# Design & Reasoning

Using legacy code based on a series of smart light switches situated in multiple locations, I created the base of the smart home. From the legacy code I was able to begin designing the system instantly changing class names. This code also included certain methods which had already been implemented in the previous system that could be recycled under the new design.

# Constraint Compliance

Infrastructure

The way in which I have built the system ensures that objects are separated from each other, i.e., frontend remains in the frontend, like the backend. I have ensured that all objects in the backend are solely for the management of data and to not deal directly with any user input. Prime examples of this are the SmartHome.java and SmartPlug.java classes including sub classes within SmartPlug.java. These do not directly deal with any user input but manage the input data that has been collected via ConsoleOutput.java.

ConsoleOutput.java is the helper class which I have implemented to separate the frontend from the backend. Methods created in this class can be used in the backend for the input of data. For example, the toggle method I have created within SmartHome requires an integer to be entered for the plugID, this method will then set the status of the plug specified to the opposite of what it currently is. This method is called within ConsoleOutputs.java in the frontend under the third option of roomOptions(); where a new variable is created and assigned to the user input.

Room/Plug Combinations

As specified in the brief, combinations for rooms and plugs must be zero, one or many plugs in any room however the same plug, i.e. plug 1, cannot be in many rooms at the same time as it is physically impossible. Creating the arrays, I have a SmartPlug array and SmartRoom array, both which hold data about each object. When asking the user to input data, I have created the system to ask which room the plug is in, this ensures that the same plug is not in multiple rooms at the same time.

Within the display statement to ensure that the plugs were shown within the correct rooms, I added an if statement to sort the plugs into their correct rooms. If the roomID is identical as the roomID that has been entered for the plug, this will only allow the plugs to be displayed if they have matched the ID.

UIX

Following the request of the Alikoti Dev team, I created the system using Java. Alikoti required a Java console interface for the user to interact with the system with nearly, or if possible, all input to be integer based rather than string. Within the system I have created, the user input is based on the ID’s of each room, plug or device which allows the user’s input to be completely integer based. This then follows into the menu options where the user is given a set of options to choose from and again, only must enter the number of the option they would like to choose.

When setting up the system, all plugs are required to be turned off, this has been configured within the system to automatically happen when the user is setting up the plugs therefor the user does not have to enter any more information when setting up the rooms, plugs and devices. Looking further into the status of the plugs and integer-based input, menu options for changing the status of plugs simply rely on which plug or room ID the user inputs, there for making the system more efficient and user friendly; there is also an option to toggle all lights on/off.

Exception Handling

As the development team are expecting good input throughout the system, I have not included exception handling within the system. I could have included it for multiple features throughout the system as with integer-based input it can be easy to input a number which does not exist. The status of plugs is another area where exception handling could have been useful, however, in this case it is managed by the system itself, rather than the user’s input rendering the inclusion useless. Including exception handling would also be irrelevant as the plan for this project is for it to be exported to a GUI created by Alikoti themselves.

Libraries

When creating the system I ensured that all processes and mechanisms were coded by myself and not taken from included libraries such as java.util.Arrays. I included both util.Scanner and StringBuilder libraries within the system as these are simple and do not come with the constraints of other Java libraries. Although I could have used the util.Arrays library for the toString() methods within the system, as instructed, I built them myself and did not use array lists but instead used arrays.

Complexity

The code which I have provided is clear, separated, and functional. When evaluating the code, I ensured the code which had been written was not too complex and was easy to understand, this is so the Alikoti team can edit the code if required. Following this, I have also included pseudo code alongside to explain what the methods carry out and again to reduce the complexity when trying to understand the program. With no complex code in place and pseudo code implemented, the code should be easily understandable when being referenced for future projects.

Extra Functionality

As per the development team’s request, when creating the system, I did not include any extra functionality such as functions within the menu which have not been originally specified by Alikoti themselves. Requirements for this project were to be strictly followed and not to be deviated from, therefor the system which has been produced represents the barebone requirements with no added features or functionality.

## System Options

Unfortunately, when creating the system options, I was unable to finish this section of the system. The system continued to return null errors suggesting a new record was being created within the array, however this was not being populated with information as required. Due to this, the system is currently limited to the pre-defined plugs, rooms and devices, however, once these features are implemented, the system will be fully functional and users can adjust arrays as needed. 