České vysoké učení technické v Praze
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

"My Doctor" - Information System

Design documentation

2nd iteration

Authors: Caraganciu Dan
Băncilă Iana
Leahu Luminița

Architecture

This chapter describes the architecture of the MyDoc system.

The web application for booking appointments to the doctor will be realized as a as a three-layered architecture. Is used Django framework, which follows the 'Model–View–Controller' architectural pattern. The layers implemented in Django are the following:
• The Model layer
• The View layer
• The Template layer

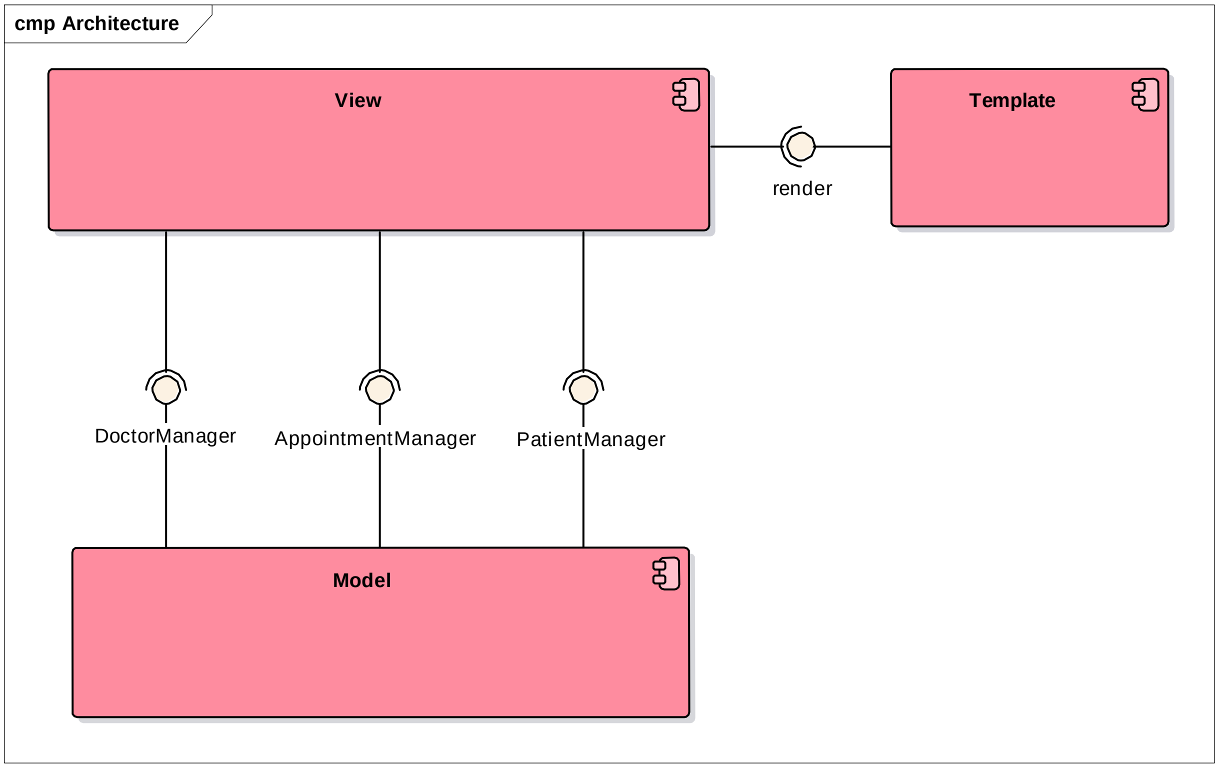

Figure: 1

Model

A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table.

Template

The template layer contains HTML content and presentation logic. It gets data from the view and outputs a web page.
i.e. the template layer provides the syntax for rendering the information to be presented to the user.

In the Django settings, we were able to specify a list of directories to check for templates with TEMPLATE_DIRS. If a template doesn't exist in the first directory, it checks the second, and so on.

View

The View layer (sometimes called 'Controller') encapsulates the logic responsible for processing a user's request and for returning the response.
It can see all the other layers. It defines URLs, maps them to functions which receive data from the web framework and use the other layers to finally answer back to the web framework. It should be kept small, because the code in it is not very reusable.
The View describes the data that gets presented to the user. It's not necessarily how the data looks, but which data is presented.

Package "Design-Django"

As an association with the three layered architecture, the model layer in our application contains the business layer and the data layer.

Also a part of the functionality of the business layer is provided by View layer. The template layer is the topmost level of the application. It can be described as the presentation layer in the three layered architecture.

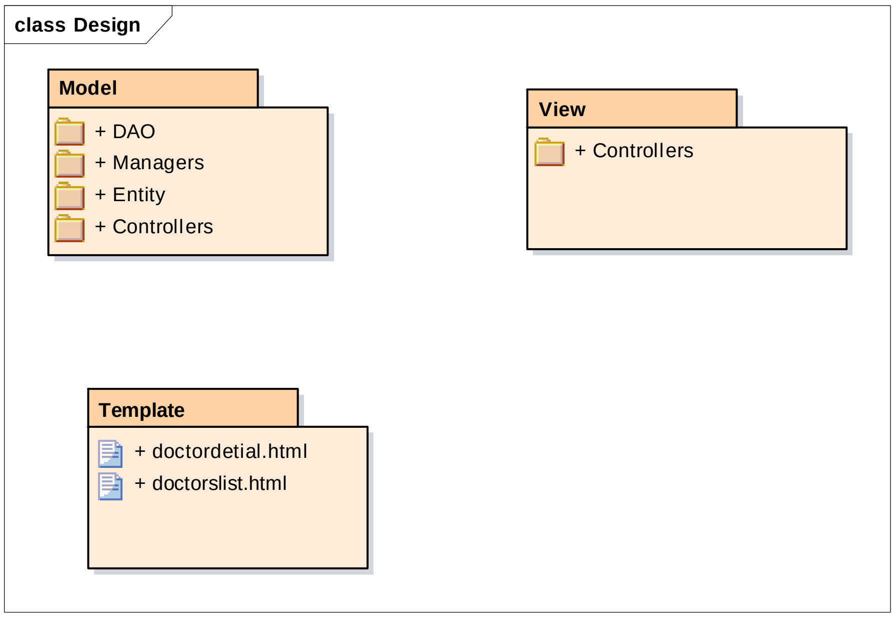

Figure: 1

Package "Model"

Package contains classes of the Model layer, which holds the model of software classes (data holders, business logic and user interface).

