Project Proposal

Radiology Examination Assistant Website

Team: Wind of Change

Chenyi Cao	**Admin / Developer**
Zanlai Hu	**Developer**
Haoling Wu	**Developer**
Han Yue	**Developer**

TABLE OF CONTENTS

EXECUTIVE SUMMARY 3
1. BACKGROUND 4
2. AIMS 5
3. SYSTEM ARCHITECTURE 5
3.1 FRONT-END 6
3.2 BACK-END 6
3.2.1 DIALOGFLOW NLP CHATBOT 6
3.3 SERVER 9
3.4 PROJECT HIGHLIGHTS 9
4. PROJECT DETAIL LAYOUTS 14
4.1 ASSUMPTIONS 14
4.2 HOMEPAGE 14
4.3 PATIENT PAGES 15
4.3.1 APPOINTMENT BOOKING PAGE 15
4.3.2 APPOINTMENT INFORMATION LIST 21
4.3.3 CLINIC INFORMATION SEARCH 22
4.4 CLINIC PAGES 22
4.4.1 CLINIC DASHBOARD 23
4.4.2 REQUEST LIST 23
4.4.3 UPLOAD REPORT 25
4.4.4 STATISTICS 25
4.4.5 PROFILE EMAIL AND PASSWORD UPDATE (BOTH PATIENT AND CLINIC USERS) 26
4.4.6 QUESTIONNAIRE AND TIMETABLE BUILD 27
5. EXAMPLE CODES 31

5.1 DJANGO PROJECT STRUCTURE 31
5.2 MODELS 31
5.3 VIEWS 32
5.4 URLS 33
5.5 TEMPLATE 34
5.6 EXAMPLE 35
5.7 GOOGLE APIS 39
5.8 FLASK API 41
5.9 BOTUI 42
6. CONCLUSION 46
7. GANTT CHART 47
8. REFERENCES 48
APPENDIX A – WEBPAGES SCREENSHOT 1
APPENDIX B – COLOUR PALETTE 15
APPENDIX C– CHATBOT SCREENSHOTS 16

Executive Summary

Web-based Chatbot for Radiology Examination is a project that explores a new way for customers to booking and preparing for radiology examinations with the assistance of today's Artificial Intelligence technology. In the twenty-first century, AI techniques have experienced a rapid development following the advances in computer performance, large amounts of various data and theoretical understanding. Nowadays, these techniques have become an essential part of the technology industry, helping to solve many challenging problems in real world.

Currently, messaging is the most used way of communication in the world thanks to the various popular social applications on the Internet. Since people have got used to gather and exchange information through messaging, many businesses are already taking advantage of this and developing different kinds of chatbots to provide more convenient and humanized services to customers. Besides, with the quick development of the computer science, chatbots are becoming faster and more intelligent. Not to mention that they are able to work 24/7 and can quickly search and provide the most appropriate answers to customers anytime and anywhere. In many cases, they can provide a better service even than human.

Nowadays, complex medical examinations require appropriate preparation of a person before they visit the clinic. In traditional system, patients have to book appointments and find preparation information for the examinations themselves through searching the web-portals. However, it is not very easy for a person to find the accurate information. Due to this inconvenience, we believe there is a high demand for users to have a specific chatbot for guiding them to make appointments and also providing corresponding preparation information.

In the meantime, many big companies have already created their own bots by using popular and mature chatbot frameworks such as Google's Dialogflow. As a result, we have got enough resources and reliable techniques to build our chatbot to improve the traditional system.

1. Background

With the growing utilization of internet service, nowadays, people can do almost everything online. In healthcare area, many large hospitals had built websites for online services, however, for smaller clinics and medical centres, people are still using traditional ways for their information management: local spreadsheets, manual making reservations and send documents. With the growing proportion of immigration, more and more patients are troubled by accent differences and medical glossaries while talking over phones.

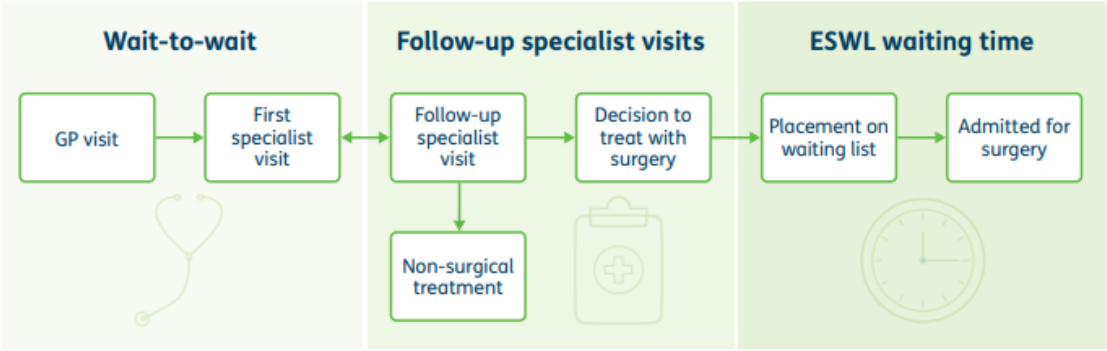


Figure 1healthcare waiting time overview in Australia (Hbf.com.au (2019))

Waiting time before medical consultation and receptionist cost are also significant. With the current flow of complete a 'medical treatment' (Figure 1), patients have to experience 2 to 3 waiting time periods before their treatment. The average waiting time before seeing a GP or specialist varies from 10 to 250 days averagely before seeing a specialist or GP (Hbf.com.au. (2019).). On the other side, $27,310 are paid to reception and documentation per employee per year, commonly 2 receptionists are hired per clinic (Work.chron.com. (2019)).

With respect to these problems within existing online healthcare services for small and medium clinics, we designed this online-assistant chatbot for radiology examinations to build a patient-friendly and clinic-convenient website. We aim to make a chatbot that is able to provide available timeslots from different radiology clinics with respect patients preferred clinic area, radiology examination type and time range. This not only reduces waiting time for patients before examination but also prevent unbalanced patient amount between clinics within similar area offering similar services. We also aim to provide documentation uploading function that enables radiologist to upload patients' reports and receipts so that patients do not have to wait for days for the posting service and achieve paperless documentation as well as reduce number of receptionists needed for clinics. Overall, the online-assistant chatbot for radiology examinations bring out from this project should achieve a 'double-win' result for both patients and clinics.

2. Aims

This full-stack website project is aimed to create a project that helps and assist both patient users and clinic users with:

1. Provide step-wise radiology examinations booking process, with google-map representation of clinic locations;
2. Help patients prepare radiology examination via chatting with on-line assistant bot;
3. Assist radiology clinic receptionists for radiology appointment management;
4. Provide clinic users report and questionnaire file uploading functions, as well as providing patient users uploaded file accessing functions;

3. System Architecture

The architecture of system can be separate to three-parts: front-end, back-end and server. The figure below shows the software composition of this project.

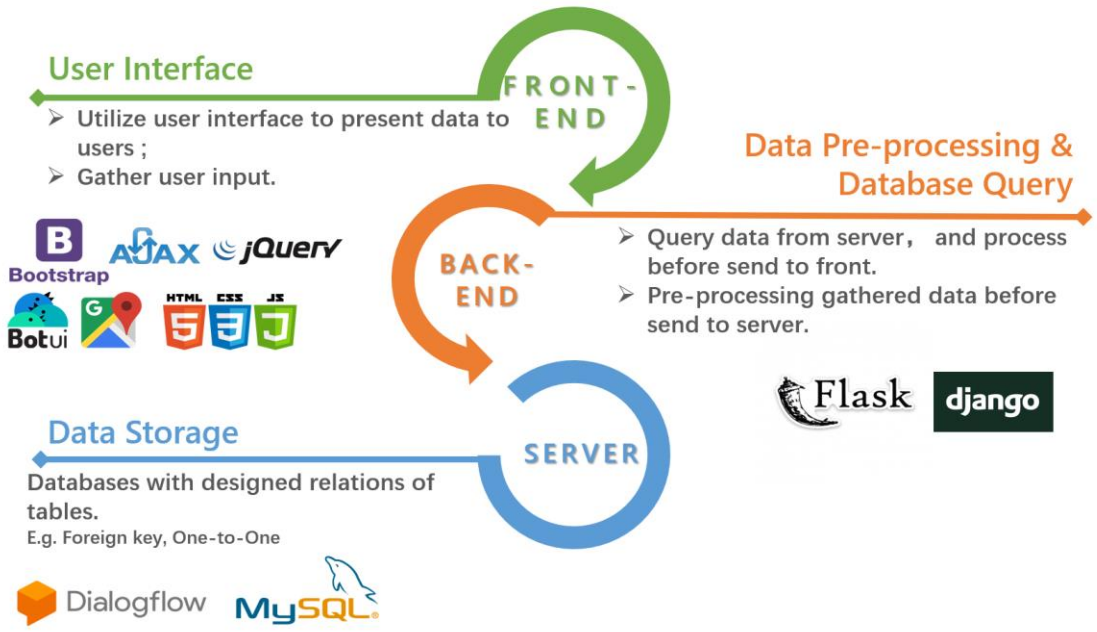


Figure 2 System Architecture

3.1 Front-end

By definition, front-end part is the user-interface designed to enable user-friendly interaction with programs *("Definition of FRONT-END", 2019)*. In this project, the front-end part is the website UIs which present back-end queried information and gather user input. As most of website UIs, our front-end composed of:

- Html5, CSS and JavaScript for UI composition and animation;
- Bootstrap for UI arrangement and layout;
- jQuery and Ajax for integrating UI information representing and transferring (to backend);
- BotUI for assistant bot layout;
- Django-styled template language for data-representation.

3.2 Back-end

Back end, or rear end part is used for data pre-processing before send to the front-end or forward to the server, as well as, data query to the database under front-end gathered query conditions. In this project, Django and Flask are used as back-end structures.

- Flask: A python based, **light-weight** WSGI web application framework, designed to make getting started quicker and easier than traditional method. In this project, Flask was mainly used to connect chatbot UI and Google Dialogflow server.
- Django: Unlike flask, python Django is a high-level Web framework that encourages rapid development, and clean, pragmatic design of web projects. This project is a Django project, which runs as a server, the back-end part is the **views** part of Django framework, which responsible for processing data and send information (in URLs).

  E.g. When a user filled his/her information in the signup page and click “Submit”, the views (back-end script) will send a “HttpPOST get” commend to front-end and grab the user input, process the information to target format(e.g. json.loads(), json.dumps() methods for string processing) and insert a new-user record to database.

For the chatbot part, the back-end service will deliver the information to Dialogflow and get the json information from it. The Dialogfow analyses information and return normal natural language in json. If the chatting information contains key words, the Dialogflow will give special json.

3.2.1 Dialogflow NLP Chatbot

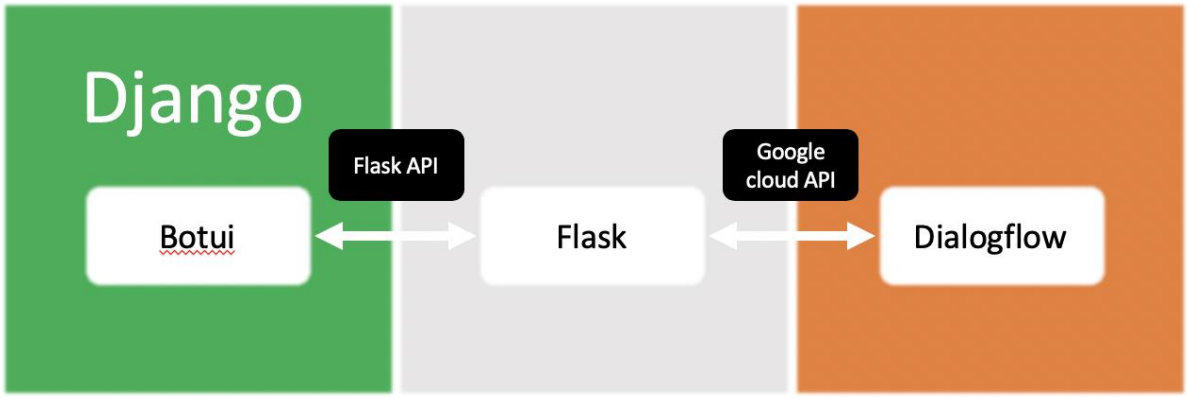


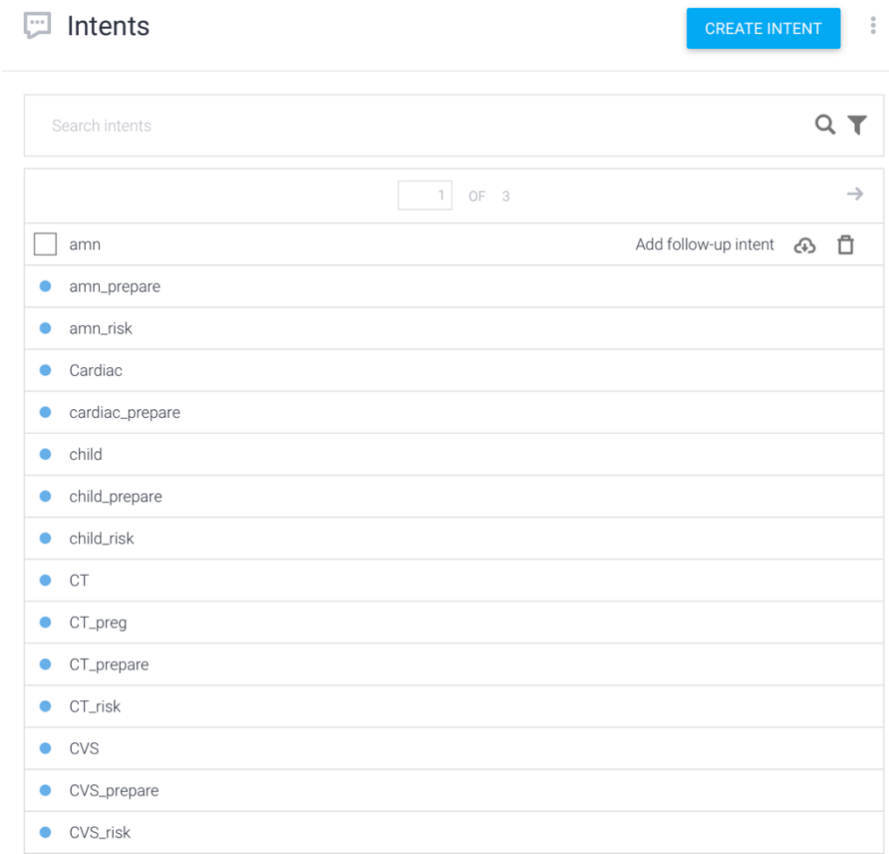
Figure 3 Chatbot architecture

Dialogflow (formerly Api.ai, SpeakToit) is a Google-owned developer of **human–computer interaction** technologies based on natural language conversations. Dialogflow is used to training the natural languages in the chatbot system to answer users' questions.
Using intents and entities to build modified training data.
Features of intents used in this system are:

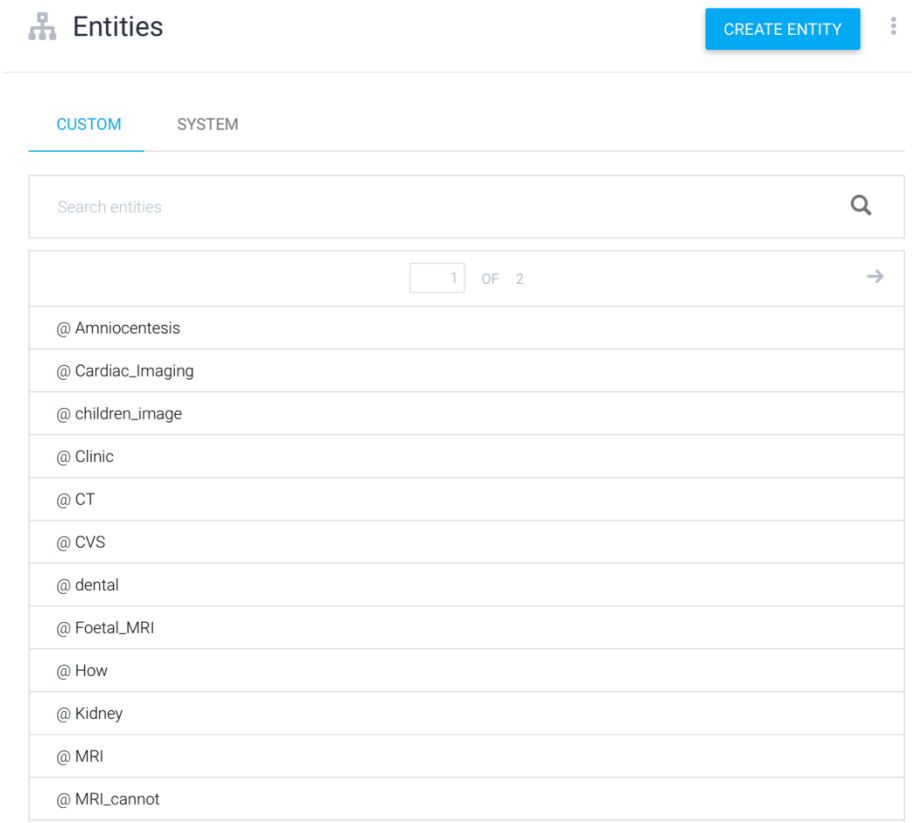
Training phrases: Create example of questions which contain key words in the sentences.
Parameters: As a single sentence might contain several key words. Only useful and important key words should be selected to define the actions.
Action: According to different selections of parameters, different actions are created to response to it.
Responses: Create unique responses to actions.

Entities are used to create keywords. For example, a 'clinic' entity contains 'hospital, 'clinic', 'medical center'. As a result, all words similar to clinic will be recognized as a clinic keyword.

Depending on the specific information, the back-end service will connect to MongoDB, and decide to add/delete/update data.



Entities are used to create keywords. For example, a ‘clinic’ entity contains ‘hospital, ‘clinic’, ‘medical center’. As a result, all words similar to clinic will be recognized as a clinic keyword.



3.3 Server

Databases and schemas are stored at server, and connected to Django project by “python3 manage.py migrate” (reflect as migration folder). Patient, clinic and appointment tables are stored in MySQL server; Keywords for chatbot are stored and structured in Google Dialogflow.

3.4 Project Highlights

This project is aimed to provide an online-booking system for patient users and corresponding booking management system for clinic users.

Instead of using the traditional “static pages” of most online-booking webpages: users fill in some kind of static form and then send this form to clinic users wait for approving or send to serve waiting for an auto-processing, this project utilize “On-page data query and presentation” for the booking parts: Instead of a “static” form or “static” page, all data from step2 (step1 is choose an examination type), are “dynamically” generated with the data queried from server and send from back-end using preceding user inputs without refreshing the webpage.

For example:

Figure 4 Example: MRI Clinic list

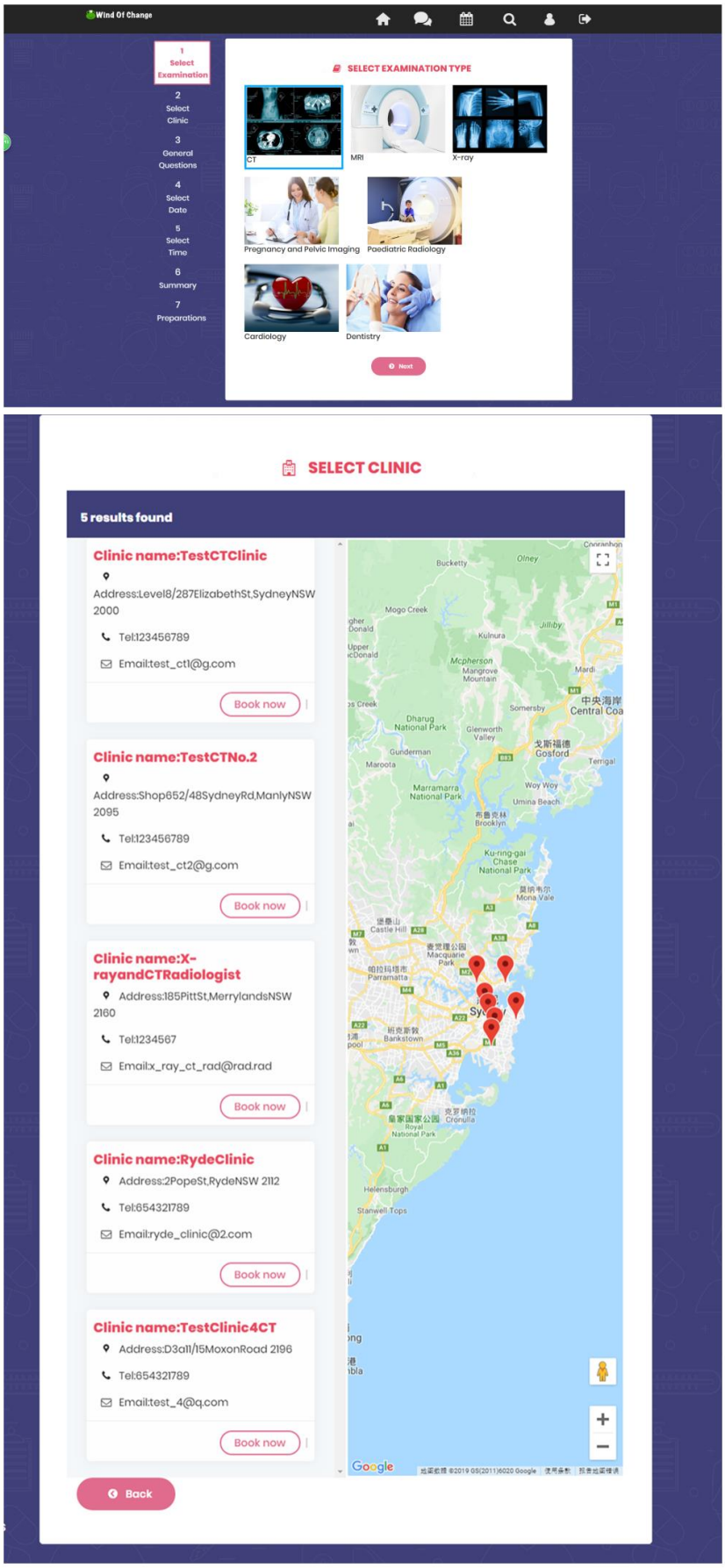


Figure 5 CT clinic list

Or selecting a different date for the same clinic will output different timeslot to choose:

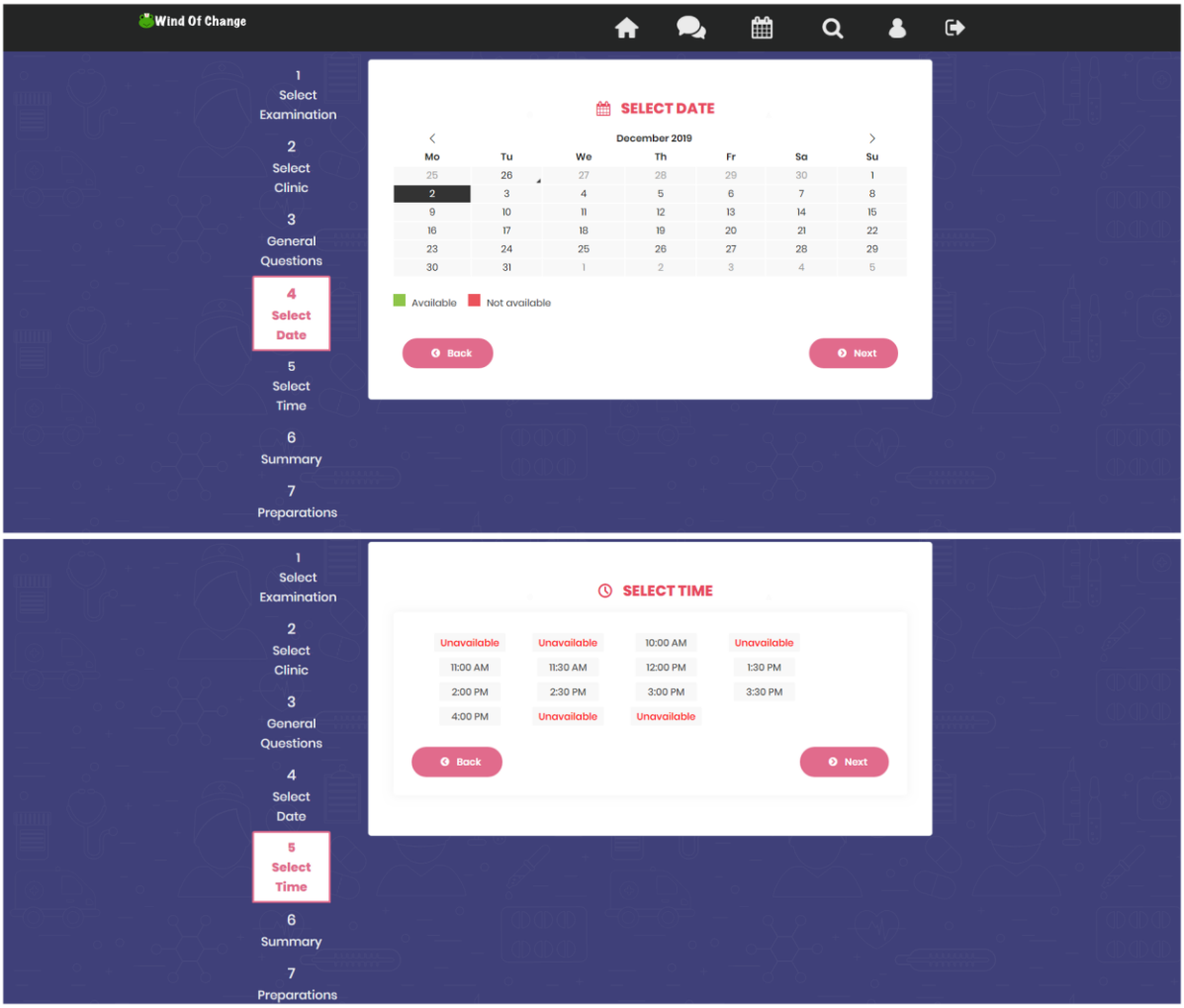


Figure 6 ClinicID 1 Dec-2nd Time-table

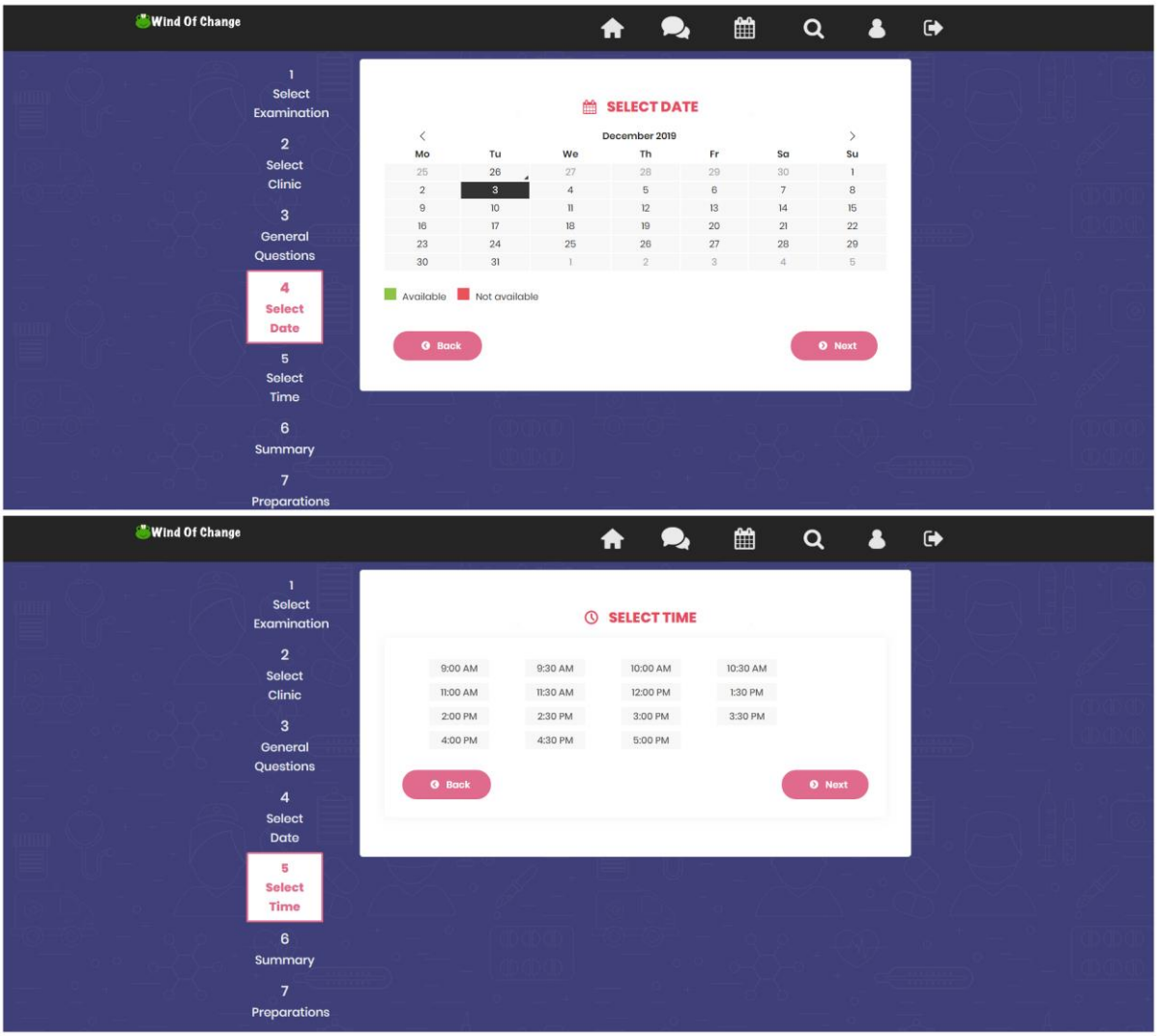
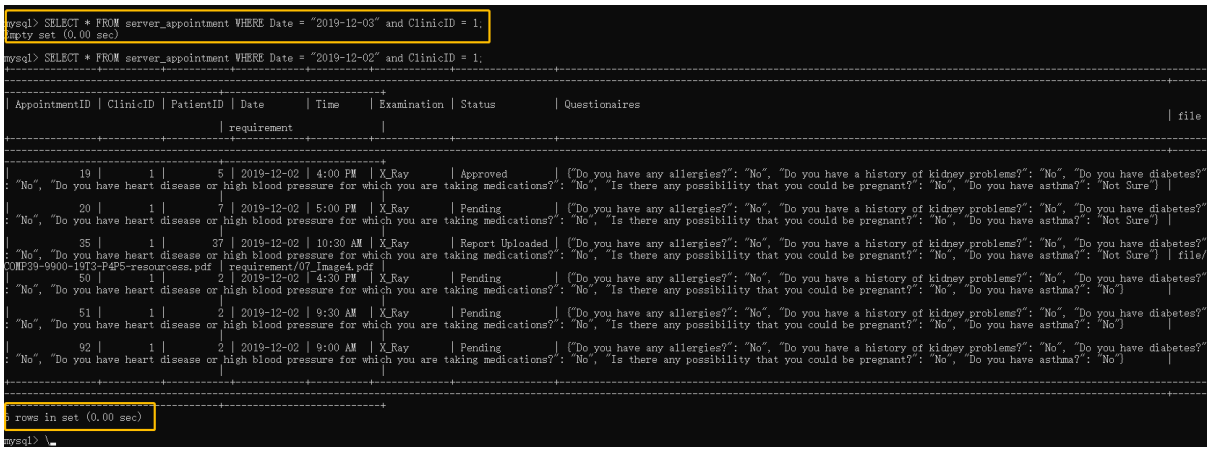


