University of Puerto Rico Mayagüez, Campus

Faculty of Engineering

Department of Computer Science and Engineering

# Team I: Find & Eat

Estefanía Torres Collado

Andrea C. Miranda Acevedo

David Carrión Beníquez

Gabriel R. Pantojas Burgos

Christopher Vegerano López

Everson Rodríguez Muñiz

## Informative Part

I. Name, Place, Date
   a. Find&Eat, Mayagüez PR, August 2020

II. Current Situation
   a. The current situation is that people are struggling to find an establishment that serves their craving that satisfies their needs based on taste, budget, availability, and location. This causes a delay on the time that a person takes to choose what to eat and where. Furthermore, a wrong decision can lead to unsatisfaction of that person and potentially deteriorate the image or reputation of the establishment.

III. Needs
   a. There is therefore a need to enhance discovering the right food establishments that are capable of increasing satisfaction of a customer and decrease the time it takes to reach a decision.

IV. Ideas
   a. The idea is to provide a platform/system where people can search and find the best rated dish of a food establishment that fulfills their craving. Dish ratings will be based by reviews and votes of numerous food critics and popularity/majority preference. From another perspective, a person could be at a restaurant but doesn't know what to eat, s/he can verify in our system the top or most preferred dishes in that establishment and can read professional reviews, if any, of each dish.

V. Scope, Span
   a. The scope is food directory. The span is the creation and management of a crowd-sourced and food critic reviewed food community to explore different food options.

VI. Synopsis
   *a.* The goal of this project is to develop a web app to tackle frequent problems when going out to eat to facilitate decision making speed and satisfaction. The best dishes from each establishment (according to users) are displayed to a user faced with indecision when presented with a myriad of choices. Food critics are also users within the user pools but with a greater weight in their opinions.

The design of the application includes an explore page, which displays the most popular dishes at the moment according to user ratings. It also includes a profile page for food establishments, in which administrators of accounts can customize their profile's information (location, name and business hours) and modify menus by creating, editing or deleting dishes (with information such as price, name, description, category and type).

The app will be implemented using React for frontend development, utilizing libraries such as Material UI for GUI and Firebase for storage of the files that the users upload. The Flask framework along with SQLAlchemy and PostgreSQL will be used for backend development. The app will be hosted either in Heroku or Firebase, the decision will be made on the final sprint of the project.

The team has been distributed in two groups, each designated to tackle frontend and backend, respectively. There are four members in the frontend team while two work in the backend. There are several aspects of agile development applied to the workflow in accordance such as: adaptive planning, evolutionary development, and flexible responses to change are encouraged.

## Descriptive Part

I. Descriptive Rough Domain Sketches

   a. A person wants to go out to eat a specific dish that the person is craving at a given moment, but that person wants to eat at the best possible location that serves the best dish that they are desiring. That person could be a tourist or someone that doesn't know the area all too well, thus reaching a problem that he or she doesn't have a good sense of direction of where to dine where they will satisfy their palate. The person proceeds to look for information on the internet or by asking friends or family members for suggestions of establishments near them that serve the dish they are craving. That way the person can gather different options to decide which is the best option. The best option will be determined by different criteria: the person's budget, distance/location, quantity, quality and/or ratings from source (friend or web). When an option is finalized, the person goes to the establishment, orders the food and proceeds to eat it.

II. Descriptive Domain Narrative

   a. In the domain of food establishments, the main notions that further describe and differentiate each from the other. These are: menu, location, price range, rating, and category. A menu is made up of one or more dishes and is created with the food establishment and can be edited. A location is the placement of the establishment in a place during a specific time. A price range is the average amount of money that is spent on one meal at the location. A rating is the overall score given by the clients. The category is the classification or grouping of the food establishment for it to be searchable. By a food establishment we understand that it is a location in which food and drink is sold to customers. A food establishment can be searched in terms of location, rating, or type. There are several operations which involve or result in food establishments: creating a new food establishment or editing a food establishment. Editing a food establishment can be triggered by a change in rating due to new reviews or a change in location or offering in the menu.

      Dishes are a serving available at an establishment that form its menu. They are also composed of other notions to further describe them. These are rating, category, and price range. A rating is the overall score given by the consumers, which is utilized for searching. The category is the classification or grouping of the dish for it to be searchable. The price range is the amount of money necessary to purchase the dish. There are several operations which involve or result in dishes. Dishes can be created, edited, or deleted. A dish's rating changes according to the overall score of the users or customers. Dishes can be deleted when an establishment no longer serves it in their menu.

III. Descriptive Domain Terminology

a. Craving –An intense desire for a particular food that the person wants to eat.
b. Rating – Score that determines the ranking of a dish based on different categories.
c. Source –Website or close friend/relative that gives you feedback of a dish.
d. Dish – Specific plate of food.
e. Establishment – Place where dishes are prepared and served for consumers that have paid.
f. Distance – Proximity between a person and an establishment.
g. Domain Entities
    i. Establishment: location where customers go and purchase food.
    ii. Person: user of the web browser and the one that goes to the establishment to satisfy their craving.
    iii. Dish: food that is served in an establishment. In the web application it is the main search engine.
        1. Price: amount of money that a person must pay for the dish in the establishment to consume it. In the web application it can be utilized as a search to show you the best dish that a person can purchase according to their budget.
        2. Rating: the voting system that is utilized in the web application where the dish can get upvoted by different user determining which establishment server the best dish that the user is searching for.
        3. Ingredients: ingredients are the components that make up the dish when they are combined.
        4. Category: is what the dish is categorized as (i.e. sandwich, burgers, legumes, etc.). In the user stories later on the document you can see all the categories listed there.
    iv. Smartphones/computer: Devices utilized to access the web application to search for the dish that they want to eat.
    v. Map: the map consists of the different location of the establishments that have been enlisted in the web application. It will also determine how long it will take you to arrive to the establishment from your current location.
    vi. Menu: List of different dishes that an establishment serve.
    vii. Drink: refreshment that a user drinks after they finish eating to soothe their thirst.
h. Domain Functions
    i. Rate_dish(dish, score): assigns/updates the rating score of the specified dish.

   ii. Search_item(): determine all the searchable items within the scope of the domain.
   iii. Get_menu(): returns the menu of a food establishment, including all the dishes which it is made up of.
   iv. Verify_reviewer(user): Verify if a dish was rated by a professional or a lay person. A user can search for dished that have been rated by a professional chef (that is not affiliated with said establishment) or by a professional food critique/taster, which is someone who makes a living out of tasting different types of dishes or drinks. These types of critiques will have a major impact on the rating of the dish due to their professional backgrounds, but certain criteria must be met before their rating is said to be professional.

i. Domain Events
   i. User gets cravings
      1. The user is eager to eat a particular dish and wonders which establishment can cater to their needs.
   ii. Return location
      1. After the user decides on the establishment that they want to dine at, the browser will give them the location for the establishment to get there through the fastest route.
   iii. Rating a dish
      1. A user can upvote a dish within its own category. This user is either a professional or a lay person, and this impacts the weight of their rating.

j. Domain Behaviors
   i. Dish rating
      1. A user can go to the website and decide to upvote an establishment's dish within that dish's category. The user can only upvote for one dish which will determine their favorite establishment that serves that dish. At any moment, the user can change their mind and upvote another establishment, but they would have to revoke their previous vote.
   ii. User eats at restaurant
      1. User chooses a particular item (salad, pizza, etc.) and orders. After eating, they decide whether they enjoyed their meal and how tasty it was.

IV. Domain Requirements

a. Search a Dish
   i. A user can search for a particular dish based on the search criteria that they decide to utilize whether it's by the dish with the highest rating, by search based on a radius of a specified location (your current location for example),

search based on the cost of the dish, or search by category (pasta, burgers, tacos, wraps, etc.).

b. Best Dish Searcher Website
    i. The end of the product should be a website where users can login and search for the best dishes due to the ratings of the general public by an upvote system where the user can only vote for a single dish of a particular type (for example mofongo, chicken, steak…). Which will help the user decide on a place that they want to go given a specific dish that they are craving now.

V. Interface Requirements

a. shared data initialization requirements – All users should have access to the same data for searching places to eat.
b. shared data refreshment requirements – Data of the dishes and establishments must be constantly updated according to new reviews changing the score of any of the items.

VI. Machine Requirements

a. Performance
    i. The machine shall serve 500 common and 200 verified users, a total of 700 users.
    ii. The machine's average response time shall be at most 1.6 seconds, when the system is on a heavy load.
b. Dependability
    i. The machine shall always be accessible for users. Every user shall have the appearance that s/he has exclusive access to the system.
    ii. The machine shall always be available, except when under maintenance.
    iii. The machine shall encrypt all stored user sensitive data (passwords, etc.).
c. Maintenance
    i. The machine's average time between failures shall be at least 30 days, and downtime due to failure shall be less than 3 hours.
d. Platform
    i. The machine shall be developed on a UNIX operating system.
    ii. The machine shall be compatible with all the libraries and 3rd party software for correct execution.
e. Documentation
    i. We shall provide installation, support, user, contribution, and development guidelines.

## VII. Software Architecture Design

The Use Case Diagram is used as a primary tool for a new software project underdeveloped. The UML use case diagram represent and specifies the expected behavior of the system. Figure 1 represent the connection and behavior of the software that will be developed. There are two kind of user, verified and common users. Verified users can see the top-rated dishes and make a review. The system will be capable of showing the top/recommended dishes based on the reviews. The common user could search a dish by its location, restaurant, or category. The expected goal is the user could get the best rated dishes of what he/she want to eat at this exact moment, or the best dish to a certain location.

