Summer 2005

Development Of A Personal Diet Plan Database Application For Persons With Severe Food Allergies

Heather Suzanne Ward
Regis University

Regis University
School for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

Acknowledgements

With thanks to my family for their love and support.

Thanks, Dad, for always being there.

Mom, thank you for not letting me give up.

Thanks, Sissy, for the much needed proof reading and grammar policing.

Special thanks to my team of volunteer users for taking the time –I truly could not have done it without you.

Abstract

This project will research, analyze, design, and implement a computerized system that will assist patients in creating a personal diet plan based upon a "rotation diet." This diet, specifically designed for patients with severe food allergies, requires that a patient may only eat a particular food every *n* days (where *n* is any number), and foods from the same biological food family every *n* days.

Currently, patients use either pen-and-paper or a computerized spreadsheet to create weekly or monthly food meal plans for the diet plan. The meal plans are usually transferred by hand to their daily or weekly shopping lists.

There are three main problems with the current system. First, many patients complain that in order to make their meal plans simple enough to follow easily, they tend to eat the same foods in the same order each week, so their diets have become very plain and uninteresting. Second, patients frequently make mistakes on their meal plans relating to which foods belong to a given food family, which defeats the purpose of the "rotation" diet plan. Finally, hand transferring the meal plans to shopping lists is time consuming and often inaccurate.

The ultimate goal of this project is to create a computerized system that will assist patients to make up a personalized diet plan that allows them to enjoy a broader range of meals, and also to quickly and accurately make up shopping lists for the meals.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Introduction and Background

This paper will document a project for the development of a personal diet plan database application for persons with severe food allergies. The project proposal was approved in May 2002, and was begun in March 2004.

The goals of this project were to research, analyze, design, and implement a database application that would assist patients in creating a personal diet plan based upon a "rotation diet." This diet, specifically designed for patients with severe food allergies, requires that a patient may only eat a particular food every *n* days (where *n* is any number), and foods from the same biological food family every *n* days.

1.2 The Rotation Diet

The "rotation diet" was specifically designed for patients with severe food allergies, and was intended to reduce the chance of these patients developing allergies to foods that are currently safe for them to eat.

The most important feature of the rotation diet is the rotation schedule. The rotation schedule requires that a patient may only eat a particular food once every *n* (where *n* is any number) days, and may also only eat foods from the same biological food family every *n* days. This is notated as *family rotation /food rotation* (4/6, 3/7, or 3/5, etc). For example, with a 3/5 day rotation, if the patient eats broccoli (mustard family) on Monday, they cannot eat broccoli again until Sunday (five days between), but may eat something else from the mustard

family, such as cauliflower, on Friday (three days between). As another example, if the patient eats string beans (legume family) on Tuesday, they may eat navy beans on Saturday, but may not eat string beans until Monday.

In order to determine which foods belong to which families, patients consult a diet manual containing a list of foods categorized by biological food families. For example, tomatoes are listed under the Nightshade family, cow's milk is under the Bovine family, and apples are under the Rose family.

1.3 Current Process

Currently, patients use either pen-and-paper or a spreadsheet to create daily meal plans for the rotation diet, usually making up one to four weeks worth of meal plans at a time. The meal plans are then transferred by hand from the meal plans to their daily or weekly shopping lists.

Below is a sample partial weekly meal plan for a 3/5 day rotation schedule.

Table 1: Partial Meal Plan Sample

Sun	Mon	Tues	Wed	Thurs	Fri	Sat
Breakfast Orange Juice, Butter, Wheat Toast	*Breakfast* Goat's Milk, Salmon	*Breakfast* Bacon, Buckwheat	*Breakfast* Tea, Banana	*Breakfast* Coffee, Chicken Egg, Honey Dew	*Breakfast* Apricot, Blueberry, Beet Sugar	*Breakfast* Orange Juice, Butter, Wheat Toast
Lunch Beef, Wheat Bun, Cow's Milk Cheese, Tomato, Butter	*Lunch* Catfish, Okra	*Lunch* Apple, Pork Chop, Green Beans	*Lunch* Broccoli, Cod, Vinegar	*Lunch* Cashew, Chicken, Curry Powder, Jicama	*Lunch* Carrots, Goat's Milk Cheese, Mutton	*Lunch* Beef, Wheat Pasta, Cow's Milk Cheese, Tomato, Butter

Patients select the foods they put on the meal plans from three sources:

1) A list of “safe” foods that do not cause the patient an allergic reaction.
2) Their diet manual
3) Past meal plans, which help patients remember when they last ate a food.

There are three main problems with the current system:

First, many patients complain that in order to make their meal plans simple enough to follow easily, they tend to eat the same foods in the same order each week. As a result, their diets have become very plain and uninteresting.

Second, patients sometimes neglect to consult their diet manual in determining the foods belonging to a particular food family, which often leads to mistakes with the rotation schedule that are severe enough to defeat the purpose of the rotation diet.

Finally, hand transferring the meal plans to shopping lists is time consuming and often inaccurate. This inaccuracy can lead to patients eating foods out of rotation because they do not wish to make another trip to the grocery store to buy the food that was actually on their meal plan. Furthermore, most patients that give up on their rotation diet state that the major contributing factor to their failure to stay on the diet was the fact that they simply did not have time to create meal plans properly.

1.4 Project Goals

The goals of this project were the development of a database application that would 1) assist patients in creating their personal diet plan based upon a rotation diet, 2) allow them to enjoy a broader range of meals, and 3) to quickly and accurately make up shopping lists for the meals.

1.5 Project Scope

The scope of the project was determined during informal user interviews, and is based on the project goals.

It was determined that the application should:

1) Assist patients in the creation of daily meal plans for their personal diet plan.
2) Allow patients to specify rules and preferences for their diet plan, in particular the rotation schedule (3/5, 4/6, etc) and the foods that they are able to eat.
3) Not allow patients to insert foods into their meal plans that break the rules of their diet plan.
4) Allow patients to easily transfer their meal plans to weekly and/or daily shopping lists.
5) Allow patients to easily print out their meal plans and shopping lists.
6) Save meal plans for future reference.

1.5 Project Resources

The primary personnel resource for the project was the student that initiated the project, fulfilling roles as the project manager, systems analyst,

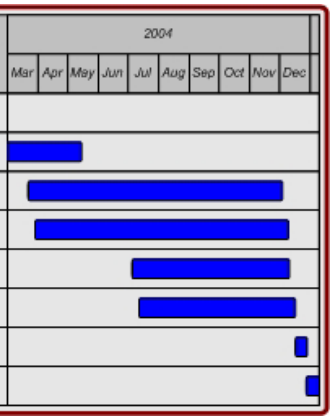
developer, and lead tester. The only other personnel resources were the team of volunteer application testers.

Hardware resources were two personal desktop computers, and the budget for the project was limited to $400.00.

1.6 Project Timeline

Figure 1: Project Timeline

ID	Task Name	Start	Finish	Duration
1	Preliminary Investigation	3/1/2002	5/1/2002	8.8w
2	Problem Analysis (Accelerated)	3/1/2004	5/14/2004	11w
3	Design (Iterative)	3/22/2004	12/3/2004	37w
4	Construction (Iterative)	3/29/2004	12/9/2004	36.8w
5	Implementation (Iterative)	7/5/2004	12/10/2004	23w
6	Analysis (Iterative)	7/12/2004	12/16/2004	22.8w
7	Implementation (Final)	12/17/2004	12/28/2004	1.6w
8	Operation and Support (Optional)	12/28/2004	6/17/2005	24.8w



2 Research

2.1 Project Methodology

One of the first decisions made during the research process was the selection of the project methodology. After a careful review of the various forms of Software Development Life Cycle (SDLC) methodologies, the student determined that since the initial research for the project indicated that it would most likely be implemented as an n-tiered database application, the most appropriate methodology for the project would be some form of Rapid Application Development (RAD).

The student chose to create a development plan based on RAD, outlined below:

1) Preliminary Investigation
2) Problem Analysis (accelerated)
3) Iterative Phases
    a. Design
    b. Construction
    c. Implementation
    d. Analysis
4) Implementation (final)
5) Operation and Support (optional)

2.2 Candidate Solutions

After the project methodology had been selected, the student conducted research to determine the business and user requirements. These were then used to complete an initial feasibility analysis of the technologies that would be

used for the project. This research was very important, as it would determine the direction the project would take.

Initial research had indicated that there were no existing off-the-shelf software applications of any sort designed for assisting patients with creating personal diet plans based on a rotation diet. Further research confirmed the lack of existing solutions, so the student determined that a custom application would have to be built.

Four categories of custom solutions were researched over the course of the project: 1) Application Implementation Technologies, 2) Integrated Development Environment (IDE) Solutions, 3) Database Solutions, and 4) Database Application Web Hosting. The top four candidates in each category were considered and scored using a candidate systems matrix. The highest scoring candidate in each matrix was chosen as the solution for that category.

2.2.1 Application Implementation Technologies

One of the most important decisions made was how the application was to be implemented, since this would directly affect which candidates would be selected in the other three categories.

The student chose four candidates for review in the application implementation category: 1) Oracle Portal, 2) Java with Web Start, 3) Java web application with the Struts framework, and 4) Java web application with the Struts and iBATIS frameworks.

2.2.1.1 Oracle Portal

The first candidate was Oracle Portal, which is part of the Oracle Application Server suite. There would be several benefits to using Oracle Portal. First, the student was very familiar with Portal, and thus could implement the application fairly quickly. Second, she already had an academic developer's license for Portal. However, there were several barriers to using Portal for this project. First, Application Server is extremely expensive to deploy, so users would not get their wish of actually using the application if the project was reasonably successful. Also, this solution would have to be deployed and accessed over the student's home network, so users would have to come to the student's home to test the application for her. Lastly, this solution would require use of the Oracle database, which is also very expensive to deploy.

2.2.1.2 Java with Web Start

The next candidate was Java with Web Start, which would be used to create a stand-alone application with an imbedded Borland JDataStore database. Users would use Java Web Start –which must be downloaded from Sun's web site– to download, install, and run the database application. There were several barriers to using this solution. First, Web Start can be confusing, and is probably too difficult for novice computer users. Also, using Java to connect to the database and pass data back and forth between the application and database would increase the amount of time the implementation would take because it requires: 1) much more code in the form of JDBC (Java Database Connectivity) SQL queries and updates than the Oracle solution, and 2) more design work than

implementing with Portal, because the user interface, database/application interface, and application design/design patterns would have to be worked out. The main benefit of this solution was the fact that it would be much cheaper to deploy since Java deployment is free. However, the JDataStore imbedded database would be about $25 per user to deploy. Also, even though Web Start would be free, users would have to download the application, so a web host would have to be selected and a domain name purchased, which would add to the costs. These costs could probably be passed on to users, however.

2.2.1.3 Java with the Struts Framework

The next candidate was to create a Java web application using the Struts framework. There would be several benefits to this solution. First, the Struts framework is designed to make the development of a large web application much easier, inherently providing all the underpinnings for the application such as design patterns (hence the name Struts). Also, the application could be supported by any of the database solutions. However, as mentioned above, using Java for database connectivity would require more time to design and implement. Also, the student would have to learn to use the Struts framework. Deployment of this solution would be free, but would also require the purchase of a domain name and the selection of a database application web host.

2.2.1.4 Java with the Struts & iBATIS Frameworks

The last implementation candidate was to create a Java web application using both the Struts and iBATIS frameworks. The addition of the iBATIS framework to the Struts framework could significantly reduce the amount of

JDBC code that would be required for the application. The iBATIS framework eliminates the need to have any actual SQL code at all within Java classes, with all of the constant parts of the SQL code held in XML (Extensible Markup Language) files and accessed only by the framework. Again, this application could be used with any of the database solutions. The only barrier would be the fact that the student would have to learn to use both the Struts and iBATIS frameworks. This solution would be free to deploy, but would also require the selection of a web host and purchase of a domain name.

2.2.1.4 Candidate Systems Matrix – Application Implementation

Feasibility Criteria	Wt.	Candidate 1 Oracle Portal	Candidate 2 Java with Web Start	Candidate 3 Java Web App with Struts	Candidate 4 Java Web App with Struts & iBATIS
Operational Feasibility (Functionality)	20%	Fully supports all user requirements. Score 100	Does not fully support all user requirements. Score 80	Fully supports all user requirements. Score 100	Fully supports all user requirements. Score 100
Technical Feasibility	20%	Requires Oracle database back-end. Would require users to have Internet access, and some training. Very well integrated with the Oracle database and Oracle developer solutions. I am very familiar with Oracle products. Would require at least one new computer. Score 80	Can be used with any of the database solutions. Can be used with any of the application development solutions except Oracle. Would require users to have Internet access, would be very difficult for most users to set up. Would require a lot of coding to get data into and out of the database. Score 60	Can be used with any of the database solutions and any of the application development solutions except Oracle. Would require users to have Internet access. Would require a lot of coding to get data into and out of the database. Will require learning the Struts framework. Score80	Can be used with any of the database solutions and any of the application development solutions except Oracle. Would require users to have Internet access. Would speed the coding required to get data into and out of the database. Will require learning the Struts and iBATIS frameworks. Score 90
Economic Feasibility **Cost to Purchase Licenses to Deploy and /or Distribute final product**	55%	(Budget $100) $2000 (for new computer) $5000+ Score 0	(Budget $100) $0 $0 Score 100	(Budget $100) $0 $0 Score 100	(Budget $100) $0 $0 Score 100
Schedule Feasibility (Can I keep to my Project Schedule)	5%	Score 100	Score 80	Score 70	Score 80
Ranking	100%	Score 41	Score 87	Score 94.5	**Score 97**

2.2.2 Integrated Development Environment (IDE) Solutions

IDEs are the software used to construct applications. The selection of an IDE is a very important decision, because the selection of the right IDE for a project can make development of applications faster and more correct.

One of the most important factors in the selection of the IDE for the project was the fact that the IDE must support whichever application implementation technology was selected. For example, if the application was to be implemented as an Oracle Portal application, it would have to be developed with Portal's integrated development tools, but if the application was to be implemented in pure Java, any number of Java IDEs might be considered.

Another important consideration was the cost of purchasing the IDE software. Since the budget for the IDE was limited to $100, most candidates proved to be far too expensive, even at an academic discount.

2.2.2.1 Oracle Portal Integrated Development Tools

The first candidate IDE was the development tools integrated with Oracle Portal. These tools are easy to use, and the student was experienced with them as she had used them in class. She also had an academic developer's license for Portal, so this solution would be free. This solution could only be used with the Portal implementation solution, and would require the use of an Oracle 9i database.

2.2.2.2 Borland JBuilder Enterprise Edition

The second candidate was Borland's powerful JBuilder Enterprise Edition. This solution could be coupled with any of the Java implementation solutions. The program would be easy to use and could greatly speed Java development for the project. However, this solution was very expensive, and even at discounted academic pricing, would exceed the budget by about 500%.

2.2.2.3 Metrowerks CodeWarrior Pro

The third candidate was Metrowerks CodeWarrior Pro. This program is not as powerful as JBuilder, but is easy to use and would speed Java development. This solution could be coupled with any of the Java implementation solutions. This candidate would exceed the budget by about 10% at discounted academic pricing.

2.2.2.4 NetBeans

The last IDE candidate was NetBeans, an open source IDE. NetBeans is not as powerful as CodeWarrior or JBuilder, but its no-frills interface would be easy to use. This solution could be coupled with any of the Java implementation solution, and is free.

2.2.2.5 Candidate Systems Matrix - Integrated Development Environments

Feasibility Criteria	Wt.	Candidate 1 Oracle Portal Tools	Candidate 2 Borland JBuilder Enterprise	Candidate 3 Metrowerks CodeWarrior	Candidate 4 NetBeans
Operational Feasibility (Functionality)	20%	Fully supports all user requirements, but cannot create a stand-alone product. May require a lot of training for end users. Score 80	Fully supports all user requirements. Can create a stand-alone product. End product may require less end user training. Score 100	Fully supports all user requirements. Can create a stand-alone product. End product may require less end user training. Score 100	Fully supports all user requirements. Can create a stand-alone product. End product may require less end user training. Score 100
Technical Feasibility	20%	Have experience with Portal. Difficult to install. Requires an Oracle DB back-end. Score 50	RAD tools. Can be used with all candidate database solutions. Uses my Java experience. Score 100	RAD tools. Can be used with all candidate database solutions. Uses my Java experience. Score 100	No RAD tools. Can be used with all candidate database solutions. Uses my Java experience. Score 100
Economic Feasibility Cost to Purchase Licenses to Deploy and Distribute final product	55%	(Budget $100) $5000+ (inc. with Portal) $0 Score 0	(Budget $100) $400 (academic price) $0 Score 70	(Budget $100) $110 (academic price) $0 Score 90	(Budget $100) Free (Open Source) Free Score 100
Schedule Feasibility (Can I keep to my Project Schedule)	5%	Score 70	Score 90	Score 90	Score 90
Ranking	100%	Score 29.5	Score 83	Score 94	**Score 99.5**

2.2.3 Database Solutions

Four candidates were considered in the database category: 1) Oracle 9i Enterprise Edition, 2) Borland JDataStore, 3) Microsoft SQL Server 2000, and 4) MySQL.

2.2.3.1 Oracle 9i Enterprise Edition

The first candidate was Oracle 9i Enterprise Edition. There were three main benefits to using this candidate. First, this was the most powerful of the database candidates. Next, the student was very familiar with Oracle 9i

Enterprise Edition, and she already had an academic developer's license for the product. Lastly, this solution would work with any of the web-based or Oracle-based implementation solutions. The only real barrier to using this solution was the fact that it would be extremely expensive to deploy.

2.2.3.2 Borland JDataStore

The second candidate database was Borland JDataStore. This solution comes built-in with Borland JBuilder 8 Enterprise Edition, but can be purchased by developers as a separate product for approximately $40. JDataStore was the only solution that could be used to create a stand alone product, because the database could be imbedded in the application. This would be much cheaper than deploying JDataStore on the web; however there would still be the problem of application installation, addressed under the implementation solutions.

