Store Selector

Dec1206
Client: Google

Chris W. Morgan
Blair M. Billings
Kerrick A. Staley
Timothy R. Kalpin
Kurt D. Kohl

Problem Statement

It's very easy to compare prices across multiple sites on the internet, but this is far more difficult with physical and local stores. Store Selector will allow customers to easily locate the best prices on the products they plan to purchase at such stores, and provide other features to optimize the shopping experience.

Market Survey

- Account Based
  - LivingSocial
  - Groupon
  - ScoutMob
- Coupon Apps
  - GeoQpons
  - The Coupon App

Our application brings together select functions of each site mentioned above and gives the user an easy-to-manage account that responds to their shopping habits and allows them to compare prices across local stores.

Group: Dec1206
Project: Store Selector
Client: Google

Concept

upload product pricing and discount information

compare stores and products, view discounts, manage shopping list

High-Level Description

- Consumer Web Interface
  - Account-based
  - Maintain shopping lists and transaction history
  - View discounts and compare store prices
- Consumer Android Application
  - Includes features of Consumer Web Interface in a convenient Android Application
  - Allows access to GPS and voice functionality

High-Level Description

- Store Manager Interface
  - Web application
  - Store managers will enter and maintain product data via this portion of our product
- Scraper
  - Dumps product descriptions and pricing from the Iowa State Bookstore; used for populating our database for initial testing

Functional Requirements

- Allow store managers to enter and maintain pricing data
- Allow customers to create and maintain item lists via Web Interface
- Allow customers to enter input via the voice interface on Android devices
- Allow customers to perform same functionality as the Web Interface on an Android device, plus GPS and voice

Non-functional Requirements

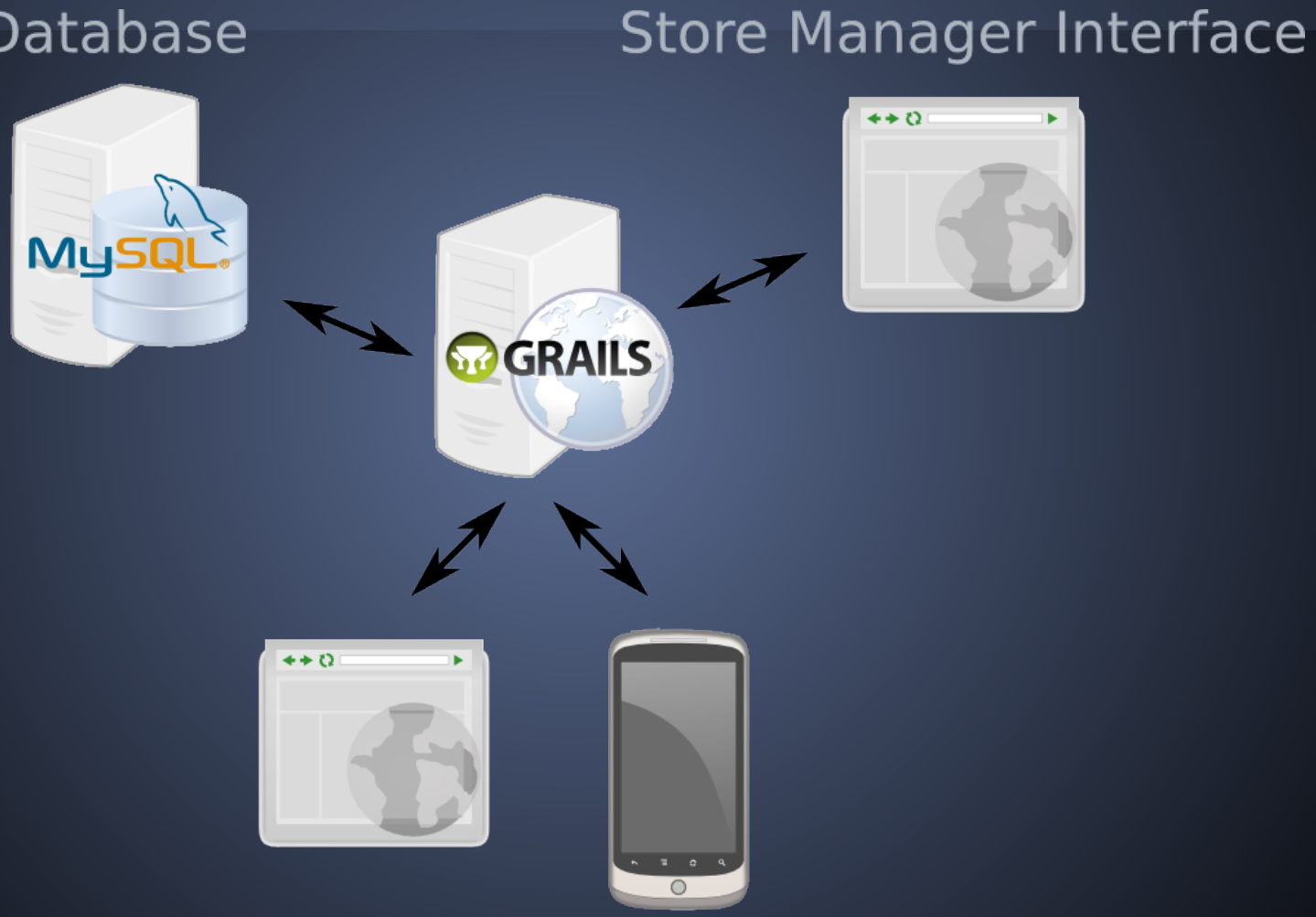
- Be easy to use and aesthetically pleasing
- Ensure price data is accurate
- Be quick and responsive
- Provide clear feedback if not enough data is available to answer query
- The code in the final product shall be modular, readable, and well-documented

