

# Endicia Webmailer Application

---

**By**  
Megan Tsai  
Gage Fleischer  
Migdoel Alvarado

**Sponsor: DYMO Endicia**  
**Advisor: David Finkel**

**3/4/2011**



## **Acknowledgements**

We would like to thank the following people for their support throughout this project:

Amine Khechfe, Endicia.

Patrick Farry, Endicia.

Emil Redzic, Endicia.

Julian Thomas, Endicia.

Jason Davey, Endicia.

Steven Tsao, Endicia.

Bhaumin Shah, Endicia.

Charles Morelle, Endicia.

And all the employees at Endicia.

Professor David Finkel, WPI.

## **Abstract**

This project collaborated with Endicia in migrating the existing DYMO Stamps application to the web. We created a set of web pages and a print engine to print stamps from a browser without endangering the securities necessary in the stamps creation process. We explored new technologies that would be beneficial to the company for future works. This project was handed off for further development and production.

Acknowledgements.....	2
Executive Summary.....	6
Chapter 1 Introduction.....	7
Chapter 2 Background.....	8
2.1 Endicia.....	8
2.2 Competitors.....	8
2.3. Current Applications.....	9
2.3.1 DYMO Stamps.....	9
2.3.2 Web Application.....	10
2.3.2.1 MonTimbrenLigne.....	10
2.3.2.2 USPS Click-N-Ship.....	10
2.4 User Base Research.....	11
2.4.1 Client Side Requirement Research.....	11
2.4.1.1 Browser Type Usage.....	11
2.4.1.2 Browser Height Usage.....	13
2.4.1.3 Browser Width Usage.....	14
2.4.2 Print Engine Requirement Research.....	15
2.4.2.1 Operating Systems Usage.....	15
2.4.2.2 Java Support Usage.....	16
Chapter 3 Requirements.....	18
Chapter 4 Design.....	19
4.1 Client Side.....	19
4.1.1 Environment.....	19
4.1.2 Design.....	20
4.1.2.1 Design Process in Endicia.....	20
4.1.2.2 Design Approach.....	20
4.2 Print Engine.....	21
4.2.1 Environment.....	21
4.2.2 Design.....	21
Chapter 5 Implementation.....	23
5.1 Client Side.....	23
5.1.1 Iteration I: Development from Wireframe.....	23
5.1.1.1 Development Process.....	23
5.1.1.2 Iteration I Testing.....	26
5.1.1.3 Iteration I Development Issues.....	26
5.1.1.4 Iteration I Result.....	27

5.1.2 Iteration II: Development from Visual.....	28
5.1.2.1 Iteration II Development Process.....	28
5.1.2.2 Iteration II Testing .....	33
5.1.2.3 Iteration II Development Issues .....	33
5.1.2.4 Iteration II Result .....	36
5.1.3 Iteration III Feedback and Enhancement .....	37
5.1.4 Integration.....	39
5.1.4.1 Integration with Endicia’s Server .....	39
5.1.4.2 Print Engine Integration.....	39
5.1.4.3 Integration Problems.....	40
5.1.5 Security .....	40
5.2 Print engine.....	41
5.2.1 Proof of Concept.....	41
5.2.2 Applet Migration and Embedding.....	41
5.2.3 Code Refactoring .....	41
5.2.4 Custom Print Dialog.....	42
5.2.5 Printing and Roll Selection .....	42
5.2.6 Printer and Job Status Querying.....	43
5.2.7 Stamp Image Retrieval and Session Sharing .....	43
5.2.8 Status Feedback .....	43
5.2.9 Final Refactoring.....	43
5.2.10 Logging.....	44
Chapter 6 Result.....	45
Chapter 7 Conclusion.....	46
References:.....	47
APPENDIX A.....	49
APPENDIX B .....	50
B-Sheet Printing.....	50
Add Stamps Pop Up.....	56
Buy Postage .....	58
Prepare for Printing.....	59
Roll Printing.....	60
Prepare for Printing Roll.....	63
Guide Me .....	64
APPENDIX C .....	66
Sheet Printing.....	66

## **Executive Summary**

Today's technologies make the passing of information between people easier and faster. Even with these technologies, the mailing of important documents and materials has not gone out of style. So when physical items have to travel from one person to another, new technologies are developed to make mailing and shipping an easier task. Endicia is a company founded in 1982 as a consulting company. Today, the company is based in Palo Alto California and creates software solutions to help take the hassle out of shipping for businesses and individuals. One of Endicia's software products is a client software called DYMO Stamps which allows users to order and print stamps from a downloaded application.

Our role in this project was to migrate the DYMO Stamps application to the web so that it is accessible to more users. A set of web pages were created with all the functionality of the application as well as new features that make the web application easier to use. We also designed and implemented the print engine that allows the application to print the stamps without compromising security necessary to prevent forgeries. A working prototype was created from this project that allows a user to navigate through the application and print stub information on the roll or sheet depending on the printer they have.

We encountered several technical challenges during this project. The web site design had some browser compatibilities issues that were surprising. Since we want to prevent forgeries, disabling the normal print dialog provided through a browser and only allowing user to print with custom constraints was necessary for the security of the application. A large part of the project was to coordinate the user interface, the Endicia's server and the print engine into a working application. The project left Endicia with a working set of web pages and a print engine that have the main architectures set up for more detailed refinement.

This project was in part a research and development effort. Therefore, we employed new technologies that were not used by Endicia and was of interest for future Endicia projects. For this reason, we were able to create and present a tutorial on CSS Grid and JQuery to the Endicia web team. This tutorial will help them understand and connect the web application we created for this project with any future addition to the website.

This result of this project is scheduled for a production release in a couple of months from the end date of the project. Therefore, the result of our project was not just a prototype and a proof of concept; it will be improved and integrated into Endicia's website. The web pages we created were designed so they can be easily connected to the server. During our final phase of integration we removed local references of data and replaced them with calls to the server. These calls to the server will later be fully developed by Endicia and will give our application the information it needs dynamically to function properly.

## **Chapter 1 Introduction**

Today's technologies make the passing of information between people easier and faster. Even with these technologies, the mailing of important documents and materials has not gone out of style. So when physical items have to travel from one person to another, new technologies are developed to make mailing and shipping an easier task. Endicia is a company founded in 1982 as a consulting company. Today, the company is based in Palo Alto California and creates software solutions to help take the hassle out of shipping for businesses and individuals. Their main software product is DAZzle and through DAZzle, users have access to all the provided services from Endicia. For users who do not want to pay for the complete coverage, client software called DYMO Stamps is also available with less functionality compared to DAZzle.

Our role in this project was to migrate the DYMO Stamps application to the web so that it is accessible to more users. A set of web pages were created with all the functionality of the application as well as new features that make the web application easier to use. We also designed and implemented the print engine that allows the application to print the stamps without compromising security necessary to prevent forgeries. A working prototype was created from this project that allows a user to navigate through the application and print stub information on the roll and sheet depending on the printer they have.

## **Chapter 2 Background**

### **2.1 Endicia**

Endicia software allows users to fill out and print complete prepaid postage for any single piece of domestic mail class offered by the US Postal Service, whether it's First-Class Mail, Priority Mail, Express Mail, Media Mail, Library Mail, Bound-Printed Matter, or Parcel Post [8]. Endicia supports printing of these types of postage on DYMO Thermal LabelWriter printers. These printers do not use ink or toner; instead they use direct thermal printing to print on the shipping label sheets provided by Endicia. Endicia also supports printing using normal inkjet or laser printers on stamp sheets provided by Endicia. Endicia offers various subscriptions providing different levels of services based on company needs. The base system provides address verification via the USPS system and auto-correction of addresses to USPS format with complete ZIP codes [5]. Additionally, postage is automatically calculated by Endicia's server based on mail class, weight of package, and destination. The postage can then be printed with a specific barcode containing postage information readable by USPS [9]. For extra services such as certified mail, COD, and Restricted Delivery, the system can calculate the extra fees necessary and adds them to the total postage [14].

For international shipping, Endicia offers an integrated form to eliminate some of the steps needed to fill out a custom form. Along with an International Mail Advisor, the user of the system can be informed about shipping restrictions for the destination country, rates for shipping the item, and the different mailing classes that exist in the destination country [12].

Shipping labels can be customized with a business logo and specific layout for a personalized look [13]. The shipping labels can be adjusted to hide the postage cost from the customer to avoid conflicts [17]. Additionally, Endicia offers their users a way to print or email return shipping labels to their customers [15]. Business reply mail is also customizable with Endicia software. This feature is especially useful for companies that rely on business reply mail and can use the extra personalization provided by the software to help the company be more memorable to the customers [7]. To simplify the package tracking process, Endicia offers customizable email notification upon shipment of the package [10]. Endicia also offers a log on the user's own computer as well as access to a web report on Endicia's website to help the user keep track of all shipments [6, 16].

Endicia does not only offer an installed software product for their customers, they also offer a variety of web services for software integration. Customers that need address verification and zip code correction can use the Dial-A-ZIP API to check the address and zip code entered right in their application. The desktop software DAZzle is not just standalone software; Endicia offers an XML interface for the customer to integrate DAZzle into their own software solutions. For customers without access to the Endicia software, the integration of the Endicia label server into an application allows anyone to print the shipping label straight from their own computer without installing Endicia software [11].

### **2.2 Competitors**

Only a few other companies offer similar Internet-based postage services. One of Endicia's competitors is Stamps.com. Stamps.com offers a variety of subscription based plans, each one targeting different types of users, from the home office user to the enterprise user. Users can purchase supplies from their website; however their services do not require them to do so. [37]

Along with printing custom and standard stamps, shipping labels, and envelopes, Stamps.com's software allows users to print automated Post Office forms. The software is also able to work with scales in order to determine the correct amount of postage needed. To minimize returned mail, address verification services are included that check against the USPS address database. They also offer package tracking services for both the shipper and the receiver. Web-based reports that summarize postage spending are also available through their website.



Stamps.com also provides data integration software that allows communication with other third party software and services. With their integration software, users are able to import order information from popular e-commerce websites such as eBay, Amazon.com, PayPal, Yahoo!, and Google Checkout to quickly print out the necessary postage and labels for transactions made through those websites. Their integration software can also import addresses from other popular software such as Outlook and QuickBooks.

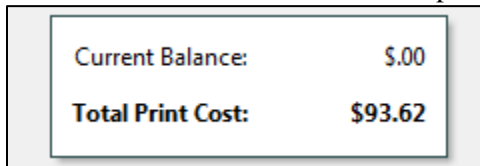
Comparable to PictureItPostage by Endicia, Stamps.com offers NetStamps, a web based solution for printing out stamps. This service requires users to purchase special stamp sheets. NetStamps allows users to print exact postage. However, NetStamps is only limited to printing out stamps and does not print out shipping labels or envelopes [38].

## 2.3. Current Applications

### 2.3.1 DYMO Stamps

Endicia currently offers a downloadable piece of software called DYMO Stamps for handling some basic postage creation, customization, and printing options. This piece of software is installed on a user's machine and communicates with Endicia's systems to provide some extra services as well as money transfers [4].

DYMO Stamps application helps the user calculate the postage by selecting a mail piece type of postcard, letter envelope, large envelope, package, flat rate envelope or flat rate box. Then the user can select a weight class for either domestic or international first class up to 13 oz. in weight. For flat rate items, the user can select between express mail and priority mail.



Current Balance:	\$0.00
<b>Total Print Cost:</b>	<b>\$93.62</b>

Figure 1 The total changes as new stamps are added

As the user select between different options in the application, the postage total is displayed and constantly updated to show the combined cost of all the postage that will be printed and charged to the user's account (Figure 1). There is the option of printing the postage selections or test print to make sure the postage will print correctly and cleanly.

The system is designed to print on USPS approved stamp sheets. Any inkjet or laser printer with stamp sheets inserted can print out stamps. The software also supports certain DYMO LabelWriter thermal paper rolls. These blank sheets or thermal rolls are purchased from Endicia's website or at certain local stores. If printing on sheets, one can choose how many sheets of stamps to print. A maximum of 10 sheets of 24 stamps can be printed in a single order. If printing from a DYMO thermal printer, the specific numbers of stamps are printed with each postage type. Unlike the thermal printer option, one can select a specific spot on the stamp sheet with a different postage amount and effortlessly print 24 different stamps on the same sheet (Figure 2).



Figure 2 Sheet printing preview in DYMO Stamps

Apart from selecting and printing postage, there are several buttons that link the user to the Endicia site. These buttons allow one to manage his or her Endicia account, buy postage sheets or rolls, add money to the postage account, and link to a help site for using the software. The software also contains a link to PictureItPostage.com for creating postage with picture customization. PictureItPostage ships the customized standard stamps to the customer and is not integrated into the software.

## 2.3.2 Web Application

### 2.3.2.1 MonTimbrenLigne

Endicia worked hand in hand with MonTimbrenLigne, a France based personal postage provider like Endicia, to create an entirely web-based system for customizing and printing stamps. This system does not support shipping labels and only offers services dealing with mail shipments from France. The application keeps an address book of all of the addresses a user has used in the creation of their stamps. The address book is also expandable via a CSV/XLS file. The application also remembers stamps users have made before so they do not have to create the commonly used postage every time they visit the site [30].

The following is an overview of the use of the application:

1. Choose from a pre-defined list of pictures to be displayed on the stamp
2. Enter location, destination, weight of package, and the print paper format (stamp paper, envelope, or DYMO label paper).
- 3-1. If the stamp paper option is chosen, users can customize the printing layout such as where on a page the stamp will be printed. Users can print multiple stamp styles (picture and rate) at the same time; the system automatically adds the new stamps to the end of previous ones.
- 3-2. For the envelope printing option, stamps can be printed directly onto an envelope along with return and destination addresses.
- 3-3. Since DYMO printer formats only allow one stamp to be printed at a time; there is no specific option for this printing method.
4. The customer can review the payment total as well as the payment method. This step requires the user to be logged in to continue.
5. Samples of the ordered stamps can be printed to ensure the customer will be satisfied with the final product and to make sure the information is correct. Once the customer hits print, the payment is charged to their account and the stamps will be printed and ready for use.

### 2.3.2.2 USPS Click-N-Ship

Click-N-Ship, the USPS web based solution, is a one stop shop for printing labels and postage developed to work with Endicia's system. Unlike the full services provided by Endicia and Stamps.com,

Click-N-Ship does not require a subscription for their services; users only need a free account to use the services and software of Click-N-Ship. Click-N-Ship seems to be geared more toward the home office and small business user since the features available are very limited. Click-N-Ship allows users to input a return and delivery address which can be saved in their Address Book for future use. Shipping costs are calculated based on the user supplied package information such as size, weight, and service options. Users can also start a batch order that allows the creation of multiple labels with the same package, weight, service options, and return address. Since this is a very simple service, no reporting or integration features are available [39].

## 2.4 User Base Research

Endicia keeps a data base of certain usage data on the customers to help the developers tailor the application to the user base. The following information was generated in January of 2011 as the most recent data on the visitors to Endicia’s website.

### 2.4.1 Client Side Requirement Research

#### 2.4.1.1 Browser Type Usage

Browser compatibility was important for this project to ensure that all users have the same experience on the web application despite the use of a different browser. Endicia keep tracked of the browser types of current Endicia users. Figure 3 contained the top 20 browsers and version used by Endicia users.

Browsers		Visitors
1.	Microsoft Internet Explorer 8	300,324 38.2%
2.	Mozilla Firefox 3.6	194,491 24.7%
3.	Microsoft Internet Explorer 7.0	86,635 11.0%
4.	Google Chrome 8.0	70,890 9.0%
5.	Safari (unknown version)	51,889 6.6%
6.	Mozilla Firefox 3.5	15,456 2.0%
7.	Microsoft Internet Explorer 6.0	14,886 1.9%
8.	Mozilla Firefox 3.0	8,603 1.1%
9.	Safari 5.0.2	5,656 0.7%
10.	Microsoft Internet Explorer 9	5,185 0.7%
11.	Google Chrome (unknown version)	4,899 0.6%
12.	Microsoft Internet Explorer 7.0 (AOL)	3,701 0.5%
13.	Mozilla Firefox (unknown version)	2,996 0.4%
14.	Opera 9.x (unknown version)	2,671 0.3%
15.	Microsoft MSN Explorer	2,603 0.3%
16.	Microsoft MSN Explorer 9.0	2,240 0.3%
17.	Safari 4.0.4	1,472 0.2%
18.	Safari 5.0.1	1,288 0.2%
19.	Safari 5.0	1,232 0.2%
20.	Safari 4.0.5	1,098 0.1%

Figure 3 Browser types usage by Endicia visitors.

We compiled the data into a graph containing the top 10 browser type and version as seen in Figure 4.

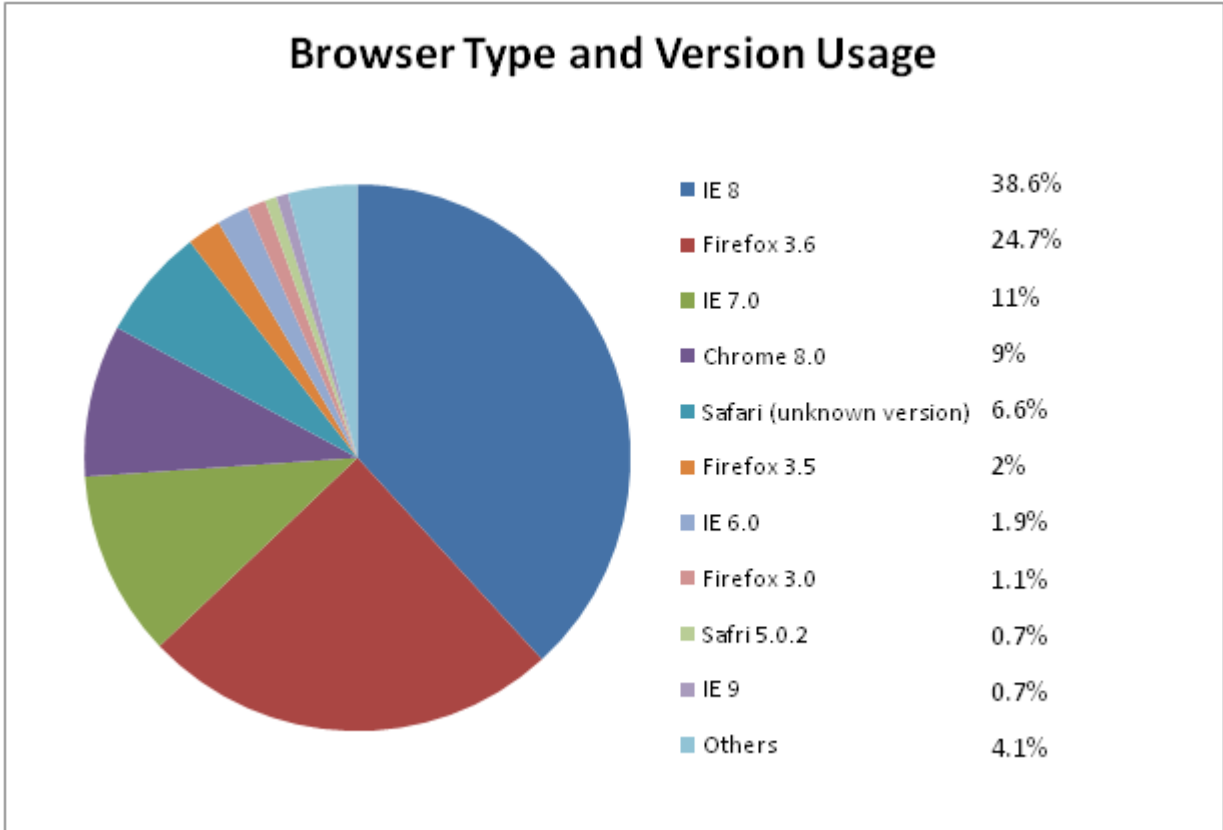


Figure 4 Graph of the top 10 browser type and version.

It is evident that the Internet Explorer (IE) 8 is the browser of choice for over a third of Endicia's customers. Mozilla Firefox (Firefox) 3.6 is also quite important with 25% of the user base covered. However, just looking at specific browser version is not a good way to see the target of our browser compatibility testing. The following graph (Figure 5) presents the information based on browser type which is more helpful when deciding what browser to support and from what version up we should support.

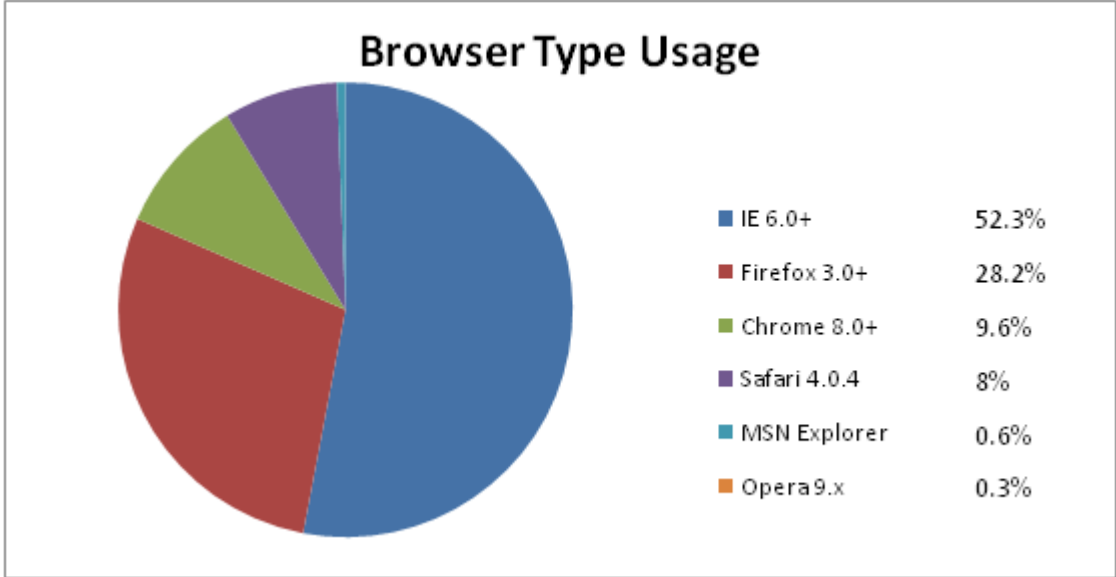


Figure 5 Graph of browser types usage.

The graph does ignore unknown versions of certain browsers because if the version is not known, we cannot test that particular version for compatibility. Looking at this graph, we can see that supporting IE 6.0 and up is important for Endicia with over half of the user base using IE. Firefox is close behind in user base with 28% of Endicia users. Originally, we thought Opera would be an important browser to support, however, from this graph we can see that only 0.3% of Endicia’s customers actually use Opera as their browser. Therefore, our focus will be placed on supporting IE 6.0+, Firefox 3.0+, Chrome 8.0+ and Safari 4.0+ with periodic testing done on Opera.

### 2.4.1.2 Browser Height Usage

Browser height is important when considering the page fold, the height of the visible portion of the page before scrolling is necessary, so the most important elements are within the fold. Figure 6 shows the statistics on the height of the browser. The numbers have been categorized into more meaningful sections in Figure 7. If the web application has a fold of 550 pixels, about 70% of the users for Endicia will not have to scroll to view the major elements on the page.

Browser Height (in pixels)		Visitors	
1.	550 to 599	131,911	16.8%
2.	800 to 899	115,410	14.7%
3.	600 to 649	97,415	12.4%
4.	650 to 699	78,629	10.0%
5.	500 to 549	64,732	8.2%
6.	700 to 749	56,004	7.1%
7.	400 to 449	45,429	5.8%
8.	300 to 399	45,117	5.7%
9.	750 to 799	41,367	5.3%
10.	900 to 999	38,295	4.9%
11.	450 to 499	34,871	4.4%
12.	1000 to 1499	22,757	2.9%
13.	200 to 299	10,282	1.3%
14.	100 to 199	2,280	0.3%
15.	Unspecified	1,313	0.2%
16.	Less than 100	713	0.1%
17.	1500 or More	220	0.0%

Figure 6 Browser heights of Endicia visitors.

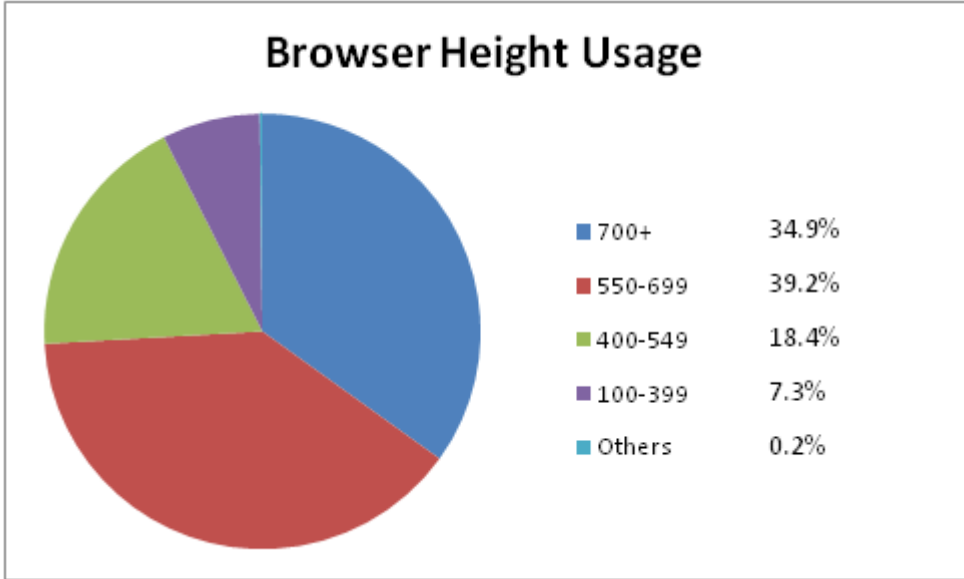


Figure 7 Graph of browser height usage.

**2.4.1.3 Browser Width Usage**

According to Usability.gov, a resource site for government website developers, “[h]orizontal scrolling is a slow and tedious way to view an entire screen” [40]. Therefore, eliminating horizontal scrolling for our pages would be important unless it is part of the design of the page. As a result, browser width is more important than height when considering the usability of web pages. Figure 8 shows the width in pixels the visitors to Endicia web sites have for their browser windows. The following graph (Figure 9) had been compiled from the data. As we can see, browser width of 1000+ is used by about 75% of the visitors to Endicia’s sites. We only cover 70% of visitors to Endica’s sites by having a web page height constrained to 550px. We can cover 90% of visitors by constraining the width of the web page to 800px. It is necessary to cover more site visitors when considering width so the visitors do not have to scroll horizontally to view all the page elements. Our goal is to cover 100% of the user base. However, to do so, most users will have a really narrow web page with unused spaces to the left and right. Because of the narrow web page, users will have to increase the amount of scrolling vertically to view all elements. Additionally, screen resolution will only get better in the future. Therefore, keeping the width at 800pixels is the optimum approach for providing the best usability to most people.

Browser Width (in pixels)		Visitors	
1.	1000 to 1499	510,380	64.9%
2.	1500 or More	105,778	13.4%
3.	750 to 799	54,073	6.9%
4.	900 to 999	42,601	5.4%
5.	800 to 899	39,074	5.0%
6.	600 to 649	8,325	1.1%
7.	700 to 749	7,628	1.0%
8.	650 to 699	6,377	0.8%
9.	550 to 599	4,585	0.6%
10.	500 to 549	2,561	0.3%
11.	450 to 499	1,807	0.2%
12.	Unspecified	1,145	0.1%
13.	300 to 399	1,003	0.1%
14.	400 to 449	721	0.1%
15.	200 to 299	426	0.1%
16.	100 to 199	178	0.0%
17.	Less than 100	83	0.0%

Figure 8 Browser widths for Endicia users.

