Zen Cart Shopper

by
Michael McMahon

Computer Science Department
College of Engineering
California Polytechnic State University
2012

Date Submitted: March 7, 2012
Faculty Advisor: Franz Kurfess

Abstract

Zen Cart is a popular e-commerce tool for building a shopping cart web site. A typical Zen Cart store is not well suited for the small touch screens commonly found on today's smart phones. Zen Cart Shopper is a solution for Android systems which employs web scraping techniques to display a store's content in a smart phone friendly interface. This report describes the design and implementation of the Zen Cart Shopper app for Android.

Table of Contents

1. Introduction

Traditionally, a website is accessed on a desktop computer and its contents are displayed on a monitor and interacted with by clicking a mouse. This traditional model is now being challenged as mobile devices become a common medium for web access.[1] Web pages with interfaces designed for desktop interaction are often unsuitable for the small touch screen displays found on mobile devices. Zen Cart stores are an example of a web interface having diminished usability when accessed on a mobile device.

Zen Cart is a software tool which generates a shopping cart web page and allows users to set up an online store.[2] With over 2 million downloads[3], the Zen Cart platform has become a popular option for implementing a web shop. A Zen Cart store runs on a framework built on PHP and mysql; the store's content is displayed in HTML. Because the HTML generated by Zen Cart was intended to be viewed on a desktop display, its usability suffers when transferred to a mobile touch screen. On such a device, navigation requires a cumbersome amount of vertical and horizontal scrolling. Additionally, the mobile display must be frequently zoomed in and out to read small sized fonts or to accurately click on elements such as links or text fields.

The purpose of this project is to remedy these problems by creating a medium for interacting with a Zen Cart store that is tailored for usability on a mobile device. The solution that was implemented is an app for the Android platform called Zen Cart Shopper. Zen Cart Shopper uses the Android framework to place a touch screen friendly interface over the standard Zen Cart HTML. Underneath this interface, web scraping techniques enable the app to mirror the core functionality of an HTML Zen Cart store. This report summarizes the process of building the Zen Cart Shopper app.

2. Requirements Specification

This section lists the program requirements that were originally envisioned at the start of the project. The first subsection, Functional Requirements, is a detailed breakdown of the intended functionality for Zen Cart Shopper. This subsection describes when content is displayed, what content is displayed, and the actions available to the user at different contexts. The second subsection, Non-Functional Requirements, defines the expected behaviors of the app which don't rely on any specific functionality.

2.1 Functional Requirements

2.1.1 Store Selection

1. Users can choose to select a store which has been previously viewed, or choose to browse a new store.
2. A new store is selected by entering the URL of the store's main page.
    a. If the URL does not lead to a Zen Cart store, the user is informed that a bad URL was entered, the URL is not saved, and the user returns to the choice of actions available in requirement 1.1.

    b. If the URL does lead to a Zen Cart store, the URL is saved and the store can be selected again when the application is restarted.
3. Previously viewed stores can be accessed from a list.
    a. Stores are listed by their store name and URL.
    b. Stores can be removed from the list of previously viewed stores.

2.1.2 Store Accounts and Login

1. After choosing a store, a user may choose to log in or choose to create a new account for the store.
    a. The user may also choose to browse the store without logging in if the store allows it.
2. If a user chooses to log in, they must provide a username and password of an already existing account for the store they have selected.
    a. The last username entered will be filled in on future logins to the same store while the application is running and after being restarted.
    b. The user can choose to have the password saved and filled in for future logins to the same store while the application is running and after being restarted.
        i. The password will only be saved after a successful login.
    c. If the login fails, the user is informed that bad credentials were provided, the password is not saved and the user returns to the choice of actions available in requirement 2.1.
3. If a user chooses to create a new account, they will be presented with a form to input account information.
    a. The new account form will retrieve and display the input fields requested by the selected store.
    b. Required fields will be indicated by an asterisk.

