IITG Hospital System

Technical Documentation

Group 2

4/22/2015

This is the technical documentation of IITG Hospital System Project being handled by the Group 2

❖ Project-Overview:

- The aim of the project is to develop an web application that caters to the needs of the IITG Hospital and we have divided the work among the two groups and our group has further divided our part of the project into 5 modules: RTC, Complaint-Booking, Follow-up, Insurance and Reimbursement, Medicine Search.

- The RTC or the Real Time Communication module involves the communication inside the hospital that is all the activities during a patient's visit to hospital i.e. the patient's appointment, sending him to the doctor, check-up (prescription), and the patient can take the prescription back at the reception. Our team has worked on the following features :

  - Reception adds the requests
  - Reception has a clear view of the pending waiting and completed requests
  - Doctor can see the current patients medical history
  - Doctor prescribes medicines
  - Doctor can see his previous day's work
  - Receptionist receives the final prescription

- The Follow up module deals with the follow up of the patient after his visit to the hospital the features of this model are as follows:
  - Patient receives the follow up request and has all his follow ups lined up which are yet to be filled.
  - The patient can delete the ones he want to skip and the ones which aren't replied/filled in time will be made invalid in short time
  - Doctor can view the follow up and he can delete the follow up's which have already been looked at.
  - Doctor can also reply to patients follow-up and he cannot edit the replies or send multiple replies to a follow up.
- The complaint Booking Module involves the complaint forum wherein the patient, doctor and other concerned users can lodge in their complaints , the features are as follows :
  - The user can lodge in complaint and view the complaint status.
  - The user can cancel the complaint and the admin can cancel an invalid complaint too

- The admin can reply to the complaints
- The user can view the replies to the complaint

- Medical Search Module deals with the search option, there are two fields for search : search by name and search by disease, there is also an auto-complete i.e. suggestions will be provided once we enter a keyword in the search field

- Insurance Module, this module deals with the insurance and reimbursement of the hospital system the patient or the user can apply for an insurance and reimbursement and view his applications status and the admin can accordingly make any changes and do the required process.

- **Technical Requirements:**

    - In this project we have used Python 2.7 language.
    - We have used Django 1.8 , a python based web framework as our platform.
    - We have test this app in Chrome , Firefox , IE 8+.
    - Additional dependency required is python pytz package

- The project can be divided into multiple layers: the model layer , the view layer, the template layer.

Technical Specifications

❖ Model Layer:

This layer deals with the structuring and manipulating of the data in the module. Each class of this layer is equivalent to the layout of a table in a database and fields of the classes correspond to the columns in the database. The various classes in the layer are listed below :

- **CRequest-class**:
  - This class is the database for the requests handled by the receptionist i.e. the appointments that are pending or waiting or completed which we shall see so. The Fields used in this class:
    - **status_choice**
    - **name**
    - **reg_no**
    - **problem**
    - **request_date**
    - **appoint_date**
    - **appoint_no**
    - **doctor**
    - **done**
    - **outsider**
    - **status**

  - **status_choice:**

    This is a mapping which maps Waiting to 0 ; Pending to 1 ; Completed to 2 ; just a choice for status of Appointment.

  - **name:**

    A CharField for the name of patient of maximum length of 140 characters.

  - **reg_no:**

The reg_no is a Foreign Key, a many to one relationship with the patient class or the database a patient can have multiple appointments but an appointment is unique to a patient thus a many to one relationship.

- **problem:**

A CharField for describing the problem of a patient. It is give maximum length of 140 characters.

- **request_date:**

A DateTimeField for taking the request date the time the request is being made i.e. the appointment is being made

- **appoint_date:**

A DateTimeField for taking the appointment date i.e. the time of the appointment

- **doctor:**

A many to one relationship(Foreign Key).A Doctor can receive appointments from different patients.

- **done:**

It is a boolean field (is filled). This is used to check if the appointment (request) has been completed or finished.

- **outsider:**

It is a boolean field (is filled). This is used to check if the patient is an outsider or from IITG.

- **status:**

It is a CharField which should be chosen from drop down menu with 3 choices described above in the status choice field.

- **methods in CRequest:**

__unicode__(self):

Object's representations are used throughout Django's automatically-generated admin so this function will return a bytestring, with characters encoded as UTF-8.

- **Prescription Class:**
  - This class is the database for the prescriptions prescribed by the doctor, later on the prescription will be printed at the reception which we shall see later on.Fields used in this class :

    - **reg_no**
    - **doctor**
    - **disease**
    - **medicine**
    - **disease**
    - **symptoms**
    - **prescription_time**
    - **nedtime**
    - **next_visit**

- **reg_no:**

  The reg_no is a Foreign Key, a many to one relationship with the patient class or the database a patient can have multiple appointments but an appointment is unique to a patient thus a many to one relationship
- **doctor:**

  A many to one field. A Doctor can give prescription to different patients.
- **disease:**

  A CharFIeld for disease name to which doctor gives description.It is given maximum length of 256 characters.
- **medicine:**

  This field is a Text-field that is for the medicines.
- **symptoms:**

  It is a TextField where the doctor writes the symptoms of the disease.
- **prescription_time:**

  It is a DateTimeField for taking the time of prescription by doctor.
- **medtime:**

  It is a Text-Field $$
- **next_visit:**

  It is an Integer-Field. $$

- **Doctor Class:**
  - This class is the doctor database i.e. it stores the doctor usernames and names and the fields in this Class:

    - **username**
    - **name**

  - **username:**

    It is a CharField for login user name of the Doctor.It is given a maximum length of 255 characters.
  - **name:**

    It is a CharField for doctor actual name.It is given maximum length of 20 characters.
  - **Methods in this class:**

    __unicode__(self): Object's representations are used throughout Django's automatically-generated admin so this function will return a bytestring, with characters encoded as UTF-8.

- **Patient Class:**
  - This class is the database of the patients that is the one that holds the data of the patients this will populated by the other group and the fields in this Class:

- **username**
- **reg_no**
- **name**
- **age**
- **height**
- **weight**

- **username:**
  It is a CharField for login user name of the patient.It is given a maximum length of 255 charracters.
- reg_no:
  It is a CharField for registration number of the patient.It is given maximum length of 10 characters.
- **name:**
  It is a CharField for actual name of the Patient.This Field is given maximum length of 20 characters.
- **age:**
  It is an IntegerField for age of the Patient.
- **height:**
  It is a CharField for height of the Patient.It has been given maximum length of 100 characters.
- **weight:**
  It is an IntegerField for weight of the Patient.

- **Reception Class:**
  - This class is the database of the receptionists and the Fields of Reception: username

  - **username:**
    It is a CharField for username of the Receptionist.It has been given maximum length of 255 characters.

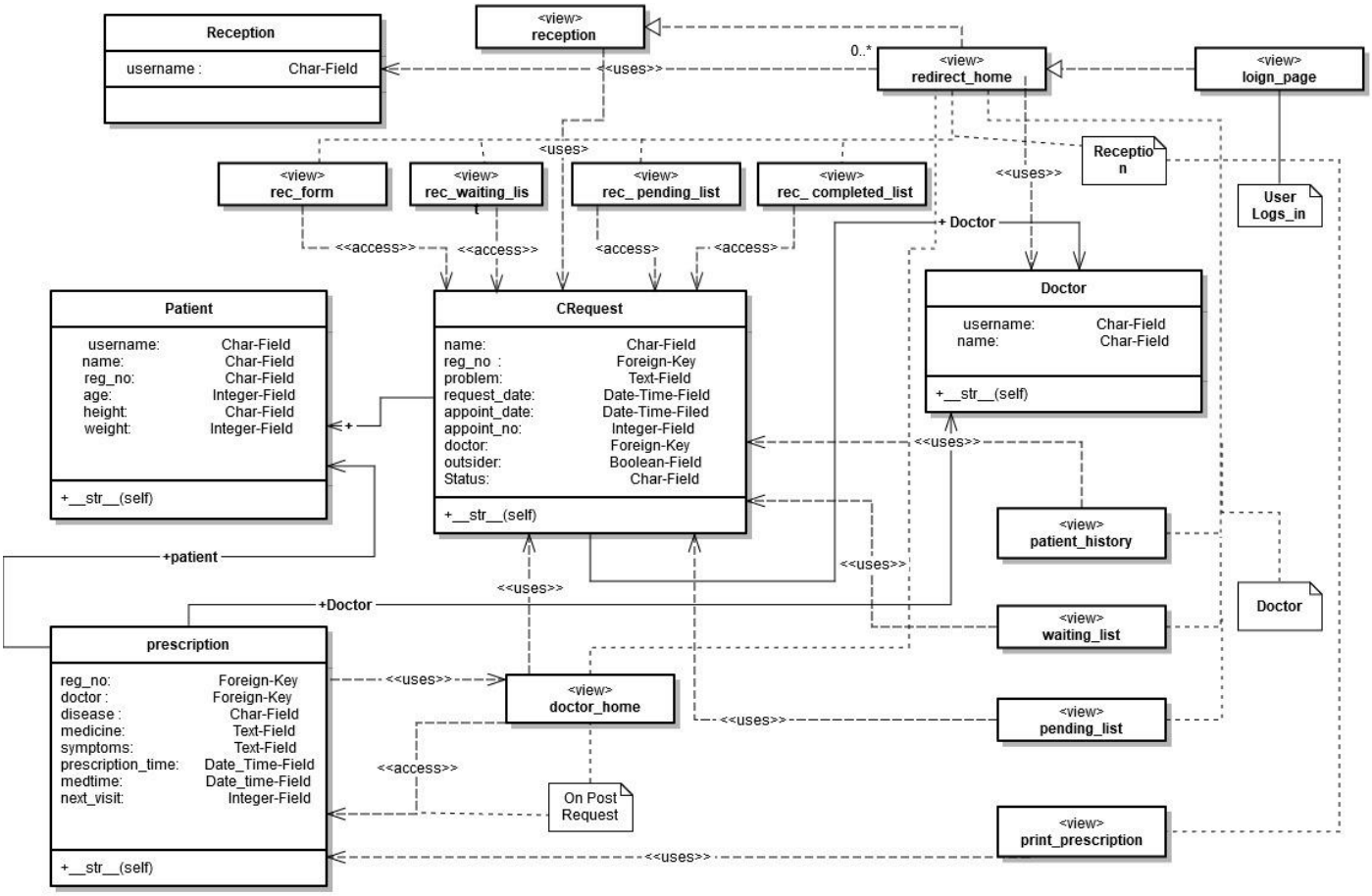
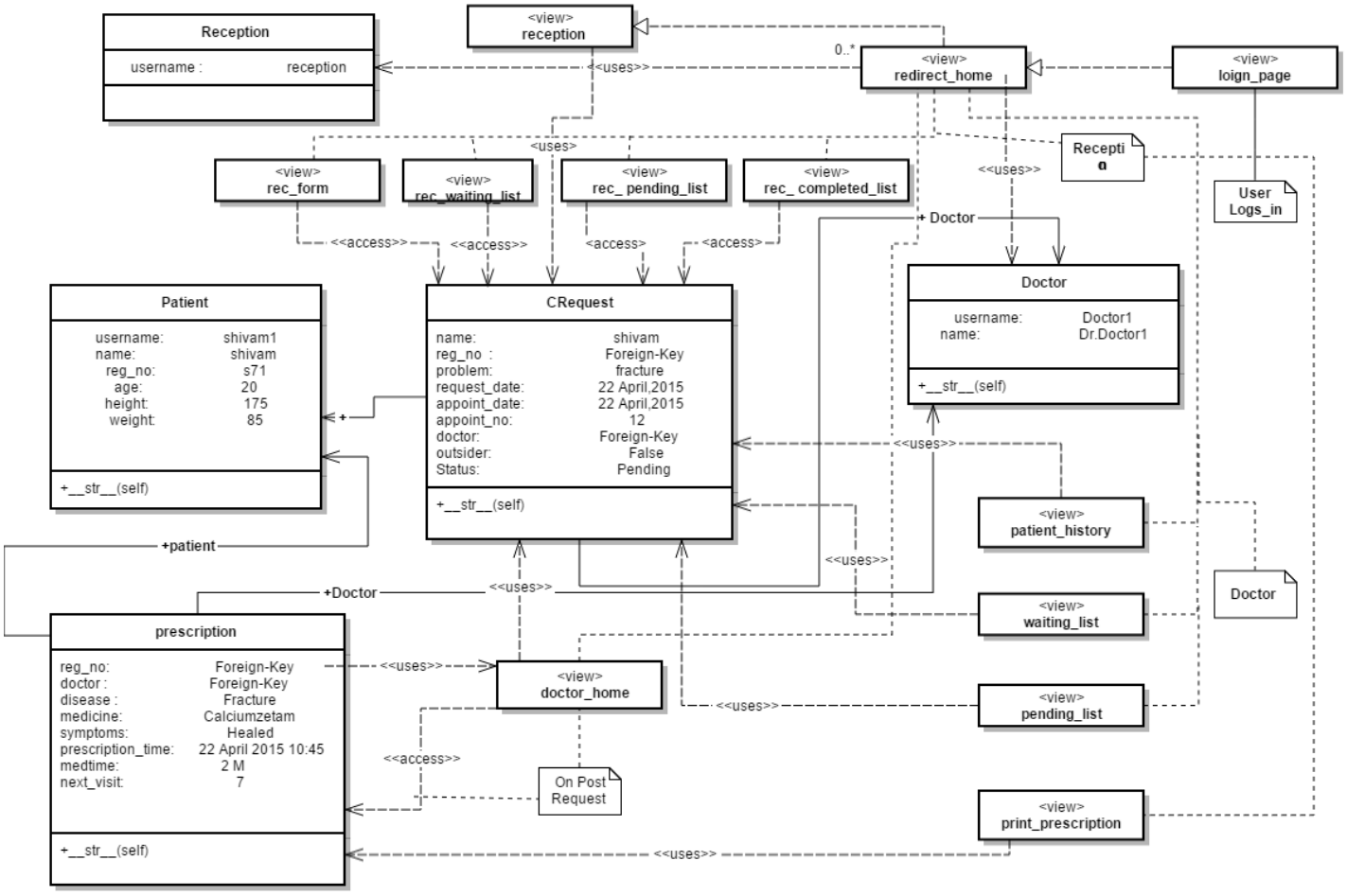


