**Functional Requirements Document**

**They Will**

**Random Food Venue App**

**22/10/2019**

**Daniel Takyi**

**Jiwoun Kim**

**Mason Lane**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Description of Change** | **Author** | **Date** |
| 1 | First instance of our FRD | Mason Lane | 22/10/2019 |
|  |  |  |  |

CONTENTS

1 INTRODUCTION [4](#_heading=h.1fob9te)

1.1 Purpose [4](#_heading=h.vx1227)

1.2 Scope [4](#_heading=h.3fwokq0)

1.3 Background [4](#_heading=h.1v1yuxt)

1.4 References [4](#_heading=h.4f1mdlm)

1.5 Assumptions and Constraints [4](#_heading=h.1t3h5sf)

1.6 Document Overview [5](#_heading=h.2u6wntf)

2 METHODOLOGY [5](#_heading=h.19c6y18)

3 FUNCTIONAL REQUIREMENTS [5](#_heading=h.3tbugp1)

4.1 Context [5](#_heading=h.28h4qwu)

4.2 User Requirements [5](#_heading=h.nmf14n)

4.3 Data Flow Diagrams [6](#_heading=h.37m2jsg)

4.4 Logical Data Model/Data Dictionary [6](#_heading=h.1mrcu09)

4.5 Functional Requirements [6](#_heading=h.46r0co2)

5 OTHER REQUIREMENTS [6](#_heading=h.2lwamvv)

5.1 Interface Requirements [6](#_heading=h.111kx3o)

5.2 Data Conversion Requirements [7](#_heading=h.3l18frh)

5.3 Hardware/Software Requirements [7](#_heading=h.206ipza)

5.4 Operational Requirements [7](#_heading=h.4k668n3)

APPENDIX A - GLOSSARY 11

1. **INTRODUCTION**

## Purpose

Our application’s purpose is to present a list of random venues (and their location) to a user based on user curated or predefined parameters. We aim to leverage the Google Places API for results based on the parameters given. Name pending.

This document will elaborate on the functional requirements that exist in order to achieve this goal. It will also discuss the required architecture. An FRD’s goal is to clearly lay out what is needed, what must be avoided, what must be leveraged and what must be created. It will explain to readers what must be done to achieve our goals. Anyone reading this should in theory be able to use our gathered requirements to develop their own similar web app. This, in essence, is a step by step guide of what must be done to ensure our project successfully delivers a functional application.

## Scope

This document is designed to assist our team and other readers in understanding the function of our application - how it is intended to work, why it works, what it needs, etc. It will change as we grow our project.

This is accomplished by clearly describing the various needs our application will possess, and presenting potential solutions or architecture; then following up on the various functional needs presented by the architecture. People reading this should have a clear grasp of what we’re trying to accomplish, what we are leveraging, what we aim to leverage, and what is in our way. Sudden additions to our team should be able to read this and understand what were creating, and know what to avoid, monitor, improve or counter when developing.

## Background

We are team They Will. It consists of Mason Lane, Jiwoun Kim, and Daniel Takyi. Our responsibility is to develop an application or minimum viable product (MVP) by the deadline specified within our semester of ENSE 374. As such, we have decided on creating our proposed web-app. Currently, we have planned to split development responsibilities by the MVC, as there are three of us.

We are producing this document to aid the comprehension of the project’s scope, architecture and needs for our team and others.

## References

<https://github.com/jiwoone/ENSE-374-Project/tree/master/Documents/UMLDiagrams>

<https://github.com/jiwoone/ENSE-374-Project/tree/master/Documents>

<https://github.com/jiwoone/ENSE-374-Project/tree/master/Milestone%201%20report%20out>

## Assumptions and Constraints

### Assumptions

This document assumes we will have access to easily leverage the Google Places API.

### Constraints

Constraints are boundary conditions on how the system must be designed and constructed. Examples include: legal requirements, technical standards, strategic decisions.

* Some degree of an MVP must be achieved by the end of our semester. Time constraint. Failure or low quality results by the deadline could result in lower marks.
* Our system mustn't infringe on any Google properties by using or paying for them incorrectly.
* We would prefer not to use a database, but will if needed.

