< MEAT >
Design Document

Authors:
Peng Cheng
Mengjiao Zeng
Sunghun Park

Group: 7

Contents

1 Introduction

The project is to create a unified management system for all of employees of the University of South Carolina. The system should be able to let authenticated users to create and manage meeting schedule inside the organization. They will have authority to create new meeting schedule and keep track of changes. All of the information will only be accessed by the authorized personnel. This system will largely mitigating employees' burden to organize and remember meetings.

1.1 Purpose

The purpose of this software design document is to provide a low-level description of the MEAT system, providing insight into the structure and design of each component. Topics covered include the following:

- Design overview and constraints
- Architectural design and Interfaces
- Database configuration and specification
- Class hierarchies and interactions, sequence behavior

1.2 System Overview

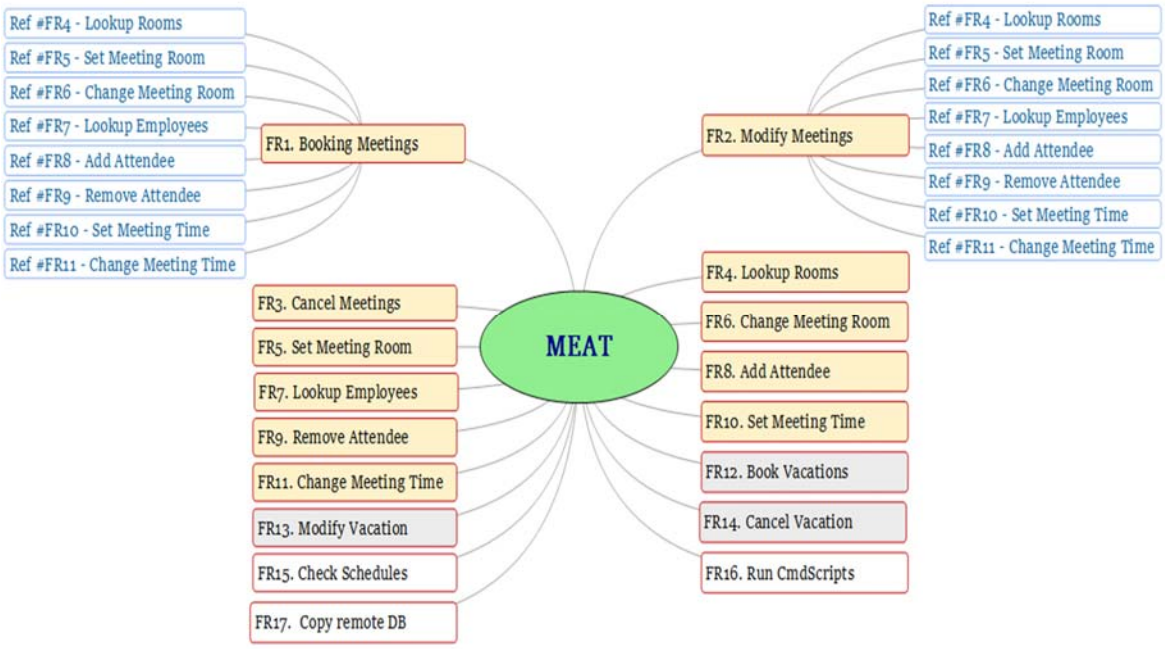
The purpose of this document is to present and provide a detailed description of the design specification of MEAT Software system which is designed to create, organize and record the meeting schedule for organizations.

1.3 Design Objectives

MEAT system provides complete required development functionality, which will enable to communicate with external master employee database and room database. It is runnable on the desktop environment of the customer's company. For the overall functions of the MEAT system:

- The system provides users to organize and book meeting schedules.
- The users enable to modify and cancel the existing meeting schedules.
- The system allows users to look up open rooms and attendees' schedules at the given time.
- The users are able to change times, room, attendees' information in the existing meeting schedules.
- The system provides users to book their vacation time or change their vacation time.
- The users enable to check all existing schedules for specific employee, or room, or universal all rooms at given time.
- The system supports automation by providing the functionality of accepting the action scripts file which contains a list of action paired with necessary input.
- The system provides the functionality of copying and storing data on local machine daily from remote master employees and rooms databases.

[Main Features Of MEAT]



1.4 References

- The first requirement document that the customer prepared.
- Online session of requirements elicitation with the customer in the moodle.
- Software requirement document (10.2.2016)
- Requirements Based Test Cases for <MEAT> (10.2.2016).

1.5 Definitions, Acronyms, and Abbreviations

- *MEAT:* The Meeting Engagement and Tracking System
- *User*: The users are company employees who will use the system
- *Meeting organizer*: An employee who organize meetings with coordinate conference rooms and assemble attendees.
- *Attendees*: Employees who are supposed to attend a meeting.

2 Design Overview

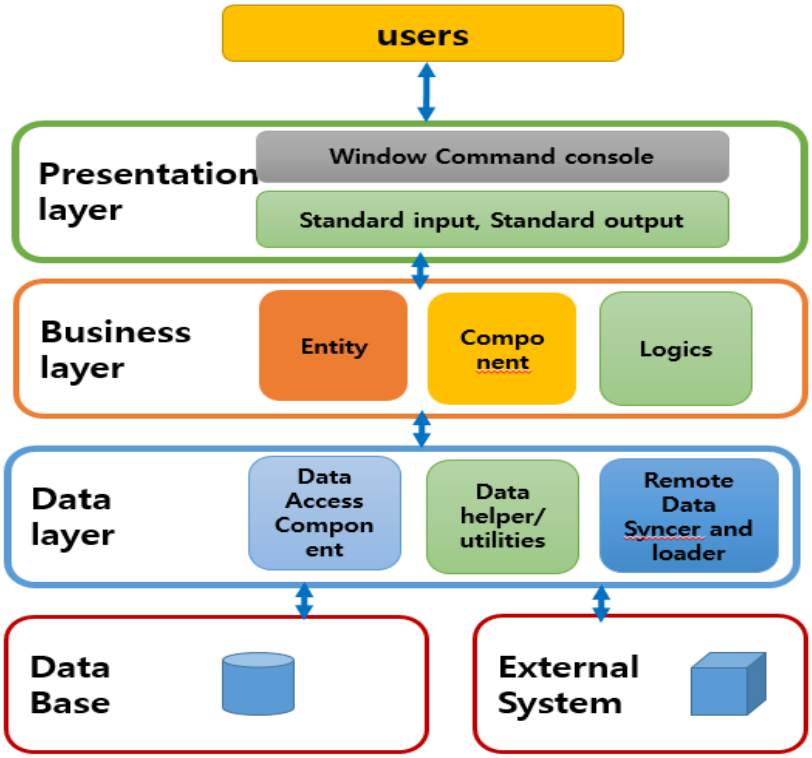
2.1 Introduction

- In the MEAT system design, the object-oriented design approach is adopted. MEAT''s internal structure is divided into two parts: server side and client side. Since the MEAT system is data-centric, the classes that handle the data will be isolated.

- The data storage of server side will use SQLite database. This database provide us light-weight database to be able to operate in local desktop PC.
- The global data structure of this system is best characterized by the database. The database structure shows the data involved in the application in its purest sense. The MEAT system user will never access this database directly; it will instead issue requests to the server.

