System Document

Healing Paws Veterinary Solution Documentation

Version 1.0

Group 15

17205985 & 17205981 & 17206254 & 17205989 & 17206001

2020/03/09

Table of Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Scope . . . 3
- 1.3 Problem Statement . . . 3
- 1.4 Technical Overview . . . 4

2 Group Work **4**
- 2.1 Work division . . . 4
- 2.2 Distance Communication . . . 4
- 2.3 How to design the project structure to facilitate the division of labor? . . . 5
- 2.4 How to deal with unsolvable technical problems . . . 5
- 2.5 Version control . . . 5
- 2.6 Work Progress . . . 5

3 Technical Implementation **6**
- 3.1 Program skeleton . . . 6
- 3.2 Authority Component . . . 7
  - 3.2.1 Module: Sign up/Login module . . . 7
  - 3.2.2 Module: Reset password module . . . 9
  - 3.2.3 Module: Redirect module . . . 10
  - 3.2.4 Module: Authority model . . . 11
- 3.3 Reservation component . . . 11
- 3.4 Language-switching Component . . . 12
- 3.5 Pet Management Component . . . 12
  - 3.5.1 Introduction of Pet in Database . . . 12
  - 3.5.2 Module: Add Pets . . . 13
  - 3.5.3 Module: Show Pets . . . 13
- 3.6 Real-time Communication Component . . . 14
  - 3.6.1 Introduction of Flask-SocketIO . . . 14
  - 3.6.2 Setting Socket.IO connection . . . 14
  - 3.6.3 Sending a message . . . 14
  - 3.6.4 Online population statistics . . . 16
- 3.7 Deployment and performance . . . 16
  - 3.7.1 Cloud Service: Huawei Cloud . . . 16
  - 3.7.2 Deployment strategy: Linux + Gunicorn + Nginx-Proxy + Eventlet . . . 17

4 Conclusions **17**

5 Appendix **17**
- 5.1 Deployment Project Via Linux+Nginx+Gunicorn . . . 17
- 5.2 Step 1 — Installing the Components from the Ubuntu Repositories . . . 18
- 5.3 Step 2 — Creating a Python Virtual Environment . . . 18
- 5.4 Step 3 — Setting Up a Flask Application . . . 18
- 5.5 Step 4 — Configuring Gunicorn . . . 20

1 Introduction

The Software Design Document is a document to provide documentation which will be used to aid in software development by providing the details for how the software should be built. Within the Software Design Document are narrative and graphical documentation of the software design for the project including use case models, sequence diagrams, collaboration models, object behavior models, and other supporting requirement information.

1.1 Purpose

The purpose of the Software Design Document is to provide a description of the design of a system fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to built.

1.2 Scope

The scope of the Software Design Document is to provide details description on the teamwork, technical stack and detailed implementation of the system. At the appendix section, additional code and referenced paper will be included and displayed.

1.3 Problem Statement

Our team are employed to design a online Management System for a Veterinary Hospital.In this system, party A demands two client-side interfaces: one for the hospital employees and the other for the customers.The document below is a initial version description for the system requirements.

As we all know, our project is a hospital system, and this means the system should be with high security, convenience and user oriented. In our daily life, if a pet needs to be treated, we will observe the extent of its injury to depend if it needs emergency. And this should be decided by the doctors of hospital. After the doctor deals with the appointment, customers can see their pets' state. So we think there will be four requirements. First, treat will be two types: emergency and standard. And this can be depended by doctors. Second, customer can make appointments for their pets, and then doctors can deal with them. Third, all appointments can be kept track, which means customer can see their pets' state, such as when the customer can be treated, etc. Forth, there will be two different pages for users to use, one is for customer, and the other is doctor. Customer can make appointment for their pets. Doctors can change the information of appointments and update them to show to customers. Besides these requirements, if customers have problems, how they can contact with us? Chatroom maybe the best choice. As for the non-functional requirements, we think there are many websites with low security, which causes privacy information of customers can be known. Secure is the first choice. And with the development of mobile internet, our system should be used by both computers and mobile phone. Third, globalization should also be considered, interfaces with English and Chinese are also needed. Cloud-based also needs a server to make our website be available on World Wide Web.

After finding the requirements, we would like to design our websites by using several technologies. In the front end, we think bootstrap is the best choice for us because it is a simple, intuitive and powerful front-end development framework, making web development more rapid and simple. And we can use some beautiful interfaces which can be found on the internet easily. As for the back end, we use flask. Flask is the technology that we learnt in the last term. Compared with PHP, we are more familiar with it. We used blueprint to separate the functions into some different sections. And everyone can be responsible for one small part

Then we'd like to introduce how we plan the progress of our project. First, we established a repository on the Github and invited all group members and TA. Then, we write the work packages for each group. There are four groups, maintenance group, leading group, customer group and code group. Leading group is responsible for make the plan for the whole group and determine the task of each week. Customer group is responsible for communicating with customers, and report the customers' needs to the group every week. Code group has the whole members and as its name, it's responsible for the implementation of the project. Maintenance group is responsible for the organizing and scheduling the iteration/release, demos and presentation. Outlining the document for progress, user instruction and do maintenance on cooperation workspace. Except these, we also made our worksheet and team agreement. Finally, to ensure every week's progress, we have a meeting on every week's Monday. We will discuss what we have done and what we will do on this week. And on the weekends, everyone also needs to report their progress to the person who is responsible for the weekly-update. Weekly-update is written by the members of group in turn.

1.4 Technical Overview

In this system, our team adopted Python as the back-end development language, Flask as the back-end framework and bootstrap as the front-end framework for system development. Flask is a microservices framework written in Python, with the core simple and easy to expand. Flask's pip package management tool is very suitable for teamwork, which can reduce the risk of dependency conflicts. At the same time, the Sqlchemy extension of the flask framework completes the ORM mapping, which simplifies our development process at the data layer. The blueprint feature of the Flask framework facilitates the division of the system's functional modules. Basically, we divide it into pet module, user authentication module, chat room module, administrator module, order processing module, and basic user interface development. This facilitates our team's division of labor and cooperation, and the technical implementation document will also elaborate on the division of labor between different modules.

Moreover, in the front end, the Bootstrap framework is used in the front end to meet the needs of mobile terminal adaptation. At the same time, the grid system is also conducive to testing and iterating as soon as possible when our team develops prototypes early. In addition to bootstarp, we also use the google font graphics library.