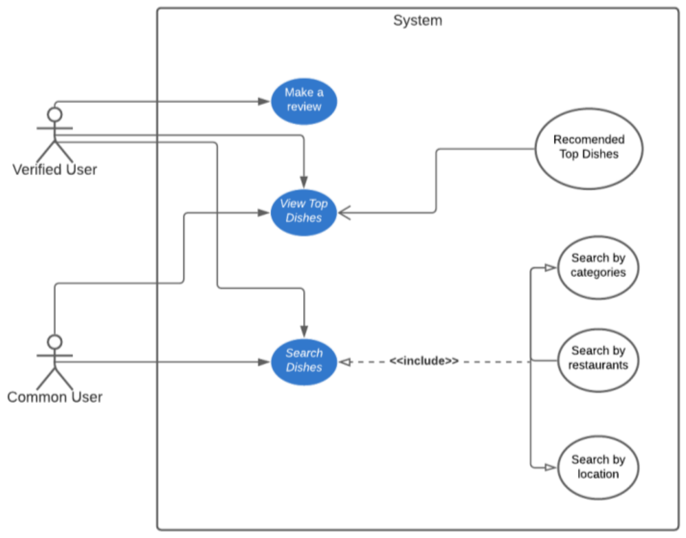


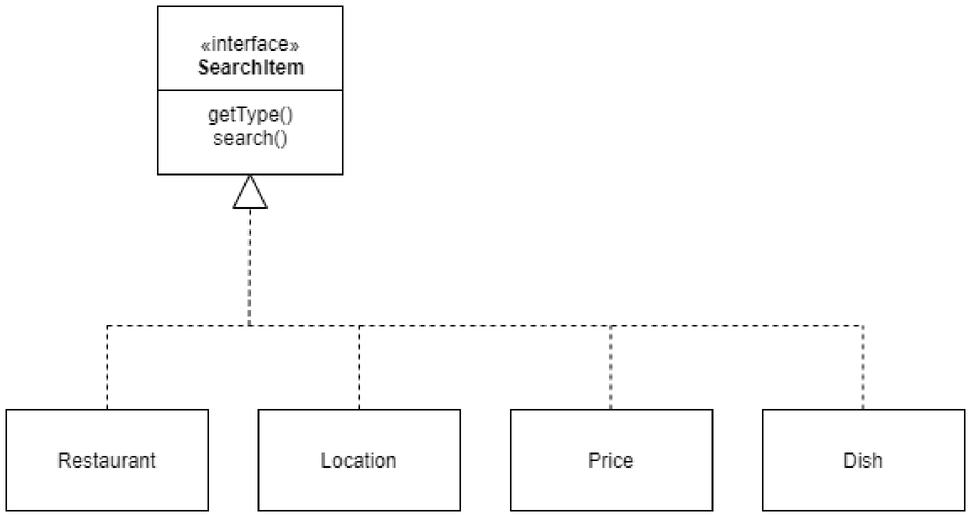
*Figure 1: UML Use Case Diagram.*

## VIII. Software Component Design

a. Principles

i. Design by Contract: It denotes that the relationship between the class and its clients constitute an agreement. In our case, if the user searches for dishes that contain the property of City = “Mayagüez”, then the Search List agrees to contain only dishes that are in Mayagüez.

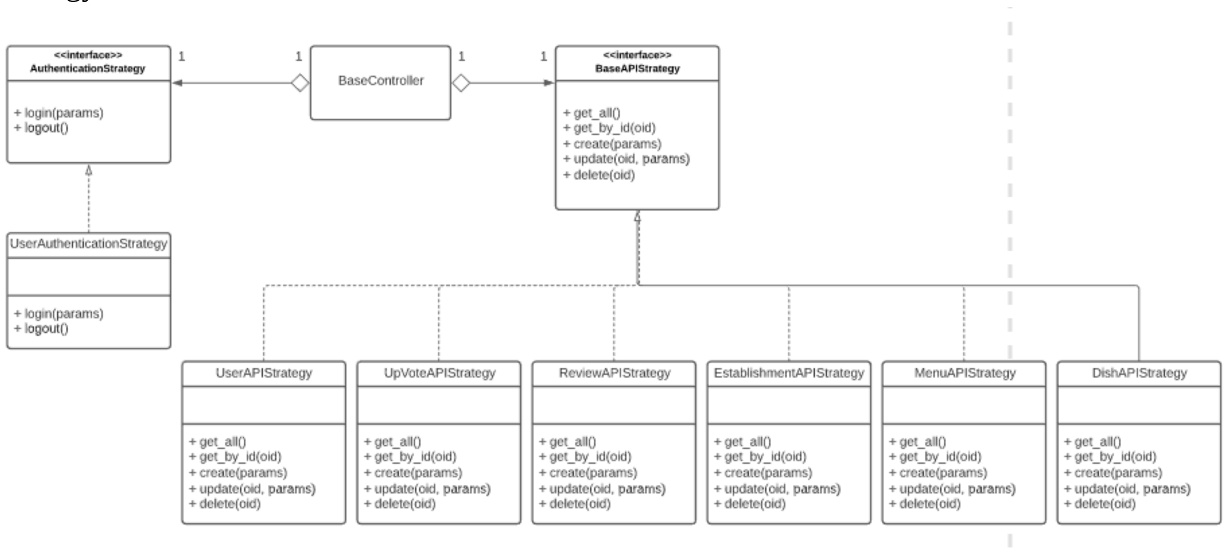
ii. Open-closed Principle: This principle establishes that there should always be room for extension of a module, but the module should be closed to modifications. Let’s take the example of a SearchItem interface. Here, it is ideal to keep adding classes that implement SearchItem, yet it is not ideal to modify the SearchItem interface when various classes already depend on it.



b. Identify and Elaborate Classes
   i. Dish - has properties like price, rating, and establishment where it is served. It also has functions like rate () to allow a user to give a rating of such dish.
   ii. Restaurant - has properties like location and rating. Has a list data structure that contains the menu items they serve. Has functions like getTopDishes() to get the best they have to offer.
   iii. Location - has properties like latitude and longitude. Has functions like getDistance() that returns the distance of that location with respect to the user.
   iv. User - have properties like name, age, isVerified (for users whose food opinions weigh more). Has functions like getLocation() to determine the current location of the user.

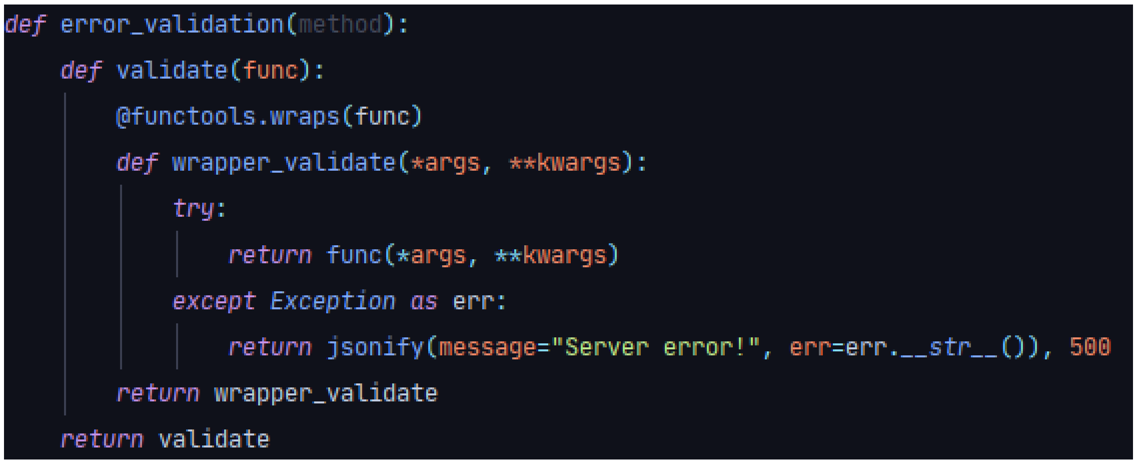
## IX. Selected Fragments of Implementation

a. Strategy Pattern:



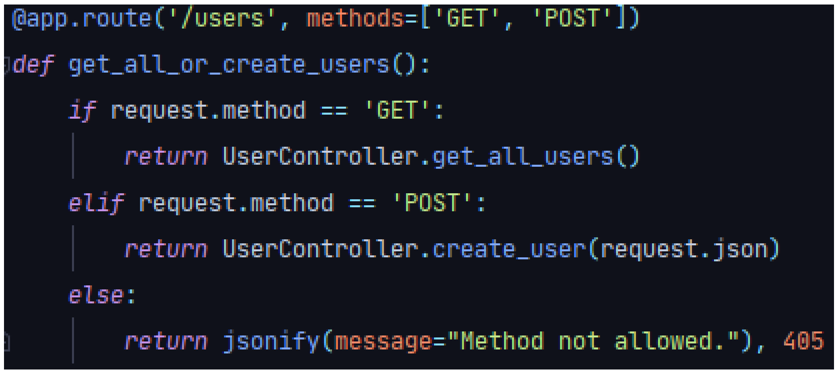
b. Implemented:
   i. Since Python is not a purely functional language, we use **decorators** to apply functional programming concepts like first-class objects.
      1. Error validation on model controllers of the api:
         a. The idea is to use the function error_validation as a decorator for every model controller method (controller contains all the CRUD methods) to validate the given user parameters and check for server errors. Its main purpose is to minimize the code in the CRUD method and make it more readable.