Constraints and Considerations

- Time constraints
  - Complete all iterations and implementations by the end of the semester
- User's mode of use
  - Android smartphone
  - Desktop computer webpage

Group: Dec1206
Project: Store Selector
Client: Google

System Diagram



Design Decisions: Architecture

Issue:

How should the overall system be structured?

Factors affecting issue:

- The design requirements are well-matched to the problems that modern web frameworks address (e.g. managing complex data, rendering the data to clients over the internet)
- A web framework yields better-structured code and reduces unnecessary work (compared to an ad-hoc setup using CGI) when used for its intended purpose

Design Decisions: Architecture

Decision:

- Straightforward: Design the system around a web framework
- The web framework will prescribe the overall architecture of the application; most frameworks follow the model-view-controller pattern

Design Decisions: Framework

Issue:

Which web framework should we use?

Alternatives:

- Django
  - Python-based
  - Emphasizes automation and avoiding duplication (DRY)
  - Uses the model-view-controller pattern
- Grails
  - Groovy-based
  - Emphasizes convention-over-configuration
  - Also uses the model-view-controller pattern

Design Decisions: Framework

Factors affecting issue:

- Python has a large ecosystem around it, and Groovy can use Java libraries and therefore leverage the Java ecosystem, so library support wasn't a concern
- Two group members had experience with Grails, and all had experience with Java (Groovy is similar to Java)
- One group member had experience with Django, and some (but not all) group members had Python experience

Decision:

- Use Grails, since it best matches the team's skill set

Design Decisions: Native/Hybrid

Issue:

How should we implement the mobile application?

Alternatives:

- Native Android application
  - Written in Java
- Hybrid Android/web application
  - Android-specific components written in Java
  - Platform-agnostic parts created with web technologies
  - Android-specific code provides an interface to native functionality (microphone, GPS) that is used by the web code

Design Decisions: Native/Hybrid

Factors affecting issue:

- A native application is conceptually simpler: fewer interacting components → fewer places for bugs
- A native application provides a look-and-feel that's consistent with the user's other apps
- A fully native application would however divert resources and cause duplicated functionality: we're already making a browser-based client, why not reuse its code

Decision:

- Make a hybrid application integrating a mobile-optimized version of our existing site with Android-specific features

Design Decisions: Scraper

Issue:

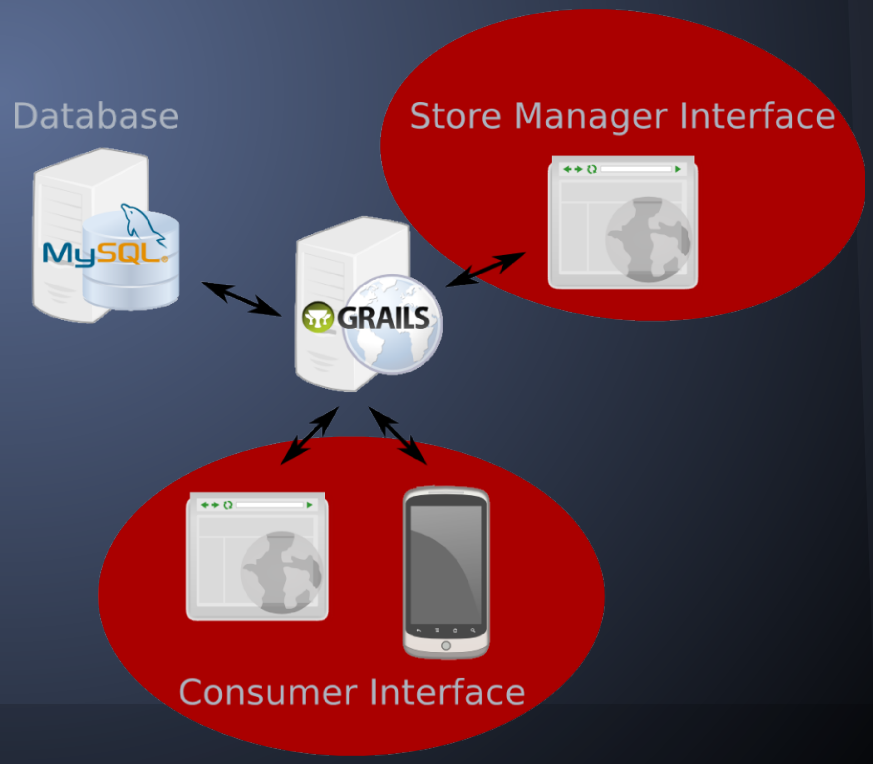
Python has powerful libraries (BeautifulSoup) for web scraping, but our application is written in Groovy; interfacing the two languages would be cumbersome.

