

Foodie Connector

Design Document

Team 25

Zequan Wu

Shuqi Ma

Shiwen Xu

Xiaonan Shen

Purpose

Ordering food delivery is not as easy as it seems. People usually waste money on the food they don't need to satisfy the minimum delivery fee or spend much time splitting bills with friends who order with them. Our project aims to build a platform, on which users can create group orders, split bills, invite friends and even find neighbors they don't know to order together. They will be able to easily meet the order minimums and save money on delivery fees.

Functional Requirements

1. Account

As a user,

- a. I would like to register for an account.
- b. I would like to log in and manage my account.
- c. I would like to reset my password if I forgot it.
- d. I would like to save my address and payment information so that I don't have to type them in every time.

2. Choosing Food

As a user,

- a. I would like to have a general address based on my current geolocation so that I don't have to type in my address before I decide.
- b. I would like to see the available restaurants, together with the delivery fee, estimated delivery time, and other related information.
- c. I would like to search for specific restaurants by name.
- d. I would like to filter or sort the restaurant list by different properties such as distance, rating, and estimated delivery time.
- e. I would like to see the menus for each available restaurant.
- f. I would like to see different options (e.g., sauce choice, meat choice) on the menu.

3. Group Order

As a user,

- a. I would like to create group orders.
- b. I would like to see all nearby public orders that I can join.
- c. I would like to split the total cost so that I only pay the price and delivery fee proportional to the food chosen by me.

As an order creator,

- a. I would like to generate a link of the order.
- b. I would like to share the link to others using text messages, emails, or other third party applications.
- c. I would like to generate a QR code of the order so that it will be more convenient for face-to-face sharing.

- d. I would like to set the order to private so that only people who know the links and QR code can see and join it.
- e. I would like to set the order public so that can be seen and joined by all nearby users.
- f. I would like to set a time limitation in which other users can join the order.
- g. I would like to be the only user who confirms the order.
- h. I would like other users set themselves as ready before I could confirm the order so that we can make sure everyone has selected the items they want.

4. Friends

As a user,

- a. I would like to add friends.
- b. I would like to invite friends from the friend list to join the order.
- c. (If time allows) I would like to receive notifications when my order status updated.

5. Tracking

As a user,

- a. I would like to track my order status.
- b. I would like to track my driver's real-time location.

6. Order History

As a user,

- a. I would like to see my order history so I could choose the same restaurant again easily.
- b. I would like to rate the restaurants from which I've ordered.

7. Administration

As an administrator,

- a. I would like to login to a dashboard.
- b. I would like to manage (add/update/remove) restaurants.
- c. I would like to manage the restaurants' menus.

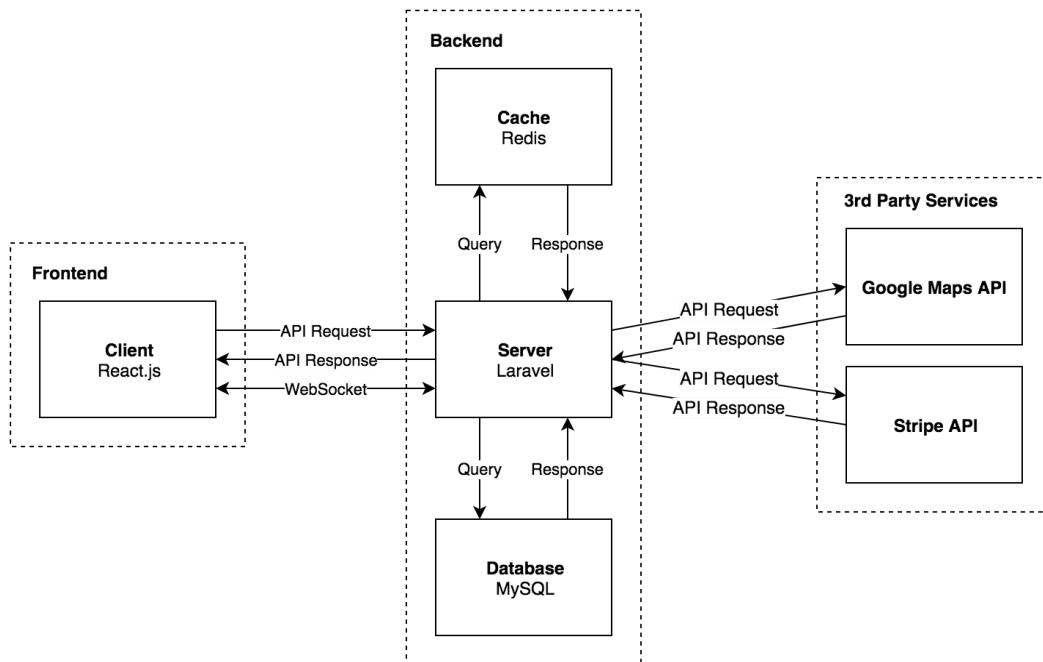
8. Restaurant

As a restaurant employee,

- a. (If time allows) I would like to see the detail of all active orders.
- b. (If time allows) I would like to update the status of orders.

