Gebze Technical University

Department of Computer Engineering

2022 Spring CSE222

Final Report

Group 4

1. Team Members

- 1801042671 - ÖZLEM SEVRİ
- 1901042604 - ABDULLAH KIRICIOĞLU
- 200104004090 - BURAK CEYLAN
- 1801042625 - HAKAN KESKİN
- 1801042660 - İBRAHİM HİLMİ AKARGÜL
- 200104004117 - YILDIRIM YANIKLARDAN
- 141044022 - MERVE DUR
- 1801042651 - MERVE HORUZ

2. Problem Definition

Due to human nature, people get sick many times throughout their lives and need to go to hospitals to get treated by doctors. Considering the population today, the number of people going to hospitals is quite high. Therefore, patients need to make an appointment and monitor appointment information. It is great of importance that patients can easily access the reports and process of the examination and make an appointment through a single application. In addition to the patients in the hospital, nurses, doctors, counseling, and management staff are of great importance in terms of monitor and manage examination, treatment and management system (registration, observation and report, etc.).

3. Users of the System

- Administrator

Performs administrative operations (add/remove) on hospital employees. Displays patient records, appointments, and employee reports.

- Doctor

By accessing appointment information, patient history and test results, the doctor uploads the necessary evaluations and reports to the system through the records. Besides, he/she can manage his/her appointments.

- Patient

The patient performs the processes of making appointments with specific doctors and hospitals, viewing examination results and doctor evaluations, and managing their appointments.

- Patient admission

Performs the procedures of admitting and enrolling patients who make an appointment on the system to the hospital's patient list.

- Nurse

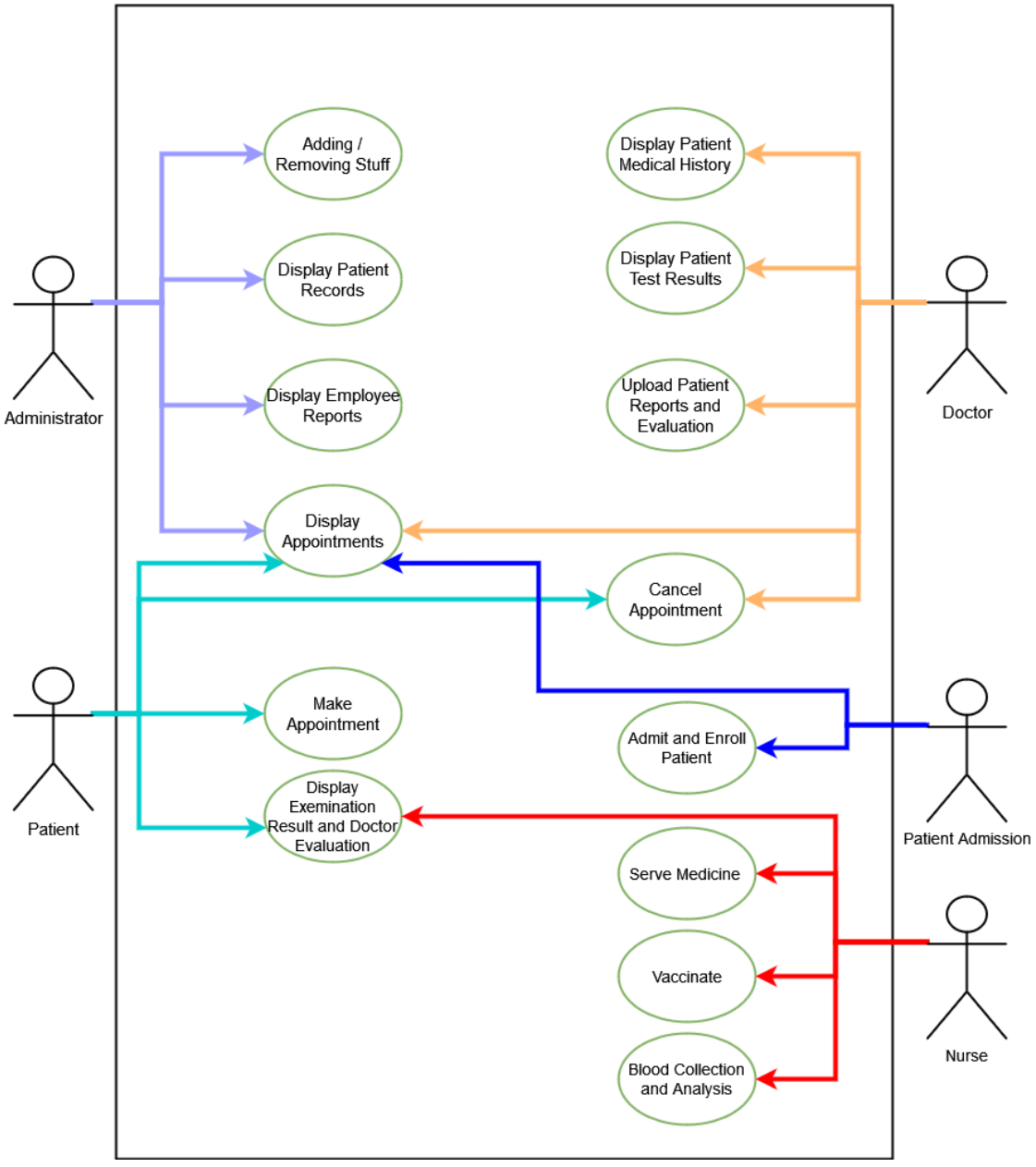
It performs drug service, vaccination, blood collection and analysis services for patients.

4. Requirements In Details

Non-Functional Requirements	
1	The system provides username which is T.C. identification number and special password.
2	The system will be developed using Java programming language.
3	The system data is stored in the text file and initially assigned to the corresponding sections in the system.
4	Viewing operations on the system are performed by restricting the viewer's position on the system within the scope of data security.

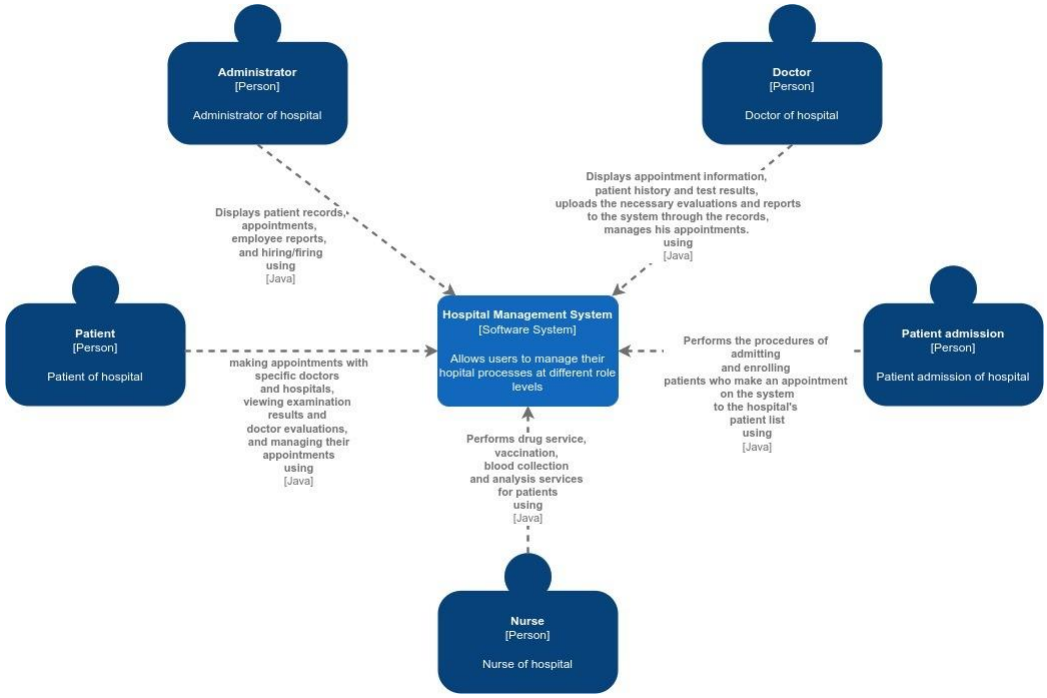
Functional Requirements	
1	The doctor can see the appointments made for him/her and add the patient's evaluation report to the appointment.
2	The doctor can view the health history and test results of his/her patients.
3	The doctor can cancel the appointment of him/her patients.
4	The patient can perform the process of adding and canceling an appointment for himself/herself.
5	The patient can view the appointment evaluations and test results uploaded by the doctor.
6	The patient admission can record patient entry information when the patient arrives at the hospital during an appointment or emergency.
7	Patient admission can register patients to the hospital in case the patient does not have a record.
8	Patient admission can view patient appointments.
9	The nurse can enter the patient's drug, vaccine or test information into the system with the permissions given according to the patient's doctor evaluations.
10	The administrator can add/delete an employee (doctor, nurse or patient admission) to/from the hospital.
11	The administrator can view patient, employee records.
12	The administrator can view patient appointments.

