



UNSW
A U S T R A L I A

COMP 9900 Information Technology project 2023 T2

Eatery Management System

Final Report

Group: 9900F15APT5D

Yue Niu (Scrum Master)

Front-end/Back-end	z5356976	z5356976@ad.unsw.edu.au
--------------------	----------	-------------------------

Haoxian Zhang

Front-end/Back-end	z5327309	z5327309@ad.unsw.edu.au
--------------------	----------	-------------------------

Lichen Zhang

Front-end	z5373704	z5373704@ad.unsw.edu.au
-----------	----------	-------------------------

Zhourui Shi

Front-end	z5348712	z5348712@ad.unsw.edu.au
-----------	----------	-------------------------

Yimin Liu

Back-end	z5228038	z5228038@ad.unsw.edu.au
----------	----------	-------------------------

4 August 2023

Contents

Overview	3
System Design	3
Functionalities	5
Function 1	5
Function 2	6
Function 3	7
Function 4	7
Function 5	8
Function 6	9
Function 7	11
Function 8	12
Brief descriptions of all third-party functionalities.....	14
UI Design	14
Figma and JS Design (http://js.design)	14
Front-end	14
React	14
Zustand	15
Tailwind CSS	15
Material-UI	15
Back-end	15
Python 3.8.....	15
Flask	15
Database	16
MySQL.....	16
MySQL Workbench 8.0	16
Jira	16
GitHub Desktop.....	16
Visual Studio Code	16
Implementation challenges	17
Implementing database	17
Architecting the Front-End by using react framework.....	17
Developing Back-end server and implementing API functions	17
Bridging Front-end functions and Back-end codes	18
Integrating Google Maps into our web page	18
User manual	19

Installation Guide	19
Creating Docker Images	19
Starting Docker Containers	19
Accessing the Website	19
User Documentation.....	20
User Registration, Login, and Profile Updating.....	20
Eatery General Coupon Issuance Function	23
Automatic Periodic Voucher Issuance Function	25
Search Eatery by Filter and Customer Book Voucher	26
Voucher Redemption Function	27
Voucher History and Eatery Review Function.....	28
Google Map Interaction.....	30
Leaderboard	31
About our team.....	33

Overview

System Design

As people's living standards improve, the opportunities for dining out and pursuing gastronomic experiences have gradually increased. Moreover, individuals tend to enjoy bargaining while making purchases. In order to assist people in promptly availing themselves of discounts, encouraging them to revisit eateries, and establishing better connections between customers and eateries, our team has opted to develop a web-based Eatery Management System called "Eathub" to meet these demands. Due to the differentiation in user requirements, the entire framework is designed to accommodate two types of users: customers and eateries. The system adopts a hybrid architecture, combining both B/S (Browser/Server) and C/S (Client/Server) modes, to cater to the distinct needs of these user categories. The entire framework is designed as follows:

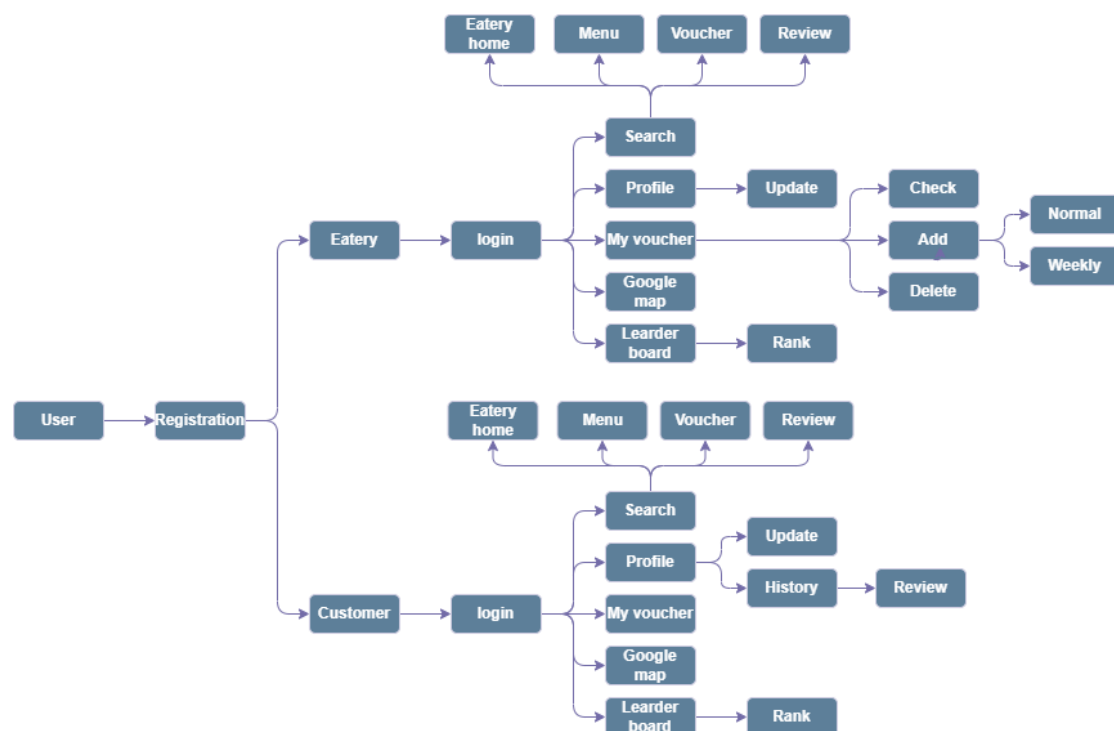


Figure 1 Framework of Eatery Management System

In order to achieve a well-structured system and enhance its maintainability, scalability, and security, Eathub can be designed using a three-tier architecture. The architecture consists of the presentation layer, application layer, and data layer, each responsible for distinct functionalities. Communication and interaction between these layers occur through interfaces, reducing the coupling between them. As a result, this approach facilitates better system design and development.

- **Presentation Layer:** The Presentation Layer is the user interface part of the system, also known as the front-end layer. It is responsible for displaying and handling the user interface, receiving user input, and passing it to the Application Layer for processing. In the Eatery Management System, this layer is responsible for displaying eateries,

voucher redemption, and showing voucher status, among other user interaction functionalities.

- **Application Layer:** The Application Layer is the core part of the system, also known as the middle layer. It contains the business logic and processing functionalities, responsible for receiving requests from the Presentation Layer and processing them, then passing the results to the Data Layer. In the Eatery Management System, this layer handles core business logic such as voucher processing, inventory management, and handling feedback.
- **Data Layer:** The Data Layer is the data storage and management part of the system, also known as the back-end layer. It is responsible for storing and retrieving data, including user information, voucher data, inventory information, etc. In the Eatery Management System, this layer can utilize a database management system to store and manage all relevant data.

Functionalities

Eathub is a restaurant management system designed with eight core features. In this section, we will dive into how to implement these features in code.

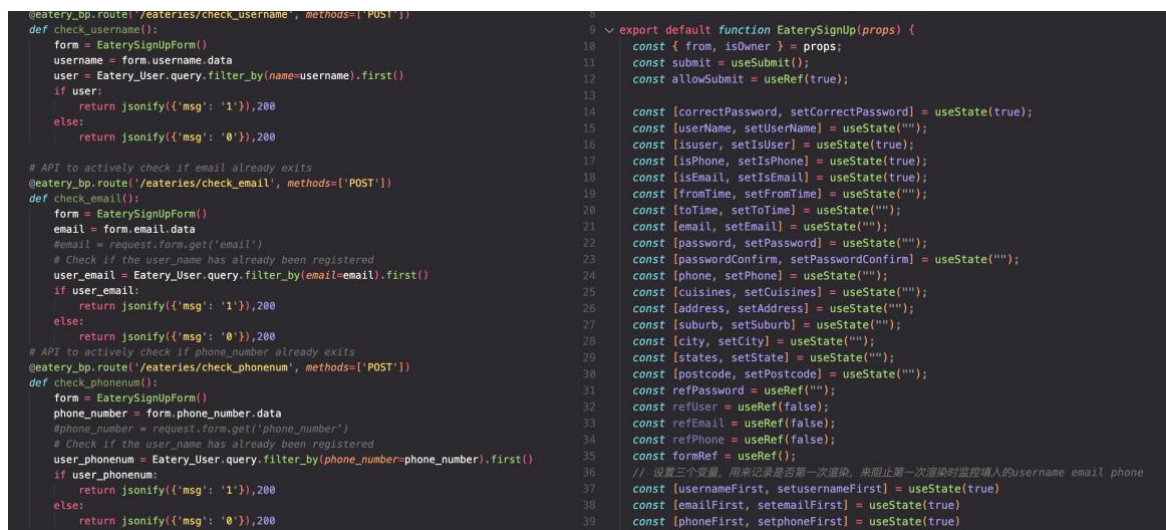
Function 1

Upon initially accessing the system, users are directed to a registration interface where they have the option to register either as a restaurant owner or as a customer. The system guides them to different pages based on this selection, each requiring specific information to be filled in.

For restaurant owners, the necessary details include essential information such as contact phone number, email address, physical address, cuisine type, operating hours, and more. On the other hand, customers need to provide basic details like phone number, email, date of birth, gender, etc.

Once registered, users can log in using their unique username created during the registration process.

The following is a detailed breakdown of the code implementation of the restaurant user registration functionality, including the frontend and backend components.



```

@eatery_bp.route('/eateries/check_username', methods=['POST'])
def check_username():
    form = EaterySignUpForm()
    username = form.username.data
    user = Eatery_User.query.filter_by(name=username).first()
    if user:
        return jsonify({'msg': '1'}), 200
    else:
        return jsonify({'msg': '0'}), 200

# API to actively check if email already exists
@eatery_bp.route('/eateries/check_email', methods=['POST'])
def check_email():
    form = EaterySignUpForm()
    email = form.email.data
    #email = request.form.get('email')
    # Check if the user_name has already been registered
    user_email = Eatery_User.query.filter_by(email=email).first()
    if user_email:
        return jsonify({'msg': '1'}), 200
    else:
        return jsonify({'msg': '0'}), 200

# API to actively check if phone_number already exists
@eatery_bp.route('/eateries/check_phonenum', methods=['POST'])
def check_phonenum():
    form = EaterySignUpForm()
    phone_number = form.phone_number.data
    #phone_number = request.form.get('phone_number')
    # Check if the user_name has already been registered
    user_phonenum = Eatery_User.query.filter_by(phone_number=phone_number).first()
    if user_phonenum:
        return jsonify({'msg': '1'}), 200
    else:
        return jsonify({'msg': '0'}), 200
  
```

```

export default function EaterySignUp(props) {
  const { from, isOwner } = props;
  const submit = useSubmit();
  const allowSubmit = useRef(true);

  const [correctPassword, setCorrectPassword] = useState(true);
  const [userName, setUserName] = useState("");
  const [isuser, setIsUser] = useState(true);
  const [isPhone, setIsPhone] = useState(true);
  const [isEmail, setIsEmail] = useState(true);
  const [fromTime, setFromTime] = useState("");
  const [toTime, setToTime] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [passwordConfirm, setPasswordConfirm] = useState("");
  const [phone, setPhone] = useState("");
  const [cuisines, setCuisines] = useState("");
  const [address, setAddress] = useState("");
  const [suburb, setSuburb] = useState("");
  const [city, setCity] = useState("");
  const [states, setStates] = useState("");
  const [postcode, setPostcode] = useState("");
  const refPassword = useRef("");
  const refUser = useRef(false);
  const refEmail = useRef(false);
  const refPhone = useRef(false);
  const formRef = useRef();

  // 设置三个变量，用来记录是否第一次渲染，来阻止第一次渲染时监控填入的username email phone
  const [usernameFirst, setUsernameFirst] = useState(true);
  const [emailFirst, setEmailFirst] = useState(true);
  const [phoneFirst, setphoneFirst] = useState(true);
  
