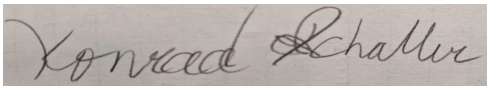
Nibbler

by

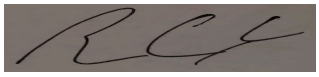
Vincent Ehemann, Konrad Schaller, and Ryan Young

Submitted to
the Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

_____ 04/18/2020
Vincent Ehemann Date



_____ 04/18/2020
Konrad Schaller Date



_____ 04/18/2020
Ryan Young Date

_____ _____
Abdou Fall, Faculty Advisor Date

University of Cincinnati
College of
Education, Criminal Justice, and Human Services

April 2019

Prepared by
Vincent Ehemann, Konrad Schaller, and Ryan Young

Students of
University of Cincinnati
College of Education, Criminal Justice, and Human Services
School of Information Technology

April 13, 2020

Table of Contents

Table List

Figure List

Abstract

Nibbler is an android application used to search and find restaurants based on specific filters applied by the user. Nibbler gives users the ability to apply multiple filters to their searches and allows those with dietary restrictions or food allergies to find restaurants that fit their specific diet. Nibbler also aims to eliminate the issue of analysis paralysis by limiting the return on food locations that are returned to the user after completing a search. The reason for this is simple, if there are too many options, then an individual can overanalyze or overthink the situation. This way the user still has everything they originally searched for, but the decision is easier due to the fact that they are limited in their options. The application is simple to use and anyone with an android device can navigate the simple UI. This allows users to store personal data and health information safely and securely.

Introduction to Problem

When you are with others or just by yourself, finding a restaurant can be difficult. Many websites or applications, such as Google or Yelp present their users with a massive list of restaurant locations that fit their search criteria. When you are presented with too many choices it becomes cumbersome to actually decide where and what you want to eat. This phenomenon is known as Analysis Paralysis and many individuals experience this every day.

In the United States, it is estimated that 32 million Americans have a food allergy according to the Center for Disease Control & Prevention and the Department of Food Allergy Research & Education. That is about 10% of the population of the US. Each year, there are over 200,000 US citizens hospitalized due to these food allergies. The issue of food allergies in the United States has steadily been growing throughout the years.

Problem Statement:

According to Dr. Barry Schwartz's book *The Paradox of Choice*, "researchers set up two displays of jams at a gourmet food store for customers to try samples. In one display there were six jams, in the other 24: 30% of people exposed to the smaller selection bought a jam, but only 3% of those exposed to the larger selection did. (Schwartz, 2005)" You may want a certain type of food or maybe you are unsure you can actually eat the food because of an allergy you have. Eventually you may find a restaurant only to realize you cannot eat there because they use a product that you are allergic to.

Solution Statement:

Nibbler is an application that gives users a list of restaurant options based on preset options and filters that the user applies. The filters cover everything from allergies/diet restrictions to the type of cuisine. Nibbler takes the restaurants and menus from the users' area and gives the user an easy to read list of locations and food types. The users can also apply filters such as lactose intolerance or the quickest restaurant nearby. Other applications such as Yelp or DoorDash limit the user to limited filters (in some cases only one). This not only is inconvenient and adds time to your search, but it also causes analysis paralysis to set in. Nibbler puts the user in the position to get food quickly, satisfy all their hunger needs, and allows them to eat with peace of mind knowing that they are not eating foods that include items that they cannot consume due to dietary restrictions or allergies. The Nibbler search algorithms will tailor user suggestions to their tastes and style. If they want something new our machine learning algorithm will analyze other users reviews to decide the next best restaurant.

Project Goals:

The goal of Nibbler was to build an app that allows users to find restaurants near them that pertain to their dietary needs. The number of individuals who are affected by the lack of applications regarding this goal is high. We want to give these individuals the ability to find these restaurants like everyone else in a simple and efficient way.

Project Overview:

The remainder of this report highlights the development, design methodologies, and

various details about the project. Some of these details include scheduling and deadlines, the budget for the project, testing done, and technical details that outline each component and how it works with the Nibbler system as a whole.

Project Concept

Nibbler is an android application that gives users the ability to research and find potential restaurant options around their area. Nibbler will allow the user to be in control by giving them several filter options, as well as giving them the option to stack filters on top of each other. This will ensure that users can find exactly what they are craving and will limit the factor of analysis paralysis. Nibbler also is an attempt to promote awareness of food allergies and other eating restrictions. We want everyone to be able to enjoy their food knowing that it will not encroach on their dietary restrictions or allergies.

Design Objectives:

Our goals for this application are to offer an easy to use and quick piece of technology that gives users peace of mind when eating out. Nibbler gives users with certain dietary restrictions the ability to research an establishment before actually getting there. Current applications can give users some false information. A big goal of Nibbler is to present accurate information, as well as update past information to ensure that our users know they can consume at a restaurant. Unfortunately, all the goals we have made did not make it into the final product. Originally, we wanted Nibbler to be HIPPA certified, but due to the application auditing fees and the slow certification process, we were unable to meet this goal. We also

wanted the application to have a feature on the backend that would allow it to gather data around the web pertaining to restaurant menus. Unfortunately, the data this feature would retrieve could have been incorrect and would have hindered our previous goal pertaining to accurate food information. The information collected would need to be verified before being put into the database, which would take much more time than just simply entering the information ourselves.

Technological Approach:

Once we understood what was achievable and what was not, we elected to design the application based around the goals above. The idea for the android application came about from goals to create a fast and reliable tool to find food. Some individuals had issues finding food that was safe or did not encroach on their dietary decisions. The backend of the application was the starting point of this process. Once created, a back office used for entering data, viewing databases, and updating restaurant information was then created. This allowed us to add items such as restaurant, dietary restriction filters, and user information. From there, development on the actual frontend of the application began. A basic flow design was created using a UI design tool. The views came from this flow design and were built using various android components.

User Profile:

Figure 1: User Profile, displays the user profile for our Nibbler application.

The figure lists users who would use the application, whether that be restaurant

owners, actual customers, or other users. The profile also depicts tasks that the users will complete when operating Nibbler.

PROJECT:

Nibbler

POTENTIAL USERS:

- Admins
- Business Associates
- Restaurant Owners
- Guests
- Adventurous hungry individuals seeking new places to eat
- Indecisive individuals seeking a place to eat
- Indecisive individuals with food allergies seeking a place to eat
- An indecisive group collectively finding a place to eat

SOFTWARE, INTERFACE, AND RELATED EXPERIENCE:

The project's core purpose is to cater towards the individuals or groups seeking a place to eat. The users can create a profile with the dietary restrictions and allergies. This profile will be submitted with every food suggestion request. When the user is ready to request a restaurant suggestion, the user must fill out a filter form which contains categories of food to choose from. Since this form contains preference and likely will change, the user will submit a new form every time a request has been made.

Restaurant owner users will have a separate interface to interact with that is independent to the one the core users will. In this interface, the restaurant owner can modify attributes of their restaurants they own. They can change the menu, description, and picture of the restaurant. The purpose of the restaurant owner role is to minimize the overhead for restaurant owners to modify attributes of their restaurants.

Admins and Business Associates will interact with a similar interface as to the restaurant owners. The difference is that there will be more access to other data. Some data can be core user information and restaurant information. They will have the freedom to change all of it to resolve any issues that may arise with other users.

EXPERIENCE WITH SIMILAR APPLICATIONS:

The core user application interface will operate similarly to most mobile applications. It will use many similar constructs found on all mobile apps. Check lists, side bars, option menus, back buttons, and more will be used in the interface. It will be simplistic and thus easy to use.

The back-office interface will be a web application using 3rd party libraries that have common designs to other web apps that exist. The application will consist of mostly forms and data so it may require some time to become acclimated with the web application. Nonetheless, the application will use common interface designs so the user will have no doubts on the functionality of the application interface.
TASK EXPERIENCE: Admins will need to maintain the infrastructure and hosting software which the application back-end serves. Code changes need to be coordinated between admins to ensure consistency.
FREQUENCY OF USE: The core users are expected to use the application the most since most of the application work will have been previously done. The nature of the data is that it won't be changed much in its lifetime.
KEY PROJECT DESIGN REQUIREMENTS THAT THE PROFILE SUGGESTS: ● Simple and easy to navigate UI for mobile application ● Reduce complexity of back-office UI to make it streamline ● Use common UI constructs to conform to User Model of the interface

Figure I: User Profile

Use Case Diagram:

Figure 2: Use Case Diagram, illustrates the use case diagrams for the Nibbler app. It depicts all the uses who would be associated with the application as well as the corresponding tasks that they would complete while using the app.

