**Indian Institute of Information Technology, Allahabad**

**Software Engineering**

SOFTWARE DESIGN SPECIFICATION

Anavadya - An Occupancy Management system for smart buildings

**GROUP MEMBERS-**

**Asha Jyothi Donga - IIT2019006**

**Sampada Manoj Kathar - IIT2019007**

**Alka Trivedi - IIT2019055**

**Sumit Bakoliya - IIT2019083**

Copyright © 2021 by the authors

TABLE OF CONTENT

1. [**Introduction**](#_30j0zll) **2**
   1. [Purpose of this document](#_3znysh7) 2
   2. [Scope of the development project](#_tyjcwt) 2
   3. [Definitions, acronyms, and abbreviations](#_1t3h5sf) 2
   4. [References](#_2s8eyo1) 2
   5. [Overview of document](#_3rdcrjn) 2
2. [**Conceptual Architecture/Architecture Diagram**](#_lnxbz9) **3**
   1. [Structure and relationships](#_1ksv4uv) 5
   2. [User interface issues](#_2jxsxqh) 6
3. [**Logical Architecture (Class Diagram, Sequence Diagram, State Diagram)**](#_3j2qqm3) **6**
   1. [Logical Architecture Description](#_4i7ojhp) 18
   2. [Class name: Login](#_1ci93xb) 21
   3. [Class Name:](#_2bn6wsx) Building Details 22
   4. [Class Name:](#_3as4poj) Present Occupancy 25
   5. [Class Name:](#_49x2ik5) Room\_x Occupancy 26
   6. [Class Name:](#_147n2zr) Warnings 27
   7. [**Execution Architecture**](#_2lwamvv) **32**
   8. [**Reuse and relationships to other products**](#_3l18frh)32

[**5.0 Design decisions and tradeoffs**](#_4k668n3) **32**

* 1. [**Pseudocode for components**](#_1egqt2p) **33**
  2. [Class name: Login](#_1ci93xb) 21
  3. [Class Name:](#_2bn6wsx) Building Details 22
  4. [Class Name:](#_3as4poj) Present Occupancy 25
  5. [Class Name:](#_49x2ik5) Room\_x Occupancy 26
  6. [Class Name:](#_147n2zr) Warnings 27

[**7.0 Appendices (if any)**](#_3mzq4wv) **48**

# **The Software Design Specification**

# **Introduction**

The Software Design Document is a document to provide documentation which will be used to aid in software development by providing the details for how the software should be built. Within the Software Design Document are narrative and graphical documentation of the software design for the project including use case models, sequence diagrams, collaboration models, object behaviour models, and other supporting requirement information.

# **Purpose of this document**

This document will define the design of the one runway simulator. It contains specific information about the expected input, output, classes, and functions. The interaction between the classes to meet the desired requirements are outlined in detailed figures at the end of the document.

# **Scope of the development project**

We describe what features are in the scope of the software and what are not in the scope of the software to be developed.

In Scope:

1. A GUI Application for computing frequently used areas in smart buildings based on occupancy in real time.
2. Admin/Caretaker can see the graphs of occupancy of multiple halls/rooms in smart buildings in a particular time.
3. Admin can manage the lights, fans, AC’s, Elevators, etc, thus saving energy consumption. .

Out of Scope:

1. Switching ON/OFF the electrical devices.
2. There is no interference of Iot devices for managing the electrical devices.

# **Definitions, acronyms, and abbreviations**

Acronyms and Abbreviations:

1. “Anavadya©” :Copyrighted app name.

Definitions:

1. Smart Buildings- Government, Private offices, Hospitals, Malls.
2. “Anavadya©” - An application for smart buildings for effective energy management.

