

Technical Documentation

General flow of the program:

The main method is only responsible for starting the application through the launch method. In the launch method, the log file is first initialized. Next, the primaryStage is configured by setting the application title and parameters.

Following this, the FXMLLoader are initialized for the Loading Scene and Main Scene. The Controller are created and their initialize methods are called and the scenes appear in the app one after another. The MainController is then stored for future operations.

The initThread is started while the Loading Scene is still active. This thread loads important data, such as drone IDs and dynamics, in the background to ensure that the key functionalities are accessible without additional loading times. Within the thread, a Dashboard object is created and passed to the MainController. Drone IDs and types are loaded, labels are reset, and upon completion, the application transitions to the Main Scene.

Finally, the initThread is started to complete the loading process.

When the program is started, a Dashboard object and the GUI main controller are created. When a Dashboard object is created, the DroneCollection Object in Dashboard automatically initializes the map of drones, drone types and a set amount of dynamics of each drone.

Via the Dashboard object the GUI can access all relevant information about the drones, such as a hashmap of the drones, a hashmap of the dronetypes as well as the currently selected drone, the currently selected drone dynamics and an offset, which indicates where in the list of dynamics the current dynamic is. Additionally, the Dashboard class provides methods to change the currently selected drone, dynamics and offset.

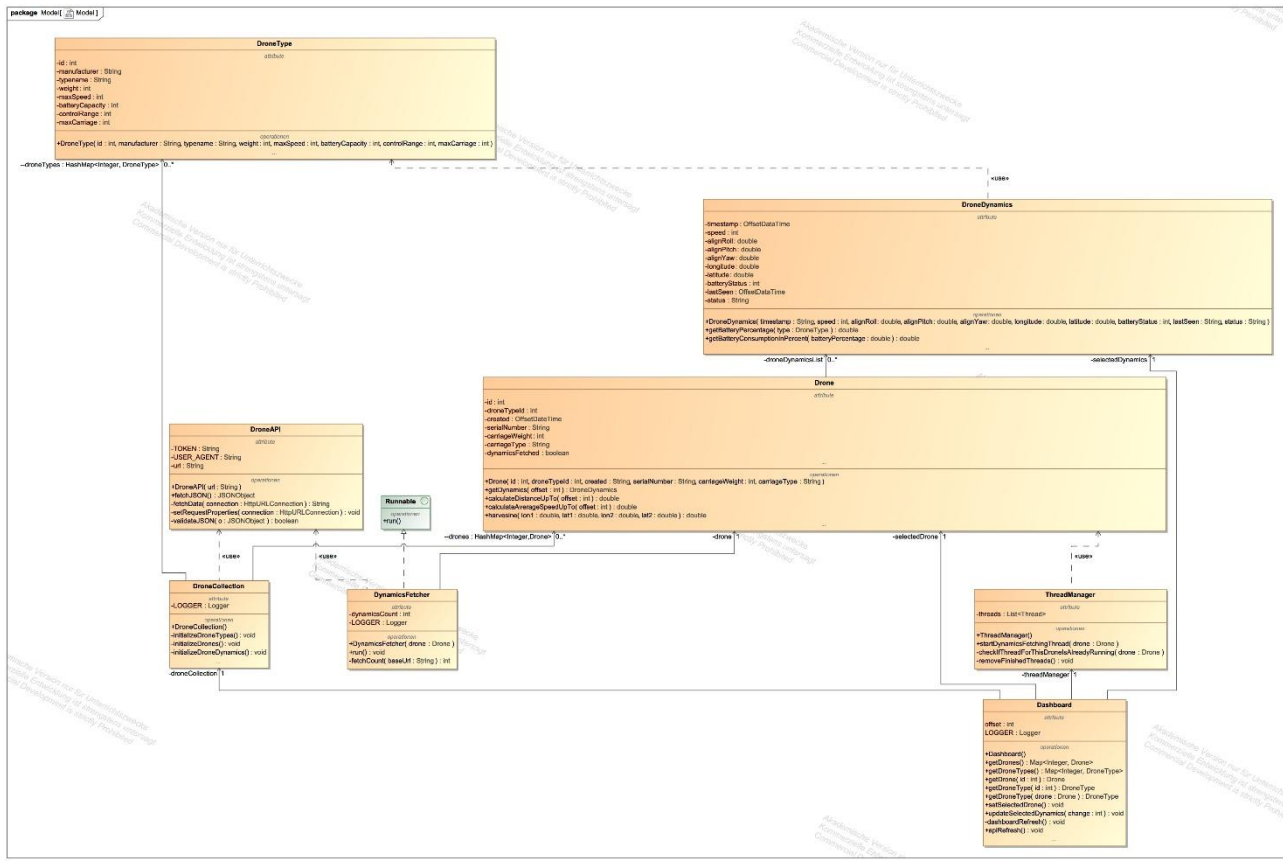
The user can browse the information about the drones and drone types as soon as the GUI has loaded. When the user wants to view the dynamics of a selected drone and the drone has not had its dynamics fetched before, a thread is started that fetches the dynamics.

When the fetching of the drones and drone types fails for any reason, the user can use a refresh button to retry. The click of the button creates a new DroneCollection object.

For more details about the class please check the section "Class details".

Class details:

Backend:



Dashboard

The Dashboard class is the intersection between GUI and the back end. The goal of this class is to have as little logic as possible in the GUI classes.

Attributes:

- **selectedDrone**: The Drone the user has currently selected.
- **selectedDynamics**: The DroneDynamics the user has currently selected.
- **offset**: The number of the selected dynamic of the drone. (offset = 0 => The selected dynamic is the first dynamic of the selected drone.)
- **droneCollection**: The object that contains a list of all available drones and drone types.
- **threadManager**: The object responsible for starting and managing the threads for fetching the drone dynamics.

Methods:

The class has methods for returning unmodifiable copies of the drone and dronetype hashmaps as well as for returning drones and dronetypes by their ID.

- **setSelectedDrone(Drone)**: sets the **selectedDrone** attribute to the passed argument and tells the **threadManager** to start fetching the dynamics of the drone.
- **updateSelectedDynamics(int)**: sets the **offset** attribute to **offset + passed integer**. Stops the offset from going below 0 or above the amount of available

Dronedynamics. Sets the selectedDynamics attribute to the dynamic of the selected drone with the number of the new offset.

