John-Paul Simonis
Halil Akin
Ben Carr
Paramjit Sandhu
Alex Phillips
Alex Eckerman
May 15, 2009
CSE 403

Cooking Special – Design Document

Revision History

Revision Number	Symbolic name	Modification date	Changed By	Description
1.0	cse403_cookingspecial_design.docx	April 24, 2009	-everyone-	Initial creation of the document
2.0	Cse403_design_v2_beta	May 15, 2009	JP	Beta Release design changes Changed sections: Database High Level Schema (reasoning for change explained in "Alternate Design Decisions" section)

Table of Contents

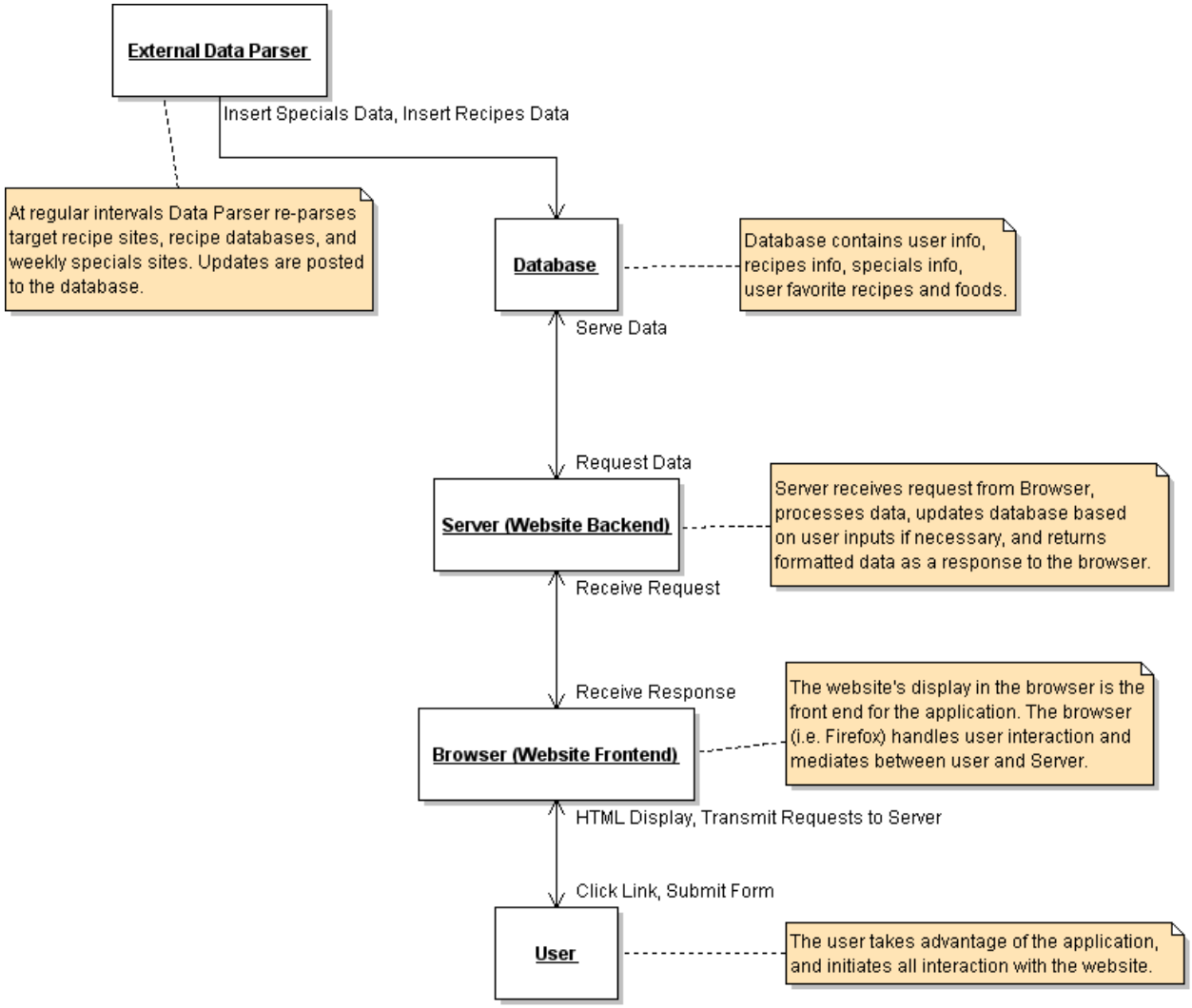
Document Abstract

This design document is a detailed definition of the Cooking Special website and its components. It identifies the major modules and interfaces between modules required to implement the system. It addresses the design of the system from both the customer's viewpoint and the developer's viewpoint.

