Ministry of High Education,
High institute of Computer Science and Information Systems,
Culture and Science City, 6-October City

المعهد العالي لعلوم الحاسب ونظم المعلومات

Graduation Project:

Medical Chatbot

Assistant:

T.A. Walaa Zaghlol

Supervised by

Prof. Dr. Sami Abdel Moneim

Project No: Academic Year: 2021/2022

Ministry of high education,
Culture and science city at Oct 6,
The High institute of computer Science & information systems

المعهد العالي لعلوم الحاسب ونظم المعلومات

Graduation Project:

Medical Chatbot

Prepared by

٤٢٠٠٨ - ريموندا روماني سمير
٤٢٠١٨ - مصطفى جمال مصطفى
٤١٠٠٨ - أبوبكر مصطفى أحمد
٤١٠٢٣ - أحمد فرج المحمدي
٤١٠٣٥ - إسراء امجد سليمان
٤١٠٥٠ - آية إبراهيم الزاهي
٤١٠٧٤ - سندس طه مصطفى
٤١١٧٥ - يوسف يحي عبدالنبي

Supervised by

Prof. Dr. Sami Abdel Moneim
T.A. Walaa Zaghlol

Project No: Academic Year: 2021/2022

Acknowledgment

We would like to thank the Board of Education, administrators and support staff of The Higher Institute for Computer Science and Information Systems led by the collage Dean: Prof. Dr. Sami Abdel Moneim for their leadership and support to the students and the continued commitment to deliver quality instructions, as the strong leadership is the foundation of our institution success. This work could not have been done without you.

This book and the research behind it would not have been possible without the exceptional support from our project Supervisor: Prof. Dr. Sami Abdel Moneim who provided us with the space to be innovative but at the same time balancing that with guidance to keep the team on the Progress Schedule.

We also acknowledge Eng. Walaa Zaghloul, who has been providing us with supervision, through her guidance we have been able to complete the project. We would like her to know that we appreciate her for being our mentor.

We are also immensely grateful to anyone who offered a helping hand, and great thanks to the team members for their effort and willingness to learn.

Abstract

The project goal is to build a medical personal virtual assistant. This healthcare management system will assist people with giving medical care support regarding two domains; the Dentistry field and general Medicine and will work as a regular self-monitoring option.

Users can access the website's chatbot to navigate them to the General Medicine section to mention their symptoms and it would give them an accurate response of what they are suffering from, after analyzing the symptoms, they are provided with the diseases diagnosis and the right medical specialty and sub specialties their diseases belong to.

The website has a dental hygiene self-check section that scans teeth photos to help users identify if their teeth are healthy or not.

The users have the ability to book appointments if their case need to step into the Doctors' office.

Table of Contents

List of Figures

List of Abbreviations

Symbol	Description
AI	Artificial Intelligence
ML	Machine Learning
CNN	Convolutional Neural Network
CSV	Comma-Separated Values
DL	Deep Learning
TF	TensorFlow

CHAPTER ONE

INTRODUCTION

Chapter 1
Introduction

Health systems worldwide are in transition to utilize technological solutions to overcome many of the difficulties they face today. These difficulties are namely, the difference in access to medical care, the increase in costs no matter how simple the medical case, how busy the modern life has made people, getting used to the technological solutions and pinging what they're experiencing to get quicker hassle-free solutions and common fear of doctors when visiting them without having a clue about diagnosis and their related symptoms.

According to [Azza A El-Housseiny and Derwi(2014)] Dental fear has not only been linked to poor dental health in children but also persists across the lifespan, if unaddressed, and can continue to affect oral and systemic health. Other research [Shimmaa Moustafa(2015)] discussed how we could assess children's oral health risks, and their dental hygiene practices. The study focused on how adults have often already acquired dental anxiety from childhood.

1.1 A thesis statement

Bearing in mind all the listed above issues, Our offered solution looks convenient, MDBOT website works as a virtual personal medical assistant that performs various tasks. It offers the users the ability to select between two medical domains; dentistry and general medicine, and get the result(diagnosis) of the given symptoms immediately, offering the option of booking a doctor in the related medical field, or if the users are content with the given report, they needn't to book any appointments.

1.2 Proposed strategy

Overview of the structure running in the background

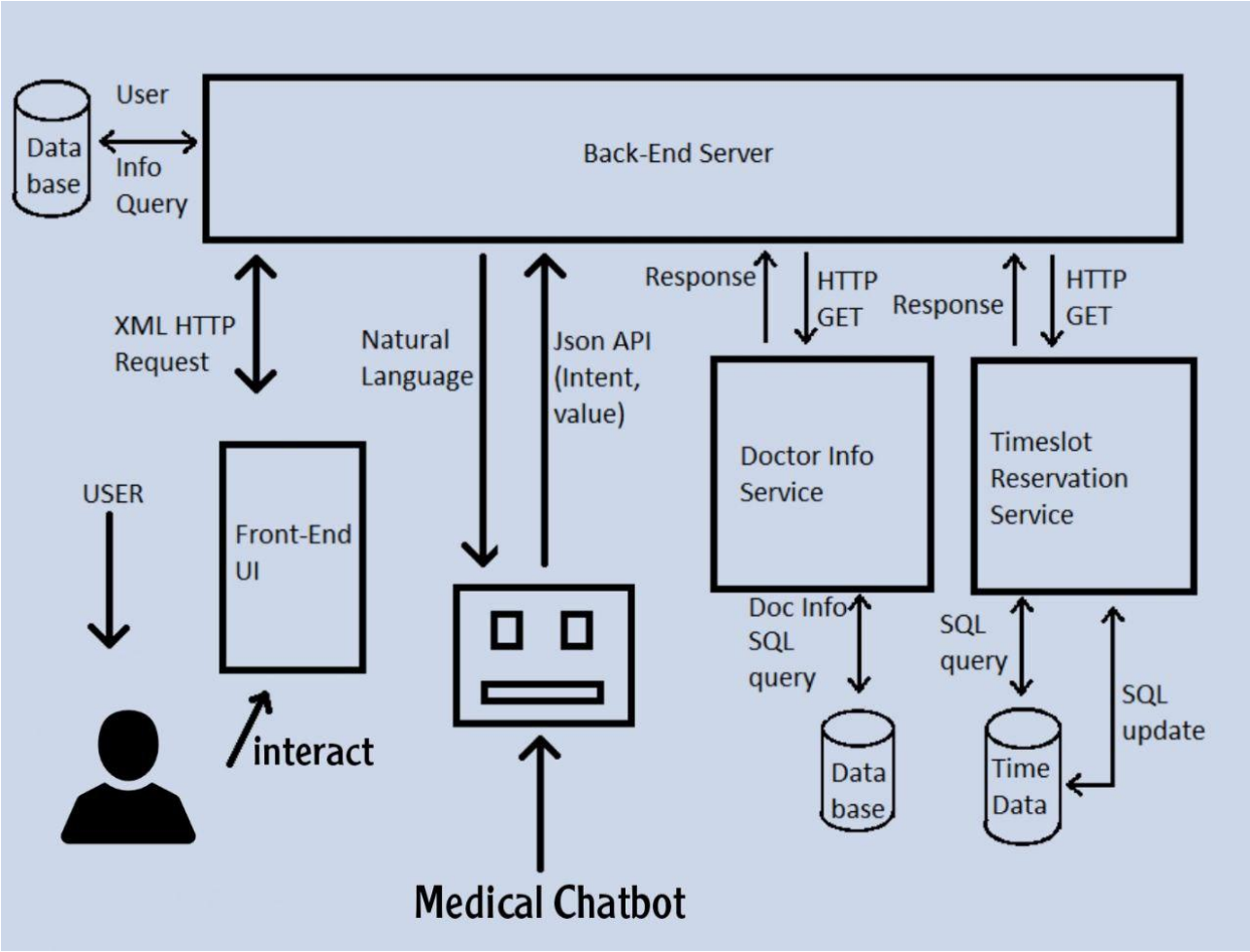


FIGURE 1.1: Structure of the microservices

1.3 Problem statement

There are deep problems that are worth our attention.

- Lack of personal hygiene
- Poor oral hygiene
- Improper Dental health
- Various Dental issues
- Need for Online booking
  The difficulty of making appointments offline.
- people Not having in depth medical knowledge
  Confusion and troublesome access to initial diagnosis in case of not having the right medical knowledge.

1.4 Types of Chatbots

Chatbots are a specific class of AI-based systems that has become popular in recent years. They are software-based systems designed to interact with humans via text-based natural language, They are the programs that interact with users utilizing natural language. The chatbot stores the data in the database to distinguish the keywords from the sentences and settle on a choice for the query and answers the query. Nowadays's [SFlora Amato(2018)] chatbots can be seen in every industry to guide the user as per their need. here are two types of chatbot:

- Rule based chatbots , also known as decision-tree bots, follow predefined paths. Think of them as a flowchart where users select an option and the chatbot responds with an appropriate answer. These can be as simple or as complex as needed.
- Conversational chatbots , or AI chatbots, are essentially like virtual assistants. They are more personalized than rule-based chatbots. They

use AI to understand the context and intent of a question to help them formulate appropriate responses.

1.5 Reasons why we need DeepLearning

In healthcare [Comendador(2015.)], Image classification is significant as we could analyse medical images and suggest whether they relate to a symptom of illness, even showing an evident disease or not. Image classification is the process where the computer has the ability to analyse images and identify the class or label each image falls under, to be more specific, the computer calculate the probability of each image being part of a class. So, relaying on the old methods of analyzing images such raw pixel data has become inefficient and it was replaced with deep learning.

Deep learning is a type of machine learning; a subset of (AI) that allows machines to learn from data. Deep learning involves the use of computer systems known as neural networks.

1.6 How a medical website could be a better solution

Our web application(MDBOT) enables users to log in, converse a Chatbot for easier access to our models, book appointments, and get a medical diagnosis report. With a smooth and user friendly UI, the user don't find it hard to explore all about the website's functionalities and how to access them.

CHAPTER TWO

THEORETICAL BACKGROUND AND TOOLS

Chapter 2
Theoretical background and tools

2.1 Existing system

Health organizations make significant investments in health applications to engage with patients for things like therapy, medication monitoring, symptom observation, diseases discovery, medical appointments booking etc. The results of those investments (outcomes) have been inconsistent. Research shows that for a health application to show potential in succeeding , it is necessary for a team of physicians to follow up with patients to ensure that they are regularly using the offered solution. This has an impact on the scalability of technology.

In today's medical websites [Divya(2017)], patients are required to complete a form with their personal information while using web-based customer service. Patients are reluctant to submit their personal information through Web-based customer support choices. Another issue with a web-based system without live chat is enraging customers with slow replies. The patient may need to make a visit to the hospital or the clinic in order to receive the information he is seeking if none of the aforementioned methods prove successful.

2.2 Proposed system

A medical virtual assistant with improved services, that is industry-specific, its main functionality is to lead users in exploring the website full functionalities, chat with patients to answer their questions, reduce time to search the web and the troublesome of translating the results not to mention the issue of actually understanding the meaning of each term.

The medical website that serves mainly Two medical fields:

- General Medicine
- Dentistry

General Medicine

Web application that has a Rule-based Chatbot in it, so it could be easier to explore the website functionalities, with a symptom-based model. It has a dedicated system for only solving problems regarding the physician role.
The Objectives:

- Reliable disease prediction based on symptoms.
- Based on the symptoms, the system will suggest doctors in the matched specialization.
- Scheduling a medical appointment

The Dentistry

The dental attendance isn't just the chat bot. It does more than simply sitting on the website,twenty-four hours a day.
It's the concept of: when you need it, it is there.
The chatbot within it helps in:

- organizing the patient flow
- booking appointments
- reliable informative tips

The Objectives:

- To develop a free dental consultation and self-check for a better dental health management service that can be accessed online at all time via MDBOT website.

- To design a chatbot system that enable users to communicate with machine like a real-life conversation to assist the user in navigating to the medical domains and the available features.

- To develop a web-based medical assistant system especially for oral and dental health, and to provides comprehensive diagnosis of diseases and management of medical conditions for adults of all ages.

- Reduce dental fear amongst people.

2.3 Software Development tools used

Python: is an interpreted high-level general-purpose programming language, it was chosen for various reasons, as the language makes sense of the data by working with Artificial Intelligence and Machine Learning in healthcare. The dynamic language offers wide variety of libraries and modules that support ML, DL and Image-based diagnostics.

FIGURE 2.1: python

Flask: The best Python-based microframework for web application. It was mainly written by and for python, that's why it's one of the most suitable technologies to run a python based web application with it. Flask is based on two key components: WSGI toolkit and Jinja2 template engine. WSGI is a specification for web applications and Jinja2 renders web pages.

FIGURE 2.2: flask

TensorFlow: is among the most popular frameworks when it comes to DL. .Since the the base language for the framework is Python, it's extremely convenient in this project. TensorFlow is a toolkit that integrates data flow and differentiable programming to handle various tasks related to DNN training and inference.

FIGURE 2.3: TF

Keras: is a simple, Open Source NN library and built in Python and consistent high-level APIs and follows best practices to reduce the cognitive load for the users. Keras works as a wrapper to TensorFlow framework.

