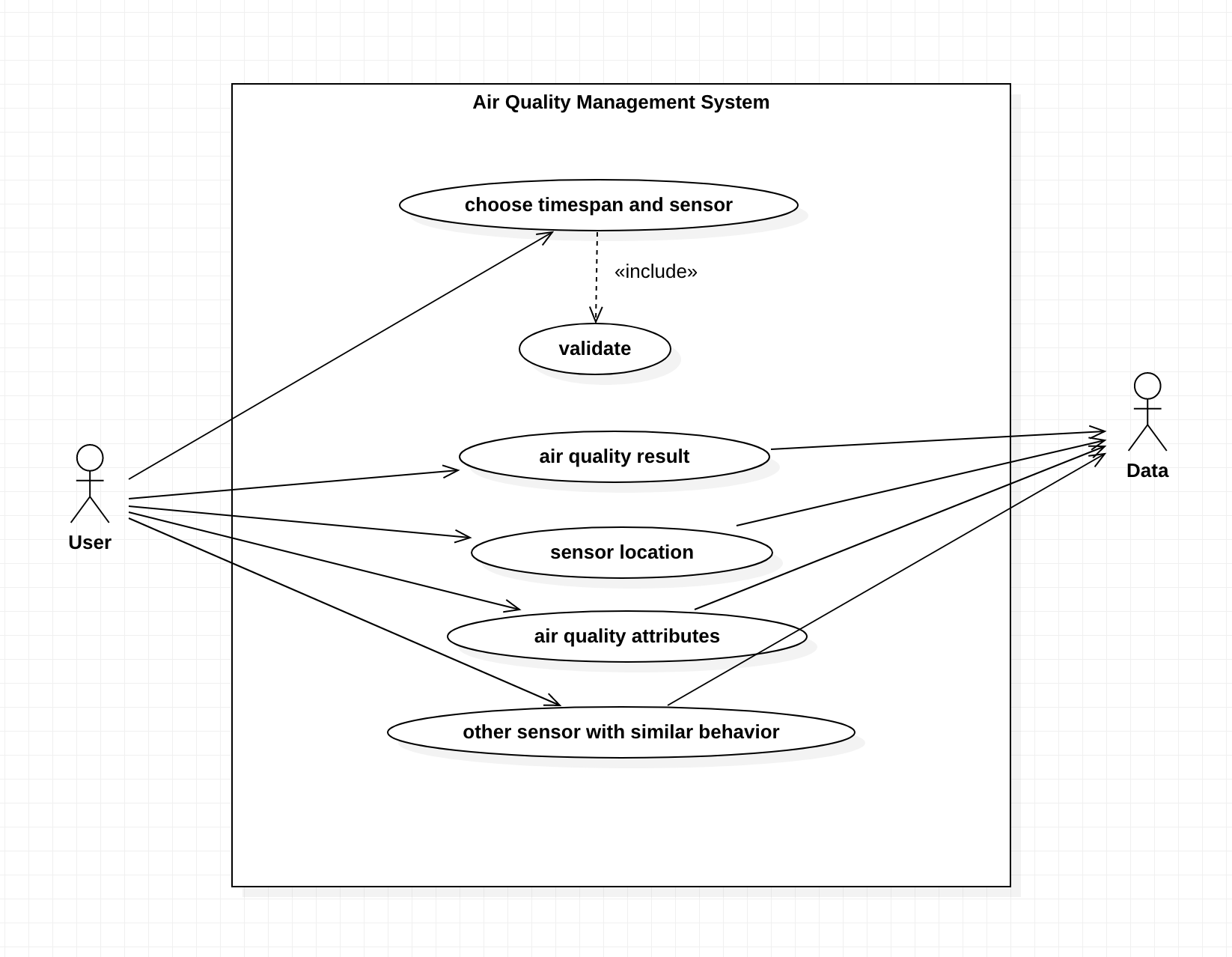
# UML Diagram and Textual Explanation

Adam Satria Adidarma; Gayuh Kautaman Rahmad

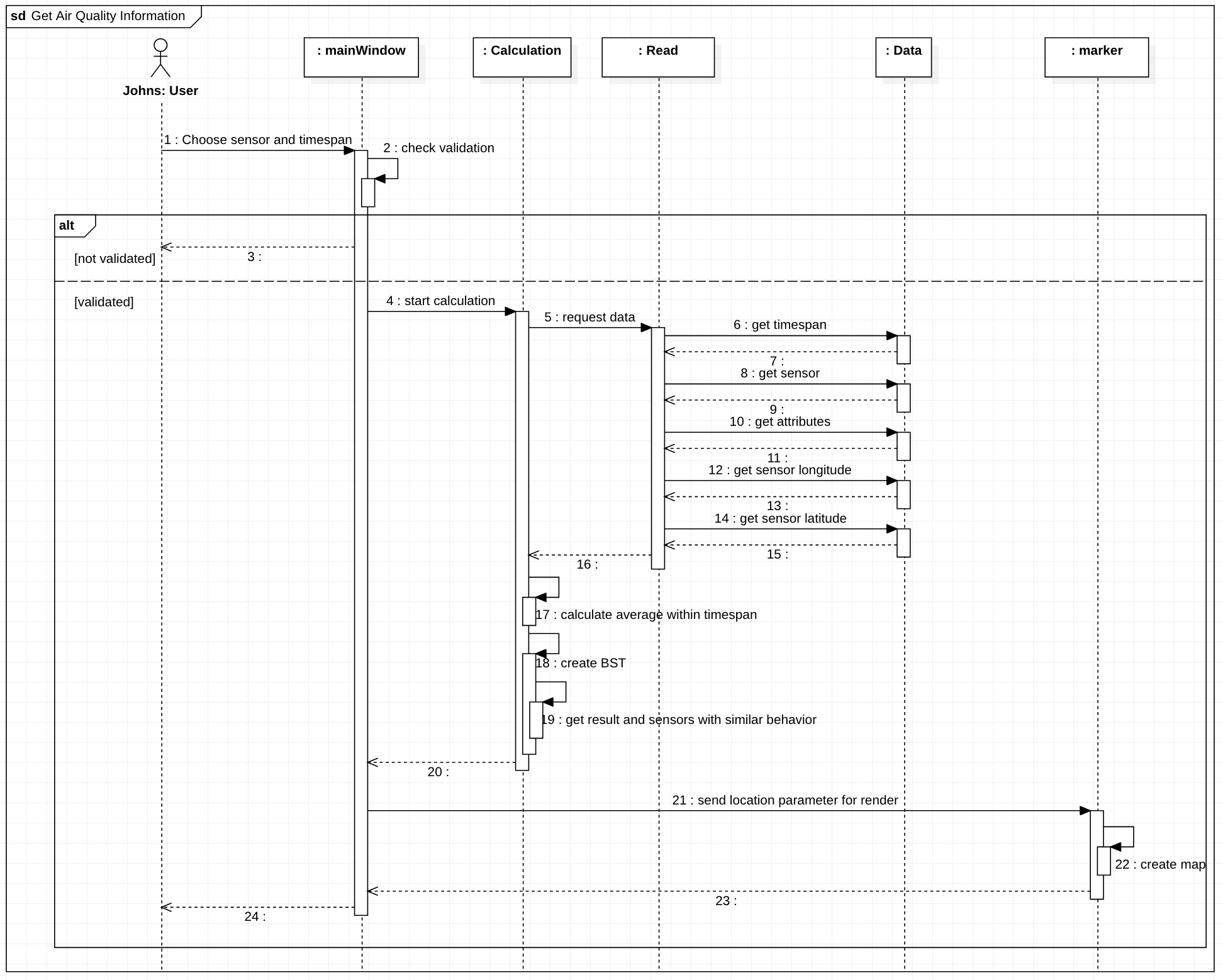
## **Behavior Diagram**

### 1.1 Use Case Diagram



This diagram model the functionality of the system using actors and use cases, these use cases are sets of action and functions that the system needs to perform. Use case diagrams are mainly used to visualize the functionality of a system but not how they are implemented. In our use case diagram, we can see all of our system’s main functionality starting from letting the main actor which is our user to choose the sensor and timespan that will be validated later on inside the system. Next, after the system receives user input of choosing the sensor and timespan, it will get the air quality result and their attributes, sensor location, and other sensors with similar behavior to the user’s chosen sensor for the user to see. The system will be able to complete these sets of actions by using an external data that is provided for the system.