- ApiRefresh(): re-initializes the DroneCollection attribute to fetch new drones and dronetypes.
- UiRefresh(): resets the attributes offset, selectedDynamics and selectedDrone.

DroneCollection

When a DroneCollection object is created, it uses the DroneAPI class to fetch all available drones and drone types into a hashmap. To make the user experience smoother, it fetches a set amount of dynamics for every drone.

Attributes:

- droneTypes: a hashmap of droneType objects. The keys are the IDs of the drone types.
- drones: a hashmap of drone objects. The keys are the IDs of the drones.

Methods:

The class has methods for returning unmodifiable copies of the drone and dronetype hashmaps as well as methods for initializing the hashmaps and the dynamics of the drones. The amount of dynamics that is supposed to be fetched during the creation of the DroneCollection object can be changed in the initializeDroneDynamics method. The rest of the program works dynamically with this value.

ThreadManager

The ThreadManager manages a list of currently running DynamicsFetcher threads. It only allows the start of a new thread if a thread for the same drone is not currently running and the dynamics for the drone have not been fetched before.

DynamicsFetcher

A runnable that fetches all remaining dynamics of a given drone using the DroneAPI.

DroneAPI

The class responsible for opening the connection to the given URL and fetching the data and returning it in JSON format.

Drone

The class represents a drone. it holds a list of all dynamics of the drone that have been fetched so far and a boolean value that indicates, whether all dynamics have been fetched or not.

Methods of interest:

- calculateDistanceUpTo(int): iteratively calculates and returns the distance the drone has traveled up until the dynamic of the given number
- calculateAverageSpeedUpTo(int): iteratively calculates and returns the average speed the drone had up until the dynamic of the given number.

DroneType

The class represents a drone type.

DroneDynamics

The class represents the dynamics of a drone at one point in time.

Methods of interest:

- `getBatteryPercentage(DroneType)`: calculates and returns the current battery charge in percent.
- `GetBatteryConsumption`: calculates and returns how much battery charge was consumed so far in percent.

Errorhandling

- `ConnectionFailedException`: thrown by the DroneAPI when the fetching fails for any reason.
- `IllegalJSONFormatException`: thrown by the DroneAPI when the fetching is successful, but the JSON object does not contain the required fields.
- `CustomFormatter`: custom formatting for the file handler of the logger
- `Logging`: provides methods for initializing the logger. The logger uses a file handler.

Util

- The parser classes contain methods to parse JSON objects into Drone, DroneType or DroneDynamic objects.
- Util contains other utility methods.

Milestone 1 - Group 22

1 Introduction

This is a progress report for our Java project interacting with a drone API. We're building an application to show data about drones and their updatable dynamics using a graphical interface.

2 Group Members

Name	Main Responsibilities
Tobias Ilcken	API Interaction
Younes Wimmer	API Interaction
Parnia Esfahani	API Interaction
Maxim Wenkemann	GUI Development
Torben Fenchel	GUI Development

3 Project Documentation

- **What has been implemented?**
 - API connection
 - Parsing JSON data into Java objects (DroneType, Drone, DroneDynamics).
 - Basic Drone and DroneType data retrieval.
 - Basic DroneDynamics data retrieval.
 - Referencing Drone & DroneTypes
 - Time difference calculation between dynamics timestamps.
 - Updating individual drone dynamics. (not sure still)
- **Individual parts:**
 - Dashboard: Gets data from the API and parses JSON.
 - Drone: Represents a drone and its data.
 - DroneType: Represents a type of drone.
 - DroneDynamics: Represents a drone's status at a time.
- **Code Snippet:**

```
public int calculateTimeStamps() {  
    String urlWithoutOffset =  
    "http://dronesim.facets-labs.com/api/dronedynamics/?limit=1";
```

```

String urlDrones = "http://dronesim.facets-labs.com/api/drones/";
int result = 0;
String[] array = new String[2];
int droneSize = 0;

JSONObject wholeFileDrones = getJSON(urlDrones);
if (wholeFileDrones != null && wholeFileDrones.has("count")) {
    droneSize = wholeFileDrones.getInt("count");
}

JSONObject wholeFileDynamics = getJSON(urlWithoutOffset);
JSONArray jsonFileDynamics =
wholeFileDynamics.getJSONArray("results");
JSONObject o = jsonFileDynamics.getJSONObject(0);
array[0] = o.getString("timestamp");

String url = urlWithoutOffset + "&offset=" + droneSize;
JSONObject wholeFileDynamics2 = getJSON(url);
JSONArray jsonFileDynamics2 =
wholeFileDynamics2.getJSONArray("results");
JSONObject o2 = jsonFileDynamics2.getJSONObject(0);
array[1] = o2.getString("timestamp");

if (array[0] != null && array[1] != null) {
    OffsetDateTime timestamp1 = OffsetDateTime.parse(array[0]);
    OffsetDateTime timestamp2 = OffsetDateTime.parse(array[1]);
    Duration duration = Duration.between(timestamp1, timestamp2);
    result = (int) Math.abs(duration.getSeconds());
}
return result;
}

```

4 Milestones

4.1 Milestone 1

- **Achievements:** API connection, data retrieval (dronetypes, drones), JSON parsing, basic DroneDynamics retrieval, time difference calculation, getting first and last individual drone dynamic.
- **Who did what:** Tobias, Younes, Parnia (API), Maxim, Torben (GUI planning).
- **Next steps:** Implement efficient data storage, full DroneDynamics pagination, start GUI (Drone Catalog, Dashboard), clean up (messy code)
- **Problems:** drone dynamics data retrieval not efficient.

4.2 Milestone 2 (Plans)

- **Planned achievements:** Finish GUI (Drone Catalog, Dashboard), calculations, implement extra features, error handling, file access, threads, clean code.

Skit
Group 22 ☹️

1.