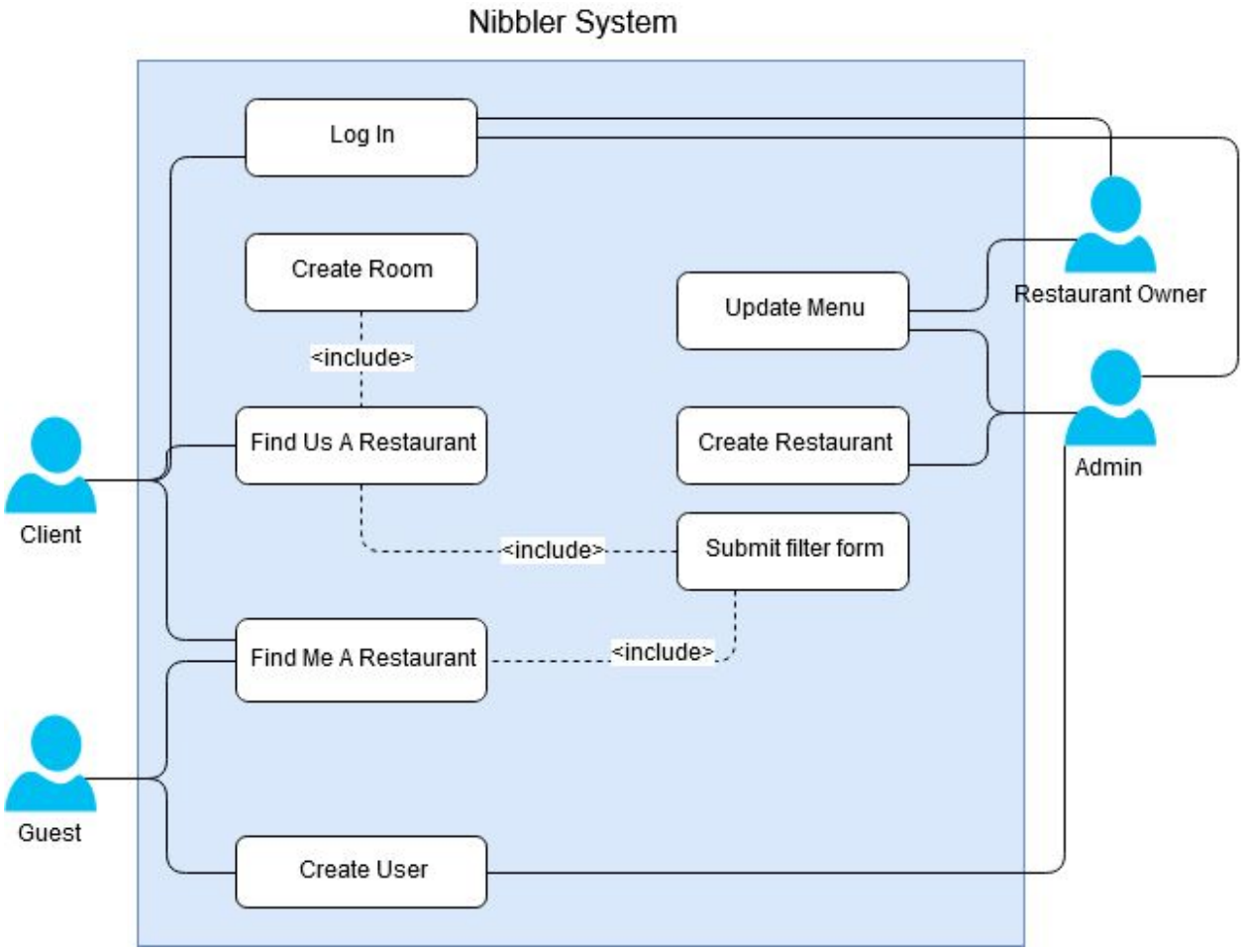


Figure II: Use Case

Use Case: Find me a restaurant

Scope: Nibbler Application
Level: User goal
Primary Actor: End user searching for restaurant
Stakeholders and Interests:

- User - Wants to find a restaurant that suits their dietary needs and cravings at the time of searching with a simple, intuitive UI to prevent frustration and quickly resolve the user's query
- Restaurants - Has an interest in increasing their customer base and an application that would potentially show their restaurant to new people looking to try a new restaurant is a great way to do this.

Preconditions: User wants to find a place to eat because they are hungry, but cannot decide where to eat.
Success Guarantee: User is presented with a short list of restaurants to choose from, one being the best match based on user dietary profile and applied filters which shows basic

information about the restaurant. Users can further select restaurants in the list to view more details; reviews, rating, open in google.

Main Success Scenario:

1. User opens application to main screen
2. User selects Find Me a Restaurant option
3. User applies filters to search
4. System searches for nearby restaurants using filters and dietary profile
5. System returns short list of restaurants
6. User selects restaurant to view details

Extensions:

3a. User does not have an account
    1. System checks if user has account
        a. When system returns there is no account, an option appears above filters to add dietary restrictions
    2. User selects dietary restrictions option
        a. Login screen appears
        b. User selects option to sign up
            i. Application authenticates the user's information
            ii. Screen redirected back to apply filters

5a. User wants to change search filters
    1. User selects back option
        a. User adds or removes filters
    2. User selects submit option

Special Requirements:

- Must have login for user dietary profile to be used
- Language internationalization for displayed text
- Must have location services enabled

Frequency of Occurrence: This will most likely be the largest percentage of user interaction

Open Issues:

- Will we inform users of requiring a login to utilize a dietary profile?
    - If so, when?
- Will we opt for displaying all restaurants in the short list the same way?
    - Each restaurant has basic details
    - Top of short list is best match

Use Case: Find us a restaurant

Scope: Nibbler Application
Level: User goal
Primary Actor: Group of end users searching for restaurant
Stakeholders and Interests:

- User - Wants to find a restaurant that suits their cravings at the time of searching with a simple, intuitive UI to prevent frustration and quickly resolve the users' query
- Restaurants - Has an interest in increasing their customer base and an application that would potentially show their restaurant to new people looking to try a new restaurant is a great way to do this.

Preconditions: Users want to find a place to eat because they are hungry, but cannot decide where to eat.

Success Guarantee: Users presented with a short list of restaurants to choose from, one being the best match based on applied filters which shows basic information about the restaurant. Users can further select restaurants in the list to view more details; reviews, rating, open in google.

Main Success Scenario:

1. Users open app to main screen
2. Users select Find Us a Restaurant option
3. One user selects Create Room option
4. Filter options appear
5. Other users join room based on RoomID
6. Other users enter filter options
7. Other users select "Done"
8. System tracks who is finished
9. User that created room closes room when all are finished
10. System returns short list of restaurants
11. Users select restaurants to view details

Extensions:

3a. Two or more users select Create Room option
    1. Multiple users select the Create Room option
    2. All but one user selects the back option to select Join Room

7a. User wants to change search filters
    1. User selects back option
        a. User adds or removes filters
    2. User selects done option

Special Requirements:

- Users must have a login for a dietary profile to be used
- Language internationalization for displayed text
- Users must have location services enabled

Frequency of Occurrence: This will be a much less used portion of the application

Open Issues:

- Will user dietary profiles be applied?
    - If so, who's?
    - Will we ask for a login if the user is not?
- Will each user in the room receive the short list of restaurants?
    - Will we consider having only the room creator see the list

Budget:

Table 1: The project budget can be seen in Table 1. Each item depicted in the budget reflects a current cost and a future cost. The current cost is obviously the current price we are paying for the item. The future cost is what we would be paying if we were to expand on the project after graduation and took the idea to market. These future costs could include “pro” versions of software or enterprise versions. We also have a labor section depicted in Table 1. The labor section depicts wages of the different team members if the project was taken to market. The labor costs are not factored into the total cost of the overall project.

Item	Description	Current Cost	Future Cost
Hardware	This includes items such as servers and devices to display the application on. Currently we are just using our personal devices and a server that was purchased by a team member.	$300	TBD
Software	Currently all the software being used is running on the free version. The only guarenteed future cost would come in the form of Amazon Web Services. The cost would be a recurring $50 every month.	$0.00	$50.00 (Monthly)
Communication	Currently the group is using GroupMe to communicate. In the future we would most likely switch to a more organized platform, such as Stack or Discord. This would allow us to have organized discussions in specific chat rooms. Both of the products are free so there would be no additional cost. WebEx is currently being used for long distance meetups. Currently this is free for us as we are UC students, but in the future an additional cost would be applied. We would most likely switch to a free platform, such as Skype, Google Hangouts, or Discord.	$0.00	TBD (Depending on desired video meeting platform.)
Wages	Currently, wages are not an issue due to the fact that this is a Senior Design project. In the future, wages would become a major cost. Three members making a salary of $66,000 each. The total cost would come out to around $198,000 per year. This is subject to change.	$0.00	$198,000*

Table I: Budget

**Totals are subject to change due to unseen expenses and wage discussions.*

Lean Startup Business Plan:

A lean startup plan is a type of business plan that roughly plans out and guides a small startup business during their startup process. It gives insight into the goals of the business, how the business plans to handle different challenges, and allows them to track and manage progress along the way. This is a rough lean startup plan for the Nibbler application.

Nibbler Lean Startup	
Identity Nibbler is a restaurant locator application that locates restaurants in your area based on specific search criteria and filters you apply.	**Problem** Users are looking for restaurants that have specific dietary restrictions or allergy free food that they can consume safely and without worry of having a reaction.
Our Solution Our application gives users the ability to find these restaurants by allowing them to tailor their search with filters and specific criteria, making their experience easy and stress free.	**Target Market** The target audience is anyone looking for restaurants or who have allergies or dietary restrictions. The age range is not limited because Nibbler is for everyone.
The Competition Nibbler is in a market that includes several other food applications. Apps such as DoorDash, Yelp, and even Google would be	**Revenue Streams** Nibbler has several ways to make money. Ads for partner restaurants would be the biggest revenue stream.

competing with Nibbler in their respected Market.	
Marketing Activities Nibbler would communicate with customers with an email newsletter (similar to DoorDash), targeted Google, and social media posts and pages.	**Expenses (Outlined in Budget)** ● Servers to store databases ● Possible advertising costs ● Amazon Web Services monthly costs ● Developer wages
Team and Key Roles Currently, Vincent Ehemann, Ryan Young, and Konrad Schaller are the only members, but Nibbler will look to add more once we begin to grow and make a name for ourselves.	**Milestones** Once the application is fully completed, Nibbler can look into sponsoring events or donating to bring attention to our app. Events would include ones such as Taste of Cincinnati.

Table II: Nibbler Lean Startup

**Expenses are outlined more specifically in Table 1 (Budget).*

**Different aspects of the plan are subject to change during the development process.*

Project Objectives

The objective of this project is to develop an android application that gives users the ability to research and find potential restaurant options around their area. Nibbler will allow the user to be in control by giving them several filter options, as well as giving them the option to stack filters on top of each other. This will ensure that users can find exactly what they are craving and will limit the factor of analysis paralysis. Nibbler also is an attempt to promote awareness of food allergies and other eating restrictions. We want everyone to be able to enjoy their food knowing that it will not encroach on their dietary restrictions or allergies

Project Schedule (Gantt Chart):

Figure 3: The Fall and Spring 2019 Gantt Chart can be seen in Figure 3. This outlines the development process and the amount of time each task takes to complete.