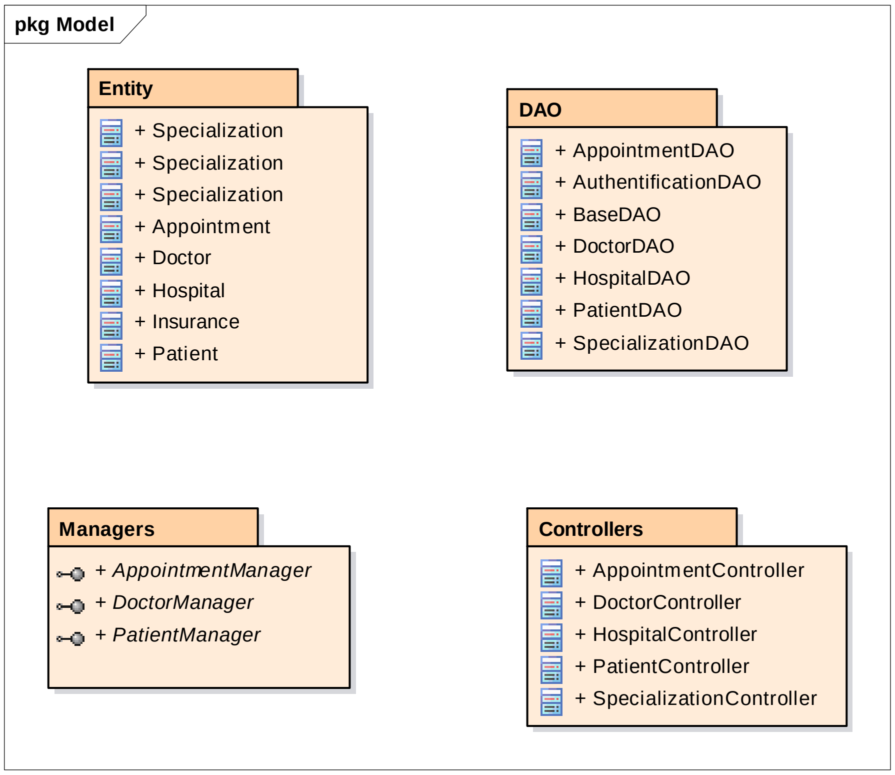

Figure: 2

Package "Managers"

Contains methods which are used for interaction between the View layer and the Model layer.

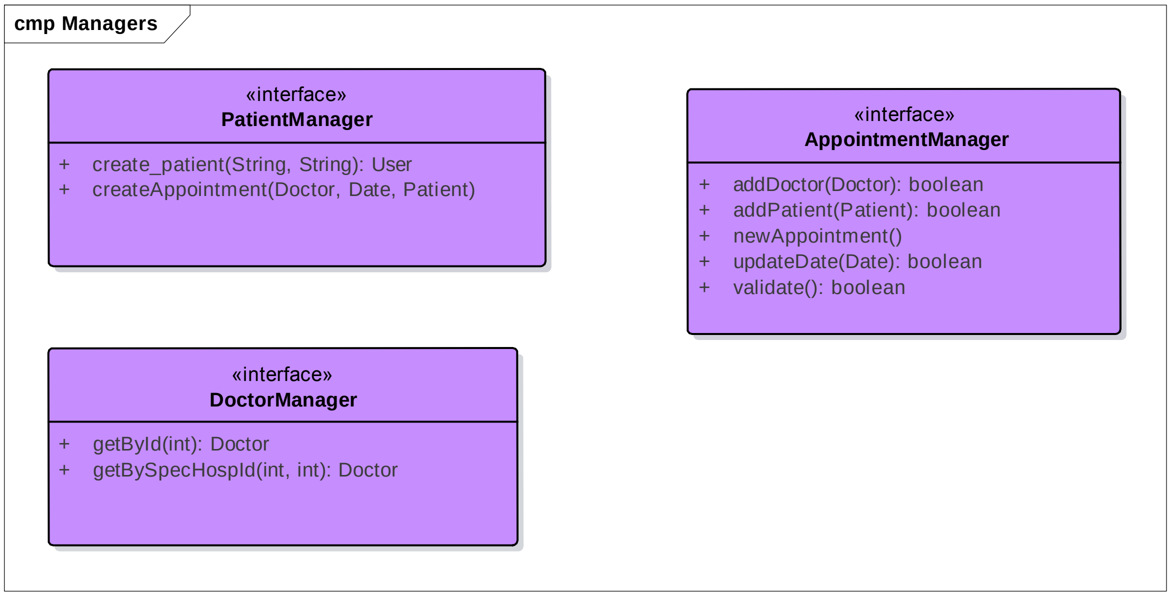
cmp Managers
«interface»
PatientManager
+ create_patient(String, String): User
+ createAppointment(Doctor, Date, Patient)
«interface»
AppointmentManager
+ addDoctor(Doctor): boolean
+ addPatient(Patient): boolean
+ newAppointment()
+ updateDate(Date): boolean
+ validate(): boolean
«interface»
DoctorManager
+ getById(int): Doctor
+ getBySpecHospId(int, int): Doctor

Figure: 3

Package "Entity"