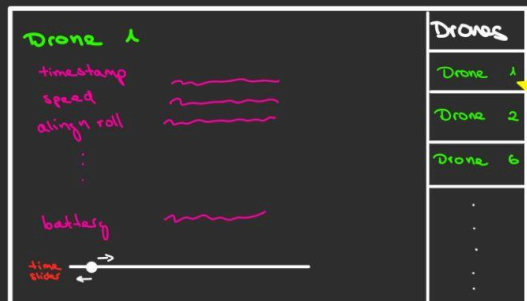
2.



Drone 1	↑
Drone 2	↑
Drone 3	↑
Drone 4	↑
Drone 5	↑
Drone 6	↑
.	
.	
Drone n	

↑ scrolling ↓

3.



Milestone - Group 22

1 Introduction

This is a progress report for our Java project interacting with a drone API. We're building an application to show data about drones and their updatable dynamics using a graphical interface.

2 Group Members

Name	Main Responsibilities
Tobias Ilcken	Backend
Parnia Esfahani	Backend
Younes Wimmer	Backend & Frontend
Maxim Wenkemann	Frontend
Torben Fenchel	Frontend

3 Project Documentation

- **What has been implemented?**

- API connection
- Drone Catalog
- Drone Dashboard
- Flight Dynamics
- Data Refresh

- **Code Snippet:**

```
@FXML
```

```
private void onNextDynamicClicked1() {  
    proceedNextDynamic(1);  
}
```

```
private void proceedNextDynamic(int steps) {  
    Drone selectedDrone = dashboard.getSelectedDrone();
```



```

if (selectedDrone == null) return;

dashboard.updateSelectedDynamics(steps);

setDroneDynamicsLabels(selectedDrone);
}

```

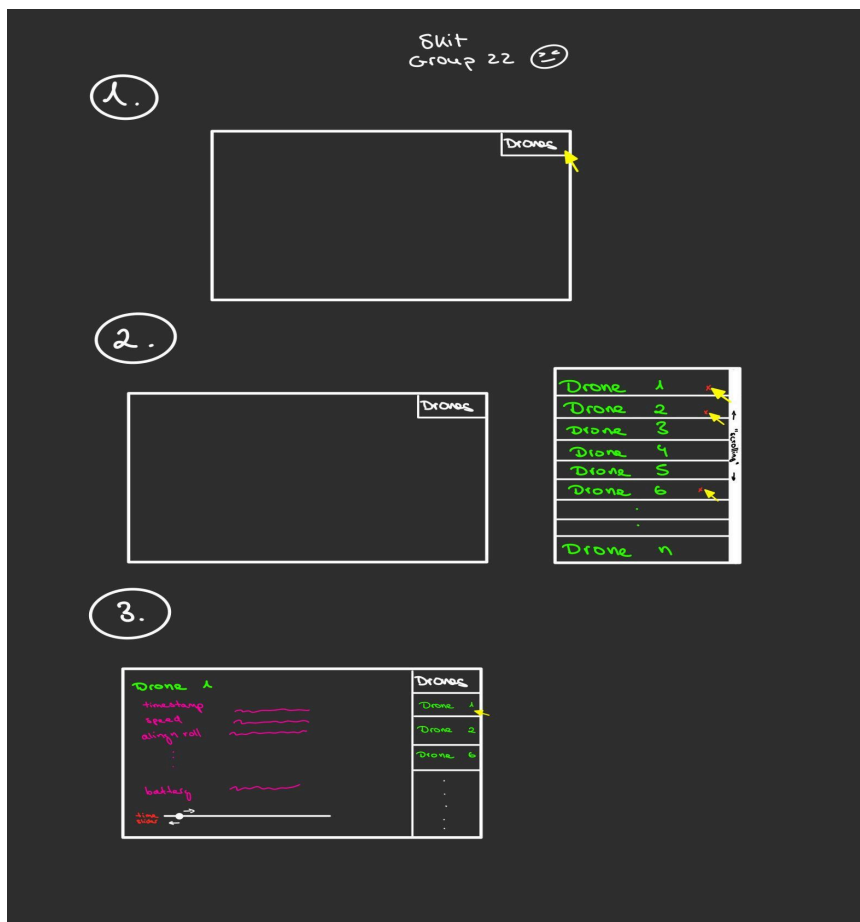
4 Milestones

4.1 Milestone 1

- **Achievements:** API connection, data retrieval (dronetypes, drones), JSON parsing, basic DroneDynamics retrieval, time difference calculation, getting first and last individual drone dynamic.
- **Who did what:** Tobias, Younes, Parnia (API), Maxim, Torben (GUI planning).
- **Next steps:** Implement efficient data storage, full DroneDynamics pagination, start GUI (Drone Catalog, Dashboard), clean up (messy code)
- **Problems:** drone dynamics data retrieval not efficient.

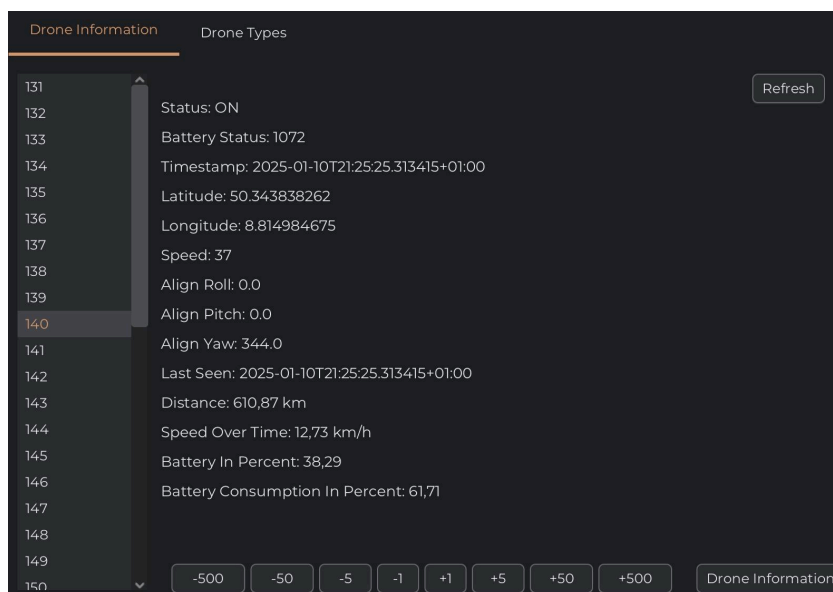
4.2 Milestone 2 (Plans)

- **Planned achievements:** Finish GUI (Drone Catalog, Dashboard), calculations, implement extra features, error handling, file access, threads, clean code.