# **References**

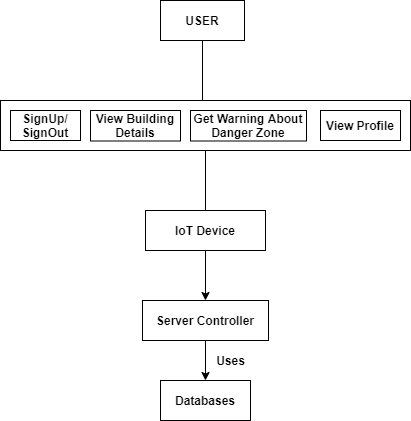
* + 1. R. S. Pressman, Software Engineering: A Practitioner's Approach, 5th Ed, McGraw-Hill, 2001.
    2. IEEE SDS template

# **Overview of document**

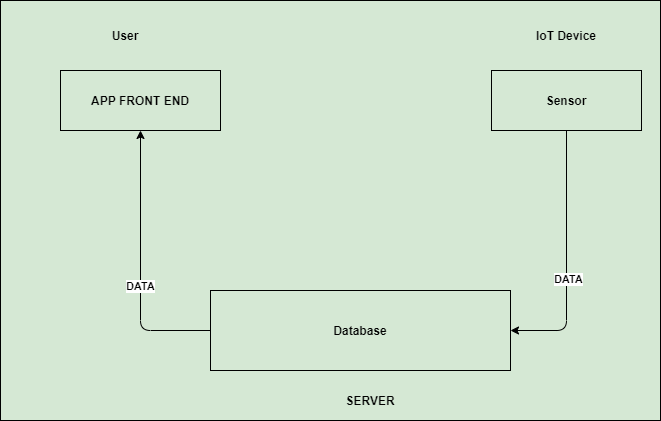
This SDS is divided into seven sections with various sub-sections. The sections of the Software Design Document are

1. **Introduction**: describes about the document, purpose, scope of development project definitions and abbreviations used in the document.
2. **Conceptual Architecture**/**Architecture Diagram:** describes the overview of components, modules, structure and relationships and user interface issues.
3. **Logical Architecture:** describes Logical Architecture Description and Components.
4. **Execution Architecture:** defines the runtime environment, processes, deployment view.
5. **Design Decisions and Trade-offs:** describes the decisions taken along with the reason as to why they were chosen over other alternatives.
6. **Pseudocode for components:** describes pseudocode, as the name indicates.
7. **Appendices:** describes subsidiary matter if any

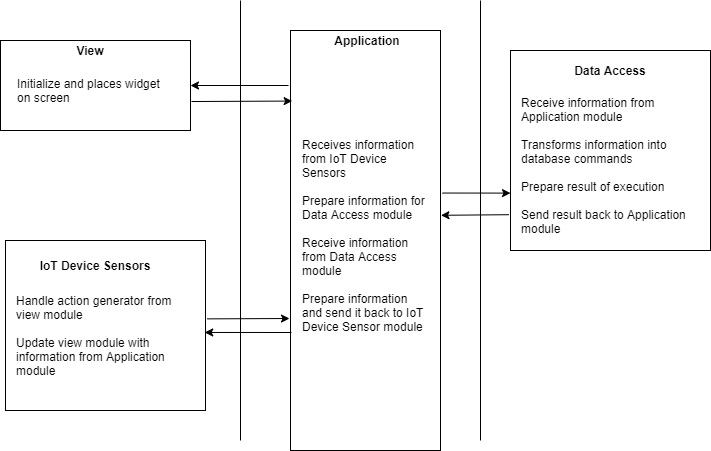
# **Conceptual Architecture/Architecture Diagram Architecture Diagram 1:**



