**Software Requirements Specification (SRS)**

Revision History:

|  |  |  |
| --- | --- | --- |
| Date | Author | Description |
| 3-17-19 | Rui Xing | Editing system capabilities |
| 3-18-19 | Shuihan Zhang | Editing system context |
| 3-19-19 | Yuru Wang | Editing quality requirements (non-functional requirements) |
| 3-20-19 | Rui Zhu | Editing fundamental assumptions |
| 3-20-19 | Rui Xing | Editing expected subsets |
| 3-21-19 | Rui Xing, Shuihan Zhang, Yuru Wang, Rui Zhu, Shijie Wen | Editing use cases |
| 3-22-19 | Rui Xing, Shuihan Zhang, Yuru Wang, Rui Zhu, Shijie Wen | Editing use cases |
| 3-23-19 | Rui Xing, Shuihan Zhang, Yuru Wang, Rui Zhu, Shijie Wen | Adding use case |
| 3-23-19 | Shijie Wen | Editing detailed requirements |
| 3-23-19 | Rui Zhu | Editing expected changes |
| 3-23-19 | Yuru Wang | Editing appendices |
| 3-24-19 | Shijie Wen | Modifying detailed requirements |
| 3-24-19 | Rui Xing | Editing introduction |
| 3-24-19 | Rui Xing, Shuihan Zhang, Yuru Wang, Rui Zhu, Shijie Wen | Modifying and improving use cases |

**1. Introduction**

**1.1 Intended Audience and Purpose**

This document is intended to provide information that guides the installation and development process to ensure that all system requirements are met. The following entities may find the file useful:

Key customers: this page details all application requirements understood by the production team. Customers should be able to determine from the information on this page that their requirements will be correctly reflected in the final product.

Server: the server will be able to use this document to the major functions contained in the database. In addition, the database will have a set of system requirements before it can be used. More information about these requirements can be found here.

Development teams: the details of the specific requirements that must be included in the final software build will be located here. Developers can use this document to ensure that the software meets these requirements.

QA team: by developing a test process based on system requirements, the QA team can create a comprehensive test solution to ensure that requirements are met.

**1.2 How to use the document**

Table of Contents:

1.Introduction

2.System Capabilities-- broad description of the purpose of the application

2.1 System Capabilities -- description in prose of all capabilities available to the user in the address book

2.2 System Context -- details any specific system requirements the application will require to run

2.3 Use cases -- A detailed look at each functional requirement, describing the application context both before and after an action is taken

3. Detailed Requirements -- How the application will interact with a user

3.1 Input and output requirements -- A description of allowed inputs and generated outputs

3.1.1 Input -- Describes any restrictions that will be placed on allowed input

3.1.2 Output -- Describes the range of outputs that can be generated

3.2 Detailed Output Behavior -- Output descriptions in prose

4. Quality Requirements - Requirements not pertaining to the function of the application will be listed here

5. Expected Subsets - Expected levels of functionality at checkpoints during development

6. Fundamental Assumptions - Some specifics about input, output, or behavior upon which other requirements are founded will be listed here

7. Expected Changes - Future features and directions the project is expected to take

8. Appendices - Details aiding the understanding of this document

8.1 Definitions and acronyms - Any technical terms or abbreviations will be spelled out here for ease of use of the document

8.1.1 Definitions - Definitions of technical or unusual terminology

8.1.2 Acronyms and Abbreviations - Any abbreviated terms will be expanded here

8.2 References - any external references necessary or helpful to understanding this document will be listed here

**2. System Capabilities**

The goal is to provide intelligent lighting system with a database that enables efficient data storage and access. It provides external access to different identities of users, sensors, actuators, rooms, lights and other information of various interfaces, so that users accessing the database can easily edit various attributes. Users need to install MySQL and Java on their computer. For more detailed capabilities of the database, read the section *System Capabilities*.