5. Use-Case Diagrams

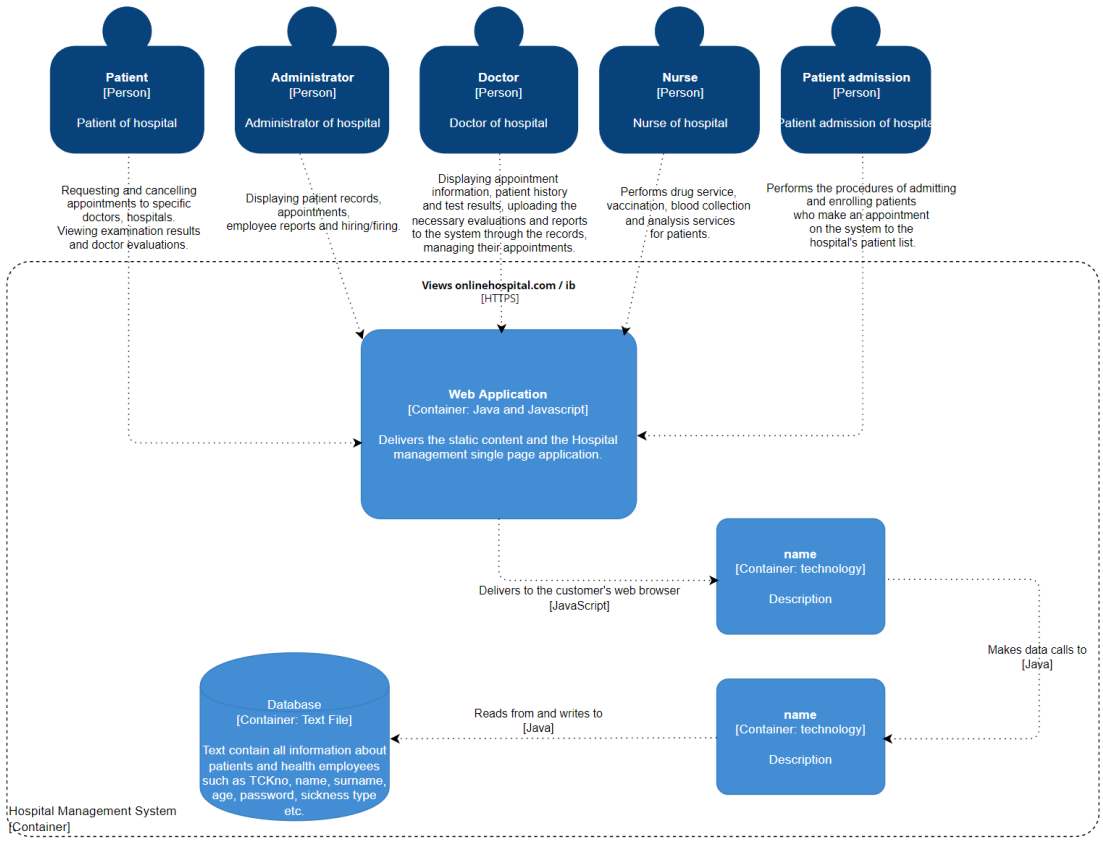


6. The C4 Model of the System

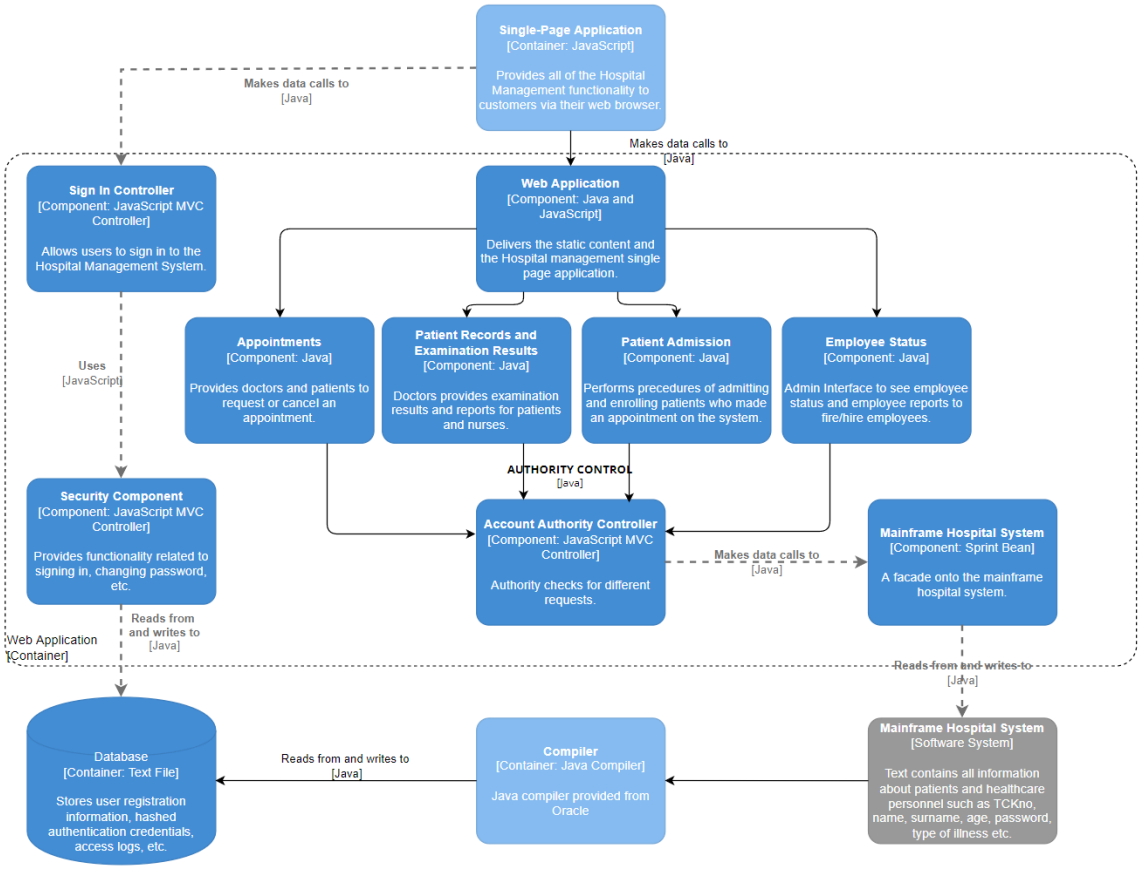
6.1. Level 1 (Context)



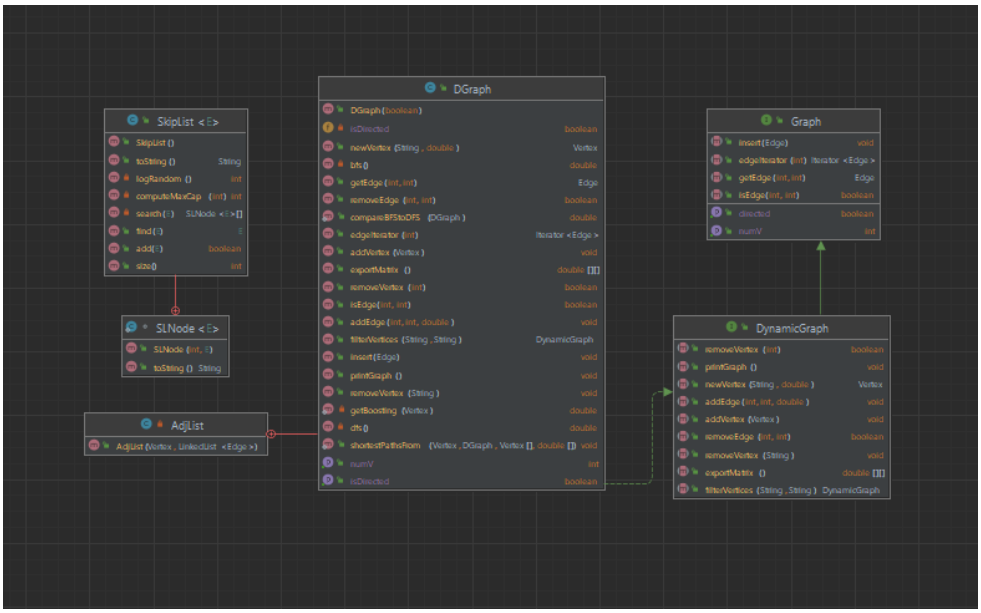
6.2. Level 2 (Container)