2.2.3.3 Microsoft SQL Server 2000

The next candidate was Microsoft SQL Server 2000. The benefits of this solution included the facts that 1) it could be coupled with any of the web-based Java implementation solutions, and 2) the student already had a developer's license for SQL Server. Unfortunately, SQL Server is also very expensive to deploy.

2.2.3.4 MySQL

The last candidate was the open source MySQL database. There would be three main benefits to using MySQL: 1) MySQL is free, 2) it would be able to do nearly everything the more expensive candidates could do, and 3) this solution could be coupled with any of the web-based Java implementation

solutions. The only barrier found was the fact that MySQL does not support the use of foreign keys for data validation. However, the student determined that foreign key data validation was not critical to the project, because data validation could be implemented in the application.

2.2.3.4 Candidate Systems Matrix - Database Solutions

Feasibility Criteria	Wt.	Candidate 1 Oracle 8i(9i) Database	Candidate 2 Borland JDataStore	Candidate 3 Microsoft SQL Server 2000	Candidate 4 MySQL
Operational Feasibility (Functionality)	20%	Fully supports all user requirements. Score 100	Fully supports all user requirements. Score 100	Fully supports all user requirements. Score 100	Fully supports all user requirements. Score 100
Technical Feasibility	20%	Can be used with any of the application implementation & development solutions. Would require users to have Internet access. Score 80	Can be used with any of the Java application implementation solutions, and all of the application development solutions except Oracle. Can be imbedded in the application to create a stand-alone application. Score 100	Can be used with any of the Java application implementation solutions, and all of the application development solutions except Oracle. Would require users to have Internet access. Score 100	Can be used with any of the Java application implementation solutions, and all of the application development solutions except Oracle. Does not support Foreign Keys. Would require users to have Internet access. Score 90
Economic Feasibility Cost to Purchase Licenses to Deploy and /or Distribute final product	55%	(Budget $100) $15,000+ $0 Score 0	(Budget $100) $0 with JBuilder $40 dev. license $25 per copy or approx. $400 for web deployment Score 70	(Budget $100) $0 (have developer lic) $5000+ Score 0	(Budget $100) $0 $0 For non-commercial use Score 100
Schedule Feasibility (Can I keep to my Project Schedule)	5%	Score 100	Score 100	Score 100	Score 100
Ranking	100%	Score 41	Score 83.5	Score 45	**Score 98**

2.2.4 Database Application Web Hosting

This category was optional, as only the Java implementation solutions would require a database application web host.

As such, all of the candidates would be able to provide all of the services required by a Java database application, namely database hosting, a JVM (Java Virtual Machine) engine and a Java servlet engine, as well as sufficient disk storage and monthly bandwidth. All the candidates also provided domain registration, which would be required, and used Linux servers.

The budget of $100 for this category was for the cost of web hosting for the duration needed for development, testing, and deployment of the project, documenting the deployment, and presenting the project, an estimated 6 months.

2.2.4.1 jspzone.net

The first web host candidate was jspzone.net. This host allowed a respectable 800MB of disk storage, 20GB of monthly bandwidth, and provided a 99.7% uptime guarantee. They also guaranteed daily web site and system backups, and used Tomcat for the servlet engine. This was the second least expensive of the candidates profiled here, but would overrun the budget by about 17%.

2.2.4.2 lunarpages.com

The next candidate was lunarpages.com. This host tied for the most disk storage allowed at 1000MB, and had the best monthly bandwidth allowance at 40GB. They used the servlet engine Resin and the older J2SE 1.4.1. The

student was much more familiar with the Tomcat servlet engine than Resin, however Resin appeared to be very similar. Lunarpages also guaranteed 99.9% uptime. This was the least expensive candidate, and but would still overrun the budget by about 1%.

2.2.4.3 assortedinternet.com

The next candidate was assortedinternet.com. This host would provide the lowest disk storage allowance at 125 MB and lowest bandwidth allowance at 5000MB, with a guarantee of 99% uptime. This host used Tomcat, which the student is familiar with. This was the most expensive candidate.

2.2.4.4 cwihosting.com

The last host is cwihosting.com. This host tied with lunarpages.com for the most disk storage allowed (1000MB) and had the second best monthly bandwidth allowance at 25GB.They used Resin with JDK 1.4.2. They also claimed a 100% uptime guarantee. This was the second most expensive candidate.

2.2.4.5 Candidate Systems Matrix – Database Application Web Hosting

Feasibility Criteria	Wt.	Candidate 1 jspzone.net	Candidate 2 lunarpages.com	Candidate 3 assortedinternet.com	Candidate 4 cwihosting.com
Operational Feasibility (Functionality)	20%	Fully supports all user requirements. Score 100	Fully supports all user requirements. Score 100	Fully supports all user requirements. Score 100	Fully supports all user requirements. Score 100
Technical Feasibility	20%	800MB storage 20GB bandwidth Unlimited email with webmail Uses Tomcat 99.7% uptime Score 100	1000MB storage 40GB bandwidth Unlimited email with webmail Uses Resin Uses older JDK 99.9% uptime Score 95	125MB storage 5000MB bandwidth Unlimited email and webmail Uses Tomcat 99% uptime Score 50	1000MB storage 25GB bandwidth 55 pop3 email accounts Uses Resin 100% uptime Score 100
Economic Feasibility **Domain Registration** **Hosting Cost** **Setup Fees** **Total for 6 months**	55%	(Budget $100 for 6 months) $15.00 year - $203.88 year or $50.97 quarterly or $16.99 month - Free $116.94 - Score 85	(Budget $100 for 6 months) $14.95 year - $119.40 year or $65.70 bi-yearly or $35.85 quarterly - $20 unless subscribe to yearly plan $100.65 - Score 100	(Budget $100 for 6 months) $15 year - $299.88 year or $29.99 month - Free $194.94 - Score 0	(Budget $100 for 6 months) $25 year - $175 year + $3 month for jsp/Resin or $14.95 month + $3 month - $20 unless subscribe to yearly plan $132.70 - Score 70
Schedule Feasibility (Can I keep to my Project Schedule)	5%	Score 100	Score 100	Score 100	Score 100
Ranking	100 %	Score 91.75	**Score 99**	Score 35	Score 83.5

2.3 Winning Solutions

After careful consideration, the best available solution was selected from the candidates for each of these four categories: 1) Application Implementation Technologies, 2) Integrated Development Environment (IDE) Solutions, 3) Database Solutions, and 4) Database Application Web Hosting.

2.3.1 Application Implementation Technology Solution

The student chose Java with the Struts and iBATIS frameworks for the application implementation technology solution. She selected this solution for several reasons. First, Struts has become a standard for Java web development and incorporates many best practices and Java design patterns. Next, though the frameworks would add some complexity, and would require the student to learn the frameworks, this would be balanced by the fact that both frameworks were well-tested, stable, and among other things, would make the project easier to document. Lastly, the iBATIS framework would allow all the SQL code for the project to be logically organized and stored in one place.

2.3.2 Integrated Development Environment Solution

Since one of the Java candidates was selected for the implementation technology solution, an IDE that supported Java development was required for the IDE solution. The student chose NetBeans. There were several benefits to using NetBeans, such as faster development, but the primary reason it was selected over the other candidates was the fact that it was free.

2.3.3 Database Solution

After careful consideration, the student chose MySQL for the database solution. She selected this solution for several reasons. First, many database application web hosts include MySQL databases in the price of hosting. Next, setup of a MySQL datasource would be very easy, and would not require any special tasks. Lastly, deployment of MySQL databases was free. The student did have some reservations about MySQL, because it would not support the use of

views or foreign keys. However, she decided that since she would be using Java JDBC for any updates to the database, she could enforce the keys programmatically.

2.3.4 Database Application Web Hosting Solution

The student selected lunarpages.com to host the project. This solution was chosen because lunarpages.com would provide by far the best value for the money. The student was somewhat concerned about using the Resin JVM, since she had never used it before, but decided that this was a minor issue.

2.4 Methods of Research

2.4.1 User Interviews

In order to have a group of users for guidance and prototype testing purposes, the student formed a team of volunteers who were: 1) actively following a rotation diet or had been following a rotation diet in the past two years, 2) computer literate, and 3) able to test the application. This user team was the source for all user interviews, and provided the user feedback that was used to revise the project requirements after major testing milestones.

2.4.2 World Wide Web

The majority of the research that was done on the project was conducted online, at software publisher's web sites, online stores, academic and technical library sites, and web hosting provider sites (web hosts). Only the best candidate solutions were included in the feasibility analysis, but many more were researched.

2.4.2.1 Software Publishers

Software publisher web sites were visited to determine the features and manufacturer's suggested retail price of software candidates.

The software publisher web sites that were visited include:

Table 2: Software Publishers

Publisher	Product(s) Researched	URL
Apache Software Foundation	Struts	http://www.apache.com
Borland Corporation	JBuilder Enterprise Edition; JDataStore	http://www.borland.com
Eclipse Foundation	Eclipse	http://eclipse.org
Helios Software Solutions	Textpad	http://www.textpad.com/
iBATIS	iBATIS framework	http://www.ibatis.com
JetBrains	IntelliJ	http://www.jetbrains.com/idea/
Metrowerks	CodeWarrior Pro	http://www.metrowerks.com
Microsoft Corporation	SQL Server	http://www.microsoft.com
MySQL AB	MySQL	http://www.mysql.com
NetBeans	NetBeans	http://www.netbeans.org
Oracle Corporation	Oracle 9i Enterprise Edition; Oracle Portal	http://www.oracle.com
PostgreSQL Global Development Group	PostgreSQL	http://postgresql.org
Sun Microsystems	Java	http://java.sun.com
Xinox Software	JCreator	http://www.jcreator.com/

2.4.2.2 Online Stores

Online stores were visited to determine the retail and/or academic prices of the software researched at the software publisher's sites. Stores visited include:

Table 3: Online Stores

Site	URL
Amazon.com, Inc.	http://www.amazon.com
Edu Tech Store	http://www.edu.com
Efollet Software Store	http://www.efollett.journeyed.com
SAM'S West, Inc	http://www.samsclub.com

2.4.2.3 Academic and Technical Web Sites

The academic and technical web sites were the source for all of the journal articles and some of the books used during the course of the project.

The primary sites visited were:

Table 4: Academic and Technical Sites

Site	Type	URL
ONJava.com	Technical Web Journal	http://www.onjava.com/
O'Reilly Network Safari Bookshelf	Subscription Technical Library	http://safari.oreilly.com/
Regis University Libraries Online	Academic Library	http://www.regis.edu/libdatabase.asp?sctn=lib&p1=empty
TheServerSide	Technical Web Journal	http://www.theserverside.com/

2.4.2.4 Web Hosts

The web site hosting provider sites were visited to research the costs of the various feature and options that would be needed when the project was finally deployed online.

Web hosts visited include:

Table 5: Web Hosts

Web Host	URL
Add2Net, Inc.	http://lunarpages.com
Assorted Internet Ventures, LLC	http://assortedinternet.com
CWIHosting.com	http://cwihosting.com
Iniquinet.com	http://iniquinet.com
JavaServletHosting.com	http://javaservlethosting.com
JSPZone.net	http://jspzone.net
KGB Internet Solutions	http://kgbinternet.com
Oxxus.net	http://oxxus.net

2.5 Summary

The project required extensive research. The bulk of the research was conducted to discover the requirements of the project and to determine candidate solutions for the project.

User interviews were conducted using a team of volunteers for requirements discovery, and the internet was used primarily in the search for candidate solutions for the project.

Four categories of candidate solutions were researched: 1) Application Implementation Technologies, 2) Integrated Development Environments, 3) Database Solutions, and 4) Database Application Web Hosting. The winning solutions for the project in each of the four categories were determined with a feasibility study using candidate systems matrices.

3 Project Methodology

3.1 Rapid Application Development

As mentioned in the previous chapter, the student felt that the most appropriate methodology for the project would be some form of Rapid Application Development (RAD).

According to Whitten, et al., the iterative phases of RAD create a development spiral or "prototyping loop" that continues "until the prototype is considered a 'candidate system' for implementation" (98-100). Some of the most important benefits of RAD are that it speeds analysis and design and actively involves users in development. Also, RAD allows for some experimentation to occur without developers having to finalize a design for a project that users may find completely unsuitable after implementation.

This experimentation proved to be critical to the success of the project; the student could not have completed the project without it. This was due in good measure to her inexperience in systems design and development, but fortunately over the course of the project the RAD methodology provided many opportunities for learning.

3.2 Phases of Rapid Application Development (RAD)

The student created a development plan somewhat loosely based on RAD, which would consist of the following phases:

1) Preliminary Investigation
2) Problem Analysis (accelerated)
3) Iterative Phases
    a. Design
    b. Construction
    c. Implementation
    d. Analysis

4) Implementation (final)
5) Operation and Support (optional)

A description of each of the phases in the development plan follows.

3.2.1 Preliminary Investigation

The preliminary investigation phase researched and defined: 1) the reasons for and scope of the project, 2) the initial requirements and constraints, including budget, and schedule, and 3) the project methodology that would be used. This phase was completed with the approval of the project proposal.

3.2.2 Problem Analysis

The problem analysis phase combined problem analysis with requirements discovery and decision analysis. During this phase formal user interviews were conducted to help determine the project requirements, and the feasibility and requirements analyses were completed.

3.2.2.1 Requirements Discovery

As mentioned in the previous chapter, in order to have a group of users for guidance and prototype testing purposes, the student formed a team of volunteers who were: 1) actively following a rotation diet or had been following a rotation diet in the past two years, 2) computer literate, and 3) able to test the application.

Once the user team was formed, she met individually with each user for an initial interview to confirm the scope of the project as well as determine the initial user requirements. The user team explained their needs and expectations for the application, including: 1) multiple lists of “safe” foods and their

corresponding rotation schedules must be allowed to be created and stored for each user, 2) meal plans and shopping lists must be printable, and 3) users should only be able to add foods to any given meal plan that are allowed by that user's rotation schedule. The full results of the interviews are compiled in User Interview Report 1 (Appendix A). This report was used as a reference in the creation of the business rules and initial application requirements, presented below.

3.2.2.1.1 Business Rules

1. Each user must be assigned a unique identification (id) number.
2. Each user must have a unique username.
3. Each user must have a password.
4. Each user must have an email address.
5. Each food must have a unique id number.
6. Each food must have a unique name.
7. Each food must be associated with a food family.
8. Each food family must have a unique id number.
9. Each food family must have a unique name.
10. Multiple food lists may be associated with each user id number.
11. Each food list must have a unique id number.
12. Each food list must have a name that is not already in use by another food list associated with the specified user id. (Name is unique only to the user's account).

13. Each food list must have a rotation schedule consisting of both the food rotation in days and food family rotation in days.
14. Multiple food list items (foods) may be associated with each food list.
15. Each food list item must be associated with a food.
16. Multiple meal plans may be associated with each food list.
17. Each meal plan must be for a specific date.
18. Only one meal plan per date may be associated with each user id number.
19. Each meal plan must have a unique id number.
20. Multiple meals may be associated with each meal plan.
21. Each meal must have a unique id number.
22. Each meal must be named, but each name must not be already in use by another meal associated with the specified meal plan. (Name is unique only within each meal plan).
23. Multiple meal items (foods) may be associated with each meal.
24. Each meal item must be associated with a food list item (in turn associated with the specified food list).
25. Only those foods that are allowed by the specified food list's rotation schedule may be added to any meal associated with the meal plan for any specified date.

3.2.2.1.1 Application Requirements

1. The food and food family tables may not be modified using the application.
2. Each user must create an account to use the application.
3. Each user must login to their account to use the application.
4. Each user can only see food lists, meal plans, and meals associated with their account.
5. Each user may create an unlimited number of food lists, meal plans, and meals.
6. Each user may modify and/or delete their food lists, meal plans, and meals as often as they wish.
7. Each display page for the application must have links to each major page of the application (Home, Meal Plans, Food Lists, and Shopping Lists).
8. Each multi-page form must contain appropriate form navigation buttons (previous, next, finish, etc).

The business rules and initial application requirements were the basis for the requirements definition report, which was updated with each iteration of the development cycle. The final Requirements Definition Report is included as Appendix B.

3.2.2.2 Feasibility Analysis

After the initial Requirements Definition Report was completed, the student conducted research to determine if there were any off-the-shelf software programs that were focused on the target user group, namely persons with

severe food allergies that had been prescribed rotation diets. She found no evidence of the availability of any such software programs, and concluded that a custom application would have to be built. Since there were no existing off-the-shelf solutions that could be used for the project; the student spent the rest of the problem analysis phase researching candidate technology solutions for the construction and implementation of a custom database application, which were presented in the previous chapter.

The student completed the initial Feasibility Analysis of the candidate technologies in March of 2004, completed the initial application design, and had begun preparations for first iteration of the construction phase.

It was at this stage that the student discovered a costly mistake in her initial feasibility study, which ultimately caused more than a month-long delay in the project schedule.

She had planned to implement the project with Oracle 9i database, Oracle 9iAS Portal, and a small amount of Java thrown in. However, she found that she had neglected to check the hardware requirements for Portal. Portal is powered by Oracle 9i Application Server (AS), which is in turn supported by an Oracle database. The student determined that her existing hardware was not powerful enough to run the AS, let alone both the AS and Portal at the same time, and could not be further upgraded. Because she had not budgeted for new hardware, a new computer would overrun her planned project budget of $400 by 200-600%. Also, the user team had indicated that if the project was successful, they would like to continue to use the completed application; however, the

licensing fees for the Oracle 9i database and Oracle Portal would be prohibitive. For these reasons, the student decided to look for alternatives to the Oracle products – which required completion of a new feasibility study.

In the course of the second feasibility analysis, the student determined that she could design and build a Java application that could replace Portal, and found several candidate IDEs (Integrated Development Environment) to speed the Java development. She also discovered several alternatives to the Oracle 9i database, and since the project would be implemented on the web, she researched several candidates for database application web hosting.

Once the second feasibility study was complete, the student reviewed her original application design, and discovered that the existing design was not going to be adequate for the project, as it could only be well implemented with Oracle Portal. Consequently, she started on a redesign in May 2004.

