Bugs

Front End

Known bugs:

- Dark mode barely works
- Camera crashes on Android, but never on iOS
- All of the posts show the email of the vendor in bold, with the subtext being the real name
- Location in post not functional, does not load sometimes
- Orders page sometimes does not show the accepted food cards
- Back buttons, such as the one in Favorites, seems to require multiple taps
- Sometimes logins take a while
- Food card uploads can not be seen immediately, loading takes a while
- Phone number displayed on the expanded food card accesses test data, instead of accessing true data
- Tags, and name can sometimes not be edited
- Need to remove entering business hours in create account.

Back End

Bugs:

[Body Data] Postman and some other services will wait for a server to respond and then send data which triggers an event in the request called `data`. However the fetch requests we are sending to the server pack their body data immediately which results in the `data` event never firing. Outside of the routes on `images` all requests need to have an immediate body otherwise they will be rejected for `Bad body, or no keys`

[Error Handling] There are many errors we are able to catch. Any time an error is thrown by a function in a breakpoint we know about one of 3 things that happens. A) the breakpoint returns the thrown error from whatever library threw the error, B) the breakpoint returns a customized error with some message pertaining to why the routine failed, C) The breakpoint specifies a known error that is documented in the `serverutility.js` file. The effects of this are that when things go wrong inside the backend the responsibility is placed onto the frontend to handle the error response. The reason I list this as a bug is because the front end will receive error information passed all the way from the backend and it will say things like “incorrect SQL syntax near line blahblahblah of sqlhandler.js”., however it really indicates that some data format was wrong on upload (like a type check)

[Card Upload] : There is no type checking for card names,tags, or anything included in the data field of a card object on upload. Rather more specifically anything that is included in the food_card.data field is stringified by the stringification of the food_card.data field. This means

that when uploading you could set the item name to be any data type and it will just be rendered as a string. So arrays, integers, objects, etc. However with the route /vendor/updateData you can pass a fixed name if you did misspell something. That being said the updateData route is also affected by the bug and you can put arrays, or whatever you want here for the name as well.

[Reservations] : Card reservation will return “Card already reserved” if a card is not found within the database. This is because the query to find the card you are reserving can return no results but the output is treated the same because there is a WHERE clause that searches for a NON-NULL reservation field.

[Cancelation] : Canceling cards is handled by a sweeping update to any card that has the reservation field set to the user's email. So if no cards are reserved by the user a successful response is returned and it will say that cards were canceled (Not entirely false because the user has no reservations anymore).

[Delete]: deleting an item that does not exist returns success 1

Suggested User Stories to Test Functionality of System:

User story: I, a user, should be able to create an account with vendor status that saves my information for a future visit, so I can reuse my account in case I need to do this again.
To prove, perform Scenario 1 on Unit Tests for Front-End.

Backend:

Using a fetch request or the provided `fetcher.html` in `/src/testing`.

Make a post request to the route `/signup` sending a body with the following ***required keys**: **[email:string, pass:string, vendor:int]***.

Regardless of error or not the server will send a status code of 200 back for the request. If you do not get 200 there is most likely a connection issue.

User story: I, a user, should be able to create an account with seeker status that saves my information for a future visit, so I can reuse my account in case I need to do this again.
To prove, perform Scenario 2 on Unit Tests for Front-End.

User Story: I, as a vendor, want to see the posts that I have made, to make sure I’m not posting the wrong ones.
To prove, perform Scenario 3 on Unit Tests for Front-End.

User Story: I, as a vendor, after posting, would like to edit the tags, name, and time available of my post so I can fix mistakes I make while working.
To prove, perform Scenario 4 on Unit Tests for Front-End.