FIGURE 2.4: Keras

Werkzeug: as a final output, it's a comprehensive WSGI compatible web application library, at its core, it consists of a collection of libraries. The tool provides a set of utilities for creating a Python app that can communicate with a WSGI server. It provides main functionalities such: Request processing, Response handling and URL routing.

FIGURE 2.5: Werkzeug

HTML: (Hypertext markup language)is extensively utilized for creating web applications. The building blocks of any HTML pages are HTML elements and the creation of a structured document is done with the help of structural-semantic text like heading, list, link etc. The content of the HTML templates is going to be displayed in the browser.

FIGURE 2.6: HTML

CSS: (Cascading Style Sheets) is a design language that enhances the aesthetic appeal of a website above plain pages. It controls visual structure and layout, whereas HTML primarily controls text content. HTML is a markup language,

FIGURE 2.7: CSS

whereas CSS is a language for creating style sheets.

JS: (Javascript) is a scripting client-side language used to create dynamic and interactive web content. It allows us to add animations and hover effects. JavaScript is the only language that runs natively in web browsers.

FIGURE 2.8: JS

SQLite: is a open-source software library and a cross-platform DBMS, that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is one of the fastest-growing database engines around. SQLite engine is not a standalone process like other databases.

FIGURE 2.9: SQLite

DB Browser for SQLite: high quality, visual, open source tool to create, design, and edit database files compatible with SQLite. The tool gives programmers the ability to create, search, and edit databases. DB4S uses a familiar spreadsheet-like interface but at the same time, so it's up the user to write complicated SQL commands or not.

FIGURE 2.10: DB Browser for SQLite

CHAPTER THREE

ANALYSIS AND DESIGN

Chapter 3
Analysis and Design

3.1 System Planning

A schedule and a resource plan to ensure project success on time.

Medical Chatbot Project Plan (7 months)			
This project plan provides the work breakdown structure that our chatbot-project team members use to deliver it. We add, adjust or delete tasks depending on the needs of the project's supervisors.			
Task	**Goal**	**Deliverable**	**Notes**
Project start-up & management (December)			
Kick-off meeting **(21/12/2021)**	Start the project, meet the teaching assistant, discuss project plan.	-Draw the initial Activity diagram. -Determine the time of the online meeting.	
Define team	Gain a basic understanding of who will be doing what so that the project can get started	Youssef & Remonda (Front-end). Esraa & Sondos (SQL DBA). Aya, Bakr (AI & Python Developer). Ahmed (AI). Mostafa (Design).	
Create project hub for core team	Establish the method to be used for project communication and sharing deliverables	Whatsapp groups	
Hold weekly status meetings & prepare weekly status reports	Keep all team members and supervisors informed of progress, next steps, critical issues and remaining	-State the final output in textual form via Whatsapp. -Determine the new tasks.	

FIGURE 3.1: Timeline[Project Management]

Task	Goal	Deliverable	Notes
Planning & Analysis Phase (2 Months)			
Read the Overview provided by the collage to understand the project	Understand needs and priorities of the Chatbot.	-Project outlines, scope, basic features and technologies.	
Determine project objectives through group meetings	Identify and prioritize project objectives.	-The chatbot is informative for all the sides. - Chat with patients. - Match patients with doctors. - provide time slots, schedule and reschedule, delete appointments. - integrate with Google calendar to notify users.	
Translate objectives into detailed Activity diagram	Capture objectives in a Who/Why/What format that describes how the chatbot will deliver on the objectives.	The final Activity Diagram that will be used as a baseline and a backbone for the other diagrams.	
Formalize decisionmaking for the diagrams and the designs	Determine who is responsible for each of the chatbot's diagrams and the process of designing.	- Ahmed (Sequence diagram). - Aya & Bakr **(**Activity, Use case, Use case tables, chatbot flow**)** - Esraa & Sondos (Class & Block diagram) - Youssef & Remonda (Architecture diagram) - Mostafa (Designs)	
Teaching Assistant feedback	Track the Analysis and designing processes of the chatbot and how well the team members are delivering the project objectives.	Edits, comments and suggested changes on the diagrams and designs.	

FIGURE 3.2: Timeline[Planning and Analysis Phase]

Task	Goal	Deliverable	Notes
Content Phase (March-June)			
Make progress with the chatbot interactions	Clearly document the types of interactions the chatbot supports and the way the interactions feel in details	A document with a series of supported chatbot interactions based on the initial chatbot flows that were delivered in the previous phase.	**MILESTONE. This document varies widely depending on the chatbot objectives. Examples: - chatbot may contain a list of question and answer pairs. -A document that demonstrates the different ways a user might interact with the chatbot. -A script for the chatbot that contains a predicted dialog from the user.**
Gather content and data	Gather all the content and data that will be delivered via the chatbot	Repository of content that will be used to populate chatbot. Ex: upload all the used files here to github.	Depending on chatbot objectives and the available data, we will determine how this step will be handled.
Define Medical chatbot personality	Define the textual tone that the chatbot will take	A few sentences describing the personality of the chatbot. Examples of words, phrases and punctuation characteristic of the personality	Keep in mind to relate the personality of the chatbot to the characteristics of the medical representative.
Choose the name	A name that can be used in the designs and in sample scripts	An agreed-upon name for the medical chatbot	

FIGURE 3.3: Timeline[Content Phase]

Develop non-core content including failures	Support successful conversation that surrounds core content, including pleasantries, edge-cases, and failures. On failure, Ask for feedback and then redirect user back to the start of the chat(restart the conversation)	Document with responses for edge-cases, failures	
Develop language model	For our medical chatbot that support natural language, develop initial language model so content population(mainly the knowledge domain) can start to form	Initial language model with intents, entities and sample utterances	Keep in mind not to maximize the use of buttons, as the chatbots that are completely button-based and do not require a language model, greatly reduce deployment time.
Edit content for chatbot use	Edit content so it is appropriate for delivery via chatbot, shorten it and add personality and keep the content/dataset in-review so it could be open for improvements.	-Edited content	
Populate content	Get content into the chatbot	Bot responds with content	
Improve the design icon and logotype	Enhance the visual identity for the chatbot that matches the medical field	The evolved designs, Icons and logotype for the medical chatbot	
Put the foundation stone of the graduation project book	-Get hands-on other graduation projects books	-Baseline graduation project book	
Follow-up meetings	-Meet the Teaching Assistant once a week. -Gather with the prof. dr. Samy once a month.		

FIGURE 3.4: Timeline[Content Phase]

Task	Goal	Deliverable	Notes
Technology Phase (March-June)			
Identify and describe new functionality necessary to make chatbot interactions come to life	Provide dev team with functional specifications so they can estimate and start on development	Series of tasks for development team to complete	
Develop custom integrations and functionality	Get chatbot to work as specified	Custom features or integrations(Google Calender)	
Configure bot to work on the website	Get a functioning working chatbot on the website	The described Website and chatbot working together	
Apply the final agreed-to look on the website	Implement those designs on the website	Fully well-designed website including the chatbot	
Demo Chatbot Alpha	Provide the first look	Controlled demo	
Follow-up meetings	-Meet the Teaching Assistant once a week. -Gather with the prof. dr. Samy once a month.		

FIGURE 3.5: Timeline[Technology]

Task	Goal	Deliverable	Notes
Launch Phase (July)			
Plan the launching	Plan activities to support the launching of the website including the chatbot	Initial launching of website and the chatbot	
Conduct a review	Review the website and ensure the chatbot delivers accurate responses.	Edited Website/chatbot	
Test & revise content	Adjust and improve bot interaction based on core team feedback	The Edited Website and chatbot after testing	Core team available to test and report issues/bugs.
Launch!	The chatbot is ready!	Live chatbot accessible by all users	
Revise content	Maintain a high-performing chatbot	Continuously improving chatbot through reviewing the logs	
Follow-up meetings	-Meet the Teaching Assistant once a week. -Gather with the prof. dr. Samy **(to be determined)**		

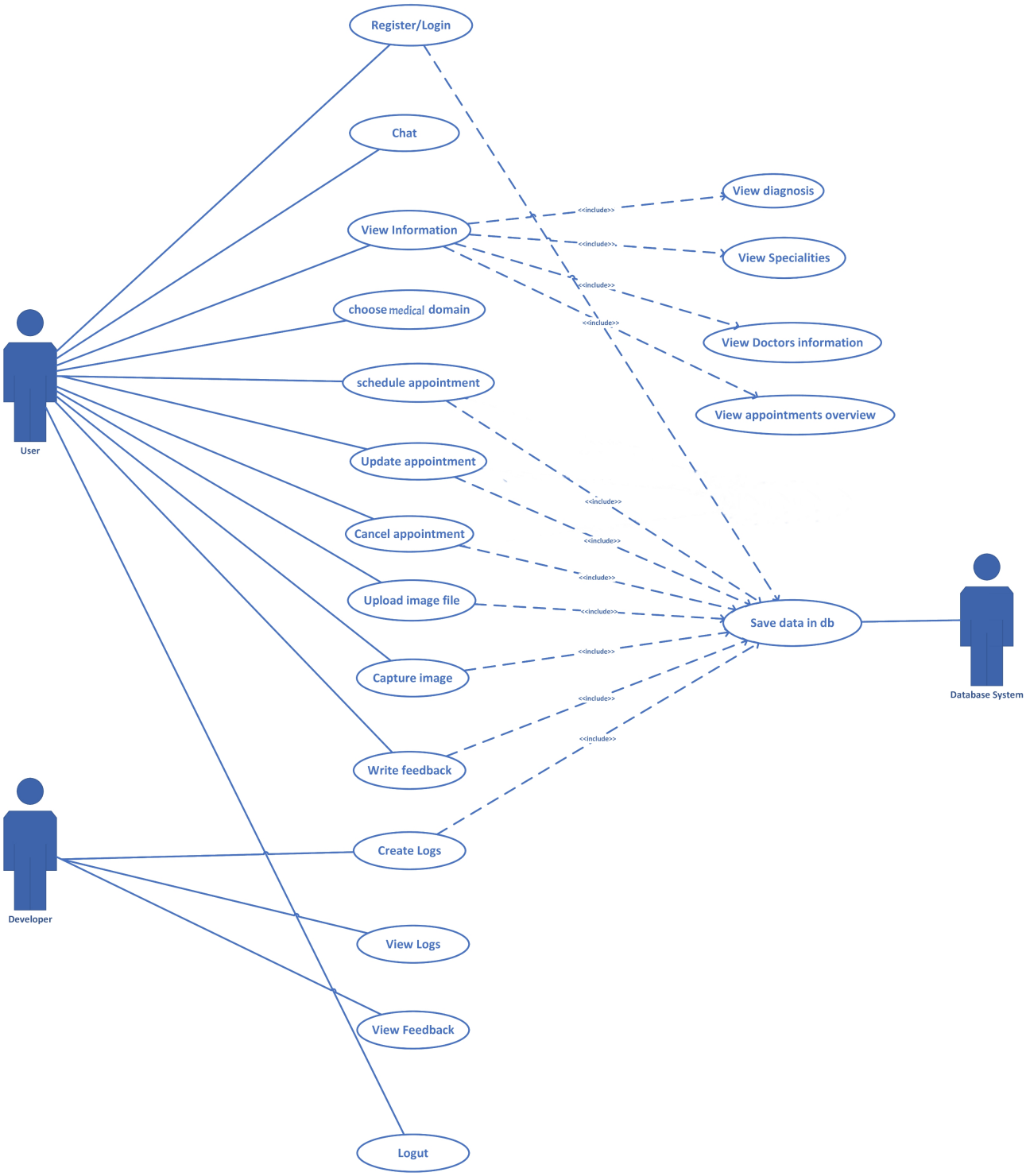
FIGURE 3.6: Timeline[Launch]

Task	Goal	Deliverable	Notes
Project wrap-up (July)			
Prepare documentation or hold training session(s)	-Final review over the graduation project book. -Training	-A ready-to-deliver graduation project book.	
Hand-off and wrap-up meeting	Pronounce the project complete. Identify what went well, what went poorly.	Project debrief report	
Follow-up meetings	-Meet the Teaching Assistant once a week. -Gather with the prof. dr. Samy **(to be determined)**		

FIGURE 3.7: Timeline[Project wrap-up]

3.2 System Analysis

3.2.1 Use Case Diagram



3.2.2 Use Cases Tables Description

Use case	**Register/Login**
Description	The user Login to his account or creates a new account
Actors	user
Included use cases	Save data in database
Preconditions	home page loaded with two options (Register or Login). user have a valid account.
Postconditions	app displays the Home page
Main flow	1. user selects Register or Login 2. app prompts user to enter Information 3. user enter his Information 4. app verifies data and Login to the user's account
Alternative flows	• User enter invalid Register/Login information ◦ app displays an Error message ◦ the user reenters the information ◦ app verifies the new information

Use case	**Save data in database**
Description	The Database System saves user's info and the Logs to use them in validation/verification or processing
Actors	Database System
Included use cases	None
Preconditions	database System is ready to receive a new data
Postconditions	the database is updated with the new information
Main flow	1. database system receives a new data 2. database system verifies the data 3. database system update the database with the new information
Alternative flows	None

Use case	**chat**
Description	The user chats with the chat-bot system through a textual form.
Actors	User
Included use cases	None
Preconditions	user has selected to chat with the chat-bot
Postconditions	bot responses to the user's inputs
Main flow	1. user chat with the chat-bot 2. bot displays messages with appropriate response to the user's inputs
Alternative flows	None