1. Design

a. Diagrams

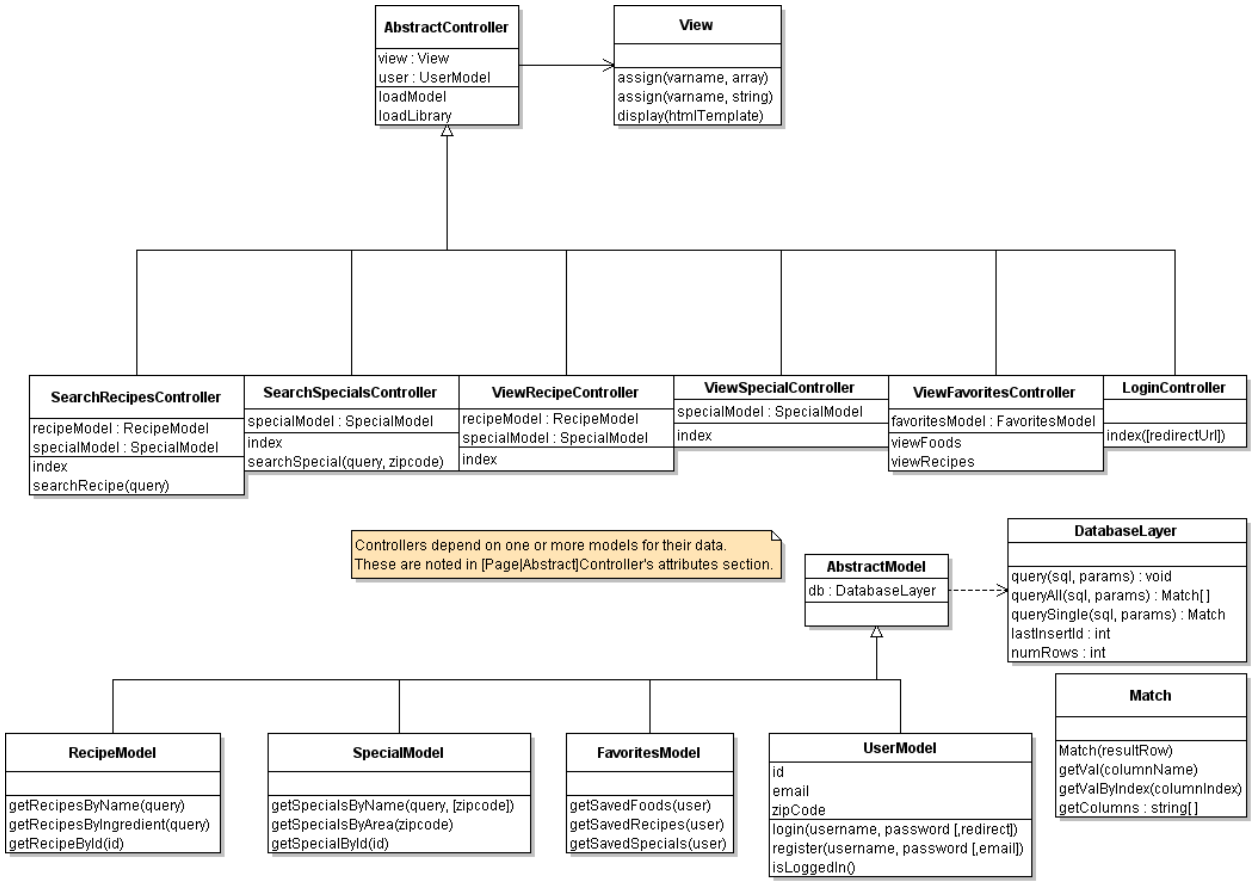
i. Module Interactions and Dependencies—High-Level View



Note: this diagram is intended to be legible and useful for both customers and developers. It provides information on interactions between each of our modules.

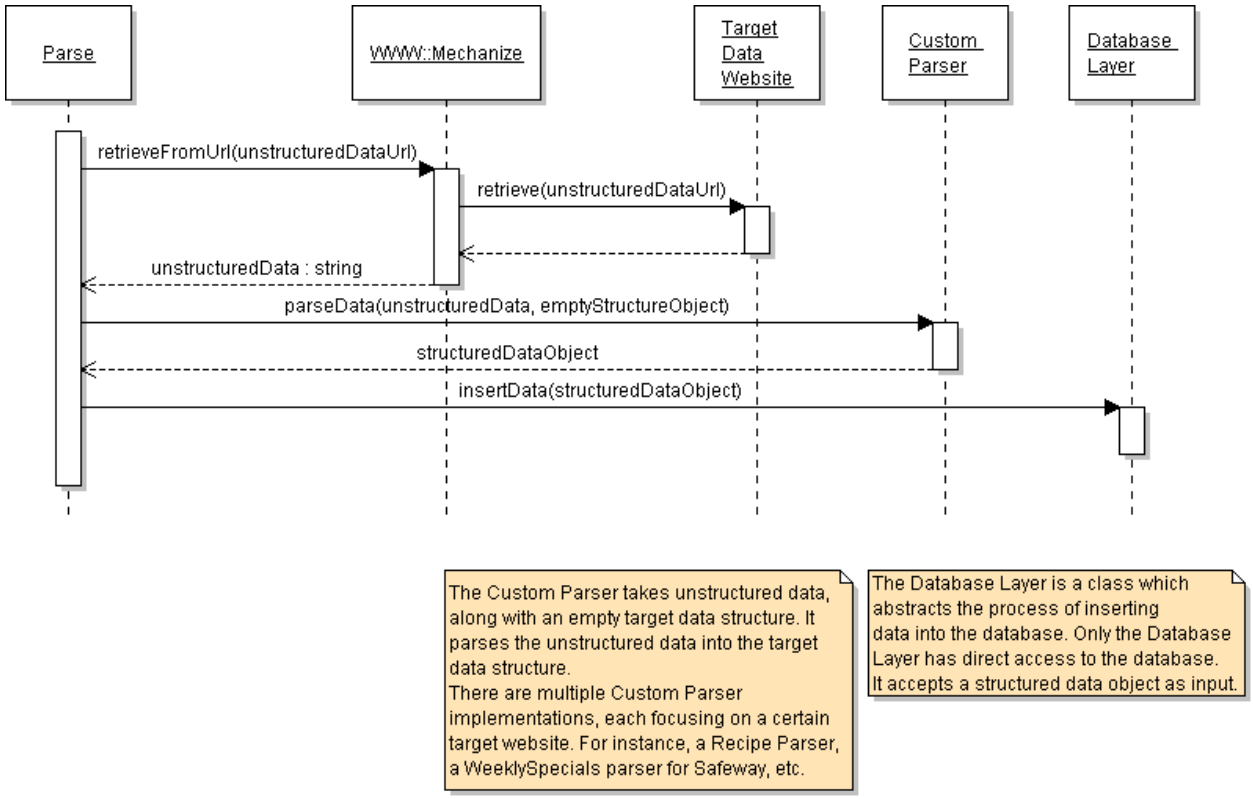
ii. UML class design diagrams

1. Server (Website Back-end) UML Class Design Diagram



The Website Back-end handles requests from the browser, and sends the appropriate response back to the user after necessary updates and changes have been considered. See the Search Recipe Use Case Sequence diagram for further information on how these classes interact with each other.