Design Outline

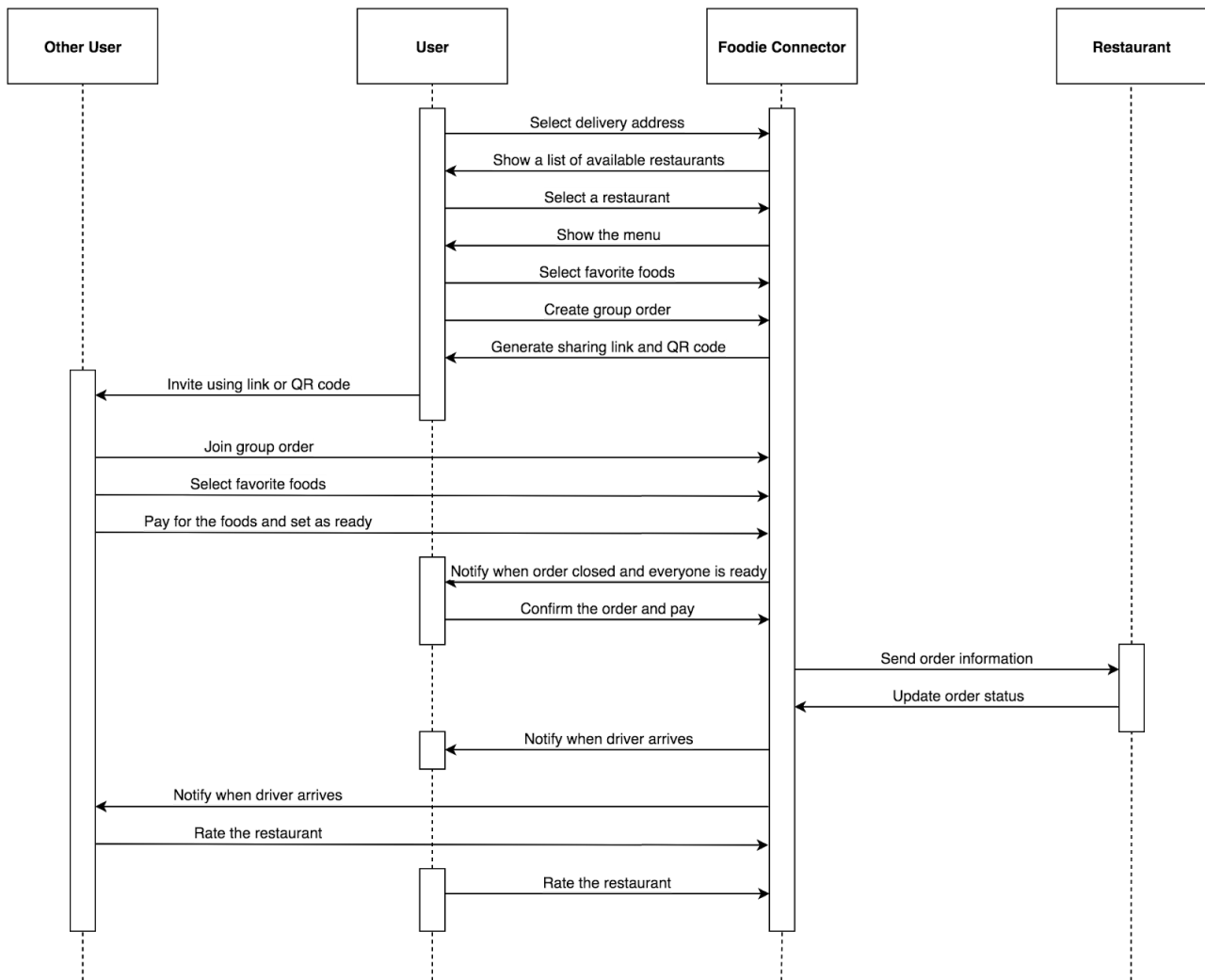
High-Level Structure



The communications between frontend and backend and between backend and 3rd party services are mainly based on API requests and responses. The WebSocket between the client and the server is used for updating drivers' real-time geolocation and pushing notifications.

- **Client**
 - The client is a React.js application. It handles the interactions with users and uses APIs to communicate with the server.
 - The UI is implemented in the client.
- **Server**
 - The server is a Laravel application. It accepts API requests from the client.
 - The backend logic is mainly implemented in the server.
 - The server handles the communication with the database, the cache, and 3rd party services.
- **Database**
 - The database stores all the persistent data.
- **Cache**
 - The cache stores all temporary data, such as sessions.
 - We use Redis as a cache.
- **Google Maps API**
 - We use Google Maps API to
 - Generate user's address based on device location;
 - Auto-complete in the address fields;
 - Determine if a user's address is in the delivery zones;
 - Calculating the distances of the restaurants.
- **Stripe API**
 - Stripe API will be used to handle all the transactions.
 - It supports major payment networks and other payment methods such as Apple Pay and Google Pay.

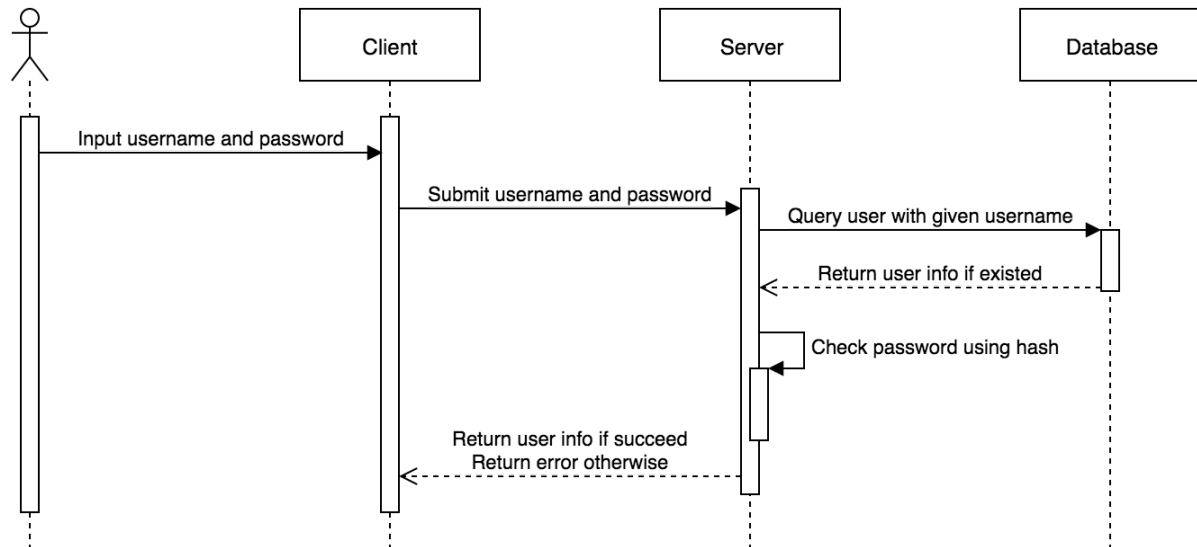
Typical Process of Ordering



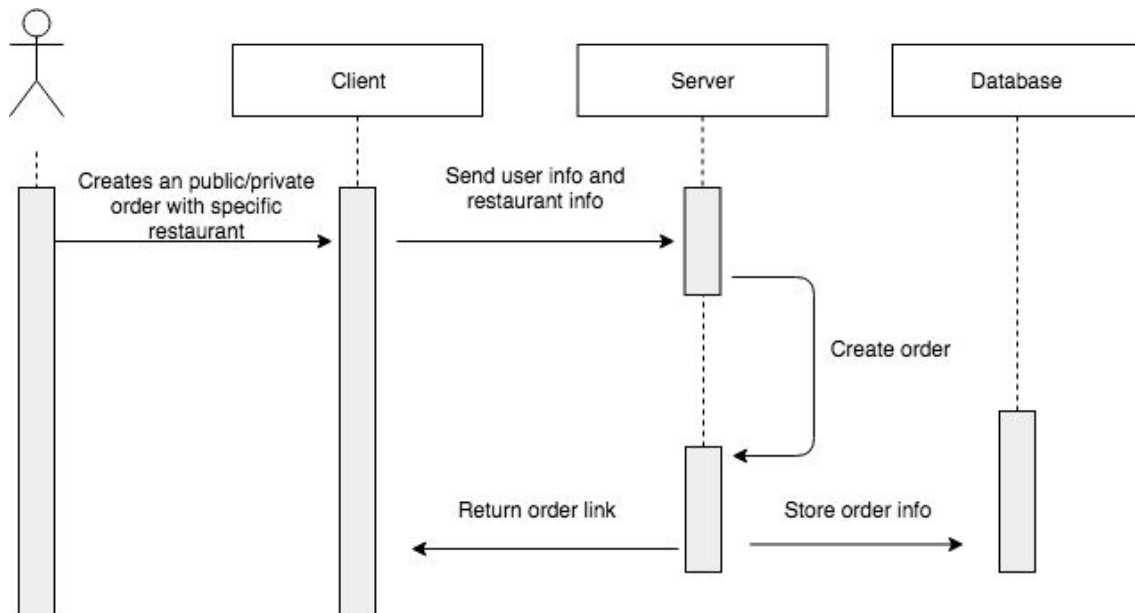
The diagram above shows a typical process of private group ordering. The details can be slightly different in public group orders, and some of the operations can be switched. For example, a user can create a group order before choosing his/her food. Some of the operations, such as register, login, and add card information, are not shown in the diagram.