In the course of the redesign, while researching design patterns for Java and the web, the student discovered the Struts framework, which has become a standard for Java web application development. It incorporates many best practices and Java design patterns. However, since the Struts framework is based on the Model 2 design pattern (a variation of the Model-View-Controller (MVC) design pattern), and in addition is designed to work with any datasource (Model-neutral), the framework does not determine how the datasource is accessed (Husted et al. (39, 41). Because of this, some research on datasource frameworks that are Struts compatible was completed, and showed that the iBATIS framework would be a good fit for the project.

After the research was completed, the student revisited the second feasibility study and added a section for application implementation technologies, such as Struts. The study confirmed her belief that the Struts and iBATIS frameworks would be an excellent choice for the project implementation.

Based on the results of her research, the student determined what she felt were the best solutions for each of the four categories mentioned above:

1) Integrated Development Environment Solution: NetBeans
2) Database Solution: MySQL
3) Database Application Web Hosting: lunarpages.com
4) Application Implementation Technology

The final revision of the feasibility study was completed in May of 2004.

3.2.3 Iterative Phases

The student had planned for the project to flow through the iterative phases sequentially in a spiral, but the reality turned out to be rather different. When the project first moved into the iterative phases, there was not a clear spiral because the project did not complete a full iteration of all the phases until the project was nearly a quarter done. In retrospect, a good deal of this confusion was due to the inexperience of the student in her role as project manager, and as she became more experienced, the phases began to spiral as planned. Following is a definition of each of the iterative design phases, followed by an explanation of how these processes actually occurred during the course of the project.

3.2.3.1 Design

The design phase defined models for the application and database. These models were continually updated as the project evolved further through the other iterative phases of the lifecycle. Deliverables for this phase were the application and database models, which were revised after each iteration of the analysis phase had been completed. The final application and database models were completed after the final implementation was approved by users. The final application model is presented as Appendix E, and the final database model is presented as Appendix C.

3.2.3.2 Construction

During the construction phase, a working prototype of the database application was built, based on the most current design. The database application prototype was the only deliverable for this phase.

3.2.3.3 Implementation

The implementation phase allowed users to test and give feedback on the latest build of the application, which was then used to refine the requirements and design of the database application. The deliverables for this phase were the user feedback from the testing.

3.2.3.4 Analysis

During the analysis phase, the requirements of the project were reviewed and updated based on the user feedback from the previous phase. Deliverables for this phase were a revised requirements report.

3.2.3.5 Explanation

As discussed in chapter 2, the student had decided to implement the application with Struts. Because Struts is based on the MVC design pattern, the student decided to design the project in the order suggested by Turner and Bedell in their book Struts Kick Start (89-105). She first designed the View (web pages), next the Model (database), and then the Controller (the application itself).

The first step was to map out and design the web pages that would form the views (jsp pages) required by the Struts application. The student drew up use cases for the project, which were used to determine the pages that would be needed, as well as the inputs and outputs required for those pages. Within each use case, the pages needed to fulfill that use case were mapped out and then designed. The use cases are presented in Appendix H.

The next step, designing the Model, was begun by determining the tables that were needed for the database. Next, the student normalized the database tables that she had created, and prepared a database design matrix. From this design matrix, she created the initial database Entity Relation Diagram (ERD).

After the Model had been designed, the student began working on the Controller (application) design while studying the use of the Struts framework. She had been taught to use UML (Unified Modeling Language) modeling for Java programming, but Struts applications cannot be easily modeled in UML, and after much frustration she came to the realization that she did not understand how Struts worked well enough to create any sort of Controller design at that point. As a result, the student decided to construct a trial version of the web pages and

Java classes required for user account creation and login, which would use a test version of the database. The experiment took about two weeks to complete, and enabled the student to see how all of the Struts components worked together; however, it also revealed a serious flaw in the initial database design.

The original database design would have required the application to create one or more tables in the database for each user, containing data such as the user's lists of acceptable foods, meal plans, and the dates each food was last eaten. However, during the Struts experiment the student quickly realized that 1) the iBATIS framework does not directly support database table creation, 2) a workaround could be implemented but would compromise the framework's built-in data security features and explode the size of the database, and 3) having to create new tables at all instead of using existing tables was a strong indication that the database was poorly designed.

Once the database had been redesigned and normalized, it was again time to undertake the Controller design. The student decided that instead of creating UML models, she would put together an Application Specification document, based on the application plan suggested by Husted, et al., in their book Struts in Action (87-89). This document is intended to track the various Struts components, including the jsp pages used for the views, the ActionForms that are passed from the views to the application, and the Action classes that make up the application itself. The final Application Specification document is included as Appendix D. This document was used as a guide for the remainder of the project, and was updated as needed.

Construction then began in earnest. As each section of the project was completed, the student implemented and tested the entire project to that point. The users were called on for tests only at major milestones, usually about once every two months.

The Iteration phases were concluded in December 2004.

3.2.4 Implementation (final)

The final implementation would be complete when the working prototype had been approved by the users, and the final build of the database application had been deployed. Deliverables for this phase were the finished design models, final requirements report, and the final build of the database application.

The final implementation was completed on January 9, 2005, with the deployment of the final build of the application to rotationdietproject.com, at the web host lunarpages.com. The final user interview is documented in Appendix F, and a small sample of the views for the project is included in Appendix I.

3.2.5 Operation and Support

The final phase of the development lifecycle would consist of maintaining the system in an operational state and providing support for users. However, this phase fell outside the scope of this project, which is focusing on development, so no deliverables were required.

3.3 Summary

The right choice of methodology proved to be critical to the success of the project. The iterative phases of the RAD methodology allowed experimentation

that gave the student time to find the best solution for the project by learning from her mistakes. RAD methodology had seemed the best choice for the project, and over the course of the project this proved to be true.

4 Project History

4.1 Beginning the Project

In October, 2002, the student received approval to begin work on her professional project, “Development of a Personal Diet Plan Database Application for Persons with Severe Food Allergies”. She felt that she was not ready to undertake the project at that time, and took several classes before she actually started the project in March of 2004.

The goal of this project was to research, analyze, design, and implement a database application that would assist patients in creating a personal diet plan based upon a “rotation diet.” The application was intended to allow patients to create a broader range of meals than currently possible with existing manual methods, and also to quickly and accurately make up shopping lists for their planned meals. The project would be considered successful if the user team felt that these goals were met.

4.2 Managing the Project

The student acted as the sole personnel resource, and her only experience as a project manager was in a classroom setting. This proved to be a definite disadvantage for the project, and many of the delays that occurred were due to inexperience and a lack of confidence.

In addition, since the student was also the systems analyst, designer, and developer, there was no-one to, for example, catch design mistakes before the developer could implement them. There were several situations in which this was

a problem; if there was a mistake it was all hers and she would have to fix it herself. This led to yet more delays.

Fortunately, as the student gained experience and confidence in her abilities in all the roles, the project went more smoothly. Admittedly, however, the user team played a large role in helping her gain confidence, because they were pleased with her work and very happy to help test the application.

4.3 What Went Right

A couple of months after the student had begun working on the project, if she had been asked what was going right with the project, she would have replied, “Nothing is going right.” Fortunately, this was not the case in the long term.

The Struts and iBATIS frameworks proved to be an excellent choice for the application, and made the development process less of a chore than it might have been. This was particularly important because of the inexperience of the student. It is very likely that if she had not found the Struts framework, she would not have been able to complete the project due to its complexity.

The volunteer user testing team was extremely positive and graciously forgiving of mistakes. Their contribution to the project was more important to the project than the student initially thought it would be. The user team was crucial to the requirements discovery process; in testing the application they not only helped find the bugs in the project but also put forward important suggestions that greatly improved the usability of the application.

4.4 What Went Wrong

So many things went wrong with the project that it is difficult to know where to begin. Most of the problems that occurred in the course of the project would have been avoided had the student had more experience in all her roles, particularly the project management and developer roles.

4.4.1 Hardware Issues

As mentioned in chapter 3, the student made a serious mistake in her first feasibility study. She had neglected to check the system requirements for the Oracle products she had intended to use for the project development and implementation. The desktop computer that she had intended to host the Oracle products was simply underpowered for the task, and furthermore, the operating system required was incompatible with her other computer. Purchasing a new computer was not an option, because it had not been included in the budget of $400.

The cost of this mistake was paid primarily in time; the project was completed in January 2005 instead of August 2004. This was for several reasons: 1) Research had to be conducted and another feasibility study completed to find alternatives to the Oracle products; this took nearly a month. 2) The student had to learn to use the selected alternatives, in particular the Struts and iBATIS frameworks, which took another month. 3) She was less experienced with Java in general than the Oracle development tools, which delayed the project another two months.

4.4.2 Development Issues

The major development issues encountered were a direct result of the fact that the student was not an experienced Java developer, furthermore, all of her Java experience was classroom based instead of real-world.

The first development issue that was encountered was the fact that the Struts framework is not a good candidate for UML based development, which was the primary method that the student had been taught in class. In fact, Struts was very difficult to model in full by any existing method. This is mainly due to the fact that the framework is based on the MVC design pattern; each of the three components (Model, View, and Controller) is completely separate from the others, and their primary interface is hidden by the XML Struts configuration file.

The second development issue was the fact that the current release of the iBATIS framework did not support a small number of standard SQL commands. In most cases, this did not present much of a problem, as alternate commands could be used. However, the inability of the framework to allow exclusive queries of the database became a problem during the development of the Java classes dealing with the rotation logic. Instead of using one exclusive query, for example, to find all of the available foods that were *not* in food families that had already been eaten within a specific time period, the student had to make three inclusive queries and sift them with Java to come up with the same data that would have been received from the single exclusive query.

The final development issue was the fact that the student had no real practical experience in Java programming. All her experience was in the

classroom, and was limited to projects that could be completed in 5 weeks or less. Fortunately, the decision to use the Struts framework helped alleviate most of the difficulties that might have arisen, because Struts was designed for simplicity.

4.4.3 Deployment Issues

There was only one real deployment issue that was dealt with, which was again primarily related to the student's inexperience.

When the student had purchased the web hosting package at lunarpages.com, she knew that the Resin servlet runner that would power the Java portion of the project was somewhat different that the Tomcat 5.0 servlet runner she was using for development, but thought that this would not be much of a problem since the current version of Resin supported all the standards required by her project. Unfortunately, she neglected to make sure that the web host was using the current version of Resin; they were not. The older version of Resin that the web host was using had two major problems: 1) it did not support the new Java JSP 2.0 expression language that her current jsp pages required, and 2) it was not as forgiving of errors in XML and JSP files as Tomcat; minor errors that Tomcat could ignore were causing fatal servlet errors.

The result of this was approximately two days spent doing nothing but recoding her jsp web pages to use the older JSP 1.2 tags instead of expression language, as well as several hours hunting down minor errors in the project's XML and JSP files.

4.5 Milestones

The project proposal was approved in May of 2002, but the project was not actually begun until March of 2004. RAD was chosen as the project methodology in March, and the first full spiral of the iterative phases of RAD was completed in July.

Also in July, the user team completed their first test of the application. Only the food list section of the application was functional at this time, but it gave the users an idea of how the finished application would look and function. They were pleased, but had a few suggestions that were incorporated into the next design iteration for the project.

After the first user test was completed, the student took a hard look at the Java packages and JSP pages within the application in light of the suggestions that the users had made. She decided that the Java packages and JSP pages were badly in need of reorganization. The reorganization was finished in early August, and greatly benefited the project because it made it easier to find a specific class or page for editing and/or debugging.

In October, further user testing was completed on the meal plan and shopping list sections of the application. Also, the student completed the final test of the logic for the rotation schedule. This was a critical milestone, because the logic for the rotation schedule had to be correct for the project to meet its goal of allowing users to create meal plans based on a rotation schedule.

Once the logic was deemed correct, the project was tested for web readiness and fully deployed online in November. After deployment, the first

round of online user testing of the entire application was completed. The users were mostly pleased but found several problems and short-comings with the application, which were addressed.

After the problems and short-comings found in the previous testing had been addressed, the final round of online user testing was completed January 5, 2005. The user team was very pleased with the application, and felt that it was ready for final deployment.

The final build of the application was deployed on January 9, 2005.

Table 6: Project Milestones

Milestone	Date
Project proposal approved	May 2002
Project begun	March 03, 2004
RAD chosen as the project methodology	March 2004
First full RAD spiral completed	July 5, 2004
Completed user testing of the Food List section of the project	July 31, 2004
Reorganization of Java packages and jsp pages	Aug 9, 2004
Completed user testing of the meal plan section of the project	Oct 02, 2004
Completed user testing of the shopping list section of the project	Oct 09, 2004
Final Rotation logic test completed	Oct 10, 2004
Web deployment readiness tests completed	Oct 20, 2004
Deployed database online	Oct 29, 2004
Application deployed online	Nov, 12, 2004
Completed first round of online user testing of the entire application	Nov 29, 2004
Completed final round of online user testing of the entire application	Jan 05, 2005
User team approves the final version of the project	Jan 05, 2005
Final deployment of application completed	Jan 09, 2005

4.6 Project Summary

The goal of the project was to research, analyze, design, and implement a database application that would assist patients in creating a personal diet plan based upon a “rotation diet.” The project was begun in March of 2004, and was

successfully concluded in January 2005 when the user team approved the final build of the application.

In spite of poor management, false starts, and numerous problems, the project was a very positive learning experience, and the student was very pleased with the successful outcome of the project.

5 Lessons Learned

5.1 Project Experience

I learned a great deal from this project. I feel that my project management skills have improved somewhat and that my systems analysis and design, as well as my Java programming skills have improved tremendously. On a personal level, I think that I am more patient when I cannot seem to find a solution for a problem, and more methodical in finding an answer.

Overall, I feel a great sense of accomplishment, because in spite of problems, confusion, and a false start, the project was more successful than I dreamed possible. I am very grateful to my user team because only their support and encouragement made it possible for me to complete this project.

5.2 Expectations

The project was able to meet all expectations, and in actuality exceeded the student's initial vision for the project when it was proposed in May of 2002.

The goals of the project, as stated in chapter 1, were the development of a database application that would:

5.2.1 Assist patients in creating their personal diet plan based upon a rotation diet.

This goal was met. According to the testers, the application was of great assistance in making up accurate meal plans for their person diet plan.

5.2.2 Allow patients to enjoy a broader range of meals

This goal was met. The testers felt that the application made it much easier for them to easily create more interesting meals for their diet plans, because they did not have to keep track of the rotation schedule themselves.

5.2.3 Allow patients to quickly and accurately make up shopping lists for the meals.

This goal was met. The testers agreed that the application made it very easy to make up shopping lists because all they had to do was select the date(s) they wanted to shop for. The application would then automatically make up a shopping list containing a list of all the foods that would be eaten on those date(s), including the number of meals that each food appeared on.

5.3 Next Evolution

There are several things that could be done to improve the application. For example, some graphics could be added, to make the application's view pages look nicer. It would also be nice to make shopping lists modifiable, so that users could adjust the amount of food they want to buy right on the screen instead of manually. Both the application and database could be updated to make better use of the meal_names table, and also to allow user initiated account deletions. The datasource connection Java classes could be more efficient and error handling could be improved.

The user team also had several excellent suggestions, which included a personal weight tracking module, and a food information reference containing information about each food such as recommended serving size, fat content, etc.

Another possibility would be making the application available online; perhaps on a donation basis, or with a monthly or yearly user fee to offset the costs of support and maintenance.

5.4 Conclusions

Despite a number of problems, the goals that were set at the outset of the project were successfully met. The student learned a great deal in the process of completing the project, and the user team was pleased with the results.

5.5 Summary

Overall, the project experience was very positive, and it gave the student opportunities to learn a great deal about project management and application development, and also to experience personal growth by working through seeming insurmountable difficulties to produce a successful project.

Bibliography

Barker, R. CASE*METHOD Entity Relationship Modelling. United Kingdom: Addison-Wesley Publishing Company, 1990.

Cavaness, C., B. Keeton. Jakarta Struts Pocket Reference. USA: O'Reilly & Associates, Inc., 2003.

Deitel, H.M., P.J. Deitel. Java How to Program. USA: Pearson Education, Inc., 2003.

Husted, T., et al. Struts In Action. USA: Manning Publications Co., 2003.

Kline, K. SQL in a Nutshell. USA: O'Reilly & Associates, Inc., 2000.

Reese, C., (2000). Database Programming with JDBC and Java. USA: O'Reilly & Associates, Inc., 2000.

Richter, C. Designing Flexible Object-Oriented Systems with UML. USA: Macmillan Technical Publishing, 1999.

Schach, S. Object-Oriented and Classical Software Engineering. USA: McGraw-Hill Higher Education, 2002.

"MySQL Manual". Official MySQL Web Site. 2004. MySQL AB. 03 Jan. 2005 <http://dev.mysql.com/doc/mysql/en/index.html>.

Stevens, P., R. Pooley. (2000). Using UML: Software Engineering with Objects and Components. United Kingdom: Pearson Education Limited, 2000.

Turner, J., K. Bedell. Struts Kick Start. USA: Sam's Publishing, 2002.

White. S., et al. JDBC API Tutorial and Reference: Universal Data Access for the Java 2 Platform. USA: Sun Microsystems, Inc., 1999.

Whitten, J. L., L. D. Bentley, K.C. Dittman. Systems Analysis and Design Methods. USA: Irwin/McGraw-Hill, 1998.

Appendices

Appendix A: User Interview 1 Report

1. What, in your opinion, is the main problem with the way you currently make up rotation menus?
    a. It is very hard to write out a menu that covers a period of time without making mistakes related to food family and rotation.
    b. Too much work.
    c. It's too time consuming.
    d. Mistakes cause cascading problems with the rotation, which often require a complete redo of the menu to fix.
    e. It is hard to find an accurate, easy to use list of what foods are in a food family.
    f. It is difficult to get all the food groups into the diet every day.

2. Thinking in general about a computer program that would make creating rotation menus and shopping lists from those menus, what features would be most important to have?
    a. Accurate food/food family lists.
    b. To be able to make multiple, customizable food lists (for each person in the family), because not everyone can eat the same foods.
    c. A customizable rotation schedule for each person in the family: not just 3/5 day rotation, need any combination rotation, including none.
    d. Rotation options should be clearly explained, for example, what will 3/5 mean in the program M/W/F, or M/F/Sun?
    e. Must be able to print out and save menus and shopping lists.
    f. Must be simple to use/idiot proof.
    g. When selecting foods for daily menus, foods/food families that would violate the rotation should not be available for selection, and preferably not even visible.

