Middlesex University London

CST2550

Software Engineering Management And Development

Deferral Coursework 1

Submission: Monday 10th August 2020, 23:55hrs

CAMPUS: Mauritius

CONTENTS

ABSTRACT

The project 'University Room Booking Management System' is a java-based application with MySQL used as database. This program aims at providing facility to store room details and bookings for a university. It includes a total of 8 features. This software provides facility to allow the user to add new bookings, update bookings, list all bookings, list all bookings for a specific lecturer, a specific room, a specific date, list available rooms and delete bookings. The functionalities implemented work well. The database was successfully implemented and runs perfectly. The database connection with the server works fine as well as the server-client connection. The server is multithreaded. Regular expressions validations were used especially user input validations were used extensively. However, the user interface used is console, as the GUI was still in development. Testing was done using the Junit framework. All the tests undergone were successfully passed. Evidence of testing and more details have been included in this report under the testing section.

INTRODUCTION

A university room booking system was built. All the project requirements were successfully met. The project was approached as follows:

1. Firstly, the ERD was constructed for the database.
2. Then the bottom up approach, Normalisation was used for the database design.
3. The database was created in MariaDB terminal.
4. After that, a server was created.
5. The server was connected the to the previously created database.
6. The Client- server connection was established.
7. The different functionalities were designed and implemented.
8. User input validations were inserted.
9. Testing was done manually and using Junit.

The program has multiple features and they are as follows:

- Add booking (allowing the user to book a new room in a specific time slot)
- Update booking (the details about existing bookings can be modified using this feature)
- LISTALL (providing the user with list of all the bookings in the database)
- LISTROOM (providing the user with a list of all bookings for a specified room)
- LISTLECTURER (providing the user with a list of all bookings for a lecturer)
- LISTDATE (providing the user with a list of all bookings for a date)
- DELETE (erasing a specified booking from the database)
- EXIT (terminating the program).

The report has 8 sections namely: the abstract, the introduction, the database design, the software design, testing and conclusion. The database design contains a written description of the database used, an entity relationship diagram and Normalisation from (UNF TO 3NF). The software design section consists of a written description, UML diagrams and GUI wireframes. The UML diagrams used are: use case diagram, activity diagram, class diagram and sequence diagram. The testing part has description and evidence of testing. The conclusion is a summary of the work completed along with the limitations of the program.

DATABASE DESIGN

Written Description

Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of data management systems. It helps in having an effective way of storing and retrieving data. There are several approaches which can be used to achieve this, however for this project, the Normalisation approached has been used. Normalisation is a bottom-up approach which starts with one big table which is later broken down to sets of tables in order to avoid redundancy and achieve a good schema.

A database has been designed to store room details and bookings for a University room booking system. The final form of the database is in the 3rd Normal Form consisting of 3 tables: Booking table, Lecturer table and TeachingRooms table.

A Top-down approach was also used for the relational model design in the form of an Entity-Relationship diagram (ERD). Contrary to Normalisation, the ERD starts from nothing and the model is built by addition of details such as entity, relationships and attributes.

NORMALISATION

UNORMALIZED FORM (UNF)

bookingId	**date**	**time**	**duration**	**reason**	**noAttendees**	**lecturerId**	**lecturerName**	**lecturerEmail**	**roomNumber**	**maximumCapacity**	**type**
B1	2020-02-14	12:00	2	Software	25	L5	Imrane	Imrane@gmail.com	C01	49	conference
B2	2020-03-25	13:00		Networking						49	conference
B3	2020-04-12	13:00	2	Covid-conference	24	L5	Imrane	Imrane@gmail.com	C02	49	conference
B4	2019-01-05	14:00	2	Tort-Law	22	L3	Smmayyah	Summayyah@gmail.com	H01	79	hall
B5	2019-01-17	14:00	3	Public-law	20	L3	Smmayyah	Summayyah@gmail.com	H02	79	hall
B6	2020-04-20	08:00	3	Music-class	21	L4	Sooltana	Sooltana@gmail.com	H01	79	hall
B7	2018-11-21	09:00	4	Violin-class	20	L4	Sooltana	Sooltana@gmail.com	H02	79	hall
B8	2020-01-02	09:00	1	Presentation	23	L2	Faiz	Faiz@gmail.com	C01	49	conference
B9	2020-10-06	14:00	2	Exam	25	L1	Hishaam	Hishaam@gmail.com	L01	39	lab
B10	2020-01-05	11:00	3	Test	27	L1	Hishaam	Hishaam@gmail.com	L01	39	lab

Table: University

FROM UNORMALIZED FORM TO FIRST NORMAL FORM (UNF TO 1NF)

- Removed repeating Groups

bookingId	date	time	duration	reason	noAttendees	lecturerId	lecturerName	roomNumber	lecturerEmail	maximumCapacity	type
B1	2020-02-14	12:00	2	Software	25	L5	Imrane	C01	Imrane@gmail.com	49	conference
B2	2020-03-25	13:00	2	Networking	25	L5	Imrane	L01	Imrane@gmail.com	49	conference
B3	2020-04-12	13:00	2	Covid-conference	24	L5	Imrane	C02	Imrane@gmail.com	49	conference
B4	2019-01-05	14:00	2	Tort-Law	22	L3	Smmayyah	H01	Summayyah@gmail.com	79	hall
B5	2019-01-17	14:00	3	Public-law	20	L3	Smmayyah	H02	Summayyah@gmail.com	79	hall
B6	2020-04-20	08:00	3	Music-class	21	L4	Sooltana	H01	Sooltana@gmail.com	79	hall
B7	2018-11-21	09:00	4	Violin-class	20	L4	Sooltana	H02	Sooltana@gmail.com	79	hall
B8	2020-01-02	09:00	1	Presentation	23	L2	Faiz	C01	Faiz@gmail.com	49	conference
B9	2020-10-06	14:00	2	Exam	25	L1	Hishaam	L01	Hishaam@gmail.com	39	lab
B10	2020-01-05	11:00	3	Test	27	L1	Hishaam	L01	Hishaam@gmail.com	39	lab

Table: University Flattened

Note: **bookingId** is nominated to act as key for the Unnormalized table.

FROM FIRST NORMAL FORM TO SECOND NORMAL FORM (1NF TO 2NF)

Functional Dependencies:

- **bookingId →** date, time, duration, reason, noAttendees, lecturerId, lecturerName, lecturerEmail, maximumCapacity, type. **(Full dependency)**
- **lecturerId →** lecturerName, lecturerEmail. **(Transitive dependency)**
- **roomNumber →** maximumCapacity, type. **(Transitive dependency)**

bookingId	**date**	**time**	**duration**	**reason**	**noAttendees**	**lecturerId**	**lecturerName**	**roomNumber**	**lecturerEmail**	**maximumCapacity**	**type**
B1	2020-02-14	12:00	2	Software	25	L5	Imrane	C01	Imrane@gmail.com	49	conference
B2	2020-03-25	13:00	2	Networking	25	L5	Imrane	L01	Imrane@gmail.com	49	conference
B3	2020-04-12	13:00	2	Covid-conference	24	L5	Imrane	C02	Imrane@gmail.com	49	conference
B4	2019-01-05	14:00	2	Tort-Law	22	L3	Smmayyah	H01	Summayyah@gmail.com	79	hall
B5	2019-01-17	14:00	3	Public-law	20	L3	Smmayyah	H02	Summayyah@gmail.com	79	hall
B6	2020-04-20	08:00	3	Music-class	21	L4	Sooltana	H01	Sooltana@gmail.com	79	hall
B7	2018-11-21	09:00	4	Violin-class	20	L4	Sooltana	H02	Sooltana@gmail.com	79	hall
B8	2020-01-02	09:00	1	Presentation	23	L2	Faiz	C01	Faiz@gmail.com	49	conference
B9	2020-10-06	14:00	2	Exam	25	L1	Hishaam	L01	Hishaam@gmail.com	39	lab
B10	2020-01-05	11:00	3	Test	27	L1	Hishaam	L01	Hishaam@gmail.com	39	lab

Since all the non-prime attributes are already fully dependent on the primary key **Booking number**, there is no Partial dependency to be removed. Therefore, the table **Database Flattened** remains unchanged from 1NF to 2NF.

FROM SECOND NORMAL FORM TO SECOND NORMAL FORM (2NF TO 3NF)

Removing Transitive dependencies

Functional Dependencies:
lecturerId → lecturerName, lecturerEmail. **(Full dependency)**

lecturerId	**lecturerName**	**lecturerEmail**
L5	Imrane	Imrane@gmail.com
L5	Imrane	Imrane@gmail.com
L5	Imrane	Imrane@gmail.com
L3	Smmayyah	Summayyah@gmail.com
L3	Smmayyah	Summayyah@gmail.com
L4	Sooltana	Sooltana@gmail.com
L4	Sooltana	Sooltana@gmail.com
L2	Faiz	Faiz@gmail.com
L1	Hishaam	Hishaam@gmail.com
L1	Hishaam	Hishaam@gmail.com

Table: Lecturer

Functional Dependencies:

roomNumber → maximumCapacity, type. **(Full dependency)**

roomNumber	maximumCapacity	type
C01	49	conference
L01	49	conference
C02	49	conference
H01	79	hall
H02	79	hall
H01	79	hall
H02	79	hall
C01	49	conference
L01	39	lab
L01	39	lab

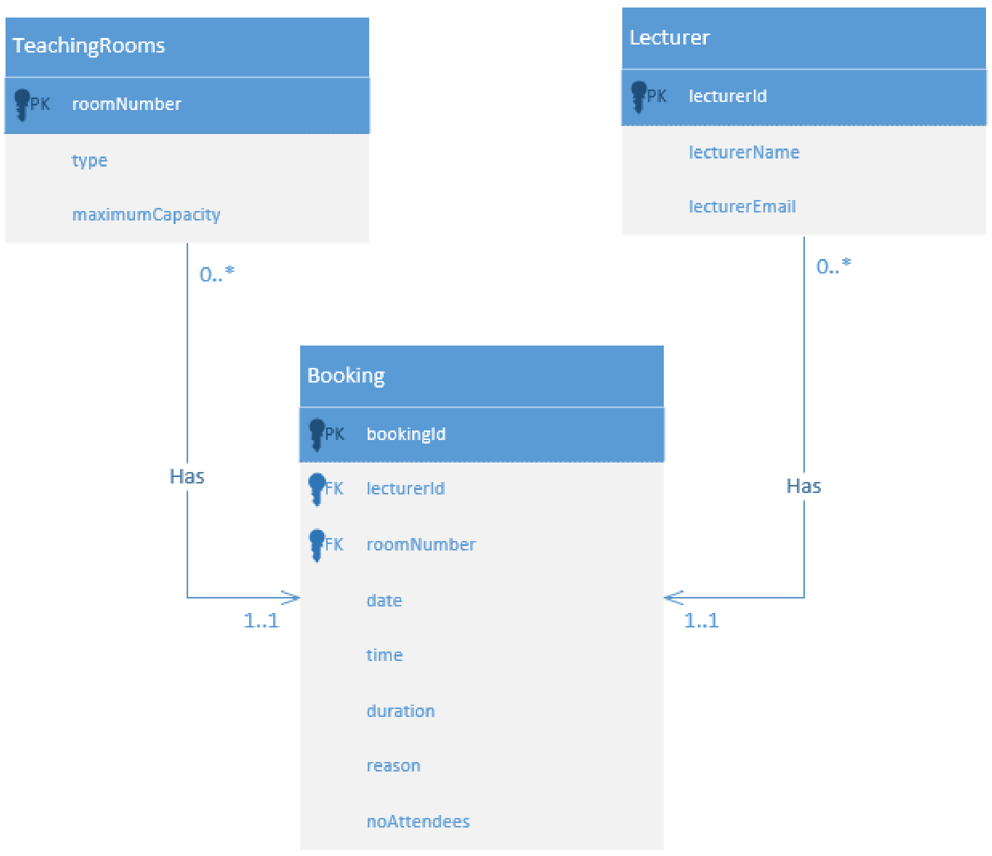
Table: TeachingRooms

Functional Dependencies:

bookingId → date, time, duration, reason, noAttendees, lecturerId, roomNumber.. **(Full dependency)**

bookingId	**date**	**time**	**duration**	**reason**	**noAttendees**	**lecturerId**	**roomNumber**
B1	2020-02-14	12:00	2	Software	25	L5	C01
B2	2020-03-25	13:00	2	Networking	25	L5	L01
B3	2020-04-12	13:00	2	Covid-conference	24	L5	C02
B4	2019-01-05	14:00	2	Tort-Law	22	L3	H01
B5	2019-01-17	14:00	3	Public-law	20	L3	H02
B6	2020-04-20	08:00	3	Music-class	21	L4	H01
B7	2018-11-21	09:00	4	Violin-class	20	L4	H02
B8	2020-01-02	09:00	1	Presentation	23	L2	C01
B9	2020-10-06	14:00	2	Exam	25	L1	L01
B10	2020-01-05	11:00	3	Test	27	L1	L01

