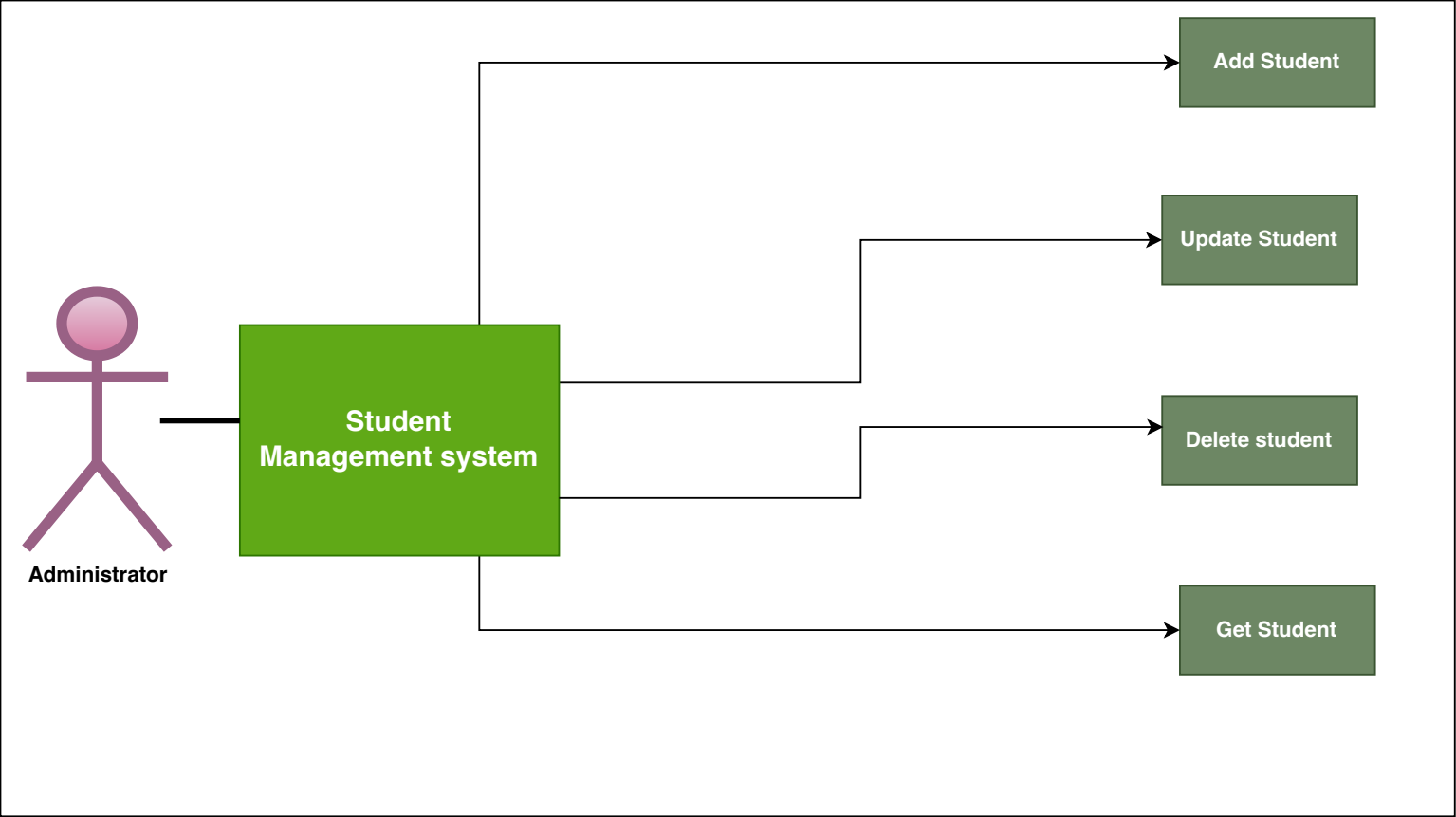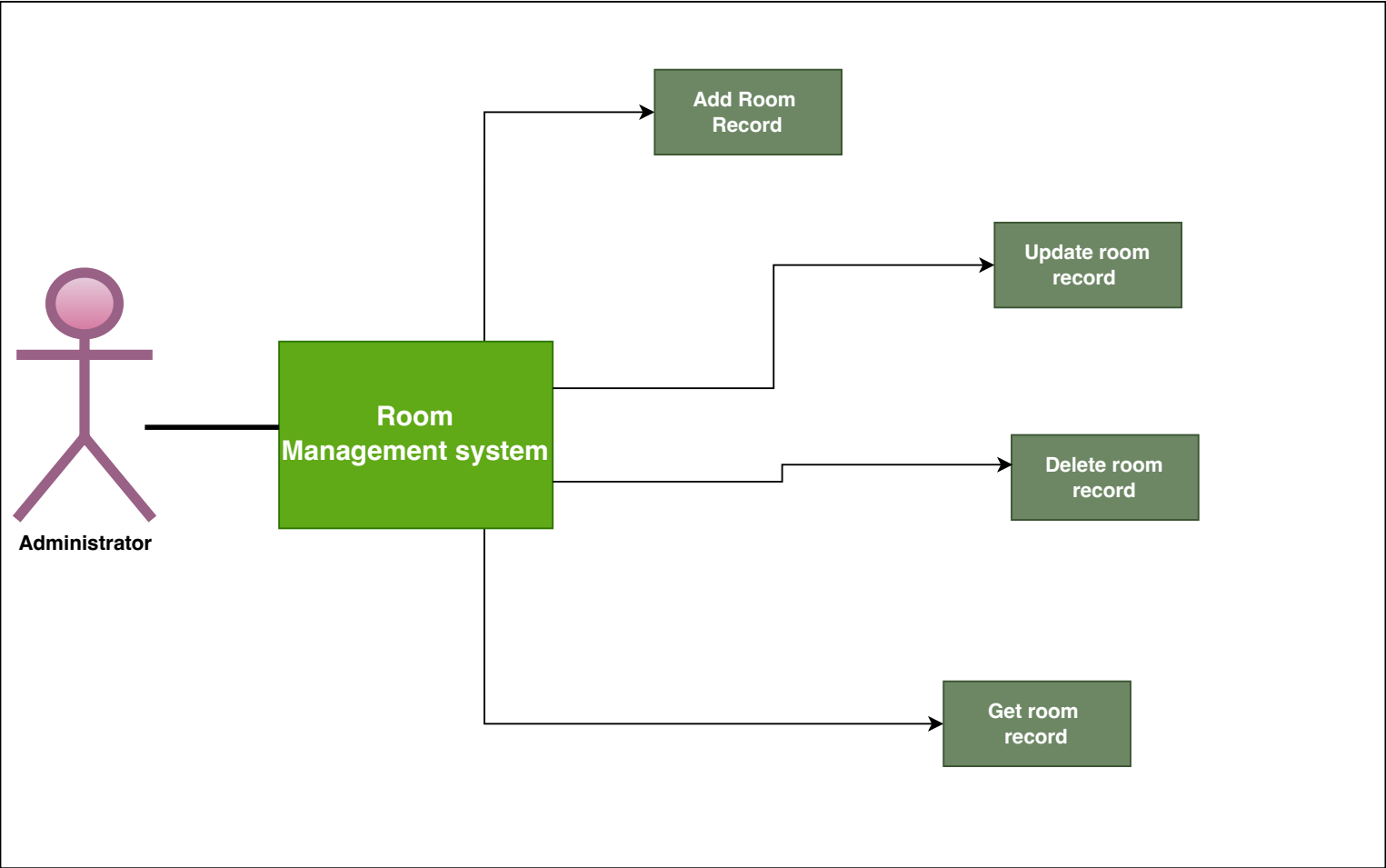| Requirements	
Functional Requirments	Non-Functional Requirments
Student Management 1) The system should allow administrators to get, add, update, and delete student records. 2) The system should store details such as student ID, first name, contact information, and assigned room.	***Usability*** 1) The system should provide help documentation and tooltips for user assistance. 2) The system should have an intuitive user interface that is easy to navigate for both administrators and students.
Room Management *1)* The system should allow administrators to get, add, update, and delete room records. 2) Each room record should include the room number, capacity, and the number of available places. 3) The system should track which students are assigned to which rooms.	***Reliability and Availability*** 1) The system should be available 24/7 with 99.9% uptime. 2)The system should implement regular backups to prevent data loss.
Administrator Management 1) The system should manage administrator records, including getting, adding, updating, and deleting administrators. 2)Each administrator should have an ID, first name, last name, and contact information.	***Scalability*** 1)The system should be able to scale to accommodate more students, rooms, and blocks as the university grows.
Accommodation Request Management 1)The system should allow students to submit accommodation requests. 2)Each request should include a unique request ID, the student's ID, preferred room, and status (e.g., pending, approved, rejected). 3)Administrators should be able to approve or reject accommodation requests.	***Maintainability*** 1)The system should be designed with modularity to facilitate easy updates and maintenance. 2) The system should include comprehensive logging to assist in troubleshooting problems.
Search and Filter 1)The system should allow administrators to search for students by name, ID, or room number. 2)The system should allow administrators to filter rooms by capacity, and availability.	***Compatibility*** 1)The system should be compatible with modern web browsers (Chrome, Firefox, Safari, Edge). 2)The system should be accessible from mobile devices.
	Security 1)The system should authenticate users (administrators and students) before allowing access. 2)The system should authorise actions based on user roles. 3)Sensitive data such as student contact information should be encrypted in the database.

