Software Design Document(SDD)
for
LOCO: Enhancing Indoor IOT Network Design to support energy Saving

Abdelrahman Samir, Fady Khaled, Rana Muhammed, Samar Saeid.
Supervised by: Dr. Mohamed Elgazzar, Eng. Silvia Soliman

January, 2019

1 Introduction

1.1 Purpose of this document

This is an SDD (Software Design Document) describes architecture and system design of LOCO, a system that aims to enhance Indoor IOT Network Design to support energy Saving. It details the components of the system, the architecture used in creating it. How the data is handled and the relations and design between the data. It also explains the algorithms used throughout the system. This document explains all these aspects with the help of Class diagrams,ERD diagrams, Activity diagram and Architecture diagrams.

1.2 Scope

Loco is a web application that aims to help decrease power consumption in IoT devices and sensors. It is run by admin, and accessed by the architect, user and auditor. it allows the architect to upload a scanned image that shows the infrastructure of the place where he is willing to place his IOT devices or sensors and the system will apply the main process of classification on several models and the output that the end user will receive will be the most optimum model recommended by the system.The auditor's role is the verification part,his authentications include verifying the image with the right number of devices and right positions and will verify the result that the user will receive.

1.3 Overview

Internet of Things (IOT), is defined as group of devices that are embedded with electronics, sensors and software allowing them to be connected to the internet and receives or sends data. It is widely considered as the future of technology and electronics. There is an increase in power consumption regardless of the carbon footprint resulting from generating electricity. We should be more socially conscious of our choices, with every device we should do our best to minimize our consumption. In this project we aim to find the best indoor placement for several IOT devices by measuring power consumption depending on many features such as the average power consumption of each device, network hops per node and battery voltage of each device and using the measurements to train the system to recommend the best model with the lowest power consumption.

1. Introduction.
2. System Overview.
3. System Architecture.
4. Data Design.
5. Component Design.
6. Human Interface Design.

7. Requirement Matrix.
8. References.

1.4 Definitions and Acronyms

SDD	System Design Document
MVC	Model View Controller
IoT	Internet of Things
ANN	Artificial Neural Network
KNN	K Nearest Neighbour

2 System Overview

In Loco we have a system that enhances the positioning of IoT devices or sensors, by accurately placing them in a way where the interference between their ranges and other factors are minimized to extend the battery life of these devices and decrease their power consumption. Loco works in two phases. The first phase, the devices' readings are stored in the database and then divided into training and testing to generate a tested model. The second phase, a floor map is uploaded. Using multiple algorithms, we randomly place the devices on the map and get the best set of placement. Finally, we give these sets to the trained model to generate the best placement for the given devices.

Figure 1: System Overview

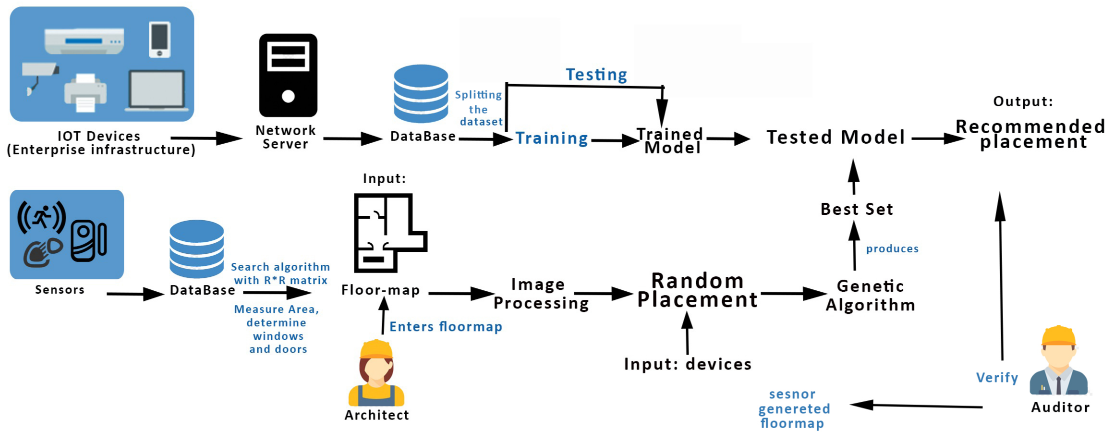
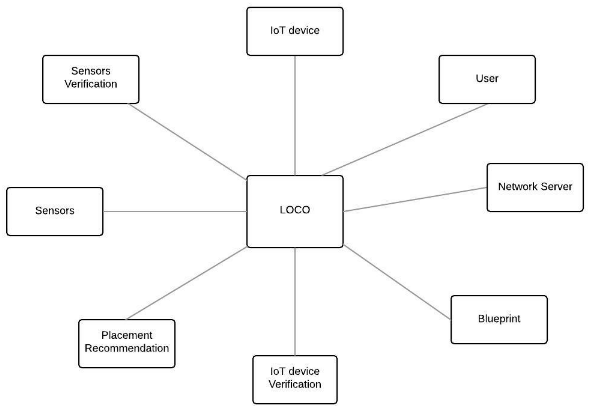


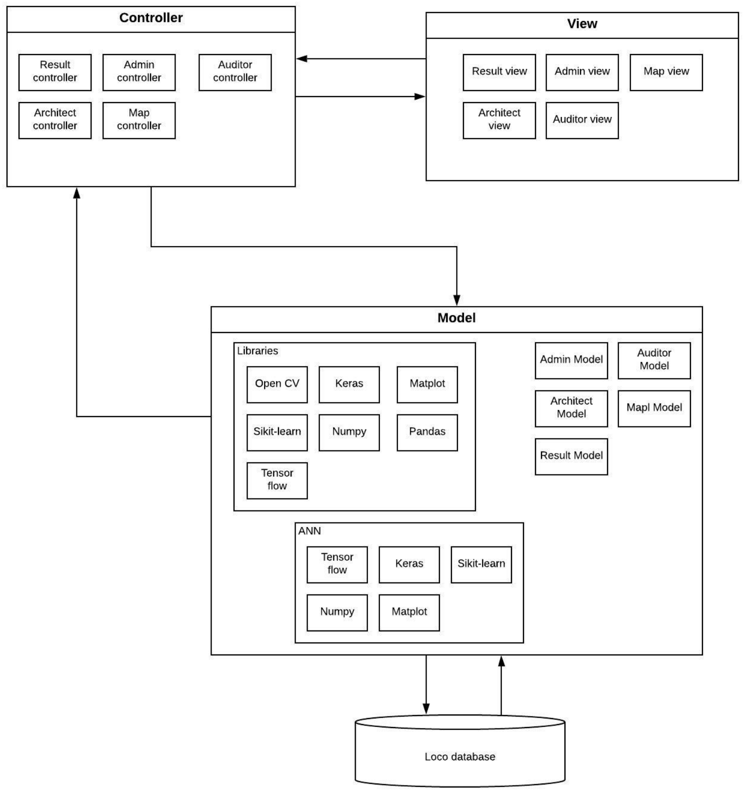
Figure 2: Block Diagram



3 System Architecture

3.1 Architectural Design

Figure 3: System Architecture



3.1.1 Model

Admin, Architect and Auditor models handle the main functionalities of the users of the system.

Map model handles the generated or uploaded floor-map of the building. it handles the number of devices and dimensions of the map.

Result model handles the generated best placement of the devices and their location on the map.

Libraries:
Numpy: library used for multidimensional arrays and matrices.

Matplot: It provides an object-oriented API for embedding plots into applications.
Scikit-learn: This is the library containing the classifier.
Keras: It is a library for dealing with deep neural network.
TensorFlow: It is used for the application of the machine learning in the neural network.

3.1.2 View

View represents the data user interface. it allows the users to view the data and manipulate it. Loco has three interfaces one is for the Admin functionalities, another one for the Architect and the last one for the auditor's functionalities.