3. What features would be nice to have, but could be done without?
    a. An option for tracking specific nutritional information such as fat, carbohydrate, protein, and caloric content for each day on a menu.
    b. It would be nice to be able to store my weight, etc., in the computer to track how well the diet is going.
    c. An option to remove foods that you do not need to buy from the shopping list before printing.
    d. Printing options for menus/shopping lists, so I can print only the number of days/weeks that I need to plan for. Such as all of next week's menus, but no shopping list, or today's menu and the shopping list for the next three days. (interviewer note: This would require saving the menus in the database)
    e. It would be nice if it was a stand alone program, so we didn't have to buy MS Access, for example, to use it.
    f. I would like to use the program on my Mac.

Appendix B: Requirements Report

1. Introduction
   1.1. Purpose
      1.1.1. The purpose of this document is to explain the user requirements for a personal diet plan database application for persons with severe food allergies.
   1.2. Background
      1.2.1. The “rotation diet” is specifically designed for patients with severe food allergies, and requires that a patient may only eat a particular food once every *n* days (where *n* is any number), and foods from the same genus family every *n* days. For example, on a 3/5 day rotation diet, if a patient eats broccoli (Mustard family) on Monday, they cannot eat broccoli again until Sunday (five days between), but may eat something else from the Mustard family, such as cauliflower, on Thursday (three days between).
      1.2.2. Currently, patients use either pen-and-paper or a spreadsheet to create weekly or monthly meal plans for the rotation diet plan. The meal plans are usually then transferred by hand to their daily or weekly shopping lists.
   1.3. Scope
      1.3.1. The scope of these requirements is the necessary features that must be included in the application, as well as development concerns such as the budget.
   1.4. Definitions and Acronyms
      1.4.1. Definitions:
         1.4.1.1. Rotation Diet: A diet for patients with severe food allergies, designed to prevent them from becoming allergic to the foods they are not already allergic to.
         1.4.1.2. Food: Refers to a specific food, such as apples or fish, not food in general.
         1.4.1.3. Food Family: Refers to the biological family of a specific food. For example, apples belong to the Rose family, and tomatoes to the Nightshade family.
         1.4.1.4. Food List: A list of all the foods that a particular person can eat.
         1.4.1.5. Rotation Schedule: Schedule that determines when a particular food may be eaten, such as 3/5 day or 4/8 day.
            1.4.1.5.1. The numbers notated in the schedule refer to the number of days *between* any day a particular food or food family is eaten and the next day the food/food family may be eaten.
            1.4.1.5.2. Example: on a 4/7 day rotation schedule, if the patient eats apples on Monday, another food from the Rose family may be eaten on Saturday (four days between), but apples may

not be eaten again until the next Tuesday (7 days between). (See point 1.2.1 above for another example.)

1.4.1.6. Meal Plan: Planned meals created for a specific date, using a rotation schedule.

1.4.1.7. Shopping List: List of foods needed for the selected meal plans.

1.4.1.8. Account: Allows the user to securely access and store their personalized food lists, rotation schedule, etc. without worries of another user accidentally (or maliciously) modifying information.

1.4.1.9. Login: The user's unique Username and password combination.

1.5. References

1.5.1. The iBATIS framework: online: http://www.ibatis.com

1.5.2. The Struts framework: online: http://struts.apache.org

1.5.3. Sun Java: online: http://java.sun.com

2. General Project Descriptions

2.1. System Objectives

2.1.1. The system shall consist of a database application that will assist patients in creating a personal diet plan based upon a "rotation diet."

2.1.2. The application will allow patients to create a broader range of meals than currently possible, and also to quickly and accurately make up shopping lists for their planned meals.

2.1.3. The application will be accessible from a standard web browser without any special configuration of the browser required.

3. Requirements and Constraints

3.1. Business Rules

3.1.1. Each user must be assigned a unique identification (id) number.

3.1.2. Each user must have a unique username.

3.1.3. Each user must have a password.

3.1.4. Each user must have an email address.

3.1.5. Each food must have a unique id number.

3.1.6. Each food must have a unique name.

3.1.7. Each food must be associated with a food family.

3.1.8. Each food family must have a unique id number.

3.1.9. Each food family must have a unique name.

3.1.10. Multiple food lists may be associated with each user id number.

3.1.11. Each food list must have a unique id number.

3.1.12. Each food list must have a name that is not already in use by another food list associated with the specified user id. (Name is unique only to the user's account).

3.1.13. Each food list must have a rotation schedule consisting of both the food rotation in days and food family rotation in days.

3.1.14. Multiple food list items (foods) may be associated with each food list.

3.1.15. Each food list item must be associated with a food.

3.1.16. Multiple meal plans may be associated with each food list.

3.1.17. Each meal plan must be for a specific date.

3.1.18. Only one meal plan per date may be associated with each user id number.
3.1.19. Each meal plan must have a unique id number.
3.1.20. Multiple meals may be associated with each meal plan.
3.1.21. Each meal must have a unique id number.
3.1.22. Each meal must be named, but each name must not be already in use by another meal associated with the specified meal plan. (Name is unique only within each meal plan).
3.1.23. Multiple meal items (foods) may be associated with each meal.
3.1.24. Each meal item must be associated with a food list item (in turn associated with the specified food list).
3.1.25. Only those foods that are allowed by the specified food list's rotation schedule may be added to any meal associated with the meal plan for any specified date.

3.2. Functional Requirements
3.2.1. The food and food family tables may not be modified using the application.
3.2.2. Each user must create an account to use the application.
3.2.3. Each user must login to their account to use the application.
3.2.4. Each user can only see food lists, meal plans, and meals associated with their account.
3.2.5. Each user may create an unlimited number of food lists, meal plans, and meals.
3.2.6. Each user may modify and/or delete their food lists, meal plans, and meals as often as they wish.
3.2.7. Each display page for the application must have links to each major page of the application (Home, Meal Plans, Food Lists, and Shopping Lists).
3.2.8. Each multi-page form must contain appropriate form navigation buttons (previous, next, finish, etc).
3.2.9. Inputs (inputs received from user unless otherwise noted)
3.2.9.1. New users must create an account containing
3.2.9.1.1. Unique username
3.2.9.1.2. Password
3.2.9.1.3. Email address
3.2.9.1.4. Confirm email address
3.2.9.2. User's Login
3.2.9.2.1. Username
3.2.9.2.2. Password
3.2.9.3. Lost Login info
3.2.9.3.1. Email address
3.2.9.4. Creating food list:
3.2.9.4.1. User's id number (from application)
3.2.9.4.2. Rotation schedule
3.2.9.4.2.1. Family rotation

3.2.9.4.2.2. Food rotation
3.2.9.4.3. Name of the food list
3.2.9.5. Selecting food list:
3.2.9.5.1. User's id number (from application)
3.2.9.5.2. Desired food list
3.2.9.6. Modifying food list
3.2.9.6.1. User-selected food list
3.2.9.6.2. Id of selected food list (from application)
3.2.9.6.3. User's id number (from application)
3.2.9.6.4. Foods that may be eaten by the user
3.2.9.6.5. Foods that the user no longer wants in the list
3.2.9.7. Deleting food list
3.2.9.7.1. User-selected food list
3.2.9.7.2. Id of selected food list (from application)
3.2.9.8. Creating Meal Plans
3.2.9.8.1. User-selected food list
3.2.9.8.2. Id of selected food list (from application)
3.2.9.8.3. Start date
3.2.9.8.4. Number of days planning meals for
3.2.9.9. Modifying Meal Plan
3.2.9.9.1. User-selected food list
3.2.9.9.2. Id of selected food list (from application)
3.2.9.9.3. User-selected meal plan
3.2.9.9.4. Id of the selected meal plan
3.2.9.10. Selecting Meal Plan
3.2.9.10.1. User-selected food list
3.2.9.10.2. Id of selected food list (from application)
3.2.9.10.3. Desired meal plan
3.2.9.11. Deleting meal plan
3.2.9.11.1. User-selected food list
3.2.9.11.2. Id of selected food list (from application)
3.2.9.11.3. User-selected meal plan
3.2.9.11.4. id of selected meal plan
3.2.9.12. Creating Meals
3.2.9.12.1. User-selected meal plan OR list of meal plans (from application)
3.2.9.12.2. Names of meals to be created
3.2.9.13. Modifying Meals
3.2.9.13.1. User-selected food list
3.2.9.13.2. Id of selected food list (from application)
3.2.9.13.3. User-selected meal OR list of meals (from application)
3.2.9.13.4. Id(s) of meal(s) to be modified
3.2.9.13.5. Foods that are allowed to be eaten according to the rotation schedule for the food list.
3.2.9.13.6. Foods that will be eaten at the meal

3.2.9.14. Selecting Meals
3.2.9.14.1. User-selected meal plan
3.2.9.14.2. Id of selected meal plan
3.2.9.14.3. Desired meal
3.2.9.15. Deleting meal
3.2.9.15.1. User-selected meal plan
3.2.9.15.2. Id of selected meal plan
3.2.9.15.3. User-selected meal
3.2.9.15.4. Id of selected meal
3.2.9.16. Creating shopping list
3.2.9.16.1. User-selected food list
3.2.9.16.2. Id of selected food list
3.2.9.16.3. Start date
3.2.9.16.4. Number of days to shop for

3.2.10. Processes

3.2.10.1. Create Account (username/password/email combination)
3.2.10.1.1. Enter unique username
3.2.10.1.2. Enter password
3.2.10.1.3. Enter email address
3.2.10.1.4. Confirm email address (double entry)
3.2.10.1.5. Inserts new account into database.
3.2.10.1.6. Forward to home page

3.2.10.2. Login to Account
3.2.10.2.1. Enter username
3.2.10.2.2. Enter password
3.2.10.2.3. Checks database for username/password combination.
3.2.10.2.4. If correct forwards to home page
3.2.10.2.5. If incorrect, prompts for correct username/password combination.

3.2.10.3. Find Lost Account
3.2.10.3.1. Enter email address
3.2.10.3.2. Checks database to see if the email address can be found.
3.2.10.3.3. If the email address is found, forwards to lost account page.
3.2.10.3.4. If not email address not found, prompts for correct email address.

3.2.10.4. Create Food List
3.2.10.4.1. Enter rotation schedule
3.2.10.4.2. Enter food list name
3.2.10.4.3. Checks to see if food list name is already in use by this user.
3.2.10.4.3.1. If yes, prompts to Enter different food list name
3.2.10.4.3.2. If no, inserts the new food list into the database.
3.2.10.4.4. Forwards to Modify Food List

3.2.10.5. Modify Food List

3.2.10.5.1. Select Food List OR selected list is received from CREATE FOOD LIST.
3.2.10.5.2. Select foods from the list.
3.2.10.5.3. Inserts the selected foods into the database.
3.2.10.5.4. Forwards to food lists home page.

3.2.10.6. Select Food List

3.2.10.6.1. Select the desired food list.

3.2.10.7. Delete Food List

3.2.10.7.1. Select Food List
3.2.10.7.2. Confirm deletion.
3.2.10.7.3. Deletes from the database all foods in the meals in the meal plans associated with the selected food list.
3.2.10.7.4. Deletes from the database all the meals in the meal plans associated with the selected food list.
3.2.10.7.5. Deletes from the database all the meal plans associated with the selected food list.
3.2.10.7.6. Deletes from the database all the foods in the food list.
3.2.10.7.7. Deletes the selected food list from the database.
3.2.10.7.8. Forwards to the food list home page.

3.2.10.8. View Food List

3.2.10.8.1. Select a Food List
3.2.10.8.2. Gets the foods in the food list and their associated food families from the database for display to the user.

3.2.10.9. Create Meal Plan

3.2.10.9.1. Select Food List
3.2.10.9.2. Enter start date
3.2.10.9.3. Enter number of days to be planned for.
3.2.10.9.4. Checks to see if a meal plan already exists for the start date
3.2.10.9.4.1. If yes, prompts to enter a different start date.
3.2.10.9.4.2. If no:
3.2.10.9.4.2.1. If only one meal plan is being created, inserts the meal plan into the database.
3.2.10.9.4.2.2. If more than one meal plans are to be created, checks to see if a meal plan exists for any of these days.
3.2.10.9.4.2.2.1. If yes, displays error message to user.
3.2.10.9.4.2.2.2. If no, inserts meal plans into the database.
3.2.10.9.5. Forwards to Create Meals

3.2.10.10. Select Meal Plan

3.2.10.10.1. Select the desired meal plan.

3.2.10.11. View Meal Plan(s)

3.2.10.11.1. Enter start date
3.2.10.11.2. Enter number of days to shop for.

3.2.10.11.3. Displays the meal plan dates, meal names, and foods in the meals to the user.

3.2.10.12. Delete Meal Plan

3.2.10.12.1. Select Meal Plan

3.2.10.12.2. Confirm deletion.

3.2.10.12.3. Deletes from the database all the foods in the meals associated with the selected meal plan.

3.2.10.12.4. Deletes from the database all the meals associated with the selected meal plan.

3.2.10.12.5. Deletes the selected meal plan from the database.

3.2.10.12.6. Forwards to the meal plan home page.

3.2.10.13. Create Meals

3.2.10.13.1. Enter the names of the desired meal(s)

3.2.10.13.2. Inserts the new meals into the database.

3.2.10.13.3. Forwards to Modify Meals

3.2.10.14. Modify Meals

3.2.10.14.1. Select Meal OR receives a list of one or more meals from Create Meals

3.2.10.14.2. For each meal

3.2.10.14.2.1. Select the foods to be eaten at this meal.

3.2.10.14.2.2. Inserts the foods into the database.

3.2.10.14.3. Forwards to the meal plan home page.

3.2.10.15. Select Meal

3.2.10.15.1. Select the desired meal.

3.2.10.16. Delete Meal

3.2.10.16.1. Select Meal

3.2.10.16.2. Confirm deletion.

3.2.10.16.3. Deletes the foods in the meal from the database.

3.2.10.16.4. Deletes the meal from the database.

3.2.10.16.5. Forwards to the modify meal plan page.

3.2.10.17. View Shopping List

3.2.10.17.1. Select a food list

3.2.10.17.2. Enter start date

3.2.10.17.3. Enter number of days to shop for.

3.2.10.17.4. Displays the foods in the meals for these dates, as well as the number of meals each food is to be eaten on.

3.2.11. Outputs

3.2.11.1. Input screens for each process requiring user input

3.2.11.2. Screens for each process requiring display output.

3.2.11.3. Food Lists

3.2.11.4. Meal Plans

3.2.11.5. Meals

3.2.11.6. Shopping Lists

3.2.12. Stored Data

3.2.12.1. Accounts

3.2.12.1.1. username
3.2.12.1.2. password
3.2.12.1.3. email address
3.2.12.1.4. user id (auto number generated by database)
3.2.12.2. Foods
3.2.12.2.1. food id (auto number generated by database)
3.2.12.2.2. food name
3.2.12.2.3. family id
3.2.12.3. Food Families
3.2.12.3.1. family id (auto number generated by database)
3.2.12.3.2. family name
3.2.12.4. Food Lists
3.2.12.4.1. user id
3.2.12.4.2. list name
3.2.12.4.3. list id (auto number generated by database)
3.2.12.4.4. rotation schedule
3.2.12.5. Food List Items (foods associated with a food list)
3.2.12.5.1. food id
3.2.12.5.2. food list id
3.2.12.6. Meal Plans
3.2.12.6.1. meal plan id (auto number generated by database)
3.2.12.6.2. food list id
3.2.12.6.3. date
3.2.12.7. Meals
3.2.12.7.1. meal plan id
3.2.12.7.2. meal id (auto number generated by database)
3.2.12.7.3. meal name
3.2.12.8. Meal Items (foods associated with a meal)
3.2.12.8.1. meal id
3.2.12.8.2. food id

3.3. Nonfunctional Requirements

3.3.1. Throughput/Response Time

3.3.1.1. Processes should not take an unreasonable amount of time to complete.

3.3.2. Ease of Use

3.3.2.1. The application should be easy to use, requiring only familiarity with the chosen operating system, an internet browser, and perhaps a minimal amount of training, which should be easily obtained by reading the application manual.