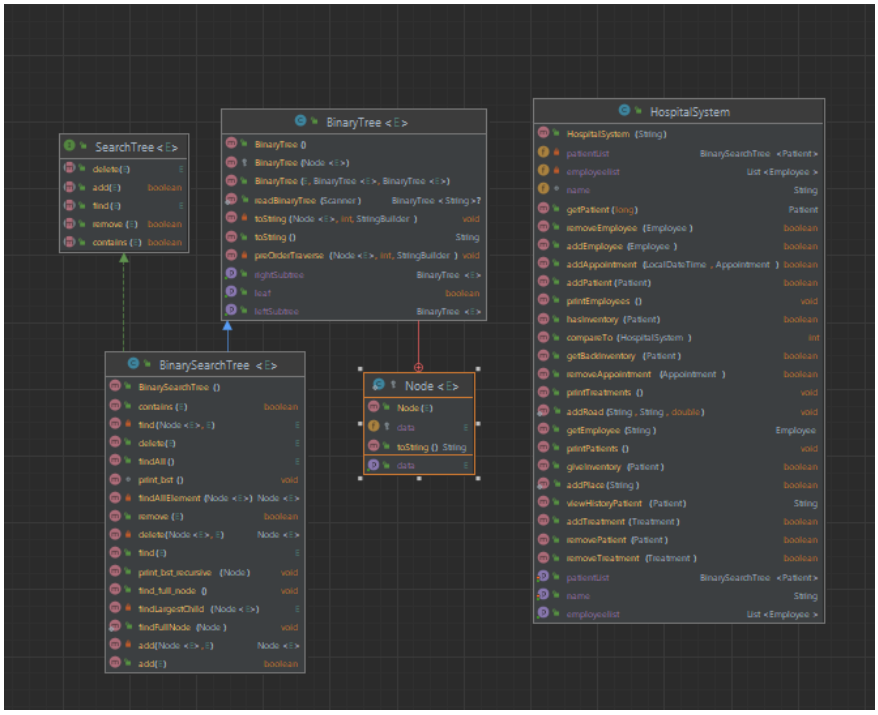


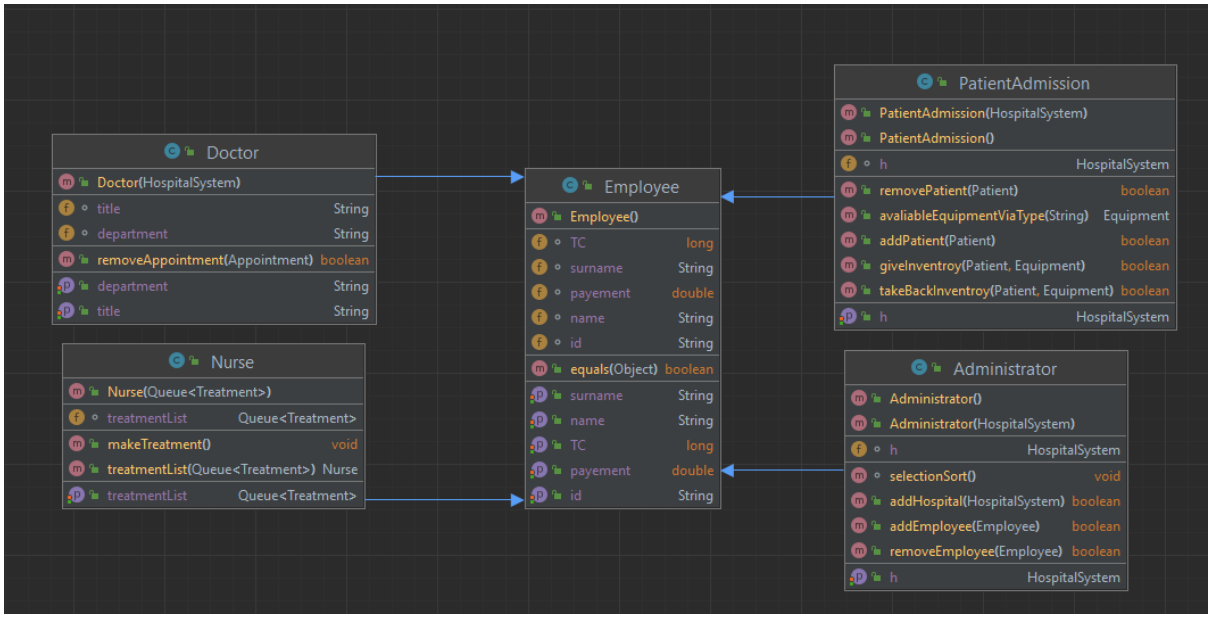
6.3. Level 3 (Component)