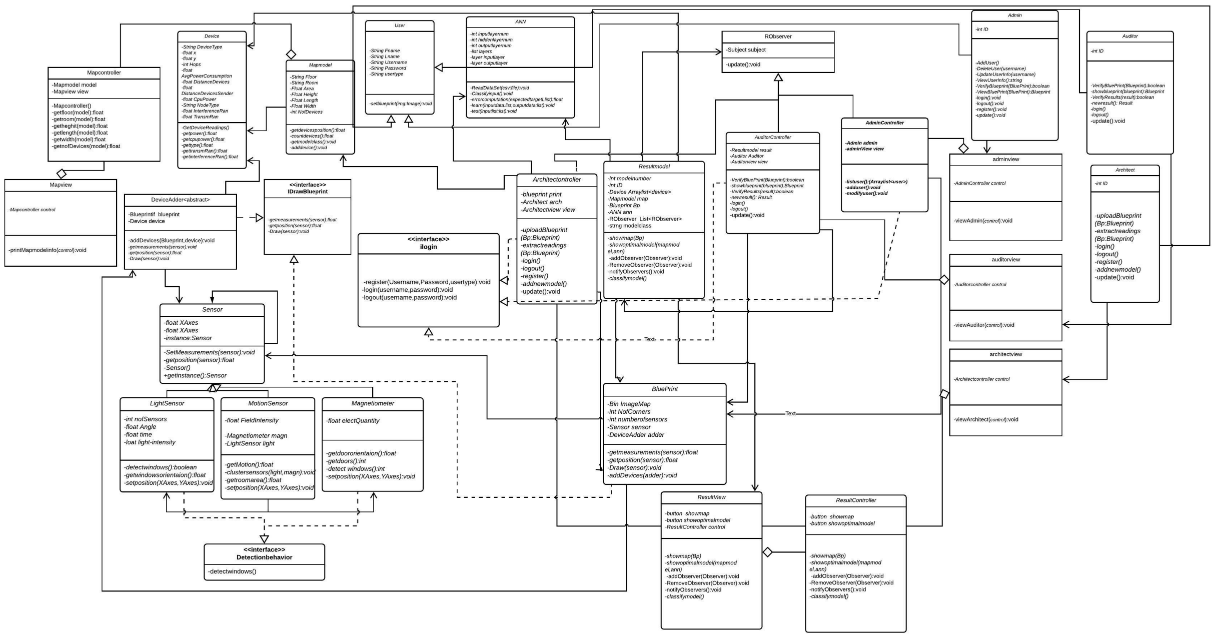
3.1.3 Controller

Controller connects the view with the model. the user sees the view and interacts with it to make commands. the model then fetches data from database and updates it to be ready for the view. Admin, Architect and Auditor controllers handles the requests made by the users.

3.2 Decomposition Description

3.2.1 Class Diagram

Figure 4: Class diagram



3.2.2 Inheritance Relationships

Class Architect inherits from class user
Class auditorController inherits from class user
Class AdminController inherits from class user
Class MotionSensor inherits from class Sensor
Class LightSensor inherits from class Sensor
Class Magnetiometer inherits from class Sensor
Class Resultmodel inherits from class RObserver
Class auditorController inherits from class RObserver
Class AdminController inherits from class RObserver

3.2.3

- **Class name:**User, Concrete.
- **List of super classes:** None.
- **List of sub classes:** Architect, Admin Auditor.
- **Purpose :** To define the basic attributes and methods of user such as his username and password and the login method and register method.
- **Collaboration:** To achieve its purpose User class should perform login process using username and password to ensure security. It will need to have the classes Architect, AdminController and AuditorController to inherit from it.
- **Attributes:** Fname,Lname,username,usertype and password will be string.
- **Operation:**
  setblueprint() it's argument: username(string),password(string),blueprint(jpg)

3.2.4

- **Class name:**ANN , Concrete.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the artificial neural network such as number of input layers and number of output layers and and number of hidden layers and a list of layers and read-dataset,classifyinput,error computation,learn and test methods.
- **Collaboration:** To achieve its purpose Device's class readings are passed by the Architect class to Ann for further analysis as the mentioned methods. It will need to have inheritance relationship with Architect class.

- **Operation:**
  ReadDataset() it's argument: Dataset(csv), ClassifyInput() it's argument: model(Resultmodel), errorComputation() it's argument: expectedTarget(List), Learn() it's argument: inputdata(list),outputdata(list) Test() it's argument: inputlist(list)

3.2.5

- **Class name:**Device , Concrete.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the positioned devices such as Device Type and Devices' positions and number of hops and the average power consumption of all devices and Distance between devices and distance between every device and the sender device (router) and cpu power consumption and node type and interference range and transmission range and getDeviceReadings, getpower, getcpupower,gettype,gettransRan and getinterferenceRan methods.
- **Collaboration:** To achieve its purpose Device's class readings are retrieved by the Resultmodel class to be verified by the auditorController for further analysis as the mentioned methods. It will need to be aggregated by Resultmodel class and inherited from Mapmodel class.
- **Attributes:** devicetype will be string, and position,hops and avgpowerconcumption and distancebetweendevices and cpupower and nodetype and interference range and transmission range would be integers.
- **Operation:**
  GetDeviceReadings () it's argument:simulationreadings(csv), getpower () it's argument: simulationreadings(csv), getcpupower () it's argument: simulationreadings(csv), gettype () it's argument simulationreadings(csv), gettransmRan () it's argument simulationreadings(csv), getinterferenceRan () it's argument simulationreadings(csv)

3.2.6

- **Class name:**Mapmodel, Concrete.
- **List of super classes:** None.
- **List of sub classes:** None.

- **Purpose :** To define the basic attributes and methods of map model such as which floor and which room and area of the room and height of the room and length of room and width of the room and number of attached devices and getdevicesposition, countdevices, getmodelclass and adddevice methods.
- **Collaboration:** To achieve its purpose Map model class should specify all the attached devices in the model and their readings to make a model with a blueprint and attached devices. It will need to have inheritance relationship to Device class and needs to be inherited from Architect class and it also needs to be aggregated from Resultmodel class.
- **Attributes:** floor,room,area,height,length,width and numberofdevices will be integers.
- **Operation:**
  getdevicesposition () it's argument: Device(device), countdevices () it's argument: Device(device), getmodelclass () it's argument: simpulation-readings(int), adddevice () it's argument:None

3.2.7

- **Class name:** Architect, Concrete.
- **List of super classes:** User.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the architect such ID and login, logout, register,addnewmodel, update,extractReadings and uploadblueprint methods.
- **Collaboration:** To achieve its purpose Architect class it should login first to be responsible of managing the blueprints and the databases. It will need to have and association to class User and Architectview class.
- **Attributes:** ID will be integer.
- **Operation:**
  ExtractReadiings () it's argument: Bp(blueprint), login () it's argument: : None, logout () it's argument: : None, register () it's argument: : None, addnewmodel () it's argument: None Update () it's argument: None

3.2.8

- **Class name:** ArchitectView, Concrete.
- **List of super classes:** None.

- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the architect view such an object from class Architect and viewArchitect method.
- **Collaboration:** To achieve its purpose ViewArchitect class it should login first to be responsible of managing the blueprints and the databases and visible on the system. It will need to be associated from class Architect and to have an aggregation with ArchiectController class.
- **Attributes:** control will be an object from class Architect.
- **Operation:**
  ViewArchitect () it's argument:control,

3.2.9

- **Class name:** ArchitectController, Concrete.
- **List of super classes:** RObserver.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the architect-Controller such as and object from blueprint,Architect and ArchitectView classes and UploadBluePrint, extractReadings,login, logout, register and addnewmodel and update methods.
- **Collaboration:** To achieve its purpose ArchitectController class it should login first to be responsible of managing the blueprints ,databases and users. It will need to have and inheritance to Mapmodel,Ann and blueprint classes and association relationship to RObserver class and needs to depend on the login interface
- **Attributes:** print,arch and view will be objects from Blueprint,Architect and Architectview.
- **Operation:**
  UploadBlueprint () it's argument: bp(Blueprint), extractReadings () it's argument: bp(Blueprint), login () it's argument: : None, logout () it's argument: : None, register () it's argument: : None, addnewmodel () it's argument: None Update () it's argument: None

3.2.10

- **Class name:** AdminController, Concrete.
- **List of super classes:** RObserver.
- **List of sub classes:** None.