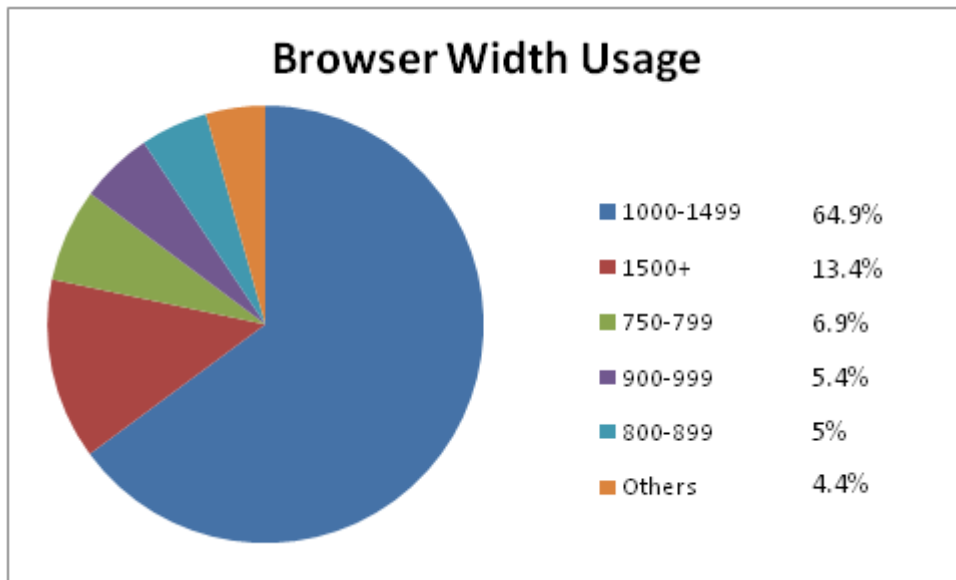


Figure 9 Graph of browser width usage.

From these statistics, the web page development on the user interface (UI) should work the same way on IE 6.0+, Firefox 3.0+, Chrome 8.0+, and Safari 4.0+ browsers. We should strive to fit all essential elements in 800 pixels by 550 pixels browser view.

## 2.4.2 Print Engine Requirement Research

### 2.4.2.1 Operating Systems Usage

Operating system (OS) usage is important for the print engine portion of this project where native OS calls are necessary to get printer information. As we can see from Figure 10, only Windows and

Macintosh operating systems make up a significant percentage of the user base. Since native OS calls are the same no matter what version of OS the user is running, the graph (Figure 11) combines the versions shown in the data for a clearer presentation of what OS has the most users. Windows clearly is the most important OS for this project with 86% of the visitors to Endicia’s sites using some version of this OS. Therefore, the focus of this project is to make the web application work with users running Windows OS. The next step for the print engine would be to support the 13% of the site visitors who have a Macintosh OS.

Operating Systems		Visitors	
1.	Windows XP	317,510	40.4%
2.	Windows 7	209,261	26.6%
3.	Windows Vista	131,520	16.7%
4.	Macintosh	100,598	12.8%
5.	Media Center 2005	12,648	1.6%
6.	Linux	6,669	0.8%
7.	Windows Server 2003 and XP x64 Edition	3,779	0.5%
8.	Windows 2000/NT 5	2,233	0.3%
9.	Not Specified	1,129	0.1%
10.	Macintosh (iPhone)	717	0.1%

Figure 10 Operating systems usage by Endicia customers.

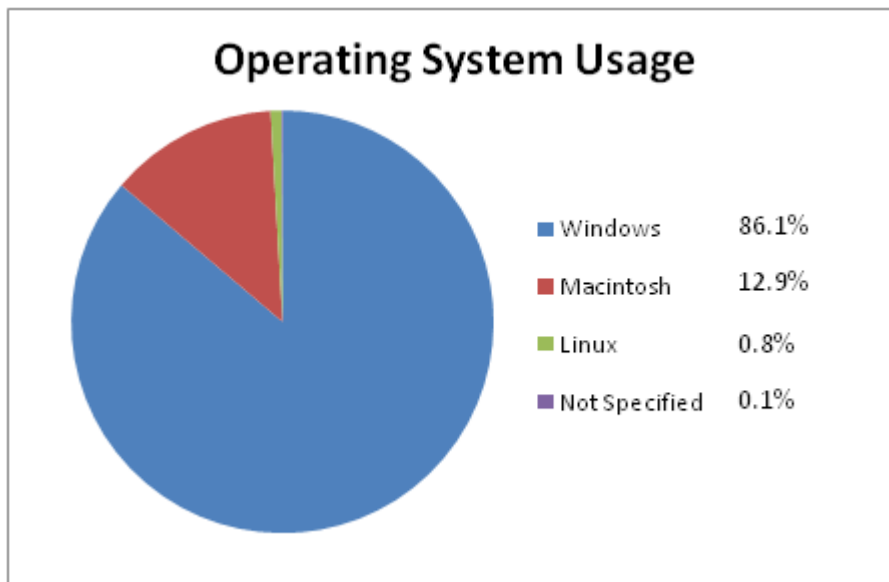


Figure 11 Graph of operating system usage by Endicia users.

### 2.4.2.2 Java Support Usage

It is imperative to Endicia that the web application should minimize the amount of resources the users have to install in order to print stamps since that is the original reason why this project is necessary. Java is absolutely essential for the print engine applet to run and allows us to print. Therefore, we must check if Java is enabled in the Endicia’s visitor’s browser. Figure 12 showed that over 80% of the Endicia’s customers enable Java. Hence, it is reasonable to use Java for printing because most users already enabled it.





Figure 12 Java usage by Endicia's users.

## **Chapter 3 Requirements**

According to the market research done by Endicia, customers prefer a web-based system over a downloadable client [24]. Therefore, it is the goal of the project to migrate the features available in the current DYMO Stamps software to the web so more users will be willing to use Endicia's stamp printing services. This web-based application targets the front office workers who do not ship often enough to purchase a subscription service and may not have the authorization to install the current software, but will be willing to utilize the service if it is more accessible [24]. Since the targeted users do not have a common technological environment, the application was designed to be able to support different browsers. For printing the stamps, the application can support print requests to normal inkjet, laser, and thermal printers using stamp sheets or rolls sold by Endicia. This project consisted of making the client side web page and developing the print engine with Endicia. Two members of the team developed the client side web page design while one member worked on the print engine. For all account management and money transfer actions performed by the user, the application employed the Endicia API using cold fusion to manage these transactions safely [18]. The web application supports all commonly used browsers. These browsers include but may not be limited to

- Internet Explorer 6.0 or better
- Firefox 3.0 or better
- Chrome 8.0 or better
- Safari 4.0 or better

It is imperative that the position of all elements on the page remain the same no matter what browser the user decides to use [19]. Screen size of the user also affects the viewing experience so the overall size of each page was placed under consideration as well. The project was done in two major development phases. First phase included the implementation of the client side and the print engine separately. The second phase will involve the incorporation of the two sides.

# Chapter 4 Design

## 4.1 Client Side

### 4.1.1 Environment

This project was done on computers provided by Endicia, running Windows 7 operating systems. The technologies used for the client side development involved HTML4, CSS2, and JQuery. HTML4 is a tag based language used for creating the actual webpage's structure. CSS2 is a presentation language that dictates how the HTML tags are presented to the viewers. JQuery is a JavaScript library that simplifies scripting involved in the creation of an interactive web page [23]. We had done a tutorial based on CSS Grid and JQuery for the web team at Endicia. See Appendix A for more of the tutorial.

WYSIWYG, a webpage creation tool that generates HTML as one drags and drops webpage elements into position on a page was used at one point during the first iteration of UI development to create a fast mock-up for usability review [43]. However, this tool was abandoned because the client side team's manager wanted to have the website hand coded to ensure all elements in the website were built the way they were supposed to be.

The IDE used for committing the project on the repository and creating the initial webpage was Eclipse. However, as the project progressed, the IDE was switched to Adobe Dreamweaver because it renders a preview of the webpage and offers better syntax checking with auto correction. Dreamweaver has three different views available. The code view shows only the code, the split view both the code and the rendered webpage preview and a design view with webpage preview only. In the split view as visible in Figure 13, Dreamweaver allows us to navigate to the code defining a specific element by selecting the element in the webpage preview.

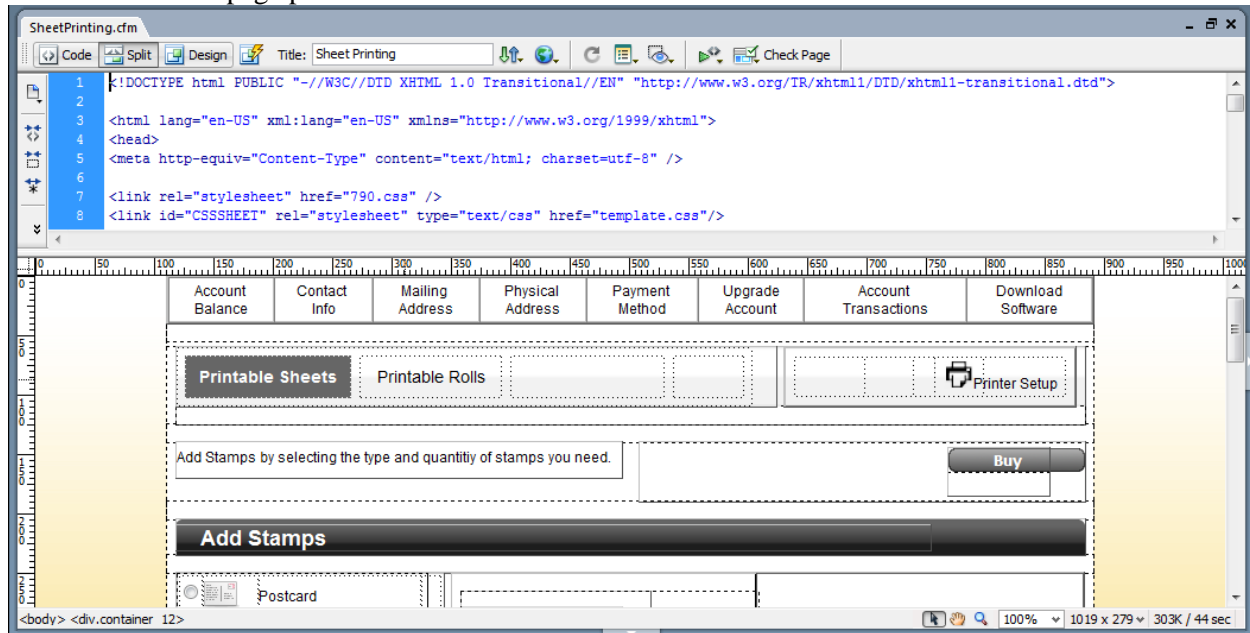


Figure 13 Split view with code in the top portion and preview in the bottom portion.

While we were making changes to the code, Dreamweaver would try to render the changes we made in its preview window. However, because we use CSS Grid for the structure of the webpage, Dreamweaver renders our pages in a way different from a common browser and thus was unreliable for the final looks of the pages. Therefore, after editing the code for our web pages, we would navigate to the locally stored web page files and open them in a browser in order to see the effect of the changes we made. The webpage preview was used to create, edit, or remove HTML tag elements as well as quickly

navigating to the code of certain elements. Dreamweaver has supports for ColdFusion commands, CSS commands, JavaScript libraries, and HTML tags to help expedite the coding process.

Coding on Eclipse was not completely abandoned because Eclipse with the Web Tools Platform (WTP) plug-in creates warnings that informed the team of cross browser compatibility issues. Eclipse showed us warnings such as invalid tag locations, extra end tags, and unrecognized tags that helped the debugging process especially involving Internet Explorer. We did most of the major coding in Dreamweaver due to its auto complete feature for CSS style, HTML tags, and JavaScript/ JQuery functions. The code was then periodically checked in Eclipse to make sure no warnings came up that could cause certain browsers to break.

File revisions were maintained on a subversion (SVN) system provided by Endicia. Eclipse with Subclipse plug-in allowed us to submit our files on SVN for safe keeping and easier circulation between teammates. SVN had prevented many major progresses lost and allowed us to view the progression of our interfaces when backtracking is necessary.

## **4.1.2 Design**

### **4.1.2.1 Design Process in Endicia**

Endicia was beginning to change the way projects within the company are carried out. Previously, projects would go through a large planning cycle where a detailed visual, a set of images that indicate the structure and styling of an application, was created and agreed upon. Then once the planning phase was over, the coding and construction was done based on the drawn up plans laid out in the visual. The project is then tested fully and submitted for final review. This design pattern works out in a perfect world but most of the time the people who were planning the project don't fully know what kind of product the project should deliver in the end. In those cases, lots of work is thrown away to accommodate new changes to the architecture of the project.

Endicia is now trying to have more iteration in the project planning process. This project is part of the experimental shift towards the rapid prototyping process. New projects, such as ours, go through a short initial planning phase where a rough wireframe or visual is made. Wireframes specify the general functionality and structure of the page with few cosmetic details while visuals offer the same information as a wireframe but with more details on the cosmetic elements of the product. Then, a short cycle of development is performed to implement features according to the wireframe. A round of revision and review by the business and management team decide on adjustments to the design necessary to make the product more in line with the project goal. Then another cycle of development is done based on the revised wireframe and other previously developed products. These steps repeat until a final cycle of product review approves the design before the project is complete and delivered. With these iterations the project has a better chance of coming out more fully realized, better designed, and possibly developed faster since less time is potentially spent on work that could be thrown away at the end. We were responsible for the fast development part of the cycle to make the wireframe into interactive web pages.

### **4.1.2.2 Design Approach**

The client side web page creation process started from a wireframe produced by SolutionSet, a marketing service company hired by Endicia, in collaboration with the Endicia marketing department. A server-less site was then created from the wireframe with as much of the user interaction built in as possible. By making a server-less site we can have the various buttons, pop-ups and input forms properly interact and function without them making calls to a database server. In this way we can have the front end and visual part of the site working without having to wait for the back end to be completed with all the security features.

The implementation of the user side of the web page was done with a pair-programming technique because both members of the client side development team were not experts in JQuery, JavaScript, or CSS Grid. One member typed up the code while another member directed progress and proof read on the fly for any typo or syntax errors not picked up by the editor. The vast majority of the code has been overviewed by both members to ensure code security and usability. In cases where a problem could not be easily resolved, the two members researched individually for possible solutions to ensure that the best possible system was developed.

The structure of the web application was created from HTML with Cascading Style Sheets defining the look and feel of the pages. CSS 960 Grid System, a tool that helps web page development through the use of a grid system to position each element with precision, was used to ensure that all browsers render the positions of every element at the same location on the screen [35]. Using the 960 CSS Grid came on recommendation from the company SolutionSet who were doing extensive usability and visual style studies on Endicia's main web pages. SolutionSet recommended Endicia use the 960 Grid systems for their pages and Endicia asked us to use this grid system for our application so they could see how it looked and worked. When defining a style for the web application we first looked at similar elements in the Endicia's current website so the new website could follow the same style as the current Endicia web pages.

Once a general structure of the pages was created, some simple cosmetic changes were made to make the page look decent. Then JavaScript and JQuery libraries were used to add user interaction, input verification, and selection animation to the page to make it more appealing and usable. All buttons, links and all manner of input fields were all linked to JQuery event handlers. With JQuery we looked to dynamically change the structure of the webpage with user input. We wanted to hide elements that were not needed at the time while showing other more important elements.

After the page is fully functional, testing was performed to make sure every element interaction was as expected. The application is then sent out thru Endicia for review and feedback. Once the feedbacks had been compiled, the webpage were changed in the same way as it was developed and the process repeats till a final review approves of the design.

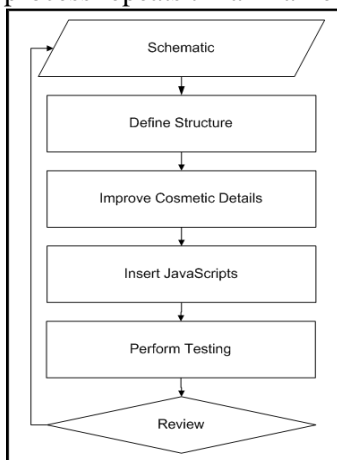


Figure 14 Flowchart of the development process

Once the print engine had been packaged up into a Java applet, it was integrated with the web pages so when the user clicks the print button, the system can perform the necessary tasks to get the stamps printed. Along with the print engine, the Endicia server was integrated with the web page so the user can transfer money and pass along the order information between the server and the web pages.

## 4.2 Print Engine

### 4.2.1 Environment

Development for the print engine functionality was also done on Endicia machines running Windows7. The development of the print engine was done in Java with the use of third party libraries. The IDE used for the print engine was Eclipse because of its extensive Java library and SVN plug-in.

### 4.2.2 Design

The printing module of the application was the focus of the print engine design. The design was restricted to mostly commonly available and standard technology. Since the market research performed by Endicia indicated that users would prefer printing their postage without additional downloads, it was

important to restrict the possible implementation platforms to those that are already downloaded on user machines such as Java, Adobe Shockwave, and Microsoft Silverlight. Because of the web-based nature of the project, this was also important so that users within different browsers and operating system environments would have the same experience.

Printing devices will vary from user to user. Asking users to input their printing device information could take away from the user experience so it was important that the print engine would be capable of automatically detecting this information and reporting it back to Endicia's server for use in support and future software versions. This meant including a communication channel between the print engine and the installed printer devices.

Postal requirements state that the image quality of printed postage has to be the best that can be currently produced. The best quality that most common printers can achieve today is around 300dpi. Therefore, the engine was designed to be capable of printing in this resolution. Final images must be at this resolution but samples and drafts can be at lower resolutions.

Because of restrictions placed by the United States Postal Service, users should not be able to print multiple copies and print out more stamps than they paid for. Because of this, browser specific printing or plug-ins capable of printing cannot be used since these include dialogs where users can input the number of copies to print out. Endicia owns a patent on "silent printing," or the ability to print to a device with a controlled dialog [19]. This patent is used in a similar product created by Endicia for La Poste, the French postal service, and was used as an example during this project's design and implementation processes for the print engine.

Printer filtering was also a key part of the print engine design. This feature was important for two reasons. The first reason was similar to the need for silent printing; virtual PDF printers are available today that can pose as hardware printers, allowing them to be displayed in a list of available printers in standard web browser print dialogs. With virtual printers, user can store the stamp images as PDF and print as many copy as they want which would be against USPS's no multiple copies policy. The second reason was to improve user experience. Since users would be able to order stamps printable on standard stamp sheets and DYMO label rolls, it was important to only display standard printers when printing to stamp sheets and only DYMO label printers when printing to DYMO label rolls. With printer filtering, the print engine would be able to detect the types of printers available, filter them out, and only display the appropriate printers. By creating a list of allowed printer, we could reduce stamp fraud and improper stamp printing.

Many errors can occur during the printing process and it is important that any printing errors reported by the printer or operating system are immediately available to the print engine. Errors such as "out of paper," "out of ink," and "printer offline" for network printers must be brought to the user's attention in order for them to take proper action. Most errors are detected at the operating system level by the print spooler and within the print queue; therefore the print engine had to be able to obtain a certain level of access to the user's operating system.

Ultimately, the print engine is responsible for retrieving the user's purchased stamp. It is able to formulate and post a valid request to the server using the browser's session data to prove that it is requesting the stamp on behalf of the user. Any errors encountered during the request formulation or stamp image retrieval is shown to the user and logged on the back end.

Web-based transactions are sometimes the target of malicious attacks and so it is important that the engine is protected from any network intrusion or hijacking attempts. The United States Postal Service and Endicia each has a set of secure coding standards [19] that were followed during the development of the system.

# Chapter 5 Implementation

## 5.1 Client Side

### 5.1.1 Iteration I: Development from Wireframe

#### 5.1.1.1 Development Process

The iteration I of the project for the client side development was done base on a wireframe from SolutionSet. The first wireframe we received, as shown in Figure 15, was missing images that indicated connection between certain elements upon user actions. The edit link for each order was one example of the lack of information on what to display to the users. While we did have details about what the add stamps action should look like, we did not have any information on what the edit should show to the user. Therefore, for the parts of the wireframe that were vague to us, a discussion between the team members were necessary to decided on the most usable presentation for the website. In the case of the edit link, we connected the order back to the Add Stamp pop up with the order's information filled in for editing.

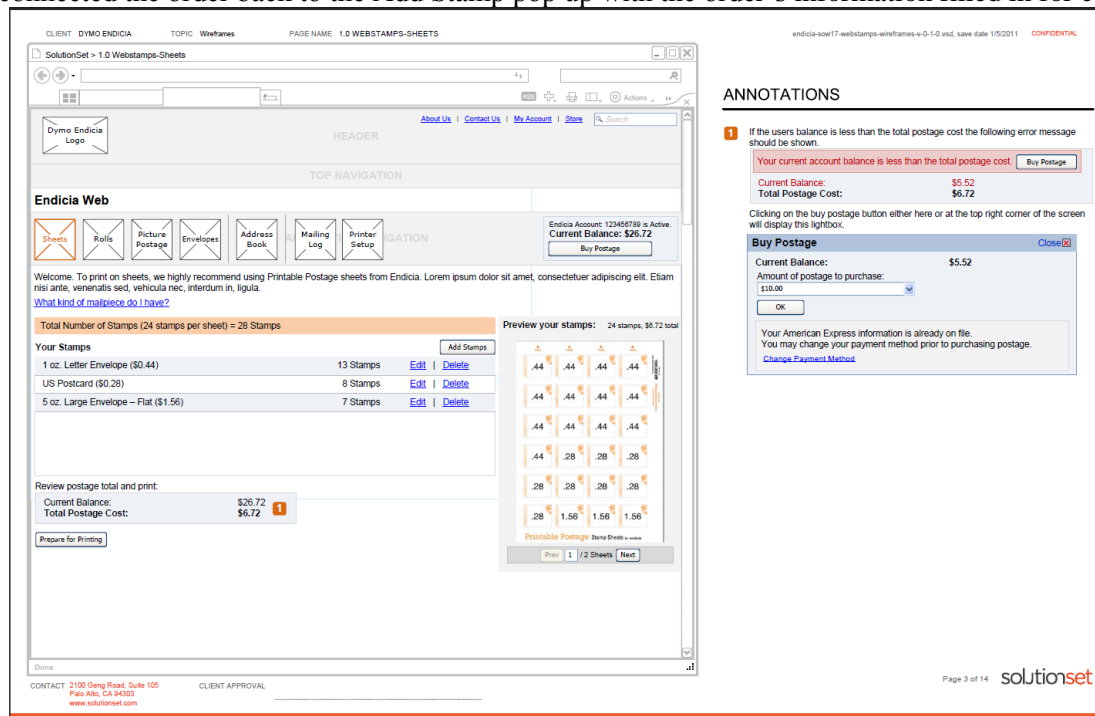


Figure 15 the sheet printing page in the iteration I wireframe.

The wireframe we were given was devoid of cosmetic detailing. Therefore we decided the webpage should mimic the CSS styling used on the current Endicia website so the pages look thematically the same when a user navigates from Endicia's main website. For elements that were not part of our project such as the integration with Picture-it-Postage page, bold color such as purple was used to highlight further extension of the page in the future.

The wireframe contained incomplete or very roughly composed application components. We used our own judgment to try and compose better elements to replace those that were not well thought out or missing clear directions. The wireframe used many hyperlinks in the page to indicate potential user actions. We replaced many of these links with buttons to give the user a greater feeling that action can be taken without being taken to a new page as most hyperlinks do on websites. For the user interactions that were essential to the flow of the application such as delete and edit, buttons were used to emphasize the available actions. We also sized many components differently in order to have the website conform more to the 960 Grid Systems we are using. Clarifying elements were also added on the page, for example, the

printed total amount for each stamp order in the order table for sheet printing was added based on our own considerations. These differences in the stamp orders table can be seen in Figure 16 and Figure 17.

Your Stamps		<a href="#">Add Stamps</a>
1 oz. Letter Envelope (\$0.44)	13 Stamps	<a href="#">Edit</a>   <a href="#">Delete</a>
US Postcard (\$0.28)	8 Stamps	<a href="#">Edit</a>   <a href="#">Delete</a>
5 oz. Large Envelope – Flat (\$1.56)	7 Stamps	<a href="#">Edit</a>   <a href="#">Delete</a>

Figure 16 the table that shows the orders in sheet printing in the wireframe.

Your Stamps				<a href="#">Add Stamps</a>	
Mailpiece	Postage Value	# of Stamps	Total	Edit	Delete
Postcard	Postcard \$(0.28)	1	\$0.28	<a href="#">Edit</a>	<a href="#">Delete</a>
Small Flat Rate Box	Small Flat Rate Box \$(5.20)	5	\$26.00	<a href="#">Edit</a>	<a href="#">Delete</a>

Figure 17 the actual table that shows the orders in Sheet Printing. Border remains to help us structure the cell width.

Since we were working without any visual resources (images) for the webpage, we had to gather images ourselves. Most of our images came from capturing pictures from Endicia’s websites and DYMO Stamps while some were from Endicia’s repositories. Although more of a cosmetic detail for a site, pictures added clarification and indicated the direction the user can proceed through our application. Pictures helped us label working elements of our rough application and gave the application a look and feel that was much closer to a finished product. We decided to spend time on putting more images into the web pages so the responses from testing would be focused on the interaction and structure and less on the visual detailing of the pages.

The wireframe was missing information such as the descriptions of each mail piece type necessary for the user to be able to pick the correct mail piece. The wireframe only provided the information for the Large Envelopes as shown in Figure 18. Therefore, the DYMO Stamp application was used as a reference to fill in the prices, descriptions and mail class for each mail piece type as seen in Figure 19. The wireframe never contained any information on what type of value is invalid for the inputs in the web pages. We used DYMO Stamps as a reference on what the maximum and minimum input values can be when determining custom value and number of stamps to order.



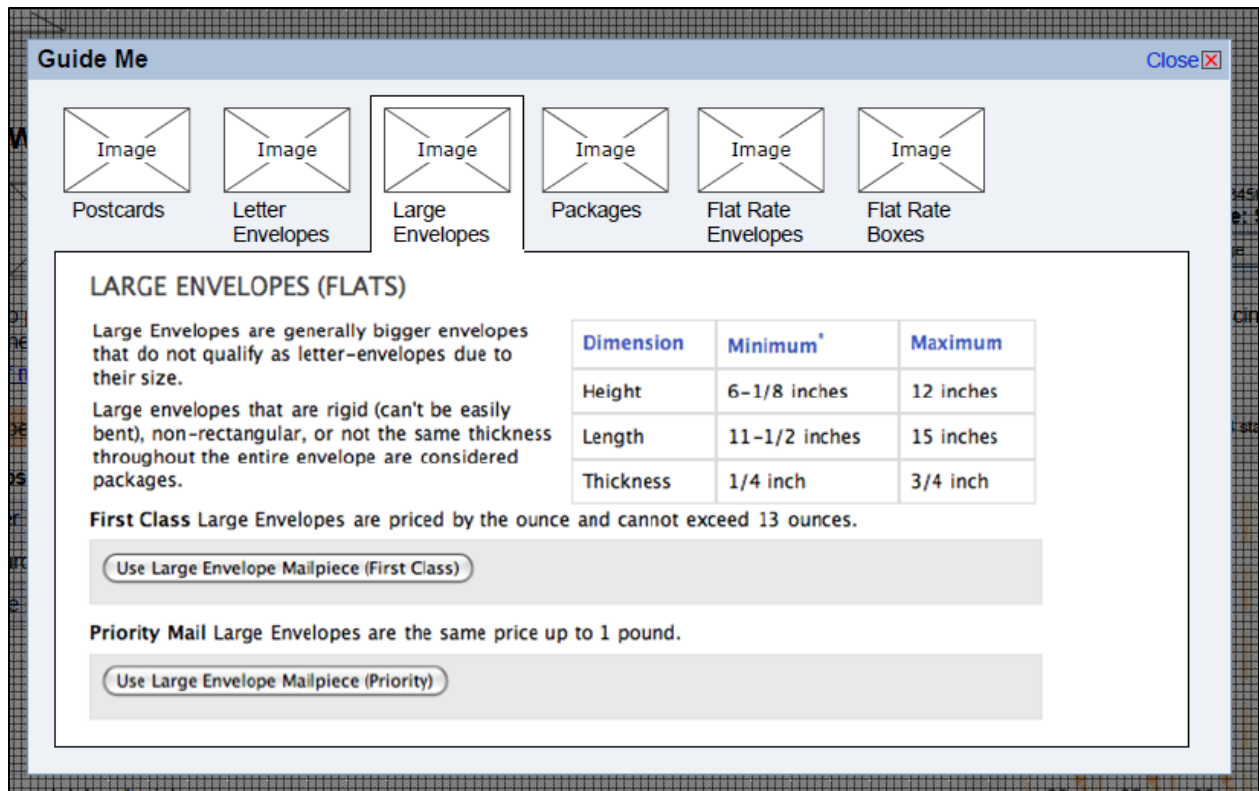


Figure 18 the guide me pop up visual from the wireframe. This is the only view available.