**2.1 System Context**

The system needs to be used on the PC side, so the system environment applied is as follows:

Windows:

* Windows 10 (8u51 and above)
* Windows 8.x (Desktop)
* Windows 7 SP1
* Windows Vista SP2
* Windows Server 2008 R2 SP1 (64-bit)
* Windows Server 2012 and 2012 R2 (64-bit)
* RAM: 8.00GB(7.85GB available)
* Processor: Intel(R) Core(TM) i5-7200U CPU @ 2.5GHz 2.70GHz

Mac OS X:

* Intel-based Mac running Mac OS X 10.8.3+, 10.9+
* Administrator privileges for installation’

**2.2 System capabilities**

First, the user enters personal information during registration, which will be stored in the user's properties table. Users with different identities are stored in the same table, but there is a type attribute to distinguish user identities. It is worth mentioning that, in order to achieve the security of the database, the user password stored in the database is encrypted.

Users can view their personal information or modify and delete it. The results of these operations are updated synchronously to the corresponding table entries in the database.

Depending on the user's identity, different permissions are granted to them. For administrators, they can view tables of sensor, room, and actuator, and they can add and remove any sensor, room, and actuator. All have access to the light table to see the light on and off. The results of the on/off operation are also updated synchronously to the database. Administrators and teachers can modify the on/off of lights without restrictions. But only administrators can add and remove lights. When the administrator adds the light, the database will record the installation time and life of the light to facilitate future maintenance and overhaul.

When adding various information, users need to strictly follow the requirements of input format. In order to ensure the validity of the data, the database will conduct input validation before it is stored in the database, and the validation result will be returned to the user.

**2.3 Use Cases**

**Case 1: Server Wants to Register an Account for End Users**

**Players:** Server

**Goals:** The server wants to register a non-existent account before.

**Preconditions:** This account does not exist before registration; the application is open and running with a client book open.

**Case:**

1.1 The server calls the add account function, which provides the user's ID, name, identity, and new password.

1.2 The database adds personal information to the client table.

1.3 Update other tables.

1.4 Return the flag of success.

**Alternate Flows:**

1.2.1 The user decides to "cancel" the workflow.

The application returns to its initial state.

**Exception Flows:**

1.3.1 The data provided by the user is inconsistent with the user name such as entering an existing name.

The application warns the administer about why his or her data has not been accepted.

The user is then able to change the data they provided and resubmit.

**Postconditions:** There is a new user in the client table. It is marked to be saved at the next save point. The user book is aware that it has been altered.

**Case 2: Server Wants to Delete a User Account**

**Player:** Server

**Goals:** The end user wants to register a new account and fill in his/her personal information. This information should be added to the database.

**Preconditions:** The server wants to delete an existing user account. The information should be deleted from the database.

**Case:**

2.1The server calls the delete account function, which provides the user's ID.

2.2 Retrieve the database by ID number and find the corresponding table items.

2.3 Delete the target table entry.

2.4 Update other tables.

2.5 Return the flag of success.

**Alternate Flows:**

2.2.1 The user chooses to "cancel" the process.

The user's personal information will not be removed from the database.

2.2.2 The user that be searched does not exist.

Return the flag of not exist.

**Exception Flows:**

2.3.2 The end user forces the deletion to terminate.

No information is removed.

**Postconditions:** The database removes the user's information and the account no longer exists.

**Case 3: Server Wants to Change a User’s Password**

**Player:** Server.

**Goals:** The server would like to change user’s password.

**Preconditions:** The user has registered, that is, personal information and password already exist.

**Case:**

3.1The server calls the change password function, which provides the user's ID and a new password.

3.2 The database looks up the corresponding table item according to the ID.

3.3 The database saves the encrypted password into the password property.

3.4 Update other tables.

3.5 Return the flag of success.

**Alternate Flows:**

3.2.1 The user chooses to "cancel" the process.

The database will keep the original password of current user.

3.2.2 The user that be searched does not exist.

Return the flag of not exist.

**Exception Flows:**

3.3.1 The user forces the modification to terminate.

No changes are saved.

**Postconditions:** If the user saves the change, the password will be changed and the next time the server search his/her password, it will get a new password.

**Case 4: Server Wants Authentication of the User ID and Password**

**Players:** Server

**Goal:** The server would like to search for username and password

**Preconditions:** The server transfer the user ID and password.

**Case:**

4.1 The server calls the login authentication function, which gives the user ID and password.

4.2 According to the user ID, database finds out corresponding user item.

4.3 Determine whether the password is the same.

4.4 If the user ID and password are correct, return the flag of correct.

**Alternate Flows:**

4.2.1 The user that be searched does not exist.

Return the flag of not exist.

