**Phood Buddy**

**High Level Design**

**COP 4331C, Spring, 2016**

Team Name: Phood Buddy

Team Members:

* William Funk
* Evan Glazer
* Timothy Flowers
* Jorge Rodriguez
* Lyudmila Sandomirskaya

Modification history:

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Who | Comment |
| v0.0 | 03/01/16 | L. Sandomirskaya | High level design diagram  Component diagram |
| v1.0 | 03/02/16 | J. Rodrigez | Design issues, Architecture Overview |
| v 2.0 | 03/03/16 | L. Sandomirskaya | Architecture description |

**Contents of this Document**

High-Level Architecture

Design Issues

**High-level Architecture**

Our original architectural decision was a Client-Server design, which would work very well for a web application, the components of the client-server web architecture would be front-end, web service, and database. Another possible architecture for our system could’ve been Repositories design: Firebase database would be the central data store and Android app, Windows app , and Web app would be the data-accessing components. After considering a few possible solutions our team decided on Layered Architectural Style for our software application.

The main advantage of layered design that’s most useful for our system is the fact that layered are independent and any internal changes within layers will not affect other layers as long as their interface remains the same, and even if the interface changes, only the layer either directly below or above will be affected.

The disadvantage of the layered architecture is a performance cost, since layers have to interact between each other passing data.

The system is 3 tiered in terms of development. There are 3 major development routes each with their own associated 3 tier blocks. Our 3 development routes are Web, Android, and Windows.

Web starts with its frontend, tailored to fit web browsers on different platforms, especially iOS web screens more so than using Bootstrap to provide responsive layouts for multiple screen types. The front-end communicates to a Logic Data layer which is designed to organize query calls (including external APIS) that users ask for or writing data to the database, sending this information through AJAX calls. The AJAX call will fuse the communication between the Frontend and the BackEnd/ PHP component of logic layer. This will secure any information being inputted into the Firebase database, also preventing users from accessing data not associated with their functionality. Firebases own internal backend security system will handle the rest from there internally and allow for a place to store data.

Android starts with its FrontEnd User-interface which communicates with the Android Logic layer. This layer is responsible for handling the query calls users ask for and writing data to the database. Layer must also be able to handle the external API calls that will be used to populate the frontend UI with requested information (Recipes, Deals, Etc.). Information relevant to our database will be sent directly to Firebase, making full use of Firebases built in security measures to add an extra layer of security between the user and the database.

Windows follows a very similar overall design as the Android development path. IT will have a FrontEnd UI which communicates with its Logic layer where it will perform the query calls for the firebase database and the external API calls and be able to write to the database. This information is processed again by the Firebase security layer and inputted into the database.

The external libraries (FatSecret, Deals API, etc.) will be accessible and partially configured in their respective Logic layers to optimize usage for each platform.



High Level Architecture Diagram



High Level Component Diagram

**Interface Description:**

Firebase Php interface is a 3rd party API that allows our php function to communicate with Firebase database.

Firebase Java interface is an API that allows Android app to communicate with Firebase database.

Firebase Sharp interface is an API that allows Windows app to communicate with Firebase database.

Ajax provides an interface between components written in Javascript and Php functions running on the back end – this interaction allows to serve user requests related to retrieval information stored in our database, authentication requests, and requests requiring to perform calls to FatSecret API (our 3rd party recipe provider), FitBit API, and Walmart API.

All three parts of our application – Web, Android, and Windows will interact with Walmart API will to get information about current deals on groceries and prices for items from the grocery list that customer can create using our software application.

FitBit API will be used to extract such data as activity log, set goals, weight, etc, to be used in user’s profile. All three parts of our software application will communicate with FitBit independently.

Google, Twitter, and Facebook APIs will interact with our application for user authentication.

**Design Issues**

1. Reusability :

Our team decided that none of the components from this application will be used in our further projects, so reusability is not our concern for this project.

1. Maintainability:
   1. API Assurance – we are relying on 3rd party APIs for some functionality of our system, so if API calls format changes, or APIs stop being accessible our system will partially lose its functionality. In such case we’ll need to find an alternative API and rewrite our calls to these APIs.
   2. Using a 3rd Party Database (Firebase) – we are relying on the 3rd party database – so in case the Firebase goes out of business we’ll have to migrate our data to a new database platform and rewrite all the functionality responsible for storing and retrieving data.
2. Testability

3.1 Difficulty Testing Accuracy of Algorithms That Tailor to Health Needs – we’ll parse recipes for ingredients that can cause allergies in case if user specified their allergy information in health profile.

3.2 Multi-Platform Testing - we need to test for web, Android, and Windows apps

1. Performance

4.1 Dependencies on 3rd party database systems and 3rd party APIs which can slow the response time and have limits on number of calls to them per day. To prevent APIs cups on number of calls we’ll get needed data in a batch versus making multiple individual calls.

1. Portability

5.1 Changing Databases and Resulting Effects

1. Safety

6.1 Misrepresented Data Retrieved from External sources – for example some recipes may contain harmful ingredients – we’ll try to prevent it by using credible sources and parsing recipes for any food allergens from users’ health profile.