Entity-Relationship Diagram (ERD)

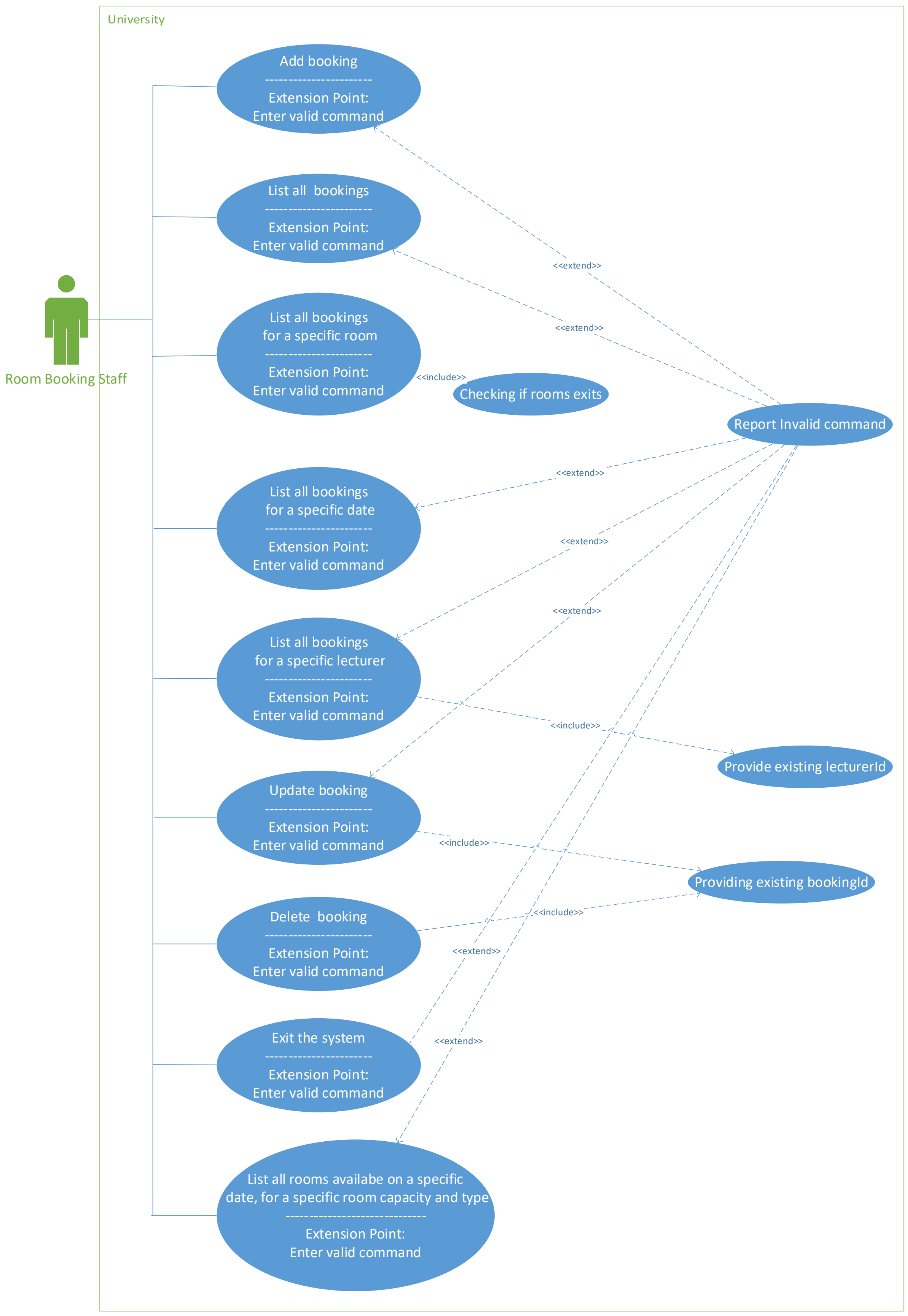


SOFTWARE DESIGN

Written description

The software design section helps give a better understanding of the structure of the program. This has been done using UML diagrams. This section includes Use-case diagram, Activity diagram, Class diagram and sequence diagram. The program is divided into 2 parts; the server and the client. The server program is the only one which has access to the SQL Database. The server run on a port (in this case, PORT:8000). A multi-threaded server was built that is several clients can use at the same time. The client can connect to the server using its host name and port number. The User at the client end requests data by writing commands in the console, data is sent to the server which reads and processes the requests accordingly. The processes data is then sent back to the client. Data are read and written using Object Input Streams and Output Streams.

USE CASE DIAGRAM



Functional Requirements

- To develop a system that will allows Teaching rooms of a particular type to be booked by lecturers on specific dates, for specific reasons and with a certain number of students.
- To allow Room booking Staff of the University to update existing bookings.
- To allow a list of all the bookings to be displayed.
- To allow a list of bookings for a specific date to be displayed.
- To allow a list of bookings for a specific room to be displayed.
- To allow a list of bookings for a particular lecturer to be displayed.
- To allow a list of available rooms of a specified type, on a particular date and of specified maximum capacity to be displayed.
- To the staff to delete particular bookings

Use case description for ADD booking.

No.	Actor	System
1.	The booking Staff enters the following add command: ‘ADD <bookingId> <lecturerId> <roomNumber> <date> <time> <duration> <reason> <noAttendees>’.	The system checks for valid command. If the command is invalid, an error message is prompted to the booking staff. Errors include: double booking, invalid input for each booking detail. If the command is valid, the system prompts a message for successful booking.

Use case description for UPDATE booking.

No.	Actor	System
1.	The booking Staff enters the following update command: ‘UPDATE <bookingId> <lecturerId> <roomNumber> <date> <time> <duration> <reason> <noAttendees>’.	The system checks for valid command. If the command is invalid, an error message is prompted to the booking staff. Errors include: booking does not exist, invalid input for each booking detail. If the command is valid, the system prompts a message for successful booking.

Use case description for LISTALL.

No.	Actor	System
1.	The booking Staff enters the following command: ‘LISTALL’.	The system checks if the command is valid. In case of invalid command, an error message is displayed to the user. Else, a list of all the bookings is retrieved from the database and displayed to the User.

Use case description for LISTROOM.

No.	Actor	System
1.	The booking Staff enters the following command: ‘LISTROOM <roomNumber>’.	The system checks if the command is valid. In case of invalid command, an error message is displayed to the user. Errors include invalid roomNumber, invalid command. Else, a list of all the bookings containing the specified roomNumber is retrieved from the database and displayed to the User.

Use case description for LISTDATE.

No.	Actor	System
1.	The booking Staff enters the following command: ‘LISTDATE <date>’.	The system checks if the command is valid. In case of invalid command, an error message is displayed to the user. Errors include invalid date, invalid command. Else, a list of all the bookings containing the specified date is retrieved from the database and displayed to the User.

Use case description for LISTLECTURER

No.	Actor	System
1.	The booking Staff enters the following command: ‘LISTLECTURER <lecturerId>’.	The system checks if the command is valid. In case of invalid command, an error message is displayed to the user. Errors include invalid lecturerId, invalid command. Else, a list of all the bookings containing the specified lecturerId is retrieved from the database and displayed to the User.

Use case description for LISTAVAILABLE

No.	Actor	System
1.	The booking Staff enters the following command: ‘LISTAVAILABLE <date> <maximumCapacity> <type>’.	The system checks if the command is valid. In case of invalid command, an error message is displayed to the user. Errors include: invalid command, invalid details for date, maximumCapacity and type. Else, a list of all the rooms containing the specified details is retrieved from the database and displayed to the User.

Use case description for DELETE.

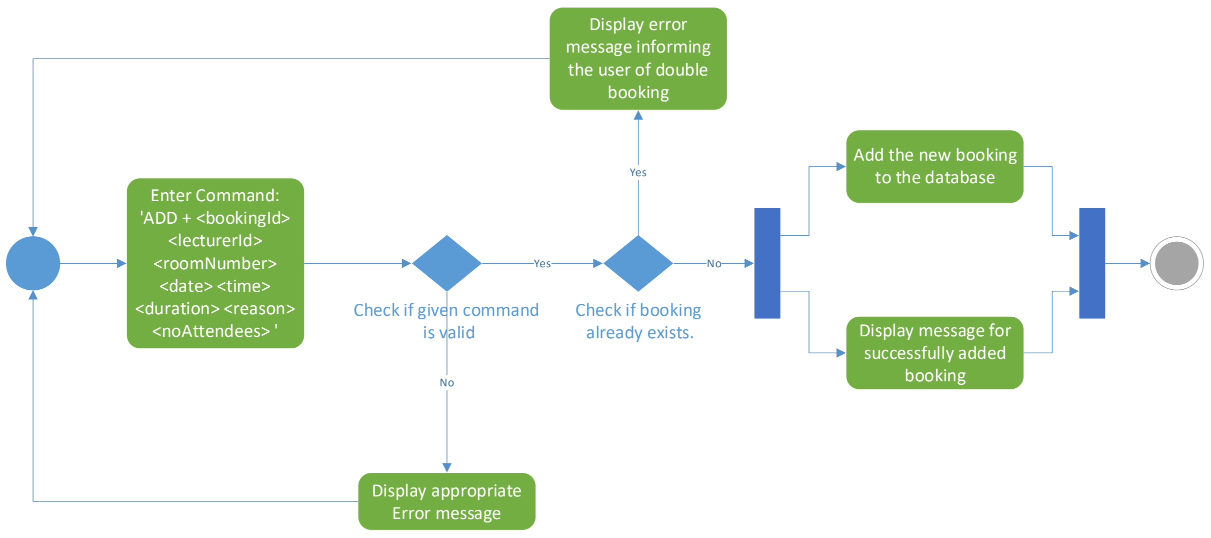
No.	Actor	System
1.	The booking Staff enters the following command: ‘DELETE <bookingId>’.	The system checks if the command is valid. In case of invalid command, an error message is displayed to the user. Errors include: invalid command and invalid bookingId. Else if command is valid, a list of all the rooms containing the specified details is retrieved from the database and displayed to the User.