4.2.2 If the user ID and password are not correct, return the flag of error.

**Exception Flows:**

4.3.1 The user forgets the password.

Proceed to case 3

**Postconditions:** The server receives the authentication result.

**Case 5: Server Wants to Add New Lights**

**Players:** Server

**Goals:** The server wants to add new lights to the list of lights he or she can control.

**Preconditions:** User is an administrator; the application is open and running with a light book open.

**Case:**

5.1 The server calls the add light function, which provides the light's ID, roomID, settime, and Life.

5.2 The database adds light information to the light table.

5.3 Update other tables.

5.4 Return flag of success.

**Alternate Flows:**

5.2.1 The user decides to "cancel" the workflow.

The light book he or she controls return to the initial state.

**Exception Flows:**

5.3.1 The data provided by the administrator is inconsistent with format of number of lights such as entering a special character in some light numbers.

The application warns the administrator about why his or her data has not been accepted.

The administrator is then able to change the data they provided and resubmit.

5.3.2 If there is another entry with the same information (e.g. light number, light’s number, sensor’s number), the light list he or she controls can’t change.

**Postconditions:** There is a new light in the light list. It is marked to be saved at the next save point. The light book is aware that it has been altered.

**Case 6: Server Wants to Remove Lights from a Room**

**Players:** Server

**Goal:** The server would like to delete some lights from light table.

**Preconditions:** The operator’s attribute is the administrator.

**Case:**

6.1 The server calls the delete light function, which gives the light ID, room ID and user ID.

6.2 According to the user ID, database determines the current user’s attribute and judge whether he has the permission.

6.3 According to the light ID and room ID, database finds out target light.

6.4 Remove the target light.

6.5 Update the other table.

6.6 Return the flag of success.

**Alternate Flows:**

6.2.1 The current user has no authority to delete the light.

Return the flag of no permission.

6.2.2 The light that be searched does not exist.

Return the flag of not exist.

**Exception Flows:**

6.3.1 The user decides to "cancel" the process after deciding to remove the light.

The database terminates the current operation.

**Postconditions:** The database removes the target light and return the flag of result.

**Case 7: Server Wants to Turn on the Light**

**Players:** Server

**Goal:** The server would like to turn on some lights from light table.

**Preconditions:** The operator’s attribute is the students or administrators and teachers. Students can do this only if there is no person in the room, while administrators and teachers do not need.

**Case:**

7.1 The server calls the turn on light function, which gives the light ID, room ID and user ID.

7.2 According to the light ID and room ID, database finds out target light.

7.3 Change the state of the target light.

7.4 Update the other table.

7.5 Return the flag of success.

**Alternate Flows:**

7.2.1 The light that be searched does not exist.

Return the flag of not exist.

**Exception Flows:**

7.3.1 The user decides to "cancel" the process after deciding to turn on the light.

The database terminates the current operation.

**Postconditions:** The database turns on the target light and return the flag of result.

**Case 8: Server Wants to Turn off the Light**

**Players:** Server

**Goal:** The server would like to turn off some lights from light table.

**Preconditions:** The operator’s attribute is the students or administrators and teachers. Students can do this only if there is no person in the room, while administrators and teachers do not need.

**Case:**

8.1 The server calls the turn off light function, which gives the light ID, room ID and user ID.

8.2 According to the light ID and room ID, database finds out target light.

8.3 Change the state of the target light.

8.4 Update the other table.

8.5 Return the flag of success.

**Alternate Flows:**

8.2.1 The light that be searched does not exist.

Return the flag of not exist.

**Exception Flows:**

8.3.1 The user decides to "cancel" the process after deciding to turn on the light.

The database terminates the current operation.

**Postconditions:** The database turns off the target light and return the flag of result.

**Case 9: Server Wants Authentication of Light Status**

**Players:** Server

**Goals:** Server view the status of the current light through the database

**Preconditions:** It is legal for students, teachers and administrators to do this operation.

**Case:**

9.1 The server sends the number of the light (LID) to the database.

9.2 According to the LID , database finds out target light.

9.3 Return the information of lamp.

9.4 Return the flag of success.

**Alternate Flows:**

9.2.1 The light that be searched does not **exist**.

Return the flag of not exist.

**Exception Flows:**

9.3.1 The user decides to "cancel" the process after deciding to the operation of viewing the status of light, the database terminates the current operation.