- **Purpose :** To define the basic attributes and methods of the Admin controller such objects from Admin and Adminview classes and AddUser, listuser and modifyuser methods.
- **Collaboration:** To achieve its purpose AdminController class it should login first to be responsible of managing the users and the blueprint and needs to depend on the login interface class and an aggregation relationship to adminview class and an association to Blueprint class and finally an inheritance to RObserver class.
- **Attributes:** admin,view will be objects from Admin and AdminView classes.
- **Operation:**
  AddUser () it's argument: an arraylist of users, addUser () it's argument: None, modifyUser () it's argument: None

3.2.11

- **Class name:** Admin, Concrete.
- **List of super classes:** User.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the Admin such ID and AddUser,deleteuser,updateUserinfo, viewuserifo,verifyblueprint,viewblueprint,login,logout,registe
  and update methods.
- **Collaboration:** To achieve its purpose Admin class it should login first to be responsible of managing the users and the blueprint and needs to an inheritance relationship to User class and an association to Adminview class .
- **Attributes:** ID will be integer.
- **Operation:**
  AddUser () it's argument: an arraylist of users, deleteUser () it's argument: (username), updateuserinfo () it's argument: (username), viewuserinfo () it's argument: (username), verifyBlueprint () it's argument: (blueprint), viewBlueprint () it's argument: (blueprint), login () it's argument: None, register () it's argument: None, logout () it's argument: None, update () it's argument: None

3.2.12

- **Class name:** AdminView, Concrete.
- **List of super classes:** None.

- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the Admin such control which is an object from AdminController and ViewAdmin method.
- **Collaboration:** To achieve its purpose AdminView class it should login first to be responsible of managing the users and the blueprint and needs to be associated from Admin class and an aggregation relationship to AdminController class.
- **Attributes:** control will be an object from Admincontroller class.
- **Operation:**
  viewAdmin () it's argument: (control),

3.2.13

- **Class name:** auditorController, Concrete.
- **List of super classes:** RObserver.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the auditor Controller such as result,Auditor and view and showblueprint, Verify-Blueprint, VerifyResults,newresult,login, logout,register and update methods.
- **Collaboration:** To achieve its purpose auditorController class it should login first to be responsible of managing the result and the blueprint. It will need to have and association to class RObserver and an inheritance relationship to Resultmodel,Blueprint classes and needs to depend on iLogin interface .
- **Attributes:** Result,Auditor and view will be Objects from Resultmodel,Auditor and view classes.
- **Operation:**
  login () it's argument: : None, logout () it's argument: : None, register () it's argument: : None, VerifyBlueprint () it's argument: Blueprint(blueprint), showBluePrint () it's argument: Blueprint(blueprint), VerifyResults () it's argument: Resultmodel(result), newResult () it's argument: Resultmodel(result)

3.2.14

- **Class name:** auditor, Concrete.
- **List of super classes:** User.
- **List of sub classes:** None.

- **Purpose :** To define the basic attributes and methods of the auditor Controller such as ID and showresult, VerifyBlueprint, VerifyResult,login, logout and register and update and newResult methods.
- **Collaboration:** To achieve its purpose auditor class it should login first to be responsible of managing the result and the blueprint. It will need to have and association to class User and an inheritance relationship to AudtitorView class.
- **Attributes:** ID will be integer.
- **Operation:**
  login () it's argument: : None, logout () it's argument: : None, register () it's argument: : None, VerifyBlueprint () it's argument: Blueprint(blueprint), showBluePrint () it's argument: Blueprint(blueprint), VerifyResults () it's argument: Resultmodel(result), newResult () it's argument: Resultmodel(result)

3.2.15

- **Class name:** auditorview, Concrete.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the auditor Controller such as control and viewAuditor method.
- **Collaboration:** To achieve its purpose auditor class it should login first to be responsible of managing the result and the blueprint. It will need to be associated from class Auditor and an aggregation relationship to AudtitorController class.
- **Attributes:** control will be an object from class AuditorController.
- **Operation:**
  ViewAuditor () it's argument: : control,

3.2.16

- **Class name:** Blueprint, Concrete.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the Blueprint such as ImageMap and number of Corners and getuserinfoand getmeasurements methods.

- **Collaboration:** To achieve its purpose Blueprint class should get an image and a number of corners . It will need to have and inheritence from class Admincontroller,Architect and auditorController.
- **Attributes:** ImageMap as a BIN, NofCornerswill be integer.
- **Operation:**
getmeasurements () it's argument:None, getuserinfo () it's argument:classtype

3.2.17

- **Class name:** Sensor, Concrete.
- **List of super classes:** None.
- **List of sub classes:** MotionSensor, Magnetiometer, LightSensor.
- **Purpose :** To define the basic attributes and methods of the Sensor of the sensors such as xAxes of every position and YAxes of every position and SetMeasurements and getposition methods.
- **Collaboration:** To achieve its purpose Sensor class It will need to associated from class MotionSensor, Magnetiometer, LightSensor.
- **Attributes:** XAxes, XAxes will be floats.
- **Operation:**
SetMeasurements () it's argument: : Sensor(sensor), getposition () it's argument:Sensor(sensor)

3.2.18

- **Class name:** LightSensor, Concrete.
- **List of super classes:** Sensor.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the LightSensor such as number of sensors and angle and time and light intensity and X axes and Y axes and detectwindows, getwindowsorientaion, and setposition methods.
- **Collaboration:** To achieve its purpose LightSensor class It will need to have and association to class Sensor
- **Attributes:** nofSensors will be integer, Angle,time,light-intensity xAxes and yAxes will be floats.
- **Operation:**
detectwindows () it's argument:None, getwindowsorientaion () it's argument:None, setposition () it's argument: : xAxes(float),yAxes(float)

3.2.19

- **Class name:** Magnetiometer, Concrete.
- **List of super classes:** Sensor.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the Magnetiometer such electric quantity and X axes and Y axes and getdoororientaion, getdoors, and setposition methods.
- **Collaboration:** To achieve its purpose Magnetiometer class It will need to have and association to class Sensor
- **Attributes:** electQuantity,xAxes and yAxes will be floats.
- **Operation:**
  getdoororientaion () it's argument:None, getdoors () it's argument:None, setposition () it's argument: : xAxes(float),yAxes(float)

3.2.20

- **Class name:** MotionSensor, Concrete.
- **List of super classes:** Sensor.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the MotionSensor such as field intensity and Magnetiometer and Light sensor and X axes and Y axes and getMotion, clustersensors, getroomarea, and setposition methods.
- **Collaboration:** To achieve its purpose MotionSensor class It will need to have and association to class Sensor also needs to aggregate both classes Lightsensor and Magnetiometer.
- **Attributes:** fieldIntensity, xAxes and yAxes will be floats and mag would be an object from class Magnetiometer and light would be an object from class LightSensor.
- **Operation:**
  getMotion () it's argument:light(lightSensor),mag(Magnetiometer), clustersensors () it's argument:Sensors, getroomarea () it's argument: None, setposition () it's argument: : xAxes(float),yAxes(float)

3.2.21

- **Class name:** Resultmodel, Concrete.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the Resultmodel such as model number and ID and an array of devices and an object from mapmodel and bp and ann and list of observers and model class and showmap, showoptimalmodel, addObserver,removeobserver,notifyobservers and classifymodels methods.
- **Collaboration:** To achieve its purpose Resultmodel class It will need to associate RObserver, Device ,ResultView,ANN and Blueprint classes.
- **Attributes:** modelnumber,ID will be an integer, devices array would be an array and mapmodel which is an object from mapModel and bp which is an object from blueprint and ann which is an object from class ANN and a list of RObserver objects and classmodel would be an integer.
- **Operation:**
  showmap () it's argument:(bp), showoptimalmodel () it's argument: (Mapmodel,ann), addObsevber() it's argument: (observer), removeObverber() it's argument: (observer), notifyObserver() it's argument:None, classifymodels() it's argument:None

3.2.22

- **Class name:** Resultview, Concrete.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the Resultview such showmap,showoptimalmodel and control and showmap, showoptimalmodel, addObserver,removeobserver,notifyobservers and classifymodels methods.
- **Collaboration:** To achieve its purpose Resultview class It will need to be associated from Resultmodel class and and aggregation relationship to Resultcontroller class.
- **Attributes:** showmap,showoptimalmodel will be buttons and control would be an object from class Resultcontroller.