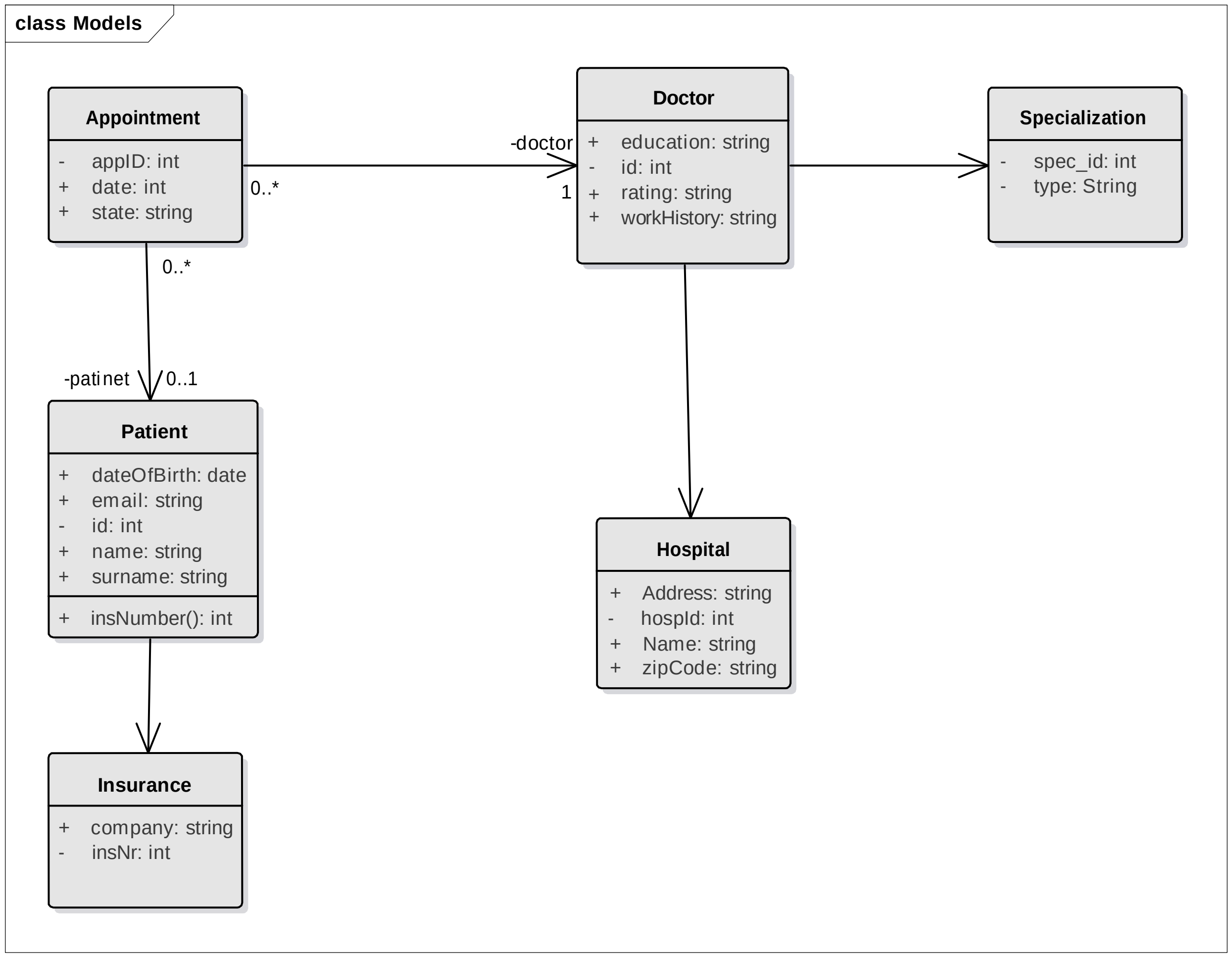

Figure: 4

Class "Appointment"

The appointment entity stores the data about each appointment made by different users/patients.

An appointment is defined by a unique ID, date and state.

Attributes

Attribute	Data type	Notes
appID	int	
date	int	
state	string	

Class "Doctor"

The Doctor entity stores the data about each doctor.

Each doctor can have 0 or multiple appointments.

Each doctor is defined by a unique id, rating, speciality, work History and education.

Attributes

Attribute	Data type	Notes
education	string	
id	int	
rating	string	
specilaty	string	
workHistory	string	

Class "Hospital"

Hospital entity stores the data about each hospital such that every one of them have an address, a unique ID, a name and a zip Code.

Attributes

Attribute	Data type	Notes
Address	string	
hospId	int	
Name	string	
zipCode	string	

Class "Insurance"

Insurance entity stores the data about the insurance used by a user/patient.

Insurance is defined by company name and insurance number.

Attributes

Attribute	Data type	Notes
company	string	
insNr	int	

Class "Patient"

Patient entity stores the data about all the patients registered in the system.

Each patient is defined by date of birth, email, unique id, name, surname and insurance number in case he has an insurance.

Every patient can book multiple appointments.

Attributes

Attribute	Data type	Notes
dateOfBirth	date	
email	string	
id	int	
name	string	
surname	string	

Methods

Method	Return type	Notes
insNumber	int	Parameters:

Class "Specialization"

Specialization entity stores the data about the type of specialization.

Attributes

Attribute	Data type	Notes
type	string	
Spec_id	int	

Package "DAO"

The DAO package provides an abstract interface to the database and some specific data operations without exposing details of the database.

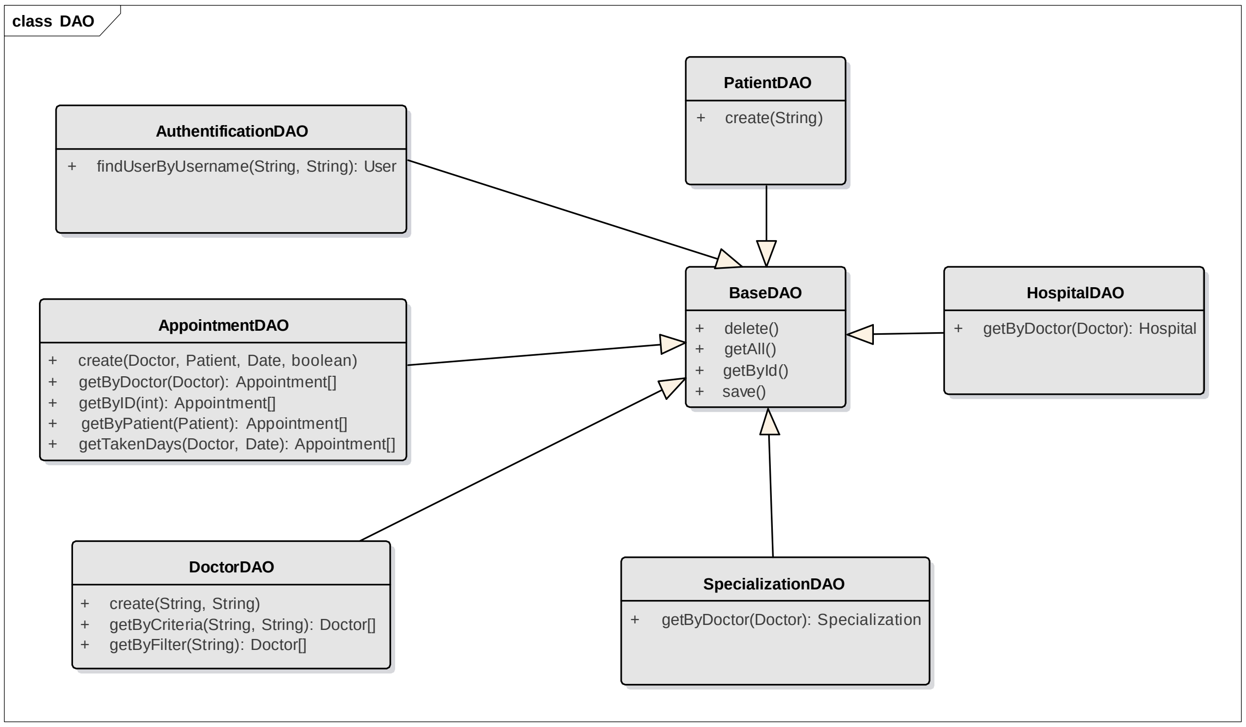
class DAO
PatientDAO
+ create(String)
AuthentificationDAO
+ findUserByUsername(String, String): User
BaseDAO
+ delete()
+ getAll()
+ getById()
+ save()
HospitalDAO
+ getByDoctor(Doctor): Hospital
AppointmentDAO
+ create(Doctor, Patient, Date, boolean)
+ getByDoctor(Doctor): Appointment[]
+ getByID(int): Appointment[]
+ getByPatient(Patient): Appointment[]
+ getTakenDays(Doctor, Date): Appointment[]
DoctorDAO
+ create(String, String)
+ getByCriteria(String, String): Doctor[]
+ getByFilter(String): Doctor[]
SpecializationDAO
+ getByDoctor(Doctor): Specialization

Figure: 5

Class "AuthentificationDAO"

Used for find in the data base information about the user which tries to log in the application.