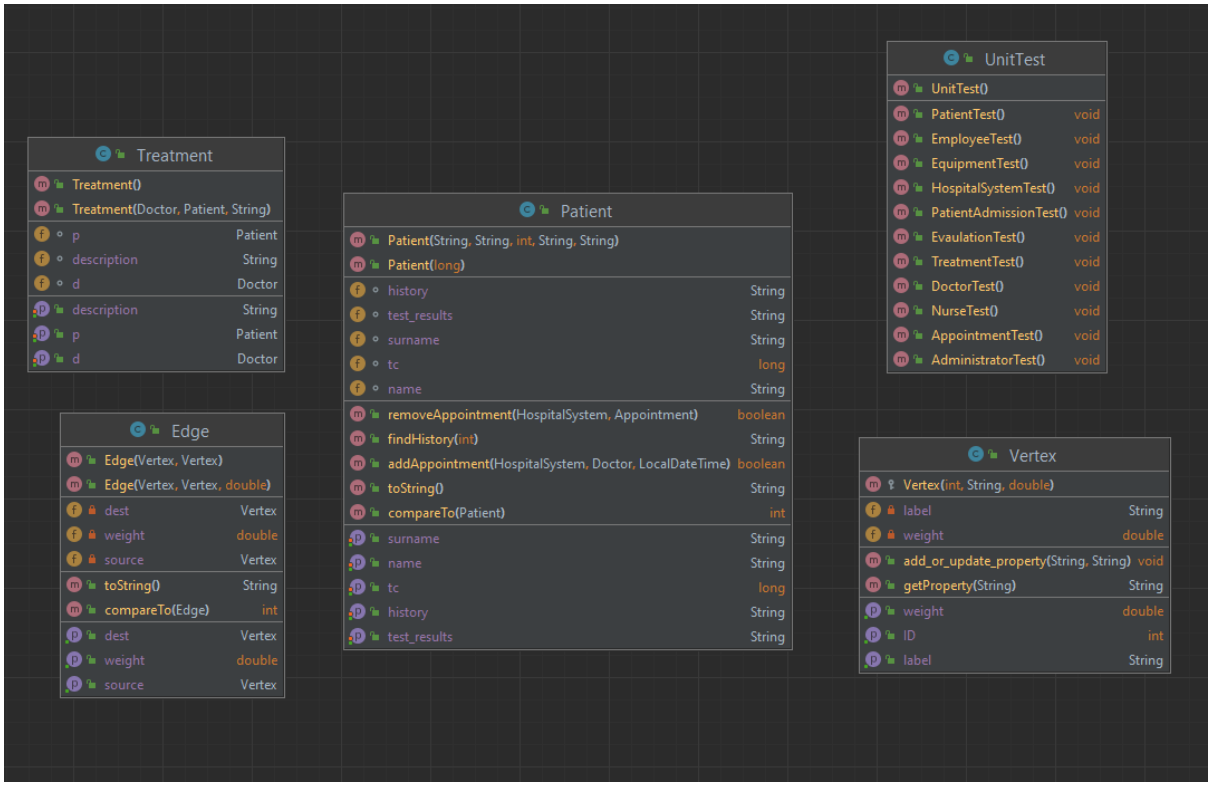


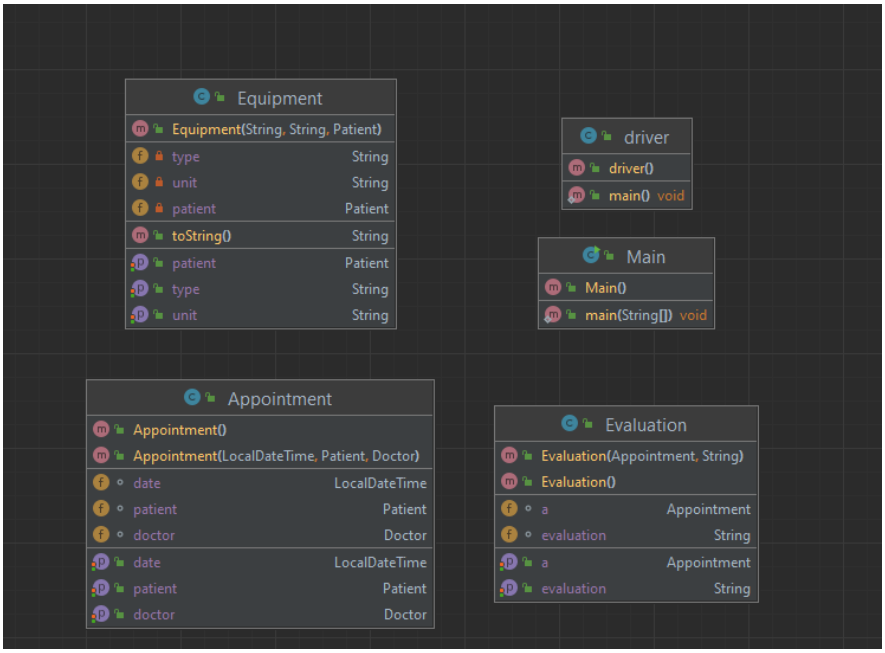
7. Class Diagrams



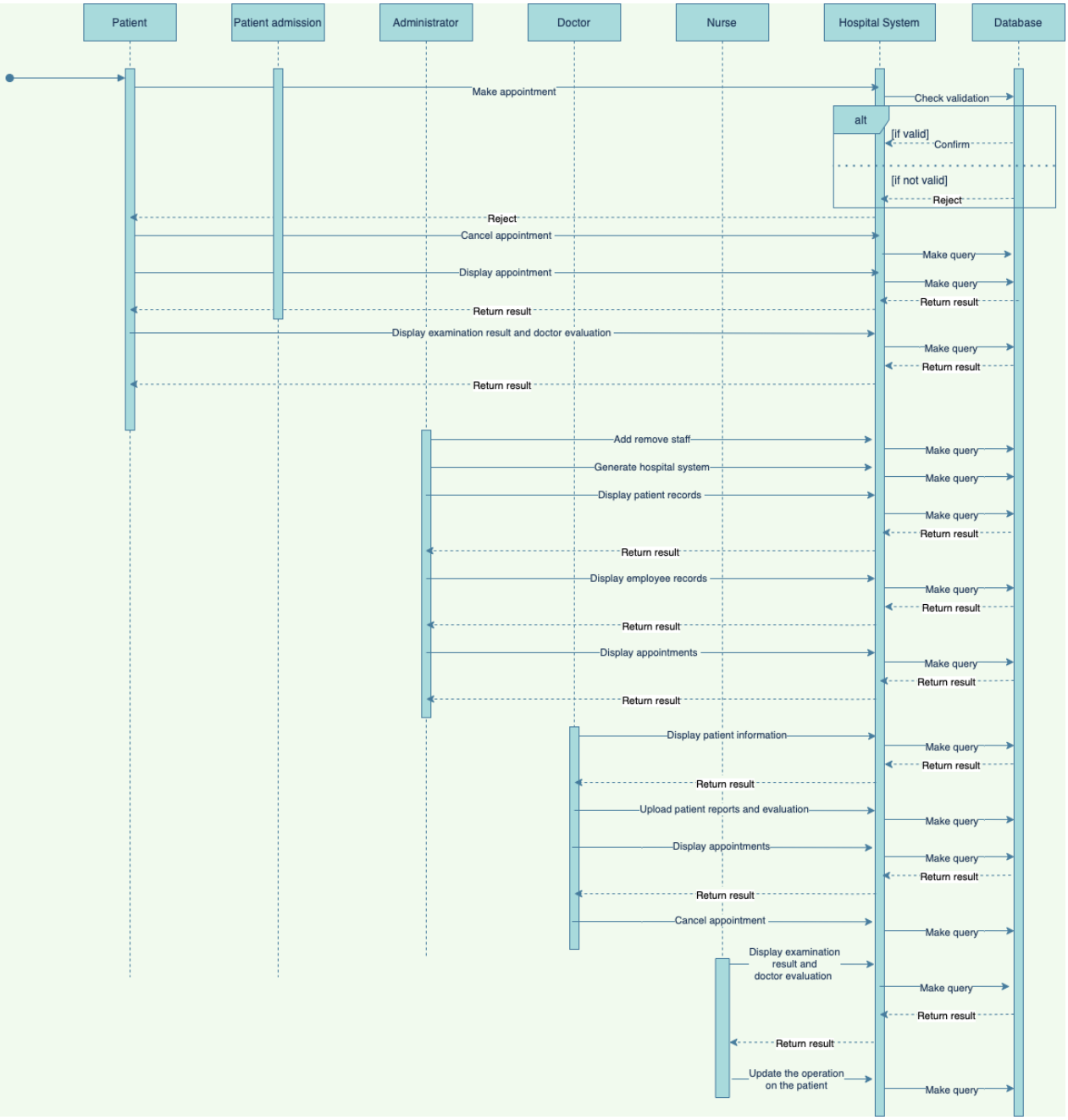

SearchTree <E>
delete(E) E
add(E) boolean
find(E) E
remove(E) boolean
contains(E) boolean
BinaryTree <E>
BinaryTree ()
BinaryTree (Node <E>)
BinaryTree (T, BinaryTree <E>, BinaryTree <E>)
readBinaryTree (Scanner) BinaryTree <String>?
toString (Node <E>, int, StringBuilder) void
toString () String
preOderTraverse (Node <E>, int, StringBuilder) void
rightSubtree BinaryTree <E>
leaf boolean
leftSubtree BinaryTree <E>
HospitalSystem
HospitalSystem (String)
patientList BinarySearchTree <Patient>
employeelist List <Employee>
name String
getPatient (long) Patient
removeEmployee (Employee) boolean
addEmployee (Employee) boolean
addAppointment (LocalDateTime, Appointment) boolean
addPatient (Patient) boolean
printEmployees () void
hasInventory (Patient) boolean
compareTo (HospitalSystem) int
getBackInventory (Patient) boolean
removeAppointment (Appointment) boolean
printTreatments () void
addRoad (String, String, double) void
getEmployee (String) Employee
printPatients () void
giveInventory (Patient) boolean
addPlace (String) boolean
viewHistoryPatient (Patient) String
addTreatment (Treatment) boolean
removePatient (Patient) boolean
removeTreatment (Treatment) boolean
patientList BinarySearchTree <Patient>
name String
employeelist List <Employee>
BinarySearchTree <E>
BinarySearchTree ()
contains(E) boolean
find(Node <E>, E) E
delete(E) E
findAll()
print_bst () void
findAllElement (Node <E>) Node <E>
remove(E) boolean
delete(Node <E>, E) Node <E>
find(E) E
print_bst_recursive (Node) void
find_full_node () void
findLargestChild (Node <E>) E
findFullNode (Node) void
add(Node <E>, E) Node <E>
add(E) boolean
Node <E>
Node(E)
data E
toString() String
data E


Doctor
Doctor(HospitalSystem)
title String
department String
removeAppointment(Appointment) boolean
department String
title String
Nurse
Nurse(Queue<Treatment>)
treatmentList Queue<Treatment>
makeTreatment() void
treatmentList(Queue<Treatment>) Nurse
treatmentList Queue<Treatment>
Employee
Employee()
TC long
surname String
payement double
name String
id String
equals(Object) boolean
surname String
name String
TC long
payement double
id String
PatientAdmission
PatientAdmission(HospitalSystem)
PatientAdmission()
h HospitalSystem
removePatient(Patient) boolean
avaliableEquipmentViaType(String) Equipment
addPatient(Patient) boolean
givelnventroy(Patient, Equipment) boolean
takeBackInventroy(Patient, Equipment) boolean
h HospitalSystem
Administrator
Administrator()
Administrator(HospitalSystem)
h HospitalSystem
selectionSort() void
addHospital(HospitalSystem) boolean
addEmployee(Employee) boolean
removeEmployee(Employee) boolean
h HospitalSystem

