Environmental Monitoring and Prediction System

Software Design Specification

Trevor Howat

8/4/2024

Table of Contents

System Overview

Problem Statement

Climate change has led to an increase in the severity and a decrease in the predictability of weather. It is imperative for the safety of communities and the functioning of our society that a system be created to monitor, predict, and communicate both short and long term environmental conditions to experts and the public.

System Functionalities

The EMPS system will collect a variety of environmental data points and process it for use in creating environmental models and for input in prediction algorithms. It will also provide a way for the user to visualize the data collected and processed, as well as the models and predictions created using the data.

Target Users

1. Research Institutions
    - Collect data to use for research
    - Analyze data to find short and long term trends
    - Use predictions and models as sources for research
2. Local Communities and Governments
    - Information on current and short term weather conditions to make public safety decisions
    - Use models for disaster preparation
3. Large Government Agencies (State or Federal)
    - Access to relevant and accurate information to use in policymaking and setting regulations
    - Access to accurate models for weather events such as hurricanes and tornadoes to assist in disaster preparedness
4. General Public
    - Access to accurate short term weather forecasts for day-to-day planning of life
    - Access to accurate models of potential disasters for personal safety
5. Non-Government Industry and Business
    - Accurate weather models and predictions in order to optimize business operations and use of resources

Business Goals

1. Support Decision Making: Provide data and predictive models to help decision makers make informed decisions about public safety, policy making, and business decisions.
2. Improve Risk Management: Provide accurate models and predictive capabilities to mitigate the effects of large scale natural disasters on the people, infrastructure, and operations of a community.

3. Facilitate Research and Development: Support Research and Development efforts by providing data and models that display the short and long term effects of Climate Change.
4. Enhance Environmental Sustainability: Provide accurate environmental readings and weather models to highlight and encourage the need for sustainable practices in industries.
5. Build Partnerships: Work with industries and governments at all levels to incorporate the EMPS system for mutual benefit.

Non-Functional Requirements

1. Performance:
    - The system shall be able to scale horizontally to add new sensors or stations and vertically to handle increased data volume as the system grows
    - The system shall have an average response time of no more than 30 seconds for data retrievals from the database and no more than 5 minutes for generating models or predictive patterns
    - The system shall be able to handle a data throughput of up to 2 Gbps
2. Maintainability:
    - The system shall be designed with low coupling and high modularity to allow systems to be substituted out in the future
    - The system shall maintain up to date documentation on both how the system works and how it is used by users
    - The system shall maintain comprehensive unit and integration tests that follow industry standards
3. Reliability:
    - The system shall be designed to handle faults gracefully to minimize downtime of the system
    - The system shall be available no less than 98% of the time during regular business hours in the area where it is deployed
4. Usability:
    - The system shall have an intuitive User Interface that follows industry standard design principles and allows for clear navigation and an organized layout.

Use Cases

Actors

Type	Actor	Goal Description
Primary	User	Register with the system and use the system's base functionality.
	System Manager	Configure system for deployed environment.
Supporting	Algorithms Server	Process data collected in the system to generate predictive models.
	Database	Stores data collected by system for use both in the system and outside of it.
Offstage	Emergency Response Agencies	Use Models and Predictions for disaster prep and warnings.
	Government Officials	Use the system to create predictive models for disaster preparations and relief.
	Academic Institutions	Use the system and the data it collects for academic and research purposes.

Use Cases

Register with System

Use Case Section	Description
Use Case Name	Register with System
Goal in Context	The user creates an account to use the system
Scope	System
Level	User-Goal
Primary Actor	User
Stakeholders and Interests	All users who wish to use the system
Preconditions	The system is installed and running in the user's environment
Success Guarantee	The user creates an account and can log into the system
Main Success Scenario	1. User navigates to registration page 2. User enters registration information 3. System validates Registration Information 4. System creates account and displays success message to User 5. User logs in.
Extensions	Registration Information is invalid 3a. System prompts user to enter valid Registration Information Account Creation Fails

	4c. System displays error to user and prompts them to start registration process over.
Special Requirements	Availability: Account Creation (step 4) should succeed 99% of the time

Collect Environmental Data

Use Case Section	Description
Use Case Name	Collect Environmental Data
Goal in Context	The user collects real time environmental data from the system
Scope	System
Level	User-Goal
Primary Actor	User
Stakeholders and Interests	Users who wish to collect real time data for study.
Preconditions	The system is installed and running and the user is logged in.
Success Guarantee	The system displays all available real time data
Main Success Scenario	1. User navigates to the data collection page 2. User selects sensor to get data from 3. User selects "Collect data" 4. System collects data and saves it to database 5. System displays requested data to the User
Extensions	<u>Data Collection Fails</u> 4c. System displays error to user <u>Save to Database Fails</u> 4d. System collects data 5d. System displays data to user with database error
Special Requirements	Availability: Data Collection should succeed 95% of the time Performance: Data collection should average no longer than 5s

Set an Alert

Use Case Section	Description
Use Case Name	Set an Alert
Goal in Context	The user configures the system to send an alert to their email if a reading meets a specified threshold
Scope	System
Level	User-Goal
Primary Actor	User
Stakeholders and Interests	Users who wish to be notified for certain data readings
Preconditions	The system is installed and running and the user is logged in.
Success Guarantee	The system is configured to send alerts when data that matches the threshold is collected
Main Success Scenario	1. User navigates to home page 2. User selects create alert option 3. System displays alert creation form 4. User enters alert information, including alert threshold 5. User selects save alert 6. System saves alert 7. System sends email to user when alert threshold is reached upon data collection
Extensions	<u>User Enters invalid threshold</u> 4a System shows error around threshold and prompts user to enter a valid threshold <u>System fails to save Alert</u> 7a. System displays error to user and prompts them to try again
Special Requirements	Availability: Alert should be sent when data reaches threshold 99.9% of the time

Configure New Sensor

Use Case Section	Description
Use Case Name	Configure New Sensor
Goal in Context	The system manager configures the system to communicate with a new sensor.
Scope	System
Level	User-Goal
Primary Actor	System Manager
Stakeholders and Interests	System Manger wishes to add new sensors to