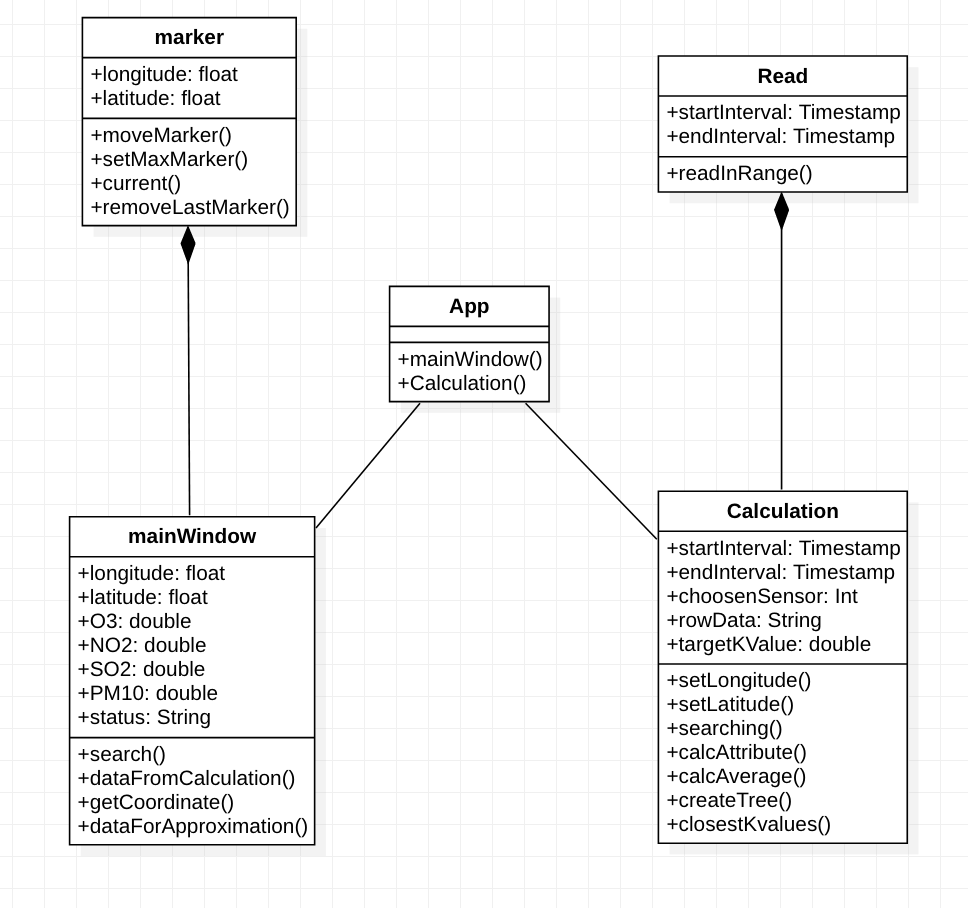
### 1.2 Sequence Diagram



This diagram describes how and in what order objects in classes work together. They help visualize how the classes interact with each other to complete a process. These diagrams are helpful to represent the details of a use case diagram and to model the logic of a sophisticated procedure, functions, and operation. In our diagram, we start with the user choosing the sensor and timespan to the mainWindow class, which triggers the start of the activation box of this class. Next, it will check for validation of user input in its own class. If the input is not validated then it will return an error message to the user and let the user input the correct inputs. If the input is validated, then it will start the Calculation class to request a calculation of these inputs. The Calculation class gets its data from the Read class where it will link to the provided CSV data to get all the components it needs for the Calculation class to start its operations. After getting all the necessary data from the Read class, the Calculation class will start calculating the average values of each attribute within the chosen timespan of each sensor in this timespan. Then it will create a BST to get the calculation result and search for sensors with similar behavior to the chosen sensor. Next, the mainWindow calls the functions of marker class by giving it sensor longitude-latitude for the marker to render the map location of this sensor. Lastly, from the mainWindow, it will display all these sets of action results for the user to see.

