

## Team 4

Sidhant Chitkara  
Alexander Meyer  
Apoorva Parmar  
Yash Pujara  
James Shao  
Michael Vieck

# Intelligent Search

September 19, 2016

## Purpose

Currently, students at Purdue must independently research dining court menus and City Bus routes ahead of time before they actually use the services. While there are apps and web services involving both, they are all essentially glorified data dumps. When using these products, students and even non-students are put to the task of manually sifting through this data, sometimes without results. Our team wants to eliminate these useless tasks by automating and creating an intelligent search functionality for all of the data. Intelligent Search's purpose aims to provide an easy user experience to Lafayette residents and Purdue Students, where the user base would simply have to worry about figuring out exactly what they want.

## Functional Requirements

1. Users can search for menus at Purdue's dining courts
2. Users can search for specific food items to check location availability
3. Users can search for food items with any allergens they might contain
4. Users can access nutritional information (calories etc.) before choosing a dining court
5. Users will be notified about any saved searches and foods they favorited
6. Users can find the nearest bus and bus stops
7. Users will be given the shortest time for a bus destination
8. Users can search for fastest routes to their destinations
9. Users can log in and save their preferences
10. Users can change their account information such as passwords and allergies

## Non-Functional Requirements

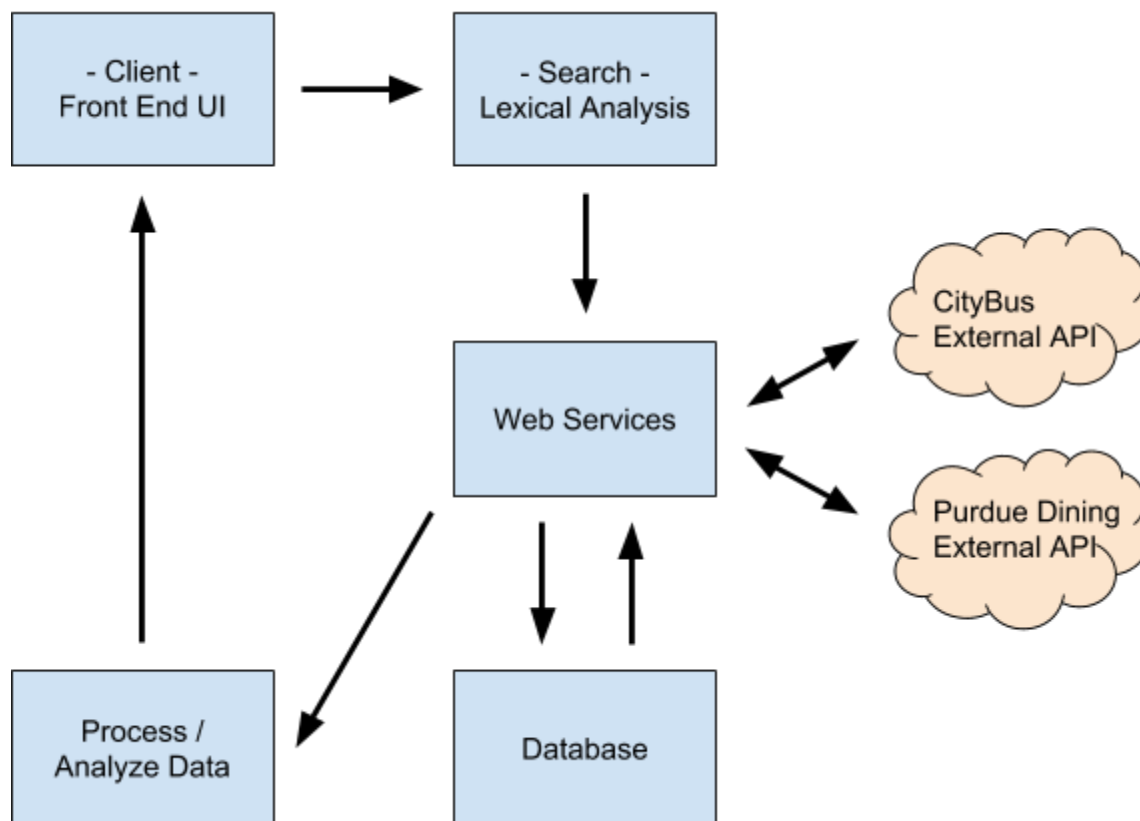
1. The database will be able to scale to meet a higher network demand.
2. The accuracy of the search will be high.
3. Users will have web as well as mobile accessibility.
4. Users' login information will be encrypted.
5. Users would have a fast response time.