Methods

Method	Return type	Notes
findUserByUsername	User	Parameters:

Method	Return type	Notes
		• **username:** String • **password:** String

Package "Controllers"

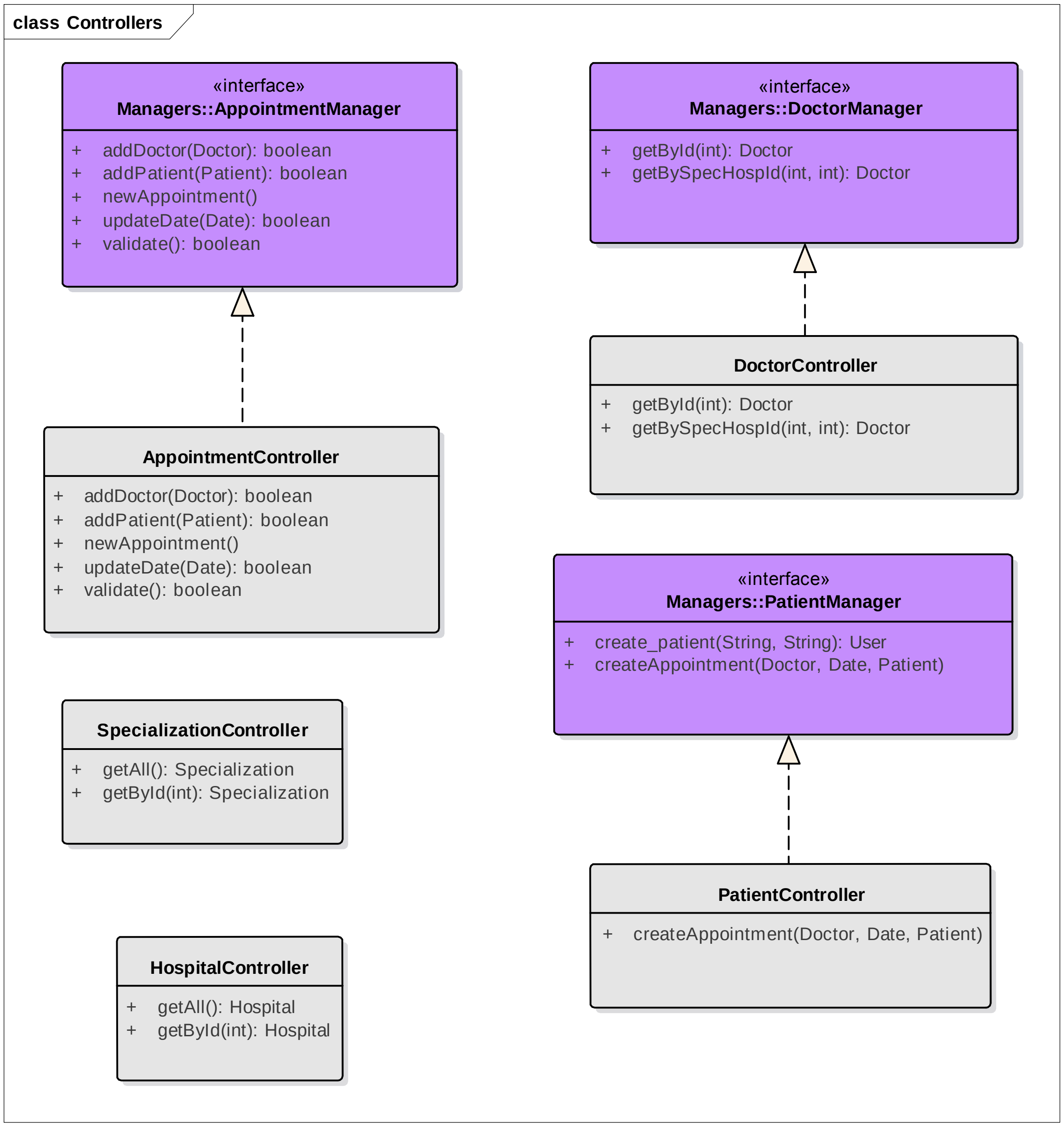
class Controllers
«interface» Managers::AppointmentManager
+ addDoctor(Doctor): boolean
+ addPatient(Patient): boolean
+ newAppointment()
+ updateDate(Date): boolean
+ validate(): boolean
«interface» Managers::DoctorManager
+ getById(int): Doctor
+ getBySpecHospId(int, int): Doctor
DoctorController
+ getById(int): Doctor
+ getBySpecHospId(int, int): Doctor
AppointmentController
+ addDoctor(Doctor): boolean
+ addPatient(Patient): boolean
+ newAppointment()
+ updateDate(Date): boolean
+ validate(): boolean
«interface» Managers::PatientManager
+ create_patient(String, String): User
+ createAppointment(Doctor, Date, Patient)
SpecializationController
+ getAll(): Specialization
+ getById(int): Specialization
PatientController
+ createAppointment(Doctor, Date, Patient)
HospitalController
+ getAll(): Hospital
+ getById(int): Hospital

Figure: 6

Package "Template"

More templates will be added during implementation, for list of appointments, appointment detail, etc.

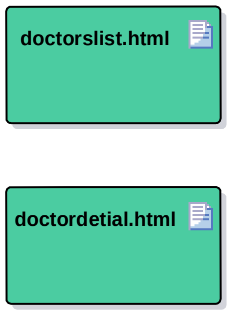

Figure: 7

Class "doctordetial.html"

The HTML template for the detail of a doctor.

Class "doctorslist.html"

The HTML template for the list of doctors.

Package "View"

Figure: 8

Package "Controllers"

Classes that reflect application state: from administrator's perspective.

