Task 2: Design patterns

2.1

- Chain of Responsibility:

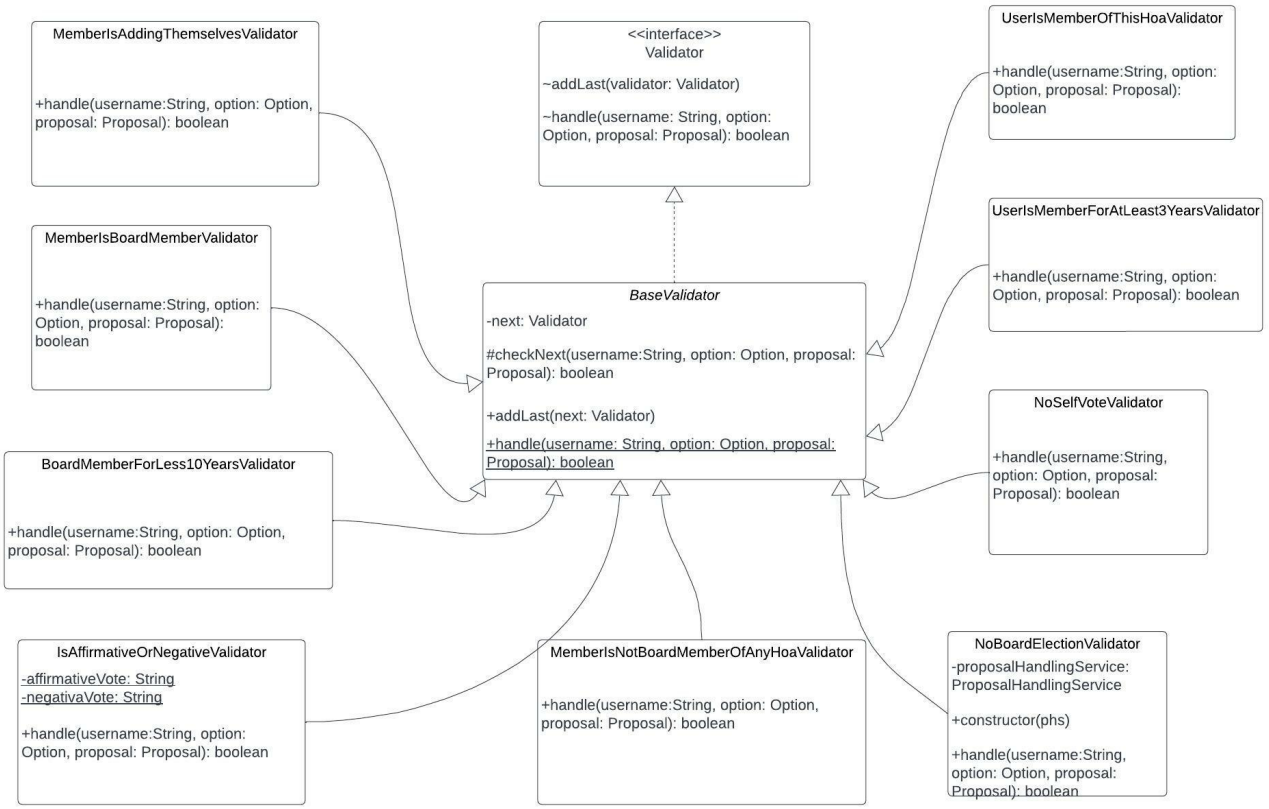
The chain of responsibility is a design pattern which aligns different checks one after the other to find a validator that can fulfil the request or pass it on to the next one for doing many different checks one after the other. In the Voting service, for each new vote and new options to a proposal, we need to do multiple eligibility checks. For example: a member cannot vote for himself, a member cannot nominate himself unless he has been in the HOA for at least 3 years, etc. The Chain of Responsibility perfectly fits the situation. We implemented different checks, or validators, which all extend from an abstract class named Validator. These validators take in users' netIDs and information about the votes, options, proposals and perform checks on these given information.