## Design Outline

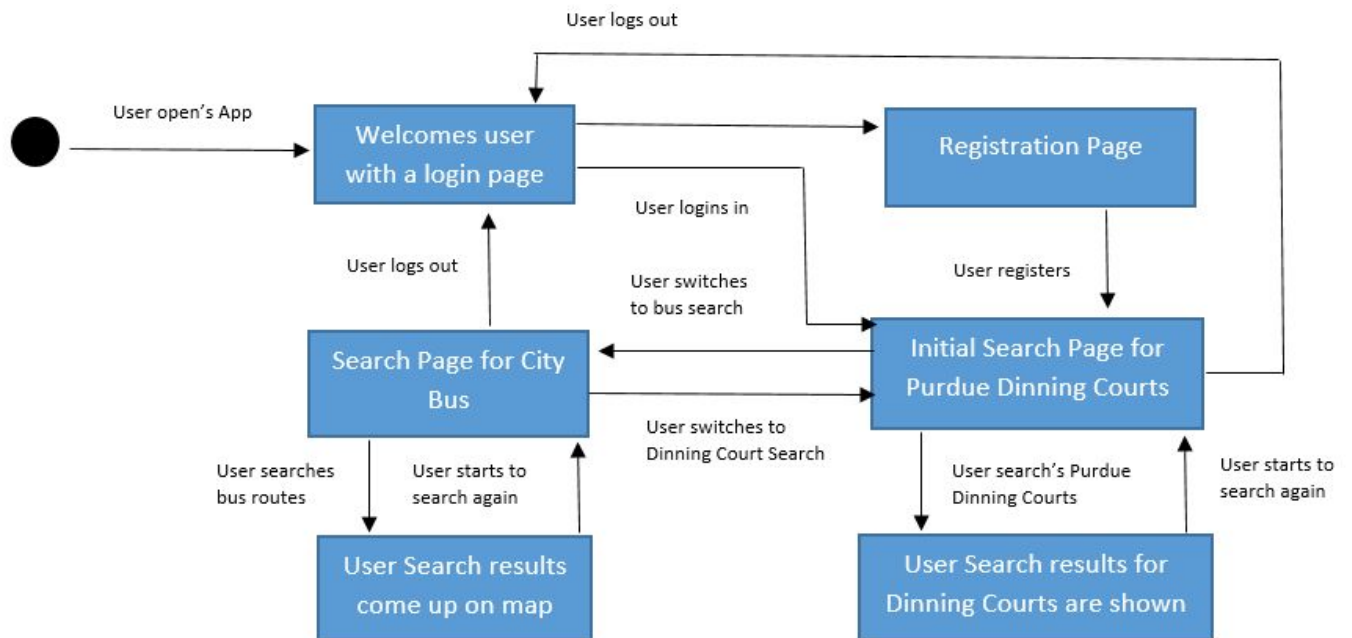
Our project goal is give users a way to search through Purdue Dining Court's API, as well as City Buses API. We will use Client Server to receive this information and send it to the users from the database as they search for it. They will be able to do this from a web site and/or an Android App. For the Dining Court information, we will update this daily from the Purdue's Dining Court API, while the City Bus information will be called from the live API whenever a search is needed. Static route and bus data will be stored on the database.

## Architecture Diagram (High Level)

The high level diagram shows all the classes and how they interact with each other in the finished product. Starting at the Front End, as users enter their search, it is passed to the lexical search for analysis. The Lexical Search then decides whether the search will require calls to an external API service or our own database services. If passed through our own web services, the service will form a query based on the search analysis and pass the data to be analyzed. This data can then be formatted or just passed to the front end for dynamic loading and presentation.



## Activity State Diagram



## Design Issues

### Functional:

- 1) Do users need to login to use the app?
  - a) Option 1: Allow login through facebook or google+
  - b) Option 2: Allow login through student's Purdue accounts
  - c) Option 3: Create a unique login feature

Decision: We decided to implement a unique login feature, with minimal information required from the user. This will allow us to store information from each user if needed in the future, without restricting the app to Purdue students, which the purdue login would do, and without security concerns, which could come up with facebook or

google logins. Ideally, having additional options of facebook and purdue logins would make the app more scalable.

- 2) Do users need to search for the same things (food items, bus routes) every time?
  - a) Option 1: Users log in so that old searches are saved in the back end
  - b) Option 2: Implementing an auto complete for most frequent searches
  - c) Option 3: Allowing users to mark favorites on food items, bus routes etc

Decision: We chose to allow users to mark favorites so that they don't have to repeat the search at all, but rather just view their favorites or get notified when one of their marked options is available again.

#### Non Functional:

- 1) Is an API web-service required between the lexical search and database?
  - a) Option 1: Use a service API that the lexical search calls once the Search has been parameterized
  - b) Option 2: Directly call the database with a query after parsing the search

Decision: Create a pseudo service that is in-between the lexical search and database. A full API is not needed because the search and database are both backend systems and the search class will be the only thing that calls the database. Instead, we will use a service class with functions that build each query, but it will not be a web accessible API.

- 2) How do we break user searches into keywords to display results?
  - a) Option 1: Use a txt file containing all expected keywords and match database accordingly
  - b) Option 2: Break the search input to find user demands and form SQL query

Decision: We chose to drop redundant/irrelevant words from the user search, and search for keywords such as the dining court name, meal timing etc. This will keep our

search results more precise to user requirements, as txt files are often used for less open ended search engines.