**Architecture Diagram 2:**



* 1. **Overview of modules / components**

****

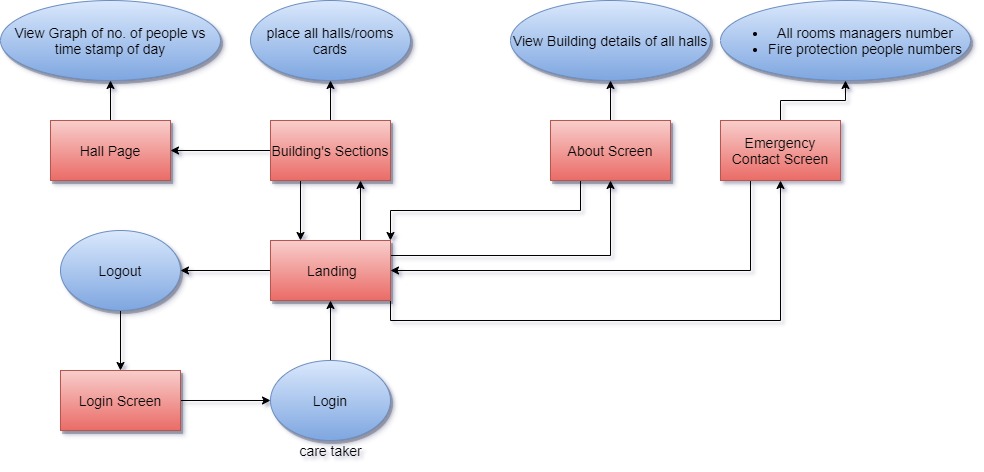
***NOTE:***

The horizontal lines represent the separation of modules.

More than one box within the same section represents sub-module

# **Structure and relationships**

* + 1. **User’s/Caretaker’s Side**



***NOTE:***

The boxes represent individual screens.

The circles represent actions that do not have screens. The arrows represent navigation between screens.

# **User interface issues**

# 

# This section will address User Interface issues as they apply to the following hypothetical users Anavadya.

User A is a 25-40 year old Care-Taker, of the smart building, who is fairly comfortable with technology. He/She is proficient with using most common computer applications, though not much aware of application development or dealing with problems. He may acknowledge that this application will be super useful for him to manage the electricity usage in the whole building.

Since User A is familiar with using web applications, Anavdya, will use common user interface conventions. For example, links between screens will use ordinary, easy to understand descriptions such as “Login”, ”Current Occupancy”, ‘’Emergency Page”,”Occupancy History” etc. To maintain consistency, any other links will also appear in the bottom half of the screen. Color combination has been so chosen, that allows the user to read all of the text on the screen in direct sunlight. Text size is reasonably larger and, therefore, more readable. Again, information fetching would be a few clicks away with minimal/no extra efforts. The user will just have to open a particular hall/room of that building and database will use the latest information obtained from the Iot devices and showcase the data numerically as well as drawing graph

* + 1. **Class Diagram explanation:**

Most of the classes extend the AppCompatActivity class which is being shown by **association** linkage. It is being shown by black-coloured diamond. The classes which extends AppCompatActivity are: Login, StudentLanding, AdminLanding, Home, Notifications, ViewProfile\_Student, JobListings, Training\_Student, StudentAbout, StudentFAQ, AdminFAQ, Training\_Admin, ViewProfile\_Admin and AdminAbout.

There is also some **composition** linkage, shown by lines with an arrow, which signifies that if the parent class is removed, the child class also loses its existence. These are: PdfAdapter depends on ArrayAdapter<Pdf>, ViewProfile\_Student and JobListings depends on StudentAccount, AboutJob depends on Home, and FilePath depends on Training\_Admin.

# **Sequence Diagram:**

Arrow line signifies there is a send message taken place. Response is being shown by dotted arrows.

* + - 1. **Login Page:**Allows User to login
      2. **Building details Page:** Allows User to view building details like number of rooms, capacity of each room, number of elevators, opening and closing times , number of people etc.
      3. **Present Occupancy Page**:Allows User to view the current frequency of occupation of all rooms of the building.
      4. **Room Occupancy Page:** Allows User to view the frequency occupation of a particular room in terms of day/week/month.
      5. **Warning Page:** Users can get the warnings about the areas which have accessibility more than capacity.