Use case	View Information
Description	The user views different information that the website presents according to the user's interaction.
Actors	User
Included use cases	view diagnosis view Specialties view Doctors information view appointments overview
Preconditions	- website presents Doctors Information - website receives Symptoms - website presents Diagnosis - chat display Specialties
Postconditions	- load Information in response to the user's query
Main flow	1. user interacts with the website and asks for Diagnosis, Doctors information, or enter Symptoms 2. website displays the right Information that the User has queered
Alternative flows	None

Use case	Choose medical domain
Description	The user chooses a medical domain
Actors	user
Included use cases	None
Preconditions	chatbot page loaded with two messages/options (Looking for a dentist/ Not sure where to go).
Postconditions	the app displays medical domain options.
Main flow	1. user chooses an option 2. the app forwards the user to the requested medical domain.
Alternative flows	None

Use case	Schedule appointment
Description	The user book an appointment.
Actors	User, Database System
Included use cases	Save data in database
Preconditions	website lists different time slots for the selected doctor
Postconditions	app book the appointment
Main flow	1. user select a doctor 2. user schedule an appointment 3. database system saves the appointment info 4. app book the appointment
Alternative flows	• update the appointment • cancel the appointment

Use case	**Update appointment**
Description	The user updates the appointment information that has been scheduled.
Actors	User, Database System
Included use cases	Save data in database
Preconditions	website displays the selected appointment information
Postconditions	app update the appointment
Main flow	1. user select an appointment 2. website displays appointment information 3. user updates the appointment 4. database system update the appointment info 5. app update the appointment
Alternative flows	• cancel the appointment

Use case	**Cancel appointment**
Description	The user cancels the appointment that has been scheduled.
Actors	User, Database System
Included use cases	Save data in database
Preconditions	website displays the selected appointment information
Postconditions	app delete the appointment
Main flow	1. user select an appointment 2. website displays the appointment information with an option for the user to cancel 3. user cancels the appointment 4. database system deletes the appointment
Alternative flows	None

Use case	**Upload image file**
Description	The user upload image files for dental diagnosis
Actors	User
Included use cases	None
Preconditions	- user allow access to the files - user has clear images for diagnosis
Postconditions	- website presents the potential diagnosis based on the image file
Main flow	1. website asks the user for images for diagnosis 2. user selects to upload image file 3. website displays Diagnosis
Alternative flows	• Capture image files

Use case	**Capture image file**
Description	The user capture image files for dental diagnosis
Actors	User
Included use cases	None
Preconditions	- user allow access to the camera - user captures clear images for diagnosis
Postconditions	- website presents the potential diagnosis based on the image file
Main flow	1. website asks the user for images for diagnosis 2. user selects to captures image file 3. website displays Diagnosis
Alternative flows	• Upload image files

Use case	**Write feedback**
Description	The user writes feedback about his experience with the app which can be used for future improvements
Actors	User, Database System
Included use cases	None
Preconditions	- website has displayed the diagnosis report - user finds the report irrelevant to dentistry
Postconditions	- website learns more and the app make future improvements based on the feedback
Main flow	1. user view report and find it not dentistry related 2. website asks for feedback 3. user enters feedback 4. database system saves the feedback
Alternative flows	None

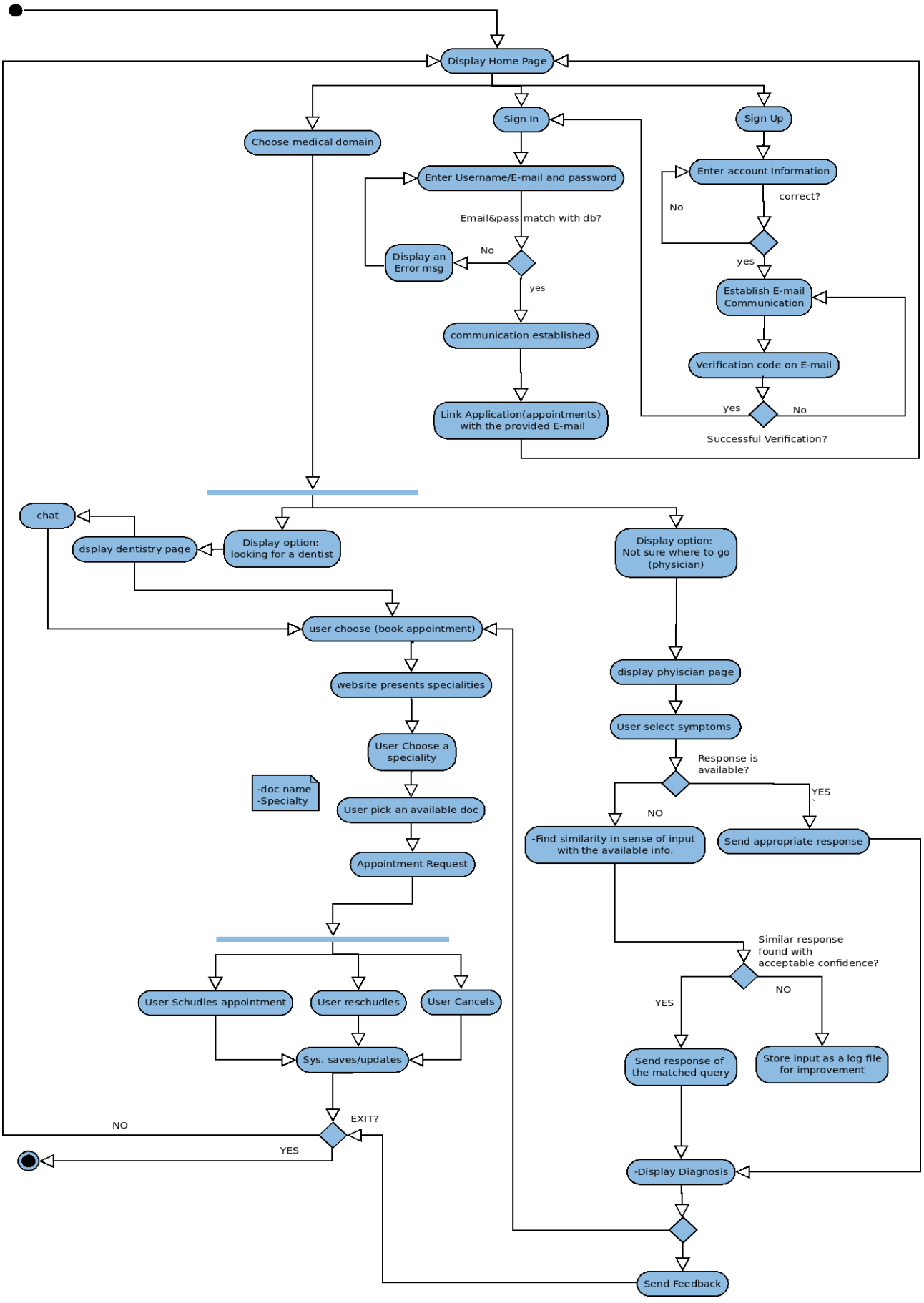
Use case	**Create Logs**
Description	system creates the logs file
Actors	Developer
Included use cases	None
Preconditions	developer adjust the system
Postconditions	logs are created and saved
Main flow	the system makes logs file automatically
Alternative flows	None

Use case	View Logs
Description	Developer view the saved logs
Actors	Developer
Included use cases	None
Preconditions	Already created Logs
Postconditions	Developer Analyze Logs
Main flow	Logs are viewed
Alternative flows	None

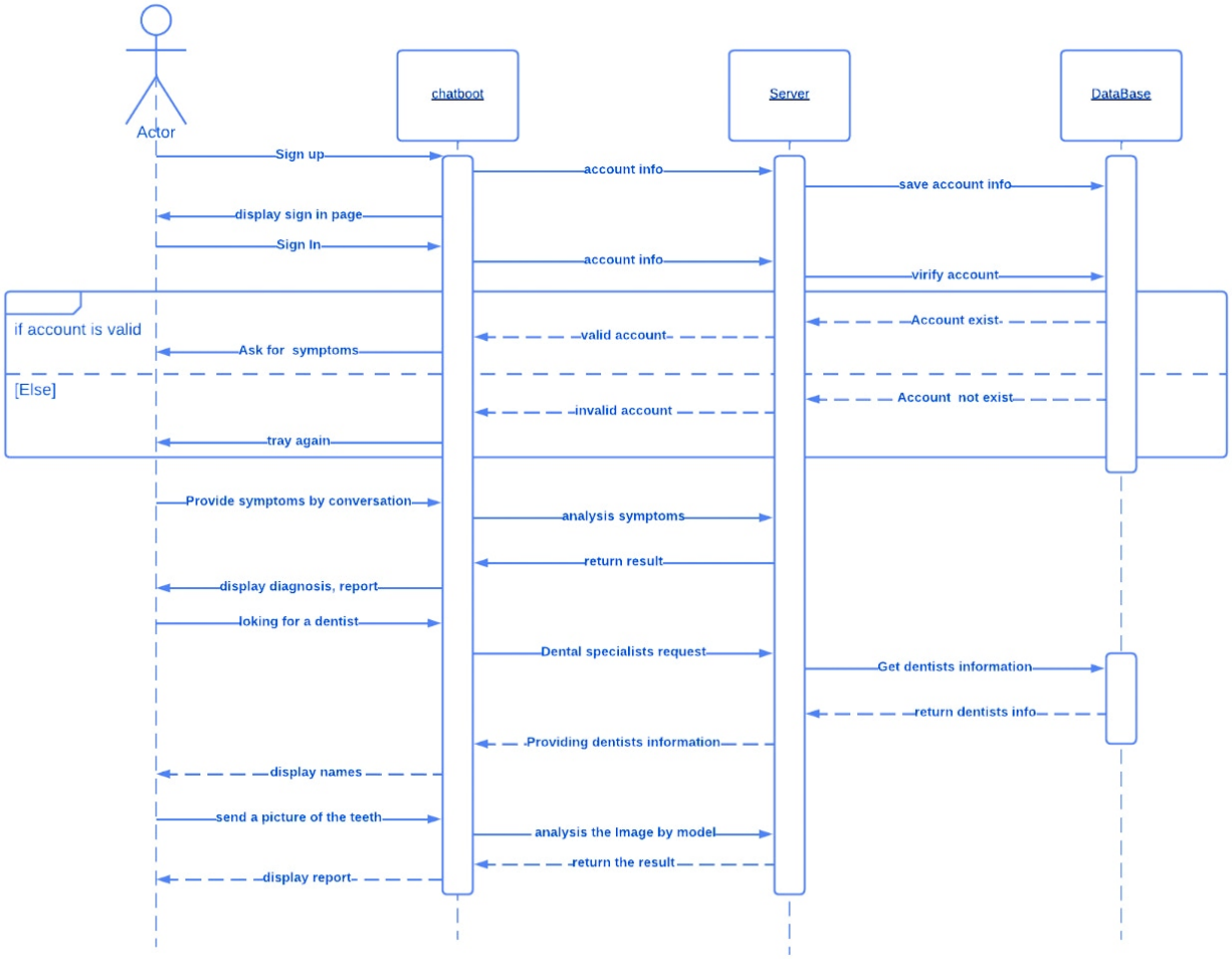
Use case	View Feedback
Description	The developer view the user's feedback to maintains the app according to the user experience
Actors	Developer
Included use cases	None
Preconditions	user left feedback after using the app
Postconditions	developer develops the app in the future according to the feedback
Main flow	1. user has written a feedback 2. developer view the feedback 3. developer update the app in response to user's feedback
Alternative flows	None

Use case	Logout
Description	The user logout from his account
Actors	user
Included use cases	None
Preconditions	user has logged in to the app with a valid account
Postconditions	the app logs out the user's account and displays the home page
Main flow	1. user open the account information 2. user select to logout 3. the app logs out the user's account 4. the app returns the user to the home page
Alternative flows	None