```

Figure 2

After successful registration, users can also modify their personal information and edit their avatars under the login status at profile page, and merchants can also upload menu images to the eatery main page, the following is the code implementation of the back-end of the restaurant's user profile editing function.

```

@eatery_bp.route('/eateries/sendAvatar', methods=['GET'])
@login_required
def send_file():
    filepath = request.form.get('avatar')
    directory = filepath.split('/')[0]
    filename = filepath.split('/')[1]
    return send_from_directory(directory, filename)

@eatery_bp.route('/eateries/update', methods=['GET', 'PUT'])
@login_required
def update():
    #if int(current_user.get_id()) == id:
    id = int(current_user.get_id())
    if request.method == 'PUT':
        user_to_update = Eatery_User.query.get_or_404(id)
        #username
        user_to_update.name = request.form.get('username')
        #password
        if not request.form.get('password'):
            return jsonify({'code': 401, 'msg': 'Password can not be empty!'})
        user_to_update.password = request.form.get('password')
        #business hours start
        bussiness_hours_start = request.form.get('bussiness_hours_start')
        if not check_hours(bussiness_hours_start):
            response = {
                'msg': 'Invalid bussiness hours format. Please use H:M.'
            }
            return jsonify(response), 400
        user_to_update.bussiness_hours_start = bussiness_hours_start
        #business hours end
        bussiness_hours_end = request.form.get('bussiness_hours_end')
        if not check_hours(bussiness_hours_end):
            response = {
                'msg': 'Invalid bussiness hours format. Please use H:M.'
            }
            return jsonify(response), 400

```

Figure 3

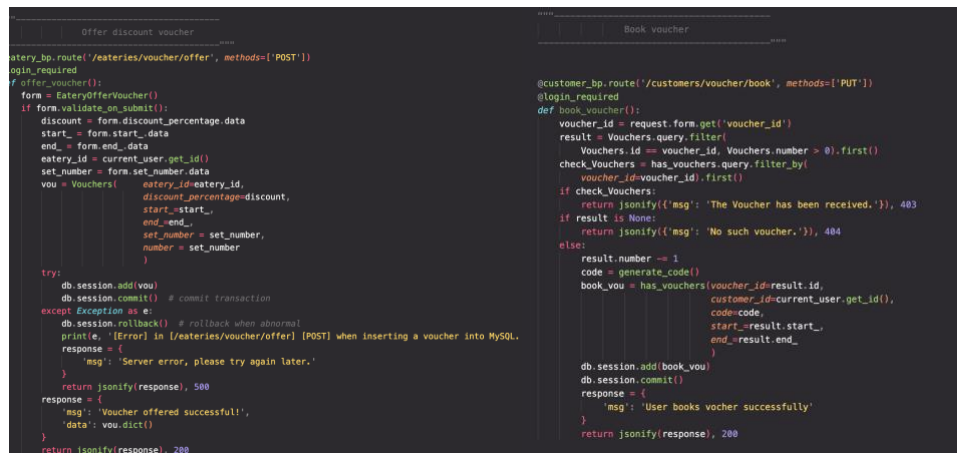
Function 2

The second function of the Eathub website allows restaurant owners to create customized discount coupons that can be distributed at any specified discount rate at any selected time. In this way, customers will be able to take advantage of the corresponding discounts at the time of checkout.

Restaurant owners can create these discount coupons in a completely customized manner. They can select the start and end time of the discount coupon on the calendar, set the discount percentage and determine the number of discount coupons to be distributed.

Customers, on the other hand, can choose to book these discount coupons on the restaurant's homepage. When the booking is successful, the system returns a message, "Coupon received." If the reservation fails, the system returns a message stating "No such coupon available". After a successful booking, the system automatically generates a unique coupon code that the restaurant can use for verification. This feature increases flexibility and interaction between the restaurant and the customer, enhancing the overall user experience.

The following is a detailed breakdown of the code implementation of the restaurant user and customer user vouchers functionality.



```

@eatery_bp.route('/eateries/voucher/offer', methods=['POST'])
@login_required
def offer_voucher():
    form = EateryOfferVoucher()
    if form.validate_on_submit():
        discount = form.discount_percentage.data
        start_ = form.start_data
        end_ = form.end_data
        eatery_id = current_user.get_id()
        set_number = form.set_number.data
        vou = Vouchers(
            eatery_id=eatery_id,
            discount_percentage=discount,
            start=start_,
            end=end_,
            set_number=set_number,
            number=set_number
        )
        try:
            db.session.add(vou)
            db.session.commit() # commit transaction
        except Exception as e:
            db.session.rollback() # rollback when abnormal
            print(e, '[Error] in [/eateries/voucher/offer] [POST] when inserting a voucher into MySQL.')
            response = {
                'msg': 'Server error, please try again later.'
            }
            return jsonify(response), 500
        response = {
            'msg': 'Voucher offered successfully',
            'data': vou.dict()
        }
        return jsonify(response), 200

@customer_bp.route('/customers/voucher/book', methods=['PUT'])
@login_required
def book_voucher():
    voucher_id = request.form.get('voucher_id')
    result = Vouchers.query.filter(
        Vouchers.id == voucher_id, Vouchers.number > 0).first()
    check_Vouchers = has_vouchers.query.filter_by(
        voucher_id=voucher_id).first()
    if check_Vouchers:
        return jsonify({'msg': 'The Voucher has been received.'}), 403
    if result is None:
        return jsonify({'msg': 'No such voucher.'}), 404
    else:
        result.number -= 1
        code = generate_code()
        book_vou = has_vouchers(voucher_id=result.id,
                                customer_id=current_user.get_id(),
                                code=code,
                                start=result.start_,
                                end=result.end_)
        db.session.add(book_vou)
        db.session.commit()
        response = {
            'msg': 'User books voucher successfully'
        }
        return jsonify(response), 200

```

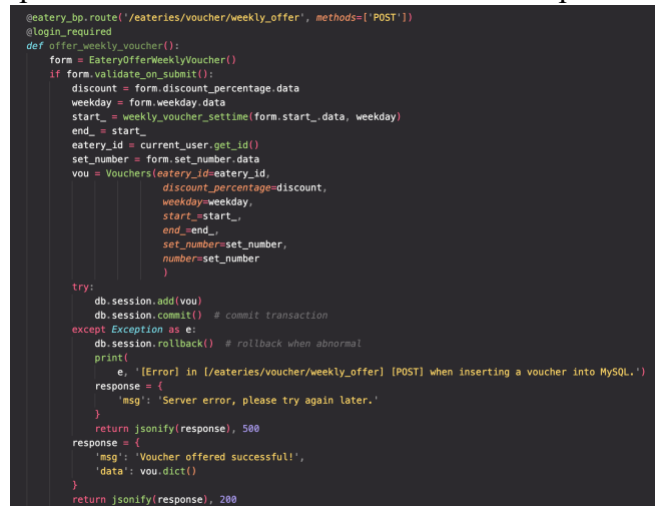
Figure 4

Function 3

The third function allows the restaurant owner to set a specific number of discount coupons to be distributed on a set day and time each week. Once on the coupon creation screen, the owner can choose between two types of coupons. The first type allows for customized times, while the second type allows the owner to select a day of the week on the calendar to distribute coupons on a regular basis.

This regular distribution of vouchers can be a powerful tool for restaurants to attract repeat customers or boost sales on typically slower business days. It's both flexible and automated, giving store owners the ability to strategically use discounts to boost business.

Below is the implementation of the backend code for this particular functionality.



```

@eatery_bp.route('/eateries/voucher/weekly_offer', methods=['POST'])
@login_required
def offer_weekly_voucher():
    form = EateryOfferWeeklyVoucher()
    if form.validate_on_submit():
        discount = form.discount_percentage.data
        weekday = form.weekday.data
        start_ = weekly_voucher_setTime(form.start_data, weekday)
        end_ = start_
        eatery_id = current_user.get_id()
        set_number = form.set_number.data
        vou = Vouchers(eatery_id=eatery_id,
                        discount_percentage=discount,
                        weekday=weekday,
                        start=start_,
                        end=end_,
                        set_number=set_number,
                        number=set_number)
        try:
            db.session.add(vou)
            db.session.commit() # commit transaction
        except Exception as e:
            db.session.rollback() # rollback when abnormal
            print(
                e, '[Error] in [/eateries/voucher/weekly_offer] [POST] when inserting a voucher into MySQL.')
            response = {
                'msg': 'Server error, please try again later.'
            }
            return jsonify(response), 500
        response = {
            'msg': 'Voucher offered successfully',
            'data': vou.dict()
        }
        return jsonify(response), 200

```

Figure 5

Function 4

The fourth function enables customers to search for discount coupons for different restaurants using keywords such as address and type of cuisine. In addition, customers can go directly to the restaurant page from the search results.

After selecting a discount coupon, the customer can clearly see all the details, including how long the coupon is valid for. If a particular coupon is claimed, the total number of available

coupons is decremented by one. It is worth noting that each customer can only claim one voucher in the same time, and if there are no remaining vouchers, they cannot be claimed.

The logic of this function consists of retrieving relevant fields from the front-end, such as discount, cuisine, restaurant name, location, and code. Depending on whether different fields have default values, the system will search and match the restaurant data corresponding to the input conditions.

This function not only facilitates users to find the ideal discount, but also improves the management efficiency of the whole discount allocation process and balances the needs of restaurant owners and customers.

The following is the code implementation of this function.

```
query = Eatery_User.query
if discount_start != 0:
    query = query.join(Vouchers, Eatery_User.id == Vouchers.eatery_id) \
        .filter(Vouchers.discount_percentage >= discount_start, \
            Vouchers.discount_percentage <= discount_end)
if cuisine and cuisine.lower() != 'all':
    query = query.filter(func.lower(Eatery_User.cuisine) == func.lower(cuisine))

if city and city.lower() != 'all':
    query = query.filter(func.lower(Eatery_User.city) == func.lower(city))

if state and state.lower() != 'all':
    query = query.filter(func.lower(Eatery_User.state) == func.lower(state))

if suburb and suburb.lower() != 'all':
    query = query.filter(func.lower(Eatery_User.suburb) == func.lower(suburb))

if postcode and postcode.lower() != 'all':
    query = query.filter(Eatery_User.postcode.ilike(f'%{postcode}%'))

if eatery_name and eatery_name.lower() != 'all':
    # regex_pattern = f'.*{re.escape(eatery_name)}.*'
    # query = query.filter(Eatery_User.name.op('REGEXP')(regex_pattern))
    query = query.filter(Eatery_User.name.ilike(f'%{eatery_name}%'))

# Execute the query and fetch the results
search_result = query.all()
if not search_result:
    return jsonify({'msg': 'No eateries.'}), 404
else:
    response_users_info = {
        'msg': 'eateries successful found!',
        'data': model_list_to_list(search_result)
    }
    return jsonify(response_users_info), 200
```

Figure 6

Function 5

The fifth function focuses on the use of discount coupons by customers during the ordering process. Customers can only use the discount coupons within the specified validity period to ensure that the discounts are applied appropriately.

Each coupon in the system has a unique code. Customers must present this code at the restaurant so that staff can verify that the voucher is valid. When a customer books a voucher, the system automatically generates the code and returns it upon successful booking.