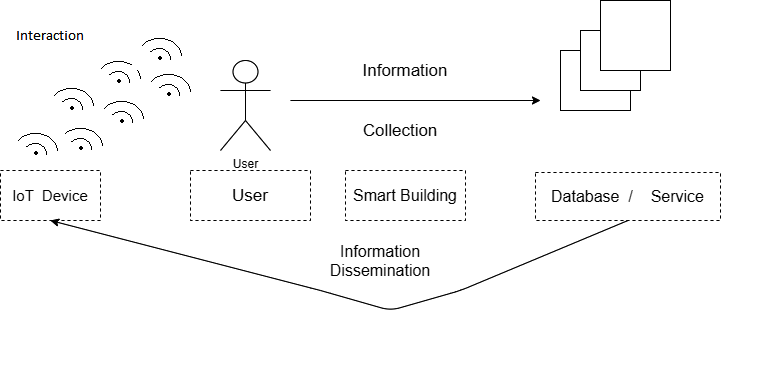
# **State Diagram:**

Initial state is being shown by starting with a black dot. Final State is being shown by the black dot surrounded by an empty circle.

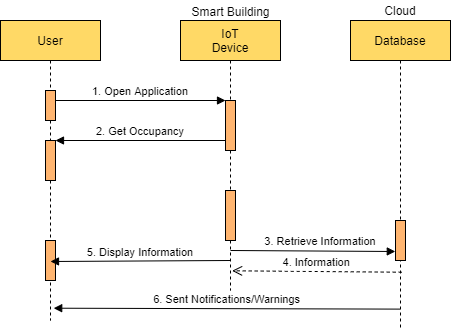
* + - 1. **Login Page:**Allows user to enter credentials, which are being checked for authentication in the back-end. After being authenticated successfully he can land on building details page and present occupation page.
      2. **Building Details Page:** Allows users to view Building details and can sign out from this page by clicking on the sign out button.
      3. **Present Occupation Page:** Allows users to view the present occupation of all rooms .For more details, users can click on the room card which lands on the room occupation page. On clicking the warnings button, it will land on the warning page.By clicking Back , it lands on the parent page. By clicking on sign out users can sign out from the software.
      4. **Room Occupation Page:** By clicking on the room card in the present occupation page it lands on this page. Allows users to view the history and present occupation of a particular room.
      5. **Warning Page:**By clicking on the warnings in the present occupation page it lands on this page.Allows users to view the warnings of present occupation.