2.1.3 Product Viewing

1. Users can choose to view products within a single category or choose to view products from all categories.
    a. Product categories are provided by the selected store, and are presented in a list after the user logs in.
    b. If a category contains sub-categories, a list of sub-categories is displayed after selecting a category.
    c. A user can also choose to view products from all categories or sub-categories.
2. Product information is displayed in a list, with each row listing details about a single product.
    a. A row lists a product's name and price.
3. A single product image is present above the textual list during product viewing.
    a. A user can scroll to another image to browse products by pictures.
    b. When a user taps on a product from the textual list, the product's row becomes highlighted and that product's image is scrolled to.
    c. When a user scrolls to a product's image, that product's textual list row becomes highlighted.
    d. When a user taps a product's image, that image is expanded to fill the screen.

4. From a categories, sub-categories, or product list a user can enter a search term by tapping the SEARCH button on their device.
    a. When the user enters a term to search for, a product list with product names or descriptions containing the search term is displayed.
5. From a product list, the user can choose to go back to view the list of categories or sub-categories from which the product list had been selected.
    a. If the product list is the result of entering a search term, the user goes back to the list of top-level categories.
6. From a sub-categories list, the user can choose to go back to viewing the categories list from which the sub-categories list had been selected.

2.1.4 Product Details and Options

1. When a user taps on a highlighted product's row from the textual product list, they are brought to a new screen where they can view the product's details, set product options, and add the product to their cart.
2. For product details, the page will show an image of the product, the product's name, the product's price, the number of units of that product in stock, and a detailed description of the product.
    a. Upon initially entering this screen, the product's detailed description is minimized under a header titled "Detailed Description". If the user taps the header, the detailed description expands underneath it.
    b. If the user taps the product's image, it will expand to fill the screen.
3. To access any product ordering options available, the user taps a button labeled "Select Product Options".
    a. For each option available on the product, a button is listed which is labeled with the option's name and the current choice for that option.
    b. If the user taps an option's button, then a new dialog overlays the screen and the user can select a new choice for that option.
4. To add the currently viewed product to their cart, the user taps a button labeled "Add to Cart"
    a. After the user taps the "Add to Cart" button, a window overlays the screen that allows them to set a quantity to add to cart and two buttons labeled "Ok" and "Cancel".
    b. The quantity is always set at first to 1, and can be increased to an amount equal to the quantity in stock.
    c. If the user taps the "Ok" button, the selected quantity is added to their cart and the view returns to the product details screen with a temporary notification that the product has been added to their cart.
    d. If the user taps the "Cancel" button, no items are added to the user's cart and the view returns to the product details screen.
    e. If product options are available, and have not been accessed, the quantity selection screen also includes a warning to the customer to inform them that options have not been set.
5. From the product details screen, the user can choose to go back to the list of items from which the product was selected.

2.1.5 Cart Viewing, Editing and Checking Out

1. From any screen after logging in, the user can choose to view their cart.
2. Viewing their cart allows a user to see a list of products they have added to their cart.
    a. Each product listed is in a single row of the list.
    b. A row displays the product's name, the quantity of the product in the cart, the price of ordering the given quantity of the product, and a button to remove the product from the cart.
    c. By taping on the quantity of the product, the user can modify the quantity of that product in their cart via an interface identical to the one used after the user selects "Add to Cart" from the product details screen
3. By tapping a row anywhere besides the quantity value, the user is taken to the product details page for the product's row they tapped on.
4. At the bottom of the screen is a button to check out the cart, labeled "Check Out".
    a. When the user taps the "Check Out" button, they are brought to a new screen that displays their shipping address and list of possible shipping methods.
        i. The user can choose to edit their shipping address from this screen.
    b. Tapping on a shipping method will bring the user to a new screen that displays a billing address, the total cost of the order with shipping and billing options.
        i. The user can choose to edit their billing address from this screen.
        ii. Supported billing options will include billing by check or money order, and PayPal.
    c. After selecting a billing option, a final screen is presented to confirm the order.
        i. The confirmation screen displays the selected shipping method and address, billing method and address, the products being ordered, and the total cost of the order.
        ii. The transaction is completed by pressing a button labeled "Confirm Order".