- **Operation:**
  showmap () it's argument:(bp), showoptimalmodel () it's argument: (Mapmodel,ann), addObsevber() it's argument: (observer), removeObverber() it's argument: (observer), notifyObserver() it's argument:None, classifymodels() it's argument:None

3.2.23

- **Class name:** Resultcontroller, Concrete.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the Resultview such showmap,showoptimalmodel and showmap, showoptimalmodel, addObserver,removeobserver,notifyobservers and classifymodels methods.
- **Collaboration:** To achieve its purpose Resultcontroller class an aggregation relationship to Resultview class is needed.
- **Attributes:** showmap,showoptimalmodel will be buttons and control would be an object from class Resultcontroller.
- **Operation:**
  showmap () it's argument:(bp), showoptimalmodel () it's argument: (Mapmodel,ann), addObsevber() it's argument: (observer), removeObverber() it's argument: (observer), notifyObserver() it's argument:None, classifymodels() it's argument:None

3.2.24

- **Class name:** ilogin, interface.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes and methods of the ilogin such as Login, Logout and register methods.
- **Collaboration:** To achieve its purpose ilogin class It will need to be depended from Architect,auditorController and AdminController classes.
- **Attributes:** None.
- **Operation:**
  Login () it's argument:username(String),password(String), Logout () it's argument: username(String),password(String), register () it's argument: username(String),password(String),usertype(int)

3.2.25

- **Class name:** IDrawBlueprint, interface.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic methods of the IDrawBlueprint such as getmeasurements, getposition and Draw methods.
- **Collaboration:** To achieve its purpose IDrawBlueprint class It will need to be depended from DeviceAdder and BluePrint classes.
- **Attributes:** None.
- **Operation:**
  getmeasurements () it's argument:sensor(Sensor), getposition () it's argument: sensor(Sensor), Draw () it's argument: sensor(Sensor)

3.2.26

- **Class name:** Detectionbehavior, interface.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic method of the Detectionbehaviorsuch as detectwindows method.
- **Collaboration:** To achieve its purpose Detectionbehaviorclass It will need to be depended from LightSensor and Magnetiometerclasses.
- **Attributes:** None.
- **Operation:**
  detectwindows () it's argument: None

3.2.27

- **Class name:** DeviceAdder, Abstract.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic attributes methods of the DeviceAdder such as blueprint, device and addDevices, getmeasurements, getposition, Draw methods.

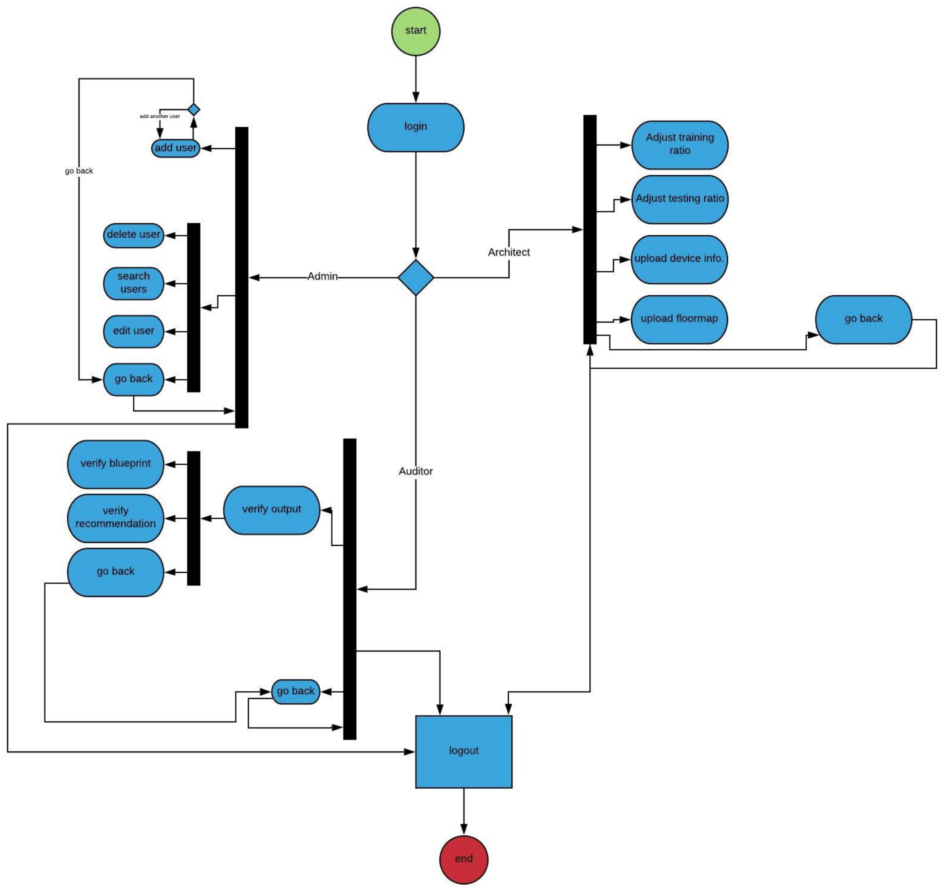
- **Collaboration:** To achieve its purpose DeviceAdder class It will need to depended from IDrawBlueprint interface and associated with Device classe.
- **Attributes:** None.
- **Operation:**
getmeasurements () it's argument:sensor(Sensor), addDevices () it's argument: Blueprint(blueprint),Device(device), getposition () it's argument: sensor(Sensor), Draw () it's argument: sensor(Sensor)

3.2.28

- **Class name:** RObserver, Abstract.
- **List of super classes:** None.
- **List of sub classes:** None.
- **Purpose :** To define the basic attribute and method of the RObserver such as subject and update method.
- **Collaboration:** To achieve its purpose RObserver class It will need to be depended from auditorController, AdminController and Architectclasses.
- **Attributes:** Resultmodel will be an object from Resultmodel.
- **Operation:**
update () it's argument: None

3.2.29 Activity Diagram

Figure 5: Activity diagram



3.2.30 Sequence Diagram

Figure 6: System sequence diagram

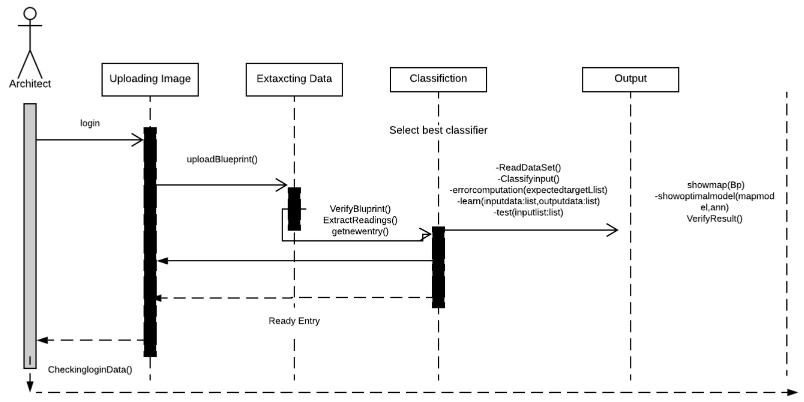


Figure 7: Generating blueprint sequence diagram

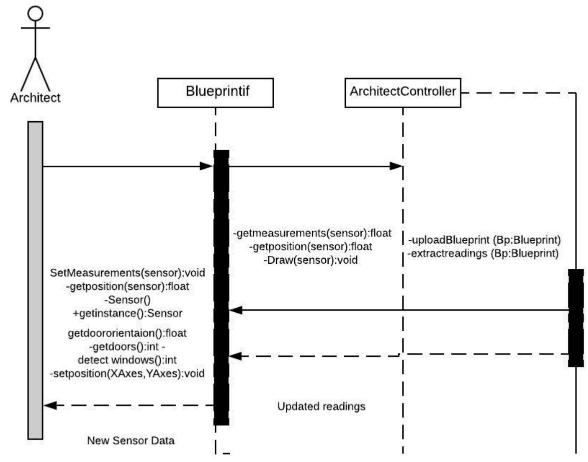
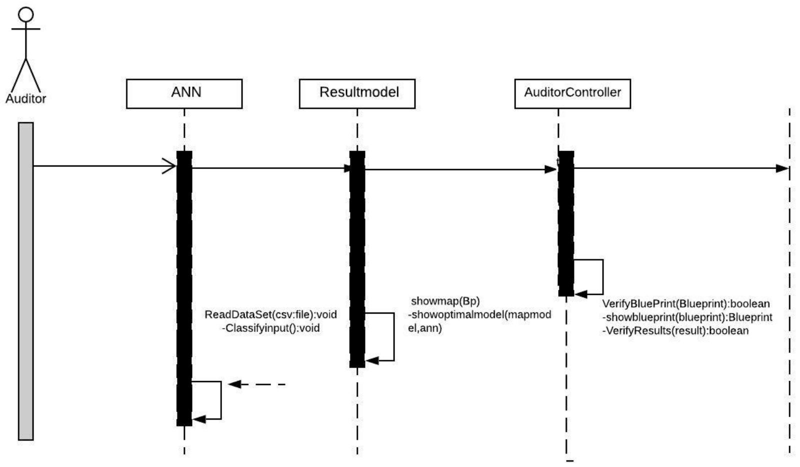