# **3. Logical Architecture (Data Flow Diagram, Sequence Diagram, State Diagram)**

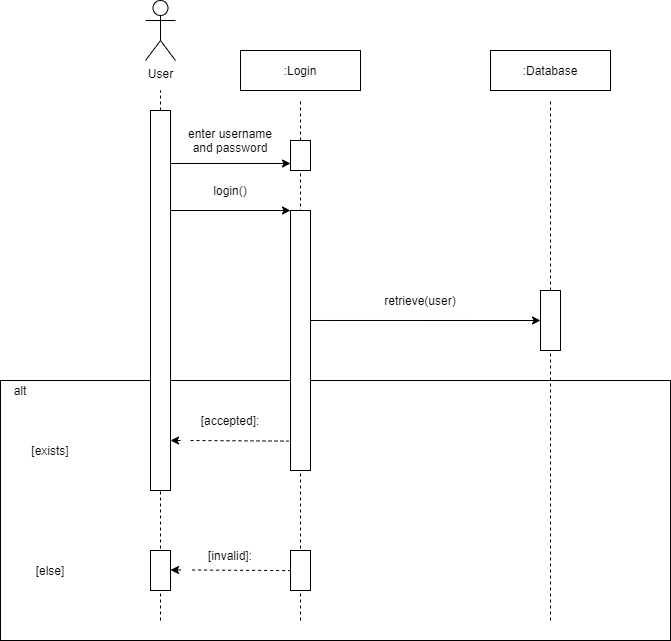
**DataFlow Diagram:**

****

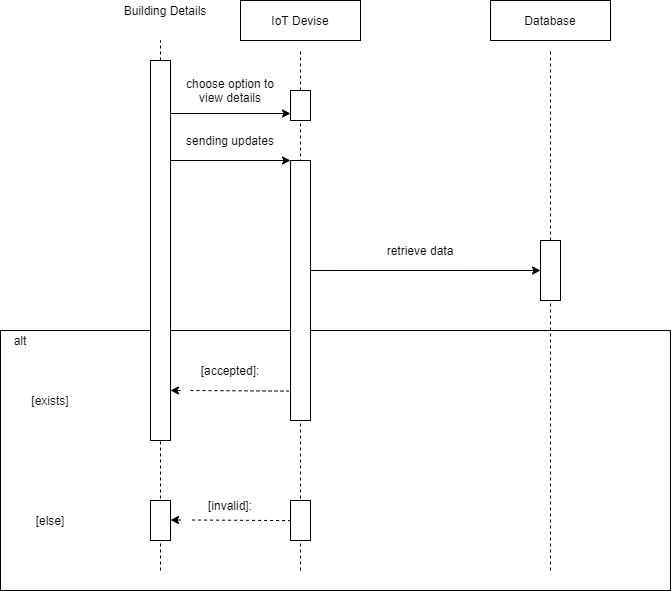
# **Sequence Diagrams:**



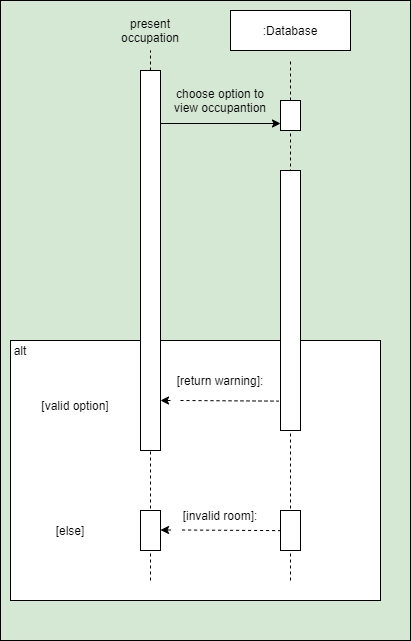
# **Sequence Diagram:Login Page**



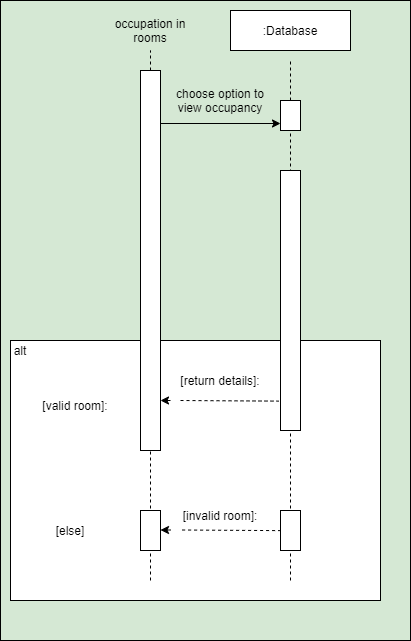
**Sequence Diagram: Building Details Page**

****

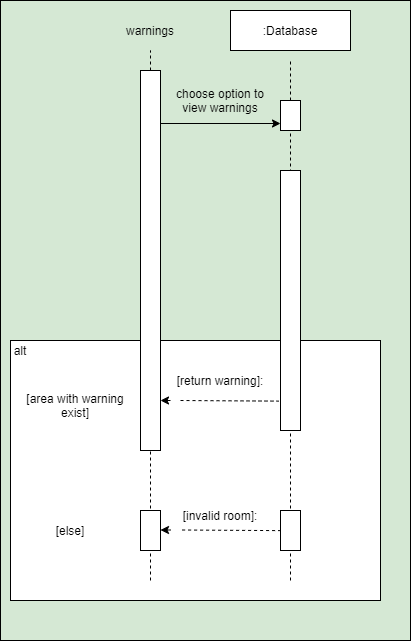
**Sequence Diagram: Present Occupation Page**