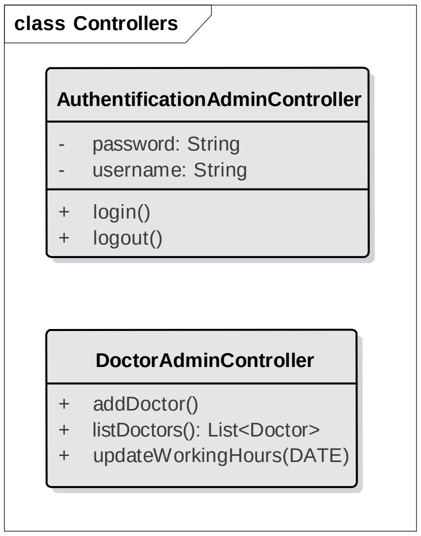

Figure: 9

RealizationModel

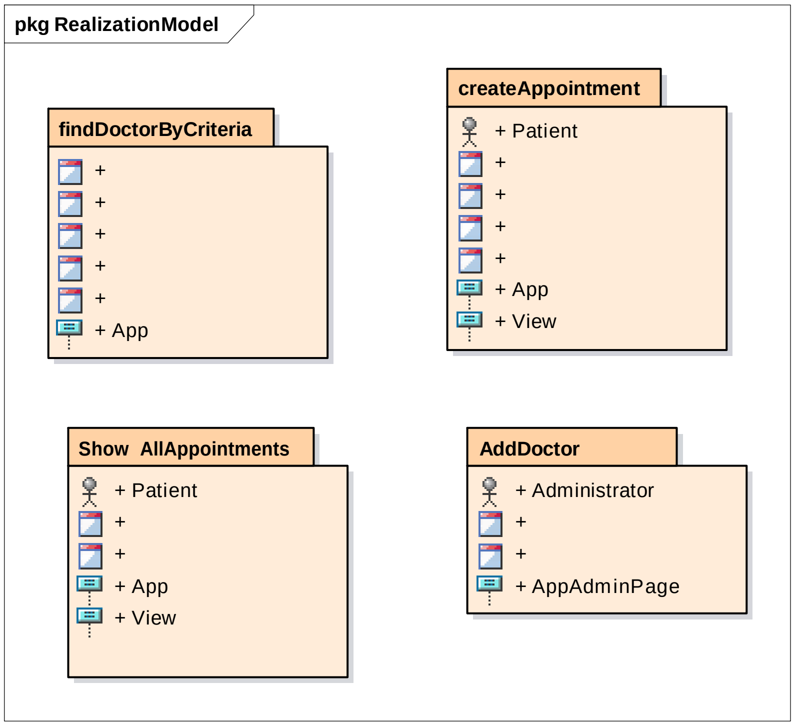

Figure: 1

AddDoctor

Our application has an admin site that lets you add, change and delete doctors.

The diagram shows the logic of the process.

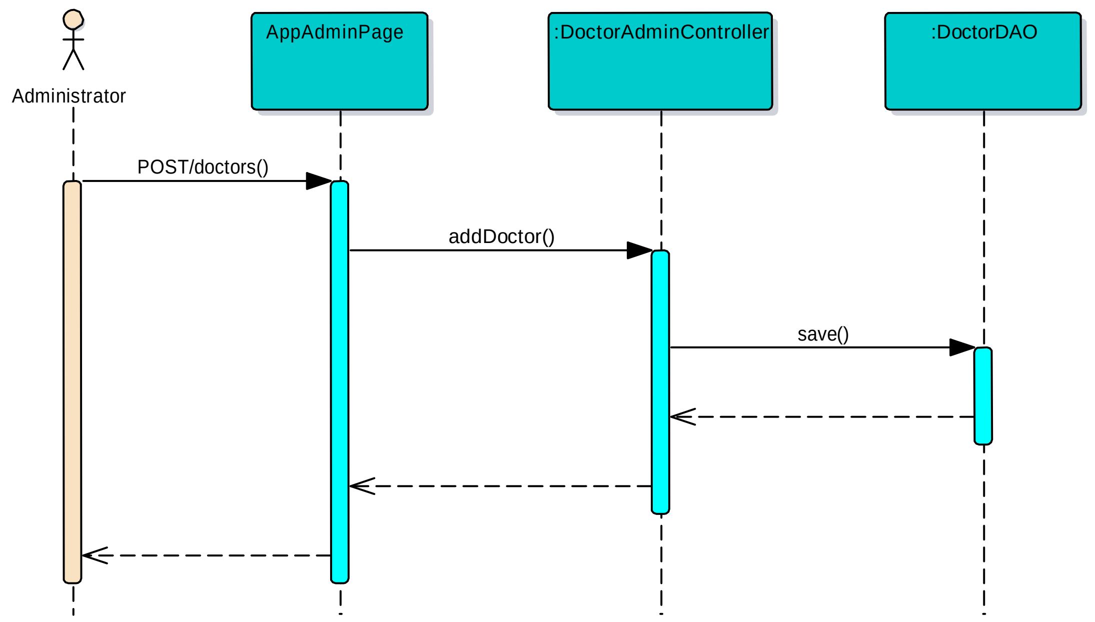

Figure: 2

ShowAllAppointments

The diagram shows the process of showing all appointments of a specific patient/doctor.

It requires PatientController to call the getAllAppointments method and then from the DataBase is retrieved only the list of appointments for the user which initiate the request. The list of appointments is searched by patient_id or doctor_id.