The deployment strategy of the system is to use the flask built-in wsgi interface, gunicorn server (we added real-time chat function later in the project, so we replaced it with eventlet asynchronous service for deployment) and nginx reverse proxy. Due to the performance problem of python web Using nginx reverse proxy can increase system performance and increase bandwidth. For cloud server providers, we chose Huawei Cloud

2 Group Work

When we carried out the group project, we met the following problems: How to assign tasks to each team member?How do team members communicate under the influence of a SARS-COV-2?How to design the project structure to facilitate the division of labor?How to deal with unsolvable technical problems?How to version control?

2.1 Work division

The team members divide the tasks reasonably among themselves.Our team was divided 4 different groups:leading group,customer group,code group and maintenance group.
Leading group is responsible to organize meeting every week ,write weekly-update,deal with issues of team work and evaluate the project progress.
Customer group is responsible to evaluate needs of customers, provide relevant functionalities and design user interface of the program, and finally test the functionality and visual effects (user interface) of the program and then improve it.
Code group is responsible to maintain and simplify design, establish integration environment and a test suite, maintains source control and ensures proper execution.
Maintenance group is responsible to organize and schedule the iteration/release, demos and presentation(including video).Outline the document for progress, user instruction and do maintenance on cooperation workspace.
Each group has a leader and group members.The leader leads the group to complete the task. The relationship between the group leader and the group members is not a general sense of leadership, but closer to the cooperative relationship between friends, and better at communication and discussion.In addition, the work assigned to each team member is based on the field that each person is good at or the work that the member is willing to undertake.

2.2 Distance Communication

The team conducts effective communication and consultation.Due to the epidemic situation, the group members could not communicate face to face. So,our group adopted online communication, which means we hold group meetings at a fixed time every week to discuss what we would do this week and the progress of the project by using Tencent-meeting tool, and set up a WeChat group to communicate some technical difficulties and bottlenecks.
In addition, since each team member has different abilities and schedules, we help each other to complete the task in the way of communication, and achieve coordination and efficiency within the group as much as possible through communication.

2.3 How to design the project structure to facilitate the division of labor?

How to design the architecture and how to design the project framework is a problem needs to be solved in the first place.A chaotic and complex project framework will inevitably increase the difficulty of assigning tasks and thus lead to the difficulty of cooperation. On the contrary, a simple and clear project framework can increase the cohesion of each code section, thus improving the efficiency of team cooperation.For example: replacing routing with blueprints, designing the structure of the project based on the functionality. These were not thought at the beginning of the project, but as the project progressed, we had to adjust and improve the structure of the project to make it more suitable for teamwork.In addition,making goals for each different periods could help guide our group to finish work.

2.4 How to deal with unsolvable technical problems

Look for appropriate references. Browsing for the right resources can be surprisingly helpful: literature, professional papers, websites, books, and instructional videos.By learning and sharing these materials, our group enables each member to understand the technical support of some extent, so as to promote the common progress and cooperation of team members.

2.5 Version control

Our team uses Sourcetree for version control because the graphical interface is easier to use and the submission records can be easily viewed. We set up a warehouse on github to store the code, and set up people who conduct weekly testing and code review. Every weekend, the code that passes the test will be submitted, and will be reviewed and changed at the regular meeting on Monday

The standard within the team (code standard, time standard). To develop a unified standard is the most important thing in team cooperation, the front-end and the back-end respectively which language to use, which programming tools to use, what software to use to upload files, what format to use code and so on are all factors to be considered.Our group used flask framework of python for writing back-end, used js and CSS to render front-end, used sourceTree to upload code to github, and used pycharm tool to write python codes.

2.6 Work Progress

week-01:
Our group has deployed two kinds of framework on server(Simply summary):
1.linux+nginx+gunicorn+flask:
After installing all pieces(include pyhton3-pip) that we need from Ubuntu repositories , we create a python virtual environment(venv).Base on these effort before ,we install Flask and configure Gunicorn use WSGI entry point. Then ,configure Nginx to pass web requests to socket file in the project directory.Finally ,get on an SSL certificate to remain secure on our server.
2.windows+apache+flask: Installing apache x64 and download VC14 windows binaries and modules. After that we start apache 2.4 service (using cmd command or tools).Download and install php ,configure php into apache which allows us access php page.This method base on windows OS.

week-02:
Our group has developed the basic structure and some functions of the website. You can sign up and sign in by using two kinds of user: customer and employee. And some members' work packages also have been updated. We divided the project into some little projects, such as: sign up, display the information, chat with employee, etc. Every member will be responsible for one little project. You can see the basic website on the github or http://117.78.11.41/

week-03:
We decide to write update report in turn on the webinar for this week. The progress keeps being pushed. Our members have finished the basic functions for their division. We are going to further develop it. Also, we learn a lot on how to cooperate in overleaf. You can see the basic website on the github or http://117.78.11.41/:5000

week-04:
Our group found some bugs after integrate pages which were coding by different members. We tried to solve most of errors and continued finishing basic functions. Basic function of "Pet" page, login and register page is finished. We designed frontend of index page. Our database was set up and regulated to contain all the models. We also created framework of chatroom to connect service and client, but for now we can only show chatroom by console. Finally, we discussed about Project Proposals for main content of Project Proposal.

week-05:
Our main concern is on our pitch. Previously, we put our main target on developing a system via pure flask, for the mobile side development we decide to develop some api for our project.

week-06:
Our group has improve our GUI of veterinary hospital base on css.And from that time,our project has a system model of our graphic design for our interface.
In addition,we also testing some part of our codes and optimize the code framework which make our code more structured,more distinct and easier to divide the work.For instance,we break up the models.py and divide into accounts.py, pets.py, messages.py ,users.py and so on which are kept in a folder named models.Besides,we generated controllers folder to render our templates and functions instead of use routes.py.
Our group add a new function which let user could retrieve password by email.This means we need a official email to send these message.And this function makes our project more people-oriented.
We also correct some coding mistakes of our database that make database more complete and more high coupling.

week-07:
We correct some mistakes on our database, add the function of changing language. The UI also be changed a lot. The additional bar can be seen on the website.

week-08:
We encapsulate the crud interface of database, finish websocket test. And the UI of some page have been upgraded.

week-09:
Set the blueprint of pets. Update the structure of pet function. Finish the implementation of reservation's add, show. Update the UI of chatroom.