Decision:

We stored scraped data in a widely-supported open-source database (MySQL), so we could dump it from Python and later access it in Groovy. The database acted as a liaison, bypassing the need for direct communication between the two languages.

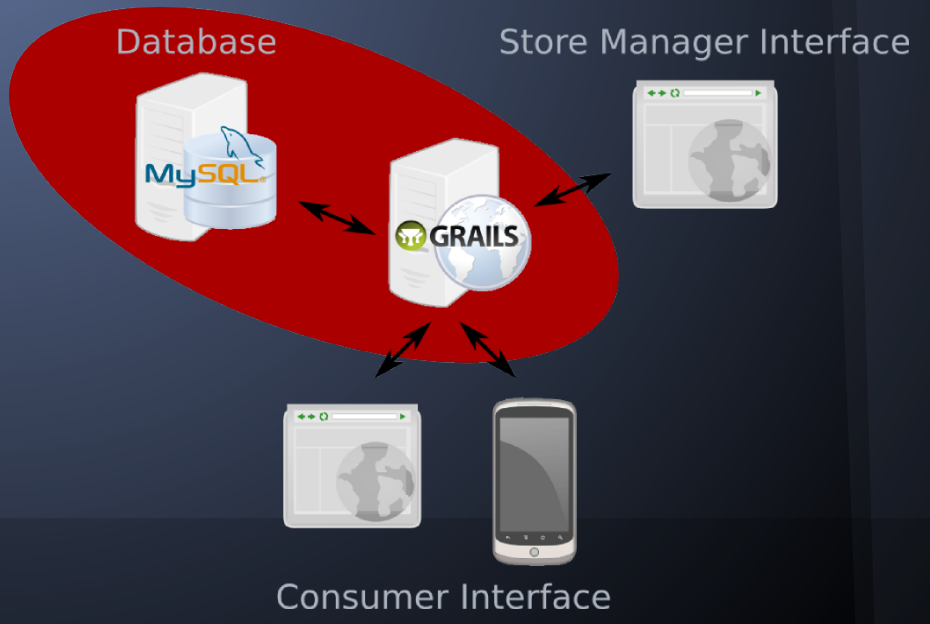
Functional Decomposition - Frontend

- Frontend
  - Hybrid Android app for mobile interface
  - HTML/CSS/Javascript web pages for browser interface