*/helpers/decorators.py*



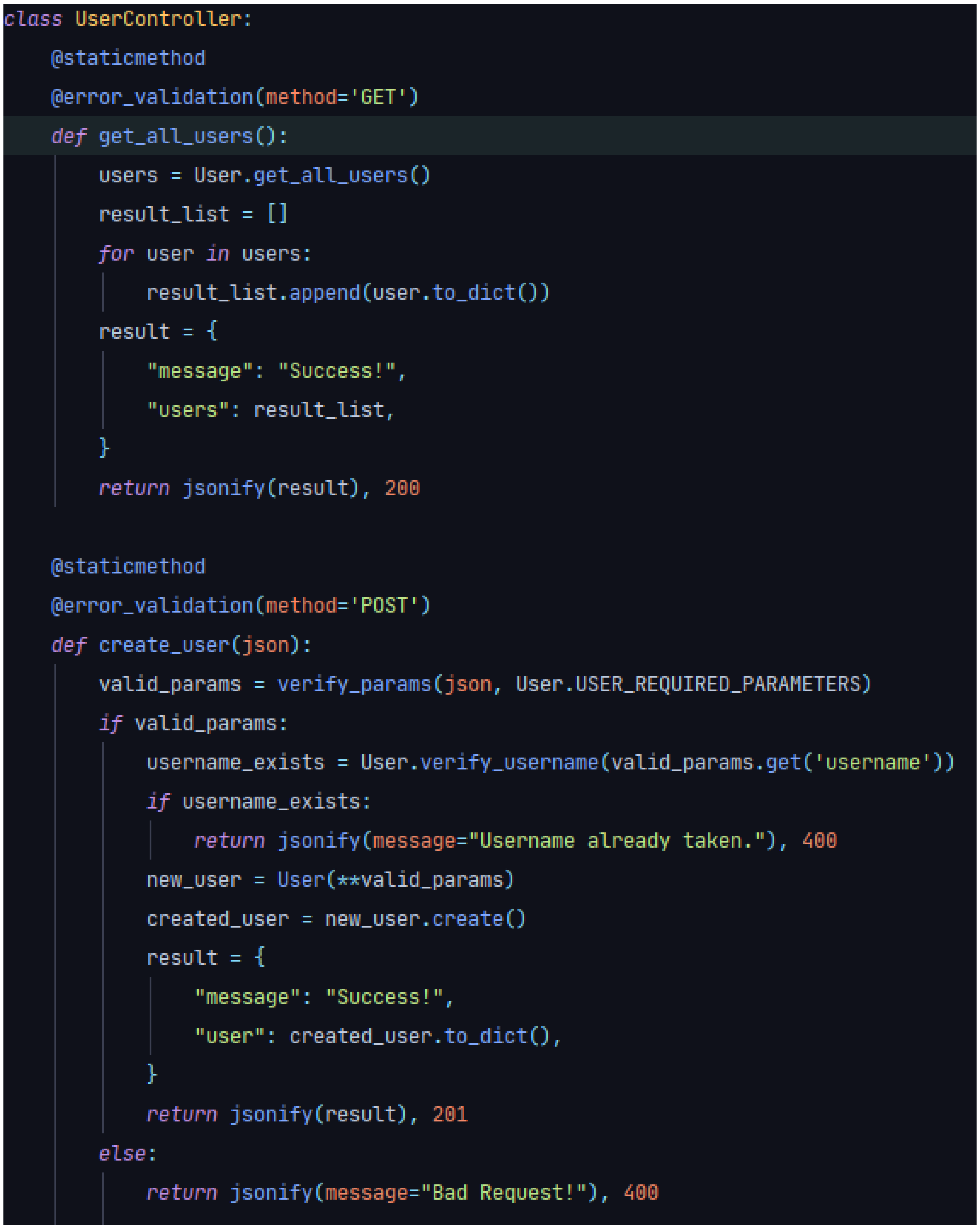
```python
def error_validation(method):
    def validate(func):
        @functools.wraps(func)
        def wrapper_validate(*args, **kwargs):
            try:
                return func(*args, **kwargs)
            except Exception as err:
                return jsonify(message="Server error!", err=err.__str__()), 500
        return wrapper_validate
    return validate
```

*app.py*



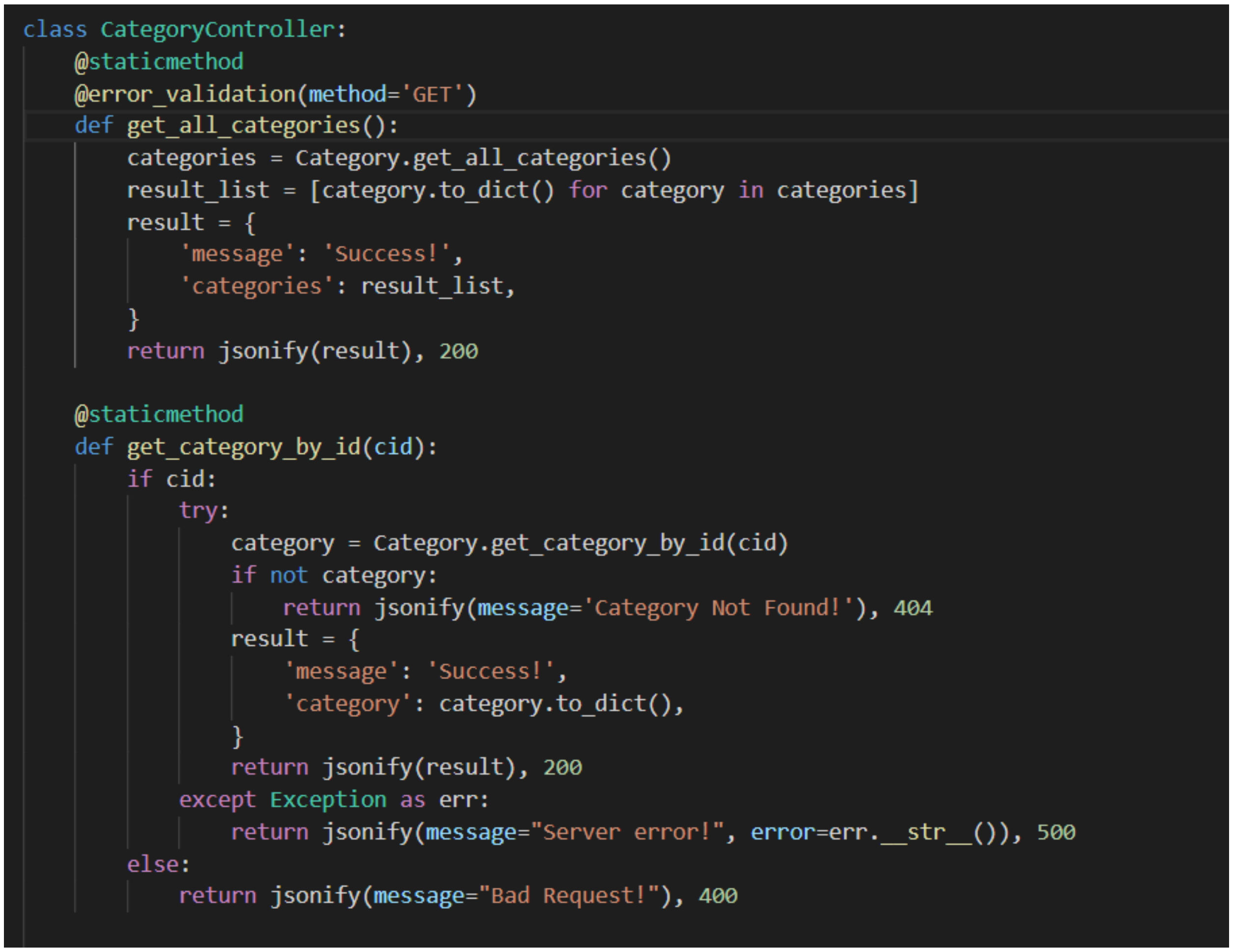
```python
@app.route('/users', methods=['GET', 'POST'])
def get_all_or_create_users():
    if request.method == 'GET':
        return UserController.get_all_users()
    elif request.method == 'POST':
        return UserController.create_user(request.json)
    else:
        return jsonify(message="Method not allowed."), 405
```

*/user/user_controller.py*


```python
class UserController:
    @staticmethod
    @error_validation(method='GET')
    def get_all_users():
        users = User.get_all_users()
        result_list = []
        for user in users:
            result_list.append(user.to_dict())
        result = {
            "message": "Success!",
            "users": result_list,
        }
        return jsonify(result), 200

    @staticmethod
    @error_validation(method='POST')
    def create_user(json):
        valid_params = verify_params(json, User.USER_REQUIRED_PARAMETERS)
        if valid_params:
            username_exists = User.verify_username(valid_params.get('username'))
            if username_exists:
                return jsonify(message="Username already taken."), 400
            new_user = User(**valid_params)
            created_user = new_user.create()
            result = {
                "message": "Success!",
                "user": created_user.to_dict(),
            }
            return jsonify(result), 201
        else:
            return jsonify(message="Bad Request!"), 400
```

