CSC540 project 1 Report

Team 6: ldong6, sxu11, shou3, yhuang34

Assumptions:

Following assumptions need to be considered to implement the University Student Housing system we designed:

1. Only full-time students, approved guests and housing office staff in our university are allowed to register. That is to say, a person can't sign up if his/her information is not recorded as one of those above in our university.

2. Only students and approved guests are eligible for university housing.

3. Students and staff (administrator) are uniquely identified by their id, and no student and staff can share the same id.

4. There are two basic housing options: residential halls and apartments. Apartments are categorized as general apartment and family apartment. General apartments provide single-room accommodation for groups of three or four students, while family apartment are rented as a whole with one, two or three bedrooms.

5. Only family students can rent family apartments.

6. In the same period of time, one student only allowed to rent one room in a hall or apartment, except when he/she rent a family apartment.

7. A student may rent a room in a hall or student flat for various periods of time.

8. To rent a room, each student must submit a request with one of the three housing preferences. Each request should follow four statuses: Pending, Processed (after approved), In Progress (after signed up by students) and Completed. Similarly, a termination request should be sent when a student wish to terminate. Termination request passed through three states: Pending, Processed and Completed.

9. Along with housing request, student may request a parking spot in one of the nearby parking lots. And the parking spots are given based on the availability and "first come first service".

10. Parking for housing students is only available to campus residents. Students may also opt for the more general student parking lots.

11. Renting information (uniquely identified using a lease number) is stored in a lease agreement between a student and accommodation office.

12. A monthly invoice is issued to each student and a final invoice is generated at the end of lease.

Problem Statement:

This project aims at designing a database application of a housing system for university students and guests. The basic function of this system is to assist students' housing choices along with parking choices, as well as the administration of the staff. In general, students and guests can view available housing options based on different criteria, initiate and check status of lease related requests, invoices and maintenance tickets. Housing staff can view and act on the different types of requests. Besides the fundamental requirement, there are other constraints need to be consider, such as different type of users, time limitation of lease and invoice, and students' preferences to choose their ideal housing. We divided all problems into two categories: fundamental and management requirements.

Fundamental requirements:

1. Store basic information of all users: students, approved guests and administrators.

2. Store information to exhibit housing options along with parking options to users, including residential halls, apartments, rooms, associated parking lots and parking spots.

3. Record students' action in the form of leasing requests, maintenance ticket, termination request and lease.

4. Record the information about which staff took any action on a lease or ticket and date.

5. Allow students and guests to view available housing options based on different criteria, initiate and check status of lease related requests, invoices and maintenance tickets.

6. Similarly, Allow administrator to view or take action such as add, approve, delete the record of all students, namely their requests, tickets and leases information.

Management requirements:

1. Access control: the system need to protect the security and privacy of students' and staff's information, which means students can't view other students' or staff's information. Housing office administrators' accesses are restricted in students' part of basic information (ex. not including password).

2. Concurrency control: the system should be multi-tasking (allow multiple staff and students to log in, view and take actions at the same time). To maintain the consistency, concurrent execution need to implement.

3. Alert and notification: the system should be able to send necessary alert to students when there is a payment due while send notification to students when there is a lease or invoice is available.

4. Crash recover: the system should be robust enough to recover from unforeseeable incidence such as power-off.

5. Multiple identities: the system has to handle multi-authentication. For example, once a person is approved for an apartment lease, the system needs to identify the student

with a single lease id or some labels else. Then load the corresponding authority and information. We can't let conflicts happen.

Entity and relationship:

Entities:

We categorized all entities and list them as follows:

1. Student
2. Guest (Visitor and scholar)
3. Housing Office Staff (administrator)
4. Halls
5. Apartments: includes general apartment and family apartment
6. Room
7. Lease
8. Parking lot
9. Parking spot
10. Invoice
11. Ticket

Relationship:

Relationship explained in detail

1. Has: Apartment **has** general apartment and family apartment; Hall **has** room; Parking lot **has** parking spot; Family **has** family member.
2. Manage: Staff **manage** hall or apartment.
3. lease_request: Student **lease_requests** lease.
4. parking_request: Student **parking_requests** parking spot.
5. termination_request: Student **termination_request** termination.
6. Generate: Lease **generate** invoice
7. Raise: Student **raise** ticket
8. Near: parking lot **near** house