3.3.3. Budget

3.3.3.1. $400.00

3.3.4. Costs

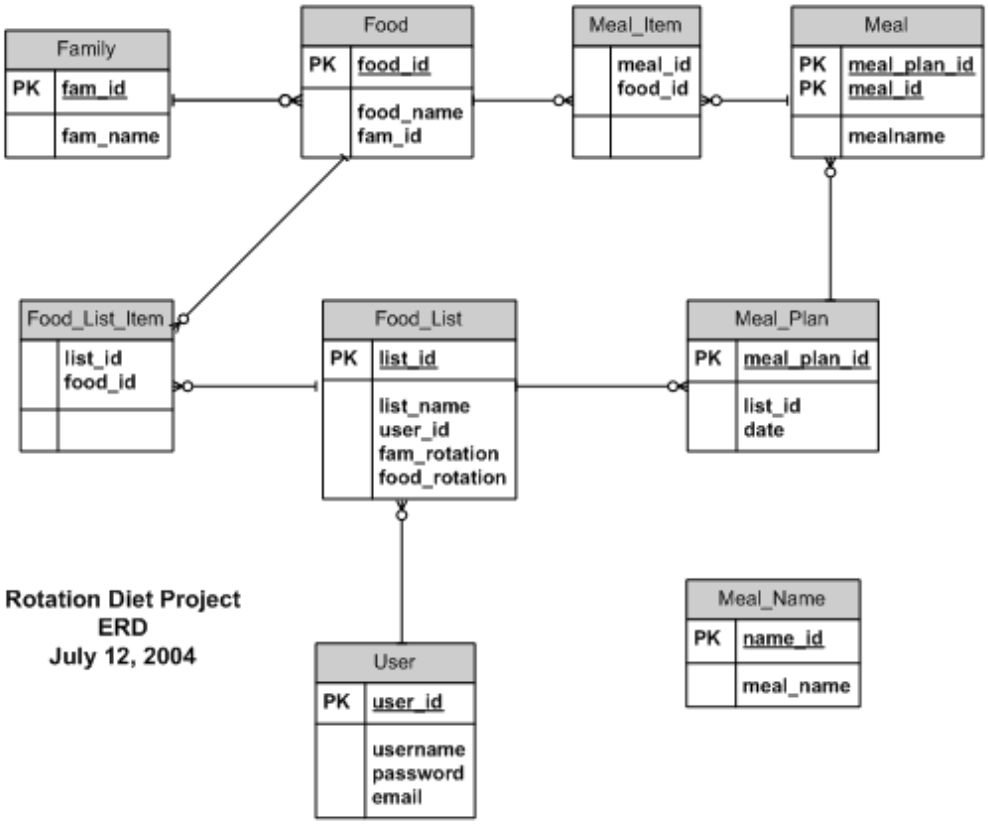
3.3.4.1. Application Development Solution

3.3.4.1.1. The software used to develop the application.

3.3.4.1.2. Budget: $100

3.3.4.2. Database Software
3.3.4.2.1. The database back-end for the application.
3.3.4.2.2. Budget: $100
3.3.4.3. Application Implementation Technology
3.3.4.3.1. The technology used to implement the application.
3.3.4.3.2. Budget: $100
3.3.4.4. Database Application Web Hosting
3.3.4.4.1. An internet web host for the database application.
3.3.4.4.2. Needed for approximately six months, until the project presentation is concluded.
3.3.4.4.3. Budget: $100
3.3.5. Cost Savings
3.3.5.1. Time Savings on tasks such as creating Meal Plans and shopping lists should be significant.
3.3.6. Timetables/Deadline
3.3.6.1. The project implementation must be completed by January 10, 2005.
3.3.7. Documentation and Training Needs
3.3.7.1. Documentation should be ongoing throughout the project.
3.3.7.2. Users should only need to be familiar with their operating system and browser to successfully use the application.
3.3.8. Quality Management
3.3.8.1. Quality will be assured and approved by the user/testers.
4. Conclusion
4.1. Outstanding Issues
4.1.1. I would like to add an action for deleting a user's account.
4.1.2. The deployed application works fine offline, but does not work online.

Appendix C: Database Entity Relationship Diagram (ERD)



Appendix D: Struts Application Specification

Struts Specifications for Rotation Diet Project
The purpose of this document is to give a general description of each building block of this Struts project (JSPs, ActionForms, Actions, etc), which will also include the general purpose of each building block.

1. JavaServer Pages
    1.1. Account Pages
        1.1.1. createAccount.jsp
            1.1.1.1. Form: CreateAccountForm
            1.1.1.2. Action: CreateAccountAction
            1.1.1.3. Allows user to enter a unique username as well as a password and valid email address, and confirm the email address.
            1.1.1.4. If the email addresses entered match, the program will check to see if the username is unique. Otherwise it will reset() the fields and prompt the user to re-enter the info.
            1.1.1.5. If the username is unique the account will be created, otherwise the user will be prompted to try a different username
            1.1.1.6. If the account is successfully created, the user will automatically be logged in.
            1.1.1.7. Links
                1.1.1.7.1. privacyPolicy.jsp
                1.1.1.7.2. createAccount.jsp
        1.1.2. login.jsp
            1.1.2.1. Form: LoginForm
            1.1.2.2. Action: LoginAction
            1.1.2.3. Allows users to Login
            1.1.2.4. Allows user to Create a New Account.
            1.1.2.5. If the user enters a correct username/password combination they will be logged in. Otherwise, the user will be prompted to try again.
            1.1.2.6. Links
                1.1.2.6.1. lostAccount.jsp.
                1.1.2.6.2. createAccount.jsp.
        1.1.3. lostAccount.jsp
            1.1.3.1. Form: LostAccountForm
            1.1.3.2. Action: LostAccountAction
            1.1.3.3. Allows user to enter and confirm the email address associated with their account to retrieve their account information.
            1.1.3.4. If the email addresses entered match the program checks the database for that address.

1.1.3.5. If the address is found, the account information should be emailed to that email address. Otherwise the fields are reset and the user is prompted to re-enter the correct email address.
1.1.4. lostAccountResults.jsp
1.1.4.1. Displays a message to the user regarding status of lost account request.
1.1.4.2. Links
1.1.4.2.1. login.jsp
1.2. Common Pages (Tiles pages)
1.2.1. footer.jsp
1.2.1.1. Displays the footer information.
1.2.2. header.jsp
1.2.2.1. Displays the header information.
1.2.2.2. Includes
1.2.2.2.1. Message display area.
1.2.2.2.2. Error display area.
1.2.3. layout.jsp
1.2.3.1. Tiles master layout page.
1.2.3.2. Determines the layout/look of the entire web site.
1.2.4. links.jsp
1.2.4.1. Contains links that should be common to all/most pages
1.2.4.2. Some links will only display if the user's id is to be present as a session variable.
1.2.4.3. Some links will only display if the user's id and a food list id are to be present as session variables.
1.2.4.4. Some links will only display if the user's id is not to be present as a session variable
1.2.4.5. Links
1.2.4.5.1. Common Links
1.2.4.5.1.1. contactUs.jsp
1.2.4.5.1.2. privacyPolicy.jsp
1.2.4.5.1.3. LogoutAction
1.2.4.5.2. Variable Links
1.2.4.5.2.1. Require user id session variable to be to be present
1.2.4.5.2.1.1. chooser.jsp
1.2.4.5.2.1.2. logout (Action)
1.2.4.5.2.2. Require user id session variable not to be to be present
1.2.4.5.2.2.1. logon.jsp
1.2.4.5.2.3. Require user id and food list id session variable to be to be present
1.2.4.5.2.3.1. foodList.jsp
1.2.4.5.2.3.2. mealPlans.jsp
1.2.4.5.2.3.3. shoppingLists.jsp
1.3. Food List Pages
1.3.1. createFoodList.jsp

1.3.1.1. Form: CreateFoodListForm
1.3.1.2. Action: CreateFoodListAction
1.3.1.3. Enter the food list name.
1.3.1.4. Enter the food family rotation in days.
1.3.1.5. Enter the food rotation in days.
1.3.2. deleteFoodList.jsp
1.3.2.1. Form: SelectorForm
1.3.2.2. Action: DeleteFoodListAction
1.3.2.3. Select Y/N to give/refuse permission to delete the food list.
1.3.3. foodLists.jsp
1.3.3.1. Form: BlankForm
1.3.3.2. Action FoodListAction
1.3.3.3. Choose a food list action:
1.3.3.3.1. Create Food List
1.3.3.3.2. View/Print Food List
1.3.3.3.3. Modify Food List
1.3.3.3.4. Delete Food List
1.3.4. modifyFoodList.jsp
1.3.4.1. Form: ModifyFoodListForm
1.3.4.2. Action: ModifyFoodListAction
1.3.4.3. Select the box beside the food name to add the food to the food list.
1.3.4.4. Deselect the box beside the food name to remove the food from the food list.
1.3.5. selectFoodList.jsp
1.3.5.1. Form: SelectorForm
1.3.5.2. Action: SelectFoodListAction
1.3.5.3. Select a Food List.
1.3.6. viewFoodList.jsp
1.3.6.1. Form: BlankForm
1.3.6.2. Action: ViewFoodListAction
1.3.6.3. Displays the foods in the selected food list.
1.3.6.4. Forwards to
1.3.6.4.1. Back to Food Lists (foodLists.jsp)
1.3.6.4.2. Home (chooser.jsp)
1.4. General Pages
1.4.1. chooser.jsp
1.4.1.1. Form: BlankForm
1.4.1.2. Action: ChooserAction
1.4.1.3. Select an option:
1.4.1.3.1. Food Lists
1.4.1.3.2. Meal Plans
1.4.1.3.3. Shopping Lists
1.4.2. contactUs.jsp
1.4.2.1. Displays the site’s contact information.

1.4.2.2. Links
1.4.2.2.1. Requires user id to be present as session variable
1.4.2.2.1.1. chooser.jsp
1.4.2.2.2. Requires user id is not to be present as session variable
1.4.2.2.2.1. login.jsp
1.4.3. privacyPolicy.jsp
1.4.3.1. Displays the site's privacy policy.
1.4.3.2. Links
1.4.3.2.1. Requires user id to be present as session variable.
1.4.3.2.1.1. chooser.jsp
1.4.3.2.2. Requires user id not to be present as session variable.
1.4.3.2.2.1. login.jsp

1.5. Meal Plan Pages
1.5.1. createMealPlan.jsp
1.5.1.1. Form: CreateMealPlanForm
1.5.1.2. Action: CreateMealPlanAction
1.5.1.3. Enter the start date.
1.5.1.4. Enter the number of days planning meals for.
1.5.2. deleteMealPlan.jsp
1.5.2.1. Form: SelectorForm
1.5.2.2. Action: DeleteMealPlanAction
1.5.2.3. Select Y/N to give/refuse permission to delete the selected meal plan.
1.5.3. mealPlans.jsp
1.5.3.1. Form: BlankForm
1.5.3.2. Action: MealPlanAction
1.5.3.3. Select a meal plan option:
1.5.3.3.1. Create Meal Plan(s)
1.5.3.3.2. Modify Meal Plan
1.5.3.3.3. View/Print Meal Plan
1.5.3.3.4. Delete Meal Plan
1.5.3.3.5. Change to a different food list.
1.5.4. modifyMealPlan.jsp
1.5.4.1. Form: BlankForm
1.5.4.2. Action: ModifyMealPlanAction
1.5.4.3. Choose an option:
1.5.4.3.1. Add a Meal
1.5.4.3.2. Modify a Meal
1.5.4.3.3. Delete a Meal
1.5.4.3.4. Select a Different Meal Plan
1.5.4.3.5. Back to Home (chooser.jsp)
1.5.5. selectMealPlan.jsp
1.5.5.1. Form: SelectorForm
1.5.5.2. Action: SelectMealPlanAction

1.5.5.3. Select a Meal Plan.
1.5.6. selectMultipleMealPlans.jsp
1.5.6.1. Form: SelectMultipleMealPlansForm
1.5.6.2. Action: SelectMultipleMealPlansAction
1.5.6.3. Enter start date.
1.5.6.4. Enter number of days.
1.6. Meal Pages
1.6.1. createMeals.jsp
1.6.1.1. Form: CreateMealsForm
1.6.1.2. Action: CreateMealsAction
1.6.1.3. Select the boxes beside the meals you want to plan.
1.6.2. deleteMeal.jsp
1.6.2.1. Form: SelectorForm
1.6.2.2. Action: DeleteMealAction
1.6.2.3. Select Y/N to give/refuse permission to delete the select meal.
1.6.3. modifyMeals.jsp
1.6.3.1. Form: ModifyMealsForm
1.6.3.2. Action ModifyMealsAction
1.6.3.3. Select the box beside the food name to add the food to the meal.
1.6.3.4. Deselect the box beside the food name to remove the food from the meal.
1.6.4. selectMeal.jsp
1.6.4.1. Form: SelectorForm
1.6.4.2. Action: SelectMealAction
1.6.4.3. Select a Meal.
1.7. Shopping List Pages
1.7.1. shoppingLists.jsp
1.7.1.1. Choose an option:
1.7.1.1.1. Create and Print a Shopping List
1.7.1.1.2. Select a different food list
1.7.1.1.3. Back to Home
1.7.2. viewShoppingList.jsp
1.7.2.1. Form: BlankForm
1.7.2.2. Action: ViewShoppingListAction
1.7.2.3. Displays the selected meal plans in shopping list format.
1.7.2.4. Forwards:
1.7.2.4.1. Back to Shopping Lists (shoppingLists.jsp)
1.7.2.4.2. Home (chooser.jsp)

2. Forms
Note: Forms with Validators are ValidatorForms, otherwise they are ActionForms.

2.1. Account Forms
2.1.1. CreateAccountForm

2.1.1.1. Action: CreateAccountAction
2.1.1.2. Page: createAccount.jsp
2.1.1.3. Input
2.1.1.3.1. username – the user's desired username
2.1.1.3.2. password – the user's desired password
2.1.1.3.3. email address – the user's email address
2.1.1.3.4. confirm email address – user re-enters email address
2.1.1.4. Validators
2.1.1.4.1. required
2.1.1.4.1.1. username
2.1.1.4.1.2. password
2.1.1.4.2. minlength
2.1.1.4.2.1. username: 5
2.1.1.4.2.2. password: 5
2.1.1.4.3. maxlength
2.1.1.4.3.1. email: 40
2.1.1.4.4. email
2.1.1.4.4.1. email
2.1.1.4.5. custom
2.1.1.4.5.1. email and confirm email must be identical
2.1.2. LoginForm
2.1.2.1. Action: LoginAction
2.1.2.2. Page: login.jsp
2.1.2.3. Input
2.1.2.3.1. username – the user's username
2.1.2.3.2. password – the user's password
2.1.2.4. Validators
2.1.2.4.1. required
2.1.2.4.1.1. username
2.1.2.4.1.2. password
2.1.3. LostAccountForm
2.1.3.1. Action: LostAccountAction
2.1.3.2. Page: lostAccount.jsp
2.1.3.3. Input
2.1.3.3.1. email – the email address the user entered when they created their account
2.1.3.3.2. confirm email – user re-enters email address
2.1.3.4. Validators
2.1.3.4.1. required
2.1.3.4.1.1. email
2.1.3.4.1.2. confirm email
2.1.3.4.2. email
2.1.3.4.2.1. email
2.1.3.4.2.2. confirm email
2.1.3.4.3. custom

2.1.3.4.3.1. email and confirm email must be identical

2.2. Food List Forms
  2.2.1. CreateFoodListForm
    2.2.1.1. Action: CreateFoodListAction
    2.2.1.2. Page: createFoodList.jsp
    2.2.1.3. Input
      2.2.1.3.1. famRotation – the food family rotation
      2.2.1.3.2. foodRotation – the food rotation
      2.2.1.3.3. listName – the user's desired food list name
    2.2.1.4. Validators
      2.2.1.4.1. custom
        2.2.1.4.1.1. famRotation must be equal to or less than foodRotation.
        2.2.1.4.1.2. listName must not already be in use by this user.
      2.2.1.4.2. required
        2.2.1.4.2.1. famRotation
        2.2.1.4.2.2. foodRotation
        2.2.1.4.2.3. listName
      2.2.1.4.3. integer
        2.2.1.4.3.1. famRotation
        2.2.1.4.3.2. foodRotation
      2.2.1.4.4. maxlength
        2.2.1.4.4.1. famRotation: 2
        2.2.1.4.4.2. foodRotation: 2
        2.2.1.4.4.3. listName: 12
  2.2.2. ModifyFoodListForm
    2.2.2.1. Action: ModifyFoodListDispatchAction
    2.2.2.2. Page: modifyFoodList.jsp
    2.2.2.3. Input
      2.2.2.3.1. preSelected – array of the foods already in the user's food list
      2.2.2.3.2. selected – array of the foods the user wants to be in the food list
    2.2.2.4. Validators
2.3. General Forms
  2.3.1. BlankForm
    2.3.1.1. This placeholder form is used by Actions that do not require form data because the Struts framework requires an ActionForm to be indicated for most Actions.
      2.3.1.1.1. ChooserAction
      2.3.1.1.2. FoodListAction
      2.3.1.1.3. InsertModifiedFoodListAction
      2.3.1.1.4. ModifyFoodListPrepareAction
      2.3.1.1.5. SelectFoodListPrepareAction
      2.3.1.1.6. ViewFoodListPrepareAction

2.3.1.1.7. ViewFoodListAction
2.3.1.1.8. MealPlanAction
2.3.1.1.9. CreateMealPlanPrepareAction
2.3.1.1.10. ModifyMealPlanAction
2.3.1.1.11. SelectMealPlanPrepareAction
2.3.1.1.12. ViewMealPlanPrepareAction
2.3.1.1.13. ViewMealPlanAction
2.3.1.1.14. SelectMealPrepareAction
2.3.1.1.15. CreateMealsPrepareAction
2.3.1.1.16. InsertModMealAction
2.3.1.1.17. ModifyMultipleMealsPrepareAction
2.3.1.1.18. ModifySingleMealPrepareAction
2.3.1.1.19. AddMealsPrepareAction

2.3.2. SelectorForm

2.3.2.1. This single field form is used by several actions.

2.3.2.1.1. SelectFoodListAction
2.3.2.1.2. DeleteFoodListAction
2.3.2.1.3. SelectMealPlanAction
2.3.2.1.4. DeleteMealPlanAction
2.3.2.1.5. SelectMealAction
2.3.2.1.6. DeleteMealAction

2.3.2.2. Input

2.3.2.2.1. selected – the selected data.

2.3.2.3. Validators

2.3.2.3.1. required

2.3.2.3.1.1. selected

2.4. Meal Forms

2.4.1. CreateMealsForm

2.4.1.1. Action: CreateMealsAction
2.4.1.2. Page: createMeals.jsp
2.4.1.3. Input

2.4.1.3.1. mealNames – array of the meal names the user may select.
2.4.1.3.2. selected – array of the meal names the user selected.