**Postconditions:** The server receives the authentication result.

**Case 10: Server Wants to Add New Sensors**

**Players:** Server

**Goals:** The server wants to add new sensors to the list of sensors he or she can control.

**Preconditions:** User is an administrator; the application is open and running with a sensor book open.

**Case:**

10.1 Server calls add sensor functions, which provide the lamp's ID, roomID, and type.

10.2 The database adds sensor information to the lamp table.

10.3 Update other forms.

10.4 Return success flag.

**Alternate Flows:**

10.2.1 The user decides to "cancel" the workflow.

The sensor books he or she controls return to the initial state.

**Exception Flows:**

10.3.1 The data provided by the administer is inconsistent with format of number of sensors such as entering a special character in some sensor numbers.

The application warns the administer about why his or her data has not been accepted.

The administrator is then able to change the data they provided and resubmit.

10.3.2 If there is another entry with the same information (e.g. sensor number, light’s number, sensor’s number), the sensor lists he or she controls can’t change.

**Postconditions:** There is a new sensor in the sensor list. It is marked to be saved at the next save point. The sensor book is aware that it has been altered.

**Case 11: Server Wants to Remove Sensors from a Room**

**Players:** Server

**Goal:** The server would like to delete some lights from sensor table.

**Preconditions:** The operator’s attribute is the administrator.

**Case:**

11.1 The server calls the delete sensor function, which gives the sensor ID, room ID and user ID.

11.2 According to the user ID, database determines the current user’s attribute and judge whether he has the permission.

11.3 According to the sensor ID and room ID, database finds out target sensor.

11.4 Remove the target sensor.

11.5 Update the other table.

11.6 Return the flag of success.

**Alternate Flows:**

11.2.1 The current user has no authority to delete the sensor.

Return the flag of no permission.

11.3.1 The sensor that be searched does not exist.

Return the flag of not exist.

**Exception Flows:**

11.3.1 The user decides to "cancel" the process after deciding to remove the sensor.

The database terminates the current operation.

**Postconditions:** The database removes the target sensor and return the flag of result.

**Case 12: Server Wants to Add New Rooms**

**Players:** Server

**Goals:** The server wants to add new rooms to the list of rooms he or she can control.

**Preconditions:** User is an administrator; the application is open and running with a room book open.

**Case:**

12.1 The server call adds the room function, which provides the roomID, Lightnum, and Sensornum.

12.2 The database adds the room information to the room table.

12.3 Update other tables.

12.4 Return success flag.

**Alternate Flows:**

12.2.1 The user decides to "cancel" the workflow.

The room book he or she controls return to the initial state.

**Exception Flows:**

12.3.1 The data provided by the administer is inconsistent with format of number of rooms such as entering a special character in some room numbers.

The application warns the administer about why his or her data has not been accepted.

The administrator is then able to change the data they provided and resubmit.

12.3.2 If there is another entry with the same information (e.g. room number, light’s number, sensor’s number), the room list he or she controls can’t change.

**Postconditions:** There is a new room in the room list. It is marked to be saved at the next save point. The room book is aware that it has been altered.

**Case 13: Server Wants to Remove Existing Rooms**

**Players:** Server

**Goal:** The server would like to delete some rooms from room table.

**Preconditions:** The operator’s attribute is the administrator.

**Case:**

13.1 The server calls the delete room function, which gives the room ID and user ID.

13.2 According to the user ID, database determines the current user’s attribute and judge whether it can be deleted.

13.3 According to the room ID, database finds out target room.

13.4 Remove the target room.

13.5 Update the other table.

13.6 Return the flag of success.

**Alternate Flows:**

13.2.1 The current user has no authority to delete the room.

Return the flag of no permission.

13.3.1 The room that be searched does not exist.

Return the flag of not exist.

**Exception Flows:**

13.3.1 The user decides to "cancel" the process after deciding to remove the room.

**Postconditions:** The database removes the target room and return the flag of result.

**Case 14: Server Wants to Change the User's Permissions**

**Players:** Server

**Goals:** Server changes the user permissions.

**Preconditions:** Server makes a request to change the user's permissions.

**Case:**

14.1 The server calls the change user identity function, which provides the user ID and the modified identity.

14.2 Based on the user ID, the user is found in the client table.

14.3 Modify the label attribute for this user.

14.4 Return the flag of success.

**Alternate Flows:**

14.2.1 The user that is searched does not exist.

Return the flag of not exist.