## Document Overview

The contents of this document are meant to clarify the current planned design, needs, limitations and precautions for the project. We will discuss how we find functional requirements, what they are, how our design will form to accommodate them, and what these accommodating functions will need to work. We will also detail the importance and guiding principles behind our applications security, reliability, recovery as well as other general needs like hardware requirements. Finally, we will also delve into more general requirements like error handling and form validation.

1. **METHODOLOGY**

This document serves as a continuously changing outline of our application’s planned architecture. It will evolve as we continue to develop our app and grow a greater understanding of what we are aiming to create. This document will be modified as we reach, solve, or re-plan milestones based on varying development hurdles. Currently, the document outlines our idealized architecture, or how we’ve envisioned it. Hurdles may require us to make drastic changes to our prototypes. This document will be a reference document as we design our application.

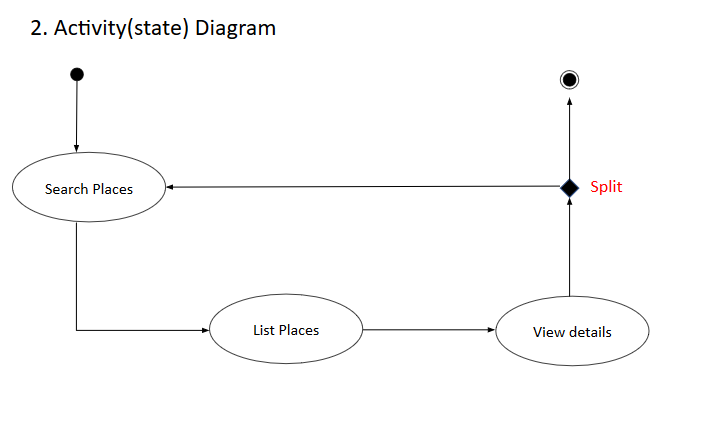
Requirements may be discovered as we continue to develop and learn about the nature of this kind of application and the Google Places API. Initial needs are based off of past knowledge and logical conclusions. Some assumptions must be made until we have developed a better understanding of the design.

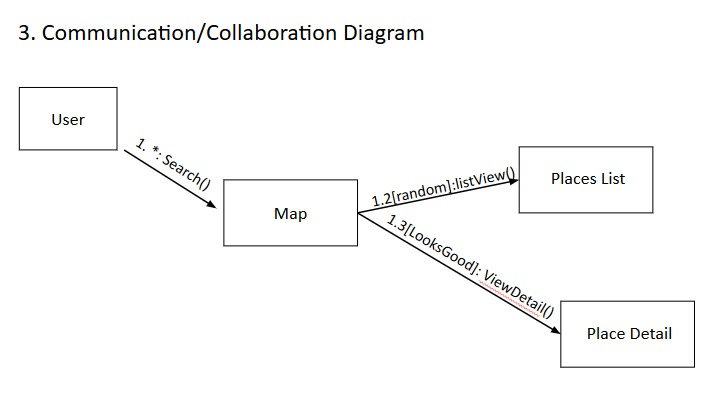
1. **FUNCTIONAL REQUIREMENTS**

## Context

[Provide a context diagram of the system, with explanations as applicable. The context of a system refers to the connections and relationships between the system and its environment.]

We currently do not have a dedicated context diagram, however our activity and collaboration diagrams may suffice for now.

**

**

## User Requirements

As an MVP, we must at minimum start with pre-queried results for the regina area, and quickly expand to allow custom areas and ranges. Users should be able to select their desired distance.

Further, we plan to enable search functions to allow user’s a broader range of uses - not limited to food locations

