System Document (draft)

Group 6

May 8, 2020

1 Abstract

The project was designed for a pet hospital called Healing Paws to meet the needs of the appointment system there. This is an online system for recording, displaying and adjusting most of the activities that take place in the chain of pet hospitals. It provides entry of account and pet information and performs online registration appointments and displays status such as appointments and surgeries to users, in addition to allowing employees to adjust the appointment information and status.

This project USES a Python based Flask framework that is lightweight, compact, and extensible. The Jinja2 architecture is used for coding and functional implementation. Jinja2 was chosen because it was more flexible. It provides control structures, expressions, and inheritance; Do not allow too much business logic to be written in templates; Templates are easy to read and easy for developers to communicate with each other. All user interfaces are implemented through JavaScript and CSS design. The flask-sqlalchemy extension is used to manage databases and provides additional options for security standards and data protection mechanisms. The application is responsible for the different modules and their associated reports, which are generated according to the applicable policies and standards proposed by management.

The development of the whole project considered the distributed client server computing technology.Eliminates all exceptions that may occur when a database transaction is performed by a general user and hospital staff.The user interface ADAPTS to various devices and provides distributed access to the entire system, making the site easily accessible to users of all devices.The internal database was chosen for SQLite because it was small and fast, and SQLite is very stable, with excellent stability due to its simplicity.Leverage the basic structure of table Spaces, clusters, and indexes to provide better consistency and reliability for data storage.SQLite is an option because it provides advanced reliability and security constructs.The entire front end is managed by the system using appropriate business rules or validation to manage data consistency.

Cloud services are adopted by ali cloud ECS. Cloud server Elastic Compute Service (ECS) is a basic cloud computing Service provided by ali cloud. Using ECS is as convenient and efficient as using water, electricity, gas and other resources. It can provide us with safe and reliable computing and data processing power.

We develop the basic functions of the website first, and then start the work on front end. Then the back end work is synchronized with the front-end work, and finally the finished product is deployed to the server. The specific implementation of all functions will be written in the document.

2 Introduction

2.1 Project Description

Healing Paws is a web-based project.The project is in the field of pet medicine.

This project is implemented and supplemented based on the functional requirements provided by users to meet the needs of the reservation system of pet hospitals. This is an online system that records, displays and adjusts most of the activity that takes place at a chain of pet hospitals.

In addition to allowing employees to adjust appointment information and status, it also provides the ability to enter account and pet information, register appointments online, and show users the status of appointments and surgeries. If someone needs to make an appointment on the pet hospital's website, this program is helpful for them, and it also provides the hospital staff with a simple, clean, visualized and strong appointment information display and adjustment function, which provides an easy way for the staff to work.

2.2 Software Solution

The Healing Paws software solution is an IT solution in a dynamic environment where business and technical strategies converge. The solution focuses on combining IT innovation with new business approaches adopted, while leveraging the organization's current IT assets. Make full use of the knowledge and available Internet resources, and implement prudent business and technical strategies in today's environment.

This project uses a Python based Flask framework that extends Flask-sqlalchemy to manage databases and provides additional options for security standards and data protection mechanisms. The application is responsible for the different modules and their associated reports, which are generated according to the applicable policies and standards proposed by management.

2.3 Function we have

Appointment for registration (emergency/ordinary)

Delete and modify customer information storage.

Show pet status (e.g. date of operation confirmed, operation completed)

The employee adjusts the appointment information or order.

Online reference services.

Chinese-English shift

2.4 Implementation

This project USES a Python based Flask framework, which itself performs only the most basic functions, so Flask is called the microFramework. It's lightweight, compact, this so-called "small", but it brings more convenience. It has a wealth of third-party libraries that make it easy to write almost any type of program, so it's extensible. The Jinja2 architecture is used for coding and functional implementation. Jinja2 was chosen because it is more flexible than template. It provides control structures, expressions, and inheritance. Unlike Mako, jinja2 only has a control structure and does not allow too much business logic to be written in the template; Jinja2 performs better than Django templates; The Jinja2 template is easy to read and easy for developers to communicate with each other. All user interfaces are implemented through JavaScript and CSS design. Advanced technologies and concepts oriented to services and components have potential to ensure the scalability of product technologies. Loose coupling between layers to make services to business processes configurable and reconfigurable.

The flask-sqlalchemy extension is used to manage databases and provides additional options for security standards and data protection mechanisms. SQLite was chosen as the internal database because it has zero configuration (no installation and management configuration); Small enough, about 130,000 lines of C code, 4.43M; Faster than some popular databases in most general database operations; SQLite is very stable, very stable because of its simplicity. SQLite is an option because it provides advanced reliability and security constructs. Leverage the basic structure of table Spaces, clusters, and indexes to provide better consistency and reliability for data storage. The development of the whole project considered the distributed client server computing technology. Eliminates all exceptions that may occur when a database transaction is performed by a general user and hospital staff. The user interface automatically ADAPTS to various devices, provides the corresponding layout for the device, enables users of all devices to easily access the site, reduces development and maintenance costs, and can be found and dealt with as soon as possible when problems arise. And provide distributed access to the entire system. The entire front end is managed by the system using appropriate business rules or validation to manage data consistency.

Adopt ali cloud ECS cloud service.Compared with ordinary IDC machine rooms and server manufacturers, aliyun USES more stringent IDC standards, server access standards and operation and maintenance standards to ensure the high availability of cloud computing infrastructure, data reliability and cloud server high availability. Aliyun has passed a variety of international security standards certification, including ISO27001, MTCS. The biggest advantage of cloud computing is its flexibility and flexibility. The elasticity of aliyun is the elasticity of computing, the elasticity of storage, the elasticity of network and the elasticity of business architecture re-planning. Using ECS is as convenient and efficient as using water, electricity, gas, and other resources. It can provide us with safe and reliable computing and data processing power.

2.5 Deploy

In order to realize the deployment of the application in the cloud server, we use the combination of nginx and gunicorn to realize the application forwarding on the server. We use SSH protocol remote control server completes

the configuration of the environment(Install python,virtualenv,flask,Nginx,gunicorn and flask requirements).To be able to use nginx and gunicorn, we opened port 80 and 5000 (show as follow):

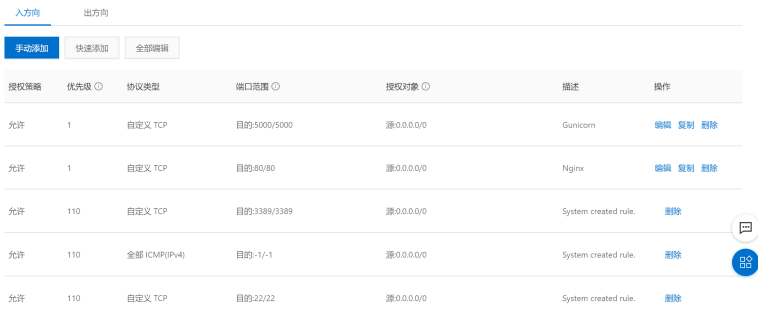

Figure 1: Port

Then change nginx's configuration as follows:

