# Assignment 1, Task 2

SEM Group 26A

Tommy Hu
Andrei Drăgoi
George Iftode
Konstantinos Stergiou
Lennart Verstegen
Ansh Kumar

# Design Patterns

## Facade Pattern

The first design pattern that we decided to use was the facade pattern. This is because in general when there is a user using this application the only aspect that they would see is the user side, while there are many more method calls and instructions happening in the background of every method. For instance, in the case that a HR representative wants to create a new contract, the application takes the information for the contract and sends it to the request microservice that handles communicating everything to the other microservices, afterwards the data is sent to the contract microservice where it is constructed and then sent back to the request microservice, which then sends the data to the notification microservice, which sends data to the user microservice. Without going through every single method call and every single request this is just the general flow of data that is done simply to just create a contract.

The primary reason that we decided to use the facade pattern is because the user of the application won't ever have to access any other microservice. They simply need to send a request to perform whatever task they want to perform, while the whole process is happening in the background, and at the end the user gets what it wants.

To summarise , we decided to use the facade pattern because for every task that the user wants to perform there are many background operations being performed that the user does not need to see. This ensures that the application is easy to use for all users regardless of their experience with these types of softwares.

## Builder Pattern

The second design pattern that we decided to use was the builder pattern. We used the builder pattern in the construction of contracts. Currently there is only one type of contract that can be constructed, a custom contract where each piece of information needs to be filled out. Although this is the case right now this would most definitely change in the future.

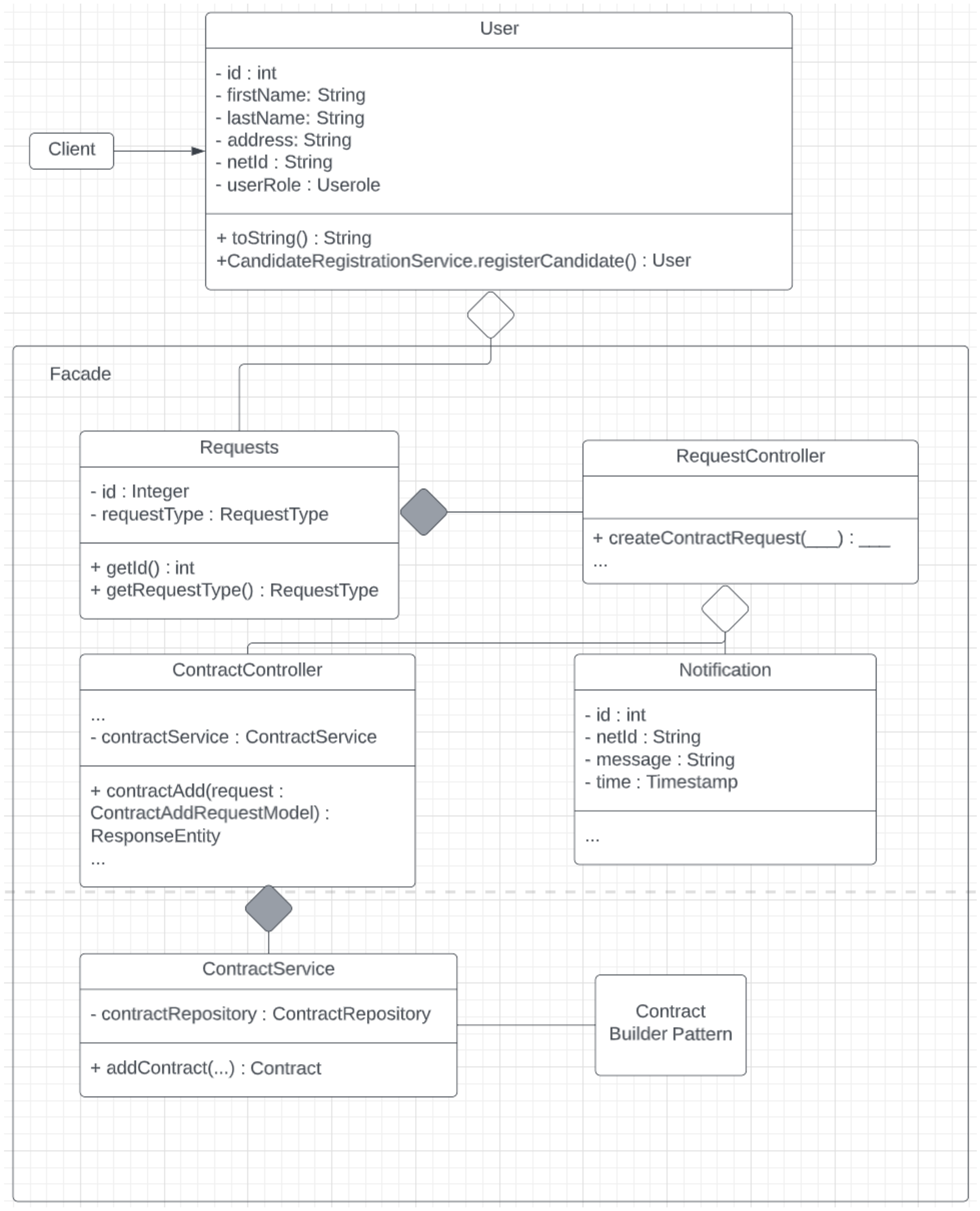
Primarily because there are multiple different types of contracts that are very common and all have the same features. A common instance of this are, summer internships. Many companies offer summer internships to students which all have the same conditions such as; salary, work hours, contract duration, etc. Therefore rather than having the HR representative write out every piece of information for every internship they can simply just create a summer internship contract where they only need to fill in the candidates name and the employers name.