3 Technical Implementation

3.1 Program skeleton

Through requirement analysis, the project is a multi-user system, so unlike a single user, the project must be composed of two ports and also include user authentication functions. Since it is an online system, the system must have a basic view protection module. At this time we need to verify the user's identity and track the user's login status.

For the functional module, on the user side, we have functions such as storing pet information, making reservations, editing order content, and consulting. We can divide the four functional modules: user, pet, order, and consultation. These modules will greatly help the design of the database. Our chosen architecture Flask gave the blueprint feature to divide the work package, the project skeleton will be arranged by a classic flask project plus the blueprint binding function module.

In large-scale Flask projects, there are mainly three common organizational structures: functional architecture, partitioned architecture and hybrid architecture. The program will use a partitioned architecture, that is, resources related to each module are placed in the same package The package is placed in the root directory of the package.

The program is mainly contained six component: authority component, hospital's dashboard, pet management, reservation component, real-time consolation system. Further more, to get a easy-to-scale structure, we use the factory mode to configure and register kinds of functional method. Beside the functional components, we add some extension for the program to be operate in the command interface.

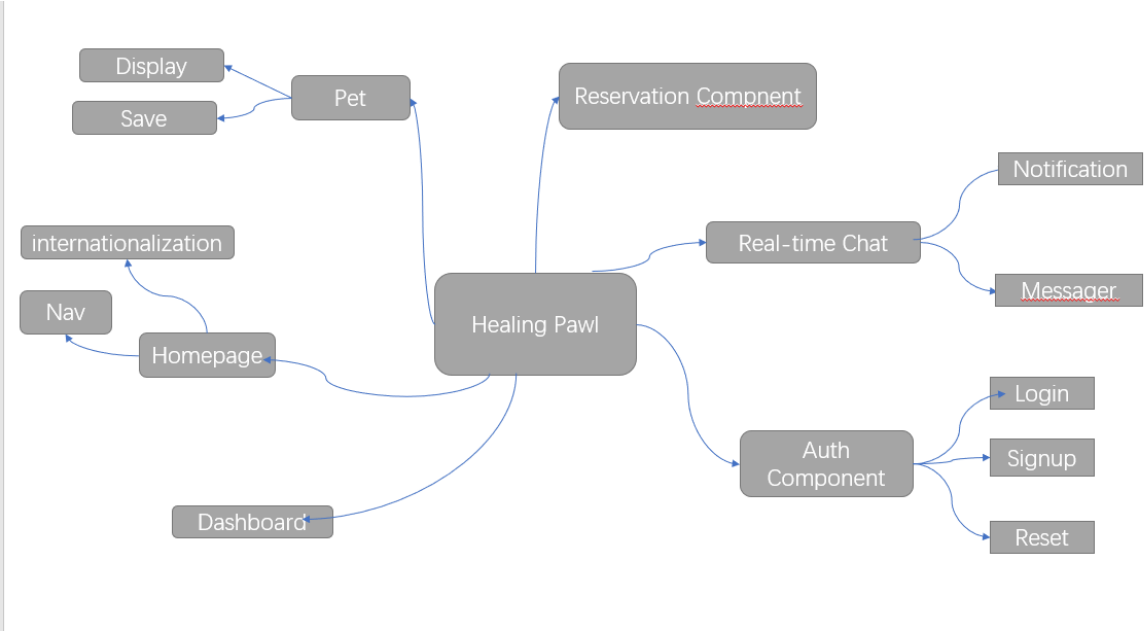


Figure 1: Program Skeleton

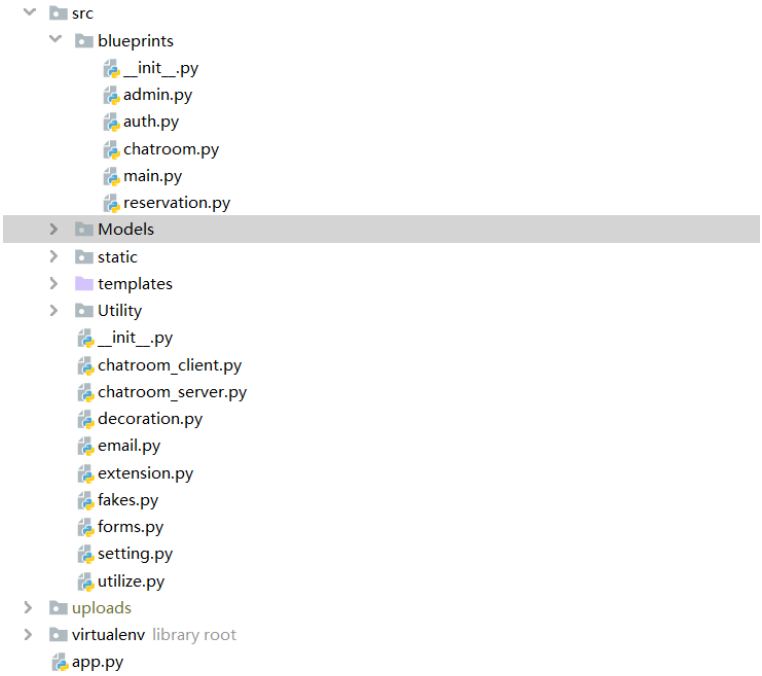


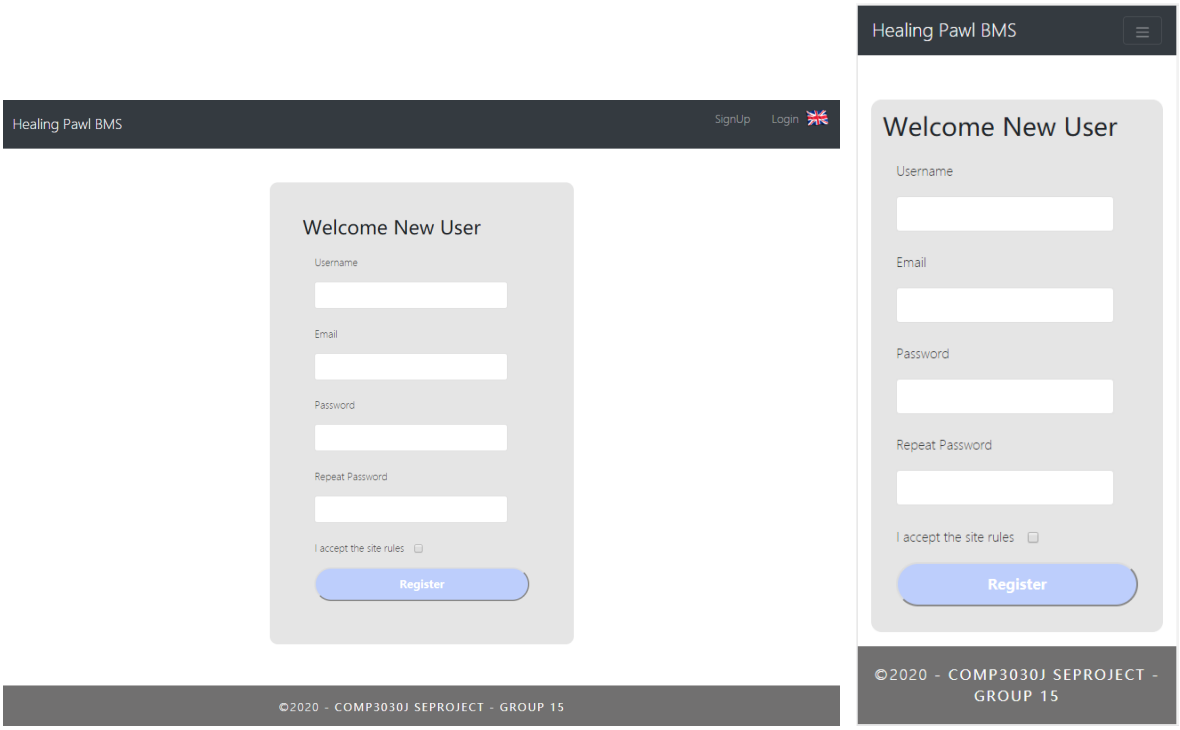
Figure 2: Program Structure

3.2 Authority Component

3.2.1 Module: Sign up/Login module

front-end

Both signup page and login page are based on bootstrap frame. Considering about the adaptation of computer and mobile, we limit the max width of main block. So, the main block can be regular for both computer and mobile. The blank space is alterable depend on the screen size.



In signup page, there are some hints showed by JS and jQuery.

Empty hint is showed by flask and form in front-end

```
    {% for error in form.username.errors %}
        <span style="color: red;" class="failure">{{ error }}</span>
    {% endfor %}e)
```

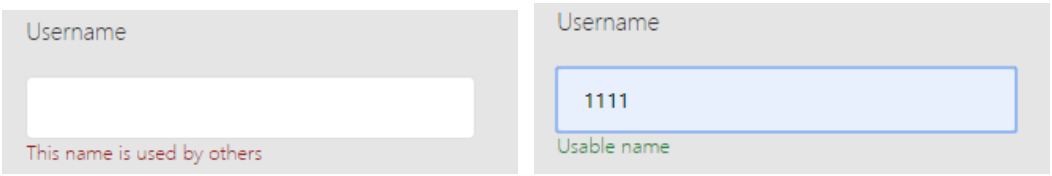
Format hint is showed by jQuery. When user focus on an input tag which has a format hint, add text in its span tag. The class and content of hint is changed in focus function.

Format is judged by Regular Expression.

```
var regex = /^[A-Za-z0-9\u4e00-\u9fa5]{3,10}$/;
```

If input text is wrong format, focus is controlled by JS and use jQuery to show hint and control color of hint by class. Then the input text is deleted and focus this input again.

Because username and email are unique, we use POST to send username and email to back-end, and receive JSON message from back-end. According to this JSON message, front-end show hint to user.



All the input and submit is generate by flask-form. In front-end, it's directly created by flask.

```
{{ form.username(size=32) }}<span id="hintuser"></span><span id="checkuser"
    ></span>
```

Back-end

First, we create form for signup and login. It's convenient for both back-end and front-end.

```
class SignupForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
```

```
    email = StringField('email', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    password2 = PasswordField('Repeat Password', validators=[DataRequired()])