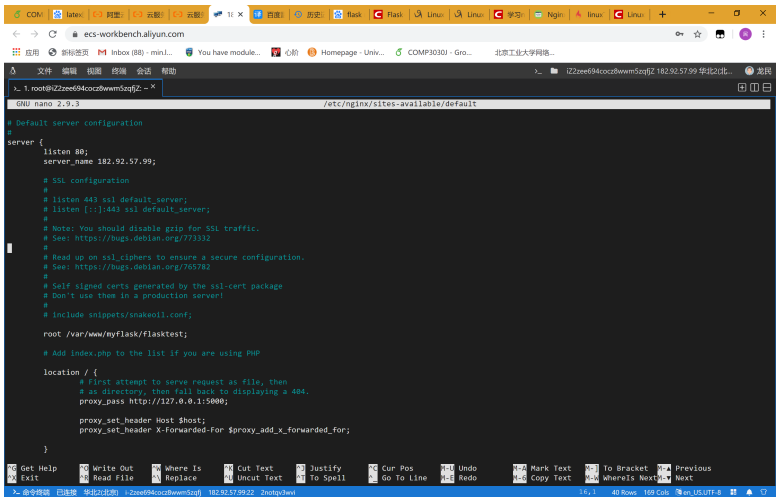

Figure 2: Caption

After start the gunicorn in the virtualenv, now the website can be accessed on the public network by the IP address 182.92.57.99 (We haven't prepared the domain name yet).

2.6 Feature

Security:

Flask's framework was used to provide security against cross-site scripting (XSS), cross-site request forgery (CSRF), json data transfer, and more.

And we use the embedded SQLite database, SQLite USES the standard C to achieve its security mechanism. By using text file to encrypt database data, the audit function is realized. Through the way of file copy to achieve database data backup and recovery; The encryption mechanism for SQLite is implemented by using the sequence encryption algorithm RC4. In addition, we used aliyun server, which can effectively prevent MAC spoofing and ARP attacks, effectively protect against DDoS attacks, can carry out traffic cleaning and black hole, provide port intrusion scanning, hanging horse scanning, vulnerability scanning and other additional services.

Reliable:

Flask has been developing for nearly a decade, during which time it has evolved into a mature, robust framework. Flask's stability and reliability are guaranteed by its increasingly sophisticated expansion package and simple design concept.

For SQLite, it takes the form of constraints to increase the stability of the database. Constraints are rules that are enforced on the data columns of the table to limit the types of data that can be inserted into the table, which ensures the accuracy and reliability of the data in the database.

We use ali cloud server is also contributing to the system's reliability, ali cloud server business built on the proprietary network, and network infrastructure will constantly evolve, make every day have updated the network architecture and function of the network, to make our business always remain in a stable state. For the peak period in daily use, if in the ordinary IDC mode, it is impossible to prepare resources immediately; Cloud computing, on the other hand, can help us get through these peaks in a flexible way. When business peaks disappear, excess resources can be released to reduce business costs. With the horizontal expansion and reduction, and with the elastic expansion of ali cloud, it is possible to achieve a fixed time and quantitative expansion, or according to the load of the business expansion.

Laconic:

The interface of our website adopts the simple and beautiful design style, and USES the elements of blood and many pets in the main page, so as to meet the use environment of the pet hospital.

There are some pictures shows style of our interface:

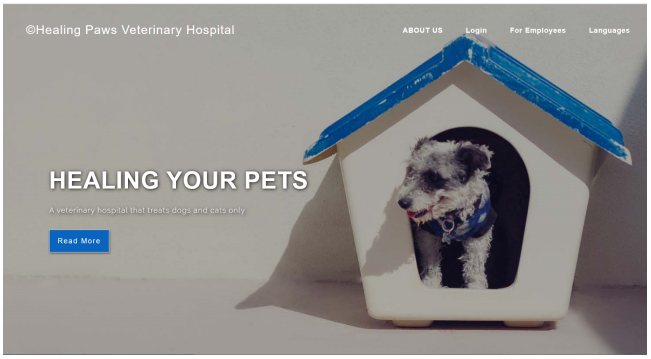

Figure 3: meet the use environment

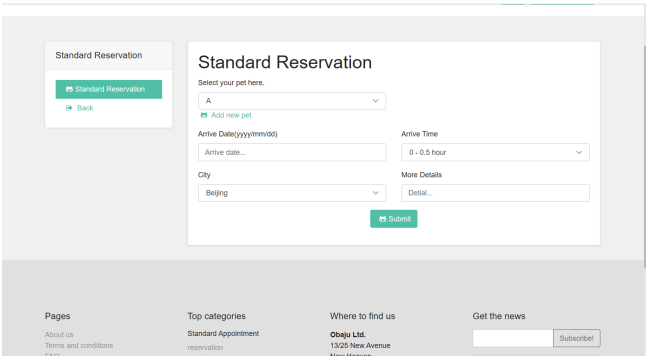

Figure 4: simple and beautiful

3 Process

3.1 Overall introduction

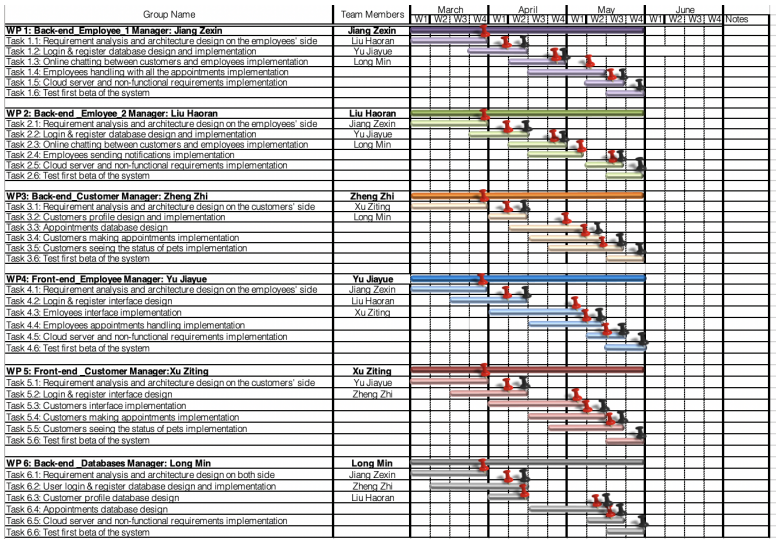

Figure 5: Gantt chart

The total duration of our project is sixteen weeks, the analysis, design and implementation works are in the first twelve weeks (as shown in the chart above). The first month, we did the analysis according to the user's requirement (problem statement). Then from week five to week six, we designed our online hospital system. From week seven till now, we have been focusing on implementing our system. In the following parts, I would introduce the process of analysis, design and implementation of our process.

3.2 Analysis

According to the user's requirement (problem statement), we divided the requirements into two parts, non-functional requirements and functional requirements.