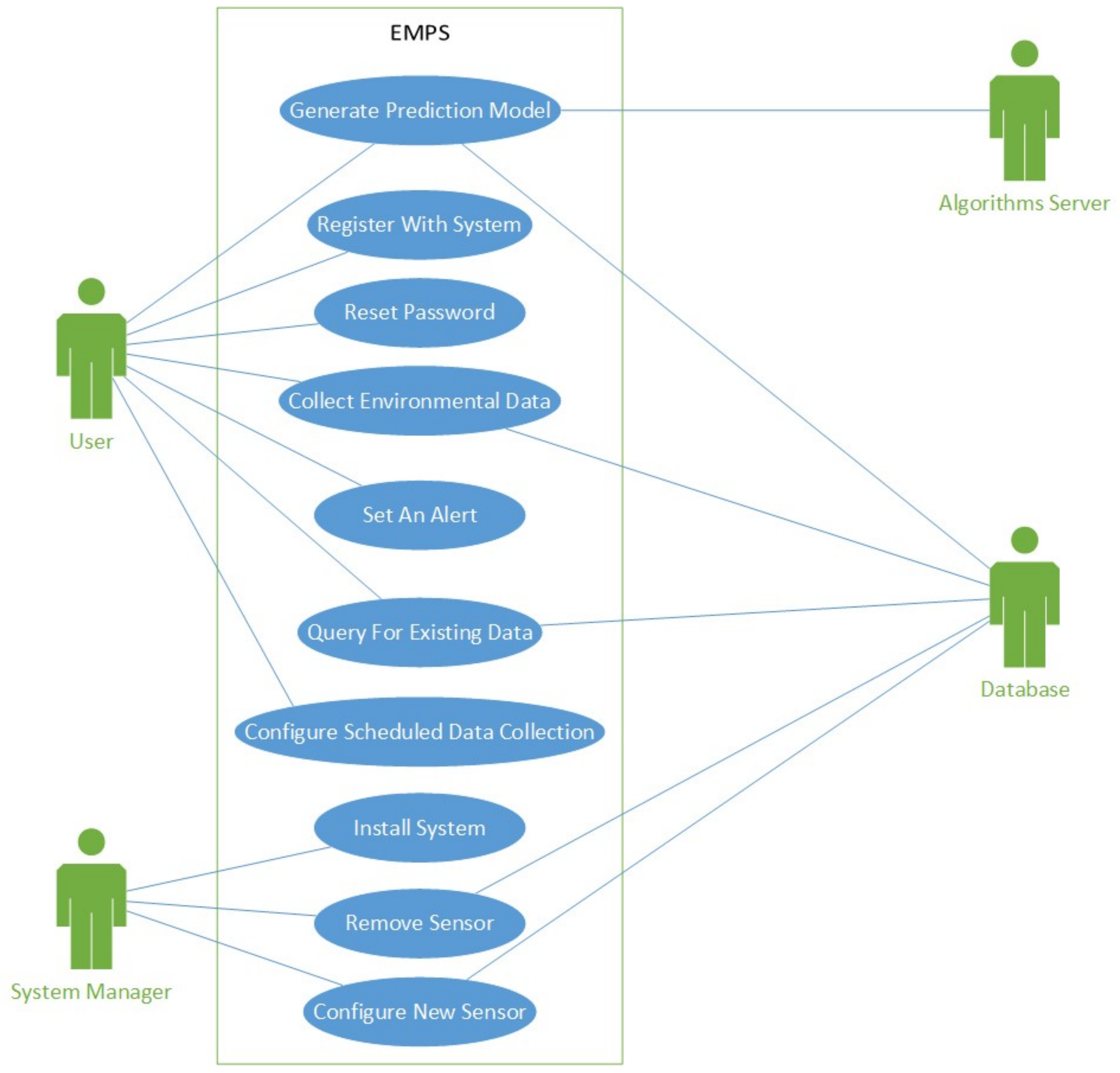
	system to broaden the scope of data that can be collected Users who wish to have more data points available to them
Preconditions	The system is installed and running in the System Manager's environment, and the sensor is deployed and hooked in to the same network as the system.
Success Guarantee	The system can collect data from the new sensor
Main Success Scenario	1. System Manager logs into system with their admin credentials 2. System Manager selects "Add New Sensor" 3. System displays sensor form 4. System Manager enters information into form 5. System Manager selects "Create Sensor" 6. System creates the sensor object and pings it for data 7. System adds new sensor to database and migrates data if needed 8. System shows success message to System Manger
Extensions	<u>System Manager Credentials are Invalid</u> 1a System displays login error to System Manager <u>System Cannot Collect Data from New Sensor</u> 6b. System displays data collection error to System Manager and new sensor is not added <u>System cannot migrate database</u> 7c. System displays database error to System Manager and new sensor is not created
Special Requirements	Performance: Creating a new sensor and migrating the database should average no longer than 3m

Generate Prediction Model

Use Case Section	Description
Use Case Name	Generate Prediction Model
Goal in Context	The user generates weather prediction models with the system.
Scope	System
Level	User-Goal
Primary Actor	User
Stakeholders and Interests	User wishes to view predictive models
Preconditions	The system is installed and running and the user is logged in and the proper data is in the database.
Success Guarantee	The system displays predictive models
Main Success Scenario	1. User selects "Generate Model" on homepage 2. System displays model generation form 3. User selects model type and region 4. System queries for data needed 5. User selects "Generate Model" 6. Algorithms server uses data to generate model and sends the model to the system 7. System displays model to the User
Extensions	<u>Insufficient Data for Model Generation</u> 6a. System displays insufficient data error and lists missing data <u>Algorithms Server Fail</u> 6b. Algorithms Server sends error to System 7b. System displays generation failure error to User
Special Requirements	Availability: Connection to the Algorithms server should be available 99% of the time

NOTE: Other Use Cases found in the use case diagram are not detailed here. This is not a comprehensive view of the system, just an example using its core functionality.

Use Case Diagram



Description

The System Manager and User serve as Primary Actors in the system, while the Algorithms Server and Database are supporting actors. The Database itself serves an interesting role, as it is created maintained by the developers of the EMPS, but also available for use by external users/clients. Thus it has roles as both internal and external to the system. For the purpose of our use cases, we can treat it as internal.

Domain Model

Conceptual Classes

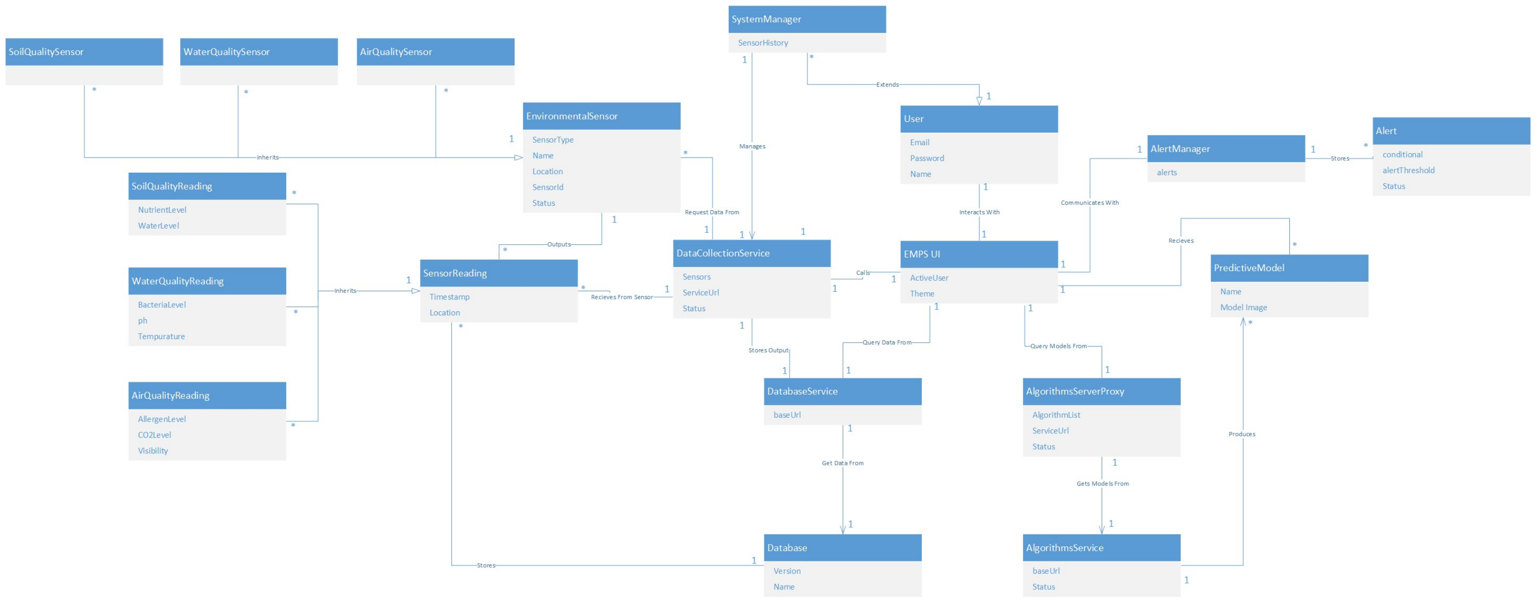
Conceptual Class Category	Example from EMPS
Physical or Tangible Objects	Environmental Sensors, EMPS UI, Soil Quality Sensor, Water Quality Sensor, Air Quality Sensor
Specifications, Designs or Descriptions of Things	Sensor Readings, Stored Data (from DB), User Account, Success Message, Sensor Data Type, Alert, Predictive Model, Data Query, System, Soil Quality Reading, Air Quality Reading, Water Quality Reading
Places	Region
Transactions	Registration
Transaction Line Items	Registration Information, Alert Information
Roles of People	User, User Data Admin, System Manager
Containers of Other Things	Registration Page, Data Collection Page, Home Page
Things in a Container	Stored Data
Other Computers/Systems (external)	Database, Algorithms Server
Abstract Noun Concepts	Account Creation Error, Data Collection Error, Database Error, Data Fields, Login Error, Insufficient Data Error, Generation Failure Error, Database Service, Algorithms Server Proxy, Alert Manager
Organizations	Government Agencies, Research Centers
Events	Data Collection
Processes	Model Generation
Rules and Policies	-
Catalogs	-
Records of Finance, Work, Contracts, Legal Matters, etc.	-
Financial Instruments and Services	-
Manuals, Books, Documents, Reference Papers	-

