Contents

1 Classes

1.1 Pilot

`Pilot` is a class that is unique to our application. The current structure of the pilot class is as follows:

- **pilot_id**: Used to refer to each unique pilot.
- **allowed_to**: A list of `ShipType`s that the pilot is permitted to board.

Other attributes for quality-of-life may include:

- **forename**: To output their name to the dashboard.
- **surname**: To output their name to the dashboard.

1.2 Tide

As harbourmasters, we are solely responsible for analysing the current tide conditions to judge whether or not it is safe to bring a ship into port. The structure of a `Tide` object is as follows:

- **tide_id**: A unique identifier for the tide.
- **until**: The time at which the current tide ends or changes height.
- **height**: The height of the tide.
- **type**: The TideStage; may not be needed.

1.2.1 Collection and Storage of Tide Data

Tide data will be stored in a simple MySQL database which is accessed via TideDAO. This concept of using DAOs also applies to other classes, but it's important to focus on the tides specifically as it requires collection from an external source, i.e. the internet.

If possible, we would like to make use of a free API to get the most up-to-date tide information with little manual effort on our end. There are many APIs available, notably the one from admiralty.co.uk for all UK ports, though it is quite expensive. As finding a free API is quite difficult, we may have to suffice for either web-scraping tide information from public websites, or simply making up our values. Regardless of how the information is obtained, it wil be stored in a way similar to how the object is structured. Example:

tide_id	until	height	type
74	2021-10-19 10:36	1.00	LOW
75	2021-10-19 13:36	4.39	HIGH

1.3 TideStage (Enumeration)

A very simply enumeration that stores what stage the current tide is considered.

- **LOW**: A low tide.
- **HIGH**: A high tide.

1.4 Ship

The design of the `Ship` class is to be agreed with the shipping line and stevedore team, making the structure absolutely subject to change. Currently, we have the following structure in mind:

- **ship_id**: A unique identifier for the ship.
- **type**: The type of ship, given by the enumeration '`ShipType`'.
- **minimum_tide_height**: The minimum tide height that the ship can be safely brought in at.
- **maximum_tide_height**: The maximum tide height before it is deemed unsafe to be brought in and must wait.

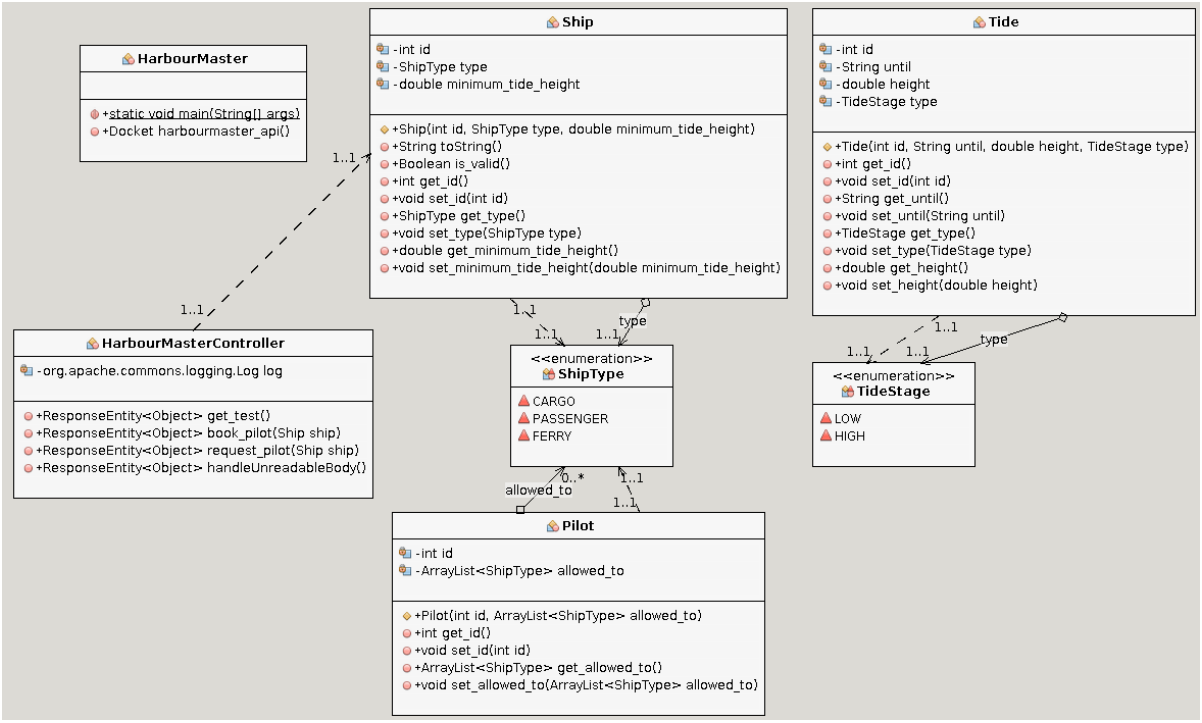
1.5 ShipType (Enumeration)

ShipType is not a class, but a simple enumeration. Similar to the `Ship` class, the design of this is also dependent on the shipping line team. We currently have:

- **PASSENGER**: A ship whose primary function is to carry out passenger on the sea.
- **FERRY**: Another type of passenger vessel.
- **CARGO**: A very large ship that carries cargo, goods, and materials from one port to another.

1.6 Class Diagram

The DAOs have not been added to our experimental project, hence the absence of them from the diagram. It is still a work in progress, and the attributes of the classes are likely to change.



2 Responsibilities

We are responsible for exposing the following REST endpoints:

2.1 /api/bookPilot (for the Shipping Line)

`/api/bookPilot` is an endpoint that is needed by the shipping line team to book a pilot to bring in the ship when they arrive at the port. This endpoint will take in a `DateTime` and `Ship` object. From here, our system will perform checks regarding time availability and ability to board the given ship. Once a list of suitable pilots has been accumulated, a random one will be chosen and their ID returned to the shipping line team where they will continue their role. If no pilots are available that clear these conditions, then an error message will be returned along with the next time that a pilot is available.

2.2 /api/requestPilot (for the Shipping Line)

WIP `/api/requestPilot` is needed by the shipping line when the time comes for a pilot to be sent out and retrieve ships that have arrived to port. We will require, as a minimum, either a ship ID or the pilot ID that has been booked, the coordinates of the destination berth, as well as the current coordinates of the ship. From here, we will calculate the distance between the pilot and the ship, the time it takes for the pilot to travel the distance (assuming a constant speed of about 100mph) to the ship, and the time for the pilot to lead the ship (depending on the speed of the ship type) to the booked berth's coordinates. We will then signal to the stevedore team that a ship has been brought in using an endpoint of theirs.

3 Requirements from others

3.1 /api/shipArrived (from the Stevedore team)

WIP `/api/shipArrived` is an endpoint that we'll need the stevedore team to expose to us so that we can signal to them when the piloting process has been complete. At the moment, we believe that they'll only need the {berth_id} of the berth that the ship has been lead to so that they can move their team there, however it's almost certain that they'll need more information about the ship to perform their job; the `Ship` object being a likely candidate to need altering. We intend to discuss with them about their requirements. They stated in their presentation that they'll need to know information about what's on board in terms of cargo, waste, passengers, etc., so we may have to revise the `Ship` class after a discussion with both the shipping and stevedore team.