On the part of the restaurant owner, they can search for that code and send a request to check if the code exists in the system. If it is found, the coupon can be redeemed, indicating that the customer used this discount.

This feature strengthens the integrity of the discount system by providing a secure, verifiable method of applying discounts. It ensures that both the customer and the restaurant owner adhere to the established rules and improves the overall efficiency and credibility of discount management within the Eathub system.

Below is the code implementation of this function.

```

=====
check code by voucher_id
=====

@eatery_bp.route('/eateries/vouchers/verify', methods=['POST'])
@login_required
def customers_show_code():
    eatery_id=current_user.get_id()
    #data = request.get_json()
    #code=data.get('code')
    data = request.get_json()
    code = data['code']

    print(code)
    voucher_record= db.session.query(Vouchers, has_vouchers, Eatery_User)\
        .join(has_vouchers, has_vouchers.voucher_id==Vouchers.id)\
        .join(Eatery_User, Eatery_User.id==Vouchers.eatery_id)\
        .filter(has_vouchers.code ==code, Eatery_User.id == eatery_id).first()
    if not voucher_record:
        return jsonify({'msg': 'No such voucher or voucher has been used.'}), 404
    else:
        response_users_info = {
            'data': redeemList(voucher_record),
            'msg': 'Voucher Available'
        }
        return jsonify(response_users_info), 200

def customers_show_code():
    eatery_id=current_user.get_id()
    data = request.get_json()
    code = data['code']
    voucher_record= db.session.query(Vouchers, has_vouchers, Eatery_User)\
        .join(has_vouchers, has_vouchers.voucher_id==Vouchers.id)\
        .join(Eatery_User, Eatery_User.id==Vouchers.eatery_id)\
        .filter(has_vouchers.code ==code, Eatery_User.id == eatery_id).first()
    voucher_record=model_list_to_list(voucher_record)

    if not voucher_record:
        return jsonify({'msg': 'No such voucher or voucher has been used.'}), 404
    eatery_id_1 = voucher_record[2]['id']
    customer_id = voucher_record[1]['customer_id']
    discount = voucher_record[0]['discount_percentage']
    weekday = voucher_record[0]['weekday']
    start = voucher_record[0]['start']
    end = voucher_record[0]['end']
    code1 = voucher_record[1]['code']
    use_voucher_time=datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    his = History(customer_id=customer_id,
                  eatery_id=eatery_id_1,
                  discount_percentage=discount,
                  weekday=weekday,
                  start=start,
                  end=end,
                  code=code1,
                  use_voucher_time=use_voucher_time)
    db.session.add(his)
    db.session.commit()

    delete = has_vouchers.query.filter_by(code=code).first()
    if delete:
        db.session.delete(delete)
        db.session.commit()

```

Figure 7

Function 6

The sixth function is that after using a coupon, users can find their usage history in the "History" screen. Under each usage record there is a "Rating" button that allows users to give a star rating from 1 to 5 and add their own comments. These reviews and ratings are displayed on the restaurant's homepage for all users to view while browsing the Eathub website.

The main business logic of this feature is as follows:

- **Ratings and Reviews:** When clicked, the frontend returns the coupon code, restaurant ID, and the added reviews and ratings to the backend. The backend adds this data to the "History" table and updates the average rating of the restaurant in the "Restaurants" table. The page then automatically jumps back to the customer's history page.
- **Edit and Delete:** After adding a review, the "Add Review" button changes to "Edit Review and Rating". The customer can click to edit or delete the review and go to the restaurant's full review and rating page by clicking on the restaurant's name in their

history. To delete a review or rating, the user can select "None" from the rating drop-down menu and empty the review box before submitting.

- Handling of inaccessible restaurants: The system includes a check for deactivated restaurant accounts. If a user attempts to navigate to a restaurant that no longer exists via their history information, the system displays the message "Restaurant Does Not Exist" and the user is unable to add a review. A similar message is displayed if the user tries to access a review for a restaurant with a deactivated account.

This function fosters a sense of community and trust within the Eathub platform, allowing customers to share their experiences and make informed dining choices. It also allows restaurants to receive direct feedback from customers, which is valuable for continuous improvement. By effectively handling a variety of user actions and edge cases, this feature enhances the robustness and user-friendliness of the system.

Below is the code implementation of this function.

```
#
@customer_bp.route('/customers/history', methods=['GET'])
@login_required
def customers_history():
    user_id = int(current_user.get_id())
    # 按照voucher使用时间降序排列history
    all_history = History.query.filter_by(customer_id=user_id).order_by(desc(History.use_voucher_time)).all()
    if all_history:
        # 构建结果字典列表
        results = []
        for history in all_history:
            eatery = Eatery_User.query.get(history.eatery_id)
            # 获取每一条history的字典
            history_dict = history.dict()
            # 在每一条字典里添加eatery_name和eatery_avatar (要判断是否有avatar)
            if eatery:
                history_dict['eatery_name'] = eatery.name
                if eatery.avatar:
                    history_dict['eatery_avatar'] = eatery.avatar
            else:
                history_dict['eatery_name'] = "Unkonwn eatery"
            results.append(history_dict)

        response = {
            'msg': 'History found!',
            'data': model_list_to_dict(all_history)
        }
        return jsonify(results), 200
    else:
        return jsonify({'msg': 'No history!'}), 404
```

```

customer_id = current_user.get_id()
review = request.form.get('review')
rating = request.form.get('rating')

history_to_update = History.query.filter_by(id=historyid).first()

if history_to_update:

    eatery_id = history_to_update.eatery_id

    #Get current time
    current_review_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # 先判断该eatery是不是已经注销了账号
    #
    eatery_user = Eatery_User.query.filter_by(id=eatery_id).first()
    # 账号存在
    if eatery_user:

        #
        # 添加评论和打分以及时间到数据库
        #
        history_to_update.review = review
        history_to_update.rating = rating
        history_to_update.review_time = current_review_time
        try:
            db.session.commit() # commit transaction
        except Exception as e:
            db.session.rollback() # rollback when abnormal
            print(e, '[Error] in [/customers/register] [POST] when inserting a user into MySQL.')
            response = {
                'msg': 'Server error, please try again later.'
            }
            return jsonify(response), 500

    #
    # 重新计算该商家的平均分并且添加到eateries表中

```

Figure 8

Function 7

The seventh function introduces map interaction, which enhances the user experience by providing geographic context. Specifically, when a user clicks on a restaurant's location on the Eathub platform, they are automatically redirected to Google Maps, where the restaurant's location is prominently displayed.

The integration with Google Maps provides a seamless way for customers to visualize the actual location of the restaurant, which is especially useful for planning a visit or understanding the distance to different dining options.

By adding this map interaction, the Eathub system provides users with an intuitive and convenient tool to connect the digital dining experience with real-world logistics. This underscores the platform's commitment to delivering a holistic user experience that goes beyond the mere processing of reservations and discounts, integrating itself into the larger context of dining planning and exploration.

Below is a demonstration of the code that calls the Google Maps API.

```

    } from "@mui/material";
    import Slider from "@material-ui/core/Slider";
    import React, { useEffect, useState } from "react";
    import EatItem from "../Eat_Item";
    import SearchRoundedIcon from "@material-ui/icons/SearchRounded";
    import CardGiftcardIcon from "@mui/icons-material/CardGiftcard";
    import CheckCircleOutlineRoundedIcon from "@mui/icons-material/CheckCircleOutlineRounded";
    import CancelIcon from "@mui/icons-material/Cancel";
    import { categories, state, cities, suburbs } from "../../constant";
    import { Link, useNavigate } from "react-router-dom";
    import useUserStore from "../../store/User";

    export default function SearchPage() {
      const [restaurant, searchRestaurant] = useState("");
      const [isFilter, setIsFilter] = useState(true);
      const [discount, setDiscount] = useState([0, 20]);
      const [category, setCategory] = useState("all");
      const [postCode, setPostCode] = useState("");
      const [states, setState] = useState("all");
      const [city, setCity] = useState("all");
      const [suburb, setSuburb] = useState("all");
      const [allEateries, setAllEateries] = useUserStore((state) => [
        state.allEateries,
        state.setAllEateries,
      ]);

      const [googleAddress, setGoogleAddress] = useState("");
      const [googleCity, setGoogleCity] = useState("");
      const [googleLocation, setGoogleLocation] = useState("");
    }

```

Figure 9

Function 8

The eighth function is designed to foster healthy competition among restaurants within the Eathub platform. In addition to the user comment and rating functionality, this feature compiles a comprehensive rating for each restaurant. This rating is calculated based on both the customer count and individual customer ratings.

These comprehensive ratings are then showcased on a leaderboard within the platform, creating a transparent ranking system that motivates restaurants to improve their service and performance. This also provides customers with a quick and clear reference for popular and highly rated dining options within the community.

The process of aggregating and displaying these rankings emphasizes data-driven decision-making, turning subjective opinions into quantifiable metrics. By offering a clear and competitive landscape, the Eathub system encourages restaurants to strive for excellence, while also providing customers with valuable insights to guide their dining choices. This feature complements the existing functionalities by adding a layer of gamification and analytics, contributing to a more dynamic and engaging user experience.

Below is the code implementation of this function.

```

1  import React, { useEffect, useState } from "react";
2  import HOC from "../HOC";
3  import bg1 from "../../assets/bg1.PNG";
4  const Home = (props) => {
5    const { city, address } = props; //you can add suburb
6    const [googleLocation, setGoogleLocation] = useState("");
7    useEffect(() => {
8      const Address = address ? address.trim().split(" ").join("+") : "UNSW";
9      const City = city ? city.trim() : "Sydney";
10     const googleSearch = `${Address},${City}+Australia`;
11     setGoogleLocation(googleSearch);
12   }, [address, city]);
13   return (

```

```
3
4 export default function ShowLeaderboard () {
5
6   /*filter相关 */
7   const [category, setCategory] = useState("all");
8   const [states, setStates] = useState("all");
9   const [city, setCity] = useState("all");
10  const [suburb, setSuburb] = useState("all");
11  const [data, setData] = useState([]);
12  const [count, setcount] = useState(0)
13  useEffect(() => {
14    const sendDataToBackend = async () => {
15      const response = await fetch(
16        "http://localhost:8081/eateries/sorted",
17        {
18          method: "POST",
19          credentials: "include",
20          headers: {
21            "Content-Type": "application/json",
22          },
23          body: JSON.stringify({
24            cuisine: category,
25            state: states,
26            city: city,
27            suburb: suburb,
28          }),
29        }
30      );
31      const data = await response.json();
32      console.log(data.data.length);
33      setData(data.data);
34      setcount(data.data.length)
35    };
36    sendDataToBackend();
37  }, [category, states, city, suburb]);
38}
```

Figure 10

Brief descriptions of all third-party functionalities

UI Design

Figma and JS Design (<http://js.design>)

We chose Figma and JS Design as our tools to design our user interface of the website for the following reasons:

- **Collaboration:** Both tools support real-time collaboration. This feature enables team members to work on the same project simultaneously, no matter where they are.
- **User-Friendly:** They both have intuitive user interfaces and a wealth of functionality, making it easy for designers to create and edit complex user interface designs.
- **Cloud-Based:** As cloud-based tools, Figma and JS Design allow access to design files from any device with internet, which simplifies file backup and sharing.
- **Prototyping:** Both tools support prototyping, enabling designers to create interactive UI prototypes for better demonstration and testing of design functionality.

Front-end

Overall, we developed our front-end server on the React framework.