2.4.2. ModifyMealsForm

2.4.2.1. Action: ModifyMealsDispatchAction
2.4.2.2. Page: modifyMeals.jsp
2.4.2.3. Input

2.4.2.3.1. preSelected – array of the foods already in the user's food list
2.4.2.3.2. selected – array of the foods the user wants to be in the food list

2.5. Meal Plan Forms

2.5.1. CreateMealPlanForm

2.5.1.1. Action: CreateMealPlanAction

2.5.1.2. Page: createMealPlan.jsp
2.5.1.3. Input
2.5.1.3.1. startDate – the first date to create meals for.
2.5.1.3.2. days – the number of days the user wants to plan meals for.
2.5.1.3.3. dates – array of the dates the meals will be created for.
2.5.1.4. Validators
2.5.1.4.1. custom
2.5.1.4.1.1. Returns an error if a meal plan for startDate is already in the database for this food list.
2.5.1.4.1.2. Returns an error if a meal plan for any of the other dates requested is already in the database.
2.5.1.4.2. required
2.5.1.4.2.1. startDate
2.5.1.4.2.2. days
2.5.1.4.3. integer
2.5.1.4.3.1. days
2.5.1.4.4. date
2.5.1.4.4.1. startDate: strict date pattern MM-dd-yyyy
2.5.1.4.5. range
2.5.1.4.5.1. days: min 1, max 7
2.5.2. SelectMultipleMealPlansForm
2.5.2.1. Action: SelectMultipleMealPlansAction
2.5.2.2. Page: selectMultipleMealPlans
2.5.2.3. Input
2.5.2.3.1. startId – the meal plan id of the selected starting date
2.5.2.3.2. days – the number of days the user wants to view meal plans for.
2.5.2.3.3. mealPlans – array of MealPlanDTOs containg data about the meal plans that are to be viewed.
2.5.2.4. Validators
2.5.2.4.1. custom
2.5.2.4.1.1. There must be at least one meal plan in the given date range.
2.5.2.4.2. required
2.5.2.4.2.1. startId
2.5.2.4.2.2. days
2.5.2.4.3. integer
2.5.2.4.3.1. days
2.5.2.4.4. range
2.5.2.4.4.1. days: min 1, max 7
2.6. Shopping List Forms
2.6.1. None needed.

Note: Some Actions exist only logically in the struts-config file, not as physical java class files. These actions will be denoted with a *.

3. Actions
    3.1. Account Actions
        3.1.1. CreateAccountAction
            3.1.1.1. Form: CreateAccoutForm
            3.1.1.2. Page: createAccount.jsp
            3.1.1.3. Adds a new user account to the database.
            3.1.1.4. ActionForward: chooser
        3.1.2. LoginAction
            3.1.2.1. Form: LoginForm
            3.1.2.2. Page: login.jsp
            3.1.2.3. ActionForward: chooser
        3.1.3. LogoutAction
            3.1.3.1. Form: None
            3.1.3.2. Page: None
            3.1.3.3. Invalidates the session, which removes from memory all variables stored in the session.
            3.1.3.4. ActionForward: login
        3.1.4. LostAccountAction
            3.1.4.1. Form: LostAccountForm
            3.1.4.2. Page: lostAccount.jsp
            3.1.4.3. Gets the user's email address for display.
            3.1.4.4. Future versions will email the user's account information to the user's email address.
            3.1.4.5. ActionForward: lostAccountResults
    3.2. FoodList Actions
        3.2.1. CreateFoodListAction
            3.2.1.1. Form: CreateFoodListForm
            3.2.1.2. Page: createFoodList.jsp
            3.2.1.3. Adds a new food list to the database.
            3.2.1.4. ActionForward: modifyFoodListPrepare
        3.2.2. DeleteFoodListAction
            3.2.2.1. Form: SelectorForm
            3.2.2.2. Page: deleteFoodList.jsp
            3.2.2.3. Deletes a food list from the database.
            3.2.2.4. ActionForward: foodLists
        3.2.3. FoodListAction*
            3.2.3.1. Form: BlankForm
            3.2.3.2. Page: foodLists.jsp
            3.2.3.3. Allows user to click a button to forward to a food list option.
        3.2.4. InsertModifiedFoodListAction
            3.2.4.1. Form: BlankForm
            3.2.4.2. Adds selected foods to the food list.
            3.2.4.3. Removes any unwanted existing foods from the food list.

3.2.4.4. ActionForward: foodLists
3.2.5. ModifyFoodListDispatchAction
3.2.5.1. Form: ModifyFoodListForm
3.2.5.2. Page: modifyFoodList.jsp
3.2.5.3. Gets user selections and sets them for use by insertModifiedFoodListAction.
3.2.5.4. ActionForward: insertModifiedFoodList
3.2.6. ModifyFoodListPrepareAction
3.2.6.1. Form: BlankForm
3.2.6.2. Sets up the data required by ModifyFoodListForm for use in ModifyFoodListAction.
3.2.6.3. ActionForward: modifyFoodList
3.2.7. SelectFoodListAction
3.2.7.1. Form: SelectorForm
3.2.7.2. Page: selectFoodList.jsp
3.2.7.3. Adds the id of the user selected food list to the session.
3.2.7.4. ActionForward: determined by source of the request.
3.2.8. SelectFoodListPrepareAction
3.2.8.1. Form: BlankForm
3.2.8.2. Sets up the data required by SelectorForm for use in SelectFoodListAction.
3.2.8.3. ActionForward: selectFoodList
3.2.9. ViewFoodListPrepareAction
3.2.9.1. Form: BlankForm
3.2.9.2. Sets up the data required by ViewFoodListAction.
3.2.9.3. ActionForward: viewFoodList
3.2.10. ViewFoodListAction*
3.2.10.1.Form: BlankForm
3.2.10.2.Page: viewFoodList.jsp
3.2.10.3.Allows user to click a button to forward to an option.
3.3. General Actions
3.3.1. ChooserAction*
3.3.1.1. Form: BlankForm
3.3.1.2. Page: chooser.jsp
3.3.1.3. Allows a user to click an option button to forward to that option.
3.4. Meal Actions
3.4.1. AddMealsPrepareAction
3.4.1.1. Form: BlankForm
3.4.1.2. Sets up the data required by CreateMealsForm for use by CreateMealsAction.
3.4.1.3. ActionForward: createMeals
3.4.2. CreateMealsAction
3.4.2.1. Form: CreateMealsForm
3.4.2.2. Page: createMeals.jsp
3.4.2.3. Adds one or more meals to the database.

3.4.2.4. ActionForward: modifyMultipleMealsPrepare
3.4.3. CreateMealsPrepareAction
3.4.3.1. Form: BlankForm
3.4.3.2. Sets up the data required by CreateMealsForm for use by CreateMealsAction.
3.4.3.3. ActionForward: createMeals
3.4.4. DeleteMealAction
3.4.4.1. Form: SelectorForm
3.4.4.2. Page: deleteMeal.jsp
3.4.4.3. Deletes the selected meal from the database.
3.4.4.4. ActionForward: modifyMealPlan
3.4.5. InsertModMealAction
3.4.5.1. Form: BlankForm
3.4.5.2. Adds foods to the selected meal.
3.4.5.3. Removes any unwanted existing foods from the selected meal.
3.4.5.4. ActionForward: Determined by the source of the modifyMeals request.
3.4.6. ModifyMealsDispatchAction
3.4.6.1. Form: ModifyMealsForm
3.4.6.2. Page: modifyMeals.jsp
3.4.6.3. Gets user selections and sets them for use by InsertModMealAction.
3.4.6.4. ActionForward: insertModMealAction
3.4.7. ModifyMultipleMealsPrepareAction
3.4.7.1. Form: BlankForm
3.4.7.2. Sets up data required by ModifyMealsForm for use by ModifyMealsAction.
3.4.7.3. ActionForward: modifyMeals
3.4.8. ModifySingleMealPrepareAction
3.4.8.1. Form: BlankForm
3.4.8.2. Sets up data required by ModifyMealsForm for use by ModifyMealsAction.
3.4.8.3. ActionForward: modifyMeals
3.4.9. SelectMealAction
3.4.9.1. Form: SelectorForm
3.4.9.2. Page: selectMeal.jsp
3.4.9.3. Gets user input to select a meal.
3.4.9.4. Adds data about the selected meal to the session.
3.4.9.5. ActionForward: Determined by the source of the selectMeal request.
3.4.10. SelectMealPrepareAction
3.4.10.1. Form: BlankForm
3.4.10.2. Sets up the data required by SelectorForm for use by SelectMealAction.
3.4.10.3. ActionForward: selectMeal

3.5. Meal Plan Actions
  3.5.1. CreateMealPlanAction
    3.5.1.1. Form: CreateMealPlanForm
    3.5.1.2. Page: createMealPlan.jsp
    3.5.1.3. Adds one or more meal plans to the database.
    3.5.1.4. ActionForward: createMealsPrepare
  3.5.2. CreateMealPlanPrepareAction
    3.5.2.1. Form: BlankForm
    3.5.2.2. Sets up the data required by CreateMealPlanForm for use by CreateMealPlanAction.
    3.5.2.3. ActionForward: createMealPlan
  3.5.3. DeleteMealPlanAction
    3.5.3.1. Form: SelectorForm
    3.5.3.2. Page: deleteMealPlan.jsp
    3.5.3.3. Deletes the selected meal plan from the database.
    3.5.3.4. ActionForward: mealPlans
  3.5.4. MealPlanAction*
    3.5.4.1. Form: BlankForm
    3.5.4.2. Page: mealPlans.jsp
    3.5.4.3. Allows user to click a button to forward to a meal plan option.
  3.5.5. SelectMealPlanAction
    3.5.5.1. Form: SelectorForm
    3.5.5.2. Page: selectMealPlan.jsp
    3.5.5.3. Gets user input to select a meal plan.
    3.5.5.4. Adds data about the selected meal plan to the session.
    3.5.5.5. ActionForward: Determined by the source of the selectMealPlan request.
  3.5.6. SelectMealPlanPrepareAction
    3.5.6.1. Form: BlankForm
    3.5.6.2. Sets up the data required by SelectorForm for use by SelectMealPlanAction.
    3.5.6.3. ActionForward: selectMealPlan
  3.5.7. SelectMultipleMealPlansAction
    3.5.7.1. Form: SelectMultipleMealPlansForm
    3.5.7.2. Page: selectMultipleMealPlans.jsp
    3.5.7.3. Gets user input to select a date range.
    3.5.7.4. Gets data about meal plans for future use.
    3.5.7.5. ActionForward: Determined by the source of the selectMultipleMealPlans request.
  3.5.8. SelectMultipleMealPlansPrepareAction
    3.5.8.1. Form: BlankForm
    3.5.8.2. Sets up the data required by SelectMultipleMealPlansForm for use by SelectMultipleMealPlansAction.
    3.5.8.3. ActionForward: SelectMultipleMealPlansAction
  3.5.9. ViewMealPlanAction*

3.5.9.1. Form: BlankForm
3.5.9.2. Page: viewMealPlan.jsp
3.5.9.3. Allows user to click a button to forward to an option.
3.5.10. ViewMealPlanPrepareAction
3.5.10.1.Form: BlankForm
3.5.10.2.Sets up the data required by ViewMealPlanAction.
3.5.10.3.ActionForward: viewMealPlan
3.6. Shopping List Actions
3.6.1. ShoppingListAction*
3.6.1.1. Form: BlankForm
3.6.1.2. Page: shoppingLists.jsp
3.6.1.3. Allows user to click a button to forward to a shopping list option.
3.6.2. ViewShoppingListAction*
3.6.2.1. Form: BlankForm
3.6.2.2. Page: viewShoppingList.jsp
3.6.2.3. Allows user to click a button to forward to an option.
3.6.3. ViewShoppingListPrepareAction
3.6.3.1. Form: BlankForm
3.6.3.2. Sets up the data required for display by viewShoppingList.jsp.
3.6.3.3. ActionForward: viewShoppingList

4. Utility Classes
4.1. Tokens
4.1.1. Holds all internal constants such as forward names, and query/update names
4.2. SqlMapConfig
4.2.1. Sets up the datasource and iBATIS framework
4.3. SqlHelper
4.3.1. Methods for creating, retrieving, updating, and deleting database information. (CRUD methods)
4.4. SelectorForm
4.4.1. A general purpose ActionForm for use when there is only one value to be passed to an action.

Other Specifications for Rotation Diet Project

1. iBATIS Framework.
1.1. Uses the SqlMapConfig and SqlHelper utility classes to expedite database traffic.
1.2. Uses java data transfer objects (DTOs) to securely transfer data.
1.3. Allows the constant parts of queries and updates to be stored in the web application, so that everything is in one place. This is much more efficient and reduces the server overhead.
2. Java Package descriptions
2.1. com.hward.rotationdiet.account

2.1.1. Provides classes for creating, retrieving, updating, and deleting user account data. (database crud actions)

2.2. com.hward.rotationdiet.dto
   2.2.1. Provides the Data Transfer Objects (DTOs) used by the iBATIS framework.

2.3. com.hward.rotationdiet.foodlist
   2.3.1. Provides classes for creating, retrieving, updating, and deleting user food list data. (database crud actions)

2.4. com.hward.rotationdiet.meal
   2.4.1. Provides classes for creating, retrieving, updating, and deleting user meal data. (database crud actions)

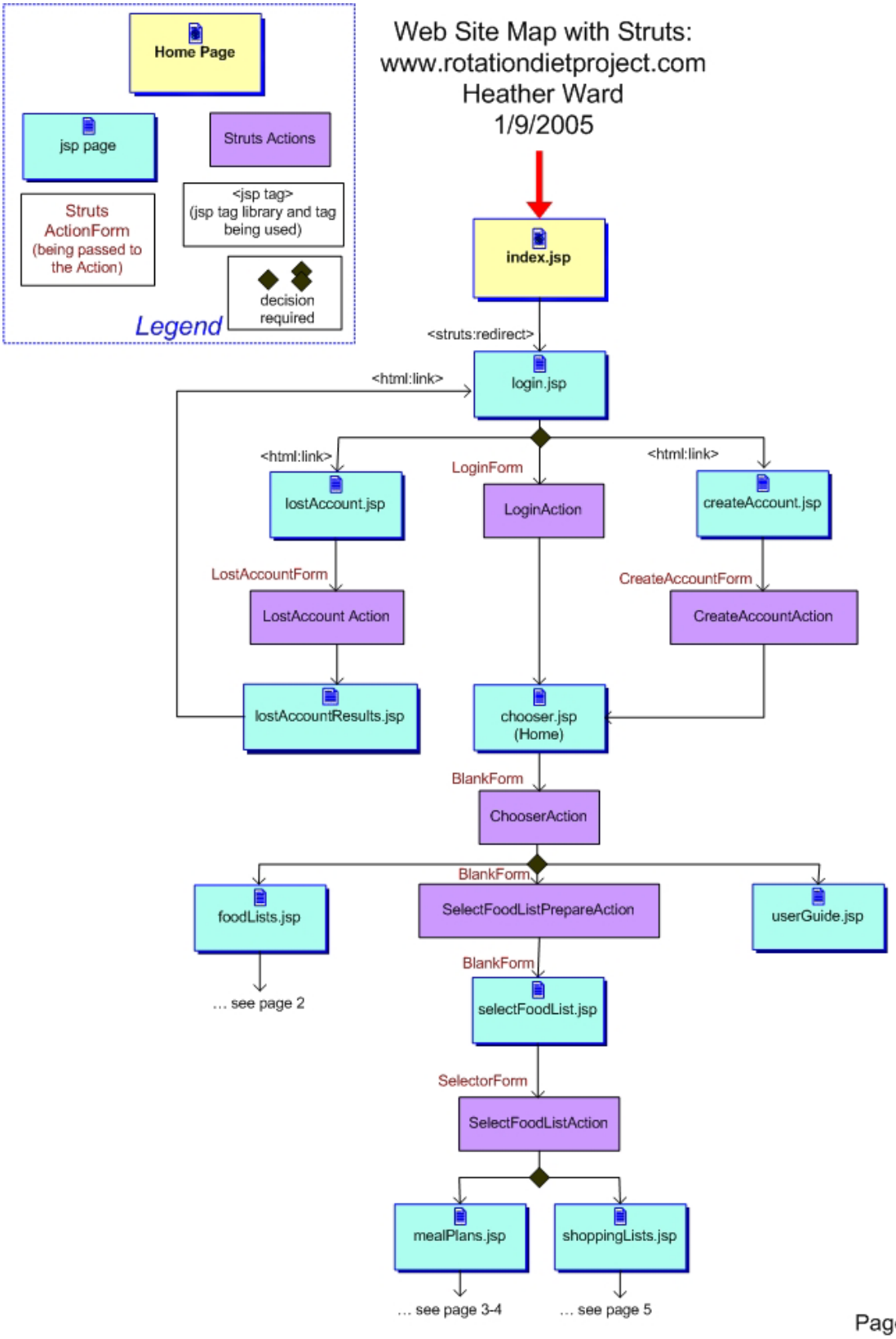
2.5. com.hward.rotationdiet.mealplan
   2.5.1. Provides classes for creating, retrieving, updating, and deleting user meal plan data. (database crud actions)

2.6. com.hward.rotationdiet.shoppinglist
   2.6.1. Provides classes for retrieving a user's shopping list data. (database crud actions)

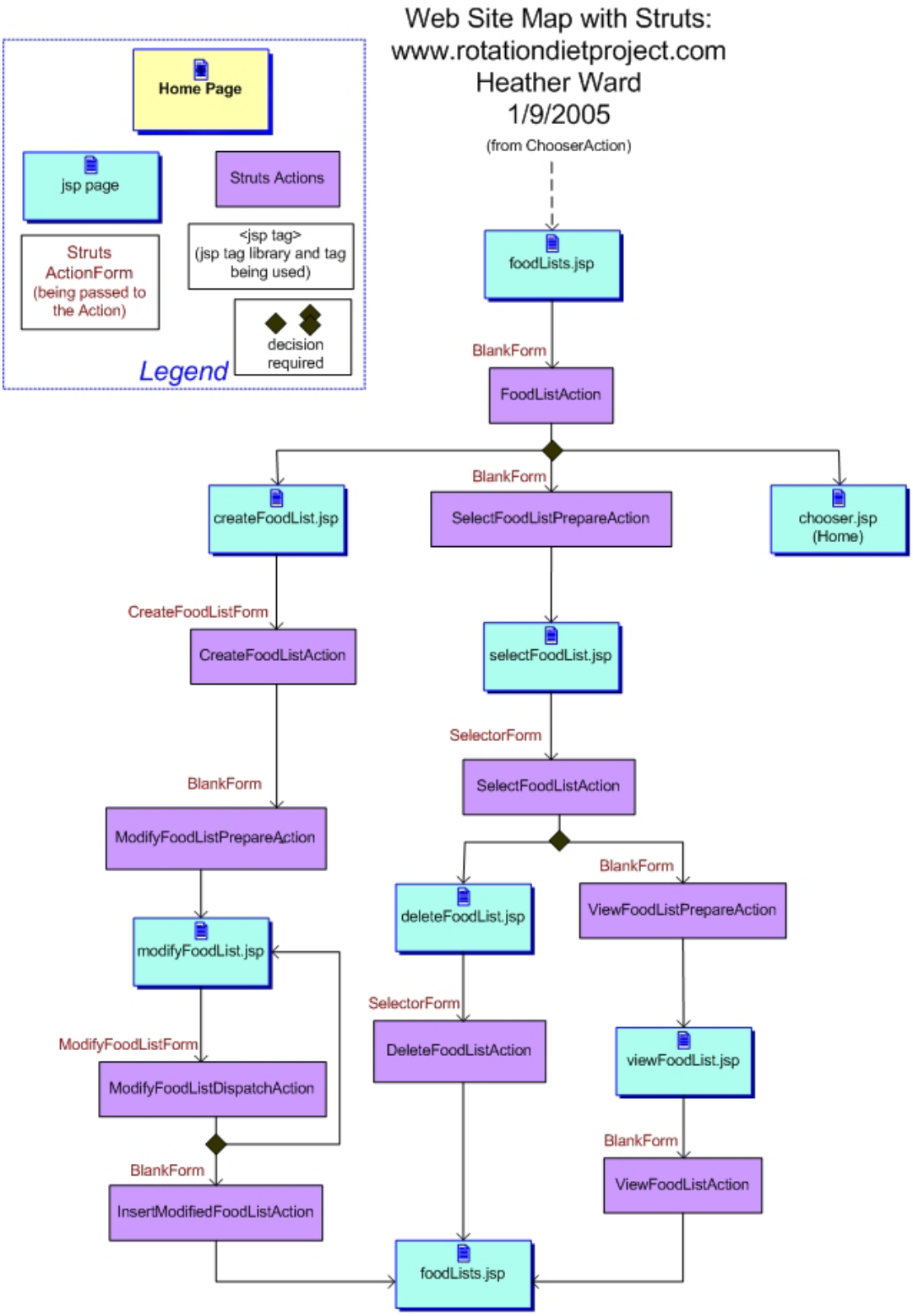
2.7. com.hward.rotationdiet.util
   2.7.1. Provides utility classes for the rotation diet application.