Fig. Class-Diagram



Object-Diagram : Rtc

- **Follow up :**
  - **prescription = models.OneToOneField('Prescription', primary_key=True)**

    A follow-up has one-to-one correspondense to a prescription, i.e. a prescription has a single follow-up for the patient to fill and a follow-up is of a single prescription (can't be for multiple prescriptions)

  - **is_filled=models.BooleanField(default=False)**

    A follow-up has a boolean field (is_filled). This is used to check if the follow-up has been filled by the user.

  - **rating=(('1','poor'),('2','Average'),('3','good'),('4','very good'),('5','excellent'))**

    This is the mapping which maps poor->1 (integer value) (for analysis purposes) and so on.

  - **patient = models.ForeignKey('Patient')**

    A many-to-one relationship. A patient may fill multiple Follow-ups from multiple doctors.

  - **doctor = models.ForeignKey('Doctor')**

    A many-to-one relationship. A doctor may receive multiple Follow-ups from multiple patients.

  - **title = models.CharField(max_length=200)**

    A title field (char field) for the follow-up of maximum length 200 chars

  - **description = models.CharField(max_length=1000)**

    A description field (for patients to give a detailed follow-up).

  - **rating = models.CharField(max_length=15, choices=rating)**

    A rating must be chosen from a dropd-down menu with five choices to choose from.

  - **Doctor_delete = models.BooleanField(default=False)**

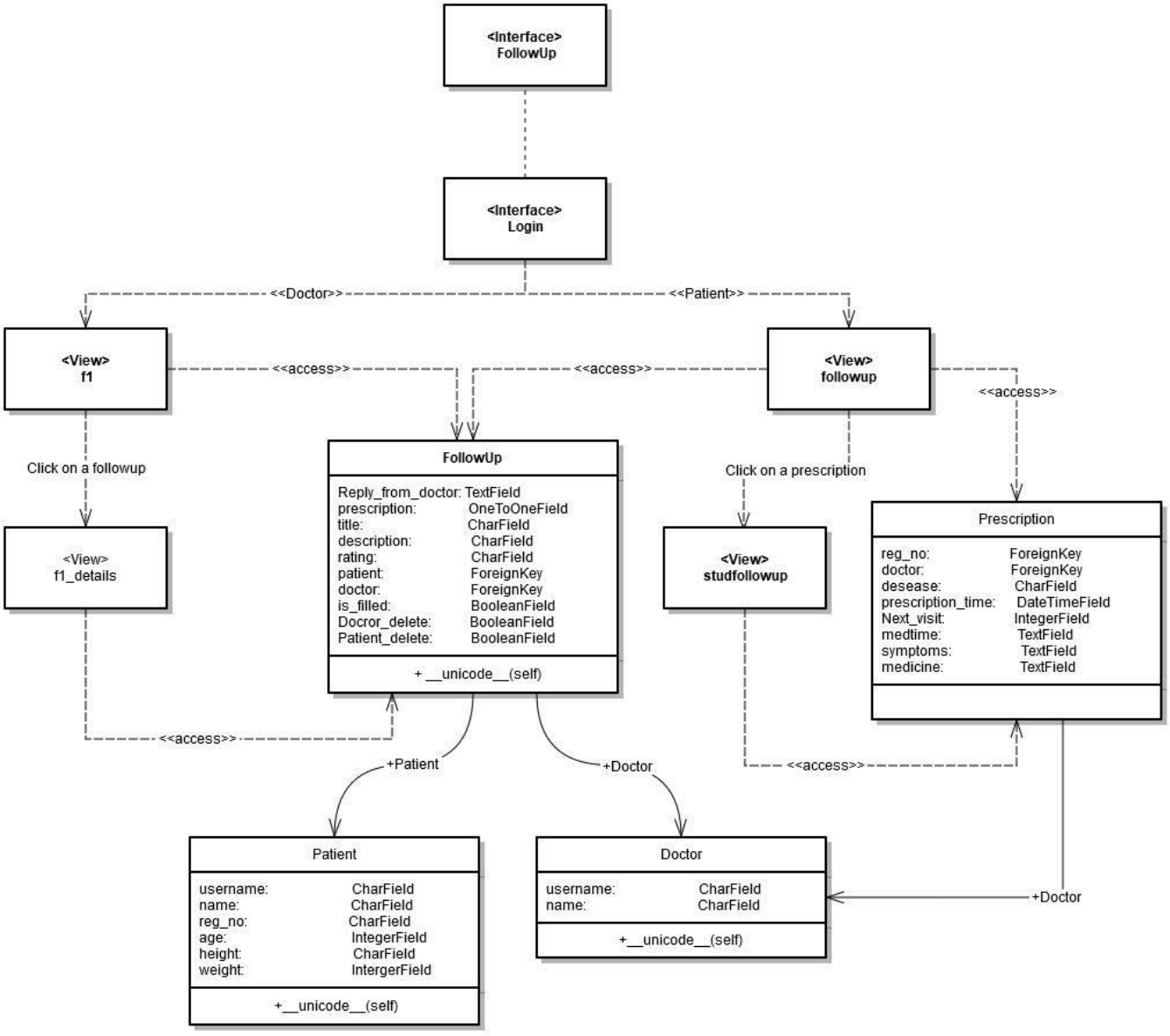
    A boolean to check if the doctor has removed a follow-up from his feed (NOTE: the deleted follow-up still remains in the datebase)

- **Patient_delete = models.BooleanField(default=False)**

  A boolean to check if the patient has removed a follow-up from his feed (NOTE: the deleted follow-up still remains in the datebase)

- **Reply_from_doctor = models.TextField(blank=True)**

  A text-field to store the reply from the doctor to the follow-up (patient)



Class-Diagram-Follow_up

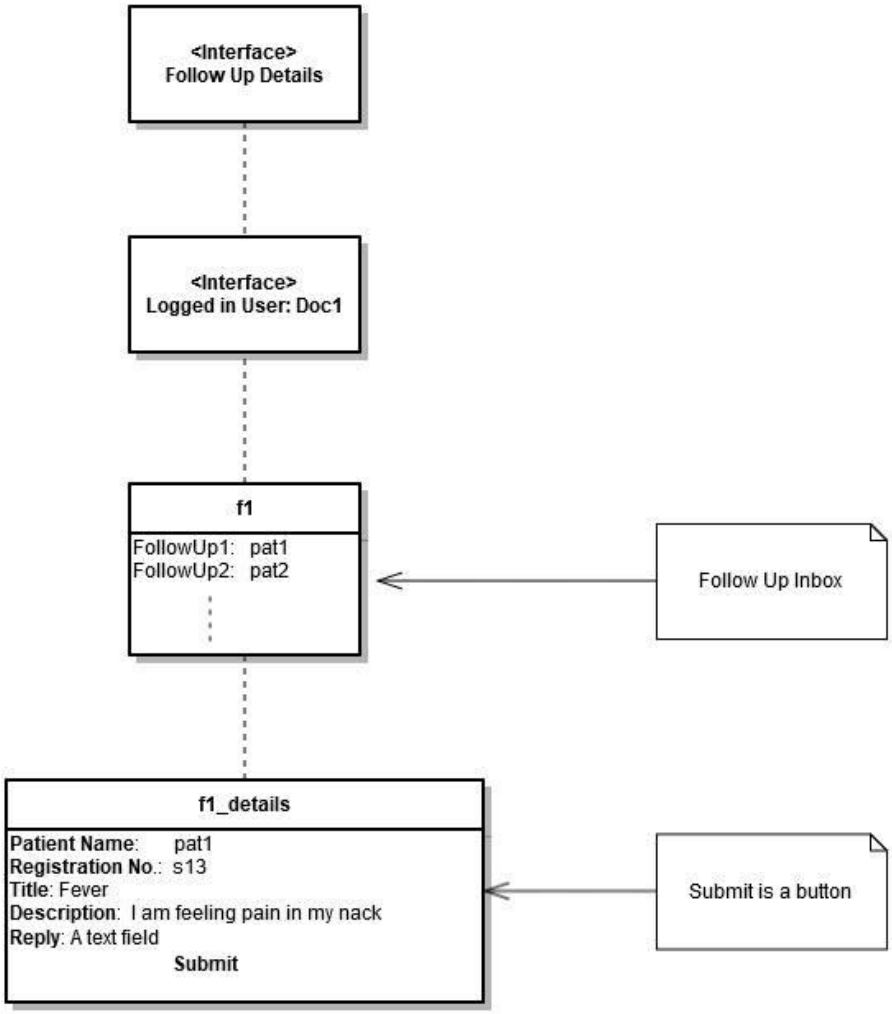


Fig-Object –Diagram: Follow-up

- **class Complaint(models.Model):**

  - **username = models.CharField(max_length=140):**

    it is username of the student when a complaint is booked against a particular category and it is not unique as a person can give complaint any number of times.It is an not null field

  - **category = models.CharField(max_length=20):**

it contains the category that is on what particular topic the person is booking the complaint it cantins fields like ambulance ,hygene,staff,medicine.it is an not null field

