HEALTHCARE PLATFORM : HEALTHY BUFFS

1) Team Members: Payoj Jain, Rahul Chowdhury, Ganesh Chandra Satish
Team Number: 33

Title of the Project: HealthyBuffs
Vision: A platform to search for doctors, pharmacies, pathologies easily. This product helps people to book appointment with doctors and pathologies instantly. People can also consult doctors online by asking over the platform and get doctors' answers.
Project Description: Implement Object Oriented Design Pattern for following use cases:

- Search doctors and look doctor details
- Book appointment with doctors
- Look up Pathologies and Pharmacies located in the city
- Update/View Medical Records for patients
- Lookup all the medical records on a single page
- Consult online: Ask general queries online

2) FEATURES THAT WERE IMPLEMENTED

Business Requirements:	
ID	**Title**
BR002	All doctors, pharmacists and pathologists should have names and address
BR003	A patient can make/edit only a single appointment to a particular doctor until that appointment time is passed.
BR004	The appointment duration can be 30 minutes.
BR006	A user can only view his/her medical history.
BR007	Pharmacies and Pathologies can view patients' history only when a doctor/customer allows.

User Requirements:	
UR001	Patients can search pharmacists and pathologists based on name or location
UR002	Patient can search doctors based on specialisation and name or location

UR003	Patients can search for medicines availability in pharmacies based on pharmacies’ names, distance
UR004	Patient can view available time slots to make appointments for a particular doctor
UR005	A patient can make an appointment to a doctor in the available time slot
UR006	Patient can view his/her past and future appointments
UR007	Patient can view medical history
UR008	Patient can view lab reports directly in the platform
UR009	Patient can ask queries to a doctor as a mail or a drop message
UR0010	Users can register/Sign up
UR0011	Doctors can view appointments for the day/week/month
UR0013	Doctors can view/update patients’ medical records
UR0015	Doctors can reply to patients’ query received on the platform
UR0017	Pharmacies can lookup patients’ medical history and previous medicine purchases before giving them medicines
UR0019	Pathologies can view patients’ medical history
UR0020	Pathologies can upload patients’ tests results
UR0023	Admin can view all the registered users to have a collective view

Functional Requirements	
ID	**Requirements**
FR001	System should not allow users to enter the system without proper authentication.
FR003	A patient cannot be allowed to book an appointment slot of a doctor which is already booked by another patient
FR004	System should stop taking appointments from any patients if the doctor has exceeded appointment hours for the day or takes a day off

FR005	System should notify patients whenever a doctor cancels an appointment due to a personal reason

Non Functional Requirements	
ID	**Requirements**
NFR01	Medical records must be secured and access control must be maintained
NFR02	Patient medical history must be available to doctors, pharmacists, pathologies etc. without observable latency (<10 ms)
NFR03	User account information and password must be stored in a secure manner
NFR04	Patients must be able to schedule an appointment using a calendar at the first page on the dashboard
NFR05	Any data from the database must be returned correctly or not at all. Incorrect data should not be returned at any cost