2.8. com.hward.util.ibatis
   2.8.1. Provides utility classes for implementing the iBATIS framework.

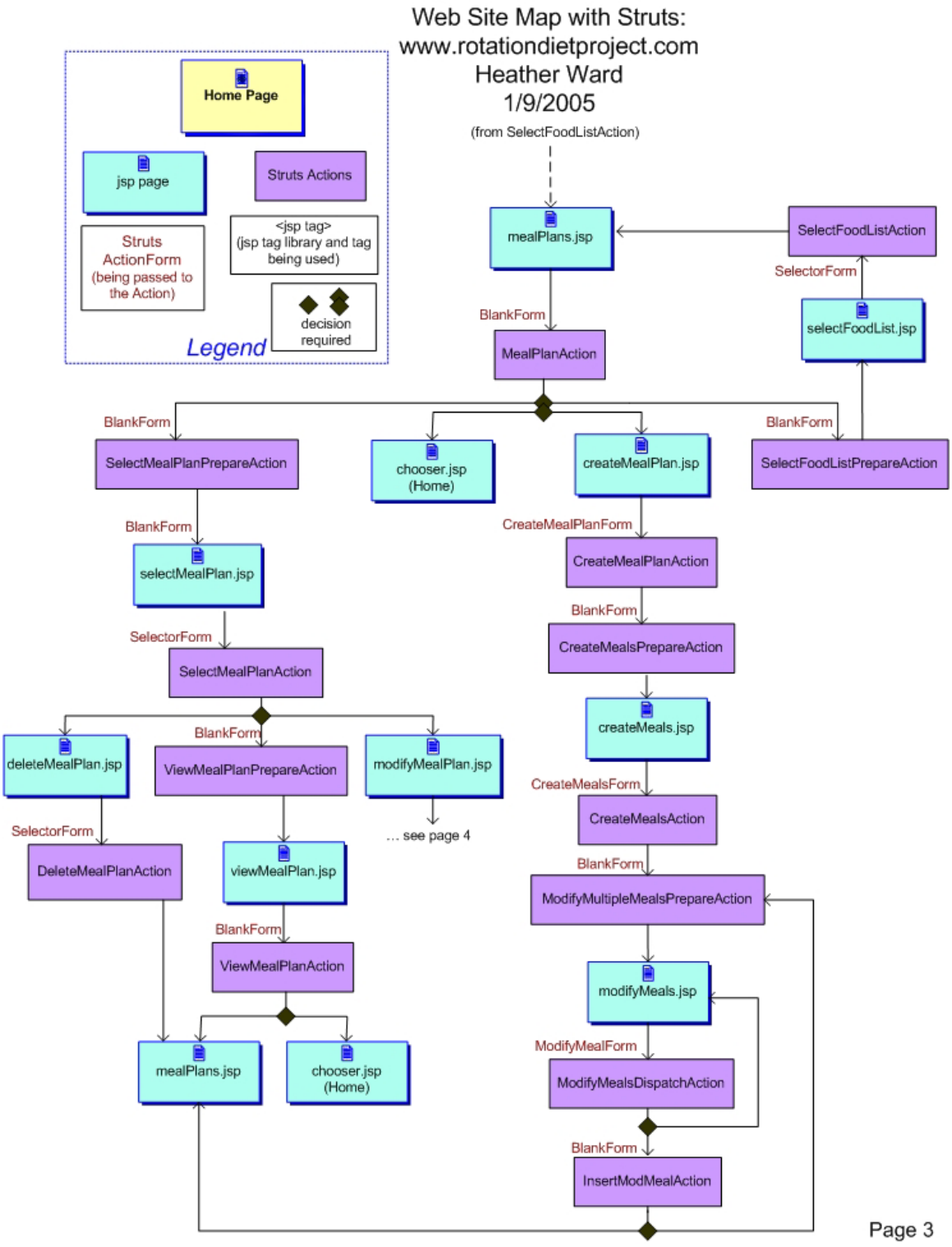
Appendix E: Struts Application Flowchart

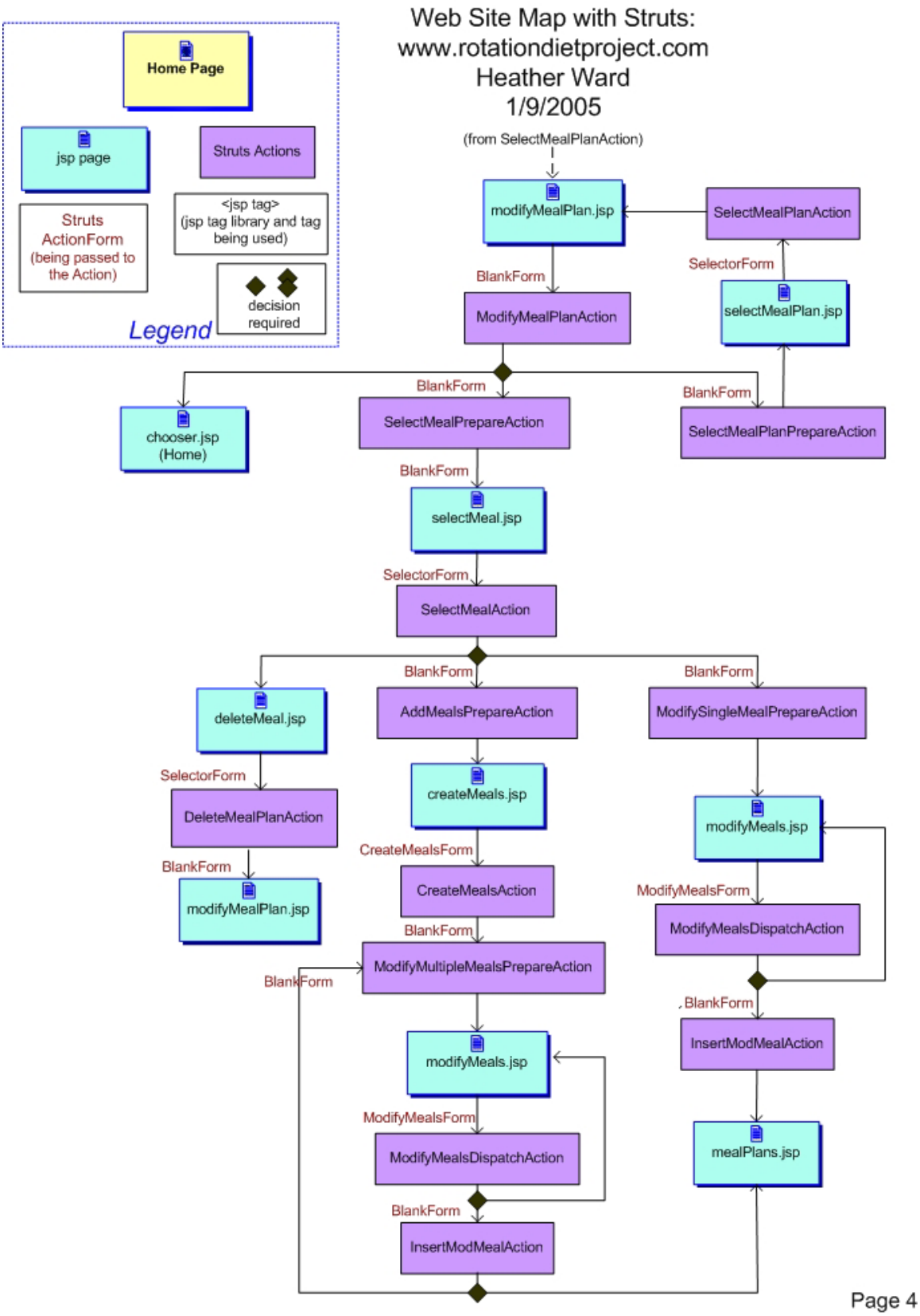


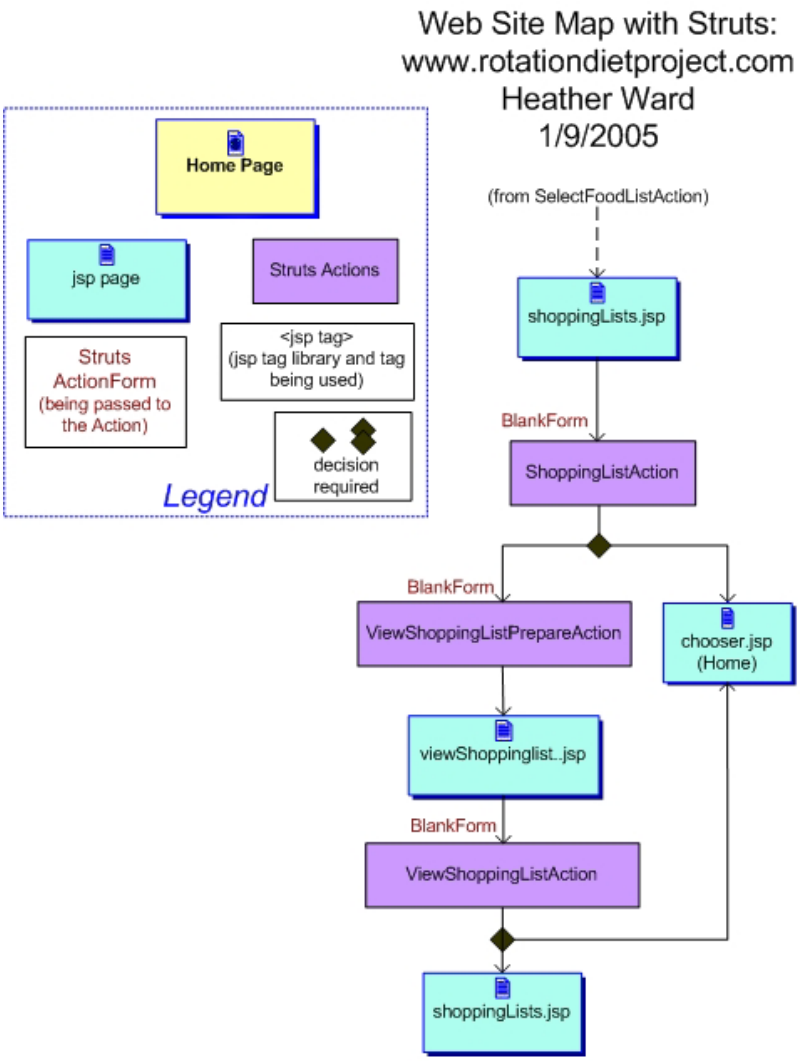
Page 1


Web Site Map with Struts:
www.rotationdietproject.com
Heather Ward
1/9/2005
Home Page
jsp page
Struts Actions
Struts ActionForm (being passed to the Action)
<jsp tag> (jsp tag library and tag being used)
decision required
Legend
(from ChooserAction)
foodLists.jsp
BlankForm
FoodListAction
BlankForm
createFoodList.jsp
SelectFoodListPrepareAction
chooser.jsp (Home)
CreateFoodListForm
CreateFoodListAction
selectFoodList.jsp
SelectorForm
SelectFoodListAction
BlankForm
ModifyFoodListPrepareAction
BlankForm
deleteFoodList.jsp
ViewFoodListPrepareAction
modifyFoodList.jsp
SelectorForm
DeleteFoodListAction
viewFoodList.jsp
ModifyFoodListForm
ModifyFoodListDispatchAction
BlankForm
ViewFoodListAction
BlankForm
InsertModifiedFoodListAction
foodLists.jsp

Page 2


Web Site Map with Struts:
www.rotationdietproject.com
Heather Ward
1/9/2005
Home Page
jsp page
Struts Actions
Struts ActionForm (being passed to the Action)
<jsp tag> (jsp tag library and tag being used)
decision required
Legend
(from SelectFoodListAction)
mealPlans.jsp
SelectFoodListAction
SelectorForm
selectFoodList.jsp
BlankForm
MealPlanAction
BlankForm
SelectMealPlanPrepareAction
chooser.jsp (Home)
createMealPlan.jsp
BlankForm
SelectFoodListPrepareAction
BlankForm
selectMealPlan.jsp
CreateMealPlanForm
CreateMealPlanAction
SelectorForm
SelectMealPlanAction
BlankForm
CreateMealsPrepareAction
createMeals.jsp
BlankForm
deleteMealPlan.jsp
ViewMealPlanPrepareAction
modifyMealPlan.jsp
CreateMealsForm
CreateMealsAction
... see page 4
SelectorForm
DeleteMealPlanAction
viewMealPlan.jsp
BlankForm
ModifyMultipleMealsPrepareAction
BlankForm
ViewMealPlanAction
modifyMeals.jsp
ModifyMealForm
mealPlans.jsp
chooser.jsp (Home)
ModifyMealsDispatchAction
BlankForm
InsertModMealAction
Page 3


Web Site Map with Struts:
www.rotationdietproject.com
Heather Ward
1/9/2005
Home Page
jsp page
Struts Actions
Struts ActionForm (being passed to the Action)
<jsp tag> (jsp tag library and tag being used)
decision required
Legend
(from SelectMealPlanAction)
modifyMealPlan.jsp
SelectMealPlanAction
BlankForm
SelectorForm
ModifyMealPlanAction
selectMealPlan.jsp
BlankForm
BlankForm
chooser.jsp (Home)
SelectMealPrepareAction
SelectMealPlanPrepareAction
BlankForm
selectMeal.jsp
SelectorForm
SelectMealAction
BlankForm
BlankForm
deleteMeal.jsp
AddMealsPrepareAction
ModifySingleMealPrepareAction
SelectorForm
createMeals.jsp
modifyMeals.jsp
DeleteMealPlanAction
CreateMealsForm
ModifyMealsForm
BlankForm
CreateMealsAction
ModifyMealsDispatchAction
modifyMealPlan.jsp
BlankForm
BlankForm
ModifyMultipleMealsPrepareAction
BlankForm
InsertModMealAction
modifyMeals.jsp
ModifyMealsForm
mealPlans.jsp
ModifyMealsDispatchAction
BlankForm
InsertModMealAction
Page 4


Web Site Map with Struts:
www.rotationdietproject.com
Heather Ward
1/9/2005
Home Page
jsp page
Struts Actions
Struts ActionForm (being passed to the Action)
<jsp tag> (jsp tag library and tag being used)
decision required
Legend
(from SelectFoodListAction)
shoppingLists.jsp
BlankForm
ShoppingListAction
BlankForm
ViewShoppingListPrepareAction
chooser.jsp (Home)
viewShoppinglist..jsp
BlankForm
ViewShoppingListAction
shoppingLists.jsp

Appendix F: Final User Interview

1 How well did the rotation diet project website meet your expectations overall?
   a. It mostly met them, but I wasn't able to test it thoroughly because of a server problem.
   b. Once I understood not to use the back button on my browser, I found the website very helpful. It made menu planning much easier and having the food families so easily available was very handy.
   c. I was pleased with how it turned out, although there are still some bugs.

2 Did you find the website easy to use? Why?
   a. Yes. The layout was logical.
   b. Yes, because it was understandable to me, a person who is barely computer literate.
   c. Yes, especially once the buttons on the food list and meal plan screens were fixed up so you could go backwards and forwards without messing things up.

3 Do you think that the site would be easy for others to use? Why?
   a. Yes. It is simple to use.
   b. Yes, because it was easy for me to use.
   c. Mostly. I think it might be harder for someone who hasn't been on a rotation diet very long to figure out. I think it needs more instructions.

4 Would you continue to use the web site to create your meal plans if the website remained available? Why?
   a. Yes, because I don't have to track the rotation manually.
   b. Yes, because it makes the meal planning much easier.
   c. Yes, because it is much easier than doing meal plans manually.

5 Was there anything that you expected to be included on the website that was not?
   a. No.

b. Being able to change the amounts of food that needs to be bought right on the shopping list and then print it.
c. Yes, I was hoping to have more instructions on the screen to tell how to do things.