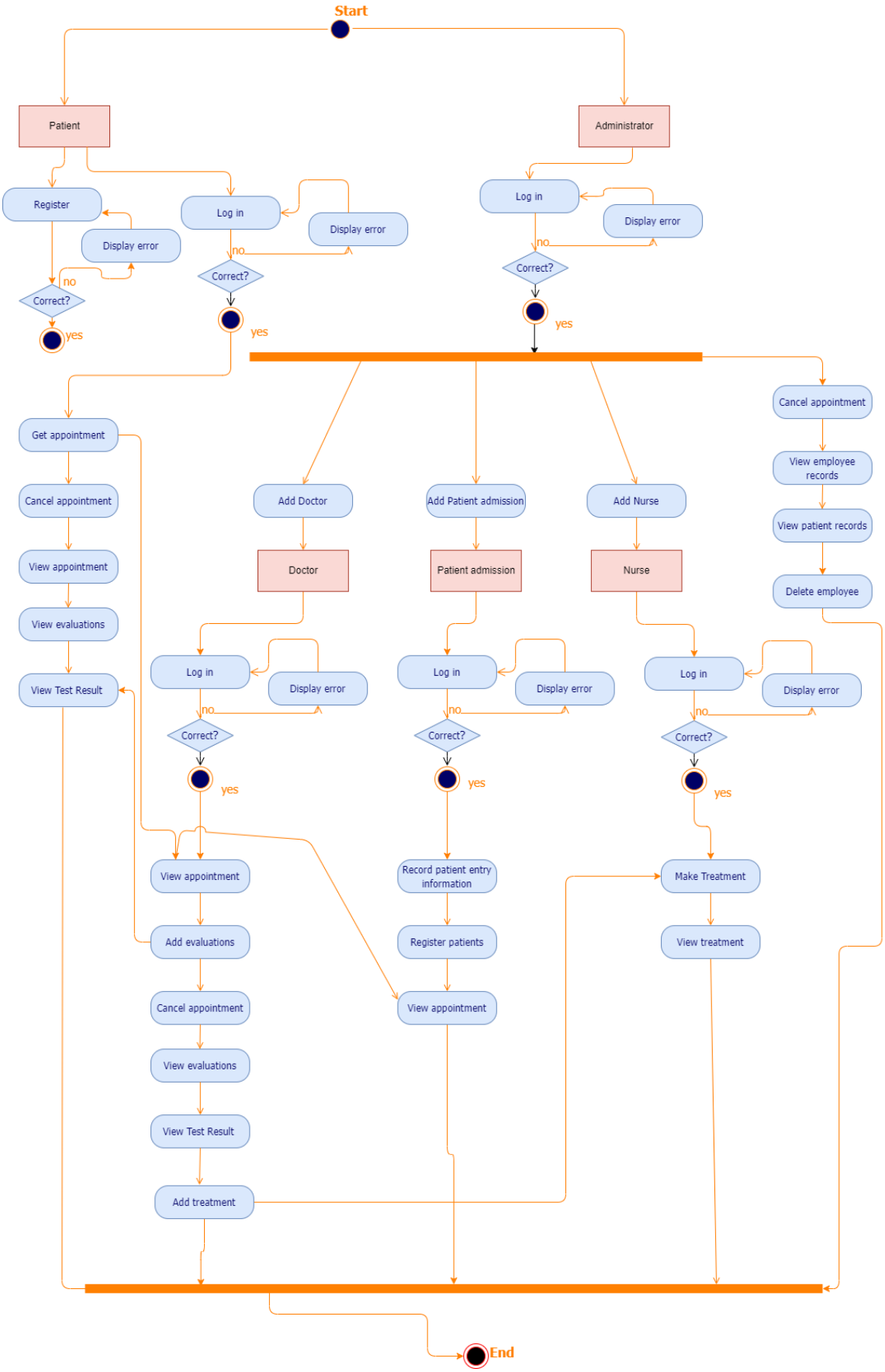

Treatment
Treatment()
Treatment(Doctor, Patient, String)
p Patient
description String
d Doctor
description String
p Patient
d Doctor
Edge
Edge(Vertex, Vertex)
Edge(Vertex, Vertex, double)
dest Vertex
weight double
source Vertex
toString() String
compareTo(Edge) int
dest Vertex
weight double
source Vertex
Patient
Patient(String, String, int, String, String)
Patient(long)
history String
test_results String
surname String
tc long
name String
removeAppointment(HospitalSystem, Appointment) boolean
findHistory(int) String
addAppointment(HospitalSystem, Doctor, LocalDateTime) boolean
toString() String
compareTo(Patient) int
surname String
name String
tc long
history String
test_results String
UnitTest
UnitTest()
PatientTest() void
EmployeeTest() void
EquipmentTest() void
HospitalSystemTest() void
PatientAdmissionTest() void
EvaulationTest() void
TreatmentTest() void
DoctorTest() void
NurseTest() void
AppointmentTest() void
AdministratorTest() void
Vertex
Vertex(int, String, double)
label String
weight double
add_or_update_property(String, String) void
getProperty(String) String
weight double
ID int
label String


Equipment
Equipment(String, String, Patient)
type String
unit String
patient Patient
toString() String
patient Patient
type String
unit String
driver
driver()
main() void
Main
Main()
main(String[]) void
Appointment
Appointment()
Appointment(LocalDateTime, Patient, Doctor)
date LocalDateTime
patient Patient
doctor Doctor
date LocalDateTime
patient Patient
doctor Doctor
Evaluation
Evaluation(Appointment, String)
Evaluation()
a Appointment
evaluation String
a Appointment
evaluation String

8. Sequence Diagram



9. Activity Diagram



10. Implementations Details

- **List**

We kept the employees as an Array List to use adding removing and searching operations through employees inside administrator.

- **Stack/Queue**

Some patients need to get treatment. We kept these treatments inside the nurse structure as a queue to make sure it performs respectively.

- **Binary Search Tree**

There patients inside the hospital and wee kept them in binary search tree inside the hospital system to make easier to search patients.

- **Priority Queue**

All appointment given to patients stored inside priority queue according to its date.

- **Set-Map**

We use an equipment structure for represent different kind of helping material of the patients inside the hospital. We store these equipment's using set and map structures. In set structure we kept patients who takes any equipment from hospital. We use map for hold the stock information about equipment.

- **Skip List**

In administrator structure we kept hospitals as a skip list to manage hospitals from administrator.

- **Graph**

We use graph for finding the distance between hospitals.

- **Balanced Binary Search Tree**

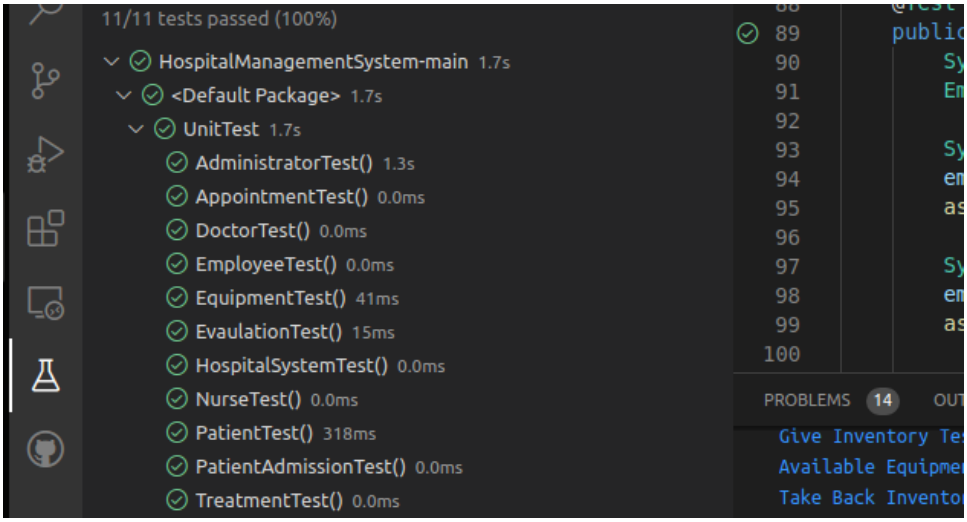
We kept patient appointments also inside the patient structure as a Tree Set.

- **Selection Sort**

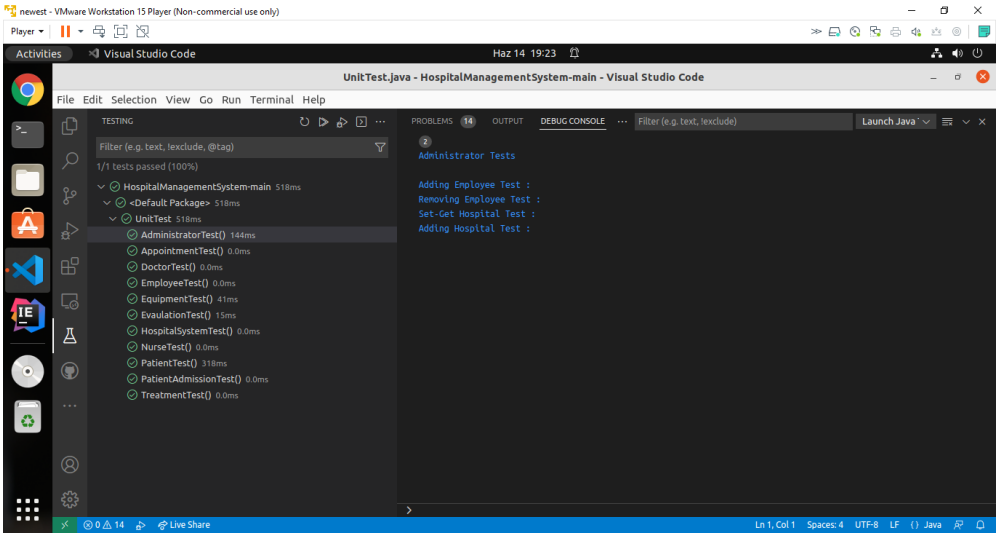
We use selection sort algorithm to sort the equipment list.

