Design Document (DD)

Acronyms			
UAB	Universitat Autonoma de Barcelona	SE	Software Engineering
UPF	Universitat Pompeu Fabra		

Revision History			
Version	**Date**	**Comments**	**Autor**
0.1	1-Apt-2019	Initial Draft	ES Team

Contents

List of Figures

List of Tables

1 Introduction

This design document will specify a class diagram and some sequence diagrams which will specify some behaviors of our software. A class diagram is a structure which describes the structure of a system by showing the classes, the attributes, operations or methods and the relationship between them. A class is the description of a group of objects with common properties (attributes), common behavior and common relationships with other objects as well as common semantics. Class diagrams are used to model:

- System vocabulary, as classes are identified as relevant abstractions
- Collaborations, as these classes must be identified so the desired behavior of the software is identified
- It's also a logic database scheme

In the class diagram for this software, you will be able to identify classes and attributes and methods inside them. You will also be able to identify visibility, some comments, the class's name and the multiplicity.

In terms of visibility, there are four types:

- public: accessible for all clients.
- protected: members of a class can only be accessed by sub-classes, friend classes or the same class.
- private: members if a class can only be accessed by friend classes or the same class.
- implementation: can only be accessed by the implementation of the package that contains it.

On the other hand, sequence diagrams can also be found in this document. UML Sequence diagrams are interaction diagrams that detail how operations are carried out. Interaction between objects can be seen in the context of collaboration. They show the order of the interaction visually.

The content of this document it:

- **Introduction**
- **Diagram**
- **Sequence Diagrams**

2 Class Diagram

The image below show a class diagram of MY-NEIGH software. The relationships between classes are defined with arrows. They are represented in the following way:

Type	Symbol	Drawing
Extension	<|--	
Composition	*--	
Aggregation	o--	
Association	--	

Figure 1: Relationships in class diagrams

```
@startuml
User <|-- Neighbour
User <|-- PropertyAdministrator
Neighbour <|-- President

PropertyAdministrator "1" -- "1..n" NA: has >
Neighbour "n" -- "1..n" NA: has >
Warning .> WarningStatus: uses >
Calendar ..> CalendarViewType
NA "1" *-- "1" WarningManagement: contains >
WarningManagement "1" o-- "n" Warning
ChatController -- Chat
NA *-- ChatController: contains >
NA "1" *-- "1" CalendarManagement: contains >
CalendarManagement -- Calendar
Calendar "1" --> "1..n" Event: uses >
NA "1" *-- "1..n" Contacts: has >
Contacts o- Suppliers
NA "1" *-- "1" AccountNA: has >
AccountNA -- Payment
Invoice "1..n" --o "1" NA
Budget "1..n" --o "1" NA
@enduml
```