Firstly, we analyzed the non-functional requirements. The user requires that the system should work on both mobile and PCs. Therefore, the system should be cloud-based. Moreover, the user requires that the system should be secure, reliable and stable. In addition, the user requires that the system should be accessible in many parts of the world, such as in Beijing and Dublin.

Secondly, we analyzed the functional requirements. In the online system, the user requires that the user interface of customer and employee should be separated, there should be two portals- one for customer and one for employee. For the customer, when making an appointment, the appointment should be distinguished by regular appointment and emergency appointment, and one customer should be able to make appointments for multiple pets at one time Furthermore, the customer should be able to see the status of their pets, for example, if the pet is under surgery or ready to release, etc. For employees, they should be able to organize and prioritize the pets which appointments have been made by their owners. Besides, employees should be able to handle emergencies immediately even if there are regular appointments at the same time. Moreover, customers and employees should be able to online chat through the system.

3.3 Design

After the analysis of the user's requirements, we think that the best option of our system is a cloud-based web application. We decided to flask as the frame work of our system, because flask have several advantages. Flask is free, flexible, and extensible, with a wide selection of third-party libraries that combine with the most popular and powerful Python libraries. Therefore, flask is ideal for small-scale web application like our system.

We divided our website into two parts, the front end and the back end. In designing the front end, we looked for some online booking sites for inspiration and sketch out the main pages. Then, we planed to use pictures from the Internet to beautify the pages.

In designing the back end, firstly, we drew use case diagrams for customers and employees separately to list the functions that we are going to implement.

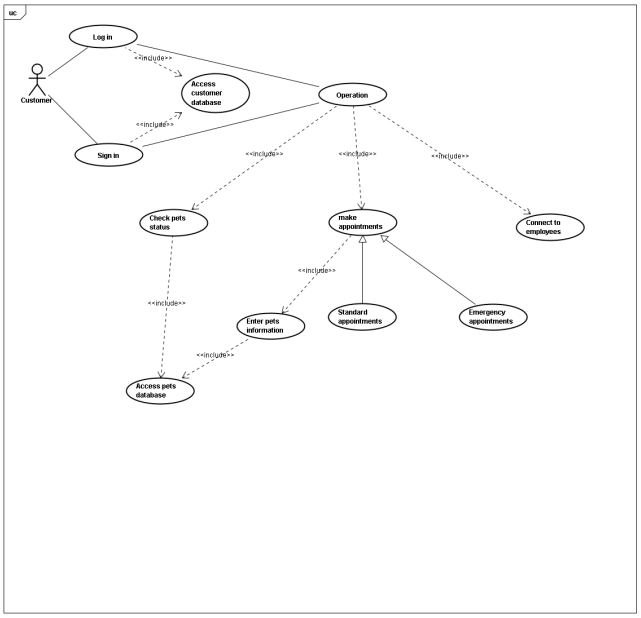

Figure 6: Customer use case

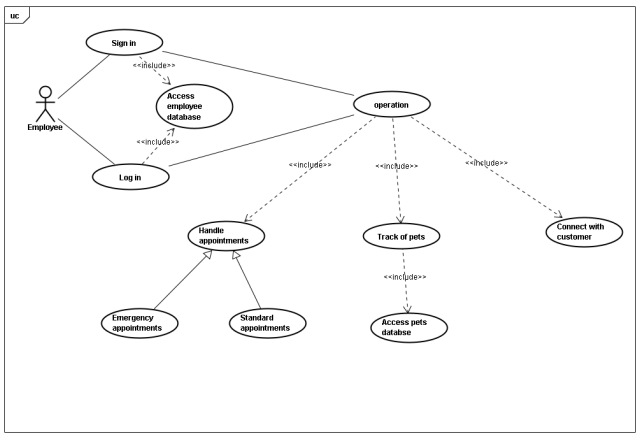

Figure 7: Employee use case

When designing the back end of the system, we focused on what the functions should the system implement. We plan to set up two entrances on the main interface, one for employees and one for customers. For both customers and employees, login and register should be achieved.

According to our design, in the customer interface, the user should be able to make regular appointment and emergency appointment, check the status of pet treatment, check the record of pets treatment, modify account information including personal information and password, add and modify pets information and online consult.

In the employee interface, the user should be able to view all the appointments and sort them by day, week and month, Sort pets according to different search criteria such as pet name and id, change pet status, change appointment status and sort appointments according to priorities. Moreover, user should be able to reply to those who had online consultation.

3.4 Implementation

Our implementation process is roughly divided into three phases: front end implement, back end implement and cloud deploy.

Phase 1: Front end implementation

We used HTML, CSS and JavaScript to generate our pages. We used HTML to create the basic framework for web page styles. Then we used CSS to decorate the pages and achieve some special effects to make the page more diverse. Then we used JavaScript to allow user to interact with the system and allow system to do input

validation. We also found some pictures from the Internet to beautify our UI.

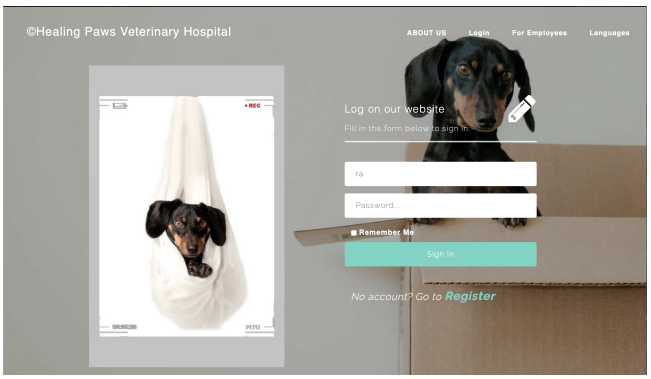

Figure 8: Main UI

This is the main page of our system. The system automatically determines whether the type of account entered by the user is a customer or an employee, and then enters different interfaces.

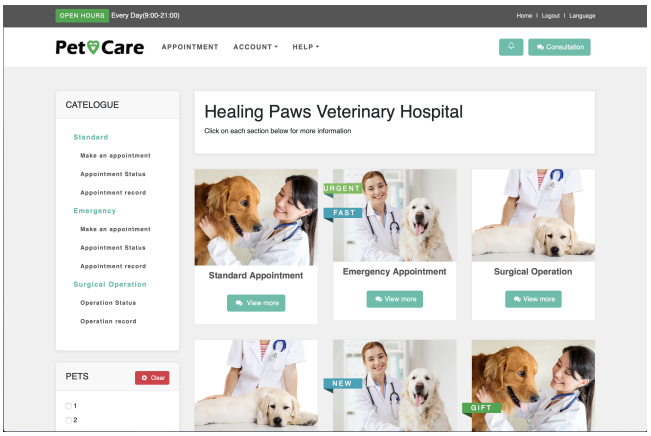

Figure 9: Customer UI

This is the main interface for customer.