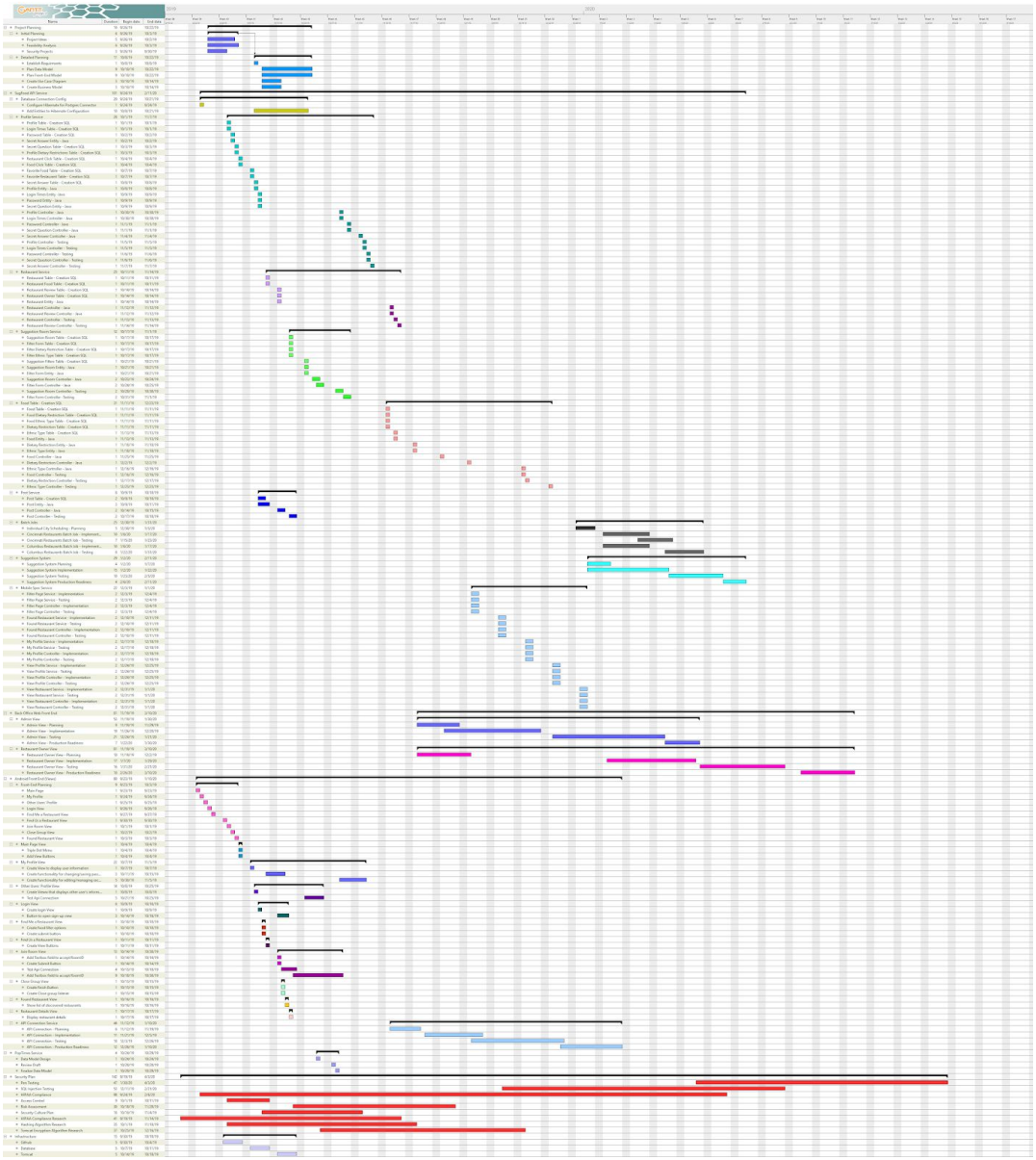


Figure III: Fall/Spring 2019 Gantt Chart

**Figure 3 is subject to change during the development process.*

Project Schedule (Work Breakdown Structure):

Table 3: The Work Breakdown Structure is displayed in Table 3. The WBS, like Figure 3, outlines the development process and presents the expected schedule for the completion of the development process.

Project Planning	19	9/26/19	10/22/19
Initial Planning	6	9/26/19	10/3/19
Project Ideas	5	9/26/19	10/2/19
Feasibility Analysis	6	9/26/19	10/3/19
Security Projects	3	9/26/19	9/30/19
Detailed Planning	11	10/8/19	10/22/19
Establish Requirments	1	10/8/19	10/8/19
Plan Data Model	9	10/10/19	10/22/19
Plan Front-End Model	9	10/10/19	10/22/19
Create Use Case Diagram	3	10/10/19	10/14/19
Create Business Model	3	10/10/19	10/14/19
SugFood API Service	101	9/24/19	2/11/20
Database Connection Config	20	9/24/19	10/21/19
Configure Hibernate for Postgres Connector	1	9/24/19	9/24/19
Add Entities to Hibernate Configuration	10	10/8/19	10/21/19
Profile Service	28	10/1/19	11/7/19
Profile Table - Creation SQL	1	10/1/19	10/1/19
Login Times Table - Creation SQL	1	10/1/19	10/1/19
Password Table - Creation SQL	1	10/2/19	10/2/19
Secret Answer Entity - Java	1	10/2/19	10/2/19
Secret Question Table - Creation SQL	1	10/3/19	10/3/19
Profile Dietary Restrictions Table - Creation SQL	1	10/3/19	10/3/19
Restaurant Click Table - Creation SQL	1	10/4/19	10/4/19
Food Click Table - Creation SQL	1	10/4/19	10/4/19
Favorite Food Table - Creation SQL	1	10/7/19	10/7/19
Favorite Restaurant Table - Creation SQL	1	10/7/19	10/7/19
Secret Answer Table - Creation SQL	1	10/8/19	10/8/19
Profile Entity - Java	1	10/8/19	10/8/19
Login Times Entity -Java	1	10/9/19	10/9/19
Password Entity - Java	1	10/9/19	10/9/19
Secret Question Entity - Java	1	10/9/19	10/9/19
Profile Controller - Java	1	10/30/19	10/30/19
Login Times Controller - Java	1	10/30/19	10/30/19
Password Controller - Java	1	11/1/19	11/1/19
Secret Question Controller - Java	1	11/1/19	11/1/19
Secret Answer Controller - Java	1	11/4/19	11/4/19
Profile Controller - Testing	1	11/5/19	11/5/19
Login Times Controller - Testing	1	11/5/19	11/5/19
Password Controller - Testing	1	11/6/19	11/6/19
Secret Question Controller - Testing	1	11/6/19	11/6/19
Secret Answer Controller - Testing	1	11/7/19	11/7/19
Restaurant Service	25	10/11/19	11/14/19
Restaurant Table - Creation SQL	1	10/11/19	10/11/19
Restaurant Food Table - Creation SQL	1	10/11/19	10/11/19
Restaurant Review Table - Creation SQL	1	10/14/19	10/14/19
Restaurant Owner Table - Creation SQL	1	10/14/19	10/14/19
Restaurant Entity - Java	1	10/14/19	10/14/19
Restaurant Controller - Java	1	11/12/19	11/12/19
Restaurant Review Controller - Java	1	11/12/19	11/12/19
Restaurant Controller - Testing	1	11/13/19	11/13/19
Restaurant Review Controller - Testing	1	11/14/19	11/14/19
Suggestion Room Service	12	10/17/19	11/1/19
Suggestion Room Table - Creation SQL	1	10/17/19	10/17/19
Filter Form Table - Creation SQL	1	10/17/19	10/17/19
Filter Dietary Restriction Table - Creation SQL	1	10/17/19	10/17/19
Filter Ethnic Type Table - Creation SQL	1	10/17/19	10/17/19
Suggestion Filters Table - Creation SQL	1	10/21/19	10/21/19
Suggestion Room Entity - Java	1	10/21/19	10/21/19
Filter Form Entity - Java	1	10/21/19	10/21/19
Suggestion Room Controller - Java	2	10/23/19	10/24/19
Filter Form Controller - Java	2	10/24/19	10/25/19
Suggestion Room Controller - Testing	2	10/29/19	10/30/19
Filter Form Controller -Testing	2	10/31/19	11/1/19
Food Table - Creation SQL	31	11/11/19	12/23/19
Food Table - Creation SQL	1	11/11/19	11/11/19
Food Dietary Restriction Table - Creation SQL	1	11/11/19	11/11/19
Food Ethnic Type Table - Creation SQL	1	11/11/19	11/11/19
Dietary Restriction Table - Creation SQL	1	11/11/19	11/11/19
Ethnic Type Table - Creation SQL	1	11/13/19	11/13/19
Food Entity - Java	1	11/13/19	11/13/19
Dietary Restriction Entity - Java	1	11/18/19	11/18/19
Ethnic Type Entity - Java	1	11/18/19	11/18/19
Food Controller - Java	1	11/25/19	11/25/19
Dietary Restriction Controller - Java	1	12/2/19	12/2/19
Ethnic Type Controller - Java	1	12/16/19	12/16/19
Food Controller - Testing	1	12/16/19	12/16/19
Dietary Restriction Controller - Testing	1	12/17/19	12/17/19
Ethnic Type Controller - Testing	1	12/23/19	12/23/19