React

React is a modern front-end architecture which makes it easier to build and maintain novel UI. Specifically, here are some main reasons why we chose React.

- The UI elements can be divided into several independent and reusable components benefiting from its componentized development model. The “props” such as JavaScript functions and React elements can be directly seen on screen. This approach is a developer-friendly way to develop and can increase efficiency.
- One of the main features of the React.js framework is the technique of virtual DOM which improves the performance and speed of the web page. Among different web development frameworks, whenever a large amount of information requires to be processed, the entire DOM tree needs to be re-rendered and the performance of the web page deteriorates. However, with React.js, it is reflected in the DOM tree only when a node changes.
- The lifecycle concept of the React class component allows developers to control the behavior easier. Furthermore, developers can choose to use either class or function components.
- React can realize client-side rendering in milliseconds, which releases the burden on the server-side and improves user experience.
- React and REST API work well together to make front-end development simpler and more manageable. REST API is a user-friendly API based on the REST architecture that can be easily migrated between servers and seamlessly switched to various databases at any time.

Zustand

During our React development, we chose to use Zustand as a state management tool. Here are the reasons. Firstly, unlike Redux, its API is designed to be very simple and easy to understand, which allows developers to get started quickly. Secondly, Zustand doesn't force developers to organize their code according to a specific pattern or structure. Thirdly, Zustand excels in performance by rendering only when necessary, which avoids unnecessary rendering and improves the performance of the application. In addition, Zustand integrates seamlessly with other features and tools in React, including React Hooks. This compatibility enhances our experience using Zustand in the React ecosystem. Finally, the active Zustand developer community is a great asset. It has provided us with not only timely help and support, but also a wealth of learning resources.

Tailwind CSS

We have chosen Tailwind CSS to style our web pages because of its practicality and efficiency. As a utility-first CSS framework, Tailwind provides many predefined classes. As a result, developers can apply styles they want directly in HTML file without a separate CSS file. This approach not only speeds up development, but also makes the code cleaner and easier to maintain. In addition, the responsive design capabilities and configurability of Tailwind CSS make it ideal for creating modern web applications.

Material-UI

Material-UI is one of the more popular React UI frameworks among developers. What attracts us to Material-UI is its extensive library of pre-built components, such as page buttons, input boxes, avatars, forms, and so on, that we frequently use in our eatery management system pages. Such pre-built components can improve the efficiency of our front-end development and UI integrity.

Back-end

When it comes to our back-end implementation, we mainly lean on Python as our programming language. Alongside Python, we utilize a variety of database technologies to manage and store our data. Our primary focus is on developing the back-end logic, managing user requests, and ensuring smooth interaction with the database, among other responsibilities.

Python 3.8

We chose Python 3.8 as our back-end API development language because of its readability, simplicity, and the large number of libraries available to us. In addition, Python's syntax is simple and easy to understand, and it has rich libraries and frameworks, such as the Flask framework, which enables our team to quickly deploy web applications. Furthermore, Python 3.8 is one of the most stable versions of Python, and stability is an important consideration for back-end development. What's more, the cross-platform nature of Python ensures that Python applications can be easily deployed and migrated across a variety of operating systems. For example, our team uses Windows and MacOS. In addition to this, our project will eventually be deployed on Docker. Therefore, Python 3.8 was the ideal choice for the development.

Flask

We have chosen Flask as our backend API development framework. The API built by Flask has no hidden logic and hence is more intuitive for developers. As a micro-framework, Flask is highly flexible and allows developers to add functionality as per their requirements. What's

more, while Flask itself does not include security measures, there are several extensions that address security issues, such as Flask-WTF for CSRF protection. Also, the configuration management allows for different settings for different purposes, such as testing, development, and production. In addition, FLASK has many off-the-shelf packages that make it easy for us to develop, such as flask-login for managing user logins in our back-end application, flask-blueprint for differentiating the APIs for different kinds of users, etc. FLASK has a lot more to offer than that, so we're going to go with the FLASK to develop the back-end server.

Database

MySQL

For database management, we chose MySQL, one of the most popular database management systems, as our data storage and management system to ensure that we can quickly retrieve the required data from the database and process it accordingly when requested by users or systems.

MySQL Workbench 8.0

We utilize MySQL Workbench, the official GUI tool of MySQL, for visual database management. MySQL provides a relatively intuitive way to help us with database design, schema writing, and modification.

Jira

Our Jira is a management tool widely used for various project management and agile development, it has the advantages of high flexibility in workflow management, support for agile development, ultra-high integration capabilities, a very rich set of analytical tools, and a well-developed plug-in ecosystem. In our team's project development process, our entire project's eight features, as well as their user stories, and the entire project's three phases are clearly defined in the Jira platform, and each member can track and update the status at any time. Jira improves the efficiency of our team's development, and saves the cost of communication, so we chose to use Jira.

GitHub Desktop

GitHub desktop makes it easy to localize and manage Git repositories. With this application we can quickly commit as well as pull project code, making it easy to manage code and maintain our repository.

Visual Studio Code

Our code is mainly edited and tested on VS Code. VS Code provides a very stable and powerful code compilation environment, with a very large number of plugins available, and vscode itself is very lightweight and doesn't require too much configuration and loading. For our project development provides a lot of convenience.

Implementation challenges

Implementing database

The initial challenge we encountered was the construction of a database. Firstly, we had to make a crucial decision regarding the selection of the appropriate database system. After careful consideration, we opted for MySQL to serve as our database server. Subsequently, we created an Entity-Relationship (ER) diagram to delineate the proposed structure of our database, outlining the number of tables, their interrelationships, and the attributes of each table. Upon finalizing this design, we proceeded to construct the actual database service and establish a connection with Flask programming.

During the programming design process, and particularly following the completion of Sprint 1, we revised our database model to incorporate additional tables and attributes, reflecting new insights and ideas.

While implementing our website, it was imperative to ensure that commands originating from the front-end were executed accurately within the database.

Architecting the Front-End by using react framework

We selected React as our front-end framework for website design. For many members of our team, JavaScript represented a new programming paradigm, with features distinct from those they were accustomed to. Additionally, mastering HTML programming proved to be a formidable challenge we needed to overcome. We utilized the component functionality provided by React to render our webpages.

Furthermore, it was necessary to implement certain functions within the front-end to send requests to the back-end, either for information retrieval or for modifying database data.

During the design phase of our webpage, one persistent issue was related to rendering. Specific problems included instances where the text or image areas exceeded the confines of the webpage. This challenge required diligent attention to resolve.

Developing Back-end server and implementing API functions

In the initial phase of our project, we conducted a thorough analysis of the project file, identifying all eight functions we intended to implement. Subsequently, through the careful design of user stories, we were able to determine the number of API functions required to be developed within our back-end server.

We selected the Flask framework to implement these back-end functions. The first challenge we encountered was establishing a connection with the MySQL database server. To achieve this, we created a models file that maps Python objects to corresponding database tables. This allowed us to utilize ORM functions for direct and convenient manipulation of database data. Additionally, we created a utils file to encapsulate useful functions, and a forms file to standardize the data received from the front-end.

Two key files, named 'customers' and 'eateries,' were created to implement all the functions necessary to satisfy the project requirements. During this phase, the primary challenge was debugging, as minor errors could lead to numerous issues within Flask programming. It was also crucial to debug ORM functionalities to ensure that we could successfully send SQL commands and receive accurate results from the database.

Bridging Front-end functions and Back-end codes

One of the most significant challenges we encountered was establishing a connection between the front-end and back-end servers. We utilized fetch functions to send requests to the back end, including headers and JSON data with specific API URLs. Additionally, it was essential to recognize the type of data being sent from the front end, which was a different process compared to using Postman to transmit data to the back-end server.

As a result, some team members were required to become proficient in both front-end and back-end programming features.

On the front-end side, we had to meticulously code within the React framework to ensure that the server sent requests and data in the correct format, and subsequently assigned the data received from the back end to the corresponding variables.

On the back-end side, we wrote code using Flask to guarantee that at the beginning of every function, the back-end server properly received data from the front end. Additionally, it was necessary to ensure the accurate transmission of required information from the database back to the front end.

This careful coordination between front-end and back-end components was crucial to the success of our project, requiring both technical expertise and effective communication within the team.

Integrating Google Maps into our web page

Initially, we had to acquire and manage a Google Maps API key. This process required us to invest time in understanding Google's billing system, as any mishandling of the key could lead to unauthorized access.

Upon successfully obtaining the API key from Google Maps, we faced another challenge: integrating it seamlessly into our website.

We elected to embed a Google Map on our main search page, displaying a list of eateries that customers might find appealing. On this page, the React server would automatically send a request to the back end to retrieve information about all eateries. However, we were confronted with the decision of how to best represent multiple eateries on a single map within the search main page. Ultimately, we chose to only display the location of the top-ranked eatery on the list.

Furthermore, when a user clicked on an individual eatery, they were directed to a dedicated page for that eatery, where the Google Map was also embedded to display its specific location. Through this approach, we believe we leveraged the capabilities of Google Maps to enhance our website's functionality effectively and aesthetically.

This experience underscores the value of careful planning and a thoughtful design process in creating a cohesive and user-friendly interface.

User manual

Installation Guide

Our project (Eatery Management System) is deployed and run on the docker container.

To simplify the installation and deployment process, we have prepared Docker files for both Flask-App and React-App and included a docker-compose.yml file in the project folder. Here are the installation steps:

Creating Docker Images

- Ensure you have Docker installed and properly configured. If not, please follow the [installation guide](#) on the official Docker website.
- Open the terminal and navigate to the **root** directory of the project folder.
- In the terminal, enter and run the following command: “**docker-compose build**”. This command will create Docker images based on the docker-compose.yml file. This process might take approximately 5 to 6 minutes.

Starting Docker Containers

- After successfully creating the images, enter and run the following command in the terminal: “**docker-compose up -d**”. This will start the Docker containers that were just created.
- Please note that our backend server runs on the default localhost: 8081:8081 port, and the frontend server runs on the localhost:80:3000 port.

Accessing the Website

- After successfully running the Docker containers, open Google Chrome or any other browser you prefer, and enter **localhost:80** in the address bar to access the project website.