Use case description for EXIT

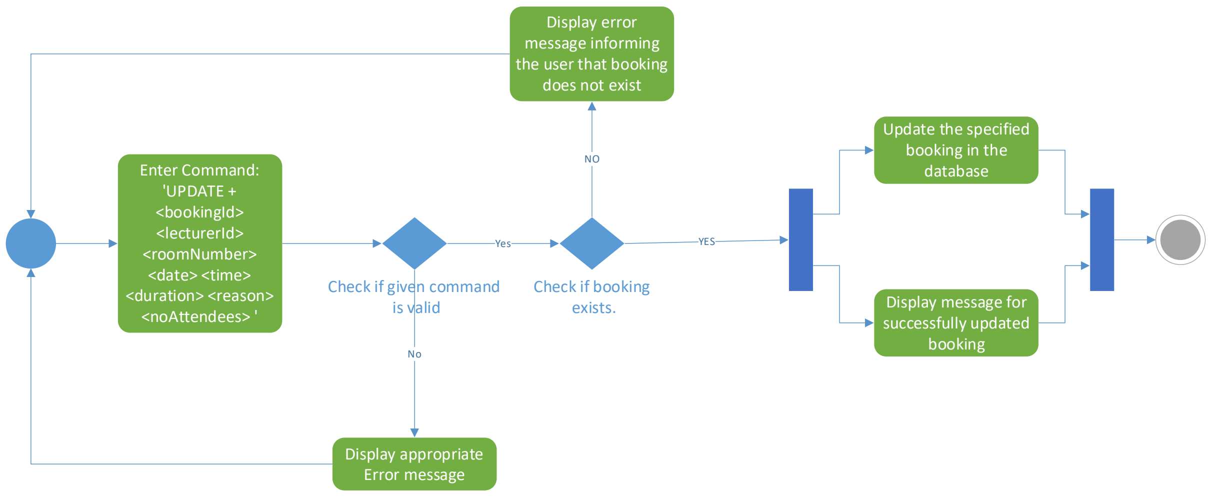
No.	Actor	System
1.	The booking Staff enters the following command: ‘EXIT’.	The system checks if the command is valid. In case of invalid command, an error message is displayed to the user. If the command is valid, a goodbye message is displayed to the user and ther program is termintated.

ACTIVITY DIAGRAMS

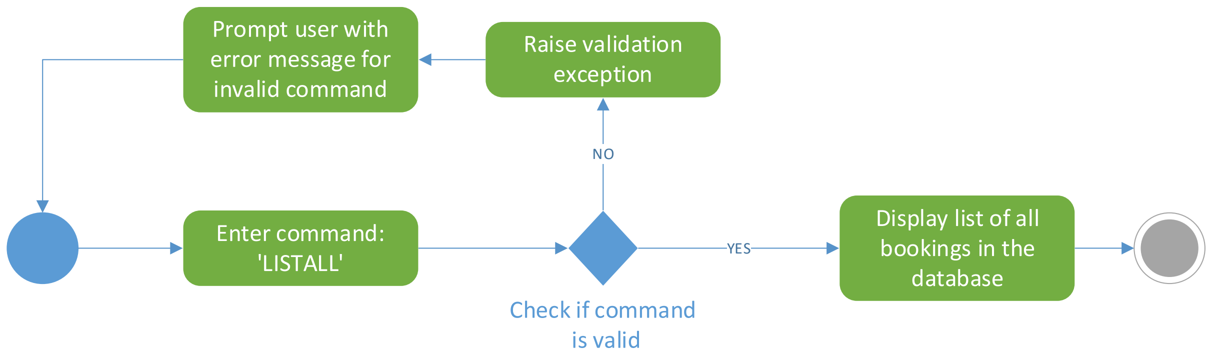
ADD BOOKING ACTIVITY DIAGRAM


Display error message informing the user of double booking
Yes
Add the new booking to the database
Enter Command: 'ADD + <bookingId> <lecturerId> <roomNumber> <date> <time> <duration> <reason> <noAttendees> '
Yes
No
Check if given command is valid
Check if booking already exists.
Display message for successfully added booking
No
Display appropriate Error message

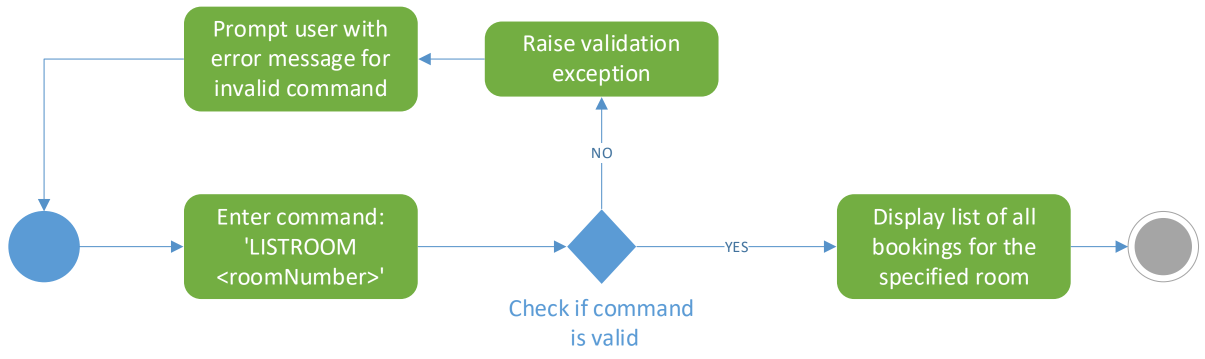
Update Booking Activity diagram


Display error message informing the user that booking does not exist
Update the specified booking in the database
NO
Enter Command: 'UPDATE + <bookingId> <lecturerId> <roomNumber> <date> <time> <duration> <reason> <noAttendees> '
Yes
YES
Check if given command is valid
Check if booking exists.
Display message for successfully updated booking
No
Display appropriate Error message

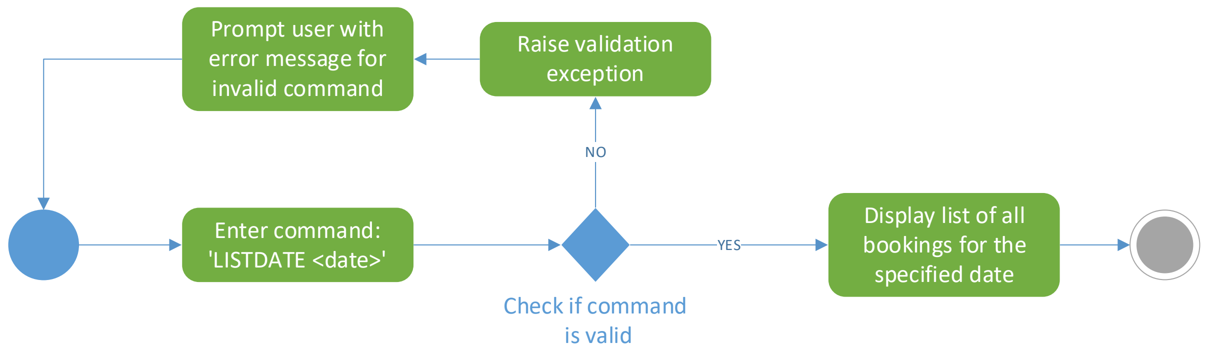
LISTALL ACTIVITY DIAGRAM


Prompt user with error message for invalid command
Raise validation exception
NO
Enter command: 'LISTALL'
YES
Display list of all bookings in the database
Check if command is valid

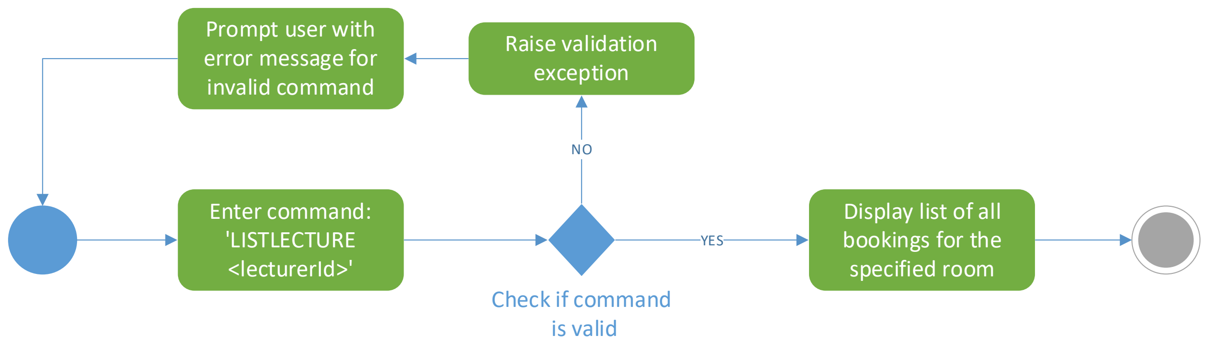
LISTROOM ACTIVITY DIAGRAM


Prompt user with error message for invalid command
Raise validation exception
NO
Enter command: 'LISTROOM <roomNumber>'
YES
Display list of all bookings for the specified room
Check if command is valid

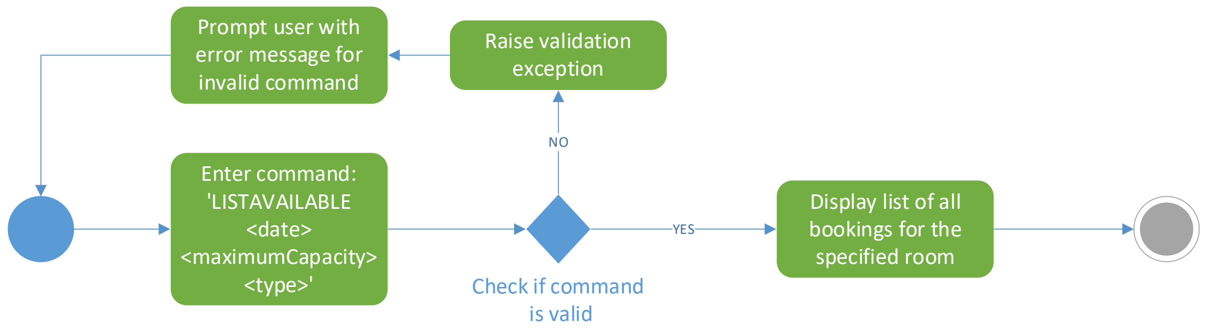
LISTDATE ACTIVITY DIAGRAM


Prompt user with error message for invalid command
Raise validation exception
NO
Enter command: 'LISTDATE <date>'
YES
Display list of all bookings for the specified date
Check if command is valid