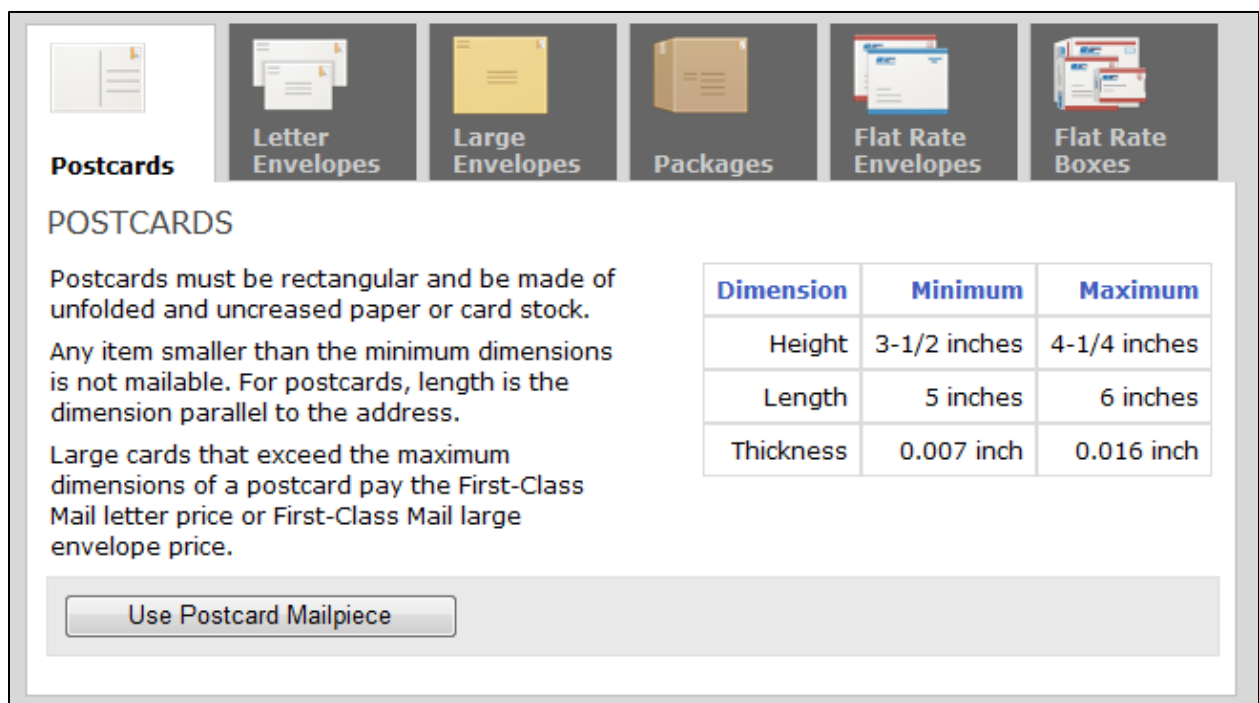


Figure 19 the guide me in DYMO Stamps contain information for all mail piece types.

The visual detailing that was done at this point in the development cycle was elements that could affect the reviewer's assessment of the application. However, many features deemed to be cosmetic were left un-finished and postponed for later work. The purpose of our work is to fully define a flow to the web

application. Cosmetic features, tweaks, and full compatibility with browsers and screen resolutions would come later once the structure and flow was agreed upon by marketing and business personnel.

### 5.1.1.2 Iteration I Testing

When testing the codes, we made sure that both members of the client side team tested all features of the application. Each of us especially tested features that the other members coded so we would not be testing the system based on the way we think it suppose to work. Without a testing bias to our own working code we could more actively test how each of our mental model of the application should work and not how they currently function.

The first round of testing was intended to cover all expected user interaction with the application. We tested the code first by looking through the page for any structural problem such as buttons displaying pass the edge of the page. Then we clicked all clickable items for proper links to Endicia sites, affiliated companies sites, and pop ups in the current page. This round of testing ensured that the code in our application functions in the expected way.

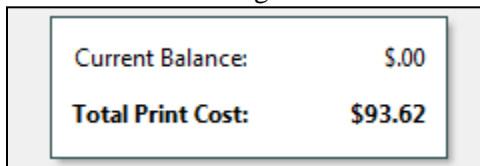
The second round of testing was performed on all input fields, drop down menus, and other interactive elements to check if the elements respond as expected. For this round of more rigorous testing, all the interactions between the displaying fields and the input fields were tested to the full extent of our knowledge. The testing was done all from the point of view of a user intending to break the system. All possible accidental mistakes and wrong doing that we could think of was done to the interface. Taking input fields to their extreme was a good deal of the testing.

Another major testing methodology was to switch focus from element to element while giving input to both elements. When a user switches focus while typing in a field the input can get ahead of some error checking. With this type of testing in place, we could prevent even more input field malicious activities.

### 5.1.1.3 Iteration I Development Issues

The two members in charge of the user interface portion of the project were not experts in the JQuery/ JavaScript library. So, certain functionality provided by such functions as `.toFixed()` were first manually coded with a series of other commands before utilizing the appropriate JavaScript function. After major version changes to the application, we did large amounts of code refactoring. It was during the refactoring of the code that repeated lines of code were moved into functions and unused variables were removed to make the code leaner.

Internet Explorer has some compatibility issues that we were not aware of due to the fact that the code was initially tested on Firefox and Chrome. The CSS float property was one of the elements of the page that was different between browsers. IE ignores the float property randomly, causing some of our formatting to lose the spacing necessary to make the page more visually appealing. As a result, CSS Grid system was used even for small elements like the numbers in the blue box showcasing the current balance and the total in the Figure 20.



Current Balance:	\$0.00
Total Print Cost:	\$93.62

Figure 20 the total cost display.

Internet Explorer also has problems with custom tags that are not part of the standard tags for HTML. Tags such as `<m>` and `<AddStampTitle>` were used to help styling the webpage in CSS without using a class. However, IE does not support fetching the object from these tags for JQuery thus breaking most of the JavaScript involved in making the webpage interactive. It is at this point in time that we realized that Eclipse with the Web Tools Platform (WTP) plug-in creates warnings that indicate potential issues web standard does not cover. After each major section of the code was completed, we went through all the warnings generated by Eclipse and changed all formatting to that of the web standard so all browsers that support the standard will render the page the same way.

### 5.1.1.4 Iteration I Result

After some extensive testing on the webpage, the application was sent to the marketing and business personnel for review. The result of the iteration I development of Sheet and Roll printing page can be seen in Figure 21 and Figure 22. The progression of the web pages as we developed it for iteration I can be seen in Appendix B.

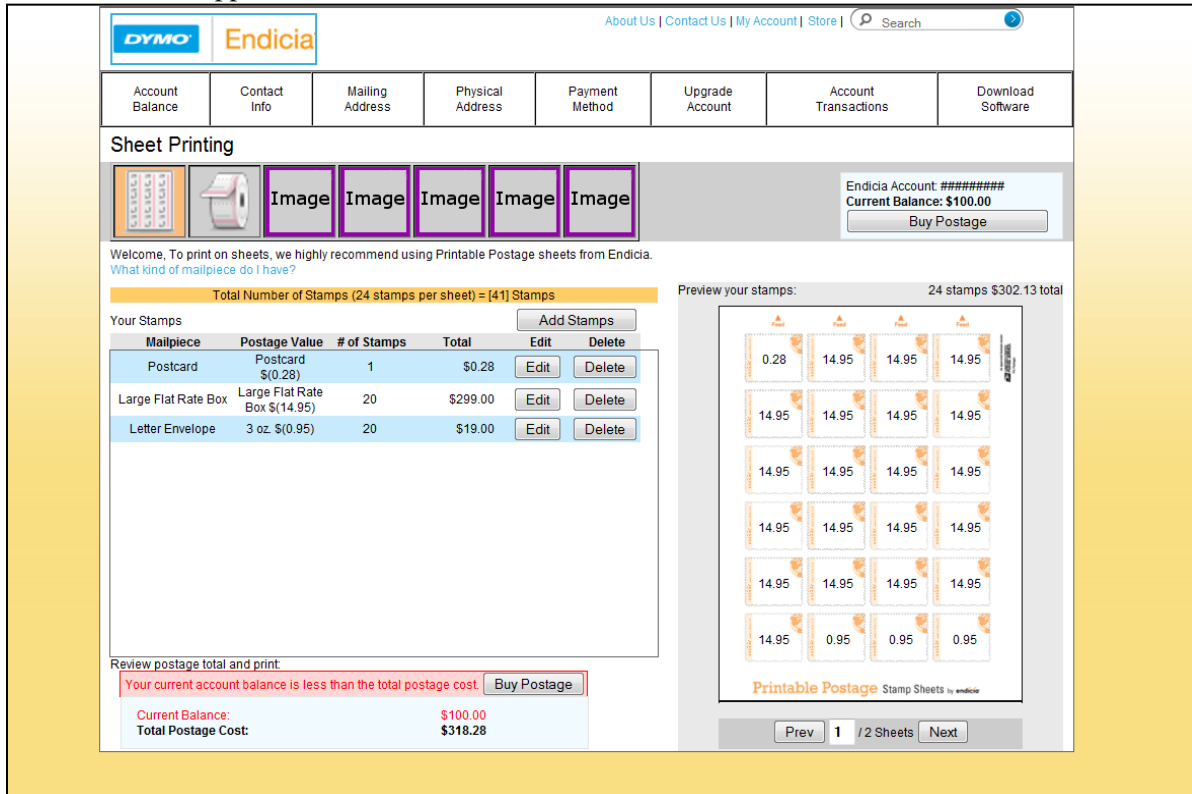


Figure 21 the final view of the sheet printing page based on the wireframe.

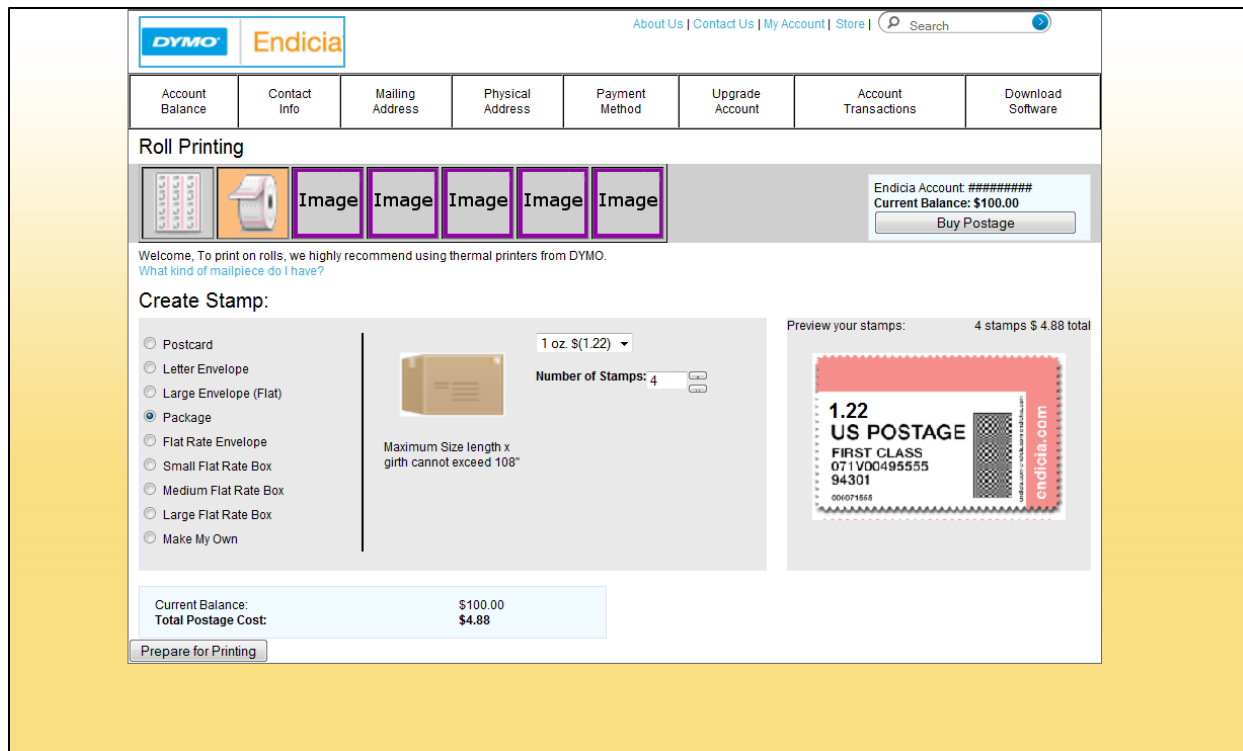


Figure 22 the final view of the roll printing page based on the wireframe.

## 5.1.2 Iteration II: Development from Visual

### 5.1.2.1 Iteration II Development Process

A quick meeting with the marketing manager revealed that the wireframe we were given in iteration I was for a future integration with a new Endicia website still in the design phase, and not for the integration with the current Endicia website that the product of this project will be added to. The marketing department gave us a visual based on the current Endicia website for a second round of development.

We talked to the company that created the wireframe and the visual for this project, SolutionSet. During this meeting the designers from SolutionSet displayed some of the reasoning behind the flow of the application for the new visual along with some of the interactions involved that are not visible in the visual wireframes they sent to us. We were also able to show them the prototype we had done so far on a previous wireframe so they could get some feedback on how their wireframe work was interpreted.

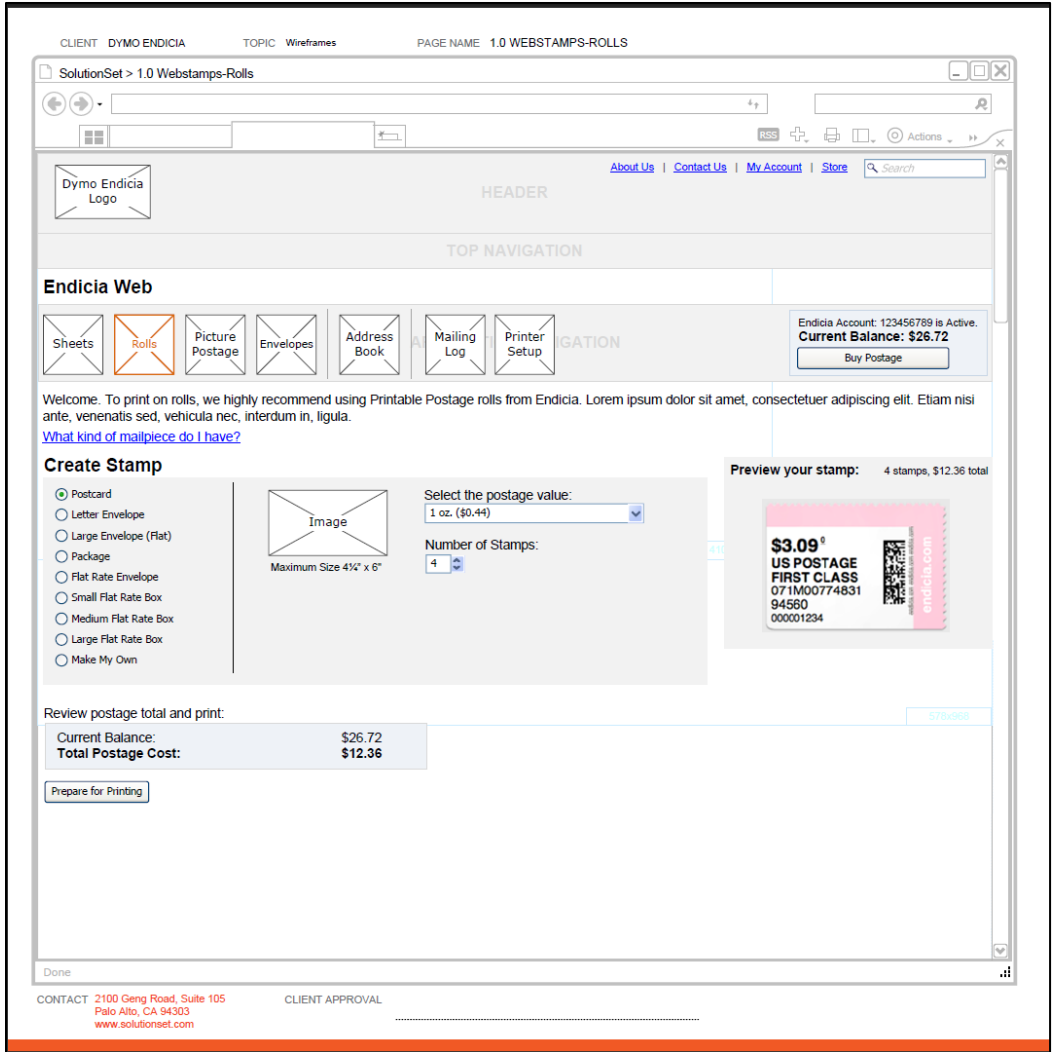


Figure 23 the Roll Printing Page from the first wireframe

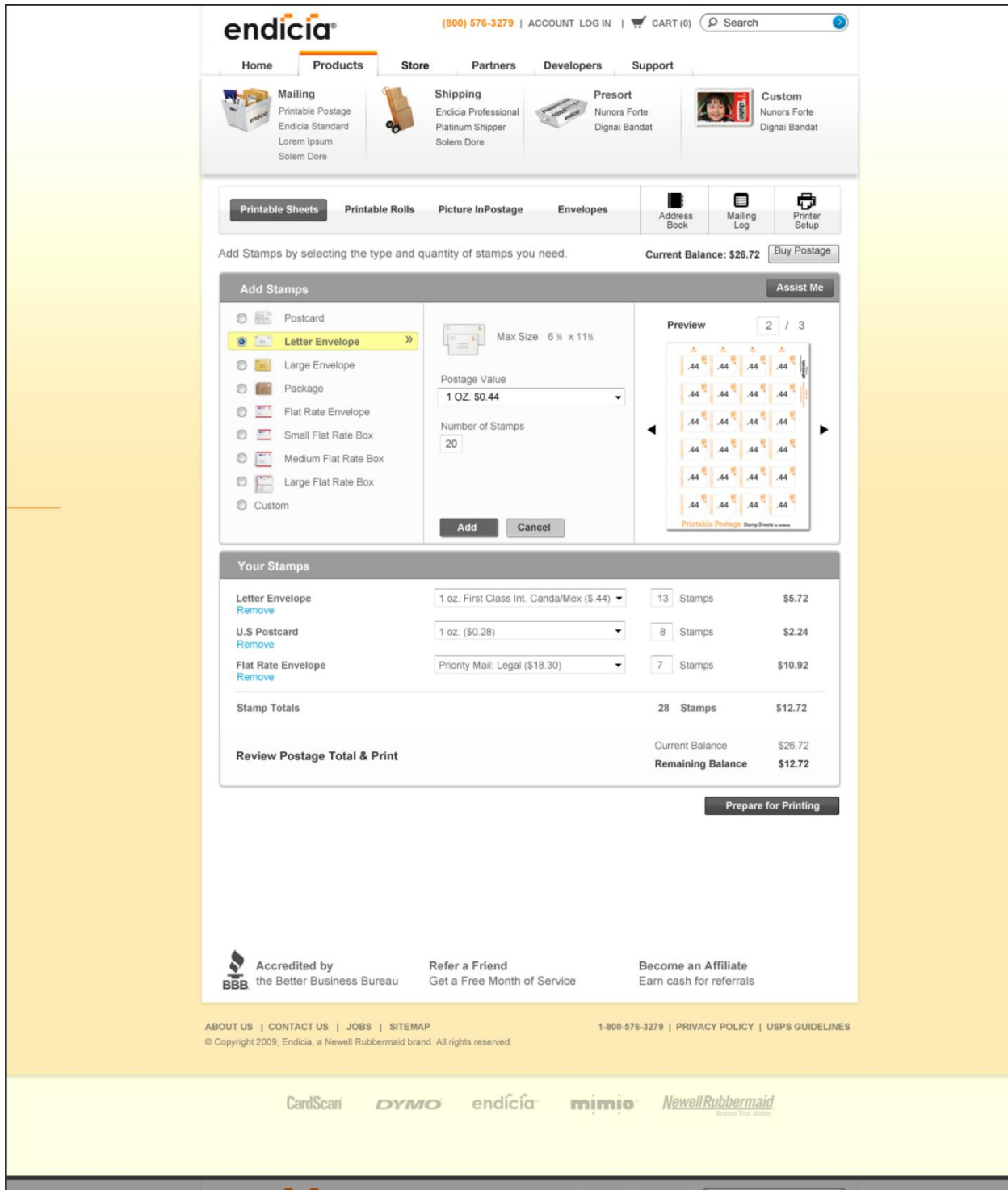


Figure 24 the sheet printing page from the new visual.

The new visual for iteration II development has a similar structure compared to the Roll Printing page of the old wireframe as in Figure 23 and Figure 24. The structure of the page remained virtually the same with certain links such as the “About US” link being moved to a new location. Since the new visual is based on a narrower web page design, we had to generate a new CSS Grid system with 790 pixels as the width of the page instead of 960 pixels. The 960 CSS Grid website contained a CSS generator that

made the job of switching the size of each grid of the 12 grids per line as easy as switching out the CSS file we were using. However, the smaller grid size made certain elements harder to read and many elements of the page required resizing and relocation to make the page viewable. The differences between the 960 pixels width and 790 pixels width can be seen in Figure 25 with 960 pixels view on the top.

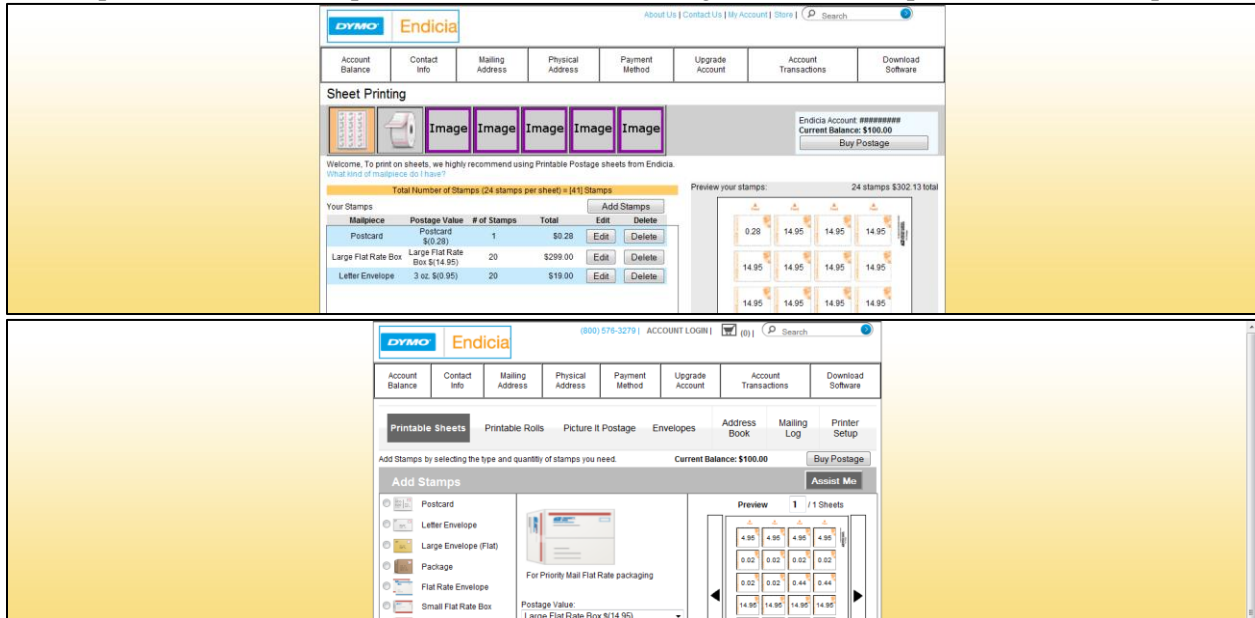


Figure 25 the old Wireframe with 960 pixels width vs. new visual with 790 pixels width.

The new visual reduced the need to add new stamps in a pop up window with the add stamp section right on the main page. Therefore, on the Sheet Printing page, every action that existed in the Add Stamps pop up (Figure 26) has to be migrated to the main page. Since all the codes already existed for adding a stamp order to a table, only minor changes such as cloning the drop down menu and adding it to the table were necessary.

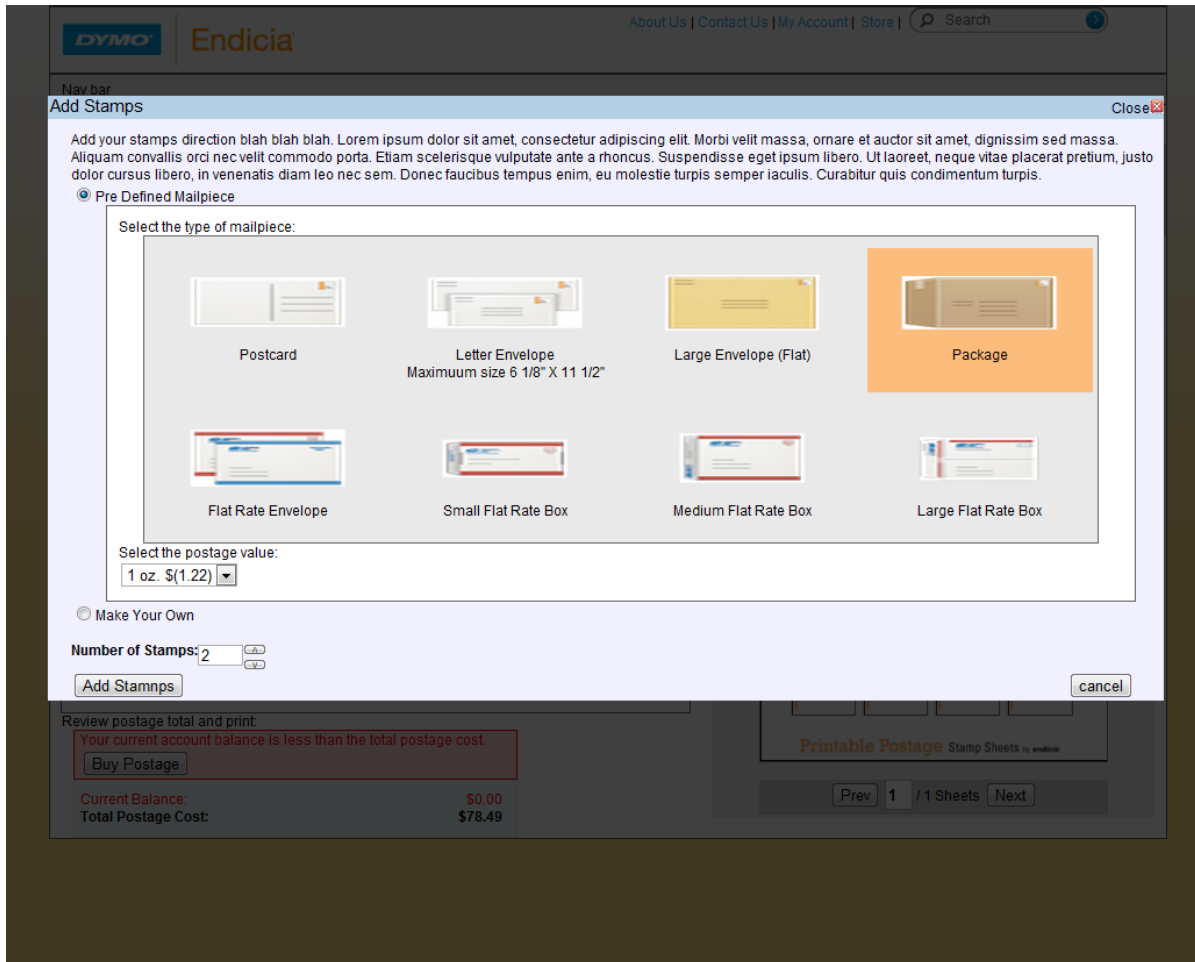


Figure 26 the add stamps pop up that displays in the Sheet Printing page when user presses the Add Stamps button.