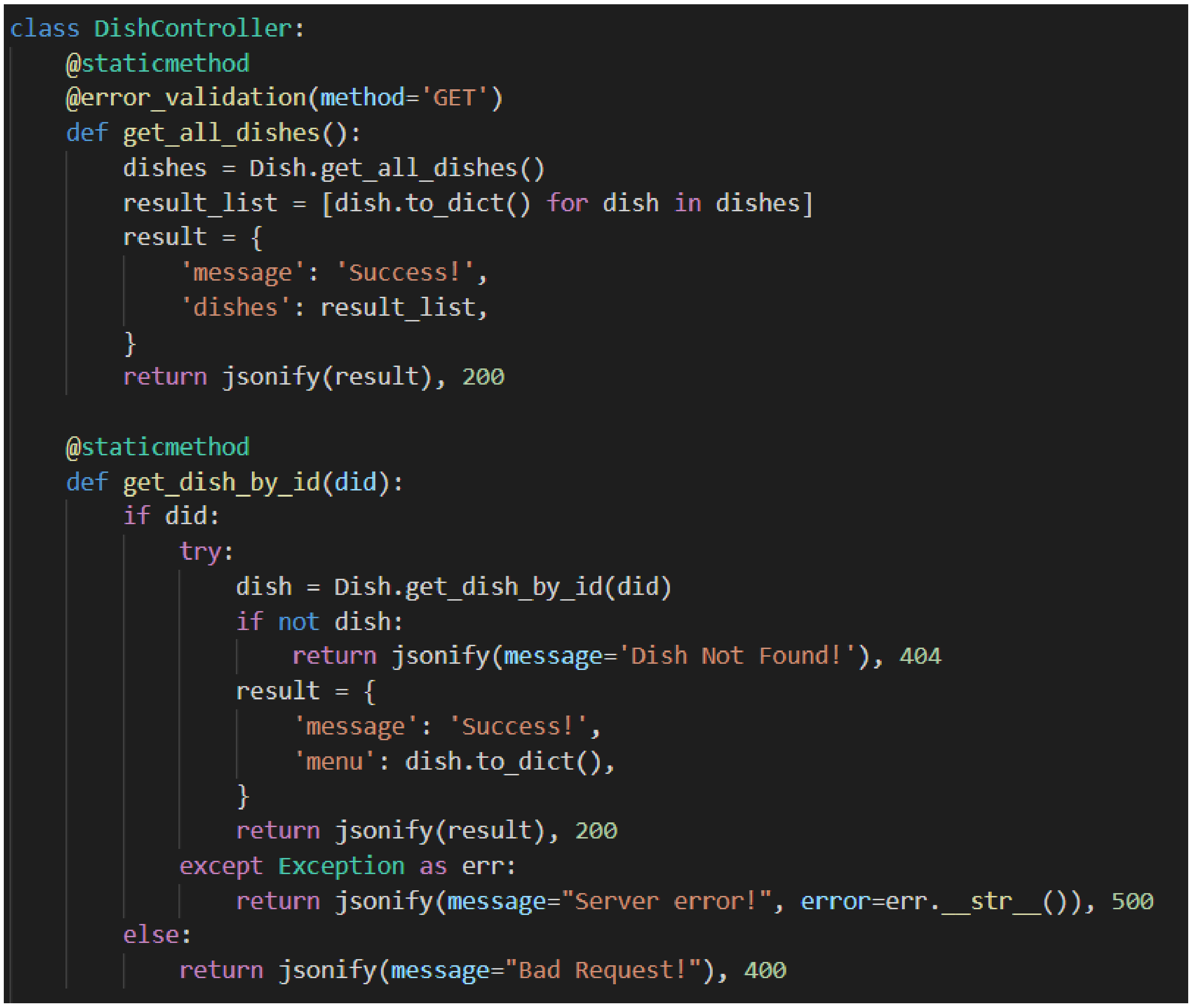
*/category/category_controller.py*



```python
class CategoryController:
    @staticmethod
    @error_validation(method='GET')
    def get_all_categories():
        categories = Category.get_all_categories()
        result_list = [category.to_dict() for category in categories]
        result = {
            'message': 'Success!',
            'categories': result_list,
        }
        return jsonify(result), 200

    @staticmethod
    def get_category_by_id(cid):
        if cid:
            try:
                category = Category.get_category_by_id(cid)
                if not category:
                    return jsonify(message='Category Not Found!'), 404
                result = {
                    'message': 'Success!',
                    'category': category.to_dict(),
                }
                return jsonify(result), 200
            except Exception as err:
                return jsonify(message="Server error!", error=err.__str__()), 500
        else:
            return jsonify(message="Bad Request!"), 400
```

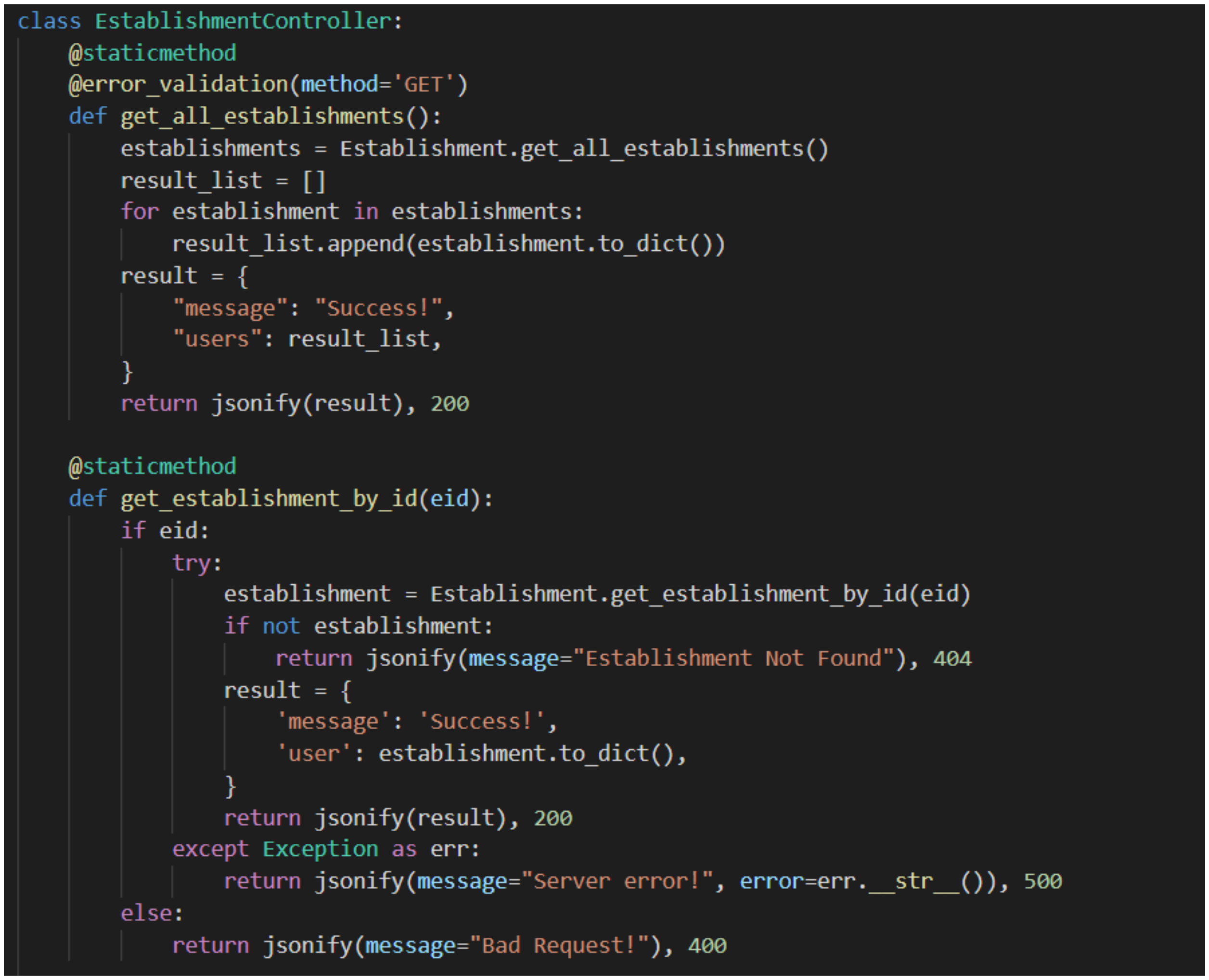
*/dish/dish_controller.py*



```python
class DishController:
    @staticmethod
    @error_validation(method='GET')
    def get_all_dishes():
        dishes = Dish.get_all_dishes()
        result_list = [dish.to_dict() for dish in dishes]
        result = {
            'message': 'Success!',
            'dishes': result_list,
        }
        return jsonify(result), 200

    @staticmethod
    def get_dish_by_id(did):
        if did:
            try:
                dish = Dish.get_dish_by_id(did)
                if not dish:
                    return jsonify(message='Dish Not Found!'), 404
                result = {
                    'message': 'Success!',
                    'menu': dish.to_dict(),
                }
                return jsonify(result), 200
            except Exception as err:
                return jsonify(message="Server error!", error=err.__str__()), 500
        else:
            return jsonify(message="Bad Request!"), 400
```