****

**Sequence Diagram: Occupation of Room Page**

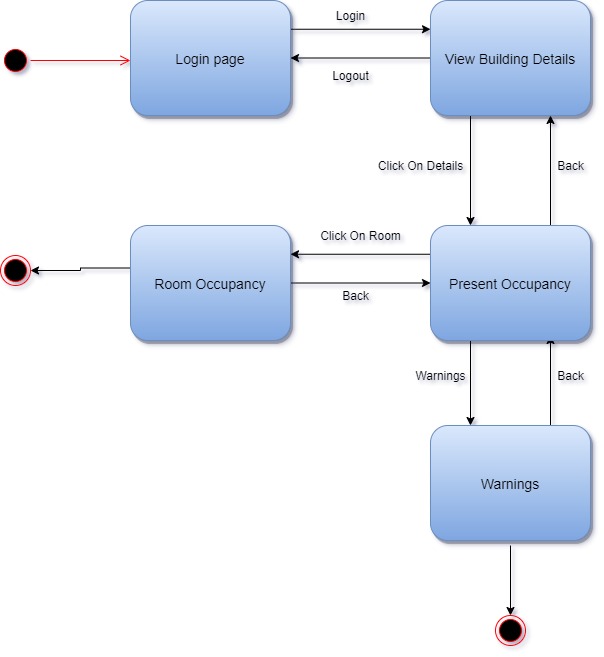


**Sequence Diagram: Warnings Page**

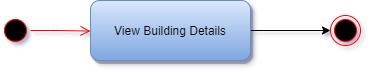


**State Diagrams:**

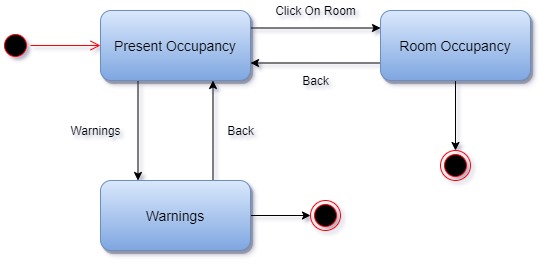
**State Diagram:Login Page**



**State Diagram: Building Details Page**



**State Diagram: Present Occupation Page**



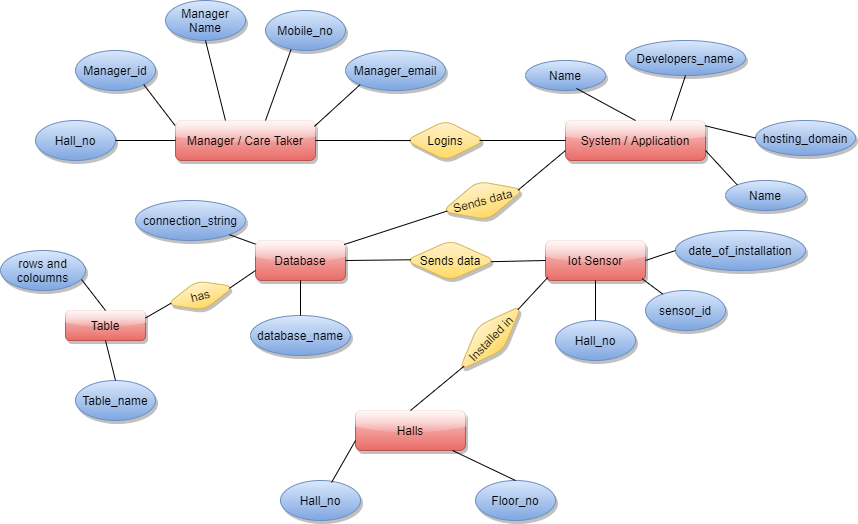
**State Diagram: Occupation of Room Page**



**State Diagram: Warnings Page**

****

**EntityRelationship Diagram:**

****

# **Class name: Login**

**Description:** This class allows the user to enter the system by authenticating the entered credentials.

# **3.2.1Method 1: onCreate()**

**Input:** savedInstanceState, Username, Password

**Output:** Launch the Activity, SignIn

**Method Description:**

When an Activity first calls or launches then onCreate(Bundle savedInstanceState) method is responsible for creating the activity. Whenever orientation(i.e. from horizontal to vertical or vertical to horizontal) of activity gets changed the object of Bundle class will save the state of an Activity. Basically Bundle class is used to store the data of activity whenever the above condition occurs in the application. SetContentView is used to fill the window with the UI provided from the layout file. This method takes input as username and password and as a result opens the landing page if login is successful.