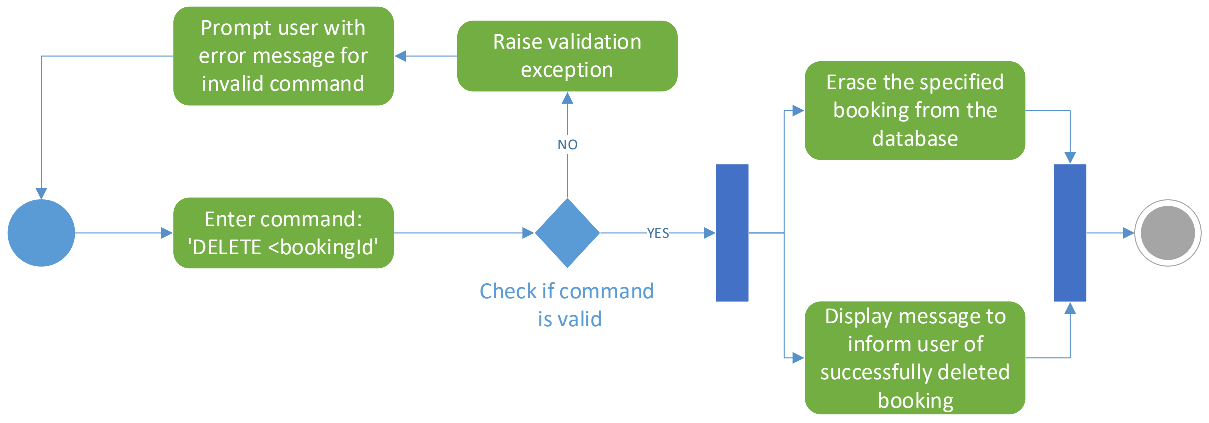
LISTLECTURER ACTIVITY DIAGRAM


Prompt user with error message for invalid command
Raise validation exception
NO
Enter command: 'LISTLECTURE <lecturerId>'
YES
Display list of all bookings for the specified room
Check if command is valid

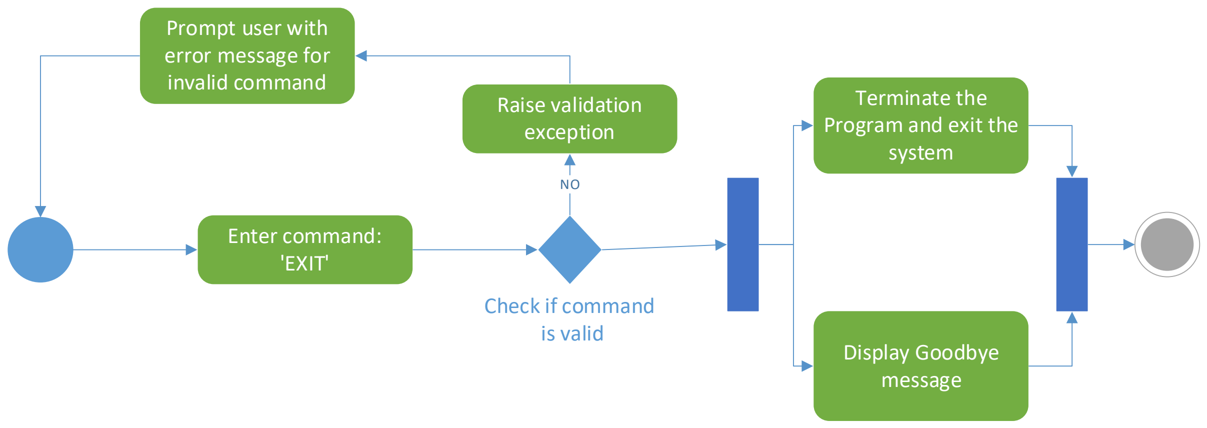
LISTAVAILABLE ACTIVITY DIAGRAM


Prompt user with error message for invalid command
Raise validation exception
NO
Enter command: 'LISTAVAILABLE <date> <maximumCapacity> <type>'
YES
Display list of all bookings for the specified room
Check if command is valid

DELETE BOOKING ACTIVITY DIAGRAM

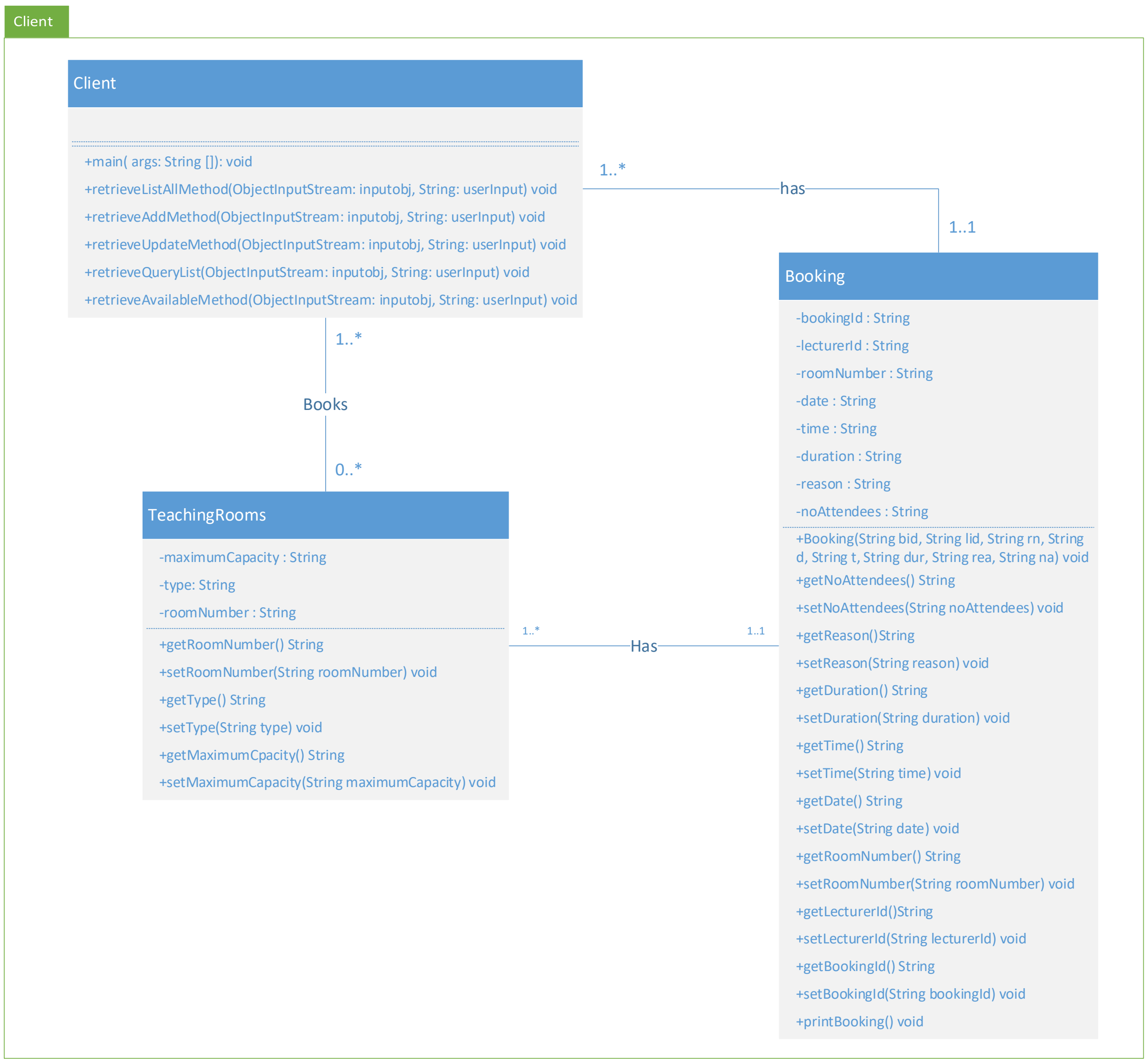

Prompt user with error message for invalid command
Raise validation exception
NO
Erase the specified booking from the database
Enter command: 'DELETE <bookingId'
YES
Check if command is valid
Display message to inform user of successfully deleted booking

EXIT SYSTEM

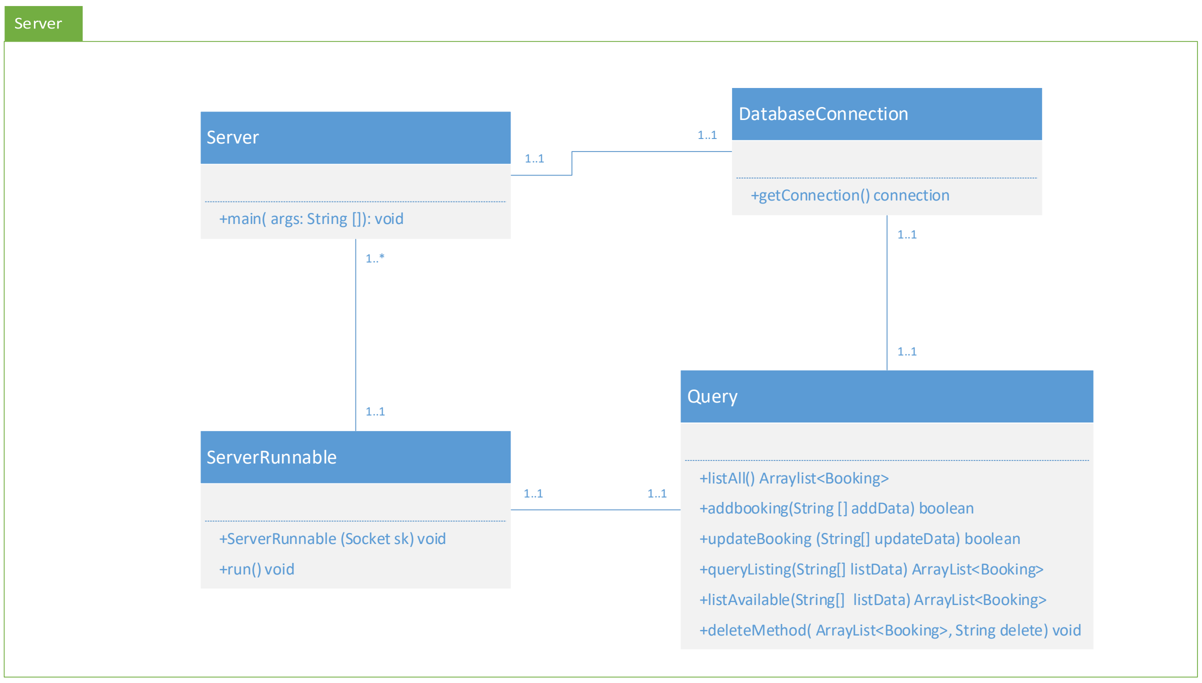

Prompt user with error message for invalid command
Raise validation exception
NO
Enter command: 'EXIT'
Check if command is valid
Terminate the Program and exit the system
Display Goodbye message