2.2 System Architecture

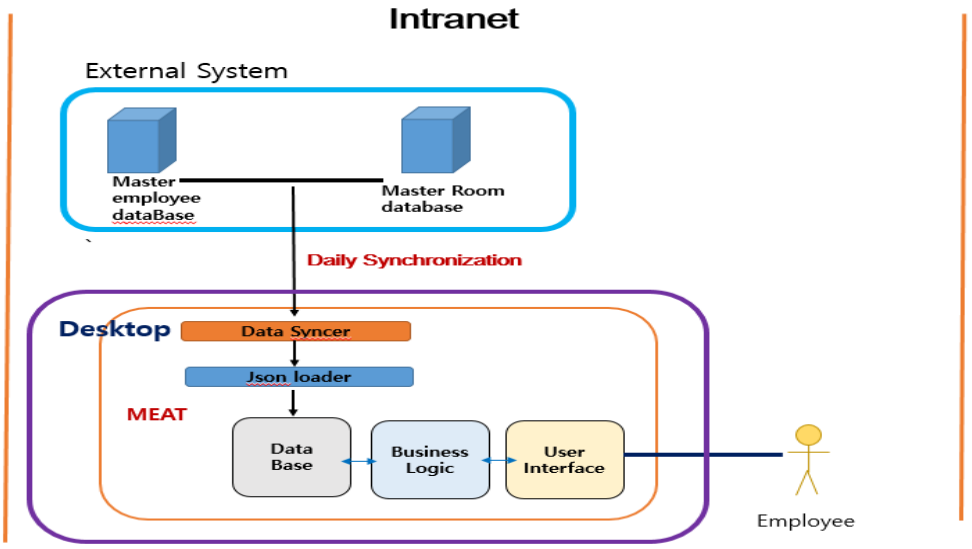
2.2.1 Logical View



√ The system consists of three major layers which are presentation layer, business layer, and data layer.

- ***Presentation layer*** *:* User interface capturing input and output from users.
- ***Business layer*** *:* Business logics are implemented in these layer.
- ***Data layer*** *:* This consists of three major component, data access control, utility classes, and data synchronizer and loader from external system.

2.2.2 Physical view



√ MEAT is executed in desktop environment with local database, and employees' data and room data is copied from remote external system.

- ***Data synchronizer*** *:* Connecting remote database and save its data records as a Json data format file.
- ***Data loader*** *:* Parsing the saved Json file and insert and update the data into local database.

2.3 Constraints and Assumptions

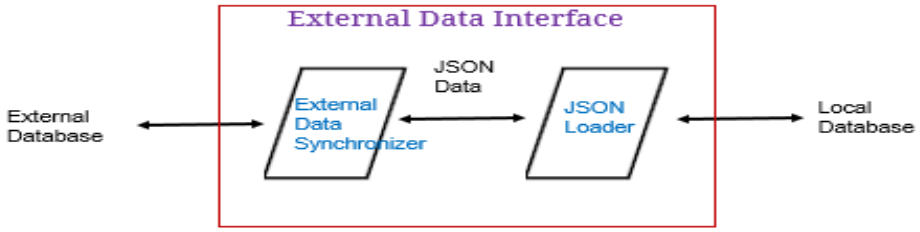
1. ***There is no different user role or role among the users.***
   - All users in the employee database have equal privileges. The system allows all users to look up, book, and modify all company schedules.
2. ***The system should be developed by JAVA language.***
   - To establish easy maintainability, all codes of the MEAT system is required to be programmed by JAVA language.
3. ***The MEAT system is used in desktop environment.***
   - The system should be executed on desktop environment of windows operating system.
4. ***The employees and master rooms database are stored in the external system.***
   - The data synchronizer should be developed.

3 Interfaces

3.1 System Interfaces

3.1.1 External Data Synchronizer Interface

· This interface is used to synchronize between external database (employees and master rooms) and local database and consists of two major components, **External Data Synchronizer** and Json **loader**.



· The remote database can be accessed by the **External Data Synchronizer** which is running in MEAT system through intranet network. The fetched data is saved as the form of Json data type and **the** Json **loader** will parse the saved data and store into the local database.

· The External Data Interface can be executed through the command line as below or the operating system scheduler.

- prompt > Java – jar MEAT.jar DBSYNC

[Employee's Json **data format]**

```
{
    "id": "nguy0621",
    "firstName": "Luan",
    "lastName": "Nguyen",
    "email": "nguy0621@cse.sc.edu",
    "title": "Senior Engineer",
    "totalVacationDays": 21
 }
```

[Room's Json **data format]**

```
{
    "id": "3D15",
    "building": "Swearingen",
    "floor": 3,
    "occupancy": 15
}
```

3.1.2 Scripting Commands interface

· This interface is used to facilitate automated meeting management by uploading and executing a script of commands to the MEAT system.

· Scripts can be executed through the command line by providing them as an argument to the MEAT system.

- prompt > ***java – jar MEAT.jar scriptfile.json***

✓ *More detailed information about available commands and specific format can be referred in the document "MEAT Scripting Interface".*

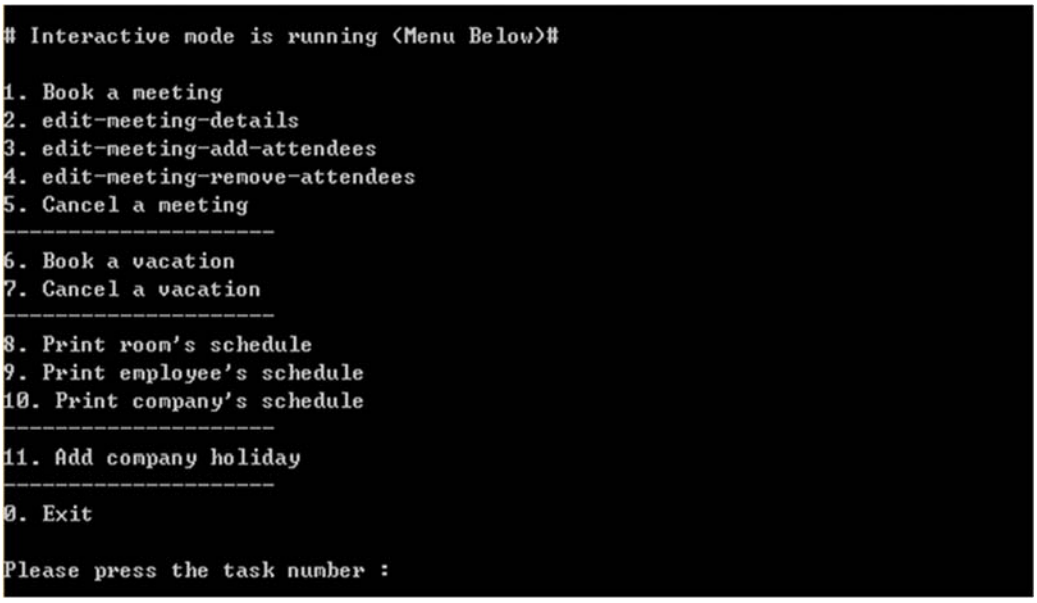
3.1.3 Interactive User interface

· Employees can use functionalities, such as booking a meet, by simply putting their choices according to the shown directives on the prompt or terminal screen.

· This interactive interface can be executed through command prompt or terminal by running without any argument when starting the MEAT system.

- prompt > ***java – jar MEAT.jar***

[Sample User-Interface for selecting a functionality]



4 Database Design

The relational database is chosen on the MEAT project because there is no need to use other characteristic databases such as hierarchical database, but simple entities relation. In addition, taking into account the scope of this project, we choose very lightweight relational database which is “sqlite”.

