COEN-6312-2184-UU

Model Driven Software Engineering

Professor: Wahab Hamou-Lhadj, Ph.D., ing.

Deliverable 3

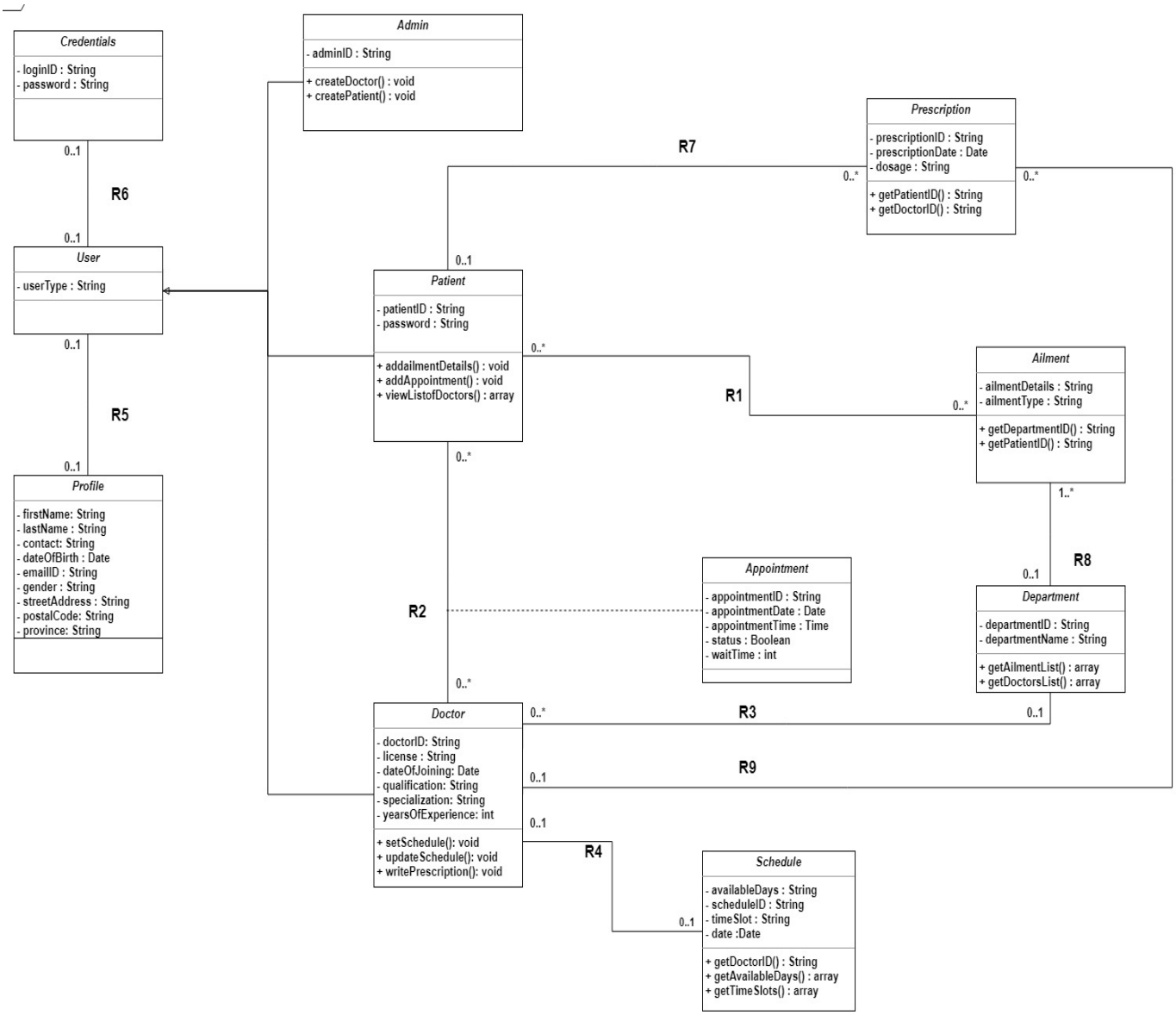
Submitted By:

NAME	ID	EMAIL
Ashish Sharma	40050452	ashish.sharma5293@gmail.com
Harmanpreet Singh	40059358	harmansandhu63@gmail.com
Navjot Kaur Bhamrah	40050459	navjotkaurbhamrah@gmail.com
Amandeep	40046716	amandpsingh03@gmail.com
Raghav Sharda	40053703	raghavsharma2926@outlook.com
Shivya Pant	40068007	shivyapant@gmail.com

Clinic Management System

1. Class Diagram

The purpose of the diagram is to show and explain clinic structure, staff, relationships with patients and patient treatment terminology. In the diagram, a Patient can fix up an appointment with a Doctor over the network record his/her ailments and choose a specialized Doctor depending on his/her ailments or health complaints.