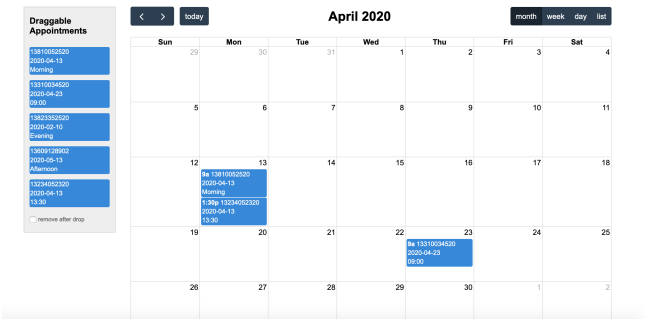

Figure 10: Employee UI

This is the main interface for employee.

Phase 2: back end implementation

As mentioned before, we are using flask as our framework. Its WSGI toolkit uses Werkzeug (routing module) and its template engine uses Jinja2. Werkzeug allows user route pages through URL, which guarantees access speed. Jinjia2 guarantees the rendering speed of the templates through pre-compiling python source code.

In order to store user's data, we designed databased for customers and employees. For database, we chose SQLite. SQLite is a lightweight embedded in-process database whose database is just a file that implements a self-contained, serverless, non-configuration, transactional SQL database engine. It is simple and small, very

suitable for our network system. In addition, In order to achieve the function of Chinese and English conversion, we created a separate database to store the corresponding Chinese and English terms, when the user needs the language conversion, the system can directly call the corresponding language from the database.

To meet the security requirements, we used hashcode when storing user's account password. The Hash algorithm converts the target text into an irreversible Hash string of the same length and stores it in the database, which means people have access to the database cannot get user's password, which guarantees the security of user's accounts.

We implemented most of the functions of our system during week 7 to week 11.

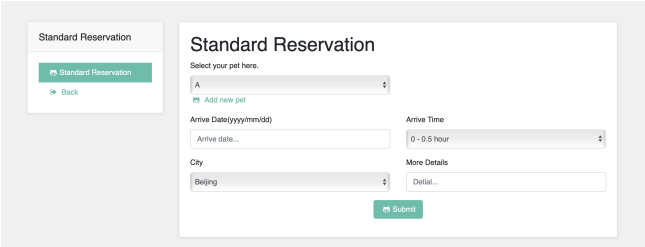

Figure 11: Make Appointment

Customer can make regular and emergency appointment.

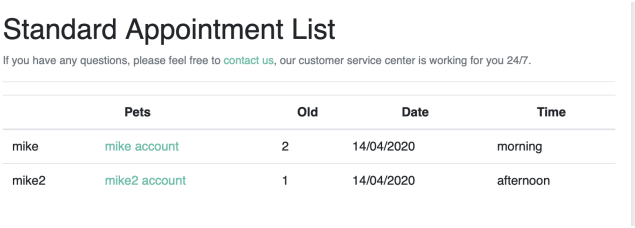

Figure 12: Appointment list

Customer can see all their appointments in a list.

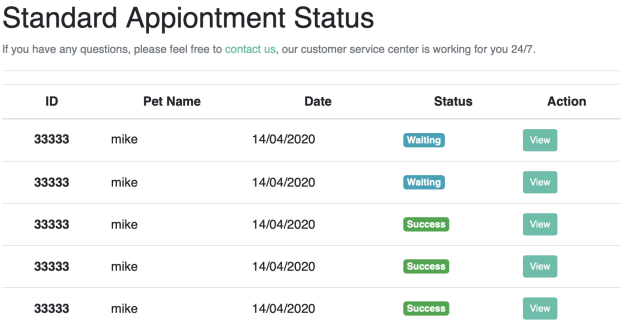

Figure 13: Pet status

Customer can check the status and treatment record of their pets.

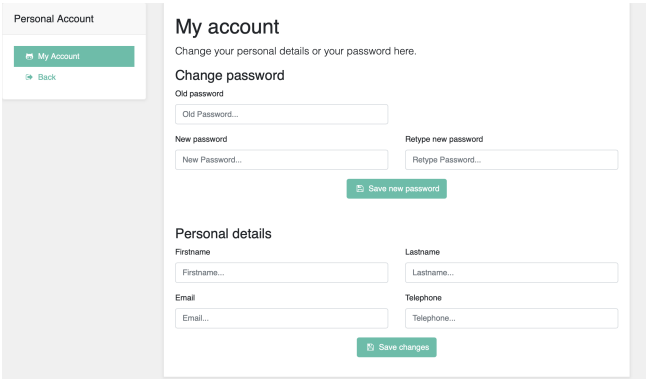

Figure 14: Change personal information

Customer can change their personal information.

Figure 15: View appointments

For employees, they can see all the appointments in a list and search, sort them by different criteria.

Phase 3: Cloud deployment

In week 10, we deployed our web application on Ali Cloud server. The domain name of our site is 182.92.57.99 and we have tested that our site worked normally in both China and Europe.

3.5 Labor division

Front-end employee manager: Yu Jiayue

· Requirement analysis and architecture design on the employees' side

· Login and register interface design

· Employees interface implementation

· Employees appointments handling implementation

· Cloud server and non-functional requirements implementation

· Test first beta of the system

Front-end customer manager: Xu Ziting

· Requirement analysis and architecture design on the customers' side

· Login and register interface design

· Customers interface implementation

· Customers making appointments implementation

· Customers seeing the status of pets implementation

· Test first beta of the system

Back-end Employee1 Manager: Jiang Zexin

· Requirement analysis and architecture design on the employees' side

· Login and register database design and implementation

· Online chatting between customers and employees implementation

· Employees handling with all the appointments implementation

· Cloud server and non-functional requirements implementation

· Test first beta of the system

Back-end Emloyee2 Manager: Liu Haoran

· Requirement analysis and architecture design on the employees' side

· Login and register database design and implementation

· Online chatting between customers and employees implementation

· Employees sending notifications implementation

· Cloud server and non-functional requirements implementation

· Test first beta of the system

Back-end Customer Manager: Zheng Zhi

· Requirement analysis and architecture design on the customers' side

· Customers profile design and implementation

· Appointments database design

· Customers making appointments implementation

· Customers seeing the status of pets implementation

· Test first beta of the system

Back-end Databases Manager: Long Min

· Requirement analysis and architecture design on both side

· User login and register database design and implementation

· Customer profile database design

· Appointments database design

· Cloud server and non-functional requirements implementation

· Test first beta of the system

4 Group Work

4.1 Communication

There are six members in our group. As the team aggrement regulates, communications between team members should be through GitHub, overleaf, Wechat group, and weekly online or face-to-face meetings. Code and written documentation tasks should be uploaded onto GitHub and overleaf respectively. Due to the current situation, there would be no face-to-face meetings temporarily. Therefore, group members will check their Wechat group messages at least twice a day and reply when necessary. What's more, The team members assigned to the relevant content will have a more detailed discussion in private. After reaching a conclusion, they will formulate the conclusion in the wechat group and listen to others' opinions and suggestions. We have also adopted a more scientific approach to collaboration. Everyone has their own branch on GitHub. Only when confirmed by other team members, can they merge their branch into master branch. This way is conducive to the control of code version and remote cooperation.