Class Pruning

Good Classes (Retained)	**Bad Classes (Pruned) - Reasoning**
Environmental Sensors	Stored Data - Redundant
EMPS UI	User Account - Redundant
Sensor Readings	Success Message – Implementation Specific
Alert	Sensor Data Type - Attribute
Predictive Model	Data Query – Implementation Specific
User	System – Vague
System Manager	Region – Attribute

Database	Registration - Vague
Algorithms Server	Registration Page – Implementation Specific
Data Collection	Data Collection Page – Implementation Specific
Air Quality Sensor	Home Page – Implementation Specific
Water Quality Sensor	Account Creation Error - Irrelevant
Soil Quality Sensor	Data Collection Error - Irrelevant
Soil Quality Reading	Database Error - Irrelevant
Air Quality Reading	Data Fields - Irrelevant
Water Quality Reading	Login Error - Irrelevant
Database Service	Insufficient Data Error - Irrelevant
Algorithms Server Proxy	Generation Failure Error - Irrelevant
Alert Manger	Model Generation - Attribute
	Registration Information – Attributes (User Metadata)
	Alert Information - Attributes
	Government Agencies - Irrelevant
	Research Centers - Irrelevant
	User Data Admin - Irrelevant

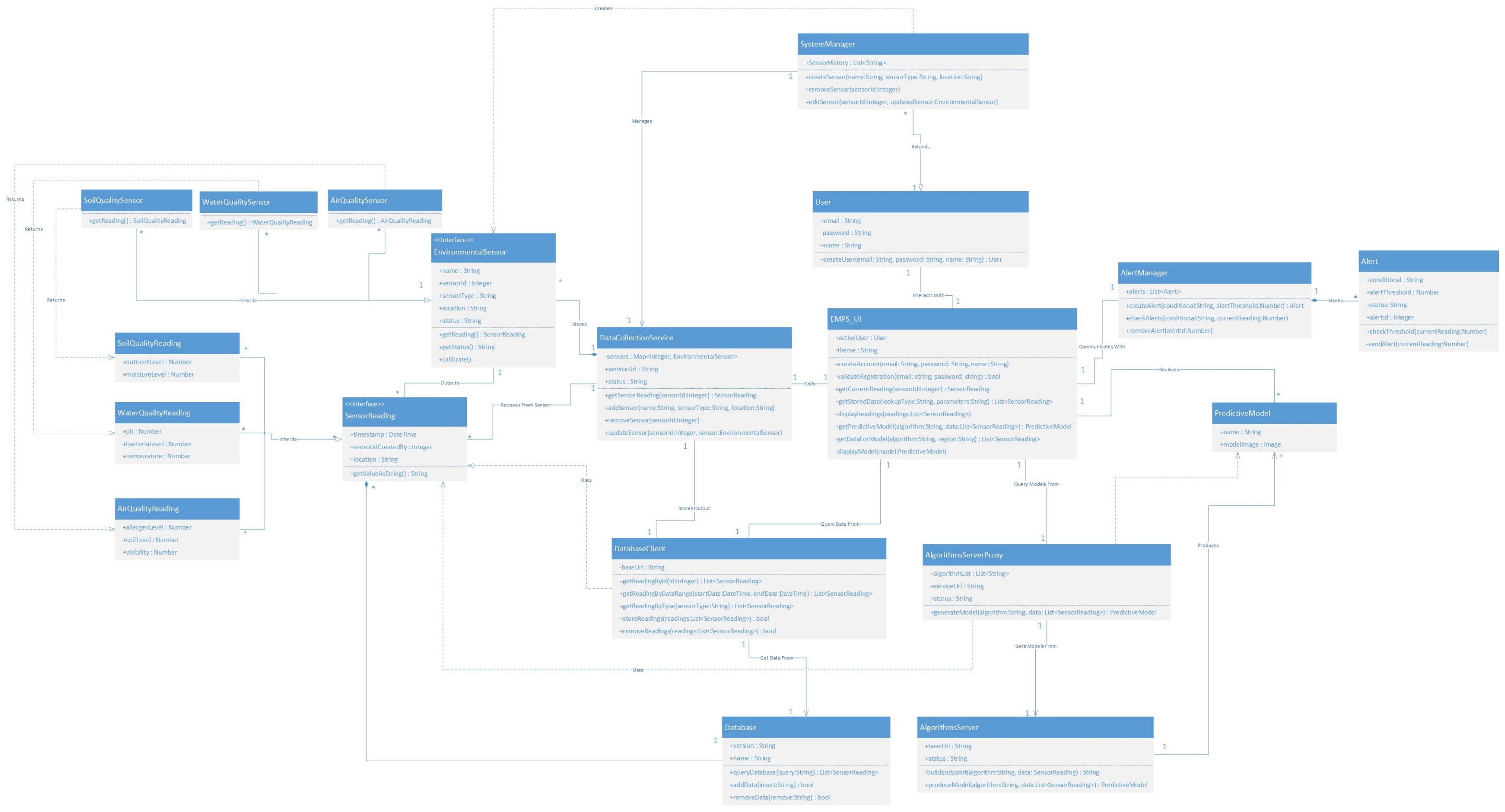
Domain Model Diagram



Description

The Domain Model flushes out the general layout of the system. It sets up the basic areas of the system, Alerts, Data Collection, Model Generation, and Database storage, all of which revolve around a central UI for the user and system manager to interact with. Since System Manager is a special type of user, it extends from the user class, a use of Generalization.