- **subject = models.CharField(max_length=140):**
  It is simply on what the complaint is about it is similar to the the subject in an email it will give an glimpse of the topic.it is an not null field

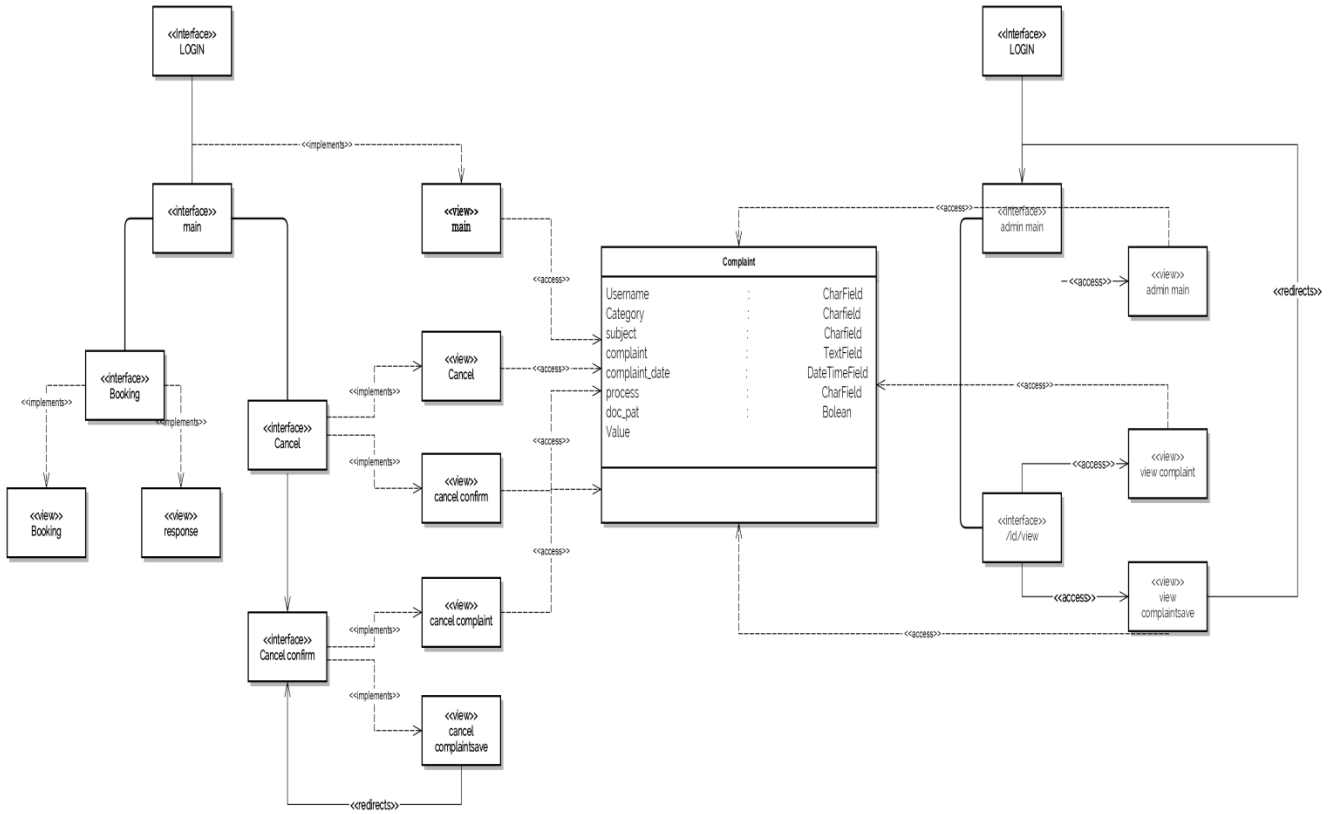
- **complaint = models.TextField(blank=True):**
  It is the description of the complaint to be booked

- **complaint_date = models.DateTimeField():**
  it contains the time and date at which the complaint is booked and it automatically stores the server time using timezone.now()

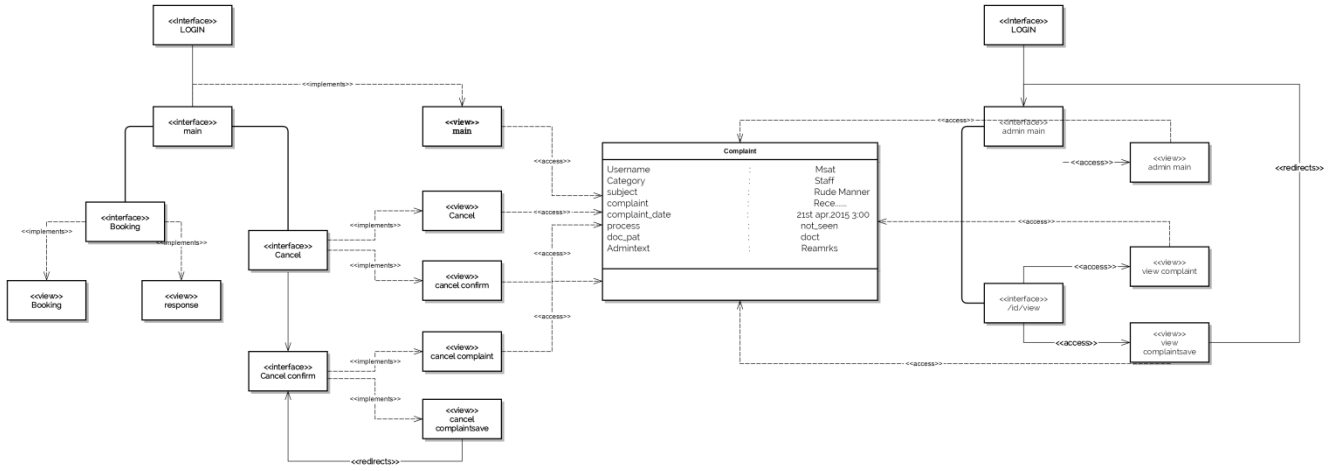
- **process = models.CharField(max_length=50 ,default='not_seen'):**
  It is the status of the complaint that is whether it is on process or completed it will be updated by the admin and the default value is not seen

- **doc_pat = models.BooleanField(default=0):**
  It stores a Boolean value of 0 for students and 1 for doctor

- **admintext = models.CharField(max_length=140,default='admin'):**
  It will store the remarks given by the admin when the complaint is processing or completed



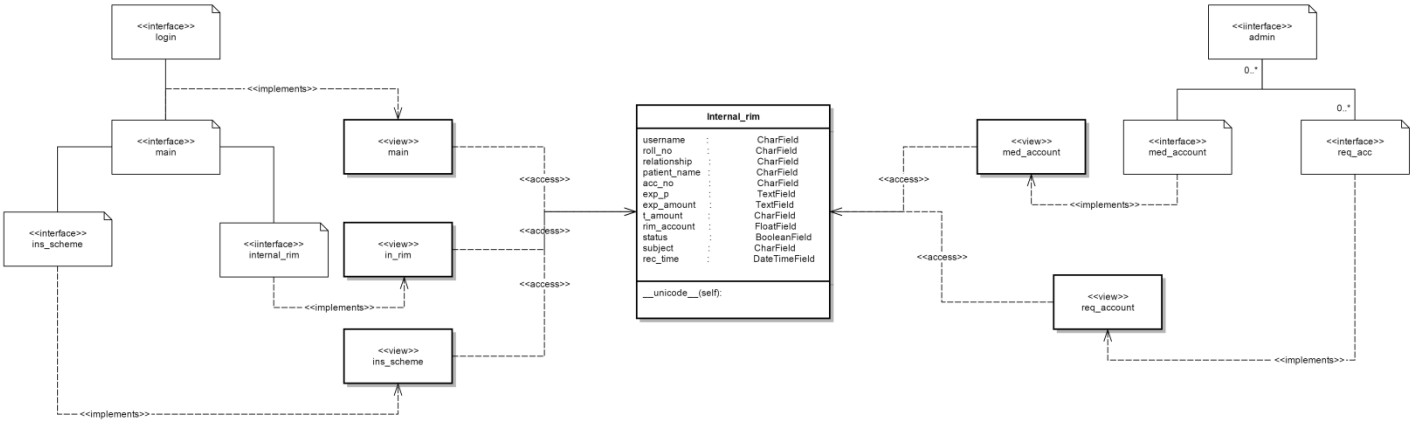
Class-Diagram-Complaint Booking



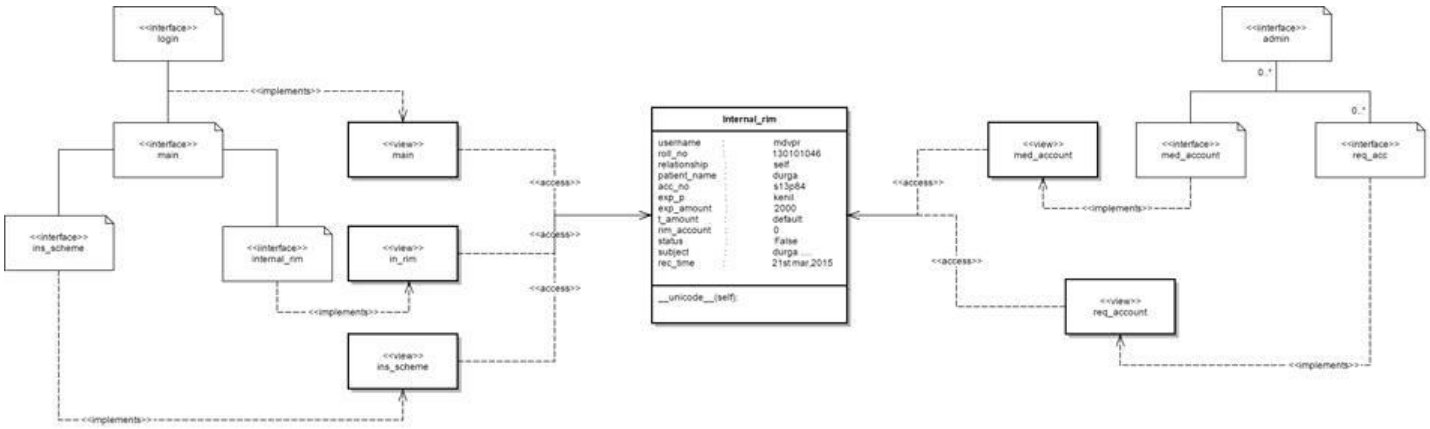
Object: Complaint-Booking

- **Internal_rim(models.Model):**

  - **username = models.CharField(max_length = 255):**
    it stores the username of the user
  - **roll_no = models.CharField(max_length = 10):**
    it stores the roll no of the user
  - **relationship= models.CharField(max_length=50, blank=True):**
    it stores the relationship of the person with the student.It will be used only when students realtives used the hospital facility with proper id and information
  - **patient_name= models.CharField(max_length=255,default='Self'):**
    it stores the patient name or user name who is applying for the reimbursement
  - **acc_no= models.CharField(max_length=50):**
    it stores the bank details that is account number
  - **exp_p=models.TextField():**
    it stores the expense parameters that is on what the expenses are made and how they are used
  - **exp_amount= models.TextField():**
    it stores the total amount spent on the parameters that are mentioned above
  - **rim_amount= models.FloatField(max_length=10, default=0):**
    the reimbursement amount that was provided by the accounts and finance section
  - **status= models.BooleanField(default=False):**
    the status of the applied reimbursement amount
  - **req_time=models.DatetimeField()**
    this shows the date and time when the request for reimbursement is filled.
  - **subject=models.CharField(max_length=255)**
    this shows the subject for the reimbursement so the user can recognize why did he applied for reimbursement.