2.2 Non-Functional Requirements

2.2.1 OS Version

The app will run on devices using the Android operating system at versions 2.1 or greater. Targeting this OS version will allow the app to run on the majority of devices in use. This OS version will still provide the API functionality needed for the app.

2.2.2 Screen Size

The app will be usable on screens of all sizes but, it will be tailored to run on a "normal" sized screen of about 4 inches in width. Building for this screen size will ensure that the user interface is optimized for the majority of devices currently in use. For devices outside of this screen size, the Android OS will automatically scale the user interface to an appropriate size.

2.2.3 Portrait and Landscape Views

The app will run in portrait or landscape view but, interfaces will be tailored to be usable in portrait view. The benefits of building user interfaces tailored to landscape view are not substantial enough to justify the amount of development time required. So that the user does not feel forced to use portrait view, the app will still allow the OS to automatically adjust the user interfaces to a landscape view.

2.2.4 Battery Usage

To avoid excessive drain of a device’s battery, the app will be developed to be energy efficient. Efficiency will be achieved by using coding practices that avoid intensive use of the device’s processor. These practices include: Avoiding the creation of unnecessary objects to hold temporary values, avoiding the use of floating point variables, using static methods and variables when possible, using for-each loops when possible, and using direct field access when possible.

2.2.5 Network Usage

Network requests will drain a device’s battery and will also use up a potentially limited bandwidth quota. The app will avoid making unnecessary network requests by not repeating a failed request and retrieving data only at the user’s discretion.

2.2.6 Security

The app will secure the user’s sensitive information when possible. Passwords will be saved only at the user’s discretion and the data will only be available to the app. The app will make connections over HTTPS or SLL protocols when possible and alert the user when this is not possible.

3. Design

This section describes the major areas of focus in the design of Zen Cart Shopper. The first subsection covers the process of creating the User Interface (U.I). The second subsection discusses the underlying architecture of the system.

3.1 User Interface

One of the major objectives for Zen Cart Shopper is to offer a Zen Cart interface that is tailored for a mobile device. The user interface of Zen Cart Shopper is designed to require the user to navigate a minimal number of screens when shopping for products and placing an order. In the early stages of this project, a U.I. prototype was constructed with this design philosophy in mind. A usability testing of this initial prototype reveled that users could not quickly discern the purpose of each screen, and that some screens presented an interface that was too dense for individual elements to be easily identified. To improve usability, an overhaul of the user interface was conducted which focused on improving the intuitiveness of each screen and avoiding crowded interfaces which overwhelmed the user. The following subsections cover important stylistic techniques which were implemented in this overhaul.

3.1.1 Large Main Action Buttons

Each screen in the app is associated with some main action which advances the app to the next screen. This main action is carried out by pressing a large, vibrantly colored button appearing at the bottom of the screen. The style of the button was designed to draw the attention of the user and establish the button as the main action for a screen. Providing a visual cue for a screen's main action made it was easier for users to understand what they were supposed to do on each screen. By giving the main action buttons a consistent appearance and position, it was easy for users to recognize the button's purpose and to understand that pressing it would advance the app to a new screen.

3.1.2 Navigation Through a Pop-up Menu

Once a user is logged in to a store, they can access a menu by pressing the "MENU" Button which is commonly built into Android hardware. This action causes a panel of three buttons to appear on the bottom of the screen. The buttons are labeled "Logout", "Find Product", and "View Cart" with a descriptive icon appearing above the text of each button. These menu items allow for quick navigation through the app by bringing the user to one of the three major actions available. These actions are: selecting and logging in to a store, searching through a product inventory, or placing an order for a selection of products in a cart. Besides faster navigation, another benefit of the menu is that it alleviates crowded interfaces by allowing some buttons to reside in an off-screen pop-up panel (instead of taking up screen real estate). For instance, the "View Cart" and "Log Out" buttons had occupied the title bar on almost every screen in the original prototype.