Class Diagram


Creates
SystemManager
+SensorHistory : List<String>
+createSensor(name:String, sensorType:String, location:String)
+removeSensor(sensorId:Integer)
+editSensor(sensorId:Integer, updatedSensor:EnvironmentalSensor)
Manages
Extends
User
+email : String
-password : String
+name : String
+createUser(email: String, password: String, name: String) : User
Interacts With
Returns
Returns
Returns
SoilQualitySensor
+getReading() : SoilQualityReading
WaterQualitySensor
+getReading() : WaterQualityReading
AirQualitySensor
+getReading() : AirQualityReading
<<Interface>>
EnvironmentalSensor
+name : String
+sensorId : Integer
+sensorType : String
+location : String
+status : String
+getReading() : SensorReading
+getStatus() : String
+calibrate()
Inherits
Stores
SoilQualityReading
+nutrientLevel : Number
+moistureLevel : Number
WaterQualityReading
+ph : Number
+bacterialLevel : Number
+tempurature : Number
AirQualityReading
+allergenLevel : Number
+co2Level : Number
+visibility : Number
Inherits
Outputs
<<Interface>>
SensorReading
+timestamp : DateTime
+sensorIdCreatedBy : Integer
+location : String
+getValueAsString() : String
Recieves From Sensor
DataCollectionService
-sensors : Map<Integer, EnvironmentalSensor>
+serviceUrl : String
+status : String
+getSensorReading(sensorid:Integer) : SensorReading
+addSensor(name:String, sensorType:String, location:String)
+removeSensor(sensorId:Integer)
+updateSensor(sensorId:Integer, sensor:EnvironmentalSensor)
Calls
EMPS_UI
+activeUser : User
-theme : String
+createAccount(email: String, password: String, name: String)
-validateRegistration(email: string, password: string) : bool
+getCurrentReading(sensorId:Integer) : SensorReading
+getStoredData(lookupType:String, parameters:String) : List<SensorReading>
-displayReadings(readings:List<SensorReading>)
+getPredictiveModel(algorithm:String, data:List<SensorReading>) : PredictiveModel
-getDataForModel(algorithm:String, region:String) : List<SensorReading>
-diaplayModel(model:PredictiveModel)
Communicates With
AlertManager
+alerts : List<Alert>
+createAlert(conditional:String, alertThreshold:Number) : Alert
+checkAlerts(conditional:String, currentReading:Number)
+removeAlert(alertId:Number)
Stores
Alert
+conditional : String
+alertThreshold : Number
+status: String
+alertId : Integer
+checkThreshold(currentReading:Number)
-sendAlert(currentReading:Number)
Recieves
PredictiveModel
+name : String
+modelImage : Image
Uses
Stores Output
Query Data From
Query Models From
Produces
DatabaseClient
-baseUrl : String
+getReadingById(id:Integer) : List<SensorReading>
+getReadingByDateRange(startDate:DateTime, endDate:DateTime) : List<SensorReading>
+getReadingByType(sensorType:String) : List<SensorReading>
+storeReadings(readings:List<SensorReading>) : bool
+removeReadings(readings:List<SensorReading>) : bool
AlgorithmsServerProxy
+algorithmList : List<String>
+serviceUrl : String
+status : String
+generateModel(algorithm:String, data: List<SensorReading>) : PredictiveModel
Get Data From
Uses
Gets Models From
Database
+version : String
+name : String
+queryDatabase(query:String) : List<SensorReading>
+addData(insert:String) : bool
+removeData(remove:String) : bool
AlgorithmsServer
+baseUrl : String
+status : String
-buildEndpoint(algorithm:String, data: SensorReading) : String
+produceModel(algorithm:String, data:List<SensorReading>) : PredictiveModel

Description

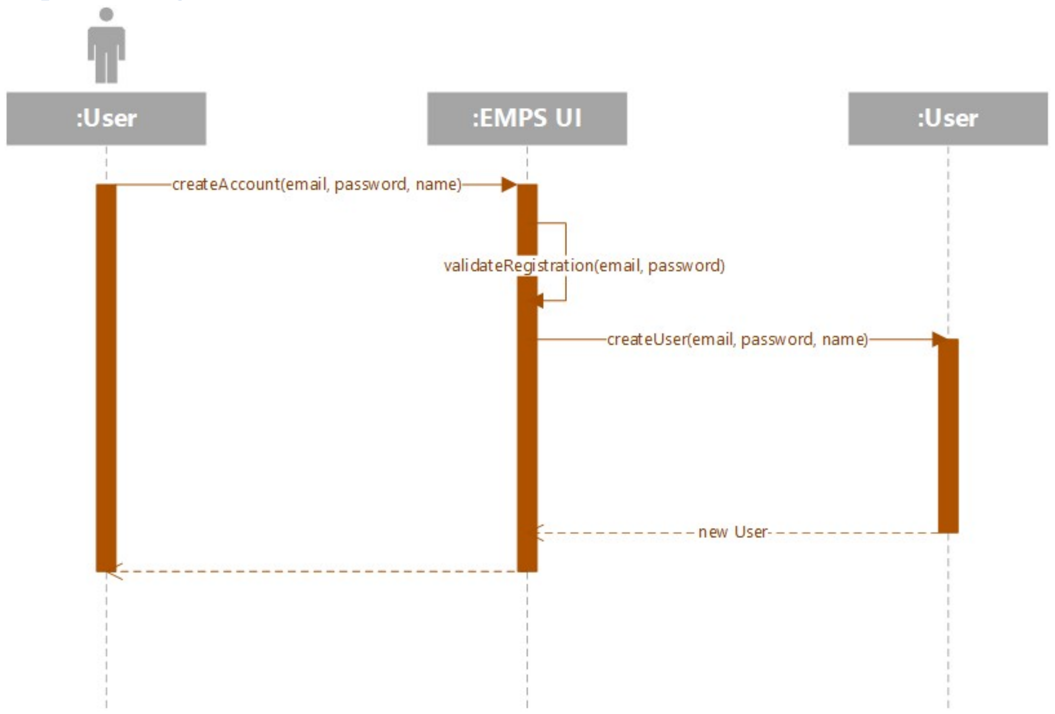
The Class Diagram expands upon the domain model to formulate how the different classes are laid out. In the Diagram, we make use of different patterns in an attempt to increase the maintainability of the system while keeping it easy to understand. In order to lower coupling between the UI and the Database/Algorithms Server, we make use of Pure Fabrication and indirection by creating classes that do not represent real world objects (DatabaseClient and AlgorithmServerProxy) whose responsibility is to communicate with their respective services.