Figure 7 ClinicID 1 Dec-3rd Time-table



```
mysql> SELECT * FROM server_appointment WHERE Date = "2019-12-03" and ClinicID = 1;
Empty set (0.00 sec)

mysql> SELECT * FROM server_appointment WHERE Date = "2019-12-02" and ClinicID = 1;
+---------------+----------+-----------+------------+----------+-------------+----------------+-----------------------
```

with the suggested date, so that the next time this patient's booking will have a smaller chance to be cancelled again.

Aside from the patient's side, on the clinic side, in order to support some clinic that has very flexible schedule, this project also allows clinic users to customize their own questionnaires and timetable with different capacities on each timeslot, so that every piece of information patient displayed with, are from the real situation of a clinic, this also helps to minimize the chance a patient's order would be cancelled, because of the information are largely synchronized on both patient and clinic ends.

4. Project Detail Layouts

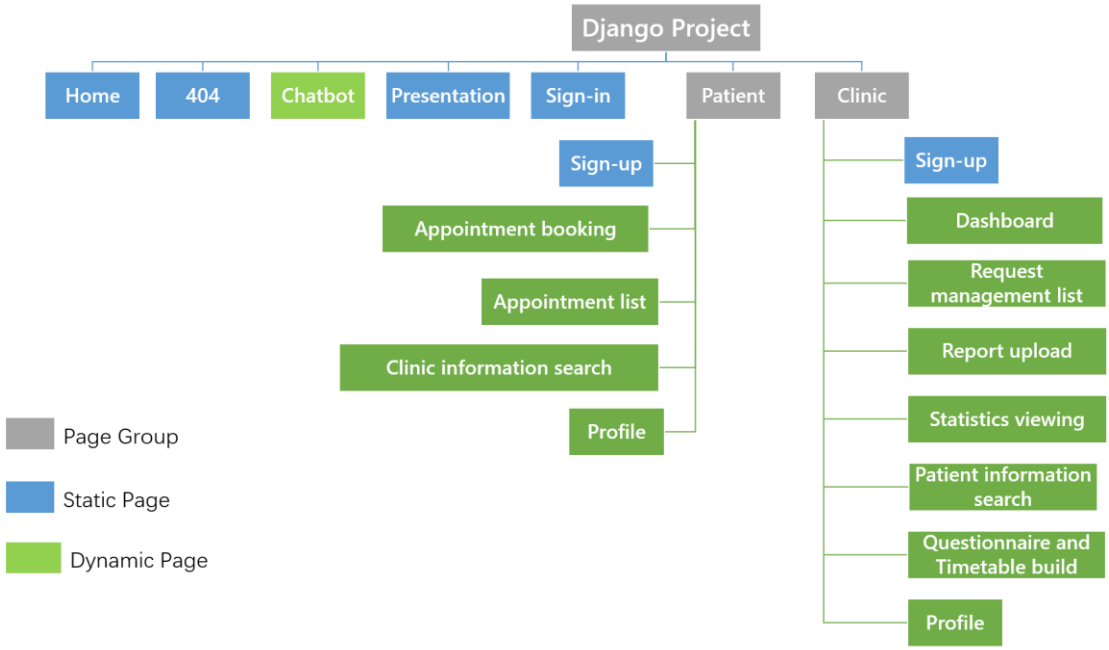


Figure 9 Webpages Structure

The overall webpages structure is shown as above, out of 18 webpages, this project contains 12 dynamic webpages and 6 static webpages. The next sections will focus on some detailed part in homepage, (Patient) appointment booking, information search and appointment listing pages, (Clinic) Dashboard, request list and report upload pages.

4.1 Assumptions

The featured functions were developed based on the assumptions:

1. Patients **already knew** which examination to take;

   Patients already went to General Patients and given a reference letter for medical imaging (Radiology examination) instructions.
2. Apart from 'online-questionnaire', patients still need to fill out other files if clinic adds **requirements.** (E.g. Some clinic specified questionnaires patients need to print out and sign);

3. The cancellation policy claims that patients should **inform clinic** if they need to cancel an approved examination;

   Avoid wrong-click and schedule mass-up caused from patient side (since receptionists are trained to update status and coordinate schedule changes).

4.2 Homepage

The home page consists by: About us section, skill utilization section, system featured functions section and examination specialization overview section (Refer to appendix for full screenshot of webpages).

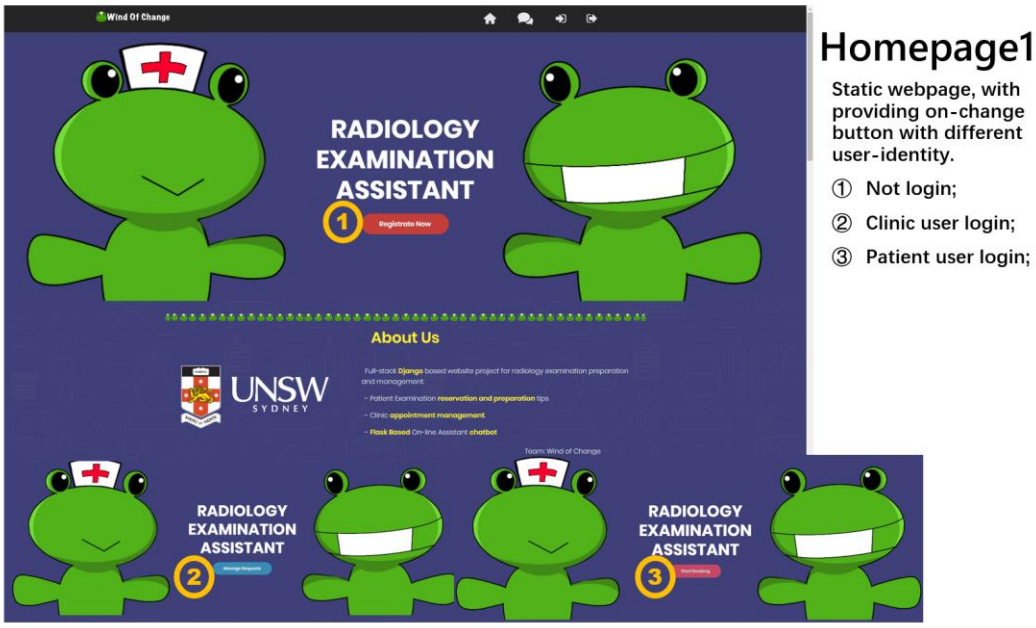


Figure 10 Homepage detail

As shown in the figure above, the short-cut button on the homepage, changes with different identity (visitor, patient, clinic) was designed to let users directly go to the main functions of website.

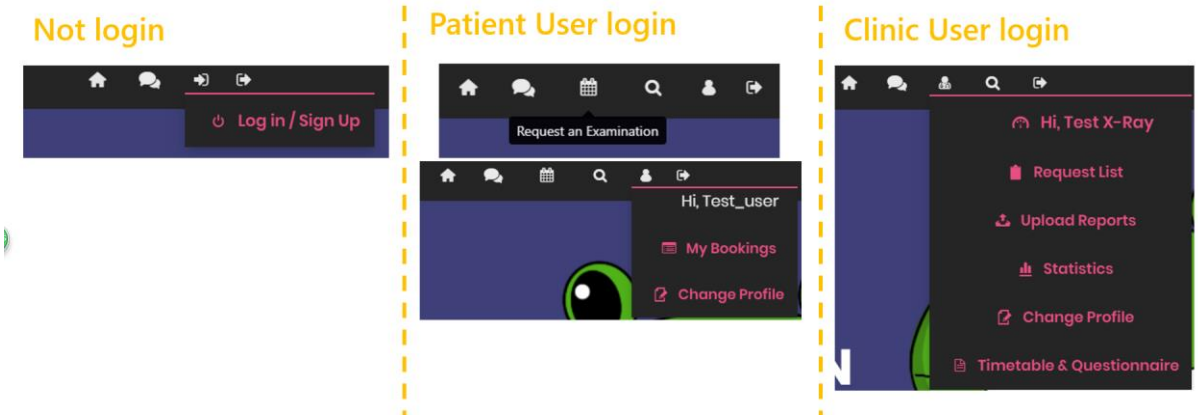


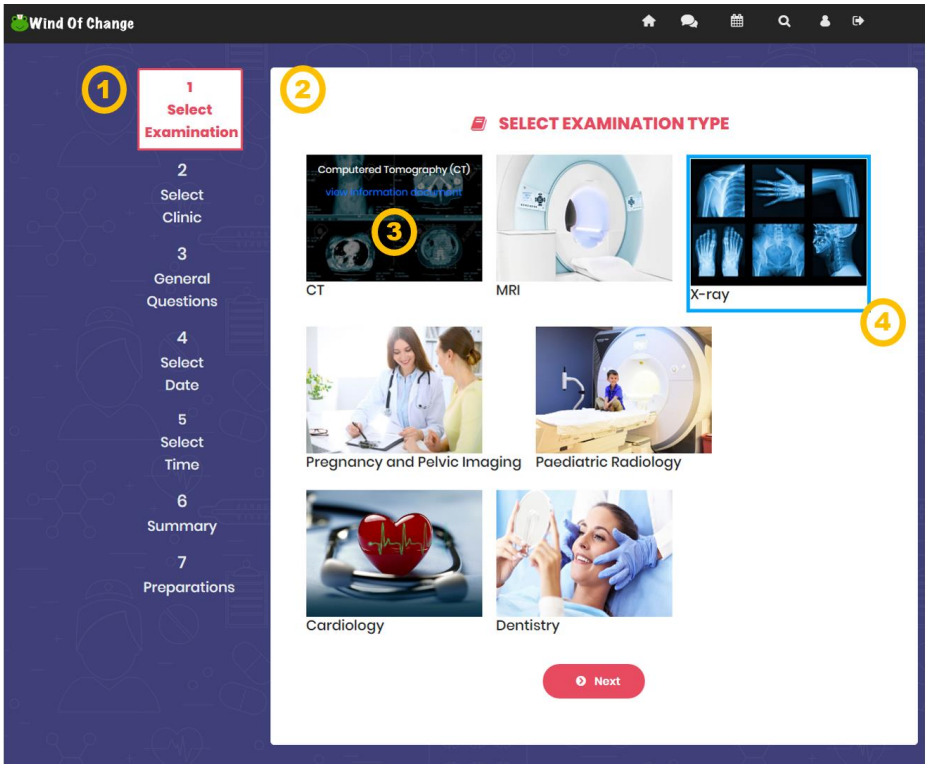
Figure 11 Menu Detail

Not only the Homepage, the menu also shows different navigation items with different login identities.

4.3 Patient pages

Patient users are provided with two main functions: appointment booking and appointment information checking.

4.3.1 Appointment booking page



Appointment Page

① Steps indicator bar;

② Content section;

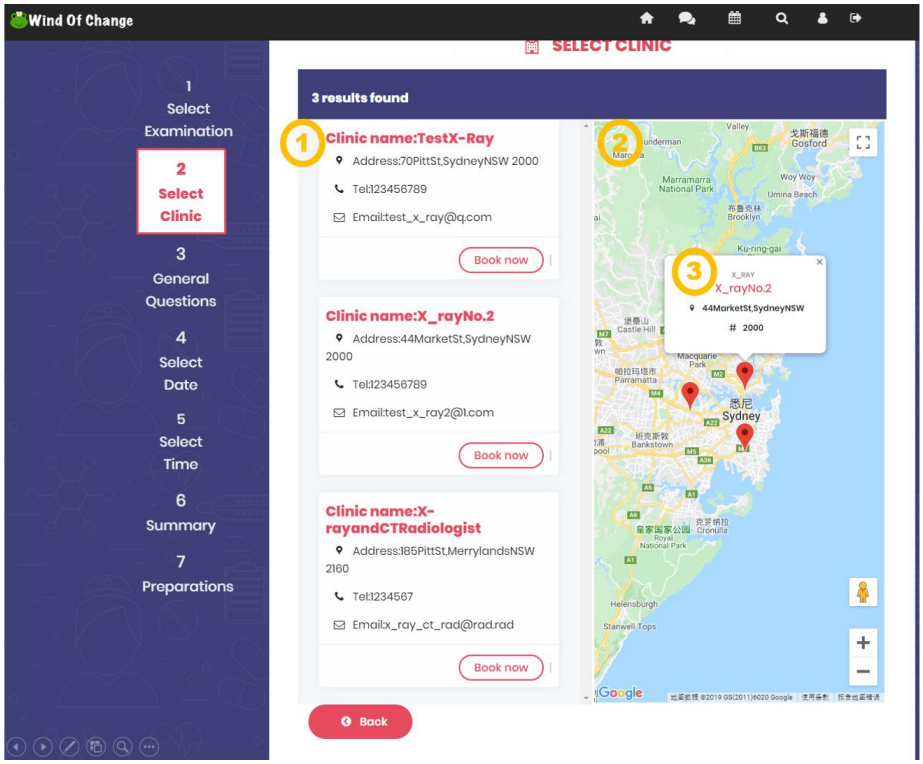
③ Hover explanations and information link;

④ Selection highlight;

Figure 12 Patient Appointment Booking Step 1

In the appointment booking page, patient users are provided with a step-wise indication bar freeze on the left, since some tabs may need more than one-screen height, if the indication bar was freeze at the top, some information might be covered;

The main feature of this webpage is that: data are dynamically queried at back-end according to users' input in each step without refreshing the whole webpage (using Ajax). The reason for making webpage not refreshing between steps is to continuously transferring data between tabs(steps) avoids repeating sending same data (e.g. clinic name, address was grabbed in step 2, but needs to be displayed in step 6) and make same query to database when the same piece of information is surely needed in following steps.



Appointment Page-2

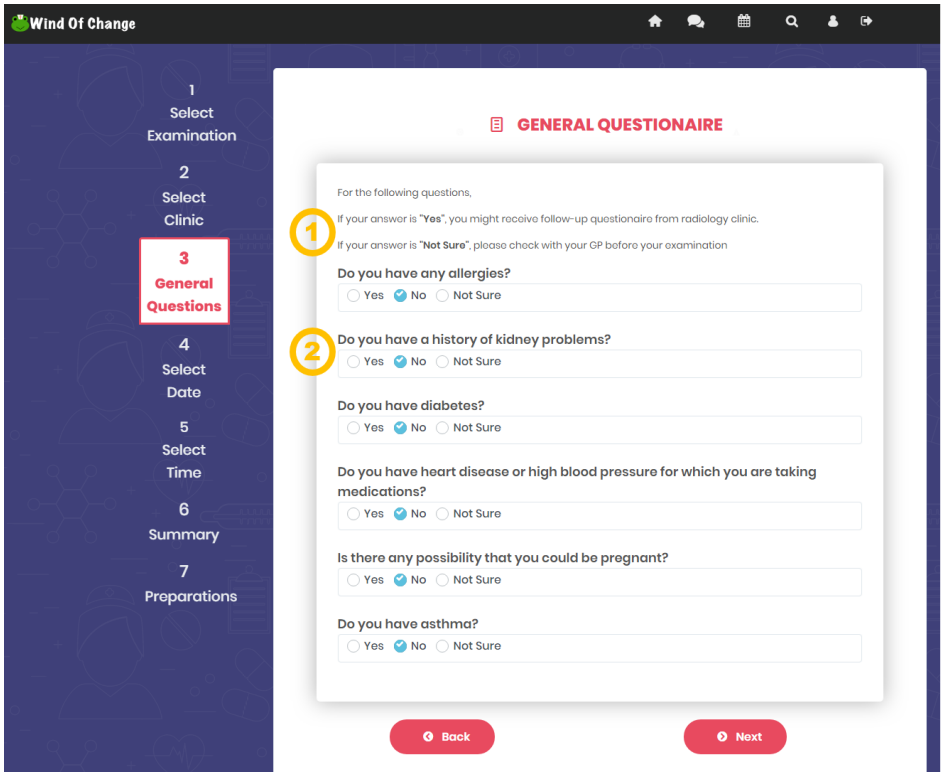
① Clinic list for selected examination;

② Google map location indication;

③ Clinic detail shown on Google map;

Figure 13 Patient Appointment Booking Step 2

In step-2, with the selected examination users selected in step-1, a list of clinics which offers such examination are shown with their detail information, as well as locations indicated in Google Map, which provide users a direct view of where the clinic is , and which clinic is in his/her preferred area(from map);



Appointment Page-3

① Instructions for questionnaire;

② Options for questionnaire;

Figure 14 Patient Appointment Booking Step 3

In step-3 patients are asked to fill a clinic-built questionnaire for some multiple-choice questions. This function was designed to act as a reminder that gives patients some ideas on which situation they need to avoid for selected examination, as well as, gives the clinic user an overview to decide if they need to ask the patient to provide more information about his/her medical history.

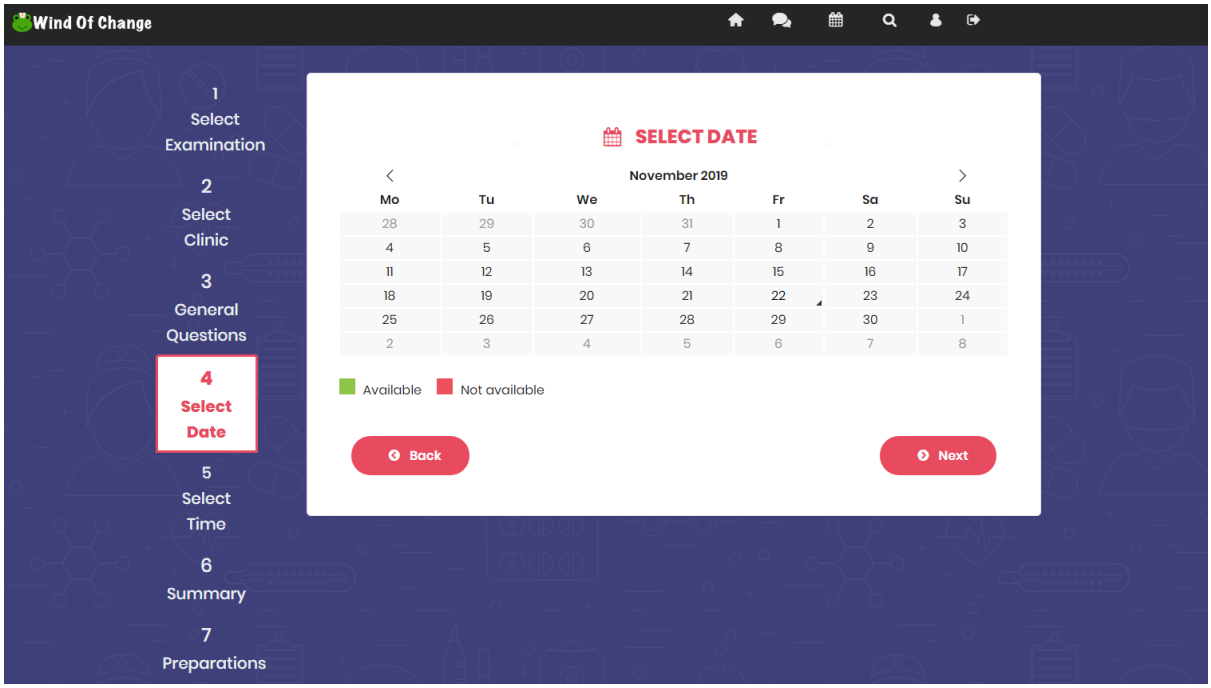


Figure 15 Patient Appointment Booking Step 4

In step-4, after submitting the questionnaire, a calendar will be provided to patient to select their preferred dates for their examinations. Error insertion of selecting a past-date and a non-working date are shown below:

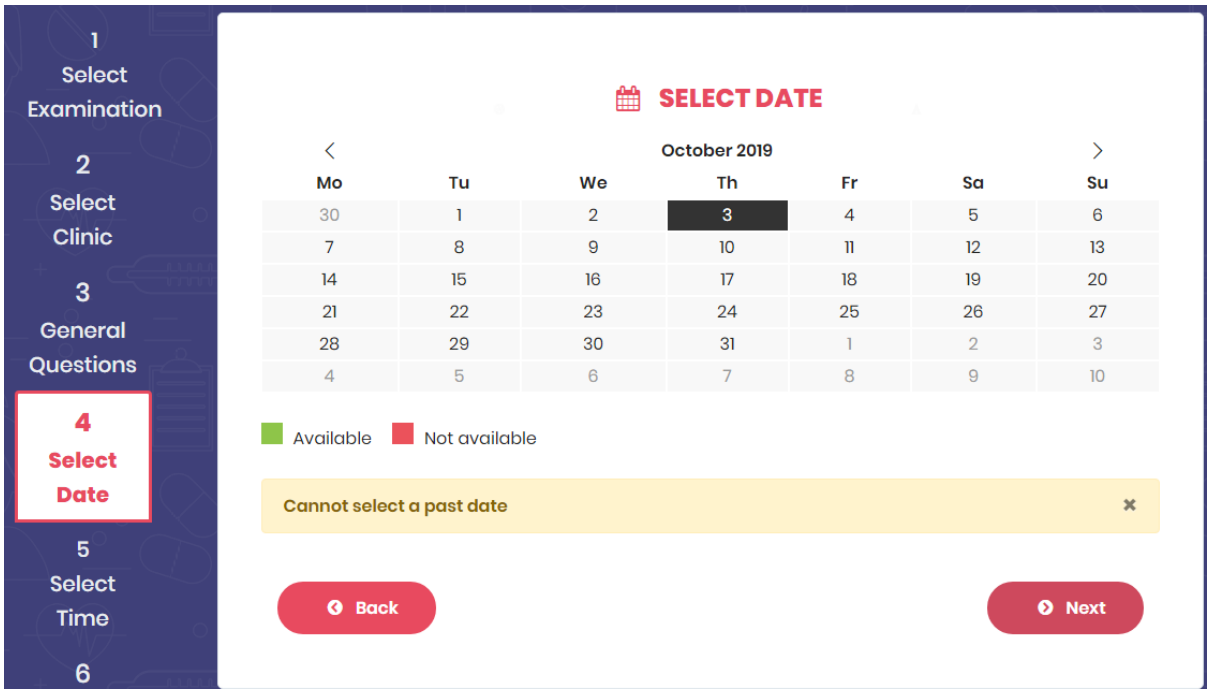


Figure 16 Patient Appointment Booking Step 4 Error insertion 1

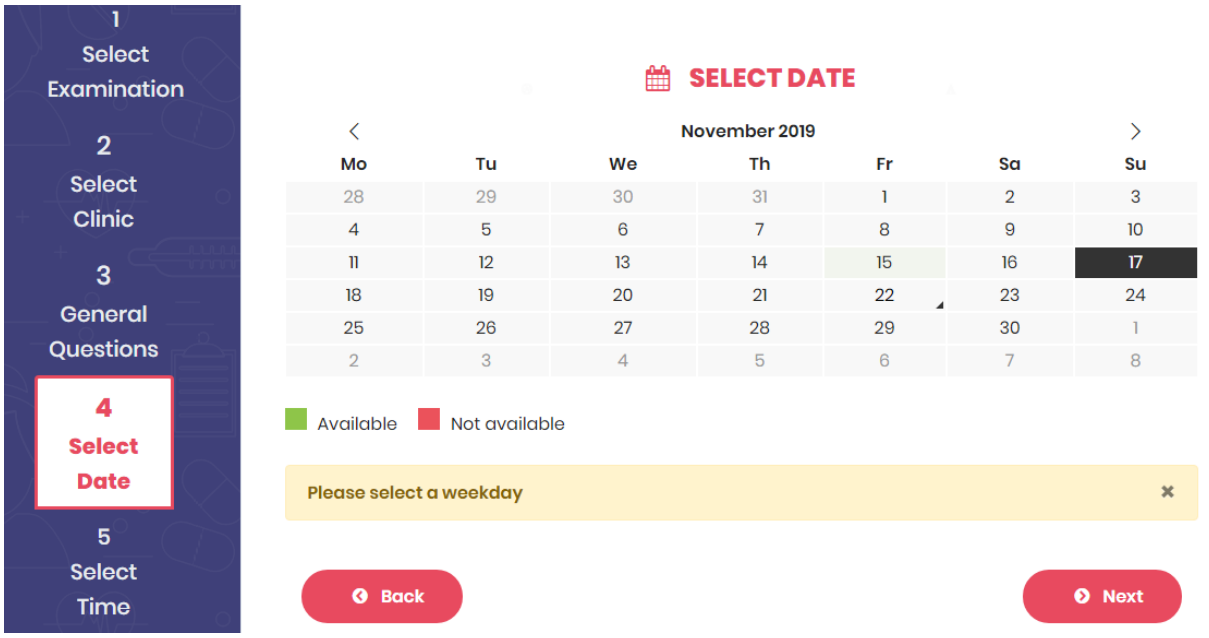


Figure 17 Patient Appointment Booking Step 4 Error Insertion 2

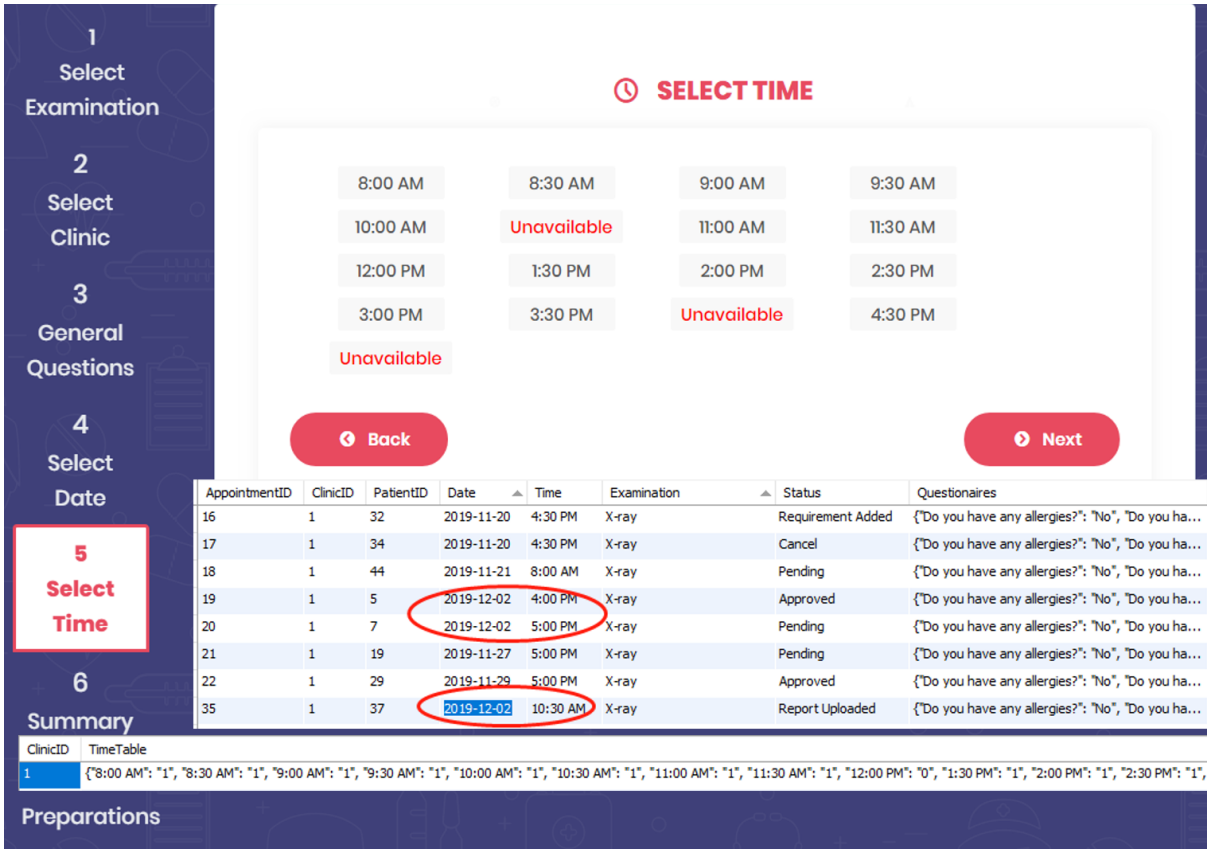


Figure 18 Patient Appointment Booking Step 5

In step-5, a time-table will be provided to patients, dynamically query the database for the time table of selected clinic on selected date and display to patients in this tab. As shown in the figure above, the selected date was “2019-12-02”, there are 3 appointments existed (No matter what status, since if the

clinic cancels someone's appointment on this date this time, that means the clinic is not available at that moment), and each timeslot, the capacity was 1, as a result, 4.00 PM, 5.00 PM and 10.30PM are "Unavailable" to book.

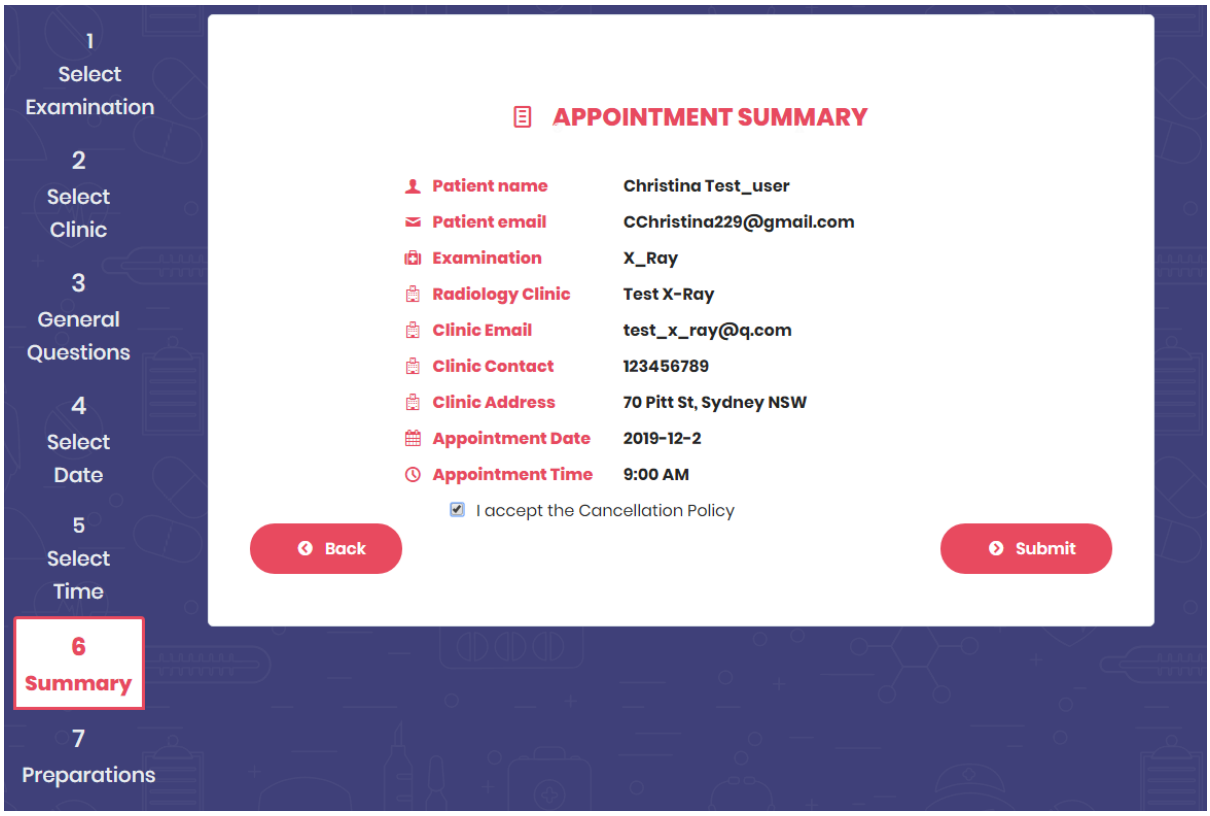


Figure 19 Patient Appointment Booking Step 6

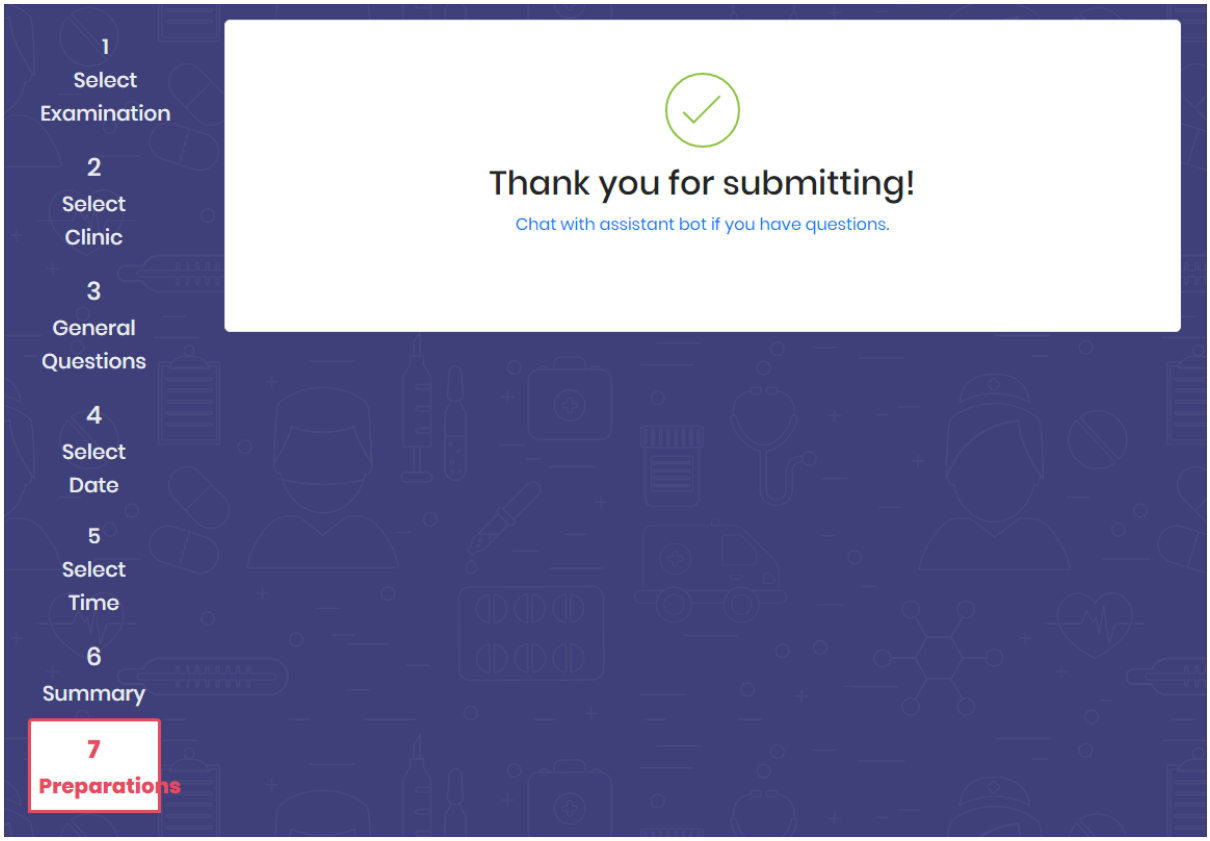
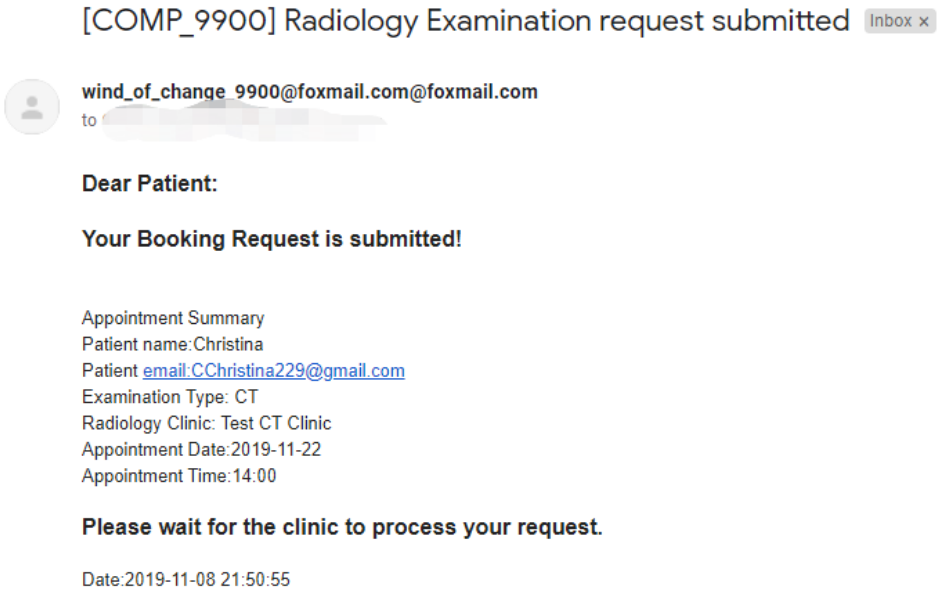


Figure 20 Patient Appointment Booking Step 7

At step-6, a summary of the appointment will be provided. After patient confirm and check the cancellation policy checkbox, a conformation page will be present, also, at the same time a record will be insert into the appointment table on the server, an email will be sent to the patient, the email sample below is an email received by patient.



[COMP_9900] Radiology Examination request submitted Inbox ×

wind_of_change_9900@foxmail.com@foxmail.com
to

Dear Patient:

Your Booking Request is submitted!

Appointment Summary
Patient name:Christina
Patient email:CChristina229@gmail.com
Examination Type: CT
Radiology Clinic: Test CT Clinic
Appointment Date:2019-11-22
Appointment Time:14:00

Please wait for the clinic to process your request.

Date:2019-11-08 21:50:55

Figure 21 Patient Appointment Booking email example

4.3.2 Appointment information list

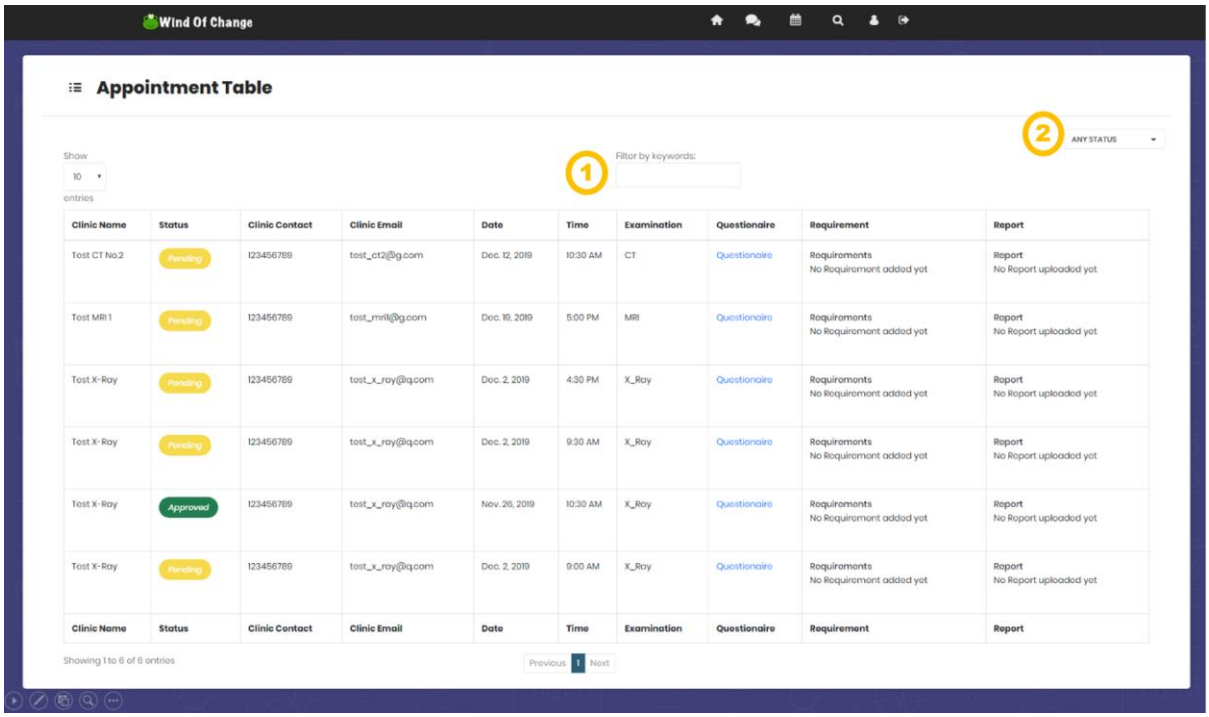
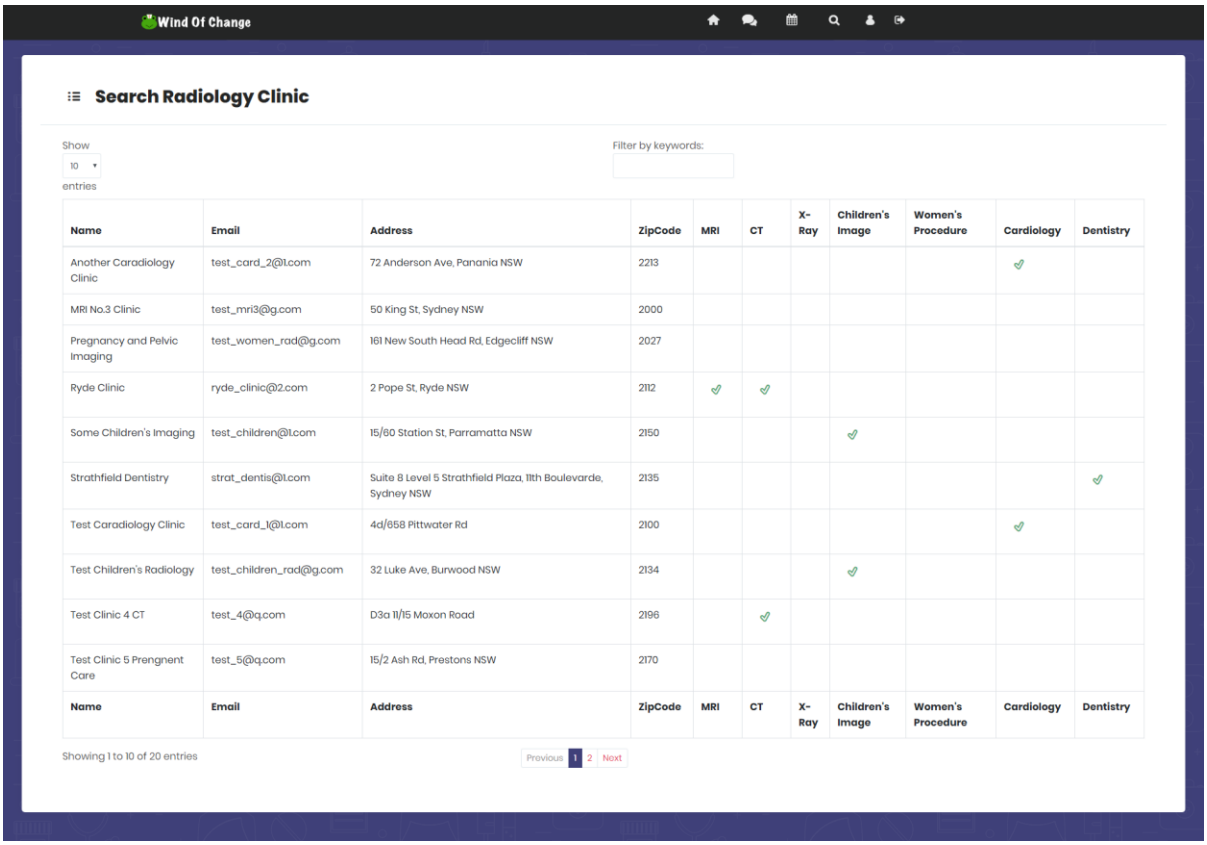


Figure 22 Patient Booking History

A patient will be able to check his/her appointment history table (like order history in shopping websites), patient can filter the table by selecting a specific status (marked as ② in the figure above) or filter by entering a keyword(marked as ①), such as: CT, a Clinic Name, or a date.

4.3.3 Clinic information search



Name	Email	Address	ZipCode	MRI	CT	X-Ray	Children's Image	Women's Procedure	Cardiology	Dentistry
Another Caradiology Clinic	test_card_2@1.com	72 Anderson Ave, Panania NSW	2213						✓	
MRI No.3 Clinic	test_mri3@g.com	50 King St, Sydney NSW	2000							
Pregnancy and Pelvic Imaging	test_women_rad@g.com	161 New South Head Rd, Edgecliff NSW	2027							
Ryde Clinic	ryde_clinic@2.com	2 Pope St, Ryde NSW	2112	✓	✓					
Some Children's Imaging	test_children@1.com	15/60 Station St, Parramatta NSW	2150				✓			
Strathfield Dentistry	strat_dentis@1.com	Suite 8 Level 5 Strathfield Plaza, 11th Boulevarde, Sydney NSW	2135							✓
Test Caradiology Clinic	test_card_1@1.com	4d/658 Pittwater Rd	2100						✓	
Test Children's Radiology	test_children_rad@g.com	32 Luke Ave, Burwood NSW	2134				✓			
Test Clinic 4 CT	test_4@q.com	D3a 11/15 Moxon Road	2196		✓					
Test Clinic 5 Prengnent Care	test_5@q.com	15/2 Ash Rd, Prestons NSW	2170							
Name	Email	Address	ZipCode	MRI	CT	X-Ray	Children's Image	Women's Procedure	Cardiology	Dentistry

Figure 23 Patient Clinic information look up

In addition, patients are also provided with a table-formatted information “database” about all radiology clinics in the server database. This function was design to provide patient a “look-up” function without go to the appointment function if they just want to look up some piece of information about some specific examination, or location.

4.4 Clinic pages

The webpages for clinic users are designed to provide:

1. Approve, cancel or add requirement files (can be another specific questionnaire from the clinic with the clinic logo and require patient to bring to the clinic on their examination day);

2. Upload report for a completed examination;

3. Look up patients’ information;

4. Questionnaire and timetable building function for appointment booking system;

5. Simple statistical summary of appointments versus timeslots, date and appointment status in line chart, bar chart and pie chart;

4.4.1 Clinic Dashboard

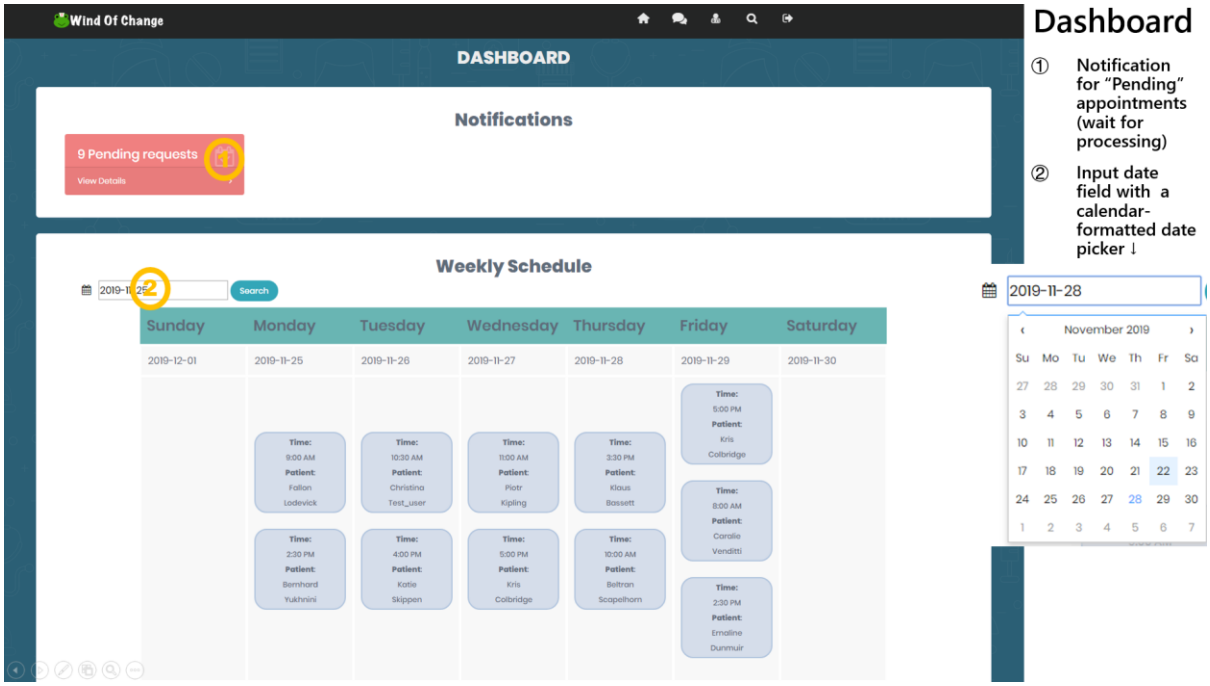
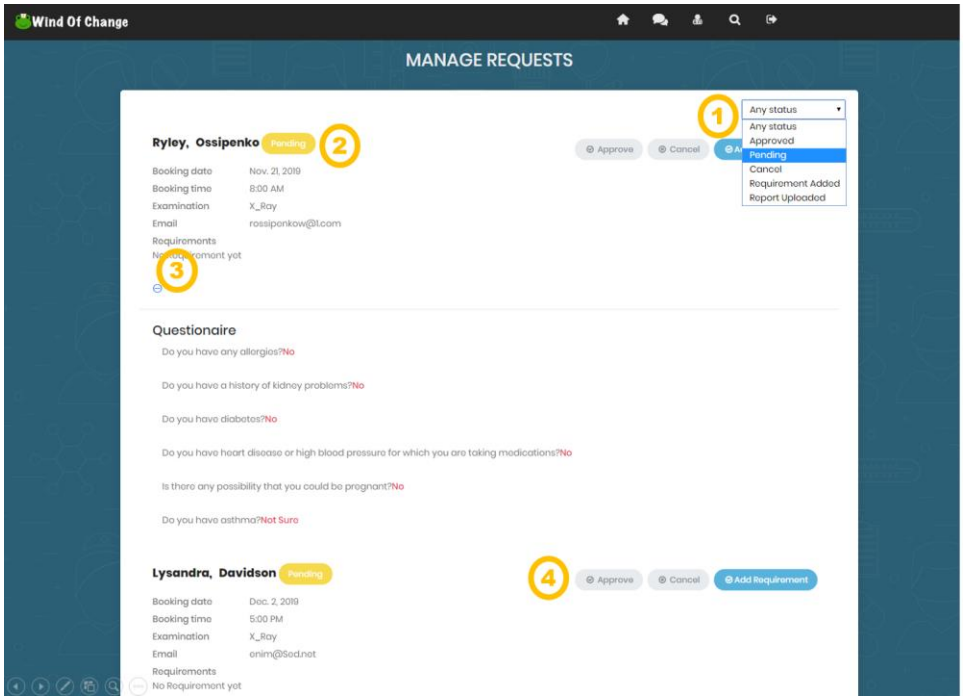


Figure 24 Clinic Dashboard

Clinic users are provided a table-formatted weekly schedule overview for "Approved" and "Requirement Added" appointments. For any input date, the weekly schedule for that date will be shown. A notification for "Pending" status appointment will be shown to notify clinic users and have them process pending requests as soon as possible.