The version of the Sheet Printing and Roll Printing page submitted for review contain images from the DYMO Stamps. The fold, the position where most of the Endicia customer can read the contents of the page without scrolling down, was not a serious concern in iteration I because the wireframe contain no indication of where the fold should be. Additionally, the design of the page in iteration I was wider than in iteration II so every element could fit above the fold without too much problem. In this view, we placed the red dash on the left of the page to indicate the fold of 600 pixels. We filled the drop down menus with the actual values in the DYMO Stamps applications so the web pages looks more like the official version for better tester experience (Figure 27).



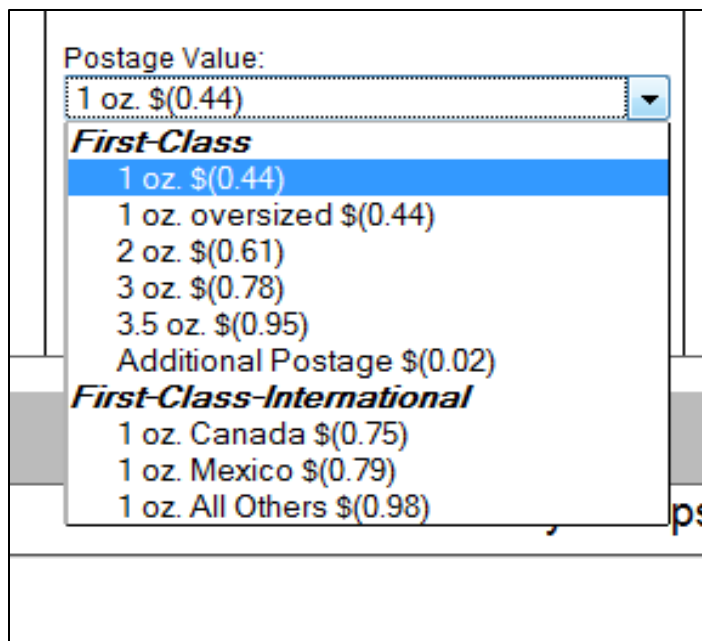


Figure 27 the drop down menu for letter envelope.

### 5.1.2.2 Iteration II Testing

The testing methodology remained the same as in iteration I. However, testing was done on all browsers we support because of the lesson we learned about IE in iteration I. Because this version of the application contain more input field and thus more area for malicious inputs, testing took longer to complete to ensure the user cannot perform any task that are not allow to.

### 5.1.2.3 Iteration II Development Issues

During testing, great effort was put into trying to get negative numbers for both stamp cost for custom stamps and the number of stamps. By having either one of these numbers negative, the total cost would display as negative, and the remaining balance check would fail, potentially allowing user to print out stamps for free. Having negative numbers is of course unacceptable and changes to calculate values beyond acceptable limits were attempted with extreme prejudice. In our attempts to reach negative numbers or really large numbers outside of acceptable ranges we took to performing unorthodox actions such as copying and pasting, repetitive input, and reloading of the page.

There were other errors that would also cause the application to break such as with negative numbers, but these would also cause the page to display incorrectly or in a very confusing way to the user. In some cases we could force input fields to hold numbers that were far greater than what would be common sense for stamp ordering process. For instance, ordering over a hundred stamps was disabled after testing because that is the maximum in DYMO Stamps application and ordering vast quantities of a single kind of stamp would cause display errors. With large numbers for stamp cost or number of stamps, total calculated cost would stretch far beyond the display boundary.

Similar errors occur when we tried to calculate with empty values in input fields. Multiplying numbers with non-numbers or empty fields will create a non-number value and will be displayed as “NaN” (not a number). Having interface elements display “NaN” disrupts the flow of the application and befuddles the user trying to place an order. Important values such as total cost, total number of stamps, number of sheets, or other such fields that derive their values from the numbers in the input fields could display “NaN” if the user does not input anything for certain fields.

All of problems described above involving input fields were important because the preview and total calculations were generated dynamically from the input values. As Figure 28 shows, when a change is made to field 1, all elements labeled 2 in the figure will change accordingly. However, by imputing a value in a text field and shifting focus away from the text field by clicking on something else in the

application we can bypass some of our error checking. During testing we recognized that the `.change()` JQuery command does not activate on a un-focus event. In the event where the user deletes the text in an input field and clicks away from it, the script that automatically puts in “1” in input field will not be triggered. We added a `.blur()` events to all script with a `.change()` events so when an element becomes un-focused, the input field is set to a valid amount. Before testing we were trying to update the page and values on it as infrequently as possible to keep things fast. After testing we realized that this infrequent updating of preview allowed for incorrect use of the application thanks to focus switching. Now we do not just update when a field is changed but we update the whole preview when there is a key up, key down, change, or blur (changing of focus) event in order to avoid getting negative, “NaN”, or extreme values for calculated fields and inputs.

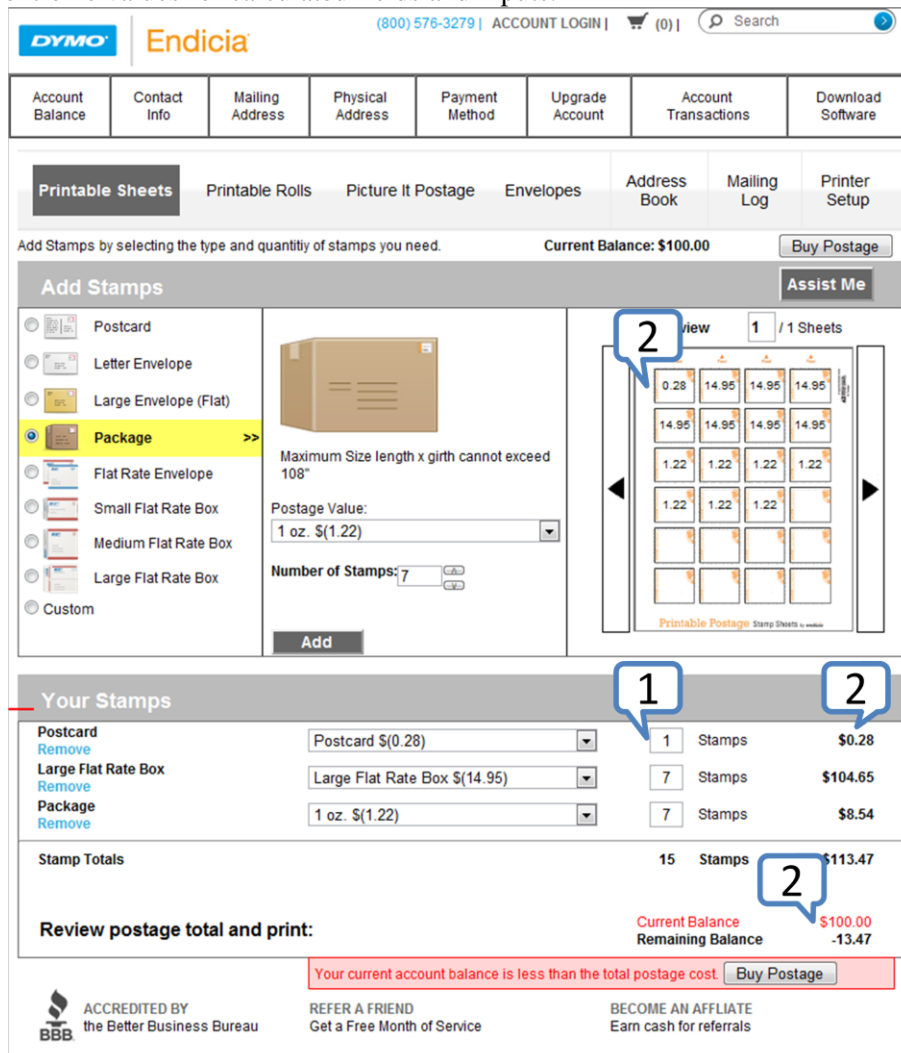


Figure 28 Sheet printing page with dynamically changing elements.

We converted our application from iteration I wireframe to iteration II visual. In doing so, we ran into some new problems. Although a lot of the functionality was the same, the JavaScript interacted with the HTML DOM in a much different way. Some functionality that was previously stored away in pop-up elements was now on the main page. However, because most of the element ids remained the same, the script associated with the add stamps interaction remained virtually unchanged. In this new view, we utilized the `.clone()` method in JQuery in lieu of the relatively complex edit button functionality we had previously used. The drop down menu was cloned so the user can change the value of the postage in

the table via these menus instead of going back up to the add stamp section of the page to change the order.

We looked for new and more proper ways to perform what we had done before. In one instance we were creating new elements, such as the text field in the order table without the use of an edit or delete button. This change in application architecture was problematic in JQuery because event handlers initialized after the webpage finished loading does not bind to dynamically created elements. When we were using buttons, we could set buttons to call a function as we create them. Now that we no longer have buttons for interacting with the elements in the order table, we needed a new way of assigning the functionalities to changes the elements in the table. With these non-button elements we had to bind the event handler to the element as they were created. After trying many inefficient and overly complex solutions we found another function unique to JQuery to help us out. The `.live()` function allowed event handlers to bind with newly created elements. This function made our job easier when dynamically created elements were necessary to make the page functional.

More browser compatibility issues were discovered when testing the second iteration of the pages based on the new visuals. We performed extensive testing on all the browsers targeted for this project along with Opera and discovered problems in Opera and Safari.

Opera does not handle a *keydown* event like other browsers. A *keydown* event occurs when any key on the keyboard is pressed down. In the web pages, we needed *keydown* events to prevent alpha characters, letters of the alphabet, from being inputted into the fields. On a *keydown* event the code will only allow certain characters, such as numbers and the delete key, through as input to the field. Opera does not allow us to prevent the default *keydown* event from happening and will display the entered key no matter what our scripts try to do [36]. However, since Opera is only 0.3% of the user base for Endicia, creating a work around just for Opera is not a good investment of our time. Additionally, input warnings and a parsing system beyond the *keydown* event were coded into the page to prevent users from performing any action that could cause a security risk. Therefore, we left the problem as it is in the current state.



Figure 29 Input Highlight in Safari



Figure 30 2 Input highlight in Firefox



Figure 31 Input Highlight in Chrome

Safari has a usability feature that interferes with our input warning system. Most browsers have subtle signs to show the user the field they are currently editing. Firefox and Chrome change the border color of the input field to blue and gold respectively. In Safari, the border is enveloped by a wider blue frame (Figure 31, Figure 31Figure 34, and Figure 35). When adding custom postage to an order, the user specifies the amount of the postage. The input field where the user enters the custom postage amount changes its border color based on the validity of the input. We create a green border for valid inputs and a red border for invalid inputs. The blue frame safari created around input fields masks the border color we set (Figure 32). If the user cannot see the border color changes easily, the system cannot inform the user of invalid inputs. However, we also have warning information popping up when the user does input an invalid amount shown in Figure 33. This pop up was developed for people with color blindness. By having this additional text warning for invalid input, Safari users should not have a different experience when using our web pages. Therefore, no special solution was created to accommodate this difference in Safari.

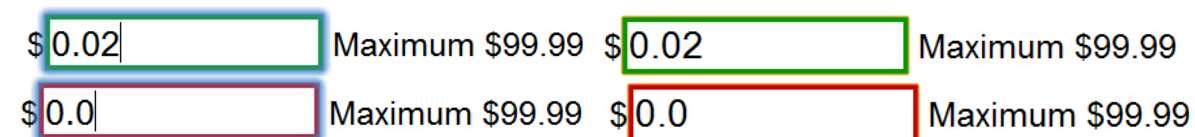


Figure 32 Safari vs. Chrome for input field highlight when border is green or red.

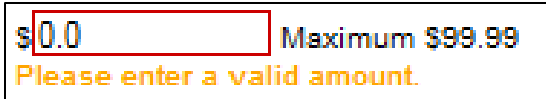


Figure 33 Warning when input is invalid

### 5.1.2.4 Iteration II Result

The result of the second iteration was sent to the marketing team once more for feed backs. The version submitted for review can be seen in Figure 34 and Figure 35. The progression of the web pages as we developed it for iteration I can be seen in Appendix C.

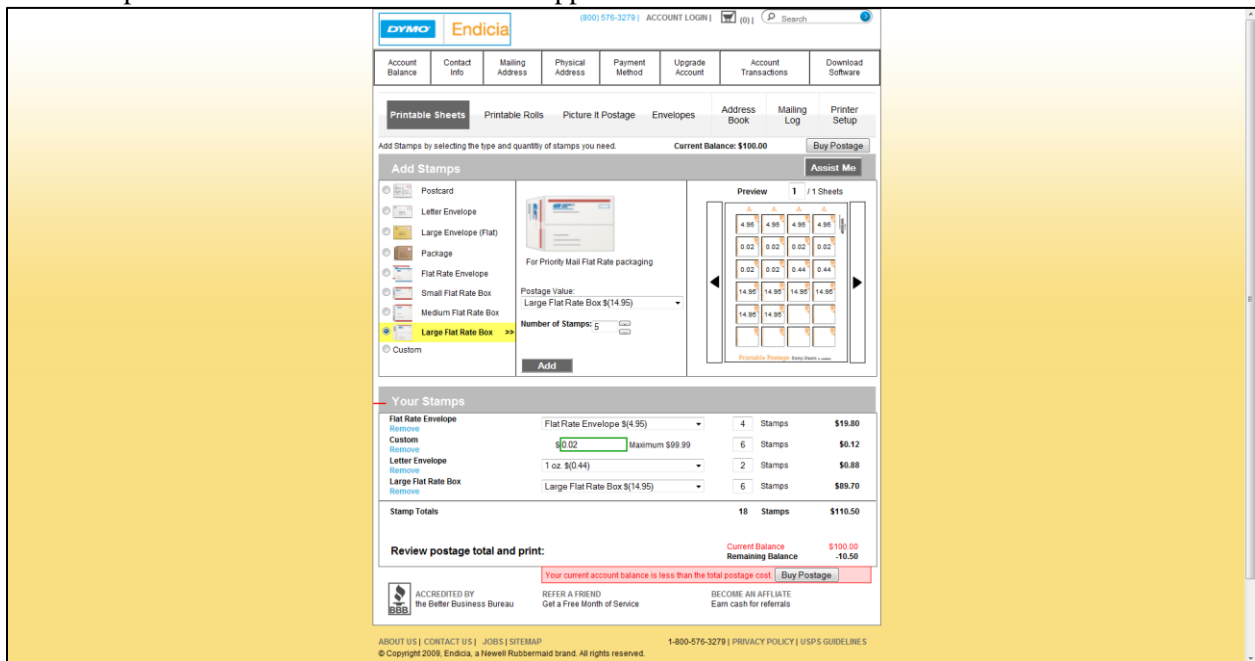


Figure 34 the sheet printing page following the new visual.

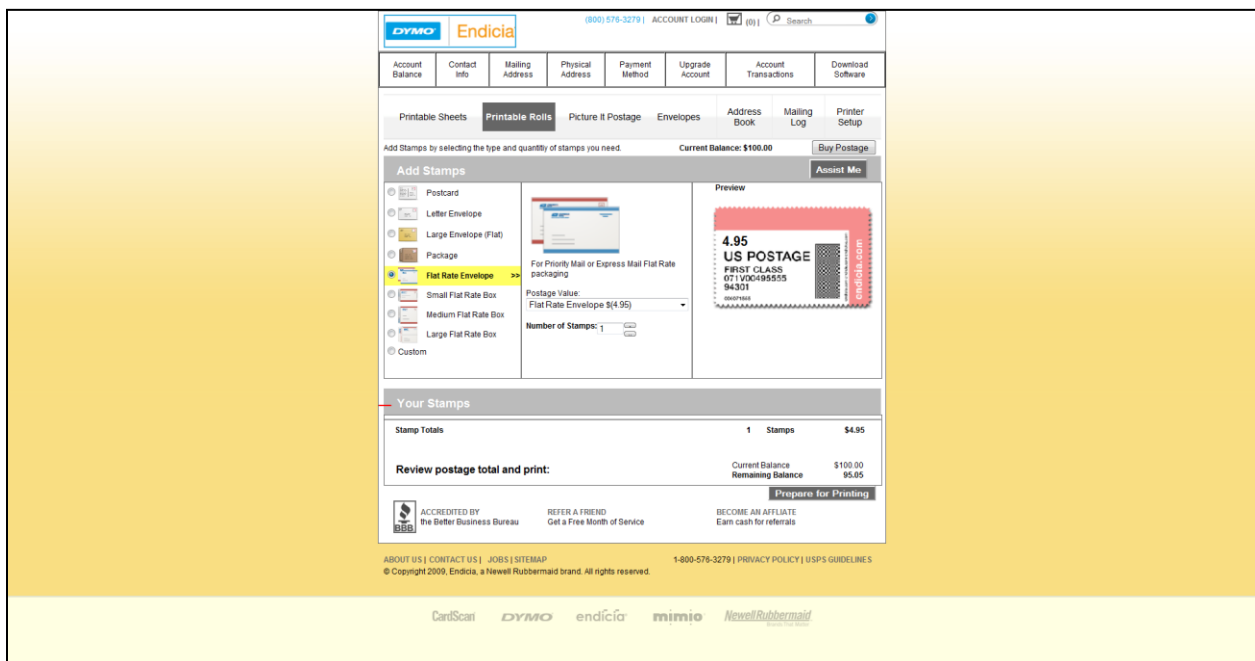


Figure 35 the roll printing page following the new visual.

### 5.1.3 Iteration III Feedback and Enhancement

Both the wireframe and the visual versions of the web pages were circulated around Endicia mostly through the marketing, product specialist and the management teams. The feedback returned from these people was mostly regarding the structure of the page. Figure 36 and Figure 37 contains the screenshot of the webpage before the feedback and the page after we made the changes base on the feedback.

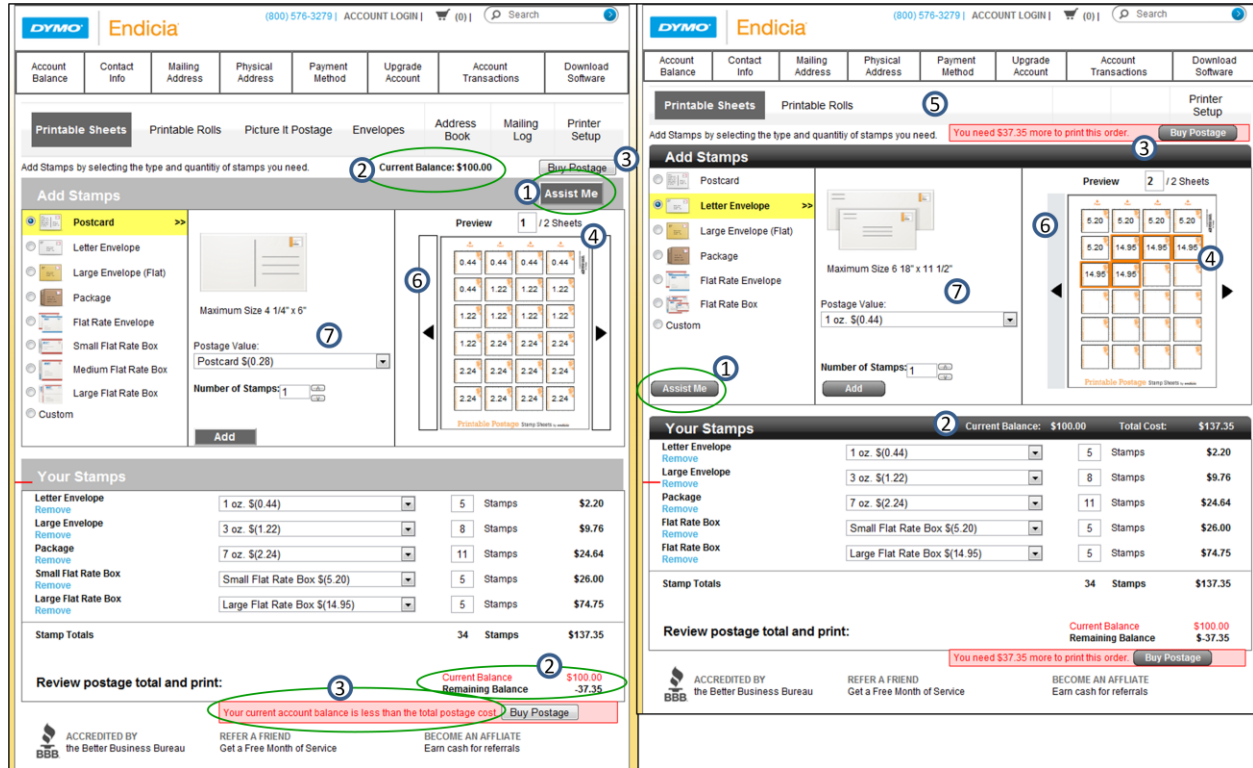


Figure 36 Left: before feed back. Right: after feedback.

1. The Assist Me button has been moved to the bottom of the radio buttons column so users who are confused about the mail piece types can spot it more easily.
2. The Current Balance and Total Cost of the order are now in the Your Stamps bar so they can be above the fold.
3. Buy postage button remained in place with an extra warning indicating low balance when necessary. The warning shows the amount of postage the user's account is missing for them to print the current order. Also, all buttons have been changed to have a uniform style.
4. When a new order is added or current order is edited, the stamps changed in the preview will have an orange border. When an order is deleted, all borders are removed. When an order is added, the stamp sheet preview will go automatically to the last sheet.
5. The buttons that indicated linking to PictureItPostage, Envelopes, Address Book, and Mailing Log were removed because these application elements will not be linked up anytime soon.
6. The preview section was too busy so the box around the arrows was removed and replaced with a background color change when the area is hovered over.
7. The default view of the page contains Letter Envelope and 1 oz values set (vs. Postcard in the current)

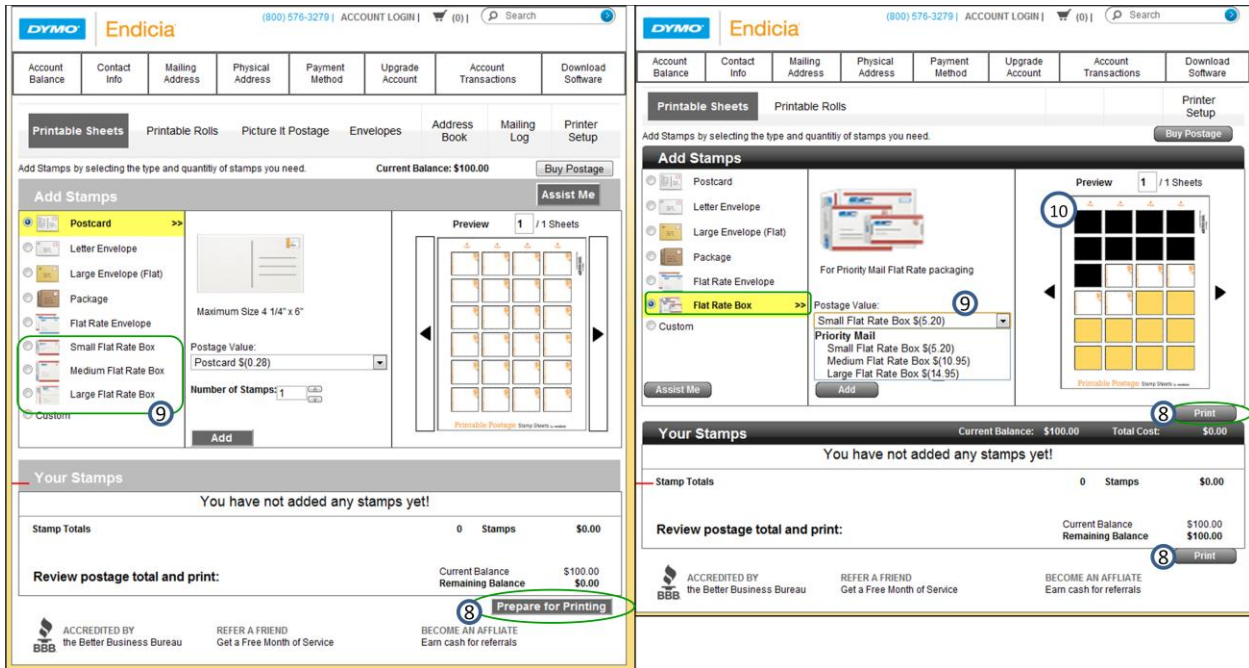


Figure 37 Left: before feed back. Right: after feedback.

8. The Prepare for Printing button was shortened to Print and was in a hard to notice location on the page. The Print button was recreated between the Add Stamps and Your Stamps sections.
9. The Small Flat Rate Box, Medium Flat Rate Box, and Large Flat Rate Box have been combined into one radio button with the three types in the drop down Postage Value menu.
10. The user can now select the starting position of the sheet printing in the preview part. This option is only available on the first sheet indicated which orange backgrounds on mouse hover. A black background is given to cells where no stamp will be printed.

Previously the fold of 600px is on the Your Stamps title bar. Spacing for the account bar, navigation bar, as well as the Add Stamps and Your Stamps title bars has been reduced. Now the first row of the order table appears above the fold (the red dash on the left side of the page).

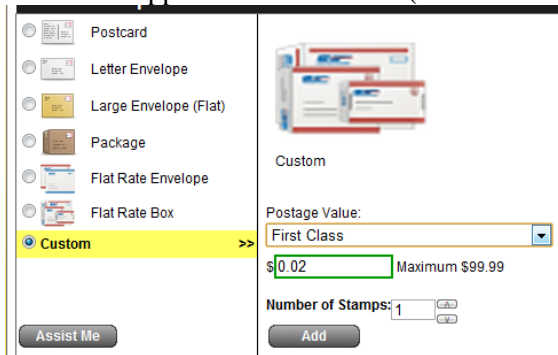


Figure 38 Custom postage with mail piece classes.

Because Endicia has to report to the USPS the type of mail piece they sold, custom postage must have first class, express and priority option as shown in Figure 38.

Your Stamps		Current Balance:	\$100.00	Total Cost:	\$0.00
Stamp Totals	Letter Envelope 1 oz. \$(0.44)			1 Stamps	\$0.44
Review postage total and print:				Current Balance	\$100.00
				Remaining Balance	\$99.56

Figure 39 Dynamically changing order information for Roll Printing.

The roll printing page was changed as well so the stamp total bar shows the current order so the user can see the order changes in multiple area of the page (Figure 39).

## 5.1.4 Integration

### 5.1.4.1 Integration with Endicia's Server

In order for our web application to be fully functional, information from Endicia's server was necessary. We called for information such as the amount of money the user has on their account, the current postage value, and the user's current payment method. When calling the backend server for information, a page refresh should not be necessary to see the updated information. Instead, we used Ajax (Asynchronous JavaScript and XML) [41] calls to reload the subsections of the web page that had the updated information. With Ajax, our page sends a request to a Cold Fusion database controller which interfaces with Endicia's database. In the request we specify an event such as 'balance request' and the Cold Fusion controller performs a database query in response to that event. The Cold Fusion controller then packages up the response it got from the database into a manageable object and sends it to the webpage. Then the webpage updates the local values with the information it gets back from the Ajax call.

For most server requests, only an event call was necessary to pass on the information. However, the server side required some way for the client side to pass on information about the stamp order that is less visible to the users when print is clicked for security and aesthetic reason. Information such as mail piece type and weight class was packaged up in Simple Object Access Protocol (SOAP) [42] format as per the request of the server designer. To do this, a variable was created with the appropriate structure set up. Since the preview and the total cost of the page were created by looping through every order on the page, the SOAP packet was recreated in the same function as the preview or the total cost generation.

The server creator suggested combining orders with the same mail type and mail class into a single order instead of allowing the user to create an order multiple times as seen in Figure 40. With a reorganization of the add stamp event handler, the system would loop through each order and check if the order already existed and update the number of stamps instead of creating a new row (Figure 41). Currently, the system does not prevent users from changing an order's mail class and value into one that already exists.

Your Stamps		Current Balance:	\$100.00	Total Cost:	\$0.88
Letter Envelope Remove	1 oz. \$(0.44)			1 Stamps	\$0.44
Letter Envelope Remove	1 oz. \$(0.44)			1 Stamps	\$0.44

Figure 40 Same order appearing twice on the order table.

Your Stamps		Current Balance:	\$100.00	Total Cost:	\$0.88
Letter Envelope Remove	1 oz. \$(0.44)			2 Stamps	\$0.88

Figure 41 Same order combined into one order in the order table.