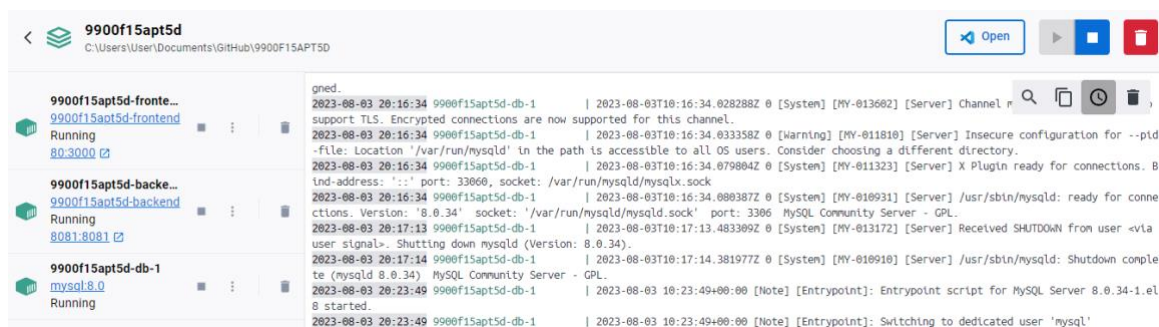


Figure 11 Project running on docker

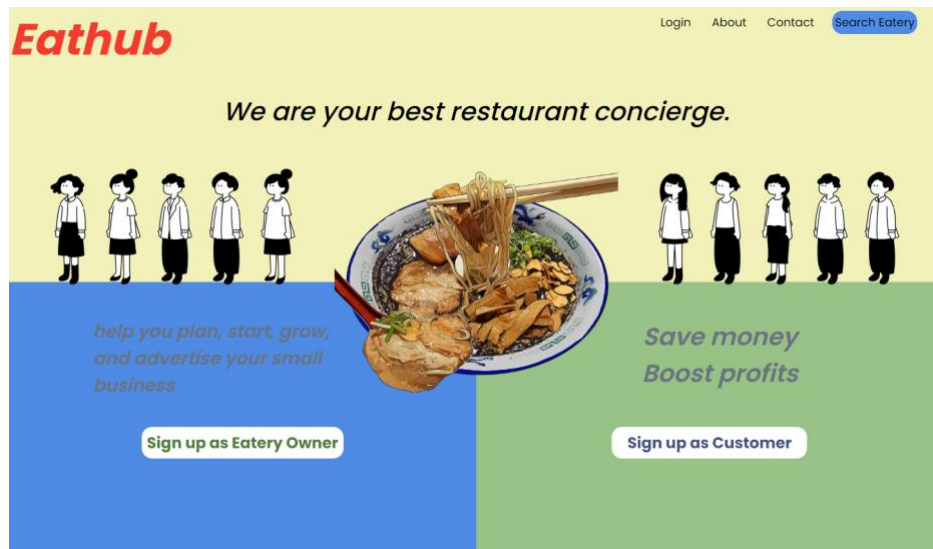


Figure 12 Eathub Home page

The project has now been successfully installed and is running!

User Documentation

User Registration, Login, and Profile Updating

The following steps guide you through the process of user registration, login, and profile updating.

Registration

- From the home page of the website, you can see two options: **"Sign up as Eatery Owner"** and **"Sign up as Customer"**. Choose the type of user you want to register as by clicking the respective button.

Figure 13 Eatery register

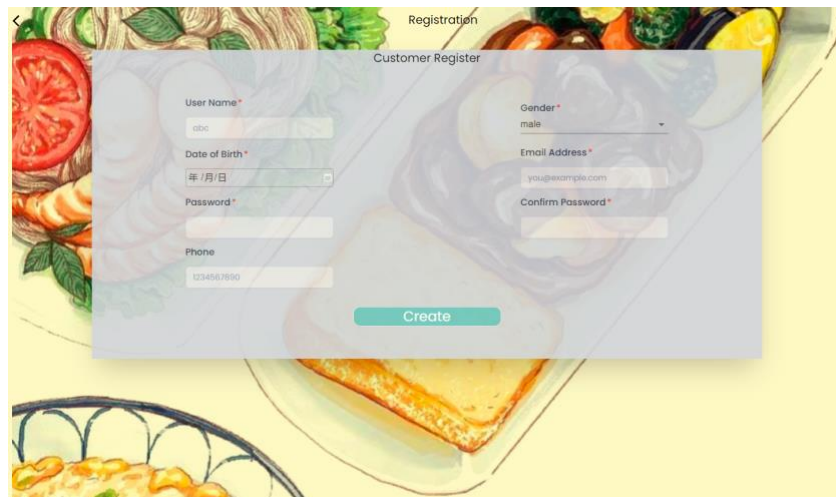


Figure 14 Customer register

- After successful registration, you will be automatically logged in and redirected to the search main page.

Login

- For already registered users, click on the **"Login"** button located at the top right corner of the **home page** to access your account.
- After successful login, click the **"Search Eatery"** button at the top right corner of the page to navigate to the search main page.

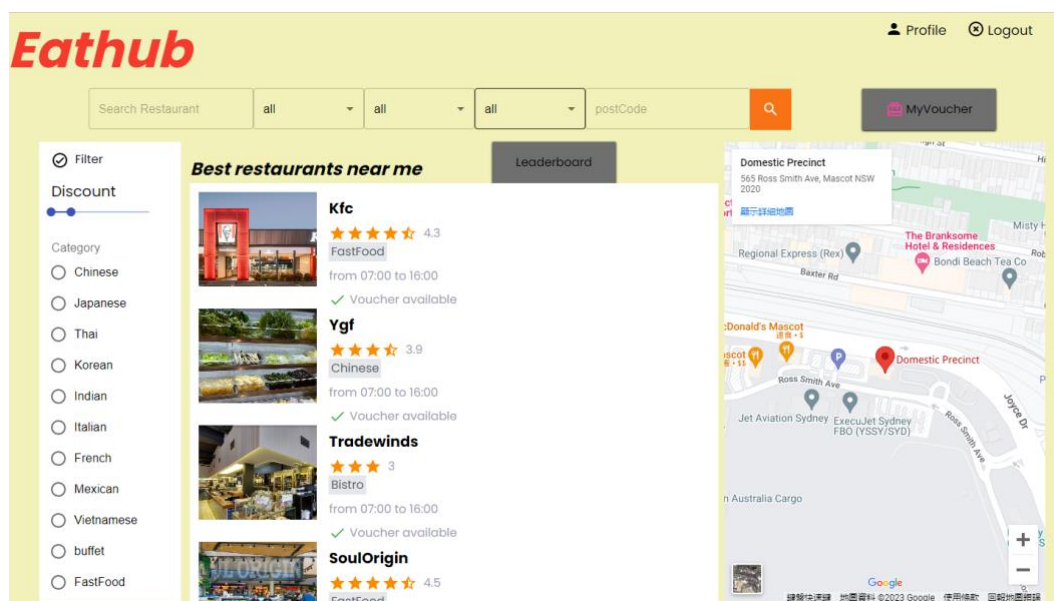


Figure 15 Eatery search main page

Profile Updating

- Once you are logged in and, on the **eatery search main page**, click on the **"Profile"** button at the top right corner of the page to access your profile.

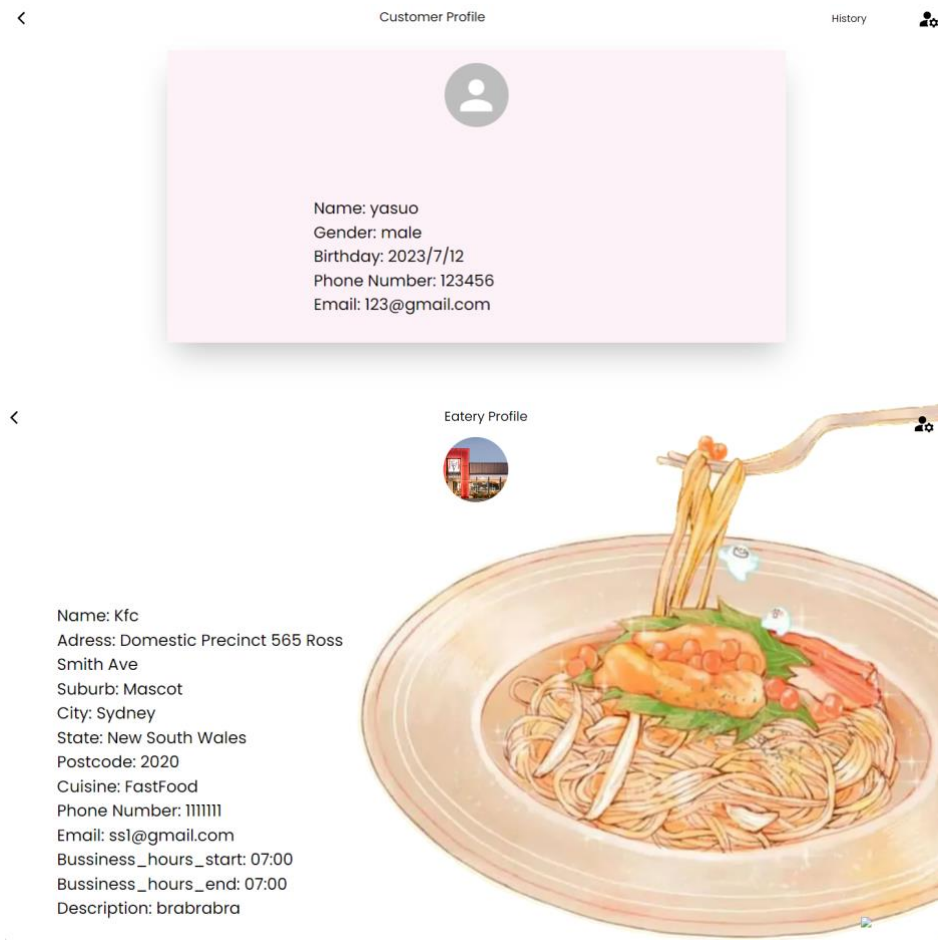


Figure 16 Profile page

- Within your profile page, click on the **edit button** located at the top right corner to enter the profile editing page.
- Here, you can update your information as needed, and even upload an avatar. Once you have finished making changes, click the "Submit" button to save your updates. You will be automatically redirected to the profile display page after your profile update is successfully submitted.

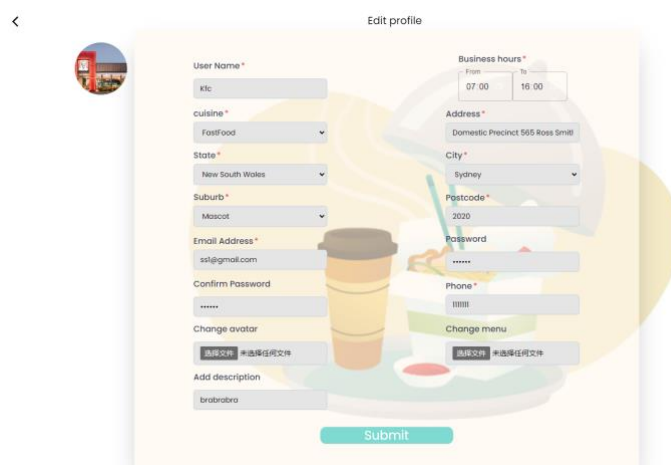


Figure 17 Eatery user profile updating page

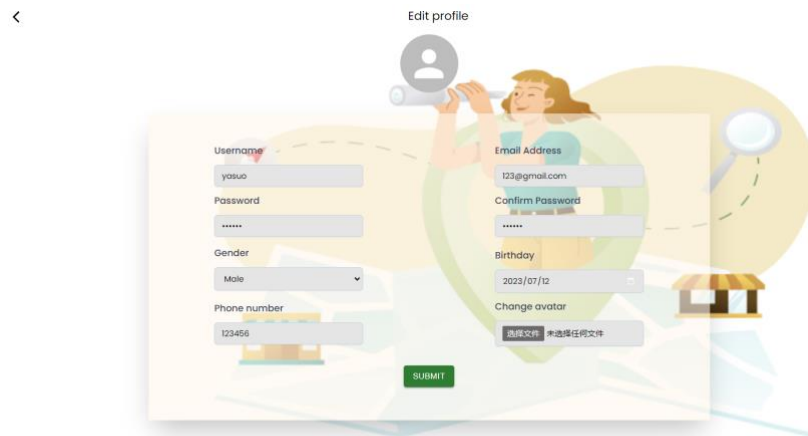


Figure 18 Customer user profile updating page

- To navigate back to the search main page, click on the **return arrow button** at the top left corner of the page, or use your browser's back button.

Eatery General Coupon Issuance Function

Our website offers a convenient method for restaurants to distribute customizable vouchers to customers at any time. The quantity, discount, and validity period of these vouchers can be freely set to meet the specific needs of each eatery. Here are the detailed steps:

Eatery creating general vouchers

- After logging in, eatery users can click on the **"My Voucher"** button in the upper right section of the search main page, which will redirect to the restaurant's own voucher page. This page displays all the vouchers that the restaurant has created.