4.2 Meeting and decision making

About the decision we have made, each idea has been discussed and understood by all members of the group before a final consensus decision is reached. It aims to make every members get benifits from the process we discussed. Sometimes there are conflicts between team members, we would use Tencents meeting where members could express their opinions by combining text, graphics and video. If we still can't agree on a problem, we will choose to record the problem and give feedback to the TA on weekly update for help. In the process of this discussion, team members need to improve their ability to express their opinions clearly and convince others, as well as their ability to accept others' opinions. This not only gives us the opportunity to learn new knowledge, but also exercises our ability of teamwork.

There is a regular meeting time every week with TA participate in. Meetings online are scheduled every Tuesday morning at 10 am. During this period, we will talk with the TA about the problems we encountered in the development of the previous week and the tasks we need to complete in the new week. The TA will also give appropriate guidance and suggestions based on our current situation. At the end of the meeting with the TA, the team members will stay to make a more specific division of the week's work and distribute it evenly to everyone. If a member cannot attend the weekly meeting, they must communicate to all members in the Wechat group 6 hours prior to the meeting. And after the meeting, through other team members, understand the contents of the meeting and the tasks they need to complete.

5 Technical implementations

5.1 Basic programming strategy

On the frontend, we used Bootstrap for the design of UI. We also plug in a FullCalendar API and implemented ajax on the javaScript part for a better user interaction with the web.

On the back-end, as it is mentioned before we decide to use the Flask web framework with Python Language, including the SQLite database engine on Flask-SQLalchemy extension which is an ORM to help the system to store and call the data, The WTForms provided by Flask to collect the imput information for users and so on.

This is our ER diagram for the database implementation:

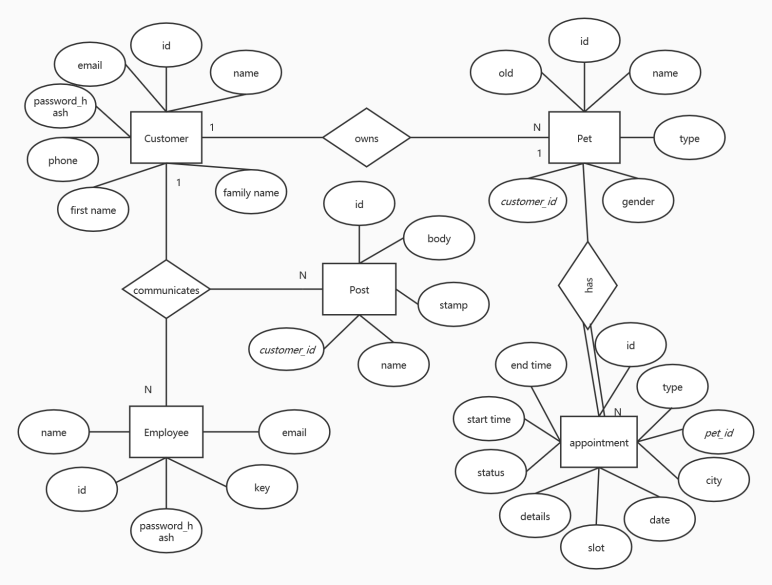

Figure 16: ER diagram

5.2 Employee side

On the Employee side there are three main functions: organizing and prioritizing and keeping track of all the appointments, assigning status and communication with customers.

5.2.1 Manage appointments

We use a full calendar for appointment organizing, prioritizing and track keeping. The FullCalendar is a JavaScript plugin with a great display of the events on the calendar. To manage all the appointments made by the customers, a full calendar is an intuitive way for employees to see whether a date is occupied or not. Therefore, we used the FullCalendar API on https://fullcalendar.io to build a calendar for appointments organizing and prioritizing.

First, we initialized a list of external events to the left side of the calendar by selecting the list id to create new event objects, adding titles for each eventData.

```
var containerEl = document.getElementById('external-events-list');
    var eventEls = Array.prototype.slice.call(
      containerEl.querySelectorAll('.fc-event')
    );
    eventEls.forEach(function(eventEl) {
      new Draggable(eventEl, {
        eventData: {
          title: eventEl.innerText.trim(),
        }
      });
    });
```

Then for the initialization of the full calendar, we set the header information of the calendar, specifying the left, center and right position of the header. The view of Month, Week and Day are all included in the API which can be taken for use directly. The navLinks are set true for clicking on a certain day name and then being navigated to that view. The editable and droppable attribute are set to be true for adding appointments onto the calendar and editing the time slot. For cancelling the appointment, we will add this functionality through the FullCalendar's API 'removeEvents' (which we have not completed and will finish this later). We may also add 'theme' which means we would add some related jQuery UI documents to beatify our calendar(which we have not completed and will finish this later). The drop function allows each dragged event to be settled onto the calendar and can be removed from the external list using the removeChild method. This will leave the appointments out that cannot be organized to the desired time.

```
var calendarEl = document.getElementById('calendar');
    var calendar = new Calendar(calendarEl, {
      plugins: [ 'interaction', 'dayGrid', 'timeGrid', 'list' ],
      header: {
        left: 'prev,next today',
        center: 'title',
        right: 'dayGridMonth,timeGridWeek,timeGridDay,listWeek'
      },
   navLinks: true, // can click day/week names to navigate views
      editable: true,
      droppable: true, // this allows things to be dropped onto the calendar
      drop: function(arg) {
        // is the "remove after drop" checkbox checked?
        if (document.getElementById('drop-remove').checked) {
          // if so, remove the element from the "Draggable Events" list
          arg.draggedEl.parentNode.removeChild(arg.draggedEl);
        }

      }
    });
    calendar.render();
```

The backend implementation of appointment data to be stored on the calendar has not completed yet. We plan to store the specific time data that the employees have organized into the database first. Then by using the sqlalchemy queries, the data can be displayed on the calendar by binding the eventData with json on the frontend. We will achieve this function later and write detailed implementation in the system document.

5.2.2 Assign appointment/surgery status

The table displays a full list of all the appointment details in the database to the employee. It is after the employee has organized the calendar and assigning different appointment/surgery status.

In order to show all the appointments in a table in the employees interface, we imported bootstrap-table library. The bootstrap table has a fixed header and a very rich set of configuration parameters, which makes the system very convenient for employees to search, sort and manage the appointments. In addition, bootstrap-table is self-adapting, which means it can be used in any browser for mobile and PC terminals.

```
<div class="wrapper">
    <div class="container" style="padding: 3em;">
        <div class="row">
            <div class="col-md-15">

                <div class="fresh-table full-color-blue">
```

The bootstrap table is capable of displaying JSON data directly into a table. In our Bootstrap-table model, calling data from the database into the table which means loading the data to be displayed into the table in the server all at once, then converting it to JSON format and sending it to the interface to be displayed, and then forwarding it to the specified interface(the employee interface). Acquire data using the EL expression when the interface is initialized, and then call the back-end related initialization function, and pass in the JSON string for display.