Architect
Blueprintif
ArchitectController
-getmeasurements(sensor):float
-getposition(sensor):float
-Draw(sensor):void
-uploadBlueprint (Bp:Blueprint)
-extractreadings (Bp:Blueprint)
SetMeasurements(sensor):void
-getposition(sensor):float
-Sensor()
+getinstance():Sensor
getdoororientaion():float
-getdoors():int -
detect windows():int
-setposition(XAxes,YAxes):void
Updated readings
New Sensor Data

Figure 8: Auditor verifying results sequence diagram

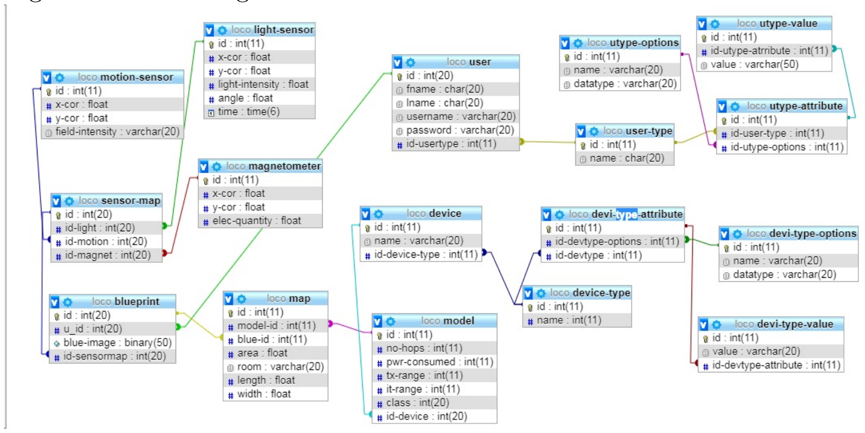

Auditor
ANN
Resultmodel
AuditorController
ReadDataSet(csv:file):void
-Classifyinput():void
showmap(Bp)
-showoptimalmodel(mapmod
el,ann)
VerifyBluePrint(Blueprint):boolean
-showblueprint(blueprint):Blueprint
-VerifyResults(result):boolean

3.3 Design Rationale

4 Data Design

4.1 Data Description

Figure 18: ERD diagram



user: this table has user id, fname, lname, username, password id-usertype as we have three different types of user (Admin, Architect and Auditor).
user-type: it has id and name to differentiate between user types.
utype-attributes:it has id, id-user-type and utype-options allowing us to know the different options of each user type.

utype-value: it has id, value and foreign keys id-user id-utype-attribute. value gives us the value of each user type attribute.
utype-options:it has id, option name and the datatype assigned to each user type option.

device:this table has the basic information of the device. it has id, name and foreign key id-device-type as there are many types of devices.
device-type: it has id and name to differentiate between device types.
devi-type-attribute:it has id, foreign keys id-devtype-options and id-devtype, allowing us to know the different options of each device type.
devi-type-value: it has id, value and foreign keys id-device id-devtype-attribute. value gives us the value of each device type attribute.
devi-type-options:it has id, option name and the datatype assigned to each device type option.

magnetometer:it has id, x-cor, y cor and elec-quantity. These readings allow us to store each reading of each installed magnetometer and know the orientation of each door.
light-sensor:it has id, x-cor, y-cor to get its location. it has light intensity, angle and time which are the readings that help us know the location of windows and doors.
motion-sensor:it takes id, x-cor, y-cor to know the location of each sensor. it also has intensity to record the variation in its readings.
sensor-map: This table is responsible for storing all the obtained information from the sensors and obtain a map from them. it has id, and foreign keys: id-light, id-motion and id-magnet.

model: This table has the data of each model of devices inserted for training and testing. It has id, no-hops, pwr-consumed, class, tx-range, it-range and foreign key id-device to know which devices are in this model.
blueprint:it has id, id-sensormap of the sensors generated map, blue-image of the blueprint/floor-map and u-id to tell which user inserted this blueprint.
map:This table stores the system recommended placement of the devices. It has id, area, room, length, width. Foreign keys model-id of the model of devices and blue-id of the blueprint.

4.2 Data Dictionary

Reliability is achieved by using singleton design pattern.

Maintainability is achieved by using Entity Attribute Value Model(EAV) in user classes (User, User-Type, Utype-Options, Utype-Attributes, Utype-Value), this allows the addition of more types of hosts in the future. EAV is also used in device classes (Device, devi-type-atrribute, devi-type-option, devi-type value).

Portability is achieved by using Model View Cotroller (MVC) design pattern.

5 Component Design

5.1 Pre-processing

In this phase we prepare the models we are going to train our model with, this happens through applying a reasonable number of experiments of IOT devices in different placements and using the results to build a training and testing dataset for the model that will be used in classification later on.
And also it includes the preparation of the input image by converting it to greyscale image using cv2Color and determining the x and y coordinates of each node to be ready for feature extraction.

5.2 Feature Extraction

In this phase our features are already set to be used as we have all the required features in the previously prepared models so all we need to do is arrange them in the proper way and labeling them according to our threshold to make the dataset ready for training and testing usage.

5.3 Classification

5.3.1 ANN

In this phase we use an ”Artificial neural network algorithm for classification and we also try another classifiers (knn and Svm) and compare the results based on accuracy,recall and precision.

The ”Artificial neural network” classifier is decomposed of:

A neuron with label J receives an input

$$p_j(t)$$

It receives it from predecessor neurons consists of the following components:

An activation :

$$a_j(t)$$

the neuron’s state, depending on a discrete time parameter.

Possibly a threshold :

$$\theta_j \theta_j$$

which stays fixed unless changed by a learning function.

An activation function:

$$a_j(t+1) = f(a_j(t), p_j(t), \theta_j)$$

And an output function:

$$o_j(t) = f_{out}(a_j(t))$$

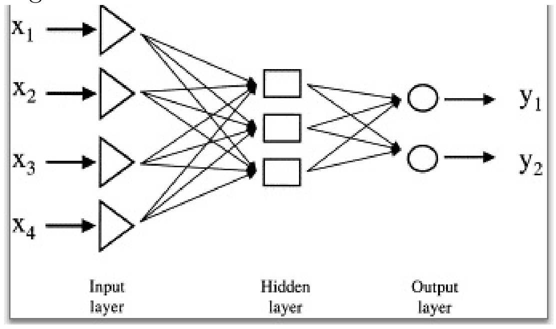
The propagation function computes the input to the neuron j from the outputs of predecessor neurons and typically has the form:

$$p_j(t) = \sum_i o_i(t) w_{ij}$$

When a bias value is added with the function, the above form changes to the following:

$$p_j(t) = \sum_i o_i(t) w_{ij} + w_{0j}$$

Figure 9: feed forward network



Learning rule:
The learning rule is a rule or an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce a favored output. This learning process typically amounts to modifying the weights and thresholds of the variables within the network.

Backpropagation:

A DNN can be discriminatively trained with the standard backpropagation algorithm. Backpropagation is a method to calculate the gradient of the loss function (produces the cost associated with a given state) with respect to the weights in an ANN.

The weight updates of backpropagation can be done via stochastic gradient descent using the following equation:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}} + \xi(t)$$

5.3.2 KNN

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming distance). In the context of gene expression microarray data, for example, k-NN has also been employed with correlation coefficients such as Pearson and Spearman.Often, the classification accuracy of k-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighbourhood components analysis.