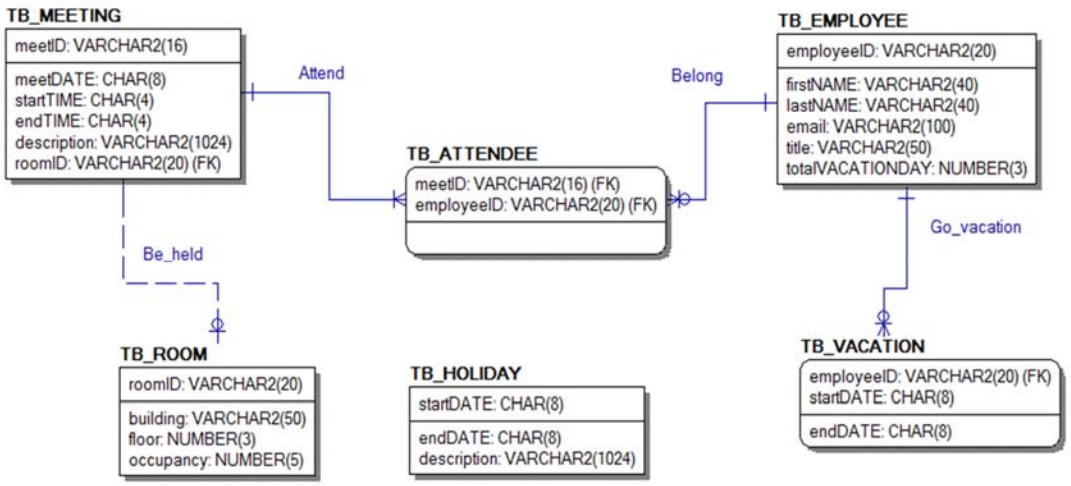
4.1 Database Overview

✓ *DBMS :* sqlite 3.15

✓ The MEAT database consists of five database tables support the system.

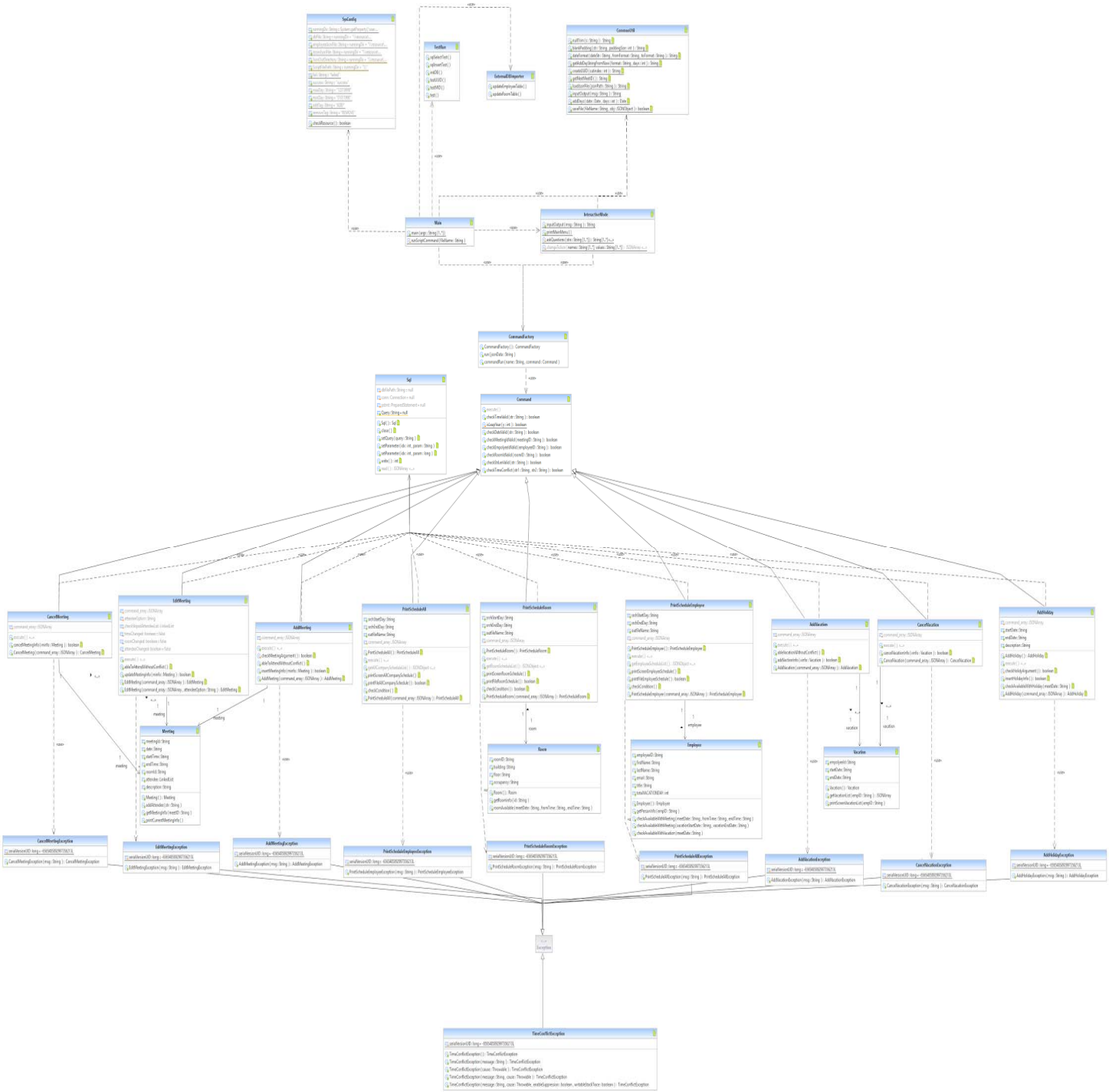
- TB_Meeting : store the information about scheduled meeting.
- TB_Attendee : store the information about attendees’ ID of scheduled meeting
- TB_Vacation : store the information about employee’s vacation.
- TB_Employee : store the information about employees from remote system.
- TB_Room : store the information about master rooms from remote system.
- TB_Holiday : store the information about the company’s holiday.

4.2 Physical E-R Diagram



5 Structural Design

5.1 Class Diagram (refer attached MEAT.png)



5.2 Classes in the client Subsystem

5.2.1 Class: Main

- Purpose: *the main entrance for the program. Get Json file input from the interface.*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.2.1.1 Attribute Descriptions (None)

5.2.1.2 Method Descriptions

1. Method: *main(String args[])*
Return Type: *void*
Parameters: *the name of input file*
Pre-condition: *low-level system is ready to run the program.*
Post-condition: *commandFactory access the Json file name successful.*
Methods called: *checkFileValid(String str), commandFactory.run(String str)*

Processing logic:
The main method just like client is very simple, just check the file is valid or not, and pass it the commandFactory class to process.

Test case 1: *input valid file name, output passed it to commandFactory successful.*
Test case 2*: input invalid file name, output the error message.*

2. Method: checkFileValid(String str)
Return Type: *boolean*
Parameters: *the name of input file*
Pre-condition: *main method has been invoked and the low-level system interface is working well.*
Post-condition: the return value return to main method successfully.
Methods called:

Processing logic:
The method check the file name is valid or not, another word Does there exist this file in the desktop.

Test case 1: *input valid file name, return true.*
Test case 2: *input invalid file name, return false*

5.2.2 Class: CommandFactory

- Purpose: *Get Json file name input from the main class, parse the Json file and call the correspond commands*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.2.2.1 Attribute Descriptions (None)

5.2.2.2 Method Descriptions

1. Method: *run(String str)*
Return Type: *void*
Parameters: *the name of input file*
Pre-condition: *parameter is not null, can parse the Json object well.*
Post-condition: each command can access Json file.
Methods called: *command.checkValid() command.execute()*

Processing logic:
get Json file name from main class, parse the file name, then call the correspond command with Json object.

Test case 1: *input: valid Json file; output: pass it to correspond command successful.*
Test case 2*: input: invalid Json file; output: the error message.*

5.3 Classes in the command Controller Subsystem

5.3.1 Class: Command

- Purpose: build the base class for commands
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.3.1.1 Attribute Descriptions (None)

5.3.1.2 Method Descriptions

1. Method: *checkValid(Json data)*
Return Type: *void*
Parameters: *Json data*
Pre-condition: *none*
Post-condition: *none*
Methods called: *none*

Processing logic:
This method must be override by child command classes.