4.4.2 Request List



Request List

① Status filtering

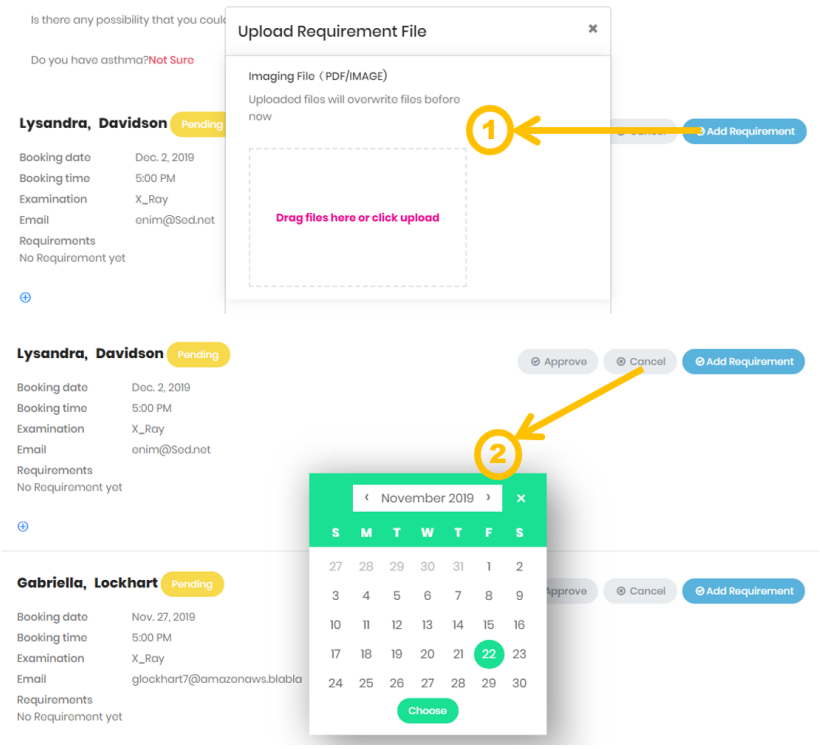
② Color-indication tag of appointment status

③ Show/hide questionnaire details

④ Buttons for clinic user's decision on appointment

Figure 25 Clinic Request List

At the request list page, clinic users will be able to filter all appointment records by status, access details of the appointment requests, look at the on-line questionnaire details from the patient's booking, and buttons for approve, cancel or add additional requirement files for the patient.



Request List-2

① Requirement upload window after clicking "Add Requirement"

② Calendar date picker for clinic user to given a suggestion date if "Cancel" button was clicked

③ Any change in status will result an email send to patient of this appointment

Figure 26 Clinic Request List details

For the appointment cancellation function, it is designed to let clinic users suggest another available date for the patient. An email will be sent to patient for any status change, as well as showing the patient clinic suggested date for a cancelled appointment.

4.4.3 Upload Report

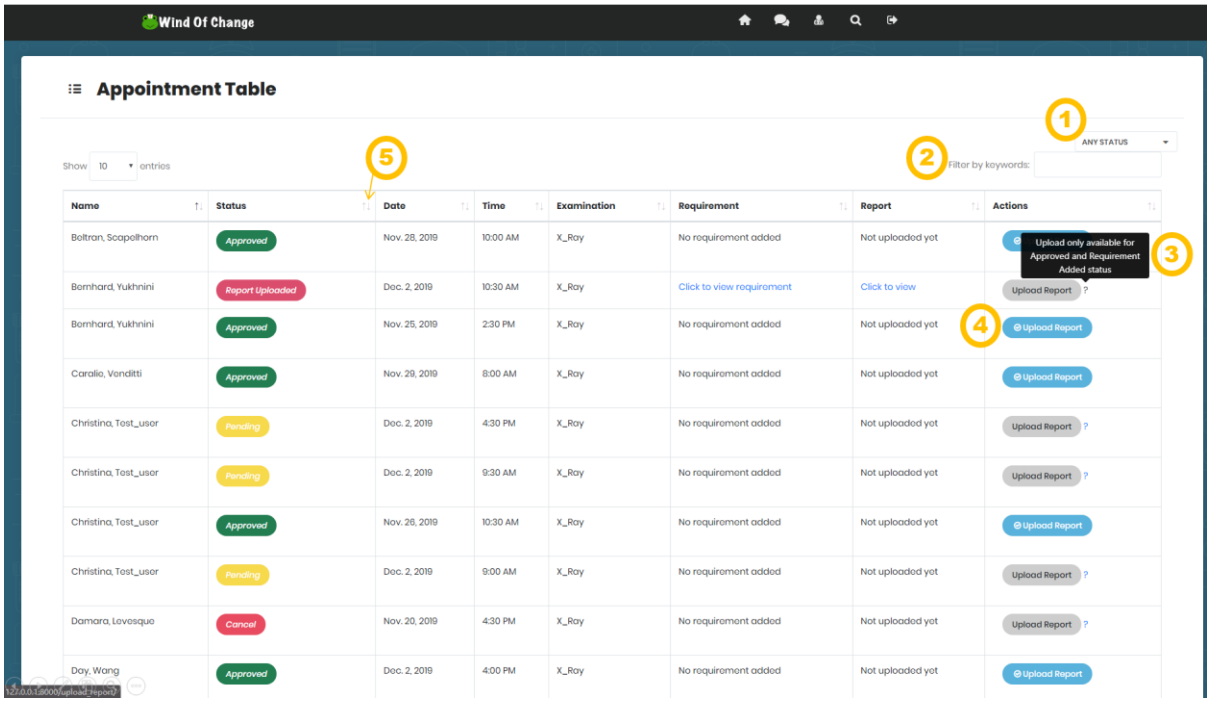


Figure 27 Clinic Upload Report

Different from request list, the upload report webpage provides a table-formatted view of all appointment for clinic users, with report upload function. The reason to design separate pages for report uploading and request processing is: 1. Avoid wrong clinic, since both pages provides file uploading function, to avoid miss-upload, the two functions are located separately and upload report function is only open for an approved or requirement added appointments (it is unlikely for a cancelled or pending appointment to have an report). 2. Request list webpage is for actions BEFORE patients' examinations, but upload report webpage is for actions AFTER patients' examinations, most clinics have more than one receptionist, and they can action on different pages at the same time for pre-appointment processing and post-appointment file transferring rather than keep refreshing each other's webpages.

4.4.4 Statistics

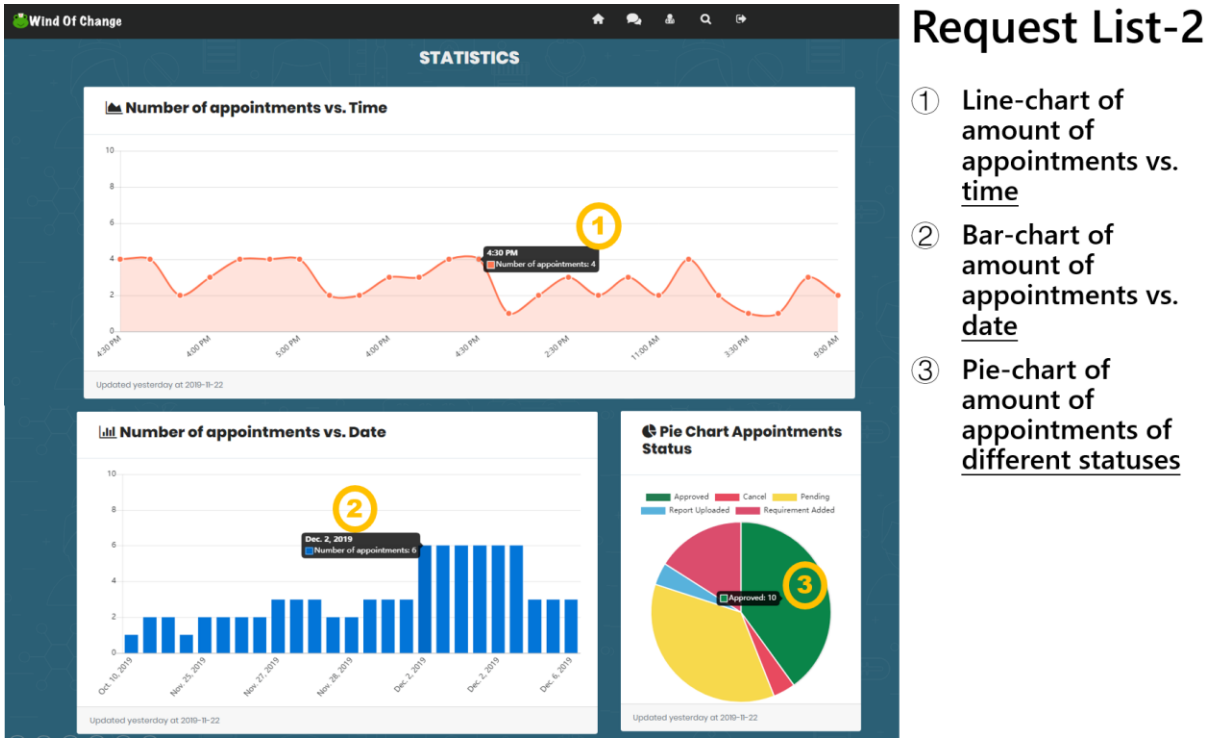


Figure 28 Clinic Statistics

Clinic users are able to view their statistical report via different charts on this webpage, these charts encourage clinic users to change their strategy if extreme situations happen. For example, if there are significantly more appointments in the morning than in the afternoon (including all status of appointments), the clinic might consider to have more capacities for morning sessions; if there are a remarkably more of “Approved” appointments than “Report Uploaded” appointments for a long time, the clinic user might consider to get their report output faster.

4.4.5 Profile Email and Password update (both Patient and Clinic users)

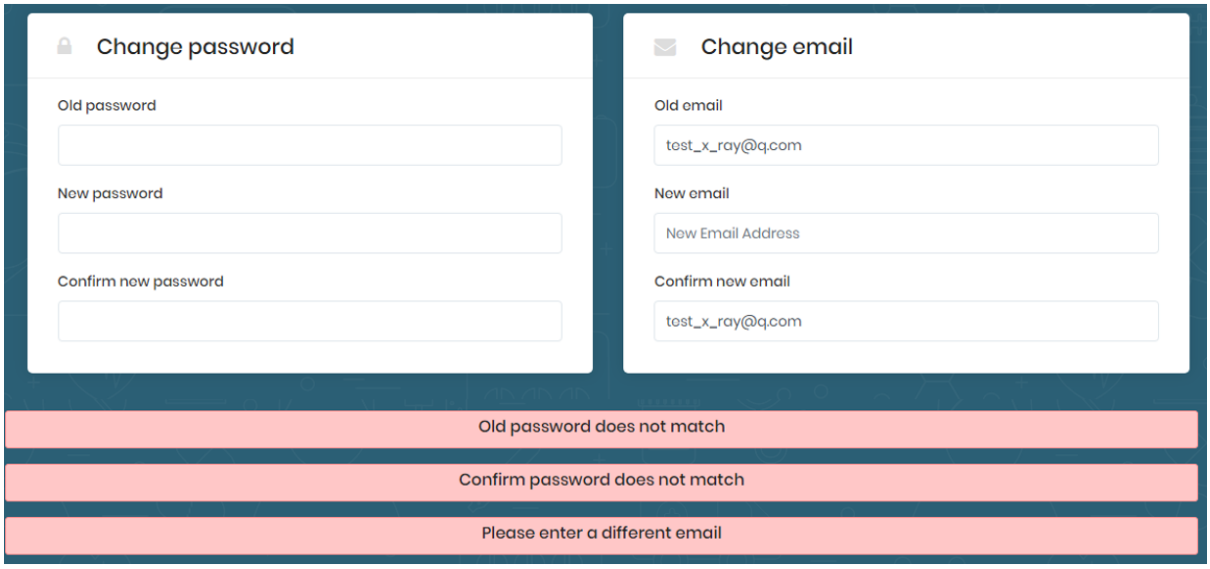


Figure 29 Change password and change email error messages

For both patient and clinic users' profile webpage, change password and change email functions are designed not only for gathering user's input data but also check users' inputs when data are sent to the backend. If the backend functions detected that there is "Invalid" input, a corresponding error message will be sent to the front-end.

4.4.6 Questionnaire and Timetable Build

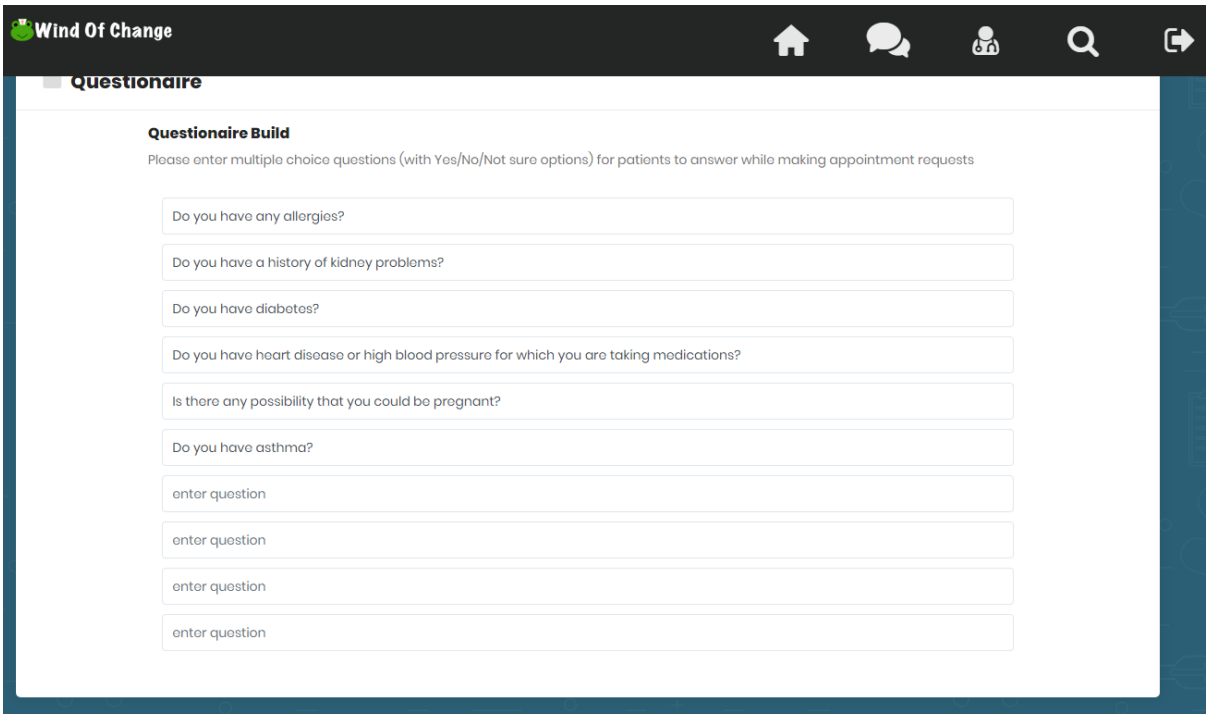


Figure 30 Questionnaire Build

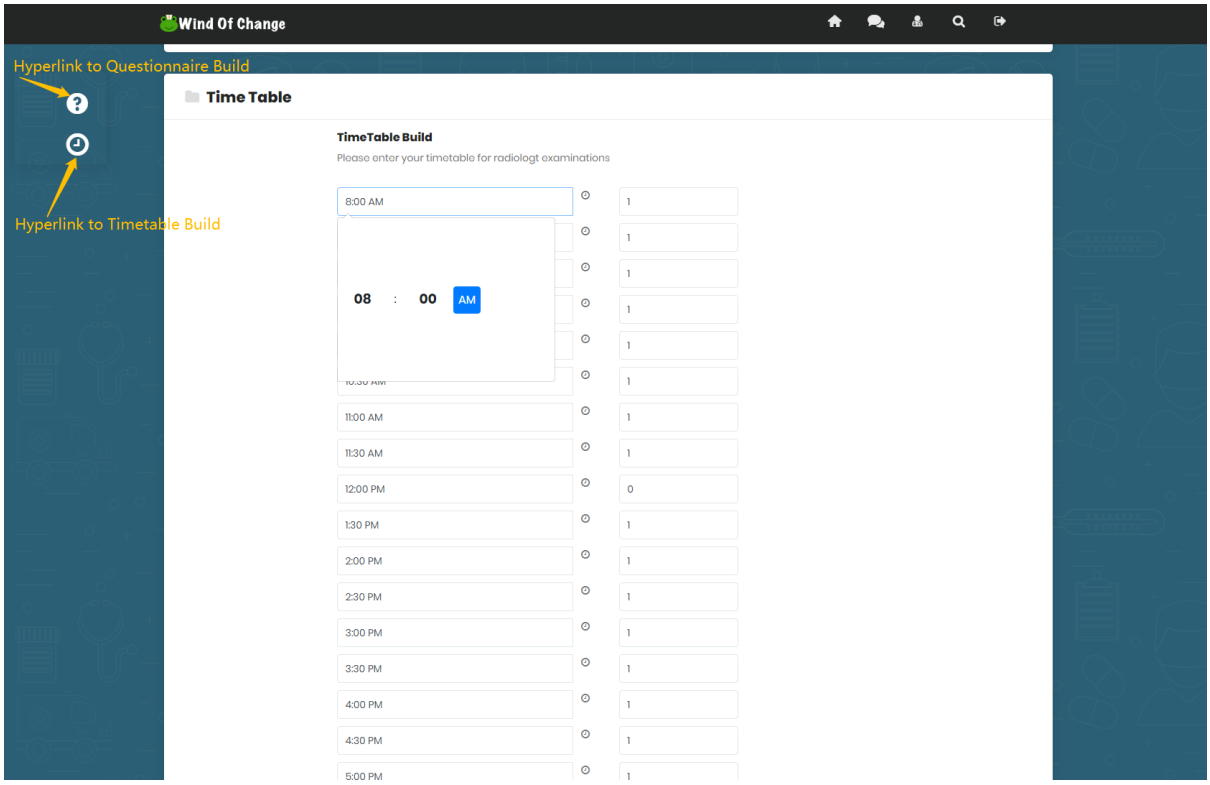


Figure 31 Timetable Build

```
mysql> SELECT TimeTable FROM server_clinic WHERE ClinicID = 1;
+-----------------------
```

For the reason that different examinations may require different questions on the questionnaire step in appointment booking and different clinics do have different schedules, Timetable and Questionnaire build page allows clinic users to build their own Yes/No questions showing on patient's appointment booking page, and build their own timetable with time slots and capacities.

For example, suppose this clinic user is from an ultra-sound specialized clinic which female patient might come and check if they are pregnant, or do some periodic pregnancy checks, thus this clinic user need to remove the question "Is there any probability that you are pregnant?", what he/she needs to do is just simply remove the question as shown below:

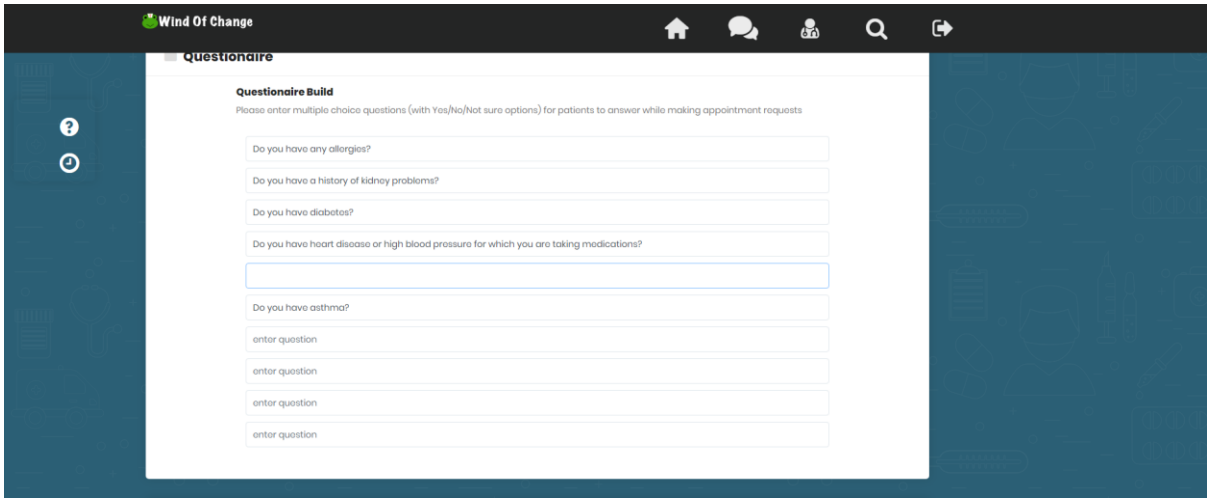


Figure 33 Change questions

Besides changing the questions, this clinic user also found that they have not a lot of patients early the day, but more patients at afternoon, so the clinic user wants to add more capacity to the timeslots in the after noon and delay the opening time until 9 am.

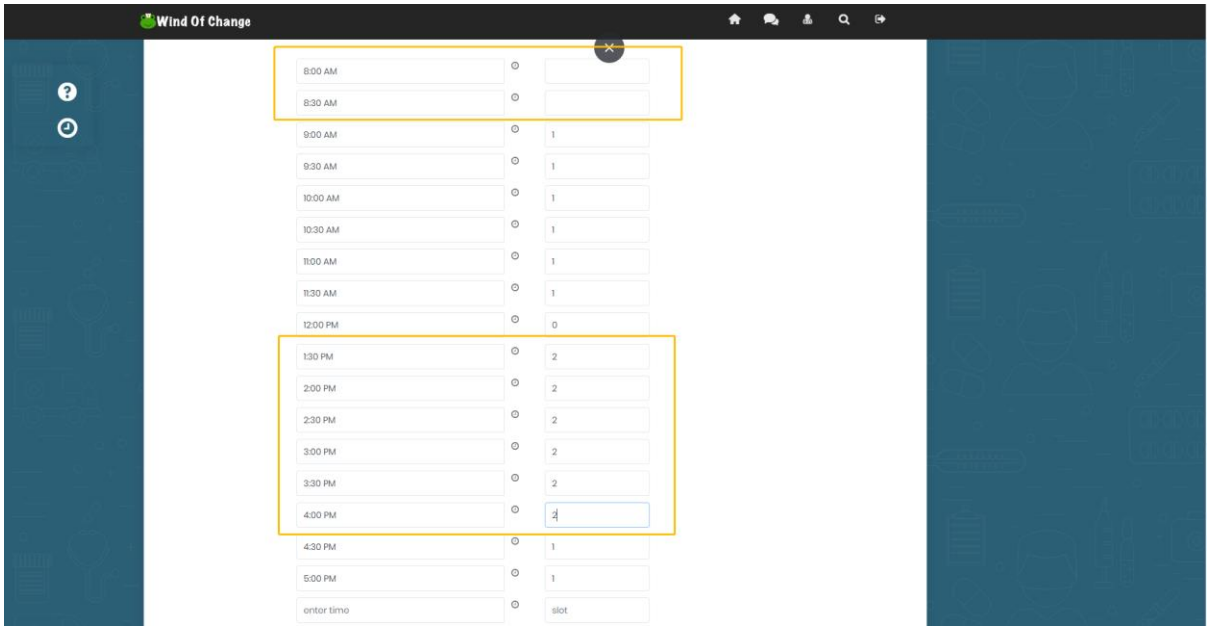


Figure 34 Change timeslots and capacities

After click “Save”, let us check the database again:

```
mysql> SELECT TimeTable FROM server_clinic WHERE ClinicID = 1;
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------+
| TimeTable                                                                                                |
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------+
| {"8:00 AM": "", "8:30 AM": "", "9:00 AM": "1", "9:30 AM": "1", "10:00 AM": "1", "10:30 AM": "1", "1
1:00 AM": "1", "11:30 AM": "1", "12:00 PM": "0", "1:30 PM": "2", "2:00 PM": "2", "2:30 PM": "2", "3:0
0 PM": "2", "3:30 PM": "2", "4:00 PM": "2", "4:30 PM": "1", "5:00 PM": "1"} |
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT Questiones FROM server_clinic WHERE ClinicID = 1;
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
--------------+
| Questiones                                                                                               |
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
--------------+
| ["Do you have any allergies?", "Do you have a history of kidney problems?", "Do you have diabetes?"
, "Do you have heart disease or high blood pressure for which you are taking medications?", "Do you h
ave asthma?"] |
+-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
--------------+
1 row in set (0.00 sec)

mysql>
```

Figure 35 Check database updates

All changes are updated to database, now when patients go and book this clinic’s examinations, they’ll see the updated questionnaires and timetables.

5. Example codes

In this section, some key programming code will be displayed and explained. However, before shown the detailed code, it is necessary to introduce Python Django framework since it is the main framework in this project, and Django does have a lot of different design from the traditional website frameworks and many unique patterns as website framework.

5.1 Django Project Structure

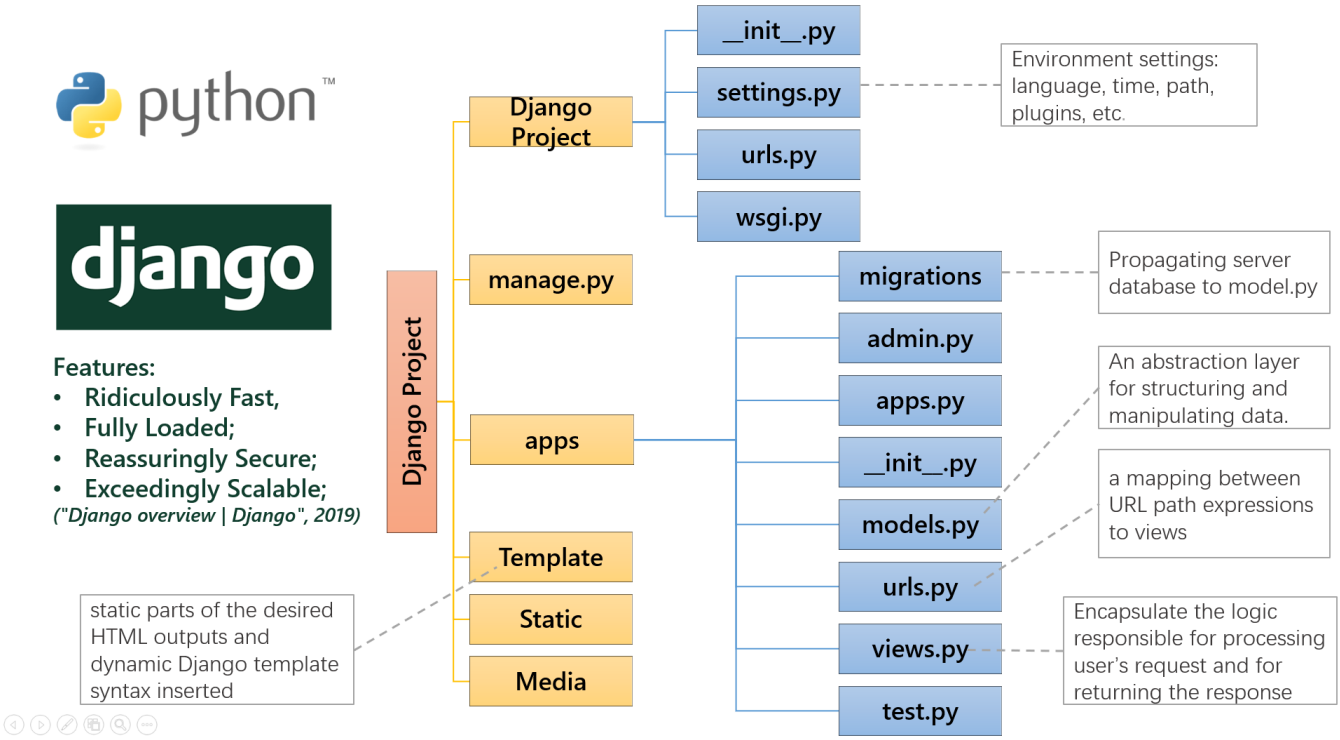
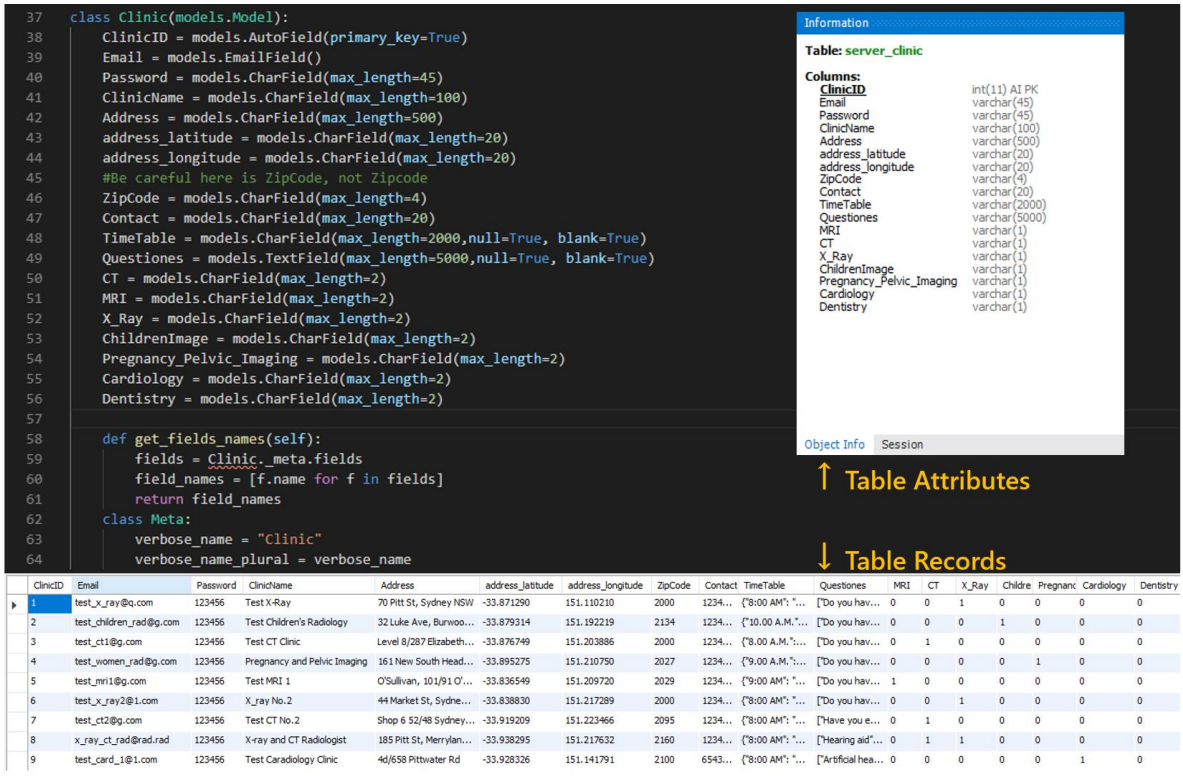


Figure 36 Python Django Project Structure

First of all, Django have its own procedure of starting a Django project. Once a Django project is created, programmers have to follow Django's structure to put code inside this project. A very unique feature about Django is, unlike other frameworks, Django is a "Model – View – Controller (MVC) framework", the model is the models.py file which contains a reference schema of projects database. Although the recommend database of a Django project is SQLite, but basically Django can adopt any database and provide users Python data-structure processing instead of a database language. The view.py is responsible for database query to the server and sending or requiring http response to the front-end, the query is based on the models built in the model file, all information(data) are transferred in the form of URLs. Url.py file contains all the url paths (e.g. "128.108.0.1/home") with corresponding functions or view classes in views file.