**3.2.2Method 2: User\_Login()**

**Input:** SignIn through Username and password

**Output:** User landing on the Building Details page.

# **Method Description:**

Intentsare asynchronous messages which allow application components to request functionality from other Web components. This method allows users to login through their account. SignIn Intent stores the resultant value by checking whether the sign from Username and password is authorized login or not from the database. If the user is not a client then the login will fail and if it is an authorized login then it will lead to a user landing page.

**3.3 Class Name: Building Details**

This class basically handles details of the smart building for the application.

* + 1. **Method1: oncreate()**

**Input:** mAuthListener

**Output:** Shows the building details like building name, number of rooms etc.

# **Method Description:**

This method takes care of the new details that this is getting from the IoT device sensors, i.e. as the number of occupancy changes in the building in any area it will immediately updated in the building details of the application.

* + 1. **Method2: view\_presentOccupancy()**

**Input:** Click Action on the view Present Occupancy option.

**Output:** User lands on the present occupancy page.

**Method Description-** On clicking this method user will be able to land the present occupancy of the building. If the present occupancy is changing frequently users will be able to see that too.

* + 1. **Method3: goback()**

**Input:** Click action on goback option

**Output:** landing page

# **Method Description-**

This method is used to go back to the login page the user is in. By clicking this method the user will be able to redirect to the login page.

# **ClassName:Present Occupancy**

# **Class Description:**

This class allows to view present occupancy of the building

* + 1. **Method 1**:**oncreate()**

**Input:** mAuthListener

**Output:** Shows present occupancy of every room in form of cards.

**Method Description:**

It fetches data from the database, which has been regularly been put by Iot Devices, and then plots the data in the cards (consisting of details of rooms).

**Method 2**: **show\_warnings()**

**Input:** Click Action on warings option.

**Output:** User is landed on the warings page.

# **Method Description:**

By clicking this method a warning will appear to the users.

* + 1. **Method 3:view\_roomXOccupancy()**

**Input:** Click action on view particular room occupancy option.

**Output:** User is landed on the room x occupancy page.

# **Method Description:**

By clicking this method a room x occupancy page will appear to the users.

* + 1. **Method 4:goBack():**

**Input:** Click Action on goback option

**Output:** User is landed on the Building details page.

# **Method Description:**

By clicking on this method the user will be redirected to the previous page. Users will be redirected to the Building Details page.

# **Class Name: Room\_x occupancy**

# **Class Description:**

This class will contain the occupancy in any particular room (for example, occupancy in room no. 1).

* + 1. **Method1:onCreate()**

**Input:** Bundle object and button clicks.

**Output:** Downloads pdf.

# **Method Description:**

It fetches data from the database, which has been regularly been put by Iot Devices, and then plots the data into graphs of No. of people Vs. Time Stamp. Here we will plot the data with a timestamp of each hour of a day, for one week. So that we get a fair idea of occupancy of a particular room for the whole week, and displays on the screen.

* + 1. **Method2:goBack()**

**Input:** Click Action on the go back option.

**Output:** User lands on the present occupancy page.

# **Method Description:**

By clicking on this method the user will be redirected to the previous page. Users will be redirected to the present occupancy page.

# **Class Name: Warning**

# **Class Description:**

This will give the user the details of the warning, i.e., if any room is exceeded maximum occupancy this method shows those areas to the user.

* + 1. **Method 1: onCreate()**

**Input:** Bundle object and button clicks.

**Output:**

# **Method Description:**

This method fetches data from the server and shows the warnings in ListView.

* + 1. **Method 2: goBack()**

**Input:** Click action on goback option.

**Output:** User will land to the present occupancy page.

# **Method Description:**

By clicking on this method the user will be redirected to the previous page. Users will be redirected to the present occupancy page.