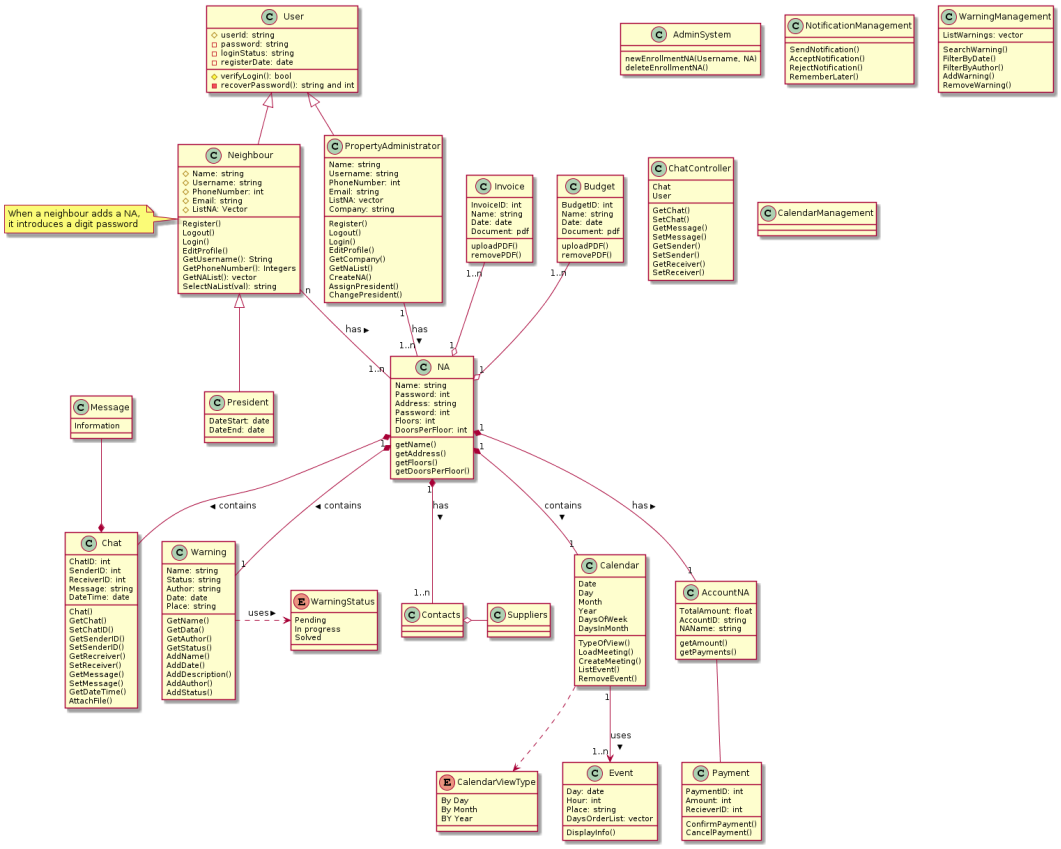

Figure 2: Class diagram tope nicee

3 Sequence Diagrams

3.1 Sequence diagram: Adding an event

The sequence diagram for the addition of an event is shown here. It describes the steps when a user decides to go to the calendar, add an event and add some information about it.

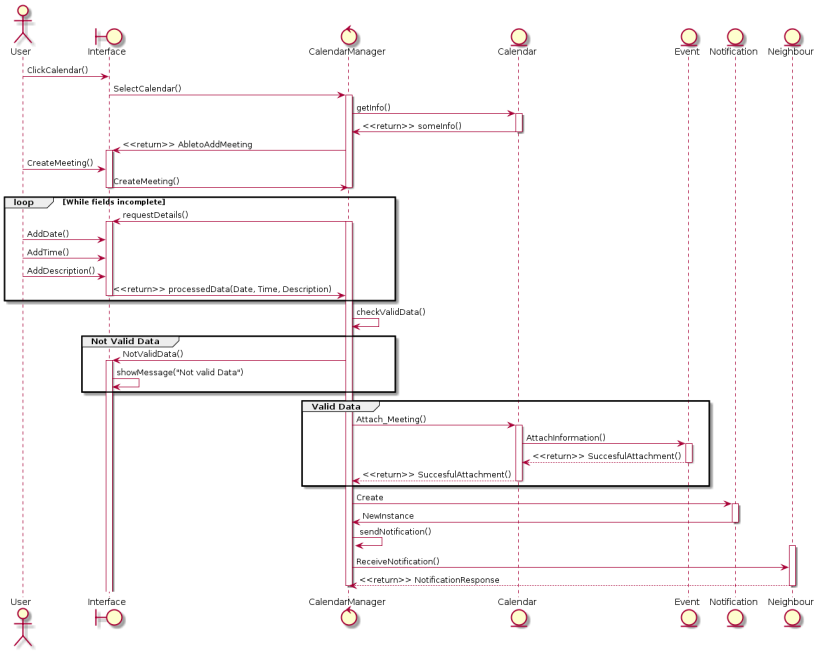

Figure 3: Sequence 1, Addition of an event

3.2 Sequence diagram: Notification management

This is the sequence diagram for the management of notifications. It describes the steps from adding a warning to receiving a notification from the point of view of another neighbour.