11. Test Cases

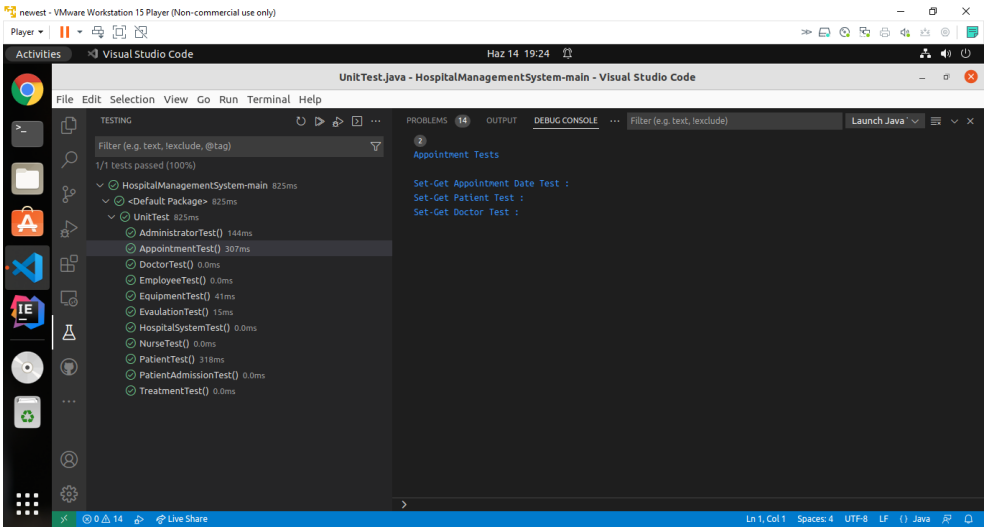
We used Junit to make the tests faster and more reliable. We wrote 11 methods for 11 classes (each method tests all the methods of the related class). You can see all methods(all classes) are passed the test in the first picture. The next pictures show each method separately with the test methods in it.



