myDocApp Documentation

Release 1.0

luminita

January 16, 2016

CONTENTS

Contents:

CHAPTER

ONE

DAO PACKAGE

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

class `doctors.dao.`**`AppointmentDAO`**
abstract interface to database object: Specialization

`create`(*patient*, *doctor*, *date*, *valid*)
method which creates new appointments specifing the Patient which created, Doctor which was booked, date of creation and validation status

`getByDoctor`(*doctor*)
method which returns from database the Appointments made by the specified Doctor object

`getByID`(*ID*)
method which returns from database the Appointments made by the specified appointment's ID (primary key)

`getByPatient`(*patient*)
method which returns from database the Appointments made by the specified Patient object

`getTakenDays`(*doctor*, *date*)
method which returns from database the days of the specified Doctor timetable which are allready taken for appointments

class `doctors.dao.`**`AuthentificationDAO`**
class working with logging in the myDoc System

`findUserByUsername`(*user*, *password*)
method for cheking if the credentials entered by user are in the database (i.e. if such a user exists).

class `doctors.dao.`**`BaseDAO`**
BaseDAO contains all the abstracts methods that should be implemented in the Controller.

`delete`()
abstract method for deleting objects from database

`getAll`()
abstract method which returns all specified types of objects from database

`getById`(*ID*)
abstract method which returns the objects from database which has the specified ID (primary key)

`save`()
abstractmethod methods which save created object to database

class `doctors.dao.`**`DoctorDAO`**
abstract interface to database object: Doctor

`create`(*name*, *spec*)
creates new objects of type Patient by and specialization

`getByCriteria`(*spec*, *zipcode*)
method which returns from database Doctor objects with specified specialization and zip_code

`getByFilter`(*gen*)
method which returns from database Doctor objects with specified gender

class `doctors.dao.`**`HospitalDAO`**
abstract interface to database object: Specialization

`getByDoctor`(*doctor*)
method which returns from database the Hospital of specified Doctor object

class `doctors.dao.`**`PatientDAO`**
abstract interface to database object: Patient

`create`(*name*)
creates new objects of type Patient by name

class `doctors.dao.`**`SpecializationDAO`**
abstract interface to database object: Specialization

`getByDoctor`(*doctor*)
method which returns from database type of Specialization of specified Doctor object

CHAPTER

TWO

CONTROLLERS PACKAGE

The controllers specify functionalities which are provided from the Model layer for the View layer.

class `doctors.controllers.`**`AppointmentController`**

AppointmentController class deals with implementing the methods specified in AppointmentDAO class

class `doctors.controllers.`**`DoctorController`**

DoctorController class deals with implementing the methods specified in DoctorDAO class

`getById`(*doc_id*)

method that retrieves Doctor objects from database by specified ID

`getBySpecHospId`(*spec_id*, *hosp_id*)

method that retrieves Doctor objects from database by specified specialization ID and Hospital ID

class `doctors.controllers.`**`HospitalController`**

HospitalController class deals with implementing the methods specified in HospitalDAO class

`getAll`()

method that retrieves all Hospital objects from database

`getById`(*ID*)

method that retrieves Hospital objects from database by specified ID

class `doctors.controllers.`**`PatientController`**

PatientController class deals with implementing the methods specified in PatientDAO class

class `doctors.controllers.`**`SpecializationController`**

SpecializationController class deals with implementing the methods specified in SpecializationDAO class

`getAll`()

method that retrieves all specializations types from database

`getById`(*ID*)

method that retrieves specializations types from database by specified ID

CHAPTER THREE

MANAGERS PACKAGE

class `doctors.managers.`**`DoctorManager`**

DoctorManager class

`create_patient``(`*name*, *password=None*`)`

method for careting new users with the role Doctor in the System

class `doctors.managers.`**`PatientManager`**

PatientManager class

`create_patient``(`*name*, *password=None*`)`

method for careting new users with the role Patient in the System

CHAPTER

FOUR

FORMS PACKAGE

“Forms describe a form and determines how it works and appears. A form class’s fields map to HTML form <input> elements (/myDoc/doctors/templates.”

class `doctors.forms.`**`BookDoc`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*)

SearchDoc Class represents a form which allows user to fill it with required details about reservation. The form uses POST method and save the introduced data as a new Appointment Object in database

class `doctors.forms.`**`SearchDoc`**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*)

SearchDoc Class represents a form which allows user to fill it with desired doctor’s specialization and zip_code. The form uses GET method and returns a list of Doctor objects from database results that fulfill the required search criteria

CHAPTER

FIVE

MODELS PACKAGE

"A model is the single, definitive source of information about the data. It contains the essential fields and behaviors of the data that is storing in the myDoc System. Generally, each model maps to a single database table. The unique ID (primary key) are craeted aoutomatically."

class `doctors.models.`**`Appointment`**`(`**args*, ***kwargs*`)`
    Stores the data about each Appointment. Each Appointment has the filds: patient which created the appointment, doctor at whom, date of creation and validation status

class `doctors.models.`**`Doctor`**`(`**args*, ***kwargs*`)`
    Stores the data about each Doctor. Each doctor is defined by name, rating, speciality, zip_code, photo, gender and education. The field password is automatically added from the DoctorManager class

class `doctors.models.`**`Hospital`**`(`**args*, ***kwargs*`)`
    Stores data about each Hospital such that every one of them has an addres, zip_code and name

class `doctors.models.`**`Patient`**`(`**args*, ***kwargs*`)`
    Stores the data about each Patient. Each doctor is defined by name, surname, insurance number, date of birth, email. The field password is automatically added from the DoctorManager class

class `doctors.models.`**`Specialization`**`(`**args*, ***kwargs*`)`
    Specialization model stores the data about the types of specialization

CHAPTER SIX

VIEWS PACKAGE

A view function, or view for short, is simply a Python function that takes a Web request and returns a Web response.

`doctors.views.`**`booking`**`(`*request*, *doctor_id*`)`
Display the confirmation message about the successfull booking process

`doctors.views.`**`booking_form`**`(`*request*, *doctor_id*`)`
Render the booking form. The form is rendering after the user requested to book a specific doctor.

`doctors.views.`**`home`**`(`*request*`)`
Render the home page of the myDoc System

`doctors.views.`**`search`**`(`*request*`)`
Render the page with the doctorlist based on the desired search requirements requested by user

`doctors.views.`**`showDoctorDetails`**`(`*request*, *doctor_id*`)`
Render the page with the details about the doctor choosen by the user from the list of doctors