6 Was there anything that you would like to see included on the website in the future?
a. I would like to see recommended serving size and basic nutrition information such as fat and carbohydrate content for each food.
b. A user guide to explain how to use the program.
c. I had a number of minor crashes where I didn't lose any information but had to log back in to get back to where I had been working.
d. It would be nice to be able to change the amounts of food on the shopping list on the screen instead of having to do it after printing.

7 Comments
a. I have set up rotation meal plans at the same time manually for three people with different allergies. It was a nightmare. This program even with a few bugs still needing to be worked out was a breeze.

Appendix G: Use Cases

Use Cases

1. Create Account (username/password combination)
    1.1. Enter unique Username
    1.2. Enter Password
    1.3. Enter Email address
    1.4. Confirm Email address (re-enter)
    1.5. Insert new account into database
2. Login to Account
    2.1. Enter Username
    2.2. Enter Password
    2.3. Validate username/password combination
3. Logout of Account
    3.1. Enter Email address
4. Create Food List
    4.1. Enter Food Rotation in Days
    4.2. Enter Food Family Rotation in Days
    4.3. Insert empty Food List into database for current user
    4.4. Modify Food List
5. Select Food List
    5.1. Select one existing Food List belonging to current user
6. Modify Food List
    6.1. Select Food List
    6.2. Add desired Foods to Food List
    6.3. Remove undesired Foods From Food List
7. Delete Food List
    7.1. Select Food List
    7.2. Delete from database:
        7.2.1. Meal Items (Foods) associated with selected Food List
        7.2.2. Meals associated with selected Food List
        7.2.3. Meal Plans associated with selected Food List
        7.2.4. Food List Items associated with selected Food List
        7.2.5. Selected Food List
8. View/Print Food List
    8.1. Select Food List
    8.2. Display selected Food List
9. Create Meal Plan
    9.1. Select Food List
    9.2. Enter Start Date
    9.3. Enter Number of Days Planning
    9.4. Create Meals
    9.5. Modify Multiple Meals
10. Modify Meal Plan

10.1. Select Meal Plan
10.2. Choose an action:
10.2.1. Add a Meal to meal plan
10.2.1.1. Create Meals
10.2.1.2. Modify Multiple Meals
10.2.2. Modify Meal
10.2.2.1. Select Meal
10.2.2.2. Modify Single Meal
10.2.3. Delete Meal
10.2.3.1. Select Meal
10.2.3.2. Delete from database:
10.2.3.2.1. Meal items (foods) belonging to selected Meal
10.2.3.2.2. Selected Meal
11. Select Meal Plan
11.1. Select one existing meal plan belonging to current user
12. View/Print Meal Plan
12.1. Select Start Date
12.2. Select Number of Days
12.3. Display Meal Plan(s)
13. Delete Meal Plan
13.1. Select Meal Plan
13.2. Delete from database:
13.2.1. Meal Items (foods) associated with selected Meal Plan
13.2.2. Meals associated with selected Meal Plan
13.2.3. Selected Meal Plan
14. Create Meals
14.1. Enter meal names
14.2. Insert empty Meal(s) into database
15. Modify Single Meal
15.1. Add desired foods to selected Meal
15.2. Remove undesired foods from selected Meal
16. Modify Multiple Meals
16.1. For each Meal:
16.1.1. Add desired foods to Meal
16.1.2. Remove undesired foods from Meal
17. Select Meal
17.1. Select one existing meal belonging to the selected meal plan
18. View/Print Shopping List
18.1. Select Start Date
18.2. Select Number of Days
18.3. Display Shopping List

Appendix H: Glossary

Account	Allows the user to securely access and store their personalized food lists, rotation schedule, etc. without worries of another user accidentally (or maliciously) modifying information.
Datasource	A source of data for an application. Commonly some form of database, but may be an xml or text file, etc.
Design Pattern	A standardized programming template that has been shown to be a successful solution in specific instances.
Entity Relationship Diagram (ERD)	A diagram that shows the relationships between database tables.
Food	Refers to a specific food, such as apples or fish, not food in general.
Food Family	Refers to the biological family of a specific food. For example, apples belong to the Rose family, and tomatoes to the Nightshade family.
Food List	A list of all the foods that a particular person can eat.
Framework	A collection of design patterns and best practices that forms a generic application template and is designed with a specific function in mind. For example, the Struts framework is designed to form the underpinnings for Java web applications.
iBATIS Framework	A framework that decouples SQL code from Java applications, improving the extensibility and service life of the application.
Integrated Development Environment (IDE)	Computer software designed to make building computer software faster, easier, and more efficient.
Java	A popular object-oriented programming language.
Java Database Connectivity (JDBC)	Standardized Java classes designed to allow Java applications to connect to and interact with databases.
Java Class	The main components of a Java program.
Java Server Page (JSP)	Text documents that can be translated into working Java servlets by a servlet runner application.
Java Servlet Runner	A computer program that powers Java web applications.
Login	A user's unique Username and password combination

Meal Plan	Planned meals created for a specific date, using a rotation schedule.
Model/View/Controller (MVC)	A design pattern that helps decouple (keep separate) the main components of an n-tier system. This can make a system more extensible so it is useful longer.
Normalization	The process of optimizing database tables.
Resin	A Java servlet runner created by Caucho Technology, Inc.
Rotation Diet	A diet for patients with severe food allergies, designed to prevent them from becoming allergic to the foods they are not already allergic to.
Rotation Schedule	Schedule that determines when a particular food may be eaten, such as 3/5 day or 4/8 day. The numbers in the schedule refer to the number of days *between* any day a particular food or food family is eaten and the next day the food/food family may be eaten. Example: on a 4/7 day rotation schedule, if the patient eats apples on Monday, another food from the Rose family may be eaten on Saturday (four days between), but apples may not be eaten again until the next Tuesday (7 days between)
Shopping List	List of foods needed for the selected meal plans.
Structured Query Language (SQL)	A programming language used to interact with databases.
Struts Framework	A framework based on the MVC design pattern.
Tomcat	A Java servlet runner created by the Apache Software Foundation.
UML	Unified Modeling Language
XML	Extensible Markup Language

Appendix I: Views Sample

Figure 2: Login Screen

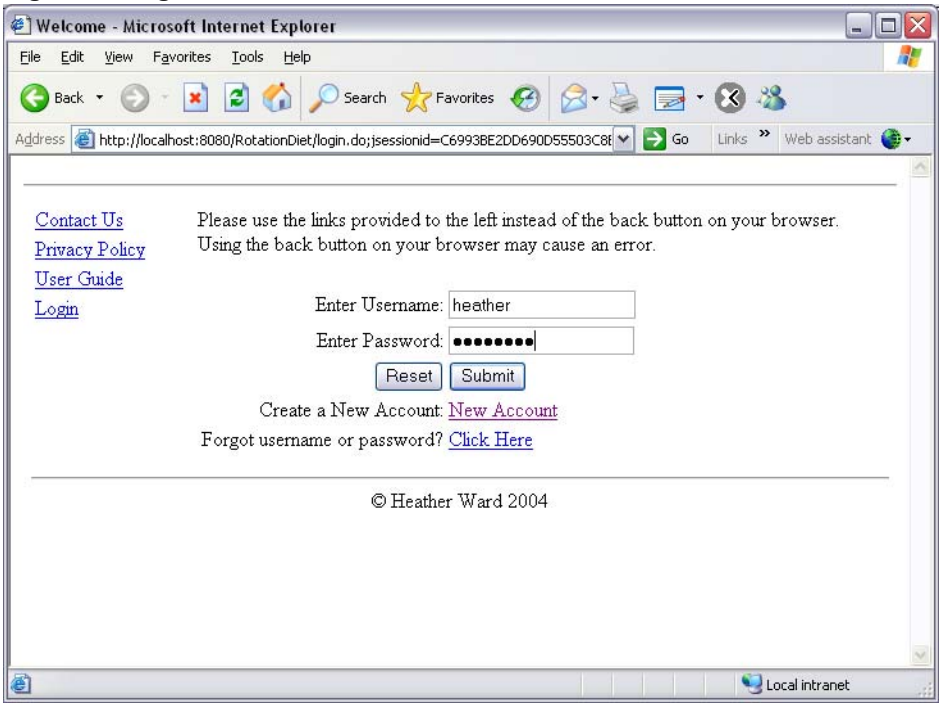


Figure 3: Create Meal Plans Screen

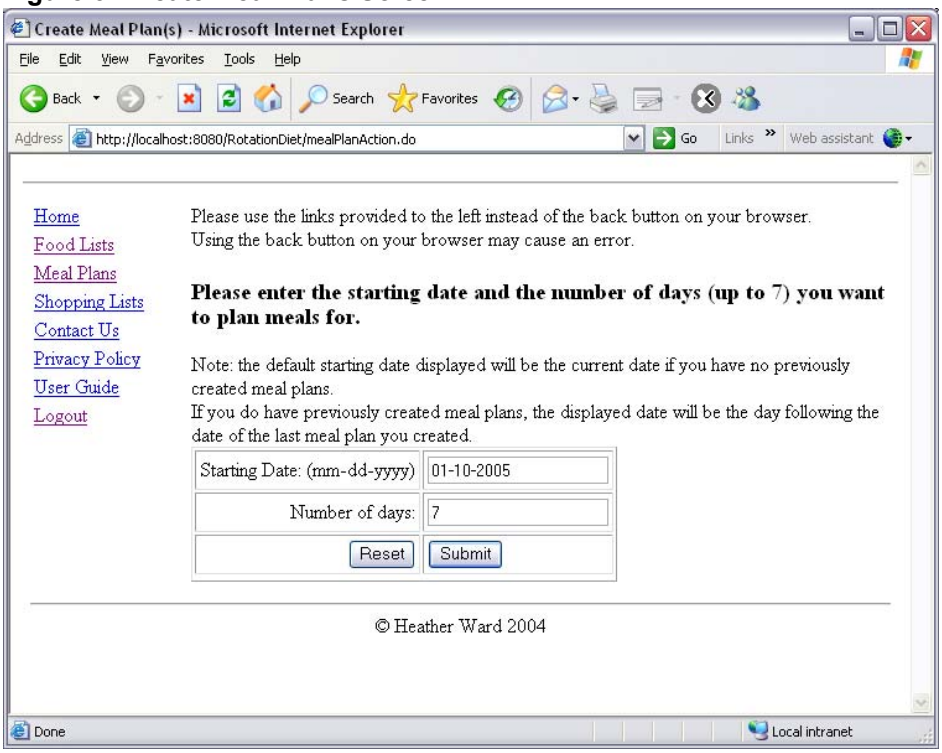


Figure 4: Choosing the Meals to be Created for the Meal Plans

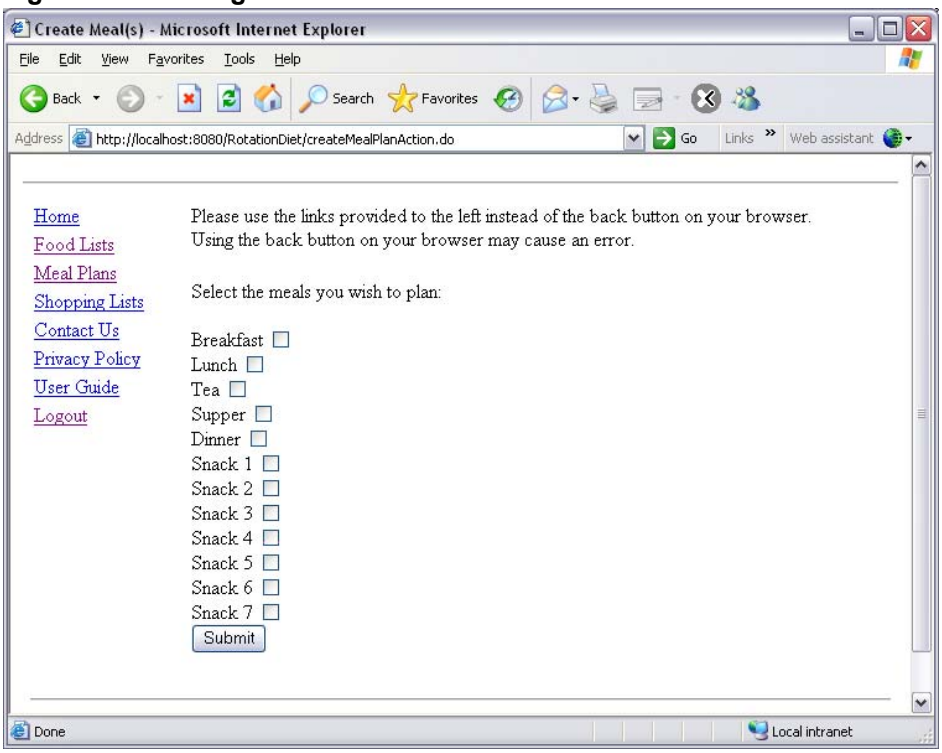


Figure 5: Selecting Foods for a Meal

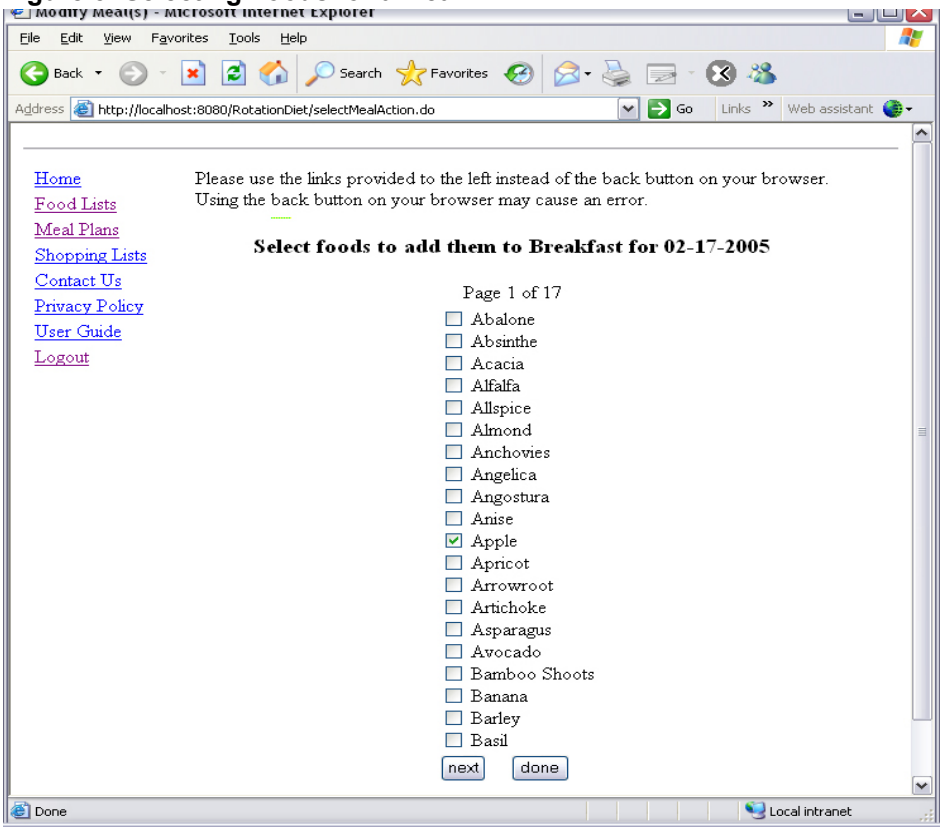


Figure 6: View of a Completed Meal Plan

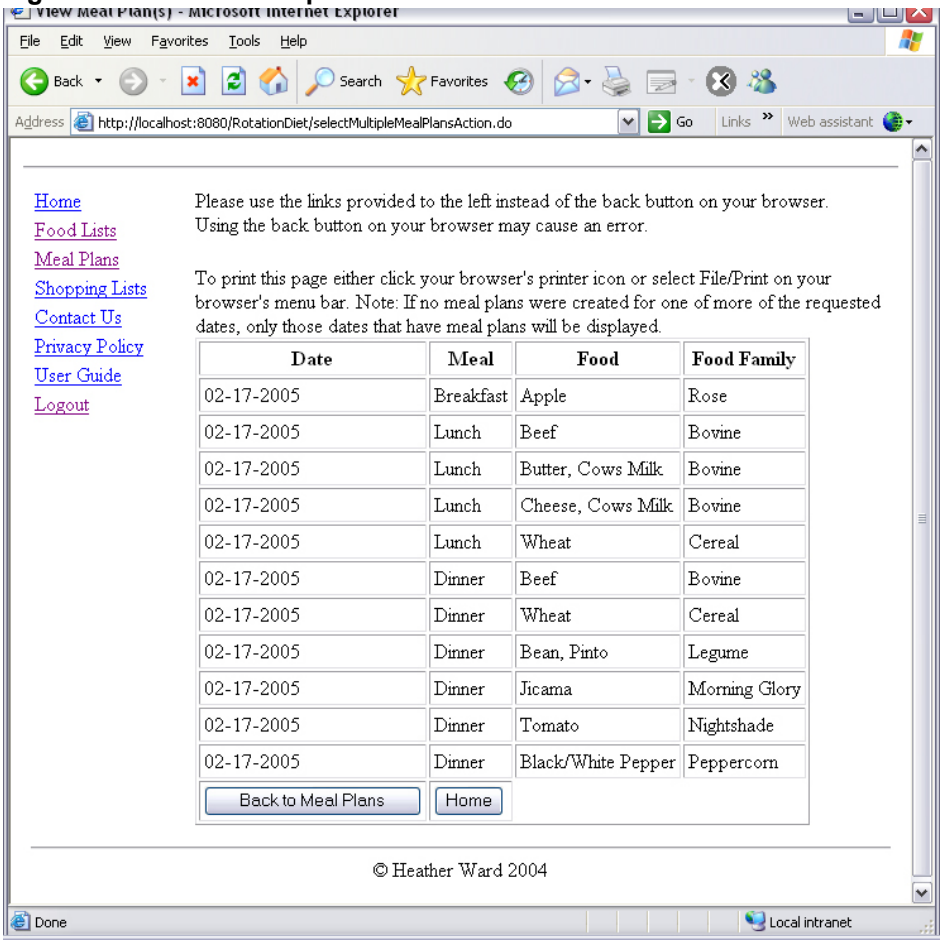


Figure 7: Shopping List for the Meal Plan Shown in Figure 6