3.1.3 Contextual Title Bar

At the top of each screen is a narrow, dark colored band containing the title of each screen in light colored text. The title text is intended to inform the user about the purpose and content of each screen. In some cases, the title bar was also used to display the context in which a screen appeared. For instance, when browsing through a tree of product categories the title bar might display the text "Television ► High Definition ► LCD Flat Screen" after the user had navigated to the LCD Flat Screen category. Providing an informative title bar assists users in the task of discerning where they are in the app and what the purpose of each screen is.

3.2 System Architecture

This section lays out the framework upon which Zen Cart Shopper is built. Subsections cover the process of content retrieval and how the major components in this process interact with one another.

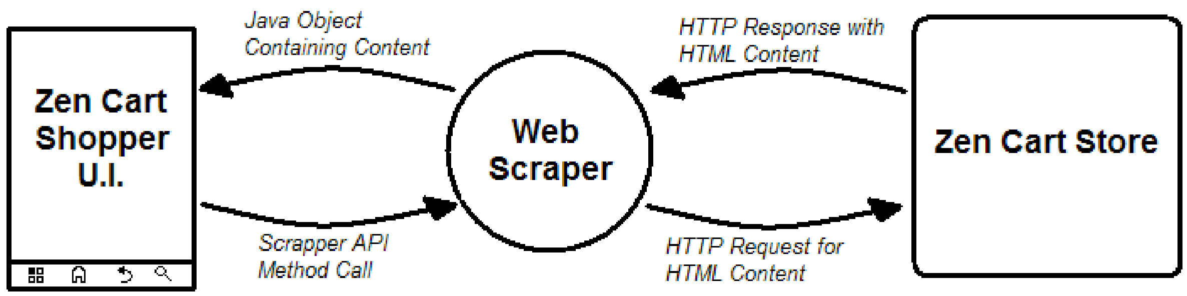
3.2.1 U.I-Scraper-Store cycle

The design of Zen Cart Shopper is built upon a system which can be broken down into three major functional components. These components are: 1) A Zen Cart store which renders content in HTML, 2) A web scraper which reads the HTML content, and 3) An interactive U.I. which displays the content. The primary goal of these components is to display a store's content in the interactive U.I. This goal is achieved through a cycle of communication between the three components, with the web scraper acting as the middle man between the U.I. and the store. The cycle works as follows:

1. The U.I. requests some content via the scraper's API.
2. The scraper then sends a request to the store via HTTP.

3. The store sends back a response to the request, also via HTTP.
4. The scraper extracts the requested content from the store's response and passes the content back to the U.I.
5. The U.I. obtains the requested content and updates its display.

This cycle is illustrated in the image below.


Zen Cart Shopper U.I.
Java Object Containing Content
Scrapper API Method Call
Web Scraper
HTTP Response with HTML Content
HTTP Request for HTML Content
Zen Cart Store

3.2.2 Data Structures

The U.I-Scraper-Store cycle described above calls for data structures which are designed to be exchanged during content retrieval. In some cases a data structure is passed from the U.I. to the scraper in order to specify what content to retrieve. In other cases the scraper creates a data structure from the content of a store and passes this structure back to the U.I. This duplex style of communication lead to the design of data structures which hold information needed by both the U.I. and the store. For instance, the Product class holds both a name that gets displayed in a product listing, and a form value to be passed to the server when the product is added to a cart.

3.2.3 HTTP Client and HTML Parser API's

Creating a framework for modular and maintainable code is the main goal for the design Zen Cart Shopper. One of the ways in which this is accomplished is by abstracting major areas of functionality into API's which act as an interface between functional components. The two most prominent API's in the design are for the HTTP client and HTML parser, these API's are contained in a classes named ZenScraper and ZenParser. ZenScraper is an interface between the app's U.I. and the store which the U.I. is displaying content from. This API contains methods intended to be called form the U.I. such as "getCart()" or "login()". ZenScraper is concerned only with retrieving a body of HTML from a store by sending the appropriate HTTP requests. Once HTML is retrieved, it is passed into a second layer of API contained in ZenParser. ZenParser contains logic for extracting content from HTML and constructing java objects which hold the content. This API contains methods which get called from ZenScraper such as "getStoreName()" or "getProductList()".