4.2 Milestone 2

- **Achievements:**
 - Threads Implementation
 - Auto API Refresh Implementation
 - GUI Implementation
 - GUI Navigation Enhancements
 - All required calculations implemented
- **Who did what:**
 - Younes (Backend,Frontend),
 - Tobias(Backend),
 - Parnia(Backend)
 - Maxim (Frontend)
 - Torben(Frontend)
- **Next steps:**
 - Error Handling
 - Refining Thread
 - Logging
 - Stream and Files
 - UI tweaks
 - Clean Code
 - Docs
- **Problems:**
 - UI Whitescreen
 - lag between picked Drones
 - Thread loading placement
 - messy code
- **Planned achievements:** Finish Project
- **How it looks:**



User Manual

for Drone Application

Compiling the Program

To compile our program, you will need Java SDK version 23 and Java version 22. While it may work with other versions, these are the versions that worked for us. Additionally, we added the following VM options to compile the program:

```
--module-path
```

```
"C:\Program Files\Java\javafx-sdk-23.0.1\lib"
```

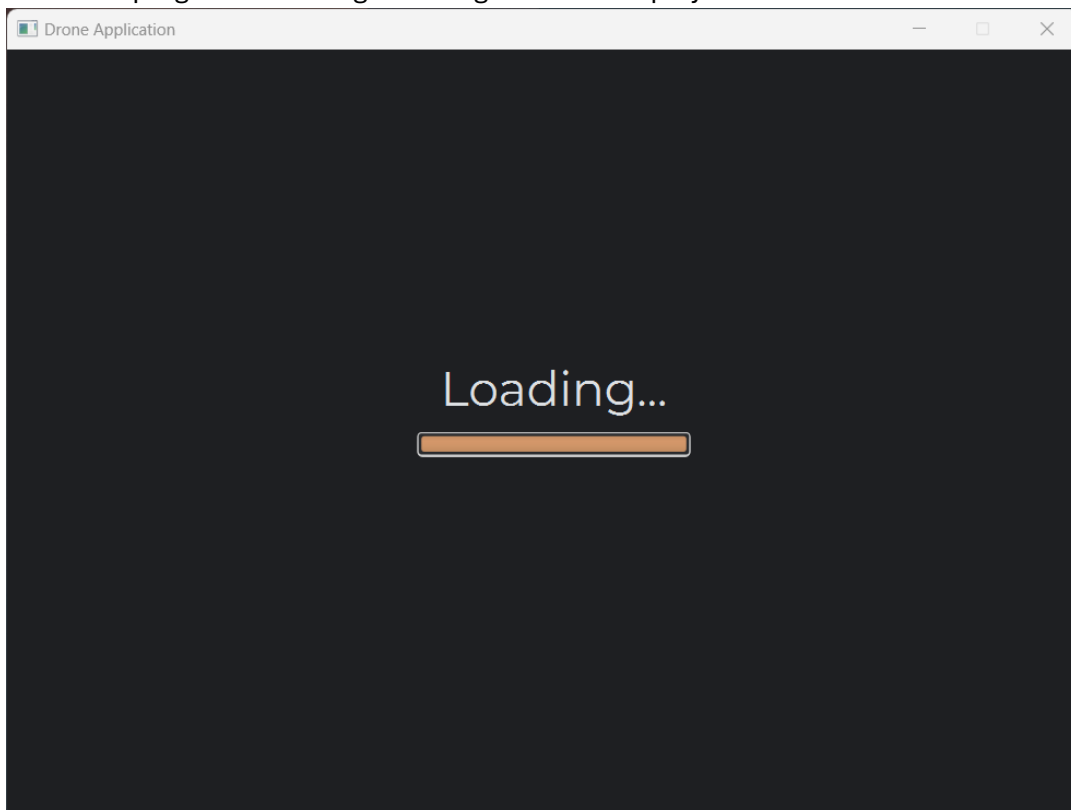
```
--add-modules
```

```
javafx.controls, javafx.fxml
```

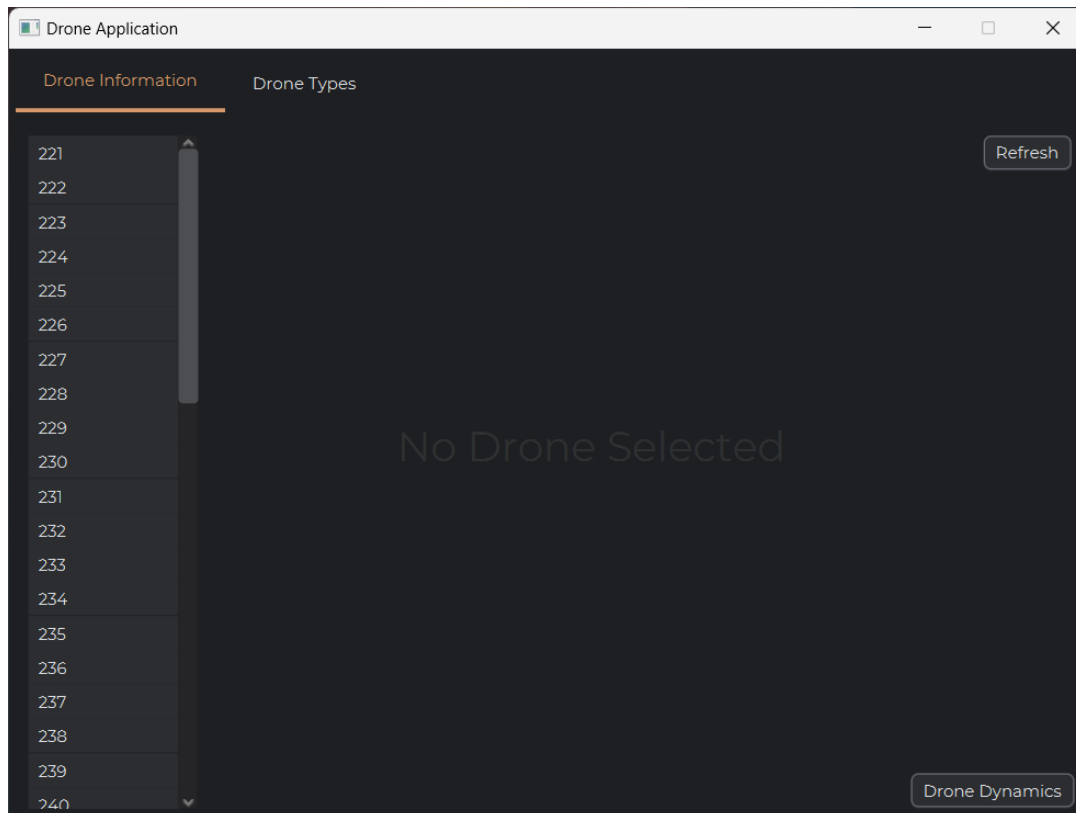
Please note that you will need to specify your own path to the lib folder in the module path.