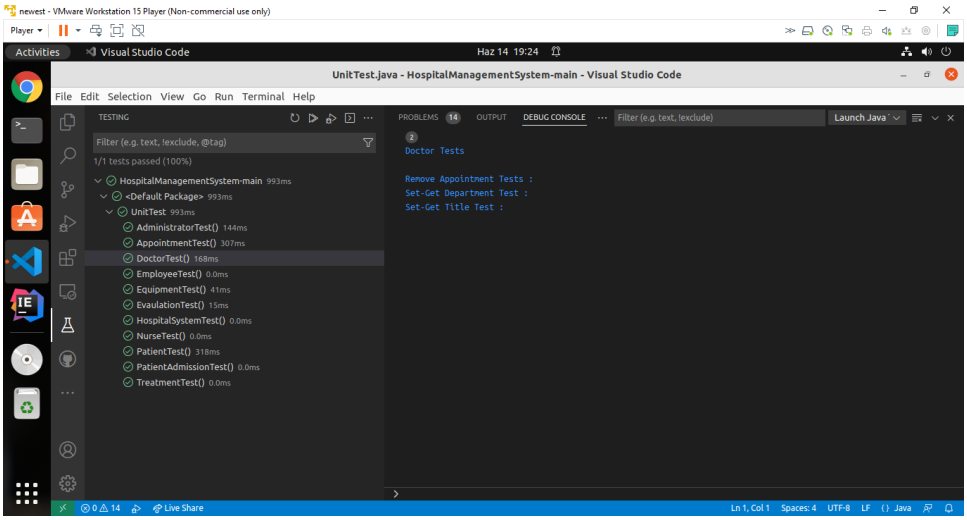
All Tests



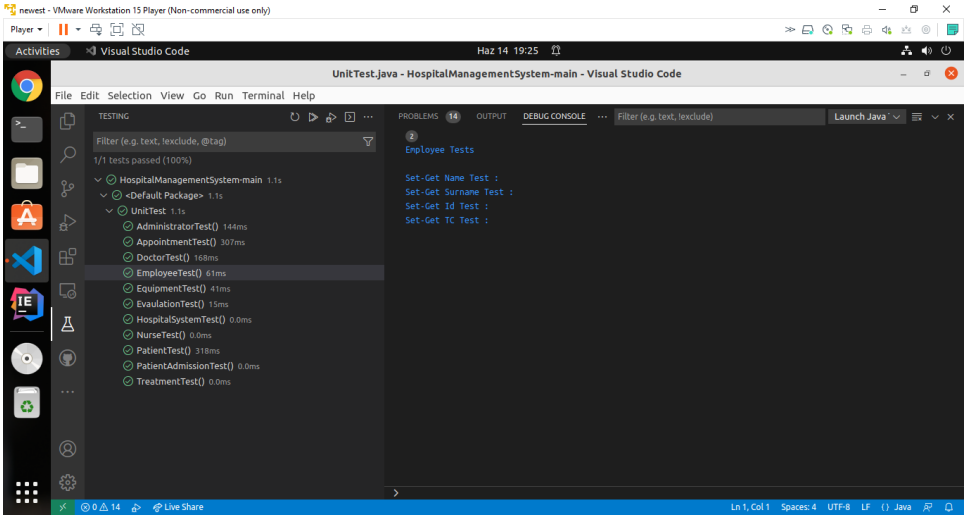
Administrator Tests



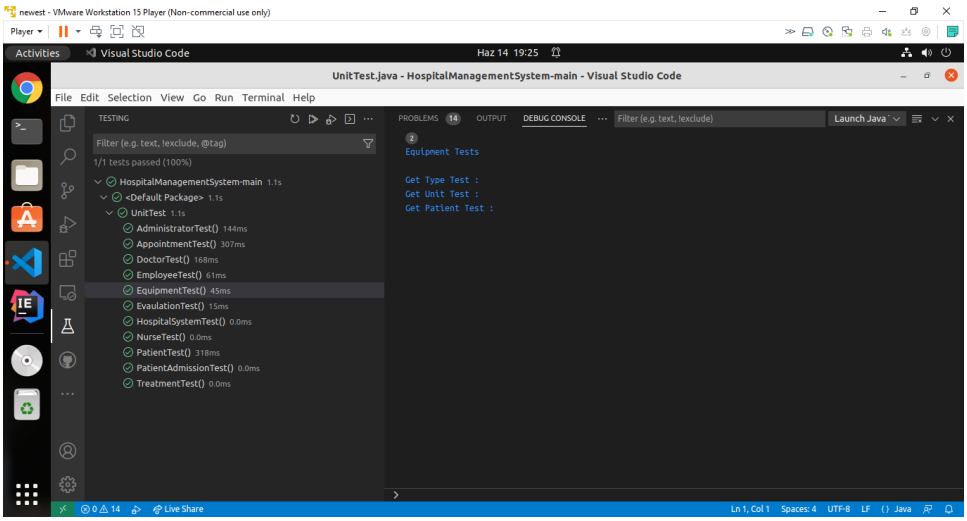
Appointment Tests



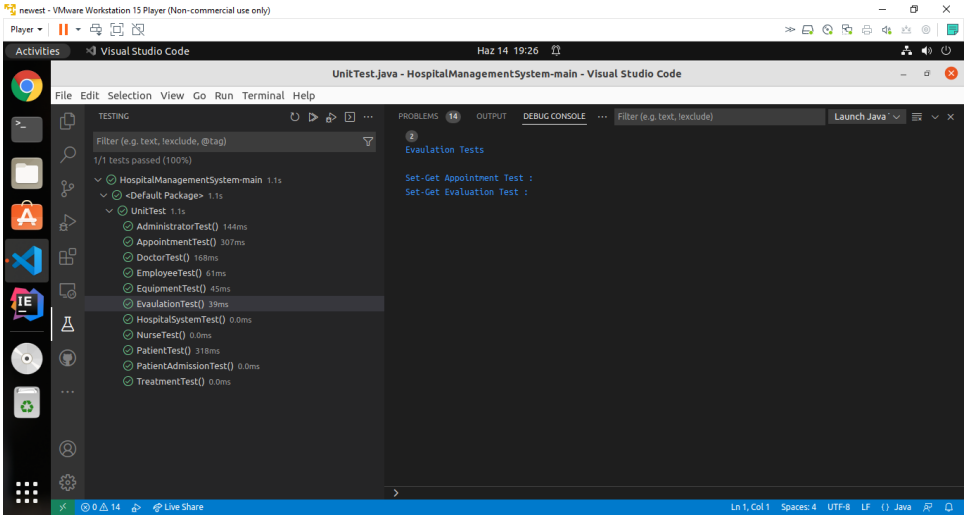
Doctor Tests



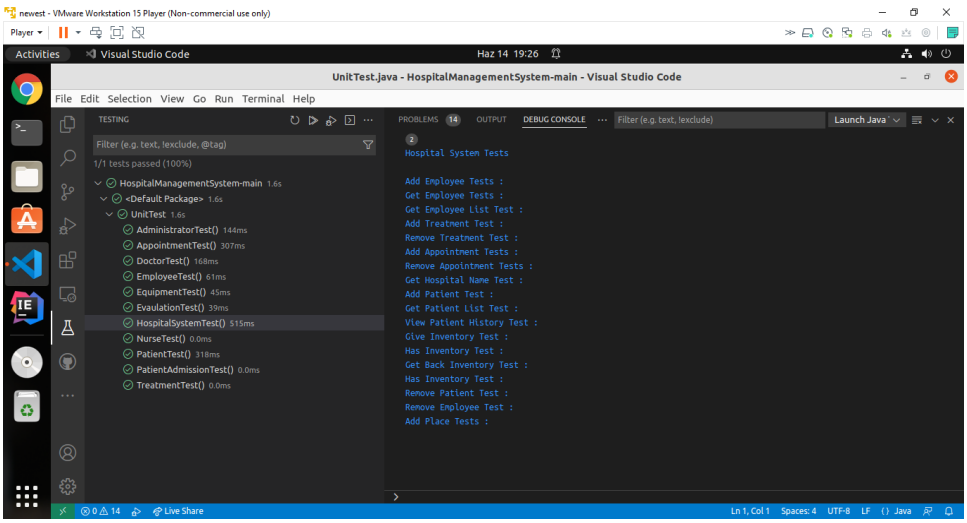
Employee Tests



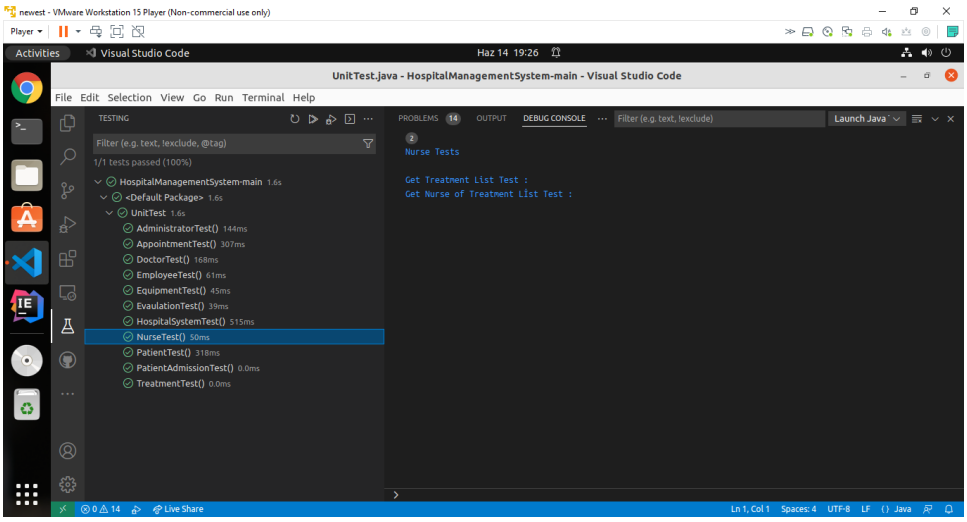
Equipment Tests



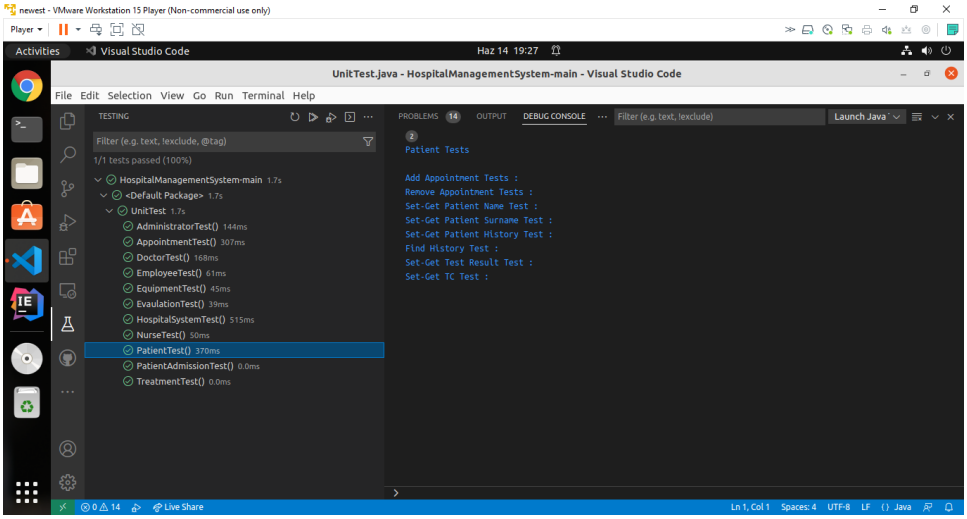
Evaulation Tests



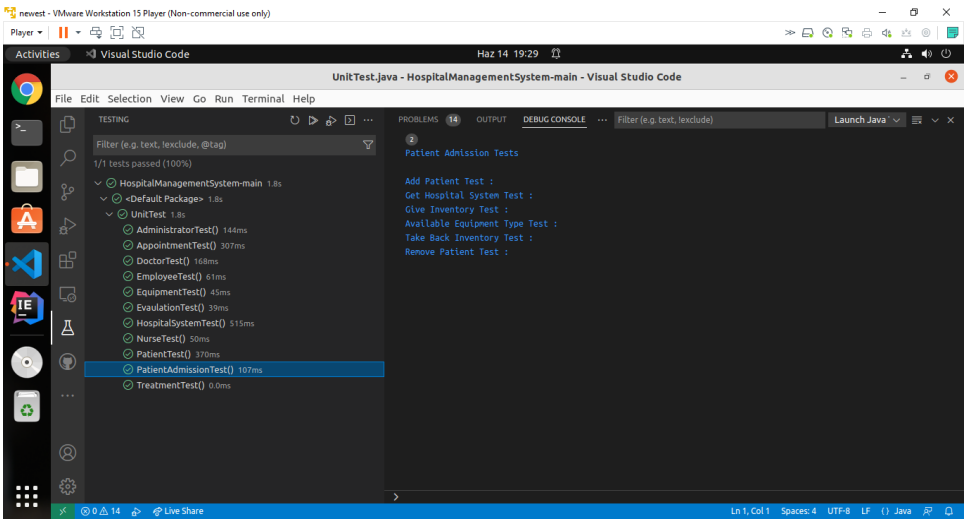
Hospital System Tests



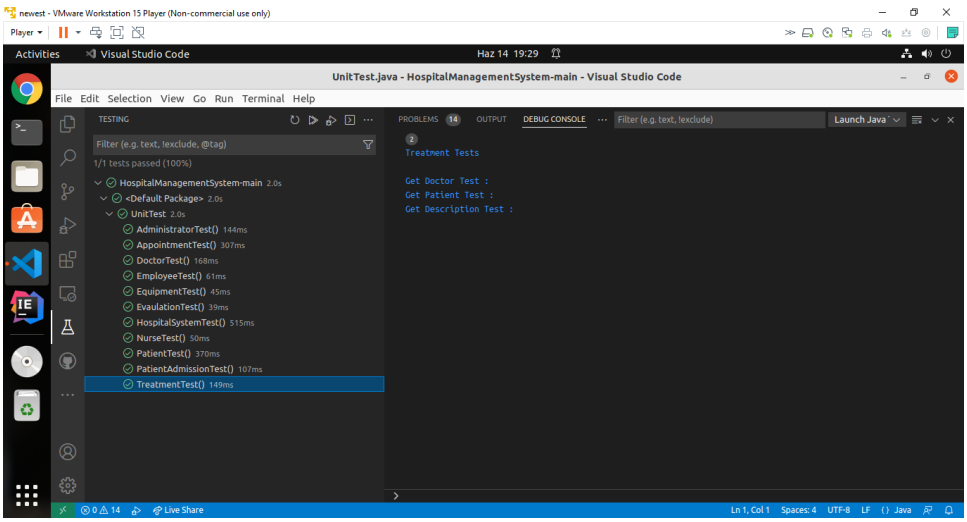
Nurse Tests



Patient Tests



Patient Admission Tests



Treatment Tests

Test Case No	Test Case	Input	Expected Outputs	Test Data	Resulting Outputs	Situation	Error No	Error
1	Administrator can display registered patient, employee records.	Patient/Employee name, surname, id number	Patient/Employee record is displayed	Patient name: Ozlem Surname: Sevri Id:1801042671	Patient/Employee record is displayed	Successful	-	-
2	Administrator try to display unregistered patient, employee records.	Patient/Employee name, surname, id number	“No such patient/employee” message should be displayed	Patient name: Ozlem Surname: Sevri Id:1111111111	Message is displayed.	Successful	-	-
3	Administrator can add employee	Employee	Employee should be added	Employee: employee	Employee is added	Successful	-	-
4	Administrator can try to add existing employee	Employee	“Employee is already added” message should be displayed	Employee: employee	Message is displayed	Successful	-	-
5	Administrator can delete existing employee	Employee	Employee should be deleted	Employee: employee	Employee is deleted	Successful	-	-
6	Administrator can try to delete employee who does not exist	Employee	“There is no such an employee” message should be displayed	Employee: employee	Message is displayed	Successful	-	-
7	Administrator can display appointments	Doctor/Patient/Date	Appointments should be displayed	Doctor: doctor/ Patient: patient /Date: date	Appointments are displayed	Successful	-	-
8	Doctor can display patient medical history	Patient Name, Surname, Id Number	Medical history of patient should be displayed	Patient Name: Ozlem Surname: Sevri Id:1801042671	Medical history of patient is displayed	Successful	-	-
9	Doctor can try to display medical history of patient that does not exist	Patient Name, Surname, Id Number	“No such patient” message should be displayed	Patient Name: Ozlem Surname: Sevri Id:1111111111	Message is displayed	Successful	-	-
10	Doctor can display patient test results	Patient Name, Surname, Id Number	Test results of patient should be displayed	Patient Name: Ozlem Surname: Sevri Id:1801042671	Test results of patient is displayed	Successful	-	-
11	Doctor can display appointments	Date	Appointments should be displayed	Date: date	Appointments are displayed	Successful	-	-
12	Doctor can cancel	Date	Appointment should be canceled	Date: date	Appointment is canceled	Successful	-	-