5.2 Models




```
class Clinic(models.Model):
    ClinicID = models.AutoField(primary_key=True)
    Email = models.EmailField()
    Password = models.CharField(max_length=45)
    ClinicName = models.CharField(max_length=100)
    Address = models.CharField(max_length=500)
    address_latitude = models.CharField(max_length=20)
    address_longitude = models.CharField(max_length=20)
    #Be careful here is ZipCode, not Zipcode
    ZipCode = models.CharField(max_length=4)
    Contact = models.CharField(max_length=20)
    TimeTable = models.CharField(max_length=2000,null=True, blank=True)
    Questiones = models.TextField(max_length=5000,null=True, blank=True)
    CT = models.CharField(max_length=2)
    MRI = models.CharField(max_length=2)
    X_Ray = models.CharField(max_length=2)
    ChildrenImage = models.CharField(max_length=2)
    Pregnancy_Pelvic_Imaging = models.CharField(max_length=2)
    Cardiology = models.CharField(max_length=2)
    Dentistry = models.CharField(max_length=2)

    def get_fields_names(self):
        fields = Clinic._meta.fields
        field_names = [f.name for f in fields]
        return field_names
    class Meta:
        verbose_name = "Clinic"
        verbose_name_plural = verbose_name
```

ClinicID	Email	Password	ClinicName	Address	address_latitude	address_longitude	ZipCode	Contact	TimeTable	Questiones	MRI	CT	X_Ray	Childre	Pregnanc	Cardiology	Dentistry
1	test_x_ray@q.com	123456	Test X-Ray	70 Pitt St, Sydney NSW	-33.871290	151.110210	2000	1234...	{"8:00 AM": "...	["Do you hav...	0	0	1	0	0	0	0
2	test_children_rad@g.com	123456	Test Children's Radiology	32 Luke Ave, Burwoo...	-33.879314	151.192219	2134	1234...	{"10.00 A.M.":...	["Do you hav...	0	0	0	1	0	0	0
3	test_ct1@g.com	123456	Test CT Clinic	Level 8/287 Elizabeth...	-33.876749	151.203886	2000	1234...	{"8.00 A.M.":...	["Do you hav...	0	1	0	0	0	0	0
4	test_women_rad@g.com	123456	Pregnancy and Pelvic Imaging	161 New South Head...	-33.895275	151.210750	2027	1234...	{"9.00 A.M.":...	["Do you hav...	0	0	0	0	1	0	0
5	test_mri1@g.com	123456	Test MRI 1	O'Sullivan, 101/91 O'...	-33.836549	151.209720	2029	1234...	{"9:00 AM": "...	["Do you hav...	1	0	0	0	0	0	0
6	test_x_ray2@1.com	123456	X_ray No.2	44 Market St, Sydne...	-33.838830	151.217289	2000	1234...	{"8:00 AM": "...	["Do you hav...	0	0	1	0	0	0	0
7	test_ct2@g.com	123456	Test CT No.2	Shop 6 52/48 Sydney...	-33.919209	151.223466	2095	1234...	{"8:00 AM": "...	["Have you e...	0	1	0	0	0	0	0
8	x_ray_ct_rad@rad.rad	123456	X-ray and CT Radiologist	185 Pitt St, Merrylan...	-33.938295	151.217632	2160	1234...	{"8:00 AM": "...	["Hearing aid"...	0	1	1	0	0	0	0
9	test_card_1@1.com	123456	Test Caradiology Clinic	4d/658 Pittwater Rd	-33.928326	151.141791	2100	6543...	{"8:00 AM": "...	["Artificial hea...	0	0	0	0	0	1	0

Figure 37 models.py example code

As mentioned in the previous section, models file reflects the database's table schema for teach table, it makes Django understands what your database architecture is. Also, programmers could also build functions (e.g. get_field_names()) and assign a verbose name of the model for database queries in views.

5.3 Views

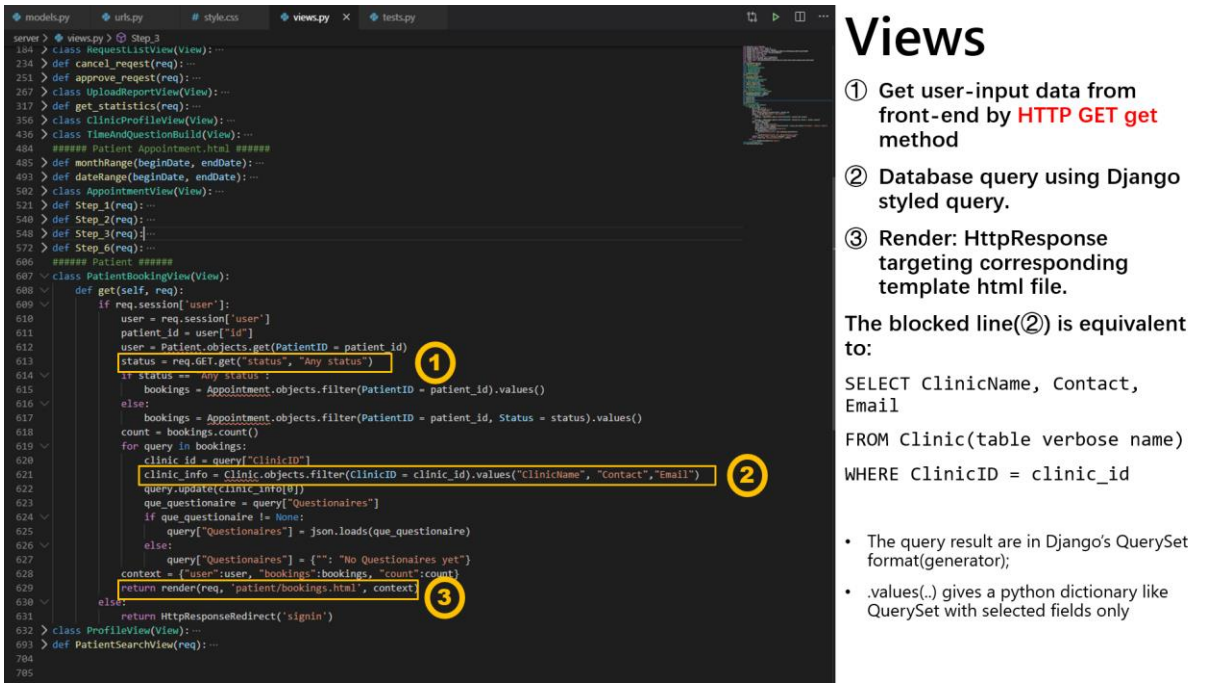


Figure 38 views.py example code

In views file, HTTPresponse, database query, string sterilizing, data pre-processing is written in class views or function views. As shown in the graph above, Django has its own Object Relational Mapper (ORM) that can be used to interact with application data from various relational databases. Using Python's advantage in writing logical functions for data processing, and Python's json module's functions for string processing, the data storing in database are not limited to "traditional" format. For example, the timetable build by clinic users are stored in database in the format of a dumped json string dictionary:

```
mysql> SELECT TimeTable FROM server_clinic WHERE ClinicID = 1;
+-----------------------
```

```
hospital > urls.py > ...
from server import views

urlpatterns = [
    url(r'^static/(?P<path>.*)$', serve,
        {'document_root': settings.STATIC_ROOT}, name='static'),
        re_path(r'media/(?P<path>.*)$', serve, {'document_root': settings.MEDIA_ROOT}),
    url(r'^media/(?P<path>.*)$', serve,
        {'document_root': settings.MEDIA_ROOT}, name='media'),
    path('admin/', admin.site.urls),
    path('', views.HomeView.as_view(), name="home"),
    path('home/', views.HomeView.as_view(), name="home"),
    path('presentation/', views.PresentationView.as_view(), name="presentation"),
    path('chat_box/',views.ChatbotView, name="chatbox"),
    path('signin/', views.SignInView.as_view(), name="signin"),
    path('signup/', views.SignUpView.as_view(), name="signup"),
    path('logout/', views.LogoutView.as_view(), name="logout"),
    #Clinic
    path('dashboard/', views.ClinicDashBoardView.as_view(), name="dashboard"),
    path('GetSchedule/',views.GetSchedule),
    path('clinic_search', views.ClinicSearchView, name="clinic_search"),
    # requestlist views
    path('requestlist/',views.RequestListView.as_view(),name='requestlist'),
    path('cancel_reqest/', views.cancel_reqest),# cancel a request
    path('approve_reqest/',views.approve_reqest),# approve a request
    ###
    path('upload_report/', views.UploadReportView.as_view(), name="upload_report"),
    path('statictics/',views.get_statistics, name='statistics'),
    path('clinic_profile/',views.ClinicProfileView.as_view(), name = 'clinic_profile'),
    path('time_and_question/', views.TimeAndQuestionBuild.as_view(), name = "time_and_question"),
    #Patient
    path('appointment/', views.AppointmentView.as_view(), name="appointment"),
    # appointment sub-views
    path('step1/',views.Step_1),#POST selected examination
    path('step2/',views.Step_2),#GET clinic information by clinic id
    path('step3/', views.Step_3),#GET timeslots from selected: examination, clinic, date
    path('step6/', views.Step_6),#POST insert new appointment record to database and send email
    ###
    path('patient_profile/', views.ProfileView.as_view(),name="patient_profile"),
    path('bookings/', views.PatientBookingView.as_view(), name="bookings"),
    path('patient_search', views.PatientSearchView, name="patient_search"),
]# + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)+ static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Figure 43 urls.py example code

Imagine every URL is the address of a view function or view class, when a link was called at the front-end, using the “urlpatterns” from the URLs file, a corresponding page can be linked to with its views.

5.5 Template

```
Base.html
{% load staticfiles %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>{% block title %}{% endblock %}</title>
    <!-- Favicons-->
    <!-- BASE CSS -->
    <link href="/static/css/font-awesome.min.css" rel="stylesheet" type="text/css">
    <link href="{% static 'css/bootstrap.css' %}" rel="stylesheet">
    <link href="{% static 'css/style.css' %}" rel="stylesheet">
    <link href="{% static 'css/menu.css' %}" rel="stylesheet">
    <link href="{% static 'css/vendors.css' %}" rel="stylesheet">
    <link href="{% static 'css/icon_fonts/css/all_icons_min.css' %}" rel="stylesheet">
    <link rel="stylesheet" href="{% static 'css/loading_2.css' %}">
    <!-- CUSTOM CSS -->
    <link href="{% static 'css/custom.css' %}" rel="stylesheet">
    <!--Pretty Checkbox-->
    <link href="{% static 'css/pretty-checkbox.min.css' %}" rel="stylesheet">
    <link href="{% static 'sass/pretty-checkbox.scss' %}">
    {% block specific_css %}{% endblock %}
</head>
<style type="text/css">
</style>
<body>
    <div id="preloader" class="Fixed">
    </div>
    <div id="page">
        <header class="header_sticky" id = "top">
        </header>
        <main style="margin-top: ">
            {% block main %}{% endblock %}
        </main>
        <footer id="foos">
        </footer>
    </div>
    <script src="{% static 'js/jquery-2.2.4.min.js' %}"></script>
    <script src="{% static 'js/common_scripts.min.js' %}"></script>
    <script src="{% static 'js/functions.js' %}"></script>
    <script src="{% static 'js/bootstrap-datepicker.js' %}"></script>
    <script>
    </script>
    <script>
    </script>
    {% block specific_js %}
    {% endblock %}
    <script>
    </script>
</body>
```

```
Dashboard.html
{% extends 'base.html' %}
{% load static %}
{% block title %} Dashboard {% endblock %}
{% block specific_css %}
    <link href="{% static 'css/admin.css' %}" rel="stylesheet">
    <link  href="{% static 'css/datepicker.css' %}" rel="stylesheet">
{% endblock %}
{% block main %}
    <style type="text/css">

    </style>
    <div class = "bg_color_3">
    </div>
{% endblock %}
{% block specific_js %}
    <script src="/static/js/bootstrap.bundle.min.js"></script>
    <script src="/static/js/jquery.easing.min.js"></script>
    <script src="/static/js/retina-replace.min.js"></script>
    <script src="/static/js/jquery.magnific-popup.min.js"></script>
    <script src="{% static 'js/datepicker.js' %}"></script>
    <script src="/static/js/admin.js"></script>
    <script type="text/javascript">
      $('[data-toggle="datepicker"]').datepicker({format: 'yyyy-mm-dd'});
    </script>
    <script type="text/javascript">
    </script>
    <script>
    </script>
{% endblock %}
```

Figure 44 Template html example code

Although most of the language used in .html files are still the three best friends: HTML5, CSS and JavaScript, there are still some Django template language used to logically present data at front-end. For example, the base.html file is not an independent html files that created for a specific webpage, it is created for loading common css and

script files, as well as header and footer part for each webpage. The unique content of each webpage is “inserted” to base.html using Django’s template language “{% %}” like inserting puzzles.

```
60      ①   {% if count > 0 %}
61          <div role="tablist" class="add_bottom_45 accordion">
62              <ul>
63      ②           {% for i in bookings %}
64                      <div class="card">
65                          <div class="card-header" role=tab id="{{i.AppointmentID}}_tab">
66                              <li><h4><strong style="font-weight: 1000; font-size: 18px;">
67      ③                       {{ i.First_name }},   {{i.Last_name}} </strong>
68                              {% ifequal i.Status 'Approved' %}
69                                  <p class="btn_1 small" style="background-color:#227D51;">{{i.Status}}</p>
70                              {% endifequal %}
71                              {% ifequal i.Status 'Pending' %}
72                                  <strong class="btn_1 small" style="background-color:#F7D94C;">{{i.Status}}</strong>
73                              {% endifequal %}
74                              {% ifequal i.Status 'Cancel' %}
75                                  <strong class="btn_1 small" style="background-color:#e84a5f;">{{i.Status}}</strong>
76                              {% endifequal %}
77                              {% ifequal i.Status 'Requirement Added' %}
78                                  <strong class="btn_1 small" style="background-color:#58B2DC;">{{i.Status}}</strong>
79                              {% endifequal %}
80                              {% ifequal i.Status 'Report Uploaded' %}
81                                  <strong class="btn_1 small" style="background-color:#DB4D6D;">{{i.Status}}</strong>
82                              {% endifequal %}
83                              </h4>
84                              <ul class="booking_details">
85                                  <li><strong>Booking date</strong>{{ i.Date }}</li>
86                                  <li><strong>Booking time</strong>{{ i.Time }}</li>
87                                  <li><strong>Examination</strong>{{ i.Examination }}</li>
88                                  <li><strong>Email</strong>{{ i.Email }}</li>
89                                  <li><strong>Requirements</strong>
90                                      {% if i.requirement %}
91                                          <a href="{% get_media_prefix %}{{i.requirement}}" target="_blank">Click to view</a>
92                                      {% else %}
93                                          <p>No Requirement yet</p>
94                                      {% endif %}
95                                  </li>
96                                  {% if i.file %}
97                                      <li><strong>Report</strong><a href="{% get_media_prefix %}{{i.file}}" target="_blank">Click to
                                        view</a>
98                                  {% endif %}
99                              </ul>
100                             <ul class="buttons">
101                             {% if i.Status == 'Pending' or i.Status == 'Cancel' %}
102                                 <li><a href="#dates"  onclick="getok('{{ i.AppointmentID}}')" class="btn_1 gray approve"><i class="
                                    fa fa-fw fa-check-circle-o"></i> Approve</a></li>
103                             {% endif %}
```

Figure 45 Template html example code 2

Not only for modulate html scripts, Django template language also used for logically generating html codes without “hard-typing” (e.g. ② Generating list items using for loop) and access returned response context by “{{ }}”, for example ③ Patients names are generated by accessing returned information with keys “First_name” and “Last_name”, which is similar to jQuery: $(‘#First_name’).html(“First_name”) but without using html ids or JavaScript objects, just using Python dictionary!

5.6 Example

Let us look at a detailed example of how the timetable was generated in patient’s appointment webpage after a date was selected and “Next” button was clicked on the date tab.

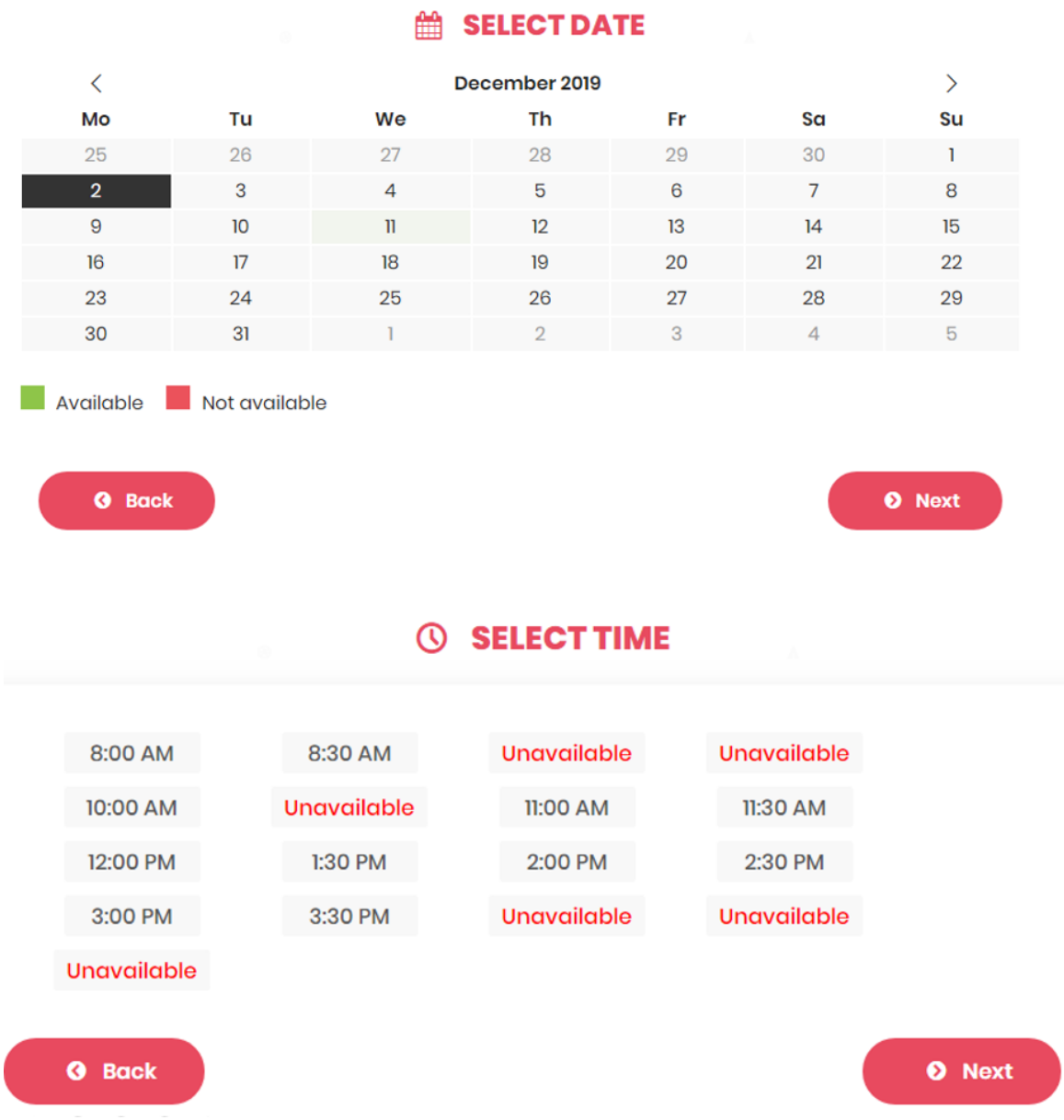


Figure 46 Example: front-end UI

```
$("#next3").click(function (event) {
    event.preventDefault();
    let date = $('#calendar').datepicker("getDate");
    let today_date = new Date();
    if (date) {
        var dateday = date.getDay();
        if (dateday != 0 && dateday!= 6 ){
            if (date >= today_date){
                let year = date.getFullYear();
                let month = date.getMonth() + 1;
                let day = date.getDate();
                $("#my_date").val(year + "-" + month + "-" + day);
                get_array(9,year + "-" + month + "-" + day);
                $("#date_error").hide();
                console.log("Array used to transfer data",myArray);
                get_clinic_and_date(myArray[9],myArray[5]);
                go_to_table("4");
            }else{
                $("#date_error").show();
                $("#date_error_text").html("Cannot select a past date");
                $("#date_error_close").click(function () {
                    $("#date_error").hide();
                })
            }
        }else{
            $("#date_error").show();
            $("#date_error_text").html("Please select a weekday");
            $("#date_error_close").click(function () {
                $("#date_error").hide();
            })
        }
    } else {
        $("#date_error").show();
        $("#date_error_text").html("Please select a date！");
        $("#date_error_close").click(function () {
            $("#date_error").hide();
        })
    }
});
```

Figure 47 Example: "next 3" click function

```
Array used to transfer data                                    (index):828
  (12) [empty, "Gil Case", "casegik@1.com", "["No","No","No","No","No","No"]", "X_Ray", 1,
▶ "Test X-Ray", "test_x_ray@q.com", "123456789", "2019-12-2", empty, "70 Pitt St, Sydney NS
  W"]
```

Figure 48 Example: consoled result at front-end

The “date” selected will be passed through a number of error testing logics at the front-end, and once no error was found, the “date” string will first be stored in the “myArray” JS array used for recording information (console.log result in *Figure 35*), and send to back-end by calling Ajax function, “get_clinic_and_date” targeting URL is the URL of corresponding view function’s URL in urls.py file.

```
//step_3 get timetable filtered by selected clinic, date
function get_clinic_and_date(date,clinic_id) {
    $.ajaxSetup({data:{csrfmiddlewaretoken:'{{ csrf_token }}'}});
    $.ajax({
        type: 'POST',
        url: '/step3/',
        data: {"date":date, "clinic_id":clinic_id},
        success: function (results) {
            renders(results)
        }
    });
}
function renders(results) {
    var html='';
    for (var i = 0; i < results['info'].length; i++) {
        var record = results['info'][i];
        html+='<li>';
        if (record == 'Unavailable'){
            html+='<input type="radio" id="radio'+i+'" name="radio_time" value="'+record+'" disabled>';
            html+='<label style = "font-size:16px; font-weight:500;color:red;" for="radio'+i+'">'+record+'</label>';
        }else{
            html+='<input type="radio" id="radio'+i+'" name="radio_time" value="'+record+'">';
            html+='<label style = "font-size:16px; font-weight:500;" for="radio'+i+'">'+record+'</label>';
        }
        html+='</li>';
    }
    $('#ket').html(html);
```

Figure 49 Example: Ajax function

```
def Step_3(req):
    clinic_id = req.POST.get("clinic_id") # ID of selected clinic
    date = req.POST.get("date") # Date string of selected date
    print("clinic id and date get from front-end", clinic_id, date)
    query_time = Appointment.objects.filter(Date=date,ClinicID=clinic_id).values_list('Time',flat=True)
    print("Existed appointment record for selected clinic at selected date", query_time)
    query_time = list(query_time)
    query_timetable = Clinic.objects.filter(ClinicID=clinic_id).values("TimeTable")
    query_timelist = list(query_timetable)[0]
    time_dict = json.loads(query_timelist["TimeTable"]) #Read dumped string to dict
    print("TimeTable dict queried from database", time_dict)
    time_keys = []
    for key,value in time_dict.items():
        if key != '':
            time_dict[key] = int(value)
            if key in query_time:
                if time_dict[key] > 1:
                    time_dict[key] -= 1
                else:
                    time_dict[key] = "Unavailable"
    for key in time_dict:
        if time_dict[key] != '':
            if time_dict[key] != "Unavailable":
                time_keys.append(key)
            else:
                time_keys.append("Unavailable")
    print("Processed Timetable using Python language and data structure", time_keys )
    return JsonResponse({"info":time_keys})
```

Figure 50 Example: view function

```
clinic id and date get from front-end 1 2019-12-2
Existed appointment record for selected clinic at selected date <QuerySet ['4:00 PM', '5:00 PM', '10:30 AM', '4:30 PM', '9:30 AM', '9:00 AM']>
TimeTable dict queried from database {'8:00 AM': '1', '8:30 AM': '1', '9:00 AM': '1', '9:30 AM': '1', '10:00 AM': '1', '10:30 AM': '1', '11:00 AM': '1
:00 PM': '1', '3:30 PM': '1', '4:00 PM': '1', '4:30 PM': '1', '5:00 PM': '1'}
Processed Timetable using Python language and data structure ['8:00 AM', '8:30 AM', 'Unavailable', 'Unavailable', '10:00 AM', 'Unavailable', '11:00 AM
ailable', 'Unavailable', 'Unavailable']
[23/Nov/2019 02:12:43] "POST /step3/ HTTP/1.1" 200 225
```

Figure 51 Example: view function printed output

The back-end function gets the clinic id “1” and date string “2019-12-2” using from HTTP POST get method, and get the data sent from ajax function, then, query the server database “Appointment” table and “Clinic” table for records. After targeted records grabbed, using a simple Python data processing structure, a list of time was created and send back to ajax using “JsonResponse”(send json string).