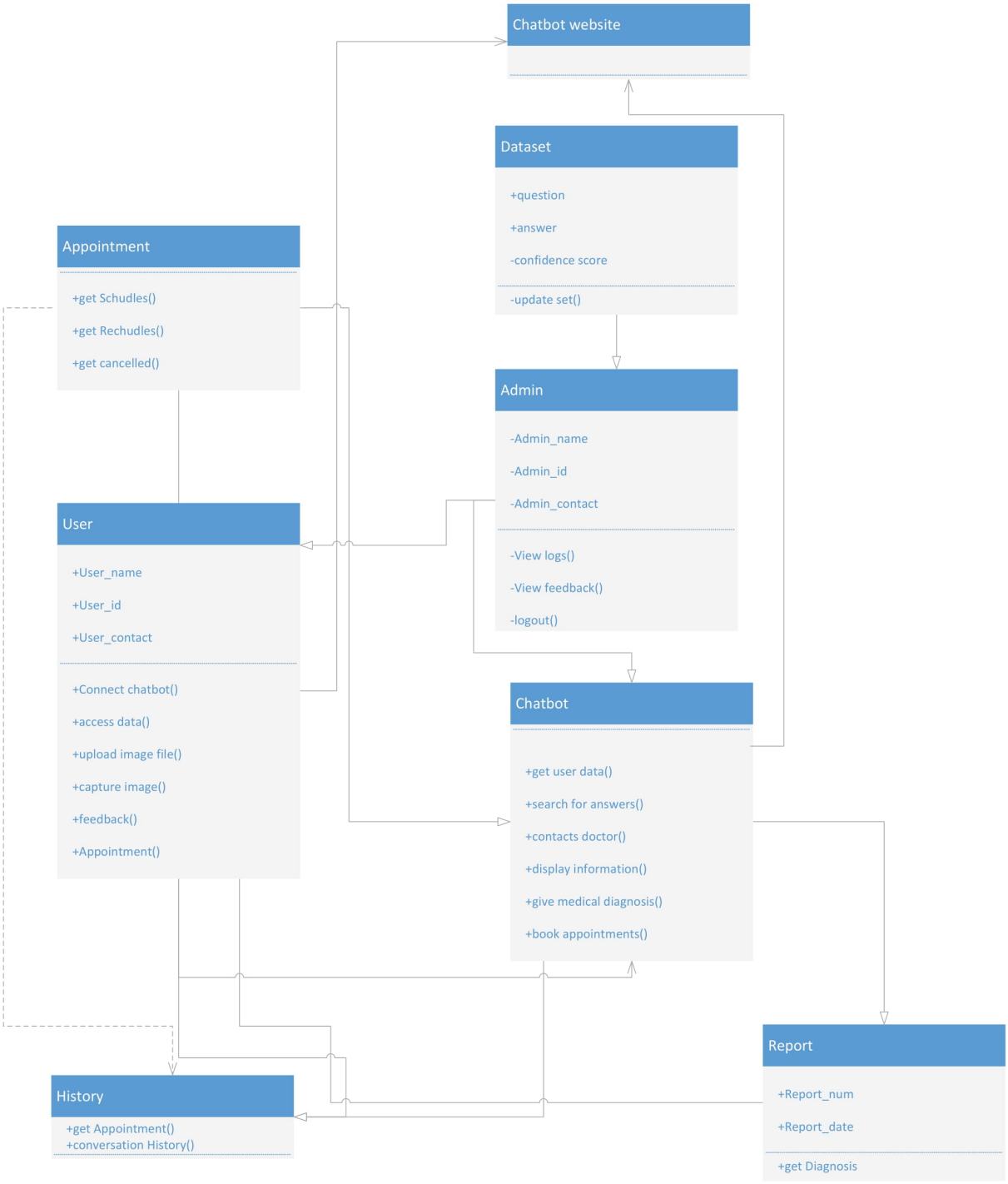
3.2.3 Activity Diagram



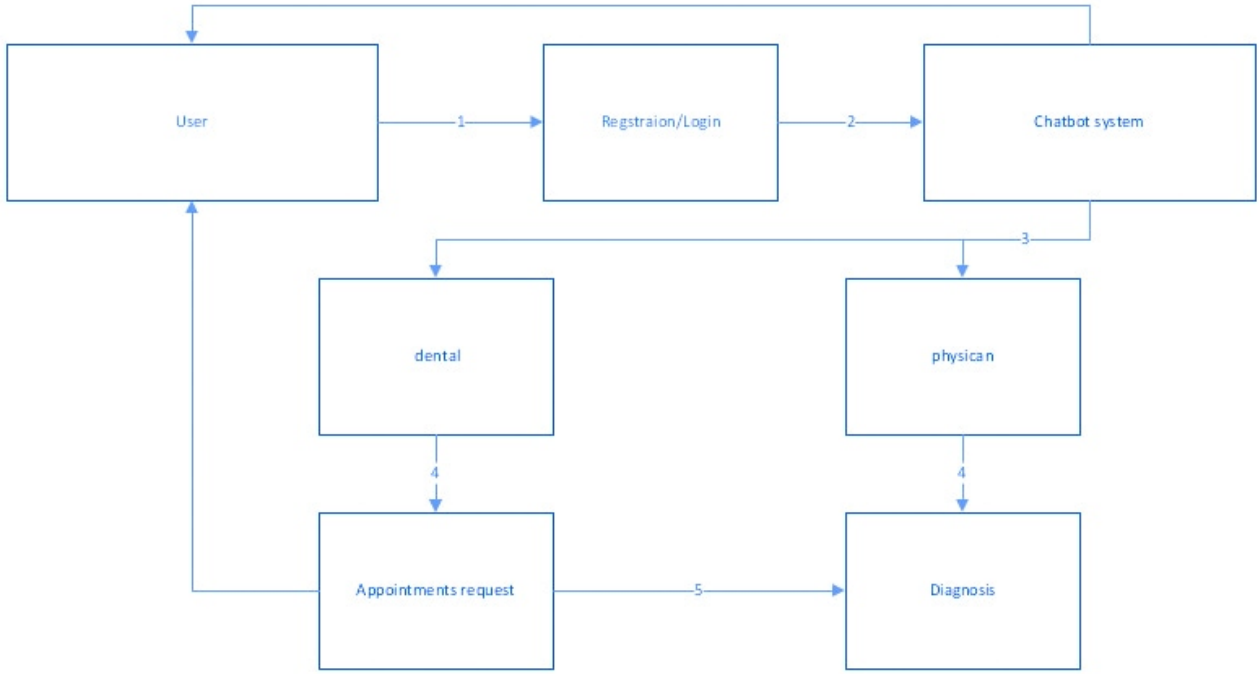
3.2.4 Sequence Diagram



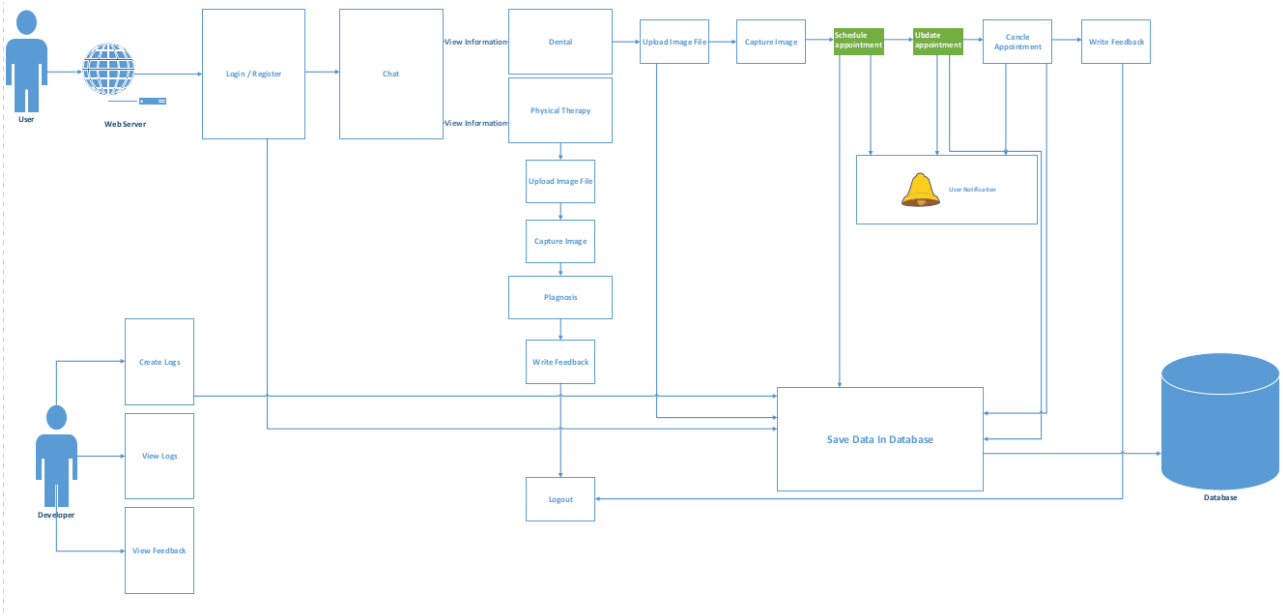
3.2.5 Class Diagram



3.2.6 Block Diagram



3.2.7 Architectural diagram

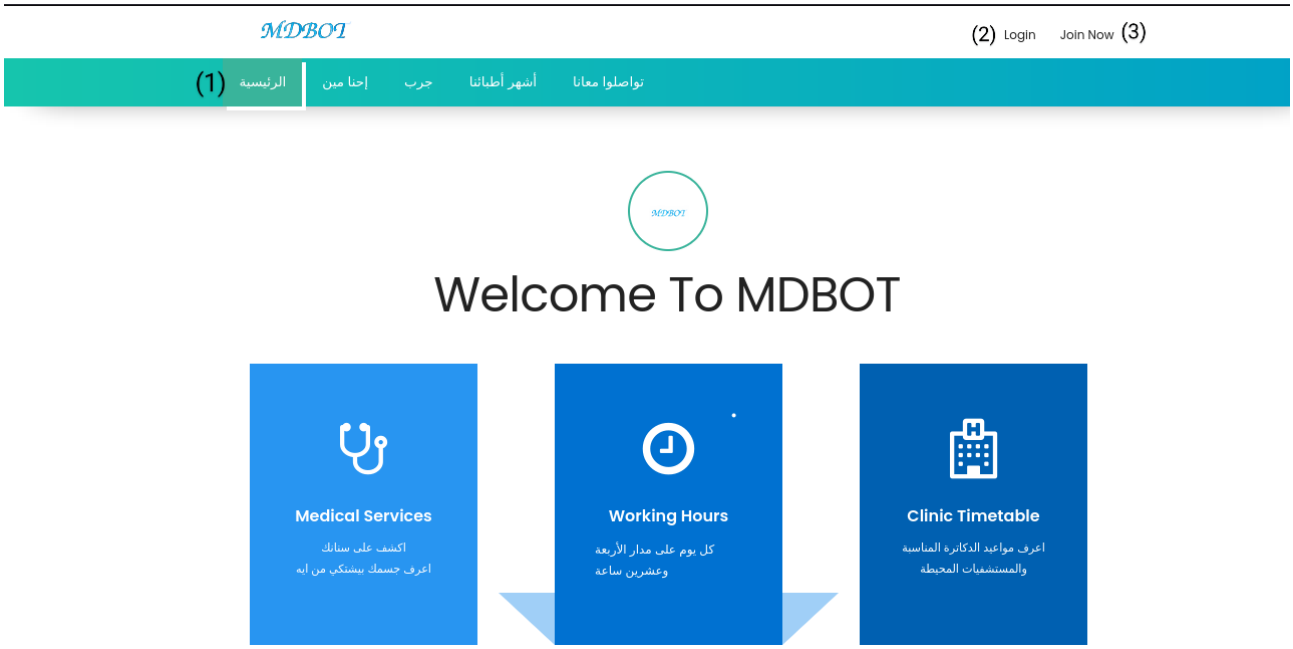


3.3 System Design

Engagement is a key concept in the design of any online application

- **Front Page**

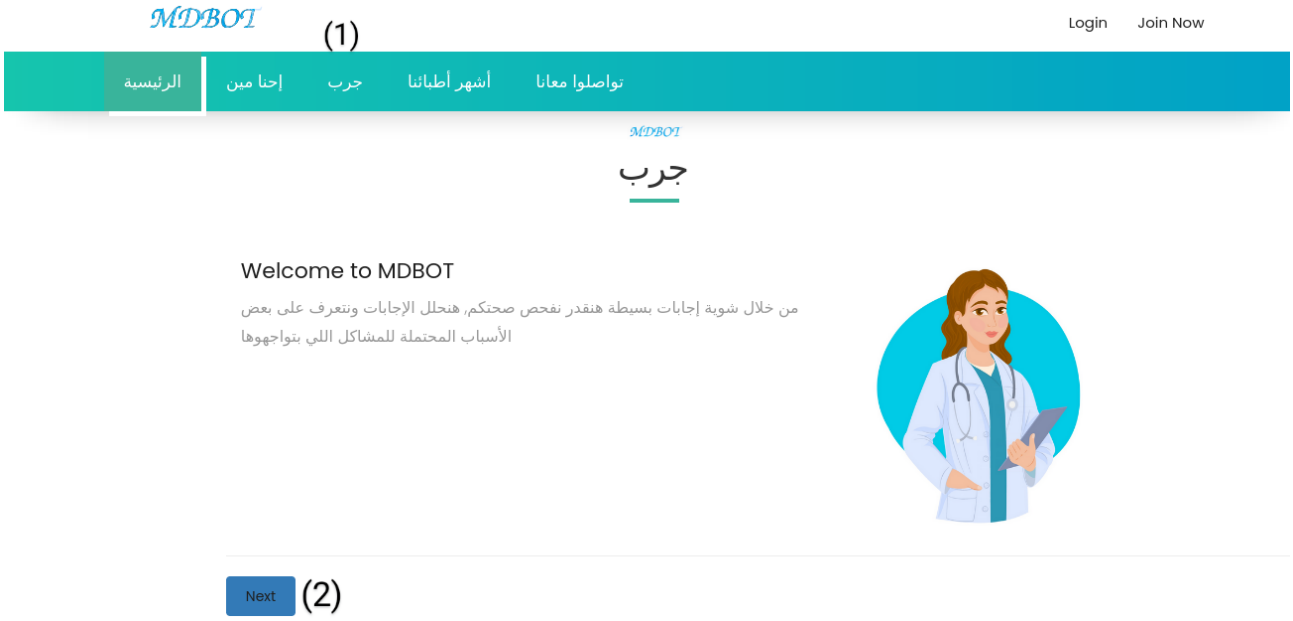
  The homepage and entry point to the website



(1) **home button**: A redirect to the main website page.