3) What language do we process the user input in?

- a) Option 1: Java
- b) Option 2: C++
- c) Javascript

Decision: The front end is in javascript, but we chose to process user searches in C++, because C++ has the option of linkage to both SQL for the backend, and javascript for frontend. So we can match keywords with our database and provide appropriate results with the quickest search algorithms.

## Design Details

Outline:

### Front End:

- LoginScreen
  - This class would be the starting activity of the app.
  - It would provide the functionality to a user who wants to login or register.
- MainScreen
  - This class would control the main display of the app.
  - It would be responsible to retrieve food items from the database and display them on the screen on the basis of search.
  - It would display a side drawer which will consist of User Settings, Favourites and logout
  - There will be a toggle button to switch between the Dining menu page and the CityBus page
  - Each page will have a search bar at the top right which does the lexical search based on the user's request.

### Search:

- I. The search will parse the user's entry for keywords and relate these keywords to gain an understanding of what data the user is asking for.

II. The keywords will determine what words to pass to the Services in order to construct the appropriate query.

**Services:**

- I. Based on information received from the Search class, this class will form a local query for the database or call an external API for the needed information.
- II. After it has received all the necessary data, it will pass everything onto the Processing class.

**Database:**

- I. The database will store all the static data used by Intelligent Search
- II. Data stored will be food item information, daily menus, and bus route information.
- III. A daily script will update the daily menu table and missing item entries will be added at this time.
- IV. A daily script will check to make sure no route changes have been uploaded.

**Data Processing:**

- I. Will receive information from the search class on whether the data should be further processed or if a data dump was queried.
- II. Will receive data from services class
- III. Will be able to manipulate data (e.g calorie calculations, shortest route, etc)
- IV. Data will be standardized before returning to the front end class



V. Will be split into many subclasses and functions that will be called depending on what kind of analysis is required.

### Interactions between Classes:

The **Front End** class will pass the User's search query to the Lexical **Search** Analysis Class. The search class will then determine what exactly the user is looking for.

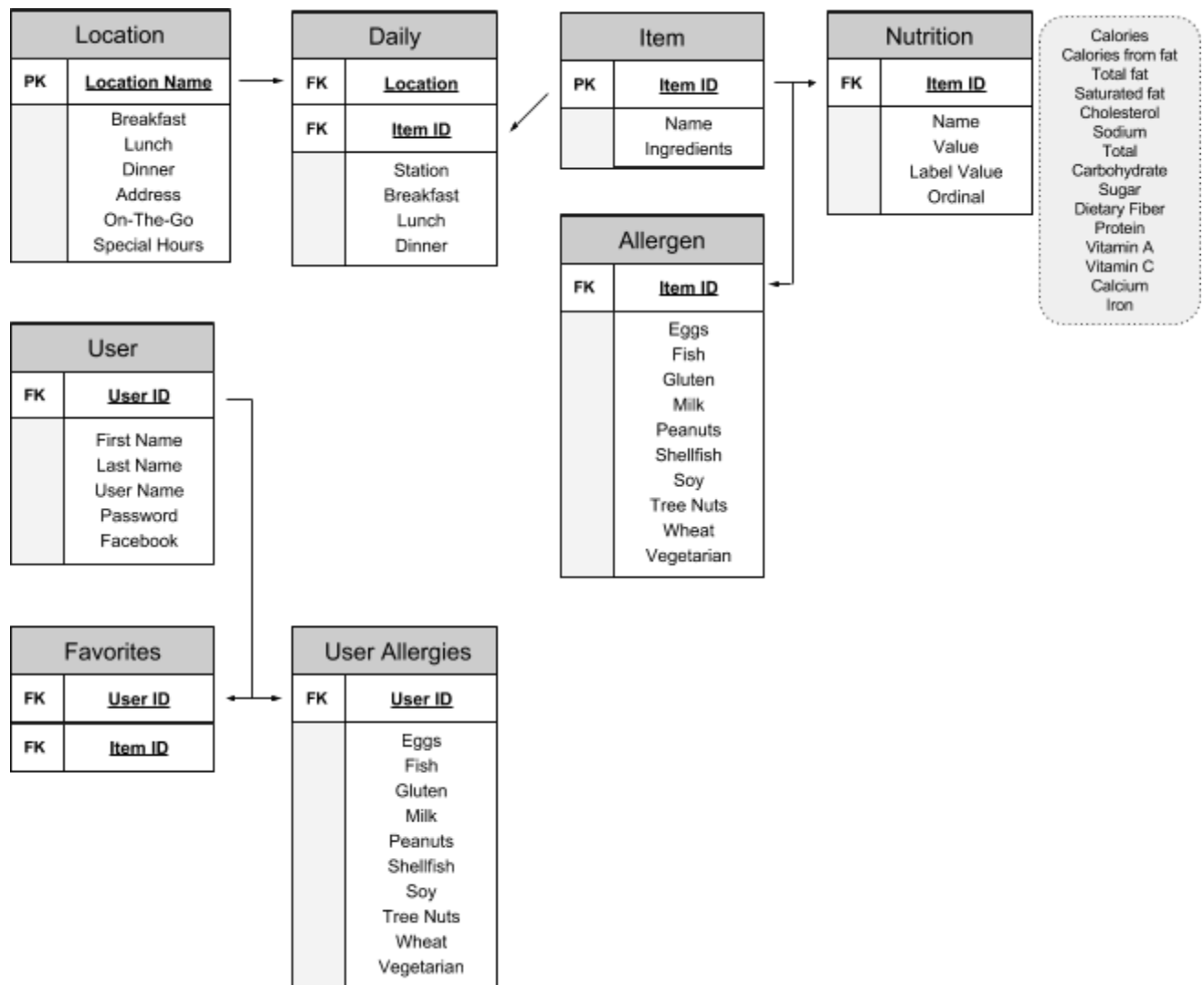
The **Search** class will be able to determine whether a local Database call is needed or if additional "live" data is needed from external APIs. It will then call the appropriate services and let the **Data Processing** class know if the user wants the data processed or not.