Class-Diagram: Insurance



Object-Diagram : Insurance

- **Medicine(models.Model):**

  - **name=models.CharField(max_length=140)**
    This stores the name of the medicine.
  - **salt=models.TextField(blank=True)**
    This stores the salts that are used in the medicine.
  - **mg=models.TextField(blank=True)**
    This stores the mg quantity or the power of each medicine.
  - **price=models.FloatField**
    This stores the price of the medicine.

- **disease=models.TextField(blank=True)**
  This stores the diseases for which this medicine is useful
- **misc= models.TextField(blank=True)**
  This stores the miscellaneous info about a medicine.
- **prsc_required=models.BooleanField()**
  This stores whether a prescription is required to get it from the pharmacy.

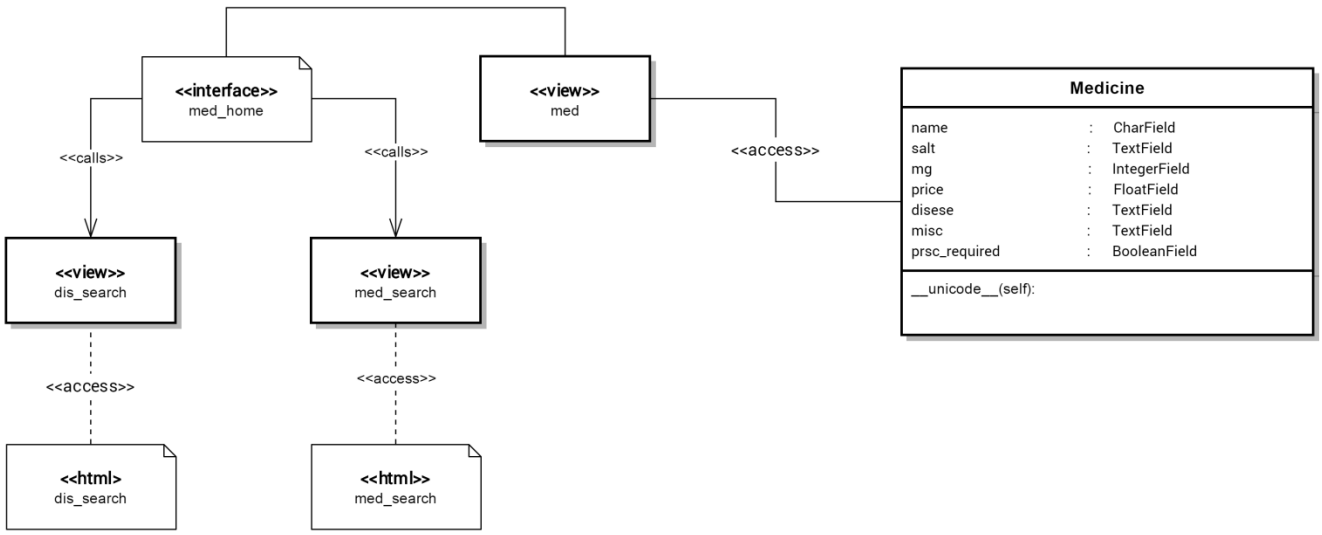
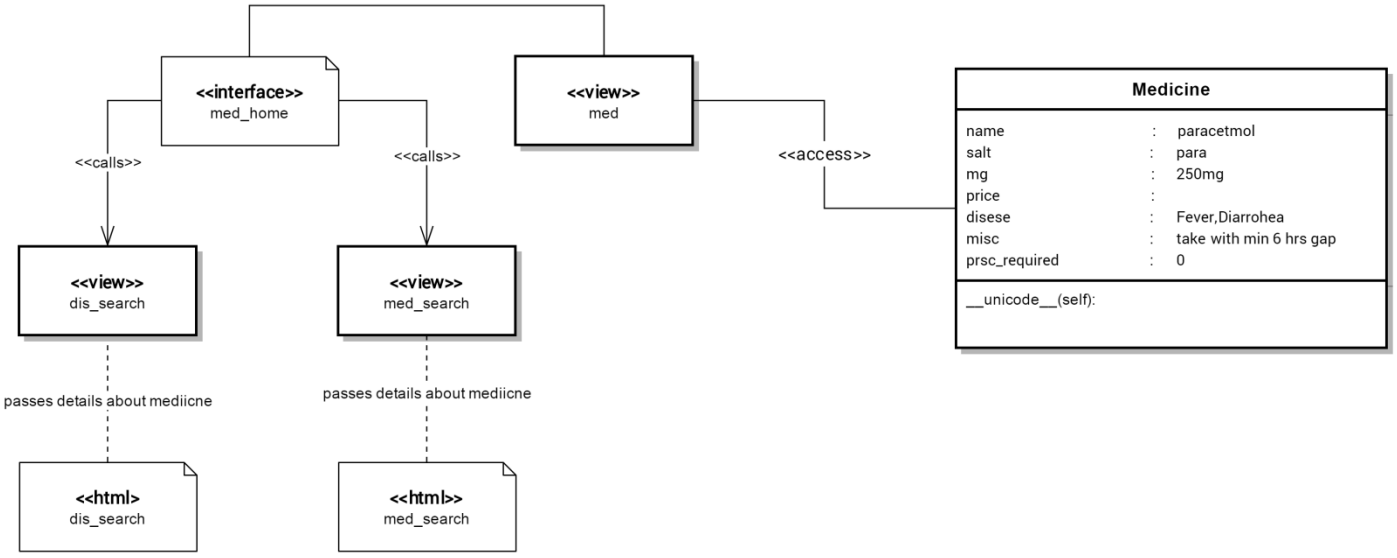


Fig-Class-Diagram-Medical-Search



Object-Diagram: Medicine-Search

❖ View Layer:

- This layer is to encapsulate the logic responsible for processing a user's request and for returning the response and the two major components are the url-confs and the view functions which are as follows :
- url-confs:
  It's a simple mapping between URL patterns (simple regular expressions) to Python functions (your views). Django determines the root URLconf module to use. Ordinarily, this is the value of the ROOT_URLCONF setting, but if the incoming Http Request object has an attribute called urlconf (set bymiddleware request processing), its value will be used in place of the ROOT_URLCONF setting then it loads the module and looks for the variable url-patterns which are as follows :
  - ✓ url(r'^$', views.home_page, name='home_page'),
  - ✓ url(r'^login/(?P<tag>\w+)$', views.login_page, name='login_page'),
  - ✓ url(r'^$', views.login, name='reception'),
  - ✓ url(r'^rec_form', views.rec_form , name='rec_form'),
  - ✓ url(r'^redirect_home', views.redirect_home , name='redirect_home'),
  - ✓ url(r'^reception', views.reception, name='reception'),
  - ✓ url(r'^doctor_home', views.doctor_home, name='doctor_home'),
  - ✓ url(r'^waiting_list',views.waiting_list,name='wl'),
  - ✓ url(r'^completed_list',views.completed_list,name='cl'),
  - ✓ url(r'^current_patient',views.current_patient,name='cp'),
  - ✓ url(r'^patient_history',views.patient_history,name='ph'),
  - ✓ url(r'^rec_completed_list',views.rec_completed_list,name='rcl'),
  - ✓ url(r'^rec_waiting_list',views.rec_waiting_list,name='rwl'),

- ✓ url(r'^rec_pending_list',views.rec_pending_list,name='rpl'),
- ✓ url(r'^print/(?P<tag>\w+)$',views.print_prescription,name='pp'),
- ✓ url(r'^logout$', views.logout_page, name='logout'),
- ✓ url(r'^f1/$', views.f1, name='f1'),
- ✓ url(r'^f1_detail/(?P<pk>[0-9]+)$', views.f1_detail),
- ✓ url(r'^reply/(?P<pk>[0-9]+)$', views.reply),
- ✓ url(r'^delete/$', views.delete, name='delete'),
- ✓ urll(r'^followup/(?P<pres_id>\d+)', views.followup , name='followup'),
- ✓ url(r'^followup/studfollowup/(?P<pres_id>\d+)', views.studfollowup , name='studfollowup'),
- ✓ url(r'^followup/$', views.followup , name='followup')
- ✓ url(r'^complaint/$',views.main , name='main'),
  - After login this is the main page ul
- ✓ url(r'^complaint/booking/$' , views.booking , name='booking'),
  - this is the url for the complaint booking form
- ✓ url(r'^complaint/booking/response/$' , views.response , name='response'),
  - when the submit button is clicked this url is used to save the data using response view
- ✓ url(r'^complaint/cancel/',views.cancel,name='cancel'),
  - this url is for the list of complaints that can be cancelled
- ✓ url(r'^complaint/(?P<question_id>\d+)/cancelcomplaint/$', views.cancelcomplaint, name='cancelcomplaint'),
  - when the cancel button is clicked this url is used to call the view cancelcomplaint which will delete the complaint
- ✓ url(r'^complaint/cancelconfirm/',views.cancelconfirm,name='cancelconfirm'),
  - this is url will be used when the complaint is deleted
- ✓ url(r'^adminmain/$',views.adminmain , name='adminmain'),
  - it is the url for the admin main page when the admin logs in
- ✓ url(r'^viewcomplaintsave/$' , views.viewcomplaintsave , name="viewcomplaintsave"),
  - it is the url used to save the data in the database it will not be visible
- ✓ url(r'^med/$' , views.med , name='med')
  - This is the home page url,i.e when the user clicks on it the first time this url pops up.
- ✓ url(r'^med/med_search/$' , views.med_search , name='med_search')
  - This url is the ajax request sent by the med page for the autocomplete box for the search by name.
- ✓ url(r'^med/dis_search/$' , views.dis_search , name='dis_search')
  - This url is the ajax request sent by the med page for the autocomplete box for the search by disease
- ✓ url(r'^med_account$', views.login, name='rim_login'),
  - this is used to to show the reimbursement requests for the account section person of the medical section.
- ✓ url(r'^req_acc', views.req_acc , name='req_acc'),
  - it is for the accountant.This is the page redirected when an account chooses to view a particular request.
- ✓ url(r'^in_rim', views.in_rim, name='in_rim'),
  - it is used for registering a patient for reimbursement
- ✓ url(r'^main', views.main, name='main'),

- this will be the main page of the user
- ✓ url(r'^ins_scheme,views.ins_scheme,name= 'ins_scheme')
  - this will be the page for all the links related to the external insurance scheme.

Once one of the reg-exes matches, Django imports and calls the given view, which is a simple Python function. The view gets passed the following arguments:

- ✓ An instance of Http Request.
- ✓ If the matched regular expression returned no named groups, then the matches from the regular expression are provided as positional arguments.
- ✓ The keyword arguments are made up of any named groups matched by the regular expression, overridden by any arguments specified in the optional kwargs argument to django.conf.urls.url().

If no regex matches, or if an exception is raised during any point in this process, Django invokes an appropriate error-handling view.

- **View-functions :**

  A view function, or view for short, is simply a Python function that takes a Web request and returns a Web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image and the view contains whatever arbitrary logic is necessary to return that response and all the views are placed in the file views.py and the views of the our modules are as follows :

  - login_page
  - reception
  - rec_form
  - logout_page
  - doctor_home
  - waiting_list
  - completed_list
  - current_patient
  - patient_history
  - rec_waiting_list
  - rec_completed_list
  - rec_pending_list
  - print_prescription
  - followup
  - studfollowup

- home

- **home :**

  This view function is for the basic home page

- **login_page:**

  On matching the url(r'^login/(?P<tag>\w+)$', views.login_page, name='login_page') we call this view function which checks if the passed parameter or tag is login and then if the request is a http request with the post method we take in the username and the password and the authenticate function is used to authenticate if the username and password are matching and the data is stored in the users database of Django we are actually using the Django's inbuilt users database which already encrypts the passwords and has inbuilt functions to check if the passed parameters are matching,if they are right we redirect them to the redirect_home

- **redirect_home:**

  This view function redirects the user to their respective pages i.e. it takes the user attribute from the request and then checks to which category the user belongs to using the page_type global variable and to determine the user type we check in all the user databases for the username and when we find one we determine which page to redirect to.