(2) **Login button**: A redirect to the login page to enable storing users' session data.

(3) **Registration button**: A redirect to the sign-up page to enable saving users' profile data by independently registering to the system.

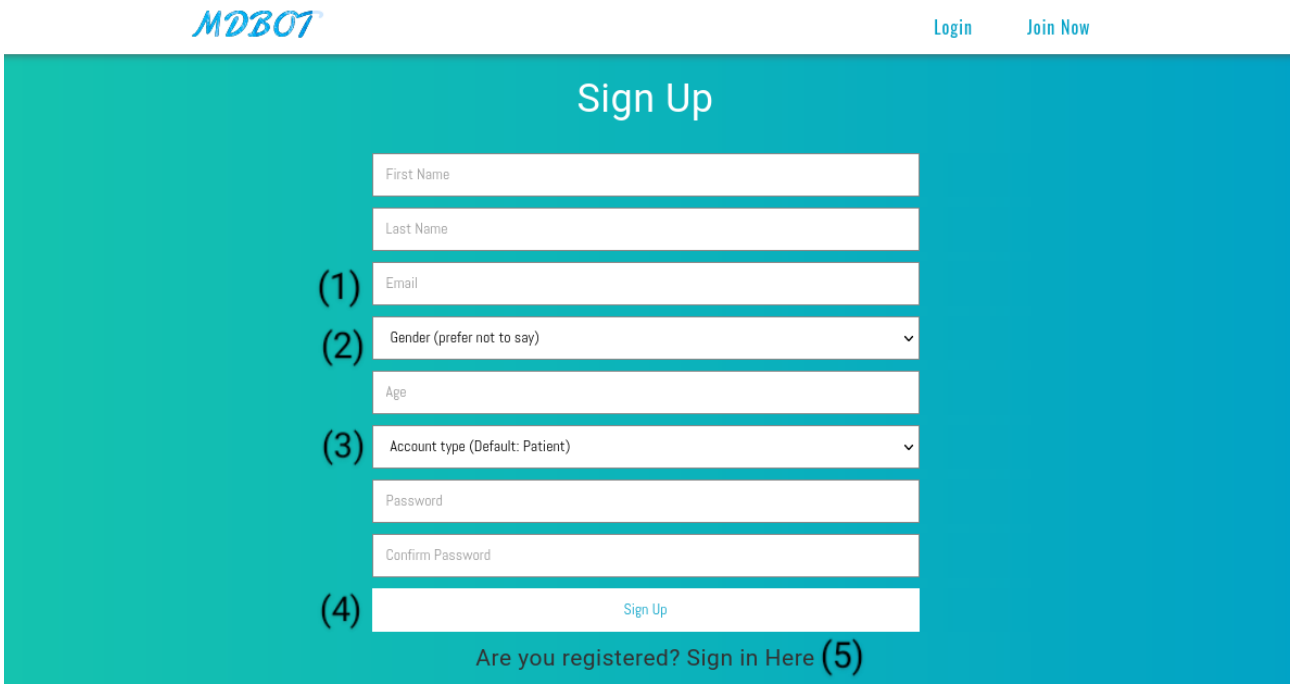


(1) جرب:A redirect to the entryway of MDBOT on the homepage to have an overview about the main functionalities.

(2) **Next button**: A redirect to the rule-based chatbot to choose a medical domain.

- **Register and Login Pages**

  Registration Form that has a list of fields that a user will input data into.



(1) **Email**: Users enter Registration emails.

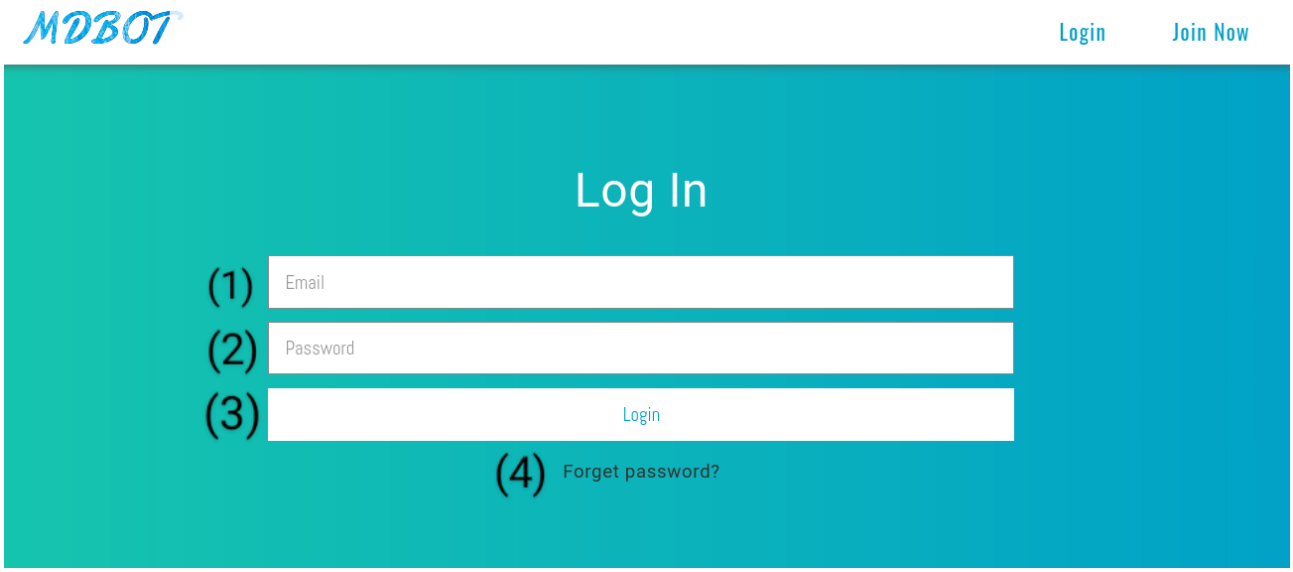
(2) **Gender**: Users enter their genders. It is needed when picking the symptoms from the gender-based model in the General Medicine section.

(3) **Account Type**: Users are either patients or doctors, each have their own functionalities and access permissions in the website.

(4) **Sign up button**: A button to accept user's input and write a new record out to the database.

(5) **Sign in button**: A redirect button to the sign in page in case user was already registered.

Login form that utilizes the credentials of users, in order to authenticate their access.



(1) **Email**: Users enter their registered emails, and then the system validates if the email address is in Database or not.

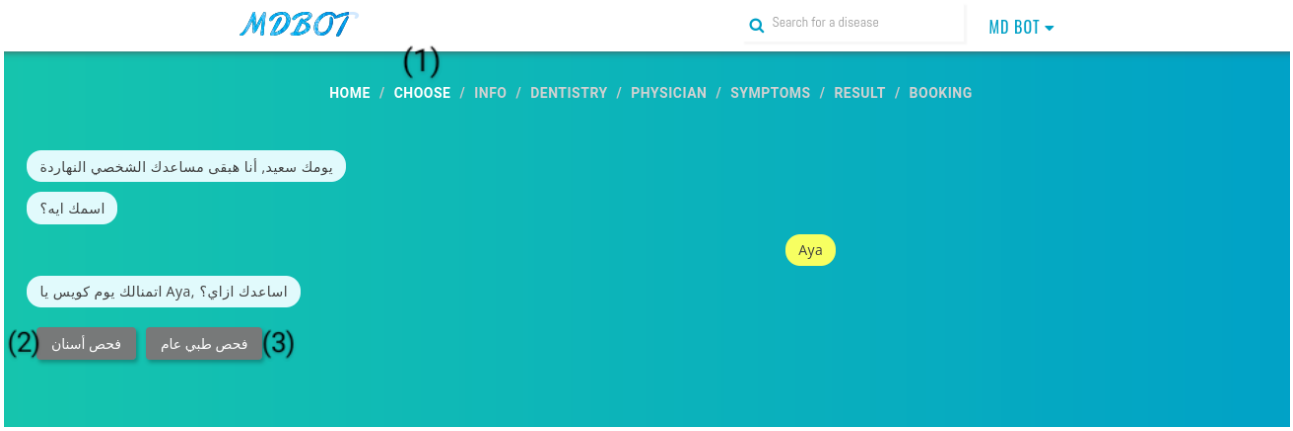
(2) **Password**: Users enter their registered passwords .

(3) **Login**: A button to read sqlite database and confirm values entered with values already in the db

(4) **Forget password?**: A button to reset the password in case users forgot their passwords

- **Choose medical field page**

  A rule-based chatbot, responsible for presenting our medical domains, and giving the user the option to choose from.



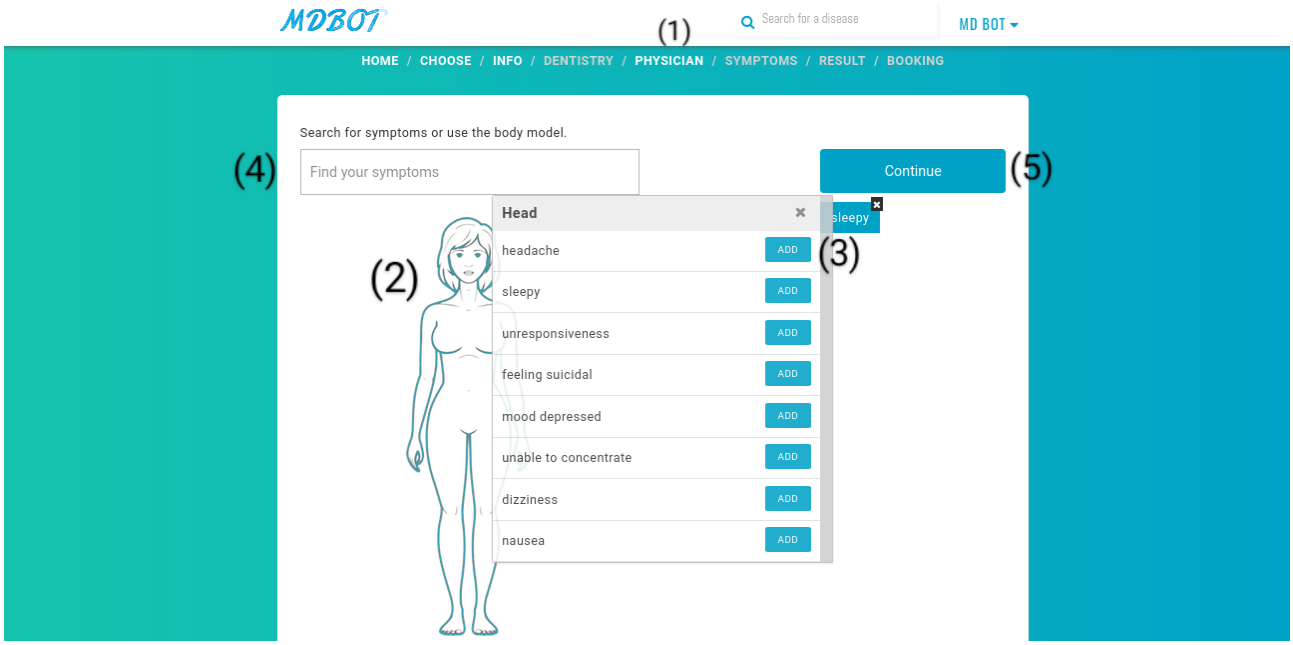
(1) **Choose**: The Medical Field selection section in the progress indicator bar.

(2) فحص أسنان: A redirect to the Dentistry section.

(3) فحص طبي عام: A redirect to the General Medicine section.

- **Symptoms Pages**

  The General Medicine section



Physician: The General Medicine Field section in the progress indicator bar.