Sequence Diagrams and Systems Operations Contracts

Use Case: Register with System

Field	Description
Name:	Create Account
Responsibilities:	Create a new User Account
Type:	System
Cross Reference:	Use Case: Register with System
Notes:	N/A
Exceptions:	If the user provides invalid information, indicate an error.
Output:	None
Pre-conditions:	The System is installed and running
Post-Conditions:	A new User account is created

Sequence Diagram


:User
:EMPS UI
:User
createAccount(email, password, name)
validateRegistration(email, password)
createUser(email, password, name)
new User

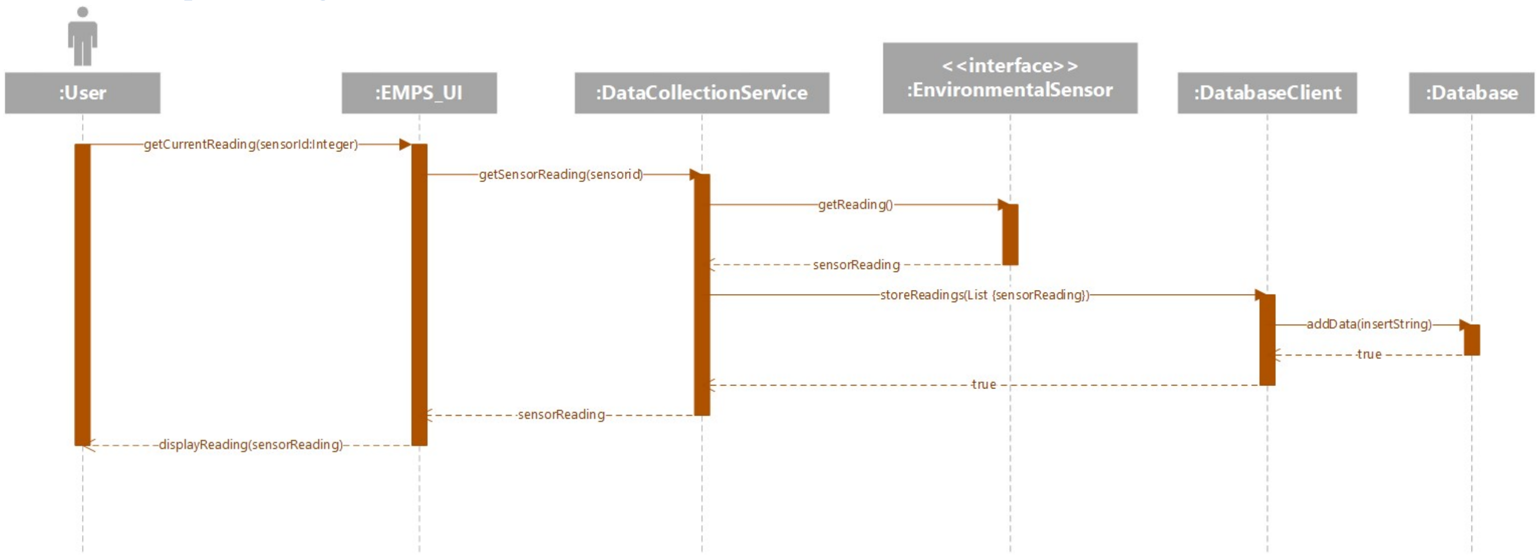
Description

This is a simple sequence of a user creating an account. The User invokes the UI which validates the data given by the user, then invokes the User object to return a new User.

Collect Environmental Data

Field	Description
Name:	Get Current Reading
Responsibilities:	Retrieve a Sensor Reading of the current sensor value
Type:	System
Cross Reference:	Use Case: Collect Environmental Data
Notes:	N/A
Exceptions:	• If the sensor fails, indicate sensor error • If Sensor passes but save to database fails, indicate database error, but still return reading
Output:	None
Pre-conditions:	The User is logged in to the System There are available sensors configured with the system.
Post-Conditions:	A new Sensor Reading is created and added to the database

Sequence Diagram



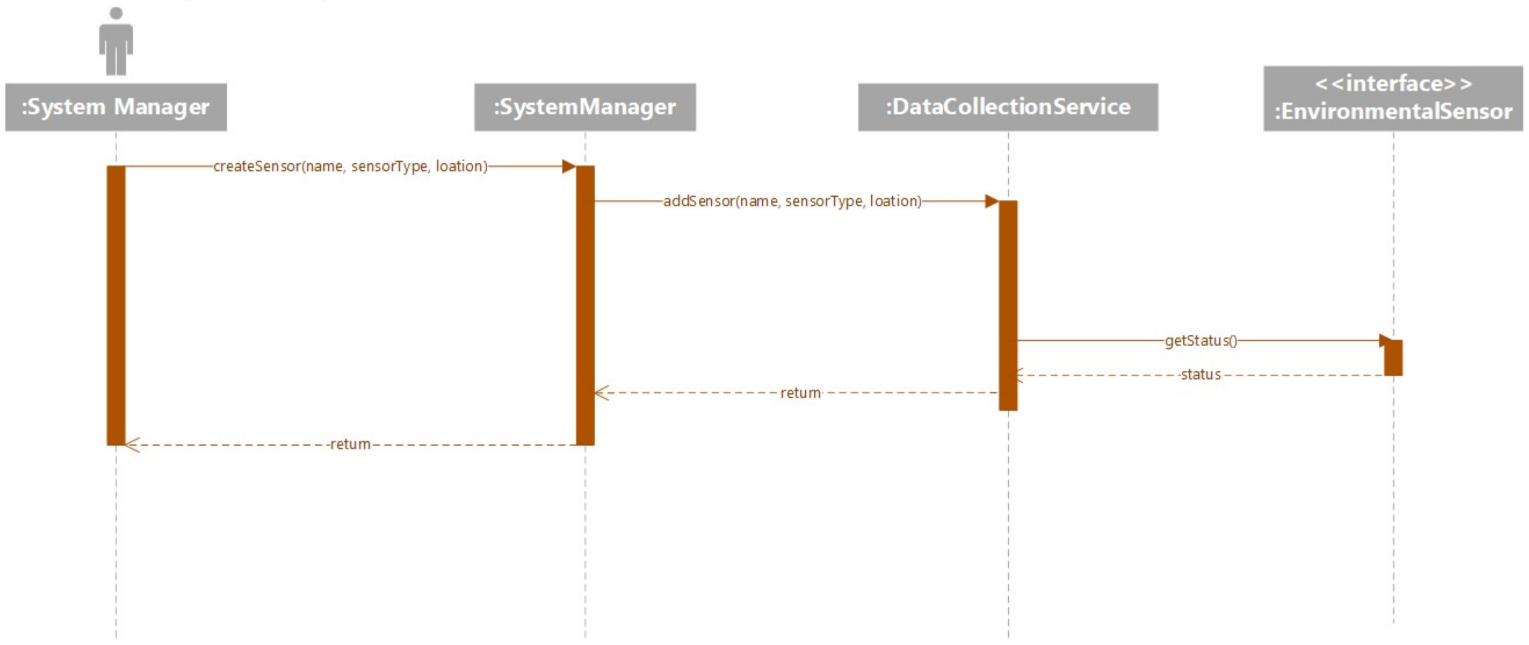
Description

This Sequence Diagram shows a user requesting a current reading from a sensor. The UI communicates with the data collection service, which gets the reading from the sensor, then saves the reading to the database before returning it to the UI to display.