    accept_rules = BooleanField('I accept the site rules', validators=[DataRequired()])
    submit = SubmitField('Register')
```

In back-end, we can receive text in form directly.

```
if form.validate_on_submit():
```

```
user = User(username=form.username.data, email=form.email.data, password_hash=passw_hash)
```

For security, password is saved in database after hash.

```
passw_hash = generate_password_hash(form.password.data)
```

In login page, first we judge if this username is exiting by database.

```
 if form.validate_on_submit():
        user_in_db = User.query.filter(User.username == form.username.data).first()
        if not user_in_db:
            flash('Wrong username or password, please check again')
            return redirect(url_for('auth.redirect_page', page='auth.login'))
```

Then judged if the password is correct. If username and password is correct, save the session to remain login status. If user choose "remember me", adjust delete time of session.

```
if check_password_hash(user_in_db.password_hash, form.password.data):
    session["USERNAME"] = user_in_db.username
    user_in_db.authIn()
    if form.remember_me.data:
        session.permanent = True
    flash('Login successfully')
```

Flash message is show in redirect page (Will be explained in detail later in redirect page module)

After login and signup, the authority will be judged (Will be explained in detail later in authority module).

3.2.2 Module: Reset password module

We use email to reset user's password with Flask-Mail. Flask-Mail is convenient to send email to users. Front-end is similar as login page. Also use flask-form to generate input.

Back-end

Before use Flask-Mail, we should set some basic information.

```
MAIL_SERVER = 'smtp.163.com'
MAIL_USERNAME = 'wsdsgbxd@163.com'
MAIL_PASSWORD = 'VLEMBIFMETRSMLCJ'
MAIL_USE_SSL = True
MAIL_PORT = 465
MAIL_DEFAULT_SENDER = ('YSY', 'MAIL_USERNAME')
```

To identify which account is resetting, a token is needed. We use pyjwt to create token.

One method used to create token by account with encode function in pyjwt, time and SECRET KEY, another used to get account from token with decode function in pyjwt.

```
def get_jwt_token(self, expires_in=6000):
    token = jwt.encode({'reset_password': self.id, 'exp': time() +
       expires_in},
                       current_app.config['SECRET_KEY'],
                       algorithm='HS256').decode('utf8')

    return token
def verify_jwt_token(token):
    try:
        user_id = jwt.decode(token,
                             current_app.config['SECRET_KEY'],
                             algorithms='HS256')['reset_password']
    except Exception as e:
        print(e)
        return
    return User.query.get(user_id)
```

We packaging send email function in another python file. Both token and email send is integrate in this function.We set some information of email first. And than use send function in Flask-Mail to send email

```
def send_email(subject, sender, recipients, text_body, html_body):
    msg = Message(subject, sender=sender, recipients=recipients)
    msg.body = text_body
    msg.html = html_body
    mail.send(msg)
def send_password_reset_email(user):
    token = user.get_jwt_token()
    send_email('reset your password',
               sender=current_app.config['MAIL_USERNAME'],
               recipients=[user.email],
               text_body=render_template('email/reset_password.txt', user=
                  user, token=token),
               html_body=render_template('email/reset_password.html', user=
                  user, token=token))
```

Then, to send a reset email, we only need to find this account by it's email address and use 'send password reset email' function.

