CELLPHONE SHOPPER

Graham Hunter
ghunter@cs.uct.ac.za

Supervisor
Dr. Hussein Suleman

Honours Report 2007

ABSTRACT

Shopping is often a process that requires more planning and frustration than it should. For example: in one household a family member decides what should be bought, but another does the actual shopping. This can lead to items being bought that shouldn't and vice versa. The Cellphone Shopper system seeks to make shopping easier by solving problems like this. By allowing users to specify what they wish bought and where, and other users being able to view and buy these items while on the move, it is hoped that shopping will no longer be a source of frustration. This report details the design of the Cellphone Shopping system, as well as the implementation and testing of the backend component.

General Terms
Documentation, Performance, Design.

Keywords
Shopping, REST, Web Services, Web Server, Database.

Table of Contents

List of Figures

1. Introduction

1.1 Problem Outline

Grocery shopping can be a burden, especially in situations where one person decides what must be bought and another does the actual shopping. Some of the problems encountered while shopping are:

- Difficulty in sharing the shopping list – what is the list written on and where is it kept?
- One person adding something to the list and another wondering who added it and why
- Going shopping, only to realise that the shopping list has not been brought
- The buyer not knowing which brand of item to buy
- Co-ordination: who does the shopping and when?
- When the buyer gets to the store, they do not know where items can be found and spend a long time wandering around the store looking for them.

The key aim of this project is to make grocery shopping easier by using technology to aid the process. This does not mean that technology will be used to automate the shopping process entirely; rather, the aim is to use it to provide a tool for shoppers to use to make shopping simpler.

Two types of communication technology are used: cellular telephony and the Internet. The goal is to allow a household to share and manipulate a shopping list stored on a central server via a Web interface or a cellphone. The typical use case is that of one user creating the shopping list for the current day or week, while another can then view that list when they go shopping. If any changes are made to the list, all the users can see them. This system will be referred to as the Cellphone Shopping system.

1.2 Solution and Division of Work

The Cellphone Shopping system is split into three components: a Web interface, a mobile application and a server component managing the data for both of these. Marc Pelteret will be implementing the Web interface, which allows users of the system to manipulate and add shopping lists, as well as more complicated features that cannot be done on the mobile interface. Tshifhiwa Ramuhaheli implemented the mobile application which allows users of the system to view and manipulate shopping lists while on the move. Finally Graham Hunter implemented the server component that dealt with integrating the interfaces and maintaining persistance of data. Given below is a high level overview of the system.

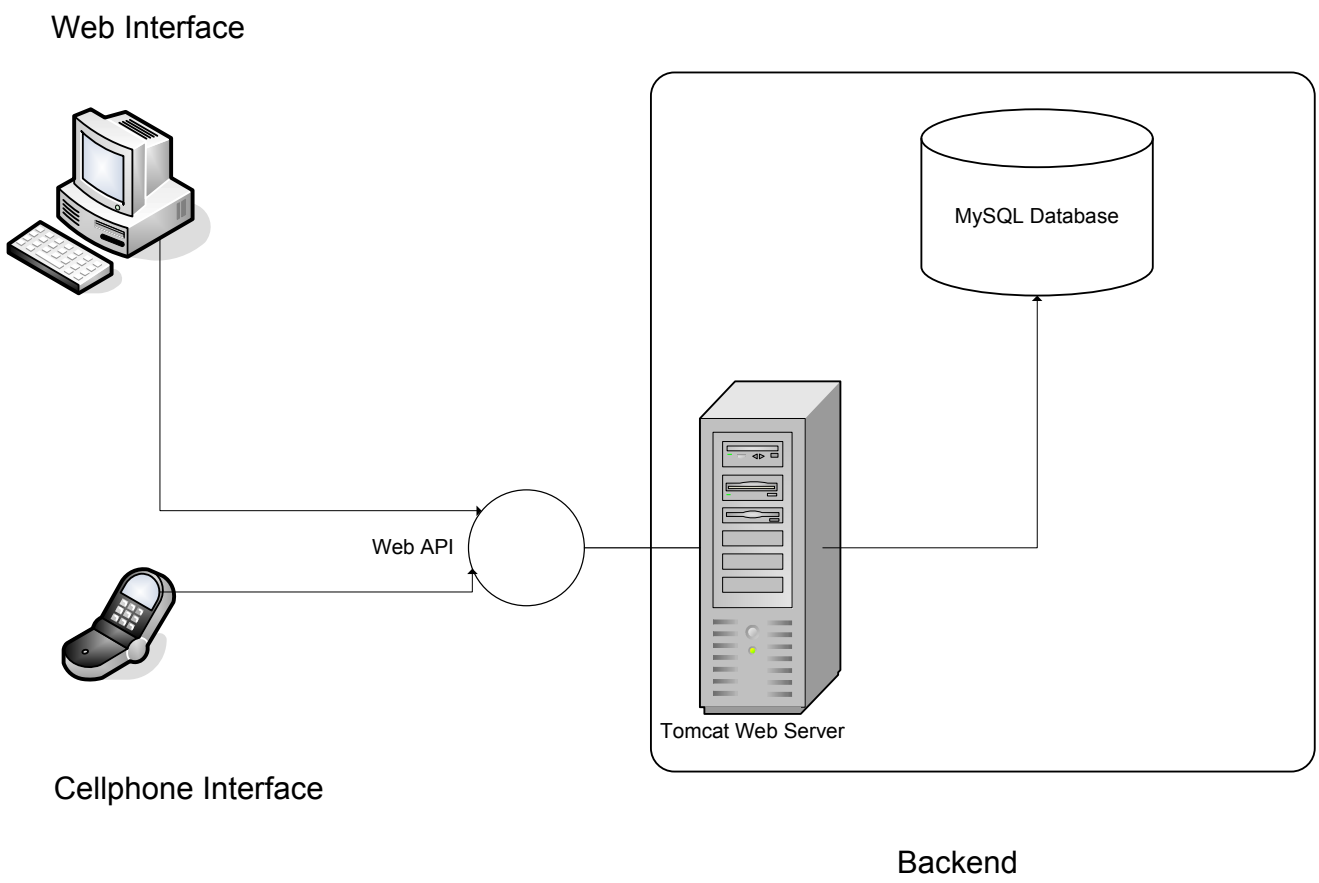

Fig 1: System Overview

1.3 Report Outline

This report deals with the overall system development, with specific attention paid to the backend component. In the next section the background literature relating to shopping behaviour will be explored as well as similar systems and technologies that could be used. In section 3 the software methodology used and the design of the system will be outlined as well as the proposed implementation. Section 4 describes the testing methodology used to evaluate the system, and a discussion of the results obtained . Section 5 will summarize the project along with the lessons learnt. Finally Section 6 will conclude the paper with possible future work.

2. Background

2.1 Shopping Behaviour

The design of the system relies on how people behave in a shopping environment, what functionality would be used and where. Based on a user study several common shopping behaviours are determined [15].

There exist two kinds of shoppers, those who make mental lists and those who make physical lists. Those who keep physical lists tend to carry it on their person. Some occasionally leave the list in the shopping cart. Many of those who keep physical lists mark off items as they are bought, showing the need for interaction between the shopper and their list. Shoppers who keep mental lists rely on browsing behaviour to remind them of what items should be bought.

Shopping typically involves three phases:

- Pre-shopping phase which is where the shopper plans to go shopping and creates a list.
- Shopping phase where the person is choosing the items.
- Checkout phase where payment is made.

Shoppers often pick up items or otherwise use their hands while shopping.

Even though mobile phones are used more often than PDAs, the size of a PDA was preferred over that of a mobile phone for use while shopping.

Shoppers tend to visit the same shop often and follow the same route.

2.2 Related Systems

Systems that are related in some way to the proposed one are discussed below.

2.2.1 Similar Architecture

Listed below are example systems that deliver data to either a Web interface or a mobile interface using a client/server approach.

2.2.1.1 Google Maps

Google Maps is a Web application that allows one to view road maps through a Web interface [11], as well as a mobile interface [12]. This application allows the user to view real-time traffic, get directions and search for locations. Using SMS mobile clients are limited to searching maps by text, but using a mobile application the client can view maps graphically, communicating with the server through the use of WAP. Much like the Cellphone Shopping system, Google Maps interacts with both a Web and mobile interface to provide information to users. The difference is that the system does not deliver personalized information.

2.2.1.2 Google Calender

Google Calender is a similar application to Google Maps, but provides functionality for creating and maintaining personal and public calendars [13]. Events can be added by using natural language such as "Brunch with mom at Java Cafe 11am on Saturday" which is then parsed. Users can choose to receive SMS or email event notifications on their mobile device. Calendars also can be shared between users,

allowing multiple people to edit the calendar depending on the access rights. The application can be accessed through a Web browser or a mobile device. Google Calendar is very similar to the Cellphone Shopping system, architecturally and functionally. It allows the management of a list through similar interfaces, provides personalized information and lets users collaborate on a single list.

2.2.2 Similar Shopping Systems

Systems that deal with providing shoppers services to help them in the store are discussed below.

2.2.2.1 Easi-Order

The Easi-Order system works by using PalmPilots to create a shopping list, then sending that shopping list to a store and having the items pre-picked [14]. The system can pre-populate the shopping list used based on the user's shopping history, as well as offer "impulse buy" suggestions. The store's products are electronically stored in a product index in categories and using a barcode scanner on the PalmPilots products can be entered into the list by scanning their barcode.

This system is similar in allowing shoppers to create and manage shopping lists. The difference is that each user has a separate list, whereas the Cellphone Shopping system allows multiple users to have one list.

2.2.2.2. U-Scan Shopper

The U-Scan Shopper system involves a monitor being attached to a shopping cart handle which can display promotions when the shopper is near certain items [16]. Additionally a bar code scanner is on the shopping cart allowing the shopper to scan items for themselves and pay at an automated station. Finally the system automatically generates a shopping list for each shopper based on their shopping history with that store, viewable on the monitor.

The similarity of this system is that it allows the shopper to view their shopping list while shopping. The difference is that the U-Scan system automatically generates this list, whereas the Cellphone Shopping system allows the user to create their own.