All relationships are listed as follows:

1. termination request(student, current lease)
2. generate(lease, invoice)
3. raise(student, tickets)
4. near(parking lot, house)
5. has(parking lot, parking spot)
6. has(hall, room)
7. has(family, family member)
8. manage(administrator, hall)
9. manage(administrator, apartment)
10. request(student, parking)

Users:

This is a housing system, which allows students to rent and help the office to manage. The only allowed user groups of our system are students, approved guests and housing staff (administrators).

Students can sign up in our system using their student id, then view all the housing options: residential halls, general apartment and family apartment even specific room. Meanwhile they can view all the available dedicated parking lots or parking spots, which are associated with specific housing option. Once a student decides to rent a room, he/she can make a lease request, and wait for it to be approved. After the request approved, the system will established a lease between the student and the Accommodation Office. Once a month, students will receive invoice with information of their rent. During the least, students can also submit a ticket to ask for maintenance. When a students wish to terminate the lease, he/she will have make a termination request, which is similar to lease request.

A university guests such as visiting scholars are able to request housing on campus once they are approved and assigned an approval id. Then they can sign up in our system using their approval id, and be treated as students.

Staff in housing office is the administrator of our system. They have the access to all the basic information of student and guest users as well as their housing information. Administrators also have the right to take action on a request, a lease or a ticket.

E-R Diagrams:

E-R model cannot capture all the constraints mentioned in the document. Following are those constraints and their elaboration:

1. Domain constraints can not be captured by ER diagram.
2. With respect to ISA hierarchies, overlap constraint and covering constraints can not be captured. In our case, all entities in subclasses collectively include all entities in the superclass, for example, there is no other kinds of apartment besides general apartment and family apartment. Also there is overlapping in some of our cases, for instance, a student can be a family student while being a guest. However by default, entity sets are constrained to have no overlap and there is no covering constraint.
3. Some specific features can't be captured: General parking plot is available to students can not be captures; Invoice generated monthly for students using monthly payment method while it generated at the end of the lease for students using semester payment; The parking spots are given based on the availability and "first come first serve" basis.