- **Reception :**

  This view function is called when url(r'^reception', views.reception, name='reception') is matching and then we check if the http request method is a get method and if it is we obtain the waiting list, pendinglist and completed list form the crequest database which will be explained later on and then the render(),which combines a given template with a given context dictionary and returns the Http Response object with that rendered text i.e the dictionary containing the above obtained lists to the reception.html template

- **rec_form:**

  This view is called when the following pattern url(r'^rec_form', views.rec_form , name='rec_form') is matched or the request is made and then we initially check if the user who made the request is actually in the reception database and if the doctor is not found we generate an error and redirect to the login page and if the user is a valid one we check for the type of HTTP request if its a Post request.

if (request. method == 'POST'):

- ✓ we get this request when the reception adds a new appointment or request
- ✓ we obtain all the data entered in the text-fields and then store them in the corresponding variables like the name in name problem in problem submit date time in the submit date time and so on, there is a boolean field to denote if a patient is an outsider and then for the reg_no we check if there actually exists an patient object with that specified field value and generate an error message if there isn't one and if there is we just continue similarly we check for the doc_name if there exists such a doctor.
- ✓ We then save all the data to a CRequest class and save it and then redirect to the page.

- **rec_waiting_list:**

  on receiving a request this view functions obtains the list of waiting patients that is the crequest objects which have a status attribute as 1 and store the query list into a dict variable 'waiting_list' and then the render (), which combines a given template with a given context dictionary and returns the Http Response object with that rendered text i.e. the dictionary containing the above obtained lists to the rec_waiting_list.html template and this can be accesed only by the receptionist , this will be displayed on the reception homepage

- **rec_completed_list:**

  similar to the rec_waiting list this computes the completed list of the patient and then the render(),which combines a given template with a given context dictionary and returns the Http Response object with that rendered text i.e the dictionary containing the above obtained lists to the rec_completed_list.html template and this can be accesed only by the receptionist

- **rec_pending _patient:**

  this view function obtains the pending patient details as discussed in th doctor_home function in view layer and then the render(),which combines a given template with a given context dictionary and returns the Http Response object with that rendered text i.e the dictionary containing the above obtained lists to the rec_pending_patient.html template.this page can be accessed only by the receptionist

- **doctor_home:**

  This view is called when the following pattern url(r'^doctor_home', views.doctor_home, name='doctor_home') is matched or the request is made and then we initially check if the user who made the request is actually in the doctor database and if the doctor is not found we generate an error and redirect to the login page and if the user is a valid one we check for the type of HTTP request if its a Get or Post request.

if (request. method == 'GET'):

- ✓ we obtain a query list stored in variable waiting_list the query list is returned by objects. filter() on the CRequest class and then we order it by appointment date similarly we obtain a query list again from the class CRequest, the completed list and the pending list ordered by appoint_date field of the class again and the filtration attribute is the status filed of the class.
- ✓ Another query list called the prescription list filtered by the doctor's username and ordered by the prescription time
- ✓ Then we are using the timedelta function to specify the time range attribute to the filter method to obtain the prescription query list of yesterday, a week ago and the past one month taking 1,7,and 30 as parameters.
- ✓ we then check if there's any patient in the pending list if there is we make him the current patient (curpat variable) and if there's none we'll make the first applicant in the waiting list as the new current patient and then generate current patient's history through getprescriptiondetails and then we update the waiting list and order it by appoint_date and then its saved and then the render(),which combines a given template with a given context dictionary and returns the Http Response object with that rendered text i.e the dictionary containing the above obtained lists to the doctor.html template

if (request. method == 'POST'):

- ✓ this request is obtained when the doctor submits the prescription
- ✓ First to obtain the current patients reg_no and the appointed doctor we filter the CRequest list and obtain the top most object when ordered by appoint_date.
- ✓ We then save the disease, symptoms, next_visit , to their respective variables named accordingly and the medicines and time fields are converted to strings i.e we have a set of strings we sort of append them Similarly the time variable that stores the duration of the medicine.
- ✓ Then we save these obtained data to the prescription class or database and then we update the waiting list and order it by appoint_date and then its saved and then the redirect to the doctor page. We also convert the current patients status to the completed.

- **waiting_list:**

  on receiving a request this view functions obtains the list of waiting patients that is the crequest objects which have a status attribute as 1 and store the query list into a dict variable 'waiting_list' and then the render (), which combines a given template with a given context dictionary and returns the Http Response

object with that rendered text i.e. the dictionary containing the above obtained lists to the waiting_list.html template

- **completed_list:**

  similar to the waiting list this computes the completed list of the patient and then the render(),which combines a given template with a given context dictionary and returns the Http Response object with that rendered text i.e the dictionary containing the above obtained lists to the completed_list.html template , this is for the doctor's page

- **current_patient:**

  this view function obtains the current patient details as discussed in th doctor_home function in view layer and then the render(),which combines a given template with a given context dictionary and returns the Http Response object with that rendered text i.e the dictionary containing the above obtained lists to the current_patient.html template

- **patient_history:**

  This view function called when the following url is matched , url(r'^patient_history',views.patient_history,name='ph'), computes the history of the current_patient by filtering the objects of the prescription table and obtain a query list and then the render(),which combines a given template with a given context dictionary and returns the Http Response object with that rendered text i.e. the dictionary containing the above obtained lists to the patient_history.html template

- **print_prescription:**

  This view functions receives a http request and a parameter i.e. the reg_no named as tag in the function when the url(r'^print/(?P<tag>\w+)$',views.print_prescription,name='pp') is matched and then we obtain the latest prescription of the patient with the passed reg_no by filtering the query list of his/her prescriptions and then pick the top most element after ordering by the time field and then as the medicine field has the string that has appended all the medicines we split it using the separator ',' and similarly med_time is separated we have already seen how they are appended in the doctor_home view function. After obtaining the required variables we form a dictionary variable 'context ' and then the render function as discussed earlier passes the context to the print2.html template .

- **logout_page:**

  this view function is to logout and this receives a request when the user tries to logout.

- **followup(request, pres_id=0):**

  if(request.method = GET)

  Get the username, registration number and patient id of the current logged in user using request.user make a list of prescriptions (sorted by 'id') for the active patient.Now, for every prescription of the active patient, get the prescription date, follow-up date (by adding the specified number of days)Thus make a list (show_id) of the active followups for the active patient.
  If it has been a month from the time the paticular follow-up was asked for, remove that follow-up from the list (show-id).
  For all followups in follow-up list, if patient_delete is true (i.e. patient has deleted the particular follow-up), remove that follow-up from the show_id list.
  Store all this information (which is to be passed) in an object (context) and then pass the context object to the followup view.

  If(request.method = POST)

  Get the username, registration number and patient id of the current logged in user.Now, according to the prescription id of the selected followup, we decide to fill of delete the follow-up.By default, pres_id passed to the follow-up page is 0, this happens only when user has clicked on the delete button, this means we have to delete the selected follow-ups (or, if nothing is selected, do nothing (this has an effect of refreshing the page))To delete the selected follow-ups, make a list of selected follow-ups, for each element in the list, get its prescription id and doctor name. Save this information in an object 'response' along with the title “Follow-up Not Filled” and redirect to the followup.html page.Else, if the pres_id follow-up page is not 0, get the values of title, description and rating made by the user (Patient in this case). Save this information in an object 'response' and redirect to the followup.html page.

- **studfollowup(request, pres_id=1):**

  Prescription (id, patient registration number) is fetched using its id and then it is passed to the studfollowup page.

- **f1(request):**

  Get the list of follow-ups which are for a particular active user (doctor) sorted by patient. This list is stored as an object 'context' and pass it to the docfollowup.html page.

- **f1_detail(request, primary_key):**

  Get the follow-up with the passed primary key and pass it as an object to the f1_detail.html page

- **delete(request):**

  Get the list of follow-ups which are for a particular active user (doctor) sorted by patient. Get the username of the active doctor and a list of checked follow-ups.
  Now for each follow-up in the checks list, the boolean field (Doctor_delete) is set to True.
  Redirect to the f1 view.

- **reply(request, primary_key):**
  Get the follow-up with the passed primary key and store it in 'follow store the value entered by the user in the reply field in the Reply_from _doctor attribute of follow.
  Redirect to the f1 view.

- **main:**
  it is used to retirive the complaints from the database Complaints that an particular user had previously booked when he logged in

- **booking:**

  this view is used to show the complaint booking form
  which will have a category,subject,complaint description fields

- **response:**

  this view used to save the form values to the database using POST method and after saving the values it will redirect to the main page

- **cancel:**

  this view is used to show the complaints that are not seen by the admin and they can be deleted it will not show the complaints that are under process or completed

- **cancelcomplaint:**

  this view is used to delete a complaint complaint using the get method i.e. the id of the complaint is passed through url and that complaint is deleted and it will redirect to cancelconfirm.html

- **cancelconfirm:**

this view is used to show the confirmation that the complaint has been deleted it will use the template cancelconfirm

- **adminmain:**

  this view is used to show the total complaints in the database and they can be viewed in different types that is sort by category sort by status etc..

- **viewcomplaintsave:**

  this view is used to save the changes that the admin made to a complaint that is about the status and the remarks given by him they will be updated in the database

- **med**:

  This view is first called when the home page is accesed.Its work is to search and provide the results to the page back which are displayed in the form of tables in the html page.

- **med_search:**

  This view is called when the ajax query try to get it's search results and if the method is post then this view searches for those medicines in the db and returns results to the med_search html page which are then populated in the med page

- **dis_search:**

  This view is called when the ajax query try to get it's search results and if the method is post then this view searches for those medicines in the db and returns results to the dis_search html page which are then populated in the med page.This page is called when the person tries to search by disease.

- **main:**

  It is used to show the main page for the patient when the he/she logins it contains or shows his/her reimbursement request applied by him and their status which will be updated by the accountant

- **in_rim:**

  This view is to generate a form for the reimbursement when we are using a get method that is when the url is activated and is also used to save to form values for reimbursement request in to the database which is activated via the post method.

- **med_account:**

  This will be used for the accountant when an accountant login he can see the reimbursement request which are pending are applied.

- **req_acc:**

  This will be used by the accountant and it wil be used to show the full details of an reimbursement request and he can change the status of the reimbursement and update the amount of the reimbursement that will be saved in the database

- **ins_scheme:**

  This will be seen by the user when he wishes to apply to for the insurance using the external scheme.All the links related to this are displayed on this page.