2.2.2.3. Mobile Shopping Assistant

The mobile shopping assistant is a system developed to allow shoppers to access Web services that are published by the store by using their mobile devices [2]. Some of the services available allow the customer to create a virtual shopping list, receive promotions and view a floor plan of the shop. The mobile devices used were normal mobile phones running a J2ME application. The mobile devices communicate with the Web services using SOAP messages, caching and compressing the messages for quicker data exchange. Some of the problems of mobile computing that this system tries to overcome include:

- Unstable network connection status.
- Limited bandwidth of mobile communication channels.
- No efficient XML parser.

Some of the problems encountered in development include local cache management on the mobile device, pre-fetching data from the Web services, data consistency and data conflicts.

In summary, there are many systems that have a similar architectural design in having a Web and cellphone interface connect to a server, and a few systems that deal with providing services to shoppers in stores.

2.3. Middleware

Middleware is a software component that sits between the network operating system and an application. It facilitates communication and management of distributed components in a network [10]. The following are examples of middleware.

2.3.1 Java Remote Method Invocation

Remote method invocation allows remote Java objects to be invoked from other Java virtual machines or remote hosts [6]. RMI has been in the Java SDK since version 1.1 and is used to develop distributed applications in Java. The benefit of using Java RMI is that it requires no additional libraries or modification of normal Java applications. One drawback is that the client needs an open port to communicate, which is not available if the server has a firewall. Compared to Web services, using RMI is significantly faster, if there are open ports to communicate in terms of network and hardware performance.

2.3.2 XML-RPC

XML-RPC is a way to make RPC-style function calls to a machine over a network [10]. XML-RPC communicates through the HTTP protocol using XML messages. The benefits of using XML-RPC is to have:

- An easy to implement standard.
- Discoverable services.
- Messages that can go through firewalls.

XML-RPC is different from other RMI protocols as it does not need stubs generated beforehand [7]. Instead primitives are used to construct requests and get responses. XML-RPC does not marshall arguments as all data is sent as text.

2.3.3. Web Services

Web services refer to an application component that can be accessed over the Internet and used remotely. Web services use XML and HTTP as these are interoperable standards [1]. Any computer that can communicate via HTTP can use a Web service. The difference between Web services and its predecessors is that it uses the SOAP protocol to communicate, has an interface defined using WSDL and is discoverable through the Universal Description, Discovery and Integration (UDDI) service. The benefit of using Web services is that of interoperability between different platforms such as .NET and

Java. A set of APIs for Java are available that can be used to implement a Web service, called the Java Web Services Developer Pack. One of these is the Java API for XML based Remote Procedure Call (JAX-RPC), which makes implementing a Web service similar to creating an RMI remote object. Web services differ from RMI in that they are executed in a Web container as servlets. The Web services classes can be converted to servlets or a servlet handler can be used to forward requests. Compared to RMI some of the features that Web services do not have include:

- Using object references.
- Distributed garbage collection.
- Dynamic class loading.
- Remote object activation.

Web services have decreased network performance compared to RMI, due to the overhead of using SOAP. Using SOAP messages, XML needs to be serialized and de-serialized, whereas RMI uses binary serialization.

Each middleware provides different functionality and varying degrees of interoperability depending on how each sends and receives messages. An overview of the message structure different middleware uses is discussed next.

2.4. Communication protocols

As bandwidth is limited when communicating with mobile clients, the messages each middleware uses should be minimal in size.

2.4.1 SOAP

SOAP requests consist of a header and a body. The body contains the XML request object, while the header has information not required for the request [7]. Given below is an example.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  xmlns:m="http://www.uct.ac.za/shop">
  <soap:Header>
    <m:ID>1234</t>
  </soap:Header>
  <soap:Body>
    <m:GetList>
      <m:ListNamee>Example</m:ListName>
    </m:GetList>
  </soap:Body>
</soap:Envelope>
```

Fig 2: SOAP Request message.

The SOAP protocol was not designed for wireless communication and does not consider the limited resources that a mobile device has [8]. SOAP is generally used over HTTP and TCP, which has the following limitations :

- Congestion avoidance in TCP assumes packet losses are because of congestion - in a mobile network this could be due to disconnections instead.
- Using TCP requires a connection to be established, resulting in increased overhead as acknowledgement packets are sent.
- The Network is not utilized well as a SOAP message that carries only a small amount of data will still require a connection being initialized.

By using client side caching and compressing the SOAP messages the performance of mobile clients can be significantly improved by up to 800%.

2.4.2 REST

REST uses standard HTTP GET requests to make calls to a Web resource [6]. The Web Server managing the resource will parse the URI and decide how to handle the request. The parameter data is included as HTTP request parameters. The protocol is the simplest but relies on the HTTP protocol.

```
GET /List?ListName=Example HTTP/1.1
Host: www.uct.ac.za
```

Fig 3. REST request message.

2.4.3 XML-RPC

XML-RPC is verbose compared to non-XML protocols like REST [9], but uses fewer bytes per message than SOAP. A request message that passes one integer is 168 bytes, of which 41 bytes are used to mark up the integer parameter. Compression can be used on these messages to reduce the size. Another drawback of XML-RPC is that retrieving data using XPath can be complicated as the elements are not named semantically [6].