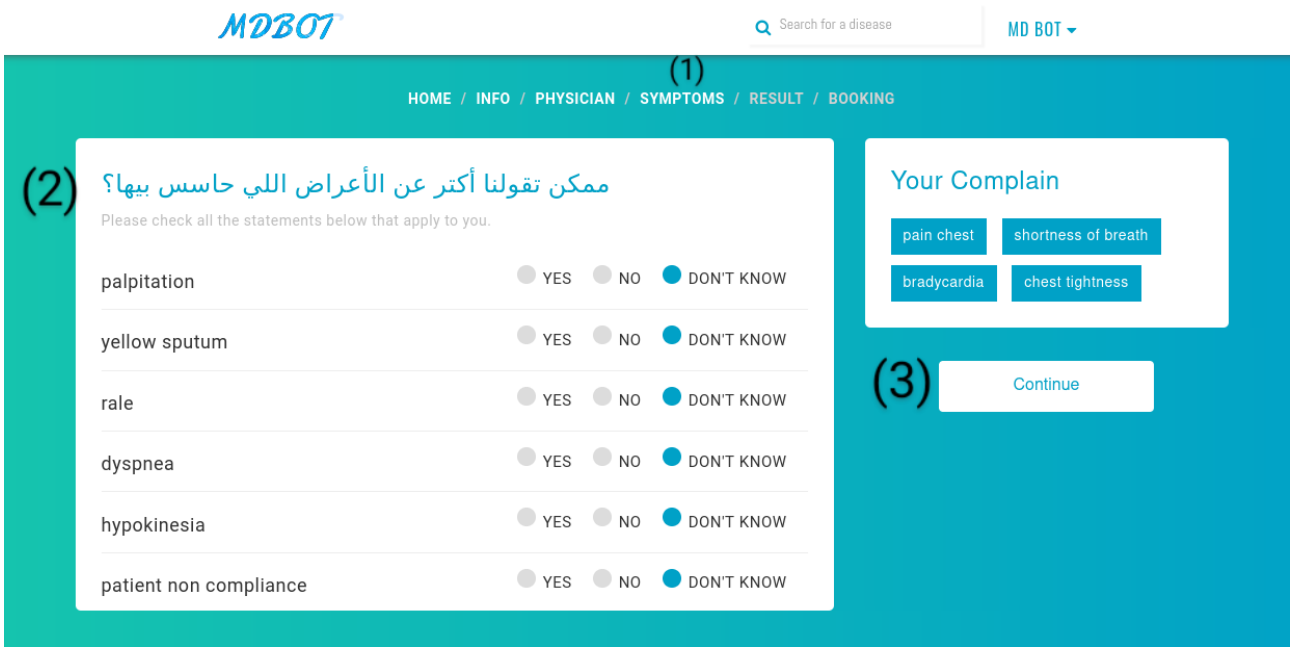
Picking body parts: Users point to the body part where they feel pain in.

Symptoms' selection: Depending on the body part, a list of symptoms appears, users could choose from.

Symptoms' Search: For users who prefer to enter the symptoms they're feeling directly without the need for the gender-based model.

Continue button: A redirect button to a more in depth questions related to the symptoms that users have chosen.

The General Medicine follow-up questions section



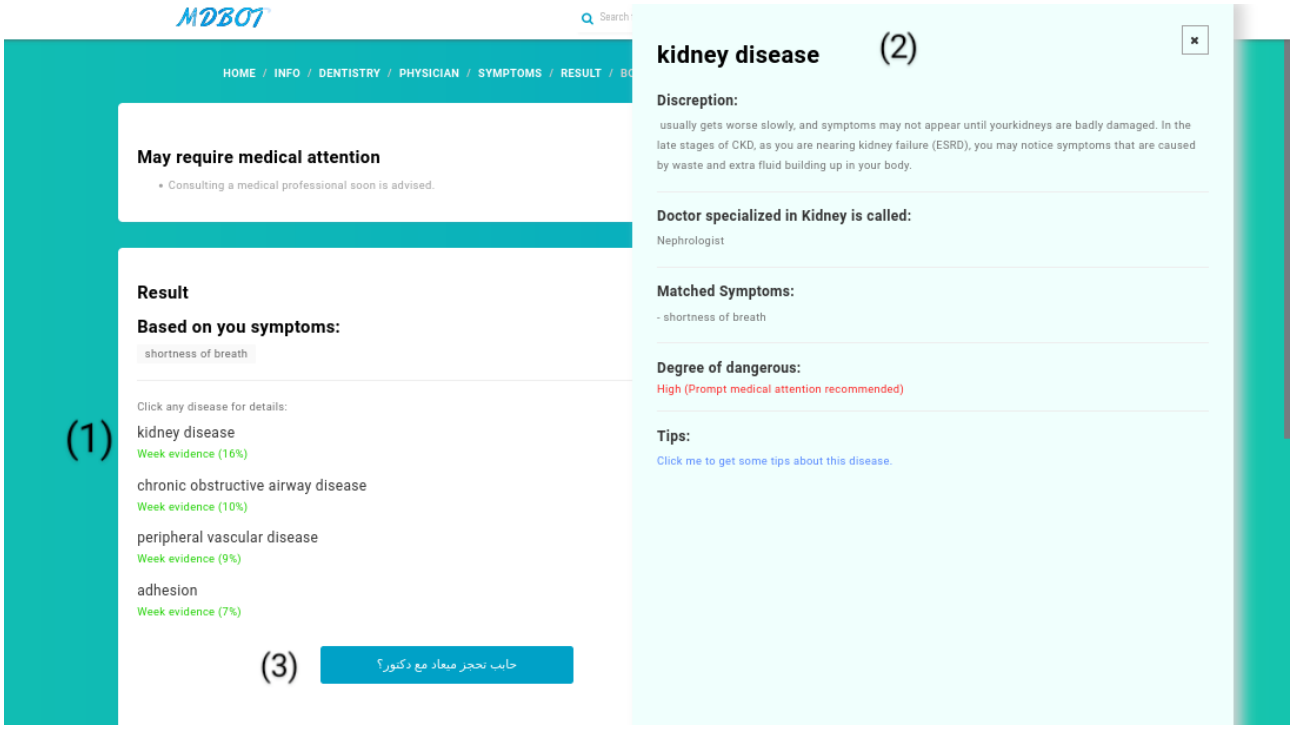
Symptoms: The follow-up questions section in the progress indicator bar.

Related Questions: More related symptoms and sub-symptoms in the form of polar/don't know questions and , users could answer to boost the accuracy of the specialization prediction.

Continue button: A redirect button to the General medicine diagnosis and medical report page.

- **Diagnosis Page**

  Information about the predicted diseases and the specializations they belong to, with an importance rank to the diseases.



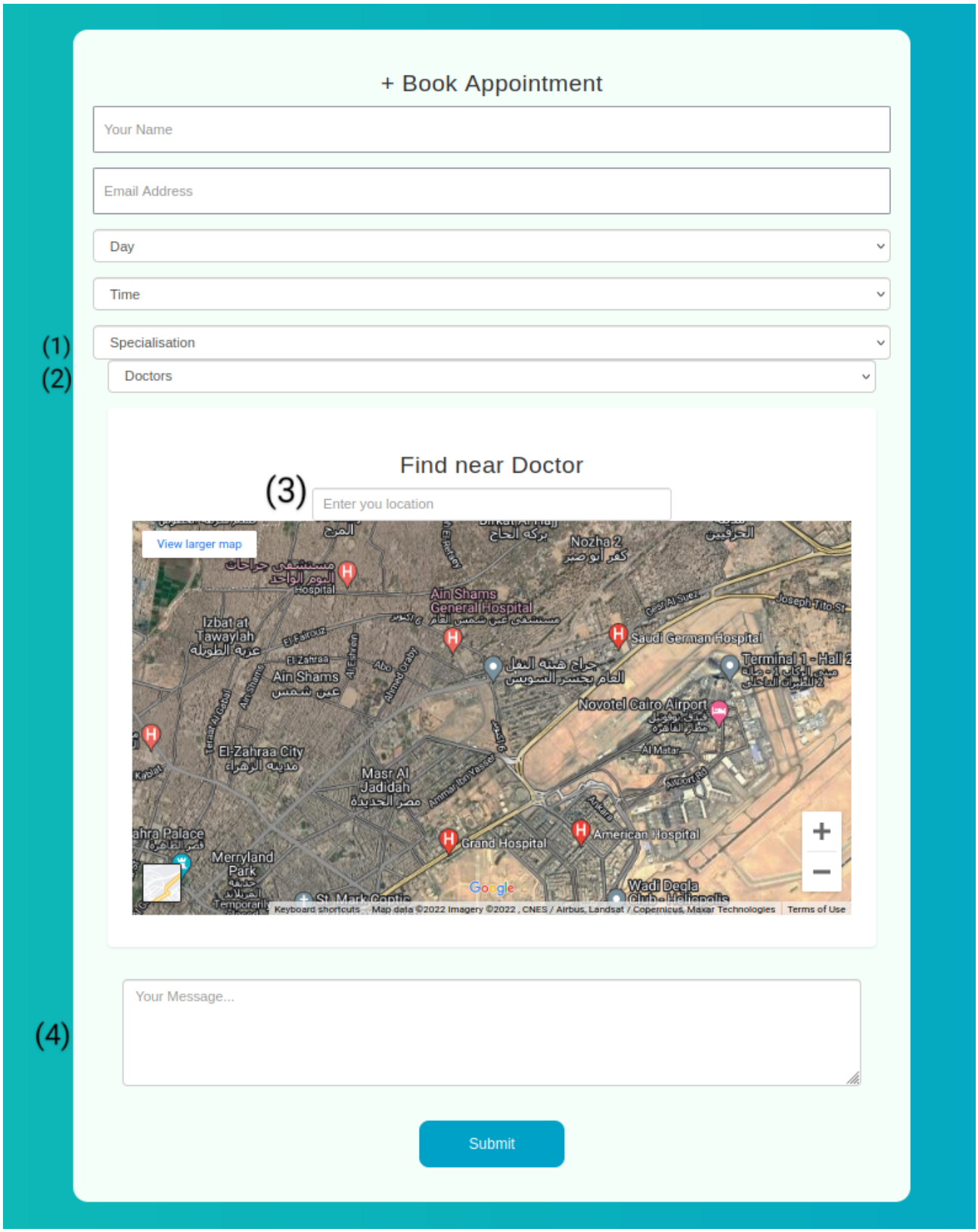
Ranked diseases: Each disease has a probability as a percentage occurrence based on the given symptoms from the General Medicine section.

Info Side page: Clicking on the diseases' names, more Information regarding the illness will appear with some functionalities such its description, rank based on which symptoms users have entered in the General Medicine section, and a "Tips" optional external link that contains more details and advices.

Booking button: A redirect button to the booking page.

- **Booking Page**

  An appointment form to book medical appointments online Online.



Specialization: According to the highest percentage disease, the specialization will be fetched.

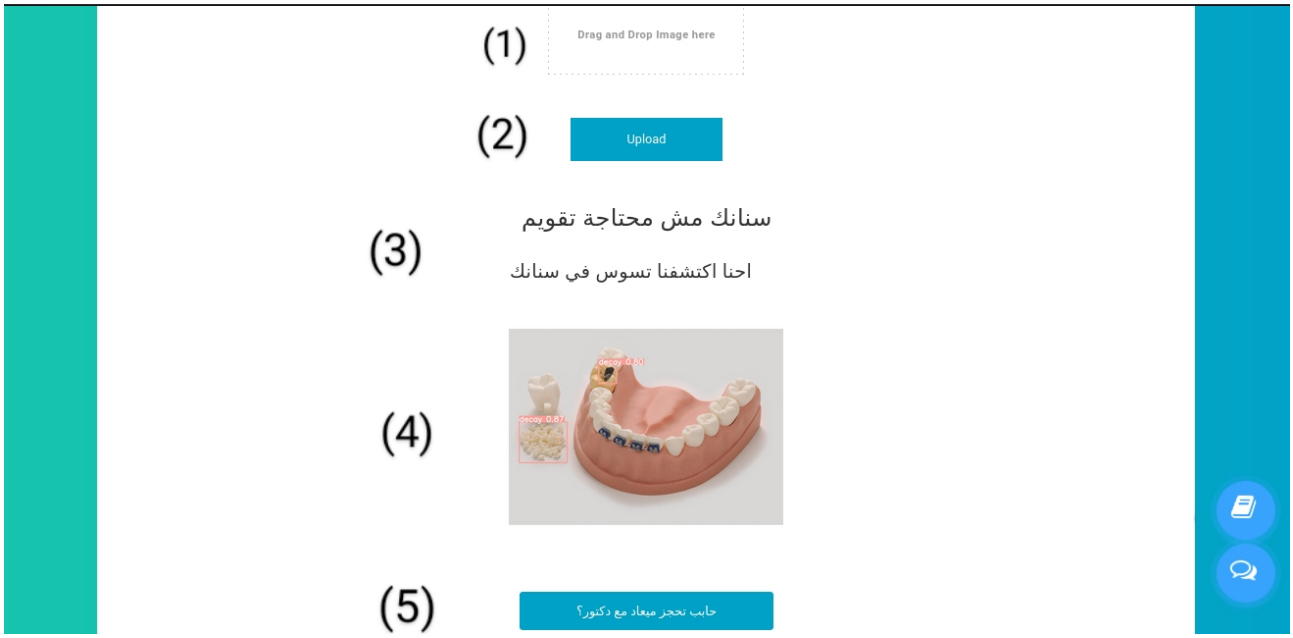
Doctors: Based on the specializations, all the doctors in system belonging to the same specialization will be retrieved.

Maps: Users could find hospitals and doctors by searching in maps and viewing the results.

Attached Message: Users could attach a private message in case they wanted to let doctor know something in advance.

- **Dentistry Page**

  The Dentistry section, offering 3 types of user communication.



Drag Drop: Users capture or upload from local files their teeth pics.

Upload button: A button used to process users' images and pass them into the prediction models to test them against tooth decay and the need for dental braces.

Prediction: The prediction results.

User Image: In case if users' teeth are suffering from any of the previous teeth problems, their teeth will be displayed with the caries percentage and Yes/No answer regarding if their teeth needs braces.

Booking button: A redirect button to the booking page.

Users identify their teeth problems from the displayed cards

Dental cases: Users could scroll through multiple Images associated with an Arabic description of the common symptoms.

Each card is clickable and redirects to the booking page if selected.

كلم البوت : A redirect fixed button if users wanted to chat about their symptoms.

Booking button : A redirect button to the booking page, where users could manually select the specialization.