❖ Template Layer :

- The template layer provides a designer-friendly syntax for rendering the information to be presented to the user and our template layer consists of the following templates :
  - ✓ RTC-

- doctor.html
- reception.html
- rec_completed.html
- rec_waiting.html
- rec_pending.html
- completed_list.html
- waiting_list.html
- current_patient.html
- patient_history.html
- rec_form.html
- login.html

✓ Follow up –

- f1.html
- followup.html
- studfollowup.html
- docfollowup.html
- student_feedback_form.html

✓ Complaint-Booking

- Admin-category.html
- Admin-main.html
- Book.html
- Admin-status.html
- Cancel.html
- Cancel.confirm.html
- Login.html
- Main.html
- Progress.html
- Viewcomplaint.html

✓ Insurance-Module

- ins_scheme.html
- internal_rim.html
- login.html

- main.html
- med_account.html
- rec_form.html
- req_access.html

✓ Medicine-Search:

- Medsearch.html

❖ **Security:**

The web application's security is looked at carefully and we have used Django's security and our own implementations too, we have the following features:

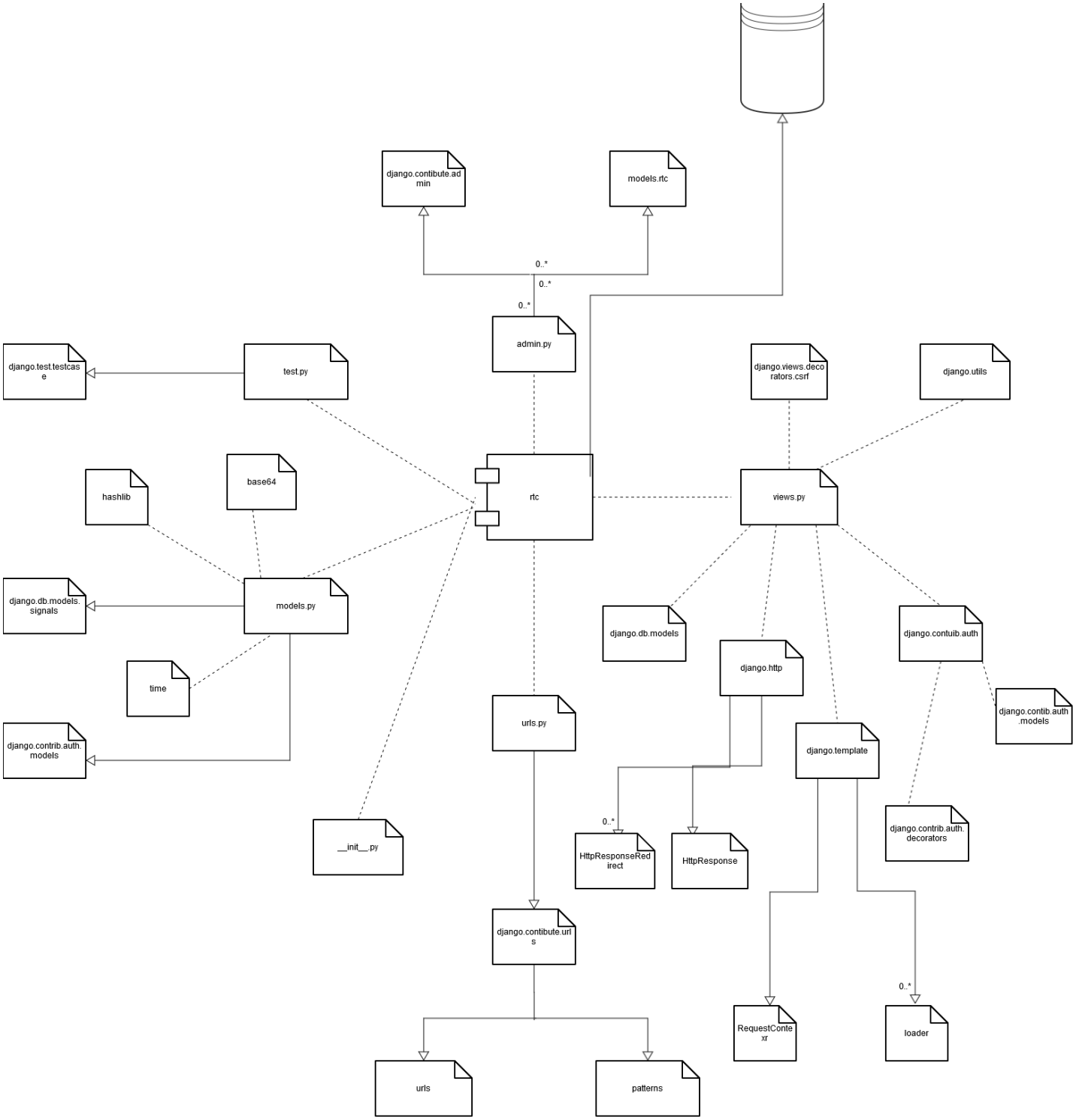
➢ CSRF attacks allow a malicious user to execute actions using the credentials of another user without that user's knowledge or consent. We used Django's

protection against most types of CSRF attacks, using it where appropriate.The protection works by checking for a nonce in each POST request. This ensures that a malicious user cannot simply "replay" a form POST to your Web site and have another logged in user unwittingly submit that form. The malicious user would have to know the nonce, which is user specific (using a cookie).When deployed with HTTPS CsrfViewMiddleware will check that the HTTP referer header is set to a URL on the same origin (including subdomain and port). Because HTTPS provides additional security, it is imperative to ensure connections use HTTPS where it is available by forwarding insecure connection requests and using HSTS for supported browsers.

- We have also used the user's database of the Django as our users database so all the usernames and passwords are encrypted using the sha256 hashing system and thus it's secure.

- And we have also handled all the exceptions if any user tries to access an irrelevant page we direct to an error page showing the user cannot access it.

- We have also implemented an Aes encryption which can be added to the project after the final integration by making the required changes for now its not included to avoid integration issues. Also we have used an asymmetric encryption algorithm RSA along with PKCS1_v1_5 to implement digital signature its a secure dsa (digital signature algorithm) this feature can also added on later on by making necessary changes to the final database structure.(YET TO BE Integrated)