```
<?xml version="l.0" encoding=
"ISO-8859-i"?>
<methodCall>
<methodName>act. GetList</methodName>
<params>
<param>
<value><string>Example</string></value>
</param>
</params>
</methodCall>
```

Fig 4: XML-RPC request message.

The protocols differ in how many bytes each message needs to be, with REST requiring the least, and SOAP the most. XML-RPC lies in the middle, but requires complex parsing.

2.5. Web Servers

The choice of what Web server software to use is important as it must support whatever middleware is chosen, as well as being easy to maintain and develop for.

2.5.1. Jakarta-Tomcat

Jakarta-Tomcat is the Apache Software Foundation's project for handling servlets and JSPs. A servlet is a program that is executed in response to an HTTP request and returns an HTTP response [5]. Servlets are object classes and are run inside the Java Virtual Machine, which means servlets can run on many operating systems. As the software runs Java it allows the use of Java-RMI as well as Web services.

2.5.2 Microsoft Internet Information Services

Microsoft Internet Information Services (IIS) is a Web Server built into the Microsoft Windows framework that can handle Web services and Web applications developed in ASP.NET [18]. As IIS does not support Java, Java RMI cannot be used. When developing Web services in ASP.NET many of the details such as creating WSDL documents, creating the SOAP messages and Web Discovery Service File are done automatically by Visual Studio .NET [17].

Summary of related work

Shopping behaviour has been explored in user studies before. Shoppers either keep a physical list or rely on the shop environment to remind them of what to buy. Meaning that a shopping system should offer a list management service and the ability to view a floor plan of the shop with items listed. Shopping can be divided into three phases: a phase where the shopper decides what to buy; a phase where they choose the items at the shop; and a phase where they pay for the items. The shopping system should provide different services at each stage. Shopping involves the frequent use of both hands and constant attention, therefore the system should be fast to use as well as usable by one hand only.

Based on this the Cellphone Shopping system included the following in the design:

- Allowing the users to view and manage shopping lists on the Web interface before shopping and on the mobile interface during shopping.
- Users can view a floor plan of the shop as well as design their own.
- Items can be labeled as bought, either while shopping or afterwards by both interfaces.
- The cellphone interface was designed so that the most common action at a certain period, for example buying an item, had the least amount of button presses.

In terms of similar systems there are many that use both a mobile and a Web interface to show data. For the mobile interface multiple communication protocols can be used such as SMS for simple text response/request messages or a mobile application that uses GPRS to interface with the server. Systems

that focus on providing mobile services while shopping have been implemented before. Some automatically generate a shopping list based on previous purchases, while others allow the shopper to create and manage their own lists. One used a PalmPilot as a mobile device, another a custom device attached to the shopping cart and finally one used a mobile phone. Neither system had a Web interface providing similar services that the mobile device had. The problems highlighted in developing these systems involved those related to mobile computing. Mobile devices have unreliable network connections, limited bandwidth and a limited XML parser. Some systems use client-side caching of data and compression of the messages used to increase performance. In the implemented system the mobile client did use client-side caching of data, as well as sending updates in batches.

Two middleware choices were covered, Java RMI which allowed a normal application to communicate remotely with another, and XML based Web Services which requires a Web Server. Performance wise Java RMI uses less network bandwidth due to the shorter messages and less CPU resources because of not needing to serialize and de-serialize XML messages. The drawback is that it is less interoperable than Web Services as the clients need to be able to run Java code. The choice of middleware is important due to the limited resources the mobile client has. Using tomcat as a server allows the use of Java RMI, Web Services and normal Web Applications, whereas using IIS without Web Services drastically reduces interoperability.

What was chosen for the project was the use of a REST based Web Service, due to its simplicity and minimal message sizes. The Web Server used was Jakarta Tomcat as it allowed the use of Java which the developers were familiar with.

3. Design and Implementation

3.1. Design Motivation

Other Web based shopping systems have been implemented before but the design focus was on providing a system specific to a shop that bought the items for the user. Similar mobile-based systems have been implemented overseas and been successful, highlighting a potential need for a shopping system in South Africa.

This is supported by the background research done, which shows that shoppers do have a need for a system that would aid in shopping. The ethnographic research done suggests that the focus for the mobile component must be ease of use. The Web site should then allow the more time consuming and harder functions to be done while the user is not shopping. The backend component is then necessary to integrate the two. The communication protocol used between the components should be minimalist due to the mobile device's low bandwidth.

3.2. Software Patterns

Software patterns provide proven and tested architecture designs that take into account issues not readily apparent. Using applicable software patterns is then important to reducing errors in the project. The software pattern used for the Cellphone Shopping system is described below.

The Model-View-Controller framework divides an application into three parts, the model where the core application is, the controller which directs the flow of the actions, and the view which handles presentation [3]. In this system the Web and mobile device interfaces would be the View, the Controller would be the Web API that links the Web and mobile application inputs to actions that the component handling the data will perform. The model responds to input from the controller, returning data from the database or changing the state of the view depending on the business model. The benefit of this model is that individuals can develop each component separately.

3.3. Software Model

The software model used throughout development was an agile method. Agile development methods involve multiple iterations and the use of prototypes with the focus being on quick development. This addresses the problem of requirements changing while the system is still in development. Agile methods also emphasize the use of small teams and face to face communication, suitable for this system.

The specific agile method used was Feature Driven Development, which focuses on a model driven short iteration process. The first stage is to develop an overall model of the system which defines its scope and context. This model is comprised of domain area models relating to specific areas of the system.Once this is done a list of features is identified, each taking no longer than two weeks to implement. Once the features have been detailed development is planned around implementing the list.

Once a feature is implemented it is tested thoroughly. The system had two iterations, with a feature gathering stage at the start of each. Shown below is a representation of the method.

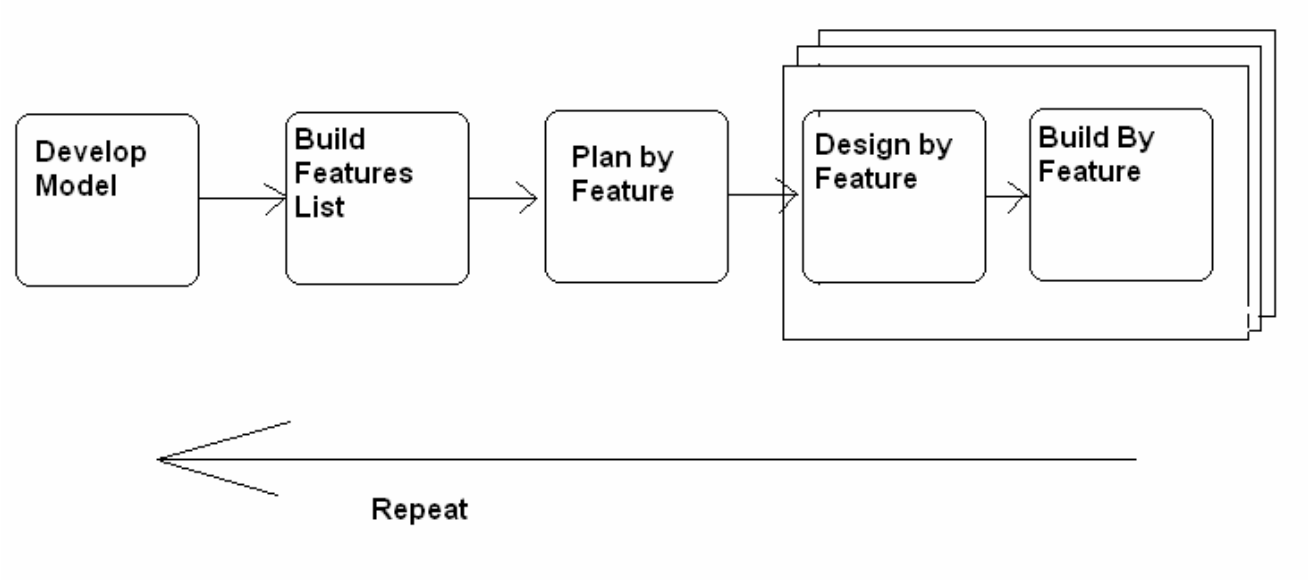

Fig 5: Feature Driven Development Lifecycle

3.4. User Requirements

3.4.1. User Interviews

To establish the features the user wanted, interviews were conducted using non-expert users with differing shopping behaviours. Users were asked about their shopping behaviour, what they would like from a Cellphone Shopping system and whether they would prefer to use a mobile device or a Web client. From this preliminary study, features were extracted which were used to develop a prototype.

3.4.2. Prototyping

Based on the initial features a prototype backend was developed to test what technology should be used in the system. This was an evolutionary prototype with the final system building on it. The prototype consisted of a Tomcat Web Server providing a SOAP Web Service that interacted with a mySQL database. This helped to determine what features were possible and their implementation.

3.5. Overview of Features

Based on the user studies done the features that each component supported in order to meet the system requirements can be divided into several categories.

3.5.1. List management

Features that fall under List management deal with creation or deletion of a list as well as modifying and maintaining a list's settings. As such the system supports the user adding multiple lists, deleting multiple lists and changing the settings of a list such as the list name, the access rights that a user has for a list and the default shop associated with the list.

3.5.2. List

Features that fall under this category have to do with adding or deleting items from a list, being able to represent the list in some way, and able to support different types of lists. Specifically the user can add an item to a list or edit an existing one. An item has attributes specific to the list it is part of, namely: quantity, the shop it should be bought from, an optional note attached it, whether it is a private item and only viewable by the user who added it and whether it is uncertain if the user wants it. Additionally the user can either delete an item, where it is removed from the list and added to deleted items, or checkout an item, where it is removed and added to previous items.

The list allows different ways of being presented, either alphabetically, by a predefined shop layout, or by item category. Special types of lists are supported: a favourites list, previous items bought, private items and deleted items.

3.5.3. Shop layout

The system allows shops to have a layout defined for them, either generated by an administrator or by a user. Each user can only have one layout per shop. The user is able to construct the layout using the website and then sort the items on a list using it.

3.5.4. User preferences

The system stores certain information about users. Each user will have a unique user name chosen by them which is used to login via the Web site or cellphone along with their password. The password is transferred in an encrypted form such that the backend cannot reverse-engineer it to plaintext.

Personal details about each user is stored: their first name and surname, their email address and their cell phone number. Each user also has a list of trusted users that they have added who they can grant access to their personal lists.

3.5.5. Reminders

Users are able to set reminders to pay items like utility bills that need to be paid. Reminders will have a name, a description and the date that it is due for as well as a notification period how long before the date the user must be notified. The reminder can also be added as an item, appearing on every list of the user except for favourite and previous item. The user will have the option of removing the reminder, which will also remove it from any lists it is part of.

3.5.6. Notes to others

The system allows users to send messages to other users using a gateway service, such as SMS or email.

3.6. System Overview

In order to implement these features the Cellphone Shopping system was divided into three components: the Web-based user interface, the mobile device application and the backend server component. The components were loosely coupled in such a way that failure of one should not stop another from fulfilling the requirements. Communication between each component will be achieved through HTTP, with the responses from the backend in XML.