Program

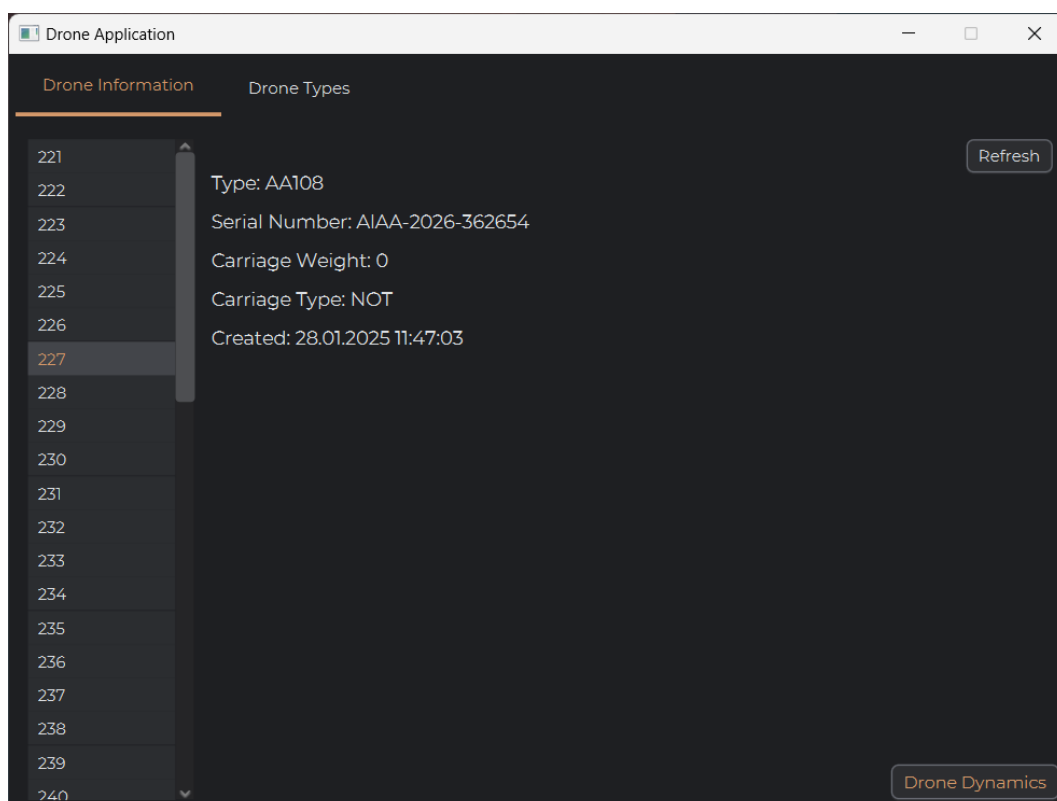
While the program is starting a loading screen is displayed.



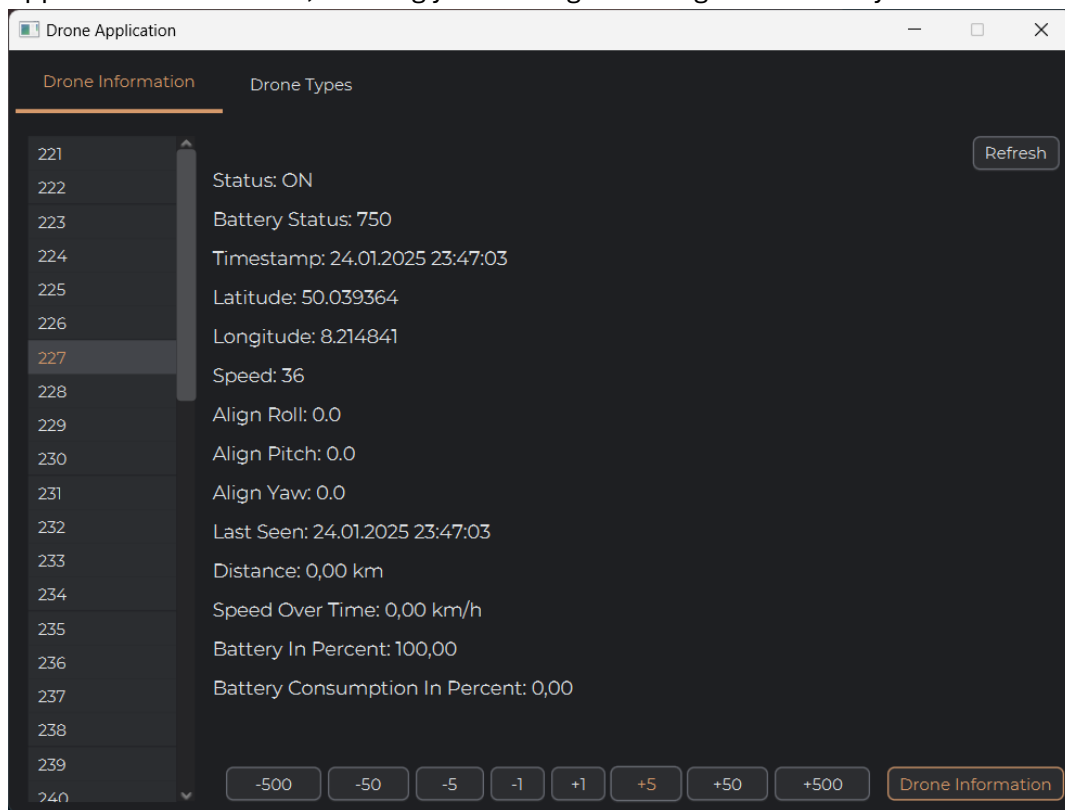
Once the program is loaded you can see a list of all DroneIDs on the left. While no DroneID is selected a message is displayed stating that no drone has been selected yet.