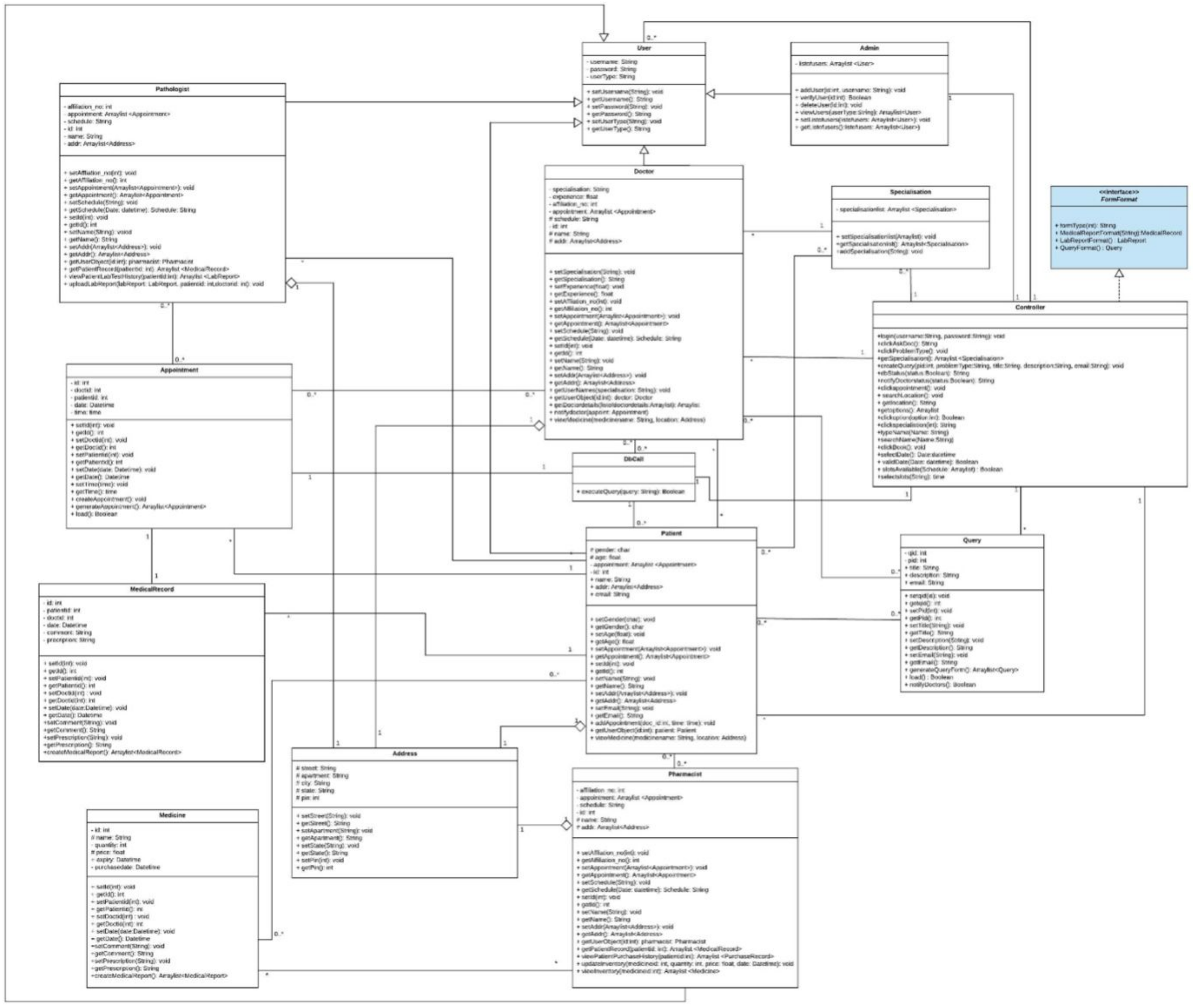
3) FEATURES THAT WERE NOT IMPLEMENTED

Business Requirements:	
ID	**Title**
BR001	Patients' login should be using CU IdentityKey.
BR005	Only Admin can sign up doctors/pharmacists/pathologists after verification.