Link in reset email has a token. After verify this token we can identify which account is going to be reset password.

```
user=User.verify_jwt_token(token0)
```

At last, we can easily change the password of this account.

3.2.3 Module: Redirect module

This page is used for show flash messages and redirect.

Front-end

We use JS to realize countdown and automatically skip with setInterval function. URL is generated according to the page information from back-end.

In this page we only show the last flash message. So, users can always get the latest message. This is showed by flask sentence in front end.

```
{% with messages = get_flashed_messages() %}
    {% if messages %}
        <h3>{{ messages[-1] }}</h3>
    {% endif %}
{% endwith %}
```

Back-end

Page that user will redirect to is saved in URL, and it's also sent to front-end by render template function.

```
@auth.route('/redirect_page/<page>')
def redirect_page(page):
    return render_template('auth/redirect_page.html', page=page)
```

3.2.4 Module: Authority model

First, we provide some email address for staffs/doctors. After signup, each account has a role id. Account that created by these email address has role id of DOCTOR, and others has role id of USER.

```
 def set_role(self):
    if self.role is None:
        if self.email in current_app.config['ADMIN_EMAIL']:
            self.role = Role.query.filter_by(name='DOCTOR').first()
        else:
            self.role = Role.query.filter_by(name='USER').first()
        db.session.commit()
```

According to the authority, user and doctor will redirect to different pages after login. Background management page checks role of accessed account and can only access by account with DOCTOR authority.

3.3 Reservation component

When user create a reservation, information of this reservation is sent to back-end by jQuery through POST by a list.

CSS color is changed for wrong input hint. With setTimeout function, the hint can be sparking. When POST is done, refresh this page.

In back-end receive this reservation and add it into database.

```
add_res = request.form.getlist("res[]")
            pet = Pet.query.filter(Pet.id == int(add_res[0])).first()
            Reservation.add_res(user_in_db, pet, add_res[1], add_res[2], "
               surgery confirmed")
```

Edit reservation is similar as add reservation.

In administrator page, reservations can be moved and the order can be saved. We use jquery.tablednd to realize these requirement.

```
$("#show_table").tableDnD({
```

After each move, "onDrop" is start. The new order is saved as a list and send to the back-end by POST. There is a python file which save the list inside and provide interfaces. When back-end receive a new, call these interfaces and update list in this python file.

```
if request.form.getlist("id_list[]") is not None and request.form.getlist("
   id_list[]") != []:
    reservation_list.update_list(request.form.getlist("id_list[]"))
```

This list is sent to front-end by render template function. Front-end show the reservations in this order.

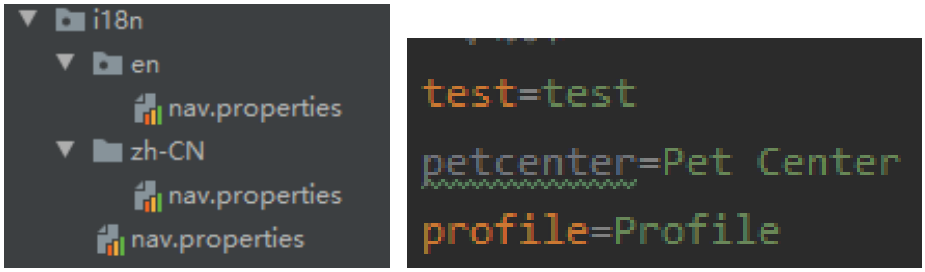
Delete reservation is realized by GET method. Id of deleted reservation is sent in URL. Back-end can easily receive this id and delete it in database.

Reservation has a final state-"finished". If a reservation's state is ready for release, users can finish this reservation. Then this reservation will not show in web pages but saved in database.

Each pet created by users can only has one ongoing reservation(is not finished). Users can only choose released pet when create an reservation.

3.4 Language-switching Component

We realize this function by jquery.i18n.properties. There are three properties files created for different languages. Each properties file contains name and content for each text in HTML.



Use meta in required pages to choose properties file.

```
<meta id="i18n_pagename" content="nav">
```

Default language is based on language of browser which is get by JS.

```
var getNavLanguage = function(){
    if(navigator.appName == "Netscape"){
        var navLanguage = navigator.language;
        return navLanguage.substr(0,2);
        }
    return false;
```

Language chosen by user is saved in Cookie. So, the chosen can be saved and user do not need to choose language again. According to the chosen language, use interface in jquery.i18n.properties to read properties files and rewrite text in HTML.

In html, provide each text a same class and it's name. In JS document we search for text with this class and change it's content according to its name.

```
insertEle.each(function() {
    $(this).html($.i18n.prop($(this).attr('name')));});
```

3.5 Pet Management Component

3.5.1 Introduction of Pet in Database

When we design the database, we think that a user can have multiple pets, but a pet can only have a reservation. We use foreign key to achieve that.

```
class Pet(db.Model):
    __tablename__='pet'
    id = db.Column(db.Integer, primary_key=True)
    petname=db.Column(db.String(100))
    petage=db.Column(db.String(100))
    petimage=db.Column(db.String(200))
    pettype=db.Column(db.String(100))
    user_id=db.Column(db.Integer, db.ForeignKey('user.id'))
    reservation = db.relationship("Reservation", backref='pet', lazy='
       dynamic')
```

To avoid user can see others' pets in their pets' page, we think that a function to examine if the user is the owner is important, so we write a function.