Configure New Sensor

Field	Description
Name:	Create Sensor
Responsibilities:	Create a New Sensor to add to the system
Type:	System
Cross Reference:	Use Case: Configure New Sensor
Notes:	N/A
Exceptions:	• If the sensor creation fails, indicate sensor error • If the sensor is added but not operational, indicate error
Output:	None
Pre-conditions:	The User is logged in to the System as a System Manager
Post-Conditions:	A new Sensor is created and added to the list of available sensors

Sequence Diagram



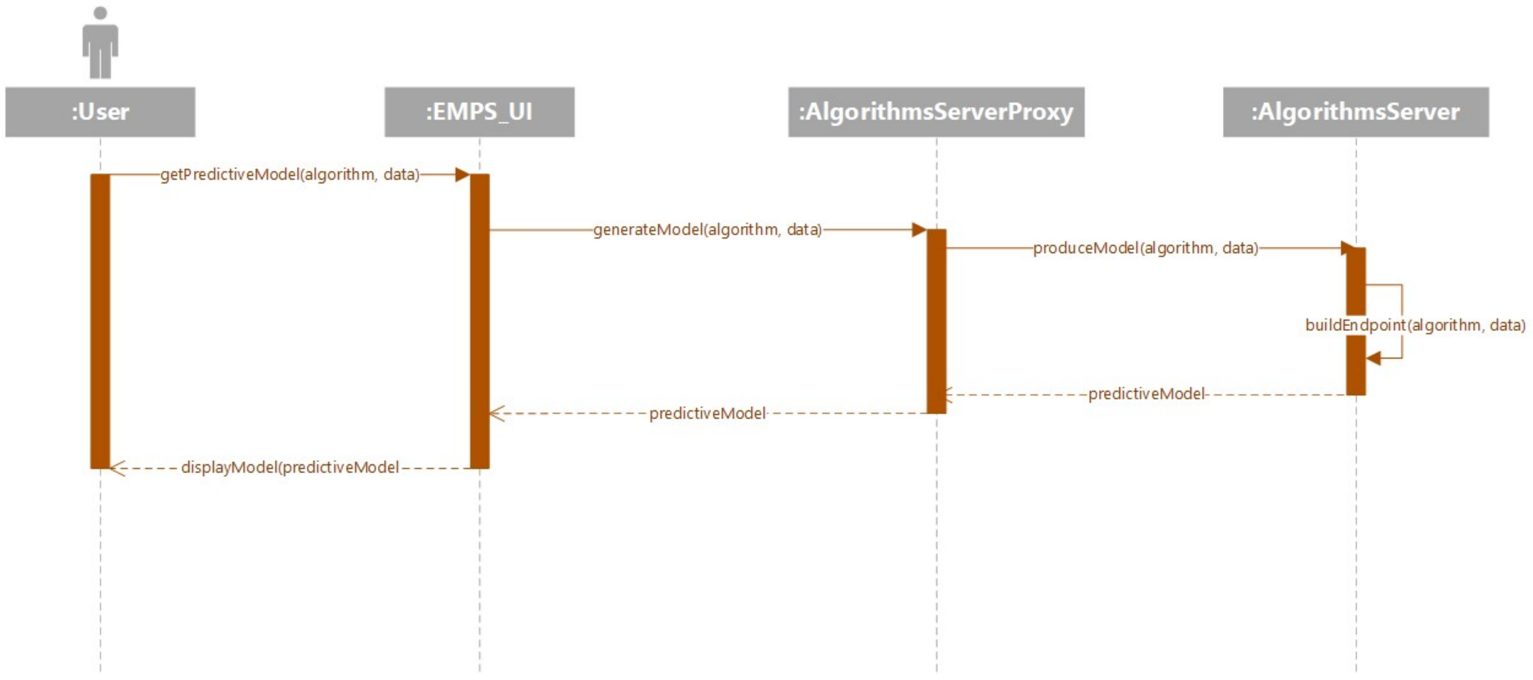
Description

This Diagram shows the process of a System Manager creating a new sensor. The Data Collection Service is used to create the new sensor and add it to the list of sensors. The sensor's status is also pinged to ensure it is operational.

Generate Predication Model

Field	Description
Name:	Get Predictive Model
Responsibilities:	Generate a Predictive Model using data
Type:	System
Cross Reference:	Use Case: Generate Prediction Model
Notes:	N/A
Exceptions:	• If connection to server fails, indicate error • If model generation fails, indicate error
Output:	Predictive Model
Pre-conditions:	The User is logged in to the System The user has collected the proper dataset needed to generate the model
Post-Conditions:	A predictive model is generated and displayed to the user

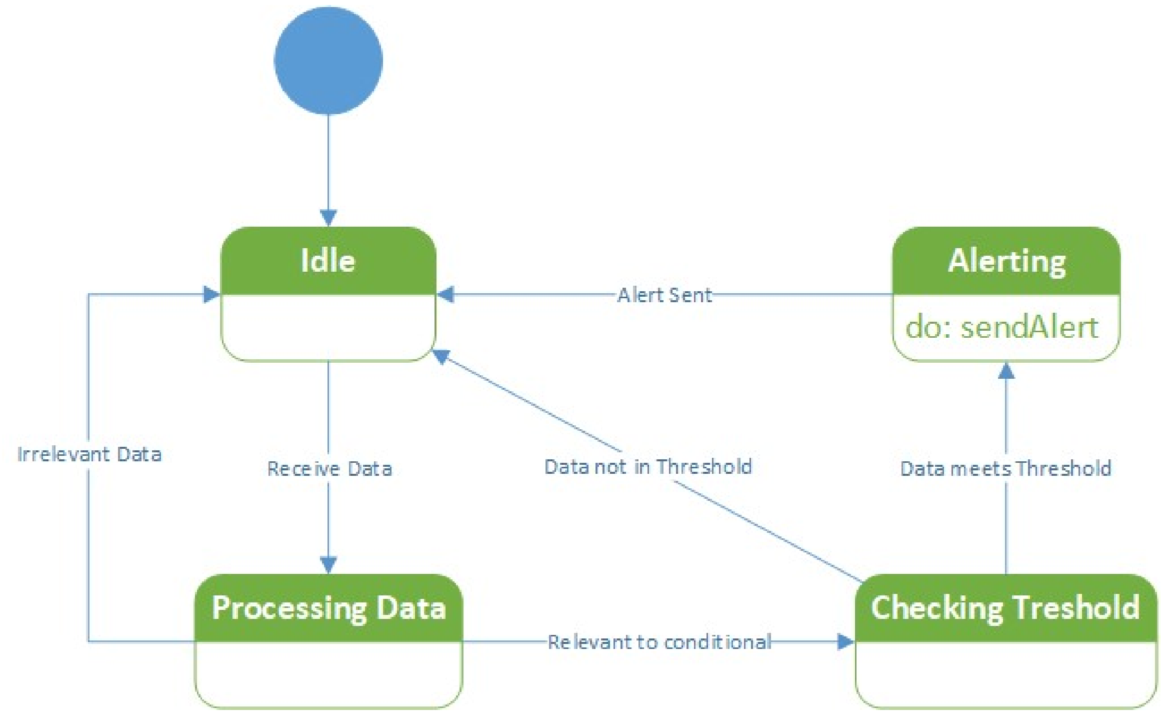
Sequence Diagram


:User
:EMPS_UI
:AlgorithmsServerProxy
:AlgorithmsServer
getPredictiveModel(algorithm, data)
generateModel(algorithm, data)
produceModel(algorithm, data)
buildEndpoint(algorithm, data)
predictiveModel
predictiveModel
displayModel(predictiveModel

Description

This Sequence Diagram shows the process of a User generating a Predictive model. This sequence assumes that the user has already queried for the data needed to generate the model. The UI makes calls to the Algorithm Server, by way of the Proxy, to generate that model before outputting it.