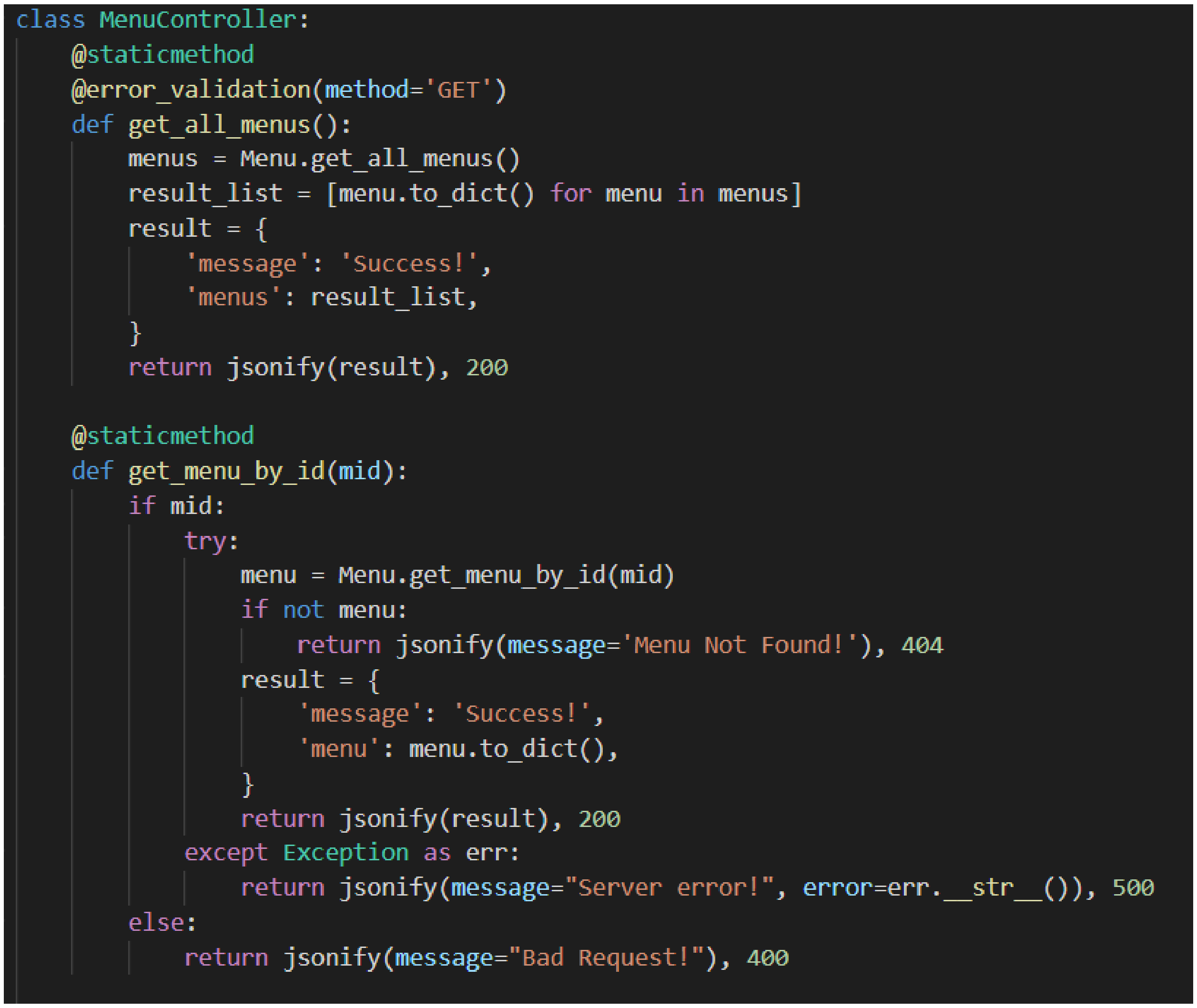
*/establishment/establishment_controller.py*



```python
class EstablishmentController:
    @staticmethod
    @error_validation(method='GET')
    def get_all_establishments():
        establishments = Establishment.get_all_establishments()
        result_list = []
        for establishment in establishments:
            result_list.append(establishment.to_dict())
        result = {
            "message": "Success!",
            "users": result_list,
        }
        return jsonify(result), 200

    @staticmethod
    def get_establishment_by_id(eid):
        if eid:
            try:
                establishment = Establishment.get_establishment_by_id(eid)
                if not establishment:
                    return jsonify(message="Establishment Not Found"), 404
                result = {
                    'message': 'Success!',
                    'user': establishment.to_dict(),
                }
                return jsonify(result), 200
            except Exception as err:
                return jsonify(message="Server error!", error=err.__str__()), 500
        else:
            return jsonify(message="Bad Request!"), 400
```

*/menu/menu_controller.py*



```python
class MenuController:
    @staticmethod
    @error_validation(method='GET')
    def get_all_menus():
        menus = Menu.get_all_menus()
        result_list = [menu.to_dict() for menu in menus]
        result = {
            'message': 'Success!',
            'menus': result_list,
        }
        return jsonify(result), 200

    @staticmethod
    def get_menu_by_id(mid):
        if mid:
            try:
                menu = Menu.get_menu_by_id(mid)
                if not menu:
                    return jsonify(message='Menu Not Found!'), 404
                result = {
                    'message': 'Success!',
                    'menu': menu.to_dict(),
                }
                return jsonify(result), 200
            except Exception as err:
                return jsonify(message="Server error!", error=err.__str__()), 500
        else:
            return jsonify(message="Bad Request!"), 400
```

## Analytic Part

I. Concept Formation
   a. There are the concrete phenomena of meals, drinks, desserts, and starters. We abstract these concepts into one representation: dishes. There are also the concrete phenomena of bars, restaurants, fast food restaurants, and "chinchorros". These can be abstracted into the concept of food establishments.

II. Validation
   a. Domain validation report:
      i. Descriptive Domain Terminology – The craving terminology is not necessary for the domain. It is not a factor present at all instances of the

domain. The source is also not always present and does not account to any other terminology such as rating.

ii. Domain Entities – The drink entity can be abstracted with dishes because they serve the same function in the scope of the food establishment, they are not treated differently by the users if they are searching a drink. It can be specified as a type of dish

iii. Domain behaviors – In the rating search it must be specified that dishes are being searched by category for the results to reflect the best rated dish in that specified category.

iv. Domain requirements – It is not specified how an account is "verified" and how much it affects the overall rating of a dish.

b. Stakeholder Validation:

Two previous restaurants chefs and owners where interviewed to gain insight in the domain of food establishments and how these types of businesses are managed.

i. Why did you decide to work in the restaurant industry?
   1. Yesenia Lopez and Alexis Cuevas: To follow the path I study

ii. How do you resolve conflicts with co-workers?
   1. Yesenia Lopez and Alexis Cuevas: By talking with my co-workers

iii. What is your favorite experience that you received from a client?
   1. Yesenia Lopez and Alexis Cuevas: Having a loyal customer that wanted me to cook for him

iv. What is the worst customer experience that you received?
   1. Yesenia Lopez and Alexis Cuevas: Messy customers

v. What things do costumers do that you find annoying?
   1. Yesenia Lopez and Alexis Cuevas: Manage employees

vi. What is the most difficult thing about managing a restaurant?
   1. Yesenia Lopez and Alexis Cuevas: That does not know about food and criticizes.

vii. What is the best thing you like about managing a restaurant?
   1. Yesenia Lopez and Alexis Cuevas: The ability of changing menus from time to time.

## III. Verification

a. Verification by informal reasoning: A dish is a representation of an item that is sold for consumption; therefore, a dish can be of different categories, it is not only defined by complete "meals", but as an item in the menu of the establishment. Those include drinks and snacks, depending on the type of establishment.

b. Verification of domain requirement 1. *Search a Dish:* In the need of searching something to eat (i.e. a *dish*), a person wants the best recommend dish for their craving and to have a great experience based on their needs. By searching for a dish, the system displays a list of dishes sorted from best to least recommended dish for his/her craving and fulfilling their needs (price, location, and rating). Thus, enhancing the searchability for a variety of dishes.

## User Stories

<u>Every user must login to his/her account.</u>

- Normal user:
    - The user can search by dish, restaurants, location, and/or price.
    - The user can search for a particular dish and see a list of different versions of that dish sorted from most to least preferred (rating and upvotes[popularity]).
    - The user can decide how many dishes the list can display, from a minimum of 1 and a maximum of 10.
    - The user can upvote only one dish for every individual dish category (Chicken, Red meat, Fish, Lamb, Pork, Crustacean, Sushi, Pasta, Noodles, Burgers, Sandwiches, Pizza, Pasta, Taco, Kebab, Wraps, Soup, Fried food, Curry, Stew, BBQ, Bread, Cheese, Doughnut, Muffin, Cake, Pie, Cookies, Pastries, Pretzel, Cheese cake, Ice cream, American, Vietnamese, Korean, Spanish, Thai, Arabic, Ethiopian, Indian, Turkish, Italian, Chinese, Japanese, Mexican, Criollo, Legumes, Vegetarian, Vegan)
    - The user can select a dish and view all the information related to the dish including reviews, establishment, location (map to the establishment), price, etc.
    - The user can view a list of most frequent upvoted dishes (3-5) on a given period (weekly and monthly).
    - The user can optionally sign out of the account after using it.
- Verified users (food critique):
    - The food critique does everything a normal user can do.
    - The food critique can create and/or edit (max 30min) his/her review of a dish.
- Restaurant user:
    - The restaurant user can create, view, edit, and or remove dishes to/from the system's data base.
    - Displayable dish.

    - The restaurant user cannot do what a normal user can (such as upvote a dish).
- Admin user:
    - Validate verified and establishment user applications.
    - The admin user can upgrade a normal user to a verified user.
    - The admin user can revoke the verified user status.

## Roles

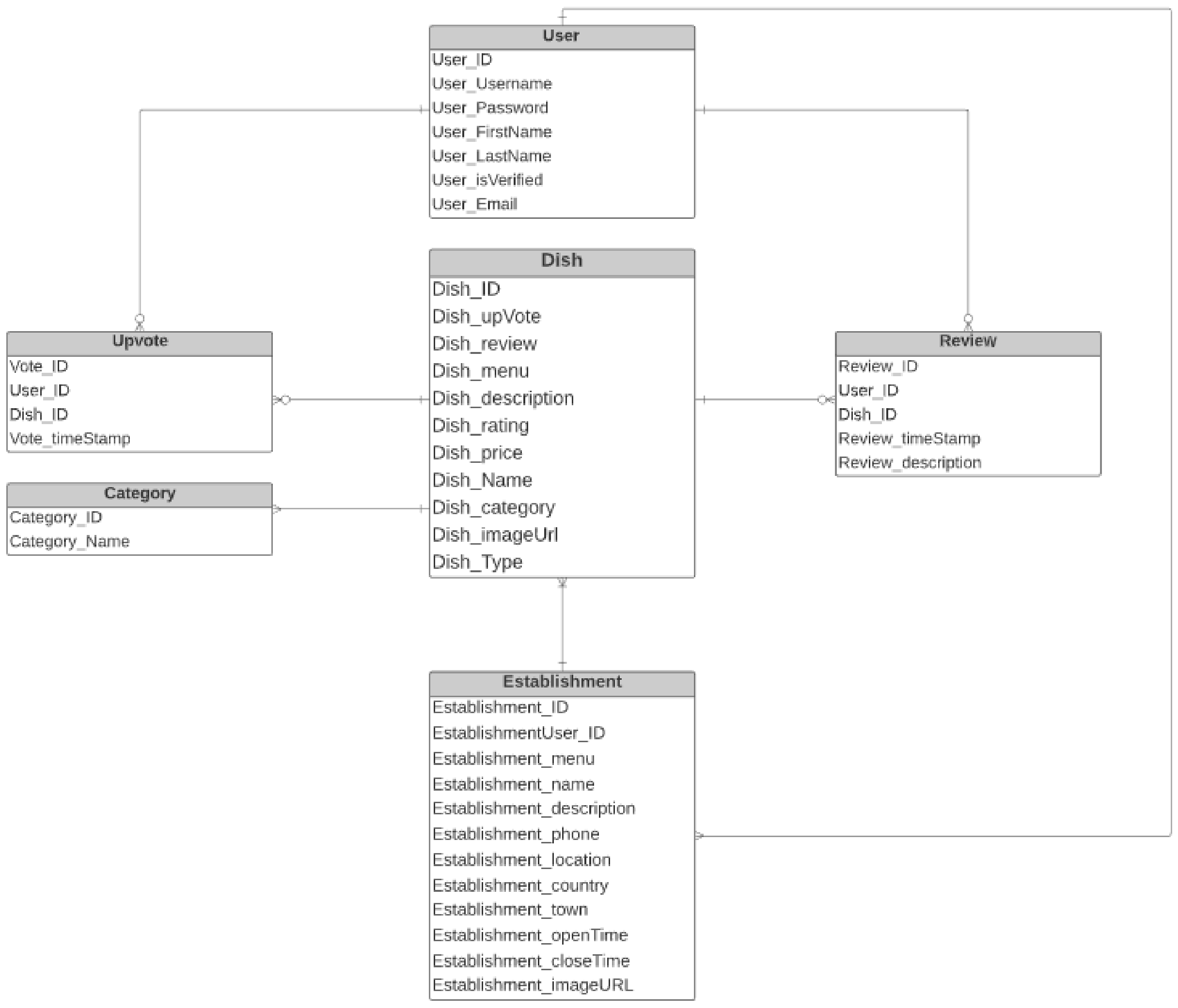
Note: the role of SCRUM Master will be changed at the beginning of each new sprint.