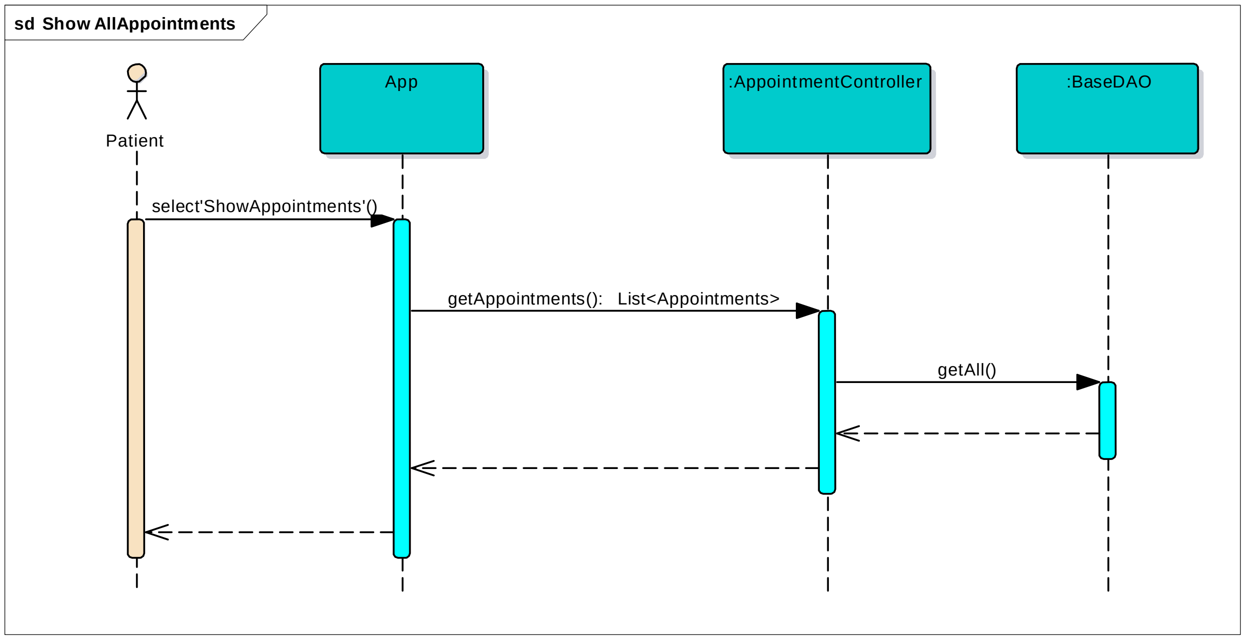

Figure: 3

createAppointment

The process starts when patient chooses a doctor from the List of Doctors generated by system based on his search preferences.

A newAppointment() is initiated, method from AppointmentController.

Second step is to choose the desired Date and Time, the time and date is stored in the new created appointment. Automatically the system search in the DataBase current user and stores information about it in the new created appointment. Information about chosen Doctor is also added automatically.

The new created record for appointment is saved to database.

After user submit the appointment it is to AppointmentController.

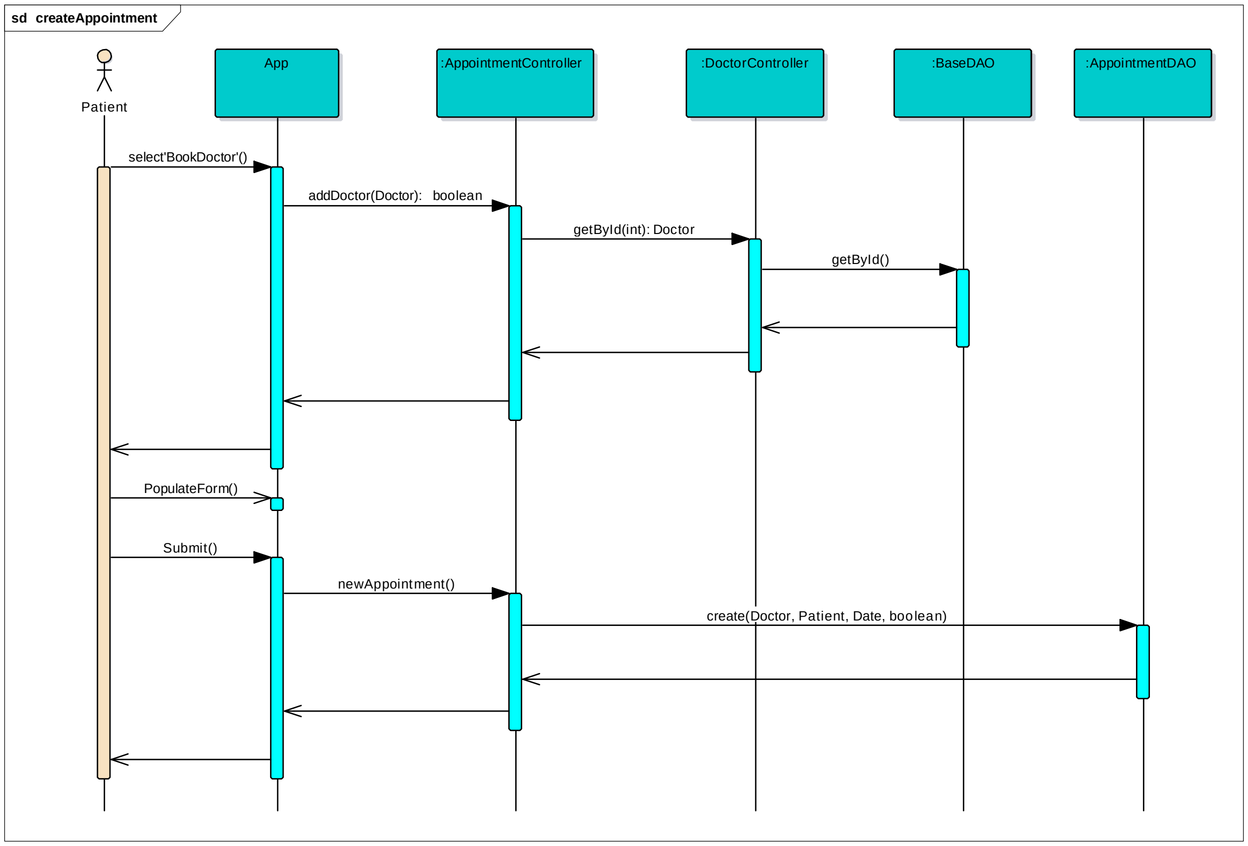

Figure: 4

findDoctorByCriteria

The process starts when Patient goes to SearchPage. The form for filled in is generated.

The Patient enters doctor's specialty and ZipCode. When he pressed button 'Find' the findbyCriteria method is called and from the DataBase is retrieved only records of doctors which have the same specialty and zipcode as specified by user.