State Diagram



Description

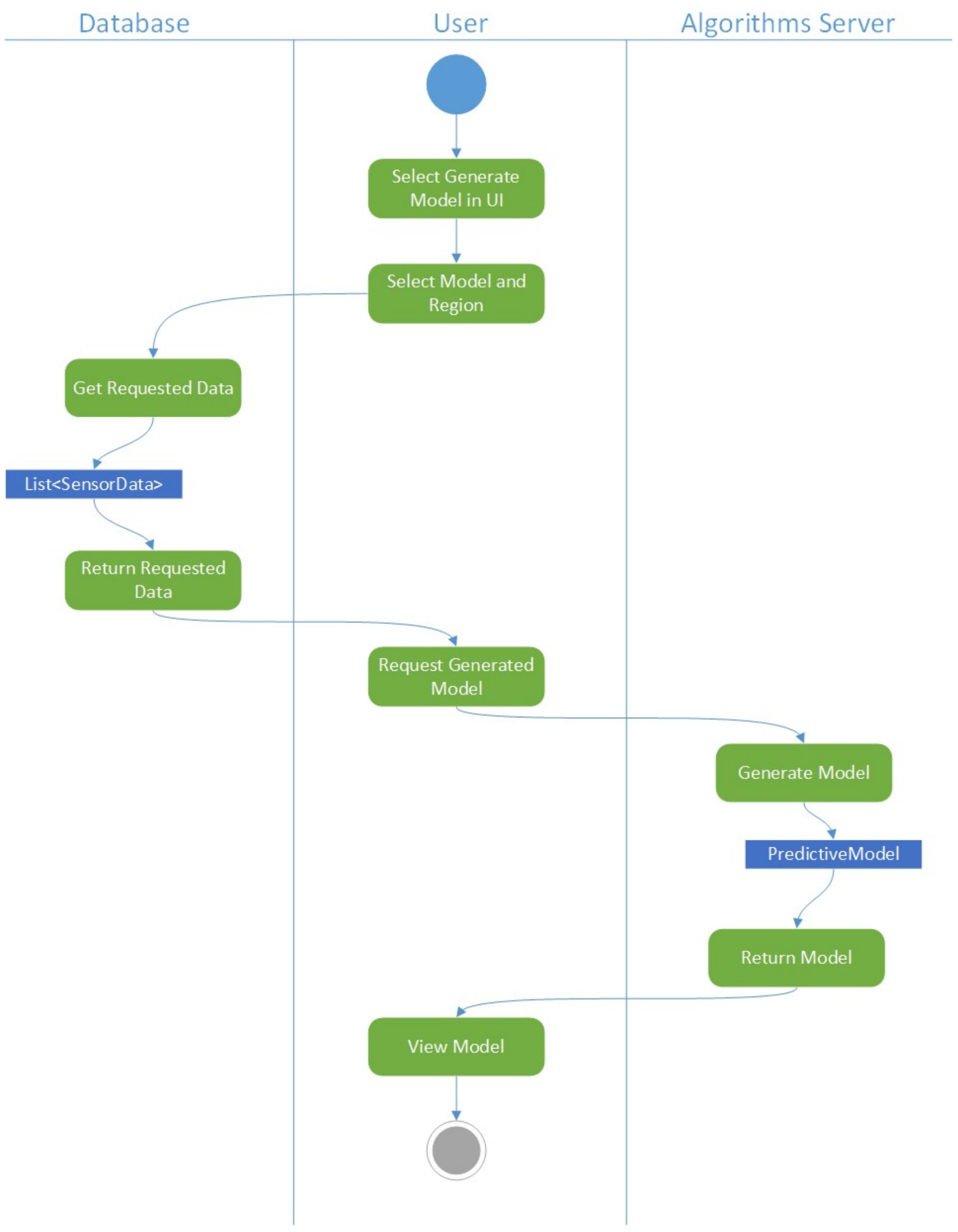
The above diagram represents the different states of the Alert Object. Initially, it starts in an idle state, waiting to receive data.

When data is received, it moves to a processing state. This state determines if the data type is relevant to the alert, that is, does the conditional of the alert pertain to this data type. For example, if an alert that is tracking the water temperature in a water quality sensor receives a piece of air quality data, that data is deemed irrelevant. If data is irrelevant, the alert returns to an idle state.

If data is relevant, it moves to a threshold check state. In this state, the Alert checks if the data falls within the specified threshold. If it is not, the Alert returns to the idle sate.

If the data does fall within the threshold, the Alert moves to the Alerting state. Here, an Alert is sent to notify the user that data has followed within the specified alert threshold. Once the alert is sent, the Alert returns to the idle state.

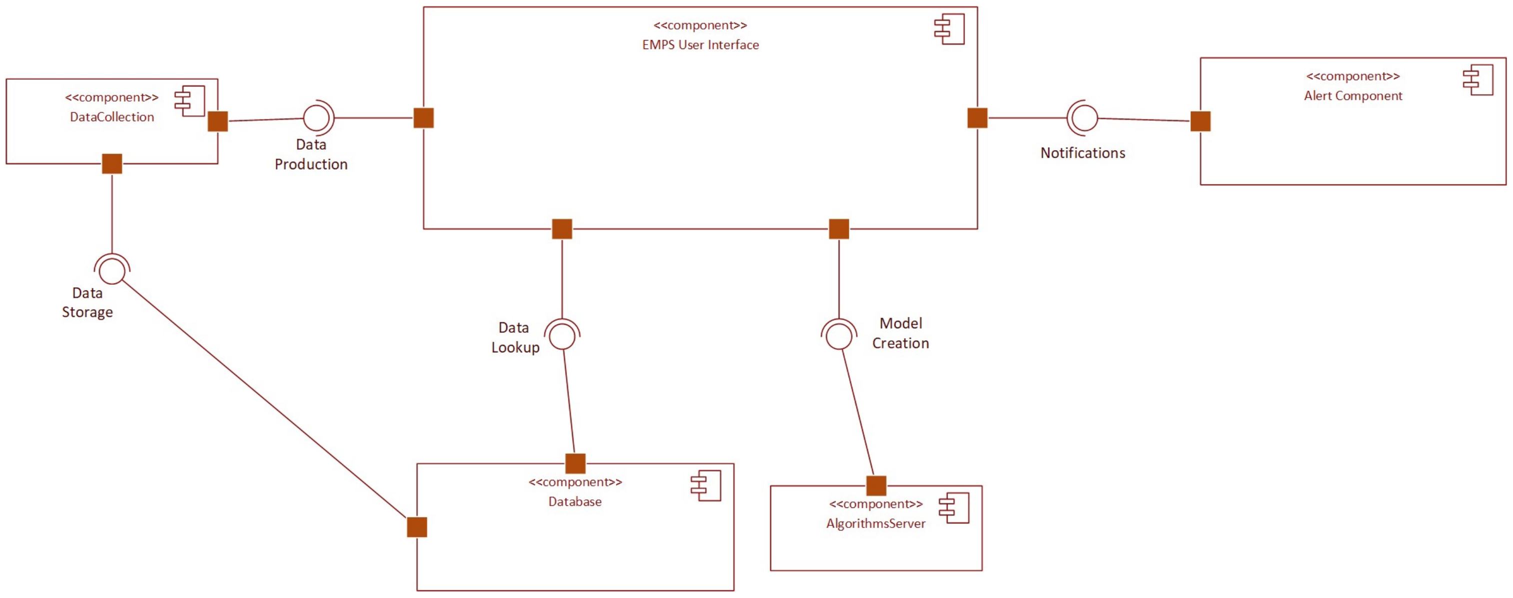
Activity Diagram



Description

This Activity diagram shows the end to end process of generating a Predictive Model using the EMPS. The Database, User, and Algorithm Server are the three roles involved in the process. First the User must collect the data needed for the Model, which is retrieved from the database. Then the user sends the data to the Algorithm Server to generate the model. Once that is complete, the User can view the generated model.