1. Estefanía Torres Collado
    a. Role: Frontend Leader
    b. Responsibilities
        i. Create wireframes for the UI
        ii. Help in the frontend development
2. Andrea C. Miranda Acevedo
    a. Role: Frontend member
    b. Responsibilities
        i. Review peer code
        ii. Work in the frontend development
3. David Carrion Beníquez
    a. Role: Frontend member
    b. Responsibilities
        i. Review peer code
        ii. Part of frontend development
4. Christopher Vegerano Lopez
    a. Role: Frontend member
    b. Responsibilities:
        i. Review peer code
        ii. Do part of the frontend development
5. Gabriel R. Pantojas Burgos
    a. Role: backend member
    b. Responsibilities
        i. Review peer code
        ii. Do part of the backend development
6. Everson Rodríguez Muñiz
    a. Role: Backend Leader
    b. Responsibilities
        i. Organize the team for weekly meetings and determine what is going to be discussed.

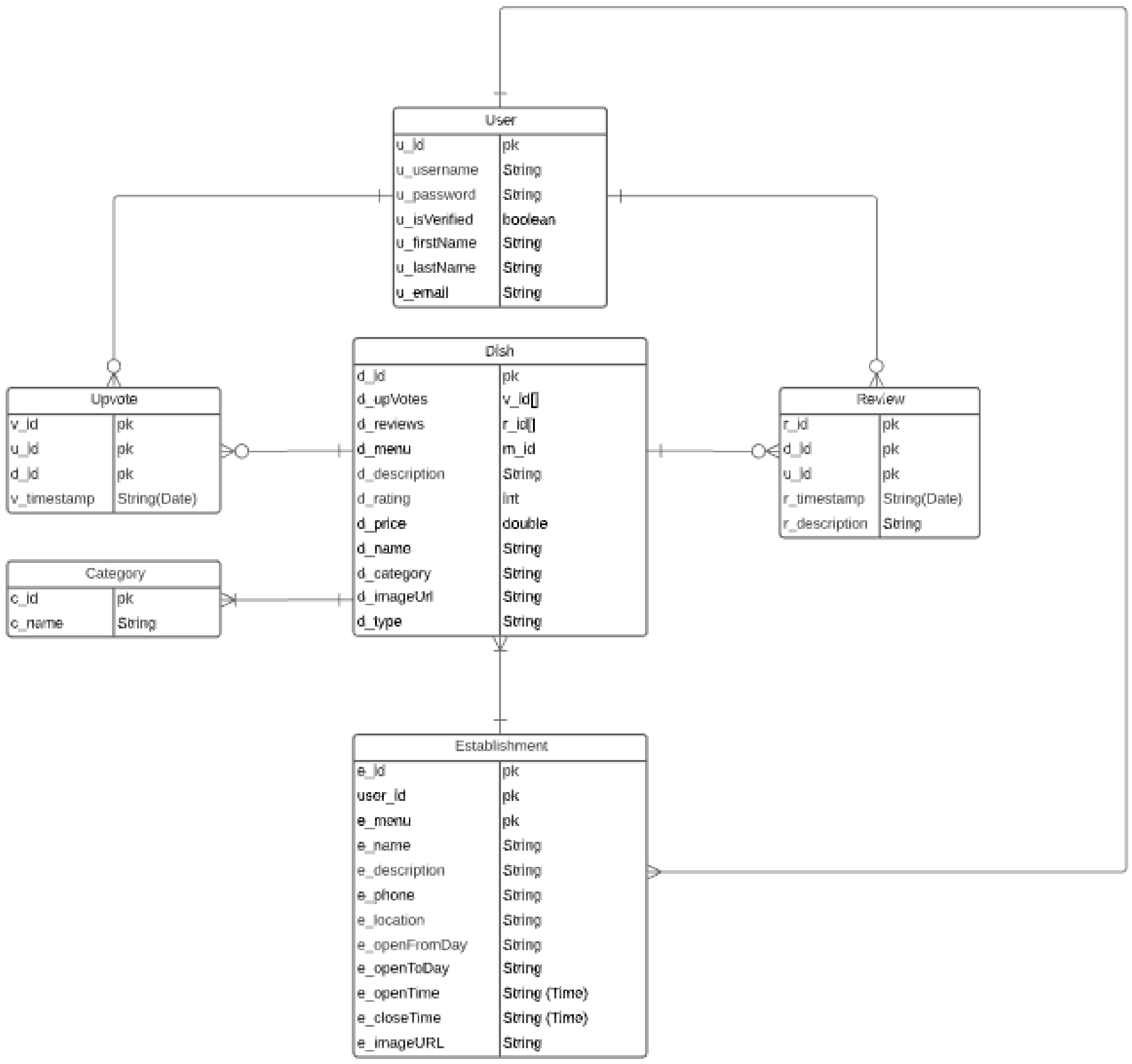
   ii. In charge of development and correct integration of the backend.
   iii. Create initial ERD diagram for discussion.
   iv. Scrum master for the first sprint (starting next week).

## Backend Entity Relationship Diagram (ERD)

1. This first ERD is the second implementation that we have done due to the feedback given by the professor during our presentation, which only describes the entities.
   a. Due to a unanimous all the member of the group, we added a new entity to the ERD to create a table for it in the backend side of the program, which is the category entity.
   b. Some minor modifications were made to the dish entity.
   c. Some minor changes were made to the establishment entity.
   d. Due to a unanimous decision by the whole group, the menu and establishment User entity were removed from the ERD because they were not necessary to have as a complete entity and they could just be added as a part of another entity or be removed as a whole.



2. This was the initial implementation that we had of the ERD that we utilized. This implementation differs from the previous one with respect that it is more implementation drive.
    a. Due to a unanimous all the member of the group, we added a new entity to the ERD to create a table for it in the backend side of the program, which is the category entity.
    b. Some minor modifications were made to the dish entity.
    c. Some minor changes were made to the establishment entity.
    d. Due to a unanimous decision by the whole group, the menu and establishment User entity were removed from the ERD because they were not necessary to have as a complete entity and they could just be added as a part of another entity or be removed as a whole.



## Technology Stack

1. Programming language
   a. We decided to utilize python,
      i. We decided that python would be the better programming language due to most of the members have had previous knowledge utilizing it, giving us a strong base where we would not have to start learning a language.
2. Framework
   a. For the backend we decided to utilize flask
      i. Since the backend team is comprised of two members, we decided to utilize flask since one of the two members already has previous experience utilizing the framework.
3. Data base

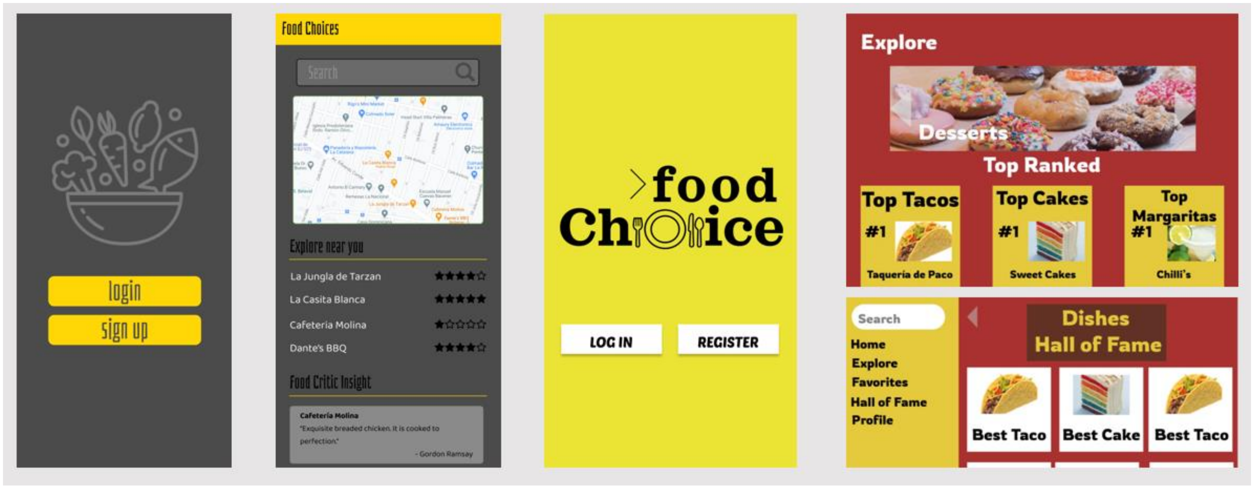
   a. To handle and store out date base we decided to use PostgreSQL
      i. This system is very highly backed up due to its high level of resilience and correctness, it is also used as the primary data store for may apps.
4. Tools
   a. In the backend we decided to utilize SQLAlchemy
      i. Since we are using python as our programming language, we decided to utilize SQLAlchemy since it will facilitate us connecting the program with a database.