4. Implementation

This section covers major areas concerning the implementation of Zen Cart Shopper. Subsections cover details about how code was constructed for the U.I, HTTP client, and HTML parser.

4.1 User Interface

The U.I. of Zen Cart Shopper is divided up into a series of screens, with each screen providing an interface for accomplishing some specific task. Interfaces are constructed using a graphical layout editor that comes with the Android Development Tools[4] plugin for the Eclipse[5] IDE. The editor outputs a graphical interface as an XML document. The XML encoded interface can then be referenced and loaded by Java classes which implement the functional end of the U.I. The Java classes which handle U.I components are extensions of the Activity[6] class, which is a staple of the Android SDK library. Each Activity class in Zen Cart Shopper was correlated to the functionality of a single screen in the app, and each screen was correlated with some action to be taken by the user.

4.2 HTTP Client

The Zen Cart Shopper app interacts with a Zen Cart store through an HTTP session between the app and the store's server. Actions by the user are sent to the store through HTTP requests, and content is extracted from the store's responses. This interaction requires the app to be able to act as an HTTP client capable of making a network connection to an HTTP server.

4.2.1 Apache HttpComponents

Zen Cart Shopper uses the Apache HttpComponents[7] library to carry out the low level programming involved with communicating with a network server. A thread safe version of Apache's HttpClient class provided an API for sending requests and receiving responses. The Apache implementation was chosen for its stability and verbose documentation.
The HttpClient used in Zen Cart Shopper is configured upon instantiation to support HTTP and SSL connections. Through out its execution, the app maintains a single instance of this client in order to support operations which rely on sequential http transactions, such as viewing a cart after a particular account has been logged in to. When the user logs out of an account, the client's connection is destroyed.

4.2.2 Background Threads

The process of retrieving an html page from a network server and then parsing content from that page can take a substantial amount of time (~5-8 seconds). In order for the user interface to remain active during this time, it is necessary to place the work of network requests and subsequent HTML parsing in a thread of execution separate from the user interface.
Communication between the content retrieving threads and the U.I. is facilitated via a Handler[8] object attached to the U.I. The procedure of retrieving and displaying content typically works as follows:

1. A user action requires the U.I. to display some content.
2. The U.I. thread launches a new thread to retrieve the content.
3. The U.I. remains active while a content retrieving thread is doing the work of making an HTTP request and parsing the result.
4. When the content retrieving thread is done working, it sends the retrieved content inside of a Message to the U.I. thread's Handler.
5. The Handler receives the Message and updates the U.I to display the content.

4.2.3 Launching Threads Simultaneously v.s. Sequentially

In some cases obtaining the content for a single screen in Zen Cart Shopper requires multiple HTTP request-response transactions with the store's server. One example of such a case is when several HTML pages of product listings must be displayed inside a single scrolling list in the app. The app has the choice of sending a request for all pages simultaneously or sending the set of requests one by one in a sequential order. While sending a volley of requests simultaneously is faster overall, a sequential request strategy is implemented instead. Sending a series of requests for content is preferable because the user can start initially start viewing a small portion of the content while the remaining portions are being retrieved. With this strategy, the user does not have to wait for the retrieval of four pages of products in order to view a product on the first page.

4.3 HTML Parsing

In order to extract content from a body of html, Zen Cart Shopper must utilize some method for distinguishing the content it wants from the rest of the HTML. One solution is to rely on a web ontology, such as Good Relations[9], which categorizes HTML content by attaching a defined set of tags to appropriate sections of the source. The feasibility of relying on a web ontology was ruled out because such a schema was not implemented by the standard Zen Cart store. The alternative method chosen was to identify patterns in HTML which would consistently surround the content desired by the app. The source of numerous stores was studied in order to identify such patterns and verify they existed across a variety of store configurations. Once these patterns had been identified, the app could employ pattern matching to parse content from a store's HTML source.

4.3.1 Parsing with jsoup

The process of HTML parsing was greatly simplified by the use of the jsoup[10] library. The jsoup library provides an API which allowed for parsing HTML source into a collection of elements represented as a Document[11] object. The Document API allows for elements contained within to be selected by attributes such as their tag, class, or id. Selected elements returned by a Document are contained in an Element[12] object. These tree-based structures can contain child Elements, and an Element's children can be searched for using an API similar to the Document object's.