2. Method: execute()
Return Type: *void*
Parameters: *Json data*
Pre-condition: *none*
Post-condition: *none*
Methods called: *none*

Processing logic:
This method must be override by child command classes.

3. Method: checkTimeValid()
Return Type: *boolean*
Parameters: *String time*

Pre-condition: *none*
Post-condition: *none*
Methods called: *none*

Processing logic:
check the input time(String) is valid or not. The time must represent in a certain format such as HH : MM or HH:MM

Test case 1: *input: valid time string; output: return true*
Test case 2*: input: invalid time string; output:return false.*

4. Method: checkDateValid()
Return Type: *boolean*
Parameters: *String date*
Pre-condition: *none*
Post-condition: *none*
Methods called: *none*

Processing logic:
check the input date(String) is valid or not. The date must represent in a certain format such as MM/DD/YY
Test case 1: *input: valid date string; output: return true*
Test case 2*: input: invalid date string; output: return false.*

5. Method: checkEmpolyeeIdValid()
Return Type: *boolean*
Parameters: *String empolyeeId*
Pre-condition: *none*
Post-condition: *none*
Methods called: *none*

Processing logic:
check the input employee Id(String) is valid or not. The employee id is integer format.

Test case 1: *input: valid employee Id string; output: return true*
Test case 2*: input: invalid employee Id string; output: return false.*

6. Method: checkRoomIdValid()
Return Type: *boolean*
Parameters: *String roomId*
Pre-condition: *none*
Post-condition: *none*
Methods called: *none*

Processing logic:
check the input employee Id(String) is valid or not. The room id must represent in a certain format, such as R108, R109.

Test case 1: *input: valid room Id string; output: return true*
Test case 2*: input: invalid room Id string; output: return false.*

7. Method: checkStrLenValid()
   Return Type: *boolean*
   Parameters: *String str*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *none*

   Processing logic:
   check the input string is valid or not. The length of string cannot exceed a certain length.

   Test case 1: *input: valid length string; output: return true*
   Test case 2*: input: invalid length string; output: return false.*

8. Method: checkTimeConflictValid()
   Return Type: *boolean*
   Parameters: *String time1, String time2*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *none*

   Processing logic:
   Compare two strings which represent time, return true if the second time after the first one. Otherwise return false.

   Test case 1: *input: second time after the first one; output: return true*
   Test case 2*: input: second time before the first one; output: return false.*

9. Method: checkIntervalValid()
   Return Type: *boolean*
   Parameters: *String[] interval1, String[] interval2*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *none*

   Processing logic:
   Compare two string arrays which represent time interval, return false if the two time-interval have overlap. Otherwise return true.

   Test case 1: *input: the two time-interval don't have overlap; output: return true*
   Test case 2*: input: the two time-interval have overlap; output: return false.*

10. Method: viewPrint()
    Return Type: *void*
    Parameters: *String str*
    Pre-condition: *none*
    Post-condition: *none*
    Methods called: *Messageout.run();*

    Processing logic:

Pass the parameter to Messageout class, Print out string message in a certain format.

5.3.2 Class: AddMeeting

- Purpose: To set up a new meeting in the meeting schedule
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.3.2.1 Attribute Descriptions

1. Attribute : meeting
   Type : Meeting
   Description : store the information will be store into database
   Constraints : should not be null, and access range should be private

5.3.2.2 Method Descriptions

1. Method: *execute()*
   Return Type: *none*
   Parameters: *Json data*
   Pre-condition: *none*
   Post-condition: *database add the meeting successfully*
   Methods called: *checkValid(), meeting.writeToSql()*

   Processing logic:
   In checkValid() method, the Json data has been checked for validity and information have been stored into meeting and vacation object. Call this method,
   1. the data store into meeting object
   2. this object will be written to database.

   Test case 1: *input: Json data; output: meeting object has been created successfully by Json data.*
   Test case 2*: input: null, output: throw exception.*

2. Method: *checkValid()*
   Return Type: *boolean*
   Parameters: *Json data*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *checkDateValid(String str), checkTimeValid(), checkIntevalValid(), checkRoomValid(), checkEmployeeIdValid(), checkStrLenValid(), new room();new employee(); room,readFromSql(); employee.readFromSql();*

   Processing logic:
   This method will check the Json data valid or not.
   1 we should parse the Json data, and check each one is valid by format requirement.
   2 check the meeting time-interval has not conflict with room time-interval, and the meeting time-interval has not conflict with employee's time-interval.
   3.If all requirements have been passed, return true.

   Test case 1: *input: valid Json data, return true.*
   Test case 2*: input time invalid Json data, return false.*

Test case 3*: input data invalid Json data, return false.*
Test case 4*: input room id invalid Json data, return false.*
Test case 5*: input employee id invalid Json data, return false.*
Test case 6*: input meeting time-interval conflict with room schedule, return false.*
Test case 7*: input meeting time-interval conflict with empolyee schedule, return false.*

5.3.3 Class: EditMeeting

- Purpose: edit a meeting from an exist meeting
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.3.1.1 Attribute Descriptions

1. Attribute : meeting
   Type : Meeting
   Description : store the information of meeting
   Constraints : should not be null, and access range should be private

5.3.3.2 Method Descriptions

1. Method: *execute()*
   Return Type: *none*
   Parameters: *String meetingId*
   Pre-condition: *the meeting exists in database.*
   Post-condition: *database update the meeting*
   Methods called: *checkValid(), meeting.writeToSql(), meeting.setDate() or meeting,setstartTime() or meeting.endTime() or meeting.setRoomId() or meeting.setAttendee()*

   Processing logic:
   In checkValid() method, the Json data has been checked for validity and information have been stored into meeting and vacation object. Call this method,
   1. update the object's field.
   2. the object write to database

   Test case 1: *input: valid data; output: meeting object has been updated successfully in database.*
   Test case 2*: input: null, output: throw exception.*

2. Method: *checkValid()*
   Return Type: *boolean*
   Parameters: *Json data*
   Pre-condition: *the meeting exists in database*
   Post-condition: *none*
   Methods called: *meeting.readFromSql(), checkDateValid(String str) or checkTimeValid() or checkIntevalValid(), checkRoomValid() or checkEmployeeIdValid() or checkStrLenValid()*

   Processing logic:
   This method will check the Json data valid or not. The exist meeting information has been stored into database.
   1. check the room id exist in the data base.
   2. parse the Json data, and check each one is valid by format requirement.

3. check the meeting time-interval has not conflict with room time-interval, and the meeting time-interval has not conflict with employee's time-interval.

4. If all requirements have been passed, return true.

Test case 1: *input: valid Json data, return true.*
Test case 2*: input time invalid, return false.*
Test case 3*: input data invalid, return false.*
Test case 4*: input room id invalid, return false.*
Test case 5*: input employee id invalid, return false.*
Test case 6*: input the length of string invalid, return false.*
Test case 7*: input meeting time-interval conflict with room schedule, return false.*
Test case 8*: input meeting time-interval conflict with empolyee schedule, return false.*

5.3.4 Class: cancelMeeting

- Purpose: cancel a meeting from an exist meeting
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.3.4.1 Attribute Descriptions (None)

5.3.4.2 Method Descriptions

1. Method: *execute()*
   Return Type: *none*
   Parameters: *String meetingId*
   Pre-condition: *the meeting exists in database.*
   Post-condition: *database has been update*
   Methods called: *Meeting.cancelMeeting();checkValid()*
   Processing logic:
   First, the Json data has been checked for validity. Call this method, the meeting data will be deleted from database/

   Test case 1: *input: meeting id; output: meeting data deleted in database.*
   Test case 2*: input: null, output: throw exception.*

2. Method: *checkValid()*
   Return Type: *boolean*
   Parameters: *String meetingId*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *new meeting();meeting,readFromSql();*

   Processing logic:
   1.This method will check the meeting can get from database or not.