**Exception Flows:**

14.2.1 The user decides to "cancel" the process after deciding to the operation of checking the number of people in the room.

The database terminates the current operation.

**Postconditions:** The user is modified to specify permissions.

**Case 15: Server Wants to Add New Actuators**

**Players:** Server

**Goals:** The server wants to add new actuators to the list of actuators he or she can control.

**Preconditions:** User is an administrator; the application is open and running with an actuator book open.

**Case:**

15.1 Server calls add actuator function, which provides the ID and roomID.

15.2 The database adds actuator information to the actuator table.

15.3 Update other tables.

15.4 Return success flag.

**Alternate Flows:**

15.2.1 The user decides to "cancel" the workflow.

The actuator book he or she controls return to the initial state.

**Exception Flows:**

15.3.1 The data provided by the administer is inconsistent with format of number of actuators such as entering a special character in some actuator numbers.

The application warns the administer about why his or her data has not been accepted.

The administrator is then able to change the data they provided and resubmit.

15.3.2 If there is another entry with the same information (e.g. actuator number, light’s number, sensor’s number), the actuator list he or she controls can’t change.

**Postconditions:** There is a new actuator in the actuator list. It is marked to be saved at the next save point. The actuator book is aware that it has been altered.

**Case 16: Server Wants to Remove Actuators from a Room**

**Players:** Server

**Goal:** The server would like to delete some actuators from actuator table.

**Preconditions:** The operator’s attribute is the administrator.

**Case:**

16.1 The server calls the delete actuator function, which gives the actuator ID, room ID and user ID.

16.2 According to the user ID, database determines the current user’s attribute and judge whether it can be deleted.

16.3 According to the actuator ID and room ID, database finds out target actuator.

16.4 Remove the target actuator.

16.5 Update the other table.

16.6 Return the flag of success.

**Alternate Flows:**

16.2.1 The current user has no authority to delete the actuator.

Return the flag of no permission.

16.2.2 The actuator that be searched does not exist.

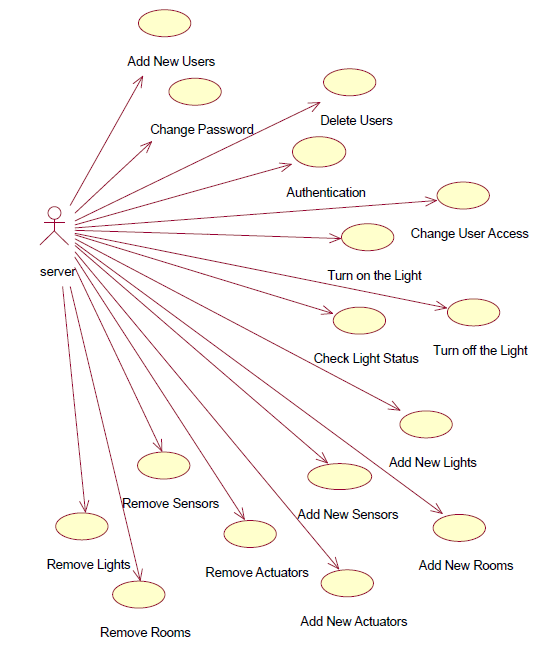
Return the flag of not exist.

**Exception Flows:**

16.3.1 The user decides to "cancel" the process after deciding to remove the actuator.

The database terminates the current operation.

**Postconditions:** The database removes the target actuator and return the flag of result.



**3. Detailed Requirements**

The goal is to specify the input and output format of each attribute in the database to ensure data validity when connected to the database. For a detailed introduction, please read the System Inputs and Outputs section.