The builder pattern is especially useful here since to implement this new summer internship contract only one simple method would have to be implemented in the Director class. There are many more types of contracts that are commonly seen such as permanent contracts, temporary contracts, etc. that could also be added.

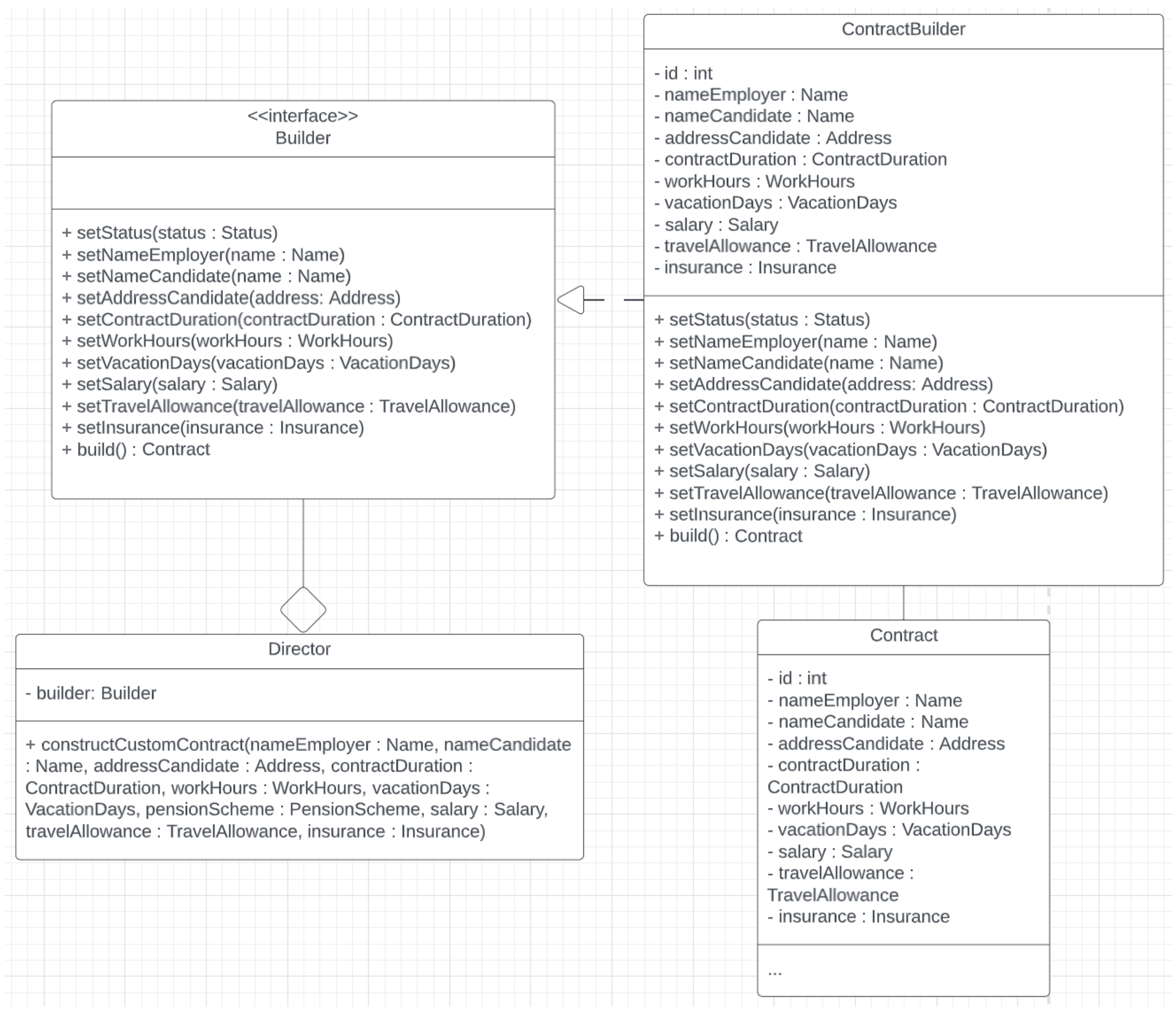
To summarise, we decided to use the builder pattern due to its scalability potential.  Since most contracts all have the same basic features, and since there are many common types of contracts, as stated above, we decided to implement the builder pattern.