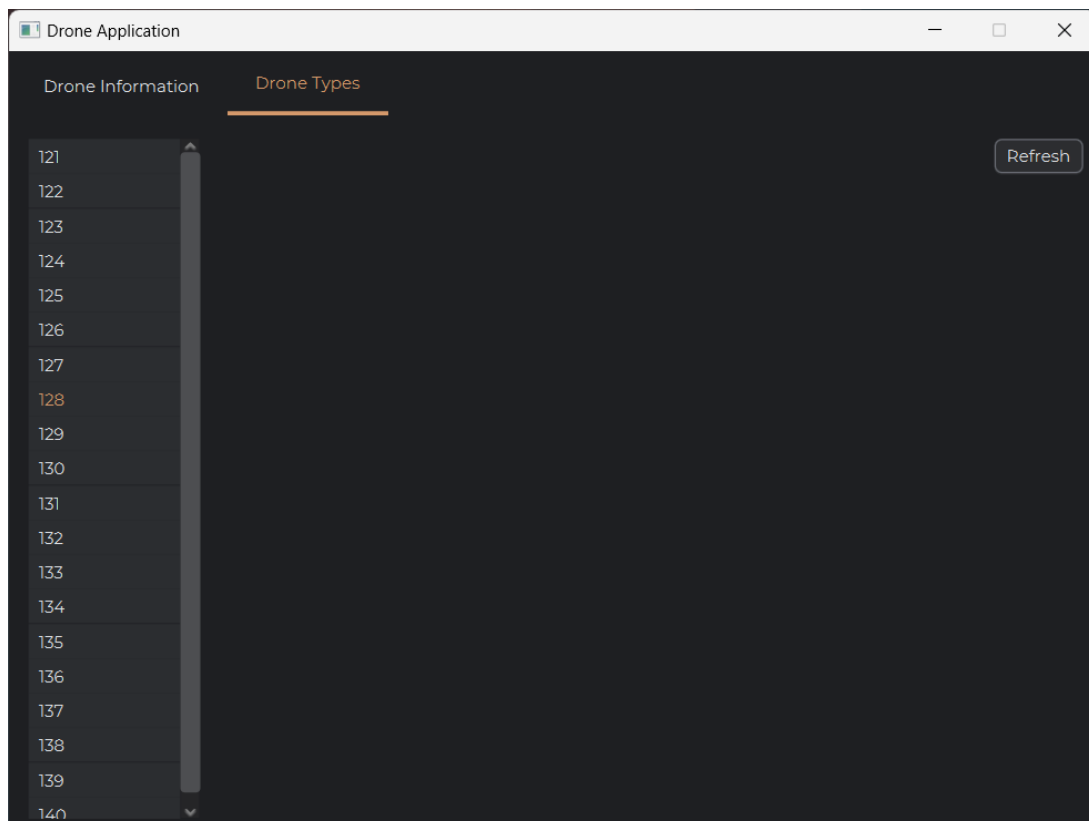
Once a drone is selected, its drone information is displayed on the right. To access the corresponding drone dynamics, click the toggle button in the bottom right corner.



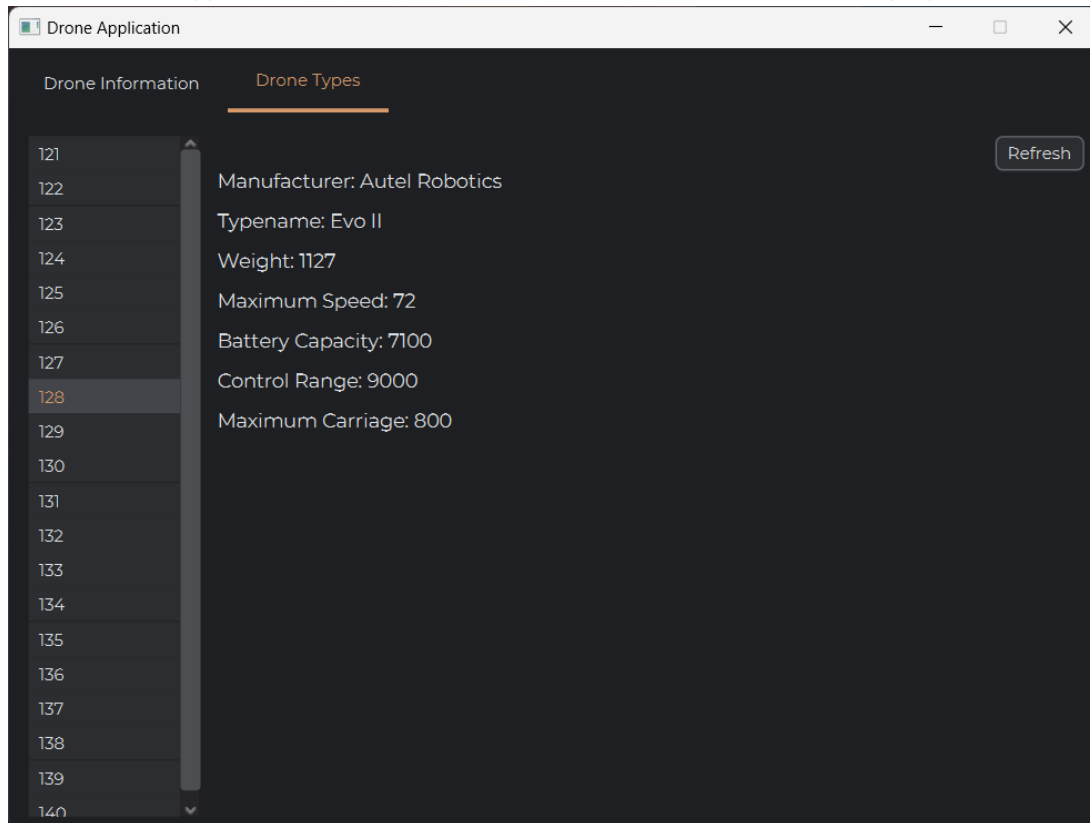
Now, the drone dynamics for the selected drone are displayed. Additionally, some buttons have appeared at the bottom, allowing you to navigate through the drone dynamics.