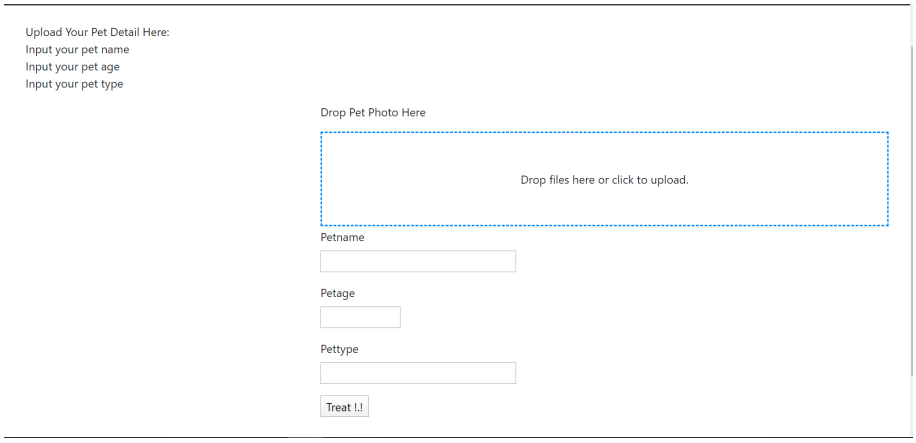
```
    def get_user_pet(id=None):
        pet = None
        if id is None:
```

```
            # pet=Pet.query.first()
            return None
        else:
            pet=Pet.query.filter(Pet.user_id ==id).order_by(Pet.id.desc()).
                all()
            # print(pet)
            return pet
```

And there are also some functions for adding and removing pets, they are very simple functions.They are for users to add and delete their pets in the system. There is also a function named $read_all()to show all pets in the database$.

3.5.2 Module: Add Pets

To add pets in the database, we create a page to do that, this page allows user to add pet's image, name, age and type into the database.



In this page, pet's name, age, type are using the forms. But pet image we use the drop zone. Because dropzone is very simple for user to use compared with FileForm.

```
  <p>Drop Pet Photo Here</p>
  {{ dropzone.create(action='main.upload') }}
```

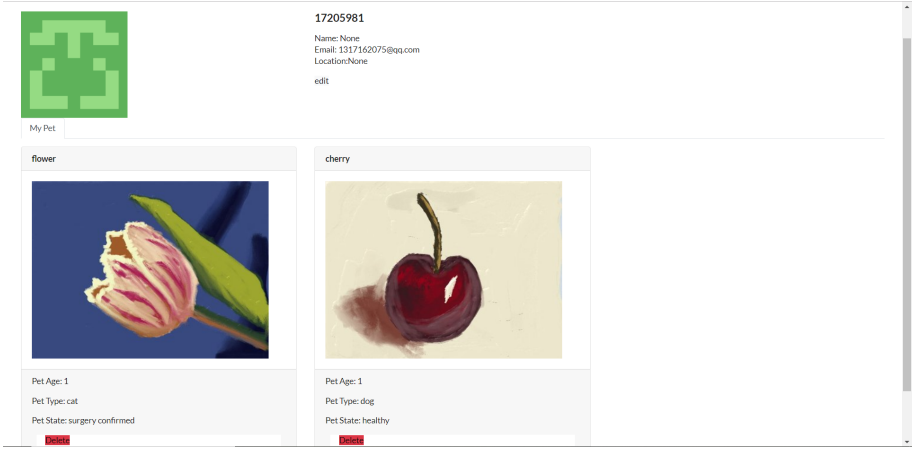
Upload also rules the size of image to make the images in the pets be normal and all the pets' images are have the same size.

```
filename_m=resize_image(f,filename,current_app.config['PHOTO_SIZE']['medium'
    ])
```

When an image putted into the database, we give the image a name and only store the name of images into the database. And when we want to show the image, we only need to read the image name in the database.

3.5.3 Module: Show Pets

In the show pets page, we think that we should not only show the information of user but also show the information of user which can be edited.



In this page, user can see the pet name, image, state and user's information on this page. And the images of users is made by the website. To achieve the reservation state of pets can be shown with pet's name and other information, we create a dictionary with key is pet and value is reservation. And actually, we put the dictionary into the website to achieve this.

```
    def get_pet_res(pets):
        if pets is None:
            return None
        else:
            dicty = {}
            for pet in pets:
                dicty[pet] = (Reservation.query.filter(Reservation.pet_id ==
                    pet.id).first())
            print(dicty)
            return dicty
```

3.6 Real-time Communication Component

3.6.1 Introduction of Flask-SocketIO

Socket.IO is an open source JavaScript library that implements real-time communication based on WebSocket. Besides supporting WebSocket, it also supports many polling mechanisms and other simulated real-time communication methods.Socket.io will automatically choose the best way to realize real-time communication according to the browser's support for the communication mechanism, so as to realize the "degraded support" of the browser.Since Flask- socket.io uses the Flask Session object internally, we want to make sure that the program sets the secret key ($SECRET_KEY$).

3.6.2 Setting Socket.IO connection

After setting up the most basic socket.io configuration, we need to establish a connection to the client, whose content will be implemented in the form of the previous client-side JavaScript.To do this, first load the socket.io resource file and then create a Socket column in the. Js file to establish the connection:

```
            var socket=io()
```

This instance will be used to execute send events, create event handlers, and so on.The io() method displays the URL of the incoming socket.io server.

3.6.3 Sending a message

In socket.IO, the data exchanged between the server and the client is called an event.The general flow of two-way communication is as follows: the client sends an event to the server by calling a emit() function and passing in data as an argument, which triggers the server to create a corresponding event handler.The server side can also send an event to the client by calling the emit() function and triggering the client's corresponding event handler.

Next, let's introduce more specifically how a news event occurs.

First step, the client sends a message.In the chatRoom.html file provides a message input box, the user by typing ENTER instead of button to send a message, the message here use socket. Emit () method to submit instead of POST or GET request.

```
var input = $('.left .input input');
var msg = $('.left .msg');
var Chat = new chatController();

input.keypress(function(e) {
  if (e.which == 13) {
    Chat.processInput();
  }
});

var socket =io()

function chatController() {
  this.processInput =  function(){
    var txt = input.val();
    input.val('');
    socket.emit('new message',txt);
  };
}
```

When the user types ENTER, the chatController() function is triggered. After the input content field is obtained through the val() method, the emit() method is called to send the event. The first parameter is passed in the event name 'new message', and the second parameter is the message content.It makes a GET request to the server.