# UML Class Diagrams

## Facade Pattern



## Builder Pattern


<<interface>>
Builder
+ setStatus(status : Status)
+ setNameEmployer(name : Name)
+ setNameCandidate(name : Name)
+ setAddressCandidate(address: Address)
+ setContractDuration(contractDuration : ContractDuration)
+ setWorkHours(workHours : WorkHours)
+ setVacationDays(vacationDays : VacationDays)
+ setSalary(salary : Salary)
+ setTravelAllowance(travelAllowance : TravelAllowance)
+ setInsurance(insurance : Insurance)
+ build() : Contract
ContractBuilder
- id : int
- nameEmployer : Name
- nameCandidate : Name
- addressCandidate : Address
- contractDuration : ContractDuration
- workHours : WorkHours
- vacationDays : VacationDays
- salary : Salary
- travelAllowance : TravelAllowance
- insurance : Insurance
+ setStatus(status : Status)
+ setNameEmployer(name : Name)
+ setNameCandidate(name : Name)
+ setAddressCandidate(address: Address)
+ setContractDuration(contractDuration : ContractDuration)
+ setWorkHours(workHours : WorkHours)
+ setVacationDays(vacationDays : VacationDays)
+ setSalary(salary : Salary)
+ setTravelAllowance(travelAllowance : TravelAllowance)
+ setInsurance(insurance : Insurance)
+ build() : Contract
Director
- builder: Builder
+ constructCustomContract(nameEmployer : Name, nameCandidate : Name, addressCandidate : Address, contractDuration : ContractDuration, workHours : WorkHours, vacationDays : VacationDays, pensionScheme : PensionScheme, salary : Salary, travelAllowance : TravelAllowance, insurance : Insurance)
Contract
- id : int
- nameEmployer : Name
- nameCandidate : Name
- addressCandidate : Address
- contractDuration : ContractDuration
- workHours : WorkHours
- vacationDays : VacationDays
- salary : Salary
- travelAllowance : TravelAllowance
- insurance : Insurance
...