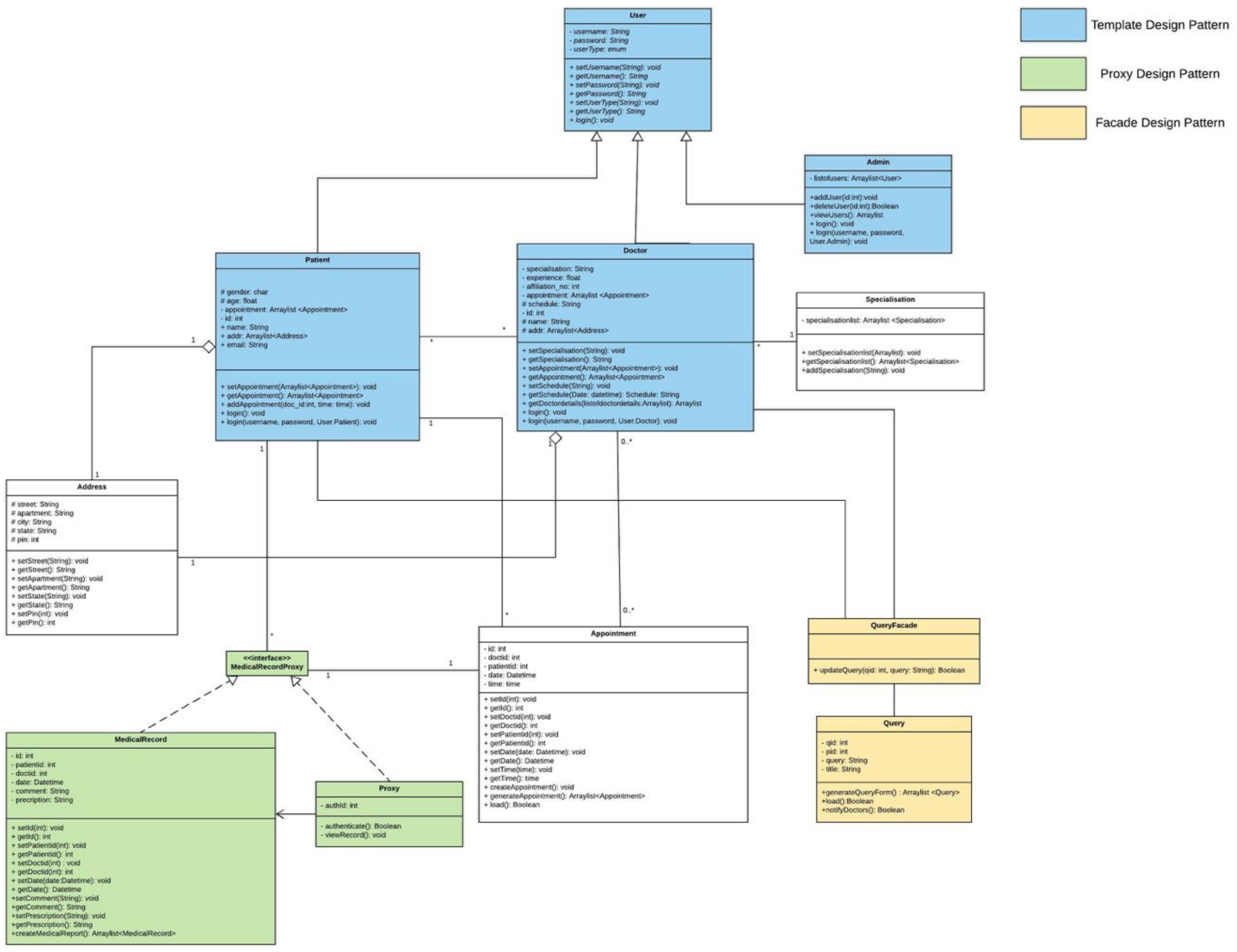
User Requirements:	
UR0012	Doctors can cancel appointments made by patients.
UR0016	Pharmacies can update their medicines availability.
UR0018	Pharmacies can update patients' medicine purchases.
UR0021	Admin should first verify doctors,pharmacies and pathologies before signing them up in the system
UR0022	Admin can delete doctors, patients, pharmacies or pathologies

Functional Requirements	
ID	**Requirements**
FR002	System should alert the pharmacist when the medicine stock gets over.