3.6.1 Technologies Used

The implementation of the backend has three layers. As the system relies on long term storage of data, a database was used. MySQL was decided upon as it is a free and stable database server which has performance comparable to other commercial products such as Microsoft SQL Server.

As the backend needs to support both a Web interface and a mobile client it was decided to provide a REST based API that would be accessible through HTTP GET and POST requests from either interface. This was achieved using Java servlets to manage calls. Responses are returned in XML, which can be parsed by both interfaces. Compared to SOAP based web services the interfaces can make calls far easier using URL encoded data, fewer bytes are used in each message which impacts the mobile client as bandwidth is limited, as well as providing the greatest degree of interoperability with both interfaces accessing the API in the same manner. To support this Tomcat was chosen as the web server as it has good support for Java servlets.

In order to implement the business logic of the system another layer between the database and the Web server was used where POJOs held the data and format it as required by the interfaces. Persistance in the system will be handled manually, as a persistence manager would increase the system overhead and was seen as unnecessary given the scope of the project.

3.6.2 Database Layer

The database was designed as described below by this ER diagram. A detailed overview of the database will be given afterwards.

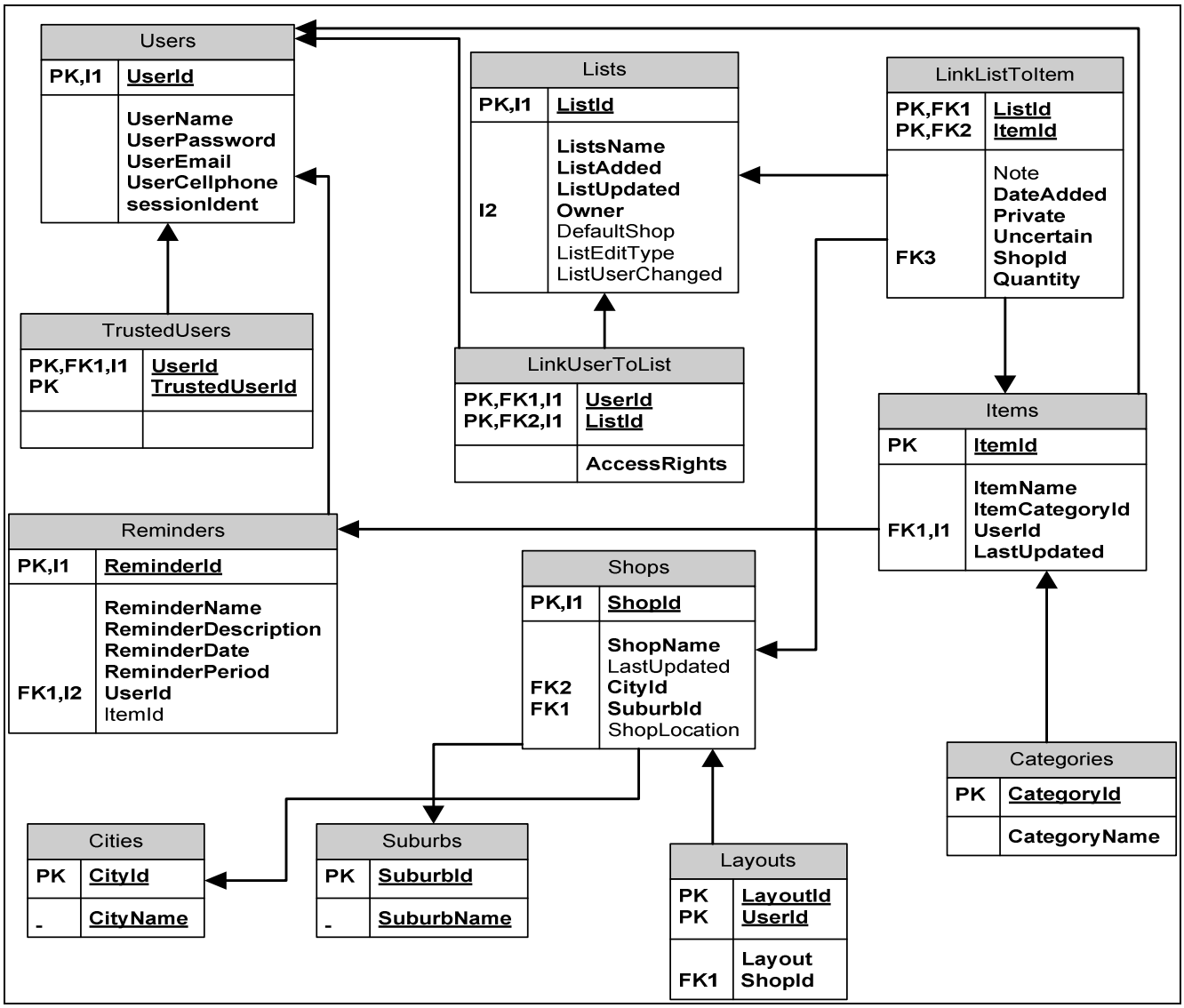

Fig 6: ER Diagram

Overview of the Database

Users: Each user needed to be represented uniquely in the system, which this table is necessary for. UserName and UserPassword are used to authenticate the user to the system; UserEmail and UserCellphone are used to send messages between users. SessionIdent is used to determine the last time the user was logged in, in order to determine what notifications they should be sent. Users has a many-to-many relationship with Lists, Shops and Trusted Users as well a one-to-many relationship with Items, Reminders and Layouts.

LinkUserToList: This table provides a many-to-many mapping between users to Lists, as one user can have many lists, and a list can have many users each with different access rights.

Lists: The Lists table stores the attributes specific to a list such as the name of the list, and the date it was added. ListUpdated is necessary to determine if a list was updated, with ListEditType specifying how it was updated and ListUserChanged which user changed it. Lists has a many-to-many relationship with items and users.

LinkListToItems: As a list can have many items, and an item can have many lists this table provides a many-to-many mapping between the two. The item attributes that are specific to its inclusion in a list are included here, such as quantity and the shop it should be bought from.

Items: This table describes the item attributes that are not specific to a list, such as the name, the category it belongs to, and the user that added it. This table has a many-to-many relationship with Lists, a many-to-one relationship with Users and a one-to-one relationship with Reminders.

Shops: This table will store the shop name, location and when it was updated, as well as providing a link to a city and a suburb. Shops will have a many-to-many relationship with Users, a one-to-many relationship with Layouts and Lists as well as a many-to-one relationship with Cities and Suburbs.

Layouts: Each layout is shop-and-user specific, with one user only allowed one layout per shop. Layouts will have a many-to-one relationship with Users and Shops.

TrustedUsers: This table stores the trusted users that a user has, with one user having many trusted users and vice versa.

Reminders: Reminders stores the attributes specific to a reminder, with one user having many reminders and one reminder having one user. As a reminder can also be linked to an item, itemid is included as a foreign key.

Cities: The Cities table will not be edited by the user but by an administrator. Cities has a one-to-many relationship with Shops. Cities will have a one-to-many relationship with Shops.

Suburbs: The Suburbs table will not be edited by the user but by an administrator. Suburbs has a one-to-many relationship with Shops.

Categories: The Categories table will not be edited by the user but by an administrator. Categories has a one-to-many relationship with Items.

Caching and Buffering of Queries

mySQL allows the use of caching database queries, which improves performance. The decision was to not use caching during system use as tables were updated too often for this to be of use. Buffering of queries, where the result set of a query is stored in a temporary table to free up table locks while the result is being returned, was handled automatically by the database optimizer.

Connection Pooling

Connection pooling refers to a technique whereby a thread requests database connections from a pool of already created connections. Each thread uses a connection exclusivly while it holds it. Once a thread has completed a transaction and the connection is idle, it is returned to the pool where other threads can utilize it. This increases performance as connections need not be created everytime, while being thread safe. Connection pooling was used in the system, with a pool size of 500.

Indexing

An index is a data structure that increases the speed of most operations. Indexes which have been created from columns allow faster random lookups, while requiring more disk space on the server. The primary keys in each table were automatically indexed: UserId; ListId; ItemId;ShopId and ReminderId. The foreign keys in each table were explicitly indexed. These were the variables used for select and update queries. The tradeoff of using indexes is that insert statements would be slower due to the index being updated, which is why ItemId was not used as an index due to the large amount of inserts being done in that table.

3.6.3. Logic Layer

The logic component comprises the Java classes that perform the business logic of the system. Java was chosen as it is the language that all involved have the most experience with. The attributes of each class is taken from the corresponding database representation. The following diagram details the classes involved in the system and their relationships.

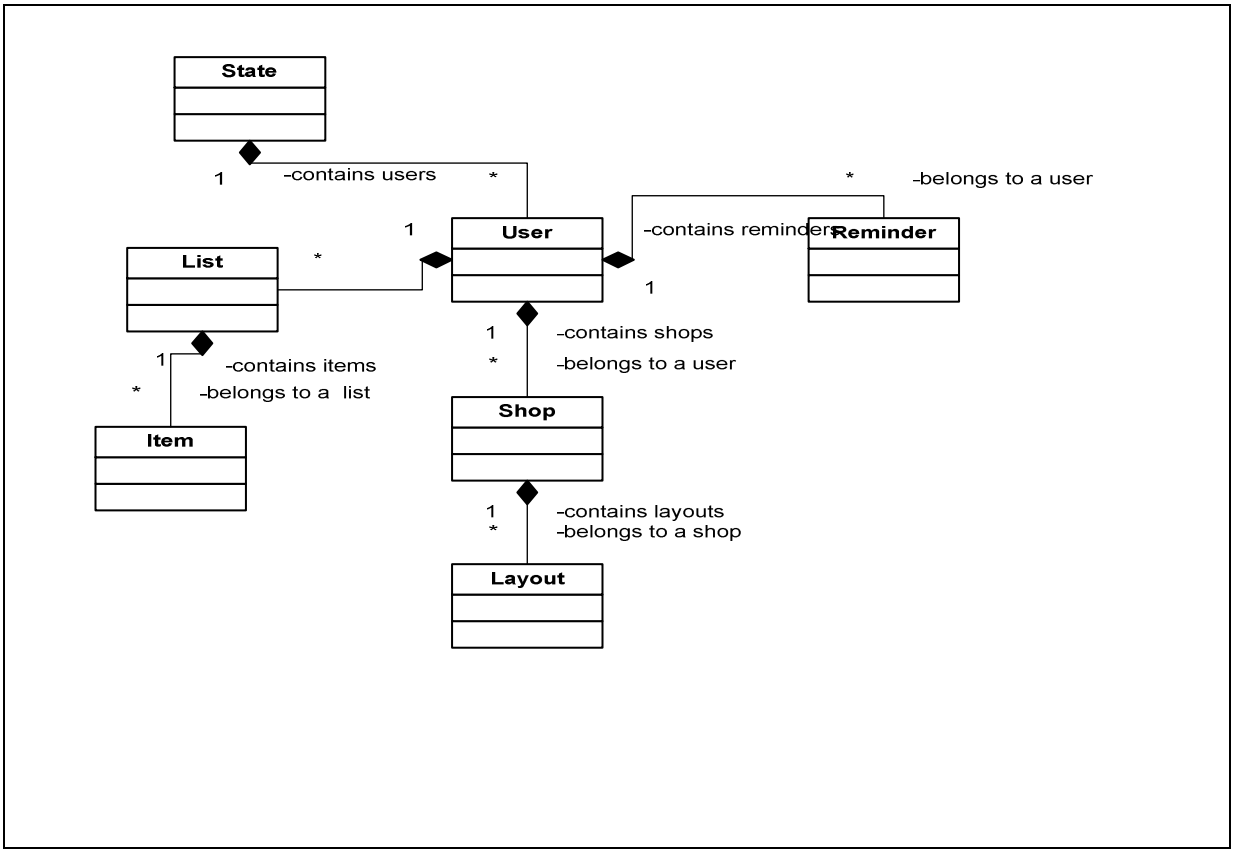

Fig 7: Class diagram

Persistance Management

When an object is instantiated, its attributes are initialized through the Load method that will load the attributes of the object from the database. This is done using JDBC to perform queries. The components of the object will also be instantiated and their Load method called when the container is initialized, resulting in a recursive initialization. This simplifies persistence in that as the User class is the super class of the others, only one method call needs to be made by the Web API. When object attributes are modified, saving the changes to the database will be done in a similar method by using the Save method, but this will not be recursive. Finally each class implements a Delete method that recursively removes from the database the rows that refer to it, and it's containing classes. Initialization is done up front, rather than lazily where it could be put off until the object was requested, as in many use cases many objects are requested in a short period of time. The sequence diagram of this process is given below.

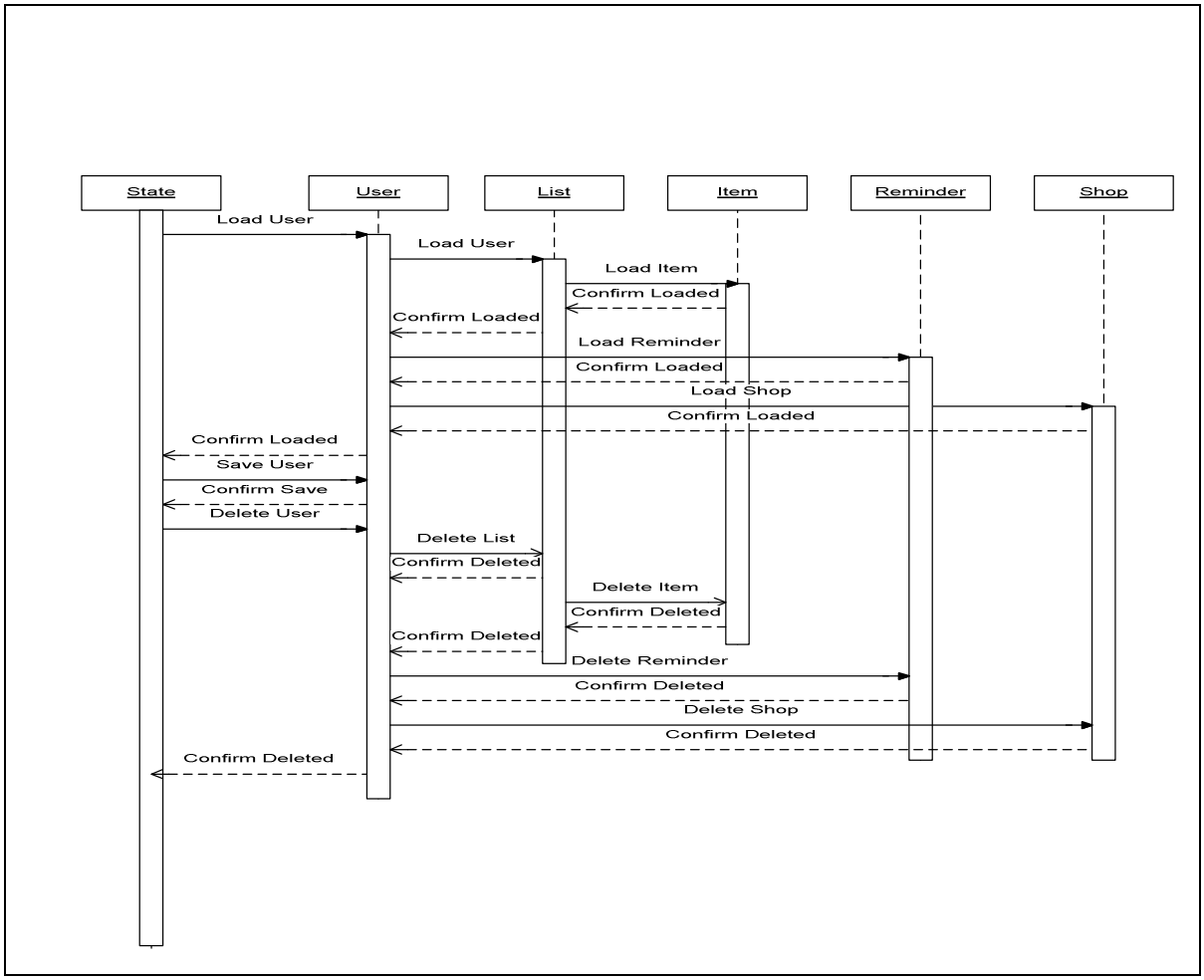

Fig 8: Persistance Mangement

Caching of Data

To reduce the amount of database connections made during system use, objects and their attributes are cached. This increases the amount of memory the system uses, but results in a greater throughput as a database connection need not be made for each request. As the User object is the super class that contains the others, a vector is used to store a soft reference to each User instance. A soft reference was used so that the object could be garbage collected if the system was running low on memory. The amount of users that are cached is limited to 100, with the older ones being replaced by the newer.

Each User object is cached for thirty minutes or when the user logs out, which is checked when another user logs in or out the system. Thirty minutes was chosen as it was seen as average shopping time that a user needs. When data needs to be accessed the appropriate user is searched for and if available it is instantiated, cached and returned. To achieve this, the vector of users is static, meaning all threads will

access the same object. To handle synchronicity a vector was chosen over a faster solution such as an arraylist or a linkedlist as it is synchronized.

To support synchronicity where shared classes are updated by different users, which will result in inconsistencies between cached objects and the database, each object checks whether the corresponding database table was updated before returning any details and if necessary will instantiate the object again. This is only applicable for returning the object information as adding and deleting operations are idempotent and do not rely on synchronicity between objects.

Authentication

When a login request is made to the Web API, a user name and password is needed for authentication. When the user object is instantiated these are compared to the database if there is a match the object will be instantiated and its id returned by the Web API. The system will assume from then on that any future calls using that id will be valid. This reduces the need of having to send the user name and password every time a request is made to the Web API. To protect the user's password it will be sent during the login request in an encrypted form using MD5, which cannot be reversed. This is done so that the backend cannot know the user's password protecting their privacy

Session Management

Each user of the system corresponds to a User object with the last time that they were logged in updated whenever the Login or Logout method is called. As users might not logout the system when they are done, sessions are closed after thirty minutes of inactivity.

Access Control

Access control in terms of the rights that a user for adding, editing and deleting items to a list, is managed in this layer as well as in the Web API layer. The rights are stored in a string with each character representing a right, with a value of 1 or 0. When a List object is initialized it loads the rights of the user that holds it, and when an operation is done on the list the rights are checked.

Notifications

The notifications that a user receives are updated based on the last time they logged in and the last time the lists and shops they have added have been updated. If the difference between the reminder date and the current date is less than the period then it will also be displayed.

Sorting

Sorting of items in a list based on their name and category is done using quicksort as this is one of the fastest algorithms for sorting with O(NlogN). The decision to sort the items in the logic layer instead of the database layer was as a result of having to sort by shop layout which required. This also has the

benefit in that multiple requests to sort a list by the same criteria means that the already sorted list can be returned unless it was updated.

Sorting by shop layout is done by sorting the items into each aisle by the categories contained in them. The path that will be suggested is that of following the aisle down to the end, then going back up the next aisle. An assumption will be made that the shopper would come from the top left side of the shop; if the entrance was to the top right the list would be read in reverse. Items not in the layout are placed last.

XML Parsing

Each class overrides the toString method and parses their attributes into XML form. This is makes it simpler for the Web layer to return results of queries. This method will not recursively call the other toString methods of other objects. An example of the result of this using the user class would be:

```
"<u id="3" n="UserName" f="Graham" s="Hunter"
e="hunter@test.com" c="083594"/>"
```

fig 9: User Attributes

Where id is the unique identifier of the object, 'n' is the user's name, 'f' is the first name, 's' the surname, 'e' the email address and 'c' the cellphone number.

Limits

To ensure that a user of the system doesn't overload it, limits are placed on the amount each container class can hold. Each list object can contain up to 40 items and each User object can contain up to 20 Lists, 20 Reminders, and 20 Trusted Users.

3.6.4. Web Layer

The Web layer deals with communicating between the two interfaces and the backend to provide the features of the system. It provides a REST-based API that will be accessible over HTTP. Web Services were used rather than RPC as this made integration easier.

Web Server

Jakarta Tomcat was used as the Web server due to its support for Servlets, as well as being open source and freely available. A Servlet is an object that receives a request and generates a response. A Servlet container, in this case Jakarta Tomcat, manages the Servlets and maps the requests to them. A Servlet is only initialized by the container once in their lifetime, where after it will service each request in a new

thread. Servlets are not destroyed after each request. Compared to CGI, Servlets are more efficient due to this, as well as allowing the use of connection pooling. This is due to Servlets not being destroyed after a request, so state can be kept.

The server was configured to have a maximum of 100 threads at one time, and the maximum idle threads was set to 50. Tomcat provides connection pooling support called DBCP. This manages database connections and was configured to remove abandoned connections when they had been idle for too long. Additionally connections were verified that they could be used whenever they were requested from the pool.

Web Service

Data is requested through GET messages, and updated using POST messages. In the initial prototype communication was done using SOAP but a REST approach was preferred as the messages were significantly shorter and marshalling the parameters was easier for both interfaces. Only GET and POST messages were used as neither interfaces had support for DELETE and PUT.

All methods are idempotent in that calling it multiple times will yield the same result except for adding an item, which will increase the quantity of that item. This makes a REST approach more applicable as it is suitable for stateless services.

The Web Services were developed using the Sun Web Developer Pack which provides an implementation of JAX-RS. JAX-RS is the Java API for RESTful Web Services that hides the lower-level details of using Servlets to implement a REST Web Service, by providing a high-level API that allows classes to use REST specific annotations to define resources and specify actions. An example of this is given below.