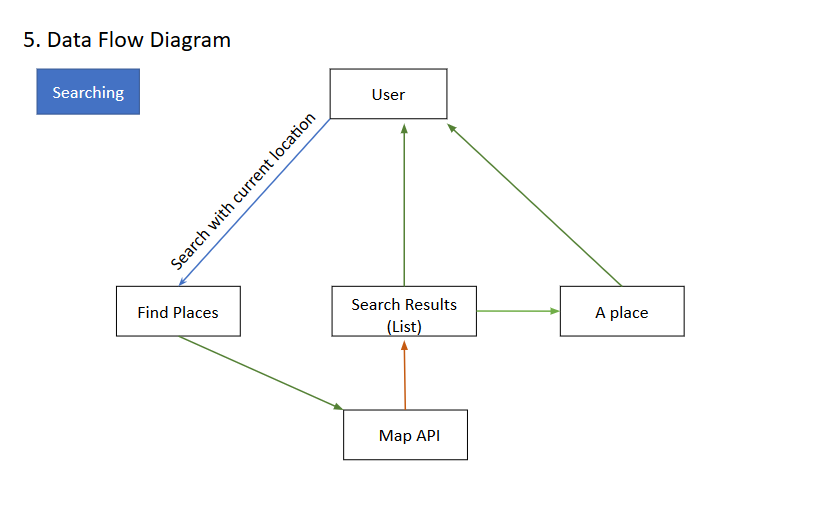
It has to be handy - something someone can use moments before getting to their garage or parking lot. So while starting with a browser based app, it still needs to be mobile friendly.

We share some concerns over the difficulty in implementing queried map results

We assume we might need some kind of access to google’s assets. Likely needing a google\_maps\_api key and the Google Places API.

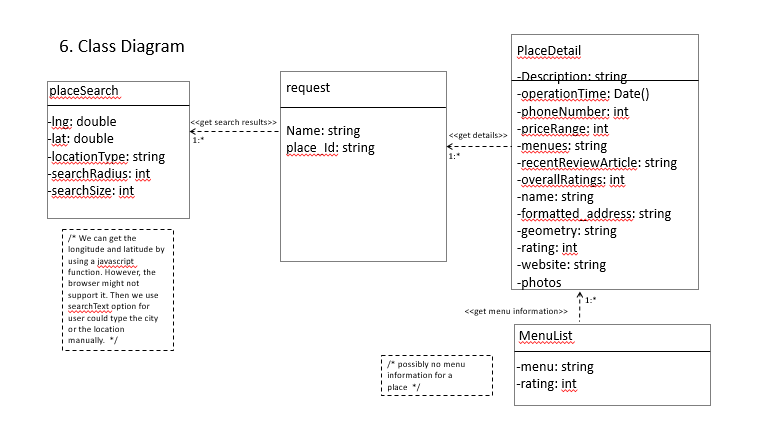
In short, a user should be able to choose between two options - one that begins a search using predefined parameters, and a second that allows custom options. Choosing the latter must allow users to curate their own search. They will choose how many results they seek, the distance allowed from their selected location, and possibly the theme or keyword of the kind of location they are looking to explore. We will use these parameters with the Google Places API to garner results. These results will then be displayed (preferably alongside a map of the area) for the user to view and select. Selecting a result must provide greater detail of the venue (menu, website, contact information, rating, etc).

## Data Flow Diagrams



## Logical Data Model/Data Dictionary

This class diagram is subject to change. (edited 12/11/2019)



## Functional Requirements

### Functional Requirements Group 1 - Random List of Venues

|  |  |
| --- | --- |
| **Section/ Requirement ID** | **Requirement Definition** |
| FR1.0. | A user must be able to view a list of varying size consisting of random venue locations within a given theme. These results are tailored for a user’s unique search specifications. |
| FR1.1 | The user should be able to select a quick search (using predefined settings), or choose to curate their own parameters. |
| FR1.1.1 | The system shall inform the user of its default parameters and attempt to interface with Google Maps using the Google Places API |
| FR1.1.2 | The system shall display a list of parameters for a user to choose from. They should be allowed to choose their location, how far results can be, the type of venue, and the number of random results. Once the parameters are submitted, the app should attempt to interface with Google Maps using the Google Places API and given parameters. |
| FR1.2 | The system shall use Google Places API results (from previous parameters) to display the relevant information and possibly a viewable map as well. Results are listed for the user to access and read further into |
| FR1.2.1 | Upon selecting a result, users should see more detailed information of the venue (ratings, menu, etc). They should be able to reach the venues menu, website or other contact information if any is available.  These results should be taken from the Google Places API and appear similar to how Google Maps displays location details. |