Post Service	8	10/9/19	10/18/19
Post Table - Creation SQL	2	10/9/19	10/10/19
Post Entity - Java	3	10/9/19	10/11/19
Post Controller - Java	2	10/14/19	10/15/19
Post Controller - Testing	2	10/17/19	10/18/19
Batch Jobs	25	12/30/19	1/31/20
Individual City Scheduling - Planning	5	12/30/19	1/3/20
Cincinnati Restaurants Batch Job - Implement...	10	1/6/20	1/17/20
Cincinnati Restaurants Batch Job - Testing	7	1/15/20	1/23/20
Columbus Restaurants Batch Job - Implement...	10	1/6/20	1/17/20
Columbus Restaurants Batch Job - Testing	8	1/22/20	1/31/20
Suggestion System	29	1/2/20	2/11/20
Suggestion System Planning	4	1/2/20	1/7/20
Suggestion System Implementation	15	1/2/20	1/22/20
Suggestion System Testing	10	1/23/20	2/5/20
Suggestion System Production Readiness	4	2/6/20	2/11/20
Mobile Spec Service	22	12/3/19	1/1/20
Filter Page Service - Implementation	2	12/3/19	12/4/19
Filter Page Service - Testing	2	12/3/19	12/4/19
Filter Page Controller - Implementation	2	12/3/19	12/4/19
Filter Page Controller - Testing	2	12/3/19	12/4/19
Found Restaurant Service - Implementation	2	12/10/19	12/11/19
Found Restaurant Service - Testing	2	12/10/19	12/11/19
Found Restaurant Controller - Implementation	2	12/10/19	12/11/19
Found Restaurant Controller - Testing	2	12/10/19	12/11/19
My Profile Service - Implementation	2	12/17/19	12/18/19
My Profile Service - Testing	2	12/17/19	12/18/19
My Profile Controller - Implementation	2	12/17/19	12/18/19
My Profile Controller - Testing	2	12/17/19	12/18/19
View Profile Service - Implementation	2	12/24/19	12/25/19
View Profile Service - Testing	2	12/24/19	12/25/19
View Profile Controller - Implementation	2	12/24/19	12/25/19
View Profile Controller - Testing	2	12/24/19	12/25/19
View Restaurant Service - Implementation	2	12/31/19	1/1/20
View Restaurant Service - Testing	2	12/31/19	1/1/20
View Restaurant Controller - Implementation	2	12/31/19	1/1/20
View Restaurant Controller - Testing	2	12/31/19	1/1/20
Back Office Web Front End	81	11/19/19	3/10/20
Admin View	53	11/19/19	1/30/20
Admin View - Planning	9	11/19/19	11/29/19
Admin View - Implementation	19	11/26/19	12/20/19
Admin View - Testing	21	12/24/19	1/21/20
Admin View - Production Readiness	7	1/22/20	1/30/20
Restaurant Owner View	81	11/19/19	3/10/20
Restaurant Owner View - Planning	10	11/19/19	12/2/19
Restaurant Owner View - Implementation	17	1/7/20	1/29/20
Restaurant Owner View - Testing	16	1/31/20	2/21/20
Restaurant Owner View - Production Readiness	10	2/26/20	3/10/20
Android Front End (Views)	80	9/23/19	1/10/20
Front-End Planning	9	9/23/19	10/3/19
Main Page	1	9/23/19	9/23/19
My Profile	1	9/24/19	9/24/19
Other Users' Profile	1	9/25/19	9/25/19
Login View	1	9/26/19	9/26/19
Find Me a Restaurant View	1	9/27/19	9/27/19
Find Us a Restaurant View	1	9/30/19	9/30/19
Join Room View	1	10/1/19	10/1/19
Close Group View	1	10/2/19	10/2/19
Found Restaurant View	1	10/3/19	10/3/19
Main Page View	1	10/4/19	10/4/19
Triple Dot Menu	1	10/4/19	10/4/19
Add View Buttons	1	10/4/19	10/4/19
My Profile View	22	10/7/19	11/5/19
Create View to display user information	1	10/7/19	10/7/19
Create functionality for changing/saving pass...	3	10/11/19	10/15/19
Create functionality for editing/managing sec...	5	10/30/19	11/5/19

Other Users' Profile View	14	10/8/19	10/25/19
Create Vieww that displays other user's inform...	1	10/8/19	10/8/19
Test Api Connection	5	10/21/19	10/25/19
Login View	6	10/9/19	10/16/19
Create login View	1	10/9/19	10/9/19
Button to open sign-up view	3	10/14/19	10/16/19
Find Me a Restaurant View	1	10/10/19	10/10/19
Create food filter options	1	10/10/19	10/10/19
Create submit button	1	10/10/19	10/10/19
Find Us a Restaurant View	1	10/11/19	10/11/19
Create View Buttons	1	10/11/19	10/11/19
Join Room View	13	10/14/19	10/30/19
Add Textbox field to accept RoomID	1	10/14/19	10/14/19
Create Submit Button	1	10/14/19	10/14/19
Test Api Connection	4	10/15/19	10/18/19
Add Textbox field to accept RoomID	9	10/18/19	10/30/19
Close Group View	1	10/15/19	10/15/19
Create finish Button	1	10/15/19	10/15/19
Create Close group listener	1	10/15/19	10/15/19
Found Restaurant View	1	10/16/19	10/16/19
Show list of discovered restaurants	1	10/16/19	10/16/19
Restaurant Details View	1	10/17/19	10/17/19
Display restaurant details	1	10/17/19	10/17/19
API Connection Service	44	11/12/19	1/10/20
API Connection - Planning	6	11/12/19	11/19/19
API Connection - Implementation	11	11/21/19	12/5/19
API Connection - Testing	18	12/3/19	12/26/19
API Connection - Production Readiness	12	12/26/19	1/10/20
PopTimes Service	4	10/24/19	10/29/19
Data Model Design	1	10/24/19	10/24/19
Review Draft	1	10/28/19	10/28/19
Finalize Data Model	1	10/29/19	10/29/19
Security Plan	142	9/19/19	4/3/20
Pen Testing	47	1/30/20	4/3/20
SQL Injection Testing	53	12/11/19	2/21/20
HIPAA Compliance	98	9/24/19	2/6/20
Access Control	9	10/1/19	10/11/19
Risk Assessment	30	10/18/19	11/28/19
Security Culture Plan	18	10/10/19	11/4/19
HIPAA Compliance Research	41	9/19/19	11/14/19
Hashing Algorithm Research	35	10/1/19	11/18/19
Tomcat Encryption Algorithm Research	37	10/25/19	12/16/19
Infrastructure	15	9/30/19	10/18/19
Github	5	9/30/19	10/4/19
Database	5	10/7/19	10/11/19
Tomcat	5	10/14/19	10/18/19

Table III: Work Breakdown Structure (WBS)

Technical Elements

Nibbler has been developed to produce results for users quickly and allow them to make the best decision based on the results they are given. The architecture of the application can be separated into three different layers; network, application, and database. Below is a short description of what occurs at each of these layers as well as a visual technical architecture diagram.

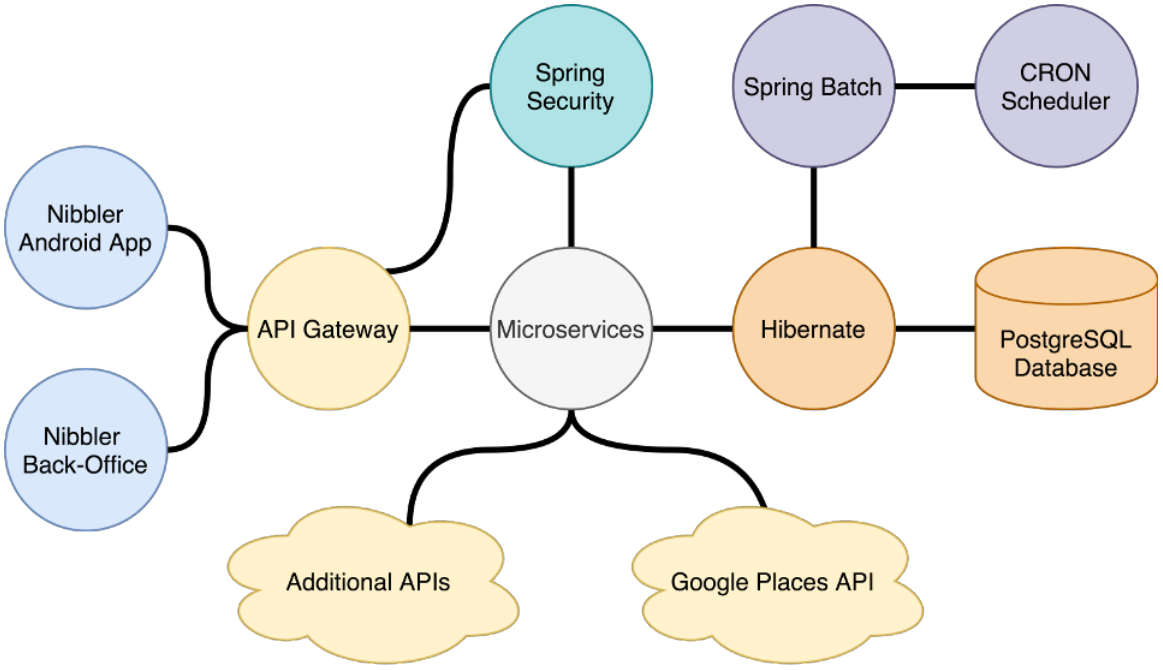


Figure IV: Technical Architecture Diagram, this diagram gives a visual representation of the application's build.

Network:

Nibbler will be deployed on AWS and our own hardware. The production environment will have it's own EC2 instance with the database running on it's own RDS instance. The entry point of the network will be an AWS load balancer that will direct traffic to one of our EC2 boxes running Nibbler servers. The load balancer and the EC2 instances will be open to the public internet while the database will be closed on a private network to prevent any outside tampering. The EC2 instances will be set up with an ansible playbook that will download and build the specified Nibbler source code. Then the Nibbler server will start running and now the setup process is complete on the servers.

The testing environment will be almost identical but all done on a machine we own. The building and deployment of the application is done manually. There is only one server and one database in this environment.

Application:

The application will be built using Java on the Android Studio platform. It will perform asynchronous network calls using a customized API to retrieve restaurant data and return a short list of Restaurants that can then be selected and viewed in more detail. This Restaurant Details View will use the Google Places API to open the restaurant on Google.