The jsoup library was chosen over other HTML parsers because of it's stability and robust documentation. Using jsoup allows Zen Cart Shopper's parsing methods to be written in a concise format which is easy to modify and maintain. Additionaly, developing with the library side-steps having to write error-prone algorithms involved with pattern matching and parsing. The jsoup library sped up development by providing a convenient and bug-free API.

The benefits brought on by jsoup did not come without a cost. One of the costs involved is a lager app size with the jsoup library adding an additional 275 kB. But the most significant drawback is a slower execution time, as jsoup's HTML parsing can consume anywhere from 3-5 seconds depending on the size of the page being parsed. The construction of jsoup's Document tree relies on an algorithm that is more complex than what is actually needed to parse the content used in Zen Cart Shopper. A better alternative to using jsoup would have been to write custom HTML scraping code which only parsed the sections of HTML that were relevant to the app. Although this alternative would have produced faster code, it is important to consider that implementing the parser in this way would have taken much longer to write and debug. This would have resulted less time spent on implementing other aspects of Zen Cart Shopper. Using the jsoup library allowed for more functionality to be written into the app within the relatively short amount of time allocated to this initial development cycle.

4.3.2 Parsing Customized Stores

The Zen Cart framework allows for a high amount of customization on top of the basic store set up. This meant that the parsing methods could not always expect a consistent structure to the HTML source of a store. Since the parser could not possibly support all current and future store customizations, it was written to fail gracefully in cases where content was displayed in an unknown format. This was handled by having the parser detect when it was reading an unknown format and simply return an empty string in place of the requested content.
This solution was reasonable since the great majority of stores use a default HTML structure which the parser is capable of reading. Most of the customizations which the parser did not understand were so uncommon that their lack of support would only affect a small number of users. In a few cases, several variations were deemed to be common enough that the parser's lack of support for them would affect a significant amount of users. These cases are discussed in the following subsections.

4.3.2.1 Parsing Currency Symbols

The parsing algorithm searches for a "$" symbol when parsing a product's price. This method is incompatible with a large number of Zen Cart stores that display prices in various non-dollar currencies. One solution to supporting varying currencies would be to search for prices with each possible currency symbol until a match was found. This solution was ruled out in this development cycle because an implementation capable of parsing all possible currencies would require the construction of a complex algorithm that would hurt execution time and detract from development time. The solution which was implemented is to always request that a store display prices in US dollars. Using this method, the parsing algorithm can read stores which display prices in any currency by simply parsing for one currency symbol. The decision to only support the dollar is discussed more in section 5.2.4 of this document.

4.3.2.2 Parsing Product Listing Grids

The default method of displaying a list of products is to use a table with each row holding a separate product entry. While most stores use this default format, a significant number are configured to display product entries in a grid format, with each entry residing in a block of the grid. A single parsing routine can not be used to read these two formats because each uses a different structure of html. To accommodate this incompatibility, the parser implements two separate parsing routines, each capable of reading one of the two popular formats. To determine which routine to use, the parser scans the html source of a product listing for a class value which indicates if the listing is using a grid display. If the class value is matched, the grid parsing routine is used, otherwise the default table parsing routine is used.

5. Results

This section will cover the results of the Zen Cart Shopper project. The first subsection describes the work that was completed. The second section discusses possible work to be done in future development.

5.1 Implemented Features

The values in the "Requirement" column in the table below refer to the requirements listed in section 2 of this document. The values in the "Completion Status" column indicate if the requirement was implemented; A value of "Partial" means some aspects of the requirement were implemented while others were not.