Sequence Diagrams

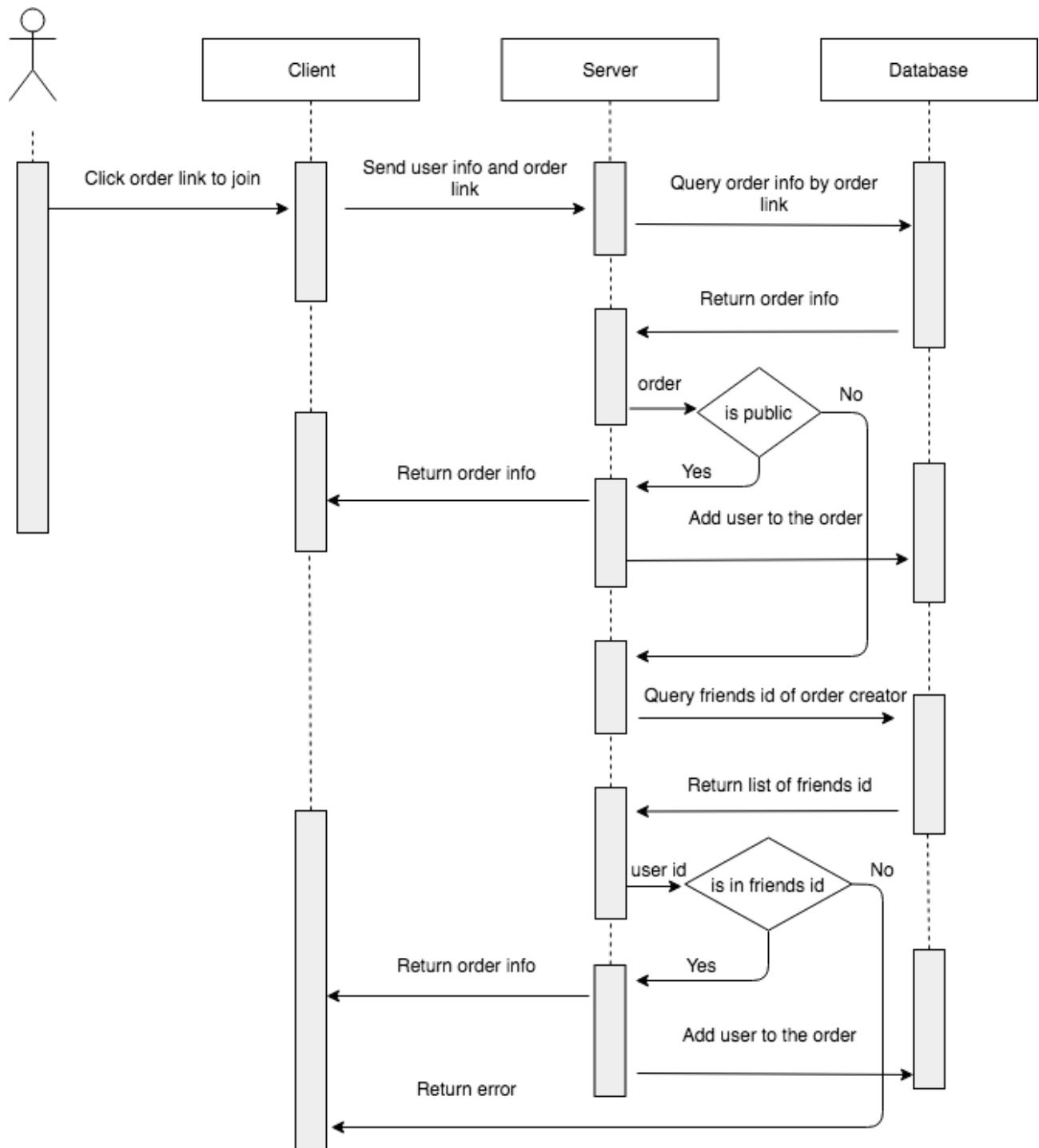
User Login



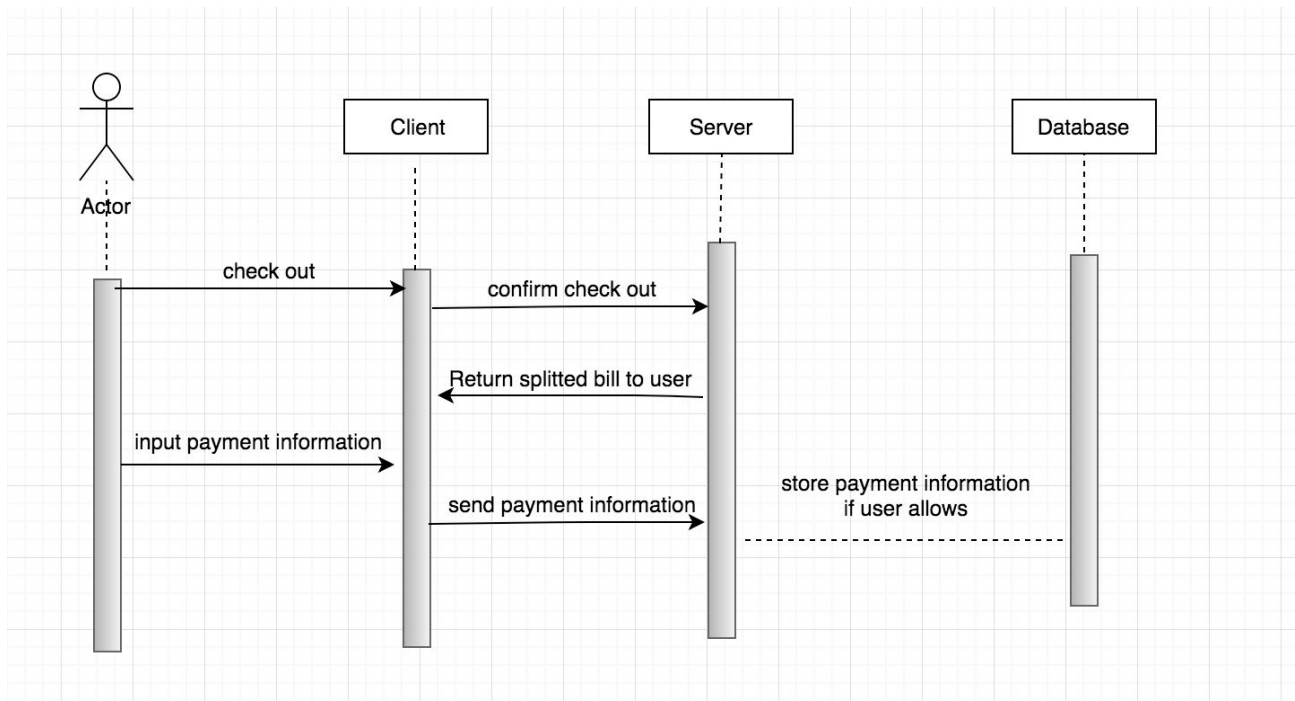
Create Order



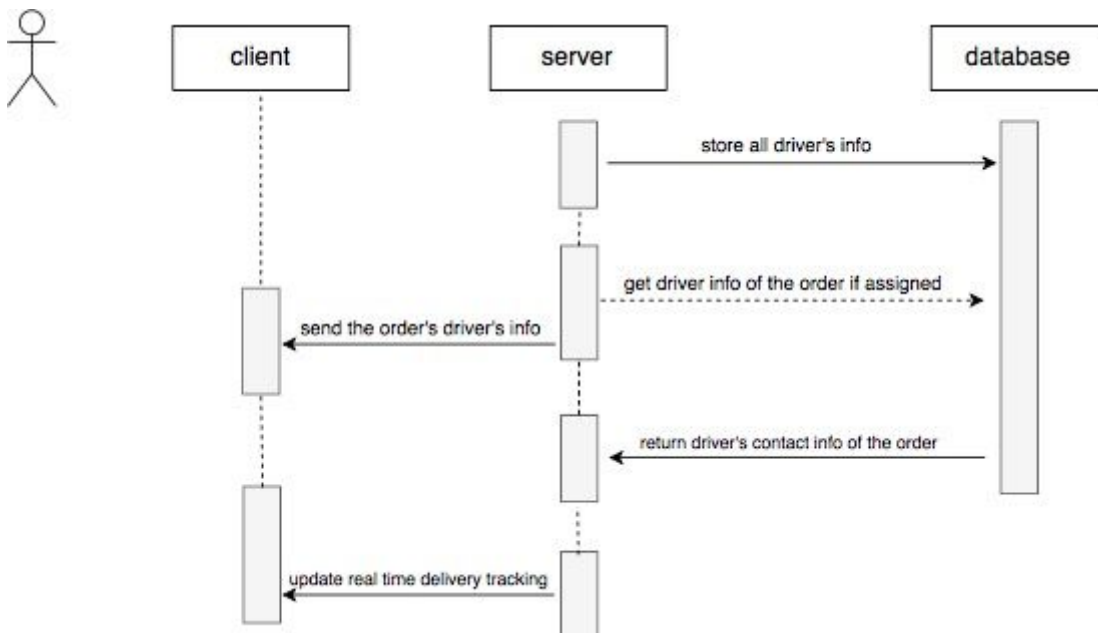
Join Order



Split Bills



Delivery Tracking



Design Issues

(Our final design decisions are bolded in the options lists as shown below)

Functional Issues

Do we need to provide an authentication system?

- **OPTION 1: Yes**
- OPTION 2: No

We require users to log in before ordering since we need authentication to protect their order histories, which can include sensitive information. Besides, their addresses and payment information will be stored so they only need to type them in once.

Does a group order have only the creator's address?

- **OPTION 1: A group order has only the creator's address**
- OPTION 2: Each member of a group order has their own address

We decided that each group order has only one address because calculating the actual walking or driving distance between different addresses can be tricky. In most cases, the driver only delivers the food to the front door of the building, so picking up at the same location won't be difficult as long as the group members live in the same or nearby buildings.

What kind of rating and feedback system do we provide?

- OPTION 1: Binary rating system (Thumbs up / Thumbs down)
- **OPTION 2: Binary rating system with optional feedback message**
- OPTION 3: Five-star rating system
- OPTION 4: Five-star rating system with optional feedback message

In five-star rating systems, people tend to give 1-star or 5-star ratings, so the results will be similar to that of binary rating systems. Besides, a binary rating system is easier to implement and easier for users to decide their ratings.