The **Service** class will construct and call the local database or will call an external API. The information received will then be passed to the **Data Processing Class**

The **Data Processing** class will receive information about the query from the **Search** analysis and data from the **Services** class. From here, it will determine if the data needs to be processed / if numbers need to be crunched. Once processed, the data is reformatted to match a standardized template to make parsing it easier. The data is then passed to the **Front End** to be displayed.

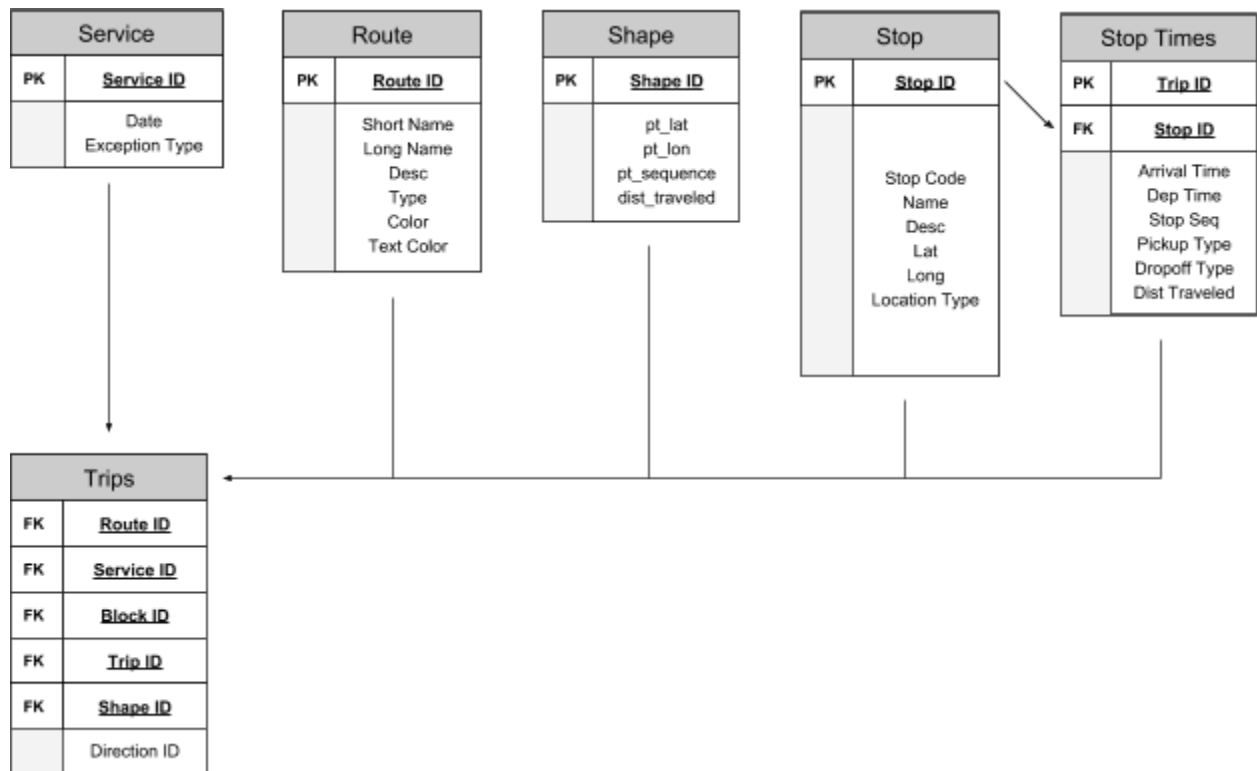
## Database Design (Dining)

The Purdue Dining Database will be a relational SQL database that holds item specific information along with the current day's dining court menus. This will help limit API calls to the purdue dining services and help reduce chokepoints. Item information will be split into multiple tables to make calls more specific. This will allow us to query only what we need and not get other miscellaneous data.