Up to now, the frontend API is already applicable and we are improving the jsonify part for status data changing and notification sending to customers.

5.2.3 Communication

A question post is always made by a customer and received by all the employees. Therefore, a foreign key(user-id) references to the Customer table. Message body, time of messaging and name of the sender are stored as attributes in this table. The representation of a Post entity can be expressed as a message like: 'Lily: Hello! (10:27)'.

```
class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    body = db.Column(db.String(140))
    timestamp = db.Column(db.DateTime, index=True, default=datetime.utcnow)
    name = db.Column(db.String(140))
    user_id = db.Column(db.Integer, db.ForeignKey('customer.id'))

    def __repr__(self):
        return '{}: {}    ({})'.format(self.name, self.body, str(self.
            timestamp)[11:16])
```

On the customer side, each message a customer send is stored in the Post database. For messages that are related to a certain customer, which include questions he asks and direct replies from the employees, are shown on the chatting page. This is achieved by using a query for filtering the correct foreign key id to select messages related to each customer.

```
if form.validate_on_submit():
    body = form.postbody.data
    user_in_db = Customer.query.filter(Customer.username == session.get("
        USERNAME")).first()
    post = Post(body=body, customer = user_in_db, name=session.get("USERNAME"
        ))
    db.session.add(post)
    db.session.commit()
    return redirect(url_for('customer_chatting'))
else:
    user_in_db = Customer.query.filter(Customer.username == session.get("
        USERNAME")).first()
    prev_posts = Post.query.filter(Post.user_id == user_in_db.id).all()
    print("Checking for user: {} with id: {}".format(user_in_db.username,
        user_in_db.id))
    return render_template('customer_chatting.html', title='Online chatting',
         prev_posts=prev_posts, form=form)
```

On the employee side, they are using a form that is slightly different from the customers'. They can specify which customer they are sending message to and their post will be stored into that customer's post database. Employees can see the posts from all the customers. This is achieved by selecting all the data from the Post table.

```
@app.route('/employee_chatting', methods=['GET', 'POST'])
def employee_chatting():
    form = PostForm2()
    if not session.get("USERNAME") is None:
        if form.validate_on_submit():
            body = form.postbody.data
            who=form.who.data
            user_in_db = Customer.query.filter(Customer.username == who).first
                ()
            post = Post(body = body, customer = user_in_db, name=session.get("
                USERNAME"))
            db.session.add(post)
            db.session.commit()
```

```
            return redirect(url_for('employee_chatting'))
        else:
            prev_posts = Post.query.all()
            return render_template('employee_chatting.html', title='Online
                chatting', prev_posts=prev_posts, form=form)
    else:
        flash("User needs to either login or signup first")
        return redirect(url_for('login'))
```

5.3 Customer side

First of all, we need to meet the requirements of customers. According to the statement of problems, the functional requirements are mainly divided into four parts: the function of customer login and registration, the function of emergency reservation and standard reservation for pets, the function of customers seeing the status of pets, and the function of customers asking questions and getting answers from employees. These functions are the main functions that need to be implemented, and there are some details that need to be implemented after the main functions are completed. After analyzing the problem statement, we draw the use case diagram of the client as figure 6.

After analyzing the problem statement in detail and listing the client functions to be completed. We started to draw a sketch of the front-end interface of customers. After looking up several online booking English courses and other booking website interfaces for reference, we initially drew a sketch of four interfaces, namely, the main index interface, the user login interface, the user registration interface and the customer index interface which is for user to make an appointment. Other pages are added after the main page is completed.

After a brief comparative study of several front-end frameworks, such as Vue and react, we choose bootstrap to develop our website. Because after analysis, using bootstrap has several advantages. All mainstream browsers support bootstrap, which is easy to use. Because we have learned basic HTML, CSS and JS, we can learn bootstrap quickly and start to practice as soon as possible. At the same time, bootstrap provides a simple and unified solution for developers to create interfaces, which includes powerful built-in components and is easy to customize. It also provides Web based customization and is open source.

Therefore, considering the simplicity of the project page function and other factors, we choose bootstrap to create a web project.Then we start to write HTML and CSS of several main interfaces. According to the original basic structure of bootstrap, the global CSS settings, basic HTML element styles and extensible classes will advance our web pages under the HTML and CSS coding specifications specified by bootstrap. In addition to using the basic CSS layout, table and form layout of bootstrap, we also refer to more than ten reusable components of bootstrap to beautify our interface, such as font icons, pull-down menus and input box groups and so on.

According to the draft, the main interface, login registration and appointment interface are well typed with HTML and CSS. We refer to more than ten customized jQuery plug-ins included in bootstrap, and using the compressed version of Bootstrap.min.js ensures that we can refer to the plug-ins we want to refer to. For example, in the top navigation bar, we refer to the plug-ins of the drop-down menu, and in the help interface, we refer to the folding plug-ins and so on.

In addition to the basic login registration and appointment interface, we have added the personal account management and pet account management interface, in order to facilitate the hospital to contact customers in a timely manner and customers do not need to fill in duplicate information for each appointment. These two pages are based on the CSS style of the previous main interface. Because both of these interfaces have forms to fill in, we use jQuery to verify whether the form content conforms to the format, such as whether the age is a positive number, whether the email and phone formats are correct, etc. We also use some other jQuery plug-ins, such as the jQuery.cookie.js to read, write and delete cookies, and the datepicker plug-in to achieve the function of customers' booking any time (which is not completed) and so on.

In order to make the interface beautiful and concise, we used the combination of font awesome (icon library) and bootstrap. At the same time, we have a unified setting for the font and color configuration of the page, with green as the main color, representing the meaning of treatment and rescue. We also selected some special art fonts, such as Poppins regular, to conform to the style of the web page and make the user more intuitive.

Because our back end uses the flash framework, we need to use the jinja2 template engine to render templates, reduce the complexity of using python, and separate the presentation logic from the business logic. The main customer page serves as the base template, appointment interface, status display interface, account management interface and other extension base templates as the derived templates.

5.3.1 Add an appointment

There are two different appointments that customers can add: Standard and Emergency. Because two types of appointments have different information required, so we divide two appointments into two pages, which is convenient for customer to choose the type and input different information. for example, emergency appointment should not provide date for user because it must be today but standard should, and standard appointment could choose morning, afternoon or evening to have free time so that we can handle it more accurately, but emergency does not need it.

However, in order to simplify the process of staff side calling different data and then integrating them, we arranged two kinds of appointments on two pages respectively for appointment on the front end, but at the back end, we realized data integration by inputting them into a database.