```
@HttpMethod(value = "GET")
@ProduceMime(value = "application/xml")
@UriTemplate("User_Details")
public String User_Details(@QueryParam("UserId") int UserId) {
```

Fig 10: Example RESTful Web service with annotations

This example responds to the URI http://example.com/User_Details?UserId=1. The annotations used in this example include @HttpMethod which defines what httpmethod it should respond to, @ProduceMime which defines what result it will return, @UriTemplate which defines what URI it will respond to and @QueryParam which specifies what parameters it expects. The URI templates will not follow REST standards which recommend not using verbs or queryfields, as since the interfaces only support GET and POST it can be ambiguous as to which method is being called.

The system makes the assumption that the user has permission to access any method in the API without authorization, with the exception of the user login and delete methods which require a user name and password. This is as a result of keeping to the stateless nature of Web communication.

Every method returns an XML response with either the object information if it is a GET request, the id of the affected objects if it was a POST request and an error message if an error was encountered because of incorrect parameters.

The Web service is split into 4 classes relating to what resource they deal with. Namely UserResource, ListResource, ShopResource and ItemResource, with each one responding to a different URI. Each of these will provide methods for creating, deleting and managing their corresponding resource. A comprehensive list of available method is given below.

User-related Methods

Name	User_Login
Description	Authenticates the user to the System, updating the last time they were logged in
Type of Request	GET
Parameters	UserName, UserPassword
Result	Returns the details of the user. “f” being the first name, “s” the surname, “e” the email address, and “c” the cellphone number.
Errors	<error id=”5”> Couldn’t Login User </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_Login?UserName=Hunter&UserPassword=123
Example returned XML	<u id="4" n="Test" f="Graham" s="Hunter" e="test@" c=" 083354 "/>

Name	User_Logout
Description	Logouts out an already logged in user of the system, updating the last time they were logged in
Type of Request	GET
Parameters	UserId
Result	Returns the id of the user
Errors	No error
Example Call	http://[IP]:[Port]/Server/resources/user/User_Logout?UserId=4
Example returned XML	<u id="4" n="Test" f="Graham" s="Hunter" e="test@" c=" 083354 "/>

Name	User_Create
Description	Create a new user in the system
Type of Request	POST
Parameters	UserName, UserPassword, UserEmail, UserCellphone
Result	Returns the id of the new user

Errors	<error id="11"> That UserName is already taken </error> <error id="15"> Couldnt create the User </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_Create?UserName=Hunter&UserPassword=123&UserEmail =hunter@gmail.com&UserCellphone=05473
Example returned XML	<id> 10034 </id>

Name	User_Edit
Description	Edit the details of a user in the system
Type of Request	POST
Parameters	UserName, UserPassword, UserFirstName, UserSurname, UserEmail, UserCellphone, UserId
Result	Returns the details of the user. “f” being the first name, “s” the surname, “e” the email address, and “c” the cellphone number.
Errors	<error id="1"> Couldn’t find user </error> <error id="12"> Incorrect Parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_Edit?UserName=Hunter&UserPassword=123&UserEmail=hunter@gmail.com&UserCellphone=05473&UserId=3
Example returned XML	<u id="4" n="Test" f="Graham" s="Hunter" e="test@" c=" 083354 "/>

Name	User_Details
Description	Get the details of a user in the system.
Type of Request	GET
Parameters	UserID
Result	Returns the details of the user. “f” being the first name, “s” the surname, “e” the email address, and “c” the cellphone number.
Errors	<error id="1"> Couldn’t find user </error> <error id="12"> Incorrect Parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_Details?UserId=3
Example returned XML	<u i=4 n=Test p=123 e=083504 c=hunter@ />

Name	User_AddList
Description	Add an existing list to the User.
Type of Request	POST
Parameters	ListId,UserId
Result	Returns the list id if successful, otherwise an error
Errors	<error id="1"> Couldn't find user </error> <error id="12"> Incorrect Parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_AddList?ListId=1&UserId=3
Example returned XML	<id> 460 </id>

Name	User_GivePermission
Description	Gives the secondUser access to a list that the first user belongs to, while adding the list to the second user.
Type of Request	POST
Parameters	UserId,ListId,SecondUserId,AccessRights
Result	Returns the id of the second user
Errors	<error id="14" > Couldnt add Trusted User </error> <error id="1"> Couldn't find user </error> <error id="12"> Incorrect Parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_GivePermission?UserId=3&ListId=1&SecondUserId=4&AccessRights=111
Example returned XML	<id> 460 </id>

Name	User_Delete
Description	Delete a user from the database, along with their lists, reminders, items, layouts, favourite shops and trusted users
Type of Request	POST
Parameters	UserName, UserPassword
Result	Returns the id of the deleted user
Errors	<error id="6"> Couldnt delete User </error> <error id="1"> Couldn't find user </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_Delete?UserName=Hunter&UserPassword=123
Example returned XML	<id> 460 </id>

Name	User_GetLists
Description	Get the details of the lists that a user has access to
Type of Request	GET
Parameters	UserId
Result	Returns the details of the lists, with the attributes: the id of the list, the id of the owner of the list, the name of the list, the access rights the user has to the list, the name of the default shop of the list, and the id of the default shop.

Errors	<error id="1"> Couldnt find User </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_GetList?UserId=3
Example returned XML	<Lists> <list id="6321" uid="1006" n="List" access="111" s="" sid="-1"/> <list id="6322" uid="1006" n="Another List" access="111" s="" sid="-1"/> </Lists>

Name	User_AddTrustedUser
Description	Add a user to another user's trusted users list, given the UserId and the TrustedUserId
Type of Request	POST
Parameters	UserId, SecondUserId
Result	Returns the id of the trusted user or one of the errors
Errors	<error id="1"> Couldnt find User </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_AddTrustedUser?UserId=3&ListId=1
Example returned XML	<id>4065 </id>

Name	User_GetTrustedUsers
Description	Get all the trusted users of a user.
Type of Request	GET
Parameters	UserId
Result	Returns the ids of the trusted users or one of the errors
Errors	<error id="1"> Couldnt find User </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_GetTrustedUser?UserId=3
Example returned XML	<id>4065 </id>

Name	User_DeleteTrustedUser
Description	Remove a trusted user from the user's list, given the UserId and TrustedUserId
Type of Request	POST
Parameters	UserId, SecondUserId
Result	Returns the id of the trusted user or one of the errors
Errors	<error id="1"> Couldnt find User </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_DeleteTrustedUser?UserId=3&ListId=1
Example returned XML	<id>4065 </id>

Name	User_AddReminder
Description	Adds a reminder to the user, given the Reminder's name, description, date (in the format dd-mm-yyyy), period of notification, and whether it should be added as an item (0 or 1).
Type of Request	POST
Parameters	UserId, ReminderName, ReminderDescription, ReminderDate, ReminderPeriod, ReminderItem

Result	Returns the reminder id if successful, or an error
Errors	<error id="8"> Couldnt Add Reminder </error> <error id=\"1\"> Couldnt find user </error> <error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_AddReminder**?**UserId=3&ReminderName=electricity&ReminderDescription=PayNow&ReminderDate=05-09-2007&ReminderPeriod=5&ReminderItem=0
Example returned XML	<id>4065 </id>

Name	User_DeleteReminder
Description	Removes a reminder from the user
Type of Request	POST
Parameters	UserId, ReminderId
Result	Returns the reminder id if successful, or an error
Errors	<error id="9"> Couldnt Delete Reminder </error> <error id=\"1\"> Couldnt find user </error> <error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_DeleteReminder?UserId=3&ReminderId=1
Example returned XML	<id>4065 </id>

Name	User_GetReminder
Description	Get a reminder from the user
Type of Request	GET
Parameters	UserId
Result	Returns the reminder details, empty set if none
Errors	<error id=\"1\"> Couldnt find user </error> <error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_GetReminder?UserId=3
Example returned XML	<Reminders> <r id="13770" n="Reminder Me" d="About This" date="30-10-2007" p="1" i="1"/> </Reminders>

Name	User_GetReminders
Description	Returns all the reminders a user has, with their details
Type of Request	GET
Parameters	UserId
Result	Returns the reminder details, empty set if none
Errors	<error id=\"1\"> Couldnt find user </error>

	<error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_GetReminders?UserId=3
Example returned XML	<Reminders> <r id="13770" n="Reminder Me" d="About This" date="30-10-2007" p="1" i="1"/> </Reminders>

Name	User_GetNotifications
Description	Get which reminders that fall in the notification period, as well as any updates to lists or shops since the user has last logged in
Type of Request	GET
Parameters	UserId
Result	Returns the list details and shop details if updates
Errors	<error id=\"1\"> Couldnt find user </error> <error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/user/User_GetNotifications?UserId=3
Example returned XML	<Notifications> <Updates/> <Reminders> <r id="13771" n="Remind me" d="Update" date="19-10-2007" p="6" i="1"/> </Reminders> </Notifications>

Item-related Methods

Name	Item_GetByUser
Description	Get all the items and their details that a specific user has added
Type of Request	GET
Parameters	UserId
Result	Returns the item details, empty set if not found
Errors	<error id=\"1\"> Couldnt find user </error> <error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/item/Item_Get?ItemId=1
Example returned XML	<Items> <I i=1 n=ItemTest c=CategoryTest f=1 usr=Hunter /> </Items>

Name	Item_Edit
Description	Edit an item details.
Type of Request	POST
Parameters	UserId, ItemName, ItemCategory

Result	Returns the item details, empty set if not found
Errors	<error id="1"> Couldnt find user </error> <error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/item/Item_Edit?ItemId=1&ItemName=Cheese&ItemCategory=Dairy
Example returned XML	<Items> <I i=1 n=ItemTest c=CategoryTest f=1 usr=Hunter /> </Items>

Name	Item_GetCategories
Description	Retrieves a list of all possible item categories
Type of Request	GET
Parameters	
Result	Returns the item categories
Errors	No error
Example Call	http://[IP]:[Port]/Server/resources/item/Item_GetCategories
Example returned XML	<Categories> <c id="4">Reminder</c> <c id="5">Baby Care</c> </Categories>

Name	Item_CheckoutItems
Description	Marks the sent items as bought. This will put the items on the user's previous items list Item ids are sent as a list deliminated by "|".
Type of Request	POST
Parameters	ItemId, ListId, UserId
Result	Returns the ids of the items
Errors	<error id="2"> Couldnt find list </error> <error id="1"> Couldnt find user </error> <error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/item/Item_Checkout?ItemId=1|2|3&ListId=1&UserId=3
Example returned XML	<id> 1|2|3</id>

List-related

Name	List_Create
Description	Create a new list, setting the user as the owner and the default shop, and adding it to the user
Type of Request	POST
Parameters	ListName, UserId, ShopId

Result	Returns the list id if successful, error otherwise
Errors	<error id=”1”> Couldnt find user </error> <error id="14"> Invalid List Name </error> <error id="12"> Incorrect parameters </error> <error id="2"> Couldnt find list </error>
Example Call	http://[IP]:[Port]/Server/resources/list/List_Create?ListName=NewList&UserId=3&ShopId
Example returned XML	<id> 36</id>

Name	List_Edit
Description	Edit the details of an existing list
Type of Request	POST
Parameters	ListName, UserId, ListId, ShopId
Result	Returns the list id if successful
Errors	<error id=”1”> Couldnt find user </error> <error id="7"> Couldnt edit: User is not the owner</error> <error id="12"> Incorrect parameters </error> <error id="2"> Couldnt find list </error>
Example Call	http://[IP]:[Port]/Server/resources/list/List_Edit?ListName=List&UserId=3&ListId=1&ShopId=3