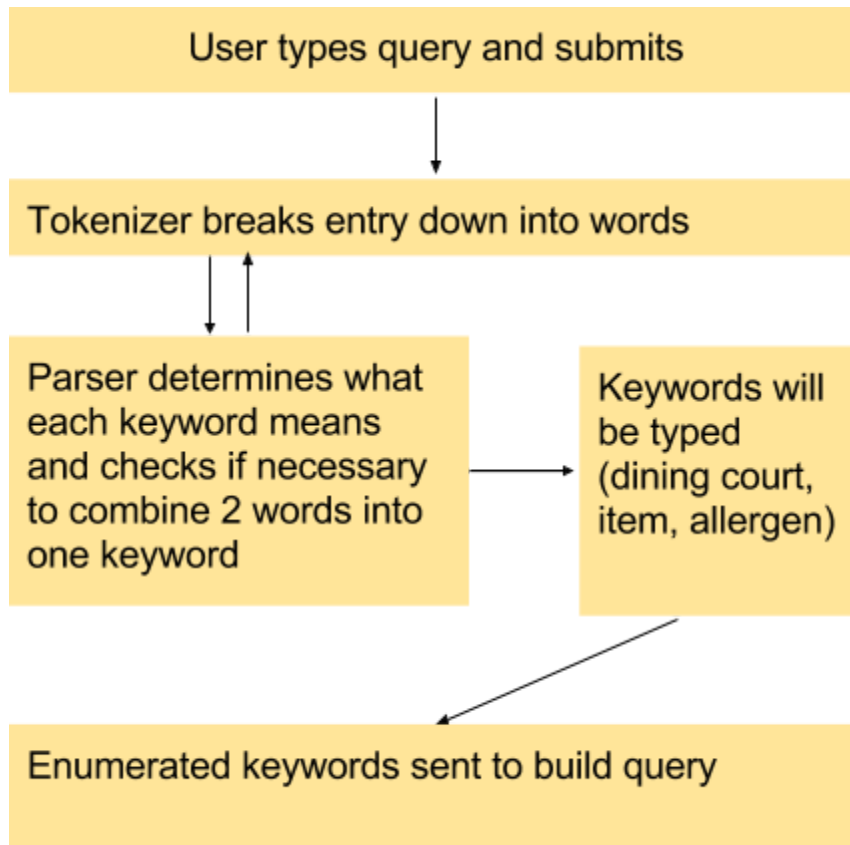
## Database Design (CityBus)

The CityBus side of the database design is based on the static data given to us. The data has already been laid out into table format. Live data will have to be queried via CityBus' own webservice / API - details that have not completely been hashed out yet. The static data will provide basic route and bus information and will all be located in the database tables.



## Search Dataflow

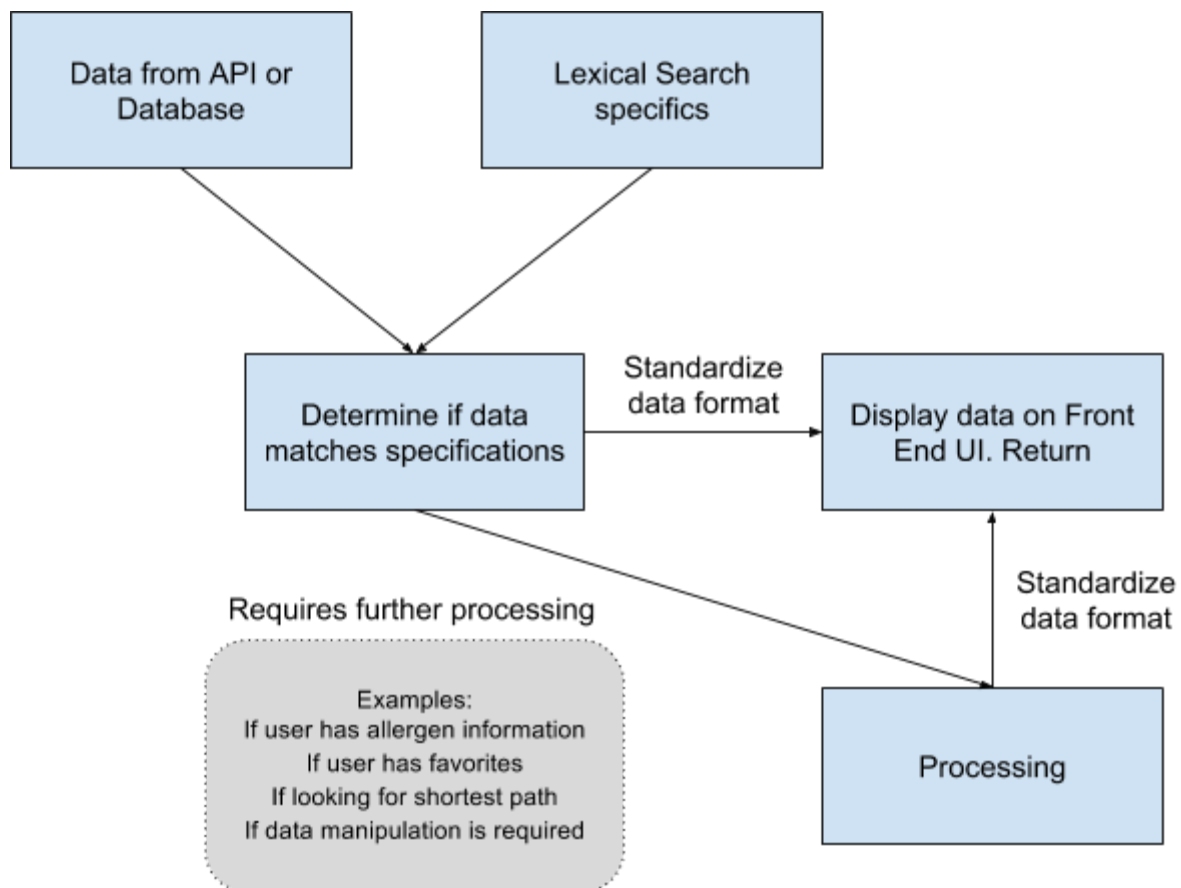
Our goal here is to take in the raw user input, and output a set of enumerated keywords that will build up the query. For example, a specific dining court name and item name should be passed to the query builder for the search “Does wiley have pizza today?”. The translation from the sentence to keywords will be broken into several parts outlined below.