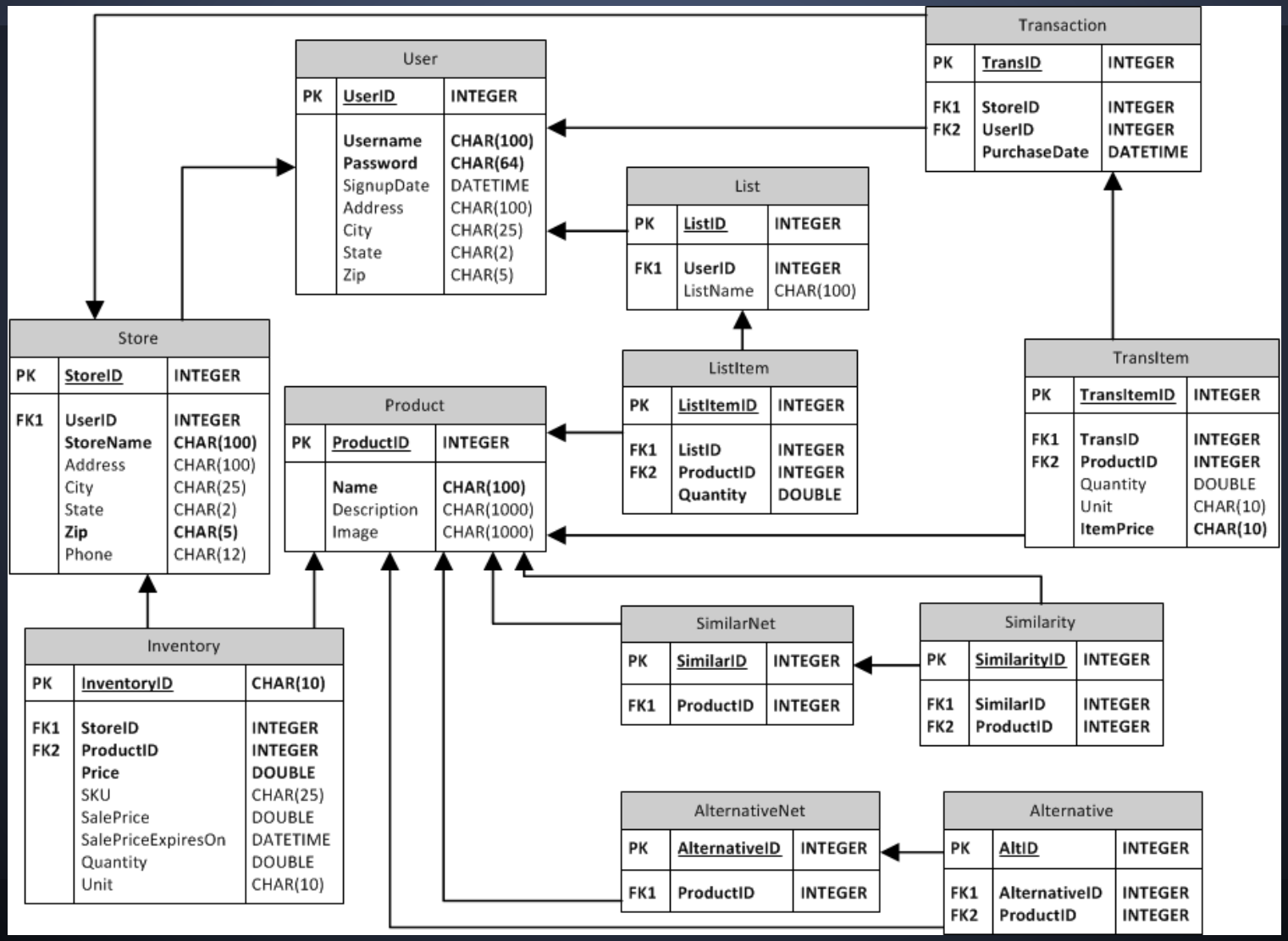
Functional Decomposition - Backend

- Backend
  - Parsers for consumers' product lists, store managers' inventories, and websites
  - Groovy Server Pages for dynamically generated/loaded web pages
  - Database