```
<div data-tab id="tab4" style="display: none">
    <div class="main_title_3" style="margin-bottom: 10px;">
        <h3 style="color: #e84a5f; font-size:24px; font-weight: 800;">
            <i class="icon-clock-5"></i>Select Time
    </h3></div>
    <div class="box_general">
        <div class="row" style='margin:auto; padding-bottom: 30px;'>
            <div class="col-8 text-center">
                <ul id="ket" style="width:150%;" class="time_select">
                </ul>
            </div>
        </div>
        <div style="display: none" id="time_error" class="alert alert-warning ">
                <button id="time_error_close" type="button" class="close">×</button>
                <strong id="time_error_text"></strong>
        </div>
        <div class="row">
            <div class="col-6" style="margin:0px;"><a href="" id="pre4" class="btn_1 medium" style="float:left;"><i
            class="icon-angle-circled-left"></i>Back</a></div>
            <div class="col-6" style="margin:0px;"><a href="" id="next4" class="btn_1 medium" style="float:right;"><i
            class="icon-angle-circled-right"></i>Next</a></div>
        </div>
    </div>
</div>
```

Figure 52 Example: html code for time list tab(step 5)

Once ajax function receives back-end feedback, it calls the render(result) function and generated time slot list to html codes, finally send these html codes to the target list with id “ket”.

Django’s ORM abled less coding for database query, sending HTTP response and data processing. Some machine learning logic can also directly be applied in Django using Python language, like Convolutional Neural Network, recommending system without “cross-language” translation between Python and JS or other languages. However, there are still some drawback about Django, sometimes the “cross-language” template coding causes “fights” between languages, and takes lots of effort to fix.

5.7 Google APIs

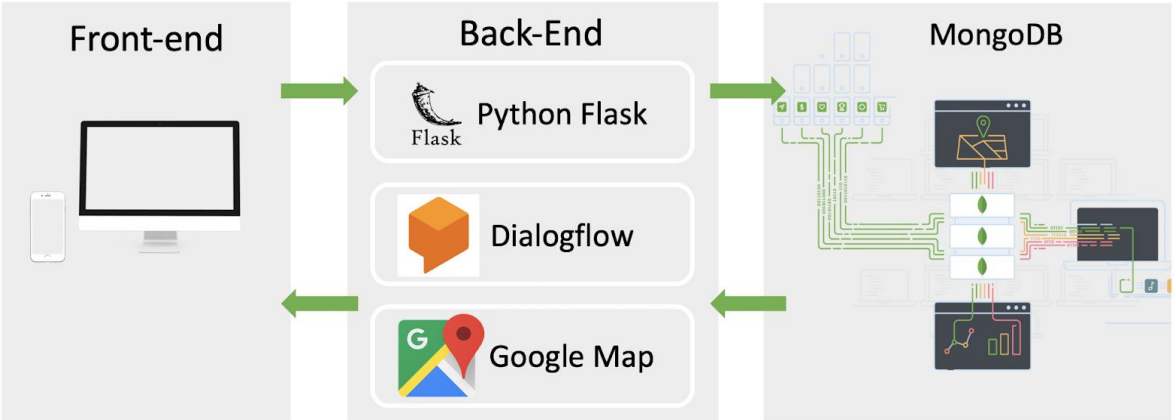
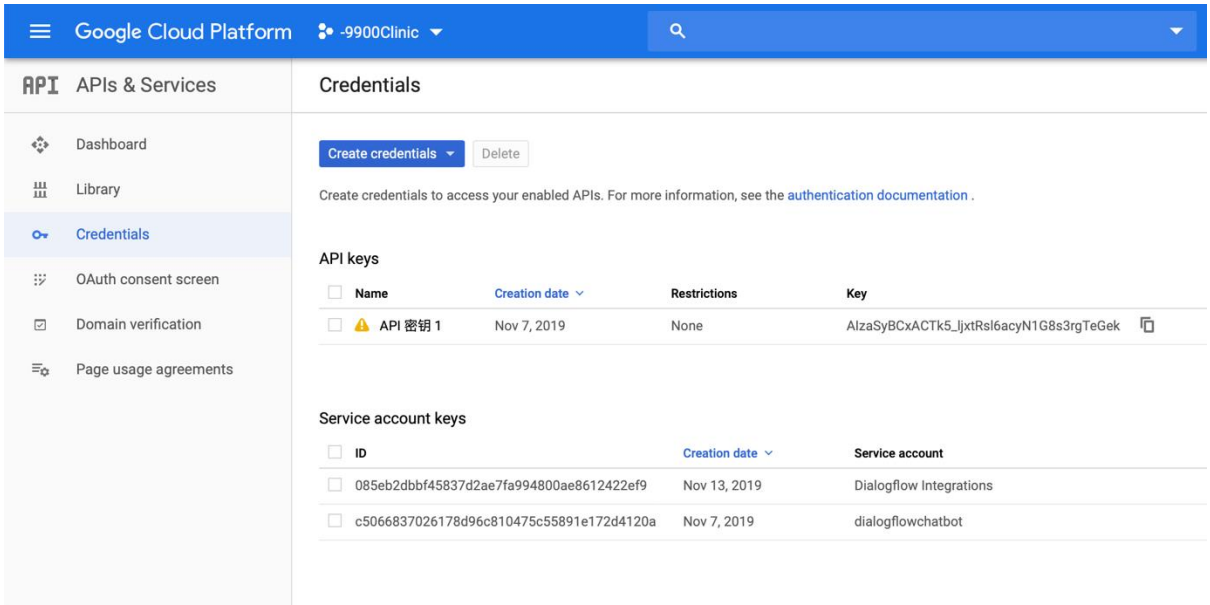


Figure 53 Google APIs structure

To create a Dialogflow API, steps below need to be implemented:

1. Create an agent in Dialogflow console;
2. Change the settings in google cloud console to make it available for rest API connection and download authentication json file to the local space.



1. Create flask rest sever and use the code below to start interaction between local flask sever and Dialogflow console:

```
def detect_intent_with_parameters(project_id, session_id, query_params, language_code,
        user_input):
    """Returns the result of detect intent with texts as inputs.

    Using the same `session_id` between requests allows continuation
    of the conversaion."""
    session_client = dialogflow.SessionsClient()

    session = session_client.session_path(project_id, session_id)
    print('Session path: {}\n'.format(session))

    # text = "this is as test"
    text = user_input

    text_input = dialogflow.types.TextInput(
        text=text, language_code=language_code)

    query_input = dialogflow.types.QueryInput(text=text_input)

    response = session_client.detect_intent(
        session=session, query_input=query_input,
        query_params=query_params
    )

    print('=' * 20)
    print('Query text: {}'.format(response.query_result.query_text))
    print('Detected intent: {} (confidence: {})\n'.format(
        response.query_result.intent.display_name,
        response.query_result.intent_detection_confidence))
    print('Fulfillment text: {}\n'.format(
        response.query_result.fulfillment_text))

    return format(response.query_result.fulfillment_text)
```

2. Create flask API so that front-end can interact information with dialogflow:

```
@api.route('/ASK_questions_to_dialogflow/<string:id>')
@api.param('id', 'The name of dentist')
```

```
class Dentists(Resource):
    @api.response(404, 'Collection was not found')
    @api.response(200, 'Successful')
    @api.doc(description="To give dentist's information about location
and specialization by dentist's name.")
    def get(self, id):
        GOOGLE_AUTHENTICATION_FILE_NAME = "-9900Clinic-
c50668370261.json"
        current_directory = os.path.dirname(os.path.realpath(__file__))
        path = os.path.join(current_directory,
GOOGLE_AUTHENTICATION_FILE_NAME)
        os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = path

        GOOGLE_PROJECT_ID = "e9900clinic-qtuqsm"
        session_id = "1234567891"
        context_short_name = "does_not_matter"

        context_name = "projects/" + GOOGLE_PROJECT_ID +
"/agent/sessions/" + session_id + "/contexts/" + \
                context_short_name.lower()

        parameters = dialogflow.types.struct_pb2.Struct()
        # parameters["foo"] = "bar"

        context_1 = dialogflow.types.context_pb2.Context(
            name=context_name,
            lifespan_count=2,
            parameters=parameters
        )
        query_params_1 = {"contexts": [context_1]}

        language_code = 'en'
        input_text = id

        response = detect_intent_with_parameters(
            project_id=GOOGLE_PROJECT_ID,
            session_id=session_id,
            query_params=query_params_1,
            language_code=language_code,
            user_input=input_text
        )
        print("response:",response)
        result = {}
        result['result'] = response

        return result, 200
```

5.8 Flask API

Send http requests to interact with the backend restful API created by flask:

```
$.ajax({
type: "GET",
dataType: "json",
url: "http://127.0.0.1:8888/ASK_questions_to_dialogflow/"+encodeURI(res.value),
success: function(data){
    return botui.message.add({ // show a message
        delay: 0,
        loading:true
        }).then(function(index){
            window.scrollTo({
                top: 0,
            });
            botui.message.update(index, {
                loading: false,
                content: data.result
            });
        }).then(function(){
            if (flag != 0){
                init(2)
            }
        })
    },
error: function(XMLHttpRequest, textStatus, errorThrown){
        alert(XMLHttpRequest.status + ":Error!");
    }
});
```

5.9 BotUI

Import BotUI framework in chatbox.html

```
<link href="{% static 'css/date_picker.css' %}" rel="stylesheet">
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/botui/build/botui.min.css" >
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/botui/build/botui-theme-
default.css" >
<script src="{% static 'js/bootstrap-datepicker.js' %}"></script>
<script src="https://cdn.jsdelivr.net/vue/latest/vue.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/botui/build/botui.js"></script>
```

Initialize an instance of BotUI:

```
var botui = new BotUI('hello-world');
```

Repeatly send http requests to flask API to interact with the chatbot:

```
function init(flag){
    //#5 API call
    if (flag == 1){
        botui.message.add({ // show a message
        delay: 200,
        loading:true,
        content: 'Hello, what can I do for you?'
        }).then(function () {
            return botui.action.text({ // show 'text' action
                action: {
                placeholder: 'Please Enter Here'
                }
            });
        }).then(function (res) {
            window.scrollTo({
                top: 0,
            });
            $.ajax({
            type: "GET",
            dataType: "json",
            url: "http://127.0.0.1:8888/ASK_questions_to_dialogflow/"+encodeURI(res.value),
            success: function(data){
                return botui.message.add({ // show a message
                    delay: 0,
                    loading:true
                    }).then(function(index){
                        window.scrollTo({
                            top: 0,
                        });
                        botui.message.update(index, {
                            loading: false,
                            content: data.result
                        });

                    }).then(function(){
                        if (flag != 0){
                            init(2)
                        }
                    })
                },
            error: function(XMLHttpRequest, textStatus, errorThrown){
                    alert(XMLHttpRequest.status + ":Parameters Error!");
                }
            });
        })
    }
    else{
```

Figure 54 BotUI example code1

```
    else{
        botui.action.text({ // show 'text' action
            action: {
                placeholder: 'Please Enter Here'
            }
        }).then(function (res) {
            window.scrollTo({
                top: 0,
            });
            $.ajax({
            type: "GET",
            dataType: "json",
            url: "http://127.0.0.1:8888/ASK_questions_to_dialogflow/"+encodeURI(res.value),
            success: function(data){
                return botui.message.add({ // show a message
                    delay: 0,
                    loading:true
                    }).then(function(index){
                        window.scrollTo({
                            top: 0,
                        });
                        botui.message.update(index, {
                            loading: false,
                            content: data.result
                        });
                    }).then(function(){
                        if (flag != 0){
                            init(2)
                        }
                    })
                },
            error: function(XMLHttpRequest, textStatus, errorThrown){
                    alert(XMLHttpRequest.status + ":Error!");
                }
            });
        })
    }