Requirement	Completion Status	Notes
2.1.1.1	Complete	
2.1.1.2	Complete	
2.1.1.2.a	Complete	
2.1.1.2.b	Complete	
2.1.1.3	Complete	
2.1.1.3.a	Complete	
2.1.1.3.b	Complete	
2.1.2.1	Partial	Account creation not implemented.
2.1.2.1.a	Not Implemented	
2.1.2.2	Complete	
2.1.2.2.a	Not Implemented	
2.1.2.2.b	Not Implemented	
2.1.2.2.c	Complete	
2.1.2.3	Not Implemented	
2.1.3.1	Complete	
2.1.3.1.a	Complete	
2.1.3.1.b	Complete	
2.1.3.1.c	Complete	

Requirement	Completion Status	Notes
2.1.3.2	Complete	
2.1.3.2.a	Complete	
2.1.3.3	Complete	
2.1.3.3.a	Complete	
2.1.3.3.b	Complete	
2.1.3.3.c	Complete	
2.1.3.3.d	Not Implemented	
2.1.3.4	Partial	Search interface is presented above category list, not via SEARCH button.
2.1.3.4.a	Complete	
2.1.3.5	Complete	
2.1.3.5.a	Complete	
2.1.3.6	Complete	
2.1.4.1	Partial	Product options not implemented.
2.1.4.2	Partial	Details screen does not display units in stock.
2.1.4.2.a	Not Implemented	Description appears in scrolling panel.
2.1.4.2.b	Not Implemented	
2.1.4.3	Not Implemented	
2.1.4.3.a	Not Implemented	
2.1.4.3.b	Not Implemented	
2.1.4.4	Complete	
2.1.4.4.a	Complete	
2.1.4.4.b	Complete	
2.1.4.4.c	Complete	
2.1.4.4.d	Complete	

Requirement	Completion Status	Notes
2.1.4.4.e	Not Implemented	
2.1.4.5	Complete	
2.1.5.1	Complete	
2.1.5.2	Complete	
2.1.5.2.a	Complete	
2.1.5.2.b	Complete	
2.1.5.2.c	Not Implemented	
2.1.5.3	Complete	
2.1.5.4	Complete	
2.1.5.4.a	Complete	
2.1.5.4.a.i	Not Implemented	
2.1.5.4.b	Complete	
2.1.5.4.b.i	Not Implemented	
2.1.5.4.b.ii	Complete	
2.1.5.4.c	Complete	
2.1.5.4.c.i	Complete	
2.1.5.5.c.ii	Complete	
2.2.1	Complete	
2.2.2	Complete	
2.2.3	Partial	Display is locked to portrait view.
2.2.4	Complete	
2.2.5	Complete	
2.2.6	Complete	

5.2 Future Work

This section discusses important features and improvements to the app which were not implemented in this development cycle. Each sub-section focuses on a specific improvement and explains why it would be good candidate for future work, and why it was not implemented.

5.2.1 Diverse Store Customization

As more customization templates are released for Zen Cart, a growing amount of work remains in writing parsing algorithms which can read the HTML structures of these custom templates. Of course, a more desirable solution would be a single parsing algorithm which identifies semantic tags that are universal across all templates. This solution is not possible without Zen Cart enforcing a web ontology. The various templates which were not supported were uncommon in use and so their compatibility was not deemed a high priority in this round of development.

5.2.2 Product Options

The ability to select varying options on a product when ordering is one feature that did not get implemented in this cycle of development. This feature would allowed for more ordering options, for example giving the user a choice between different sizes and colors of a shirt. Although an interface was created for this feature, not enough development time was allocated to write the parsing algorithms needed to obtain the various types of options which could exist. While many stores used product options when ordering, the majority of products did not have any options to be configured. For this reason, the feature was not deemed to be critical to the app's functionality and was not implemented in this development cycle.

5.2.3 Landscape Orientation Layouts

Android supports two different layouts for viewing content based on screen orientation. These layouts are the tall and narrow portrait mode, and short and wide landscape mode. The height and width of the screen can affect how content is displayed and some users will prefer one orientation over the other. Zen Cart Shopper locks the orientation to portrait view and so an improvement can be made by allowing landscape layouts. During prototyping, it was determined that the content displayed in the app was more suited for a portrait view, and so interfaces were designed for that orientation.