Group: Dec1206
Project: Store Selector
Client: Google

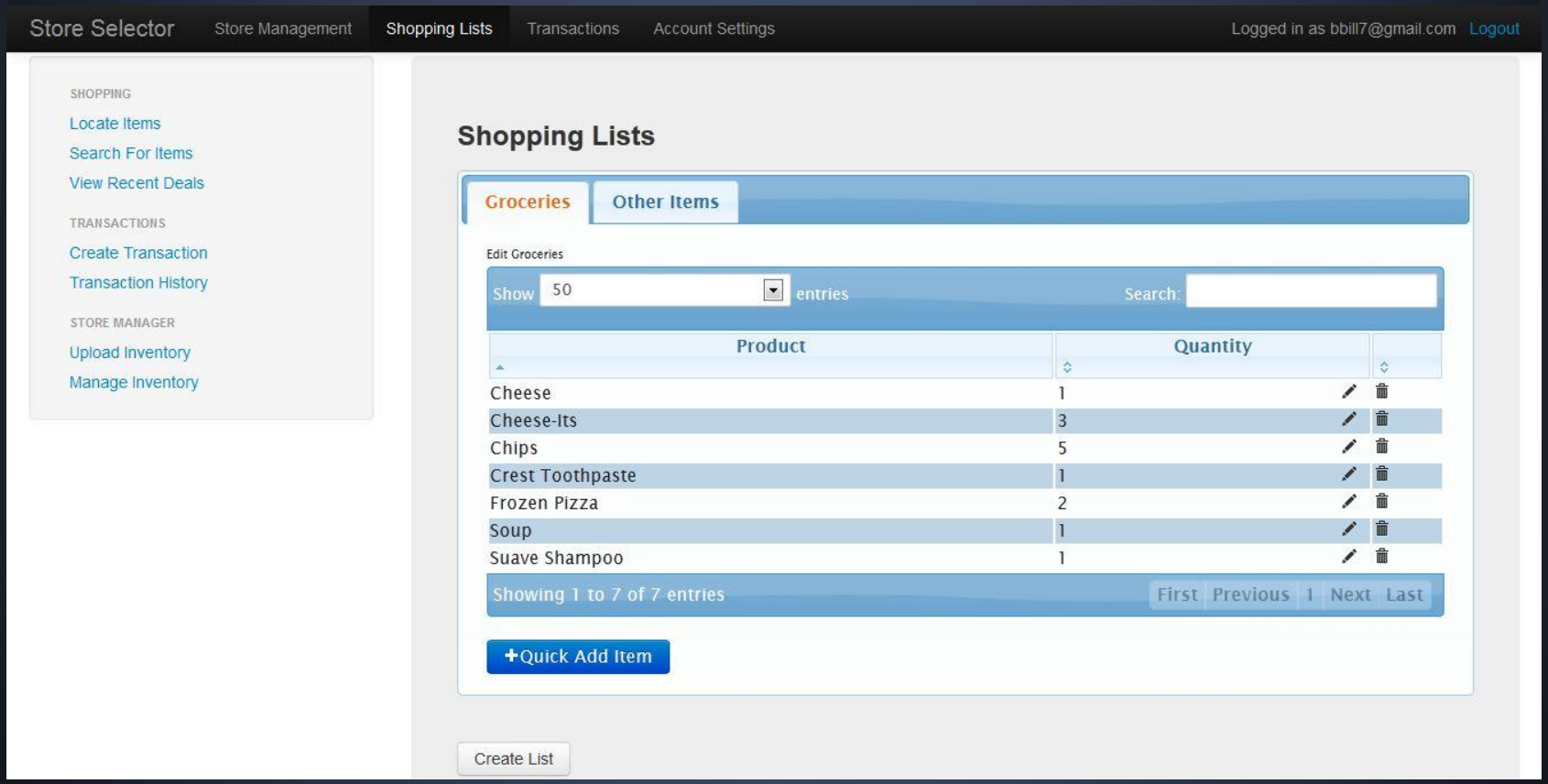
Database Schema



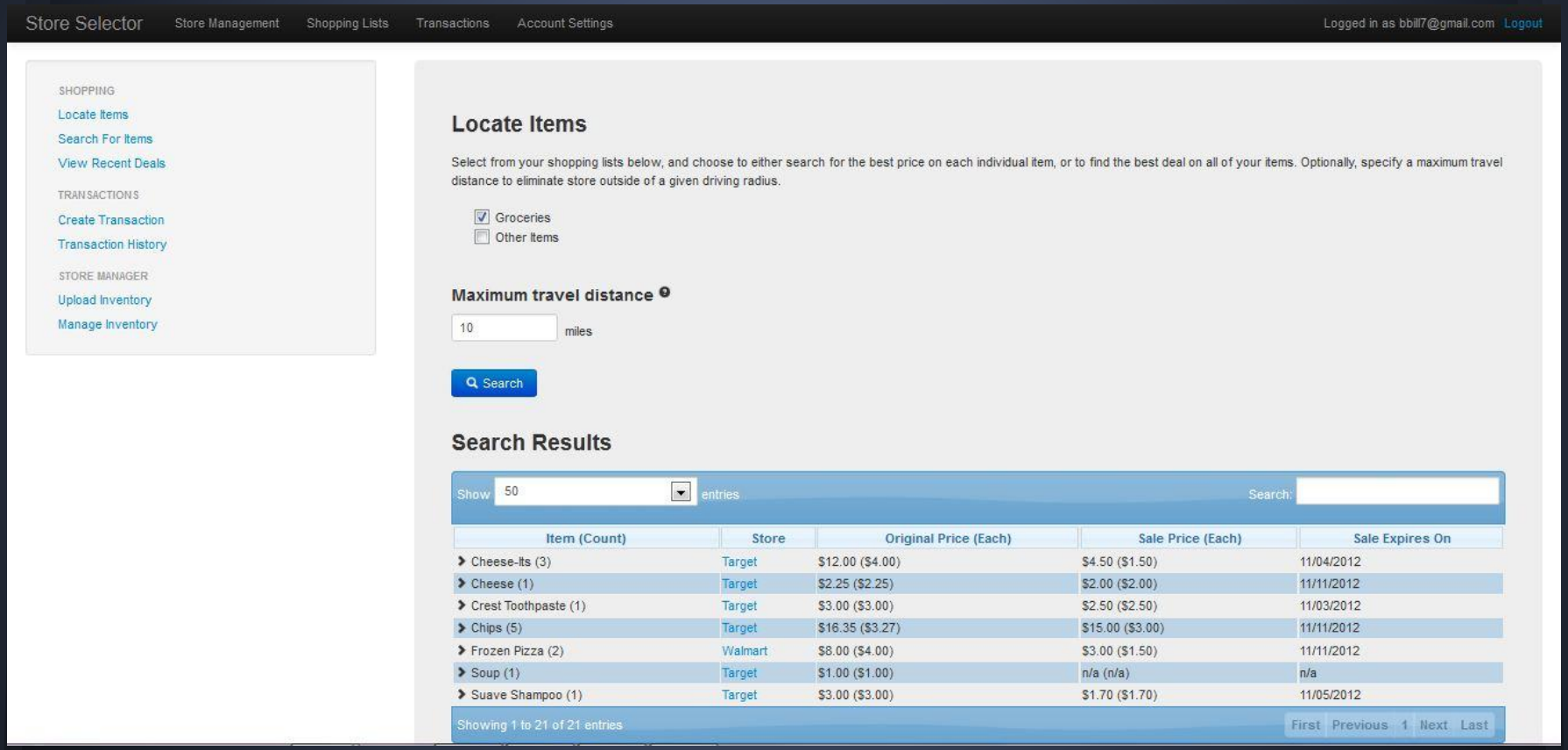
Testing

- We tested the Android app across multiple Android devices to ensure cross-device compatibility
- We tested the website in several popular browsers
- We conducted functional and usability testing across the various components of our application
- We would have liked to complete more field and usability testing, but our desire to complete more features cut our testing time