A Conversational AI chatbot, where users could chat with the bot about their dentistry-related symptoms and diseases. It could answer questions about its creation and other selected general topics as a try to humanize the flow of conversation.

Send button : Users click on it to send their messages to the bot or they could press Enter on the keyboard.

Clear button : A button to erase the ongoing conversation and stay on the same page.

CHAPTER FOUR

PROJECT IMPLEMENTATION

Chapter 4
Project implementation

- **Initialization & Home page**

  Creating the flask App by passing the __name__ variable to the Flask class. Then establishing the Configuration attribute and Assigning the Routes.

```python
app = Flask(__name__)

# Some Definitions
app.config['SECRET_KEY'] = '4f9a5c9c0b0e66722f11b04bbfb4949f'
app.config["UPLOAD_FOLDER"] = os.getcwd() + "/static/images/user_upload"
ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg', 'webp', 'tiff', 'tif', 'bmp'])

# Routes
@app.route('/')
@app.route('/home')
def home():
    return render_template('front_index.html')
```

- **Login**

  Defining the endpoint route for the Login page as "login". Then validating the account. The user logs in if the account is a valid one.

```python
@app.route('/login', methods=['GET','POST'])
def login():
    form = LoginForm()
    if request.method == 'GET' and 'email' in session:
        return redirect(url_for('home'))
    if request.method == 'POST':
        session.pop('visitor', None)
        if form.validate_on_submit():
            con = sqlite3.connect("mydatabase.db")
            cur = con.cursor()
            query_string = """SELECT * FROM user WHERE email=? AND password=?"""
            cur.execute(query_string, (form.email.data,form.password.data,))
            users = cur.fetchall()
            con.close()
            if len(users) > 0:
                session['email'] = form.email.data
                session['firstname'] = users[0][1]
                session['lastname'] = users[0][2]
                session['sex'] = users[0][5]
                session['age'] = users[0][6]
                session['accType'] = users[0][7]
                flash(f'Welcome back {users[0][1]}!','success')
                return redirect(url_for('home'))
            else :
                flash(f'Login Faild!','danger')
    return render_template('login.html', form=form)
```

- **Register**

  Defining the endpoint route for the Register page as "reg". Then validating the information. The user forwarded to login page if the information is valid.

```
@app.route('/reg', methods=['GET','POST'])
def register():
    form = RegistrationForm()
    if request.method == 'GET' and 'email' in session:
        return redirect(url_for('home'))
    if request.method == 'POST':
        if form.validate_on_submit():
            con = sqlite3.connect("mydatabase.db")
            cur = con.cursor()
            query_string = """SELECT * FROM user WHERE email=?"""
            cur.execute(query_string, (form.email.data,))
            users = cur.fetchall()
            if len(users) > 0:
                flash(f'Email used before!','danger')
                return render_template('reg.html', form=form)
            query_string = """INSERT INTO user (firstName,lastName,email,sex,age,password,accountType) VALUES(?,?,?,?,?,?,?)"""
            cur.execute(query_string, (form.firstName.data,form.lastName.data,form.email.data,
                                                    form.gender.data,form.age.data,form.password.data,form.acc_type.data))
            con.commit()
            con.close()
            flash(f'Account created for {form.firstName.data}!','success')
            return redirect(url_for('login'))
    return render_template('reg.html', form=form)
```

- **Symptoms**

  Defining the endpoint name for the Symptoms route as "symps". Then connecting to the Database to fetch the Symptoms. Following this by passing the Symptoms for template rendering.

```python
@app.route('/symps', methods=['POST','GET'])
def symptoms():
    con = sqlite3.connect("mydatabase.db")
    cur = con.cursor()
    cur.execute("SELECT DISTINCT symptom FROM symptoms")
    symptom = cur.fetchall()

    if request.method == 'POST':
        session['visitor'] = 'visitor'
        session['sex'] = request.form.get('gender','udefined')
        if session['sex'] == "udefined":
            flash(f'Please select your gender!','danger')
            return redirect(url_for('home'))
        session['age'] = request.form['age']
        return render_template('symps.html', symptom = symptom)
    if request.method == 'GET' and 'email' in session:
        return render_template('symps.html', symptom = symptom)
    else:
        return redirect(url_for('info'))
```

- **Diseases Prediction**

  Defining the AI models (multinomial naive bayes and decision tree classifiers), for predicting the possible diagnoses based on the entered symptoms

```
import Data_Analysis
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier

def predict(mylist3):
    data=Data_Analysis.analysis()

    df_pivoted=Data_Analysis.analysis2(data)
    df_pivoted.to_csv("Scraped-Data/test2.csv")

    cols = df_pivoted.columns
    cols = cols[1:]
    x = df_pivoted[cols]
    y = df_pivoted['Source']

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)
    mnb = MultinomialNB()
    mnb = mnb.fit(x_train, y_train)
    mnb.score(x_test, y_test)

    mnb_tot = MultinomialNB()
    mnb_tot = mnb_tot.fit(x, y)

    disease_pred = mnb_tot.predict(x)
    disease_real = y.values

    dt = DecisionTreeClassifier()
    clf_dt=dt.fit(x,y)

    importances = dt.feature_importances_
    print(importances)
    indices = np.argsort(importances)[::-1]
    features = cols

    temp=[]
    count=0
    feature_dict = {}
    for i,f in enumerate(features):
        if(i!=0):
            feature_dict[f] = i

    l=len(mylist3)
    for i in range(l):
        sym = mylist3[i].replace('Ã','')
        sym = sym.replace('Â','')
        sym = sym.replace('\xa0',' ')
        sym = sym.replace('\x82','')
        v=feature_dict[sym]
        sample_x = [i/v   if i ==v else i*0 for i in range(len(features))]
        temp.append(v)

    for i in temp:
        count=count+1
    for j in range(count):
       del sample_x[temp[j]]
       sample_x.insert(temp[j],1)

    sample_x = np.array(sample_x).reshape(1,len(sample_x))
    return dt.predict(sample_x)
```

- **Dentistry**

  Displaying the dent page, or processing and verifying the user uploaded image to be used as input for the trained models for prediction.

```python
@app.route('/dent', methods=['POST','GET'])
def dent():
    if request.method == 'POST':
            file=request.files['file']

            if file and allowed_file(file.filename):
                filename = secure_filename(file.filename)

                if filename in os.listdir(app.config['UPLOAD_FOLDER']):
                    pred1 = teeth_predict.predict(os.path.join(app.config['UPLOAD_FOLDER'], filename))
                    pred2= teeth_predict.model2(os.path.join(app.config['UPLOAD_FOLDER'], filename))
                    path = pred2.save()
                    limit = len(pred2.pandas().xyxy[0].to_json())
                    return render_template('dent.html', pred1=pred1, limit=limit, path=os.path.join(path, filename))
                file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
                pred1 = teeth_predict.predict(os.path.join(app.config['UPLOAD_FOLDER'], filename))
                pred2 = teeth_predict.model2(os.path.join(app.config['UPLOAD_FOLDER'], filename))
                path = pred2.save()
                limit = len(pred2.pandas().xyxy[0].to_json())
                return render_template('dent.html', pred1=pred1, limit=limit, path=os.path.join(path, filename))

    return render_template('dent.html')
```

- **Train the BOW model**

  Processing the user natural language input to create the Bag Of Words model. Then, training the model to be used for pattern-detection in the user input.

```
import random
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Sequential
import numpy as np
import pickle
import json
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download("punkt")
nltk.download("wordnet")

print('Libraries loaded')

# init file
words = []
classes = []
documents = []
ignore_words = ["?", "!"]

data_file = open("intents.json").read()
print(type(data_file))
intents = json.loads(data_file)

# words
for intent in intents["intents"]:
    for pattern in intent["patterns"]:

        # take each word and tokenize it
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        # adding documents
        documents.append((w, intent["tag"]))

        # adding classes to our class list
        if intent["tag"] not in classes:
            classes.append(intent["tag"])

# lemmatizer
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))

classes = sorted(list(set(classes)))

print(len(documents), "documents", documents)

print(len(classes), "classes", classes)

print(len(words), "unique lemmatized words", words)

pickle.dump(words, open("words.pkl", "wb"))
pickle.dump(classes, open("classes.pkl", "wb"))

# initializing training data
training = []
output_empty = [0] * len(classes)
for doc in documents:
    # initializing bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]
    # lemmatize each word - create base word, in attempt to represent related words
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
    # create our bag of words array with 1, if word match found in current pattern
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # output is a '0' for each tag and '1' for current tag (for each pattern)
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])
# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)
# create train and test lists.
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
model.save("chatbot_model.h5", hist)
print("model created")
```

- **ChatBot Response**

  Building the chatbot response, after lemmatizing the arabic words and cleaning up the sentences.

```
lemmatizer = WordNetLemmatizer()
# chat initialization
model = load_model("chatbot_model.h5")
intents = json.loads(open("intents.json").read())
words = pickle.load(open("words.pkl", "rb"))
classes = pickle.load(open("classes.pkl", "rb"))
app.secret_key = 'chatsecret'
responsed_once = []

@app.route("/chat")
def chat_home():
    session['scenario']='chat'
    session['mode']=''

    responsed_once.clear()
    return render_template("chatbot_index.html")

@app.route("/get", methods=["POST"])
def chatbot_response():
    msg = request.form["msg"]

    ints = predict_class(msg, model)

    session['intent'] = ints[0]['intent']

    res = getResponse(ints, intents)

    if session['intent'] == 'add':
        session['mode']='formreply'
        session['field']=0
        form_field = 'What is the name?'
        res = res+'. '+form_field
    return res

# chat functionalities
def clean_up_sentence(sentence):
    sentence = normalize(sentence)
    sentence_words = nltk.word_tokenize(sentence)
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence
def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0] * len(words)
    for s in sentence_words:
        for i, w in enumerate(words):
            if w == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
                if show_details:
                    print("found in bag: %s" % w)
    return np.array(bag)


def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words, show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i, r] for i, r in enumerate(res) if r > ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list
```

- **Layout**

  HTML page used as a base layout for all other pages.

```
        <nav class="navbar navbar-default navbar-fixed-top">
            <div class="container">
            <!— Brand and toggle get grouped for better mobile display —>
            <div class="navbar-header">
                <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-
navbar-collapse-1" aria-expanded="false">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <div class="text-container">
                      <a href="/"><h1>MDBOT</h1></a>
                </div>
                </div>
                <!— Collect the nav links, forms, and other content for toggling —>
                               <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
                <ul class="nav navbar-nav navbar-right">
                    {% if session['email'] %}
                    <li class="search-header">
                        <form action="/search-disease" class="myForm" method="POST">
                            <i class="fa fa-search"></i>
                            <input type="text" class="form-control" name="diseaseName" placeholder="Search for a disease"
value="">
                        </form>
                    </li>
                    <li class="nav-item dropdown">
                        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                             {% if session['accType'] == "doctor" %}Dr.{% endif %} {{session['firstname']}}
{{session['lastname']}}
                            <i class='fa fa-caret-down'></i>
                        </a>
                        <div class="dropdown-menu" aria-labelledby="navbarDropdown">
                          <a class="dropdown-item" href="/edit-profile">Edit Profile</a>
                          {% if session['accType'] == "doctor" %} <a class="dropdown-item" href="/add-disease">Add
Disease</a> {% endif %}
                          {% if session['accType'] == "patient" %}<a class="dropdown-item" href="/doctor-contacts">Contact
Doctors</a>{% endif %}
                          <a class="dropdown-item" href="/my_appointments">My appointments</a>
                          <div class="dropdown-divider"></div>
                          <a class="dropdown-item" href="/logout">Logout</a>
                        </div>
                    </li>
                    {% else %}
                    <li><a href="/login">Login</a></li>
                    <li><a href="/reg">Join Now</a></li>
                    {% endif %}
                </ul>
                <!— <form action="/search-disease" class="myForm" method="POST">
                    <input type="text" class="form-control" name="search-disease" placeholder="Search for a disease"
value="">
                </form> —>
                </div><!— /.navbar-collapse —>
            </div><!— /.container —>
        </nav>
```