## Backend

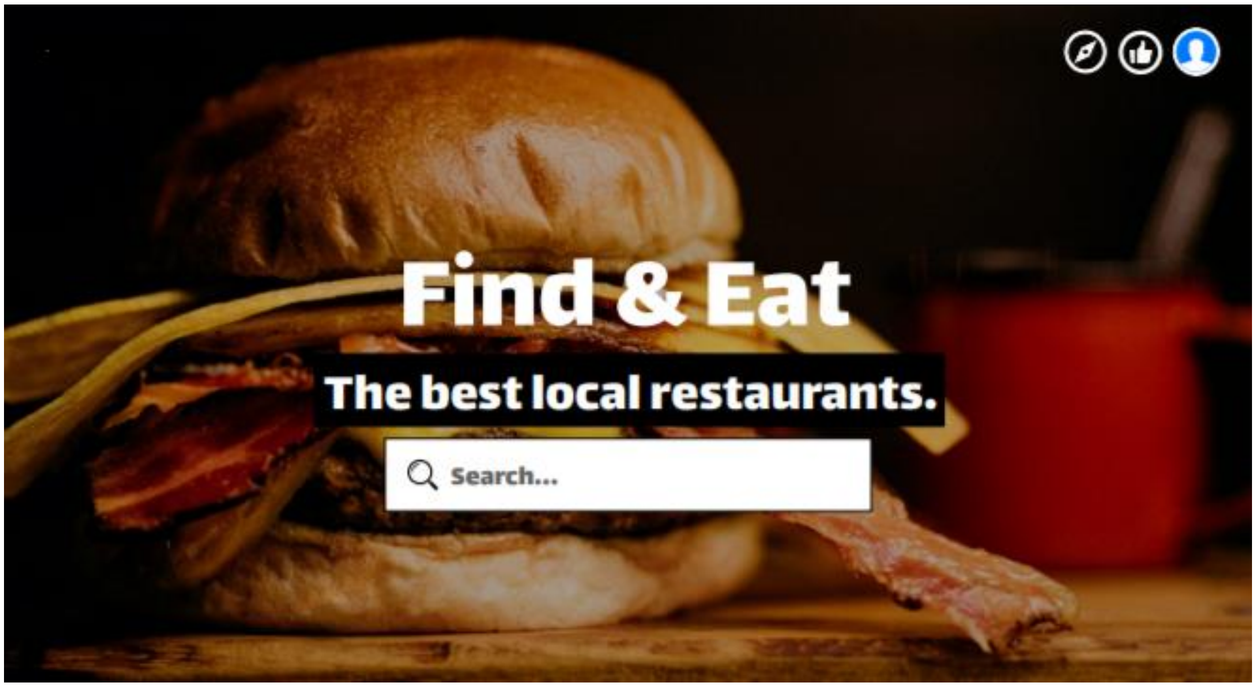
1. The backend team has already defined the data access objects and the connection to the database.
2. Creation of tables and relationships.
3. All tables and relationships have been defined.
4. All the endpoints have been defined.
5. The backend is working on the final calculation of the ratings (which is the final thing that needs to be implemented so the backend is completely done) so that it will be finished for the demo recording that is due on November 30, 2020.

## Frontend

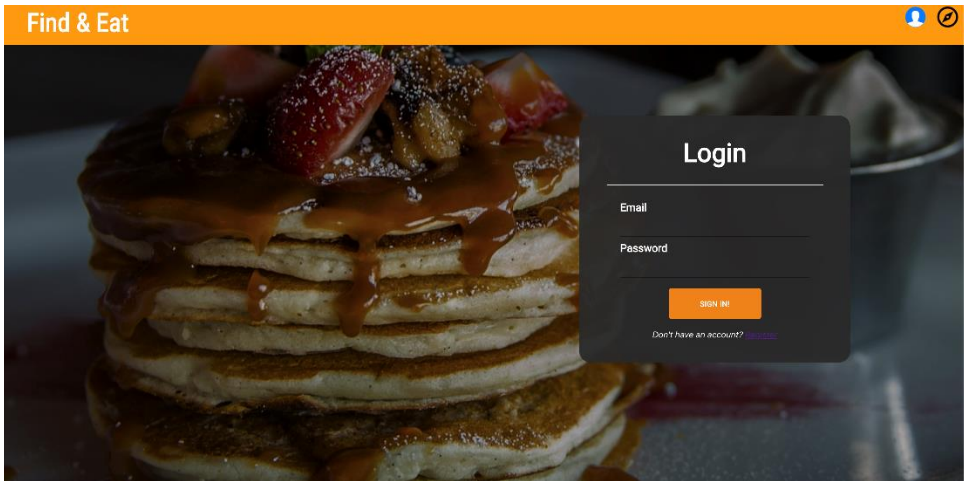
1. Each member of the frontend team was assigned to design a wireframe for the application
   a. From all the wireframes that were created these were the ones that did not make the cut

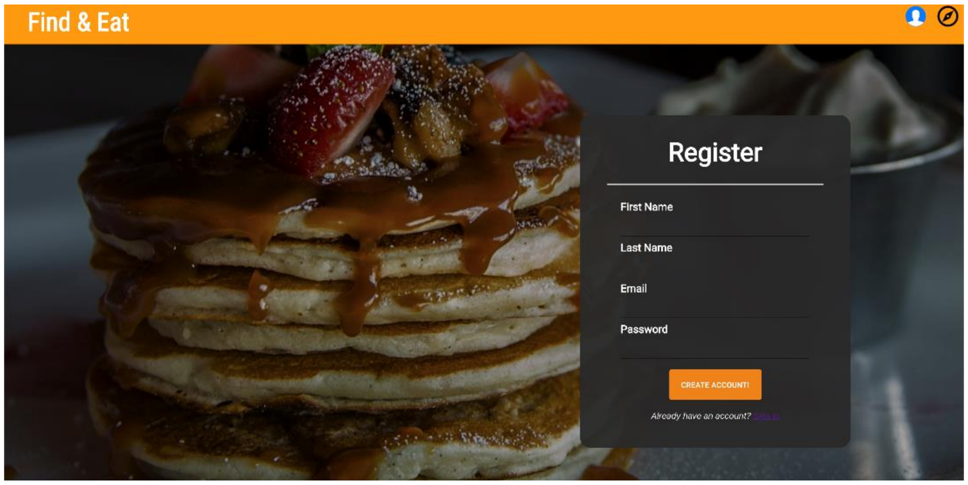


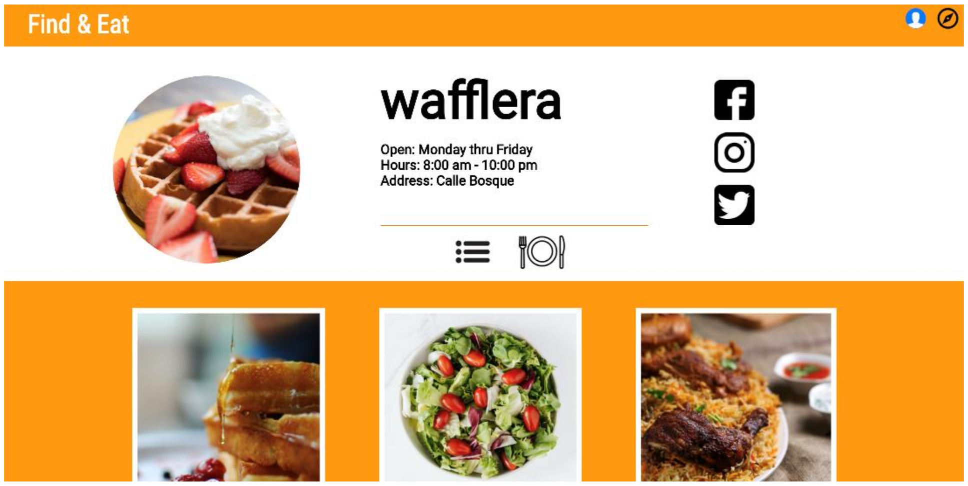
      i. Even though they did not make the cut the wireframe that won will be updated to use some of the feature that these previous wireframes utilized to make it as appealing as possible for the users.
   b. The following wireframe is the one that won by decision of the whole team



2. Frontend Progress
    a. The frontend team started the development using React with typescript.
    b. The Material UI library was used for the styling of things such as the text fields, icons, and tables.
    c. The axios react library was used for the http request.
    d. The main feature of the app page has been implemented but due to some minor bugs the frontend team has yet to push that portion of the exam to the repository.
    e. The frontend also implemented the log-in page, sign-up page (also known as the register page), the restaurant manager page, restaurant page, update the user page, and the dish page.
    f. The frontend is fixing some minor bugs on the implementation to have the whole web page up and running perfectly for the demo that is due on November 30, 2020.