In the second step, the server receives the message and broadcasts it.We use the on () decorator provided by Flask-socketIO to register the event handler used to receive events from the client.Create a new message event handler in the chatroom.py file to handle the new message event sent by the client.

```
@socketio.on("new message")
def new_message(message_body):
    message=Message(user=current_user(),body=message_body)
    db.session.add(message)
    db.session.commit()
    emit('new message',
             {'message_back':'{}'.format(message.body),
              'user_name':'{}'.format(message.user)},broadcast=True)
    print(message_body)
```

The parameter received by the On() decorator is the event name.The data sent by the client is passed in through the parameters of the event handler. We use the message-body parameter in the new-message() function to get the message body, then create the message object and save it to the database.Then the emit() function is called to send the event.In the emit() function, the first parameter is used to specify the event name, and when the server-side emit() function is called, the corresponding event handler in the connected client is triggered.The second parameter is the data to be sent, which is passed in the dictionary-type data according to the client's needs.

Third, the client receives the message.On the client side, we also need to create event handlers in order to receive events from the server side.Here we use the socket.on() function, whose first argument is the name of the listener event.

```
socket.on('new message',function (data){
    console.log("get")
    var newMsg = msg.clone();
    newMsg.find(".name").text(data.user_name);
    newMsg.find(".text").text(data.message_back);
```

```
            msg.parent().append(newMsg);
        })

        socket.on('user count',function (data){
             $(".number").html(data.count);
        })
```

The two socket.on() functions, one to get the new message event from the server and the other to get the user name of the message, both execute callbacks when triggered.In the client's new message event handler, we insert the HTML code for the message into the list of messages on the page using the append() method.
After that,the basic message transfer is completed.

3.6.4 Online population statistics

In chat rooms, the number of people online is a useful status message.A user is a client that connects to the socket.io client on the server side, which is called a Socket in socket.io.The number of people online is the number of clients connected to the server side.Since Flask - SocketIO does not have such functionality built in, we need to get the number ourselves.
Our component updates the online number with the built-in connect and disconnect events,the connect event are triggered when the client connects to the server.The disconnect event is triggered when the client disconnects.In addition, we use the primary key id in the User table to represent the online User.The id value can be easily obtained by extending the flask-login supplied current-user object.Our code is as follows:

```
        @socketio.on('connect')
        def connect():
            global online_users
            global online_doctors
            user = current_user()
            if user.id not in online_users:
                online_users.append(user.id)
                if user.isAdmin():
                    online_doctors.append(user.id)
            emit({'user count':len(online_users)},broadcast=True)

        @socketio.on('disconnect')
        def disconnect():
            global online_users
            global online_doctors
            user = current_user()
            if user.confirmed and user.id in online_users:
                online_users.remove(user.id)
                if user.isAdmin():
                    online_doctors.append(user.id)
            emit('user count',{'count':len(online_users)},broadcast=True)
```

On the client side, we create a user count event handler to handle the user count event sent from the server side.When triggered, it takes the count value representing the number of people online from the event data and updates it to the element with the id number:

```
        socket.on('user count',function (data){
             $(".number").html(data.count);
        })
```

3.7 Deployment and performance

3.7.1 Cloud Service: Huawei Cloud

In cloud service choice, we choose Huawei Cloud Services. Founded in 1987, Huawei is a leading global provider of information and communications technology (ICT) infrastructure and smart devices. Which are committed to bringing digital to every person, home and organization for a fully connected, intelligent world. There are nearly 194,000 employees, and it operates in more than 170 countries and regions, serving more than three

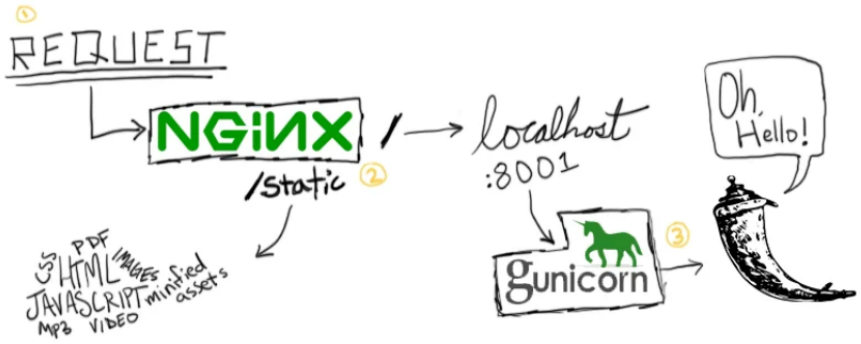
billion people around the world. HUAWEI CLOUD provides a powerful computing platform and easy-to-use development platform to support Huawei's full-stack strategy.It's no doubt that the service provider can offer solid server service to keep the whole program work in an reliable way.

3.7.2 Deployment strategy: Linux + Gunicorn + Nginx-Proxy + Eventlet

When we have a running Ubuntu operating system, we need to set up a web server on top of the operating system to serve static files - like stylesheets, Javascript files, and images - to end users. If we use nginx as our web server. Since a web server connect communicate directly with our program sever(here is Flask), the gunicorn needed to act as a medium between the server and python/flask.

Nginx is an http server. It is a high-performance http server and reverse proxy server developed in C language. Nginx is a high-performance http server / reverse proxy server and email (IMAP / POP3) proxy server. Developed by Russian programmer Igor Sysoev, the official test nginx can support 50,000 concurrent links, and the resource consumption of CPU and memory is very low, and the operation is very stable.

Well, in our case we wanted to handle WebSocket requests(chat room part). We needed support for the faster SPDY and HTTP2 protocols. We also wanted to use the Flask web framework. So, after spending some time with traditional apache, we eventually settled on NGINX open source service.



4 Conclusions

As a platform to assist the work of pet hospital, 'Healing Paws Veterinary Hospitia' reservation platform enables users to carry out user information registration, pet information registration, reservation, online consultation and communication; the staff and doctor of the pet hospital can manage and process the order through this platform to complete order. In terms of platform performance, we adopt Elastic Cloud Server (ECS) as the server of this platform, which can meet the requirements of the platform for network access capacity and data storage that can make the application environment more flexible and efficient, and ensure the server to run continuously and stably.

For the security, the data information encryption technology and access control technology is used to guarantee the security of users' and employees' personal information to a certain extent and restrict unauthorized operations. The functions of the current platform are only the function groups designed after the preliminary analysis. In the future, the functions will be modified or new functions will be developed to solve the new problems in the implementation.

5 Appendix

5.1 Deployment Project Via Linux+Nginx+Gunicorn

Introduction

In this guide, you will build a Python application using the Flask microframework on Ubuntu 18.04. The bulk of this article will be about how to set up the [Gunicorn application server](http://gunicorn.org/) and how to launch the application and configure [Nginx](https://www.nginx.com/) to act as a front-end reverse proxy.