The back-end server will consist of three applications: Suggestion Food API, Fastest Food API, and Restaurant Discovering Daily Batch. All these applications will work in conjunction to deliver valuable data for our front-end mobile application.

The Suggestion Food API is a core back-end application in Nibbler. This application will serve the data that will be most used by the users. It will provide the mobile application with food filters, profile access, and restaurant suggestions.

The Fastest Food API is a secondary application that will deliver along-side the Suggest Food API. This application will deliver on finding food that is most accessible with a time constraint.

Restaurant Discovering Daily Batch is the application that will automatically pull restaurant data on per city basis. It will grab details of a restaurant as well as all their food items available. This will populate our database. The application will run regularly to keep the database up-to-date on current menus at restaurants.

Database:

For data stores, the application will use PostgreSQL. This database implementation uses a relational model to store data, this is important since there will be a lot of restaurant data and will make for fast queries. We will be using index features for access to commonly accessed data. The application will be using Hibernate for data access.

Security:

The security of Nibbler users was a high priority when developing the Nibbler applications. Not only can users find exactly what they are looking for when using the application, but they can do this with peace of mind knowing their information is stored safely and securely. Nibbler uses SHA 512 to encrypt user passwords stored in our database. SHA 512 is a secure hash function that allows passwords and other sensitive information to be stored in an encrypted format instead of in plain text. Along with SHA 512, Nibbler utilizes a Salt to ensure that no hash is the same. This added security step will ensure that attackers are unable to recognize two identical passwords after they pass through the hash function. Nibbler also protects data being transferred from our PostgreSQL database and the client by utilizing SSL, or Secure Socket Layer. Essentially, SSL creates a secure connection between both the user and our web server when data is being transmitted. This link between the server and the user is encrypted, allowing for data to be transmitted safely and securely.

User Interface

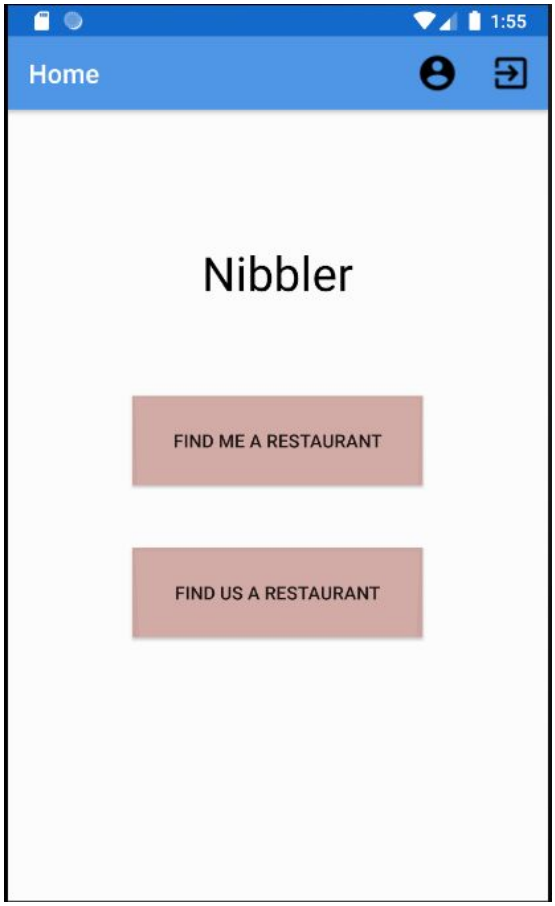


Figure V: Main Page View

Main Page View:

The main page view is a basic setup. The current design of the main page can be seen in Figure 4. The page will feature a “Find Me A Restaurant” button, a “Find Us A Restaurant” button, and a “Sign In” button, given that no user is currently signed in. Each button will take the user to different screens to complete specific tasks.

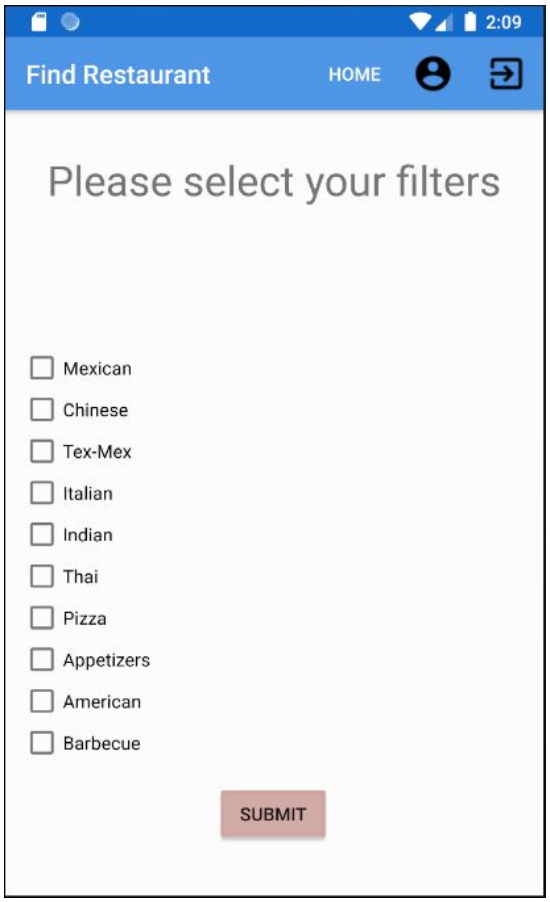


Figure VI: Find Me a Restaurant View

Find Me A Restaurant View:

The "Find Me A Restaurant" view will feature a list of food filter options. There will also be allergy and dietary restriction filters applied if the user is signed in. Each filter will be checked in their corresponding check box and then the user will submit the search by simply hitting the submit button. This will then return a list to the user of a few options that relate to the filters in their search.

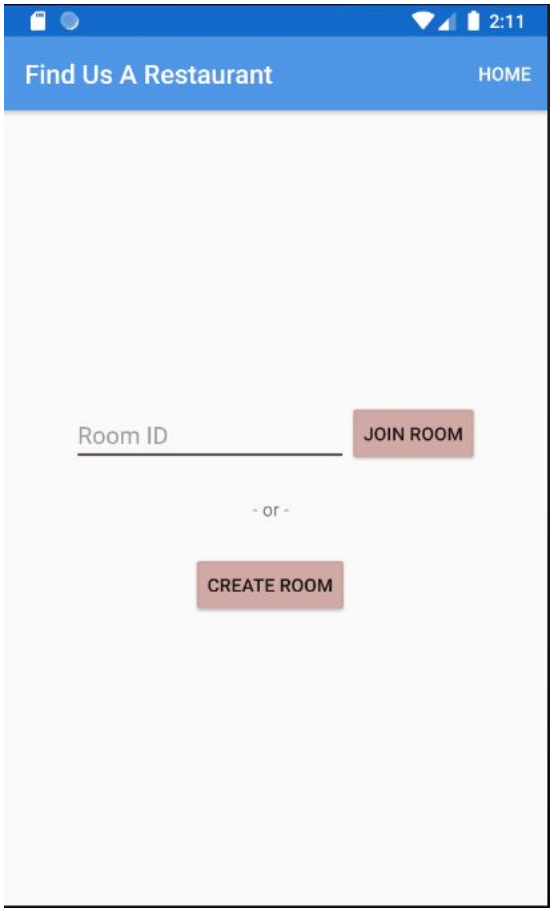


Figure VII: Find Us a Restaurant View

Find Us A Restaurant View:

The “Find Us A Restaurant View” is a lot simpler than the previous view. This page allows users to create a room. Other users can join this room and the group as a whole can vote on specific food options to make the decision-making process faster and easier. This view features two buttons that when pushed bring up the corresponding views. The first button is the “Create Room” button and the second is the “Join Room”. They function just as they sound. The only difference between the two is one requires you to enter a specific link to ensure that you join the correct room.

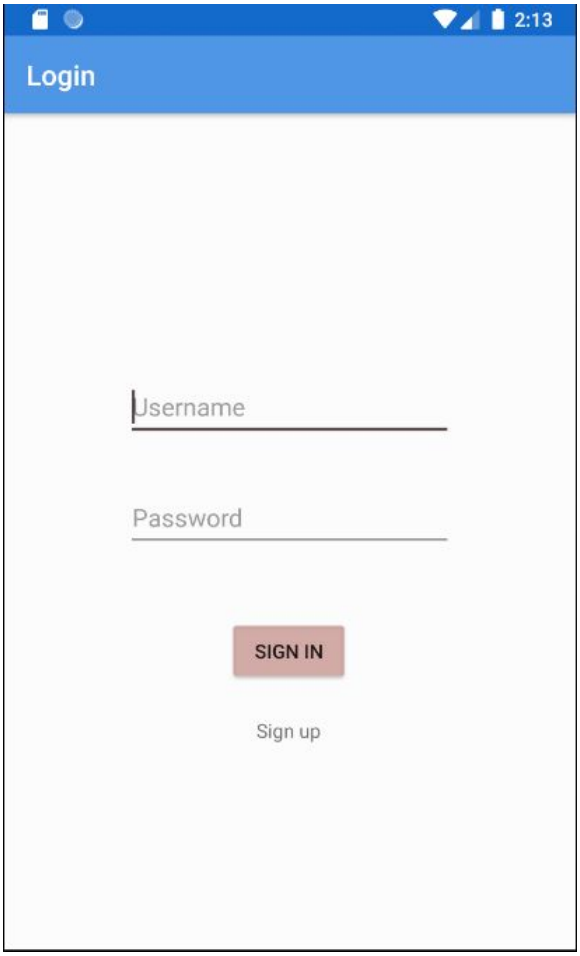


Figure VIII: Sign In View

Sign In View:

This view appears when the “Sign In” option is selected. The page is extremely simple and has four options to choose from. The options consist of two text input boxes (one for a username and the other for a password), a submit button to complete the sign in process, and a profile create button that allows the user to create a profile if they do not already have one.