### 5.1.4.2 Print Engine Integration

The print engine applet was written in Java and could only interact with the web application through JavaScript function calls. Since the print engine could not trigger an event in the web page, whenever the print engine wants the web pages to change the display, it had to call JavaScript functions such as *showPrintWaiting()* which shows the pop up with a “please wait” message. Through several JavaScript functions, the print engine can supply the user with meaningful responses that will help the user understand the current status of the application.

### 5.1.4.3 Integration Problems

When the web pages we created were merged within Endicia’s main web pages, we realized that the CSS that was defining the style of Endicia’s web pages was interfering with the styling of our application. For example, the CSS property *border-collapse* was globally set for every table on the Endicia web page. With this property set there was no spacing between the cells in a table. We do not want this property and many other globally set properties affecting our pages. Therefore, each element in the web page was looked through and extra CSS properties were reset to default as necessary.

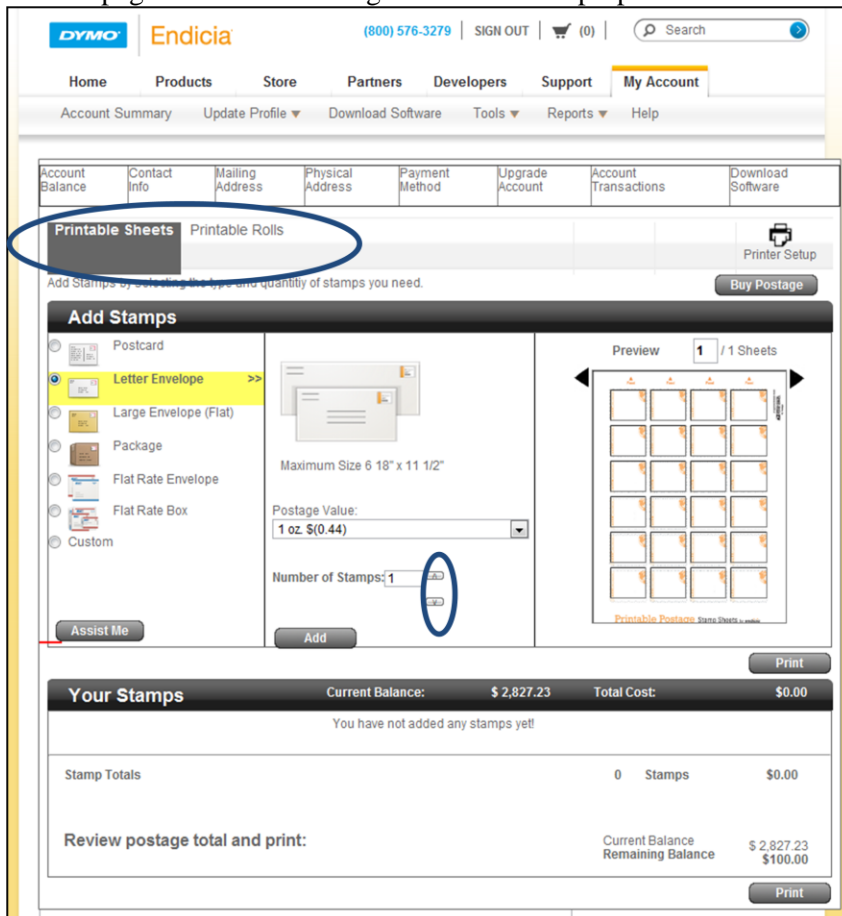


Figure 42 The web page with messed up CSS styling

During integration, many JavaScript components stopped working and had to be fixed for the applet to be integrated within Endicia’s web pages. A major problem in the integration process was the uncertainty of the structure of the information the server requires to process the order. For each stamp order, information such as the mail piece type, postage value, mail class, and the stamp amount would have to be passed to the server for a set of PDF images to be passed back to the client. Because no previous system existed, we decided to use SOAP packets that could be easily produced by the web pages and processed by the server.

### 5.1.5 Security



Security is an important part of this project. We performed most of the testing in the point of view of a user trying to break the system. However, security is not really a big concern for the web page from the user interface perspective because users will have to log in to their accounts before they can access the application. Any and all transactions the application executes will be through the Endicia API so that information is transferred in a secure manner. Therefore, UI development was focused more on minimizing the risk by eliminating more ways for the user to perform inaccurate actions.

## 5.2 Print engine

### 5.2.1 Proof of Concept

The printing module of the project required some level of access to specific printer information obtainable only through native operating system library calls. For security reasons, many common browser plug-ins are not capable of accessing this kind of information at all or without additional downloads. Fortunately, the Java Applet framework can gain elevated privileges with a signed certificate and the approval of the user. Access to native libraries was made possible with Java Native Access (JNA), a library for easily accessing operating system native libraries [21]. Using the Microsoft Developer's Network [29] as a reference, mappings from Java classes and methods to Windows API data types, structures and functions were created to meet the printer communication requirements.

A proof of concept was written and run locally to demonstrate and explore the kind of information available through the native API. It was shown that through the *winspool.drv* library, information such as a list of available printers, printer name, model, device driver version, and paper type, orientation and positioning was obtainable. Since we were unfamiliar with both JNA and the Windows native API, we implemented the proof of concept as a stand alone Java application rather than an applet; this helped ease the development by eliminating extraneous possible points of failure.

### 5.2.2 Applet Migration and Embedding

In order to embed the application into a web page, migration of the code to an applet was necessary. Using the applet Endicia developed for La Poste, we created a similar applet for our project. This included moving the core logic to the Applet class' *init()* and *start()* methods. Next, the applet was embedded into and run from a vanilla HTML page to simulate the user experience. Java applets are by default considered un-trusted code where they are run using a stricter security policy than other Java applications [20]. Operations such as reading local files on the user's computer and loading native libraries are not allowed unless a Java applet is elevated to trusted code. To relax these security restrictions, the applet must be signed by the applet provider and verified by the user (or the user's browser) [32]. The proof of concept applet was self signed and configured to request the needed security permissions to perform its functions when loaded in a browser.

The configuration required the use of the *keytool* and *jarsigner* tools available in the Java Development Kit (JDK) and involved creating a public/private key, generating a self-signed certificate (done with *keytool*) and signing the packaged jar with the generated self-signed certificate (done with *jarsigner*). Every time a new jar file was created from the source, it had to be signed again with the certificate.

### 5.2.3 Code Refactoring

With successful web page integration with the proof-of-concept, we performed code refactoring. Repeated code was abstracted away into methods for easy reuse and utility. Classes were created mostly for code involving the Windows API in order to hide intricacies and quirks involved in calling them and error handling. Many API methods involve multiple calls to the same method in order to achieve different things. For example, the *GetPrinter* [26] method is usually called twice; the first time to obtain the amount of memory required to store all of the structures returned in the second call, and the second time to populate the user allocated memory with the array of structures requested. Some of the Printer API methods also require a handle to a printer which is obtained through the *OpenPrinter* [28] method. The utility classes hide this multiple step process.

During this code refactoring phase, we also further abstracted the operating level specific calls to a common interface that any operating level API can adhere to. Interface methods such as `getPrinterDriverName` and `getPrinterStatus` were created that operating level specific API mappings can implement. With this abstraction, we were able to implement a factory method pattern that would generate the appropriate native API mappings based on what operating system the print engine detects on the user's machine. As the system grows to support more operating system environments, this design pattern can provide an advantageous separation between the higher level operations of the system from the lower level system calls.

#### **5.2.4 Custom Print Dialog**

Applying what we learned from the proof-of-concept, we implemented a custom print dialog using the refactored code to display only the list of printers that are supported by Endicia. The standard print dialog provided by the Windows operating system includes a list of all available printers on a user's computer. This list may include print devices unsupported by Endicia or virtual printers that can be used to circumvent the silent printing capabilities. With the custom print dialog, these unsupported devices, and virtual printers can be excluded from the list so that users would not be able to print to them. Ideally, this list of "prohibited" printers would be separate from the system and be available as a service through SOAP or as an XML file. This way, the list could be easily updated in a manner that is independent of the print engine's implementation without having to redeploy the entire system; the user would simply refresh the browser. However, for the sake of time and easier testing, this list is currently hard coded into the system. When this service is implemented, it will be included in the applet.

We were able to filter out the printers based on the printer driver names. A much simpler way would have been to simply retrieve the "friendly" printer name through the Java Printer Service API. However we decided against this; a user is capable of changing this name (through the Windows Control Panel, for example) and so depending on this as the filtering criterion would leave it susceptible to savvy users who are aware of the filtering policy or users who unwittingly change the friendly name of the printer. On the other hand, printer driver names are not changeable by the user, making these more reliable.

#### **5.2.5 Printing and Roll Selection**

We then began implementing the portion of the system responsible for printing the rendered stamp images. This portion required the ability to what roll to print from for dual roll DYMO label printers. Printer information up to this point was used purely for our consumption; none of it was modified or written back to the operating system's settings. In order to provide the ability to modify printer settings, we implemented more native mappings to the Windows API functions along with the necessary structures and constants.

Initially, we followed an approach using the *DocumentProperties* native Windows function to attempt to change the printer properties. This function is used to create application specific printer settings. The advantage in using this function is that we will be able to modify printer settings that will not affect user or global default settings. However, this function required postage images to be rendered for print preparation and printed directly using the Windows API. This seemed like too much work for us considering that a PDF print library was already available in Java and used in Endicia's La Poste web page.

The alternative that we chose instead of the above was to modify the user default settings with the *SetPrinter* API function. . The *DocumentProperties* function can also be used for validation of requested printer setting modifications and so was still used but in this capacity instead. Since changing user default settings would be visible to other applications, we made sure that the settings were reverted back to their initial values upon completion of the process. However, other applications could still see the changed settings if they spooled the print jobs at the right time, introducing a race condition. As of now, we are accepting this risk until another solution is available. When changes are made to printer settings at the native level, however, they were not immediately reflected in Java if the Java Print Service API's *PrinterService* objects were already initialized. In order to make the changes visible, we made sure to

refresh the printer information by forcing the *PrintService* objects to initialize again after the native API calls were made.

### **5.2.6 Printer and Job Status Querying**

To ensure that users are given the best possible feedback and providing rich logging details to the server, we included printer and job status querying capabilities in the print engine. While the Java Print Service API provides an interface for retrieving exactly this kind of information [32], we were unable to retrieve detailed information about the printers we were developing with. We chose to again leverage the native print API, using the previously mapped *GetPrinter* call to access printer status information and mapping calls to *EnumJobs* to access print job queue information. According to the Microsoft Developer's Network [link here], printer status information and job status information differs slightly; both are retrieved in our print engine.

However the detailed the status information retrieved, it is still possible that no status information is posted by the printer back to the operating system. The type of information reported back to the operating system by the printer driver is ultimately determined by the implementation of the printer driver. During the implementation of the status retrieval portion of the print engine for DYMO printers, we encountered issues where our applet was not detecting out-of-paper errors. After contacting DYMO driver and firmware developers, we learned that specific conditions must be met before DYMO printer reports out-of-paper errors. As a result of this, we included a portion of code that asks the user if the print was successful or not and presents the option to print the same stamp image again. We followed the example in the La Poste applet and set a limit to the amount of reprint attempts to reduce fraud.

### **5.2.7 Stamp Image Retrieval and Session Sharing**

Stamp image retrieval is currently not implemented. What is currently in place is a stubbed out interface to the server that simulates the functionality of the implemented interface when it is available. From the applet's perspective there is no difference; the applet simply makes the call to the interface and receives a response from the server as a result of the request. The interface will rely on session data passed to it from the applet as a form of customer identification; it will map a customer (session) to their order and return the stamp images corresponding to that customer's order. From the server's perspective, however, an HTTP request from the applet originates from a different location than a standard request from the user's browser, and the server therefore sees the browser and applet requests as coming from two different users. This is because session information is not shared between the browser and the applet. HTTP is a stateless protocol and relies on cookies to provide state from one HTTP request to the next. When the applet is started, we pass session information to it through JavaScript. This way, when the applet formulates a request to the server for stamp retrieval, it can populate the request with the session information from the user's browser so that the server sees the applet's request as coming from the user's browser.

### **5.2.8 Status Feedback**

It was pointed out to us during a review of the user interface that the print engine provided very little feedback during the printing process. During our demo, those in attendance were confused when we pressed the print button and waited without any kind of "waiting" dialog while the print engine was running. In order to eliminate this kind of confusion in production, a collaborative effort was made between both the user interface and print engine team to design a system of dialog boxes in HTML that the print engine could use to let the user know that the print engine was performing a long operation but was still responsive. Specifically, dialog boxes for waiting, errors, and reprinting were made for the print engine to use. We were able to display these HTML dialog boxes through the print engine and provide a seamless experience to the user.

### **5.2.9 Final Refactoring**

With the addition of status feedback, it became evident that the logic for the print engine had outgrown the structure of the original La Poste applet, which we had been using since it was migrated to an applet from a standard Java application. We performed a final refactoring, removing most of the logic from the La Poste applet except what was used for page setup.

### **5.2.10 Logging**

We worked together with the web team to design and implement web interfaces for posting back information from the applet to the backend. Since we want to be able to trace all print attempts, it was important to include these calls so that whenever an error occurs during or after a stamp image request, the backend is able to log as much detail about the error occurrence as possible. The other interface provided was created for reporting printer information to the server. These calls will help both in fraud detection and providing support to the user.

## **Chapter 6 Result**

From the project work, we were able to create and present a tutorial on CSS Grid and JQuery to the Endicia web team. This tutorial will help them understand and connect the web application we created for this project with any future addition to the website.

This project is scheduled for a production release in couple months. Therefore, the result of our project went beyond a prototype or a proof of concept and will be improved and integrated into Endicia's website. The web pages we created were designed so it they can be easily connected to the back end servers of Endicia. During our final phase of integration we removed local references of data and replaced them with calls to the server. These calls to the server will later be fully developed by Endicia and will give our application the information it needs dynamically to function properly.

## **Chapter 7 Conclusion**

We were able to create two prototypes of the application and get the marketing department to review the prototypes. The web pages we created were a good starting point for Endicia to push for a product launch in the near future. The first few weeks of the project were confusing with two sets of schematics to build the web pages from. However, we created both with the set requirements from Endicia without too many problems. It is regrettable that we were not able to see the full integration between the server and the web pages. We do hope that the architecture we left behind for server integration would make the integration easy to perform.

Several things were left unfinished for one reason or another. The web pages were only reviewed by Endicia employees. The application will need a full user test before it can be pushed to production. Before that can occur, the web pages will need more polishing up with real instructions in the web pages. Several elements in the web pages will need to be configured to match the current Endicia web page for full integration. The buy postage pop up contains a payment method box that the users can read to review their payment method before adding money to their balance. The pop up currently has no connections to the server and therefore has no structure set up to show a form to change the payment method.

One of the main goals of the print engine was to explore what kind of printer communication was available from a browser. The main concern was whether or not we could get enough printer information to be able to provide adequate error checking and user feedback. The end result shows that it is indeed possible to print from a browser without the using provided printer dialogues. The print engine development also provided Endicia with a benchmark as to what is needed to achieve communication with a printer. Like the front end, the print engine will serve as a good starting point for web application development for Endicia. The implementation of the Windows API mappings will also serve as an example when other native API mappings are later implemented.

Several portions of the print engine were left unfinished. The printer setup page in the application was not included and will need to be implemented on both the user interface side and the print engine side to allow the user to set up their printer before printing. Currently, the print engine only supports 32bit Windows 7 machines. For the application to be usable by most users, the print engine will need other OS native API support such as for Macintosh OS. Since security is a big part of this project, a code review and security check will have to be performed to ensure that the print engine can stand up to cyber attacks.

We hope that this project left Endicia in a better position than before with a new solution available for a new user base. We certainly enjoyed everything we learned in this project and hope that the people we worked with had benefited from our presence.

## **References:**

1. Adobe, "Adobe ColdFusion Builder" <<http://www.adobe.com/products/coldfusion/cfbuilder/features/>> Date accessed Dec. 7, 2010
2. CFEclipse, "About CFEclipse" <<http://cfeclipse.org/index.cfm/about-cfeclipse/>> Date accessed Dec. 7, 2010.
3. CKSource, "What is CKEditor" <<http://ckeditor.com/what-is-ckeditor/>> Date accessed: Dec. 4, 2010
4. *DYMO Printable Postage*. Computer software. Endicia Inc. 2010.
5. Endicia Inc. "Address Verification" <<http://www.endicia.com/Features/AddressVerification/>> Date accessed: Nov. 20, 2010.
6. Endicia Inc. "Bulk Acceptance Scans" <<http://www.endicia.com/Features/BulkAcceptanceScans/>> Date accessed: Nov. 20, 2010.
7. Endicia Inc. "Business Reply Mail" <<http://www.endicia.com/Features/BusinessReplyMail/>> Date accessed: Nov. 20, 2010.
8. Endicia Inc. "Domestic Mail" <<http://www.endicia.com/Features/DomesticMail/>> Date accessed Nov. 20, 2010
9. Endicia Inc. "DYMO Printable Postage" <<http://www.endicia.com/Features/PrintablePostage/>> Date accessed: Nov. 20, 2010.
10. Endicia Inc. "Email Notification" <<http://www.endicia.com/Features/EmailNotification/>> Date accessed: Nov. 20, 2010.
11. Endicia Inc. "Endicia Developers" <<http://www.endicia.com/Developers/>> Date accessed: Nov. 20, 2010.
12. Endicia Inc. "International Mail" <<http://www.endicia.com/Features/InternationalMail/>> Date accessed: Nov. 20, 2010.
13. Endicia Inc. "Label Customization" <<http://www.endicia.com/Features/LabelCustomization/>> Date accessed: Nov. 20, 2010.
14. Endicia Inc. "Optional Services" <<http://www.endicia.com/Features/OptionalServices/>> Date accessed: Nov. 20, 2010.
15. Endicia Inc. "Return Shipping Labels" <<http://www.endicia.com/Features/ReturnShippingLabels/>> Date accessed: Nov. 20, 2010.
16. Endicia Inc. "Software Postage Log" <<http://www.endicia.com/Features/SoftwarePostageLog/>> Date accessed: Nov. 20, 2010.
17. Endicia Inc. "Stealth Postage" <<http://www.endicia.com/Features/StealthPostage/>> Date accessed: Nov. 20, 2010.
18. Endicia Inc. Telephone Conference. Dec. 3, 2010.
19. Patrick Farry, "RE: Front End Technologies" Emailed to Endicia MQP group. Dec. 6, 2010
20. David Flanagan, "Java in a Nutshell", 2<sup>nd</sup> Edition, 1-56592-262-X, May 1997. Date Accessed, Jan 19, 2011.
21. Java Native Access, "Pure Java access to native libraries" <<https://jna.dev.java.net/>> Date accessed: Jan. 19, 2011.
22. jUSB, "Class HID JavaDoc" <<http://jusb.sourceforge.net/apidoc/usb/linux/HID.html>> Last Modified: Jul. 24, 2002. Date Accessed Dec. 7, 2010
23. The JQuery Project, "JQuery" <<http://jquery.com/>> Date accessed Dec. 4, 2010.
24. Amine Khechfe, "MQP Project Summary for Silicon Valley Project January Team" E-mail to David Finkel. Oct. 4, 2010.
25. Steve McGowan, "HID Point of Sale Usage Tables" USB Implementers' Forum Version 1.02 Mar. 5, 2001
26. Microsoft Corporation, "GetPrinter Function" <[http://msdn.microsoft.com/en-us/library/dd144911\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/dd144911(v=vs.85).aspx)> Date accessed: Feb. 20, 2011
27. Microsoft Corporation, "Microsoft Visual Studio 2010 Express" <<http://www.microsoft.com/express/Web/>> Date accessed: Dec. 4, 2010

28. Microsoft Corporation, “OpenPrinter Function” <[http://msdn.microsoft.com/en-us/library/dd162751\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/dd162751(v=vs.85).aspx)> Date accessed: Dec. 4, 2010
29. Microsoft Corporation, “Windows API Reference”, < [http://msdn.microsoft.com/en-us/library/aa383749\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa383749(v=vs.85).aspx)> Date Accessed: Jan. 19, 2011.
30. MonTimbrenLigne, “Buy Stamps” <<https://www.montimbrenligne.laposte.fr/>> Date accessed Nov. 24, 2010.
31. Moxiecode Systems, “TinyMCE – JavaScript WYSIWYG Editor” <<http://tinymce.moxiecode.com/>> Date accessed: Dec. 4, 2010
32. Oracle, “Registering for Events” <<http://download.oracle.com/javase/1.4.2/docs/guide/jps/spec/printing.fm5.html>> Date accessed Feb 24, 2011.
33. Oracle, “Understanding Signing and Verification”, <<http://download.oracle.com/javase/tutorial/deployment/jar/intro.html>>, Date Accessed: Jan. 19, 2011.
34. OWASP, “OWASP Top 10 for 2010” <[http://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)> Last Modified: Nov. 9, 2010. Date Accessed Dec. 7, 2010
35. Nathan Smith, “960 Grid System” <<http://960.gs/>> Date accessed Dec. 4, 2010
36. Quirksmode.org, “keydown, keypress, keyup” < <http://www.quirksmode.org/dom/events/keys.html>> Date accessed: Feb 18, 2011.
37. Stamps.com Inc. “Plans” <<http://www.stamps.com/postage-online/plans-comparison/>> Date accessed: Nov. 20, 2010.
38. Stamps.com Inc. “NetStamps” <<http://store.stamps.com/Store/support/faq/about/netstamps.jsp>> Date accessed: Nov. 20, 2010.
39. United States Postal Service, “Print Shipping Labels” <<https://sss-web.usps.com/cns/landing.do>> Date accessed: Nov. 20, 2010.
40. Usability.gov, “Scrolling and Paging” < <http://www.usability.gov/pdfs/chapter8.pdf>> Date accessed: Feb 18, 2011.
41. W3Schools, “AJAX Tutorial” <<http://www.w3schools.com/ajax/default.asp>> Date accessed: Feb 24, 2011.
42. W3Schools, “SOAP Tutorial” <<http://www.w3schools.com/soap/default.asp>> Date accessed: Feb 24, 2011.
43. WYSIWYG Web Builder 7, “The Web Design Solution that makes Building Web Sites Easy and Fun!”<<http://www.wysiwygwebbuilder.com/>> Date accessed: Jan 15, 2011.

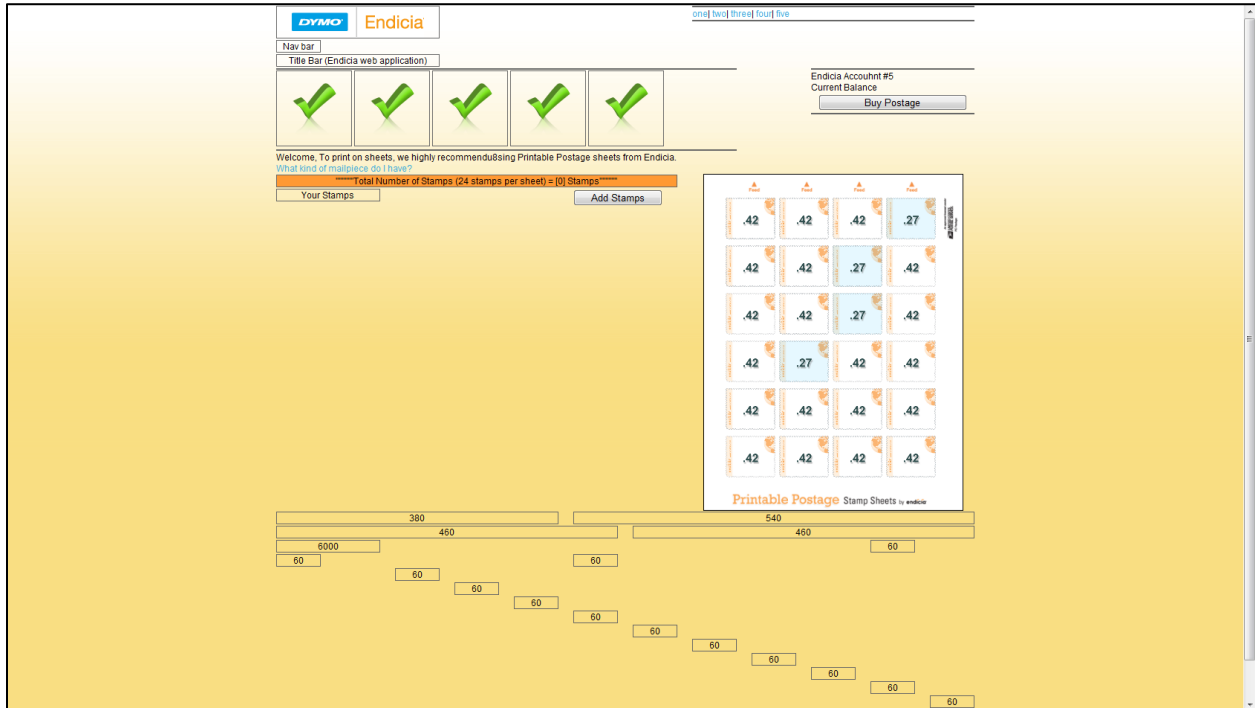


## **APPENDIX A**

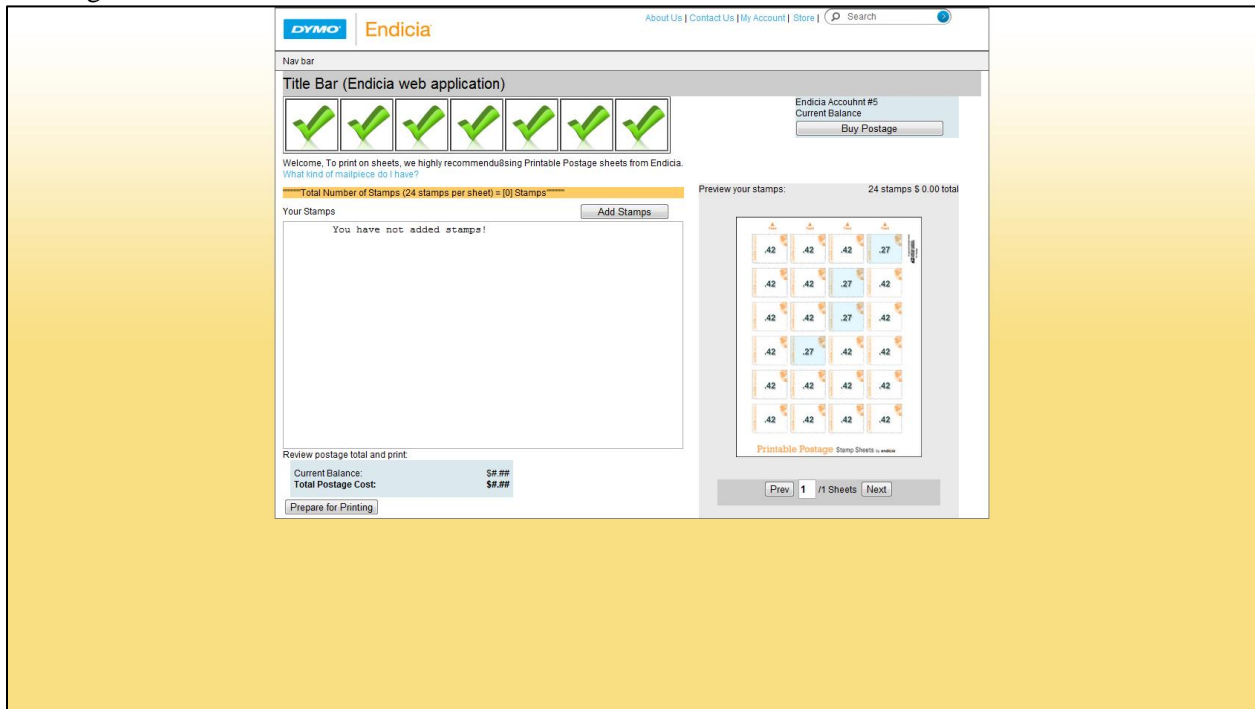
See attached zip file “CSSGrid-JQuery” for the presentation and all associated materials.