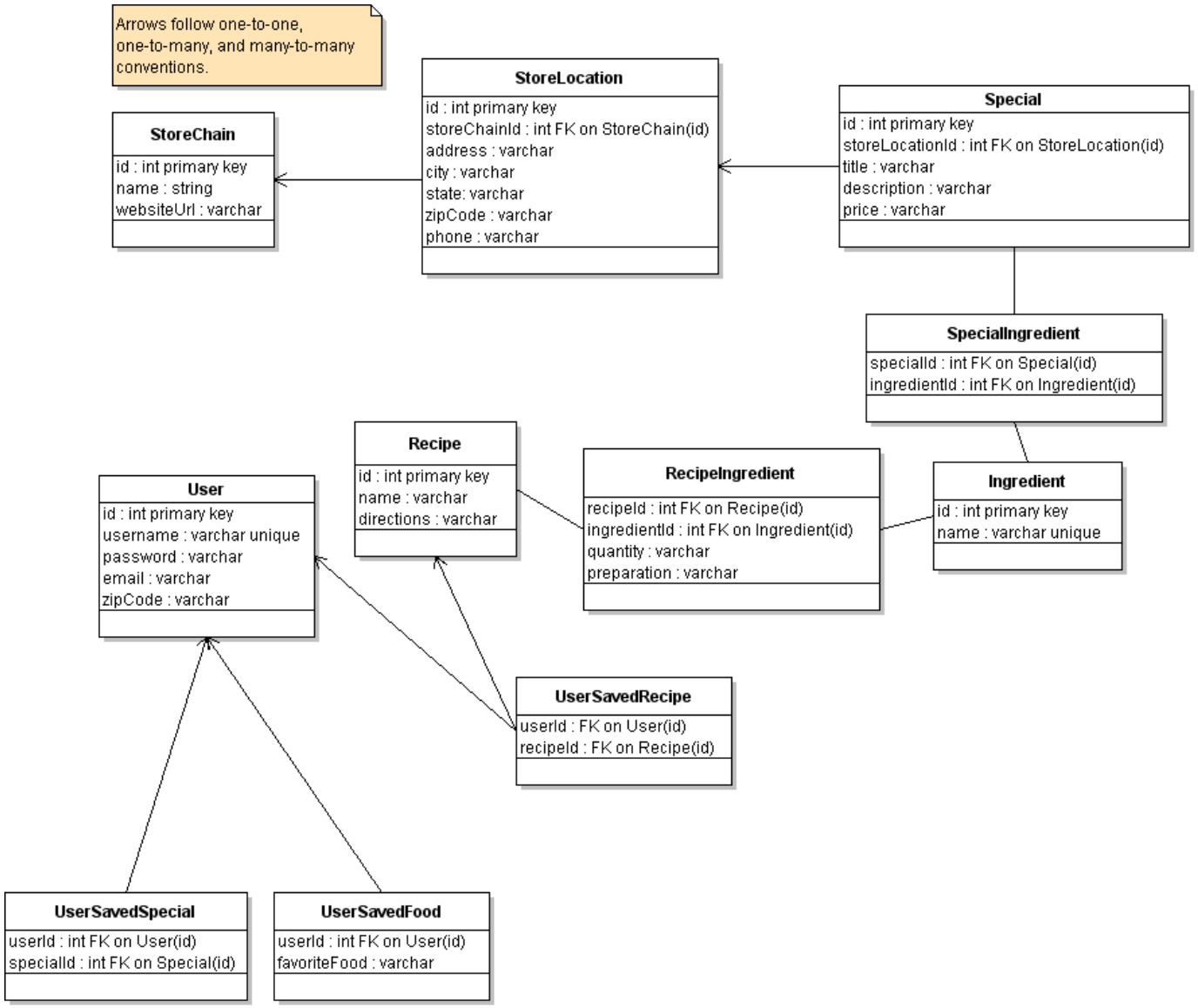
2. External Data Parser UML Diagram



Note: we found it more useful and pertinent to represent the External Data Parser's flow using a Sequence Diagram. This enables us to list the actors (Classes) above, and be able to see the intended interaction between classes through specific method calls. The actual Parser will follow the above format. Moreover, this data parser updates the database at regular intervals and as a result the website end does not need to explicitly interact with the Data Parser module. The actors in this diagram are:

- **Parse: Super** class of all specific Custom Parser classes.
- **WWW::Mechanize**: Perl module that is capable of simulating a web browser as well as parsing HTML.
- **Target Data Website**: Websites such as Albertsons, QFC, and Safeway.
- **Custom Parser**: Specific parser targeted towards a weekly special website.
- **Database Layer**: To connect to and update the database in a reasonable manner.

iii. Database High Level Schema



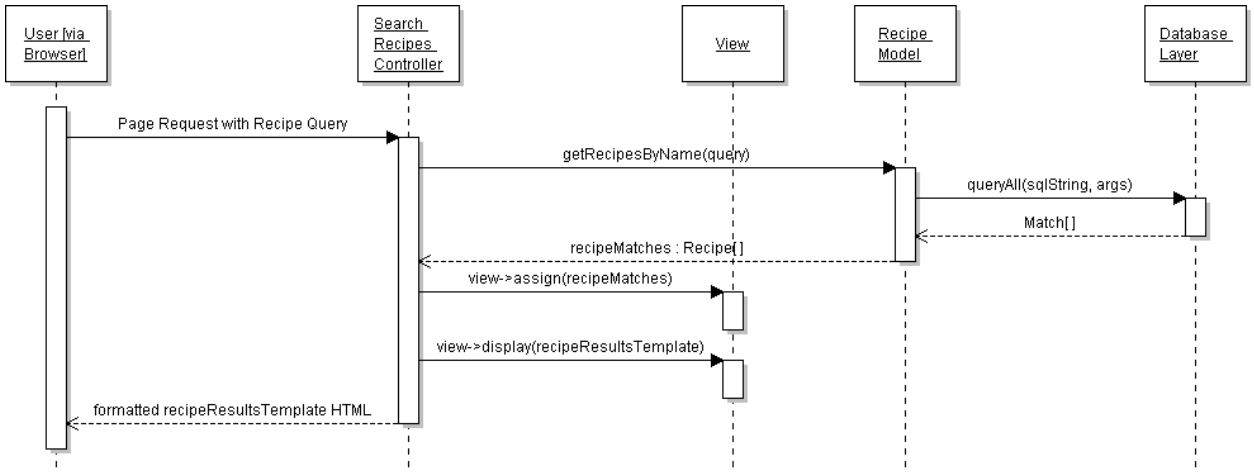
Legend:

- Boxes represent tables.
- Attributes in boxes represent columns.
- FK on User(id) is equivalent to “Is a Foreign Key referencing the id field from the User Table”
- → = one-to-many arrow.
- --- = many-to-many arrow.
- ←→ = one-to-one arrow.