Before starting this guide, you should have:

- A server with Ubuntu 18.04 installed and a non-root user with sudo privileges. Follow our [initial server setup guide](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-18-04) for guidance.
- Nginx installed, following Steps 1 and 2 of [How To Install Nginx on Ubuntu 18.04](https://www.digitalocean.com/commun to-install-nginx-on-ubuntu-18-04).
- A domain name configured to point to your server. You can purchase one on [Namecheap](https://namecheap.com/) or get one for free on [Freenom](http://www.freenom.com/en/index.html). You can learn how to point domains to DigitalOcean by following the relevant [documentation on domains and DNS](https://www.digitalocean.com/docs/netw Be sure to create the following DNS records:

- An A record with ‘your$_d$omain‘ $pointingtoyourserver's publicIPaddress.AnArecordwith$ ‘www.your$_d$omain‘ $pointingtoyours$ $//www.digitalocean.com/community/tutorials/how-to-set-up-uwsgi-and-nginx-to-serve-python-apps-on-ubuntu-14-04definitions-and-concepts)$ $coversWSGIinmoredetail.$

5.2 Step 1 — Installing the Components from the Ubuntu Repositories

Our first step will be to install all of the pieces we need from the Ubuntu repositories. This includes ‘pip‘, the Python package manager, which will manage our Python components. We will also get the Python development files necessary to build some of the Gunicorn components.

First, let’s update the local package index and install the packages that will allow us to build our Python environment. These will include ‘python3-pip‘, along with a few more packages and development tools necessary for a robust programming environment:

```
sudo apt update
sudo apt install python3-pip python3-dev build-essential libssl-dev libffi-
    dev python3-setuptools
```

With these packages in place, let’s move on to creating a virtual environment for our project.

5.3 Step 2 — Creating a Python Virtual Environment

Next, we’ll set up a virtual environment in order to isolate our Flask application from the other Python files on the system. Start by installing the ‘python3-venv‘ package, which will install the ‘venv‘ module:

```
sudo apt install python3-venv
```

Next, let’s make a parent directory for our Flask project. Move into the directory after you create it:

```
mkdir ~/myproject
cd ~/myproject
```

Create a virtual environment to store your Flask project’s Python requirements by typing:

```
python3.6 -m venv myprojectenv
```

This will install a local copy of Python and ‘pip‘ into a directory called ‘myprojectenv‘ within your project directory.

Before installing applications within the virtual environment, you need to activate it. Do so by typing:

```
source myprojectenv/bin/activate
```

Your prompt will change to indicate that you are now operating within the virtual environment. It will look something like this: ‘(myprojectenv)user@host: /myproject‘.

5.4 Step 3 — Setting Up a Flask Application

Now that you are in your virtual environment, you can install Flask and Gunicorn and get started on designing your application.

First, let’s install ‘wheel‘ with the local instance of ‘pip‘ to ensure that our packages will install even if they are missing wheel archives:

```
pip install wheel
```

Note

Regardless of which version of Python you are using, when the virtual environment is activated, you should use the

‘pip‘command (not‘pip3‘).

Next, let’s install Flask and Gunicorn:

```
    pip install gunicorn flask
```

Now that you have Flask available, you can create a simple application. Flask is a microframework. It does not include many of the tools that more full-featured frameworks might, and exists mainly as a module that you can import into your projects to assist you in initializing a web application.

While your application might be more complex, we’ll create our Flask app in a single file, called ‘myproject.py‘:

```
    nano ~/myproject/myproject.py
```

The application code will live in this file. It will import Flask and instantiate a Flask object. You can use this to define the functions that should be run when a specific route is requested:

```
    ~/myproject/myproject.py
    -------------------------------
    rom flask import Flask
    app = Flask(__name__)

    @app.route("/")
    def hello():
        return "<h1 style='color:blue'>Hello There!</h1>"

    if __name__ == "__main__":
        app.run(host='0.0.0.0')
```

This basically defines what content to present when the root domain is accessed. Save and close the file when you’re finished.

If you followed the initial server setup guide, you should have a UFW firewall enabled. To test the application, you need to allow access to port ‘5000‘:

```
    sudo ufw allow 5000
```

Now you can test your Flask app by typing:

```
    python myproject.py
```

You will see output like the following, including a helpful warning reminding you not to use this server setup in production:

```
   Output
    Serving Flask app "myproject" (lazy loading)
     * Environment: production
       WARNING: Do not use the development server in a production
          environment.
       Use a production WSGI server instead.
     * Debug mode: off
     * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Visit your server’s IP address followed by ‘:5000‘ in your web browser:

```
http://your_server_ip:5000
```

When you are finished, hit ‘CTRL-C‘ in your terminal window to stop the Flask development server.

Creating the WSGI Entry Point

Next, let’s create a file that will serve as the entry point for our application. This will tell our Gunicorn server how to interact with the application.

Let’s call the file ‘wsgi.py‘:

```
(myprojectenv) $nano ~/myproject/wsgi.py
```

In this file, let’s import the Flask instance from our application and then run it:

```
from myproject import app

    if __name__ == "__main__":
        app.run()
```

Save and close the file when you are finished.

5.5 Step 4 — Configuring Gunicorn

Your application is now written with an entry point established. We can now move on to configuring Gunicorn.

Before moving on, we should check that Gunicorn can serve the application correctly.

We can do this by simply passing it the name of our entry point. This is constructed as the name of the module (minus the ‘.py‘ extension), plus the name of the callable within the application. In our case, this is ‘wsgi:app‘.

We’ll also specify the interface and port to bind to so that the application will be started on a publicly available interface:

```
(myprojectenv)cd ~/myproject
   (myprojectenv)gunicorn --bind 0.0.0.0:5000 wsgi:app
```

You should see output like the

following:

```
Output
    [2018-07-13 19:35:13 +0000] [28217] [INFO] Starting gunicorn 19.9.0
    [2018-07-13 19:35:13 +0000] [28217] [INFO] Listening at: http
       ://0.0.0.0:5000 (28217)
    [2018-07-13 19:35:13 +0000] [28217] [INFO] Using worker: sync
    [2018-07-13 19:35:13 +0000] [28220] [INFO] Booting worker with pid:
       28220
```

Visit your server’s IP address with ‘:5000‘ appended to the end in your web browser again:

```
    http://your_server_ip:5000
```