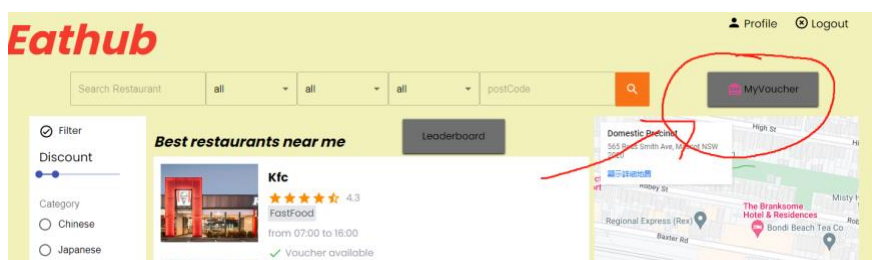


Figure 19

- To create a new voucher, users can click the **"Add a voucher"** button, which leads to the voucher creation page.

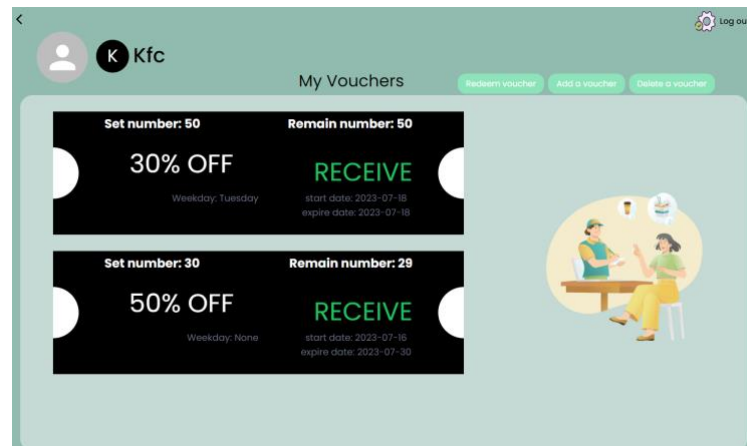


Figure 20 Eatery voucher page

- On the voucher creation page, eatery users can set the discount rate, quantity, and validity period of the voucher according to their needs. After completing the edits, click on the "save" button. The newly created voucher will then be displayed on the eatery's voucher page.

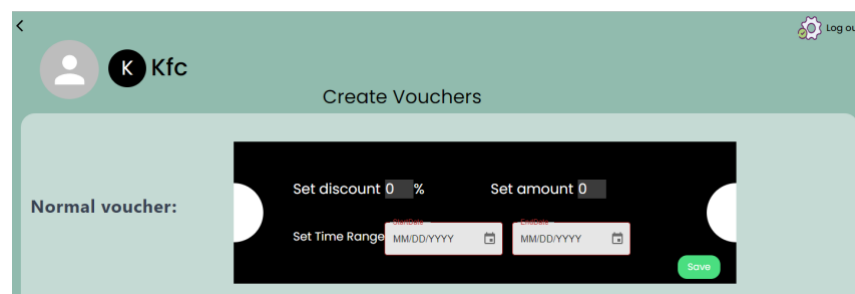


Figure 21 Eatery normal voucher creating page

For customer users, we also provide a straightforward voucher booking feature:

- Booking a voucher:** Customer users can click on an eatery they're interested in from the search main page, which will take them to that eatery's page. On this page, customer users can click on the "book a voucher" button at the top, which will scroll the page down to the restaurant's voucher area. Customers can then choose the voucher they want to reserve and click the "Book" button. If the booking is successful, the system will provide a success prompt.

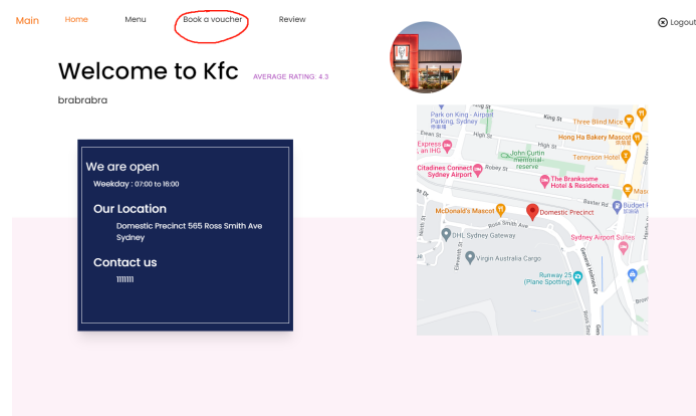


Figure 22 Eatery home page

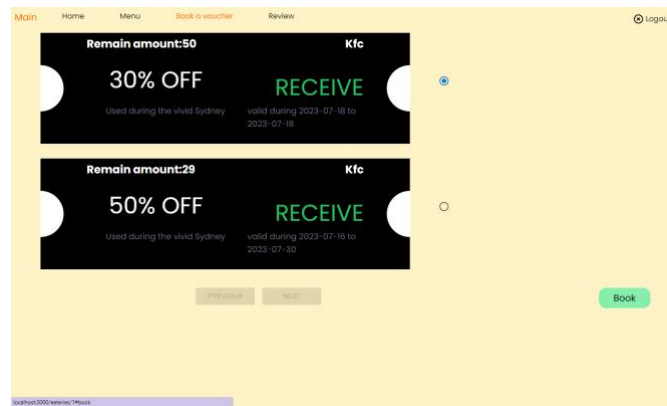


Figure 23 Book voucher page

- Please note that each customer user can only reserve one voucher from the same restaurant.
- **Viewing reserved vouchers:** After successfully booking a voucher, customer users can click on the "My Voucher" button from the search main page, which will redirect to their own voucher page. On this page, customer users can view all the vouchers they have reserved.

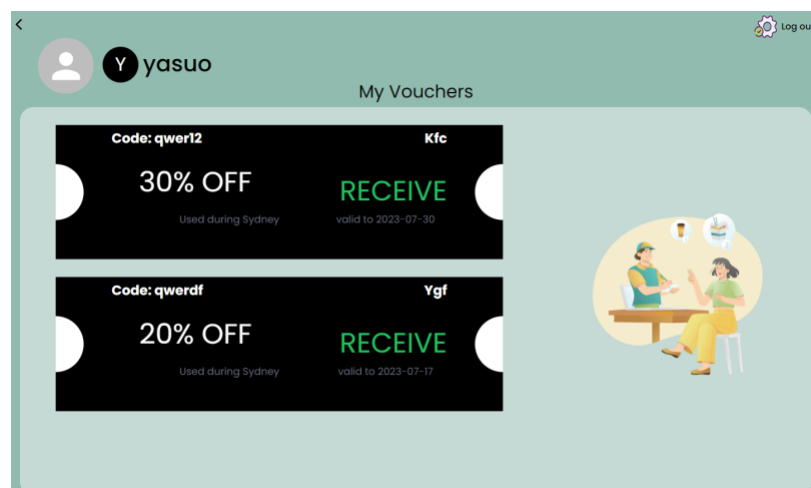


Figure 24 Customer self-voucher page

Automatic Periodic Voucher Issuance Function

Our platform also provides a unique feature that allows eatery users to set up vouchers that are automatically issued on a periodic basis. Here's how to use this feature:

- After logging in, **eatery users** can click on the **"My Voucher"** button to enter the eatery vouchers page. Click on the **"Add a voucher"** button to navigate to the page where new vouchers can be added.
- On the voucher addition page, the second voucher template is designed for setting up weekday vouchers. Eatery users can customize the discount, number of vouchers, the time to start the voucher issuance cycle, and the specific day(s) of the week to issue the voucher. After setting these details, click **"save"** to successfully add the voucher.

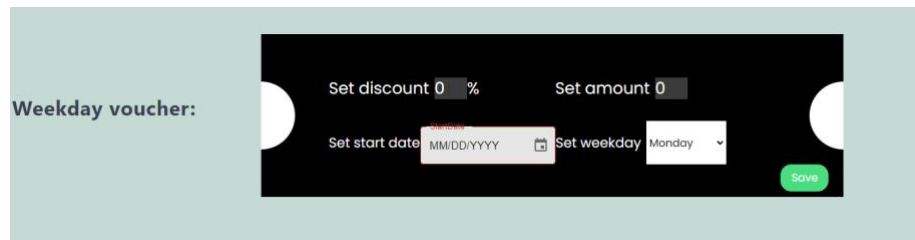


Figure 25 Weekday voucher

Search Eatery by Filter and Customer Book Voucher

One of the key features we offer is the ability for customer users to search for interested merchants using various filters, reserve vouchers from the merchant's page, with the stipulation that a customer can only reserve one voucher from the same merchant at the same time. Here are the steps to perform this operation:

Search Eatery by Filter

- After the customer user logs in and navigates to the search main page, they can **slide the discount range** on the left side of the page and **select cuisine** types to filter merchants.
- Additionally, the **search bar** at the top of the page can be used to further refine the search. In the first field of this search bar, users can **use keyword** search to find all merchants containing the keyword. The subsequent fields allow users to select state, city, suburb, or enter a postcode to filter merchants.

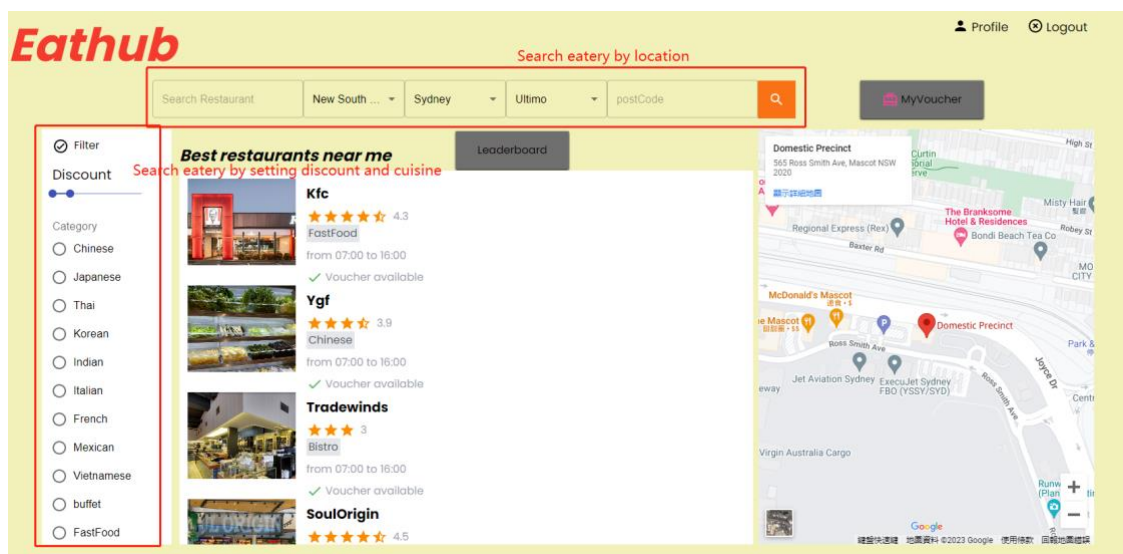


Figure 26 Search eatery

Book Voucher

After selecting and entering an eatery's page, users can click on the **"book a voucher"** button in the navigation bar at the top of the eatery homepage to scroll to the eatery's voucher page. Here, users can choose the voucher they want and click the **"Book"** button to reserve the voucher. See Figure 22 and Figure 23.