iv. Use Case Sequence Diagrams

1. Search Recipes

a. Diagram

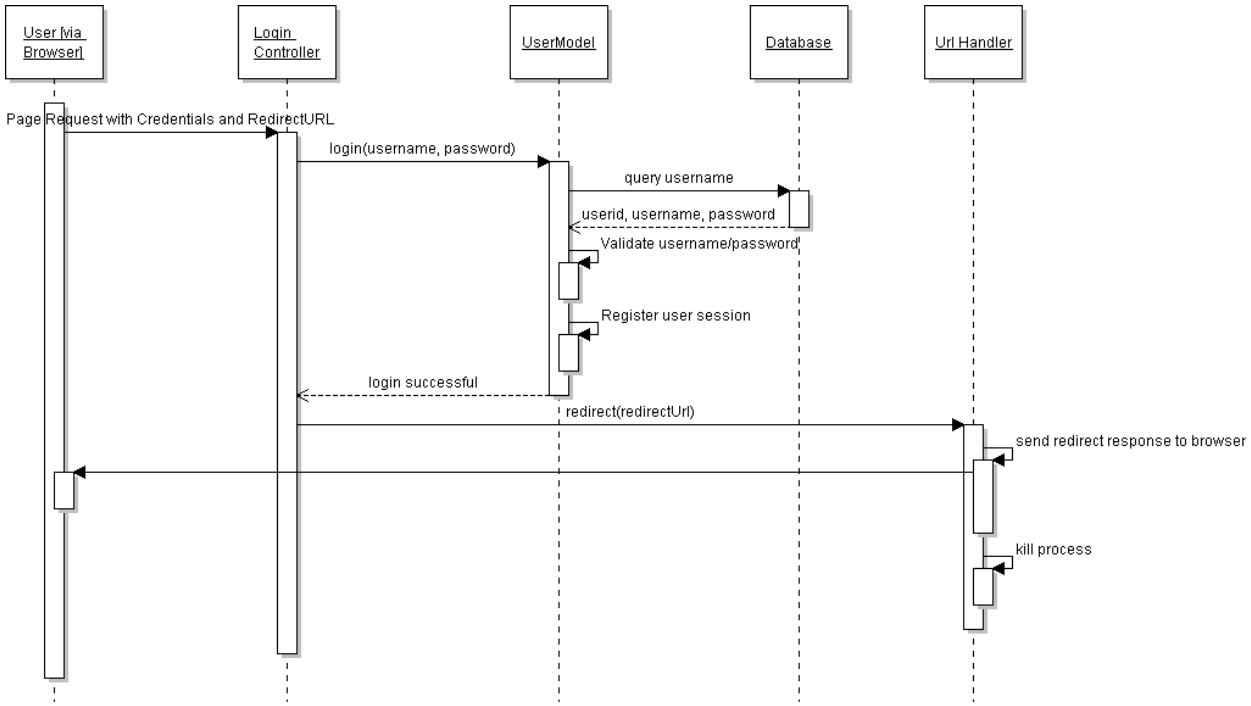


b. Pseudocode Description

1. User submits query via Browser. This occurs by typing in a query and clicking Search in the Browser.
2. Controller receives Request from Browser, including the query string.
3. Controller asks Model for recipe matches using getRecipesByName.
4. Model asks Database for recipe matches using sql query, and the original recipe query as an argument.
5. Model returns matches to Controller
6. Controller sends matches as structured array data to View using assign.
7. Controller asks View to display web page using display.
8. View displays by inserting previously assigned data into an html template (with special formatting to accommodate assigned data).
9. View sends formatted HTML output to User via Browser.

2. User Login Process

a. Diagram



b. Pseudocode description

1. User submits username and password via Browser form in original page. The originating URL is included in request as hidden data.
2. Controller receives Request, along with username, password, and redirectUrl.
3. Controller asks Model to log the user in using login.
4. Model logs user in by verifying username/password with the database, and registering user session to remember user information over multiple page views.
5. Model sends redirect Response to browser. The browser then initiates a new Request to the original page and the user returns to the original page.

Note: The original page may be any page on the website, since the login form is included on all website pages.

b. Alternate Design Decisions

With the Database Schema, we had the choice between having a separate list of ingredients for each recipe, and having a master list of ingredients to which the recipe links. Having a master list is inherently difficult because the same ingredient might have several different spellings, not to mention misspellings and different varieties. It is also difficult to programmatically determine if an ingredient has already been entered beyond just doing a direct string comparison. However, the advantage is that we can link both recipes *and* specials to certain ingredients. In this way, the task of displaying specials along with the recipe ingredients is made programmatically much easier. Having a separate list of ingredients for each recipe is programmatically easier to insert into the database, but makes the process of linking recipes to specials much more difficult. In order to take advantage of easy linking between recipes and specials, we chose to implement the master list of ingredients.

A store object has the ability to search sales within itself. This method would guarantee a many-to-one relationship between sales and stores (where a store would be a QFC or Albertsons), and would allow any sale to be easily tracked back to a specific store. Moreover, all data could be queried by store. The con of this however is the additional state we would need to store to keep track of this data, and in light of the particular needs of the web application this was not justifiable. Another possible route was to have a StoreSearch which had stores which it contained. In this method a single StoreSearch could be queried and the results would be returned from multiple stores at once. An example of this would be a single StoreSearch having Stores within it, which could be Albertsons or QFC. Each StoreSearch could in turn have associated data that might be useful, such as a zip code. This approach would have been convenient in some respects because a single object could be queried to return all sales data, but once again the additional overhead would have been unnecessary given our needs. If we really wanted to pursue an object-oriented model the former approach would have been superior, as it would have given additional benefits at lesser cost.