CLASS DIAGRAMS

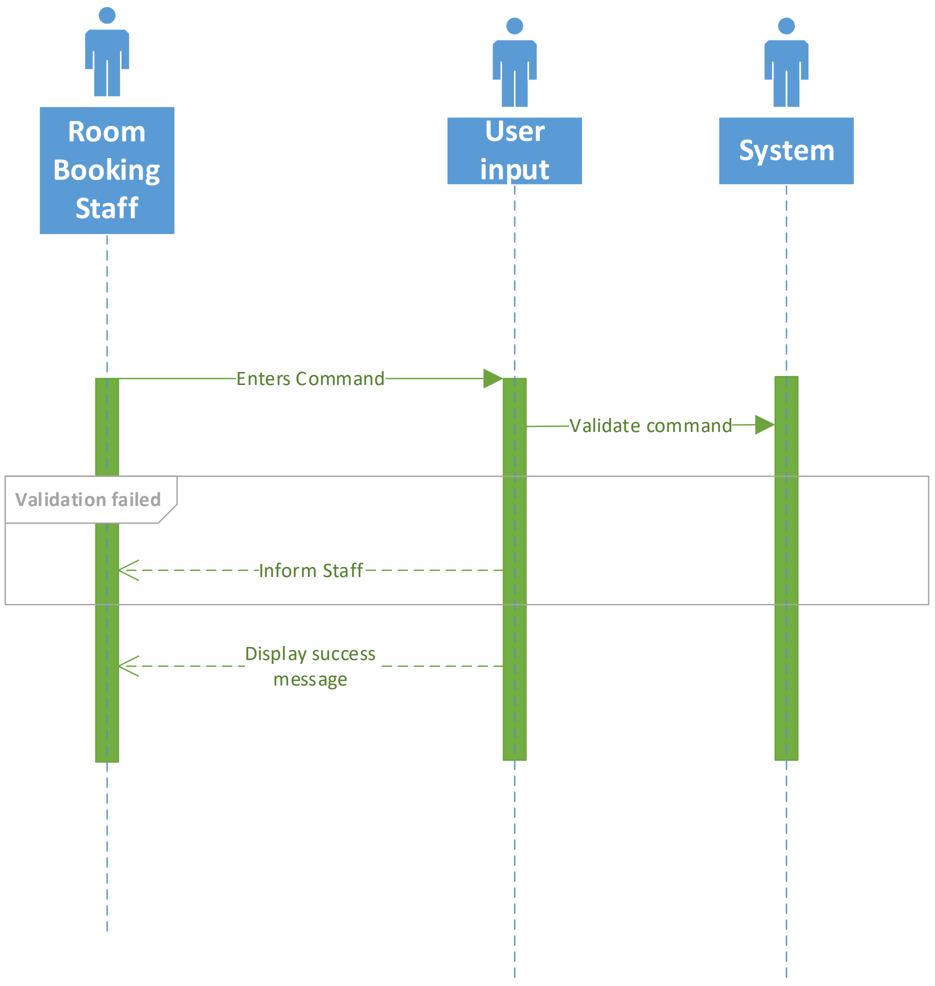
Client


Client
Client
+main(args: String []): void
+retrieveListAllMethod(ObjectInputStream: inputobj, String: userInput) void
+retrieveAddMethod(ObjectInputStream: inputobj, String: userInput) void
+retrieveUpdateMethod(ObjectInputStream: inputobj, String: userInput) void
+retrieveQueryList(ObjectInputStream: inputobj, String: userInput) void
+retrieveAvailableMethod(ObjectInputStream: inputobj, String: userInput) void
1..*
has
1..1
Booking
-bookingId : String
-lecturerId : String
-roomNumber : String
-date : String
-time : String
-duration : String
-reason : String
-noAttendees : String
+Booking(String bid, String lid, String rn, String d, String t, String dur, String rea, String na) void
+getNoAttendees() String
+setNoAttendees(String noAttendees) void
+getReason()String
+setReason(String reason) void
+getDuration() String
+setDuration(String duration) void
+getTime() String
+setTime(String time) void
+getDate() String
+setDate(String date) void
+getRoomNumber() String
+setRoomNumber(String roomNumber) void
+getLecturerId()String
+setLecturerId(String lecturerId) void
+getBookingId() String
+setBookingId(String bookingId) void
+printBooking() void
1..*
Books
0..*
TeachingRooms
-maximumCapacity : String
-type: String
-roomNumber : String
+getRoomNumber() String
+setRoomNumber(String roomNumber) void
+getType() String
+setType(String type) void
+getMaximumCpacity() String
+setMaximumCapacity(String maximumCapacity) void
1..*
Has
1..1

Server

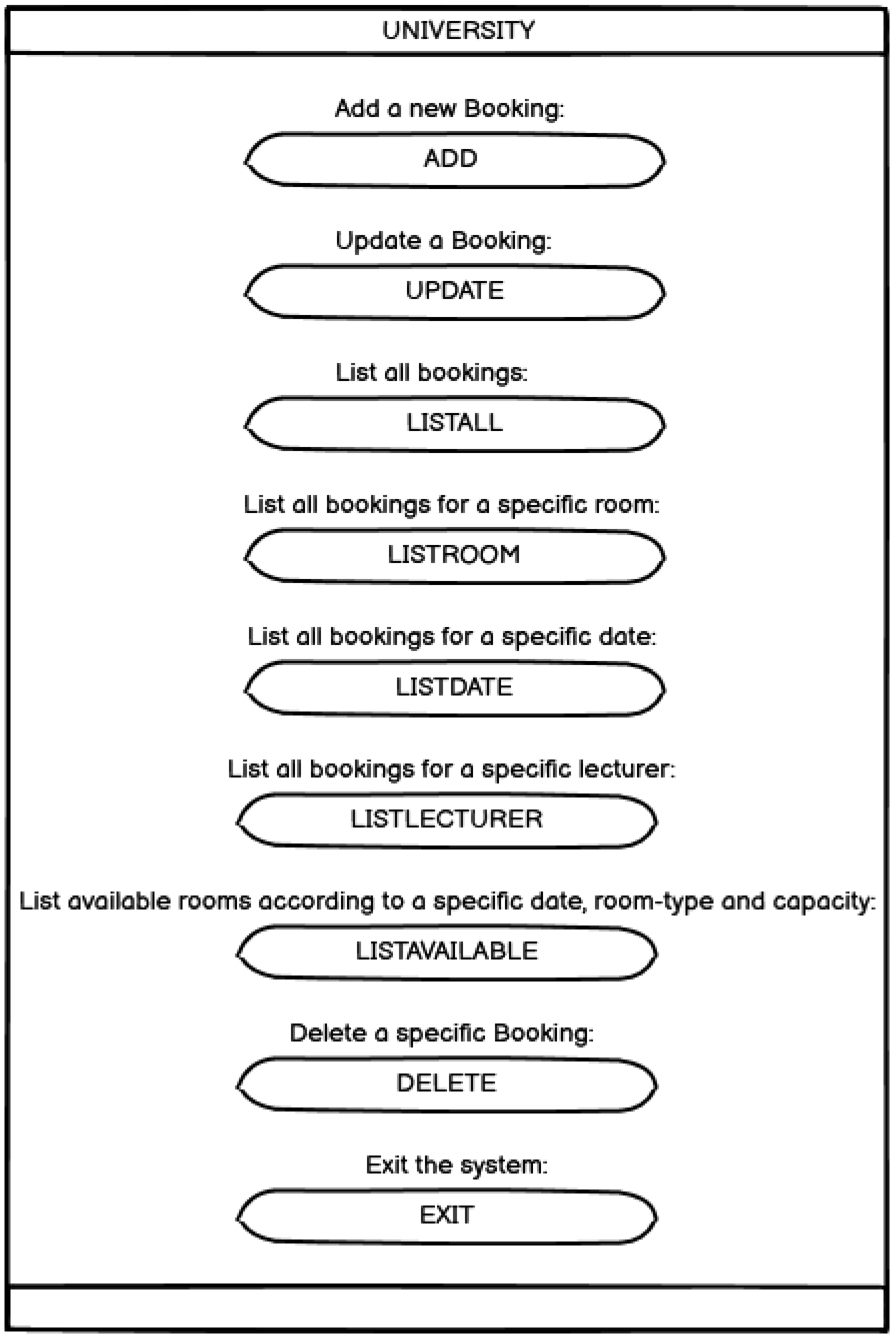

Server
Server
+main(args: String []): void
DatabaseConnection
+getConnection() connection
ServerRunnable
+ServerRunnable (Socket sk) void
+run() void
Query
+listAll() Arraylist<Booking>
+addbooking(String [] addData) boolean
+updateBooking (String[] updateData) boolean
+queryListing(String[] listData) ArrayList<Booking>
+listAvailable(String[] listData) ArrayList<Booking>
+deleteMethod(ArrayList<Booking>, String delete) void
1..1
1..1
1..*
1..1
1..1
1..1
1..1
1..1

SEQUECE DIAGRAM



GUI Mockups

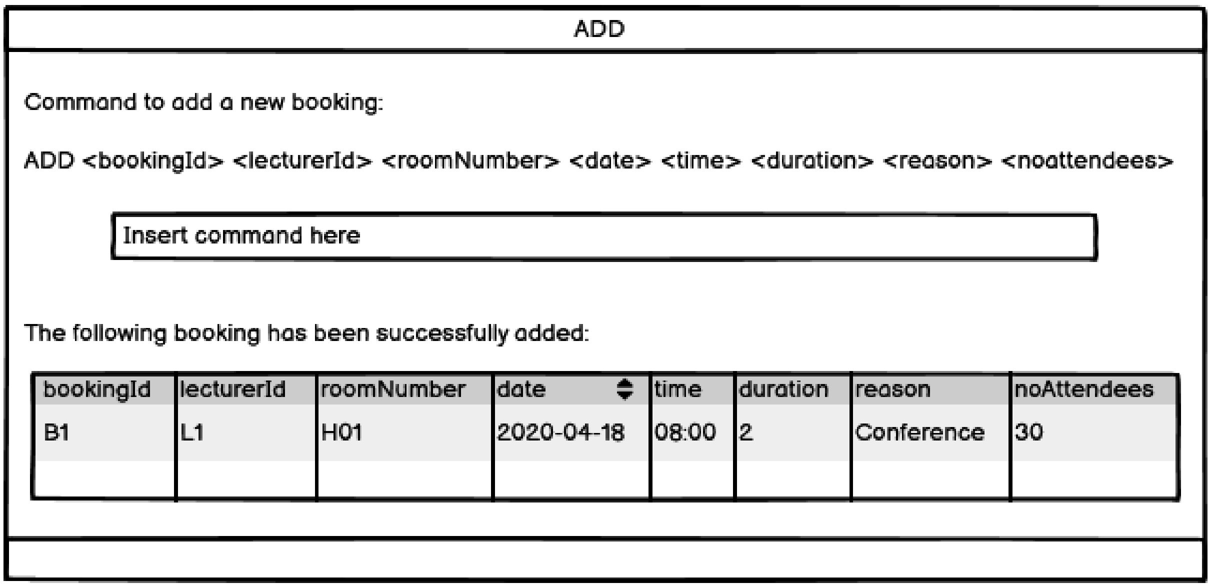
HOME


UNIVERSITY
Add a new Booking:
ADD
Update a Booking:
UPDATE
List all bookings:
LISTALL
List all bookings for a specific room:
LISTROOM
List all bookings for a specific date:
LISTDATE
List all bookings for a specific lecturer:
LISTLECTURER
List available rooms according to a specific date, room-type and capacity:
LISTAVAILABLE
Delete a specific Booking:
DELETE
Exit the system:
EXIT

Description:

This is the home page containing the list of commands that the program can execute.