Optional feedback messages are important since it helps the restaurants know what they need to improve. In many cases, customers don't want to spend time calling the restaurants and complaining about their issues. They simply choose other restaurants the next time. The optional feedback message is easy to use and doesn't cost much time, so it can encourage users to provide feedback.

Non-Functional Issues

Which language should be used to implement the backend?

- **OPTION 1: PHP**
- OPTION 2: Node.js
- OPTION 3: Go
- OPTION 4: C#

There are plenty of languages can be used to implement a reliable backend. We are most familiar with PHP and Go. They both provide powerful built-in libraries for backend development and run very fast. We choose PHP because it has a longer history so there are more stable and mature frameworks we can use to speed up our development. We will use Laravel, one of the most popular PHP frameworks.

Which database should we use?

- **OPTION 1: MySQL**
- OPTION 2: PostgreSQL
- OPTION 3: Microsoft SQL
- OPTION 4: MongoDB

Since there are a lot of relations in our database schema, we will not use NoSQL, such as MongoDB. Besides, Microsoft SQL can introduce problems in deploying to a Linux environment, so we don't want to use that as well. MySQL and PostgreSQL are very similar. We chose MySQL only because we are more familiar with its management and related tools.

Which cache server should be used?

- **OPTION 1: Redis**
- OPTION 2: Memcached

Both Redis and Memcached are key-value databases that run on memory. The main reason we chose Redis is that Redis supports data structures like List and Set, which makes the development more convenient.

Do we need HTTPS?

- **OPTION 1: Yes**
- OPTION 2: No

HTTPS provides encryption in the communication between clients and the server. It helps prevent Man-in-the-middle attacks. Since our application is going to transfer users' addresses and credit card information, encryption is obviously needed. Besides, we can use Let's Encrypt to get a free SSL certification, so using HTTPS won't be hard.