RTC- Component Diagram

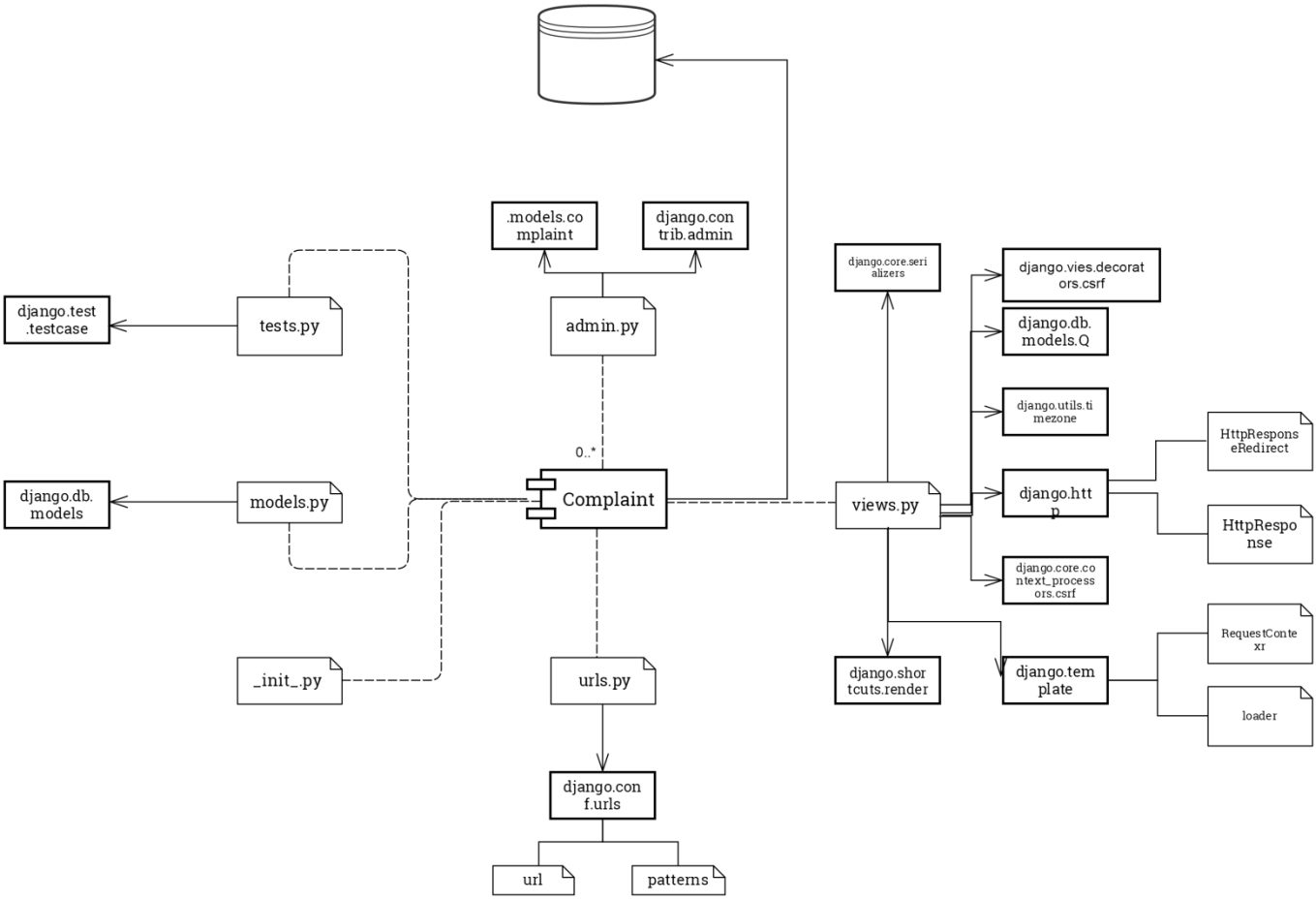


Fig-Complaint-booking components

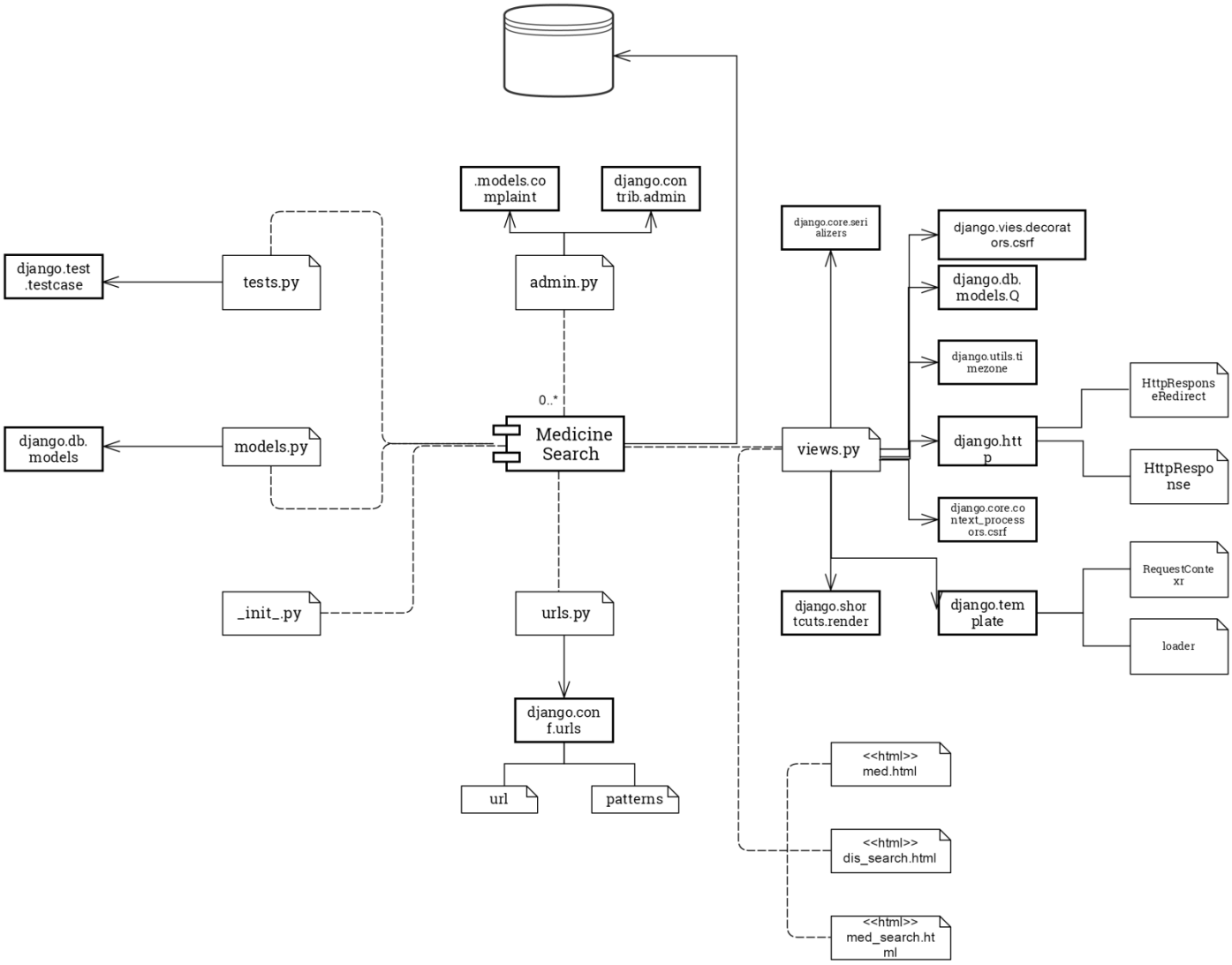
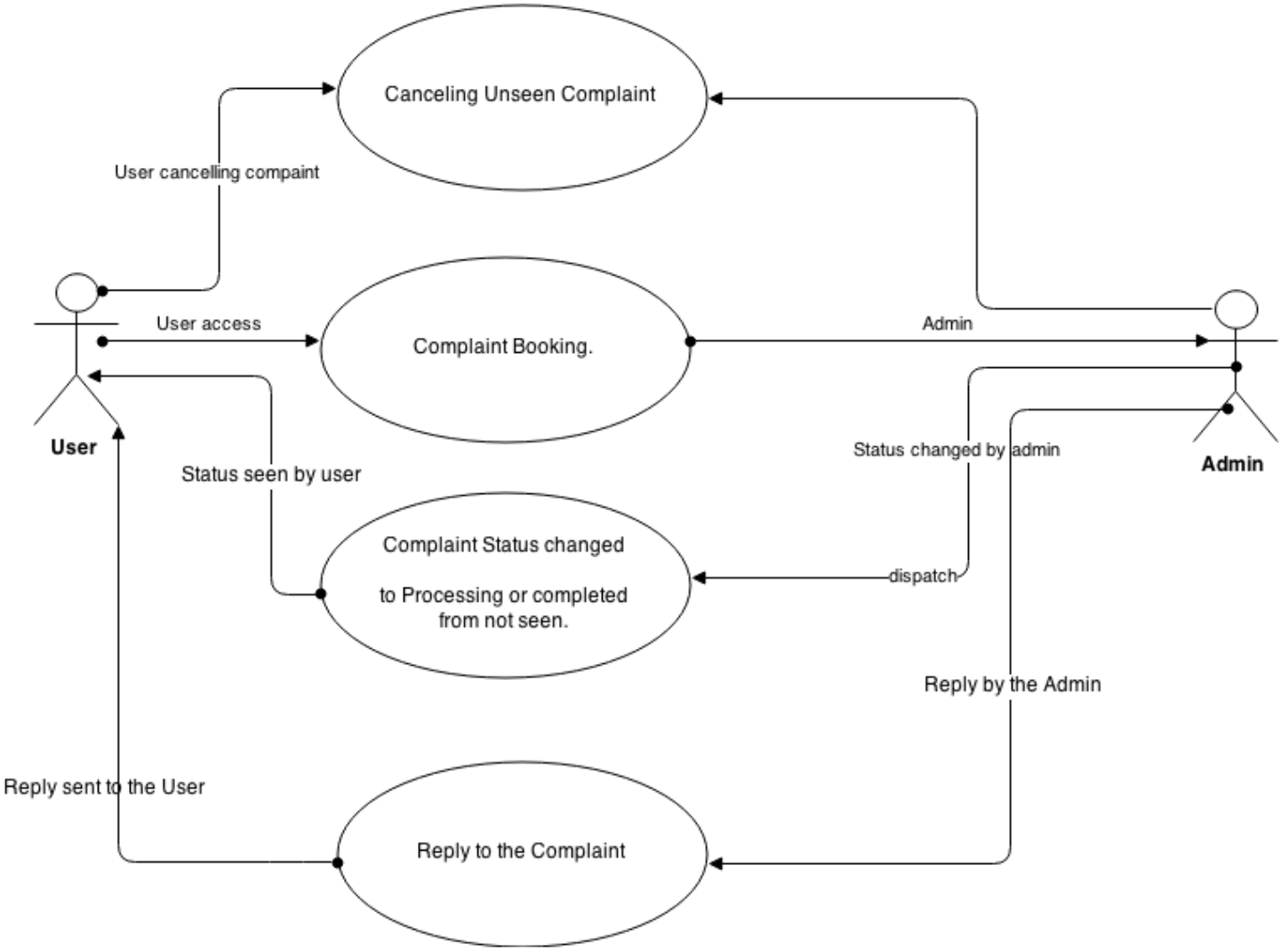
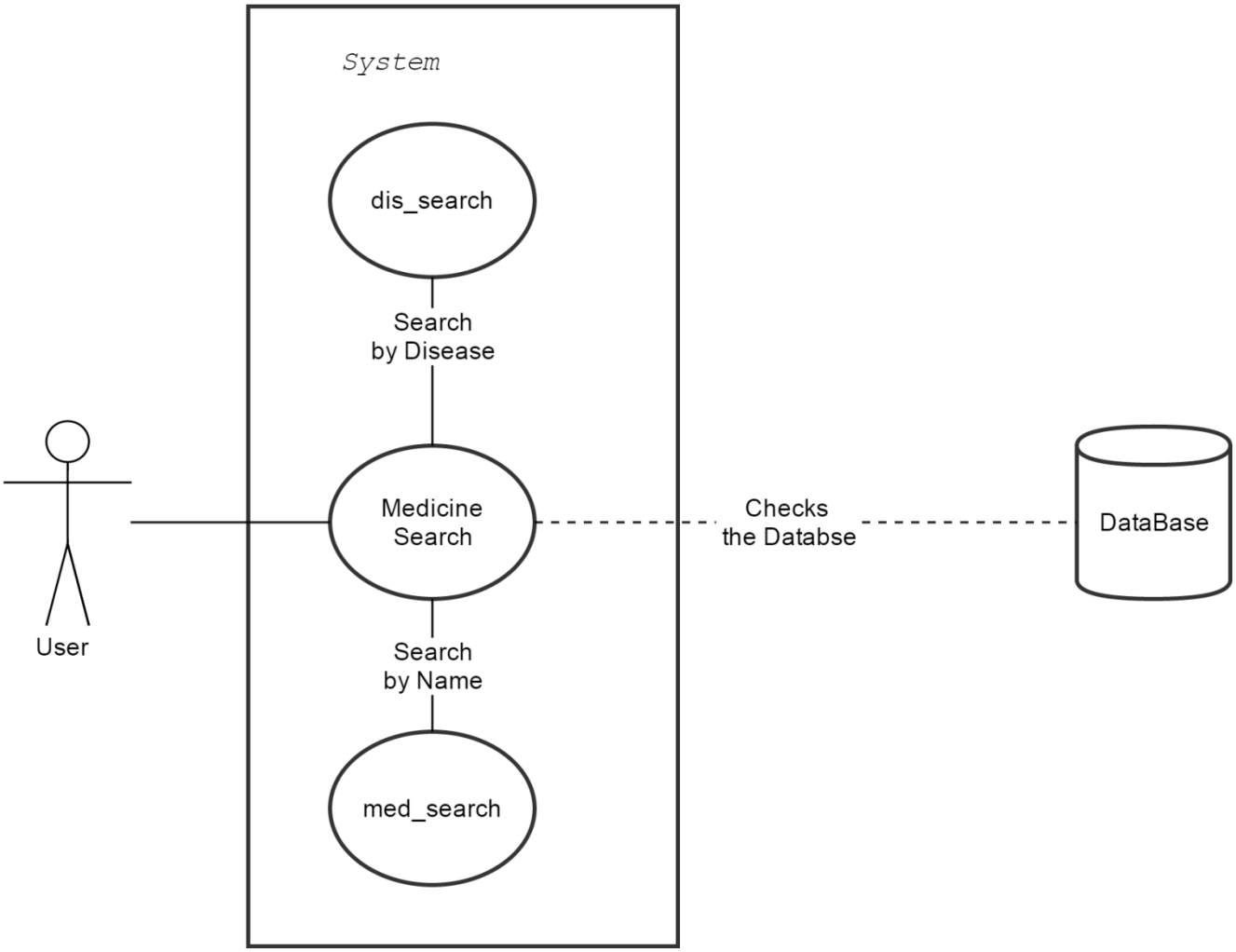


