and a superior contribution	**2 points** The benefit / the idea and the novelty of the innovation is somehow visible and a contribution	**1 points** The benefit / the idea and the novelty of the innovation is barely visible and the contribution is minor	**0 points** The benefit / the idea and the novelty of the innovation is not visible and the contribution is non-existent
Execution / Implementation **8** possible points	**8 points** The execution and implementation are excellent regarding usage of tools and all other aspects of software engineering	**5 points** The execution and implementation are moderate / usable regarding usage of tools and all other aspects of software engineering	**3 points** The execution and implementation are not to a high standard regarding usage of tools and all other aspects of software engineering	**0 points** Usage of tools and all other aspects of software engineering are not recognizable / very poor
Completeness **4** possible points	**4 points** The solution covers most aspects of the anticipated scenario	**2 points** The solution covers some aspects of the anticipated scenario	**1 points** The solution covers hardly any aspects of the anticipated scenario	**0 points** The solution does not cover the anticipated scenario
Style & Architecture **4** possible points	**4 points** Excellent style in design and overall architecture	**2 points** Usable, good style in design and overall architecture	**1 points** Somehow visible style in design and overall architecture	**0 points** No visible style in design and overall architecture

CW4 (12 points)

Criteria	Excellent	Satisfactory	Unsatisfactory
Background elaboration / target audience **2** possible points	**2 points** Excellent description of the background and the target audience for the solution (for whom and because of what...)	**1 points** Moderate description of the background and the target audience for the solution (for whom and because of what...)	**0 points** Non-satisfactory description of the background and the target audience for the solution (for whom and because of what...)
Problem statement **2** possible points	**2 points** Excellent description of the problem statement (why...)	**1 points** Moderate description of the problem statement (why...)	**0 points** Non-satisfactory description of the problem statement (why...
Explanation of the student's choice **4** possible points	**4 points** Excellent description how the solution was built. which tools used and why	**2 points** Moderate description how the solution was built. which tools used and why	**0 points** Non-satisfactory description how the solution was built. which tools used and why
Presentation **4** possible points	**4 points** Excellent general presentation with good choice for content, colors, background, style and the tool presentation as such	**2 points** Moderate general presentation with good choice for content, colors, background, style and the tool presentation as such	**0 points** Non-satisfactory general presentation with good choice for content, colors, background, style and the tool presentation as such