}

window.scrollTo({
                top: 0,
            });
init(1);
```

Figure 55 BotUI example code2

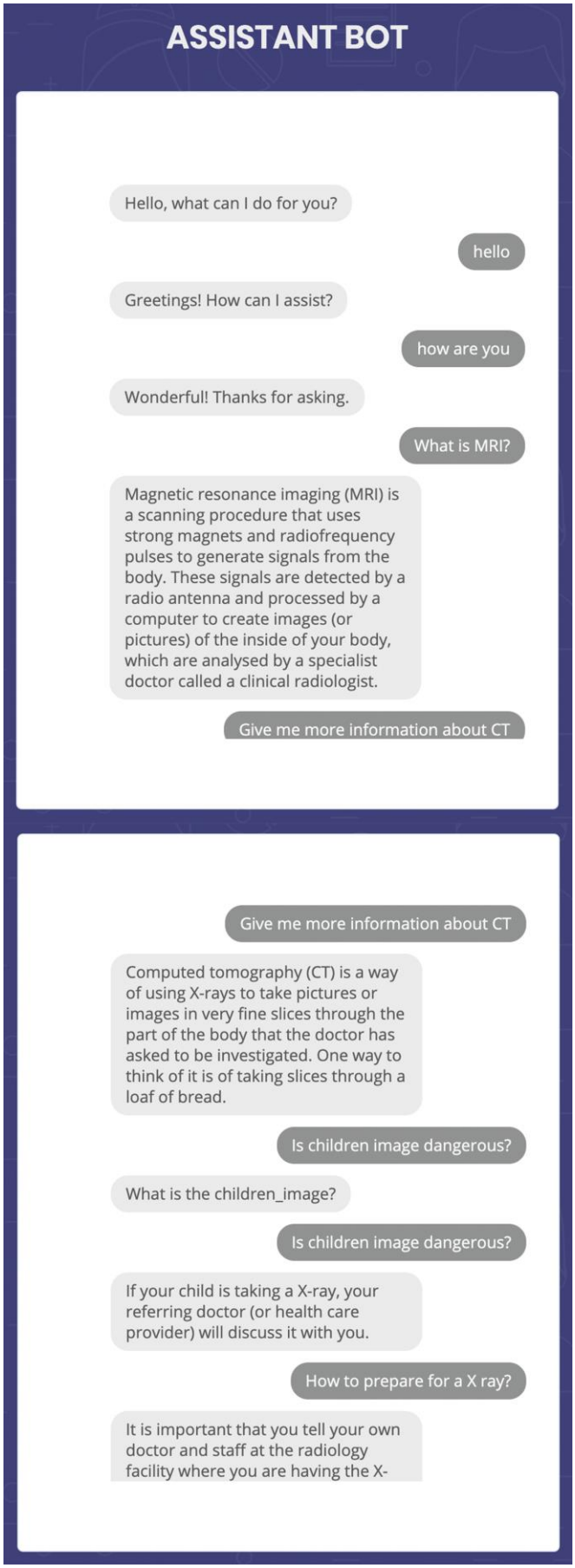


Figure 56 Chatbot chatting example

The steps of the how answers were generated are:

1. The information in the chatting window will be received by BotUI html.
2. The BotUI will send the information to Flask server through Flask API.
3. Once the Flask server has received a request, it will send the message to the Dialogflow console through Dialogflow API.
4. The Dialogflow will response the result to Flask server.
5. The Flask server send back the result.
6. The BotUI will show the information, which received from Flask server, on the chatting window.

6. Conclusion

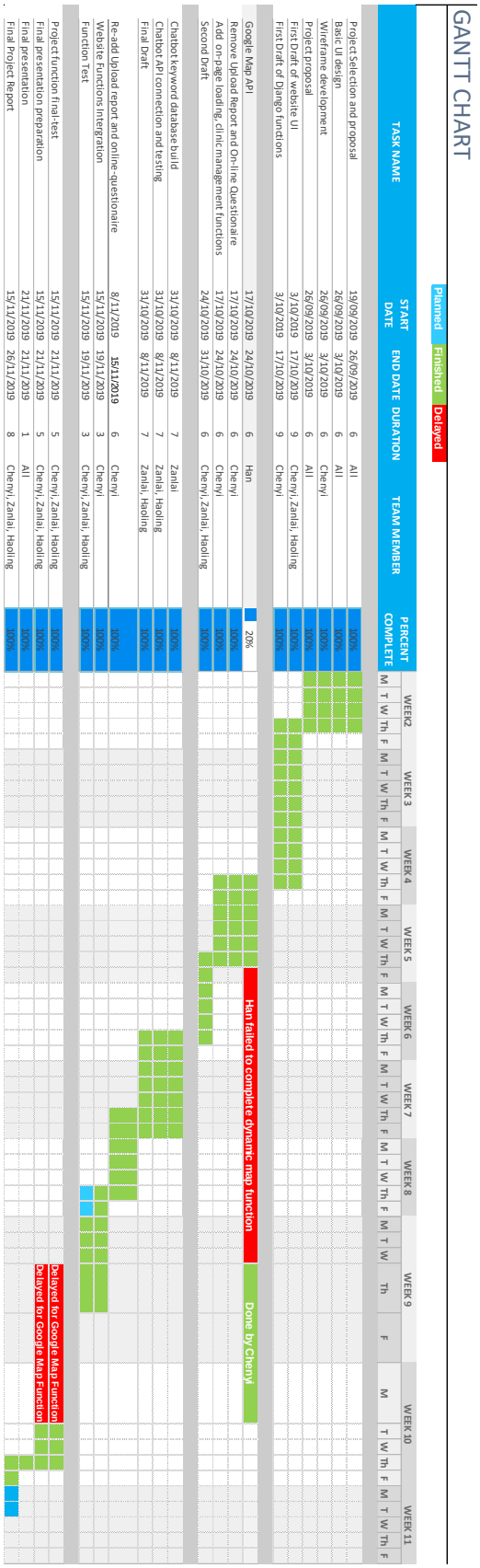
Comparing to project proposal, all most all functions were implemented and lots of extra functions (e.g. upload file) were also implemented by requirements from supervisors.

For future application of this full-stack project, the most ideal situation is that, all radiology clinics has their databases connect to the server database, so that all changes made in server database can be synchronized with clinics' databases, which avoids human mistake when transferring data, and time-zone delay. However, there has to be a high-fee from database security maintenance.

Moreover, because of the time limited caused by trimester, we run out of time to implement a check-list to help patients prepare the examination (not in the project scope, an other add-on function), if there's any chance in the future that we can put this project on-line, we are looking forward to finish this function.

7. Gantt Chart

Table 1 Project Gantt Chart



8. References

What Salary Do Health Care Receptionists Tend to Make? [online] Available at: https://work.chron.com/salary-health-care-receptionists-tend-make-9699.html [Accessed 7 Oct. 2019].

Wait times for public and private hospitals. [online] Available at: https://www.hbf.com.au/-/media/files/reports/hbf-wait-times-report-2018.pdf [Accessed 7 Oct. 2019].

QuerySet API reference | Django documentation | Django. (2019). Retrieved 22 November 2019, from https://docs.djangoproject.com/en/2.2/ref/models/querysets/