The 1-nearest neighbor classifier:
The most intuitive nearest neighbour type classifier is the one nearest neighbour classifier that assigns a point x to the class of its closest neighbour in the feature space, that is:

$$C_n^{1nn}(x) = Y_{(1)} C_n^{1nn}(x) = Y_{(1)}$$

Figure 10: KNN flow chart

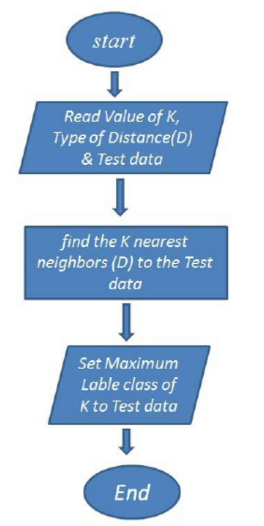
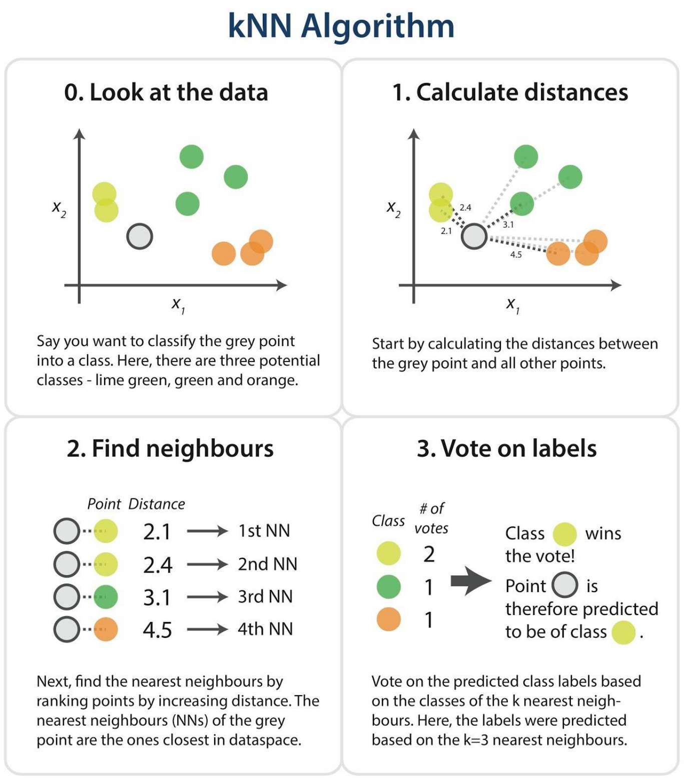

start
Read Value of K, Type of Distance(D) & Test data
find the K nearest neighbors (D) to the Test data
Set Maximum Lable class of K to Test data
End

Figure 11: KNN Algorithm


kNN Algorithm
0. Look at the data
x_2
x_1
Say you want to classify the grey point into a class. Here, there are three potential classes - lime green, green and orange.
1. Calculate distances
2.4
2.1
3.1
4.5
Start by calculating the distances between the grey point and all other points.
2. Find neighbours
Point Distance
2.1 → 1st NN
2.4 → 2nd NN
3.1 → 3rd NN
4.5 → 4th NN
Next, find the nearest neighbours by ranking points by increasing distance. The nearest neighbours (NNs) of the grey point are the ones closest in dataspace.
3. Vote on labels
Class
# of votes
2
1
1
Class wins the vote!
Point is therefore predicted to be of class .
Vote on the predicted class labels based on the classes of the k nearest neighbours. Here, the labels were predicted based on the k=3 nearest neighbours.

As the size of training data set approaches infinity, the one nearest neighbour classifier guarantees an error rate of no worse than twice the Bayes error rate (the minimum achievable error rate given the distribution of the data).

The k-nearest neighbour classifier can be viewed as assigning the k nearest neighbours a weight

$$1/k1/k$$

and all others 0 weight. This can be generalised to weighted nearest neighbour classifiers. That is, where the ith nearest neighbour is assigned a weight

$$w_{ni}w_{ni}, with\sum_{i=1}^{n} w_{ni} = 1\sum_{i=1}^{n} w_{ni} = 1.$$

An analogous result on the strong consistency of weighted nearest neighbour classifiers also holds.

$$C_n^{wnn}C_n^{wnn}\{w_{ni}\}_{i=1}^{n}\{w_{ni}\}_{i=1}^{n}.$$

Subject to regularity conditions on the class distributions the excess risk has the following asymptotic expansion

$$\mathcal{R}_{\mathcal{R}}(C_n^{wnn}) - \mathcal{R}_{\mathcal{R}}(C^{Bayes}) = \left(B_1 s_n^2 + B_2 t_n^2\right)\{1 + o(1)\}, \mathcal{R}_{\mathcal{R}}(C_n^{wnn}) - \mathcal{R}_{\mathcal{R}}(C^{Bayes}) = \left(B_1 s_n^2 + B_2 t_n^2\right)\{1 + o(1)\}$$

for constants B1 and B2 where:

$$s_n^2 = \sum_{i=1}^{n} w_{ni}^2 and t_n = n^{-2/d}\sum_{i=1}^{n} w_{ni}\{i^{1+2/d} - (i-1)^{1+2/d}\}t_n = n^{-2/d}\sum_{i=1}^{n} w_{ni}\{i^{1+2/d} - (i-1)^{1+2/d}\}$$

5.4 Result Generation

In this phase we Generate our result which is the optimum placement recommended for the IOT devices ,this happens by trying a number of models on the given infrastructure and classifying each model of them until finding an optimum model ,if no optimum model is found between them the models generation process restarts.

6 Human Interface Design

6.1 Overview of user Interface

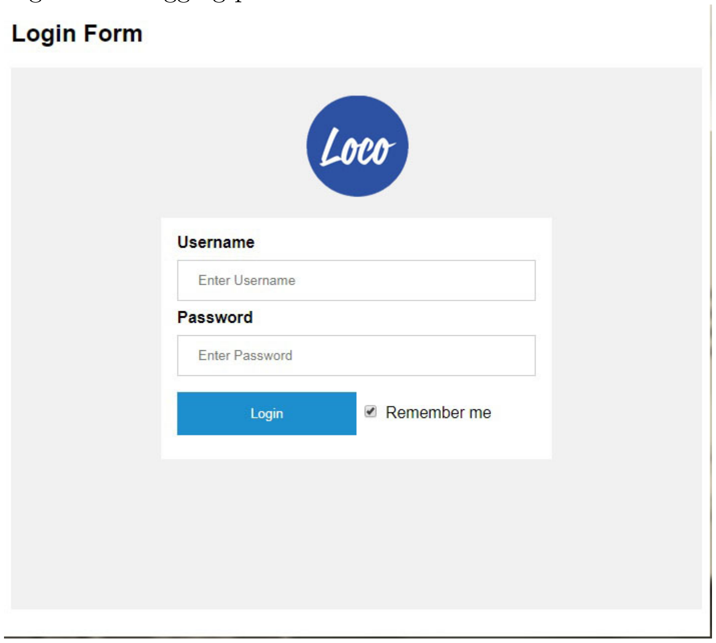
As loco has three different users, it also have three different interfaces with different functionalities. However, they all share the logging interface and the logout button. Logging is done by entering the username and password in the

text fields then pressing the login button. all users will have a notification button and a logout button.
If the user is an Admin, they will have a list of all existing users and all existing buildings. each one with an edit and a delete button. Admin will have two buttons one called Add user which will take them to another page with text fields to enter the name of the new user and a drop-down menu to select the user's type and a save and a return button. The other button is called add building, it allows the admin to add new building by entering the project's id in a text field and then press a save button.
Architect will have navigation menu one leads to a page where they can upload a data-set and split it. Another to upload device's info through text fields and save through a button. last one is to upload the floor-map/blueprint and upload its information through text fields and save through a button.
Auditor will have side navigation menu with a list of all blueprints and their state. upon clicking on them auditor will have new page that views the floor-map and radio buttons to approve or disapprove the map.

6.2 Screen Image

These are Wire-Frames of some of the pages.

