Store Selector

Team: Blair Billings
Timothy Kalpin
Kurt Kohl
Chris Morgan
Kerrick Staley

Client: Google
Advisor: Manimaran Govindarasu

Version	Editor	Date	Peer Reviewers	Notes
0.1	Kurt Kohl	March 22, 2012		Draft 1
1.0	Chris W. Morgan	April 23, 2012		Final Version

Table of Contents

Section 1 - Introduction

1.1 Document Purpose

The purpose of this document is to define the scope and implementation of the Store Selector project. Included in the document is a table of revision tracking, because the document will change through several iterations. Visual elements are also incorporated to clearly display concepts.

1.2 Acknowledgements

Our senior design team would like to thank the hard work and efforts of Muthu Muthusrinivasan of Google. We appreciate the opportunity we were given, as well as all of his help on the project. Our team would also like to thank our advising professor, Dr. Manimaran Govindarasu, for his wise counseling, thoughts, and direction during the entire planning and implementation process.

1.3 Problem Statement

There are many ways that people gather information about shopping deals and pricing information, and there are many ways that people purchase these items that they are interested in. However, there is not one source to view, store, and remember this information that is convenient and easy to use. Businesses desire their information to be seen by all, but there is no guarantee that people who don't read the newspaper will see their print ads. There is no way to guarantee infrequent internet users will see web advertisements that don't make it into a version of printed media. There is also no existing system that allows consumers to view pricing information for stores in their local community.

In general, there are too many gaps between buyers of products and businesses that want to get their deals and prices out there for all to see.

1.4 Terms and Abbreviations

[No Major Terms or Abbreviations]

1.5 Operating Environment

Our system will need to be hosted on a platform that can run all of our code and each module that we create without any issues. This web application will be public, but each User's personal data will need to be protected and privatized securely. Our host will also need to be able to handle several users concurrently. The platform that we use should also be able to handle updates to our code seamlessly. Our system will be very modular and compatible with many platforms, however. This should allow our system's components to get a greater portion of computational resources.

1.6 Target Users

This application will have a very large range of Users. Ideally, anyone who wants to receive information about deals, weekly ads, or general pricing information from businesses that they regularly (or infrequently) purchase items from could use this application. Realistically, Users will have to access the internet quite frequently to use this application. This factor probably narrows the range of Users to those people that frequently use their computers, smartphones, or other internet devices.

There will also be administrative Users who can go into the application and monitor inappropriate behavior, or abuse of the application. This will provide a better experience in the application overall.

1.7 Assumptions and Limitations

1.7.1 Assumptions

- Data on deals and products will be provided to us
- Users desire a centralized location for prices, deals, and product advertisements
- There will be deals that interest the Users of this application
- There will be a way to enter deals on the application

1.7.2 Limitations

- Competing deal advertising products
- Limited means to share the application with people
- Lack of time to complete all desired features
- Resources that we need (such as server space) may not be available

1.8 Deliverables

- Database (for storing User data)
- Web Interface
- Mobile (Android) Interface
- Voice Interface (for inputting shopping list)
- Web Entry Page/Email scrubber

This web application will follow a pretty standard Model, View, Controller pattern of design. It will include a database model, a series of web views, a mobile (Android) interface, and several controllers to handle web services. To make our system more modular, interactions between the layers of the application will be very standardized and documented. There will be a rich and easy-to-use web interface presented to the User as well.

Section 2 - Approach and Product Design Results

2.1 Approach Used

2.1.1 Design Objectives

For this project, the system we produce has to be highly modular and scalable so it can be easily adapted to new uses and more users. Our design centers on this objective.

2.1.2 Functional Requirements

1. The system shall allow users (e.g. shoppers) to find the total price of items they are seeking, using their Android device.
    a. The system shall accept a product name via voice and/or keyboard input.
    b. The system shall match this input to its database of known products.
    c. If there are multiple matching products, the system shall present a list for the user to select from.
        i. For keyboard input, this list will be generated as the user types (i.e. it will autocomplete the input).
    d. The system shall present a means of adjusting item quantity.
    e. The system shall build an on-screen list of items during this process.
    f. The list shall allow deletion of entered items.
    g. Once the user has completed the list, the system shall calculate the total price of the items at various nearby stores, and display this information on a map.
    h. The system shall allow saving, opening, and modification of product lists.
2. The system shall allow users (e.g. store managers) to submit price information for their products via a web interface.
    a. The system shall allow users to upload a price table in .odt, .xls, or .csv format or to link to a Google Doc containing price information.
    b. The system shall allow users to indicate which columns in the table indicate the name and price of each product.
    c. The system shall allow users to view, search, and edit uploaded data.