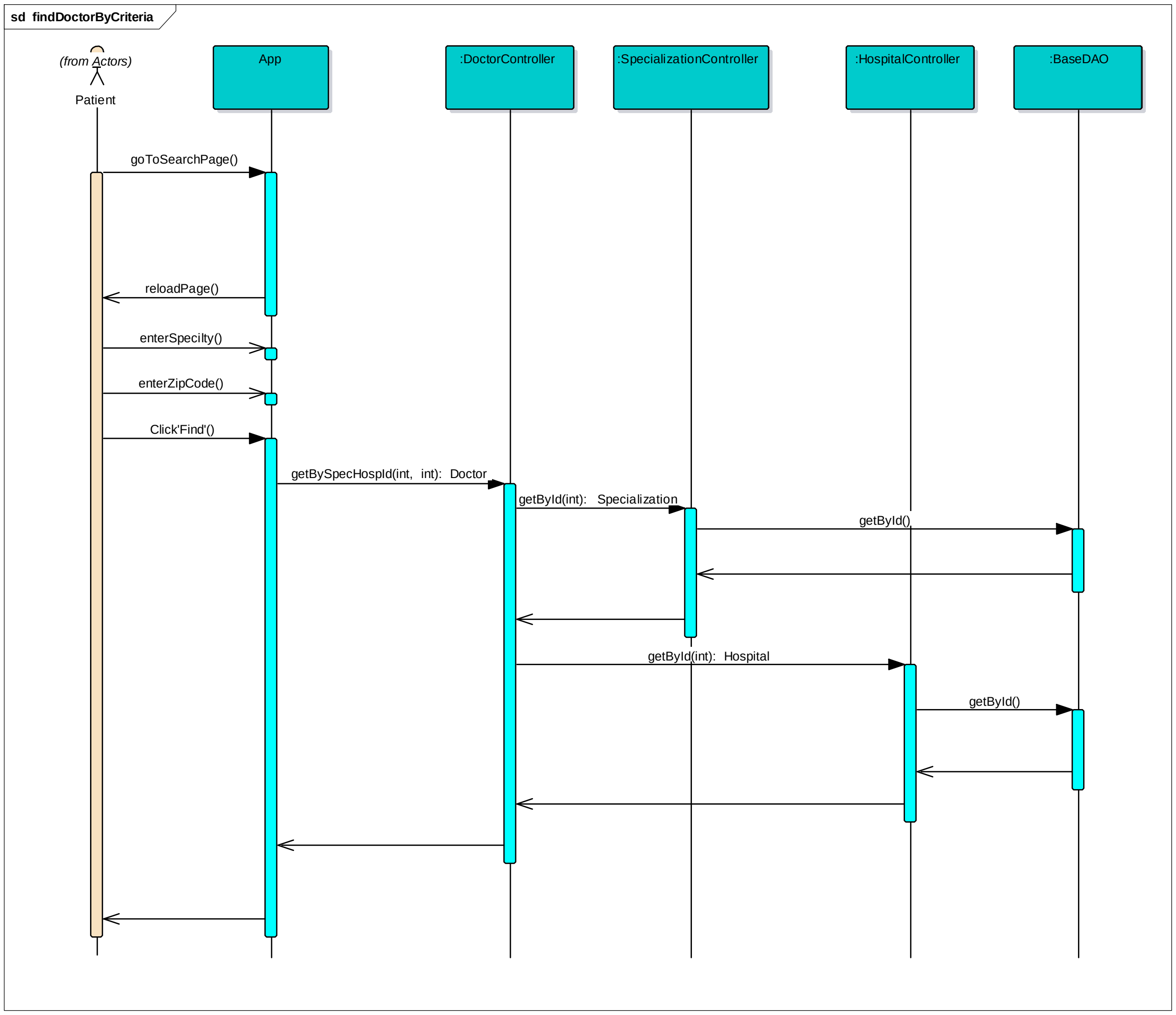

Figure: 5

Package "Database"

A database is a collection of data. This is a place in which we will store information about users, doctors, appointments. We will be using a SQLite3 database to store our data. This is the default Django database adapter.

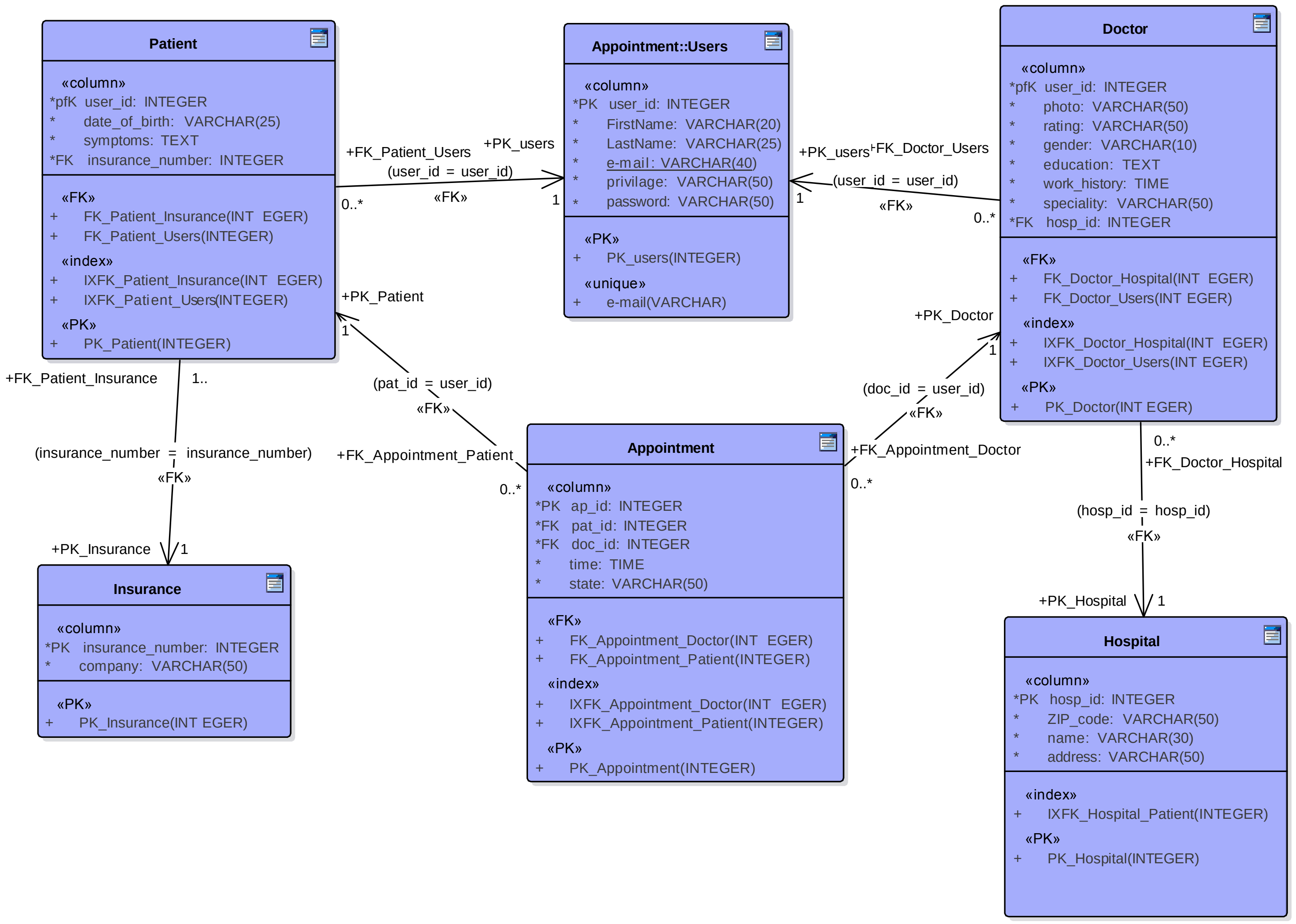

Figure: 1

This package describes the structure of the MySQL database used MyDoctor system.