Voucher Redemption Function

This feature allows customers to redeem vouchers during their validity period by providing a unique code associated with the voucher. This ensures that only valid vouchers are used and aids in seamless voucher management. Here's how to use this feature:

Customer Show Voucher Code

As a customer user, after logging in, you can click on the **"My Voucher"** button from the search main page to enter the voucher page. Here, you can provide the unique code associated with the voucher you wish to redeem to the merchant. See Figure 24.

Eatery Redeem Voucher

As a restaurant user, after logging in, you can click on the **"My Voucher"** button from the search main page, then click on the **"Redeem voucher"** button. In the search box that appears, input the code provided by the customer. If the code is outside of its validity period or has already been used, there will be **no search results**. If the code is valid, you can click on the **"redeem it"** button next to the voucher, successfully redeeming it.

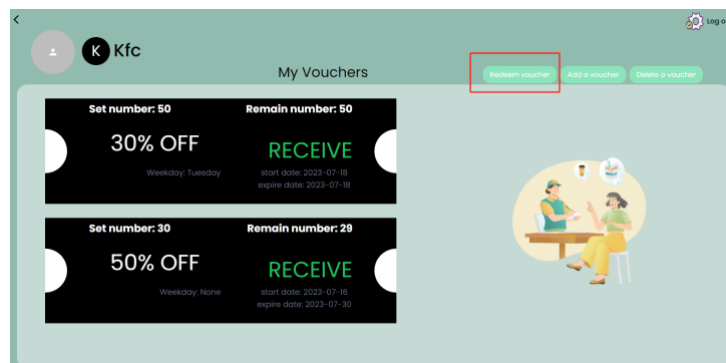


Figure 27 Eatery voucher page

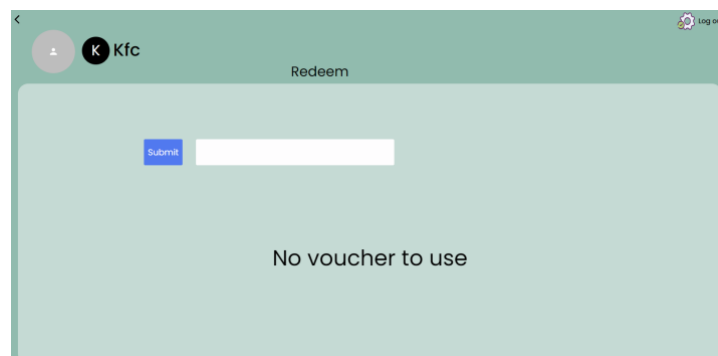


Figure 28 No result notation

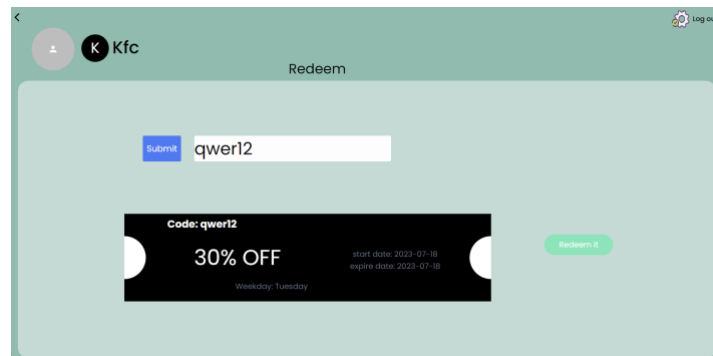


Figure 29 Redeem a voucher

Voucher History and Eatery Review Function

This feature enables customer users to track their voucher usage history, and for each record, they have the option to leave a review and rating for the merchant. All users can view all reviews and the average rating for each merchant. Here's how to use this feature:

Customers View Their Voucher Usage Histories

- As a customer user, after logging in and navigating to the search main page, click on the **"Profile"** button in the upper right corner.
- In the profile page, click on the **"History"** button, located also in the upper right corner. Your voucher usage history will be displayed on this page, and you can use the **"Next"** and **"Previous"** buttons to navigate between pages.

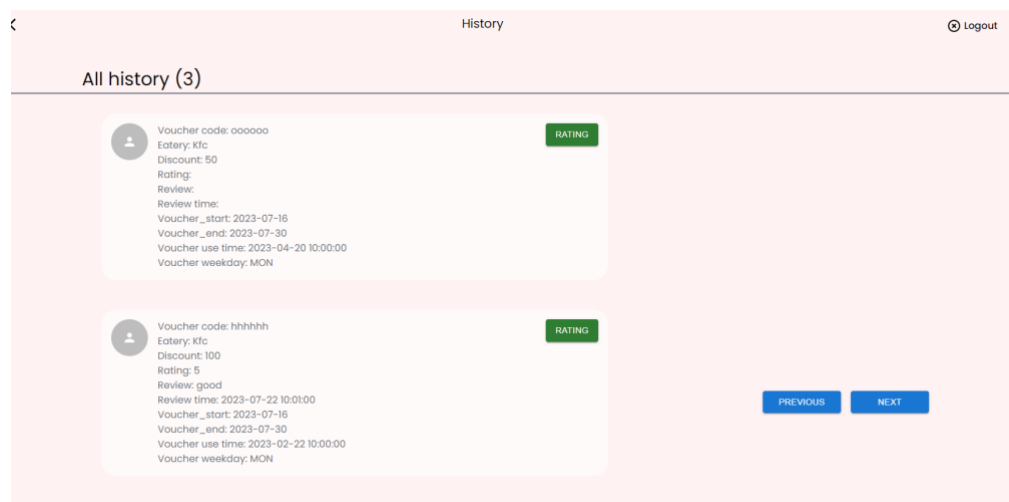


Figure 30 Customer history page

Customers Adding Review and Rating

- To leave a review, select the voucher usage record you wish to review and click the **"Rating"** button. You will be redirected to the review page.

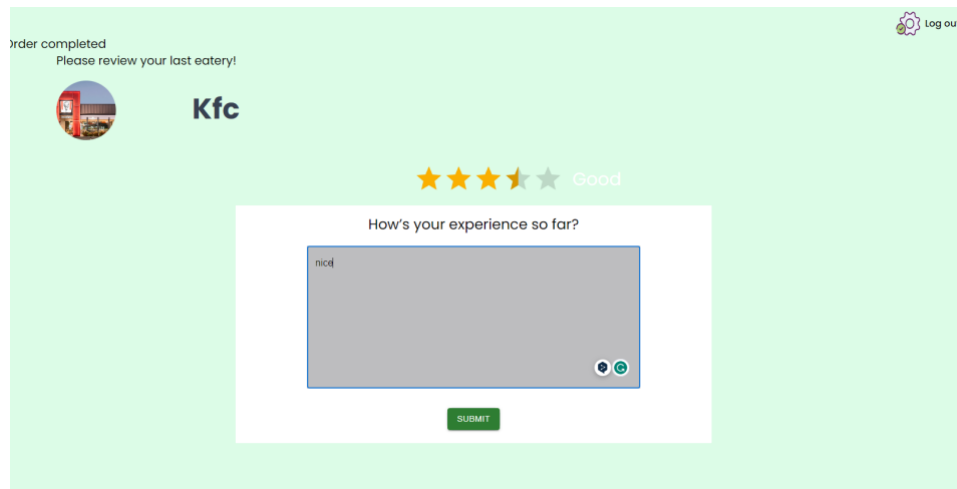


Figure 31 Adding review and rating page

- Input your rating (up to five stars) and write your review. After clicking the **"submit"** button, your review will be posted, and you will be automatically redirected to the eatery's review display page, where you can see all user reviews and ratings for that eatery.

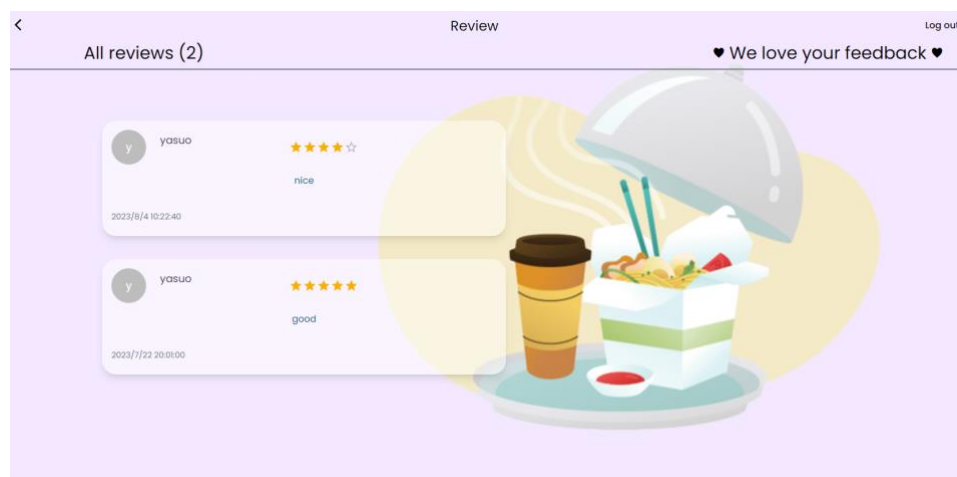


Figure 32 Eatery review page

For restaurant users and other customer users:

- After logging in and navigating to the search main page, click on an interested **eatery** to enter their **main page**. The **average rating** of the merchant will be displayed on the right side of their name.

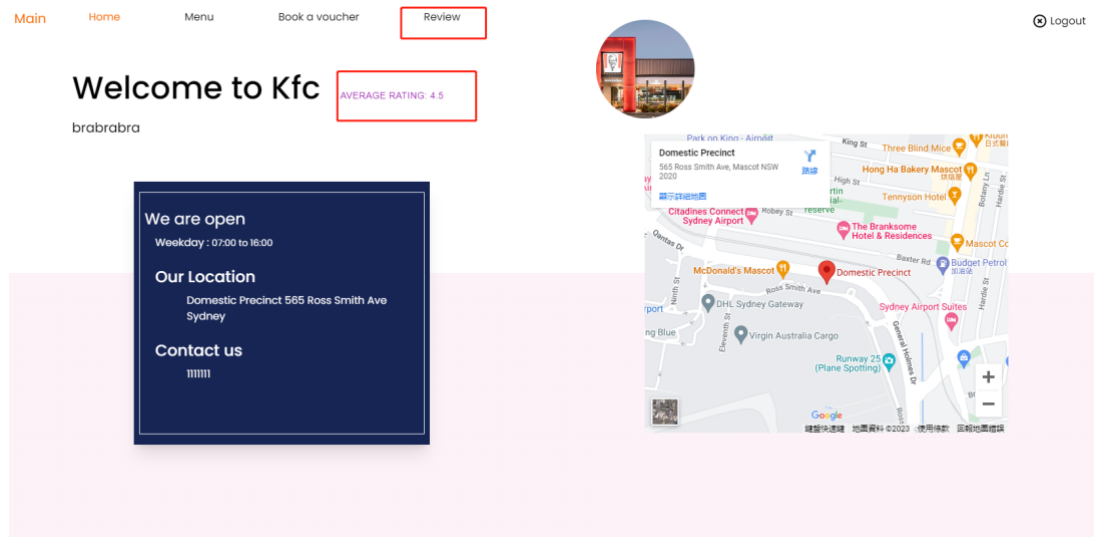


Figure 33 Eatery home page

- By clicking on the **"Review"** button in the top navigation bar, the page will scroll to the merchant's review section where all reviews are displayed.