   Test case 1: *input: valid meetingId, return true.*
   Test case 2*: input invalid meetingID,, return false.*

5.3.5 Class: AddVacation

- Purpose: To set up a new vacation in the empolyee schedule
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.3.5.1 Attribute Descriptions

Attribute : vacation
Type : Vacation
Description : store the information of vacation
Constraints : should not be null, and access range should be private

5.3.5.2 Method Descriptions

1. Method: *execute()*
   Return Type: *none*
   Parameters: *Json data*
   Pre-condition: *none*
   Post-condition: *database add the vacation successfully*
   Methods called: *checkValid(), vacation.writeToSql()*

   Processing logic:
   In checkValid() method, the Json data has been checked for validity and information have been stored into meeting and vacation object.. Call this method, the data will be store into vacation object, then this object will be written to database.

   Test case 1: *input: Json data; output: vacation object has been created successfully by Json data.*
   Test case 2*: input: null, output: throw exception.*

2. Method: *checkValid()*
   Return Type: *boolean*
   Parameters: *Json data*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *checkDateValid(String str), checkEmployeeIdValid(), checkStrLenValid(), new vacation();new employee(); vacation.readFromSql(); employee.readFromSql();*

   Processing logic:
   This method will check the Json data valid or not.
   1.check the vacation exist in database or not.
   2.we should parse the Json data, and check each one is valid by format requirement.
   3.check the vacation time-interval has not conflict with employee schedule.
   4.If all requirements have been passed, return true.

   Test case 1: *input: valid Json data, return true.*
   Test case 2*: input data invalid return false.*
   Test case 3*: input employee id invalid, return false.*
   Test case 4*: input the length of string invalid, return false.*
   Test case 4*: input vacation time-interval conflict with employee schedule, return false.*

5.3.6 Class: cancelVacation

- Purpose: cancel a vacation from an exist vacation
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.3.6.1 Attribute Descriptions(none)

5.3.6.2 Method Descriptions

1. Method: *execute()*
   Return Type: *none*
   Parameters: *String vacationgId*
   Pre-condition: *the vacation exists in database.*
   Post-condition: *database has been update*
   Methods called: *Vacation.cancalVacation(),checkValid()*
   Processing logic:
   First, the Json data has been checked for validity. Call this method, the vacation data will be deleted from database/

   Test case 1: *input: vacation id; output: vacation data deleted in database.*
   Test case 2*: input: null, output: throw exception.*

2. Method: *checkValid()*
   Return Type: *boolean*
   Parameters: *String VacationId*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *new vacation(), vacation,readFromSql();*

   Processing logic:
   This method will check the vacationId can get from database or not.

   Test case 1: *input: valid vacationId, return true.*
   Test case 2*: input invalid vacationID,, return false.*

5.3.7 Class: AddHoliday

- Purpose: To set up a new vacation for all employees
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.3.7.1 Attribute Descriptions

Attribute : vacationList
Type : Vacation[]
Description : store the information of vacation
Constraints : should not be null, and access range should be private

5.3.7.2 Method Descriptions

1. Method: *execute()*
   Return Type: *none*
   Parameters: *Json data*
   Pre-condition: *none*
   Post-condition: *database add the vacations successfully*
   Methods called: *vacation[i].writeToSql(), checkValid().*

   Processing logic:
   First, the Json data has been checked for validity. Call this method,

1. the data will be store into vacation object, then this object will be written to database.
2. Each employee will add a new vacation (it may be overlap with his/her old vacation).

Test case 1: *input: Json data; output: all vacation objects has been created successfully by Json data.*
Test case 2*: input: null, output: throw exception.*

2. Method: *checkValid()*
Return Type: *boolean*
Parameters: *Json data*
Pre-condition: *none*
Post-condition: *none*
Methods called: *checkDateValid(String str), checkStrLenValid(), meeting.readFromSql();*

Processing logic:
This method will check the Json data valid or not.
1 we should parse the Json data, and check each one is valid by format requirement.
2 check the vacation time-interval has not conflict with meeting schedule.
3If all requirements have been passed, return true.

Test case 1: *input valid data, return true.*
Test case 2*: input data invalid, return false.*
Test case 4*: input the length of string invalid, return false.*
Test case 4*: input vacation time-interval conflict with meeting schedule, return false.*

5.3.8 Class: PrintScheduleEmployee

- Purpose: print out the schedule for one employee
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.3.8.1 Attribute Descriptions

Attribute : employee
Type : Employee
Description : store the information of employee
Constraints : should not be null, and access range should be private

5.3.8.2 Method Descriptions

1. Method: *execute()*
Return Type: *none*
Parameters: *Json roomId*
Pre-condition: *employee id exist in database*
Post-condition: *none*
Methods called: *Message.out(), checkValid()*

Processing logic:
In checkValid() method, the Json data has been checked for validity and information have been stored into *employee* object. Call this method,
1.print out these data by Json format.

Test case 1: *input: employee id; output:* print out the schedule for this *employee*
Test case 2*: input: null, output: throw exception.*

2. Method: *checkValid()*
   Return Type: *boolean*
   Parameters: *Json roomid*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *checkroomIdValid(),Sql.read(),*

   Processing logic:
   This method will check the Json data valid or not.
   1. check the *employee* id exist in the data base.
   2. we should parse the Json data, and check each one is valid by format requirement.
   3. if all requirements have been passed, return true;

   Test case 1: *input: employee id valid, return true.*
   Test case 2*: input: employee id invalid, return false.*

5.3.9 Class: PrintScheduleRoom

- Purpose: print out the schedule for one room
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.3.9.1 Attribute Descriptions

Attribute : room
Type : Room
Description : store the information of room
Constraints : should not be null, and access range should be private

5.3.9.2 Method Descriptions

1. Method: *execute()*
   Return Type: *none*
   Parameters: *Json roomId*
   Pre-condition: *room id exist in database*
   Post-condition: *none*
   Methods called: *Message.out(), checkValid()*

   Processing logic:
   In checkValid() method, the Json data has been checked for validity and information have been stored into room object. Call this method,
   1.print out these data by Json format.

   Test case 1: *input: roomid ; output:* print out the schedule for this room
   Test case 2*: input: null, output: throw exception.*

2. Method: *checkValid()*
   Return Type: *boolean*
   Parameters: *Json roomid*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *checkroomIdValid(),Sql.read(),*

Processing logic:
This method will check the Json data valid or not.
1. check the room id exist in the data base.
2. we should parse the Json data, and check each one is valid by format requirement.
3. if all requirements have been passed, return true;

Test case 1: *input: room id valid, return true.*
Test case 2*: input: room id invalid, return false.*

5.3.10 Class: PrintScheduleAll

- Purpose: print out the schedule for one room
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.3.10.1 Attribute Descriptions

Attribute : meeting
Type : Meeting[]
Description : store the information of meetings
Constraints : should not be null, and access range should be private

Attribute : vacation
Type : Vacation[]
Description : store the information of vacations
Constraints : should not be null, and access range should be private

5.3.10.2 Method Descriptions

1. Method: *execute()*
   Return Type: *none*
   Parameters: *Json roomId*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *Message.out(), checkValid()*

   Processing logic:
   In checkValid() method, the Json data has been checked for validity and information have been stored into meeting and vacation object. Call this method,
   1.print out these data by Json format.

   Test case 1: *input: valida data ; output:* print out the schedule for this room
   Test case 2*: input: null, output: throw exception.*

2. Method: *checkValid()*
   Return Type: *boolean*
   Parameters: *Json roomid*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *checkroomIdValid(),Sql.read(),*

   Processing logic:
   This method will check the Json data valid or not.