c. Assumptions in Design

- The External Data Parser assumes that the target webpage is always up. In the code, actual provisions will be made to deal with this error.
- The External Data Parser assumes that the webpage has not changed its data format in a significant way. This is a significant risk which is dealt with in the risks section.
- The Website Backend's recipe search sequence diagram assumes that the database has data on specials and recipes. Behavior in these alternative cases is dealt with in the Requirements document's use case section.
- The Website Backend diagrams assume that there are no internal errors while processing the browser request and sending the response. In reality, there will be a need for error landing pages to enable the user to continue using the site without being confused by error messages.
- In our general design, we assuming the user can see/interact with the website, and there is no explicit disability support. However, we will follow XHTML strict guidelines, and this enhances compatibility with screen readers and other products for people with disabilities.
- We assume that all of the modules in the module dependency diagram are working properly and are up, whereas there is the possibility that one or more of the modules is not available.

d. Coding Style Guidelines

We will be using three primary languages:

- PHP will be used chiefly for the website backend. We will use the PHP Style from Apache SVN: https://svn.apache.org/repos/asf/incubator/shindig/trunk/php/docs/style-guide.html
- Perl will be used chiefly for the external data parser. We will use the Perl Style from the Perl manual: http://www.perl.com/doc/manual/html/pod/perlstyle.html
- We will use a MySQL DBMS. SQL queries are minimal, but should conform to the MySQL official documentation at http://dev.mysql.com/doc/

2. Process

a. Top Five High-Risk Development Areas

i. Parsing weekly special web site data

- **Introduction**
  This risk was mentioned in the software requirements document as well. Automated gathering of the weekly special data from different stores is the main reason why our website will save time and money of the user. If this feature is not available then we have the risk of losing our customers. The main problem with this risk in the SRS document was that we were not sure about even parsing of HTML or getting to the website however now this risk is reduced to adapting our parser to the different variations in which (different) stores present the weekly special data.
- **Chance of occurring**
  There is a low chance that we are unable to gather weekly special data programmatically. However the chances that data parsing breaks when website changes its interface are very high.
- **Impact**
  If the weekly special store website changes its interface then the whole product will be unusable. Hence the impact of this risk is very high.
- **Steps taken to decrease chances of occurring**
  Due to the seriousness of this risk we have dedicated two of our team members on parsing of weekly special data. These team members have already experimented with ways of gathering data from one of the stores website, giving us confidence that this risk will not become our show stopper. Moreover we are planning on using Perl::Mechanize library to parse the website, which already has a HTML parser built in. As such changing website interface will certainly have a down time, but we should be able to recover faster.
- **Mitigation Plan**
  It is unlikely that all weekly specials website's interface change at the same time. If one of them changes, we can rely on the other stores. Otherwise, we can provide this information manually if possible or using our users to provide with special information until the parser is fixed or automated. In the unlikely event that all stores change their

interface will have to fall back to providing the links to different stores where the user can conveniently buy the product such that our website is at least usable.

ii. Recipe data store not available

- **Introduction**

We definitely need recipes available on our website whose ingredients our users can find cheaply. Our product relies on the availability of such recipes, either through other recipe websites, users, or even our own data storage. Due to the complexity of parsing recipes from other websites and the time it takes, our product uses its own database of recipes. Now such a database of realistic recipes might not be available or it may be proprietary.

- **Chance of occurring**
  The chances that we are unable to find any recipe data store are medium to low.
- **Impact**
  If we do not have any recipe data store available then we will not be able to provide any recipes. However our users will still be able to search for weekly specials which implies that the impact will be high for others who cook and perhaps medium for those who don't.
- **Steps taken to decrease chances of occurring**
  We have searched the internet for possible cooking recipe data stores. We have found many open source cooking software and web sites whose databases we can potentially use.
- **Mitigation plan**
  As a backup plan, if the recipes in the open source database are complex to make for an average user, we can rely on bootstrapping with our own easy recipes until we allow user to share their own recipes (which can be post beta) or we find a better recipe database.

iii. Usability of User interface

- **Introduction**
  A good user interface is the key reason why the users will actually come back and use our website. However it is usually hard to design a good user interface.
- **Chance of Occurring**
  The chances of this occurring are currently about medium, mostly depending on how much we pay attention to user feedback.
- **Impact**
  Users will not want to come back to our website, as such the impact is high.
- **Steps taken to decrease chances of occurring**
  We are very fortunate enough to get feedback from one potential user during the paper prototype presentation. We have critically evaluated the feedback and incorporated it in the way we are going to be designing the user interface. Moreover, we have planned milestones where we will gather more user data to make sure that users are comfortable with our interface. Having two members of our team working on the user interface based on interaction with customers and the other members constantly testing it will make sure our user interface is not just good but great.
- **Mitigation plan**

In the scope of our current project, we will have to continue with the user interface if it not usable. However, the only mitigation plan is to change the user interface by incorporating different aspects of the user feedback.

iv. Extensibility of our product

- **Introduction**
  One of the risks in the future is how well our product will withstand the wheel of time.
- **Chance of Occurring**
  Based on the software developed in the real world, it is very hard to make a truly extensible product. This is mainly because as the product gets complex it becomes exponentially hard to keep track of it. As such chances of this occurring in our project are about medium.
- **Impact**
  From the user's perspective the impact of this is low since they are mostly agnostic of the software. However, from a developer's perspective this risk poses a medium impact since it is very likely that we will be adding (subtle) features in the future.
- **Steps taken to decrease chances of occurring**
  On the website backend, we are using up to date technologies and frameworks which are continually updated such as the Smarty templating engine to simulate the View component. We are making sure that our product design is modular and there is relatively loose coupling between components, such that if there is a need for any future change it can be accommodated with a minimal amount of change in code. However it may be necessary in the future to apply widespread changes. We will deal with this by releasing new versions as they are necessary, and as website profits allow. On the external data parser there is some volatility as far as data sources is concerned, and we may need to brace ourselves for some data sources disappearing.
- **Mitigation plan**
  We will mitigate this by investigating new data sources as they become available, and applying some of the strategies in risk #1. Another extreme strategy is redesigning the whole system or small modules; however in our project this is not possible because of the limited time.

v. Security and protection of user data

- **Introduction**
  Another risk is dealing with the protection of user data. We are going to be storing zip code, email, username, password, favorite or special food, and special recipes for a user.
- **Chance of Occurring**
  With a greater concern on security these days, chances that some will try to attack our website is high.
- **Impact**
  Depending on why type of information is stored and consequently compromised the impact can range from low to high.
- **Steps taken to decrease chances of occurring**