Table "Appointment"

Table Appointment contains the relation of an appointment to a patient and a doctor.

Attributes

Column	Data type	Len.	Prec, Scale	Notes
ap_id	INTEGER		,	Primary key of the table Appointment.
pat_id	INTEGER		,	Foreign key to the Patient table.
doc_id	INTEGER		,	Foreign key to the Doctor table.
time	TIME	0	,	Calendar with working time of the doctor.
state	VARCHAR	50	,	Status of availability of the doctor.

Constraints

Constraint	Notes
FK_Appointment_Doctor	Parameter: -doc_id: INTEGER Parameters: • **doc_id:** INTEGER
FK_Appointment_Patient	Parameter: -pat_id: INTEGER Parameters: • **pat_id:** INTEGER
IXFK_Appointment_Doctor	Parameters: • **doc_id:** INTEGER
IXFK_Appointment_Patient	Parameters: • **pat_id:** INTEGER
PK_Appointment	Parameter: -ap_id: INTEGER Parameters: • **ap_id:** INTEGER

Figure: DocDB

Table "Users"

Users represents information about a user which may be an doctor or a patient.
A user must be always at least a patient or a doctor, but may be both at the same time.

Attributes

Column	Data type	Len.	Prec, Scale	Notes

Column	Data type	Len.	Prec, Scale	Notes
user_id	INTEGER		,	Primary key of the Users table.
FirstName	VARCHAR	20	,	First name of a patient or a doctor.
LastName	VARCHAR	25	,	Last name of the users - family name or surname.
e-mail	VARCHAR	40	,	E-mail of users, which is also a login to the system.
privilage	VARCHAR	50	,	When a new user registers to the system he identifies himself like doctor or patient.
password	VARCHAR	50	,	Password of a user.

Constraints

Constraint	Notes
PK_users	Parameter: -user_id: INTEGER Parameters: • **user_id:** INTEGER
e-mail	Parameter: -e-mail: VARCHAR Parameters: • **e-mail:** VARCHAR

Table "Doctor"

Table Doctor represents a description of a doctor. It contains all necessary information for the patient about the doctor. Doctor works in a certain hospital and can several appointments in one day.

Attributes

Column	Data type	Len.	Prec, Scale	Notes
user_id	INTEGER		,	Primary key of the Doctor table and also a foreign key to the Appointment table.
photo	VARCHAR	50	,	Photo of the doctor.
rating	VARCHAR	50	,	Rating tells the patient how well doctor in his profession, based on an objective evaluation of previous patients.
gender	VARCHAR	10	,	Gender of a doctor.
education	TEXT		,	Knowledge, skills medical degree of the doctor.
work_history	TIME	0	,	Work history is a detailed report of all the jobs a

Column	Data type	Len.	Prec, Scale	Notes
				doctor have held, including the company name and job title.
hosp_id	INTEGER		,	It is the foreign key for the table Hospital.

Constraints

Constraint	Notes
FK_Doctor_Hospital	Parameter: -hosp_id: INTEGER Parameters: • **hosp_id:** INTEGER
FK_Doctor_Users	Parameter: -user_id: INTEGER Parameters: • **user_id:** INTEGER
IXFK_Doctor_Hospital	Parameters: • **hosp_id:** INTEGER
IXFK_Doctor_Users	Parameters: • **user_id:** INTEGER
PK_Doctor	Parameter: -user_id: INTEGER Parameters: • **user_id:** INTEGER
FK_Doctor_Specialization	Parameter: -spec_id: INTEGER Parameters: • spec_id: INTEGER

Table "Hospital"

Hospital is a health care institution providing patient treatment with specialized staff.

Attributes

Column	Data type	Len.	Prec, Scale	Notes
hosp_id	INTEGER		,	Reference to hospital
ZIP_code	VARCHAR	50	,	Each hospital has ZIP code.
name	VARCHAR	30	,	Name of the hospital.
address	VARCHAR	50	,	Adress of the hospital.

Constraints

Constraint	Notes
IXFK_Hospital_Patient	Parameters: • **hosp_id:** INTEGER
PK_Hospital	Parameter: - hosp_id: INTEGER Parameters: • **hosp_id:** INTEGER

Table "Specialization"

Attributes

Column	Data type	Len.	Prec, Scale	Notes
spec_id	INTEGER		,	Reference to specialization
type	VARCHAR	50	,	The type of specialization

Constraints

Constraint	Notes
PK_Spec	Parameter: - spec_id: INTEGER Parameters: • **spec_id:** INTEGER

Table "Insurance"

Insurance is a type of insurance coverage that covers the cost of an insured individual's medical expenses.
The insured patient is the owner of the health insurance policy, the person with the health insurance coverage.

Attributes

Column	Data type	Len.	Prec, Scale	Notes
insurance_number	INTEGER		,	Insurance_Number is a numerical identification code and also primary key of the Insurance table.
company	VARCHAR	50	,	Company that offers insurance policies to the patient.

Constraints

Constraint	Notes
PK_Insurance	Parameter: -insurance_number: INTEGER Parameters: • **insurance_number:** INTEGER

Table "Patient"

Patient is the person who can be registered in the system and make an appointment to the desired doctor. Patient can have an insurance and make several appointments at the same time.

Attributes

Column	Data type	Len.	Prec, Scale	Notes
user_id	INTEGER		,	Primary key of the Patient table and also a foreign key to the Appointment table.
date_of_birth	VARCHAR	25	,	Date of birth of the patient.
symptoms	TEXT		,	Symptoms are departure froms normal function or feeling which is noticed by a patient, reflecting the presence of an unusual state, or of a disease.
insurance_number	INTEGER		,	Insurance number is the foreign key for the table Insurance. Patient can provide the insurance number when he make an appointment to the doctor.

Constraints

Constraint	Notes
FK_Patient_Insurance	Parameter: -insurance_number: INTEGER

Constraint	Notes
	Parameters: • **insurance_number:** INTEGER
FK_Patient_Users	Parameter: -user_id: INTEGER Parameters: • **user_id:** INTEGER
IXFK_Patient_Insurance	Parameters: • **insurance_number:** INTEGER
IXFK_Patient_Users	Parameters: • **user_id:** INTEGER
PK_Patient	Parameter: - user_id: INTEGER Parameters: • **user_id:** INTEGER