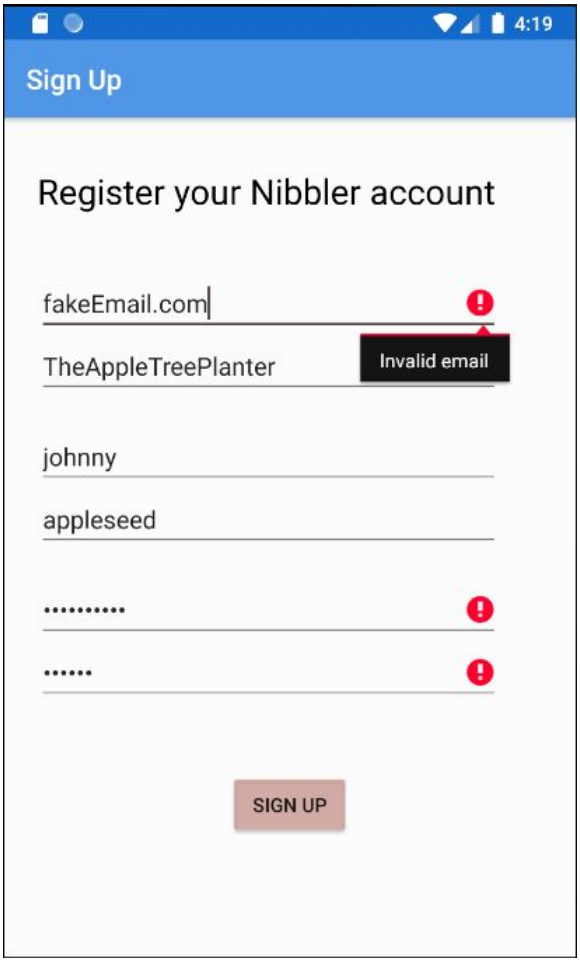


Figure IX: Sign Up View

Sign Up View:

This view appears when the “Sign up” option is selected from the Sign In view. The page asks for user information and provides validation on all input before submitting the information to the server. In Figure 8, the email and passwords were purposely input with invalid information to display the validation function; a user may select each of the red warning symbols to view that error. Once the user signs up and all information is validated, the application signs them in and takes them back to the view they had open.

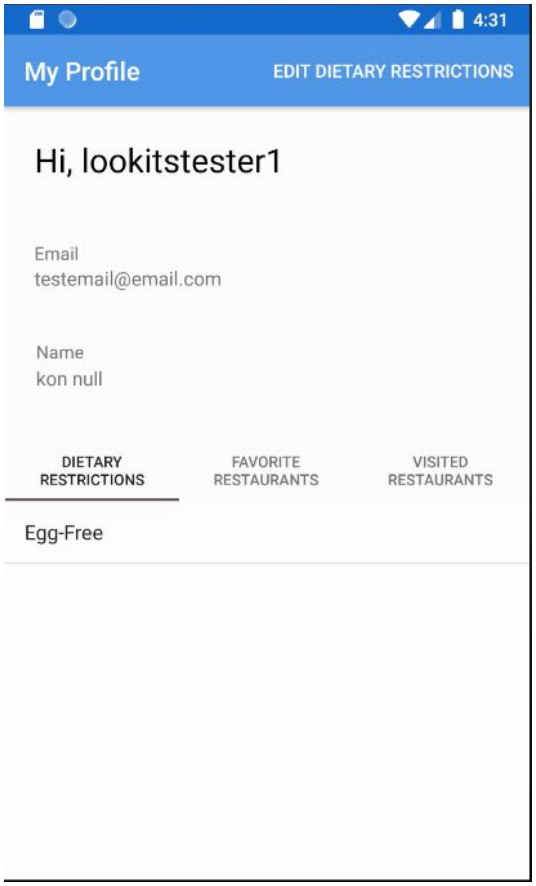


Figure X: My Profile View

My Profile View:

This view appears when the profile icon is selected from the main menu bar. It displays the user's basic information like their username, email, dietary restrictions, and their favorite or visited restaurants. From this view, the user may either edit their dietary restrictions or view the restaurant's detail page by selecting the restaurant from either the list of favorites or visited.

Figure XI: Edit Dietary Restrictions View

Edit Dietary Restrictions View:

This straightforward view provides the user a list of dietary restrictions to choose from and displays the previously selected restrictions by checking the box that represents those restrictions. When the user selects “Update,” the app will take the user back to their profile and display an updated list of dietary restrictions.

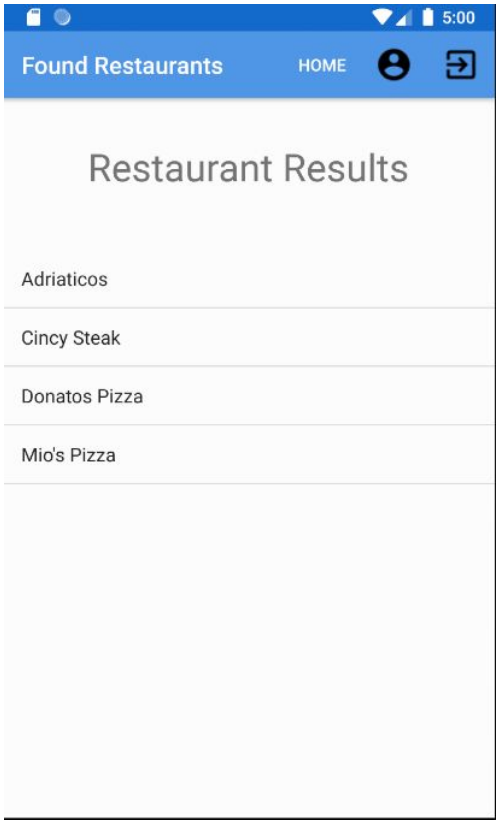


Figure XII: Found Restaurant View

Edit Dietary Restrictions View:

The Found Restaurant view as displayed in Figure 11 shows a short list of restaurants that the user might wish to check out based on their dietary restrictions if the user is signed in. From here, the user can then select one of the listed restaurants to view the details of the restaurant or go back and edit their filter selection.

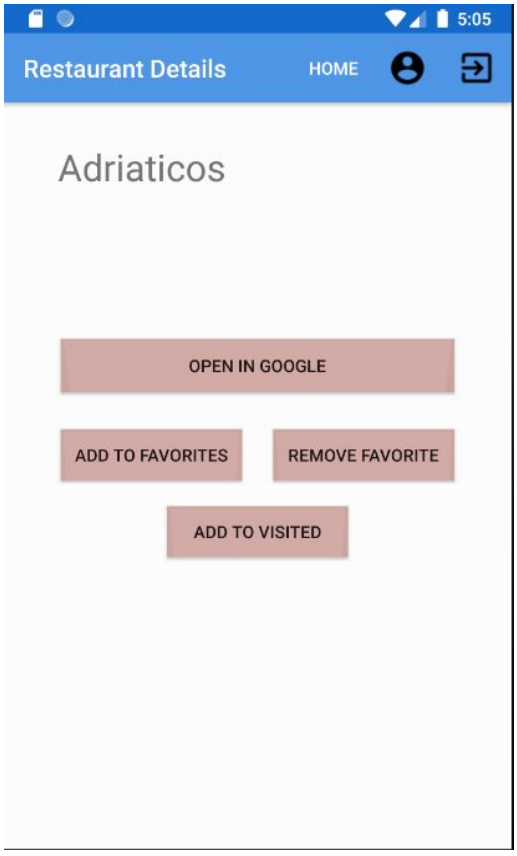


Figure XIII: Restaurant Details View

Restaurant Details View:

The Restaurant Details View allows the user to add the restaurant to either their favorites or visited, and remove from favorites. The user may also open the restaurant page in Google for more details. This view could be something expanded upon in the future with more details and other information from Google.

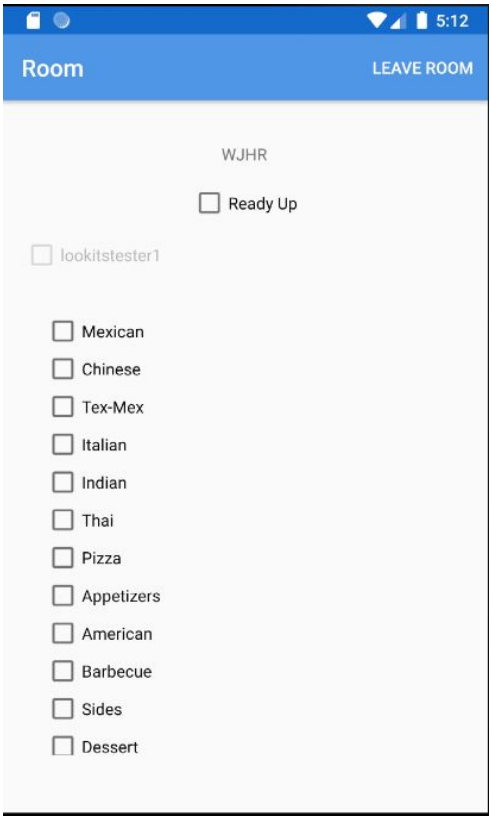


Figure XIV: Room View

Room View:

The Room view acts as the lobby screen for all users involved whether joining an existing room or creating their own room; this is the view that is displayed for both types of users and will display how many users are currently in the lobby. At the top, the room code is displayed, which is how other users will join the room. Once users select their filters, they may "Ready Up." This will cause the room's filter list to submit once all users are readied. Then a list of restaurants is displayed in the Found Restaurants view that is pertinent to the whole group. Users may also leave a room, which will take them back to the Main Page view.

Testing

Testing Objectives:

The main goal of this testing document is to outline the development teams testing procedures. These tests will ensure that the Nibbler application is running with the least amount of bugs as possible and will help to fix any massive issues before an official launch. Each test was completed with each update that was pushed through the backend. For example, after Ryan pushed the update through which allowed group rooms to be utilized on the frontend of the app, I retested all the SQL injection codes below on both the application itself and the back office.