Figure 12: Logging panel



6.2.1 Admin

Figure 13: Add User

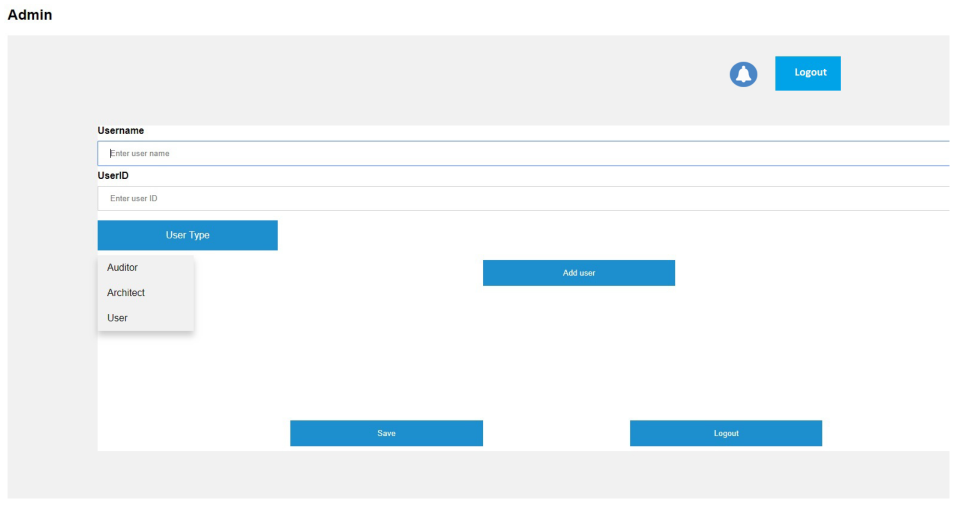
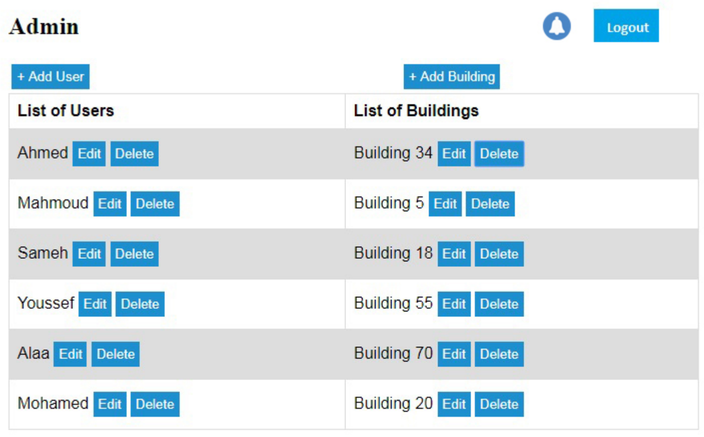


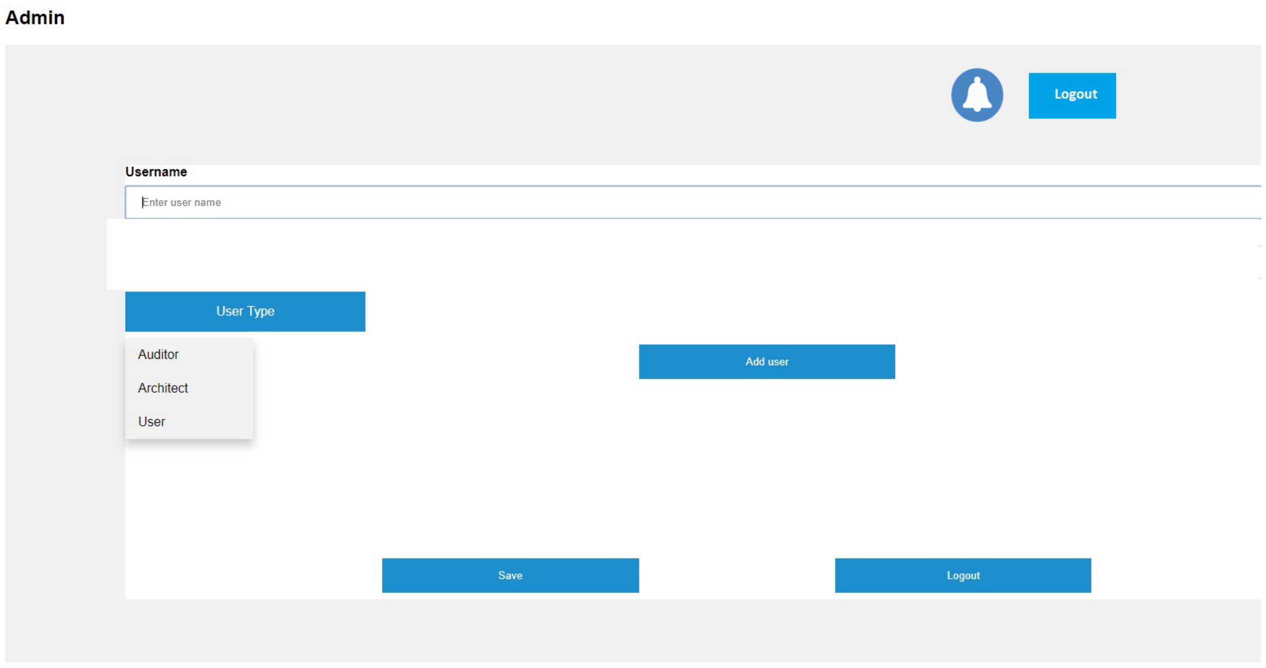
Figure 14: Admin lists



List of Users	List of Buildings
Ahmed Edit Delete	Building 34 Edit Delete
Mahmoud Edit Delete	Building 5 Edit Delete
Sameh Edit Delete	Building 18 Edit Delete
Youssef Edit Delete	Building 55 Edit Delete
Alaa Edit Delete	Building 70 Edit Delete
Mohamed Edit Delete	Building 20 Edit Delete

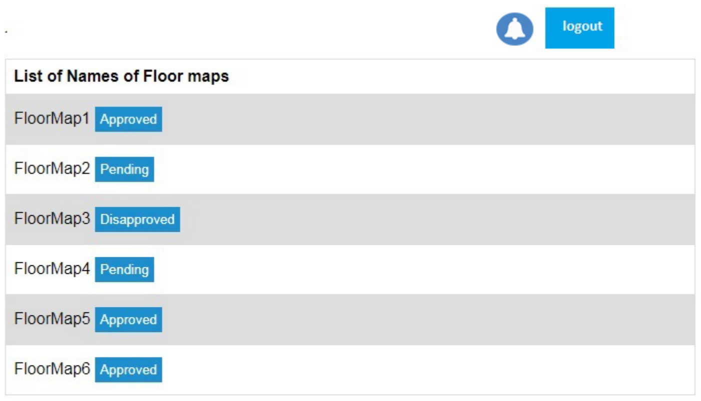
6.2.2 Architect

Figure 15: Uploading floor-map



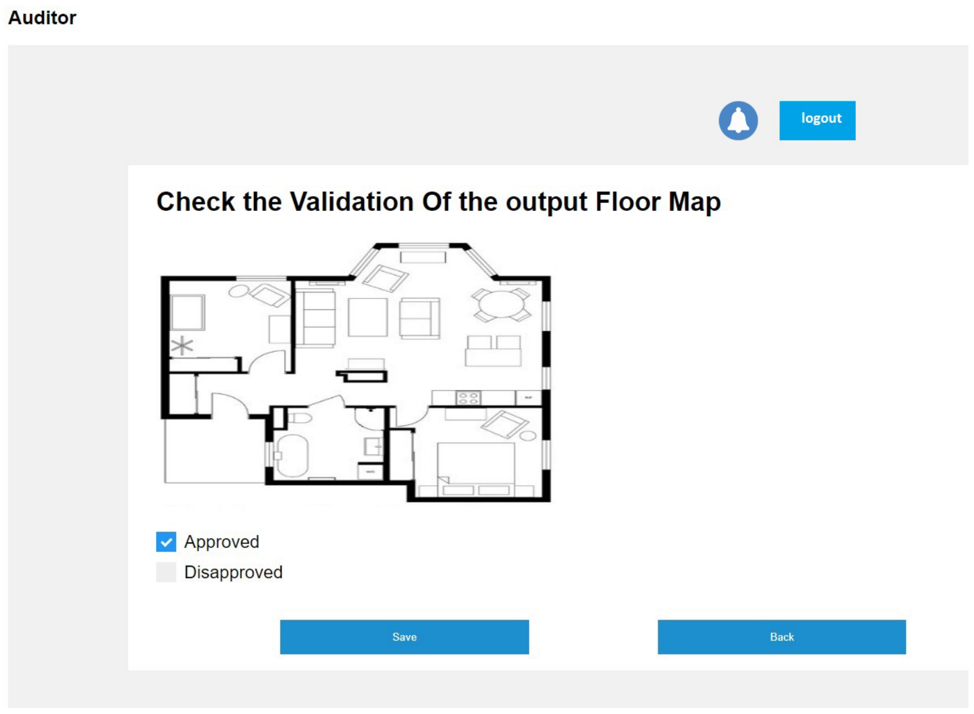
6.2.3 Auditor

Figure 15: auditor viewing all maps



List of Names of Floor maps
FloorMap1 Approved
FloorMap2 Pending
FloorMap3 Disapproved
FloorMap4 Pending
FloorMap5 Approved
FloorMap6 Approved