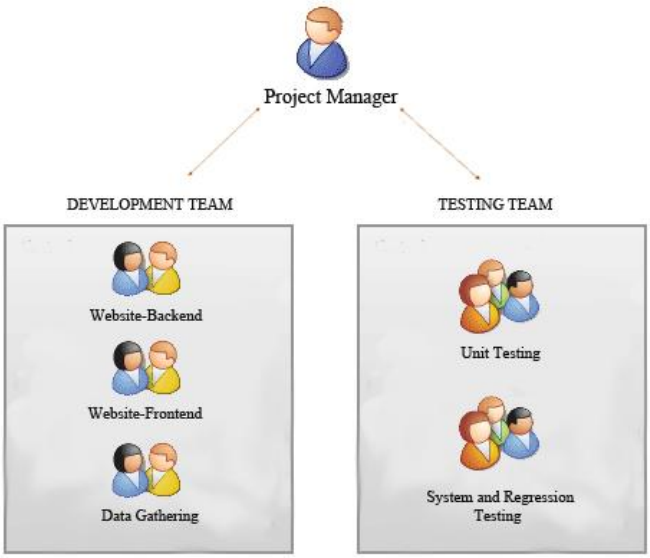
Because the user does not enter their real name, it is more difficult to associate the user as represented in the database with a real life person's identity. Nevertheless, we will not allow SQL injection attacks by using a database layer which prevents them by separating query and arguments. Also, user information will be visible only to user; other users of the site will not even know that the user exists. From a legal perspective, we do not perceive any problems with storing a minimum of user data and preferences, but will investigate the necessary privacy statements and agreements, to be presented to the user before they are allowed to register.

- **Mitigation plan**
  If this occurs then we can try to increase the security of the website as well as the database by using SSL or the latest encryption methods.

b. Changes since Requirements Document:

- The risks in this design document are more focused on specific development risks and how they can be mitigated, whereas the requirements document focused on both development risks and general project risks.
- We have added several development-oriented risks and clarified the mitigation methods by proposing development strategies that will account for them.
- Our first two risks have changed. After experimenting with shell scripts and Perl for parsing and retrieving the websites, we have reduced the first risk from very high to medium. Moreover, we have located more data store for recipes such as open source cookbook, hence reducing the second risk as well.

c. Team Structure and Task Assignments



The team is composed of main big sub-teams, Testing and Coding. We have a fairly straight forward team structure as seen in above figure.

i. Roles

On top of these sub-teams, project manager role will be filled by **John-Paul**. The main job for this role is to coordinate the actions of the team in such a way that

1) The project is progressing on schedule
2) The risks are managed as far as each development task is concerned.
3) The features of the product, roles, timeline or whatever else is needed to assure that the project is going to be completed on time, are arranged and met by requirements.
4) All members of the group are working towards specific and measurable goals, and there are no obstacles to reach these goals.

He is also going to provide expertise in different levels of our structure (specifically database and PHP web development areas).

Alex Philips will be responsible for Website-Backend section of our structure in collaboration with **Halil Akin**. Website-Backend section requires PHP and MYSQL knowledge and he is experienced with these areas. He will also help Website-Frontend section in case help is needed.

Halil Akin will deal with Website-Backend section in collaboration with **Alex Philips**. He will also be responsible for front-end development and communication between these sections in case help is needed.

Paramjit is mainly responsible for Website-Frontend section. This section includes codes and design of user interface using XHTML,CSS and other relevant technologies. He will also implement the communication between front-end and backend layers of development section.

Alex Eckerman and **Ben** will develop data gathering section. This section is about data parsing and will be implemented using Perl. The data gathering includes 1) Weekly Specials from sites such as Albertsons and Safeway

and 2) recipes from different recipe sites. The data gathering effort may require the use of other programming languages in order to take advantage of open source recipe databases. This section will need to interface with the back-end development section in order to coordinate data formats and transfer.

All members of our team will be responsible for testing. Unit tests will be executed by their developers. Integration and system tests are going to be carried out by everyone. These tests include implementation of the website's UI, correctness of the back-end's user preferences and search results sections, and also on the integrity and consistency of weekly specials data.