2. we should parse the Json data, and check each one is valid by format requirement.
3. if all requirements have been passed, return true;

Test case 1: *input: data valid, return true.*
Test case 2*: input: data invalid, return false.*

5.4 Classes in the model Subsystem

5.4.1 Class: Meeting

- Purpose: the model object for meeting
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.4.1.1 Attribute Descriptions

Attributes : meetingId, data, startTime, endTime, roomId, attendee, description
Type : String
Description : store the information of each field.
Constraints : should not be null, and access range should be private

5.4.1.2 Method Descriptions

1. Method: *Meeting()*
   Return Type: *none*
   Parameters: *String[] args*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called:
   Processing logic: Constructor

2. Methods: *getMeetingId(), getData(), getStartTime(), getEndTime(), getRoomId(), getAttendee(), getDescription().*
   Return Type: *void*
   Parameters: *null*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called:
   Processing logic: get each field information.

3. Methods: *setMeetingId(), setData(), setStartTime(), setEndTime(), setRoomId(), setAttendee(), setDescription().*
   Return Type: *String*
   Parameters: *null*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called:
   Processing logic: set each field information.

4. Methods: *writeToSql()*
   Return Type: *boolean*
   Parameters: *null*
   Pre-condition: *none*

Post-condition: *none*
Methods called: *Sql.setquery() Sql.write().*
Processing logic: write meeting information to database according sql class.

Test case 1: *input: data valid, return true.*
Test case 2*: input: null, return false.*

5. Methods: *readFromSql()*
   Return Type: *boolean*
   Parameters: *null*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *Sql.setquery() Sql.read().*
   Processing logic: read meeting information from database according sql class.
   Test case 1: *input: data valid, return true.*
   Test case 2*: input: null, return false.*

5.4.2 Class: Vacation

- Purpose: the model object for vacation
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.4.2.1 Attribute Descriptions

Attributes : vacationId, employeeId, startDate, endDate, description
Type : String
Description : store the information of each field.
Constraints : should not be null, and access range should be private

5.4.2.2 Method Descriptions

1. Method: *Meeting()*
   Return Type: *none*
   Parameters: *String[] args*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called:
   Processing logic: Constructor

2. Methods: *getvacationId(), getemployeeId(), getStartDate(), getEndDate (),getDescription().*
   Return Type: *String*
   Parameters: *null*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called:
   Processing logic: get each field information.

3. Methods: *setvacationId(), setemployeeId(), setStartDate(), setEndDate (),setDescription().*
   Return Type: *void*
   Parameters: *null*
   Pre-condition: *none*

Post-condition: *none*
Methods called:
Processing logic: set each field information.

4. Methods: *writeToSql()*
Return Type: *boolean*
Parameters: *null*
Pre-condition: *none*
Post-condition: *none*
Methods called: *Sql.setquery() Sql.write().*
Processing logic: write vacation information to database according sql class.
Test case 1: *input: data valid, return true.*
Test case 2*: input: null, return false.*

5. Methods: *readFromSql()*
Return Type: *boolean*
Parameters: *null*
Pre-condition: *none*
Post-condition: *none*
Methods called: *Sql.setquery() Sql.read().*
Processing logic: read vacation information from database according sql class.
Test case 1: *input: data valid, return true.*
Test case 2*: input: null, return false.*

5.4.3 Class: room

- Purpose: the model object for room
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.4.1.1 Attribute Descriptions

Attributes : meetings
Type : Meeting[]
Description : store the information of each field.
Constraints : should not be null, and access range should be private

5.4.1.2 Method Descriptions

1. Method: *room()*
Return Type: *none*
Parameters: *String[] args*
Pre-condition: *none*
Post-condition: *none*
Methods called:
Processing logic: Constructor

2. Methods: *getMeeting()*
Return Type: *Meeting[]*
Parameters: *null*
Pre-condition: *none*
Post-condition: *none*
Methods called:
Processing logic: get each field information.

3. Methods: *setMeetings()*
   Return Type: *void*
   Parameters: *null*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called:
   Processing logic: set each field information.

4. Methods: *writeToSql()*
   Return Type: *boolean*
   Parameters: *null*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *Sql.setquery() Sql.write().*
   Processing logic: write room information to database according sql class.
   Test case 1: *input: data valid, return true.*
   Test case 2*: input: null, return false.*

5. Methods: *readFromSql()*
   Return Type: *boolean*
   Parameters: *null*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *Sql.setquery() Sql.read().*
   Processing logic: read room information from database according sql class.
   Test case 1: *input: data valid, return true.*
   Test case 2*: input: null, return false.*

5.4.4 Class: employee

- Purpose: the model object for room
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.4.4.1 Attribute Descriptions

Attributes : meetings
Type : Meeting[]
Description : store the information of each field.
Constraints : should not be null, and access range should be private

Attributes : vacations
Type : Vacation[]
Description : store the information of each field.
Constraints : should not be null, and access range should be private

5.4.4.2 Method Descriptions

1. Method: *employee ()*
   Return Type: *none*
   Parameters: *String[] args*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called:

Processing logic: Constructor

2. Methods: *getMeetings(), getVacations()*
   Return Type: *Meeting[] or Vacation[]*
   Parameters: *null*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called:
   Processing logic: get each field information.

3. Methods: *setMeetings(), setVacations()*
   Return Type: *void*
   Parameters: *null*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called:
   Processing logic: set each field information.

4. Methods: *writeToSql()*
   Return Type: *boolean*
   Parameters: *null*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *Sql.setquery() Sql.write().*
   Processing logic: write employee information to database according sql class.
   Test case 1: *input: data valid, return true.*
   Test case 2*: input: null, return false.*

5. Methods: *readFromSql()*
   Return Type: *boolean*
   Parameters: *null*
   Pre-condition: *none*
   Post-condition: *none*
   Methods called: *Sql.setquery() Sql.read().*
   Processing logic: read employee information from database according sql class.
   Test case 1: *input: data valid, return true.*
   Test case 2*: input: null, return false.*

5.4.5 Class: Sql

- Purpose: *store new data and fetch stored data using sql query using JDBC* .
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.4.5.1 Attribute Descriptions

1. Attribute: *dbFilePath (String)*
   Type : String
   Description : store db file path
   Constraints : access range should be private

2. Attribute : conn
   Type : Connection
   Description : Connection to database
   Constraints : should be closed when exits, and access range should be private

3. Attribute : pstmt
   Type : PreparedStatement
   Description : Query statement wrapper
   Constraints : should be closed when exits, and access range should be private

4. Attribute : Query
   Type : String
   Description : store query string to execute in database
   Constraints : access range should be private

5.4.5.2 Method Descriptions

1. Method: *setQuery(String)*
   Return Type: *void*
   Parameters: *parameter*
   Pre-condition: *none*
   Post-condition: none
   Methods called:

   Processing logic:
   Call the method with query string, then the parameter is stored in the member variable "queryString". .

   Test case 1: *invoke setQuery with "select 1 from dual ", set the query parameter into member variable (queryString) successfully.*
   Test case 2*: null query output the error message.*

2. Method: *write()*
   Return Type: *void*
   Parameters: *none*
   Pre-condition: *local database shall be running*
   Post-condition: none
   Methods called:

   Processing logic:
   After setting the queryString, the method load JDBC driver and configure connection information, and then execute the query, using the connection, finally save data into database..

   Test case 1: *input booking meeting query, save data into database successfully.*
   Test case 2*: malformed query, output the error message.*

3. Method: *read()*
   Return Type: *JSONArray*
   Parameters: *none*
   Pre-condition: *local database shall be running*
   Post-condition: none
   Methods called:

   Processing logic:

After setting the queryString, the method load JDBC driver and configure connection information, and then execute the query, using the connection, finally read data from database and transform the fetched recordset into JSONArray and return it.