2.1.3 Non-Functional Requirements

1. The user interface shall be easy to use and aesthetically pleasing.
2. The backend shall process 120 queries per minute, and be able to scale to process 10,000 queries per minute with sufficient hardware.
3. The code in the final product shall be modular, readable, and well-documented.

2.1.4 Testing Approach

- To test the frontend for shoppers, we will find at least 10 volunteer testers, install a beta version of the app on their phones, and ask them to use the app upon their next shopping trip to the ISU Bookstore. The beta app will log detailed information about which products they query and how quickly/easily they do so. We will inform the volunteers that the app will log their behavior.
- To test the frontend for store owners, we will ask the ISU Bookstore manager to upload pricing information for all the products in the bookstore. The upload form will also monitor how quickly/easily the store owner interacted with it.
- To test the backend, we will write a script that initiates 10 simultaneous queries every 5 seconds for 5 minutes, and monitor the response times and server load.

2.2 Detailed Design

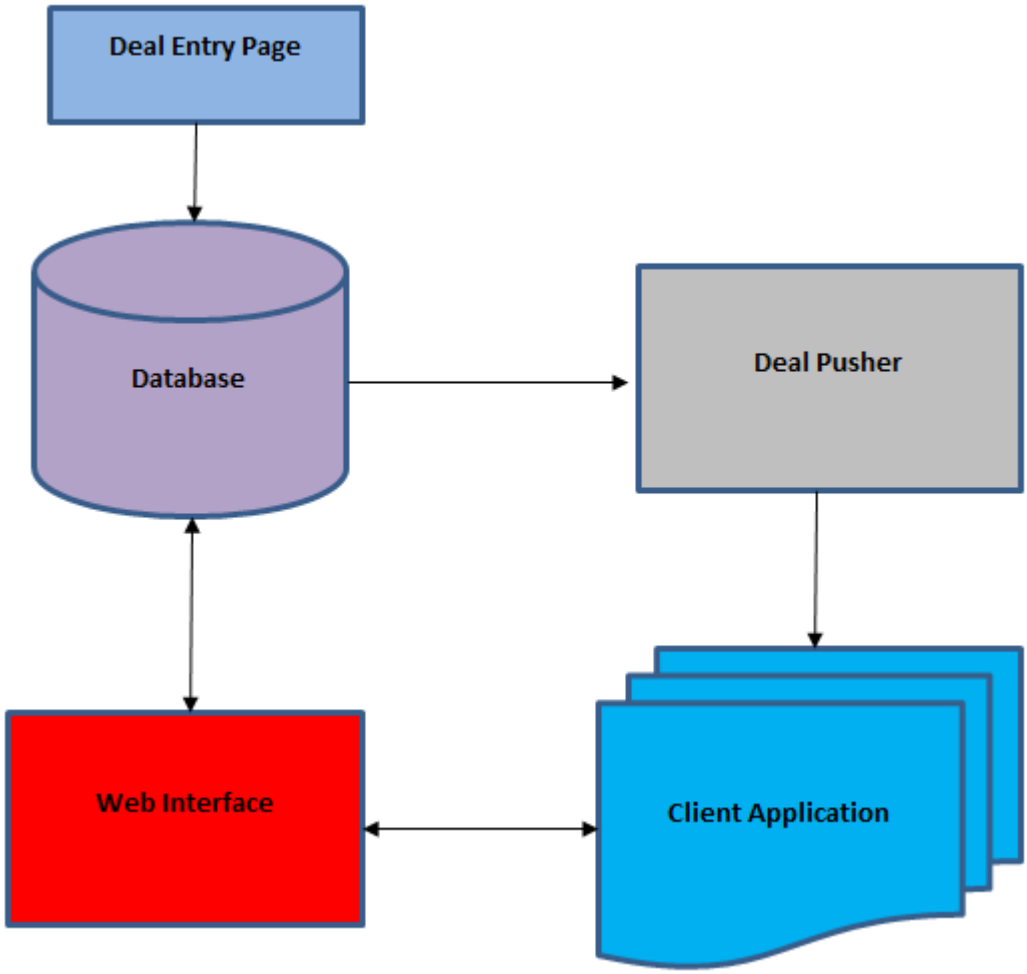

Figure 2.2.1.1 - Web Service diagram

2.2.1 Data Web Service & Database

The data web service manages user interactions with the database. It will take care of requests for information, formatting and relaying those requests on to the database, then formatting the database results. The web service will also keep track of login information and saving individual user's preferences.