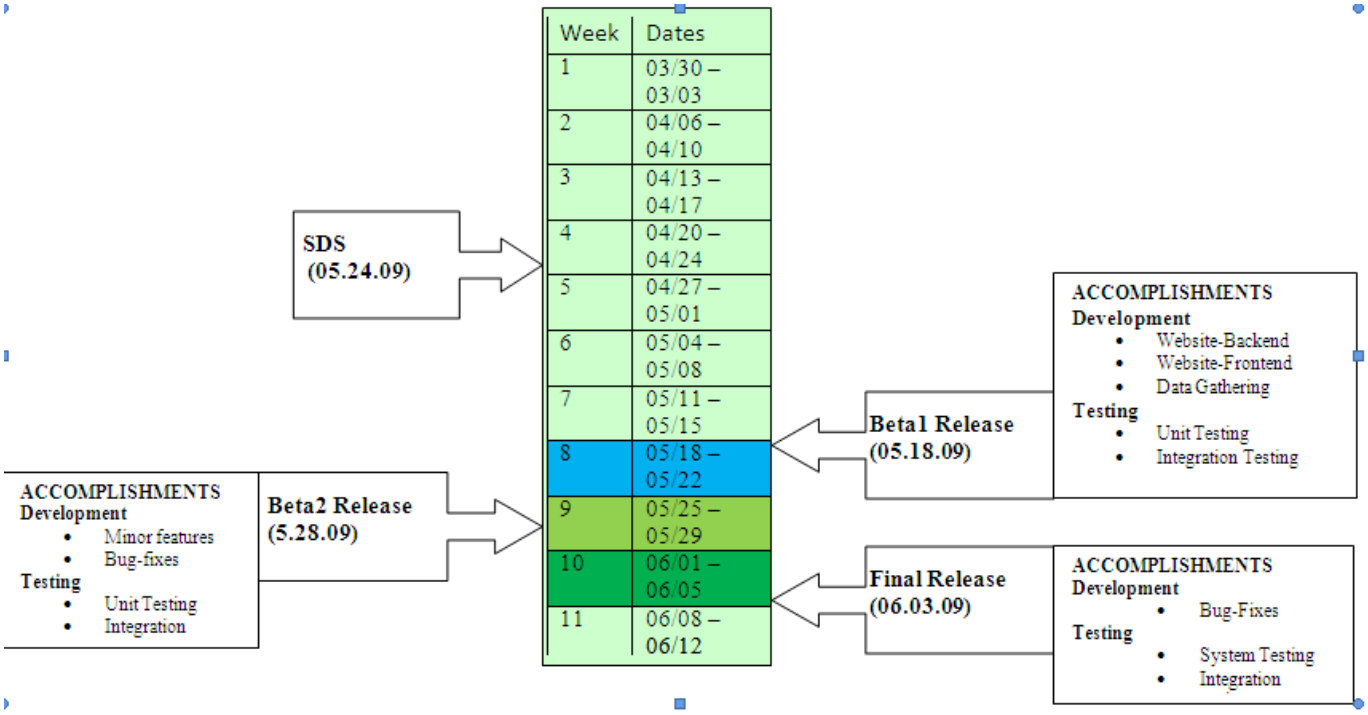
ii. Communication

Communication will occur on a regular basis over email using the cse403-cooking-special@cs.washington.edu mailing list. Individual questions can be conveyed over email:

"John-Paul Simonis" <jp.simonis@gmail.com>,
"Alex Eckerman" <acesuw@gmail.com>,
"Alex Phillips" <fluteman06@gmail.com>,
"Paramjit Singh Sandhu" <paramsan@gmail.com>,
"Ben Carr" <cbnilrem@gmail.com>,
"Halil Akin" <halilakin@gmail.com>,

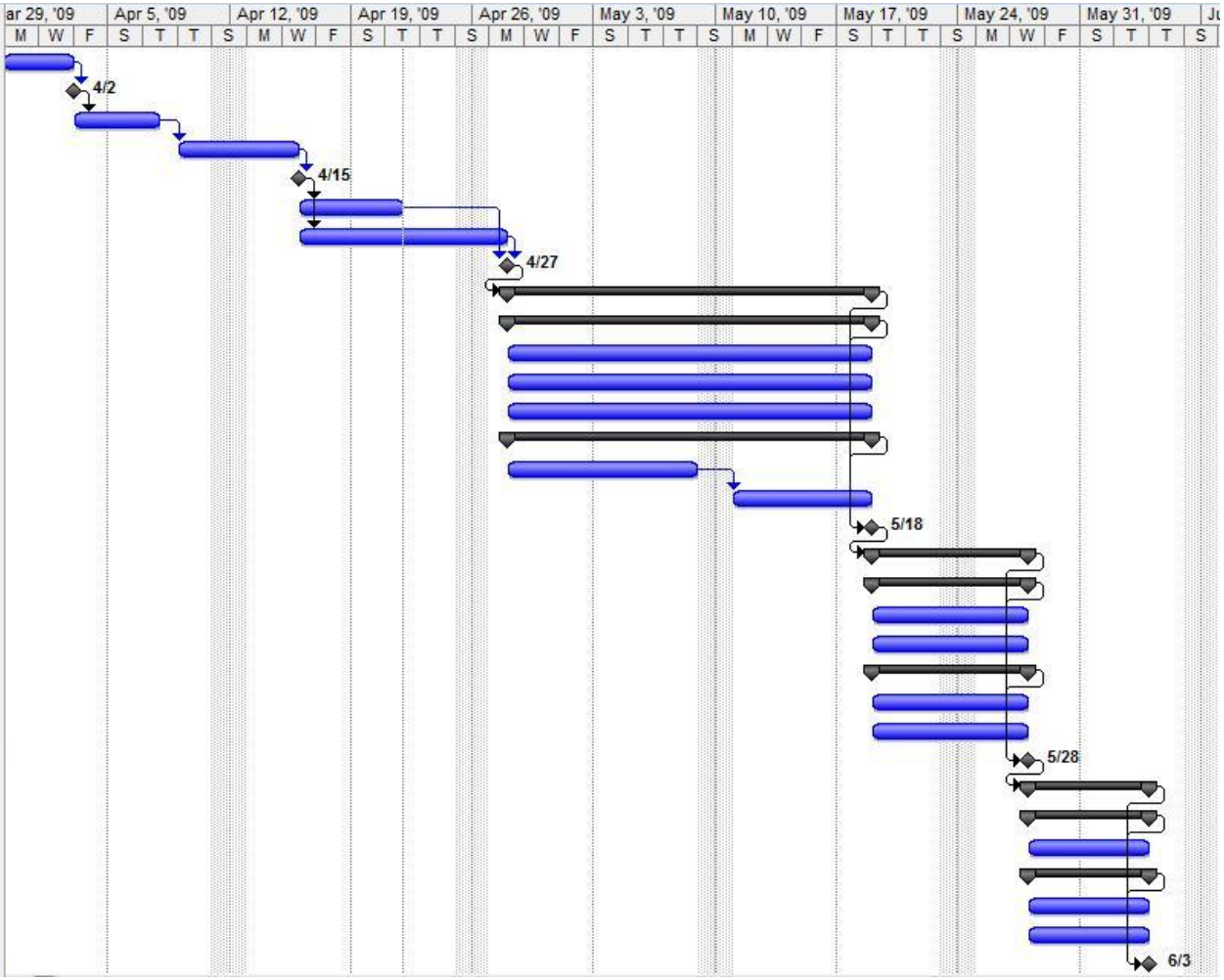
iii. Scheduling

We have 2 beta releases followed by the final release. In our releases all three sections are going to be developed simultaneously. Now, we are going to explain our tasks and milestones in more detail. For the complete scheduling process, please check the Gantt Chart. Here is the summary of the Gantt chart.