Test case 1: *existing booking meeting select query, fetch data from database and return it as the form of resultList successfully.*
Test case 2*: malformed query, output the error message.*

5.5 Classes in the View Subsystem

5.5.1 Class: Messageout

- Purpose: *print out the message in a certain format*
- Constraints: *None*
- Persistent: *No (created at system initialization from other available data)*

5.5.1.1 Attribute Descriptions (None)

5.5.1.2 Method Descriptions

11. Method: *print(String args[])*
Return Type: *void*
Parameters: *String or Json*
Pre-condition: *none*
Post-condition: *none*
Methods called:

Processing logic:
Print out the information in a certain format.

5.6 Justification and Explanation

In the Structure Class Design, some design patterns have been implied to make the software high cohesion and low coupling.

In the class hierarchy design, The MVC pattern has been used to separate application's concerns, In the model subsystem, the model represents each object which contains data, the objects can update the data when it changes. In the View subsystem, its goal is to visualize the data of model. In the controller subsystem, it acts on both view and model, it controls the logical data flow and update the view. It separates the model and view. In this hierarchy, the coupling between the layers is lower, and code maintenance is easier.

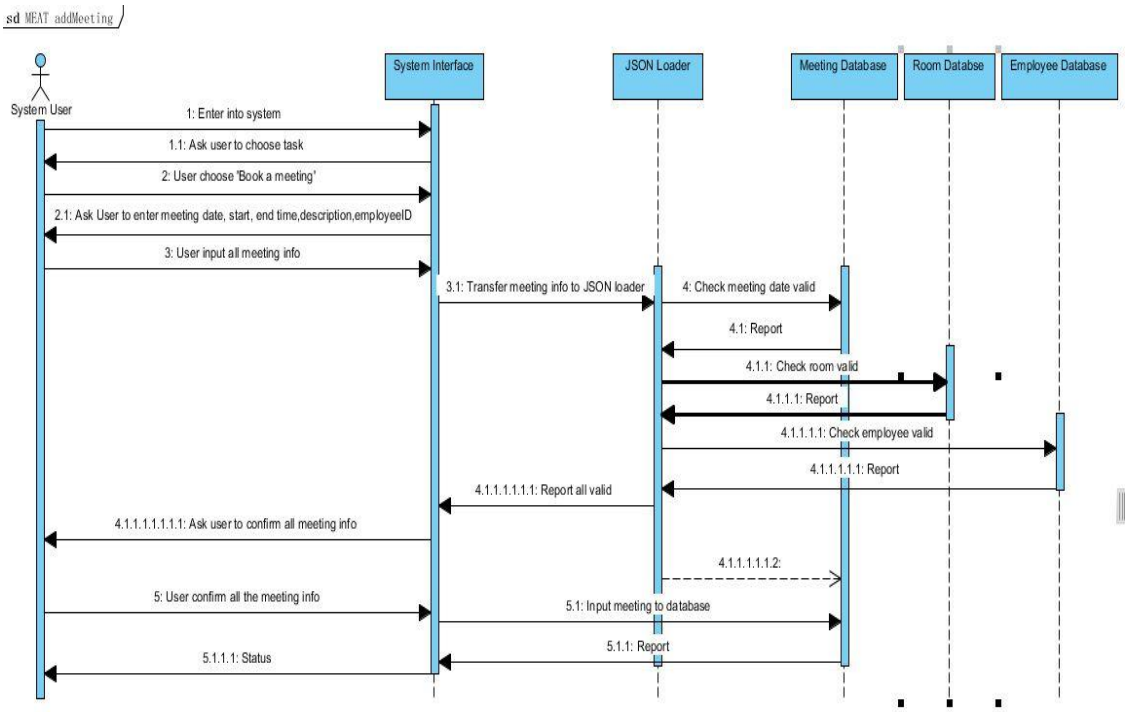
This project also applies factory pattern. The Command as base class will be called by CommandFactory, each command inherits the base Command and override some methods. The client will ask the CommandFactory to create each command. In this design pattern, the command creation logic will not expose to the client and newly created command will use the common super class. Just change CommandFactory code when a new one created.

Command pattern is good way to implement the business logic, the CommandFactory looks for the appropriate object which can process this command, then the correspond object will execute this command. Such as the Command is the base class, other commands are inherit the base Command. This pattern can make the object has high cohesion.

For the connection of database, we use the Data Access Object Pattern. It will separate low level data accessing API from high level business services. Like Sql class is the Date Access Object concrete class, the meeting, room, employee, vacation are model objects. This pattern can make the system has lower coupling.

6. Sequence Diagram

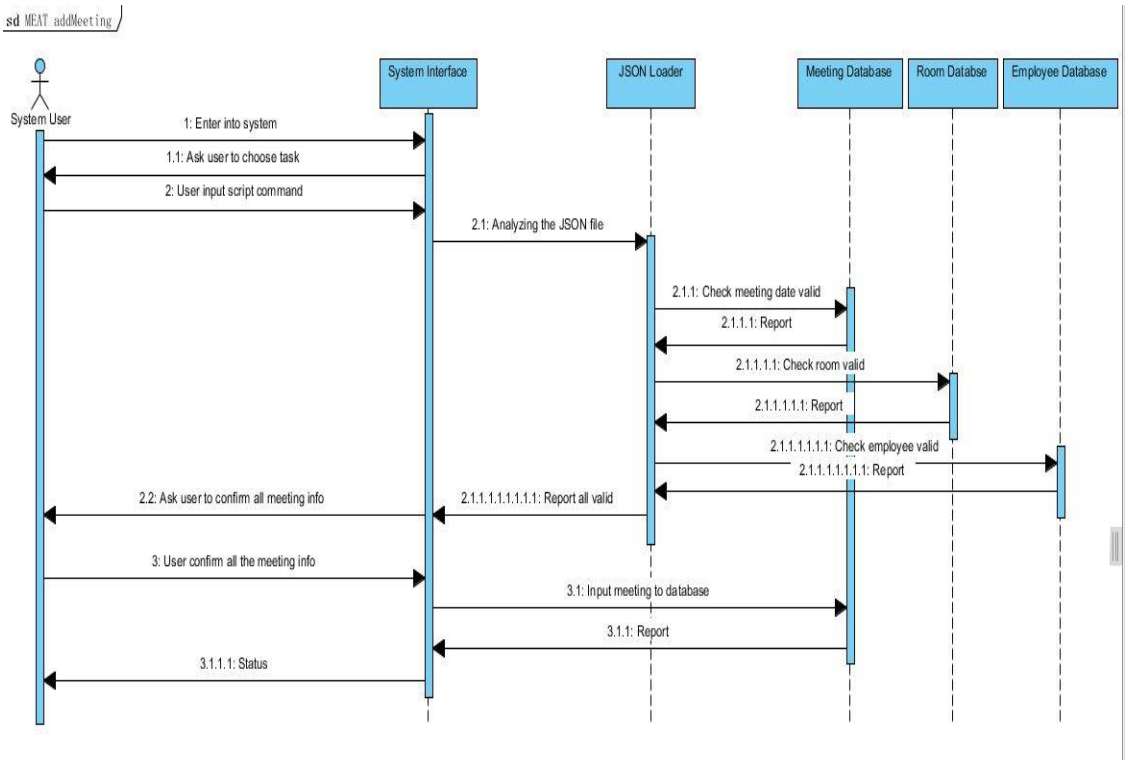
6.1 AddMeeting (interactive mode)



Description:

User enter into the system, the system interactive interface will ask user to choose one task. The user choose to add a meeting by input 1. System ask user to input meeting date, start time, end time, description and employee ID. User use command line to input these information. System receives these input and transfer these meeting information to JSON loader. JSON loader analyzing these data. Firstly, JSON ask Meeting database to check whether the input meeting date and meeting time is valid. The meeting database send back the status of the input meeting time. If it is valid, the JSON will transfer room data into room database to ask room database to check these input is valid and do not have conflict with the existing schedule. The room database report the feedback to the JSON loader. JSON loader then transfer attendee(employee ID) data into employee database to ask employee database to check these input is valid and do not have conflict with the existing schedule. After JSON receive all the positive feedback from database, JSON will send positive status to system interface. System get all of the required information for the meeting and ask user to confirm all the information. After receiving the confirmation, the system will input this meeting into meeting database. Finally, the database report the feedback to interface, and interface send the confirm status back to user.