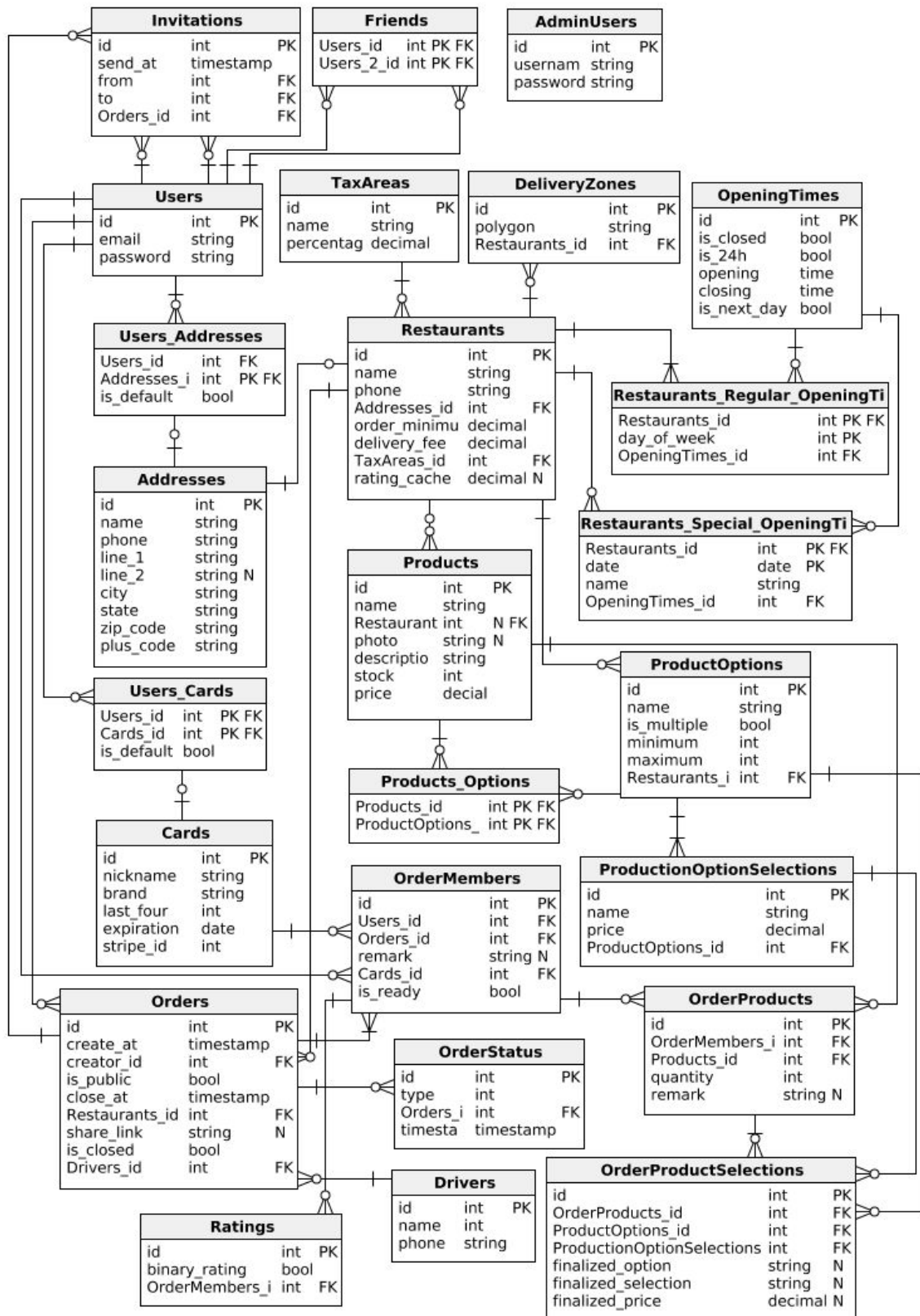
Which framework do we use to implement the frontend?

- **OPTION 1: React**
- OPTION 2: Angular
- OPTION 3: Vue.js

All the above frameworks are popular and can be used to implement user interfaces that have complex logics. We don't have much experience in using either of them building complex web applications. However, we do have some experience in using React, and it is easy to learn. Therefore, we decided to use React.

Design Details

Database Diagram



Descriptions of Database Tables

1. AdminUsers:
 - a. The table stores the username and password of admin users. Admin users are used to login to the dashboard.
2. Users:
 - a. The table contains the user information for login (email and password).
3. Friends:
 - a. If user1 and user2 are friends, there will be one record in this table.
 - b. We make sure that Users_id < Users_2_id
4. Addresses:
 - a. Each record can be either a delivery address of a user or the address of a restaurant.
 - b. The plus codes (<https://plus.codes/>) are introduced by Google. Using plus codes, we can easily describe the detailed address and calling Google Maps APIs.
5. Users_Addresses:
 - a. It is a table describes the addresses each user has.
 - b. One user can have multiple addresses, but at most one default address.
6. Cards:
 - a. The table stores the credit card information.
 - b. The strip_id is used to make payment through Stripe APIs.
7. Users_Cars:
 - a. it is a table describes the cards each user has.
 - b. One user can have multiple cards, but at most one default card.
8. Restaurants:
 - a. It contains basic information about the restaurants.
9. TaxAreas:
 - a. It stores the tax rate of each area.
10. DeliveryZones:
 - a. It uses polygons to describe the delivery zones of restaurants.
 - b. The addresses inside the geometric polygons can be delivered to.
11. OpeningTimes:
 - a. Each record contains the time of opening in one day.
12. Restaurants_Regular_OpeningTimes:
 - a. The table stores regular opening times of the restaurants.
 - b. Each restaurant has exactly 7 records in this table, representing the opening times of 7 days of a week.
13. Restaurants_Special_OpeningTimes:
 - a. The table stores special opening times of the restaurants. It is quite useful to describe the temporary change of opening times due to vacations or special events.
14. Products:
 - a. It describes the menus of the restaurants.
15. ProductOptions:
 - a. Product options can be any kind of extra options such as sauce choice and meat choice.
 - b. Product options can be radio or multiple selections.
16. ProductOptionSelections:
 - a. Each product option has several selections.
 - b. The selections can have extra prices.

17. Products_Options:

- a. The table describes which options a product has.
- b. One product can have multiple options. Multiple products can also have the same option.

18. Orders:

- a. The table stores the basic information of the orders.
- b. Orders can be set as public or private.

19. OrderStatus:

- a. Order status includes 'Payment Received', 'Cooking', 'Delivering', 'Delivered', etc.

20. Drivers:

- a. Only the driver's basic information.
- b. The real-time geolocation of the driver will be stored in Redis.

21. OrderMembers:

- a. The table stores each member's payment method, remark, and ready status.

22. OrderProducts:

- a. Each record describes a product selected by a user, along with its quantity and remark

23. OrderProductSelections:

- a. Products can have options, and this table is used to store the users' selections.
- b. It stores the finalized option name, selection name, and selection price after the order is placed. This ensures that the information in the order history is accurate even if the options or selections in the menu have been changed after ordering.