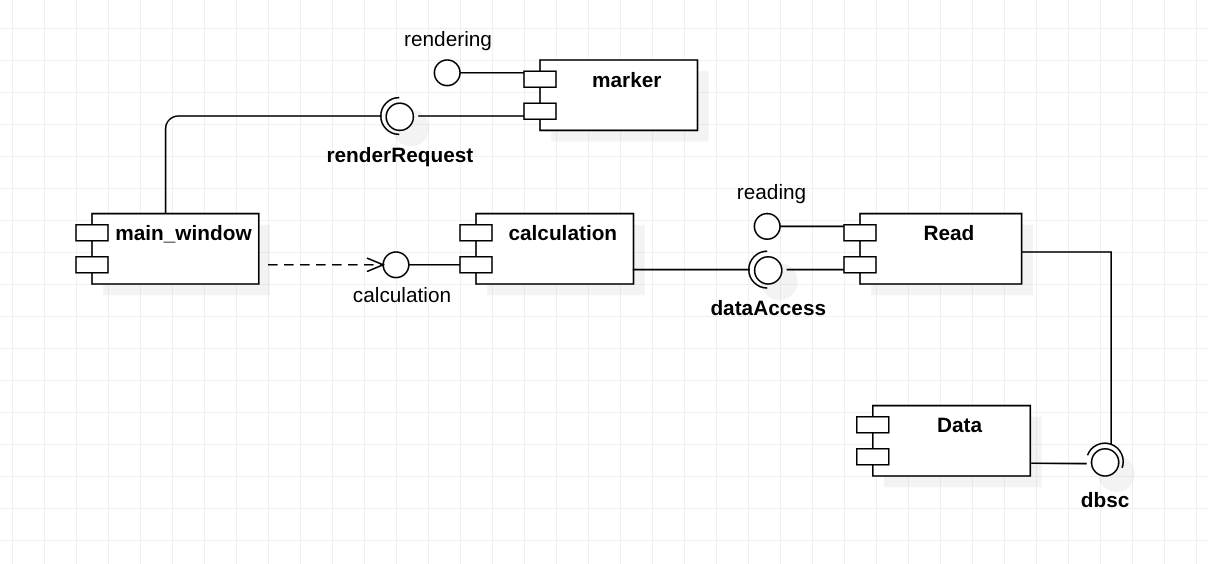
## **Structure Diagram**

### 2.1 Class Diagram



This diagram maps out the class, attributes, operations, and relationship structure of our system. Here in our case, we have 4 main classes with the app class being the API of our system. This API helps communication between the mainWindow and Calculation class with the marker being the parent class of mainWindow and Read as the parent class of Calculation. Both of these parent-child classes have a composition relationship since the child classes cannot operate without their parent classes. The calculation class is responsible to perform all the calculations that this system needs using the calcAttribute and calcAverage operations, it’s also responsible to find sensors that behave similarly with the chosen sensor using createTree and closestKvalues operation. The calculation class receives its required data via Read class which has the readInRange operation to get the data from the CSV file that is provided for the system. All these calculations will be called in the mainWindow class using the dataFromCalculation and dataForApproximation operations. Lastly, the mainWindow class will use the marker class to render the map location of the chosen sensor by passing the coordinates for this sensor to the marker’s operation.

### 2.2 Component Diagram



This diagram is used to see how components interact with each other through the system, or to put it simply how information flows through the system. In the picture above, we can see how components interact with each other within our Air Quality Management System. Our flow starts from the main\_window component which has dependencies to the calculation component since it can’t do anything before receiving data from the calculation component, and the calculation component can do these calculations by requesting the dataAccess services that are provided by the read component and in turn, the read component relies on the data component to provide those data which has the database services to provide read component needs. Lastly, the main\_window will ask for a renderRequest services to the marker component to render the map location of the sensor.