- **Diagnosis page**

  A page for displaying the potential diagnosis according to the user's symptoms with some accurate percentages, feedback, and appointment options.

```
<div class="diagnosis">
  <h3>Result</h3>
  <h3>Based on you symptoms:</h3> {% for s in symptoms %} <span class="user-symps">{{s}}</span> {% endfor %}
  <hr>
  <div class="disease-info">
    <p>Click any disease for details:</p>

    {% for d in diseases %}
    <div class="disease">
      <h4 class="iamDisease">{{d}}</h4>
      {% if diseasesInfo[loop.index0] =/= false %}
        <div class="diseses-data" >
            <i class='fa fa-close closeDis'></i>
          <h3 class="dis-name">{{diseasesInfo[loop.index0][1]}}</h3>
          <div class="dis-disc">
              <h4>Discreption:</h4>
              <p>{{diseasesInfo[loop.index0][2]}}</p>
          </div>
          <hr>
          <div class="dis-organ">
              <h4>Doctor specialized in {{diseasesInfo[loop.index0][4]}} is called: </h4><p>{{specialist[loop.index0]}}</p>
          </div>
          <hr>
          <div class="dis-sym">
              <h4>Matched Symptoms: </h4>
              {% for ds in diseasesSyms[loop.index0] %}
                {% if ds in symptoms %}
                  <p>⟵ {{ds}}</p>
                {% endif %}
              {% endfor %}
          </div>
          <hr>
          <div class="dis-deg">
              <h4>Degree of dangerous: </h4>
              {% if diseasesInfo[loop.index0][3] == 'H' %}
                <h5 style="color:#ff0000da;">High <span>(Prompt medical attention recommended)</span></h5>
              {% elif diseasesInfo[loop.index0][3] == 'M' %}
                <h5 style="color: #ff7700;">Medium <span>(Medical attention recommended)</span></h5>
              {% else %}
                <h5 style="color:#1ed200;">Low <span>(Medical attention not needed)</span></h5>
              {% endif %}
          </div>
          <hr>
          <div class="dis-tips">
            <h4>Tips: </h4> <a href="{{diseasesTips[loop.index0]}}" target="_blank">Click me to get some tips about this
disease.</a>
          </div>
        </div>
        {% endif %}
        {% if percent[loop.index0] > 60 %}
        <h5 style="color:#ff0000da;">Strong evidence <span>({{percent[loop.index0]}}%)</span></h5>
        {% elif percent[loop.index0] > 30 %}
        <h5 style="color: #ff7700;">Moderate evidence <span>({{percent[loop.index0]}}%)</span></h5>
        {% else %}
        <h5 style="color:#1ed200;">Week evidence <span>({{percent[loop.index0]}}%)</span></h5>
        {% endif %}
      </div>
      {% endfor %}
    </div>
```

- **ChatBot page**

  A page for displaying the chatbot system, that the user interacts with seeking dental specialty information.

```
{% extends "layout.html" %}
{% block content %}
<!DOCTYPE html>
<html>

<head>
  <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/chatbot_style.css')}}" />
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
</head>
<ul class="breadcrumb">
    <li class="active"><a href="/home">Home</a></li>
    <li class="active"><a href="/medicalDomain">choose</a></li>
    <li class="active"><a href="/dent">dentistry</a></li>
    <li class="inactive">Symptoms</li>
    <li class="inactive">Result</li>
</ul>

<body>

  <canvas id="particles-js"></canvas>

  <form>
        <div id="chatbox">
            <div class="col-md-12 ml-auto mr-auto">
                <p class="botText"><span>
                أهلا! أنا هبقى مساعدك الشخصي علشان تقدر تعرف التخصص المناسب لأسنانك
                </span></p>
            </div>
        </div>
        <div id="userInput" class="row">
            <div class="col-md-10">
                <input autocomplete="off" id="text" type="text" name="msg" placeholder="Message" class="form-control"><br>
                <button type="submit" id="send" class="btn btn-warning">Send</button>
                <button type="submit" id="send" class="btn btn-warning" style="margin-left:20px"><a href="/chat">Clear</a>
</button>
            </div>
        </div>
  </form>

<script>
    $(document).ready(function() {
        $("form").on("submit", function(event) {
            var rawText = $("#text").val();
            var userHtml = '<p class="userText"><span>' + rawText + "</span></p>";
            $("#text").val("");
            $("#chatbox").append(userHtml);
            document.getElementById("userInput").scrollIntoView({
                block: "start",
                behavior: "smooth",
            });
            $.ajax({
                data: {
                    msg: rawText,
                },
                type: "POST",
                url: "/get",
            }).done(function(data) {
                var botHtml = '<p class="botText"><span>' + data + "</span></p>";
                $("#chatbox").append($.parseHTML(botHtml));
                document.getElementById("userInput").scrollIntoView({
                    block: "start",
                    behavior: "smooth",
                });
            });
            event.preventDefault();
        });
    });


</script>
<script src="{{ url_for('static', filename='js/chatbot_main.js')}}"></script>
</body>

</html>
{% endblock content %}
```

- **Add Disease page**

  AddDisaease page appears for the logged-in doctors' accounts only, where they can add the Symptoms and their diseases and the diseases' organs. Then the data is saved in the Database.

```
{% extends "layout.html" %}
{% block content %}

<div class="container">
    <div class="header addDisease">
        <h3>Add New Disease</h3>
        <div class="alertMSG danger">
            All fields are required!
        </div>
        <form action="/add-disease" class="myForm" method="POST">
            <div class="addDisSym">
                <i class="fa fa-plus"></i>
                <span>Add Symptom</span>
            </div>
            <input type="text" name="dName" placeholder="Disease Name" class="form-control" required>
            <textarea name="dDefinition" id="" cols="30" rows="3" class="form-control" placeholder="Disease Definition"
required></textarea>
            <select name="organ" id="" class="form-control" required>
                <option value="" disabled selected>Disease Organ</option>
                {% for o in organs %}
                    <option value="{{o[0]}}">{{o[0]}}</option>
                {% endfor %}
            </select>
            <select name="degree" id="" class="form-control" required>
                <option value="" disabled selected>Disease degree of dangerous</option>
                <option value="H">H</option>
                <option value="M">M</option>
                <option value="L">L</option>
            </select>
            <input type="text" name="tips" placeholder="Enter link for disease tips" class="form-control" required>
            <input type="text" name="dSym[]" placeholder="Enter Symptom" class="form-control" required>
            <input type="submit" value="ADD" class="btn btn-primary" id="addDis-submit">
        </form>
    </div>
</div>

{% endblock content %}
```

- **Style with CSS**

  A snapshot for General external style sheet referenced with another HTML pages for designing the way they look.

```
/* #96daff */
a,a:active,a:hover{text-decoration: inherit; color: inherit;}
label{font-weight: normal;}
body{
    background: linear-gradient(to right, #16c5ad, #00a1c7);
    line-height: 1.8;
    overflow-x: hidden;
    letter-spacing: 0.5px;}
.btn-primary{
    display: block;
    margin: 35px auto;
    background: #00a1c7;
    border: unset;
    color: #fff;
    padding: 13px 72px;
    font-size: 16px;
    transition: all 0.7s;}
.alert-danger {
    color: #f5f5f5;
    background-color: #ec5e5e;
    border-color: #ffffff;}
.alert-success {
    color: #f5f5f5;
    background-color: #75bf57;
    border-color: #ffffff;}
.nicescroll-cursors{
    width: 8px !important;
    background-color: rgb(117, 124, 128) !important;
    border-radius: 0px !important;
    border: none !important;}
/* ---------------------- Breadcrumb ------------------------------*/
.breadcrumb{
    background: unset;
    padding: 0 0 25px;
    text-align: center;
    font-family: 'Roboto', sans-serif;
    font-weight: 600;
    text-transform: uppercase;
    letter-spacing: 0.5px;
    margin: 0;}
.breadcrumb .active{color: #fff;}
.breadcrumb .inactive{color: #d2d2d2; user-select: none;}
.error-alerts{margin-top: 80px;}
```

- **Custom JS**

  A snapshot for Custom External script, used for dynamic interaction in the front page.

```
/******** File Name: custom.js ************ */
(function($) {
    "use strict";
    /* =========== AFFIX ======================== */
    $('.megamenu').affix({
        offset: {
            top: 0,
            bottom: function() {
                return (this.bottom = $('.footer').outerHeight(true))
            }
        }
    })
    /* ======= BACK TOP ============================ */
    jQuery(window).scroll(function() {
        if (jQuery(this).scrollTop() > 1) {
            jQuery('.dmtop').css({
                bottom: "75px"
            });
        } else {
            jQuery('.dmtop').css({
                bottom: "-100px"
            });
        }
    });
    /* ================ LOADER ================================ */
    $(window).load(function() {
        $("#preloader").on(500).fadeOut();
        $(".preloader").on(600).fadeOut("slow");
    });
    /* ============= US ============================ */
    function count($this) {
        var current = parseInt($this.html(), 10);
        current = current + 50; /* Where 50 is increment */
        $this.html(++current);
        if (current > $this.data('count')) {
            $this.html($this.data('count'));
        } else {
            setTimeout(function() {
                count($this)
            }, 30);
        }
    }
    $(".stat_count, .stat_count_download").each(function() {
        $(this).data('count', parseInt($(this).html(), 10));
        $(this).html('0');
        count($(this));
    });
    /* ================= TOOLTIP ======================== */
    $('[data-toggle="tooltip"]').tooltip()
    $('[data-toggle="popover"]').popover()
```

- **Symptoms, Database table**

  SQL executable queries which apply different CRUD operations to manipulate the diseases and their symptoms within the system.

```
CREATE TABLE "symptoms" (
    "id"    INTEGER NOT NULL UNIQUE,
    "disease"   TEXT,
    "symptom"   TEXT,
    PRIMARY KEY("id" AUTOINCREMENT)
);

INSERT INTO symptoms (disease,symptom) VALUES ("Sciatica","Sciatica is the name
given to any sort of pain caused by irritation or compression of the sciatic
nerve.")

SELECT symptom FROM symptoms WHERE disease="biliary calculus"
```

- **User, Database table**

  SQL executable queries which apply different CRUD operations to manipulate different user accounts in the system.

```sql
CREATE TABLE "user" (
    "id"    INTEGER NOT NULL UNIQUE,
    "firstName" TEXT NOT NULL,
    "lastName"  TEXT NOT NULL,
    "email" TEXT NOT NULL,
    "password"  TEXT NOT NULL,
    "sex"   TEXT,
    "age"   INTEGER,
    "accountType"   TEXT,
    "specialization"    TEXT DEFAULT 'none',
    PRIMARY KEY("id")
);

INSERT INTO user (firstName,lastName,email,sex,age,password,accountType)
VALUES ("Dina","Walid","DinaWalid@gmail.com","female",25,POMDinaWalid125","doctor")

ELECT * FROM user WHERE email = "DinaWalid@gmail.com"

UPDATE user SET
firstName="newName",lastName="newName",age="26",password="newPass245" WHERE
email="DinaWalid@gmail.com"

SELECT firstName,lastName,email,specialization FROM user WHERE accountType =
'doctor'
```

Conclusion

The growth of AI-driven chatbots and virtual assistants suggests a future of a personalized fluid user experience. It will be one of the most effective ways we'll be able to deliver content that is contextually aware.

Virtual Healthcare Assistant are very useful tools to maintain the health records of the patients and information about doctors. It maintains the patient's or users' personal details. The system offers reliability and accessibility, it can be used as a basis for enhancing applications.

Chatbots are great tools for conversations, effective when it comes to Healthcare and It will be one of the most effective methods we'll be able to deliver content that is contextually aware.

Doctors should work hand in hand with technology providers in order to find better agreed ways to automate repetitive tasks without feeling threatened by AI taking their roles totally because AI was never here for that reason, It's here to build tools that help people, facilitate their work, and their interaction with computers using natural language.

The medical system that is symptom-based and personalized could be integrated with other organizations' products.

The effectiveness of the healthcare system at recognizing and predicting will be enhanced and increased by adding support for additional diseases and diagnostic features such symptoms intensity and detailed explanation of the symptoms.

References

[Azza A El-Housseiny and Derwi(2014)] N. M. F. Azza A El-Housseiny, Najlaa M Alamoudi and D. A. E. Derwi. *Characteristics of dental fear among Arabic-speaking children: a descriptive study*. 2014.

[Comendador(2015.)] B. V. Comendador. *Pharmabot: A pediatric generic Medicine consultant Chatbot*. 2015.

[Divya(2017)] I. P. Divya, Indumathi. *A SelfDiagnosis Medical Chatbot Using Artificial Intelligence*. 2017.

[SFlora Amato(2018)] S. M. SFlora Amato. *Chatbots meet eHealth: automat zing healthcare proceeding of diet*. 2018.

[Shimmaa Moustafa(2015)] H. A. Shimmaa Moustafa. *School Children Dental Health, Dental Fear and Anxiety in relation to their Parents' Dental Anxiety: Comparative Study*. Zagazig University, Egypt, 2015.

Appendices

This project submitted to the High Institute of Computer Science Information Systems, in partial fulfilment of the requirements for the degree of Bachelor of Computers and Information.

GitHub Link:
MDBOT

LaTeX Book Link:
MDBOT book

"الشات بوت الطبي"

الهدف من المشروع هو إنشاء مساعد إفتراضي طبي شخصي. سيساعد نظام إدراة الرعاية الصِّحية للأشخاص عبر دعم العناية الطبية المتعلقة بمجالين؛ مجال طب الأسنان والطب العام, سيعمل المُساعِد كخيار إعتيادي للمراقبة الطبية الذاتية.

يمكن للمستخدمين الوصول إلى روبوت الدردشة التفاعلية (الشات بوت) الخاص بالموقع الإلكتروني للانتقال بهم إلى قسم الطب العام لذكر أعراضهم وسيتم منحهم إجابة دقيقة لما يعانون منه، وبعد تحليل الأعراض، يتم تزويدهم بتشخيص للأمراض والتخصص الطبي المناسب والتخصصات الفرعية التي تنتمي إليها أمراضهم.

يحتوي الموقع الإلكتروني على قسم للفحص الذاتي لنظافة الأسنان يقوم بفحص صور الأسنان لمساعدة المستخدمين على تحديد ما إذا كانت أسنانهم صحية أم لا.

يتمتع المستخدمون بالقدرة على حجز المواعيد إذا احتاجت حالتهم الصحية زيارة للطبيب.