# APPENDIX B

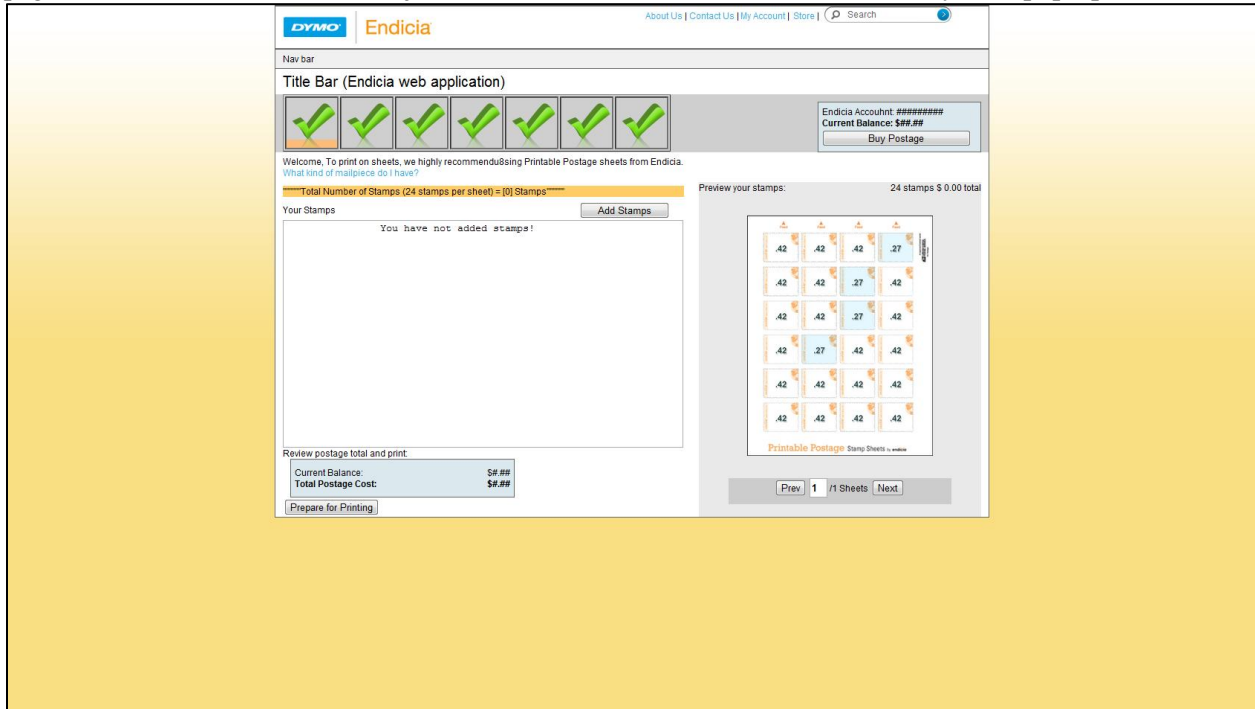
## B-Sheet Printing



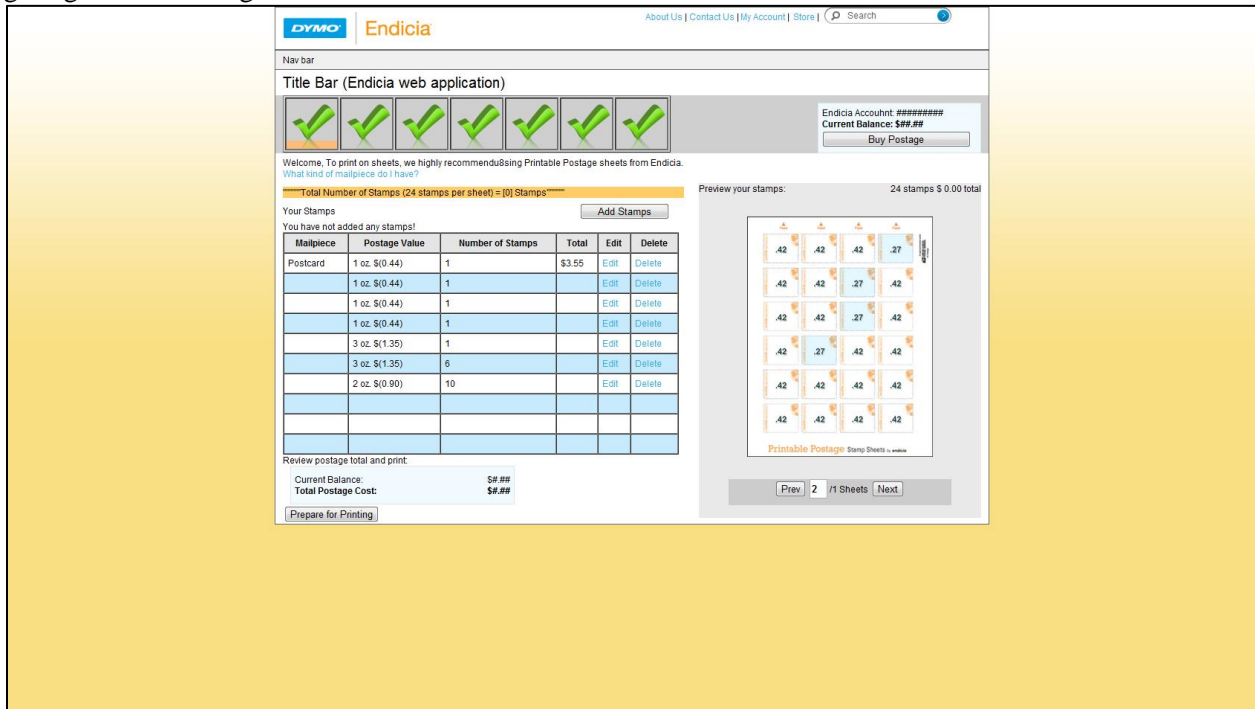
Revision 27, 1-7-11: Here is the creation of the page. We were creating the page out of a 960 CSS grid demo file and left in some of the demo grids as references at the bottom of the page. There were no working links or buttons at this time.



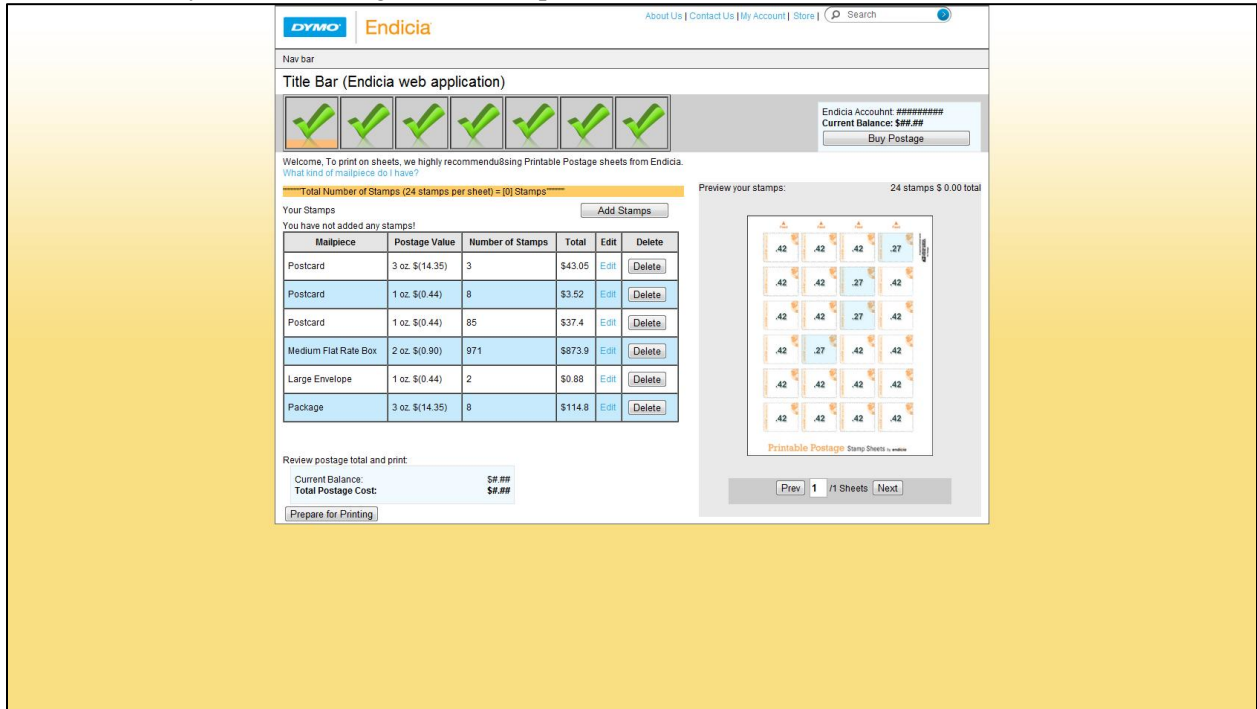
Revision 34, 1-11-11: Here we finished the structuring of the page using 960 grid coming very close to what the wireframe actually looked like. We started putting in some placeholder images and coloring the page to fit the wireframe. We had just started to add button and link functionality with pop-ups.



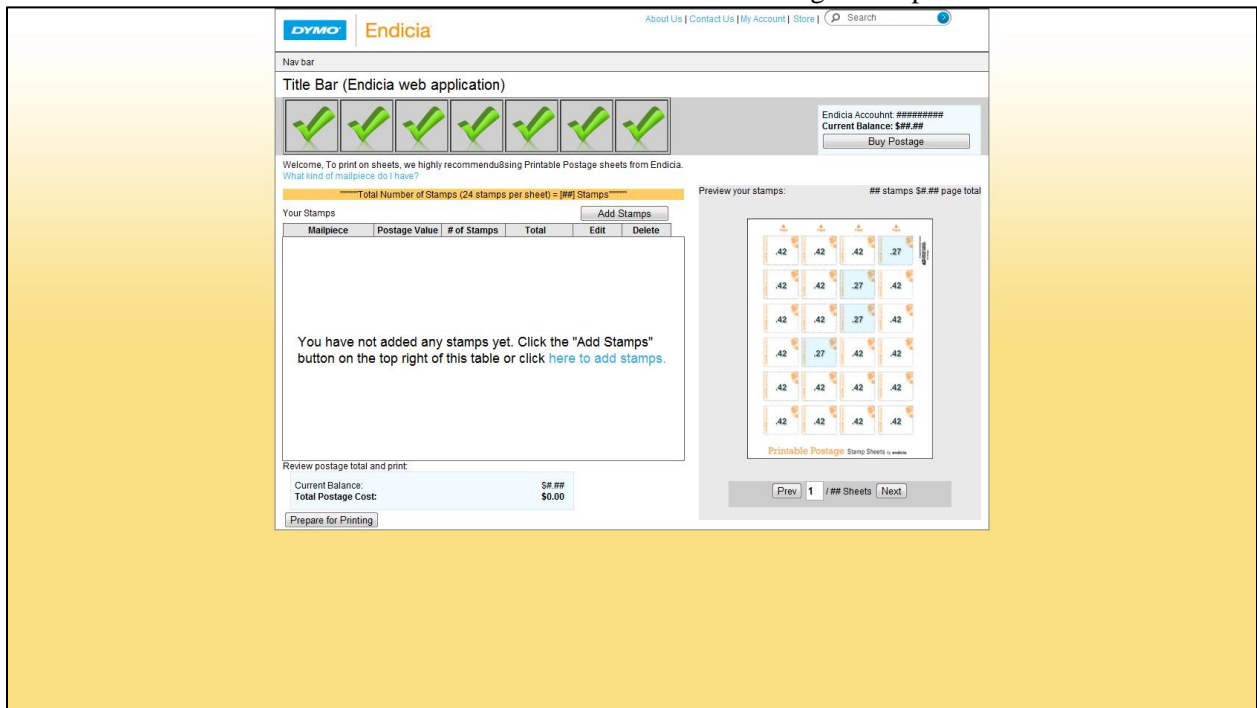
Revision 38, 1-12-11: More correct coloring and structuring went into this version as we started to space elements correctly. Here all three pop ups (buy postage, add stamps, and prepare for printing pop ups) show up correctly with black overlay to prevent the user from clicking on the element underneath. The application navigation bar (all the checkmarks) also had the added functionality of selecting them and giving them an orange underline.



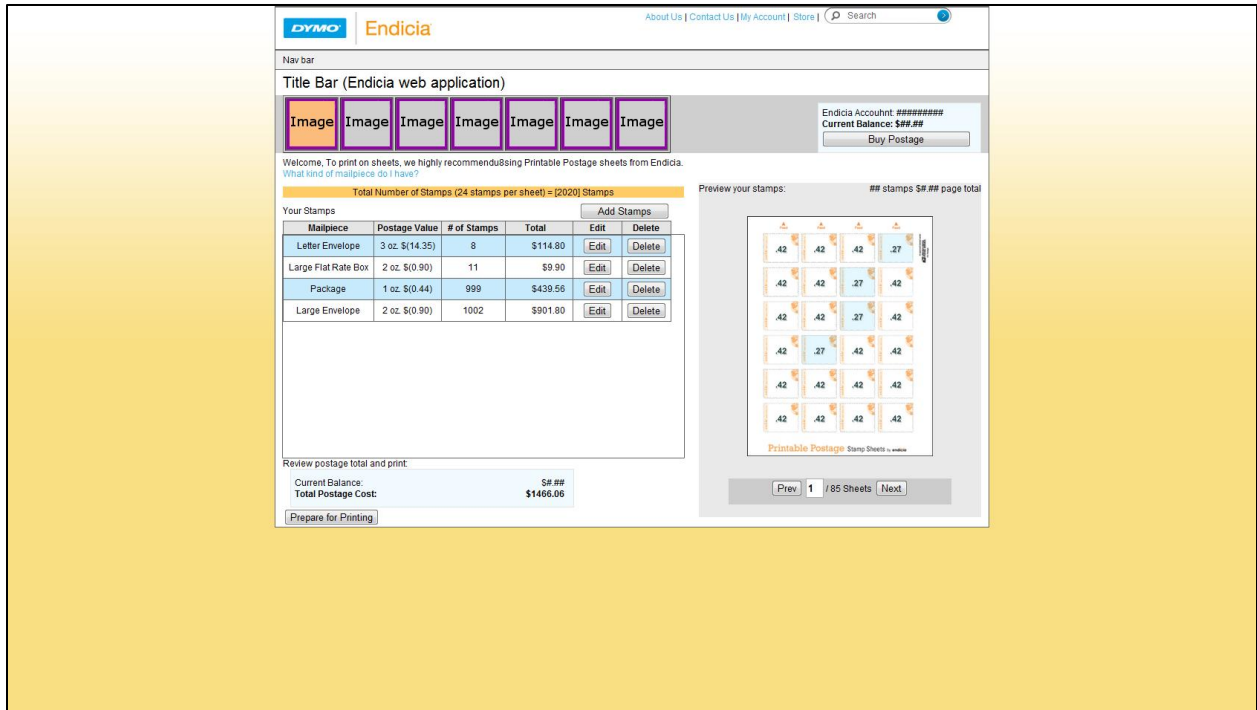
Revision 40, 1-13-11: Here we began adding the functioning table for holding stamp orders. We could get most information from the add stamps pop up window. The later to be buttons for edit and delete for each row were merely malfunctioning links at this point in time.



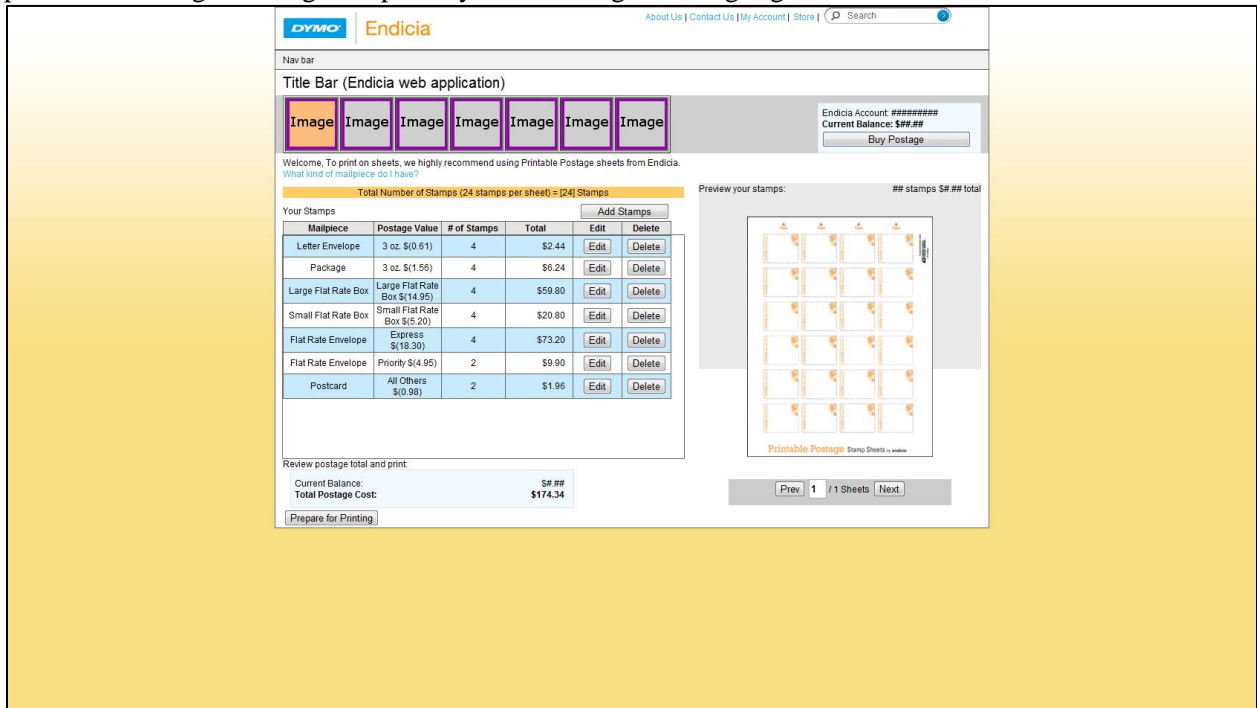
Revision 43, 1-14-11: The table was now functioning properly for the most part. We could get all the information we needed after adding stamps from the popup. We had working delete buttons that got rid of entire rows. The calculation for the total cost for the row was working at this point.



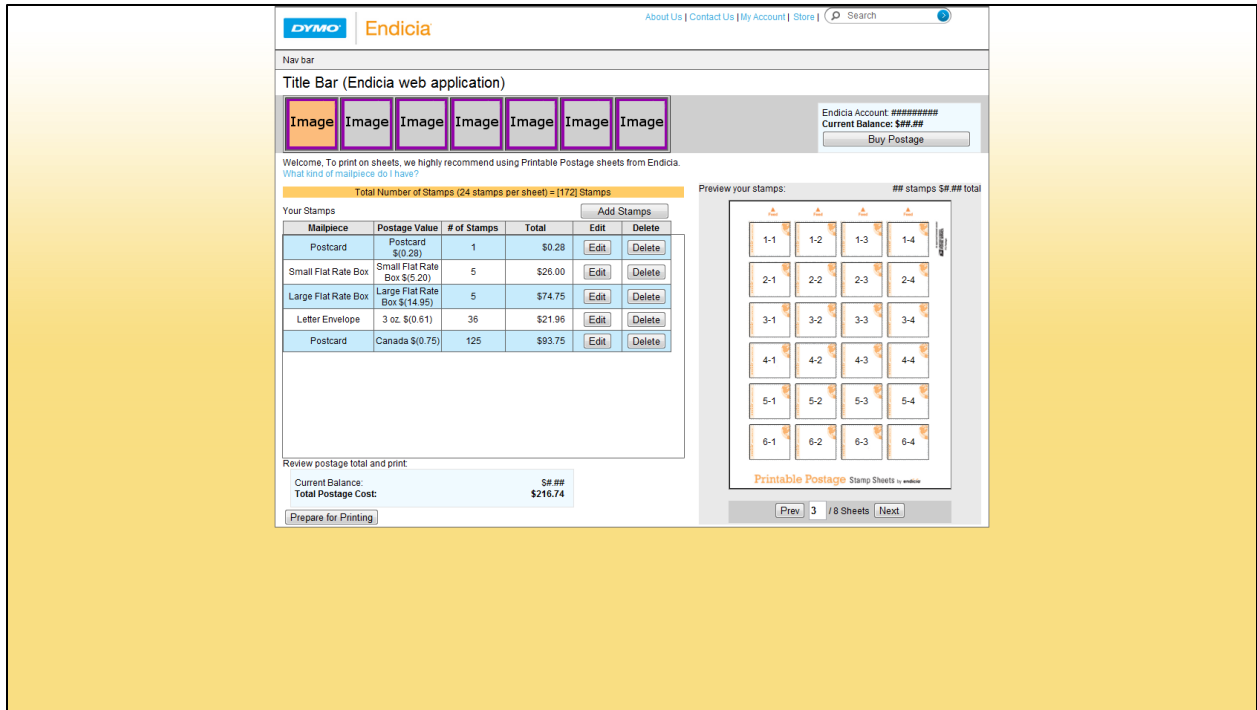
Revision 46, 1-19-11: Here we had a large refactoring of the code making it more compatible with all browsers except for IE and changing the structure internally, resizing elements and table components. We were now calculating total postage cost of all elements in the table and had the empty table dialogue.



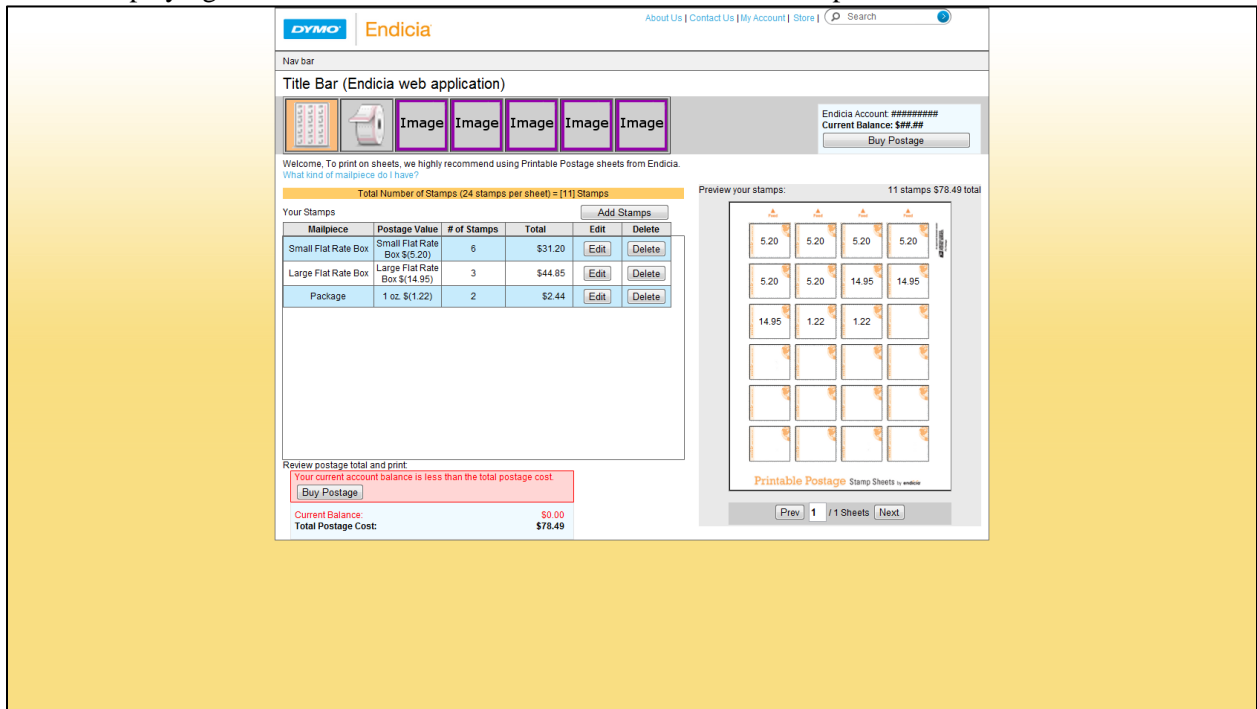
Revision 55, 1-20-11: Here we had a fully functioning table with working edit and delete buttons. We are also calculating the total number of stamps on the page as well as displaying the number of pages that is necessary to print the requested stamp order. The checkmarks have been replaced with handmade placeholder images testing transparency of the background highlight.



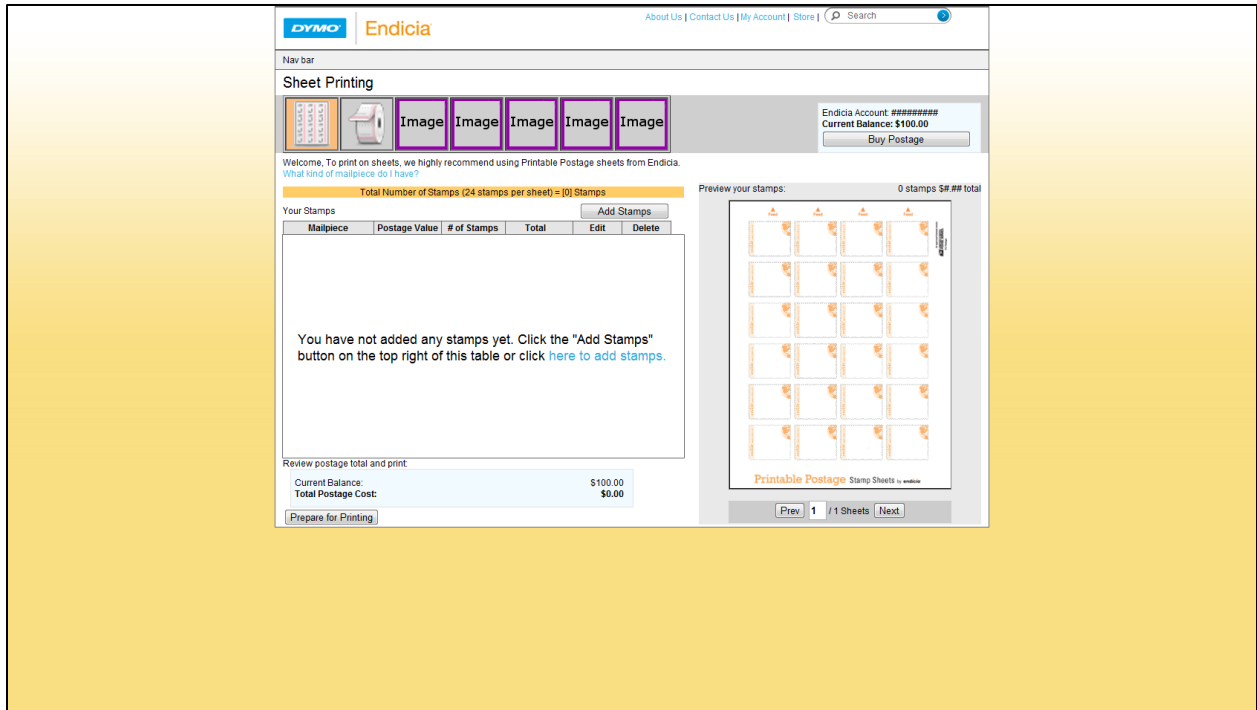
Revision 81, 1-21-11: Here we were just beginning work on the preview by creating a blank sheet image to later place numbers on. We were also getting all of the unique postage values from the improved add stamps pop-up.