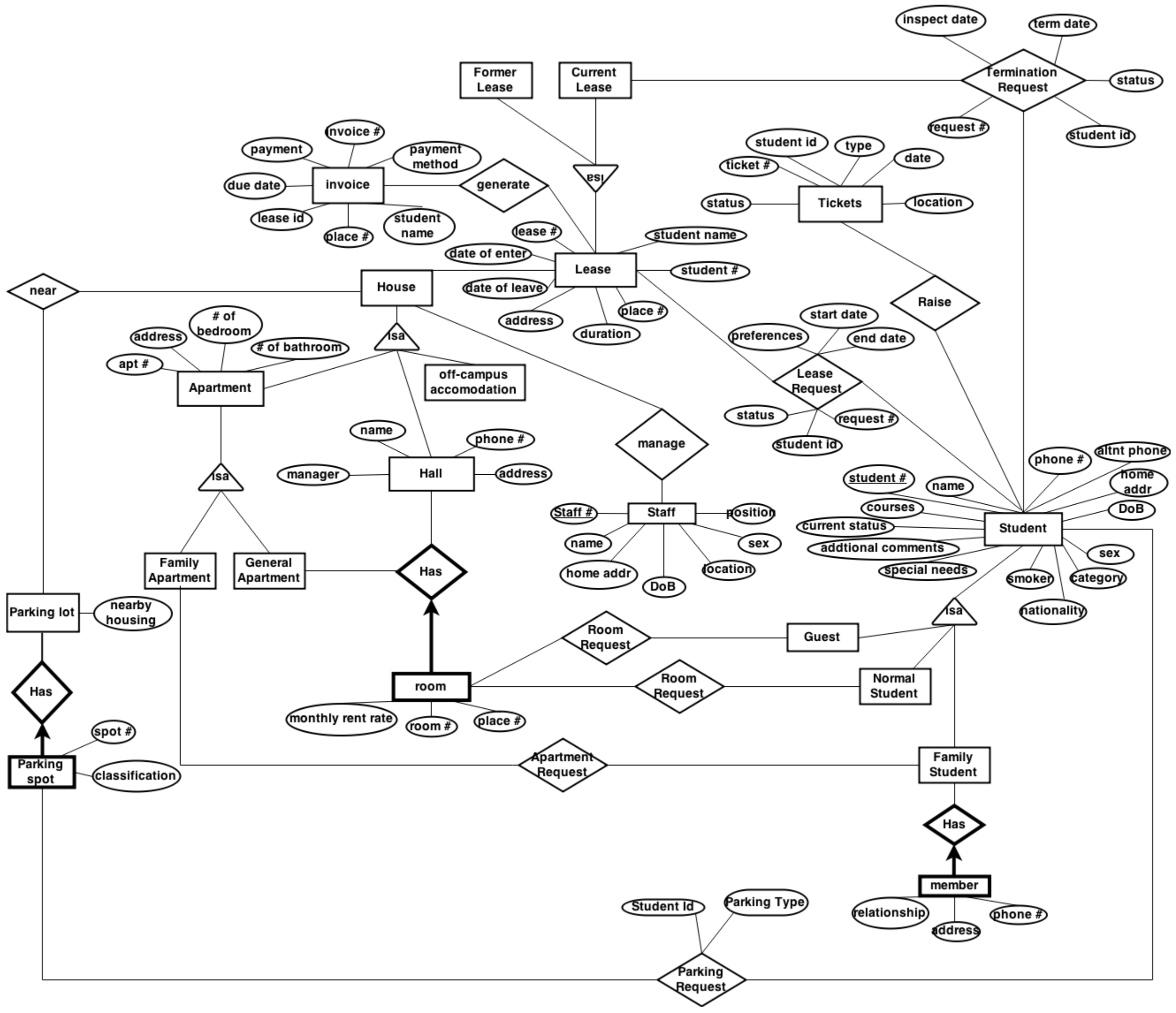


Figure 1 Housing E-R diagram

Relationship Schema:

Based on E-R model represented in Figure 1, we associate with other constraints mentioned in project descriptive document to render the relational schemas as follows, which are also tables we created in database:

1. family_aparment (<u>id: integer,</u> name: string, addr: string, room_count: interger, bath_count: interger, month_rate: real)
2. family_member(<u>id: integer</u>, <u>student_id: integer,</u> name: string, birth_date: date) general_aparment (id: integer, name: string, addr: string, tel: string, room_count: interger, bath_count: interger)

3. invoice (id: integer, lease_id: integer, pay_date: date, pay_method: string, status: string, penalty: real, damage_charge: real, late_fee: real, total: real)
4. kin_info (id: integer, student_id: integer, name: string, relationship: string, addr: string, city: string, post_code: string, tel: string)
5. lease(id: integer, student_id: integer, house_id: integer, room_id: integer, room_num: string, duration: string, enter_date: date, leave_date: date, deposit: string, payment: string, penalty: string, inspect_date: date, parking: integer)
6. lease_request(id: integer, student_id: integer, preference1: string, preference1: string, preference2: string, preference3: string, payment_method: string, status: string, start_date: date, end_date: date)
7. nearby(id: integer, lot_id: integer, house_id: integer)
8. parking_lot(id: integer, num_parking_spot: integer)
9. parking_request(id: integer, vehicle_type: string, nearbyt: string, handicapped: string, status: string)
10. parking_spot(id: integer, lot_id: integer, classification: string, fee: real, availability: string)
11. residence_hall(id: integer, name: string, addr: string, tel: string, manager_id: integer)
12. room(id: integer, room_number: string, month_rate: real, house_id: integer)
13. staff(id: integer, password: string, fname: string, lname: string, addr: string, city: string, post_code: string, birth_date: date, gender: string, position: string, work_at: string)
14. student(id: integer, password: string, fname: string, lname: string, type: string, gender: string, tel: string, alter_tel: string, addr: string, city: string, post_code: string, birth_date: date, category: string, nation: string, smoker: string, need: string, comment: string, status: string, courses: string, kin_id: integer)
15. ticket(id: integer, type: string, student_id: integer, date: date, location: string, status: string, description: string)
16. termin_req(id: integer, lease_id: integer, reason: string, date: date, status: string, inspection_date: date, extra_fee: real)
17. guest(id: integer, password: string, fname: string, lname: string, type: string, gender: string, tel: string, alter_tel: string, addr: string, city: string, post_code: string, birth_date: date, category: string, nation: string, smoker: string, need: string, comment: string, status: string, courses: string, kin_id: integer)