Definition of FRONT-END. (2019). Retrieved 21 November 2019, from https://www.merriam-webster.com/dictionary/front-end

Django overview | Django. (2019). Retrieved 21 November 2019, from https://www.djangoproject.com/start/overview/

Django's Structure – A Heretic's Eye View - Python Django. (2019). Retrieved 22 November 2019, from https://djangobook.com/mdj2-django-structure/

Appendix A – Webpages Screenshot

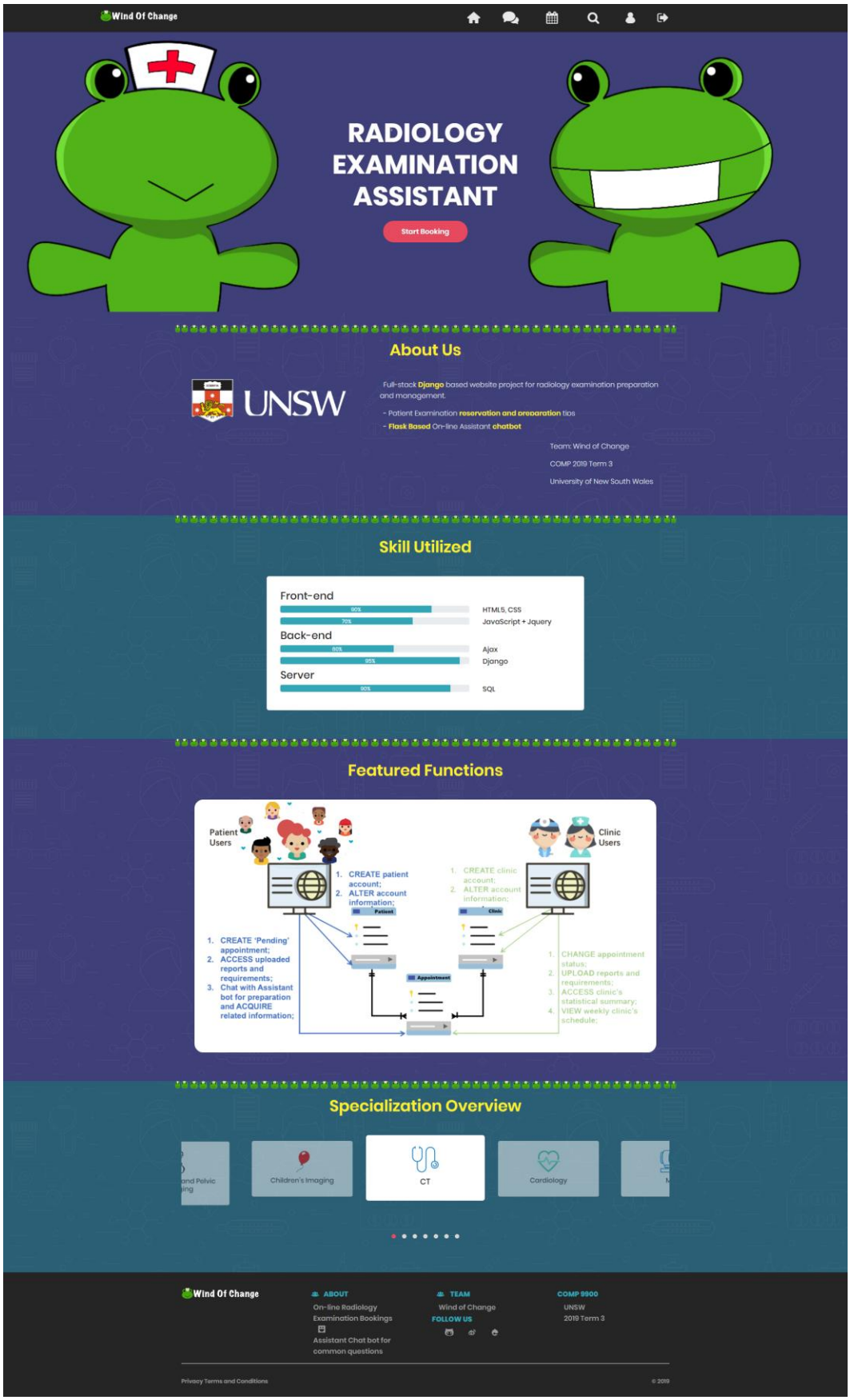


Figure 57 Homepage

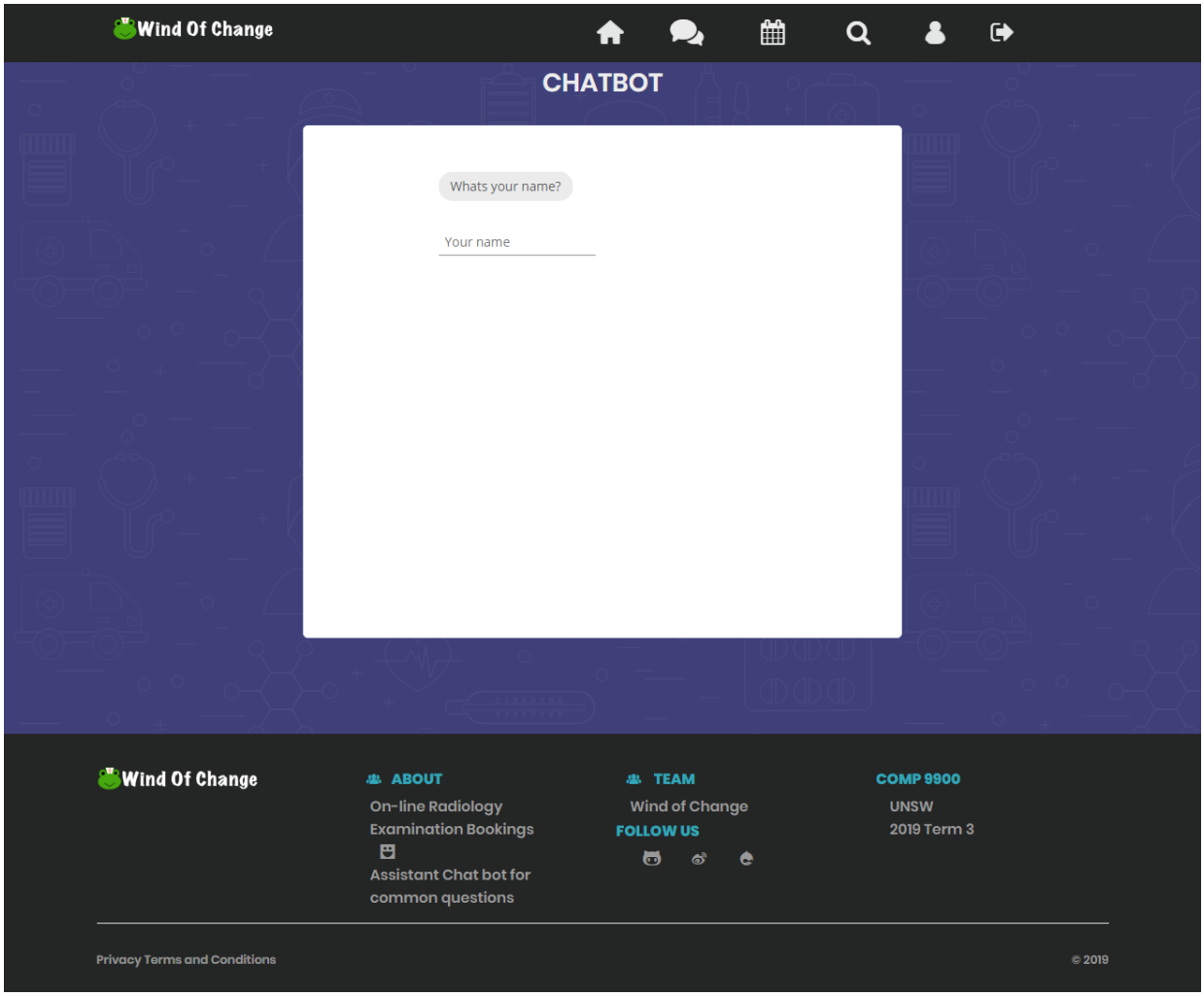


Figure 58 Chatbot

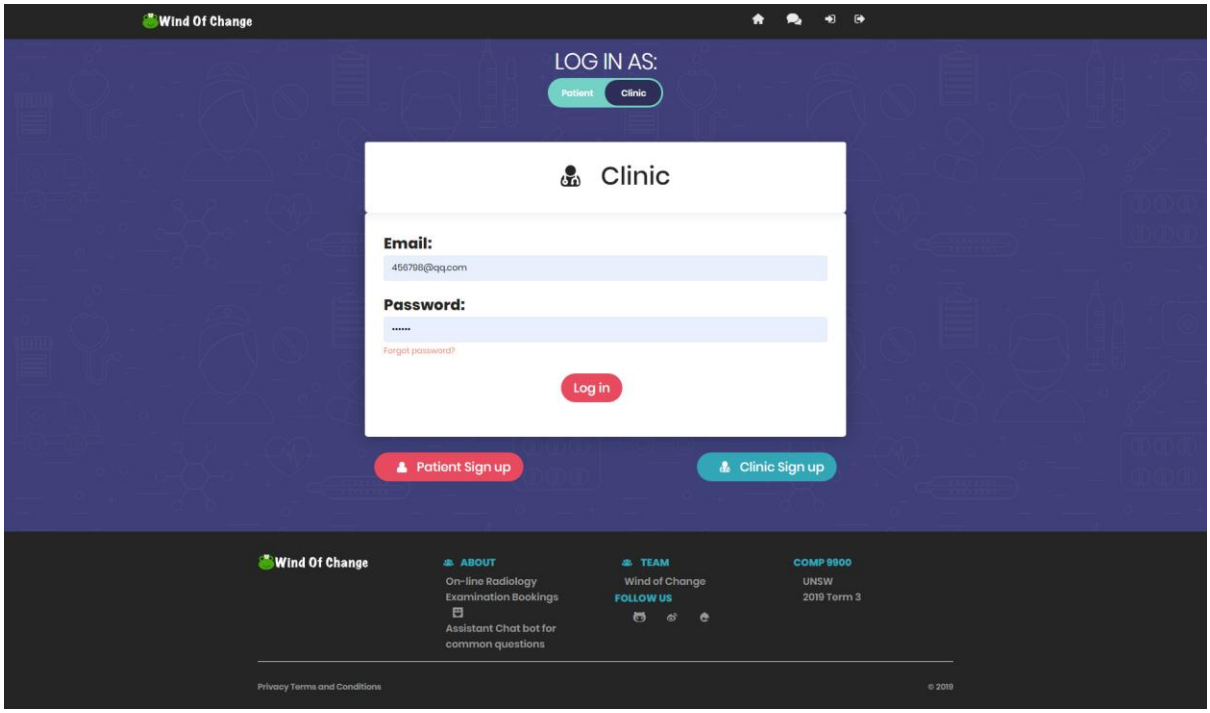


Figure 59 Sign-in

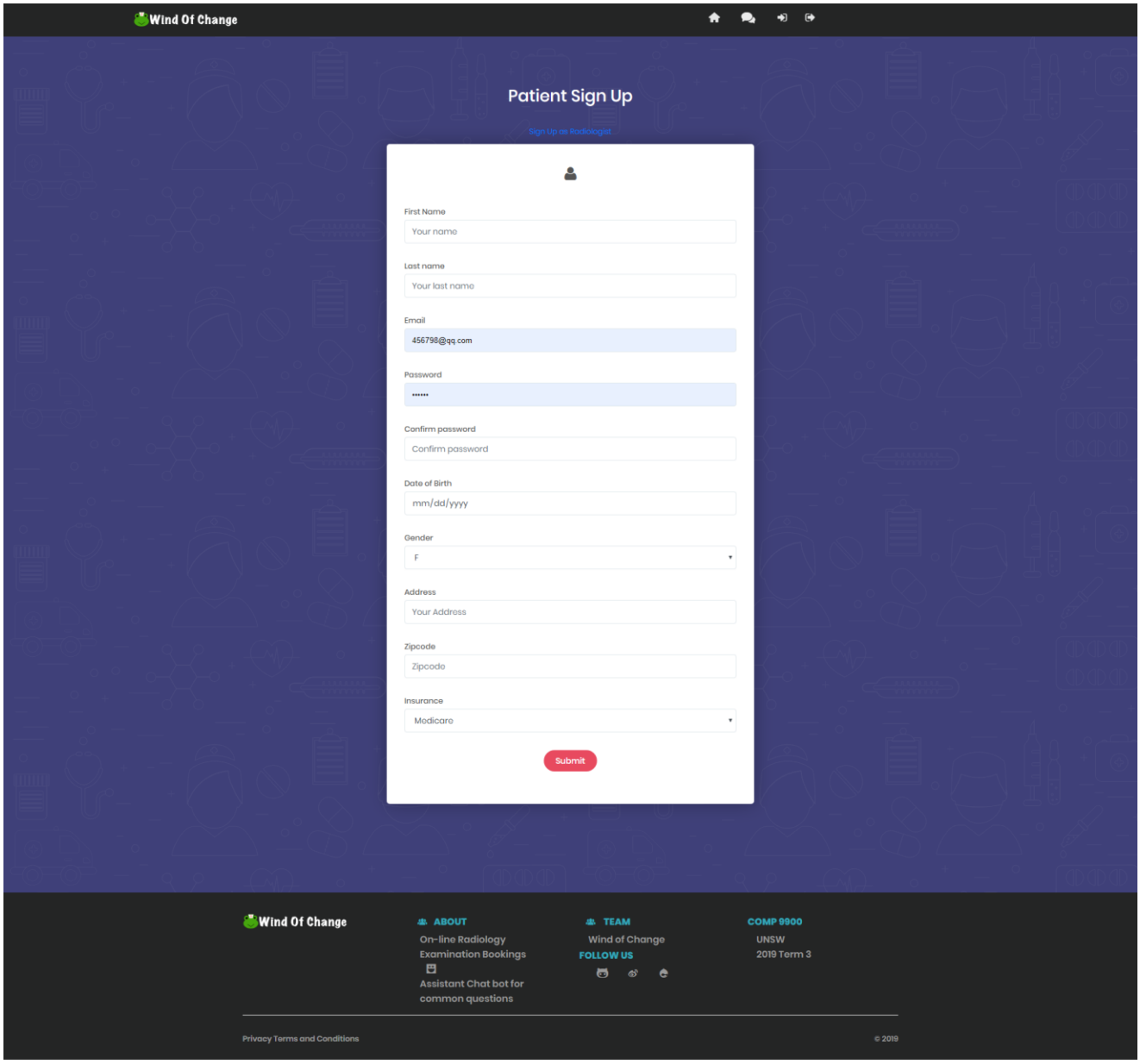


Figure 60 Patient Sign-up

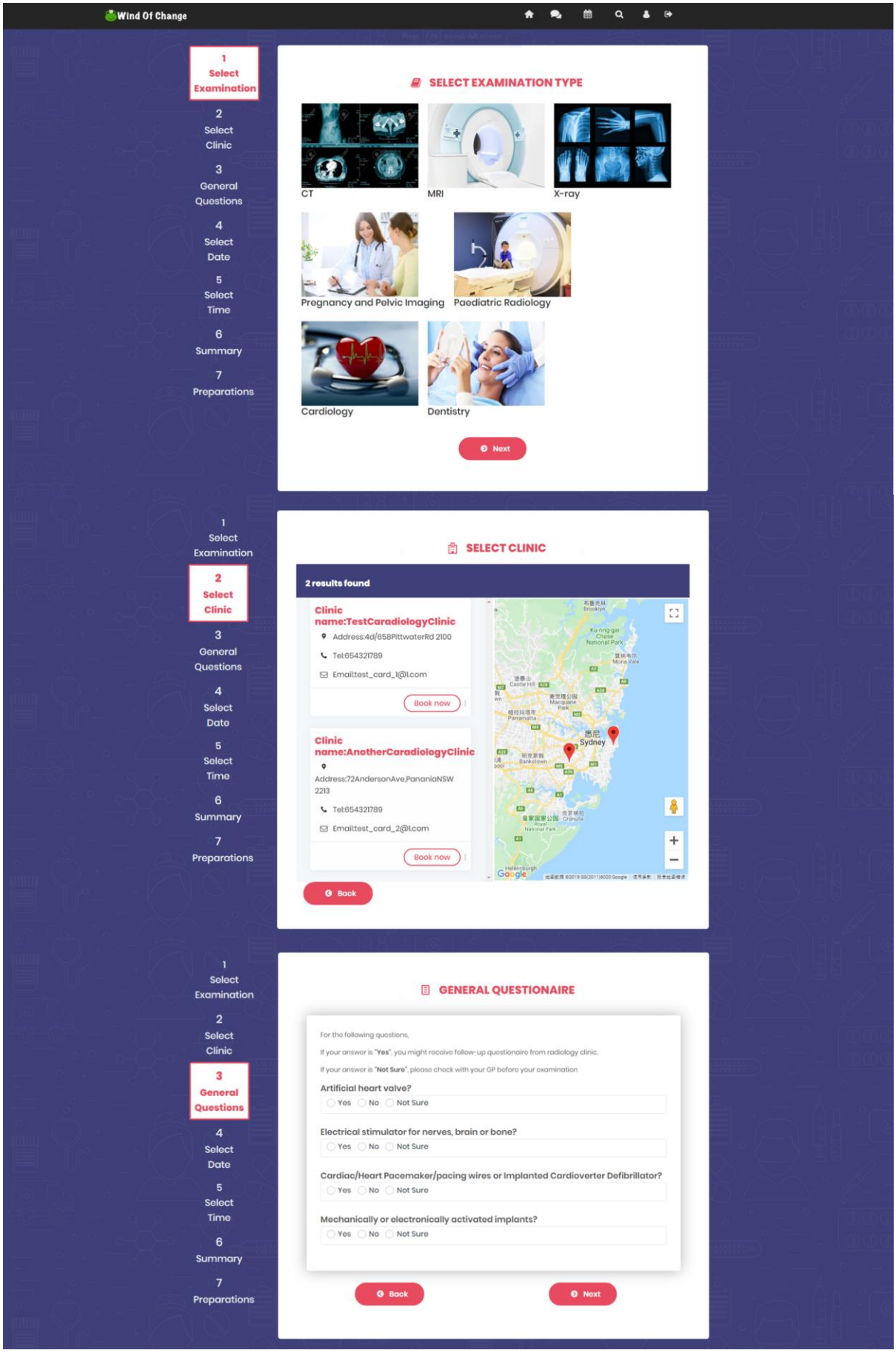


Figure 61 Appointment Booking 1

Figure 62 Appointment Booking 2

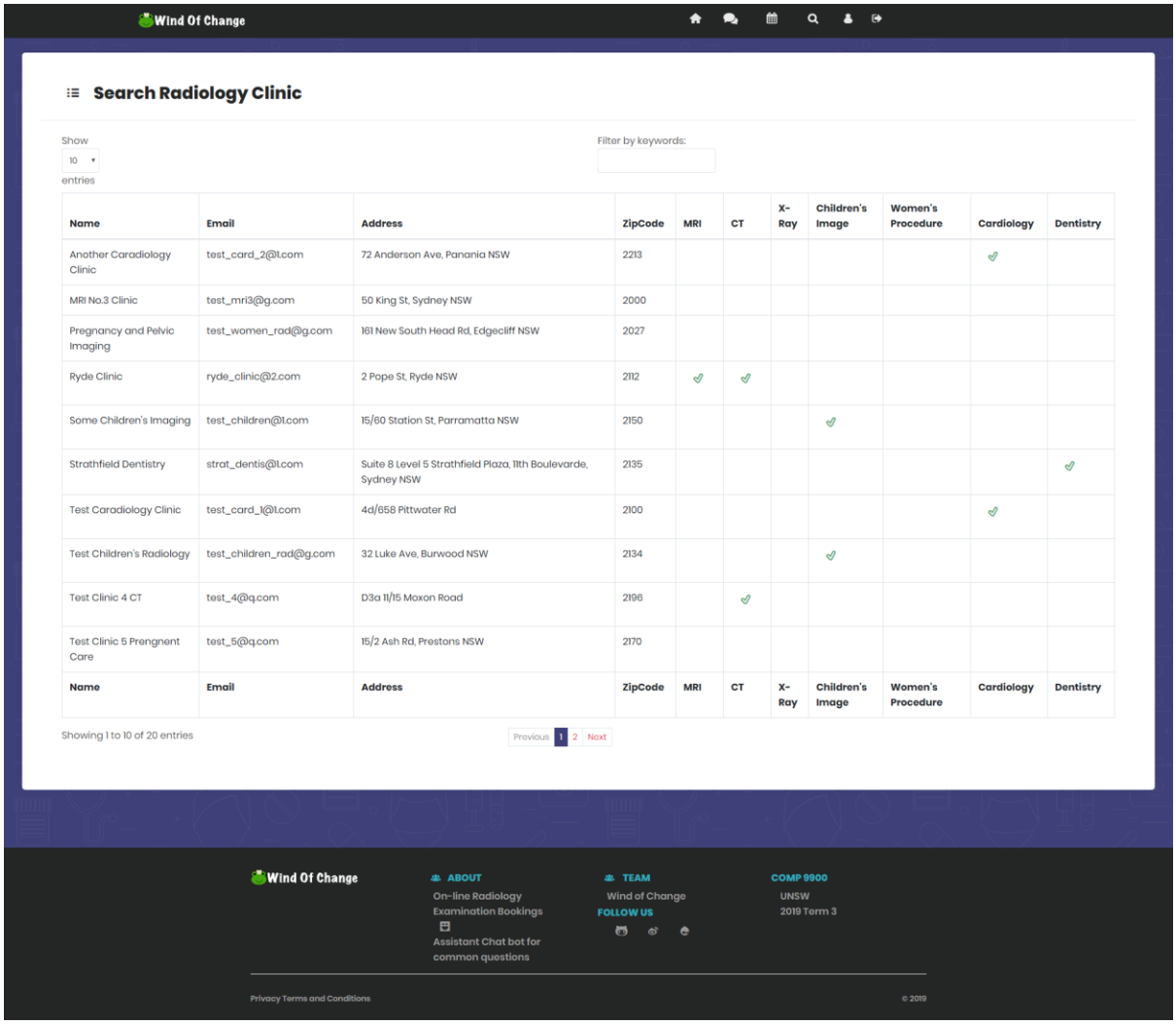


Figure 63 Clinic information look-up

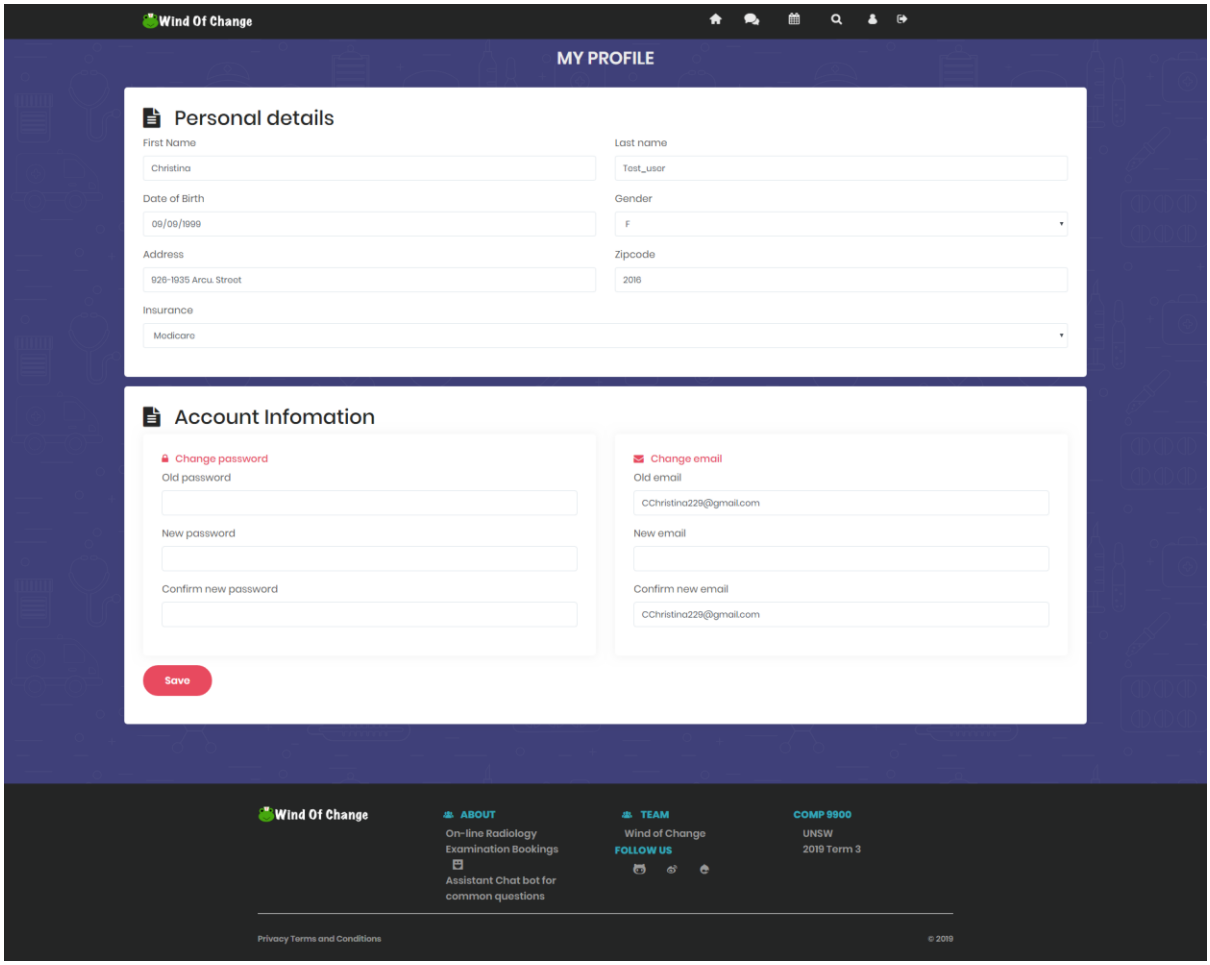


Figure 64 Patient Profile

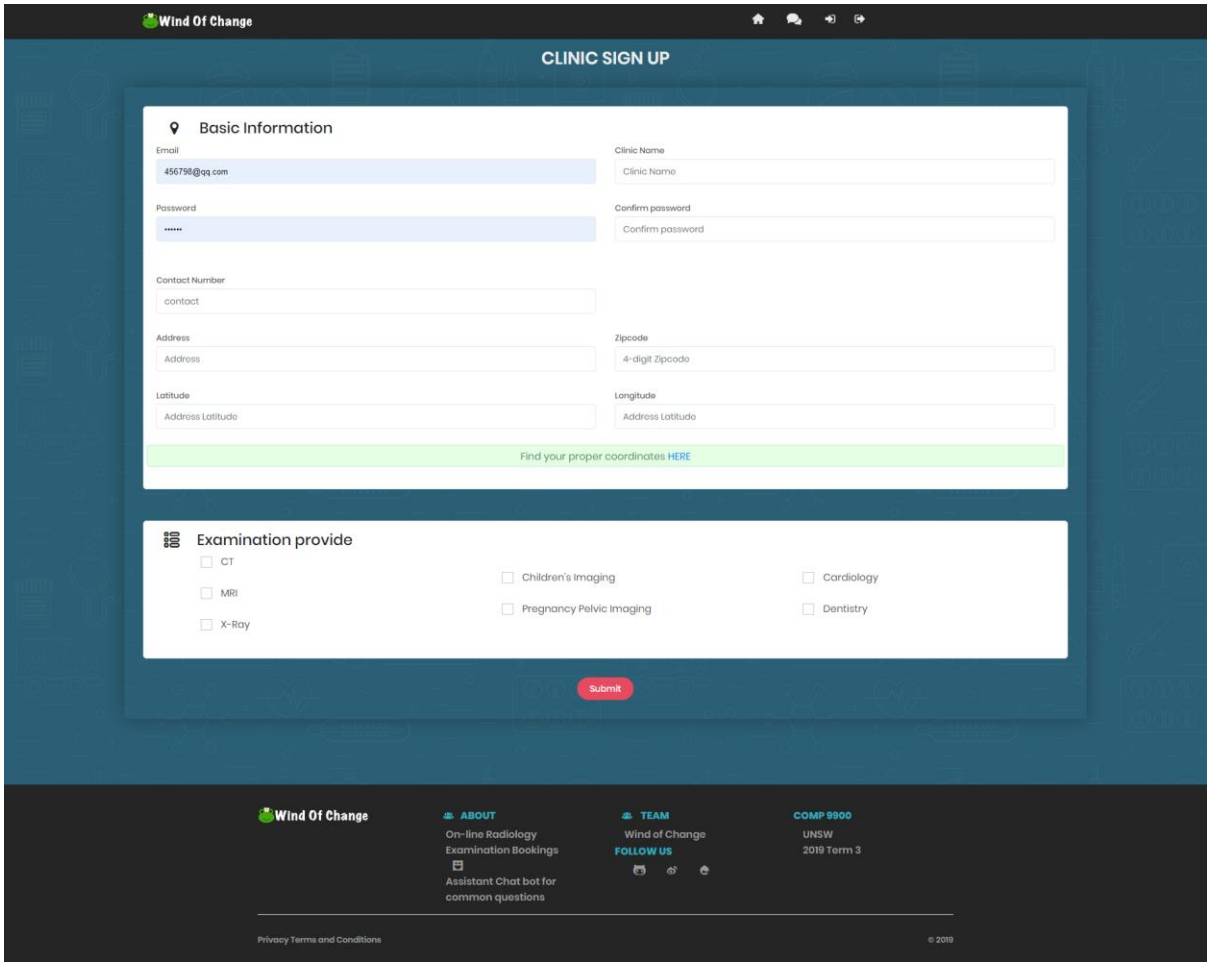


Figure 65 Clinic user sign up

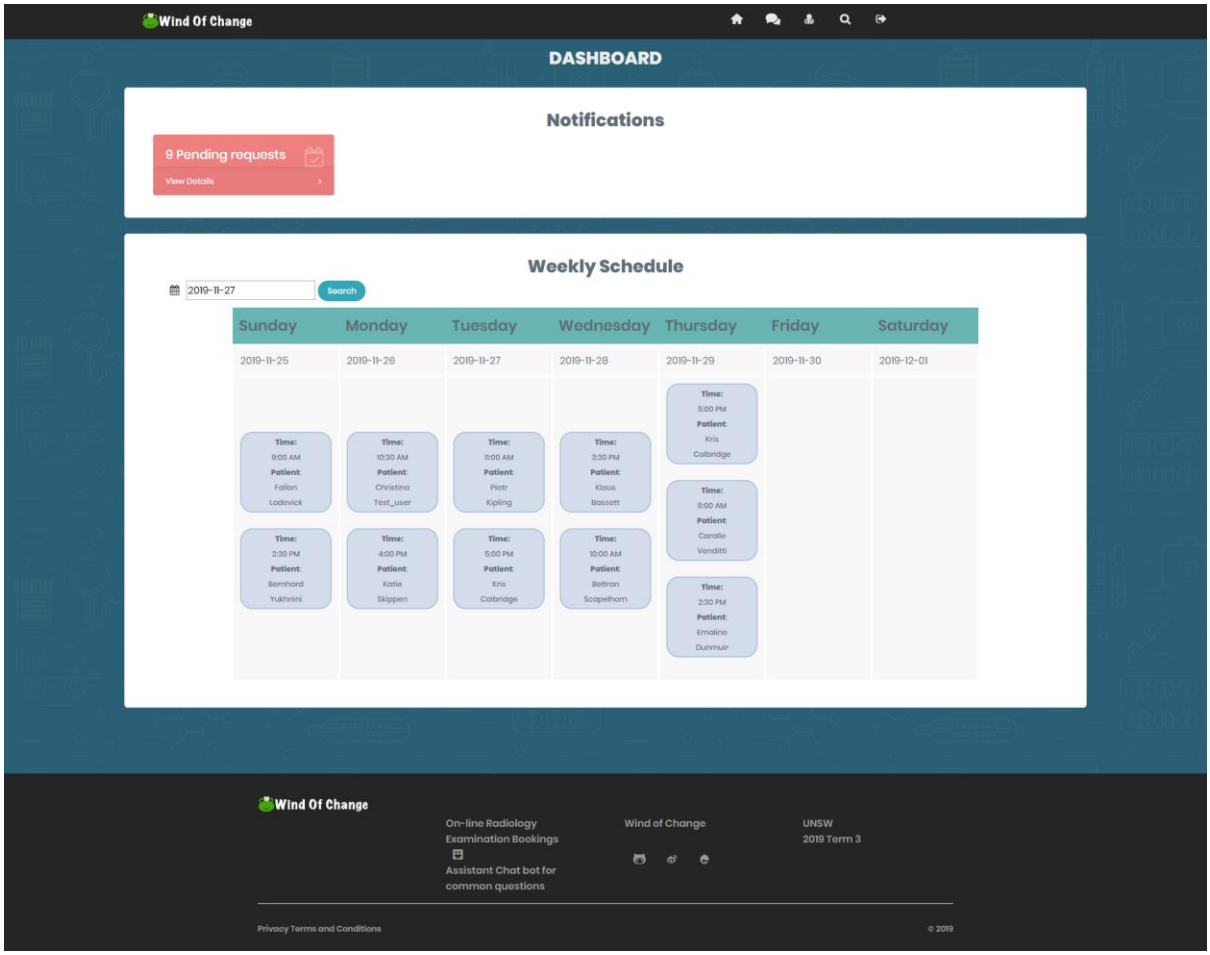


Figure 66 Clinic user dashboard

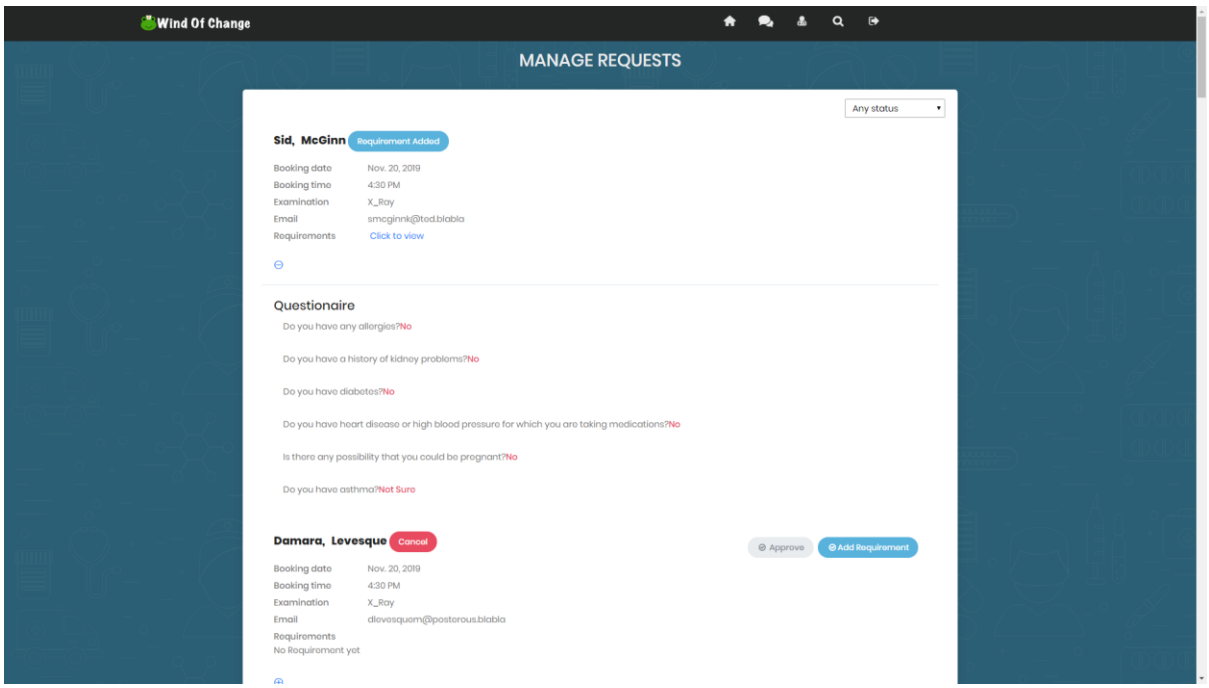


Figure 67 Clinic user Request List 1

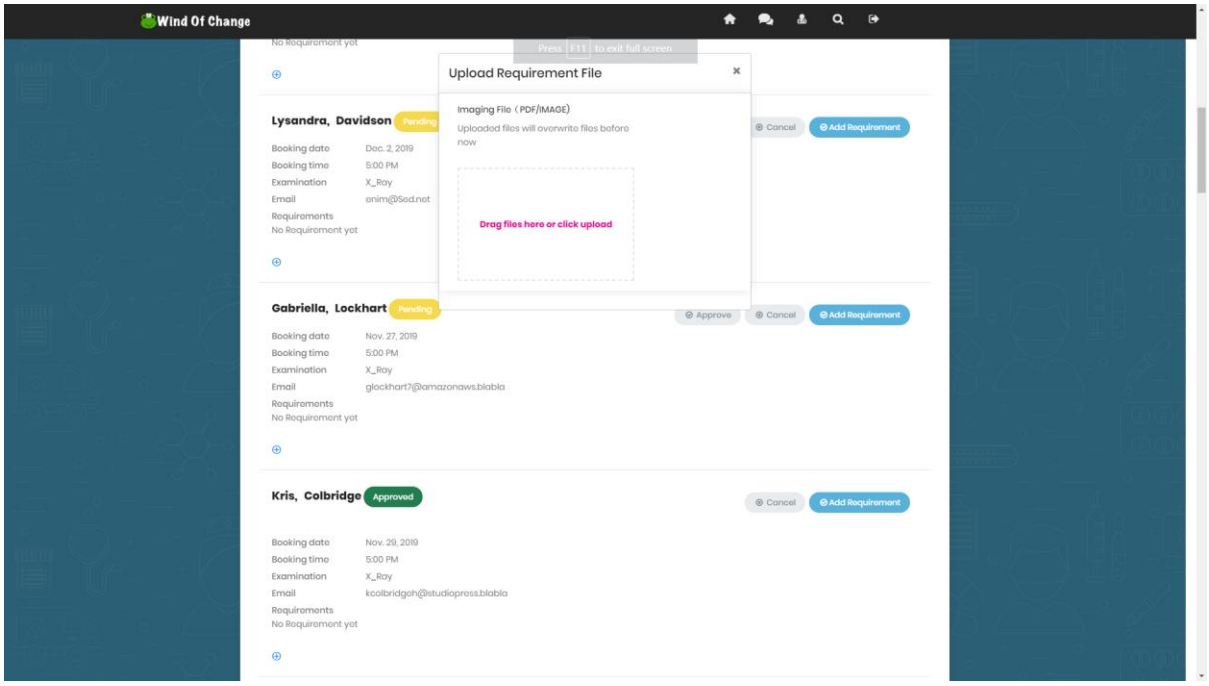


Figure 68 Clinic user Request List 2

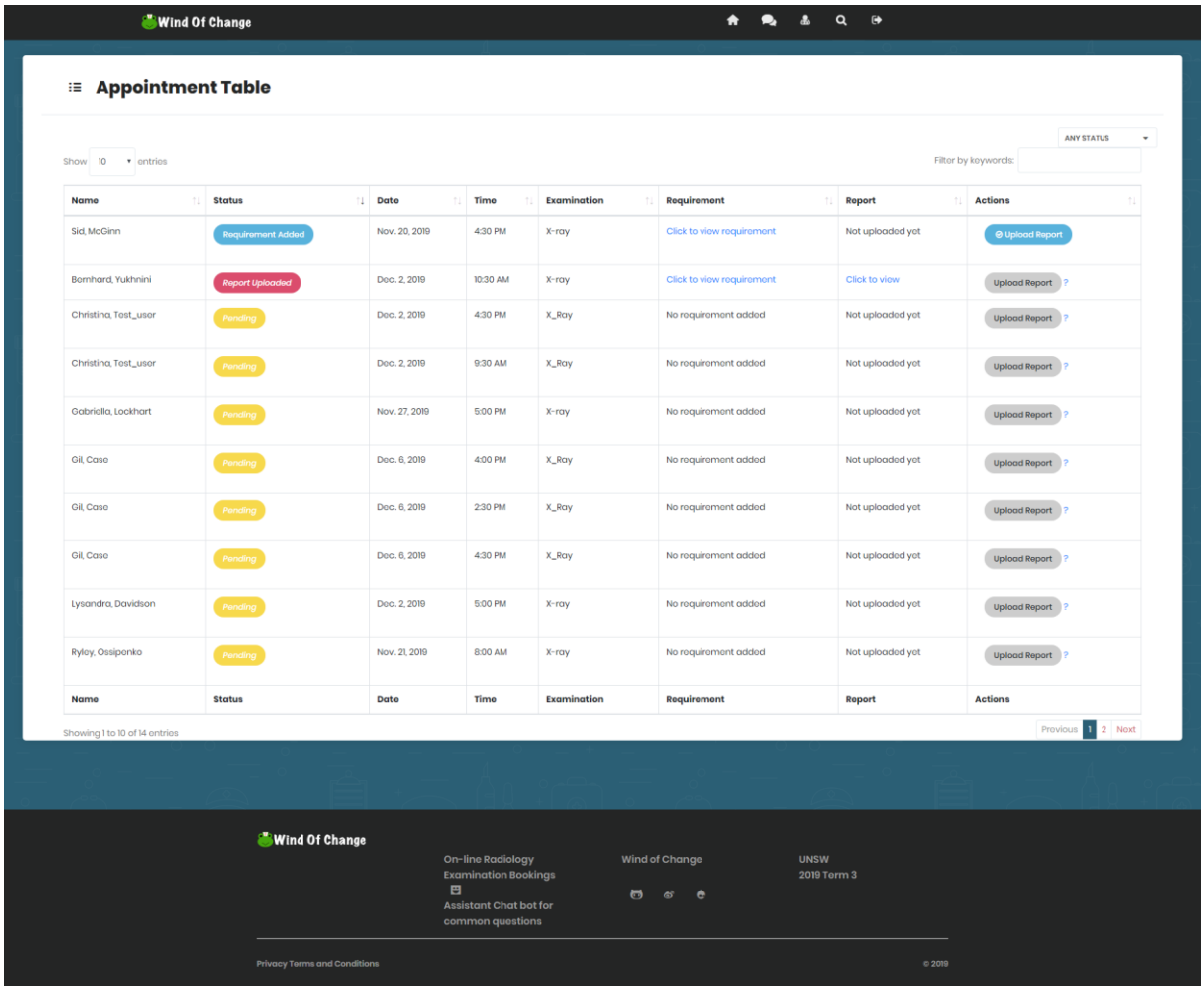


Figure 69 Clinic user Upload Report

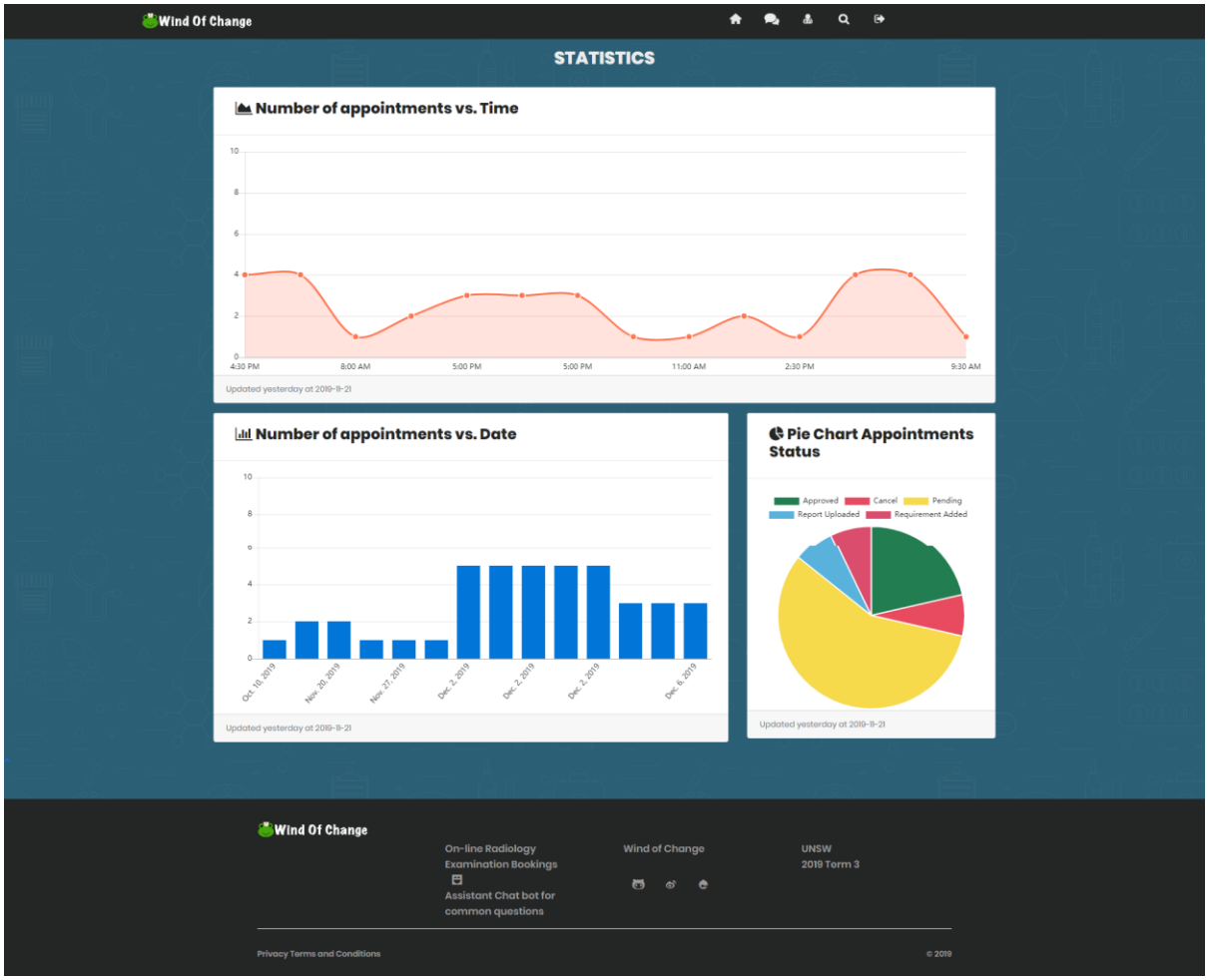


Figure 70 Statistics

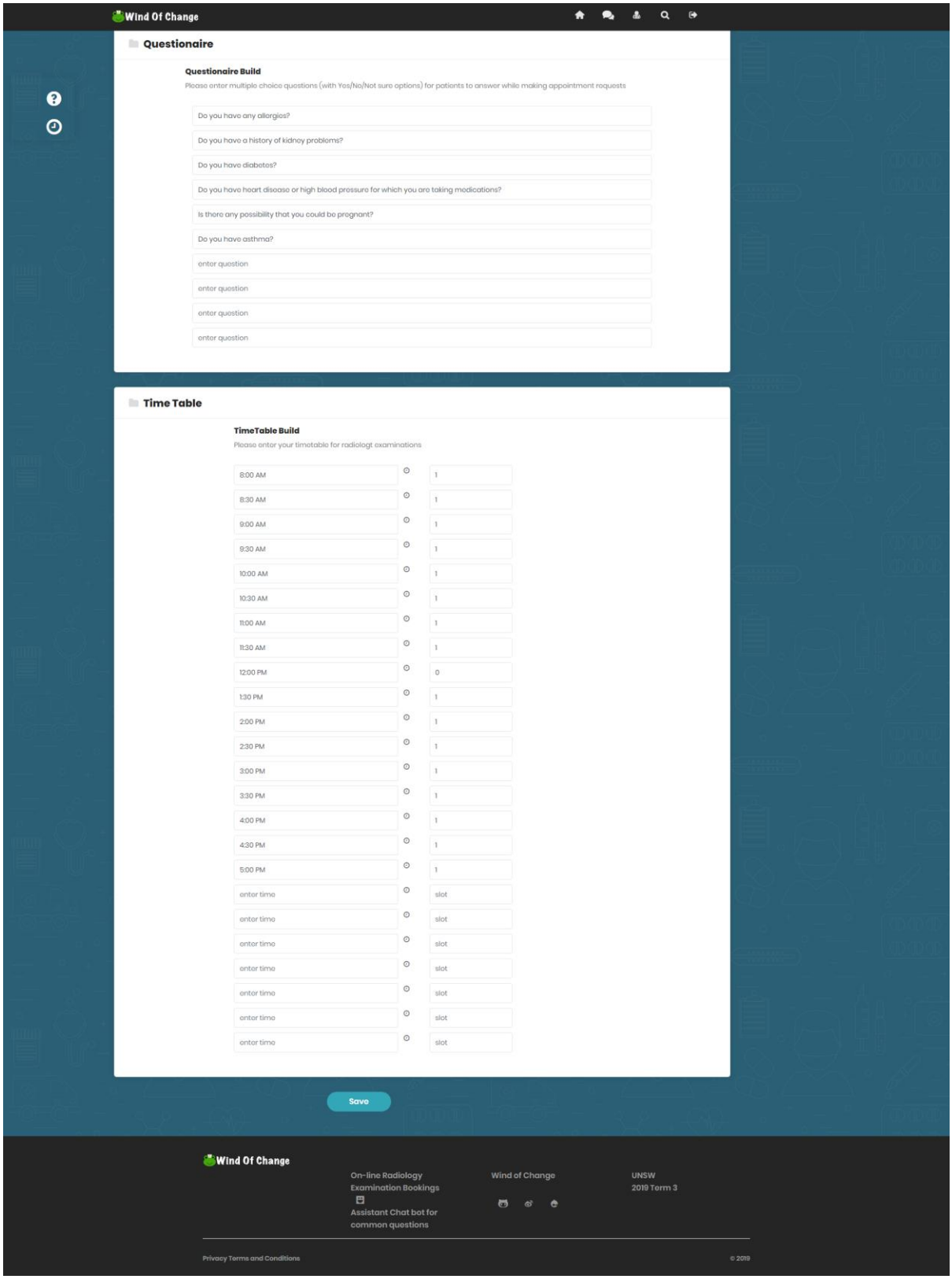


Figure 71 Questionnaire and Timetable Build

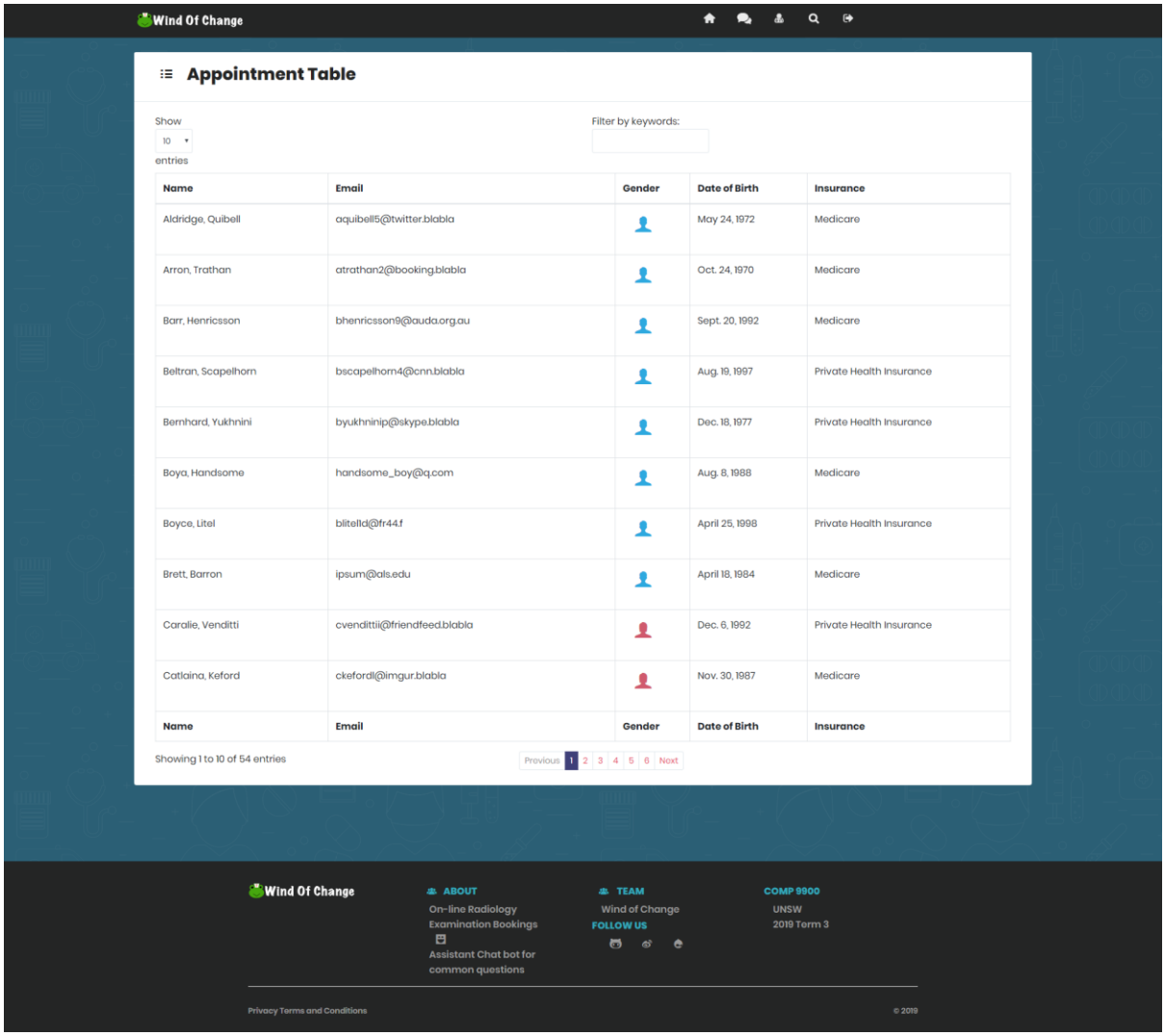


Figure 72 Patient Information look-up

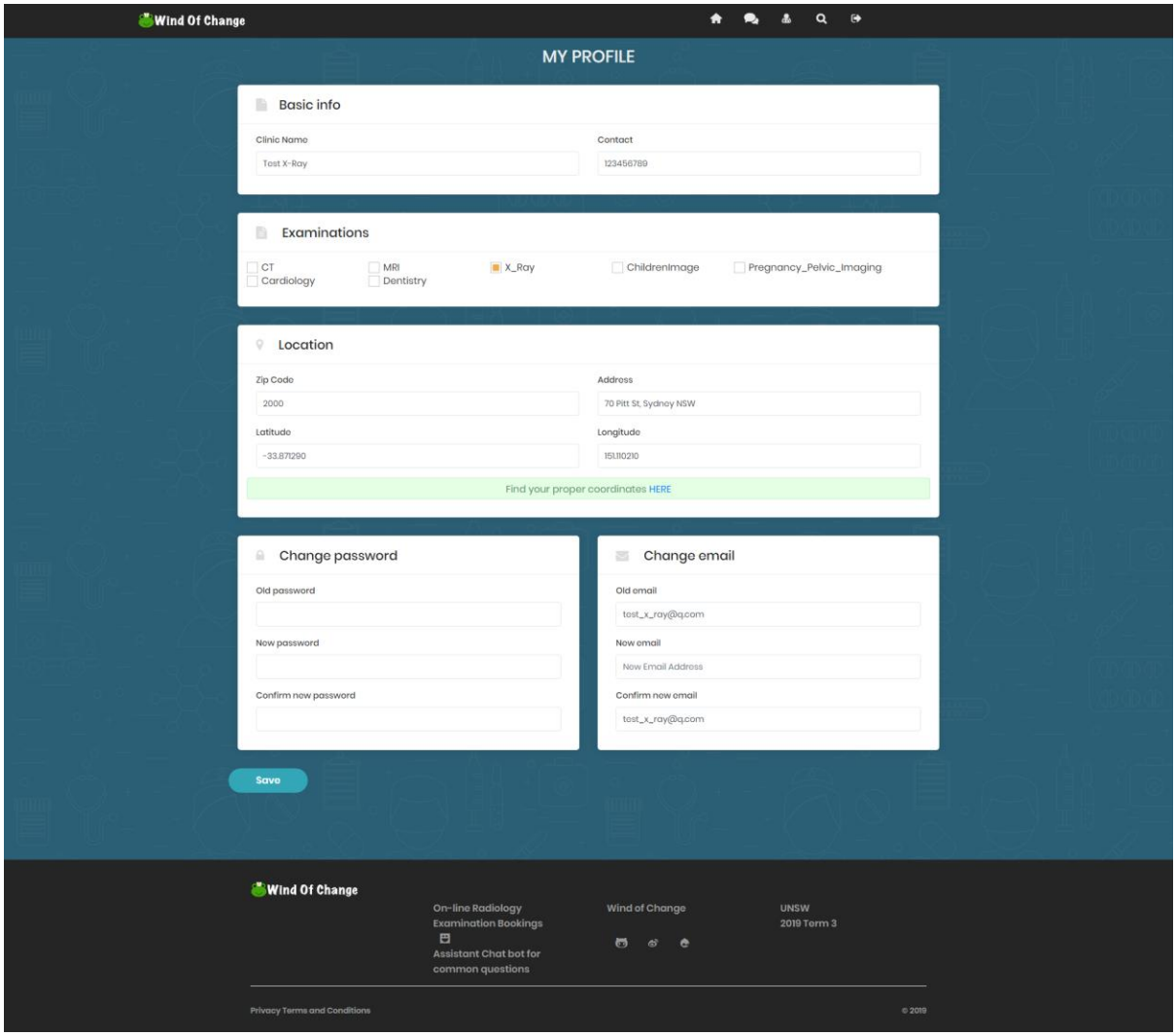


Figure 73 Clinic user profile

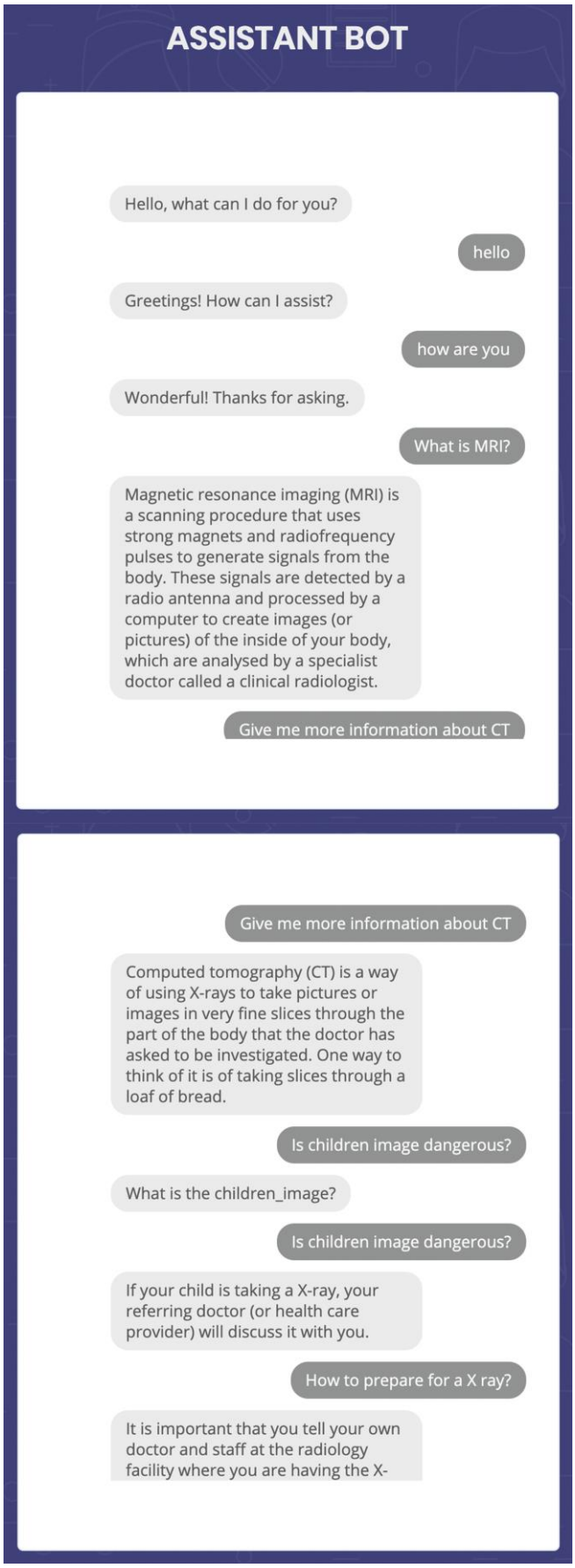
Appendix B – Colour Palette

COLOR PALETTE

HTML	Colorname	Hex Value
	Patient Background	#3f4079
	Clinic Background	#2B5F75
	Cancel	#e84a5f
	Pending	#F7D94C
	Approved	#348F34
	Requirement	#58B2DC
	Report	#DB4D6D
	Theme Color	#3f4079
	Link/ Highlight/ Button	#E16B8C

Figure 74 Project Colour Palette

Appendix C– Chatbot Screenshots



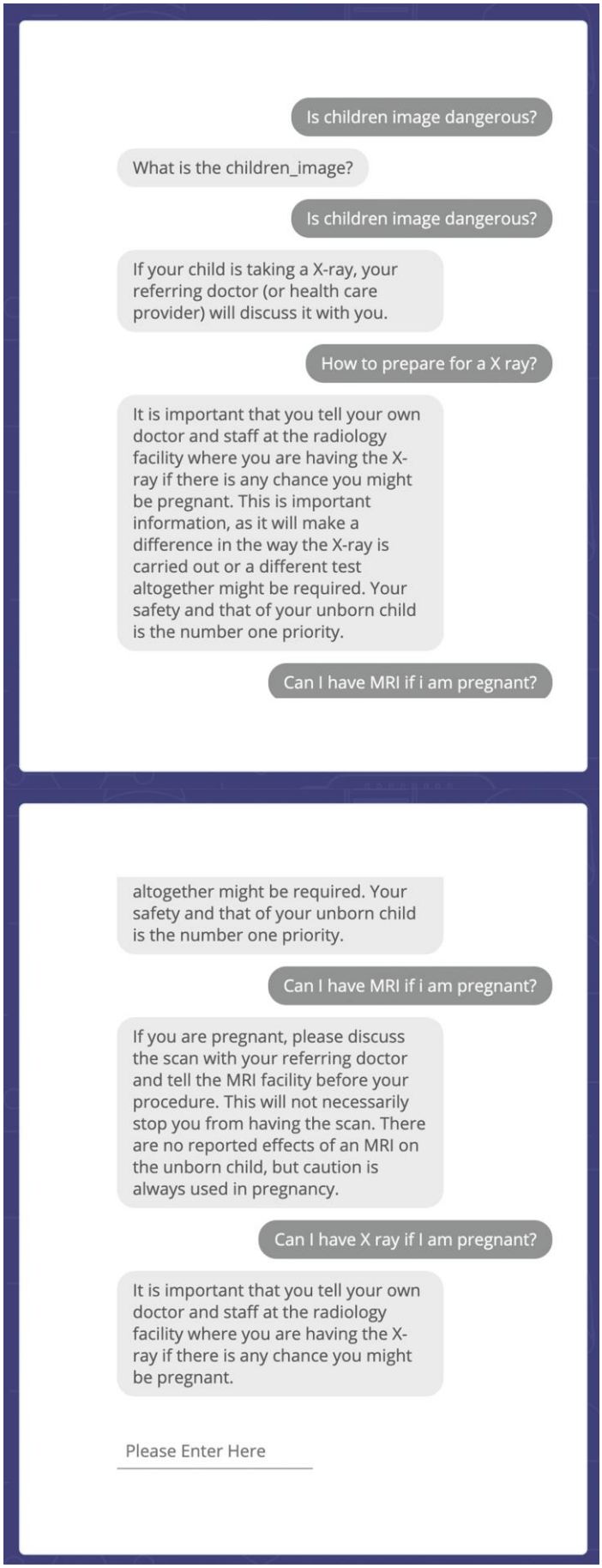


Figure 75 Chatbot example: Normal Chatting