1.1 The various classes used in this class diagram are explained below:

1.1.1 User Class

The User is an abstract class. Therefore, there cannot be any instance of User itself. However, there can be instances of its subclasses, namely Admin, Patient and Doctor. It contains the information for registering and validating the user. All users can login into the CMS application by using a unique login-id and password (operations to be supported based on type of user) and also able to change their password.

- **Admin -** This user instance will manage the system once it has been deployed. Also, this user can update the functionality of the CMS, change status of appointment from pending to confirmed, update schedule etc.

- **Patient -** This user instance will mostly use the system for its services. It contains its credentials as patientID and password attributes. To use the CMS services, it contains the methods like:

  - addailmentDetails() - It adds the health problems of the patient into the database for which they can book appointment later.

  - viewListofDoctors()- It shows the list of doctors for different departments and check their profile to book appointment later.

  - addAppointment()- It books the appointment for a particular schedule with a doctor based on patients ailment and doctor's specialization.

- **Doctor –** This user instance contains the doctor's data as attributes like doctorID, dateofJoining, qualification, specialization and yearsofExperience. It provides services to patient class, by methods like:

  - setSchedule()- It sets the schedule of a doctor based on his/her availability.

  - updateSchedule()- It updates the schedule of a doctor if comes any emergency or doctor is going for leave.

  - writePrescripton()- It adds and updates the prescription of a patient based on his/her ailments after the appointment. Basically, the attributes of prescription class are being added or updated by this method.

- **Appointment class –** It is an association class between Patient and Doctor class. These classes can access its attributes anytime, as whenever an appointment is scheduled, this class attributes are updated and accessed by its associated classes.

1.1.2 Schedule class

This class contains the schedule data of doctor and is bi-related to Doctor class. The various attributes of this class are scheduleID , timeslot, date and availableDays (all are of String type). These attributes guide towards the doctor's schedule. Its functionality is:

- getDoctorID()- A getter method to get the doctor id based on which doctor's schedule is written or updated.
- getAvailableDays()- It gets the available days of a particular doctor filtered by specialization.
- getTimeSlots()- It gets the doctor's timeslot for an appointment.

1.1.3 Department class

This class contains the department data and doctor's list based on each department. . It is related to Ailment class and Doctor class, as department is based on ailments patients face and department has doctors in it. Its attributes are departmentID and departmentName. Its functionality is:

- getAilmentList()- It gets the list of health problems based on each specialization(department) to check for doctor's in that department.
- getDoctorList()- As per the department, it gets the list of doctor's in that department.

1.1.4 Ailment class

It has the ailment data for patients and is linked to Patient class and Department class. It has attributes ailmentDetails and ailmentType (String type) which contains information of health problems of patient and checks for its related department. Its functionality contains:

- getDepartmentID()- As per the ailments described, it gets the related department by getting the department id.

- getPatientID()- It gets the id for patient to store ailments for each patient which can be further accessed by that patient id.

1.1.5 Prescription class

This class is related to patient class and Doctor class, as it contains the prescription data which has to be written by Doctor and accessed by both. The attributes prescriptionID, prescriptionDate and dosage (String and Date type) has prescription for each patient updated by Doctor after the appointment. Its methods are:

- getPatientID()- It gets the id of patient to link the prescription with it.

- getDoctorID()- Gets the doctor id so that prescription can be linked to it for record of which doctor gave the prescription.

1.1.6 Profile class

It has the profile data of each user by relation of one to one, as each user have one unique profile. It contains the Name, Address and Date of Birth attributes which are updated whenever a user registers with the CMS system (except Admin).

1.1.7 Credentials class

This class has credentials of each user which are needed every time when a user login the CMS system. It has loginID and password attributes to login (for each user). It's one on one relation with User class as each user should have unique id and password.

The relationships are defined as: Fig -1

Class 1	Relationship	Class 2	Relationship and Cardinality Details
User	Generalize	Admin	Admin can be a type of user.
User	Generalize	Patient	Patient can be a type of user.
User	Generalize	Doctor	Patient can be a type of user.
Patient	R2	Doctor	Patient can use multiple Doctor's for appointment and vice-versa.
Doctor	R4	Schedule	One to one as each Doctor has unique schedule.
Doctor	R3	Department	Multiple Doctors can have one single department, so one to many cardinalities.
Doctor	R9	Prescription	Single Doctor can give multiple prescriptions.
Department	R8	Ailment	Multiple ailments can be related to single department.
Patient	R1	Ailment	Multiple patients can have multiple ailments.
Patient	R7	Prescription	Single patient can be given multiple prescriptions.
Profile	R5	User	One to one related as each user has unique profile.
Credentials	R6	User	Each user has unique credentials so related one to one.

Fig.1

2. Design Pattern Used - Model View Controller

Design Architecture - The design pattern we wish to use in our application is MVC. The model view controller is a kind of pattern where the controller acts as a mediator between the model and view. The user will have direct interaction to the application through view which is the UI of the application. The model helps in fetching and displaying the values requested by the user or relevant to the user. While the Controller will provide the functionality to the application.