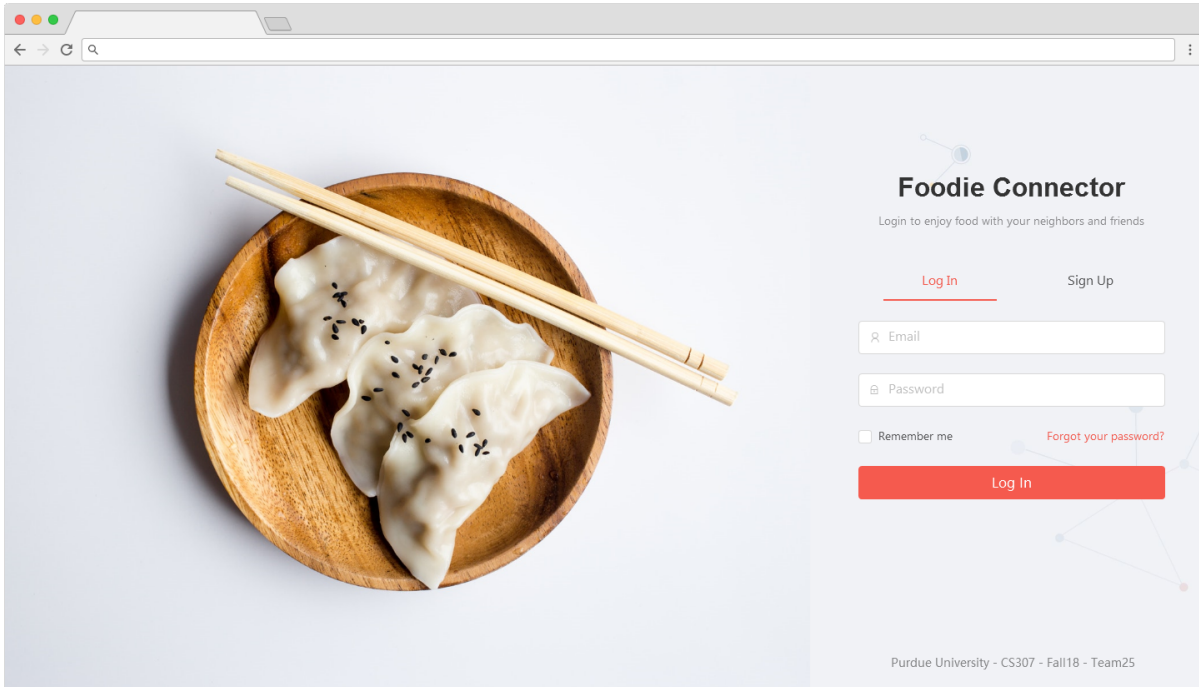
24. Ratings

- a. Each member of the order can rate the restaurant and provide feedback after the order is finished.
- b. The binary_rating stores the rating given by the user. True means thumbs up and false means thumbs down.

UI Mockup

Log In

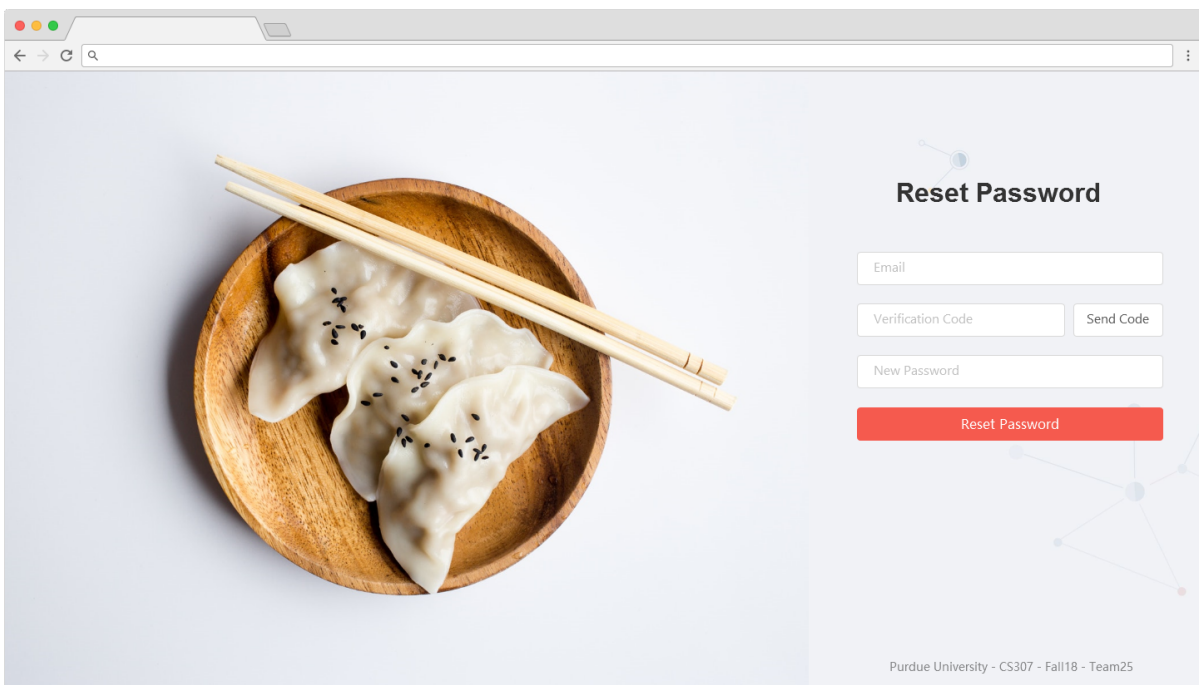
A user can enter username and password to log in or sign up. Also, we designed the option for user who forgets password.



The mockup shows a web browser window with a light gray background. On the left, there is a large image of a wooden bowl containing dumplings and a pair of chopsticks. On the right, the 'Foodie Connector' login form is displayed. The form has a title 'Foodie Connector' and a subtitle 'Login to enjoy food with your neighbors and friends'. Below the subtitle, there are two tabs: 'Log In' (active) and 'Sign Up'. The 'Log In' tab contains an 'Email' input field, a 'Password' input field, a 'Remember me' checkbox, and a 'Forgot your password?' link. A red 'Log In' button is at the bottom of the form. The footer of the page reads 'Purdue University - CS307 - Fall18 - Team25'.