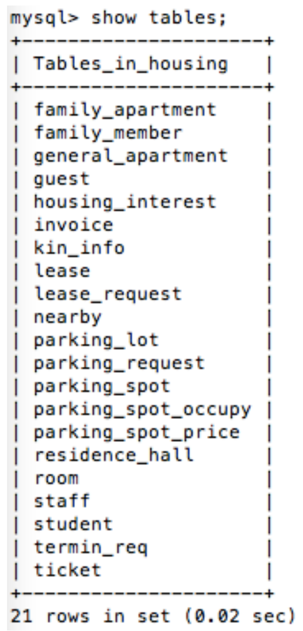
Function Dependencies and Normal Forms:

According to relational schemas and tables we created in database (shown in Figure 2), we can extract functional dependencies and list them as follows (each item corresponds a table):

1. family_apartment_id → (name, addr, tel, vacancy, apt_num, bed_count, month_rate, deposit)
2. family_member_id, student_id → (name, birth_date)
3. general_apartment_id → (name, addr, tel, room_count, bath_count, rent, deposit)
4. guest_id, password → (fname, lname, type, gender, tel, alter_tel, addr, city, post_code, birth_date, category, nation, smoker, need, comment, status, courses,

kin_id)

5. invoice_id, lease_id → (pay_date, pay_method, status, penalty, damage_charge, late_fee, total)
6. kin_info_id,s tudent_id, name → (relationship: string, addr: string, city: string, post_code: string, tel: string)
7. lease_id, student_id, house_id, room_id → (room_num, duration, enter_date, leave_date, deposit, payment, penalty, inspect_date, parking)
8. lease_request_id → (student_id, preference1, preference1, preference2, preference3, payment_method, status, start_date, end_date)
9. nearby_id, lot_id → (house_id)
10. parking_lot_id → (num_parking_spot)
11. parking_request_id → (vehicle_type, nearbyt, handicapped, status)
12. parking_spot_id, lot_id → (classification, fee, availability)
13. residence_hall_id → (name, addr, tel, manager_id)
14. room_id → (room_number, month_rate, house_id)
15. staff_id, password → (fname, lname, addr, city, post_code, birth_date, gender: string, position, work_at)
16. student_id, password → (fname, lname, type, gender, tel, alter_tel, addr, city, post_code, birth_date, category, nation, smoker, need, comment, status, courses, kin_id)
17. ticket_id → (type, student_id, date, location, status, description)
18. termin_req_id → (lease_id, reason, date, status, inspection_date, extra_fee)



```
mysql> show tables;
+----------------------+
| Tables_in_housing    |
+----------------------+
| family_apartment     |
| family_member        |
| general_apartment    |
| guest                |
| housing_interest     |
| invoice              |
| kin_info             |
| lease                |
| lease_request        |
| nearby               |
| parking_lot          |
| parking_request      |
| parking_spot         |
| parking_spot_occupy  |
| parking_spot_price   |
| residence_hall       |
| room                 |
| staff                |
| student              |
| termin_req           |
| ticket               |
+----------------------+
21 rows in set (0.02 sec)
```

Figure 2 All tables in housing database

Looking though all above functional dependences, it is obvious that the left parts of the formulas are all primary keys. Based on the definition of Normal Form, we can safely conclude that all FDs are in BCNF. Then of cause they are also in 3NF, 2NF and 1NF.

APIs and SQL queries:

APIs:

MVC (Model–view–controller) is a software architectural pattern for implementing user interfaces. It consists of three parts: model, view, and controller.