Here is the Appointment table Model:

```
class Appointment(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    type = db.Column(db.String,index=True)
    date = db.Column(db.DateTime, index=True, default=datetime.utcnow())
    time = db.Column(db.String(64), index=True)
    city = db.Column(db.String(64), index=True)
    details = db.Column(db.String(120), index=True)
    pet_id = db.Column(db.Integer, db.ForeignKey('pet.id'))
    start = db.Column(db.DateTime, index=True, default=datetime.utcnow())
    end = db.Column(db.DateTime, index=True, default=datetime.utcnow())
    status = db.Column(db.String(64), index=True, default='appointment
        submitted')
```

And Here are the routes:

```
app.route('/reservation_e', methods=['GET','POST'])
def reservation_e():
    user = {'username': 'User'}
    form = REForm()
    if not session.get("USERNAME") is None:
        if form.validate_on_submit():
            customer_in_db = Customer.query.filter(Customer.username == session
                .get("USERNAME")).first()
            emergency_appointment = Appointment(type='Emergency', time=form.
                time.data , city=form.city.data, details=form.detail.data,
                pet_id=form.pet_id.data)
            db.session.add(emergency_appointment)
            db.session.commit()
            return redirect(url_for('customer_index'))
        return render_template('reservation_e.html', title='reservation', user
            =user, form=form)
    else:
        flash("User needs to either login or signup first")
        return redirect(url_for('login'))


@app.route('/reservation_s', methods=['GET','POST'])
def reservation_s():
    user = {'username': 'User'}
    form = RSForm()
    if not session.get("USERNAME") is None:
        if form.validate_on_submit():
            customer_in_db = Customer.query.filter(Customer.username == session
                .get("USERNAME")).first()
            standard_appointment = Appointment(type='Standard', time=form.time.
                data, city=form.city.data, details=form.detail.data, date=form.
                date.data, pet_id=form.pet_id.data)
            db.session.add(standard_appointment)
```

```
            db.session.commit()
            return redirect(url_for('customer_index'))
        return render_template('reservation_s.html', title='reservation', user
            =user, form=form)
    else:
        flash("User needs to either login or signup first")
        return redirect(url_for('login'))
```

As you can see in the code, we set a "type" attribute to distinguish between standard and emergency: when the customer submit a standard appointment, the system will automatically add the "Standard" type to this piece of information and add it into database and the emergency is the same. Such storage greatly facilitate the employee the call and make unified operation, because in the setting beginning and end of the time operation and changing status operation on the employee side, the two different appointments are actually the same entity, so for employees side, get a copy of such data is more conducive to the next operation, at the same time, also won't affect the future calls on the both side.

The start time, end time and status attributes are not required for customer, so when add an appointment these information should not be input in and the only thing to do is just set a default. The default of the status is "appointment submitted", which is the first status of all the appointment. On the Employee side the status can be changed by users, and the two times will also be set by employee on the timetable.

5.3.2 Manage accounts

About managing accounts, actually every customer needs to manage two kinds of information: one is the personal information such as name, phone number, password, and the other is the animal information, which contains name, old, gender,type of every pet. So at the top of the customer web page, we set a overall option called Account and there are two sub options: personal account and pet account.

The personal account has two functions: manage the personal details and change the password. Firstly, personal details. We provide several forms to hold all the required information including first name , last name, phone number and email. If it is the first time you come in this page, your personal information should be blanks. Once you submit one time, you may find that all the information you put in is in the form, means that you can check what you submit and change over and over again.

For the password changing, this feature requires more back-end content than personal information, because the process of password change requires first verifying that the old password is entered correctly, and second that the new password is entered twice and conforms to the password rules. After all conditions are complete, changing the password could be successful. So there's a little bit of validation involved in this functionality, and for a better customer experience, we might use JavaScript and JQuery. At present, the most basic tips and verification have been completed, and we will use the remaining time to improve the convenience of customers.

```
@app.route('/customer_password', methods=['GET', 'POST'])
def customer_password():
    user = {'username': 'User'}
    form = CustomerPasswordForm()
    if not session.get("USERNAME") is None:
        if form.validate_on_submit():
            customer_in_db = Customer.query.filter(Customer.username == session
                .get("USERNAME")).first()
            if not (check_password_hash(customer_in_db.password_hash, form.
                password.data)):
                flash('Incorrect Password')
                return redirect(url_for('customer_password'))
            if form.password2.data != form.password3.data:
                flash('Passwords do not match!')
                return redirect(url_for('customer_password'))
            else:
                customer_in_db.password_hash = generate_password_hash(form.
                    password2.data)
                db.session.commit()
                return redirect(url_for('customer_index'))
        return render_template('customer_password.html', title='pet', user=
            user, form=form)
```

```
    else:
        flash("User needs to either login or signup first")
        return redirect(url_for('login'))
```

As we can see the route, we firstly check if the the old password is correct, so we need to check the input password with the hash code in database using check-password-hash function. Then we judge these two new password whether they are the same or not. After two checking, finally the new password is legal and we will change it to hash code and save in the related place in database.

5.3.3 Check appointments

Besides inputting information, there are still some functions that only need to call the data from database. For customers, it is needed to check their appointments with the status of every appointment, and the arranged time by employees for the appointment, which decide the arriving time or communicate with employees when time conflict. There is also a requirement for customer to check their appointments which are finished before, helping them to record the health statement of their pets and convenient to check if needed in the future.

Because we divide the appointment into two different types, so for both status and record we also set two types under the two appointments sections. So totally there are four related functions: standard appointment status, standard appointment record, emergency appointment status, emergency appointment record. Because these four functions have the similar functionality, so the implementation of them are mostly the same.

Now we use standard appointment status as an example to show how these kind of functions are achieved. First, we should reserve a table to display the data in the front HTML file. The table is blank.Then through the back-end route database operation, such as join, get the list of required information, and then pass it to the front-end table to complete the final display.

Here is the front-end code:

```
{% for post in prev_posts %}
                    <tr>
                      <th>{{post.Pet.id}}</th>
                      <td>{{post.Pet.name}}</td>
                      <td>{{post.Appointment.date}}</td>
                      <td><span class="badge badge-info">{{post.Appointment.
                          status}}</span></td>
                      <td>< a href=" " class="btn btn-primary btn-sm">View</
                          a></td>
                    </tr>
                    {% endfor %}
```

As we can see in the code, we need a loop to transferring several rows of the data, and in each ¡td¿, we set what information will be showed in this blank so that the rest of unusable information would not be seen by customer.

Here is the back-end code:

```
@app.route('/status_a')
def status_a():
 user = {'username': 'User'}
 if not session.get("USERNAME") is None:
  customer_in_db = Customer.query.filter(Customer.username == session.get("
      USERNAME")).first()
  pet_in_db = Pet.query.filter(Pet.customer_id == customer_in_db.id).all()
  prev_posts = db.session.query(Pet, Appointment).filter(Pet.id ==
      Appointment.pet_id ).all()
  # a = prev_posts.filter(Pet.id == pet_in_db.id).all()
  return render_template('status_a.html', title='record', user=user,
      prev_posts=prev_posts )
 else:
  flash("User needs to either login or signup first")
  return redirect(url_for('login'))
```

At the back-end, there are some queries using to find the useful data. in SQLalchemy it is presented by different type with MySQL that we learned before. After several selecting, we can find the list that contains all the required information, then transfer this to the front-end HTML page.

5.4 Other implementation

5.4.1 Ajax technique

The ajax technique can be used for the form validation and notification.