Reset Password

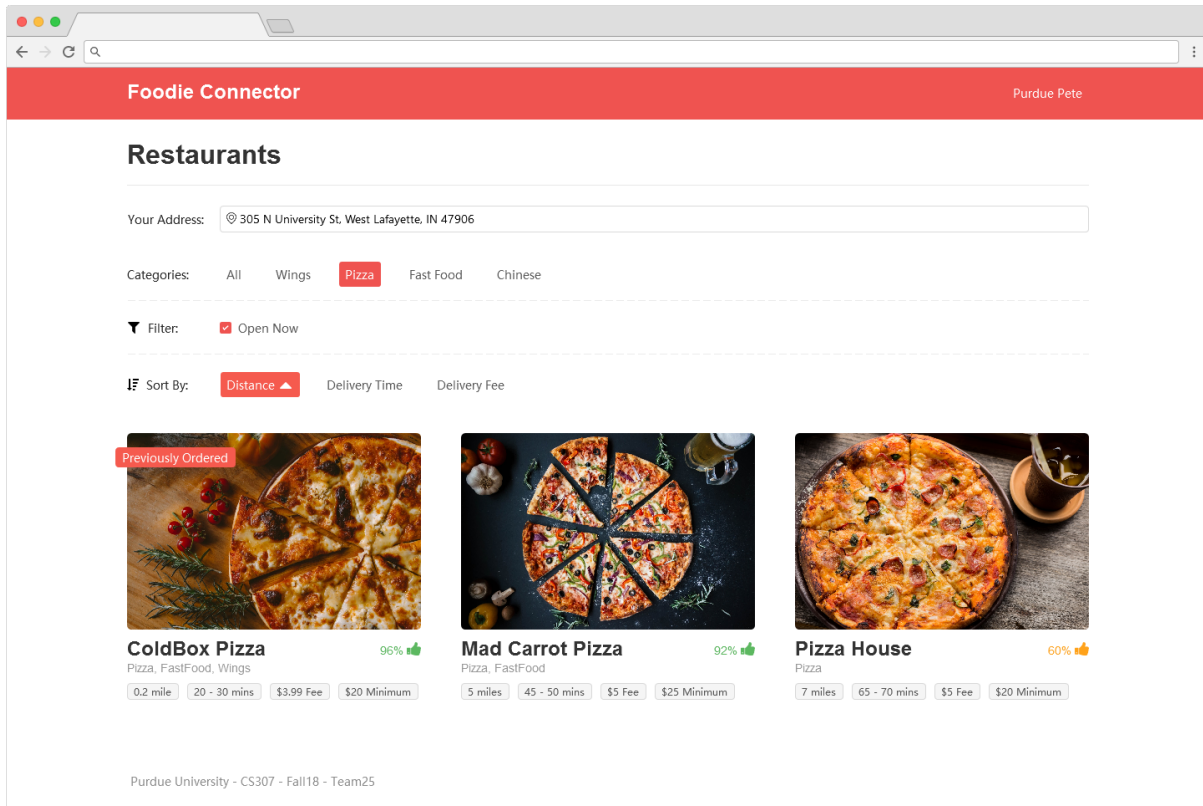
If the user forgot the password, he/she can input their email address. We will send an email containing a verification code that can be used to reset the passwords.



The mockup shows a web browser window with a light gray background. On the left, there is a large image of a wooden bowl containing dumplings and a pair of chopsticks. On the right, the 'Reset Password' form is displayed. The form has a title 'Reset Password'. Below the title, there are three input fields: 'Email', 'Verification Code', and 'New Password'. A 'Send Code' button is located next to the 'Verification Code' field. A red 'Reset Password' button is at the bottom of the form. The footer of the page reads 'Purdue University - CS307 - Fall18 - Team25'.

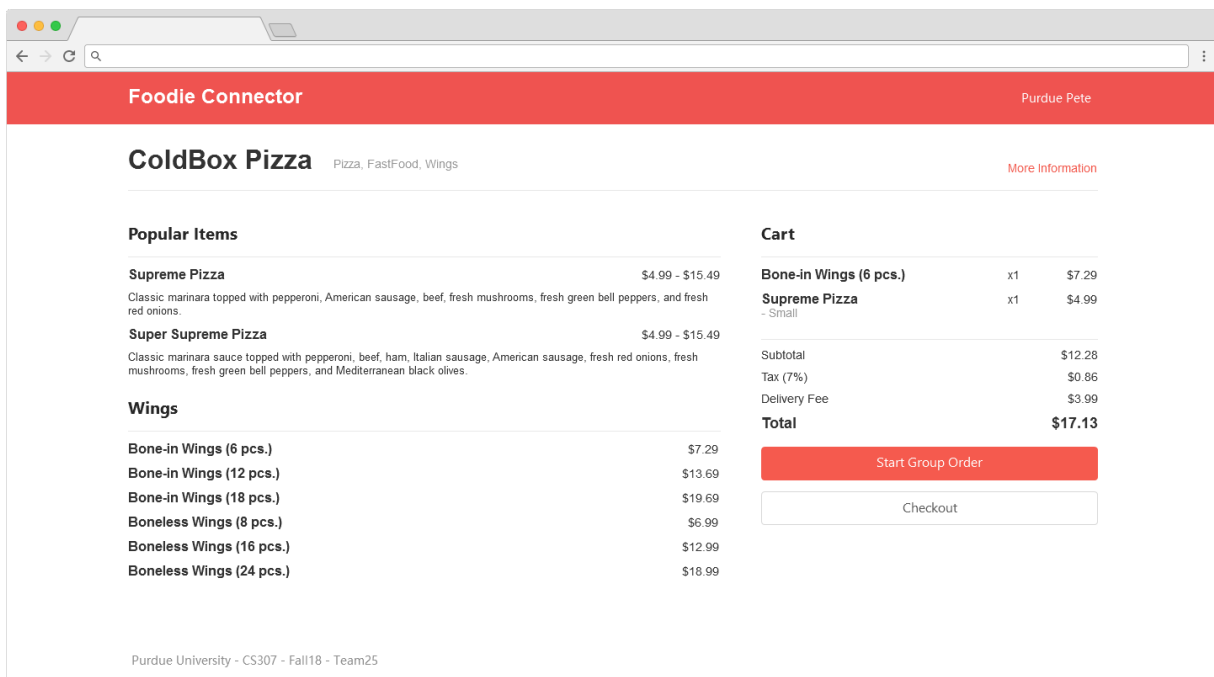
Restaurant List

After the address is detected or entered by the user, all available restaurants will be shown in the list. Users can also filter or sort the list. Besides, previously ordered restaurants will be tagged, so that users can easily find their favorites.



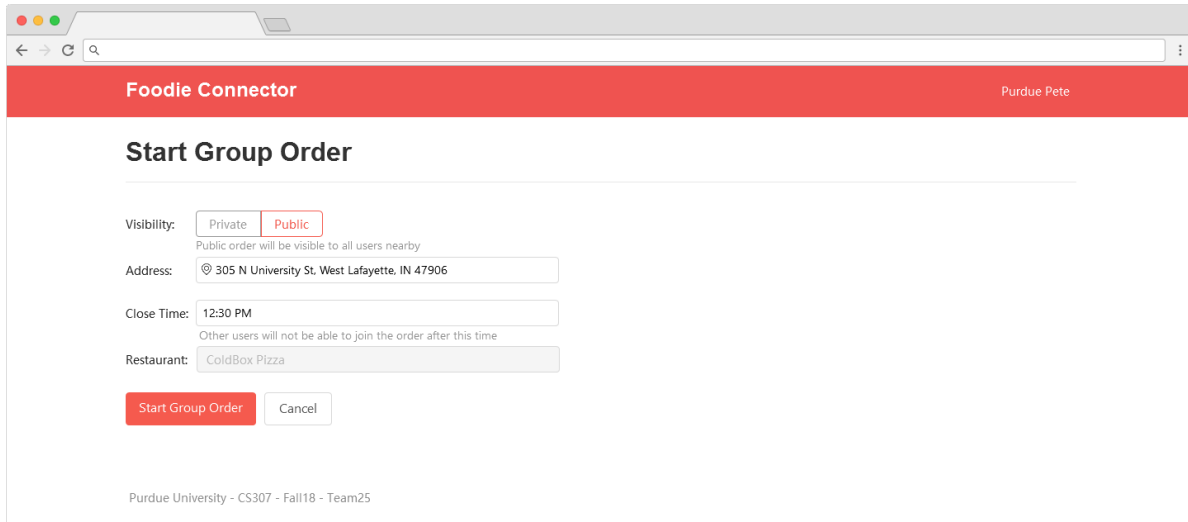
Menu

This is the menu page. Users can choose items they like. After that, they will be able to start a group order or checkout directly.