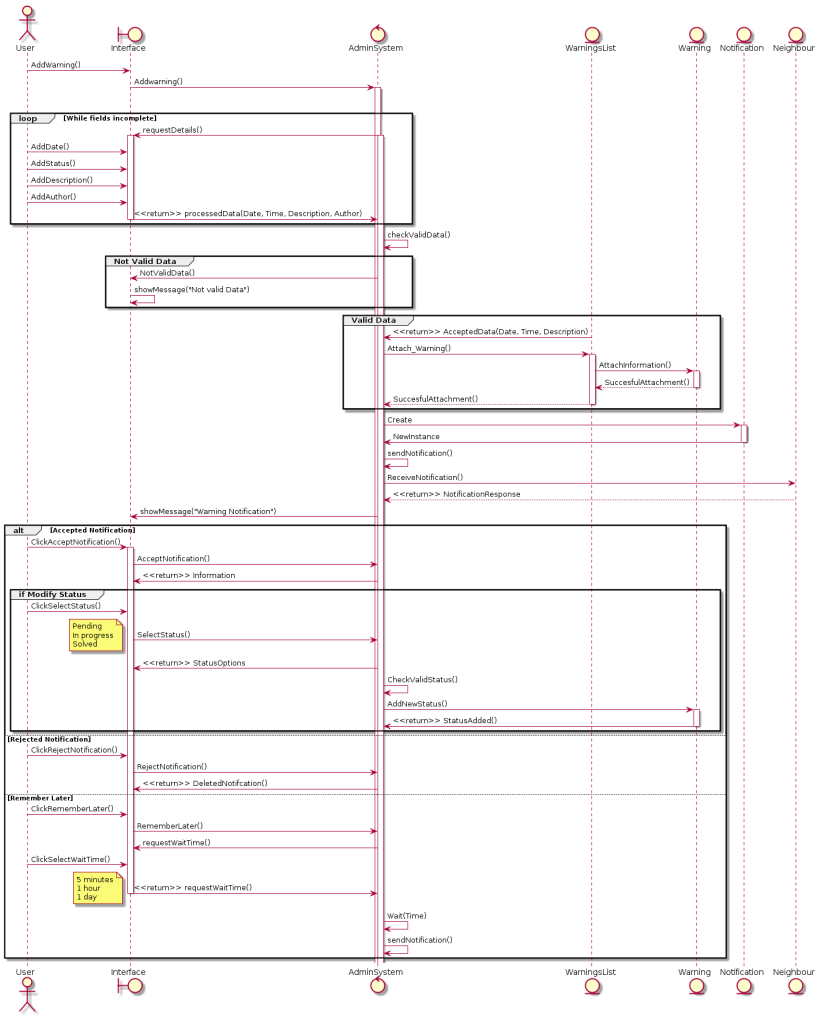

Figure 4: Sequence 2, Notification Management

3.3 Sequence diagram: User management

This is the sequence diagram for user management. The APP administrator will be able to delete a user from the database.

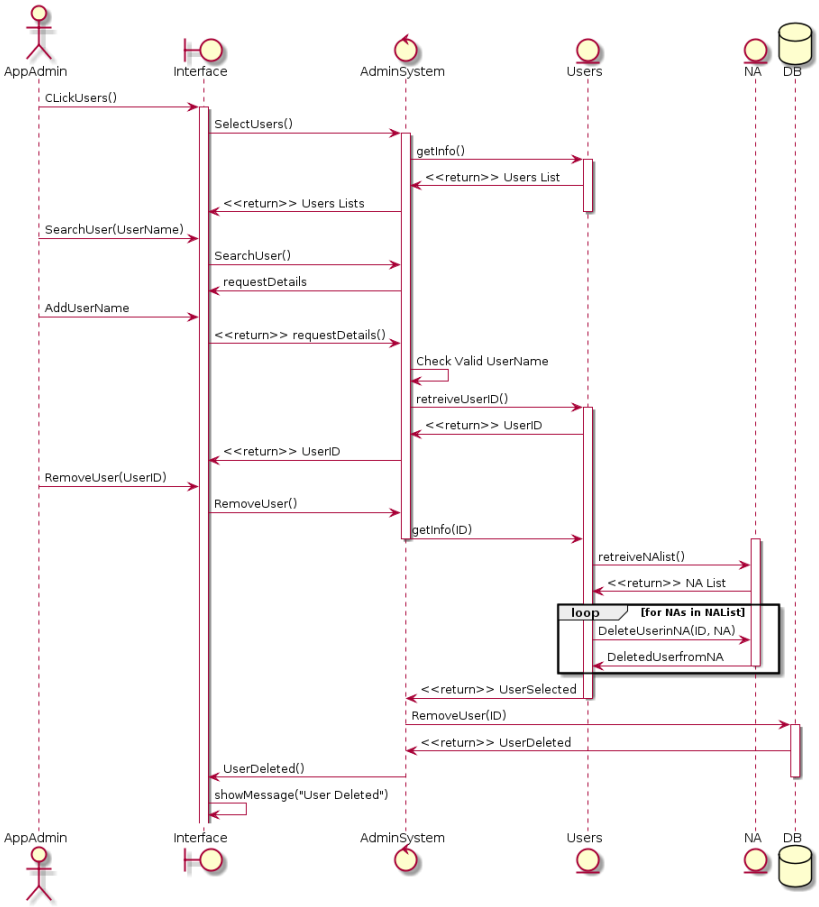

Figure 5: Sequence 3, User Management