4) Part 2 class diagram:



Final Class Diagram


User
- username: String
- password: String
- userType: enum
+ setUsername(String): void
+ getUsername(): String
+ setPassword(String): void
+ getPassword(): String
+ setUserType(String): void
+ getUserType(): String
+ login(): void
Template Design Pattern
Proxy Design Pattern
Facade Design Pattern
Admin
- listofusers: Arraylist<User>
+addUser(id:int):void
+deleteUser(id:int):Boolean
+viewUsers(): Arraylist
+ login(): void
+ login(username, password, User.Admin): void
Patient
# gender: char
# age: float
- appointment: Arraylist <Appointment>
- id: int
+ name: String
+ addr: Arraylist<Address>
+ email: String
+ setAppointment(Arraylist<Appointment>): void
+ getAppointment(): Arraylist<Appointment>
+ addAppointment(doc_id:int, time: time): void
+ login(): void
+ login(username, password, User.Patient): void
Doctor
- specialisation: String
- experience: float
- affiliation_no: int
- appointment: Arraylist <Appointment>
# schedule: String
- id: int
# name: String
# addr: Arraylist<Address>
+ setSpecialisation(String): void
+ getSpecialisation(): String
+ setAppointment(Arraylist<Appointment>): void
+ getAppointment(): Arraylist<Appointment>
+ setSchedule(String): void
+ getSchedule(Date: datetime): Schedule: String
+ getDoctordetails(listofdoctordetails:Arraylist): Arraylist
+ login(): void
+ login(username, password, User.Doctor): void
Specialisation
- specialisationlist: Arraylist <Specialisation>
+ setSpecialisationlist(Arraylist): void
+getSpecialisationlist(): Arraylist<Specialisation>
+addSpecialisation(String): void
Address
# street: String
# apartment: String
# city: String
# state: String
# pin: int
+ setStreet(String): void
+ getStreet(): String
+ setApartment(String): void
+ getApartment(): String
+ setState(String): void
+ getState(): String
+ setPin(int): void
+ getPin(): int
<<interface>>
MedicalRecordProxy
MedicalRecord
- id: int
- patientid: int
- doctid: int
- date: Datetime
- comment: String
- precription: String
+ setId(int): void
+ getId(): int
+ setPatientid(int): void
+ getPatientid(): int
+ setDoctid(int) : void
+ getDoctid(int): int
+ setDate(date:Datetime): void
+ getDate(): Datetime
+setComment(String): void
+getComment(): String
+setPrescription(String): void
+getPrescription(): String
+createMedicalReport(): Arraylist<MedicalRecord>
Proxy
- authId: int
- authenticate(): Boolean
- viewRecord(): void
Appointment
- id: int
- doctid: int
- patientid: int
- date: Datetime
- time: time
+ setId(int): void
+ getId(): int
+ setDoctid(int): void
+ getDoctid(): int
+ setPatientid(int): void
+ getPatientid(): int
+ setDate(date: Datetime): void
+ getDate(): Datetime
+ setTime(time): void
+ getTime(): time
+ createAppointment(): void
+ generateAppointment(): Arraylist<Appointment>
+ load(): Boolean
QueryFacade
+ updateQuery(qid: int, query: String): Boolean
Query
- qid: int
- pid: int
- query: String
- title: String
+generateQueryForm() : Arraylist <Query>
+load():Boolean
+notifyDoctors(): Boolean
1
*
0..*