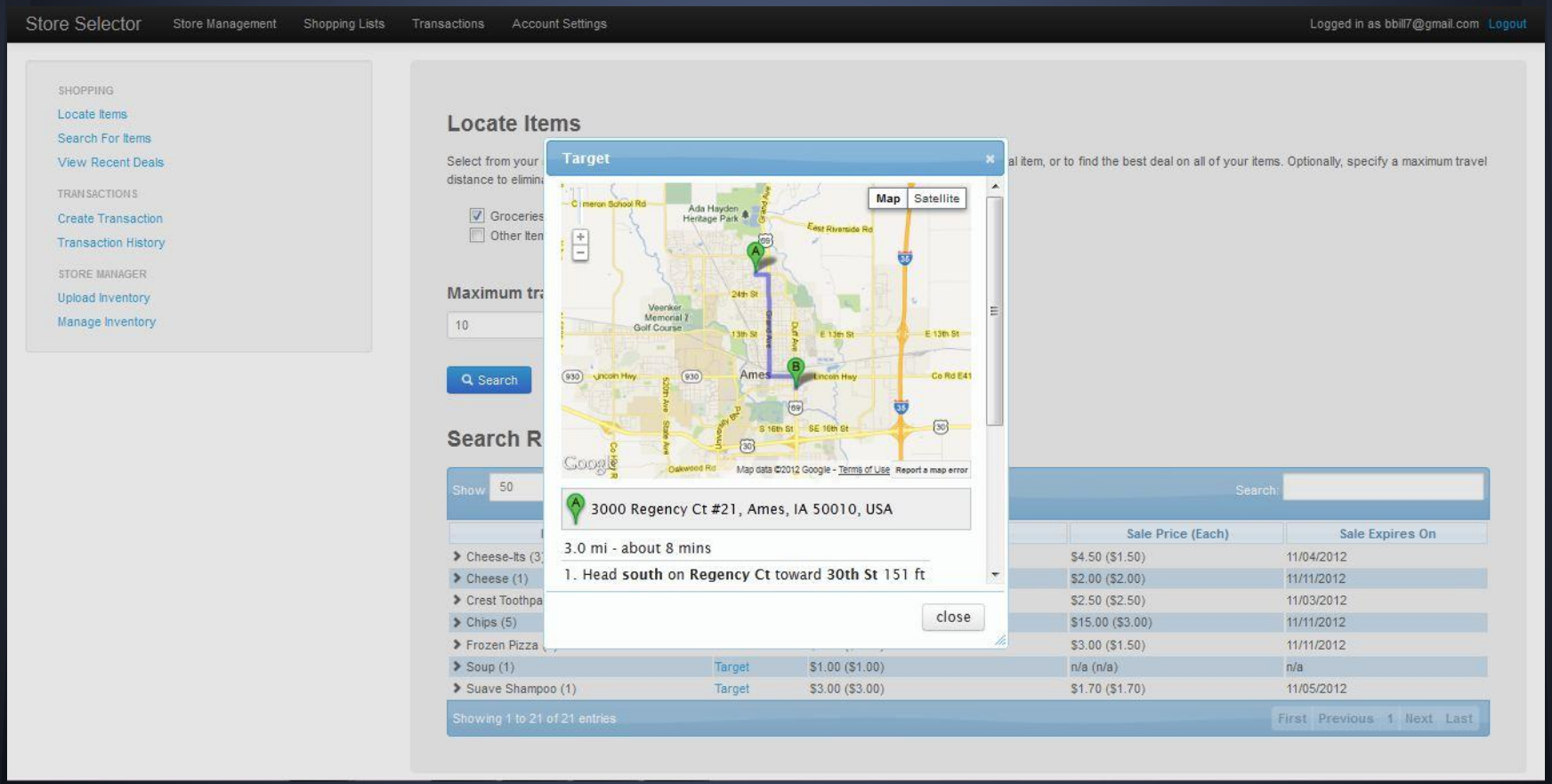
Screen - My Lists



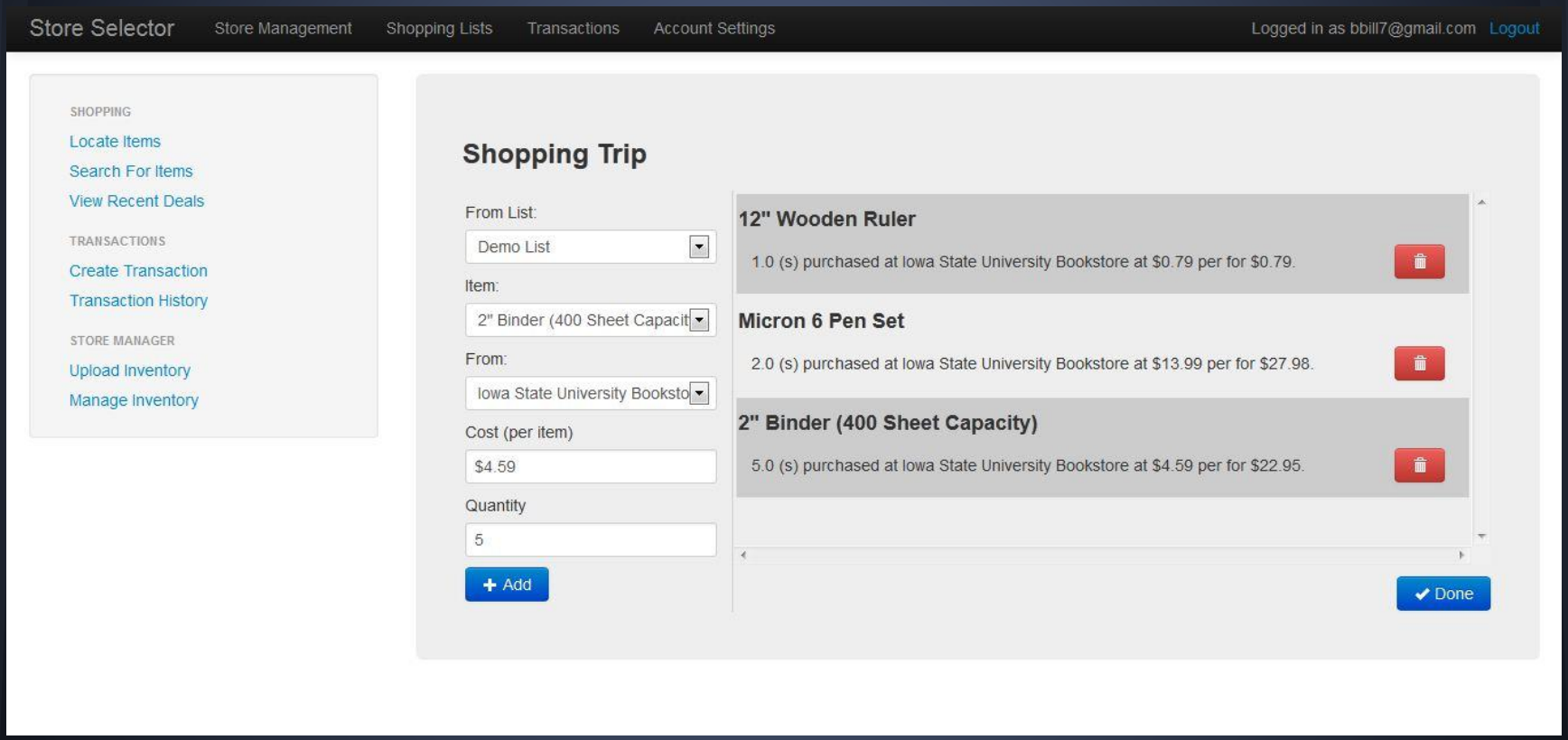
Screen - Locate Items



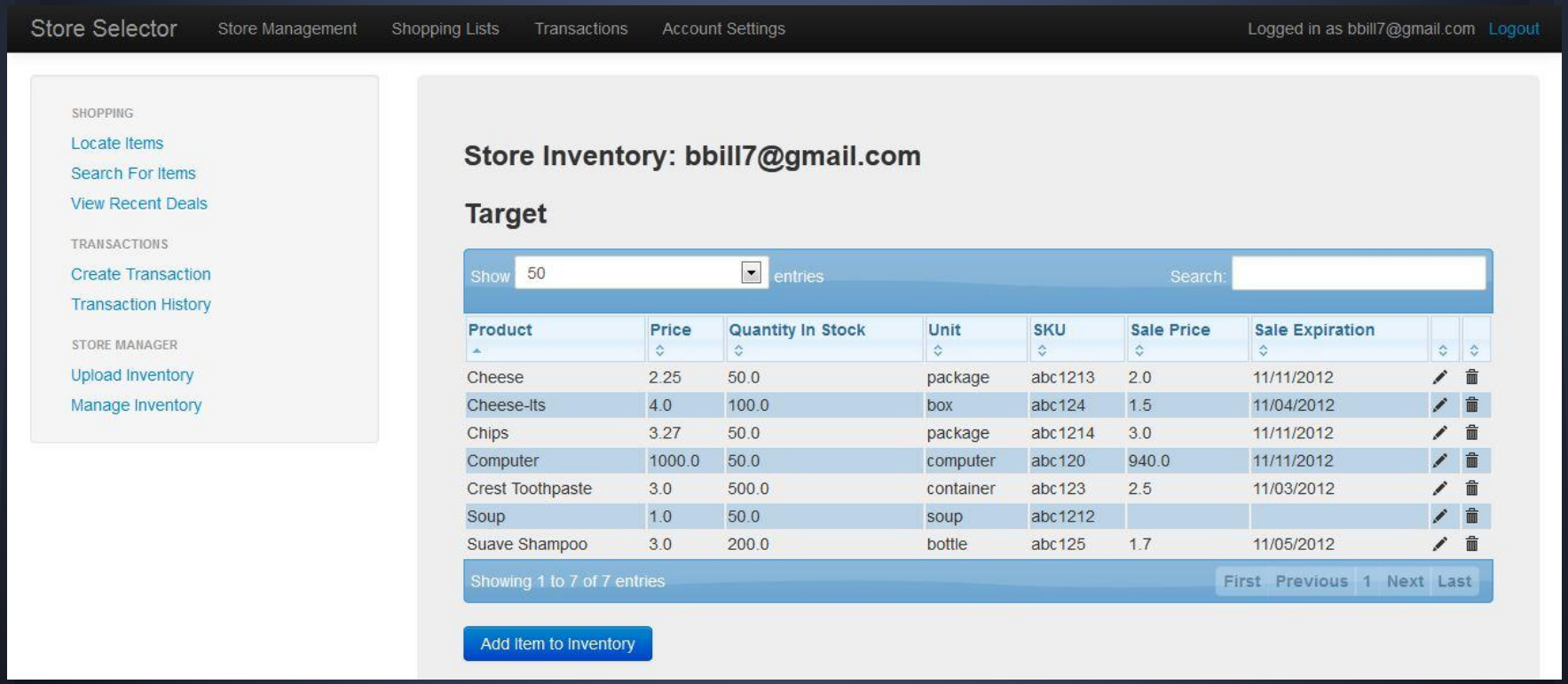
Screen - Locate Items (with map)



Screen - Create Transaction



Screen - Manage Inventory



Future Work

- Community-driven Suggestions
- Bundled Offers
- Wiki-style deal entry for users
- Deal rating system
- Multi-platform (mobile) support
- Provide an API to access users' saved lists

Group: Dec1206
Project: Store Selector
Client: Google

Questions ??