Application Testers:

The testers are the individuals responsible for testing the functionality of the Nibbler environment. These users, developers, and peers of the development team give their input on how the application is running and functioning. Their feedback has been crucial to the development of Nibbler and has ensured that not only the application runs smoothly, but also that the interface is user friendly.

SQL Injection Tests:

The stability of our databases relies on the fact that they are not vulnerable to attacks such as SQL injections. SQL injections are commands sent to an SQL database that have malicious intent. These attacks can go either way, getting rid of or gathering information. We have specific parameters in place to block these malicious attacks and these parameters will be tested here.

SQL Command	Command Used	Results (Pass/Fail)
"1=1" Always True	**Command Template:** (User ID number) OR 1=1 **Actual Command:** Username: 2 OR 1=1	Pass(0/4 Fails)
"=" Always True	**Actual Command:** Username: "or" "=" Password: "or" "="	Pass(0/4 Fails)
Drop a Table	**Command Template:** (User ID number); DROP (Table Name) **Actual Command:** Username: 2; DROP UserName	Pass(0/4 Fails)
Logging in with Admin Status	**Command Template:** Username: admin--' Password: (random password) **Actual Command:** Username: admin--' Password: P@$$word	Pass(0/4 Fails)

Table IV: SQL Injection Testing, the table displays the results to the questions asked above. If the injection was successful, the results read fail. If the injection attack was not successful, the results read pass. The point of these SQL injection tests was to test common injection attacks against our database. For the final test involving admin status, we have accounts that we created with "P@$$word" as our password. That is why we used this as the test password.

Hash and Salt Testing:

To ensure our Hash functions and salt functions were working properly, four test accounts were created to demonstrate their functionality. The hash function applies a hash to the user password to encrypt the actual password so it is not displayed in plain text. The salt is an extra step of encryption to ensure that two passwords that are exactly the same do not have the same hash value. An example of this would be two different accounts having the password

“P@$$word”. A salt and hash would ensure that these two accounts would not have the same hash for their passwords. An example of our salt and hash functions working properly within our database can be seen below.

48	Salt Test	$2a$10$41zD1dsUdAn/j9WHEikdteDS/euRw.yU41CAzmtoWqfSPaIt/Ec8u	Salttester@uc.edu
49	Salt Tester 2	$2a$10$GvZZ4LP/hm01zp7cnM95D.YqU5Ne.taii7MpN6i71URJNTu6hxrmO	Salttester2@uc.edu

Figure XV: Salt and Hash Functions, the image above illustrates the functionality of our salt and hash functions. As you can see, both passwords are stored under different hashes. This demonstrates that the encryption of passwords and their hashes is working properly

Android Build Stability Tests:

The stability of the application is of utmost importance. It requires a level of stability that will deliver a seamless and crash-free user experience. The application will need to be stable on an Android emulator and a live device. In Nibbler's case, these will be an Android 8.1 emulator and Google Pixel 3a device with Android 10 respectively.

Device type	Test Duration	Crashes	Slow Downs
Android Emulator 8.1	1 hour	0 crashes	0 slowdowns
Android Emulator 8.1 (2)	2 hours	0 crashes	1 slowdown; Find Me A Restaurant content loaded 0.5 sec slower
Android Emulator 8.1 (3)	3 hours	0 crashes	0 slowdowns
Google Pixel 3a	1 hour	0 crashes	0 slowdowns
Google Pixel 3a (2)	2 hours	0 crashed	0 slowdowns
Google Pixel 3a (3)	3 hours	0 crashes	0 slowdowns

Table V: Android Build Stability Testing, the numbers above display the results from all stability testing. Each test passed as there were no crashes and the only slowdown that occurred did not result in a connection timeout with the server.

UI Tests:

The Nibbler application's user interface needs to be bug-free, with every event and component interaction to be handled and functional. This test will also consider the usability of each activity and speed is as consistent as possible. Each interactive component will be tested 10 times on both Emulator and Google Pixel 3a.

UI Test	All Components Working	Activity Switch Slowdown	Navigation Time
Sign up as new user	Yes	No	3.45 sec from start activity to sign up; 0.42 sec from registration back to start activity
Find restaurants individually, select one restaurant	Yes	No	7.40 sec from Main Page to a selected restaurant's details page
Choose 3 new dietary restrictions	Yes	No	7.31 sec from start activity to Edit Dietary Restrictions, selecting 3 restrictions, the returning to profile page
Remove a restaurant from favorites starting at profile	Yes	No	6.14 sec from profile to details of favorite restaurant selected, removing the restaurant, then returning to profile
Start a room and submit filters	Yes	No	10.33 sec from Find Us a Restaurant to create room, select filters, then ready up and be displayed results of restaurants
join room but then leave	Yes	No	5.41 sec from Main Page to join room and then leave room

Sign in, then sign out	Yes	No	5.53 sec to sign in, 0.24 sec to sign out
Add new restaurant to favorites, then from profile remove old restaurant	Yes	No	25.61 sec to go through entire flow
View 3 restaurants from search	Yes	No	23.21 sec to view all three restaurants
View restaurant in Google	Yes	No	15.42 sec to view a found restaurant and return to details page

Table VIa: UI Testing, each test passed and all components are working in the Android 8.1 emulator. There were no slowdowns during the activity lifecycle changes and no bugs or failures.

UI Test	All Components Working	Activity Switch Slowdown	Navigation Time
Sign up as new user	Yes	No	3.68 sec from start activity to sign up; 0.38 sec from registration back to start activity
Find restaurants individually, select one restaurant	Yes	No	7.23 sec from Main Page to a selected restaurant's details page
Choose 3 new dietary restrictions	Yes	No	8.01 sec from start activity to Edit Dietary Restrictions, selecting 3 restrictions, the returning to profile page
Remove a restaurant from favorites starting at profile	Yes	No	6.10 sec from profile to details of favorite restaurant selected, removing the restaurant, then returning to profile

Start a room and submit filters	Yes	No	10.51 sec from Find Us a Restaurant to create room, select filters, then ready up and be displayed results of restaurants
join room but then leave	Yes	No	5.38 sec from Main Page to join room and then leave room
Sign in, then sign out	Yes	No	6.01 sec to sign in, 0.25 sec to sign out
Add new restaurant to favorites, then from profile remove old restaurant	Yes	No	26.56 sec to go through entire flow
View 3 restaurants from search	Yes	No	23.19 sec to view all three restaurants
View restaurant in Google	Yes	No	15.52 sec to view a found restaurant and return to details page

Table VIb: UI Testing, the live Google Pixel 3a device results are above. There were no bugs, issues, or slowdowns between the activities. Any differences between the navigation times of both application builds is minimal.

Connection Tests:

The Nibbler application requires successful connections to our server and must be responsive to information exchange. We tested the connection with the Login function several times consecutively and recorded the time on both the Android 8.1 emulator and Google Pixel 3a.

Connections Test	Login Success	Failures	Transfer Speed
Emulator Test 1	Yes	No	0.32 sec
Emulator Test 2	Yes	No	0.26 sec
Emulator Test 3	Yes	No	0.28 sec
Emulator Test 4	Yes	No	0.24 sec
Emulator Test 5	Yes	No	0.21 sec
Emulator Test 6	Yes	No	0.26 sec
Emulator Test 7	Yes	No	0.23 sec
Emulator Test 8	Yes	No	0.20 sec
Emulator Test 9	Yes	No	0.25 sec
Emulator Test 10	Yes	No	0.19 sec

Table VIIa: Connection Testing, the results above display concurrent successes on the Android 8.1 emulator.

Connections Test	Login Success	Failures	Transfer Speed
Google Pixel 3a Test 1	Yes	No	0.41 sec
Google Pixel 3a Test2	Yes	No	0.39 sec
Google Pixel 3a Test 3	Yes	No	0.34 Sec
Google Pixel 3a Test 4	Yes	No	0.31 sec
Google Pixel 3a Test 5	Yes	No	0.35 sec
Google Pixel 3a Test 6	Yes	No	0.36 sec
Google Pixel 3a Test 7	Yes	No	0.34 sec
Google Pixel 3a Test 8	Yes	No	0.31 sec
Google Pixel 3a Test 9	Yes	No	0.28 sec
Google Pixel 3a Test 10	Yes	No	0.25 sec

Table VIIb: Connection Testing, the results above display concurrent successes on the Google Pixel 3

Backend Tests:

The backend tests entail a thorough assessment of the integrity and availability of the data that will provide data to the front-end applications. Unit tests have been created to test the business logic and Hibernate connections with the database. Hibernate validates the database schema to ensure consistency for tests that run shortly after. These tests run every time the application is built. Manual tests are done with curl to examine api endpoints for data integrity. Five trials of each test will be documented in the table below. Details of each test segment are outlined after this section.

Test Collection	Trials	Passes	Fails
Business Logic	5	5	0
Hibernate Connection	5	5	0
DB Schema Validation	5	5	0
Endpoint	5	5	0

Table VIII: Unit Tests

Business Logic Unit Tests:

Tests have been created for each application use case used in the front-end applications. Any data manipulation performed in this code is checked for in the output of the test runs.

Service Test	Trials	Passes	Fails
Profile Service	5	5	0
Restaurant Service	5	5	0
Role Service	5	5	0
Food Service	5	5	0
Dietary Restriction Service	5	5	0
Cuizine Service	5	5	0

Table IX: Business Logic Unit Tests

Hibernate Connection Tests:

The hibernate test ensures that the connection and queries to the database are working. The queries include all CRUD operations as well as any custom queries. Any failures in the tests

result in rollbacks in the database. Tests are done on an isolated test database. A test has been made for all distinct records in the database.