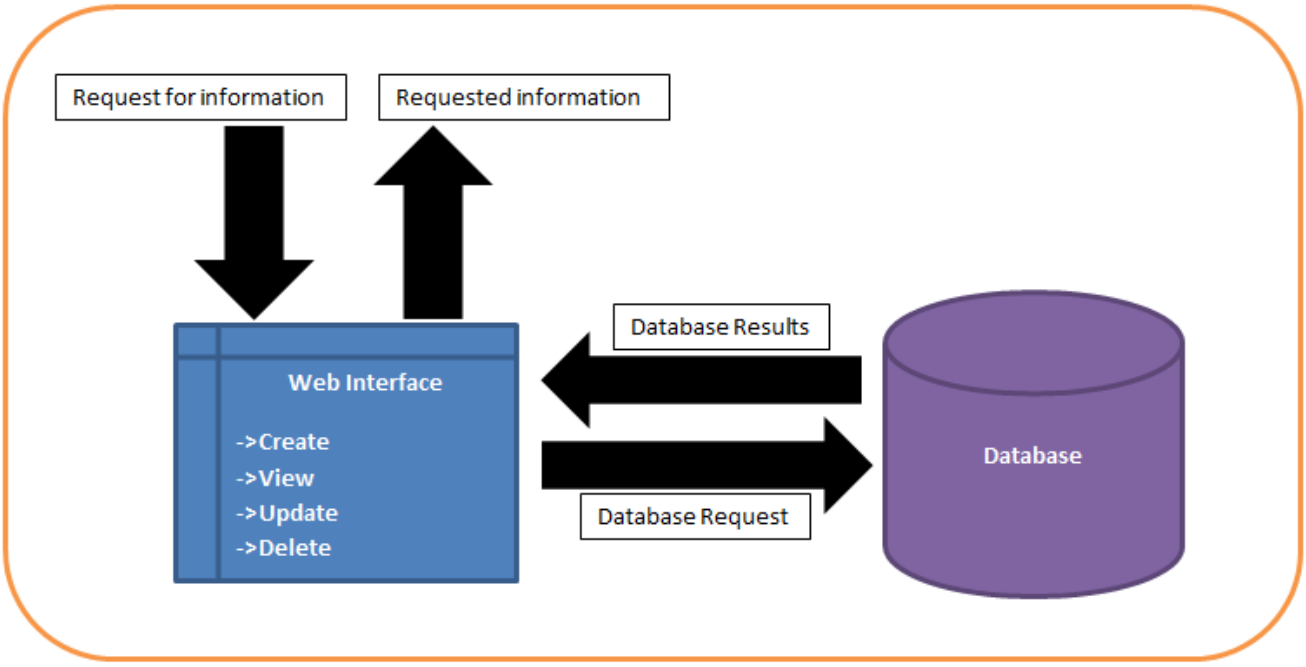

Figure 2.2.1.2 - Web Service diagram

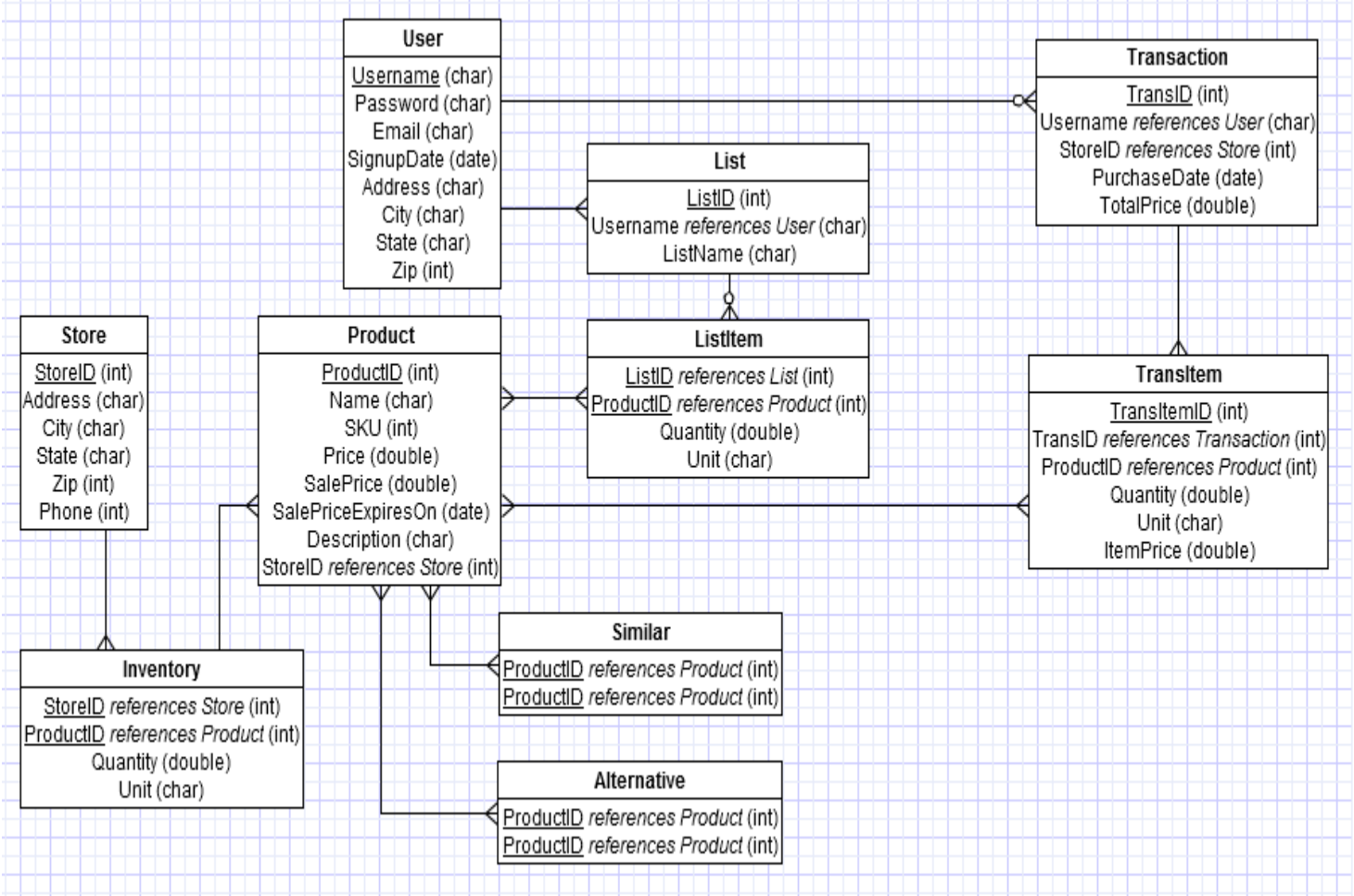

Figure 2.2.1.3: Database Schema

2.2.2 User Interface

Our product will be used by customers via a web page interface targeted both for desktop and mobile browsers, as well as a mobile interface for the Android platform. Through these interfaces, all of our intended functionality will be available to our users: signing in, browsing deals, saving deals, viewing saved deals, and changing preferences.

Section 3 - Future Work

3.1 Community-driven Suggestions

The eventual goal of the Similar and (to an extent) the Alternative relations is that they would be driven by community purchase histories. For example, if many people purchase a toothbrush and toothpaste at the same time, and you purchase a toothbrush or add a toothbrush to your shopping list, the system will suggest that you purchase toothpaste also. Also, if you have a toothbrush and/or toothpaste in your purchase history, the system will be able to notify you if deals on toothbrushes or toothpaste are available.

For our project, we will be adding similar relations to our database manually, as a community-driven system is out of our scope.

3.2 Bundled Offers

Bundled offers (e.g., buy one get one free promotions, etc.) are something we would be easily able to add to the system in future due to the methodologies we have chosen. It would require a trivial addition to the database schema, but the additional data handling code required puts it out of scope for our project.

3.3 Additional Recommendations Regarding Project Continuation or Modification

- Wiki style deal entry for users
- Deal rating system
- Multi-platform support.
- Provide an API to allow access to users' saved deals.

Section 4 - Closure Material

4.1 Project Team Information

4.1.1 Client

Google
Muthu Muthusrinivasan
muthup@google.com

4.1.2 Advisor

Manimaran Govindarasu
3227 Coover
Ames, IA 50011-3060
515-294-9175 (Office)
gmani@iastate.edu

4.1.3 Team Members

Blair Billings
Computer Engineering
bbill7@iastate.edu

Timothy Kalpin
Computer Engineering
trkalpin@iastate.edu

Kurt Kohl
Computer Engineering
kdkohl@iastate.edu

Chris Morgan
Computer Engineering
cwmorgan@iastate.edu

Kerrick Staley
Computer Engineering
kerrick@iastate.edu

4.2 Closing Summary

This project plan demonstrates our proposal for the Store Selector and the implementation thereof. The product has been designed such that features can be added without breaking the existing implementation.

The design has been broken down into three modules for ease of implementation and use; additional modules may be added at a later date. The implementation will utilize free tools and languages, but we will require dedicated server systems to handle the hosting, especially considering the potential user scope of any Google product.