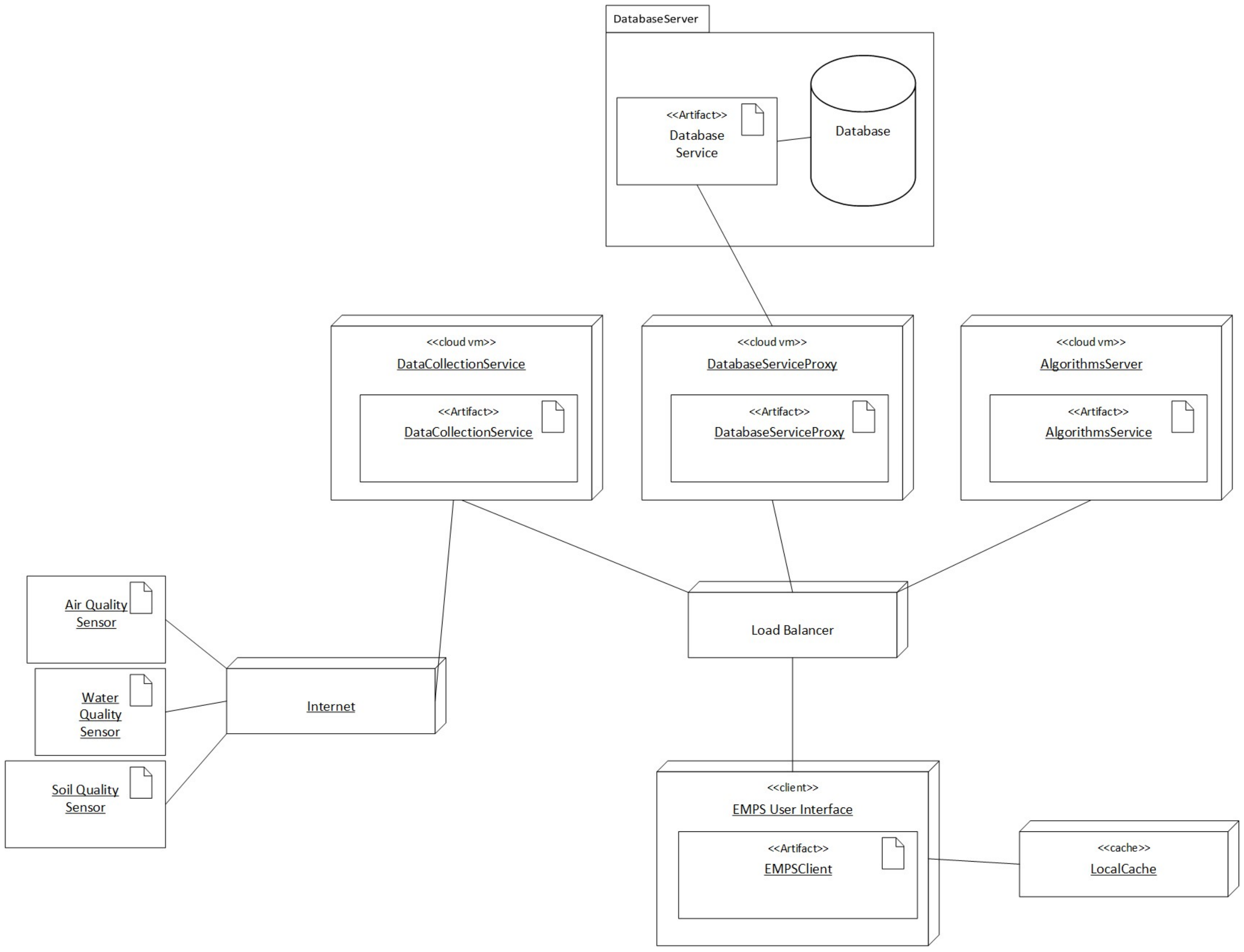
Component Diagram



Description

The EMPS system consists of 5 major components. The UI is the central component that the User uses to interact with the other components. The Data Collection component provides an interface for producing new data and has a receiving interface for storing that data. The Database component provides the UI with an interface for data lookup. The Algorithms Server component provides an interface for creating models. Finally, the Alert Component provides a notification system to the UI. These components can be swapped out with minimal damage done to the system around them, since coupling between components is low.

Deployment Diagram



Description

The EMPS consists of three main microservices deployed on cloud VMs that communicate with the EMPS Client via a Load Balancer. These services correspond to the Algorithms Service, Data Collection Service, and a Database Proxy Service. The Data Collection Service is also connected via the internet with a variety of physical sensors spread out around the world. This service communicates with these sensors to collect data. This data is retrieved by the client through the service. The Database Service Proxy communicates with a database located on an external server. The proxy makes calls to the local database service deployed alongside the database. The proxy contains generic endpoints (create, update, delete, get) that will need minimal changes if the database is migrated.

A Load Balancer is used to balance traffic to these services from the various clients. Load Balancing is important since many clients may be attempting to use the services simultaneously. Availability of the system needs to be optimizes so that a service isn't overloaded with traffic.

A Local Cache is also used by the Client. This may be especially helpful for users who are attempting to generate many different weather models for the same area. Many of these models may use the same data, so storing that data in the cache shortens retrieval and limits the amount of visits to the Database.

Skeleton Diagrams and Database Tables

Skeleton Diagrams and Database tables for the EMPS can be found in the repository under their respective folders.

Design Patterns

GRASP

One example of using GRASP patterns in this diagram is use of the Creator pattern in the Alert Manager. The alert manager contains a list of all alerts created, so it is assigned the responsibility of creating those alerts. Another example in the Alert Manager is the use of Low coupling. The Alert Manager needs to check alerts when new data is received from the Data Collection Service, but rather than couple Alerts to the Data Collection Service, we simply have the UI invoke the Alert Manager when it receives Data. Now the Data Collection Service is not coupled to the Alert Manager. Finally, as mentioned above in the description for the class diagram, Pure Fabrication and Indirection are used by creating proxy objects used to handle client side communications to the Database and Algorithms Server.

GOF

Indirection also mirrors the GOF Proxy Design Pattern, which can be used to accurately describe the Proxy objects used in the EMPS.

Microservice

As mentioned in the Deployment Diagram description, the API Gateway Design pattern was used for communication with the various microservices. Specifically, a load balancer was used.

Others

Finally, through each step of this project, I followed KISS and YAGNI principles. I didn't want to risk the project becoming overly complex by trying to shoehorn in features or design patterns that did not obviously fit the system that was being build. Keeping it simple was the overarching message I had throughout all phases of design.

Lessons Learned

The biggest lesson I learned throughout this project is that designing a software system is complex and takes lots of practice. There were points in the middle of the project where I looked back and saw mistakes or weaknesses in the design that could have been improved on, but would require reworking many subsequent diagrams – something I simply did not have time to do. Practice makes perfect, and these mistakes are certainly something I will have a better eye for in future design work. For example, late in the project, I worried that my UI object had become a "God Object" with too many responsibilities that could have been spun out into different classes. This would have reduced cohesion as well, as the UI would not be so engrained into every aspect of the system. This is something I will be wearier of in the future. I also learned that it is important to have a team building the design when possible, as opposed to one individual. Different engineers have different experiences and perspectives, mistakes I made could have been pointed out earlier in the process to allow space to correct them and shift gears. Overall, this project taught me a lot about the process of designing software and the many aspects that go into it.