Project Task Schedule Gantt Chart

	Task Name	Duration	Start	Finish	Predeces
1	Project Proposal	4 days	Mon 3/30/09	Thu 4/2/09	
2	Project Presentation	0 days	Thu 4/2/09	Thu 4/2/09	1
3	Team Groups	3 days	Fri 4/3/09	Tue 4/7/09	2
4	Requirement Analysis	5 days	Thu 4/9/09	Wed 4/15/09	3
5	SRS Delivery	0 days	Wed 4/15/09	Wed 4/15/09	4
6	Paper Prototypes	4 days	Thu 4/16/09	Tue 4/21/09	5
7	Software Design Analysis	8 days	Thu 4/16/09	Mon 4/27/09	5
8	SRS Delivery	0 days	Mon 4/27/09	Mon 4/27/09	7,6
9	**Beta1**	**15 days**	**Tue 4/28/09**	**Mon 5/18/09**	**8**
10	**Development**	**15 days**	**Tue 4/28/09**	**Mon 5/18/09**	
11	Website-Backend	15 days	Tue 4/28/09	Mon 5/18/09	
12	Website-Frontend	15 days	Tue 4/28/09	Mon 5/18/09	
13	Data Gathering	15 days	Tue 4/28/09	Mon 5/18/09	
14	**Testing**	**15 days**	**Tue 4/28/09**	**Mon 5/18/09**	
15	Unit Testing	9 days	Tue 4/28/09	Fri 5/8/09	
16	Integration Testing	6 days	Mon 5/11/09	Mon 5/18/09	15
17	Beta1 Release	0 days	Mon 5/18/09	Mon 5/18/09	9,10,14
18	**Beta2**	**7 days**	**Tue 5/19/09**	**Wed 5/27/09**	**17**
19	**Development**	**7 days**	**Tue 5/19/09**	**Wed 5/27/09**	
20	Minor Features	7 days	Tue 5/19/09	Wed 5/27/09	
21	Bugfixes	7 days	Tue 5/19/09	Wed 5/27/09	
22	**Testing**	**7 days**	**Tue 5/19/09**	**Wed 5/27/09**	
23	System Testing	7 days	Tue 5/19/09	Wed 5/27/09	
24	Regression Testing	7 days	Tue 5/19/09	Wed 5/27/09	
25	Beta2 Release	0 days	Thu 5/28/09	Thu 5/28/09	18,19,22
26	**Final**	**5 days**	**Thu 5/28/09**	**Wed 6/3/09**	**25**
27	**Development**	**5 days**	**Thu 5/28/09**	**Wed 6/3/09**	
28	Bugfixes	5 days	Thu 5/28/09	Wed 6/3/09	
29	**Testing**	**5 days**	**Thu 5/28/09**	**Wed 6/3/09**	
30	Regression Testing	5 days	Thu 5/28/09	Wed 6/3/09	
31	System Testing	5 days	Thu 5/28/09	Wed 6/3/09	
32	Final Release	0 days	Wed 6/3/09	Wed 6/3/09	26,27,29



Beta1 Release
Our first beta release will take three weeks. Our three development sections will proceed simultaneously. As far as coding is concerned, we will focus on major features of our project. Minor features will be omitted in this release. As for testing, unit tests will be followed by integration tests.

Beta2 Release
In this release we are going to implement our minor features. After having released Beta1, we will also be able to see bugs in our system. Therefore coding part also includes bugfixes. We will run regression tests to see whether our modifications have affected our system and we will run system testing again.

Final Release
In this release we are not going to add any more codes but we will try to have most stable system. Coding will only include bugfixes. Our system testing will be carried out again to test the complete system.

d. Test Plan

Tests will be used to verify functionality of individual components, particularly in the interfaces between the main regions of the codebase. In particular unit tests need to be able to verify the edge cases where users perform behavior that might not be correctly handled by the libraries that we will be using. Moreover, as our product exists across multiple applications (database, web process, and external data) we must test cases where one or more of these is unavailable.

As our system is intended to store and retrieve specialized data, we must in particular write tests that handle the proper retrieving, possible conversion, and displaying of that data. In particular we must handle the case where user data may contain characters of special significance to our code. We must also verify the functionality of our retrieval system, which handles the scraping of data from foreign sites. This data must be returned in a way that is parseable by the other subsystems of the product. Finally, we must verify the overall product and the UI in particular by running usability tests.

The following can be considered system tests, as they handle behavior that extends over multiple realms:

- **User caused errors**
    - Must make sure that all database fields can be properly created and searched when a recipe title, ingredient or list of instructions has characters outside of the standard ASCII set
    - Make sure all search boxes are properly sanitized before being used
    - Make sure all search boxes can properly handle characters outside the ASCII set
    - Make sure all login/password boxes are properly sanitized
    - Make sure that all queries properly handle the case where the user has entered nothing
    - Make sure all queries properly handle very long strings
- **Situational Errors**
    - A query properly returns no results
    - A query properly returns 1 result
    - A query properly returns many results
    - A query properly handles a huge number of results
    - A query handles when the database is unavailable
    - A login or registration attempt handles when the database is unavailable
- **External errors**
    - An attempt at data scraping properly handles an external site being down

Unit tests must be written to handle very small pieces of every system. The following tests are some unit tests. The list may not be comprehensive depending on how the systems eventually become implemented:

- **Site functionality**
    - The site must be visible to the user
    - Data entry forms must be seen to accept text

    - Buttons must be seen to perform actions
    - A query of some kind must be sent when the user clicks a query button
    - The page must be seen to respond when it receives data
    - The data must be parsed in such a way that it is legible to the user
    - Attempting to log in must be seen to perform some sort of action
    - If the site code receives an appropriate response from elsewhere, the user must be logged in
    - Logged in users must be able to see their staple menu
    - The staple menu must be seen to reflect previous choices
    - A user must be able to enter a new staple in a text box
- **Database functionality**
    - Requests to store data must be seen to have an affect.
    - Data requested must be returned in the proper format.
    - Data should not be modified between insert and request
- **Parsing functionality**
    - Data on foreign sites must be scraped into local database.
    - Data must be queryable for item name and sale data.
    - Sale data must be distinguishable based on store type.

Usability tests will be run by giving users unfamiliar with the product a reasonable period of time to become acquainted with the overall layout of the product, then asking them to perform a particular task. The user will be timed (without his/her knowledge) and any missteps in attempting to perform the task will be noted.

This project will be using either the provided **Bugzilla** interface for defect tracking and resolution, or the **Mantis** solution for the same tasks. This is currently being discussed.

e. Documentation Plan

Documentation should include:

- An installation guide for the configuration of the database and a script to setup the database tables and structure.
- An overview of the code structure.
    - Guides to adding a new page to the website, and all related functionality such as writing additional Controller and Model implementations. Any part of the website should be extensible using these guides.
- In-code comments including method stubs and class summary.
- This (continually updated) design specification.