Spring is a popular application framework written in Java and is used by many developers to create quality applications. This framework consists of many parts which provide different services, and it helps developers to pay attention to the business logic in a proper manner. The framework of spring separates MVC in an efficient manner, and it also has the ability to provide many controllers in order to be used as the base classes.

Another tool we use in our project is hibernate. Hibernate is an object – relational mapping and persistence framework for Java that allows developers to map plain old Java objects to relational database tables. Hibernate mainly aims at relieving the developer from the common data persistence related tasks. With the help of hibernate, developers can get maximum effects of data query and retrieval facilities, because hibernate maps the objects in Java with tables in database in an efficient manner. In a word, hibernate provides improved productivity, performance, maintainability, and portability.

SQL queries:

Reporting Queries

1. For each hall or apartment, display the total number of nearby parking spots (both available and not available)

```
SELECT H.name, COUNT(DISTINCT S.id) spot_num
FROM residence_hall H, nearby N, parking_spot S
WHERE H.id = N.house_id AND N.lot_id = S.lot_id
GROUP BY H.name
UNION
SELECT GA.name, COUNT(DISTINCT S.id)
FROM general_apartment GA, nearby N, parking_spot S
WHERE GA.id = N.house_id AND N.lot_id = S.lot_id
GROUP BY GA.name
UNION
SELECT FA.name, COUNT(DISTINCT S.id)
FROM family_apartment FA, nearby N, parking_spot S
WHERE FA.id = N.house_id AND N.lot_id = S.lot_id
GROUP BY FA.name;
```

2. For each hall, display total number of graduate students on the request list that do not yet have approved leases.

```
SELECT H.name, COUNT(DISTINCT S.id) grads_num
FROM  residence_hall H
LEFT JOIN lease_request R ON R.preference1 = H.name OR R.preference2 = H.name
OR R.preference3 = H.name
LEFT JOIN student S ON R.student_id = S.id  AND S.category = "Graduate" AND
R.status = "pending"
GROUP BY H.name;
```

3. For each student renting, print their next invoice.

```
SELECT *
FROM invoice I, lease L
WHERE I.due_date > CURDATE() AND L.id = I.lease_id AND L.status = "current"
GROUP BY I.student_id;
```

4. For each available parking spot, print its information (id, lot, location and type)

```
SELECT S.id spot,S.lot_id lot,H.addr location, S.classification type
FROM parking_spot S, nearby N, residence_hall H
WHERE S.lot_id = N.lot_id AND N.house_id = H.id AND S.availability =
"available"
GROUP BY S.id
UNION
SELECT S.id spot,S.lot_id lot,GA.addr location, S.classification type
FROM parking_spot S, nearby N, general_apartment GA
WHERE S.lot_id = N.lot_id AND N.house_id = GA.id AND S.availability =
"available"
GROUP BY S.id
UNION
SELECT S.id spot,S.lot_id lot,FA.addr location, S.classification type
FROM parking_spot S, nearby N, family_apartment FA
WHERE S.lot_id = N.lot_id AND N.house_id = FA.id  AND S.availability =
"available"
GROUP BY S.id
UNION
SELECT S.id spot,S.lot_id lot,"General" location, S.classification type
FROM parking_spot S
WHERE S.lot_id = "7" AND S.availability = "available"
GROUP BY S.id
```

Retrieval Queries

1. Find the most popular hall or apartment (hall or apartment with the most number of requests approved, pending, or denied)

```
SELECT temp.name,MAX(temp.number)
```

```
FROM (SELECT H.name AS name, COUNT(DISTINCT R.student_id) AS
number
FROM residence_hall H, lease_request R
WHERE R.preference1 = H.name OR R.preference2 = H.name OR R.preference3
= H.name
GROUP BY H.name
UNION
SELECT GA.name, COUNT(DISTINCT R.student_id)
FROM general_apartment GA, lease_request R
WHERE R.preference1 = GA.name OR R.preference2 = GA.name OR
R.preference3 = GA.name
GROUP BY GA.name
UNION
SELECT FA.name, COUNT(DISTINCT R.student_id)
FROM family_apartment FA, lease_request R
WHERE R.preference1 = FA.name OR R.preference2 = FA.name OR
R.preference3 = FA.name
GROUP BY FA.name) AS temp
```