Use Cases

Student Management

Administrator	**Student Management system**

Add Student

Update Student

Delete student

Get Student

Room Management

Administrator	**Room Management system**

Add Room Record

Update room record

Delete room record

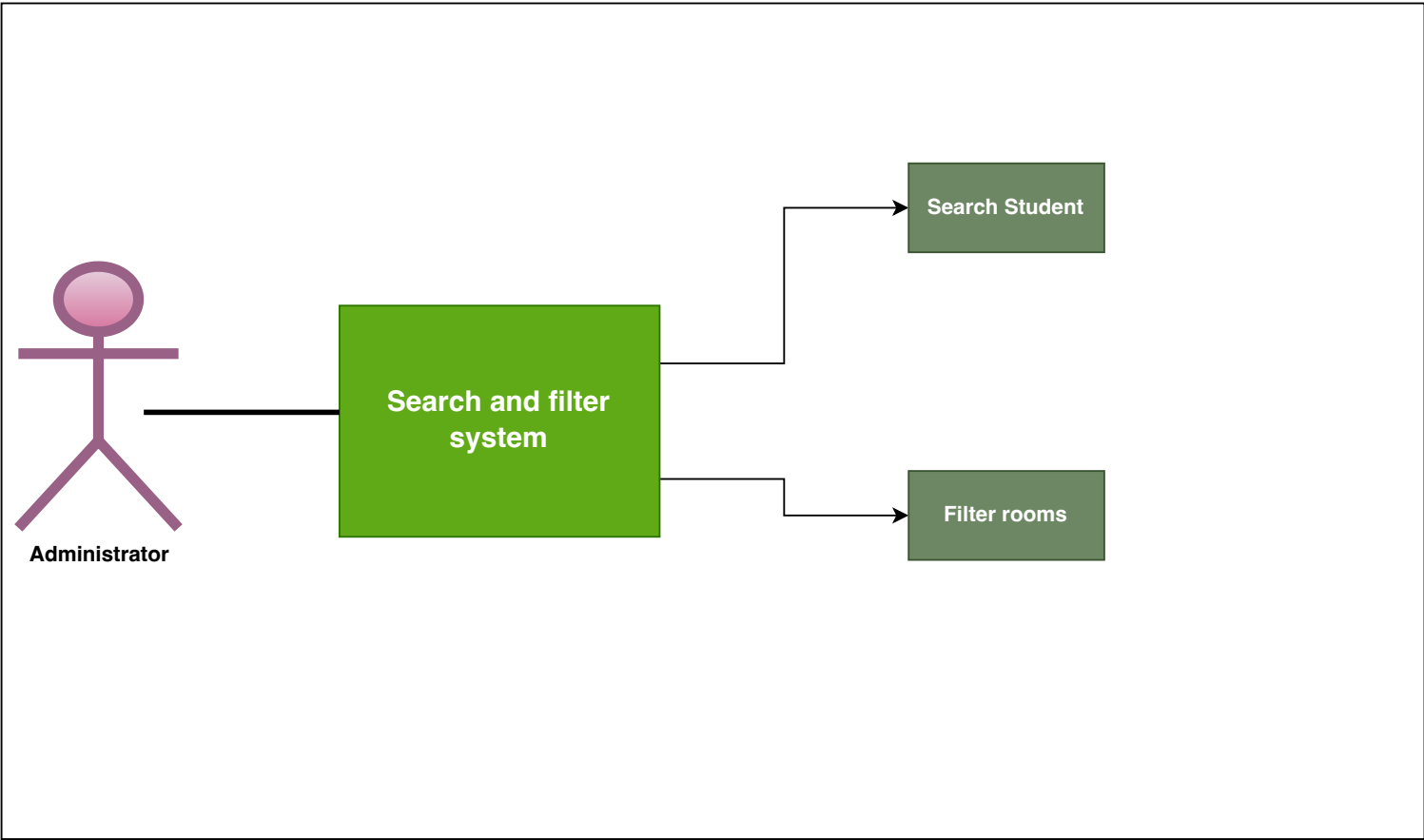Get room record

Administrator Management
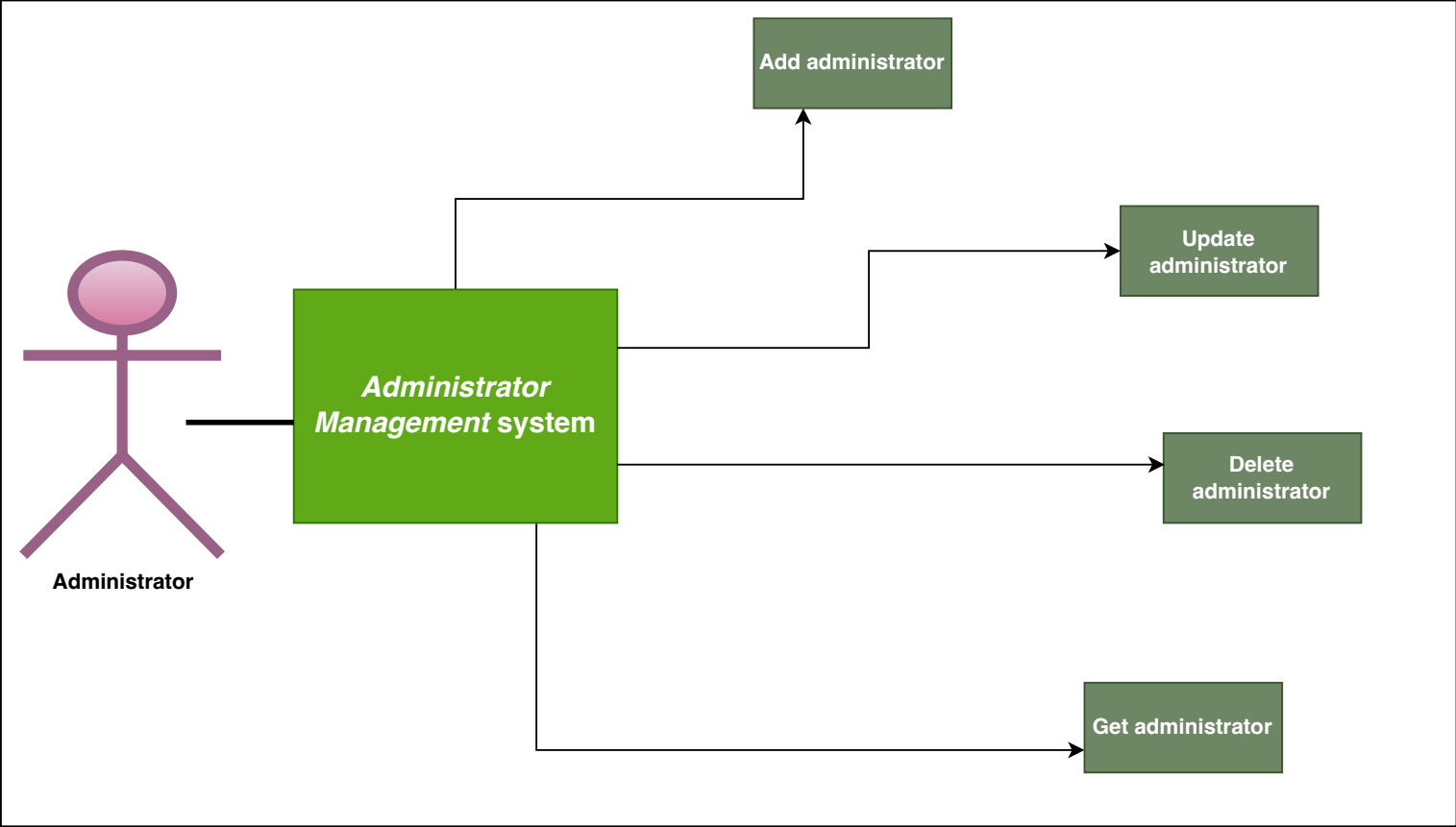
Accommodation Request Management

Search and Filter

Administrator

Search and filter system

Search Student

Filter rooms

System diagram

Student

- studentId: int
- name: String
- contactInfo:String
- id: long
- assignedRoom: Room

+ getDetails(): Student
+ addStudent(student:Student): void
+ updateStudent(student:Student): void
+ deleteStudent(studentId:int): void

Room

- roomId: int
- roomNumber: int
- capacity:int
- availablePlace: int
- assignedStudents: List<Student>

+ getDetails(): Room
+ addRoom(room:Room): void
+ updateRoom(room:Room): void
+ deleteRoom(roomId:int): void

* **1**

*

AccommodationRequest

- requestId: int
- studentId: int
- preferredRoom : Room
- status: String

+ submitRequest(request: AccommodationRequest): void
+ approveRequest(requestId: int): void
+ rejectRequest(requestId: int): void

1

Administrator

- adminId: int
- name: int
- contactInfo: String

+ getDetails(): Administrator
+ addAdmin(admin: Administrator): void
+ updateAdmin(admin:Administrator): void
+ deleteAdmin(adminId:int): void

*

*

1

SearchAndFilter

+ searchStudentByName(name: String): List<Student>
+ searchStudentById(studentId: int): Student
+ searchStudentByRoom(roomNumber: int): List<Student>
+ filterRoomByCapacity(capacity: int): List<Room>
+ filterRoomByAvailability(): List<Room>