This is an attempt at a high level description of the simcaPy server.

# Startup

The `server` module is the main entry point for the server. It needs to be called with a path to a configuration json file. It then sets up three routes (url endpoints for the frontend to access the backend) for the server. Afterwards, the server is started.

For demonstation purposes another thread is started, which will periodically send data to the server, to allow using the webapp, even if there is no real process generating data.

## The routes

One them simply resets the internal state and the other one just returns data about the configuration.

The last and most important one is used to send or request data to/from the server. It is parametrized and thus allows to insert data for a specific production step at a specific equipment, or to retrieve such data. If getting the data, the returned data will be in a forat suitable for a plotting library in the frontend. If setting the data, a complete row of measurements (of all sensors of the equipment) needs to be provided.

## The config file

The config file contains a list of all production steps at the desired plant. Each step is made up of a name and a list of equipments used for the step. Each equipment has a name and various properties, including a list of all sensors.

# The plant model

Just like in the configuration file, a plant is only defined by the production steps it carries out and each step is again defined by the equipments it uses.

Both the Plant and Step classes don't hold much functionality aside from being a container and allowing the turn the internal data into dictionaries.

The equipment however is different.

## The Equipment

An equipment is defined by the sensors it has, as well as more general information about which columns in a dataframe represent which data. It also contains a PLSModel based on some known-good data (the data is more or less hardcoded).

It also has more functionality:

* It allows adding a measurement (this is done though one of the server routes). A measurement is a single row containing values for the sensors in this equipment.

Upon adding a measurement, its data will just be recorded in the object.

* After adding a measurement, the new data can be processed (and will be).

Processed means that a lot of PLS related variables (t2, dmodx) are calculated and added to a "cache"/list of the previous versions of these variables. This allows plotting them later.

* Updating the status. This just sets an internal variable, based on the most recent t2 score. as well as how much the most recent measurement deviates from the mean +- 3 sigma.
* Starting a new batch. This just resets most internal paramters/lists.
* A few functions to create/arrange the data in a way that the frontend can easily plot it.

## The PLS analysis

This is mostly independant of the other modules. To create a PLSModel, you need some known-good data and define time/batch/x columns, as well as the number of components.

In this implementation, it does not matter whether the data is evenly spaced or not, since it will be transformed into an evenly spaced version through interpolation. Also `nan` values will also be filled by interpolation.

With the known-good data, a model is created, which allows us to project new data and calculate all sorts of parameters.

For the t2 contributions, it is possible to choose one of multiple different ways of computing them. Additionally, the dModX of some data can also be computed.

### Additional remarks:

This module also contains some utility functions to join datasets on float values. This is only necessary because data might not be evenly spaced and needs to be joined to the evenly spaced time grid.

# During Operation

The basic operation is like this:

* The server listens on the defined routes.
  + If someone sends data to a specific equipment of a specific step, the data will be added to that equipment and all PLS metrics will be updated.
  + If someone requests data from a specific equipment, the (already existing) pls metrics as well as the measurement data will be packed into some datastructure and be sent back.
* Other than that, nothing happens in the backend