**3.1 System Inputs and Outputs**

**3.1.1 Inputs**

The input to the database comes from the server. The input to the database comes from the server. There are 5 tables in the database, namely client table, light table, sensor table, room table and actuator table. The input requirements for each attribute of each table are as follows.

|  |  |  |
| --- | --- | --- |
| Name | Type | Explanation |
| UID | int[1] | UID is the user's account number, which is an integer less than max\_int. |
| name | char[20] | name is a string of up to 20 lengths representing the user name |
| password | char[50] | The password is to save the password of each user. It should be encrypted. |
| label | int[1] | label saves the attribute identification of each user, indicating that he is a student, teacher, or administrator account. |
| LID | int[1] | LID is the light's number in a room, which is an integer less than max\_int. |
| roomID | int[1] | roomID should be generated when adding rooms. They cannot be modified and they are different. |
| State | int[1] | State is an integer that holds the state of the lamp on, off, or damaged |
| Settime | string | SetTime represents the installation time of the bulb, which should be a string limited to yyyy-mm-dd format |
| Life | int[1] | Life is an integer representing the life of a light bulb in hours |
| SID | int[1] | SID is the number of sensor, which is an integer less than max\_int. |
| Type | int[1] | Type is an integer describing the type of sensor |
| Lightnum | int[1] | Lightnum is an integer describing the number of bulbs in a room |
| sensornum | int[1] | sensornum is an integer describing the number of sensors in a room |
| AID | int[1] | AID is the number of actuator, which is an integer less than max\_int. |

**3.1.2 Outputs**

The output of the database is provided to the server. The following table specifies the specific form of the output that will be provided to the server.

|  |  |  |
| --- | --- | --- |
| Name | Type | Explanation |
| UID | Int[1] | UID is the user's account number, which is an integer less than max\_int. |
| name | Char[20] | name is a string of up to 20 lengths representing the user name |
| password | Char[50] | The password is to save the password of each user. It should be encrypted. |
| label | Int[1] | label saves the attribute identification of each user, indicating that he is a student, teacher, or administrator account. |
| LID | Int[1] | LID is the light's number in a room, which is an integer less than max\_int. |
| roomID | Int[1] | roomID should be generated when adding rooms. They cannot be modified and they are different. |
| State | Int[1] | State is an integer that holds the state of the lamp on, off, or damaged |
| Settime | string | SetTime represents the installation time of the bulb, which should be a string limited to yyyy-mm-dd format |
| Life | Int[1] | Life is an integer representing the life of a light bulb in hours |
| SID | Int[1] | SID is the number of sensor, which is an integer less than max\_int. |
| Type | Int[1] | Type is an integer describing the type of sensor |
| Lightnum | Int[1] | Lightnum is an integer describing the number of bulbs in a room |
| sensornum | Int[1] | sensornum is an integer describing the number of sensors in a room |
| AID | Int[1] | AID is the number of actuator, which is an integer less than max\_int. |
| Flag | Bool[1] | Flag is a flag indicating whether the operation on the database is successful |

**3.2 Detailed Output Behavior**

The database provides various access interfaces to the server. This section details the capabilities of these interfaces and their possible output formats.

* **Function1**: query the corresponding account information according to the user UID

Query the client-database with UID as the primary key.

1. If the user of UID does not exist in the database, return null.

2. If the user exists, return the output value.

* **Function2**: query the light information according to LID and roomID

Query the light-database with LID and roomID as the primary key.

1. If the light of SID does not exist in the database, return null.

2. If the light exists, return the output value.

* **Function3**: query light information in a room through roomID

Query the information of all the bulbs in the database whose room number equals the query value

1. If no light bulb has the same room number as the query value, return empty.

2. In other cases, list all light bulb information with room number equal to query value.

* **Function4:** query the sensor information according to the sensor SID

Query the sensor-database with SID as the primary key.

1. If the sensor of SID does not exist in the database, return null.

2. If the sensor exists, return the output value.

* **Function5:** query sensor information in a room through roomID

Query the information of all the bulbs in the database whose room number equals the query value

1. If no sensor with room number equal to the query value is found in the database, return empty

2. In other cases, list all sensors information with room number equal to query value.

* **Function6:** query room information by roomID

Query the room-database with roomID as the primary key.

1. If the user of rommID does not exist in the database, return null.

2. If the user exists, return the output value.

* **Function7:** list all the rooms

input: no iuput

output: roomID(int[1]), lightnum(int[1]), sensornum(int[1])

Detailed output：

Traverse the room database and output all information.

1. If the database is empty, return null.

2. Output all information of the room database.

* **Function8:** query the sensor information based on the actuator AID

Query the actuator with AID as the primary key.