Example returned XML	<id> 436</id>

Name	List_AddItems
Description	Add items to a list, where the items are defined by a “|” separated list.
Type of Request	POST
Parameters	ListId,UserId,ItemNameList, ItemCategoryList,ItemNote,ItemUncertain,ItemPrivate,ShopId,ItemQuantity
Result	Returns the item ids if successful
Errors	<error id=”1”> Couldnt find user </error> <error id="8"> Couldnt add: User has no permissions </error> <error id="12"> Incorrect parameters </error> <error id="2"> Couldnt find list </error>
Example Call	http://[IP]:[Port]/Server/resources/list/List_AddItems?ItemName=New&ItemCategory=Milk&UserId=3&ListId=1&ItemNote=1&ItemPrivate=0&ItemFavourite=0&ItemUncertain&0&ItemItemQuantity=1
Example returned XML	<id> 1357</id>

Name	List_AddExistingItems
Description	Add Existing items to a list, by value. The items are define by a “|” seperated list of ids
Type of Request	POST
Parameters	ListId,UserId, ItemIdList
Result	Returns the item ids if successful
Errors	<error id=”1”> Couldnt find user </error> <error id="8"> Couldnt add: User has no permissions </error>

	<error id="12"> Incorrect parameters </error> <error id="2"> Couldnt find list </error>
Example Call	http://[IP]:[Port]/Server/resources/list/List_AddExistingItems?ItemIdList=1,2,3&UserId=3
Example Returned XML	<id> 1|2|3</id>

Name	List_DeleteItems
Description	Remove items from a list, where the items are defined by a "|" separated list.
Type of Request	POST
Parameters	ListId,UserId,ItemIdList
Result	Returns the item ids if successful
Errors	<error id="2"> Couldnt find list </error> <error id="1"> Couldnt find user </error> <error id="9"> Couldnt delete: User has no permissions </error> <error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/list/ListDeleteItems?ItemIdList=1,2,3&UserId=3&ListId=1
Example returned XML	<id> 1|2|3</id>

Name	List_GetItems
Description	Returns the list of items, sorted by a criteria defined by OrderBy. Orderby can be "Name", "Category" or "Layout"
Type of Request	GET
Parameters	ListId,UserId,OrderBy
Result	Returns the item details if successful
Errors	<error id="1"> Couldnt find user </error> <error id="2"> Couldnt find list </error> <error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/list/List_GetItems?ListId=3&UserId=3&OrderBy=Layout
Example returned XML	<Items> <I id="13027" n="MooStars" q="1" c="" sh="" sid="-1" p="0" u="0" usr="harry" uid="1006" dt="17-10-2007"> _</I> <I id="25580" n="Remind me" q="1" c="Reminder" sh="" sid="0" p="0" u="0" usr="harry" uid="1006" dt="19-10-2007"> Update</I> </Items>

Shop-related Methods

Name	Shop_AddLayout
Description	Adds a user layout to the specified shop. Layouts are given in the format 1s2|3|4, 2s|5|6|7, where s represents the aisles and the item categories are represented by the "|" seperated list.

Type of Request	POST
Parameters	ShopId, UserId, Layout
Result	Returns the layout id or an error
Errors	<error id=”1”> Couldnt find user </error> <error id="4"> Couldnt find Shop </error> <error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/shop/Shop_AddLayout?ShopId=1&UserId=3&Layout=1s2|3|4, 2s|5|6|7
Example returned XML	<id> 327 </id>

Name	Shop_GetLayout
Description	Returns a shop layout that the user has added
Type of Request	GET
Parameters	ShopId, UserId
Result	Returns the layout
Errors	<error id=”1”> Couldnt find user </error> <error id="4"> Couldnt find Shop </error> <error id="12"> Incorrect parameters </error>
Example Call	http://[IP]:[Port]/Server/resources/shop/Shop_AddLayout?ShopId=1&UserId=3&Layout=1s2|3|4, 2s|5|6|7
Example returned XML	Layout sid="1" id="1" uid="3"> <aisle id="1"> <s>10</s><s>63</s><s>72</s><s>17</s> </aisle> <aisle id="2"><s>8</s><s>11</s><s>13</s> </aisle> </Layout>

4. Testing and Evaluation

This section will cover the testing and evaluation of the system, specifically the backend component. Detailed below is the test plan that will be used to evaluate how well the system meets its requirements. Testing will involve both white box testing, where the test cases are based on internal structure and seek to test the different paths in a program, and black box testing where input is given and the output tested for correctness.

4.1. Methodology

4.1.1. Objectives

The objective of this test plan is to provide a description of the tests that will be done to measure the systems compliance with the original project specifications. These specifications include fulfilling the features detailed in the design chapter, handling of multiple users at once as well as providing a stable and reliable system under extreme load.

4.1.2. Scope

The tests covers what was described above, with the focus being on testing the backend server functionality and performance. The other components of the system will be tested by their respective programmers.

4.1.3 Hardware and Software Requirements

The Server component used in the test was:

- Intel Pentium 4 3.0GHZ 512MB RAM

The Requester machine used in the test was:

- Intel Pentium 4 3.0GHZ 512MB RAM

These are the following software tools installed on the test machine

- Server Windows XP with SP2 and running an Apache Tomcat Server.
- Database: MySQL

4.1.4. Testing

The key areas of importance to the backend are tested. Database integrity was checked in terms of allowing multiple reading and writing at once. System testing determined that the system does implement the required features. Business Cycle testing verifying that operations related to the life cycle of the system is correct. Performance testing was done to test the response time of the system under a normal and abnormal load - important for communicating with the Web and mobile interfaces as if the response time is too high the requests will time out. Load testing was done to ensure that the system can

handle the theoretical traffic it would receive in production. Finally security and access control was tested to ensure correctness.

Data and Database Integrity Testing	Verify simultaneous record read accesses. Verify simultaneous record updates.
System Testing	Verify a new user can be created - System Use Case. Verify a user can login - System Use Case. Verify a user can create a list - System Use Case. Verify a user can add items to a list - System Use Case. Verify a user can checkout an item - System Use Case. Verify a user can delete an item - System Use Case. Verify a user can create a reminder as an item – System Use Case. Verify a user can delete a list – System Use Case. Verify a user can add a layout to a shop - System Use Case. Verify a user can sort a list by name - System Use Case. Verify a user can sort a list by layout - System Use Case. Verify a user can sort a list by category - System Use Case. Verify a user can logout the system - System Use Case.
Business Cycle Testing	Verify that reminders are shown within their notificaton period. Verify that updates to a list or shop are shown. Verify that reminder are deleted when they expire. Verify that users are logged out when the login period expires.

Performance/Load Testing	Verify response time for user creation. Verify response time for login by Web interface. Verify response time for login by Mobile interface. Verify response time to access items in a list. Verify response time for sorting a list by item name. Verify response time for sorting a list by shop layout. Verify response time for adding a reminders. Verify response time when 10 users try to access the same list at the same time. Verify system response when loaded with 200 logged on users. Verify system response when 10 simultaneous users try to sort the same list. Verify Memory and CPU usage is no more than 70% of the total resources available.
Security and Access Control Testing	Verify Logon security through user name and password mechanisms. Verify List Access control by trying to add and delete an item without permission.

Test Strategy

Data and Database Integrity Testing

Test Objective:	Ensure multiple objects can read the same record
Technique:	• Invoke multiple database access methods simultaneously using a test script
Completion Criteria:	All database access methods allow simultaneous access of records

Test Objective:	Ensure multiple objects can update the same record
Technique:	• Invoke multiple database update methods simultaneously using a test script
Completion Criteria:	All database update methods allow simultaneous updates of records

System Testing

Test Objective:	Ensure proper application navigation, data entry, processing, and retrieval.
Technique:	• Execute each system use case specified using the web interface • The expected results occur when valid data is used. • The appropriate error / warning messages are displayed when invalid data is used.
Completion Criteria:	• All planned tests have been executed. • All identified defects have been addressed.

Business Cycle Testing

Test Objective	Ensure proper application and background processes function according to required business models and schedules.
Technique:	• Reminders will be added so that their notification date and the notification period are such that they will show up. • Reminders will be added so that their notification date and the notification period are such that they will not show up. • Reminders will be added so that their notification date has expired • A users login time will be set so that they will be logged off.
Completion Criteria:	• All planned tests have been executed. • All identified defects have been addressed.

Performance/Load Testing

Test Objective:	Validate System Response time for designated transactions under a the following two conditions: - normal anticipated volume - anticipated worse case volume Validate Memory and CPU usage
Technique:	• Use JMeter test plan to simulate user loads and requests. • Increase the number of user making requests
Completion Criteria:	• Single Transaction / single user: Successful completion of the test scripts without any failures and within the expected / required time allocation (per transaction) • Multiple transactions / multiple users: Successful completion of the test scripts without any failures and within an acceptable time allocation of 20 seconds.

Security and Access Control Testing

Test Objective:	Function / Data Security: Verify that user can access only those functions / data for which their user type is provided permissions. System Security: Verify that only those users with access to the system are permitted to access it.
Technique:	• Create a test user account with limited access and attempt to perform functions which they have no rights to. This includes adding items to a list they don't have access to, deleting items they don't have access to, and deleting a list that they do not own. • Create a test user account with access to the same functions and attempt to perform them • Attempt to login with a test user with incorrect password
Completion Criteria:	For each known user type the appropriate function / data are available and all transactions function as expected

Tools

The following tools were for testing the system:

	Tool	Version
Functional Testing	Web interface	--
Performance Testing	JMeter	2.3
Test Coverage Monitor or Profiler	JMeter	2.3
DBMS tools	mySQL-Front	2.3

JMeter Test Plan

JMeter is a freely available desktop application that can simulate a heavy load on a Web server by making multiple requests. As JMeter is muli-threaded, allowing simultaneous requests of the same method can be made. The JMeter simulation used was derived from the above stated performance testing plan. A typical use case was constructed which involved: a user creating their account; logging in; creating a list; adding twenty items to that list; adding a shop layout; adding a reminder; sorting the list by name; sorting the list by category and sorting it by the shop layout just added. A thread count of 200 was used with a ramp up period of 10 seconds.

4.2. Findings

Database testing showed that mySQL could perform with good response times and allowed simultaneous access. This was to be expected as the queries performed by the system were only simple SELECT and UPDATE queries, not requiring transactional security.

System testing of the system occurred throughout the system's development due to the Feature driven model used. A Feature was thorougly tested before development began on another. During the final integration period most of the remaining errors were found and quickly fixed by using various use cases simulating the user's actions. Errors encountered usually involved a mismatch between the cached objects and the persistent store.