## Data Processing / Analytics

Once data is returned from the database / external APIs, the data processing class will have to determine in what format the data should be returned. It also has to decide whether the data needs further processing in order to better fit the user's query. This decision making process will be done via user preferences and the lexical search. The data is then modified

into either bus / dining friendly formatting to allow the front end class to parse it.  
Standardized data formats will let the front end do less work before presenting the data.



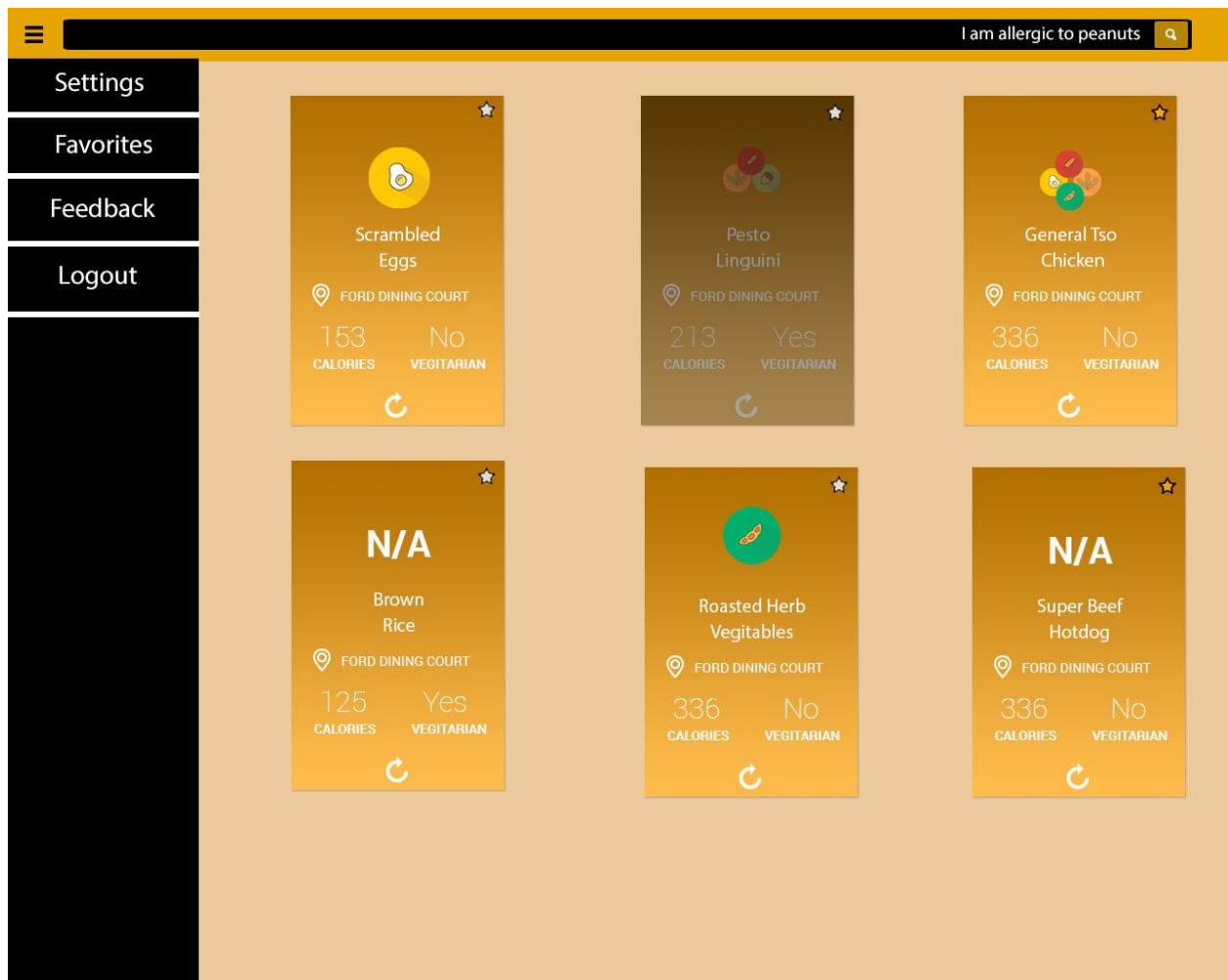
UI Mockups

Web Mockups

## Login Page

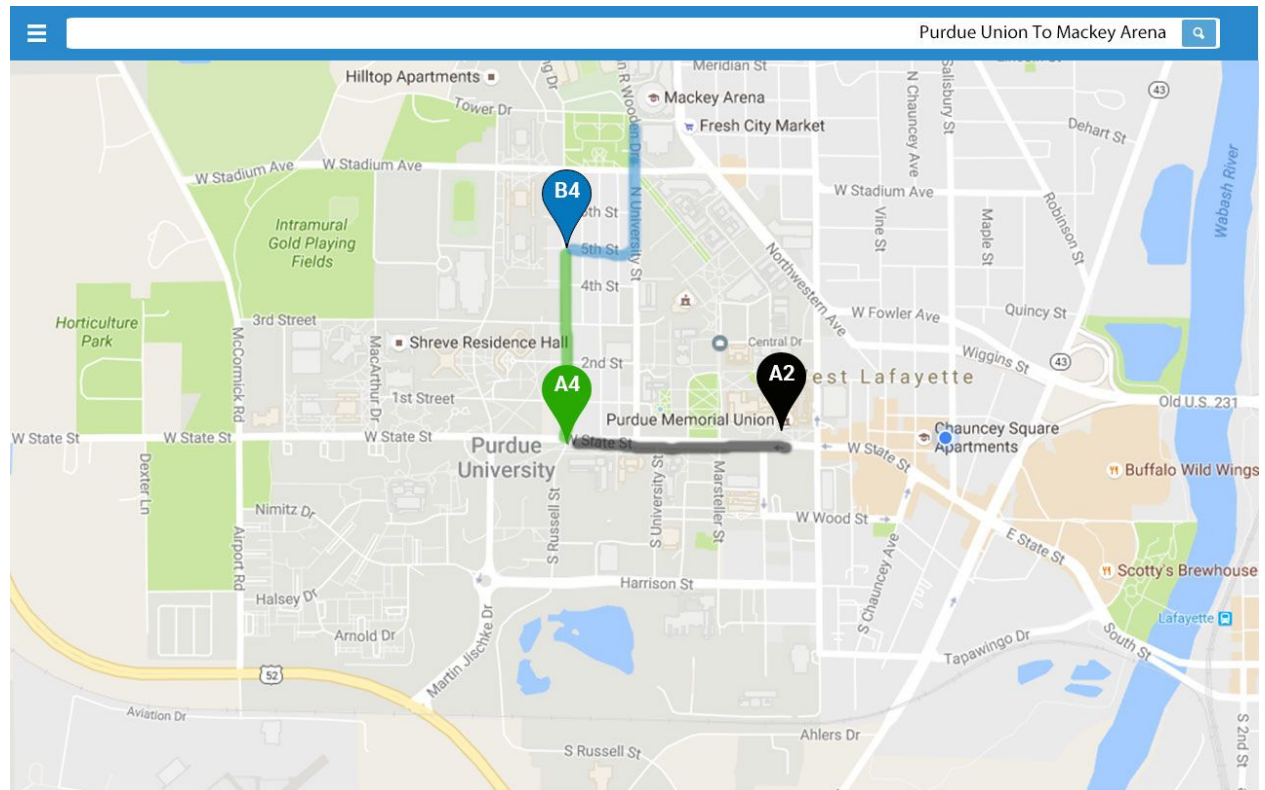
The image displays two side-by-side login and registration forms. The left form, titled 'Member Login', is set against a blue background and features a user icon. It contains input fields for 'Username' and 'Password' (masked with asterisks), a red 'LOGIN' button, and links for 'Forgot Password?' and 'Registration Needed?'. At the bottom, there are buttons for 'Sign in with Facebook' and 'Sign in with Purdue Login'. The right form, titled 'Register', is set against an orange background and features a pencil icon. It contains input fields for 'Username', 'Password', and 'Confirm Password', a red 'REGISTER' button, and a 'Member Login' link. A large black arrow points from the 'Registration Needed?' link on the left form to the 'Register' form on the right.