What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

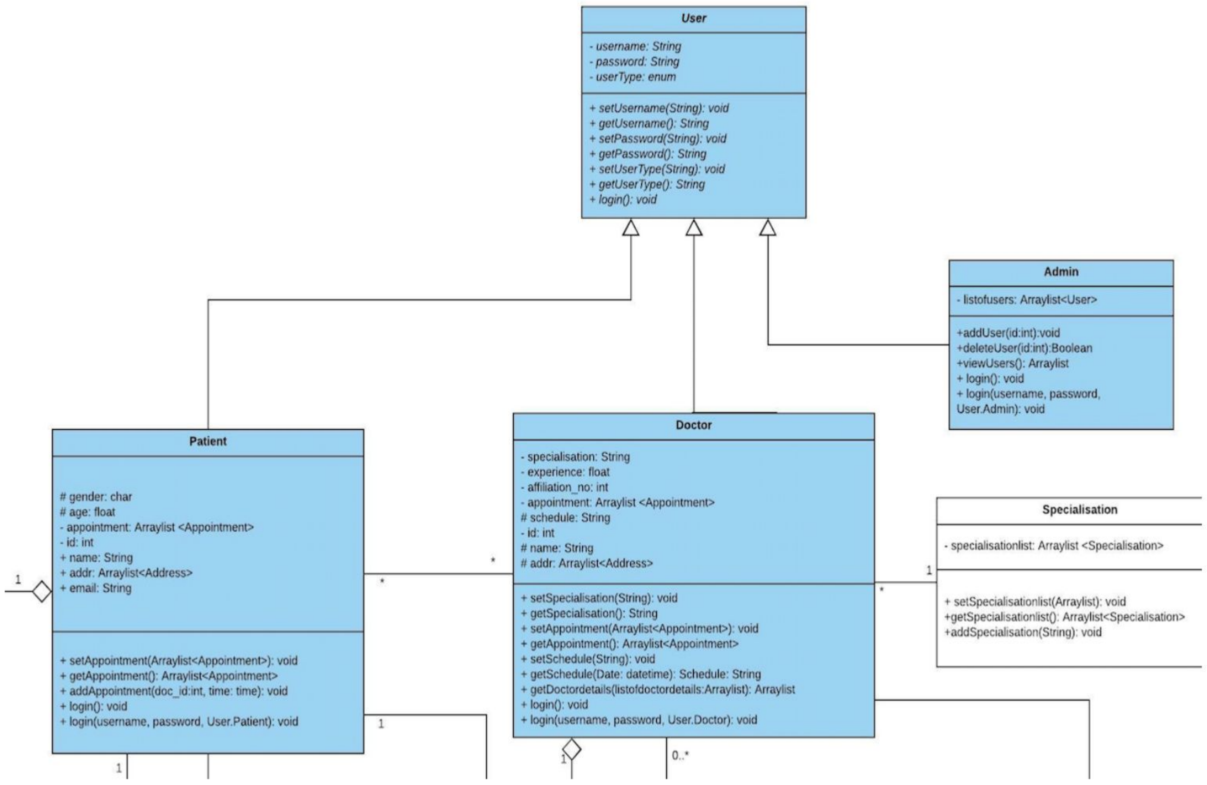
- The old diagram did not mention all the correct attributes, methods and return types of the classes. This was corrected in the final class diagram.
- The associations and inheritance remained the same in the final class diagram compared to the old one.
- All attributes of every class were implemented as Models in Spring MVC.
- Methods performed by every actor were enclosed in Controllers for every class
- There is no DBCall class in the final class diagram since it is not a good representation to have a class just for all DB queries. **We learned this in refactoring class exercise.**

- Refactoring - Changes in the Query and and the Appointment class to suit our implementation process.
- Changes as suggested in Project 2 Feedback - (Null is used as a return type instead of void).