Business Cycle testing showed that reminders and notifications operate as expected, as well as showing that users would be logged off after 30 minutes.

The Performance testing was done using a JMeter test case. Two of the results are given below.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughp...	KB/sec	Avg. By...
Create User	121	471	0	3939	571.02	0.00%	1.0/sec	0.03	30.4
Login	119	323	0	4939	536.53	0.00%	1.0/sec	0.07	72.7
Create List	119	471	15	10676	1418.20	0.00%	1.0/sec	0.01	13.0
Add 20 item...	2251	290	0	7346	463.48	0.00%	19.0/sec	0.47	25.6
Add layout	108	274	15	1922	336.18	0.00%	1.0/sec	0.01	11.1
Add Reminder	107	300	31	4924	522.10	0.00%	59.9/min	0.01	14.0
Add item to li...	105	302	0	4611	539.58	0.00%	59.2/min	0.03	26.3
Checkout Item	104	1158	0	15256	2861.85	0.00%	59.7/min	0.03	26.5
Search by N...	104	416	0	4205	637.69	0.00%	1.0/sec	1.75	1755.9
Search by C...	103	478	0	3782	755.85	0.00%	1.0/sec	1.74	1741.8
Search by L...	103	76	0	938	180.85	0.00%	1.0/sec	0.04	36.8
TOTAL	3344	334	0	15256	756.68	0.00%	27.9/sec	3.63	133.3

Fig 11: JMeter Performance Test 1

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughp...	KB/sec	Avg. Bytes
Create U...	200	293	0	6821	546.96	11.50%	59.9/min	0.31	313.6
Login	200	163	0	2565	322.53	0.00%	1.0/sec	0.07	71.1
Create List	200	145	15	2597	275.37	0.00%	1.0/sec	0.02	15.6
Add 20 it...	4000	144	0	7712	334.29	0.00%	19.4/sec	0.50	26.2
Add layout	200	98	0	657	90.99	14.00%	1.0/sec	0.36	361.0
Add Rem...	200	147	31	1580	181.82	0.00%	1.0/sec	0.02	18.6
Add item ...	200	123	0	1251	157.78	0.00%	1.0/sec	0.03	26.5
Checkou...	200	175	0	1173	208.23	0.00%	1.0/sec	0.03	26.7
Search b...	200	122	0	2535	208.59	5.00%	1.0/sec	1.80	1823.5
Search b...	200	137	0	1721	234.10	5.50%	1.0/sec	1.79	1819.8
Search b...	200	19	0	641	74.92	3.50%	1.0/sec	0.12	127.0
TOTAL	6000	144	0	7712	314.62	1.32%	28.5/sec	4.76	170.9

Fig 12: JMeter Performance Test 2

The response times obtained in the results do not take into account network latency, as the machine making the requests was calling the backend directly. As can be seen in the first figure, the average response for a request is 334 milliseconds, well within acceptable limits. The 0 obtained for a minimum response time shows that the test machines were located near each other, not taking into account network latency. The highest maximum obtained in both results was 15256 milliseconds which was likely an outlier given the high standard deviation in that method, while still within the acceptable rate of 20 seconds. The deviation of results was due to the server load being increased as the test went on. Due to the setup of the test where the number threads were slowly ramped up most of the results for throughput were quite low, except for the adding 20 items loop, with a throughput of 19 requests per second.

As the number of threads reached 160, new database connections could not be made fast enough due to the default JVM heap size of 64MB being reached, resulting in the errors in the second figure. By increasing the heap size that tomcat is allowed, the threads it can support, and the number of database

connections performance of the system can be increased. This would result in much more memory being needed. The recommended amount of RAM that the backend would need is estimated to be around 1GB to support a load of 500 threads. The amount of memory required by the system can be reduced by having more functionality in the database layer as opposed to the logic layer.

5. Conclusions

The Cellphone Shopper system's aim outlined in the introduction was to provide a means to make shopping easier. To find what features the system would need to do this user studies were done in various iterations of the project. Once these features were detailed, a feature driven development model was used to ensure that each one was implemented successfully. The final architecture chosen for the implementation was a REST Web service that allowed the Web and cellphone interfaces to call methods on the server. The server used was Jakarta Tomcat with a mySQL database.

The system was implemented successfully with all the features specified in the design. The backend component fulfilled the initial requirements outlined in the project proposal except for performing statistical analysis. To recap, the requirements were that the backend should:

- Send and receive messages to/from both the mobile interface and the Web interface in XML.
- Retrieve information from the database on request from either interface.
- Translate data from the database to XML and send it to either interface.
- Perform statistical analysis on user data and present the results to either user interface.

As to why the statistical analysis was not done, this was due to time constraints and a difficulty in showing the results in the Web interface. In the user interviews done this was the feature that was least required.

In development of the system the following problems were encountered:

- While RESTful Web services are seen as simpler to implement than SOAP based Web Services, difficulty was had in finding any documentation on standards and sample code to implement it. The JAX-RS API that was used was just out of development and was poorly documented, meaning that trial and error was used to implement it.
- Integration of the components was the hardest part of the project. The sequence of methods called that the Web API assumed would be used was different to the sequence that the interfaces used. This resulted in some methods being merged as well as new ones being made. For instance the Web API assumed an item would be created first, then added to a list. This resulted in problems for the mobile interface as sometimes the request to create an item would fail. The failure rate of the mobile client in sending requests was much higher than expected, which led to methods being created to update the database in batches.
- The attributes that the interfaces needed to be returned were not defined specifically meaning that the API had to be frequently modified as the interface was developed.
- As REST was used there was no WSDL resource generated to show the methods and their parameters. When changes to existing methods were made or new ones added this had to be communicated to the other members of the team. A resource was created midway through the integration phase that listed the methods and their parameters which greatly helped the process.
- The choice of not using a persistence manager proved to hamper implementation, as various attributes were often added to classes frequently which had to be manually stored and loaded in the database. This led to unnecessary bugs being created, due to the mismatch between an objects data and the corresponding data in the database, which had to be accounted for.

- Jakarta Tomcat needed a number of tweaks in order to run a reasonable amount of concurrent threads. The default heap size proved insufficent, with an optimal size not being found.

In conclusion, the choice of the technologies for the system was largely correct. The Tomcat server and the MySQL database proved easy to integrate together, and the JAX-RS API was well suited for working with Tomcat. Both interfaces had no trouble in requesting resources as a normal URI was used. Initial problems were encountered when trying to send form encoded data to update the system but these were overcome. The message sizes of each request and response was greatly reduced compared to a similar SOAP message which was a big advantage. In future the implementation of RESTful Web services in Java will be easier due to the inclusion of JAX-RS in future Java EE and SE releases and better documentation being made available.

Communication between project members was vital in integrating the components, this done through a combination of face-to-face talks and online communication. Communicating online was extremely beneficial as most of the team members worked remotely, but more face-to-face discussions would have better helped clear misunderstandings.

In hindsight the following approaches should have been used in implementing the system:

- A persistence manager should have been used, specifically Hibernate should have been used instead of MySQL. Hibernate manages the use of persistence classes in Java while still allowing the use of normal SQL queries. This would have helped during development where rapid changes where made to classes. Enterprise Java Beans could have been used, but the overhead would have meant that the system requirements in terms of RAM needed would have been higher.
- A session token should have been generated and given to the interfaces upon login.This would be a required parameter for access to the API, making more secure.
- Sorting should be done in the database layer, as optimization can be done there.
- Certain objects that used more memory than most should have been lazily loaded, such as previous lists.

6. Future Work

If the system were to be developed further there are several feature that could be included.

- Statistical analysis of shopping history that was not implemented.
- Integration of the system with retail shops where items can be bought online.
- A shop route can be specified before hand, and the items sorted based on the route and not the layout.
- Admin system can be implemented where the administrator can add shops, cites, suburbs, item categories and default shop layouts.
- Allow the use of the cellphone camera to scan in barcodes of items: instead of having to enter the details in manually it can be automatically added to the system, or checked out by scanning the item.
- The choice of which shop to buy an item from can be recommended by the system, based on previous users or prices.
- Update the system to use Hibernate

7. References

1. Chavda, K. F. 2004. Anatomy of a Web service. J. Comput. Small Coll. 19, 3 (Jan. 2004), 124-134.
2. Wu, H. and Natchetoi, Y. 2007. Mobile shopping assistant: integration of mobile applications and Web services. In Proceedings of the 16th international Conference on World Wide Web (Banff, Alberta, Canada, May 08 - 12, 2007). WWW '07. ACM Press, New York, NY, 1259-1260.
3. Cugola, G. and Jacobsen, H. 2002. Using publish/subscribe middleware for mobile systems. SIGMOBILE Mob. Comput. Commun. Rev. 6, 4 (Oct. 2002), 25-33.
4. Lerner, R. M. 2001. At the Forge: Server-Side Java with Jakarta-Tomcat. Linux J. 2001, 84es (Apr. 2001), 10.
5. Juric, M. B., Kezmah, B., Hericko, M., Rozman, I., and Vezocnik, I. 2004. Java RMI, RMI tunneling and Web services comparison and performance analysis. SIGPLAN Not. 39, 5 (May. 2004), 58-65.
6. SOAP, REST and XML-RPC. http://www.kbcafe.com/rss/?guid=20060704042846
7. Allman, M. 2003. An evaluation of XML-RPC. SIGMETRICS Perform. Eval. Rev. 30, 4 (Mar. 2003), 2-11.
8. Phan, K. A., Tari, Z., and Bertok, P. 2006. A benchmark on soap's transport protocols performance for mobile applications. In Proceedings of the 2006 ACM Symposium on Applied Computing (Dijon, France, April 23 - 27, 2006). SAC '06. ACM Press, New York, NY, 1139-1144.
9. Hanslo, W. and MacGregor, K. 2004. The efficiency of XML as an intermediate data representation for wireless middleware communication. In Proceedings of the 2004 Annual Research Conference of the South African institute of Computer Scientists and information Technologists on IT Research in Developing Countries (Stellenbosch, Western Cape, South Africa, October 04 - 06, 2004). G. Marsden, P. Kotze, and A. Adesina-Ojo, Eds. ACM International Conference Proceeding Series, vol. 75. South African Institute for Computer Scientists and Information Technologists, 279-283.
10. XML-RPC Specification. Last visited July 2007. http://www.xmlrpc.com/spec
11. Google Maps. http://maps.google.com/
12. Google Mobile Maps. http://www.google.com/gmm/index.html
13. Google Calendar. http://www.google.com/googlecalendar/overview.html
14. CASE STUDY: Customers scan their palms before http://specials.ft.com/ln/ftsurveys/q4a2e.htm
15. Newcomb, E., Pashley, T., and Stasko, J. 2003. Mobile computing in the retail arena. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Ft. Lauderdale, Florida, USA, April 05 - 10, 2003). CHI '03. ACM Press, New York, NY, 337-344.
16. The U-Scan Shopper: System Components and Functionality. http://www.kleverkart.com/html/system.html.
17. VanLengen, C. A. and Haney, J. D. 2004. Creating Web services using ASP.NET. J. Comput. Small Coll. 20, 1 (Oct. 2004), 262-275.
18. Internet Information Services. http://www.microsoft.com/iis