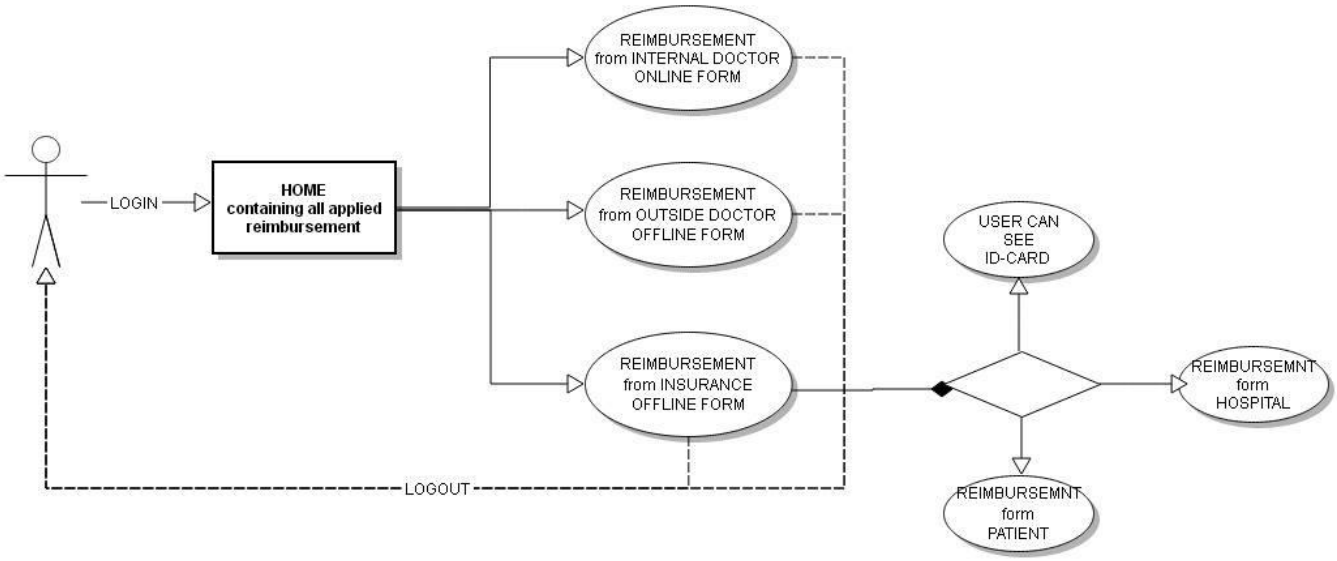
Fig-medical search components



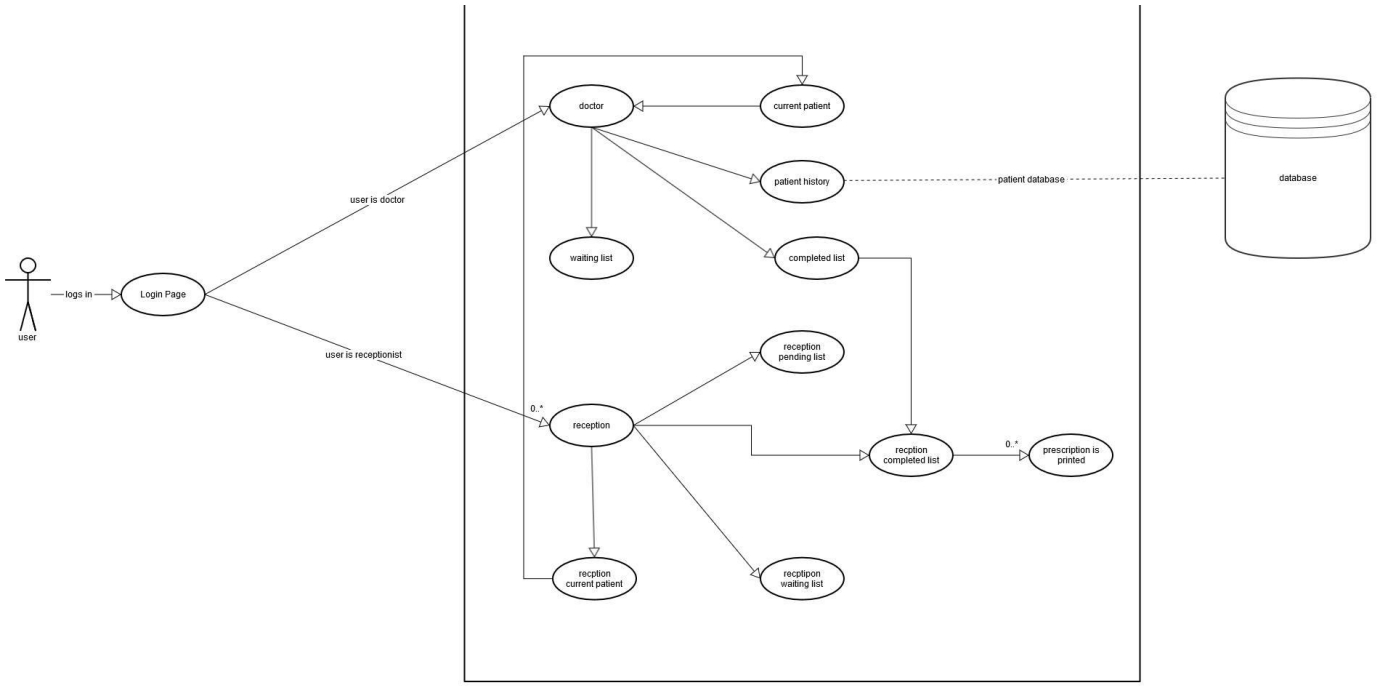
Use-case - Complaint -Booking



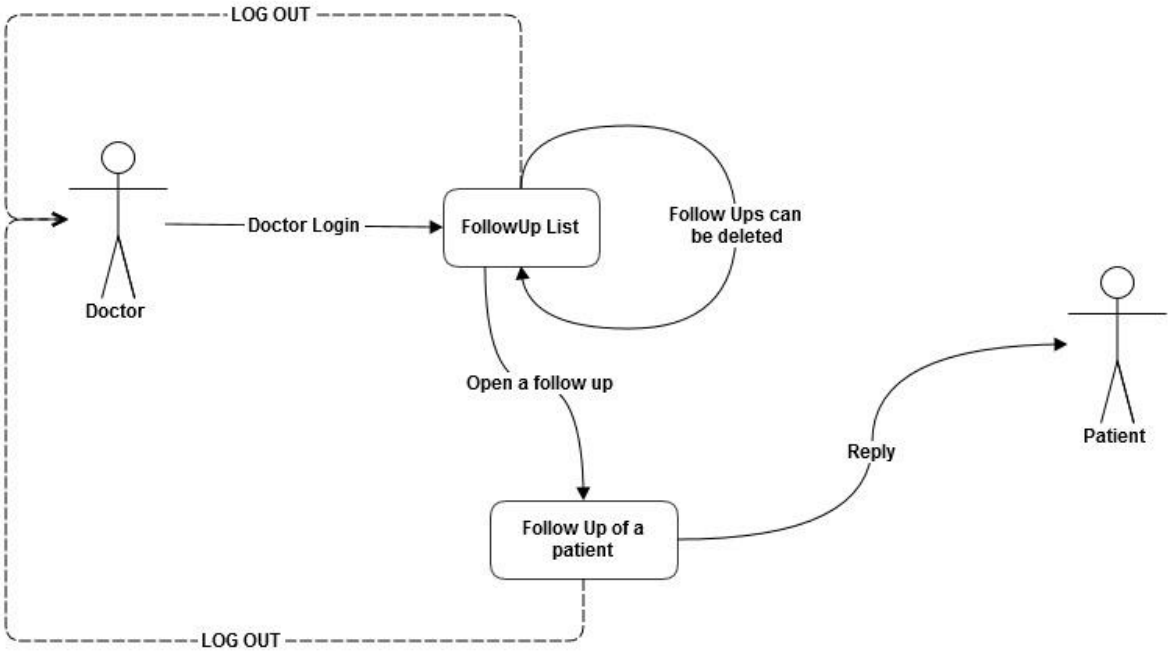
Use-case : Medicine-Search



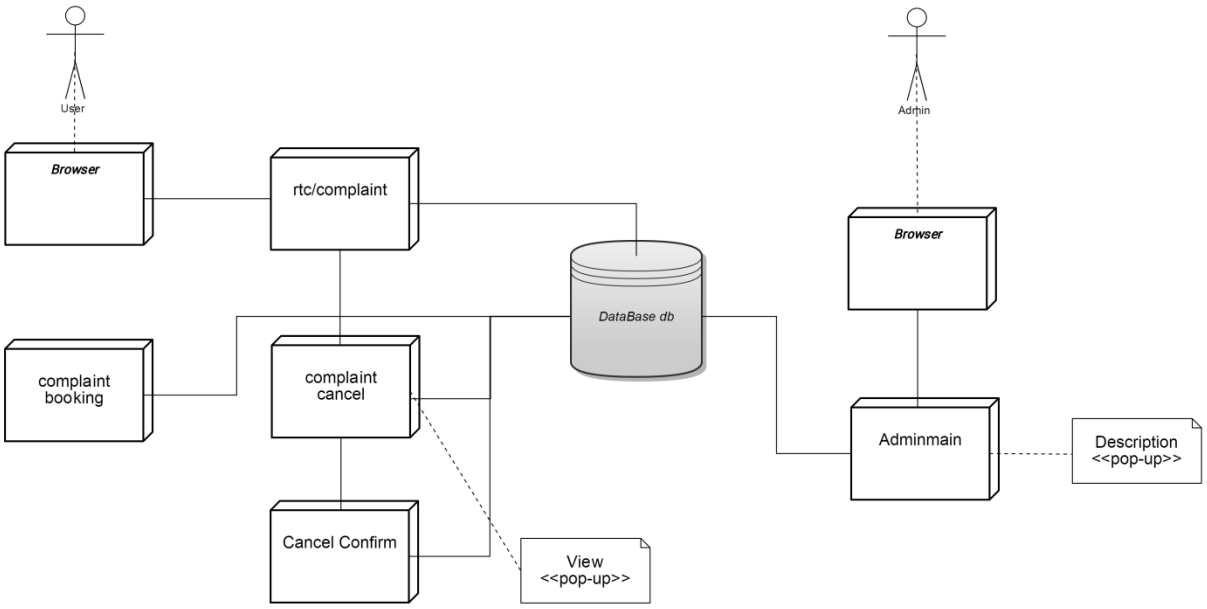
User-case: Insurance



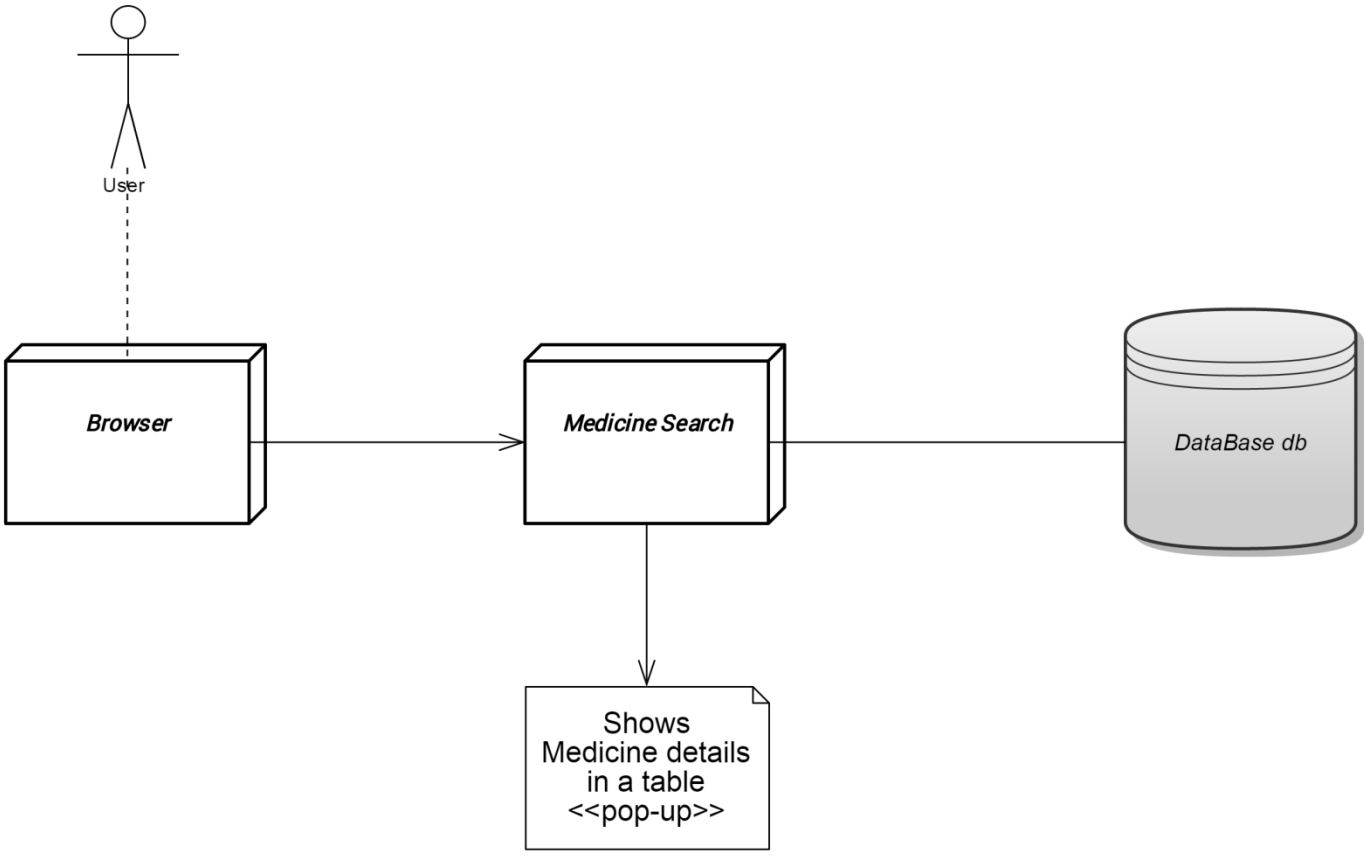
Rtc – Use cases



Follow-up : Use case 1



Deployment-Complaint Booking



`

Deployment Diagram - medicine