Figure 16: Auditor approving a map



Check the Validation Of the output Floor Map

[✓ Approved]
Disapproved

6.3 Screen Objects and Actions

6.3.1 Buttons

Figure 17: Admin's add user or buildings buttons

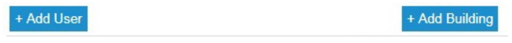


Figure 18: logging out button

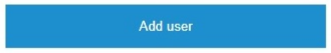
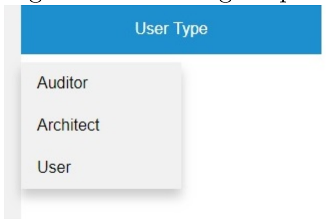


Figure 19: Notification button

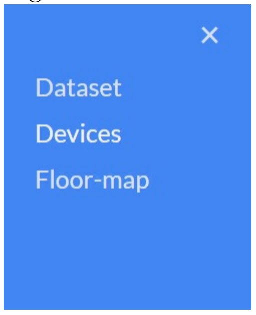
6.3.2 Drop-down List

Figure 20: Building drop down menu

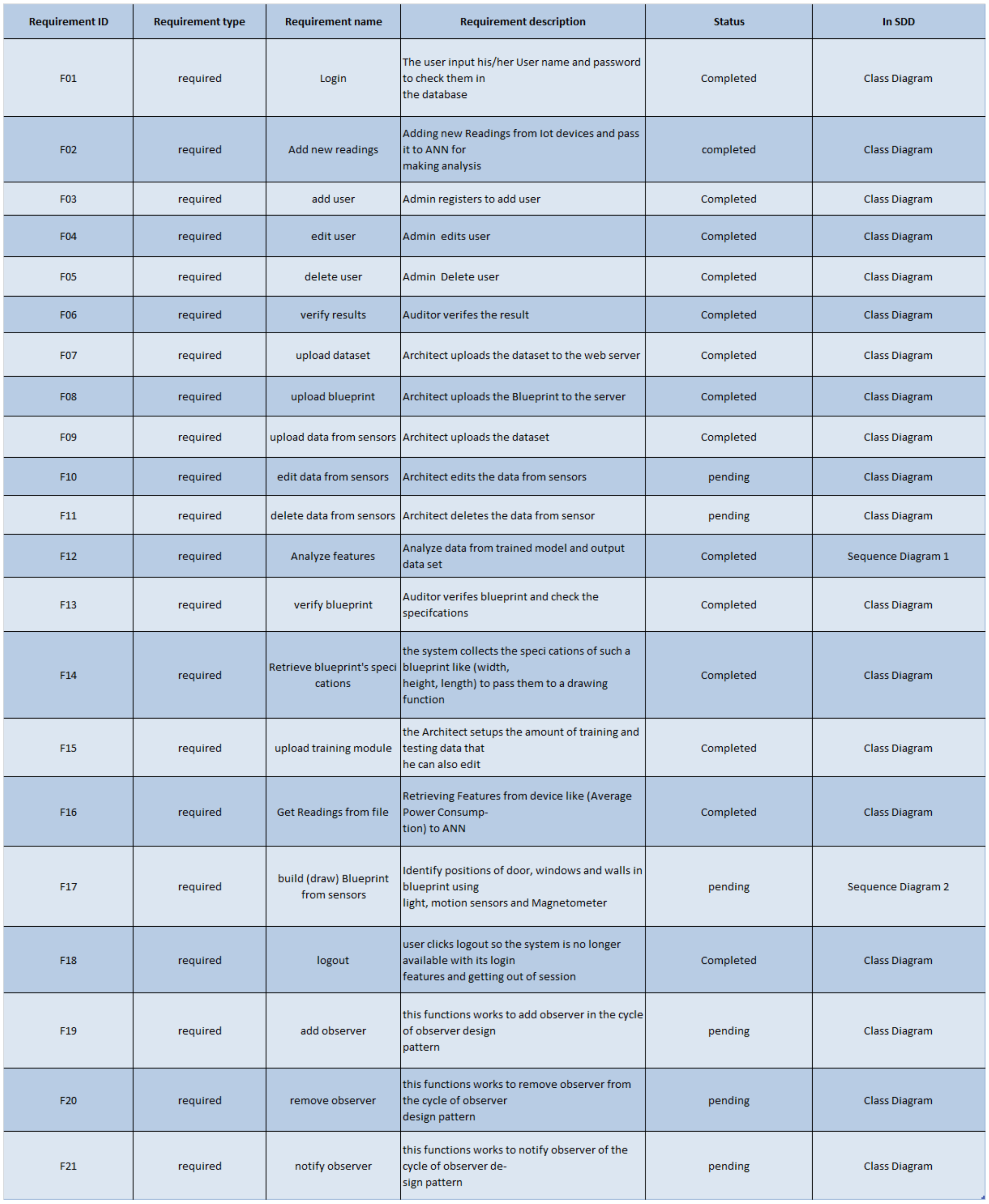


6.3.3 Navigation Menu

Figure 21: Architect's side navigation menu



7 Requirements Matrix



Requirement ID	Requirement type	Requirement name	Requirement description	Status	In SDD
F01	required	Login	The user input his/her User name and password to check them in the database	Completed	Class Diagram
F02	required	Add new readings	Adding new Readings from Iot devices and pass it to ANN for making analysis	completed	Class Diagram
F03	required	add user	Admin registers to add user	Completed	Class Diagram
F04	required	edit user	Admin edits user	Completed	Class Diagram
F05	required	delete user	Admin Delete user	Completed	Class Diagram
F06	required	verify results	Auditor verifes the result	Completed	Class Diagram
F07	required	upload dataset	Architect uploads the dataset to the web server	Completed	Class Diagram
F08	required	upload blueprint	Architect uploads the Blueprint to the server	Completed	Class Diagram
F09	required	upload data from sensors	Architect uploads the dataset	Completed	Class Diagram
F10	required	edit data from sensors	Architect edits the data from sensors	pending	Class Diagram
F11	required	delete data from sensors	Architect deletes the data from sensor	pending	Class Diagram
F12	required	Analyze features	Analyze data from trained model and output data set	Completed	Sequence Diagram 1
F13	required	verify blueprint	Auditor verifes blueprint and check the specifcations	Completed	Class Diagram
F14	required	Retrieve blueprint's speci cations	the system collects the speci cations of such a blueprint like (width, height, length) to pass them to a drawing function	Completed	Class Diagram
F15	required	upload training module	the Architect setups the amount of training and testing data that he can also edit	Completed	Class Diagram
F16	required	Get Readings from file	Retrieving Features from device like (Average Power Consump- tion) to ANN	Completed	Class Diagram
F17	required	build (draw) Blueprint from sensors	Identify positions of door, windows and walls in blueprint using light, motion sensors and Magnetometer	pending	Sequence Diagram 2
F18	required	logout	user clicks logout so the system is no longer available with its login features and getting out of session	Completed	Class Diagram
F19	required	add observer	this functions works to add observer in the cycle of observer design pattern	pending	Class Diagram
F20	required	remove observer	this functions works to remove observer from the cycle of observer design pattern	pending	Class Diagram
F21	required	notify observer	this functions works to notify observer of the cycle of observer de- sign pattern	pending	Class Diagram

References

[1] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," *Journal of pharmaceutical and biomedical analysis*, vol. 22, no. 5, pp. 717–727, 2000.

[2] P. Soucy and G. W. Mineau, "A simple knn algorithm for text categorization," in *Proceedings 2001 IEEE International Conference on Data Mining*, pp. 647–648, IEEE, 2001.

[3] K. M. Desai, S. A. Survase, P. S. Saudagar, S. Lele, and R. S. Singhal, "Comparison of artificial neural network (ann) and response surface methodology (rsm) in fermentation media optimization: case study of fermentative production of scleroglucan," *Biochemical Engineering Journal*, vol. 41, no. 3, pp. 266–273, 2008.

[4] T. Srivastava, "Introduction to knn, k-nearest neighbors : Simplified," Mar 2018.

[5] V. Paruchuri, "K nearest neighbors in python: A tutorial," Feb 2018.
(1)
(2)
(3)
(4)
(5)