Group: Dec1206
Project: Store Selector
Client: Google

Platforms Used

- Mobile Platform: Android OS
- Web Framework: Grails
- Database: MySQL

GRAILS

MySQL®

Design Tradeoffs

- A hybrid web/native app made optimizing for Android more difficult, but it enabled us to work more efficiently on the mobile version and would make extending the application to other platforms (e.g., iOS) much easier in the future.

Design Issues

Problem:

Implementing the mobile app as a native Android app would divert resources and create duplicated functionality.

Solution:

We created a hybrid web/native app. The app's chrome and navigation were generated by native code, but most information was presented as mobile-optimized web pages in embedded web views. The native component also exposed microphone and location functionality to the web component.

Risks & Mitigation

- Have to support several different interfaces
  - Focus on Android app for customers
  - Focus on web app for store managers
- Unclear pricing availability on the web.
  - Allow users to flag incorrect data for modification or deletion.
  - Encourage and allow companies and businesses to submit their own information.

Responsibilities

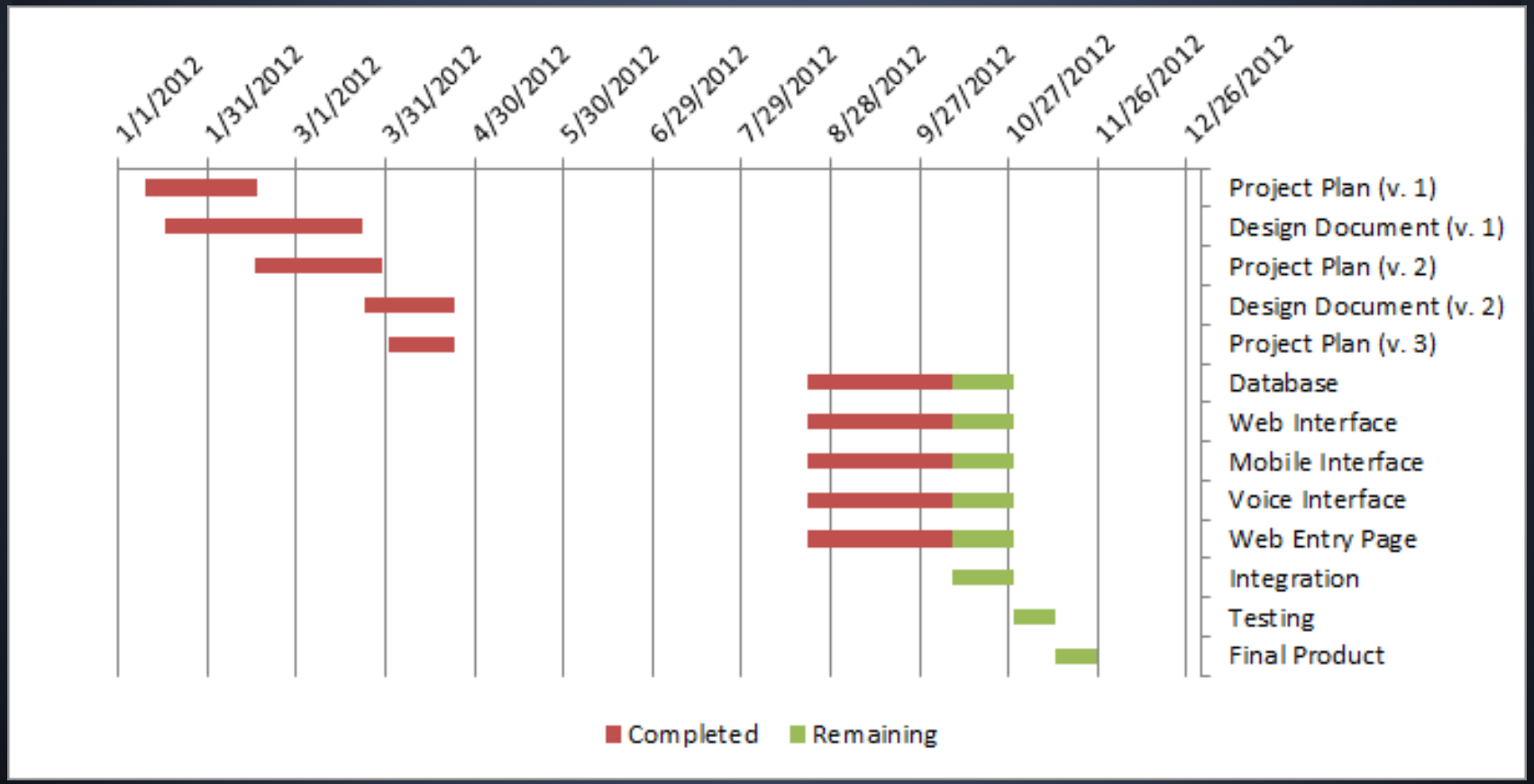
- Database - Chris
- Web Interface - Blair
- Mobile Interface - Kurt
- Voice Interface - Timothy
- Web Scraper and Miscellany- Kerrick

Status

- Scraped data from ISU Bookstore
- First iteration completed
  - Web interfaces
  - Android app
  - Voice interface
- Server in progress
- Integration begun

Group: Dec1206
Project: Store Selector
Client: Google

Schedule



Testing

- Front End Testing
  - Usability testing
  - Manual testing
  - Integration testing between controllers and views
- Back End Testing
  - Stress testing
  - Fuzzing (random data)