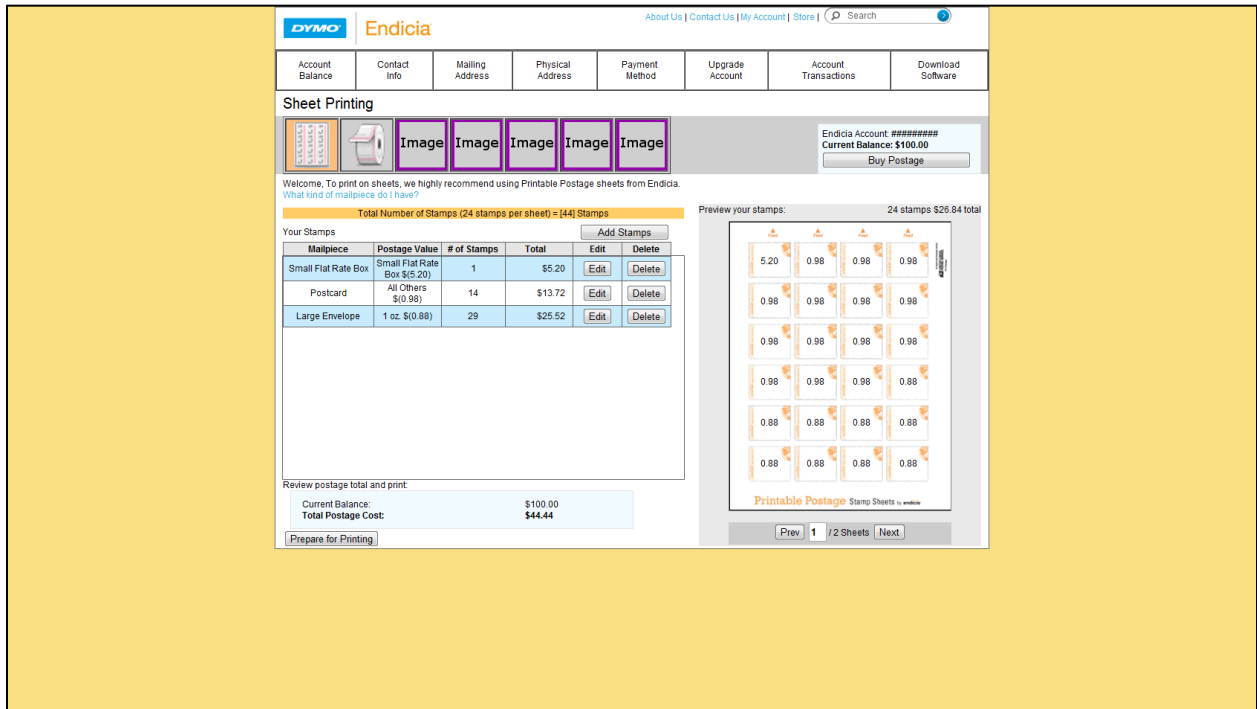
Revision 92, 1-22-11: Here we are successfully created a table that matched the cell on the image exactly. We are displaying text to our blank sheets but without the correct stamp values.



Revision 105, 1-23-11: Here we have a working warning message that is displayed when the value of the requested order is more than the current balance. While the warning message is displayed it disables the printing button. The preview is now fully functioning with the option to view different preview pages.

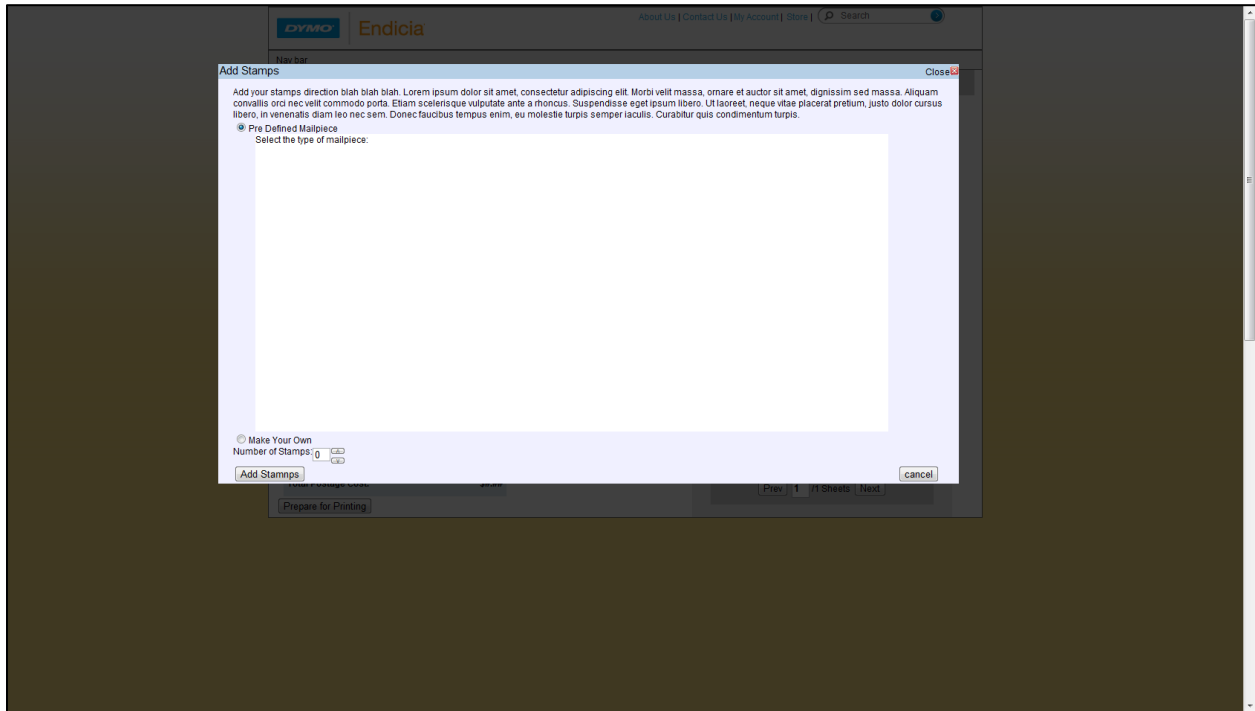


Revision 129, 1-26-11: At this point we did a lot of bug fixing that mainly had to do with our input fields allowing large values that were very excessive and caused problems with our table. We also stopped the users ability to add stamps if they had any incorrect fields. We added a max and min to the number spinners for the preview and number of stamps as well as gave the application a set current balance of \$100.

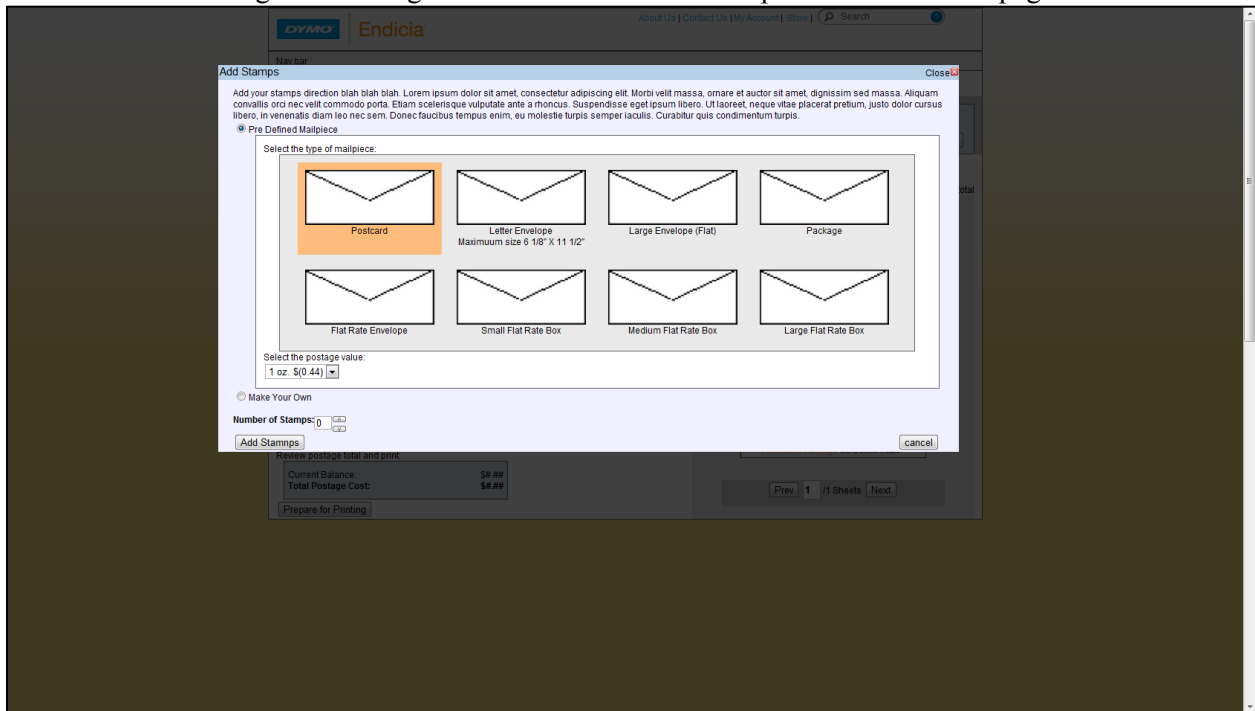


Revision 144, 1-28-11: We updated the code to work with Internet Explorer. Float CSS property has been switched to grid because IE cannot work with Float. The navigation bar has been added and it also contains the correct buttons for account management. After this we went over to creating the application following the new visuals.

## Add Stamps Pop Up

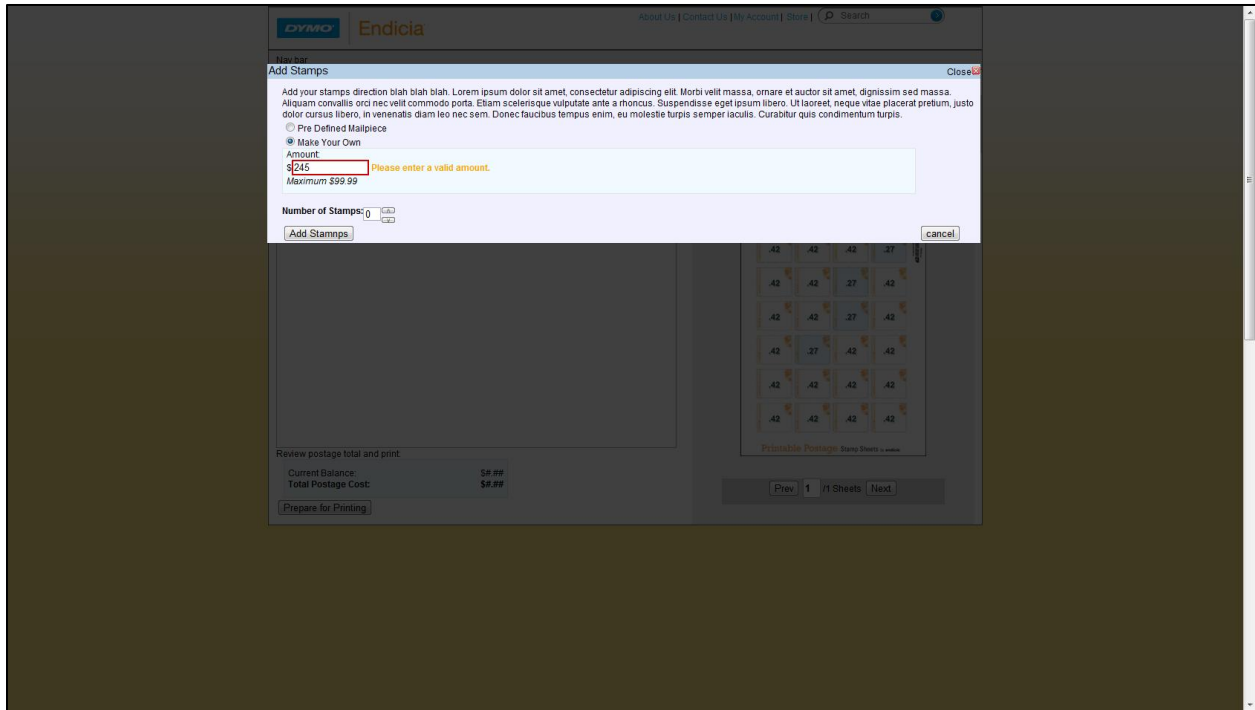


Revision 34: The creation of the Add Stamps pop-up window. There is an operational number spinner with no error checking and clicking either “cancel” or “add stamps” would close the page.

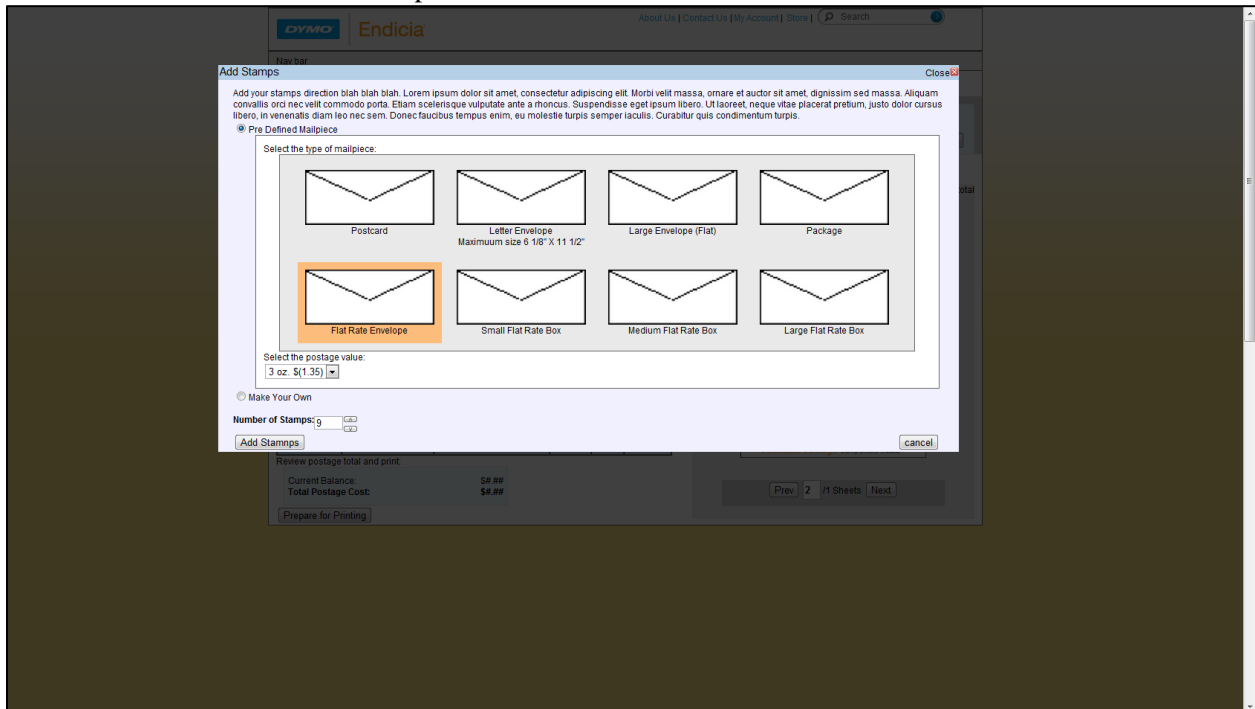


Revision 38: We could now visually select a mail piece type and drop down postage value but it still had no effect on the main page. The number spinner now had rudimentary error checking on its value so user cannot type in characters.

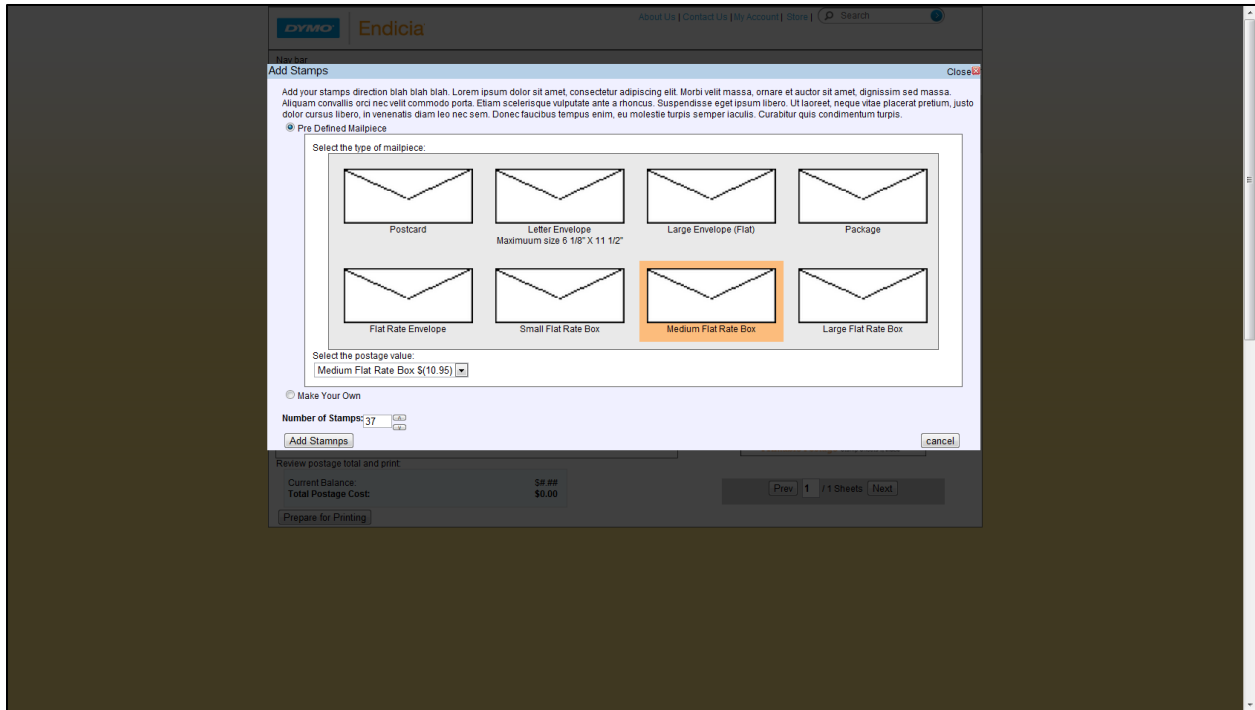




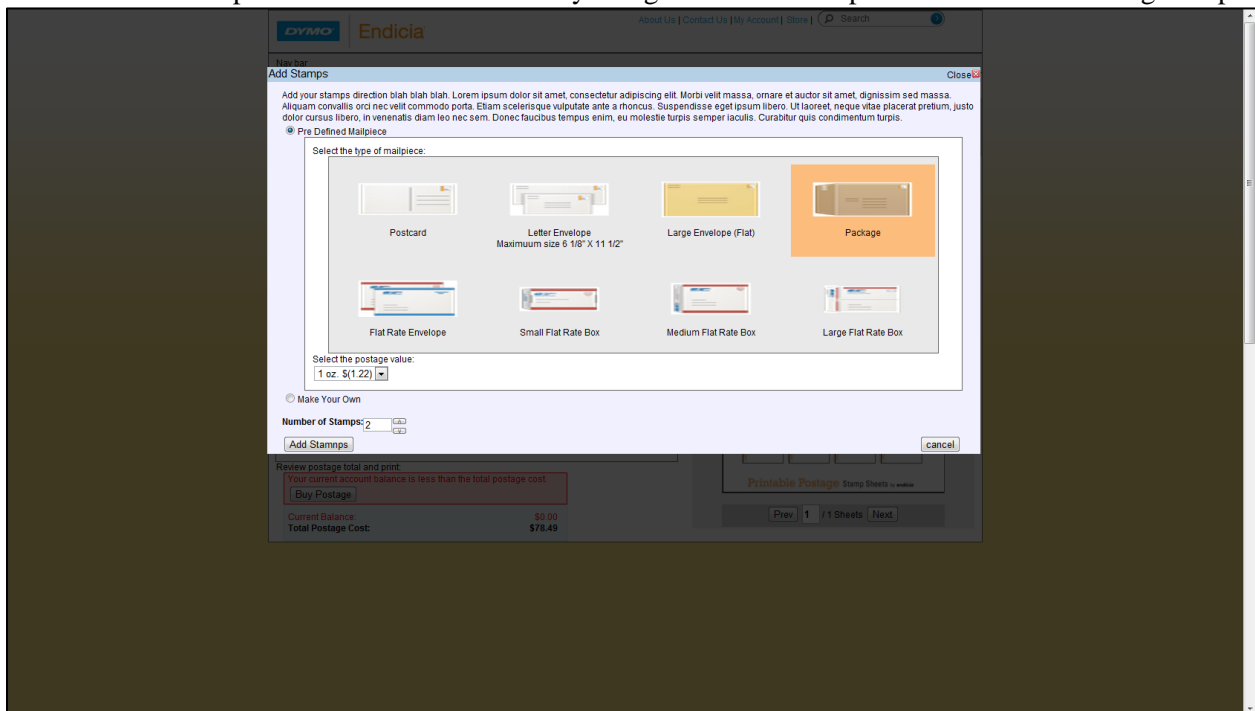
Revision 38: Here we have the make your own stamp amount input field. There is full regular expression checking on the value so we will only allow proper money amounts up to \$99.99 and display a warning when the field detects an invalid input.



Revision 40: We now had selectable mail piece types and could send our information to the main page to be added to the order table.

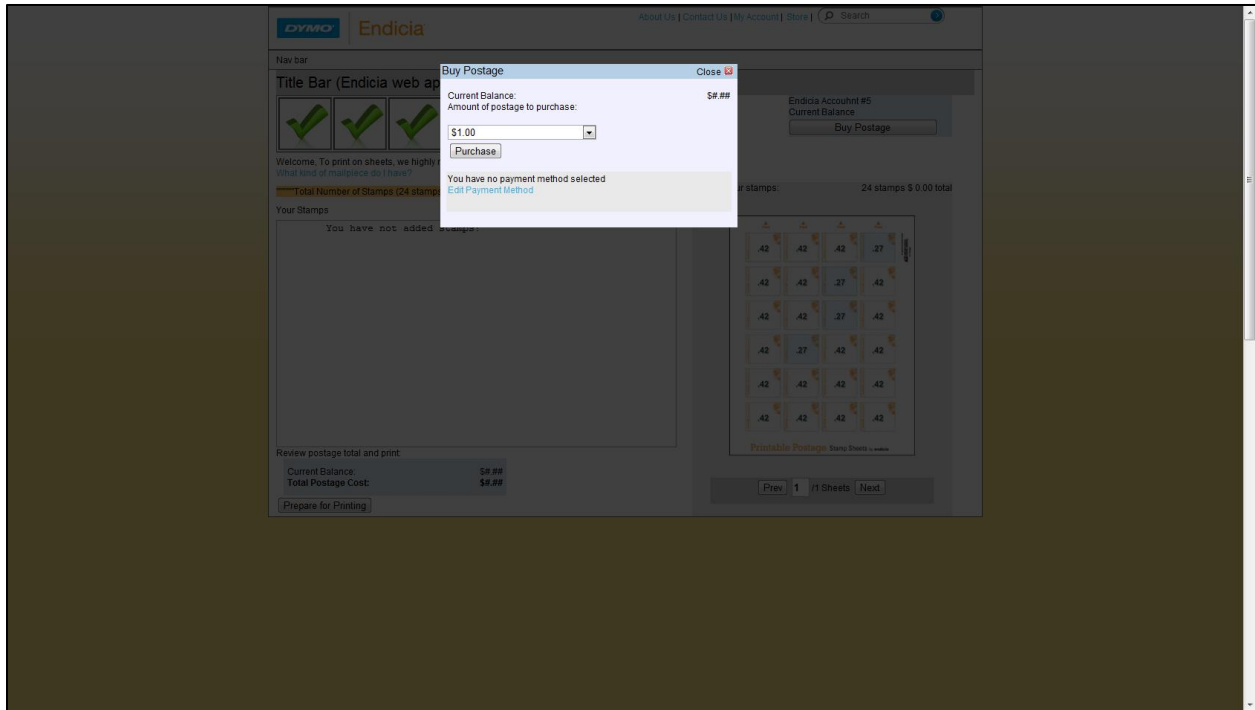


Revision 81: Now we had unique drop down menus for every selectable postage types. These unique values from the drop down box were successfully being sent to the stamp order table when adding stamps.

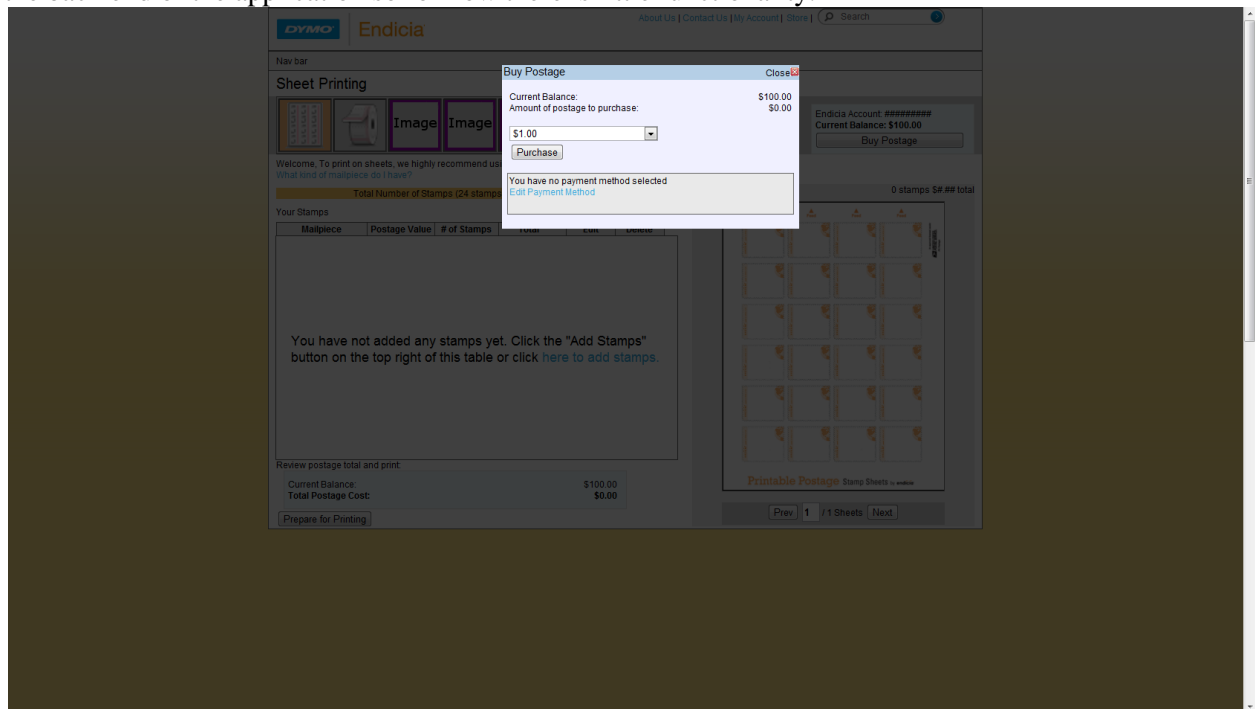


Revision 105: Here we added some final touches such as pictures and element padding. We also added better error checking for both the number spinner and custom postage input box.

## Buy Postage

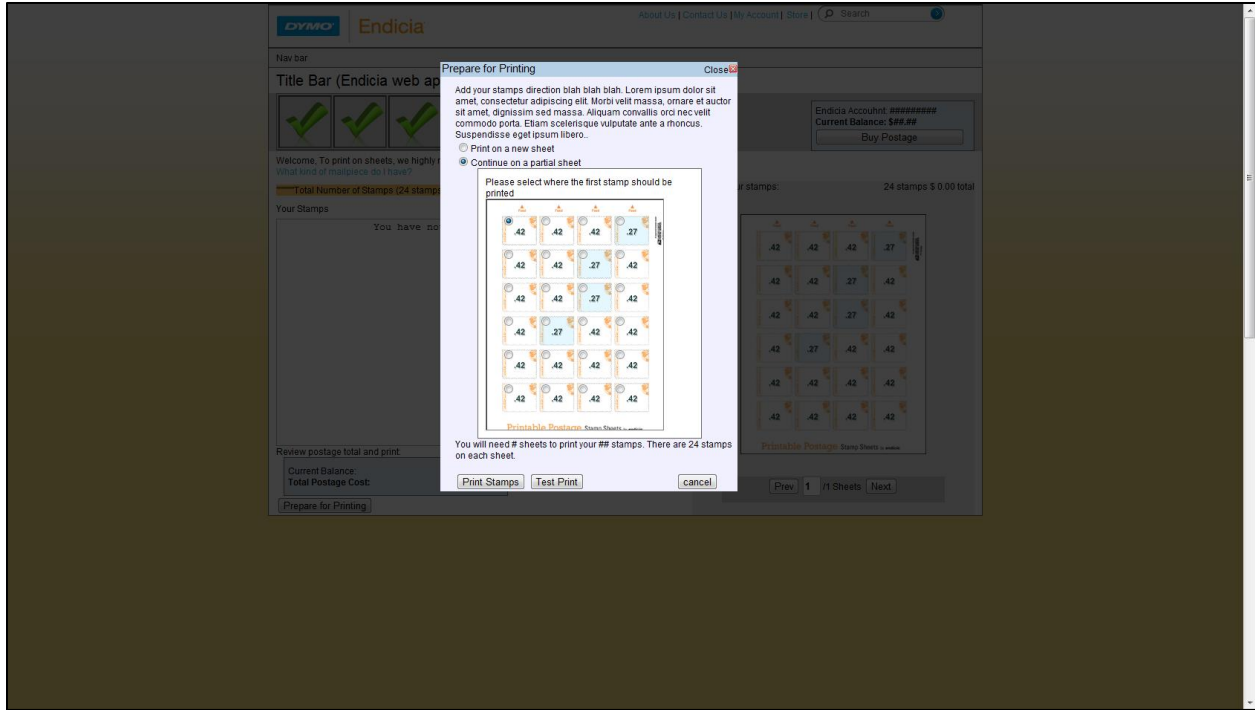


Revision 34: The creation of the Buy Postage pop-up. This pop-up will eventually connect directly with the back end of the application so for now there is little functionality.

