# Sprint 1 - Design

## Group task 14P

Team Name: Team Grotle
Tutorial Class: Tue 12:30 EN310
Tutor: Naveed Ali
Team members:

| **Group members** | **Student ID** |
|---|---|
| Hamish | 103607352 |
| Dilni | 103616345 |
| Kamar | 103607585 |
| Melanie | 103489466 |
| Justin | 102589705 |
| Tevy | 103139978 |
| Cormac | 102581060 |

# Software Component Design

The following class diagram shows the design of the software components for GotoGro-MRM that have been completed in Sprint 1. It should be noted that we made a Java application for Sprint 1 and the diagram below would reflect that. However, within Sprint 2 the application would switch from Java to PHP so all team members can contribute to the code.

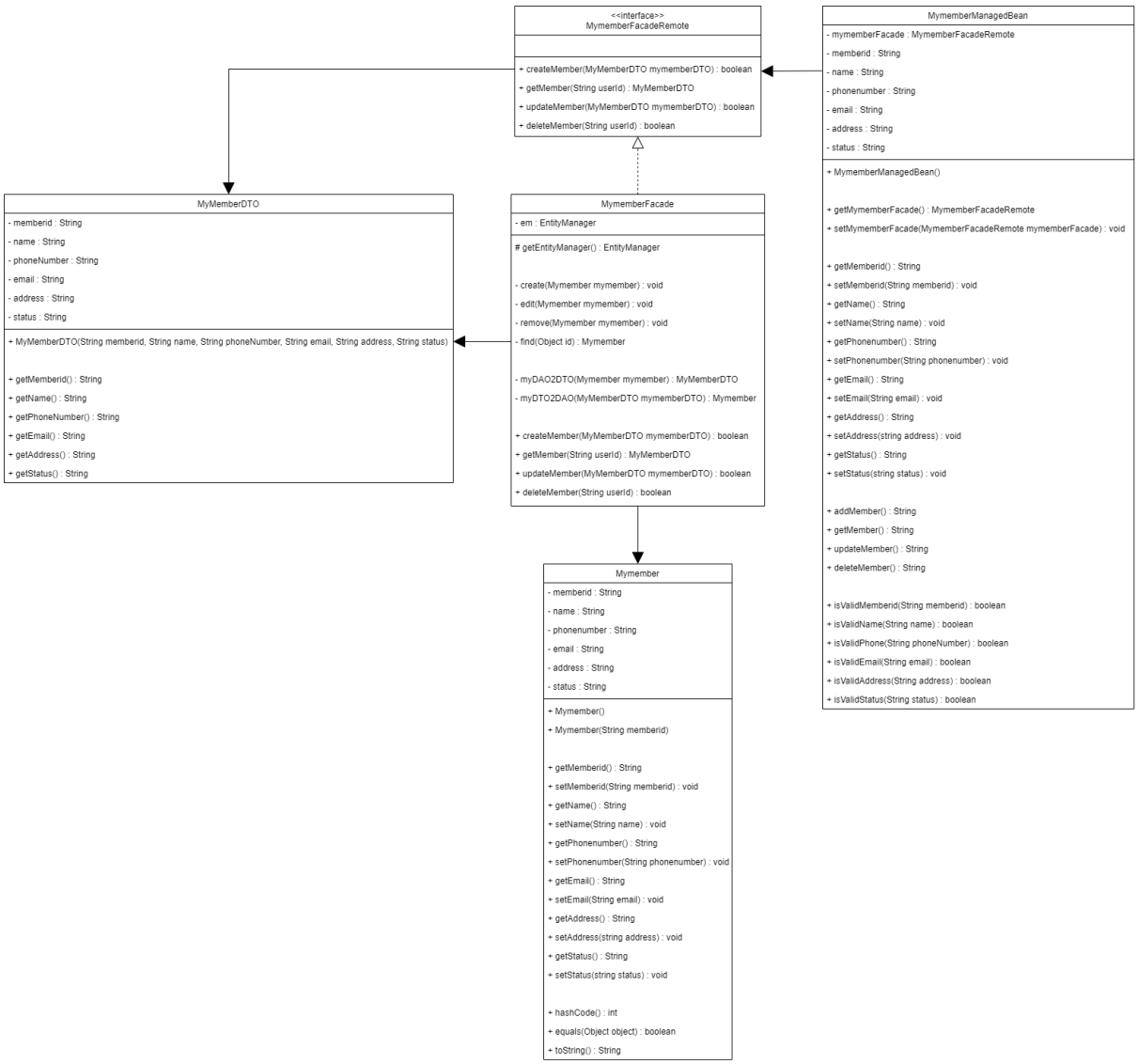


*Diagram 1: Class Diagram of GotoGro-MRM after Sprint 1*

In order to showcase the interaction between classes from the class diagram above, we have created a sequence diagram that shows the deletion process of a member below:

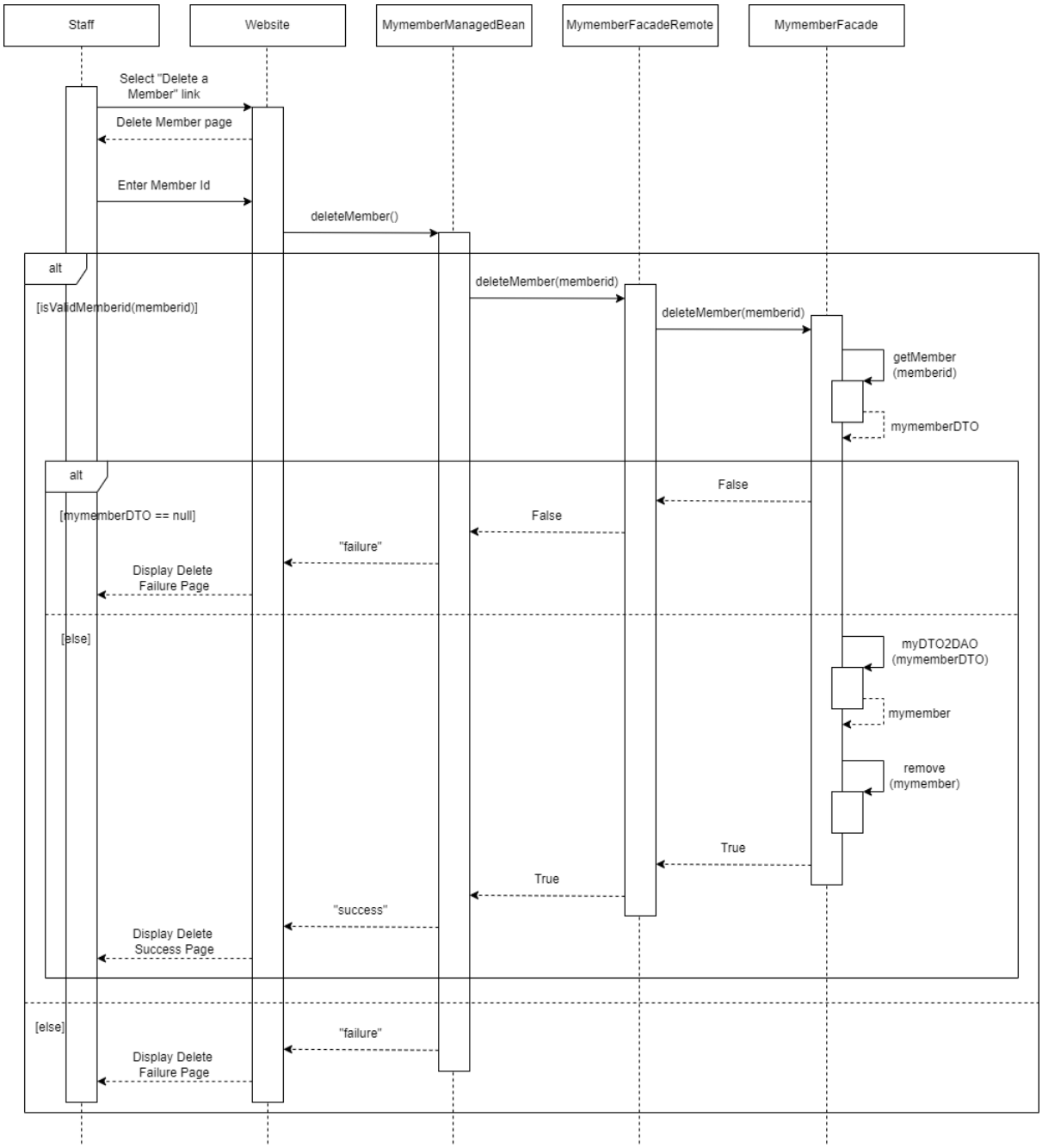


*Diagram 2: Sequence Diagram - Delete Member*

# Design Principles

## Strong Cohesion and Weak Coupling

Cohesion refers to the degree to which the elements inside a module belong together. We have achieved strong cohesion in this application by binding related attributes and methods together into classes, which can be seen in the UML class diagram in Diagram 1.

Coupling refers to the level of interconnection between modules. We have achieved weak coupling in this application the same way we achieved strong cohesion: by binding related attributes and methods together into classes so that they do not depend on other modules to perform operations.

## Object-Oriented (OO) Principles

Our design follows good design principles by having OO principles. We have implemented the following OO principles in our design:

### Encapsulation

We have implemented encapsulation in our design by bundling data and methods into single units (classes). Furthermore, each class contains private attributes that cannot be directly accessed from the outside. These attributes can only be read or updated by using the class's public getter and setter methods.

Using Mymember class as an example, Diagram 3 highlights the class's private attributes and public getters and setters.

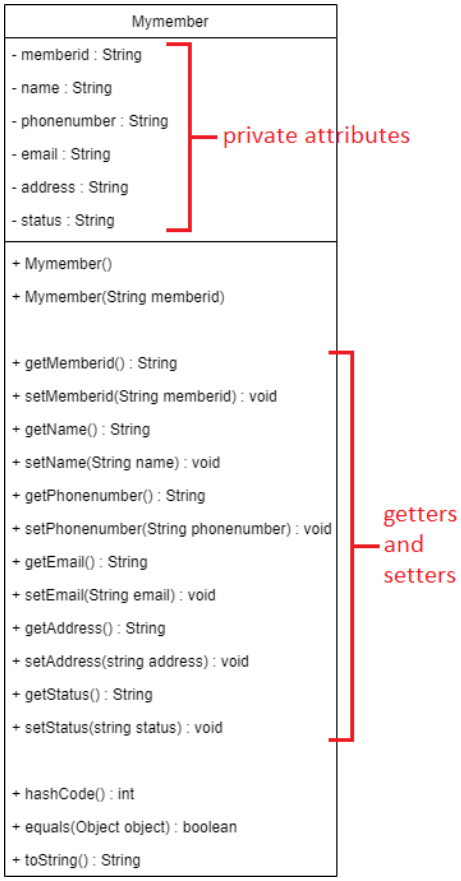


*Diagram 3: Mymember class*

## Abstraction

Our design has achieved abstraction by implementing the MymemberFacadeRemote interface. The interface tells us what needs to be done by declaring methods, but the implementation details of these methods are hidden.

The methods that have been declared (but not implemented) in MymemberFacadeRemote are createMember, getMember, updateMember, and deleteMember. The implementation of these methods are located in the MymemberFacade class.

## Inheritance

The MymemberFacadeRemote interface has also helped our design achieve inheritance, as the MymemberFacade class inherits the methods declared in the MymemberFacadeRemote interface.