We have already implemented the form validation. This is an example of validating user registration field:

```
$.post('/checkuser', {
      'username': chosen_user.val() //field value being sent to the server
   }).done(function (response){
      var server_response = response['text']
      var server_code = response['returnvalue']
      var u=/^[a-zA-Z0-9_]+$/;
      if (server_code == 0 && u.test(chosen_user.val()) == true){ // success
          : Username does not exist in the database
         $("#email").focus();
         $("#checkuser").html('<span>' + server_response + '</span>');
         $("#checkuser").addClass("success");
      }else if (server_code == 0 && u.test(chosen_user.val()) == false) {
         chosen_user.val("");
         chosen_user.focus();
         $("#checkuser").html('<span>' + "Username format is not correct" +
             '</span>');
         $("#checkuser").addClass("failure");
      }else { // failure: Username already exists in the database
         chosen_user.val("");
         chosen_user.focus();
         $("#checkuser").html('<span>' + server_response + '</span>');
         $("#checkuser").addClass("failure");
      }
   }).fail(function() {
      $("#checkuser").html('<span>Error contacting server</span>');
      $("#checkuser").addClass("failure");
   });
```

We will complete the notification part using this technique as well.

5.4.2 Work on different platforms

The basis of bootstrap's adaptive function is the "fence" mode, which divides the browser in the form of rows and columns: there are 12 columns in total, and the number of rows is customized. According to the elements you want to display, determine the display size of each element, that is, the number of columns you need. If it exceeds the range, it will automatically change rows. The size of each column is automatically averaged by bootstrap based on the size of the current browser. According to the size of the browser, bootstrap has four types of fence class names, which are the same as the CSS style sheet class name selector style call

xs: col-xs-1 — col-xs-12,

sm: col-sm-1 — col-sm-12,

md: col-md-1 — col-md-12,

LG: col-lg-1 — col-lg-12.

No matter what, bootstrap will automatically allocate column width according to the width of the browser to match the rendering mode we want.

5.4.3 Language switching

Make a session store a language value, judge the language from the value of the session, then put the value of the relevant language in the template, use a router to change the language, write variables in the front end, do not judge, and use variables in the text display. The button is directed to the routing address changelang written in the back end, which is used to determine the Lang value in the session. En becomes zh and then redirects to the home page. Because the value of the home page render is determined by the Lang value of the session, the values of all languages can be put into a language variable, and the dictionary includes two values en and zh. Through this, render can be written in both Chinese and English, and variables can be written in two separate python files locales-zh.py and locales-cn.py.

5.5 Non-functional implementations

5.5.1 Security:

The passwords for each account are converted using a hash algorithm, which converts them to an irreversible hash string (or message digest) of the same length to protect the user's password.First, we read the password entered by the user in the front end, pass it to the back end through the form, and then save it in the database after the conversion through the hash algorithm. The implementation code is as follows:

```
        return redirect(url_for('customer_register'))
    passw_hash = generate_password_hash(form.password.data)
    user = Customer(username=form.username.data, email=form.email.data, password_hash=passw_hash)
    db.session.add(user)
    db.session.commit()
```

Figure 17: Hash algorithm

In order to ensure that the user's data is not tampered with by others, we use the session object. A session object is basically an object containing key-value pairs, that the server uses to store (and retrieve) information about some user. In this way, we can determine the current user through session, bind the database associated with the user through the user's information, and ensure that each user can only change his own data. First, we create a session. The code is as follows:

```
if (check_password_hash(employee_in_db.password_hash, form.password.data)):
    # flash('Login success!')
    session["USERNAME"] = employee_in_db.username
    return redirect(url_for('employee_base'))
```

Figure 18: Session

Then determine whether the session returns a value. If the session object returns a value, you know you have started a session. Otherwise, there is no session. The code is as follows:

```
if not session.get("USERNAME") is None:
```

Figure 19: Session

And then finally pop the session out in logout otherwise there is no session.

5.5.2 Stability:

The stability of the data is critical to the system. So we used the stability of high ali cloud server, ali a time server stability of ECS from infrastructure first, their availability of the data center is a requirement of paranoia, both power supply and supply network, the best way to obtain high stability is redundant, several routes at the same time supply electricity and network, and all of the switches is redundant architecture. In addition to infrastructure, the flying data operating system is also aliyun's "unique skills." In this operating system, we build a closed loop through big data machine learning and the ability to automate operations and maintenance. We can truly detect and actively prevent failures through scheduling and migration, pattern recognition, and provide ultra-high stability for our system. Aliyun service provider through enabling customers to the infrastructure topology transparent to customers, so that customers can effectively build their own disaster architecture. Compared with the traditional server in terms of stability, ECS has 99.9 percent service availability and 99.99 percent data reliability. It supports downtime migration, data snapshot backup and rollback, and system performance alarm.

At the same time, high stability of the database is also crucial. SQLite was actively developed by some really great software engineers. It takes the form of constraints to increase the stability of the database. Constraints are rules that are enforced on the data columns of a table to limit the types of data that can be inserted into the table, thereby ensuring the accuracy and reliability of the data in the database. High-quality new features are being added at an amazing rate. Just recently, SQLite added support for JSON data with the json1 extension. SQLite has also released an improved version of the full-text search extension, which includes results ranking using the BM25 algorithm. In addition to adding new features, SQLite developers are also working to make the library more performance. In version 3.8.11, the release notes include this small introduction: SQLite is now running twice as fast as version 3.8.0 and three times as fast as version 3.3.9. Despite all these changes and improvements, SQLite rarely introduces bugs. And SQLite's test suite is widely considered to be one of the best in the industry.

6 Conclusion

In order to complete the project and meet the needs of customers, This requires the full cooperation of the team, especially in the period affected by the epidemic. We have adopted a more scientific, efficient way of remote collaboration to make up for the weakness of no face-to-face communication. In this process, our group followed the learning strategy of PBL (Problem based learning), combined with the knowledge learned in the university curriculum to build the web page. In order to meet the needs of customers, we have self-taught such as using Ajax to achieve the asynchronous update of information without refreshing; using nginx and gunicorn to complete the deployment of applications in the cloud server; using JavaScript to realize the self-adaptive of web pages on different devices, etc.This has increased our knowledge of Technology.Our current project still has many functions that have not been implemented (such as online chat and switching between Chinese and English), which requires us to invest more energy. And we haven't done a stress test yet to see the highest concurrent processing power of the server. These are all problems to be solved in the next few weeks. After that, we will conduct a series of stress tests to ensure the stability and reliability of the program. If there is still plenty of time, we will optimize our JavaScript and CSS to achieve a better user experience, and also optimize the back-end code to achieve faster information processing speed.

But what is more important is that we have enriched the experience of team cooperation development and learned the role of each team member in the whole process from the beginning to the realization of a project. We think that we can get benefit more in the future from experiencing the process of team development, learning a more efficient and scientific way of communication, and accumulating experience than the growth of pure technology. We think this is the biggest harvest after this period of study.

...