ADD



ADD

Command to add a new booking:

ADD <bookingId> <lecturerId> <roomNumber> <date> <time> <duration> <reason> <noattendees>

Insert command here

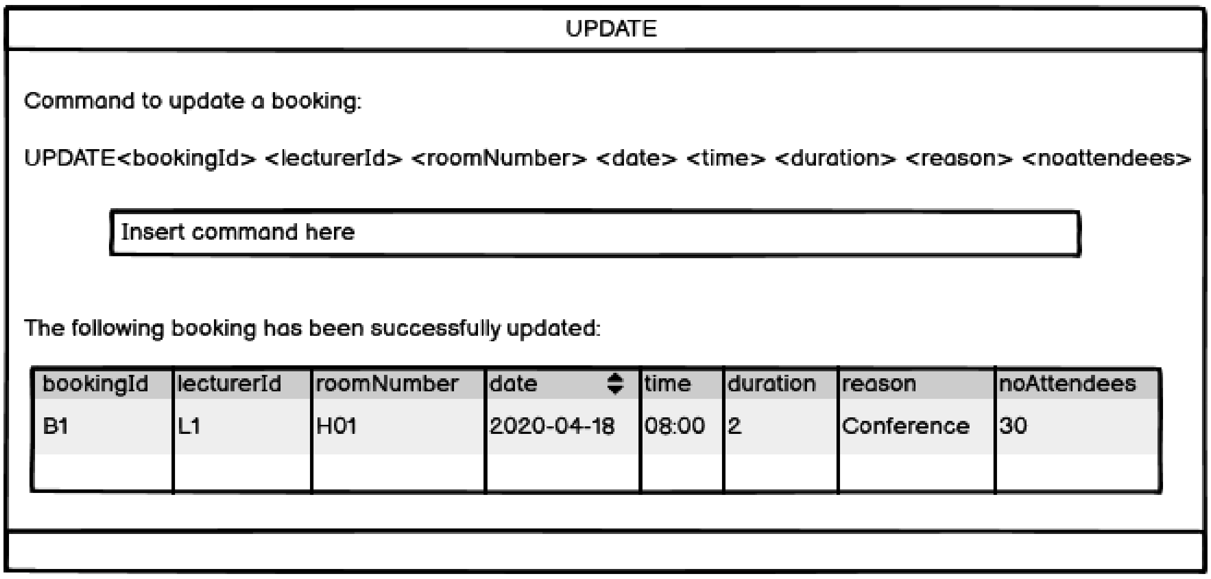
The following booking has been successfully added:

bookingId	lecturerId	roomNumber	date	time	duration	reason	noAttendees
B1	L1	H01	2020-04-18	08:00	2	Conference	30

Description:

This is the ADD command, allowing the user to add a new booking by inserting the details: bookingId, lecturerId, roomNumber, date, time, duration, reason, noAttendees.

UPDATE



UPDATE

Command to update a booking:

UPDATE<bookingId> <lecturerId> <roomNumber> <date> <time> <duration> <reason> <noattendees>

Insert command here

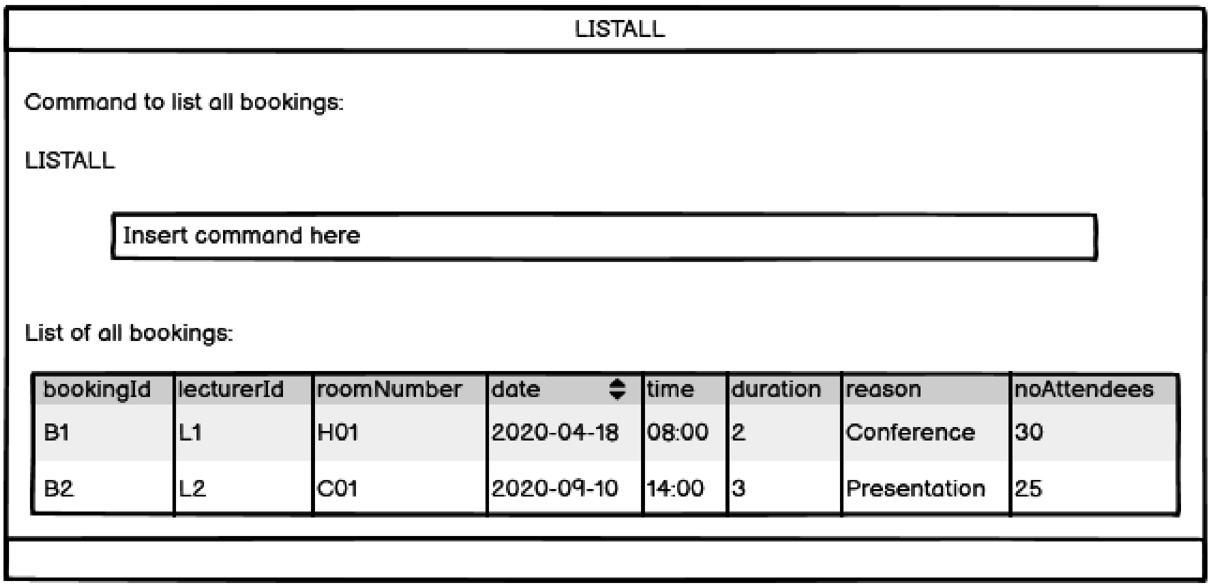
The following booking has been successfully updated:

bookingId	lecturerId	roomNumber	date	time	duration	reason	noAttendees
B1	L1	H01	2020-04-18	08:00	2	Conference	30

Description:

This is the UPDATE command, allowing the user to update an existing booking by inserting the details: bookingId, lecturerId, roomNumber, date, time, duration, reason, noAttendees.

LISTALL



Command to list all bookings:

LISTALL

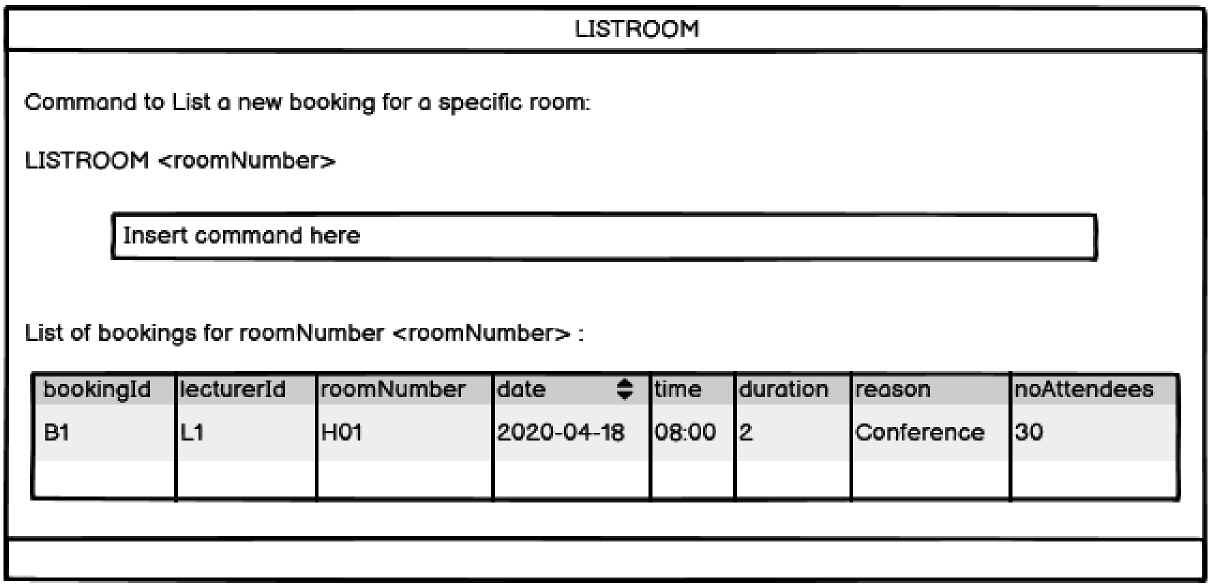
List of all bookings:

bookingId	lecturerId	roomNumber	date	time	duration	reason	noAttendees
B1	L1	H01	2020-04-18	08:00	2	Conference	30
B2	L2	C01	2020-09-10	14:00	3	Presentation	25

Description:

The LISTALL command, lists all the bookings.

LISTROOM



Command to List a new booking for a specific room:

LISTROOM <roomNumber>

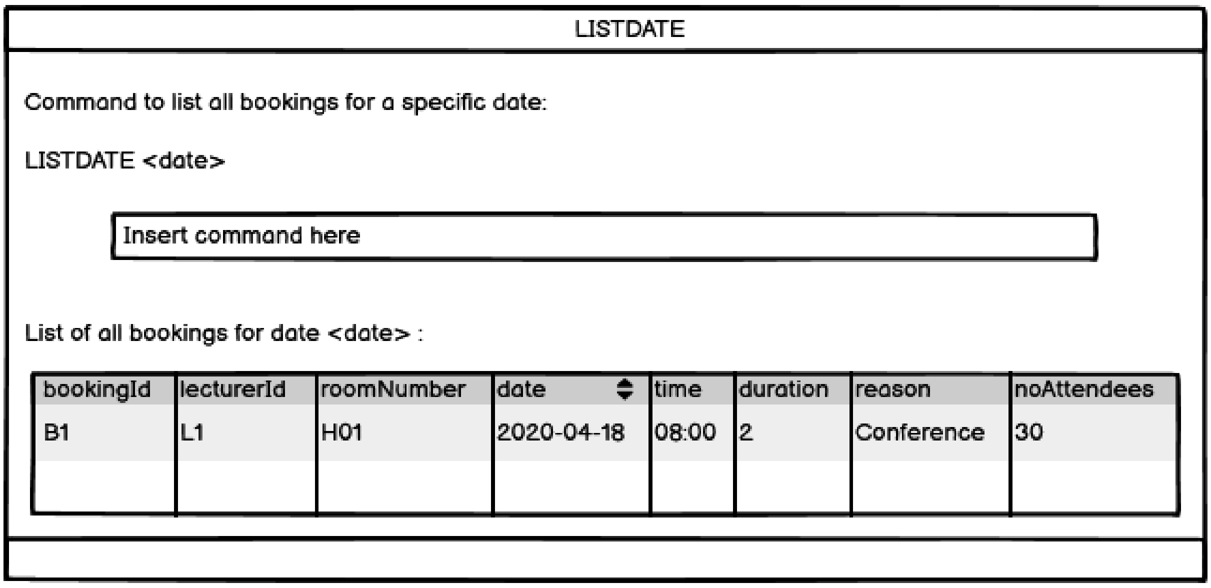
List of bookings for roomNumber <roomNumber> :

bookingId	lecturerId	roomNumber	date	time	duration	reason	noAttendees
B1	L1	H01	2020-04-18	08:00	2	Conference	30

Description:

The LISTROOM command, lists all bookings for a particular room.

LISTDATE


LISTDATE

Command to list all bookings for a specific date:

LISTDATE <date>

Insert command here

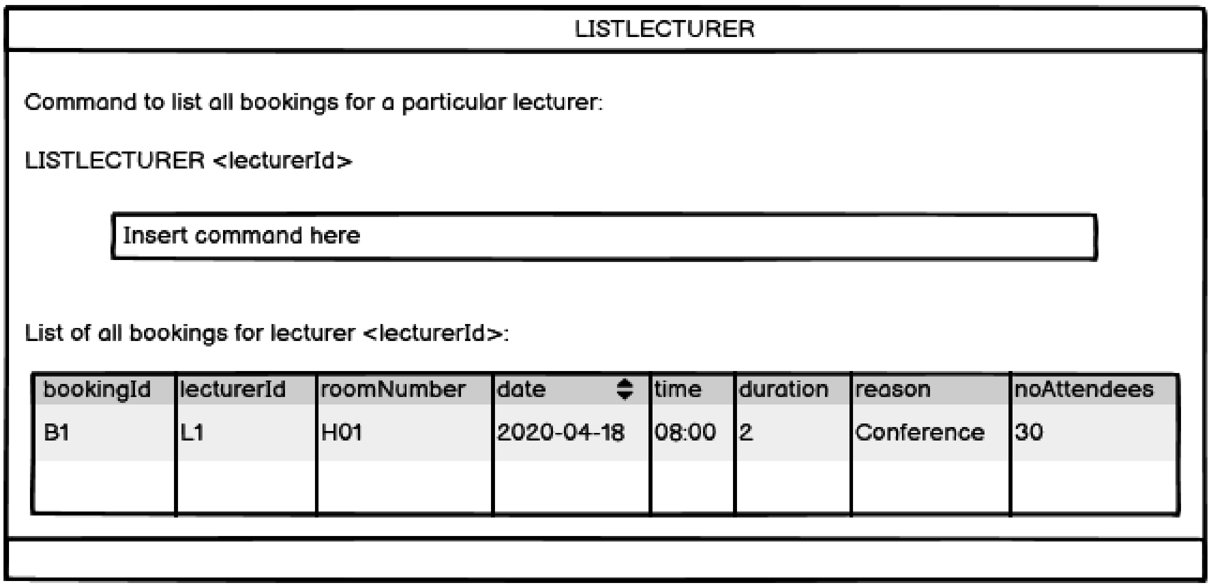
List of all bookings for date <date> :

bookingId	lecturerId	roomNumber	date	time	duration	reason	noAttendees
B1	L1	H01	2020-04-18	08:00	2	Conference	30

Description:

The LISTDATE command, lists all bookings for a particular date.