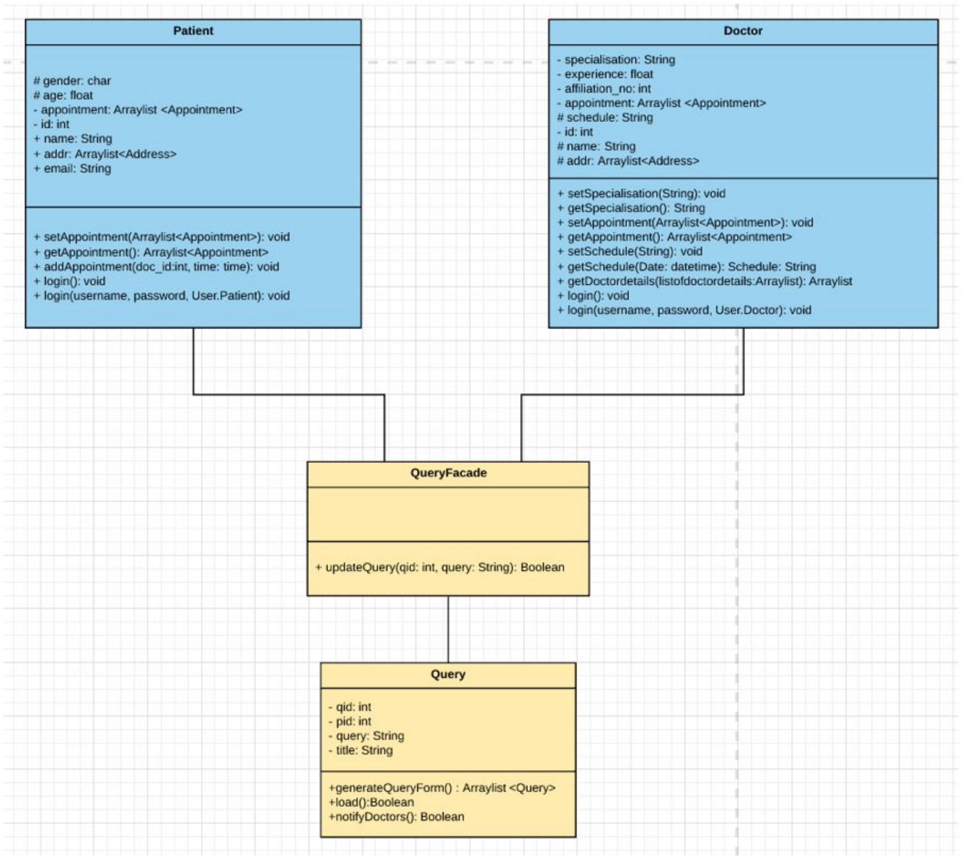
5) DESIGN PATTERNS IMPLEMENTED:

We incorporated the following design patterns into our implementation: Template, Facade and Proxy

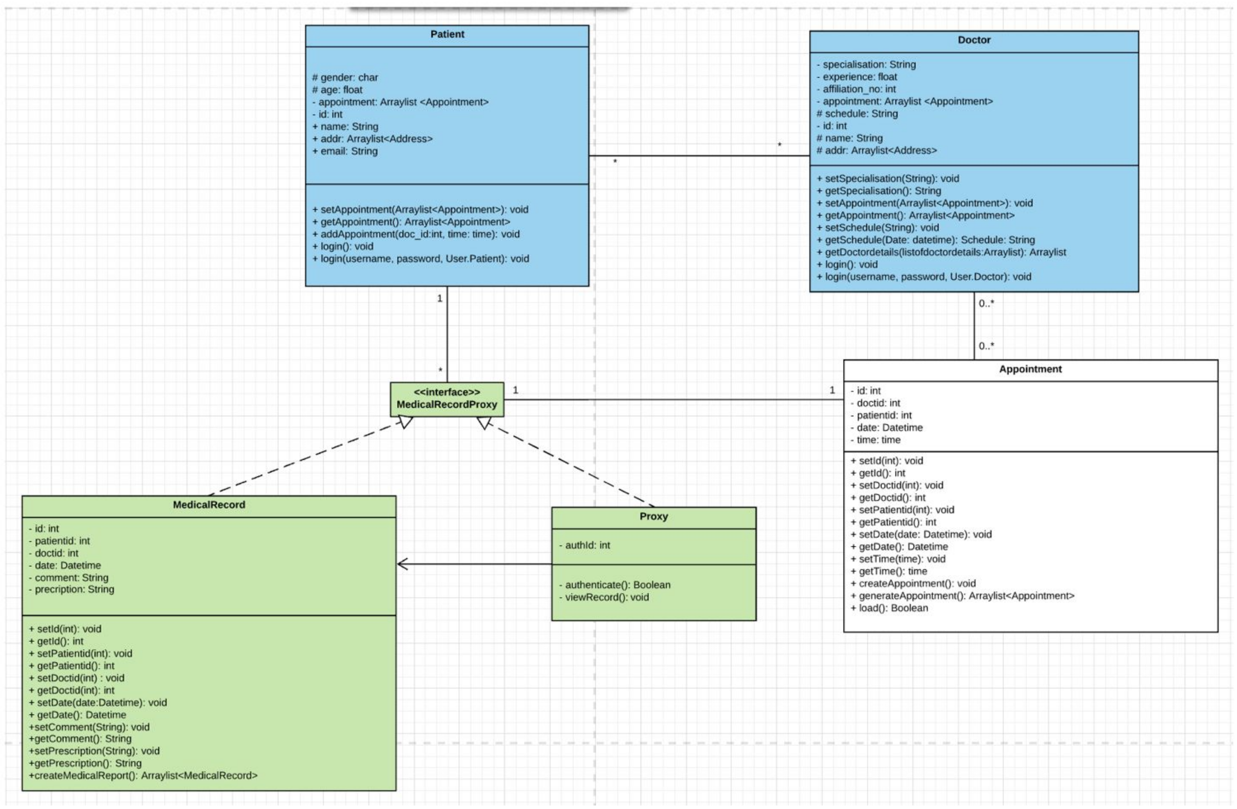
1) **Template Design Pattern** - We currently have three types of users which are being handled by the User class. The steps involved in logging-in is similar except the login header which contains the information about the type of user logging into the platform. **Participants:** Patient, Doctor, Admin, Pathologists