### Functional Requirements Group 2 - Random List of Venues Without API

|  |  |
| --- | --- |
| **Section/ Requirement ID** | **Requirement Definition** |
| FR1.0. | A user must be able to view a list of varying size consisting of random restaurant locations. These results are tailored for a user’s unique search specifications. |
| FR1.1 | The user should be able to select a quick search (using predefined settings), or choose to curate their own parameters. |
| FR1.1.1 | The system shall inform the user of its default parameters and use a premade database of locations within the regina area to generate results based on the parameters |
| FR1.1.2 | The system shall display a list of parameters for a user to choose from. They should be allowed to choose how far results can be based on predefined distances, and the number of random results. Once the parameters are submitted, the app should attempt to interface with our database for results using the given parameters. |
| FR1.2 | The system shall use DB results (from previous parameters) to display the relevant information and possibly a viewable map as well. Results are listed for the user to access and read further into |
| FR1.2.1 | Upon selecting a result, users should see more detailed information of the venue (ratings, menu, etc). They should be able to reach the venues menu, website or other contact information if any is available.  These results should be taken from our database and appear similar to how Google Maps displays location details. We will likely use google maps results to write the database information manually |

### 

1. **OTHER REQUIREMENTS**

## Interface Requirements

A user should be able to interface with the application through a web browser. They will have various popup menus to manipulate and view search results.

### Hardware Interfaces

Basic mobile devices and PCs should be able to interface with our application through appropriate web browsers. Our app will ideally be made to accommodate varying screen sizes.

### Software Interfaces

Our application will interface with the Google Places API for geographical data (lat, long) and relevant search results. External owner: Google.

### Communications Interfaces

We will need to access a devices location, this may involve their communication interfaces.

## Data Conversion Requirements

We do not expect to perform and data conversion. Results from the API are sent in JSON.

## Hardware/Software Requirements

Our application must be supported by contemporary web browsers that can understand HTML, CSS and JS. Our site will be hosted on one of our University of Regina directories. We will need the Google Places API to send search parameters and receive location results.

## Operational Requirements

### Security and Privacy

A. Consequences of the following breaches of security in the subject application:

1. Loss or corruption of data

* Expect minimal consequences besides impairment of convenience. No

beaches in privacy

1. Disclosure of secrets or sensitive information

* NOTE: as of dec 4, 2019, we have be unable to secure the google maps API, as our hosting site (uregina) is not secure. By not being https, we cannot run our geographical queries effectively and must therefore do them from the source code. As such, we cannot lock our api to a specific web link. We are currently finding alternatives.
* No foreseeable reason for a user’s location to be revealed to attackers, though we will endeavor to see how an attacker might manipulate google API requests in order to see the results of other traffic.
* Serious risk of API abuse. Need to find a way to securely hide our API from the average user. Malicious parties could easily find our API and use it beyond our monthly limit – this would result in unwanted fees.

1. Disclosure of privileged/privacy information about individuals

* No foreseeable reason for a user’s personal information to be revealed to attackers, though we will endeavor to see how an attacker might manipulate google API requests in order to see the results of other traffic.

1. Corruption of software or introduction of malware, such as viruses

* This constitutes the most likely danger in our app. We will need to make sure attackers cannot manipulate input forms for malicious manipulation of our software.

1. State the type(s) of security required. Include the need for the following as appropriate:
2. Access by user role or types.

* We may not need user roles for this app
* If we wanted a way to view trends and other metrics, we would then implement an admin system

1. State access control requirements by data attribute. For example, one group of users has permission to view an attribute but not update it while another group of users has permissions to update or view it.

* No known access control requirements, unless we extend the scope of our app to record analytics like trends and traffic.

1. State access requirements based on system function. For example, if there is a need to grant access to certain system functions to one group of users, but not to another. For example, "The system shall make Function X available to the System Administrator only".

* No known access control requirements, unless we extend the scope of our app to record analytics like trends and traffic.

1. State if there is a need for certification and accreditation of the security measures adopted for this application]

* If we decide to use HTTPS we will need to certify our website

### Audit Trail

[List the activities recorded in the application’s audit trail. For each activity, list the data recorded.]

[WIP]

### Reliability

If our app fails to perform its function in general, users might find themselves annoyed, or still indecisive. Users should be able to find alternative solutions to the problems we aim to fix, however.

Our app should be reliable to ensure a fun user experience.