# 

# **4.1 Execution Architecture**

Runtime environment required is any device supporting Browsers such as Chrome, FireFox, Safari, etc.

# 

# **Pseudocode for components**

# **Class Name: Login**

Method 1: **onCreate()**

**Pseudo-code:**

**Input:** Email/Username, Password

**Output:** Launch the Activity, SignIn

1. onCreate(savedInstanceState);
2. setContentView(layout.activity\_main);
3. button = stores the ID of Username sign-in button
4. mAuth = get passwordAuth instance
5. editTextUsername = stores the ID of Admin editTextUsername
6. editTextPassword = stores the ID of Admin editTextPassword
7. buttonSignIn = stores the ID of Admin SignInButton
8. progressDialog = create new progressDialog object
9. setOnClickListener(new View.OnClickListener()
10. public void onClick(View v)
11. signIn() ) // Admin sign in
12. mAuthListener = new passwordAuth.AuthStateListener()
13. void onAuthStateChanged( passwordAuth )
14. if passwordAuth.getCurrentUser() != null then

start another Activity

end if

1. if passwordAuth.getCurrentUser() != null then

finish();

start another Activity

end

**Method 2: User\_Login()**

**Pseudo-code:**

**Input:** view Username, password

**Output:** Admin landing page of login successful

1. String Username = get username from the user
2. String password = get password from the user
3. if string username!=null then

print the username of the user

return

1. end if
2. if string password !=null then

print the password of the user

return

end if

1. Shows progress print “Login in please wait”
2. verify (email, password) by sending it to firebaseAuth
3. result= get result from firebaseAuth
4. public void onComplete()
5. Dismiss the progress dialog
6. if result is Successful then

finish();

start another Activity leading to Admin Landing page

end if

# **Class Name: Building Details**

**Method 1:onCreate (savedInstanceState)**

**Input:** mAuthListener

**Output:** landing page of building details

**Pseudo-code:**

1. onCreate(savedInstanceState);
2. If user!=null

show further options

else

return login page

**Method2: view\_presentOccupancy()**

**Input:** Click Action on the view Present Occupancy option.

**Output:** User lands on the present occupancy page.

**Pseudo-code:**

if view\_present\_occupancy

return person.cnt;

else

return building.landing\_page;

**Method3: goback()**

**Input:** Click action on goback option

**Output:** landing page

**Pseudo-code:**

If room\_x\_present\_occupancy

return building.landing page

else

return room.present\_occupancy

# **Class Name: Present Occupancy.**

**6.1.4Class Name:room\_x occupancy**

Method 1: **onCreate (savedInstanceState)**

Pseudo-code:

**Input:** Room object fetch from database.

**Output:** Show particula room occupancy

1. Call method ‘setContentView(layout\_name)’.
2. Initialize string variable and ‘InputString’ object to NULL.
3. TRY:
4. A) Make ‘DefaultHttpClient()’ object and set ‘HttpPost’ object to point to the ERP database php url to fetch data from. And print data in the form of strings.
5. B) Make ‘HttpResponse’ object ‘response’ and ‘HttpEntity’ object.

**Method 2**: **goback()**

Pseudo-code:

Input:Bundle object and button clicks.

Output: goes to present occupancy page.

1. Call method ‘oncreate(present\_occupancy)’ .

* + 1. **Class Name: Warning**

**Method 1**: **onCreate ( savedInstanceState)**

**Pseudo-code:**

**Input:**, data fetched from database server.

Output: Shows warnings.

1. Call method ‘setContentView(layout\_name)’.
2. Initialize string variable and ‘InputString’ object to NULL.
3. TRY:
4. A) Make ‘DefaultHttpClient()’ object and set ‘HttpPost’ object to point to the ERP database php url to fetch data from. And print data in the form of strings.
5. B) Make ‘HttpResponse’ object ‘response’ and ‘HttpEntity’ object.

**Method 2**: **goback()**

Pseudo-code:

Input:object and button clicks.

Output: goes to present occupancy page.

1. Call method ‘oncreate(present\_occupancy)’ .