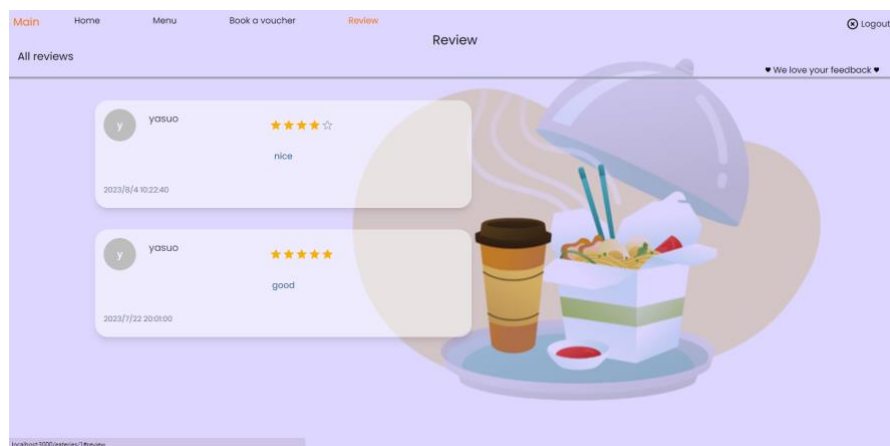


Figure 34 Eatery review page

Google Map Interaction

Our platform incorporates a feature that integrates with Google Maps to provide a clear geographic context for each restaurant. Here's how to use this feature:

- Default map view:** On the eatery search main page, the Google map displayed on the right side of the webpage defaults to show the geographical location of the first restaurant listed in the eatery's column.
- Map view for selected eatery:** When users click on a restaurant they're interested in, they are redirected to the restaurant's home page. The map on the right side of the page will then update to display the restaurant's location.

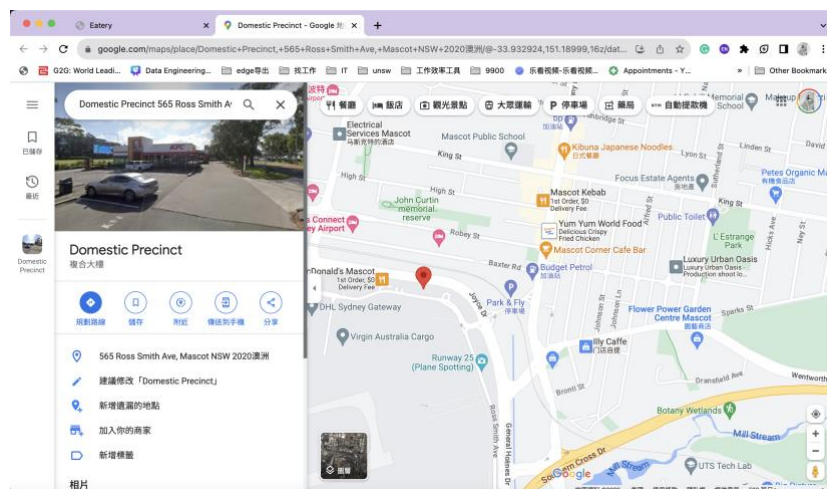
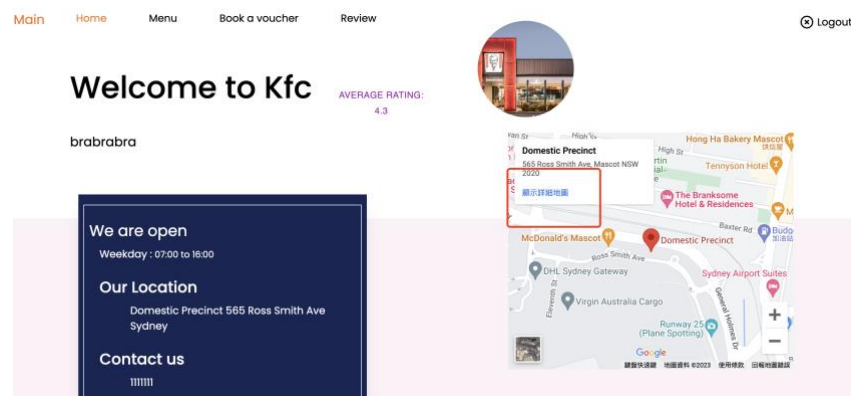
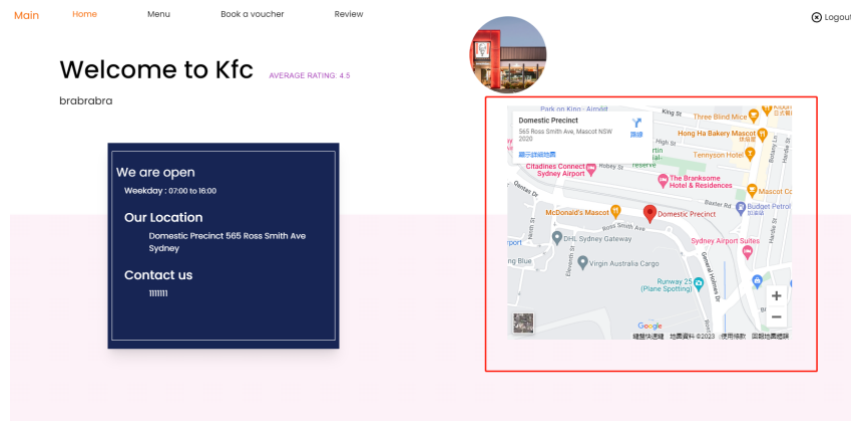


Figure 35 Google map interaction

Leaderboard

Our platform features a leaderboard that ranks all registered eateries based on various metrics, offering both logged in and non-logged in users an overview of the most popular or highly rated eateries. Here's how to use this feature:

- Whether you're logged in or not, upon entering the eatery search main page, you can click the **"Leaderboard"** button located above the eateries list to navigate to the leaderboard page.

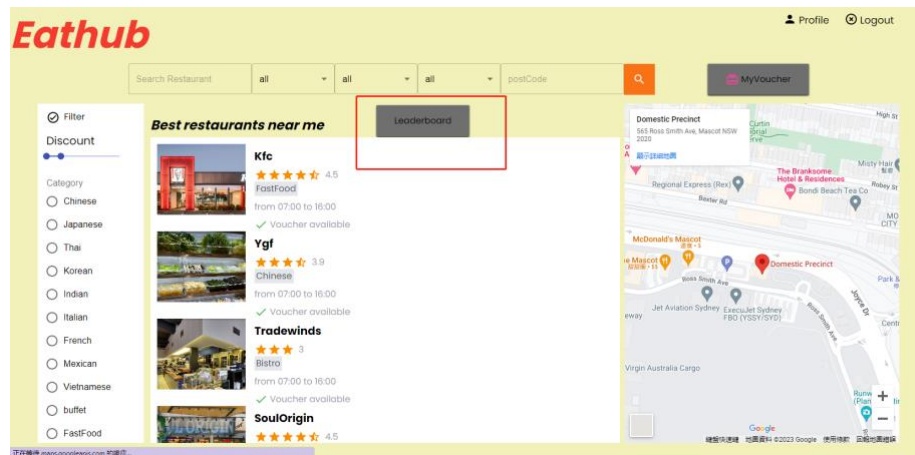


Figure 35 Leaderboard button

- The leaderboard displays a ranking of all the eateries registered on the website. Users can clearly see the eateries' rankings, names, profile pictures, and average scores.








1		Macca	5.00	★★★★★	FastFood
2		CornerstoneCafeUTS	4.90	★★★★★	Cafe
3		CurryCraze	4.60	★★★★☆	Indian
4		Kfc	4.50	★★★★☆	FastFood
5		SoulOrigin	4.50	★★★★☆	FastFood
6		Treetop	4.40	★★★★☆	Cafe
7		WallabiesThai	4.00	★★★★☆	Thai

Figure 36 Leaderboard items

- We've also implemented **four filters** at the top of the leaderboard - **cuisine, state, city, and suburb**. Users can set these filters to view the rankings of eateries within specific categories or geographical areas.

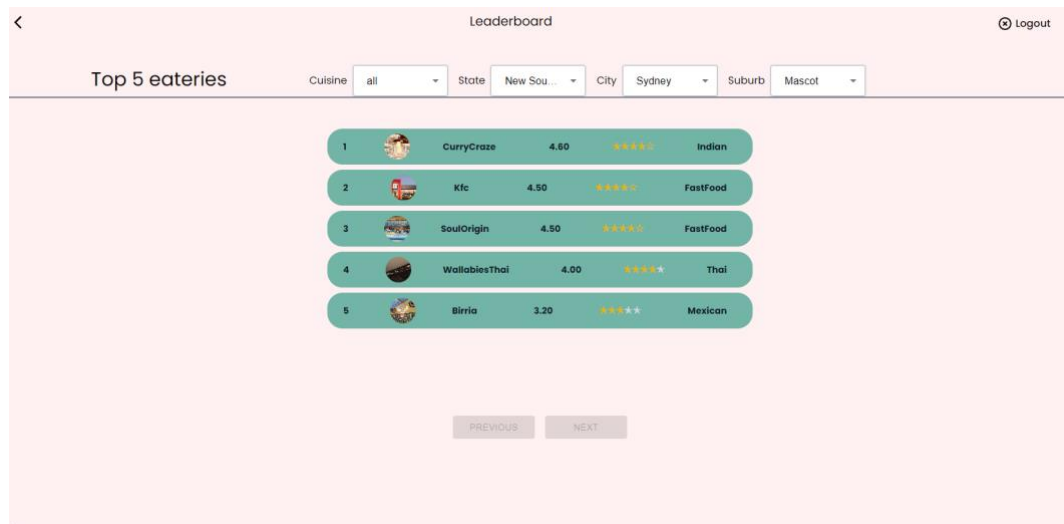


Figure 37 Ranking by setting filter

About our team

On the home page, there is an “About” button, press it and you can navigate to the About page.



Figure 38

On this page, you'll find a succinct introduction to our team, along with a detailed account of each member's contributions to this project. This has been a collaborative endeavor, with every team member working in unison to bring this capstone project to fruition.

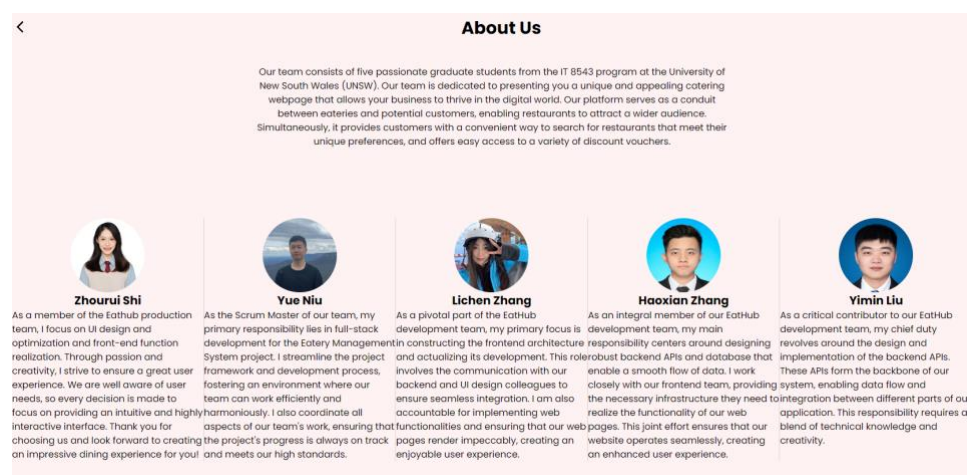


Figure 39 About page