Hibernate Test	Trials	Passes	Fails
Profile Table	5	5	0
Restaurant Table	5	5	0
Role Table	5	5	0
Food Table	5	5	0
Dietary Restriction Table	5	5	0
Cuizine Table	5	5	0
Secret Question Table	5	5	0
Restaurant Owner Table	5	5	0
Login Times Table	5	5	0
Post Table	5	5	0
Favorite Restaurant Table	5	5	0
Visited Restaurant Table	5	5	0
Food Dietary Rest Table	5	5	0
Food Cuizine Table	5	5	0
Favorite Food Table	5	5	0
Restaurant Review Table	5	5	0

Table X: Hibernate Connection Test

Database Validation:

These tests are run by hibernate everytime the back-end application is built and run. Hibernate checks all the registered JPA classes in the project to base it's validation on.

Database Validation Test	Trials	Passes	Fails
Profile Table	5	5	0
Restaurant Table	5	5	0
Role Table	5	5	0
Food Table	5	5	0
Dietary Restriction Table	5	5	0
Cuizine Table	5	5	0
Secret Question Table	5	5	0
Restaurant Owner Table	5	5	0
Login Times Table	5	5	0
Post Table	5	5	0
Favorite Restaurant Table	5	5	0
Visited Restaurant Table	5	5	0
Food Dietary Rest Table	5	5	0
Food Cuizine Table	5	5	0
Favorite Food Table	5	5	0
Restaurant Review Table	5	5	0

Table XI: Database Validation Test

Endpoint Tests:

The endpoint tests are manually done with curl calls. The curl calls are prepared for every endpoint to run at any time. Once the curl has run, the output is validated by a team member.

Endpoint Test	Trials	Passes	Fails
Suggestion Food API	5	5	0
Suggestion Food Lobby API	5	5	0

Table XII: Endpoint Tests

Problems and Future Plans

Problems Encountered:

There were several problems that we encountered during the development process. Below you can see how we adapted or adjusted our development around these issues:

- More complex UI components to display the restaurant search results, like the RecyclerView. This would have improved our user experience and the overall design of the application. The problem remains unsolved for now as each attempt to implement the RecyclerView with our own data would not work.
- There were certain times that the application's validation cookie, which was persisted in the device, would not work in authenticating a signed in user. The solution to this was to ask the user if they would like to provide credentials again to retrieve a new session cookie and continue on the application. This solution is not ideal, but it was the only solution that could be implemented in the time allotted.
- Automatic Data retrieval for restaurant menus was incorrect. We solved this problem by changing the way we collected data. Instead of automatically collecting this data, we

decided to enter all data by hand. With the amount of time we were given and the issues with incorrect information, we decided the best way to ensure that all our information was correct, was to collect and enter it by hand. This ensured that nothing would get past the development team unchecked.

- The original idea was to create a database containing all Cincinnati area restaurants as well as all the dietary information pertaining to their menus. Unfortunately, with the resources we were working with and the short amount of time we had, this goal was not going to be obtainable. We decided to restrict the boundaries to the Clifton area.

Future Recommendations:

In the future, we have several features and ideas that we would like to add to our current build. One of these ideas is extending the area of restaurants. Currently we are restricting the app to the Clifton area. We first want to include the city of Cincinnati and then move on to its surrounding suburbs and towns. After this, we would look forward to adding new cities such as Columbus, Ohio or Louisville, Kentucky. Another future plan would be adding more user functions regarding restaurants. For example, reserving a table. Finally, the biggest update we are looking forward to is to edit our UI. It currently does not utilize Android's full UI component library. We decided to look at functionality before the actual UI design because of time constraints. Luckily, this is a simple change and can be done in the near future. If we were to do this project all over again, we probably could have done this project entirely in Android Studio, as opposed to also having an API. If we used this, we could have utilized Firebase, which could serve as a database as well as a user authentication tool.

Conclusion (Spring 2020)

The development of the application has been a difficult and tedious process, but we all have been able to create something great from a simple idea. After several meetings and multiple ideas thrown back and forth, we finally began developing the application that we have today. All of us have developed new skills throughout development. These skills range from items such as learning how to perform SQL injections, to learning how to apply more complex Android programming concepts such as using shared preferences, to utilizing session cookies. Each member of the development team researched, learned and practiced new techniques and abilities throughout the development lifecycle of Nibbler. We started out with the idea of a restaurant finder that would allow users to search for restaurants with specific allergy filters applied to the search. This then escalated to applying more than one filter to a search because many mainstream restaurant finders do not allow this. This again escalated to a new level. The final product was an application that completes all these tasks, but also eliminates the issue of Analysis Paralysis. As we have previously stated, Analysis Paralysis can halt the decision making process because it overwhelms the user with options. We realized if we limited users options then not only would it make deciding easier, but also would promote trying new restaurants. This is the main difference with our application when compared to others. Where applications such as Yelp, Google, or Doordash want to give you as many options as possible, Nibbler wants to limit them to make the decision easier. Combining this main focus on eliminating Analysis Paralysis with applying multiple filters, dietary restriction filters, and food allergy filters and you have a new kind of restaurant finder. Maybe the most unique and useful restaurant finder to date.

Conclusion (Fall 2019)

The development of the application has been a difficult and tedious process, but we all have been able to create something great from a simple idea. After several meetings and multiple ideas thrown back and forth, we finally began developing the application that we have today. We started out with the idea of a restaurant finder that would allow users to search for restaurants with specific allergy filters applied to the search. This then escalated to applying more than one filter to a search because many mainstream restaurant finders do not allow this. This again escalated to a new level. The final product was an application that completes all these tasks, but also eliminates the issue of Analysis Paralysis. As we have previously stated, Analysis Paralysis can halt the decision making process because it overwhelms the user with options. We realized if we limited users options then not only would it make deciding easier, but also would promote trying new restaurants. This is the main difference with our application when compared to others. Where applications such as Yelp, Google, or Doordash want to give you as many options as possible, Nibbler wants to limit them to make the decision easier. Combining this main focus on eliminating Analysis Paralysis with applying multiple filters, dietary restriction filters, and food allergy filters and you have a new kind of restaurant finder. Maybe the most unique and useful restaurant finder to date.

References

"Facts and Statistics." *Food Allergy Research & Education (FARE)*, Nov. 2017.

Schwartz, Barry. *The Paradox of Choice: Why More Is Less*. Ecco, an Imprint of HarperCollins Publishers, 2016.

"Products - Data Briefs - Number 10 - October 2008." *Centers for Disease Control and Prevention*, Centers for Disease Control and Prevention, 6 Nov. 2015.

Insausti, Sebastian. "How to Secure Your PostgreSQL Database - 10 Tips." Severalnines, 21 Aug. 2019.

Nohe, Patrick. "The Difference between Encryption, Hashing and Salting." *Hashed Out by The SSL Store™*, 27 Aug. 2019.

Appendix A

1. UI: User Interface
2. HIPPA: Health Insurance Portability and Accountability Act
3. AWS: Amazon Web Services
4. EC2: Amazon Elastic Compute Cloud
5. RDS: Amazon Relational Database Service
6. API: Application Programming Interface
7. SQL: Structured Query Language
8. SHA: Secure Hashing Algorithms
9. SSL: Secure Socket Layers
10. CRUD: Create, Read, Update, and Delete
11. JPA: Java Persistence API
12. FARE: Food Allergy Research & Education
13. CDC: Center for Disease Control & Prevention

Appendix B

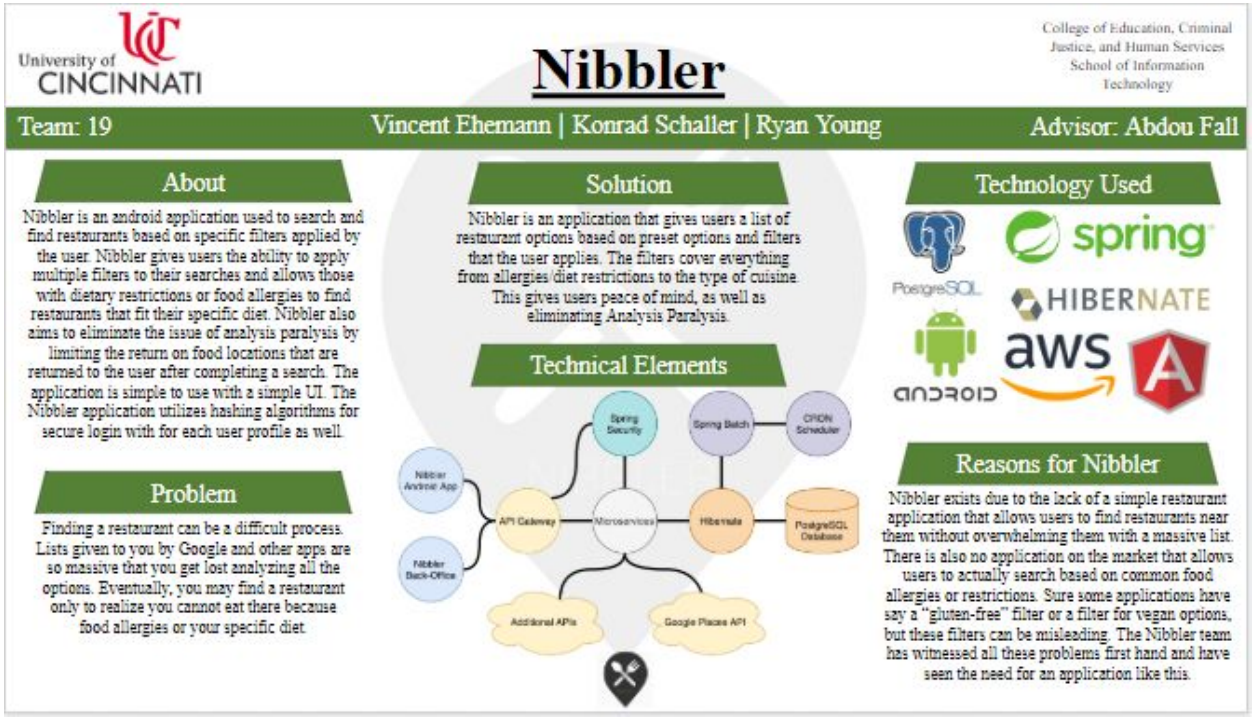


Figure XVI: Poster