LISTLECTURER


LISTLECTURER

Command to list all bookings for a particular lecturer:

LISTLECTURER <lecturerId>

Insert command here

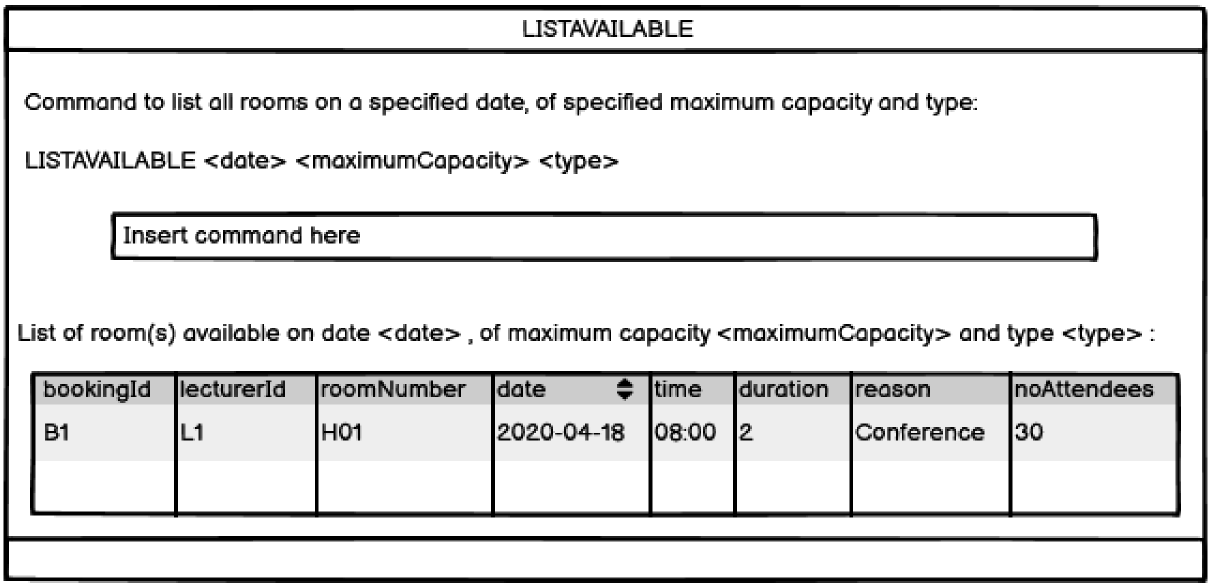
List of all bookings for lecturer <lecturerId>:

bookingId	lecturerId	roomNumber	date	time	duration	reason	noAttendees
B1	L1	H01	2020-04-18	08:00	2	Conference	30

Description:

The LISTLECTURER command, lists all bookings for a particular lecturer.

LISTAVAILABLE


LISTAVAILABLE

Command to list all rooms on a specified date, of specified maximum capacity and type:

LISTAVAILABLE <date> <maximumCapacity> <type>

Insert command here

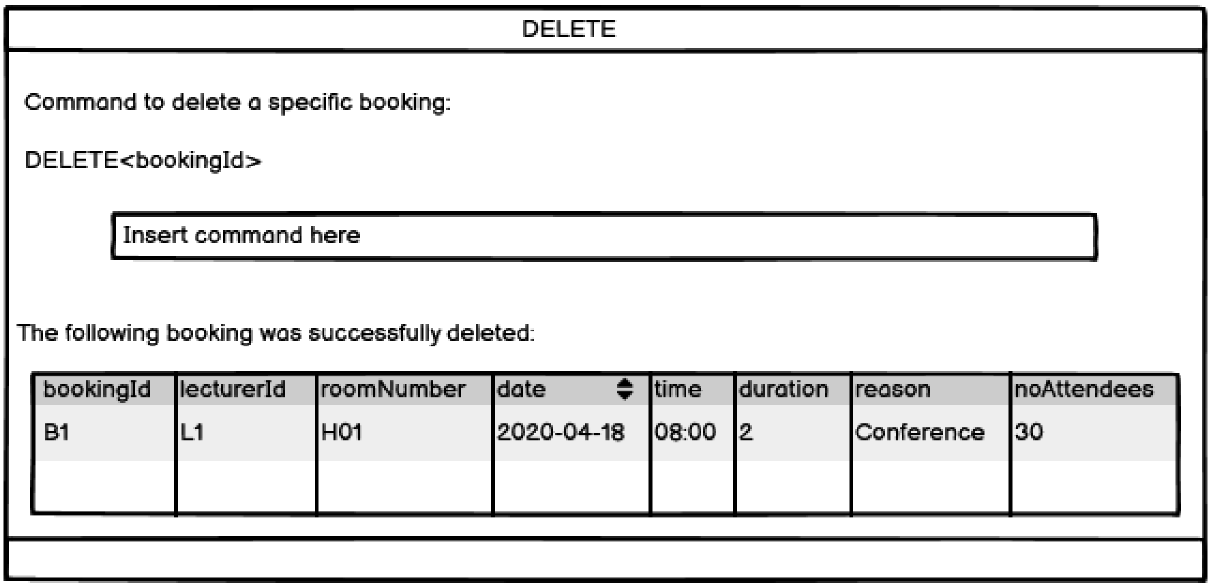
List of room(s) available on date <date> , of maximum capacity <maximumCapacity> and type <type> :

bookingId	lecturerId	roomNumber	date	time	duration	reason	noAttendees
B1	L1	H01	2020-04-18	08:00	2	Conference	30

Description:

The LISTAVAILABLE command, lists all rooms for a particular date, of specified maximum capacity and type.

DELETE


DELETE

Command to delete a specific booking:

DELETE<bookingId>

Insert command here

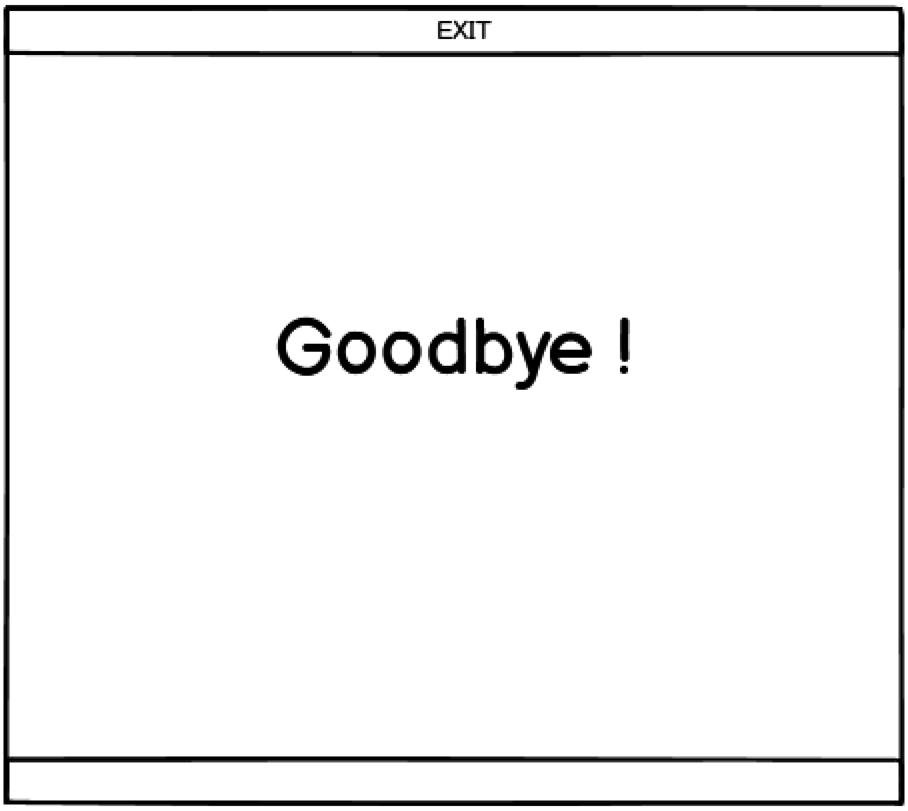
The following booking was successfully deleted:

bookingId	lecturerId	roomNumber	date	time	duration	reason	noAttendees
B1	L1	H01	2020-04-18	08:00	2	Conference	30

Description:

The DELETE command, deletes a specific booking.

EXIT



Description:

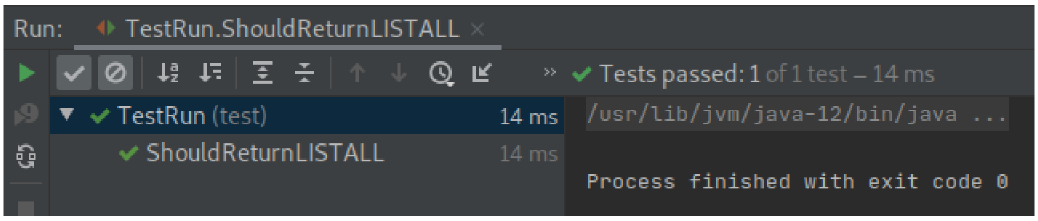
The exit command displays a goodbye message.

TESTING

Testing was performed using Junit framework. 9 tests were carried out using Junit and other manual tests were also performed.

1. Testing the LISTALL user input.

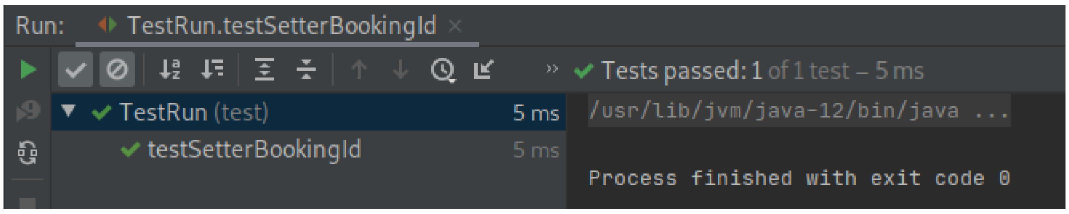
   The test was carried out to verify whether whenever the user inserts the LISTALL command in console, the data is actually well received.



```
Run:  TestRun.ShouldReturnLISTALL
Tests passed: 1 of 1 test – 14 ms
TestRun (test)          14 ms
  ShouldReturnLISTALL   14 ms
/usr/lib/jvm/java-12/bin/java ...

Process finished with exit code 0
```

2. Testing the setter method for bookingId

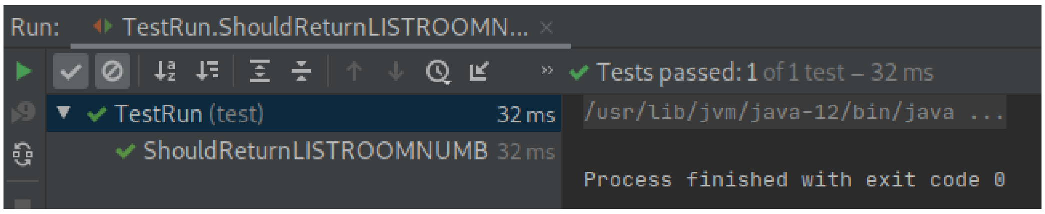


```
Run:  TestRun.testSetterBookingId
Tests passed: 1 of 1 test – 5 ms
TestRun (test)          5 ms
  testSetterBookingId   5 ms
/usr/lib/jvm/java-12/bin/java ...

Process finished with exit code 0
```