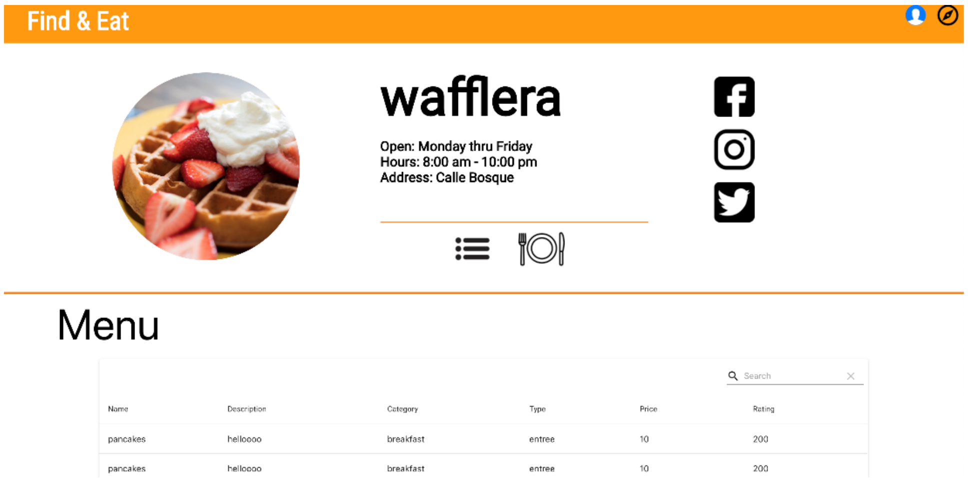






wafflera

Open: Monday thru Friday
Hours: 8:00 am - 10:00 pm
Address: Calle Bosque



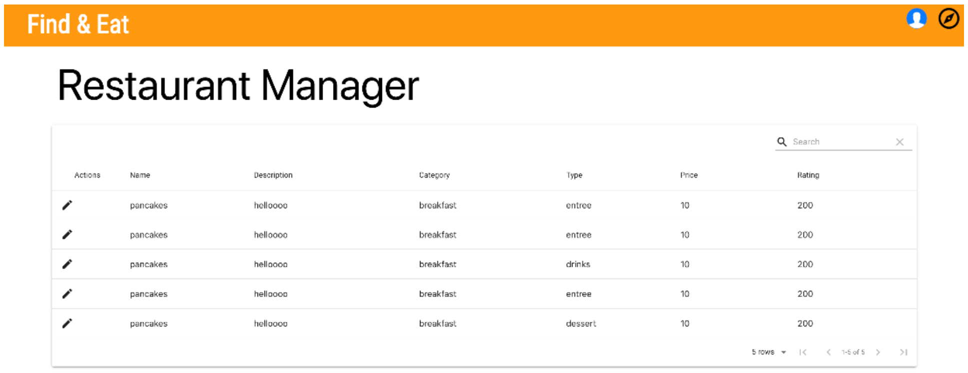
wafflera

Open: Monday thru Friday
Hours: 8:00 am - 10:00 pm
Address: Calle Bosque

Menu

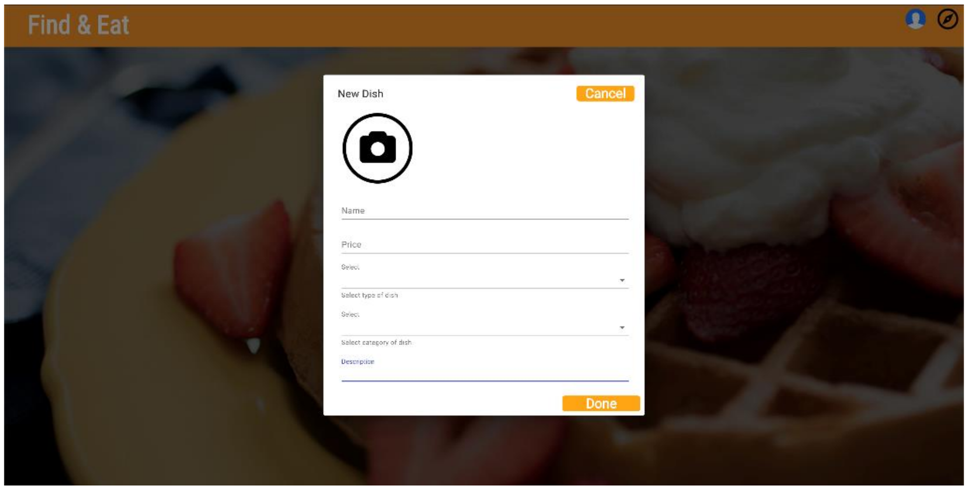
Search

| Name | Description | Category | Type | Price | Rating |
|---|---|---|---|---|---|
| pancakes | helloooo | breakfast | entree | 10 | 200 |
| pancakes | helloooo | breakfast | entree | 10 | 200 |


Find & Eat

Restaurant Manager

| Actions | Name | Description | Category | Type | Price | Rating |
|---|---|---|---|---|---|---|
| ✎ | pancakes | helloooo | breakfast | entree | 10 | 200 |
| ✎ | pancakes | helloooo | breakfast | entree | 10 | 200 |
| ✎ | pancakes | helloooo | breakfast | drinks | 10 | 200 |
| ✎ | pancakes | helloooo | breakfast | entree | 10 | 200 |
| ✎ | pancakes | helloooo | breakfast | dessert | 10 | 200 |

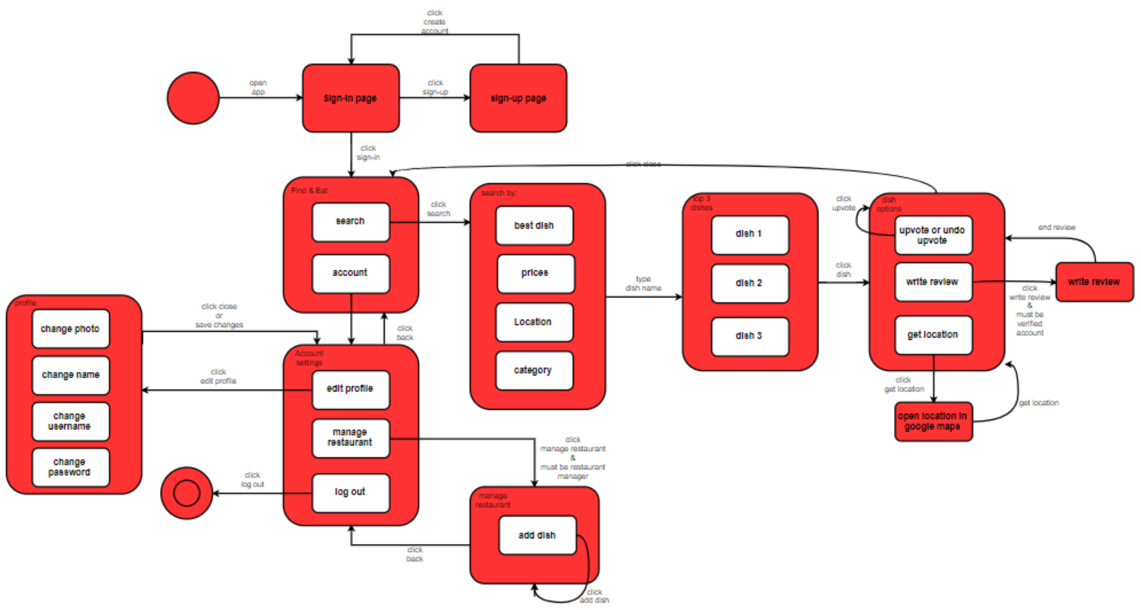


## Feedback implementations/corrections (for phase 2):

- The synopsis was rewritten to cover more on the requirement prescription, software design, implementation, and how the project is going to run.
- The domain entities, domain functions, domain events, and domain behavior were all added as fields inside the terminology.
- Functions such as delete dish and create dish were deleted from the domain functions due to them being functions in the application layer and they do not belong in the domain functions.
- Dish was searched was removed from the domain events and a new event was added.
- Fixed boxes issues in the use case diagram.
- Validation was made with people who are currently chefs.

## State Chart

1. This State chart was made with the purpose for a basic overview of how the whole web application works. On the next points the state chart will be explained in detail:
    a. To access the application a user must first open the application.
    b. Once a user has opened the app the User can log-in to their account if they already have one, if not the application will give you the option to create an account.
        i. To create an account the user just has to click on sign-up and fill all of the fields that are required and afterwards you are able to log in.
    c. After logging in to your account you will be send to the main Find&Eat where you are able to search for a particular dish or you are able to manage the account.
    d. If a user decides to search for a particular dish, they can search based on different search engines (best dish, prices, location, and category).
        i. Once they have searched for the dish a list of the top 3 dishes will appear(based on their preference of the search engine) afterwards you can view each dish individually and once you are on the dish page, you can upvote it(basically making that dish your favorite of its category), you can write a review on the dish(this field is only applicable for verified users, users that have a background of the food industry: chefs, food critiques, etc.) and/or you can get the location of the establishment where the dish is served.
    e. If a user decides to enter the account field, they can edit their profile, manage restaurant, and/or log out.
        i. If the user decides to edit their profile, they can change their picture, change their name, their username, and/or password.
        ii. If the user decides to manage their restaurant, they are able to add dishes to their establishment, write a description for the dish, add to what category they belong too, and they can add some information of their establishment like its operating hours and they days that the establishment is open or closed.
        iii. Finally, once the user is finished using the application then they can log out of their account finishing the whole process of using the application.



click create account
open app
Sign-in page
click sign-up
sign-up page
click sign-in
click close
Find & Eat
search
account
click search
search by:
best dish
prices
Location
category
type dish name
top 3 dishes
dish 1
dish 2
dish 3
click dish
click upvote
dish options
upvote or undo upvote
write review
get location
end review
write review
click write review & must be verified account
click get location
open location in google maps
get location
profile
change photo
change name
change username
change password
click close or save changes
click back
Account settings
edit profile
manage restaurant
log out
click edit profile
click log out
click manage restaurant & must be restaurant manager
manage restaurant
add dish
click back
click add dish