2. Find all students who have paid their rent after the due date for any of the past three months.

```
SELECT  S.fname,S.lname
FROM invoice I, lease L, student S
WHERE I.pay_date > I.due_date AND DATEDIFF(NOW(),I.due_date) < 180
L.id = I.lease_id AND L.student_id = S.id
GROUP BY S.id;
-- Use NOW() return current date and time and DATEDIFF(date1,date2)
return the day difference between date1 and date2
```

3. Print information about pending lease requests where student have not requested a parking spot.

```
SELECT *
FROM lease_request LR
WHERE LR.status = "pending" AND LR.student_id NOT IN
(SELECT PR.student_id
FROM parking_request PR);
```

Usecase Scenarios:

The work flow graph of our system is showed as below:

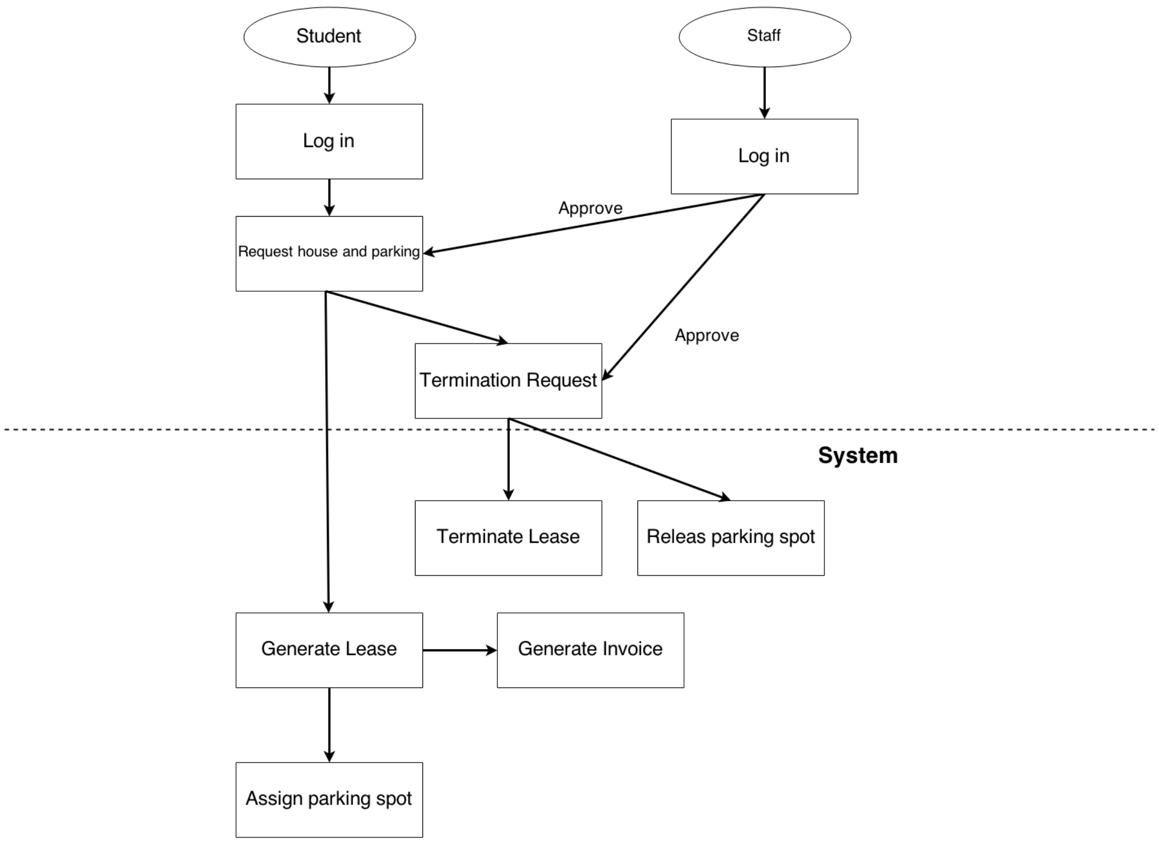


Figure 3 Work flow