3. Testing the LISTROOM user input.

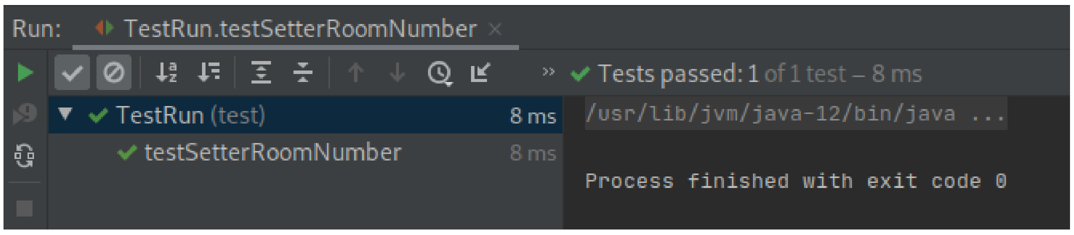
   The test was carried out to verify whether whenever the user inserts the LISTROOM command in console, the data is actually well received.



```
Run:  TestRun.ShouldReturnLISTROOMN...
Tests passed: 1 of 1 test – 32 ms
TestRun (test)              32 ms
  ShouldReturnLISTROOMNUMB  32 ms
/usr/lib/jvm/java-12/bin/java ...

Process finished with exit code 0
```

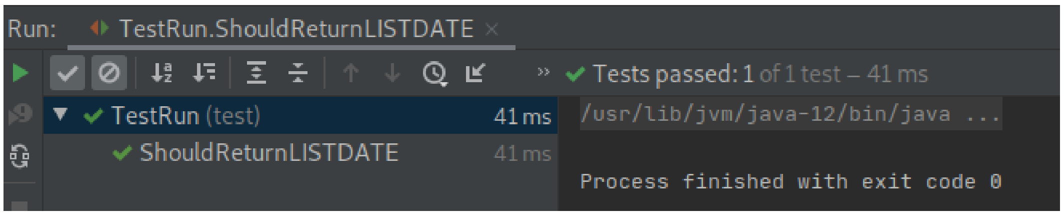
4. Testing the setter method for roomNumber



```
Run:   TestRun.testSetterRoomNumber
Tests passed: 1 of 1 test – 8 ms
TestRun (test)              8 ms
  testSetterRoomNumber      8 ms
/usr/lib/jvm/java-12/bin/java ...

Process finished with exit code 0
```

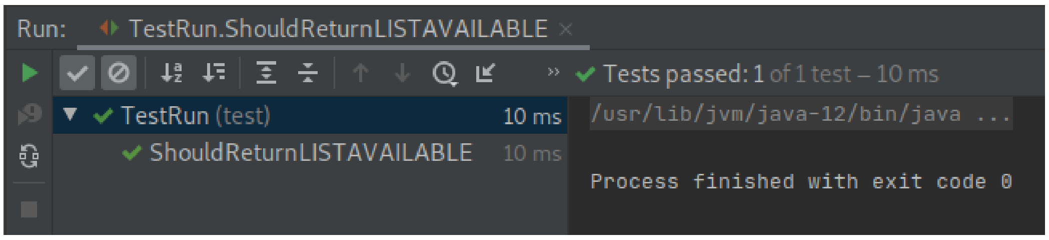
5. Testing the LISTDATE user input.
   The test was carried out to verify whether whenever the user inserts the LISTDATE command in console, the data is actually well received.



```
Run:   TestRun.ShouldReturnLISTDATE
Tests passed: 1 of 1 test – 41 ms
TestRun (test)              41 ms
  ShouldReturnLISTDATE      41 ms
/usr/lib/jvm/java-12/bin/java ...

Process finished with exit code 0
```

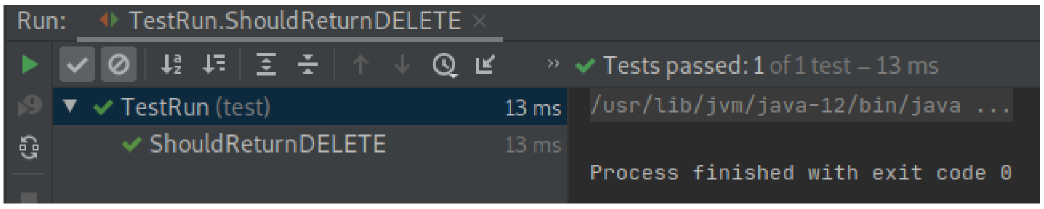
6. Testing the LISTAVAILABLE user input.
   The test was carried out to verify whether whenever the user inserts the LISTAVAILABLE command in console, the data is actually well received.



```
Run:   TestRun.ShouldReturnLISTAVAILABLE
Tests passed: 1 of 1 test – 10 ms
TestRun (test)                   10 ms
  ShouldReturnLISTAVAILABLE      10 ms
/usr/lib/jvm/java-12/bin/java ...

Process finished with exit code 0
```

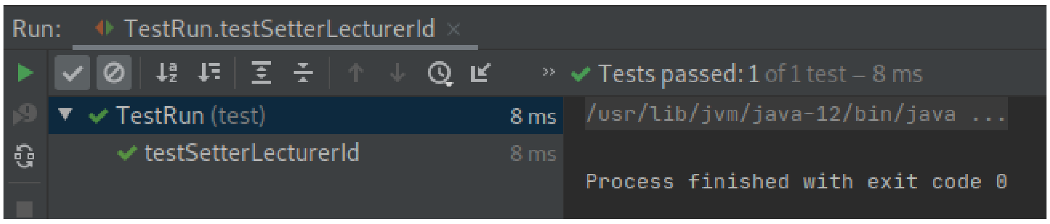
7. Testing the LISTDELETE user input.
   The test was carried out to verify whether whenever the user inserts the LISTDELETE command in console, the data is actually well received.



```
Run:   TestRun.ShouldReturnDELETE
Tests passed: 1 of 1 test – 13 ms
TestRun (test)              13 ms
  ShouldReturnDELETE        13 ms
/usr/lib/jvm/java-12/bin/java ...

Process finished with exit code 0
```

8. Testing the setter method for lecturerId


Run:
TestRun.testSetterLecturerId
Tests passed: 1 of 1 test – 8 ms
TestRun (test) 8 ms
testSetterLecturerId 8 ms

```
/usr/lib/jvm/java-12/bin/java ...

Process finished with exit code 0
```

Below are some of the manual tests which were carried out:

Test	Expectation	Result	Pass or Fail ?
Type invalid command	Invalid command	Invalid command	pass
Type “quit” or “exit”	Socket close and program exits	Socket close and program exits	Pass
LISTALL with more than 1 argument	Too many arguments	Too many arguments	Pass
LISTLECTURER with more than 2 arguments	Too many arguments	Too many arguments	Pass
LISTROOM with more than 2 arguments	Too many arguments	Too many arguments	Pass
LISTDATE with more than 2 arguments	Too many arguments	Too many arguments	Pass
LISTAVAILABLE with more than 4 arguments	Too many arguments	Too many arguments	Pass
ADD with more than 9 arguments	Too many arguments	Too many arguments	Pass
UPDATE with more than 9 arguments	Too many arguments	Too many arguments	Pass
DELETE with more than 2 arguments	Too many arguments	Too many arguments	Pass
EXIT with more than 1 argument	Too many arguments	Too many arguments	Pass
In Add command, when bookingId does not start with letter ’B’	Error bookingId must start with ‘B’	Error bookingId must start with ‘B’	Pass

In UPDATE command, when bookingId does not start with letter ’B’	Error bookingId must start with ‘B’	Error bookingId must start with ‘B’	Pass
In DELETE command, when bookingId does not start with letter ’B’	Error bookingId must start with ‘B’	Error bookingId must start with ‘B’	Pass
In ADD command, when argument lecturerId is not correct	Error lecturer does not Exist	Error lecturer does not Exist	Pass
In UPDATE command, when argument lecturerId is not correct	Error lecturer does not Exist	Error lecturer does not Exist	Pass
In ADD command, when argument roomNumber is not correct	Error room does not Exist	Error room does not Exist	Pass
In UPDATE command, when argument roomNumber is not correct	Error room does not Exist	Error room does not Exist	Pass
In ADD command, when argument date is not is in wrong format	Invalid date format	Invalid date format	Pass
In UPDATE command, when argument date is not is in wrong format	Invalid date format	Invalid date format	Pass
In ADD command, when argument duration is not is not a Number	Error, duration can only be in Numbers	Error, duration can only be in Numbers	Pass
In UPDATE command, when argument duration is not is not a Number	Error, duration can only be in Numbers	Error, duration can only be in Numbers	Pass
In ADD command, the maximumCapacity is exceeded for its corresponding roomNumber	Error <roomNumber> can only accommodate <maximuCapacity> people	Error <roomNumber> can only accommodate <maximuCapacity> people	Pass
In UPDATE command, the maximumCapacity is exceeded for its corresponding roomNumber	Error <roomNumber> can only accommodate <maximuCapacity> people	Error <roomNumber> can only accommodate <maximuCapacity> people	Pass

CONCLUSION

Summary

This university room booking management system was completed with the following features successfully added:

- Add booking (allowing the user to book a new room in a specific time slot)
- Update booking (the details about existing bookings can be modified using this feature)
- LISTALL (providing the user with list of all the bookings in the database)
- LISTROOM (providing the user with a list of all bookings for a specified room)
- LISTLECTURER (providing the user with a list of all bookings for a lecturer)
- LISTDATE (providing the user with a list of all bookings for a date)
- DELETE (erasing a specified booking from the database)
- EXIT (terminating the program).

The program also has interesting features such as extensive validations for user input. Moreover, the program successfully avoids double booking.

Limitations

The program does not use a proper graphical user interface such as JavaFx to make the user experience better. Instead, console is being used. Consequently, there are some drawbacks such as:

- The staff will have to remember and write long syntaxes
- If an information is wrongly entered, the staff will not be able to modify the input he has already added but instead he/she has to re rewrite the whole command.

The program does not allow multiple bookings to be deleted at the same time, the user has to delete one at a time which is not very time effective in cases where a lot of bookings need to be erased.

The program has come long commands that should be written, for example the ADD and UPDATE command. This could have been avoided if for example, for the add command the user was asked and prompted with one detail at a time.

Some of the validations were hard coded for the validations. For example, for the maximum room capacity, this could have been worked around using SQL commands.

Instructions could have been more elaborated for the commands. This would have helped for a better user experience.

Future Approach

In the future while working on such a project, an automatic backup could be implemented so as to avoid data loss or wrongly deleting or updating a date, it will also be much more time efficient. A proper complete GUI program could have been developed to allow full feature of the database and allow staff to even add new customers. Some validations for the program can be implemented using SQL commands instead of using regular expressions which is a too direct approach. A register and login functionality can be added for further security. Moreover, the login feature will help to better keep track of the work of each employee who uses the system. Features like deleting multiple bookings at once could have been implemented. Functionalities to sort through the different lists could have been implemented. For example, sorting through the 'All bookings list' by date or by time or by lecturer or by room number instead of sorting by bookingId only.