	appointment							
13	Doctor can try to cancel appointment which does not exist	Date	"There is no such an appointment" message should be displayed	Date: date	Message is displayed	Successful	-	-
14	Patient can display appointments	Date	Appointments should be displayed	Date: date	Appointments are displayed	Successful	-	-
15	Patient can cancel appointment	Date	Appointment should be canceled	Date: date	Appointment is canceled	Successful	-	-
16	Patient can try to cancel appointment which does not exist	Date	"There is no such an appointment" message should be displayed	Date: date	Message is displayed	Successful	-	-
17	Patient can make appointment	Doctor, Date	Appointment should be made.	Doctor: doctor Date: date	Appointment is made.	Successful	-	-
18	Patient try to make appointment while there is an appointment or no such a doctor.	Doctor, Date	"There is already an appointment" message should be displayed.	Doctor: doctor Date: date	Message is displayed	Successful	-	-
19	Patient admission can display appointments	Doctor/Patient/Date	Appointments should be displayed	Doctor: doctor/ Patient: patient /Date: date	Appointments are displayed	Successful	-	-

12. Performance Analysis

13.Function Name 14.	Function Description	Time Complexity	Explanation
Administrator.addEmployee	Add the employee to the list.	O(n), where n is the number of employees.	Search in ArrayList takes O(n) time, adding takes amortized constant time.
Administrator.removeEmployee	Removes the employee from the list.	θ(n), where n is the number of employees.	Both search and shift operations on ArrayList takes O(n) time.

BinarySearchTree.contains	Returns true if the item is included in the tree.	Average O(log(n)), where n is the number of elements.	Tree is balanced in average case, so the search operation takes O(log(n)) time.
BinarySearchTree.remove	If the item is in the tree, removes it and returns true.	Average O(log(n)), where n is the number of elements.	Tree is balanced in average case, so the remove operation takes O(log(n)) time.
BinarySearchTree.find	Searches for the item in the tree and returns a reference to it if found.	Average O(log(n)), where n is the number of elements.	Tree is balanced in average case, so the search operation takes O(log(n)) time.
BinarySearchTree.add	If the item is not included in the tree, adds it and returns true.	Average O(log(n)), where n is the number of elements.	Tree is balanced in average case, so the add operation takes O(log(n)) time.
BinarySearchTree.delete	If the item is included in the tree, deletes and returns a reference to it.	Average O(log(n)), where n is the number of elements.	Tree is balanced in average case, so the delete operation takes O(log(n)) time.
BinarySearchTree.print_bst	Prints the contents of the tree making an inorder traversal.	$\theta(n)$, where n is the number of elements.	Assuming that printing 1 element takes constant time, inorder traversal and printing operations takes $\theta(n)$ time.
BinaryTree.getLeftSubtree	Returns the left subtree of this tree.	$\theta(1)$.	Simple statements without any loops.
BinaryTree.getRightSubtree	Returns the right subtree of this tree.	$\theta(1)$.	Simple statements without any loops.
BinaryTree.isLeaf	Returns true if the root has no child.	$\theta(1)$.	Simple statements without any loops.
BinaryTree.toString	Returns the string representation of the tree.	Average $\theta(n.\log(n))$, where n is the number of elements.	Assuming that printing 1 element takes constant time, printing all the elements with a variable number of leading spaces depending on the depth takes $\theta(n.\log(n))$

			time. (Approximately n operations for each of log(n) level.)
BinaryTree.readBinaryTree	Constructs a BinaryTree and returns a reference to it.	θ(n), where n is the number of elements / lines to be read.	Reading data proportional to the size of the tree and copying it takes θ(n) time.
Doctor.removeAppointment	Removes the appointment from the priority queue.	O(n), where n is the number of appointments in the priority queue.	Searching an element in PriorityQueue (and removing it if found) takes O(n) time. (Removal takes O(log(n)) time which is dominated by O(n).)
HospitalSystem.addEmployee	Adds the employee to the list if it is not already included.	O(n), where n is the number of employees in the list.	Searching in the ArrayList (and adding the employee if not found) takes O(n) time.
HospitalSystem.getEmployee	Returns a reference to the employee if it is in the list.	O(n), where n is the number of employees in the list.	Searching in the ArrayList takes O(n) time.
HospitalSystem.addTreatment	Adds the treatment to the queue.	θ(1).	Adding an element to the end of a LinkedList takes constant time.
HospitalSystem.removeTreatment	Removes the treatment from the queue.	O(n), where n is the number of treatments in the queue.	Searching an element in the LinkedList (and removing it if found) takes O(n) time for a LinkedList.
HospitalSystem.addAppointment	If possible, adds the appointment to the priority queue and returns true.	O(n+log(m)), where n is the number of employees in the list and m is the number of appointments in the priority queue.	Searching an element (Doctor) in ArrayList takes O(n) time proportional to size of the list. Adding an element (Appointment) in PriorityQueue takes O(log(m)) time

			proportional to size the queue.
HospitalSystem.removeAppointment	Removes the appointment from the priority queue.	O(n), where n is the number of appointments in the priority queue.	In a PriorityQueue, searching takes O(n) time and removal takes O(log(n)) time, which is dominated by the search operation.
HospitalSystem.printTreatments	Prints the treatments in the queue.	θ(n), where n is the number of treatments in the queue.	Assuming that printing an element takes constant time, traversing through the LinkedList and printing all the elements takes θ(n) time.
HospitalSystem.printPatients	Prints the patients in the tree.	θ(n), where n is the number of patients in the tree.	Converting a binary search tree to string (and printing it) takes θ(n) time.
HospitalSystem.printEmployees	Prints the employees in the list.	θ(n), where n is the number of employees in the list.	Assuming that printing an element takes constant time, printing all the elements in an ArrayList takes θ(n) time.
HospitalSystem.addPatient	Adds the patient to the tree and returns true if possible.	Average O(log(n)), where n is the number of patients.	In average case, the tree is balanced, so the add operation takes O(log(n)) time.
HospitalSystem.removePatient	Removes the patient from the tree if is found.	Average O(log(n)), where n is the number of patients.	In average case, the tree is balanced, so the remove operation takes O(log(n)) time.
HospitalSystem.viewHistoryPatient	Returns the patient history of the specified patient.	θ(1).	Simple statements without any loops.
HospitalSystem.getPatient	Returns a reference to the patient whose TC is given if found.	Average O(log(n)), where n is the number of patients.	In average case, the tree is balanced, so the search operation takes O(log(n)) time.

Nurse.makeTreatment	Removes the first treatment from the treatment queue.	$\theta(1)$.	Removing the first element from LinkedList takes constant time.
Patient.addAppointment	Makes an appointment with the given parameters and adds it to the containers if possible.	O(n+log(m)+log(k)), where n is the number of employees in the list, m is the number of appointments in the priority queue and k is the number of appointments in the tree.	Uses HospitalSystem.addAppointment which takes O(n+log(m)) time. Additionally, adds the appointment in a TreeSet which takes O(log(k)) time.
Patient.removeAppointment	Removes the appointment from the containers if possible.	O(n+log(k)), where n is the number of appointments in the priority queue and k is the number of appointments in the TreeSet.	Uses HospitalSystem.removeAppointment which takes O(n) time. Additionally, removes the appointment in the TreeSet which takes O(log(k)) time.
PatientAdmission.addPatient	If possible, adds the patient to the tree and return true.	Average O(log(n)), where n is the number of patients in the tree.	Uses HospitalSystem.addPatient, which takes O(log(n)) time.
PatientAdmission.removePatient	Removes the patient from the tree if it is found.	Average O(log(n)), where n is the number of patients in the tree.	Uses HospitalSystem.removePatient which takes O(log(n)) time.