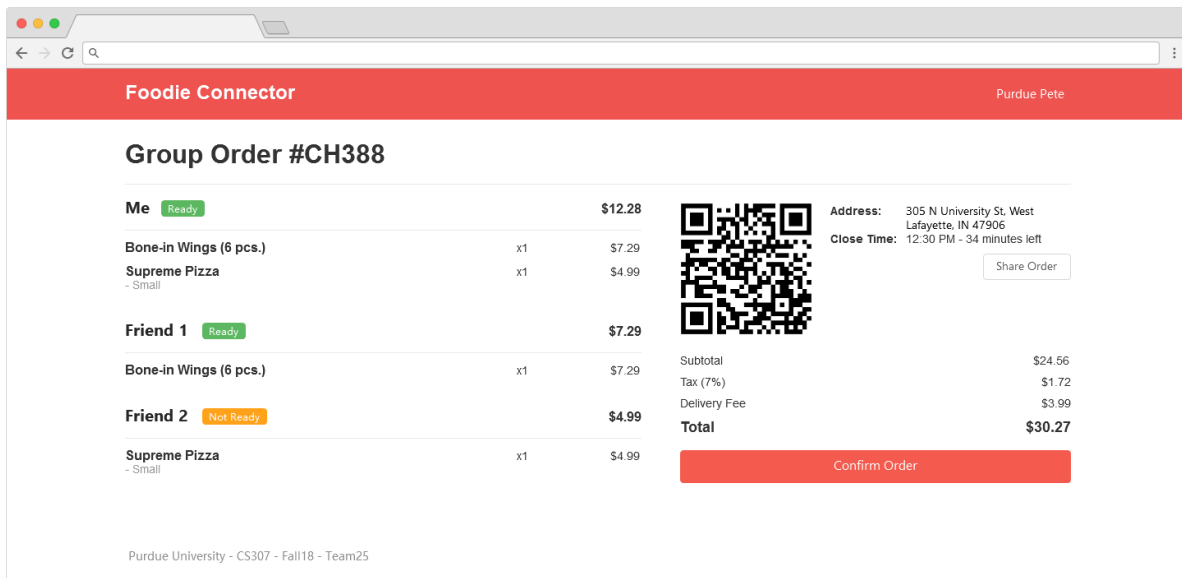
Group Order

To start a group order, user can select its visibility, address, and close time.



The screenshot shows a web browser window with the 'Foodie Connector' header. The user 'Purdue Pete' is logged in. The main heading is 'Start Group Order'. Below it, there are several input fields: 'Visibility' with 'Private' and 'Public' (selected) buttons; 'Address' with a location pin icon and the text '305 N University St. West Lafayette, IN 47906'; 'Close Time' with a time picker set to '12:30 PM' and a note 'Other users will not be able to join the order after this time'; and 'Restaurant' with a dropdown menu showing 'ColdBox Pizza'. At the bottom, there are two buttons: 'Start Group Order' (red) and 'Cancel' (white). The footer text reads 'Purdue University - CS307 - Fall18 - Team25'.

In the group order detail page, all the users in the order, together with their selections, will be shown. There will be a QR code and a share button, which can be used to invite friends to join the group order.



The screenshot shows the 'Group Order #CH388' detail page. It lists the order items and their status. The 'Me' section shows 'Ready' status and a total of \$12.28 for 'Bone-in Wings (6 pcs.)' (x1, \$7.29) and 'Supreme Pizza - Small' (x1, \$4.99). The 'Friend 1' section shows 'Ready' status and a total of \$7.29 for 'Bone-in Wings (6 pcs.)' (x1, \$7.29). The 'Friend 2' section shows 'Not Ready' status and a total of \$4.99 for 'Supreme Pizza - Small' (x1, \$4.99). A QR code is displayed in the center. To the right of the QR code, the address '305 N University St. West Lafayette, IN 47906' and the close time '12:30 PM - 34 minutes left' are shown, along with a 'Share Order' button. At the bottom right, a summary table shows: Subtotal (\$24.56), Tax (7%) (\$1.72), Delivery Fee (\$3.99), and Total (\$30.27). A red 'Confirm Order' button is at the bottom. The footer text reads 'Purdue University - CS307 - Fall18 - Team25'.

User	Status	Item	Quantity	Price
Me	Ready	Bone-in Wings (6 pcs.)	x1	\$7.29
		Supreme Pizza - Small	x1	\$4.99
Friend 1	Ready	Bone-in Wings (6 pcs.)	x1	\$7.29
		Supreme Pizza - Small	x1	\$4.99
Friend 2	Not Ready	Bone-in Wings (6 pcs.)	x1	\$7.29
		Supreme Pizza - Small	x1	\$4.99

Public Group Order List

Users can see nearby public group orders and select one to join.

Foodie Connector

Purdue Pete

Nearby Public Group Orders

Your Address:

305 N University St, West Lafayette, IN 47906

#CH388 ColdBox Pizza	Distance 0 mile	Creator Purdue Pete	Close Time 12:30 PM - 34 mins left	Join
#XYG93 ColdBox Pizza	Distance 0 mile	Creator Purdue Pete	Close Time 12:30 PM - 34 mins left	Join
#GSKWI ColdBox Pizza	Distance 0 mile	Creator Purdue Pete	Close Time 12:30 PM - 34 mins left	Join
#22WDS ColdBox Pizza	Distance 0 mile	Creator Purdue Pete	Close Time 12:30 PM - 34 mins left	Join
#OKJW0 ColdBox Pizza	Distance 0 mile	Creator Purdue Pete	Close Time 12:30 PM - 34 mins left	Join

<

1

2

3

4

5

6

7

8

9

>

Purdue University - CS307 - Fall18 - Team25

Tracking Order

Users can see their drivers' real-time geolocation.

Foodie Connector

Purdue Pete

Tracking Order #CH388

Deliverying

Driver:

Purdue Pete

Arrival Time:

In 15 mins

Purdue University - CS307 - Fall18 - Team25