Search parameters \*must\* not provide broken results. We do not want users to waste gas driving to the wrong location, or finding themselves stuck on a highway. Distance options must work almost all of the time, or not be implemented. If distance sliders are not included, results must be weighted by distance

Currently does not reliably fetch all possible results (unreliable and variable # of results)

### Recoverability

Our app is not a vital app and serves more to accommodate novelty and convenience. It does not need immediate restoration, but shouldn’t stay down long enough to become a major inconvenience. It is unknown how long is too long, but it would depend on how often the app is used - and how much it serves a user on a daily basis.

Unless we implement an account system or other features, we do not expect to use any database information. We will however rely on Google’s API, which we expect will not stay down long if the worst case occurred.

If we cannot leverage the Google API, we might implement a database with pre-recorded location information of various Regina locations. If this happened, our scope would be too small for any consequences from a corrupted database to matter.

Given the simplicity of our application, it should not be too hard to restore our processing site. However, the need for restoration is not urgent given the scope of this app.

Caveat: The urgency of restoration for any recoverable part of our application will depend on user quantity and use frequency. Current estimates are based on the assumption that only a small classroom of individuals will be aware of the app, but have made room for the possibility of its sudden popularity.

### System Availability

Our app should be running almost daily, especially during breakfast, lunch and dinner. Downtime for maintenance and updates should be reserved for midnight. Depending on the app’s popularity around the world, the timezone for this midnight would change. If central Canada used our app the most, we would reserve downtime for midnight in central Canada.

Expected peak times: 6:00 - 9:00 am, 11:00 am - 2:00 pm, 4:00 - 9:00 pm

### General Performance

Specific performance requirements, related to a specific functional requirement, should be listed with that functional requirement.

Required response time is unknown, but results as fast as google maps queries would be ideal.

Errors should almost immediately inform the user.

If our web service extended beyond academia, we would expect frequent traffic given its free and novel purpose. Exact numbers are unknown. We expect no more than ~100 visits per day at peak, given the size of our class. This is assuming all students that are aware of our app frequently use it.

### Capacity

[List the required capacities and expected volumes of data in business terms. Do not state capacities in terms of system memory requirements or disk space—if growth trends or projections are available, provide them]

[WIP]

Google places API charges $25 for every 1000 clicks. This means our website will be limited to 1000 clicks which should be enough to demo our MVP by the end of the semester.

Google places nearby search results only allow a total of 60 results.

### Data Retention

Currently, data should not be retained for longer than the current session. May add location saving/setting memory in future iterations. If data is retained, it would consist of saved search parameters, or favorited locations. Might use cookies for this.

### Error Handling

We will need error handling for our search boxes (keywords, etc). The rest (radio buttons, etc), should not incite errors.

Google places API also handles much of the error handling on their side, so we may not need to worry about poor user input (though may still implement some kind of system to be more user friendly).

When entering a location other than a user’s default, they may need to accurately spell the location’s name (API may help with inferencing a users search intention).

We will likely use JS for error handling.

We are currently considering limiting user input to preset options, removing some of the error handling requirements. Examples include, option sliders, check boxes, etc.

### Validation Rules

We may require the user to choose between their location or a preset location. We may also have max search sizes.

We will likely handle form validation through JS.

### Conventions/Standards

We intend on following css and markup standards, using validation to prove. Google may also have their own conventions/standards regarding the use of their API.**APPENDIX A - GLOSSARY**

API: a set of functions and procedures allowing the creation of applications that access

the features or data of an operating system, application, or other service.

CSS: Cascading Style Sheets, used to format the layout of Web pages.

FRD: Functional Requirements Document

Google Places API: “The Places API is a service that returns information about places

using HTTP requests.” <https://developers.google.com/places/web-service/intro>

HTML: Hypertext Markup Language, the standard language used to create documents on

the World Wide Web.

JS: JavaScript, an object-oriented computer programming language commonly used to

create interactive effects within web browsers.

JSON: JavaScript Object Notation, a lightweight format for storing and transporting data.

MVC: Model - View - Controller, an “application design model comprised of three

interconnected parts. They include the model (data), the view (user interface), and

the controller (processes that handle input). https://techterms.com › definition › mvc

MVP: Minimum Viable Product (The smallest possible product whose function is still

viable for users)