1. If the actuator of AID does not exist in the database, return null.

2. If the driver exists, return the output value.

* **Function9**: add/remove/modify a light

First use the roomID as the primary key to query the room-database, and then use the roomID and the LID as the primary key to query the light-database.

1.If the room dose not exist, the flag is false.

2.If the LID in the room has exist, the flag is false.

3. Else the flag is true

* **Function10:** add/delete/modify a room

Query the room-database with roomID as the primary key.

1.If the roomID has already exist ，the flag is false.

2 Else the flag is true

* **Fuction11:** add/remove/modify a sensor

First use the roomID as the primary key to query the room-database, and then use the room number and the SID as the primary key to query the light-database.

1.If the room does not exist, the flag is false.

2. If the SID in the room has exist, the flag is false.

3. Else the flag is true.

* **Fuction12:** add/delete/modify an actuator

First use the roomID as the primary key to query the room-database, and then use the room number and the AID as the primary key to query the light-database.

1. If the room does not exist, the flag is false.

2. If the AID in the room has exist, the flag is false.

3. Else the flag is true.

* **Fuction13:** add/delete/modify a user

input: SID(int[1]), roomID(int[1])

output: flag(bool[1])

Detailed output：

1. If the UID has already exist, the flag is false.

2. In other condition, the flag is true.

**4. Quality Requirements (Non-functional Requirements)**

**Behavioral (run-time) qualities:**

* Performance: The comprehensive query time of supporting users to concurrent access to data is less than 1 s. Average response time of key business during peak period is as short as possible. The system needs to run continuously for 7 x 24 hours. After running for a period of time, the overall execution efficiency will not decrease.
* Security: In the transmission of data information, when passing through unsafe networks (such as INTERNET network), it is necessary to provide integrity checks for the data information transmitted. The database should have a sound authority management strategy and support the principle of minimizing authority and reasonable authorization. Implementing Secret Storage of Important Business Data by Encryption
* Availability: Satisfy the server’s basic request of data access and processing. Shorten the response speed.
* Reliability: Minimizing deadlocks as much as possible. Keep transactions short and try not to make a transaction too complex to read and write. If the transaction is too complex, it will occupy more resources and increase processing time. It will easily conflict with other transactions, and ultimately increase the deadlock probability. Database need to back key data up. When the database breaks down, data recovery time is less than critical value. Relevant information of different tables in database should be consistent

**Developmental (design-time) qualities:**

* Modifiability: Layer software structure and consider enough extension ports.
* Maintainability: Ensure standardized document structure, clear code structure and optimized System Architecture
* Portability: The system consists of environment-independent and environment-related codes. MySQL database is written in C and C++ and tested with various compilers to ensure the portability of source code.
* Reusability：Ease of extending application capabilities. Application should be modularized such that adding/extending features and functions only require changes to a single component and the interface with that component, if applicable.
* Understandability: All kinds of design documents the project members write have complete information and conform to specifications. Code has detailed comments and distinct structure

**5. Expected Subsets**

* Ability to add user information.
* Administrators can add and remove sensors, lights, rooms, actuators to the database.
* Ability to view the status of lights.
* Administrators and teachers can modify the status of lights.

**6. Fundamental Assumptions**

* The system can run on any device that is capable of running MySQL.
* The database will not terminate when all windows are closed.
* Data updates are changed by the control module using DAO.

**7. Expected Changes**

Features to Add:

* User Defined Fields
* Links to Social Media
* Deletion of Many Lights at Once
* Addition of Many Lights at Once
* Deletion of Many Sensors at Once
* Addition of Many Sensors at Once
* Deletion of Many Rooms at Once
* Addition of Many Rooms at Once
* Allow users to have different role and permission

Future Platforms:

* Write Front End For Android
* Port Application to iOS

**8. Appendices**

**8.1 Definitions and acronyms**

**8.1.1 Definitions**

|  |  |
| --- | --- |
| **Keyword** | **Definitions** |
| Modifiability | The ease with which a program can be modified correctly. |
| Reusability | The same component can be reused without modification or minor modifications. |
|  |  |
|  |  |

**8.1.2 Acronyms and abbreviations**

|  |  |
| --- | --- |
| **Acronym or**  **Abbreviation** | **Definitions** |
| ID | Identity |
|  |  |
|  |  |

**8.2 References**