Using the tabs in the top left corner, you can switch between the Drone Information and the drone types. In the Drone Types tab, information is only displayed once a drone type has been selected.

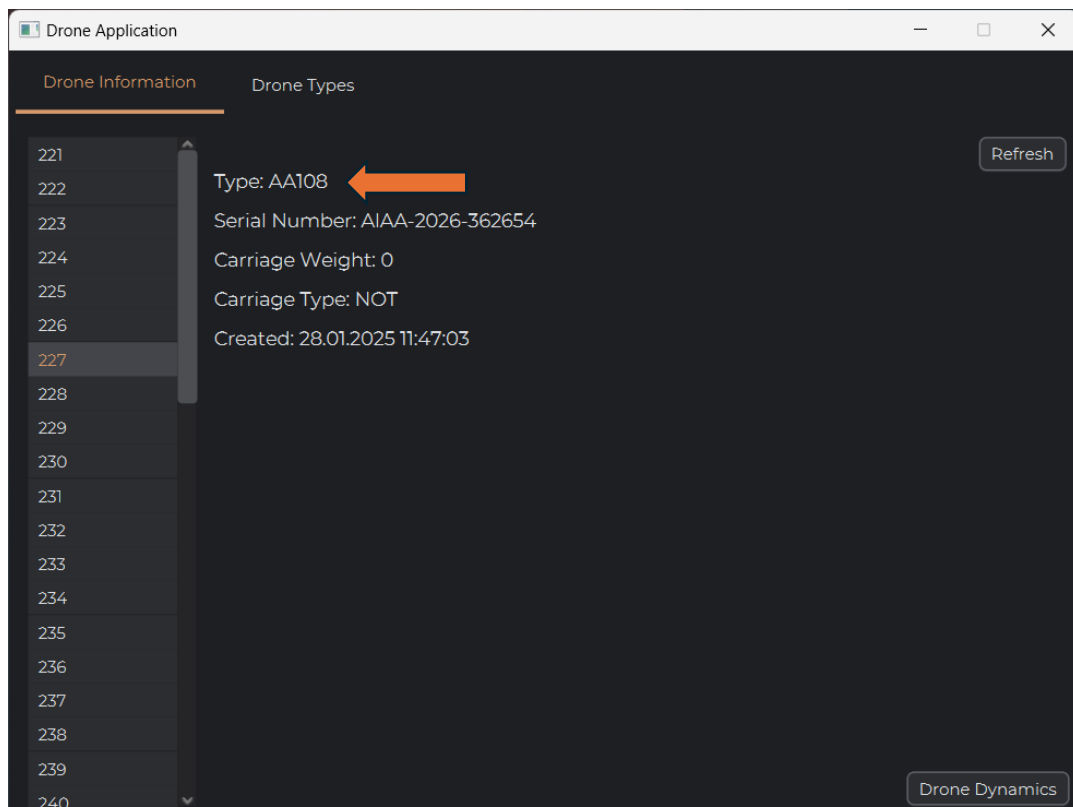


Once a drone type has been selected, its associated information is displayed.

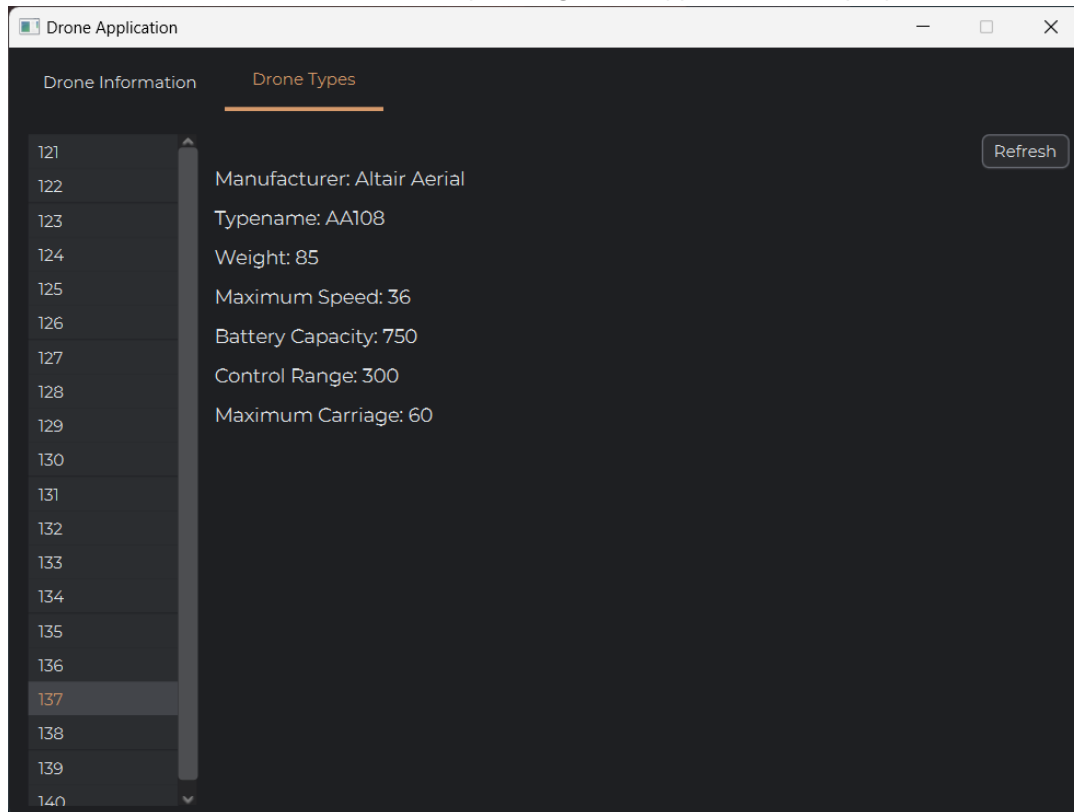


Another way to view the information for a drone type is:

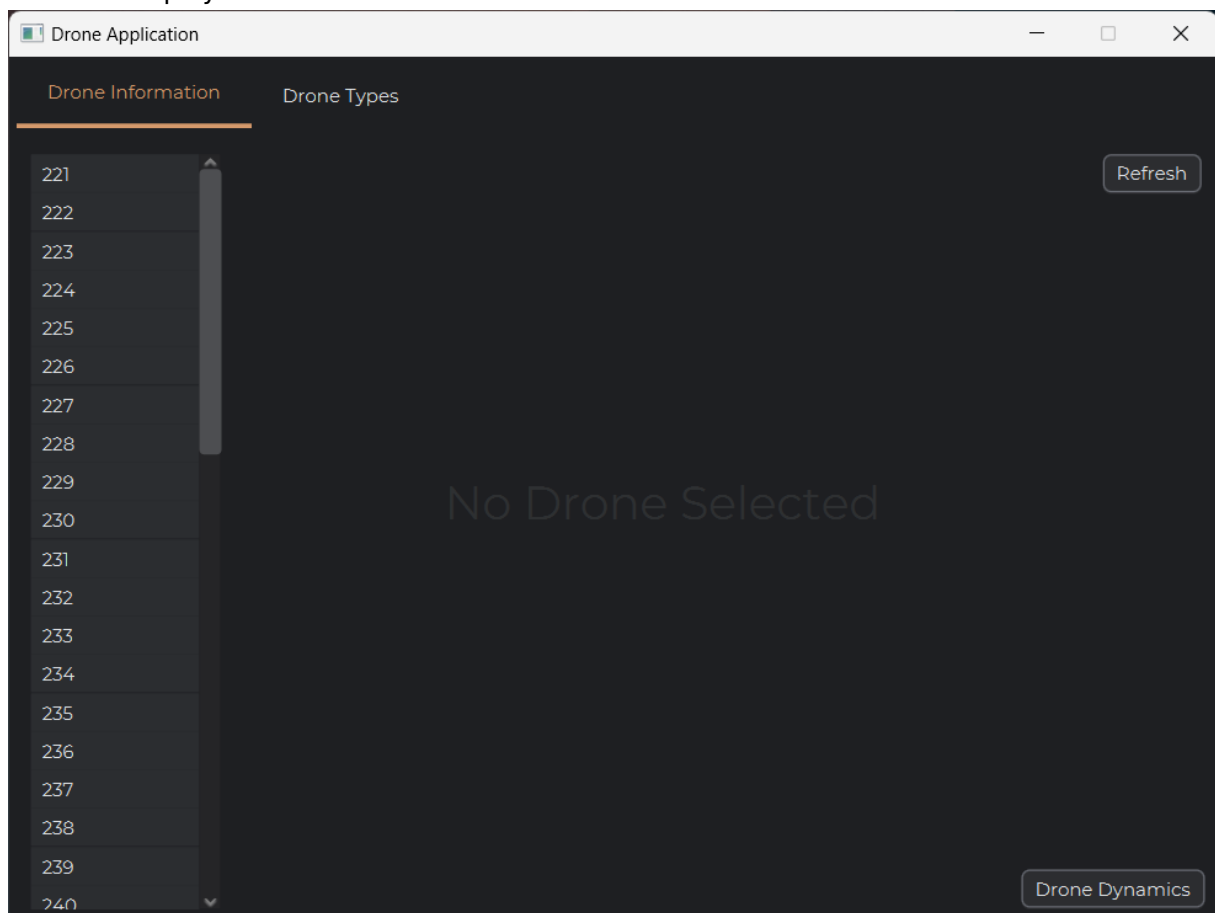
If a drone is selected in the drone information tab, you can click on the displayed type, which will take you directly to the drone type.



Further information about the corresponding drone type is then displayed.



At any time, you can click the refresh button in the top right corner. This will reload all drones and reset the displayed information.







Name	Tasks
Younes Wimmer	Runnable Program (Backend), Connecting Backend & GUI, Milestones, Object-Orientation, Collections, Streams and Files, Threads, Clean Code, Documentation
Tobias Ilcken	Runnable Program (Backend), Utilization, Object-Orientation, Collections, Error Handling, Logging, Streams and Files, Threads, Clean Code, Documentation
Parnia Esfahani	Runnable Program (Backend), Milestones, Documentation
Maxim Wenkmann	Runnable Program (GUI), Connecting Backend & GUI, Initializing Main, Documentation
Torben Fechner	Runnable Program (GUI), Connecting Backend & GUI, GitHub, Documentation

Feature	Class	Information
Abstract method	BaseParser	Line 22
interface	Parser	Used by BaseParser
Collections	DroneCollection, ThreadManager	DroneCollection: Line 23, 24 ThreadManager: Line 14
Error Handling	ConnectionFailedException, IllegalJSONFormatException	
Streams and Files	Logging	Line 28
Threads	DynamicsFetcher	Line 36

Declaration of Authorship

Example for a declaration of authorship ("Ehrenwörtliche Erklärung"):

<p>I hereby declare that the submitted project is my own unaided work or the unaided work of our team. All direct or indirect sources used are acknowledged as references.</p> <p>I am aware that the project in digital form can be examined for the use of unauthorized aid and in order to determine whether the project as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future projects submitted. Further rights of reproduction and usage, however, are not granted here.</p> <p>This work was not previously presented to another examination board and has not been published.</p>	
<u>Ganes Wimmer</u> First and last name	<u>FFM, 04.02.25</u>  City, date and signature
<u>Paria Estahani</u>	<u>FFM, 04.02.25</u> 
<u>Maxim Wenkemann</u>	<u>FFM, 04.02.25</u> 
<u>Tobias Kden</u>	<u>FFM, 04.02.25</u> 
<u>Torben Fechner</u>	<u>FFM, 04.02.25</u> 