User
- username: String
- password: String
- userType: enum
+ setUsername(String): void
+ getUsername(): String
+ setPassword(String): void
+ getPassword(): String
+ setUserType(String): void
+ getUserType(): String
+ login(): void
Admin
- listofusers: Arraylist<User>
+addUser(id:int):void
+deleteUser(id:int):Boolean
+viewUsers(): Arraylist
+ login(): void
+ login(username, password, User.Admin): void
Patient
# gender: char
# age: float
- appointment: Arraylist <Appointment>
- id: int
+ name: String
+ addr: Arraylist<Address>
+ email: String
+ setAppointment(Arraylist<Appointment>): void
+ getAppointment(): Arraylist<Appointment>
+ addAppointment(doc_id:int, time: time): void
+ login(): void
+ login(username, password, User.Patient): void
Doctor
- specialisation: String
- experience: float
- affiliation_no: int
- appointment: Arraylist <Appointment>
# schedule: String
- id: int
# name: String
# addr: Arraylist<Address>
+ setSpecialisation(String): void
+ getSpecialisation(): String
+ setAppointment(Arraylist<Appointment>): void
+ getAppointment(): Arraylist<Appointment>
+ setSchedule(String): void
+ getSchedule(Date: datetime): Schedule: String
+ getDoctordetails(listofdoctordetails:Arraylist): Arraylist
+ login(): void
+ login(username, password, User.Doctor): void
Specialisation
- specialisationlist: Arraylist <Specialisation>
+ setSpecialisationlist(Arraylist): void
+getSpecialisationlist(): Arraylist<Specialisation>
+addSpecialisation(String): void
1
*
*
1
*
1
1
1
0..*

2) **Facade Design Pattern**- Query changes are common which should not affect the functionalities of the rest of the classes. For example, client code should have the same behaviour even if query system changes. Absence of Facade would cause changes in the query to be reflected on the Doctor and Patient where it should not. Hence, Facade is necessary to keep the functionalities of the Query independent from the other classes that use the Query class. **Participants:** Doctor and Patient