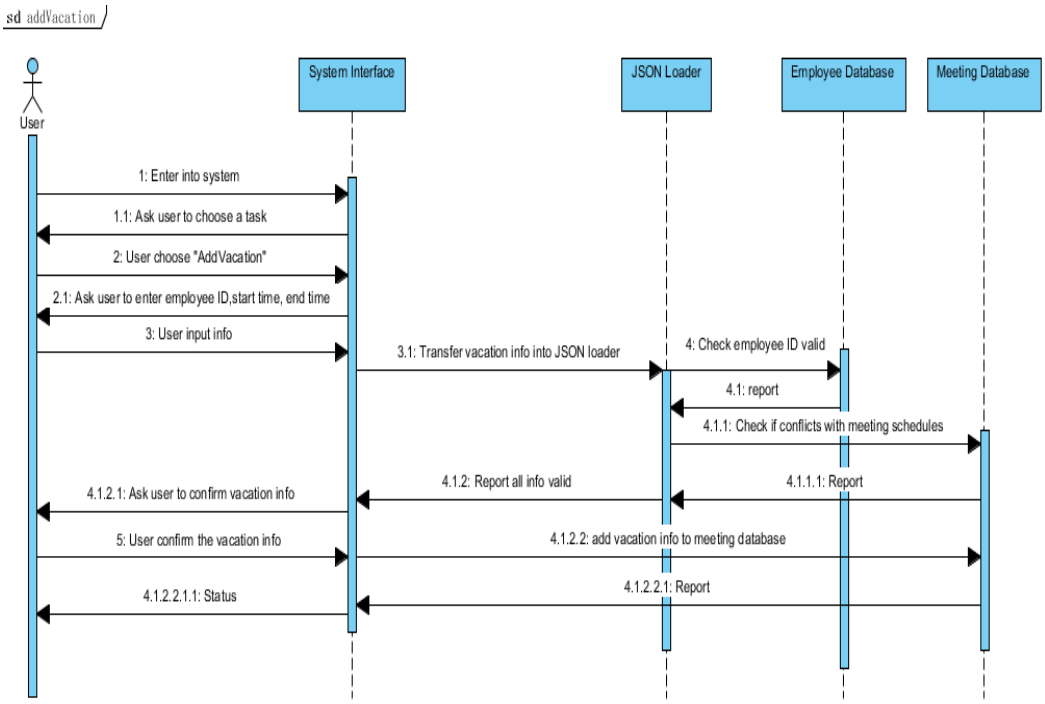
6.2 AddMeeting (Script running mode)



Description:

User enter into the system, the system interactive interface will ask user to choose one task. The user input script command line to read the .json file. System receives json file and transfer json file into JSON loader. JSON loader separate the JSON array into individually meeting info block. Then JSON loader analyzing these data. Next, JSON ask Meeting database to check whether the input meeting date and meeting time is valid. The meeting database send back the status of the input meeting time. If it is valid, the JSON will transfer room data into room database to ask room database to check these input is valid and do not have conflict with the existing schedule. The room database report the feedback to the JSON loader. JSON loader then transfer attendee(employee ID) data into employee database to ask employee database to check these input is valid and do not have conflict with the existing schedule. After JSON receive all the positive feedback from database, JSON will send positive status to system interface. System get all of the required information for the meeting and ask user to confirm all the information. After receiving the confirmation, the system will input this meeting into meeting database. Finally, the database report the feedback to interface, and interface send the confirm status back to user.

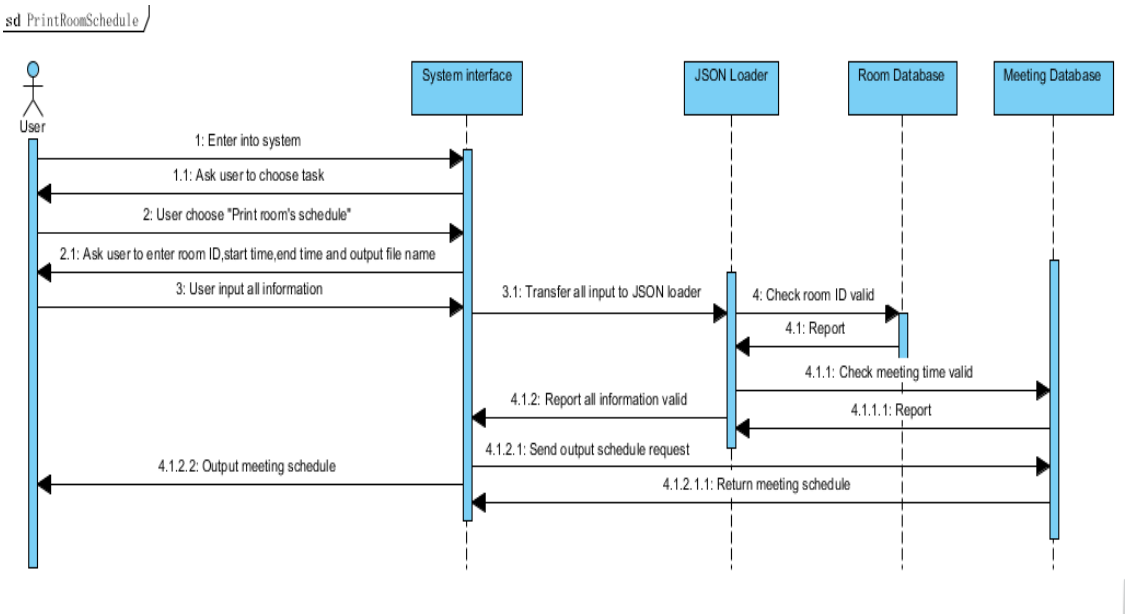
6.3 AddVacation



Description:

User enter into the system, the system interactive interface will ask user to choose one task. The user choose to add vacation by input. System ask user to input employee ID, start time and end time. user use command line to input these information. System receives these input and transfer these information to JSON loader. Then JSON loader analyzing these data. Next, JSON send employee information to employee database to check these input is valid in the employee database. The employee database report the feedback to the JSON loader. Then JSON ask meeting database to check the time input is valid and do not have conflict with the existing schedule. The meeting database report the feedback to the JSON loader. JSON report all information is valid to system interface. System interface get all of the required information for the vacation and ask user to confirm all the information. After receiving the confirmation, the system will input this meeting into meeting database. Finally, the database report the feedback to interface, and interface send the confirm status back to user.

6.4 PrintRoomSchedule



Description:

User enter into the system, the system interactive interface will ask user to choose one task. The user choose to print room schedule by input. System ask user to input room ID, start time, end time and output file name.. user use command line to input these information. System receives these input and transfer data to JSON loader. JSON loader analyzing data and ask room database to check these input is valid in the room database. Room database report the input room is valid. Then JSON ask meeting database to check time input is valid in the meeting database. The meeting database report the feedback to the JSON loader. JSON report all information is valid to system interface, and have meeting schedule in this time range. system will take the task and request meeting database to return meeting information. Finally, the database report the information to interface, and interface output meeting schedule to selected output file name..