5.2.4 Localization

Additional work remains to be done in localizing Zen Cart Shopper. Zen Cart stores are used internationally, and so a use for this app exists outside of the English speaking American users it has been built for. A basic improvement could be made by supporting currencies besides the US dollar. Complete localization could be accomplished by translating the app's interface into other languages. The initial release of Zen Cart Shopper was mainly available to users in the US and so localization was not a top priority and did not get implemented.

6. Conclusion

Developing Zen Cart Shopper was an education experience for this student. The project served as an exploration in the design and implementation of a full-fledged Android app. Areas of

Android development which were studied include structuring a relationship between U.I. and data processing components, developing threaded processes, and designing an atheistic U.I. Another major area of study focused on creating a web scraper. The implementation of this component called for research on HTTP transactions as well as the structure of HTML.
One important lesson learned was that a typical web page is not intended to be accessed like a program accesses an API; A web page it is meant to be operated by a human. In the same vein, HTML is only intended to define a layout and appearance for content, defining what the actual meaning of the content is another job intended to be done by a human. Web scraping is an attempt to have the meaning of the content understood by a computer. Unless HTML includes machine-readable semantics, understanding its content is not a straight forward task for a computer program. Had Zen Cart implemented some type of web semantics, it's content could be extracted by simpler, more reliable algorithms than what was implemented in Zen Cart Shopper. Working on this project was an exploration in the challenges of building a web scraper and a thought exercise on possible solutions for overcoming those challenges.
The goal of this project was to create a functional Zen Cart interface designed for usability on a mobile touch screen. The product delivered at the end of this development cycle utilized design techniques which resulted in an intuitive and easy to use U.I. The product was also able to mirror the critical functionality of an HTML Zen Cart store through a web scraping API. The solution implemented by Zen Cart Shopper fulfilled the project's goal and so the project can be considered a success.

7. References

1. MmobiThinking, *Global mobile statistics 2012: all quality mobile marketing research, mobile web stats, subscribers, ad revenue, usage, trends…*. N.p., 2012. Web. 1 Mar 2012. <http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats#mobilepageviews>
2. *Zen cart*. N.p., 2003. Web. 1 Mar 2012. <http://www.zen-cart.com/>.
3. "Download Statistics: All Files." *SourceForge*. N.p., 1 Mar 2012. Web. 1 Mar 2012. <http://sourceforge.net/projects/zencart/files/stats/timeline?dates=2003-06-19 to 2012-03-01>.
4. Android Developers, *Adt plugin for eclipse*. N.p., 2012. Web. 5 Mar 2012. <http://developer.android.com/sdk/eclipse-adt.html>.
5. "Overview." *Eclipse - The Eclipse Foundation*. The Eclipse Foundation, 2011. Web. 5 Mar 2012. <http://www.eclipse.org/>.
6. "Activity | Android Developers." *Android Developers - Reference*. Google, 05 Mar 2012. Web. 5 Mar 2012. <http://developer.android.com/reference/android/app/Activity.html>.
7. "Apache HttpComponents." *The Apache Software Foundation*. N.p., 2012. Web. 6 Mar 2012. <http://hc.apache.org/>.
8. "Handler | Android Developers." *Android Developers - Reference*. Google, 05 Mar 2012. Web. 5 Mar 2012. <http://developer.android.com/reference/android/os/Handler.html/>.
9. "GoodRelations: The Professional Web Vocabulary for E-Commerce." *GoodRelations*. N.p., n.d. Web. 5 Mar 2012. <http://www.heppnetz.de/projects/goodrelations/>.
10. Hedley, Jonathan. "jsoup Java HTML Parser, with best of DOM, CSS, and jquery." *jsoup*. N.p., n.d. Web. 5 Mar 2012. <http://jsoup.org/>.

11. Hedley, Jonathan. “Document." *jsoup*. N.p., n.d. Web. 6 Mar 2012. <http://jsoup.org/apidocs/org/jsoup/nodes/Document.html>.
12. Hedley, Jonathan. "Element." *jsoup*. N.p., n.d. Web. 6 Mar 2012. <http://jsoup.org/apidocs/org/jsoup/nodes/Element.html>.