Patient
# gender: char
# age: float
- appointment: Arraylist <Appointment>
- id: int
+ name: String
+ addr: Arraylist<Address>
+ email: String
+ setAppointment(Arraylist<Appointment>): void
+ getAppointment(): Arraylist<Appointment>
+ addAppointment(doc_id:int, time: time): void
+ login(): void
+ login(username, password, User.Patient): void
Doctor
- specialisation: String
- experience: float
- affiliation_no: int
- appointment: Arraylist <Appointment>
# schedule: String
- id: int
# name: String
# addr: Arraylist<Address>
+ setSpecialisation(String): void
+ getSpecialisation(): String
+ setAppointment(Arraylist<Appointment>): void
+ getAppointment(): Arraylist<Appointment>
+ setSchedule(String): void
+ getSchedule(Date: datetime): Schedule: String
+ getDoctordetails(listofdoctordetails:Arraylist): Arraylist
+ login(): void
+ login(username, password, User.Doctor): void
QueryFacade
+ updateQuery(qid: int, query: String): Boolean
Query
- qid: int
- pid: int
- query: String
- title: String
+generateQueryForm() : Arraylist <Query>
+load():Boolean
+notifyDoctors(): Boolean

3) **Proxy Design Pattern** - To authenticate the doctors' and patients' access to the medical record. Only particular patients and doctors can access medical records of patients. To protect this access, we implemented "Protection Proxy". The presence of Proxy ensures that only authorized doctors can access the medical record. **Participants:** Doctor and Patient


Patient
# gender: char
# age: float
- appointment: Arraylist <Appointment>
- id: int
+ name: String
+ addr: Arraylist<Address>
+ email: String
+ setAppointment(Arraylist<Appointment>): void
+ getAppointment(): Arraylist<Appointment>
+ addAppointment(doc_id:int, time: time): void
+ login(): void
+ login(username, password, User.Patient): void
Doctor
- specialisation: String
- experience: float
- affiliation_no: int
- appointment: Arraylist <Appointment>
# schedule: String
- id: int
# name: String
# addr: Arraylist<Address>
+ setSpecialisation(String): void
+ getSpecialisation(): String
+ setAppointment(Arraylist<Appointment>): void
+ getAppointment(): Arraylist<Appointment>
+ setSchedule(String): void
+ getSchedule(Date: datetime): Schedule: String
+ getDoctordetails(listofdoctordetails:Arraylist): Arraylist
+ login(): void
+ login(username, password, User.Doctor): void
*
*
1
*
0..*
0..*
<<interface>>
MedicalRecordProxy
1
1
Appointment
- id: int
- doctid: int
- patientid: int
- date: Datetime
- time: time
+ setId(int): void
+ getId(): int
+ setDoctid(int): void
+ getDoctid(): int
+ setPatientid(int): void
+ getPatientid(): int
+ setDate(date: Datetime): void
+ getDate(): Datetime
+ setTime(time): void
+ getTime(): time
+ createAppointment(): void
+ generateAppointment(): Arraylist<Appointment>
+ load(): Boolean
MedicalRecord
- id: int
- patientid: int
- doctid: int
- date: Datetime
- comment: String
- precription: String
+ setId(int): void
+ getId(): int
+ setPatientid(int): void
+ getPatientid(): int
+ setDoctid(int) : void
+ getDoctid(int): int
+ setDate(date:Datetime): void
+ getDate(): Datetime
+setComment(String): void
+getComment(): String
+setPrescription(String): void
+getPrescription(): String
+createMedicalReport(): Arraylist<MedicalRecord>
Proxy
- authId: int
- authenticate(): Boolean
- viewRecord(): void

6) What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

1. The course took us through some crucial phases of software development
2. We learned the importance of the various design patterns that we learned in class when we went forward and started implementing them in our project.
3. We also got hands-on experience in implementing the core principles of object oriented design such as composition, aggregation etc. with OO Principles like polymorphism, inheritance, information hiding.
4. Overall, this subject has been a huge learning curve as it instilled in us to understand the importance of designing a software project before going for implementation.