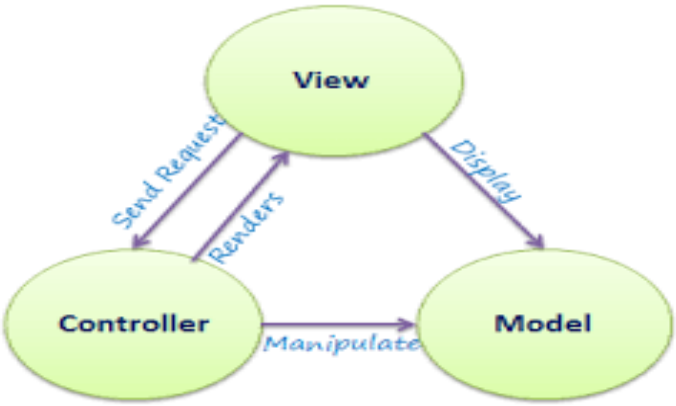


Fig 2.

3. *OCL Constraints*

3.1 There cannot be two different Doctors with the same medical license number.

```
context Doctor:

inv : Doctor.allInstances() -> forAll(d1 , d2 : Doctor | d1 <> d2 implies
d1.license <> d2.license)
```

3.2 For a particular timeslot on a particular date, there can only be one Patient.

```
context Patient:

inv : allInstances(self.R2.R4) -> forAll(S1  , S2: Schedule | S1.date =  S2.date)
implies  S1.timeSlot <>  S2.timeSlot
```

3.3 In case the Patient does not show up within the wait time (taken as a final variable of type int) of 15 minutes the appointment should be cancelled.

```
context Patient:

inv : self.Appointment.status="False" implies self.Appointment.waitTime > 15
```

3.4 The Patient should be taken care of by the department specific to his ailment.

```
context Patient:

inv: self.R2.R3 -> includes(self.R1.R8)
```

3.5 All the Users must have unique login ID credentials.

```
context User:

inv:  allInstances -> forAll (u1 ,u2 :User|u1<>u2  implies
u1.R6.loginID<>u2.R6.loginID)
and

self.R6.loginID = if(self.userType=”Patient”) then self.Patient.patientID endif
and

self.R6.loginID = if(self.userType=”Admin”)  then self.Admin.adminID endif
and

self.R6.loginID = if(self.userType=”Doctor”) then self.Doctor.doctorID endif
```

3.6 Each Patient will have a unique patientID.

```
context Patient:

inv: allInstances -> forAll(p1, p2: Patient | p1 <> p2 implies p1.patientID <>
p2.patientID)
```

3.7 Each Doctor will have a unique doctorID.

```
context Doctor:

inv: allInstances -> forAll(d1, d2: Doctor | d1<>d2 implies d1.doctorID <>
d2.doctorID)
```

3.8 Each Admin will have a unique adminID.

```
context Admin:

inv: allInstances -> forAll(a1, a2: Admin | a1<>a2 implies a1.AdminID <>
a2.AdminID)
```

3.9 A Patient can only have a prescription if the appointment is booked. Prescriptions cannot exist if the appointment for a Patient does not exist and there can be at most 1 prescription.

```
context Patient:

inv : self.R7 ->size()<= if (self.Appointment->notEmpty()) then 1  endif
```

3.10 A Patient can only book an appointment if the appointment doesn't exist already.

```
context Patient :: bookAppointment(a : Appointment )

pre  : self.Appointment -> excludes(a)
post : self.Appointment -> includes(a)
```

3.11 A Patient can only add an ailment if the ailment doesn't exist already.

```
context Patient :: addAilment( a:Ailment )

pre  : self.R1 -> excludes(a)
post : self.R2 -> includes(a)
```

3.12 A Patient or an Admin cannot have an empty string as a login.

```
context Patient
inv: allInstances -> Patient.self.oclIsKindOf(User).R6 -> forAll(p | not(p.loginID
= " "))

context Admin
inv : allInstances -> Admin.self.oclIsKindOf(User).R6 -> forAll(a | not(a.loginID
= " "))
```

3.13 The Schedule ID will always be unique.

```
context Schedule:

inv : allInstances() -> forAll(S1 , S2 : Schedule | S1<>S2  implies S1.scheduleID
<> S2.scheduleID )
```

3.14 A Doctor cannot be a patient at the same time.

Already enforced.

3.15 Only ten Patient and two Admin can register at a time.

```
context Admin:

inv : self.oclIsKindOf(User) implies self.R5.register -> Size()<=2

context Patient:

inv : self.oclIsKindOf(User) implies self.R5.register  -> Size()<=10
```

References

[1]. http://cs.ulb.ac.be/public/_media/teaching/infoh302/oclnotes.pdf
[2].https://www.academia.edu/37603896/MVC_MODELVIEWCONTROLLER_DALAM_PEMBANGUNAN_APLIKASI_WEB