## Dining Page



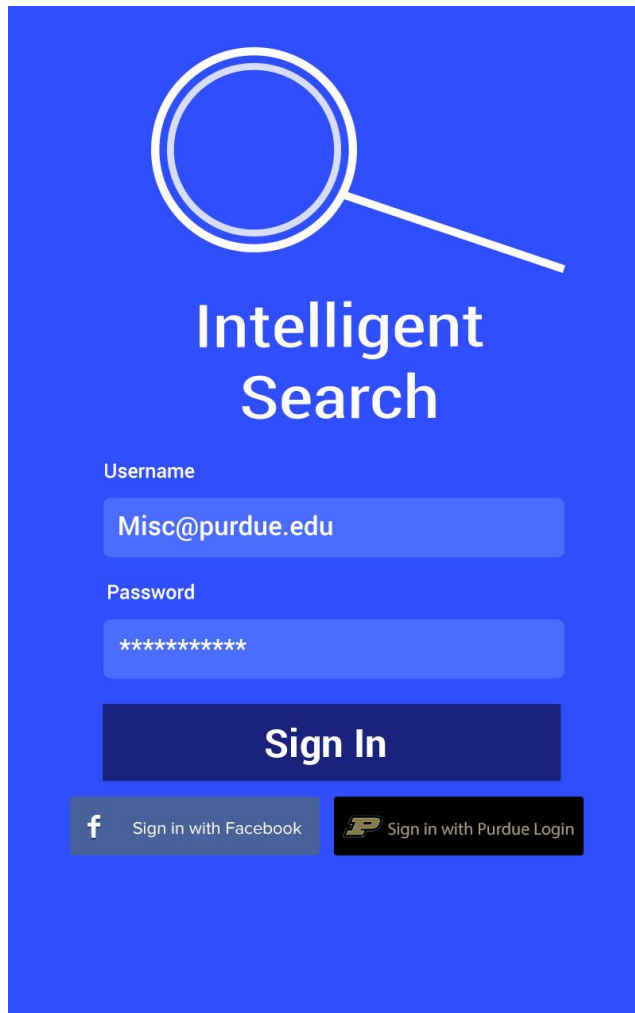


## Bus Page



## Android Mockups

## Login Activity



The login interface features a blue background with a white magnifying glass icon at the top. Below the icon, the text "Intelligent Search" is displayed in white. The form includes a "Username" field with the text "Misc@purdue.edu", a "Password" field with masked characters "\*\*\*\*\*", and a dark blue "Sign In" button. At the bottom, there are two buttons: "Sign in with Facebook" (with a Facebook 'f' icon) and "Sign in with Purdue Login" (with a Purdue 'P' logo).

Intelligent  
Search

Username

Misc@purdue.edu

Password

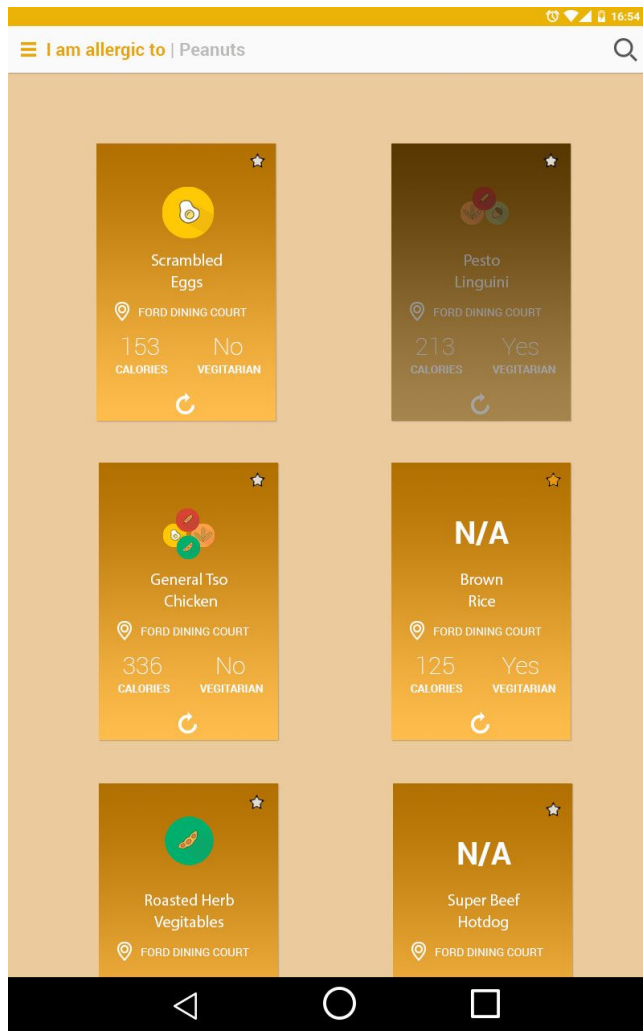
\*\*\*\*\*

Sign In

f Sign in with Facebook

P Sign in with Purdue Login

## Dining Activity



## Map Activity