- Strategy:

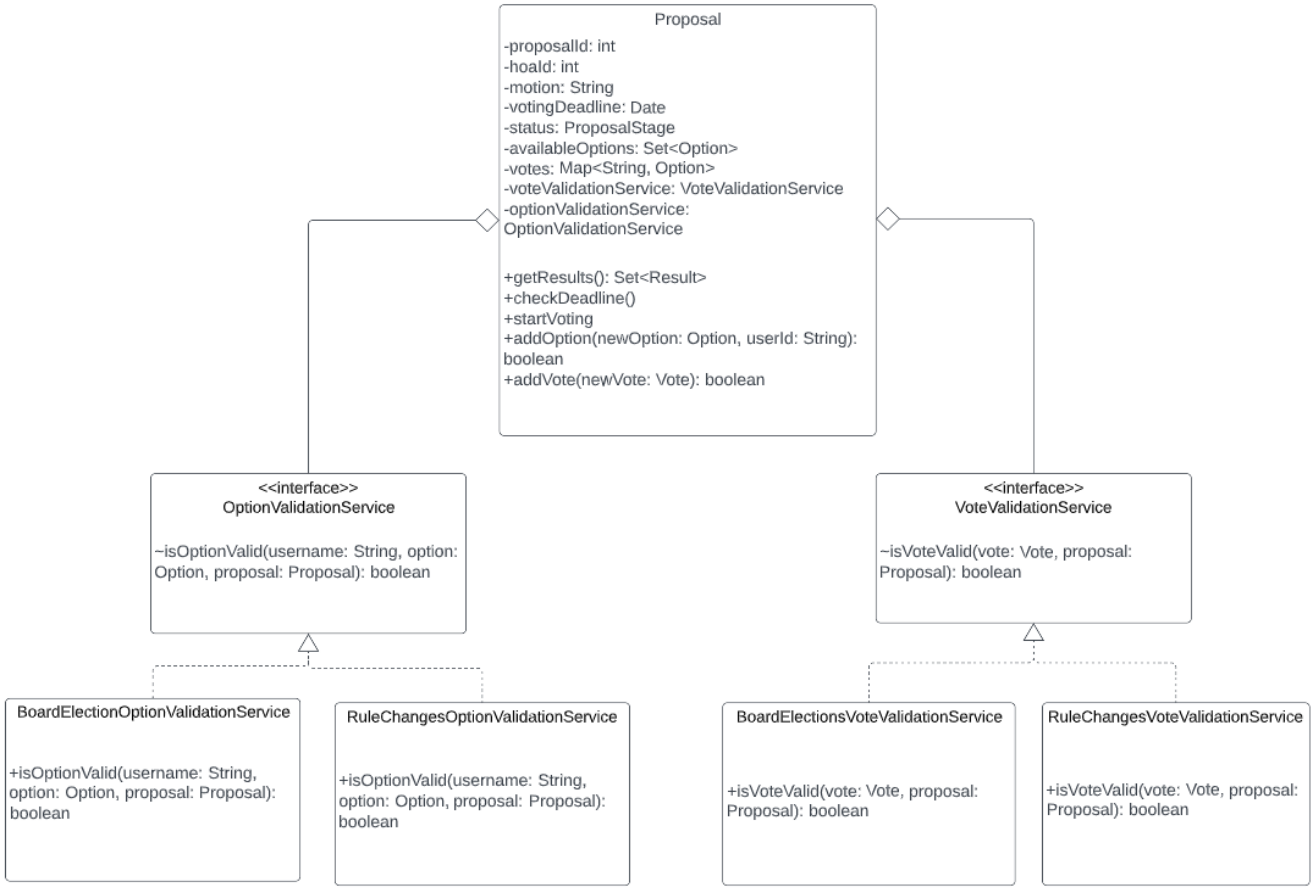
The strategy pattern enables an algorithm's behaviour to be selected at runtime by defining a family of algorithms, encapsulating them and making the algorithm interchangeable. The Voting microservice is responsible for 2 types of voting procedures: one for board election and one for rule changes election. Although the logic for counting the votes is the same for the 2 types of elections, each of them has different eligibility checks for new votes and new options. For example, for an incoming vote, the board election checks whether the voter is voting for himself while the rule changes election checks whether the voter is a member of a specific HOA. Therefore we wrote the OptionValidationService and VoteValidationService interfaces, each has different concrete implementations for board election and rule changes election. The Proposal object stores its corresponding Validation services. When a request is passed, the endpoints decide whether the request should go towards one or the other.

2.2)

- Chain of Responsibility



- Strategy



2.3) We mention the location of where the design patterns' implementation can be found here:
Chain of responsibility: the folder voting-microservice\sem\voting\domain\services\validators contains the implementation classes of the validators together with the *interface* **Validator** and the *abstract class* **BaseValidator**.
Strategy: the folder voting-microservice\sem\voting\domain\services contains the interfaces for the two validation services and the subfolder \implementations contains the concrete classes for the two strategies: **RuleChanges*** and **BoardElection***.