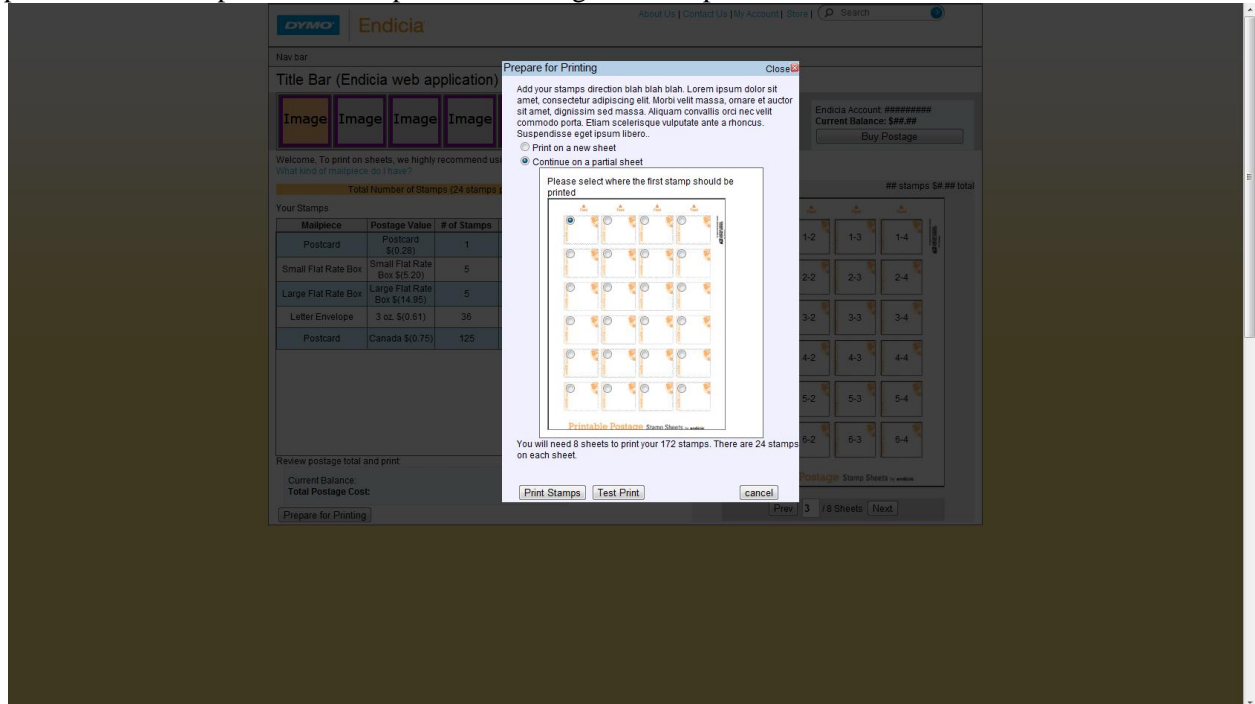


Revision 129: The Buy Postage pop-up now has both the current balance (although it does not change) and total amount of postage to purchase.

## Prepare for Printing

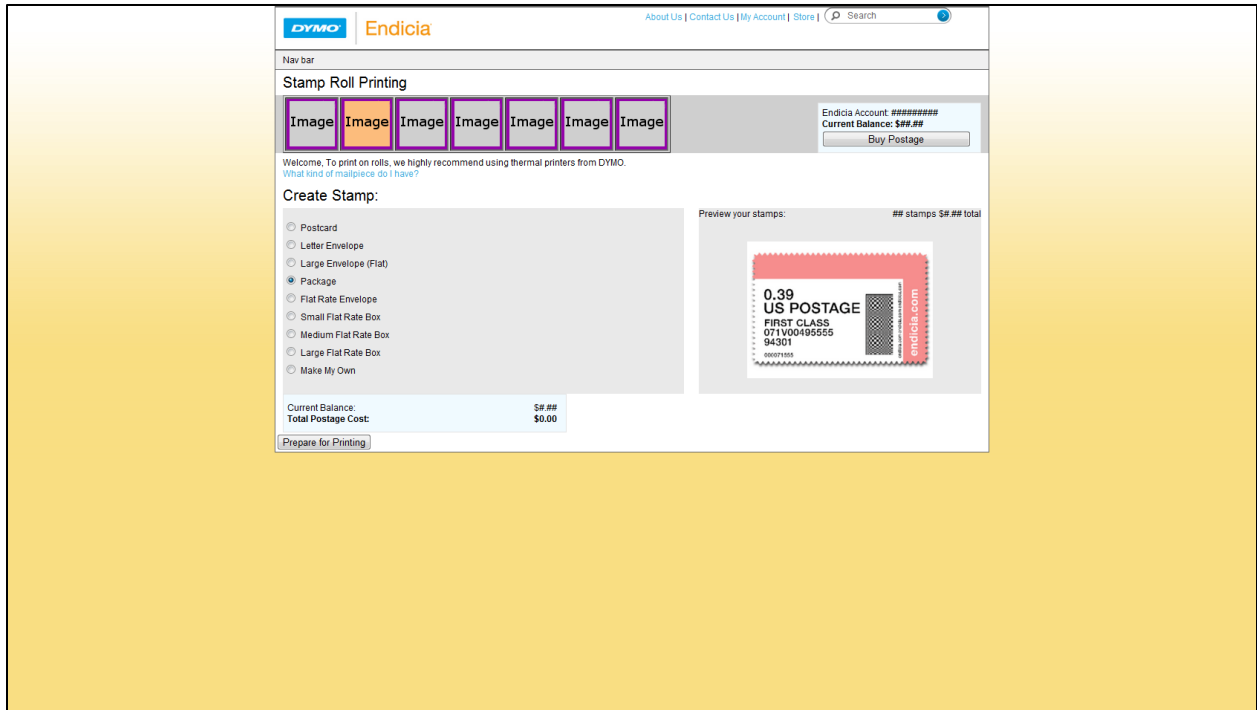


Revision 38: Prepare for Printing pop-up was created. Here the user can decide on a new sheet or select the starting location on an already used sheet. At the moment this pop-up does not interact with any printer. It shows up when the Prepare for Printing button is pressed.

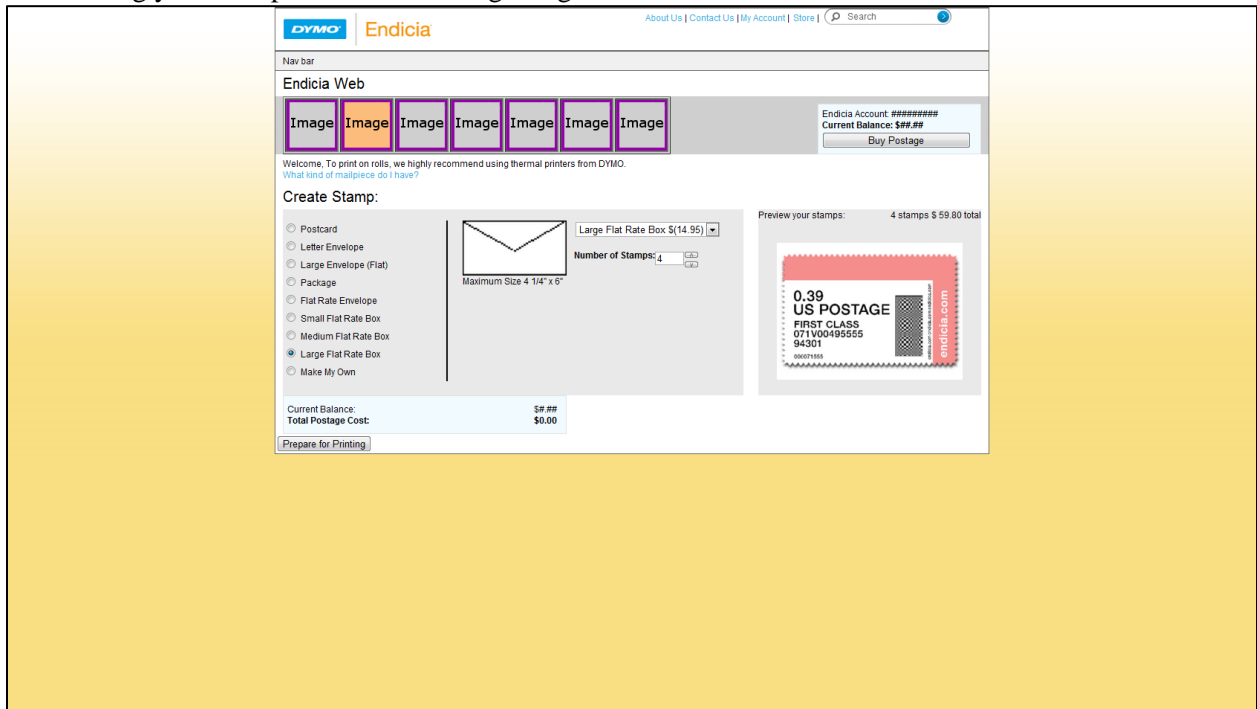


Revision 92: Here the Prepare for Printing pop-up gets information from the main page and displays the total postage cost of the order as well as the number of stamps and how many sheets it will take to print this order.

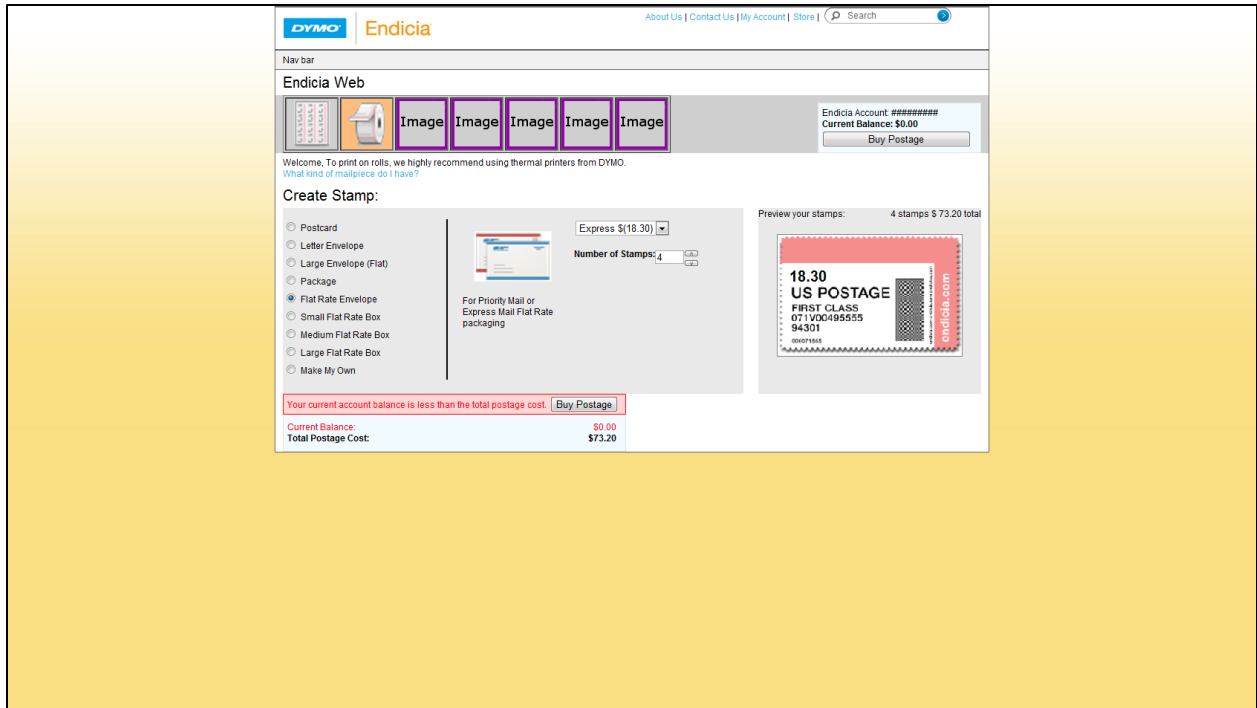
## Roll Printing



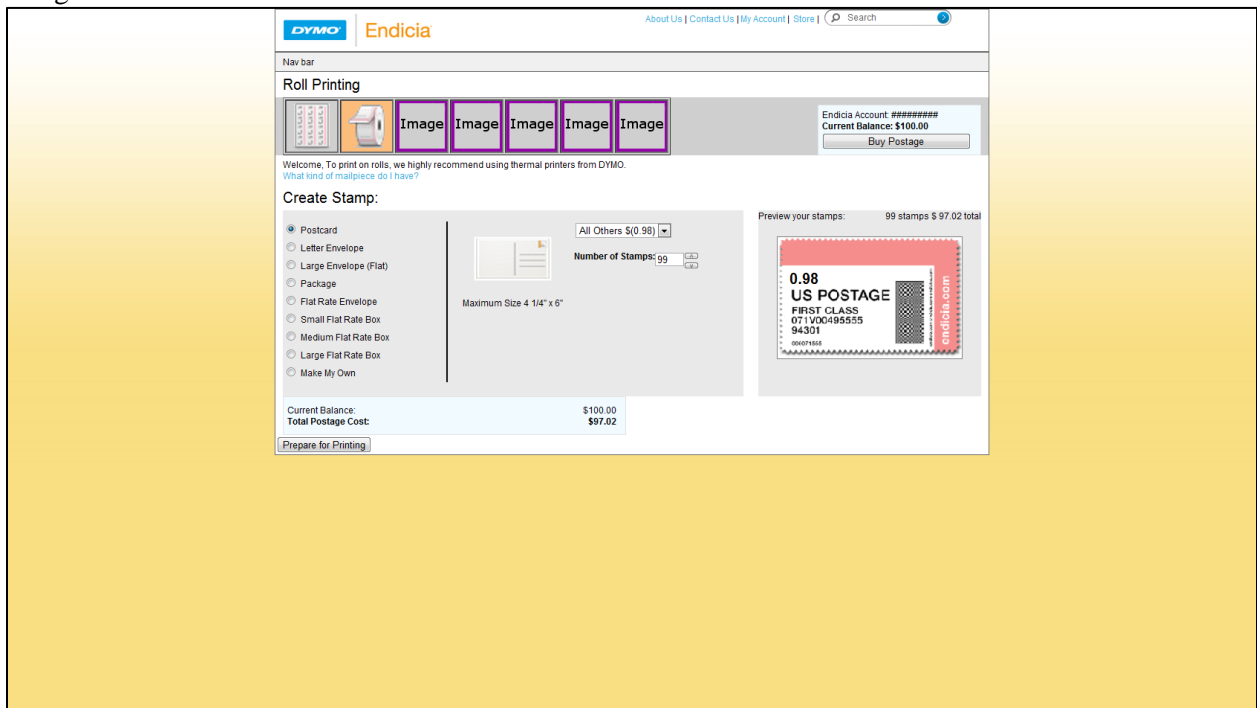
Revision 81: The creation of the roll printing page was able to carry over a lot of the format and functionality of the sheet printing page. All buttons linked to pop-ups and the very beginnings of customizing your stamp order was coming to light.



Revision 92: Here we can select between mail piece types and even create our own custom postage. The total cost is calculated above the stamp preview. The preview does not change to reflect the selected type of postage.

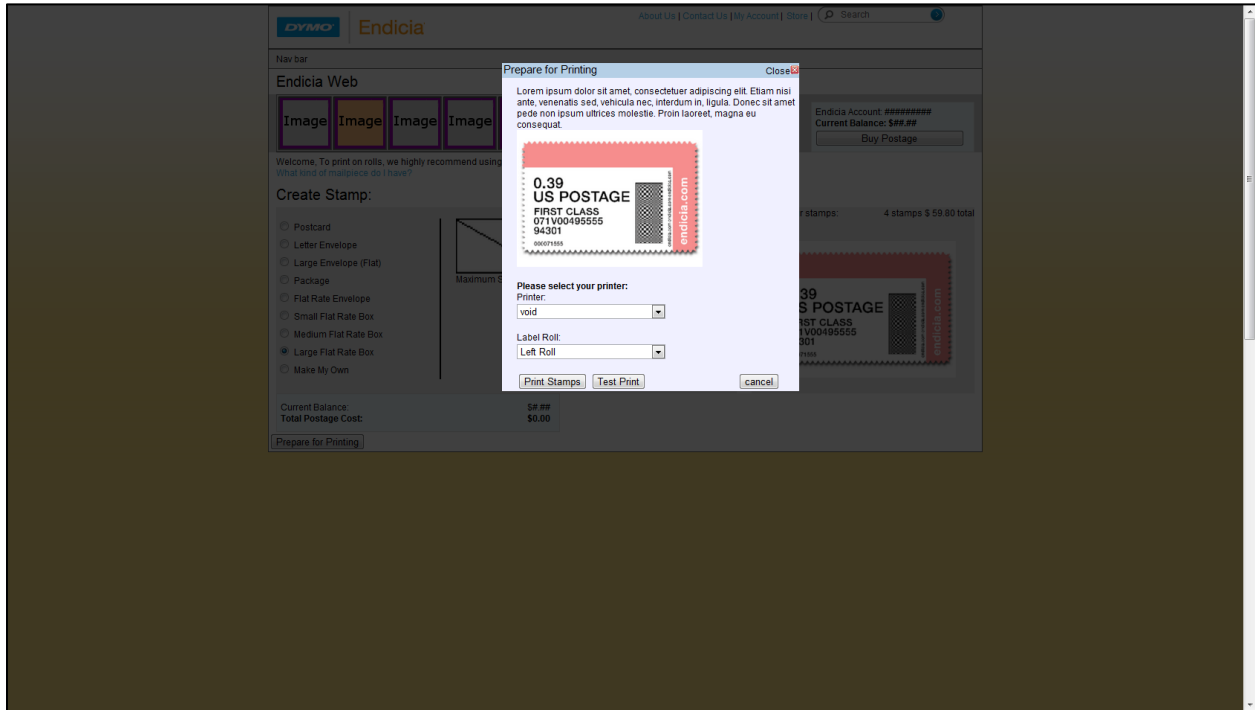


Revision 105: The preview of the order stamp now changes in value to reflect the chosen mail piece type. A warning message also shows up and blocks printing if the ordered amount of stamps would be larger than the current balance on the account. Images have been added for the selected mail piece types and the navigation bar.

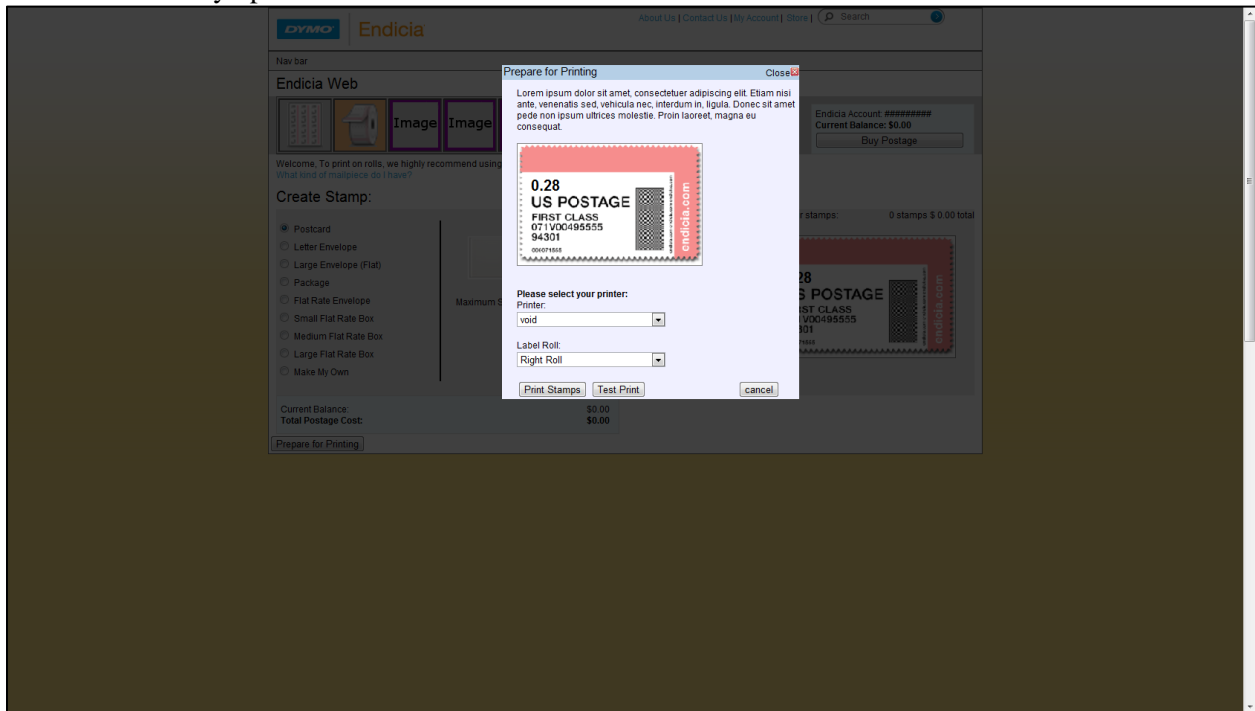


Revision 129: Here we have given the application a set current balance as well as fixed many errors concerning the input fields. Here we see that the maximum number of stamps in an order is 99. The same constraints added to the input fields in sheet printing have been added here in roll printing.

## Prepare for Printing Roll

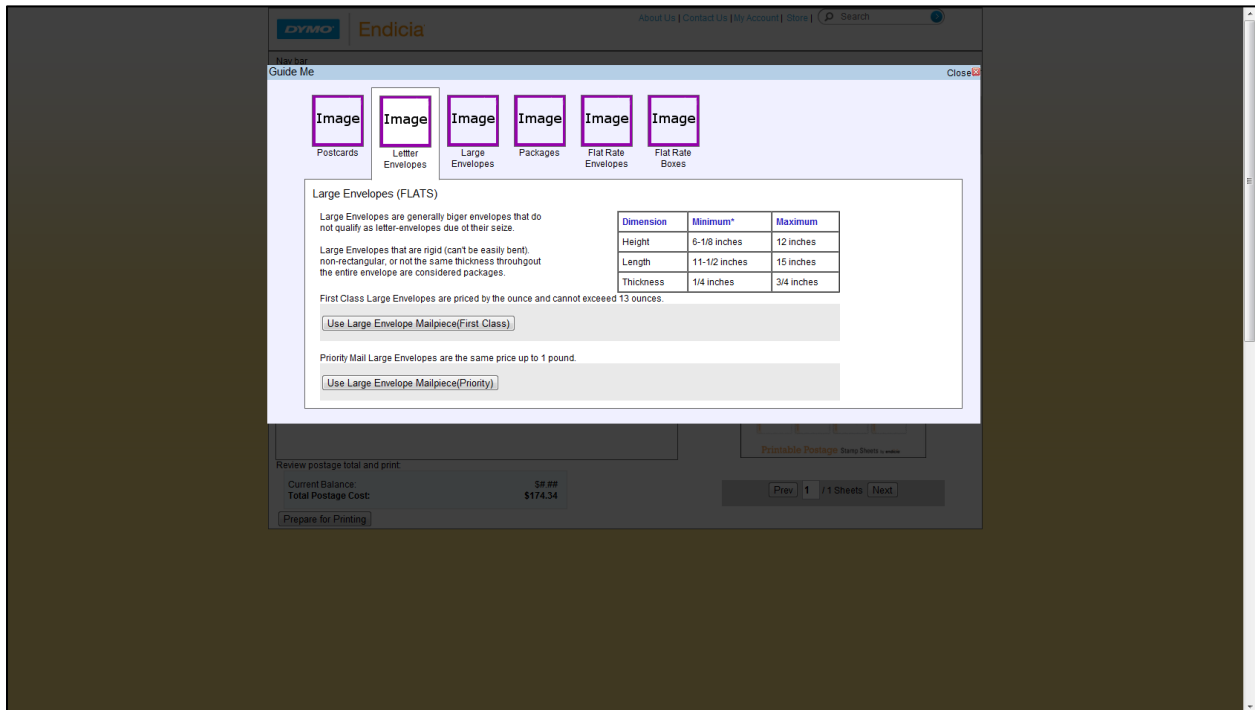


Revision 92: The creation of the Prepare for Printing stamp roll pop-up dialogue box. This is where the user would set up their DYMO stamps thermal printer before printing their order of stamps. At this point the user could only open and close the box.

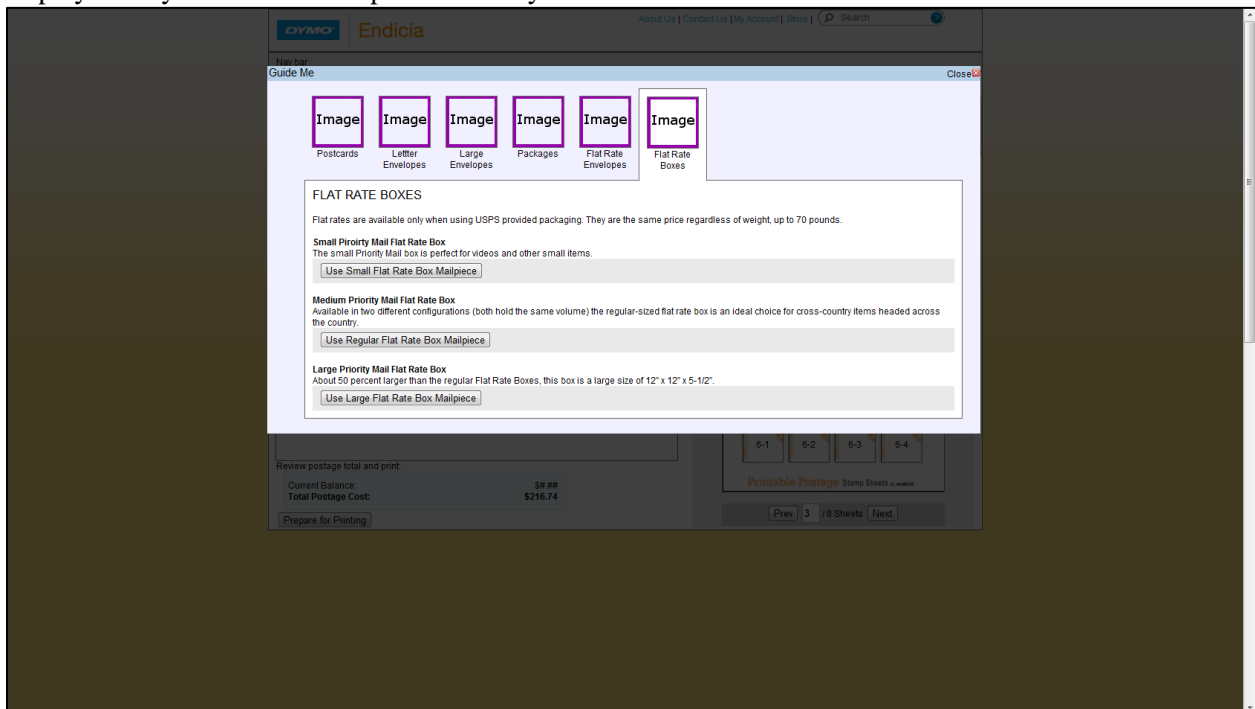


Revision 105: With this version we were displaying the same stamp preview image that we display on the main roll page. Button functionality remains the same.

# Guide Me

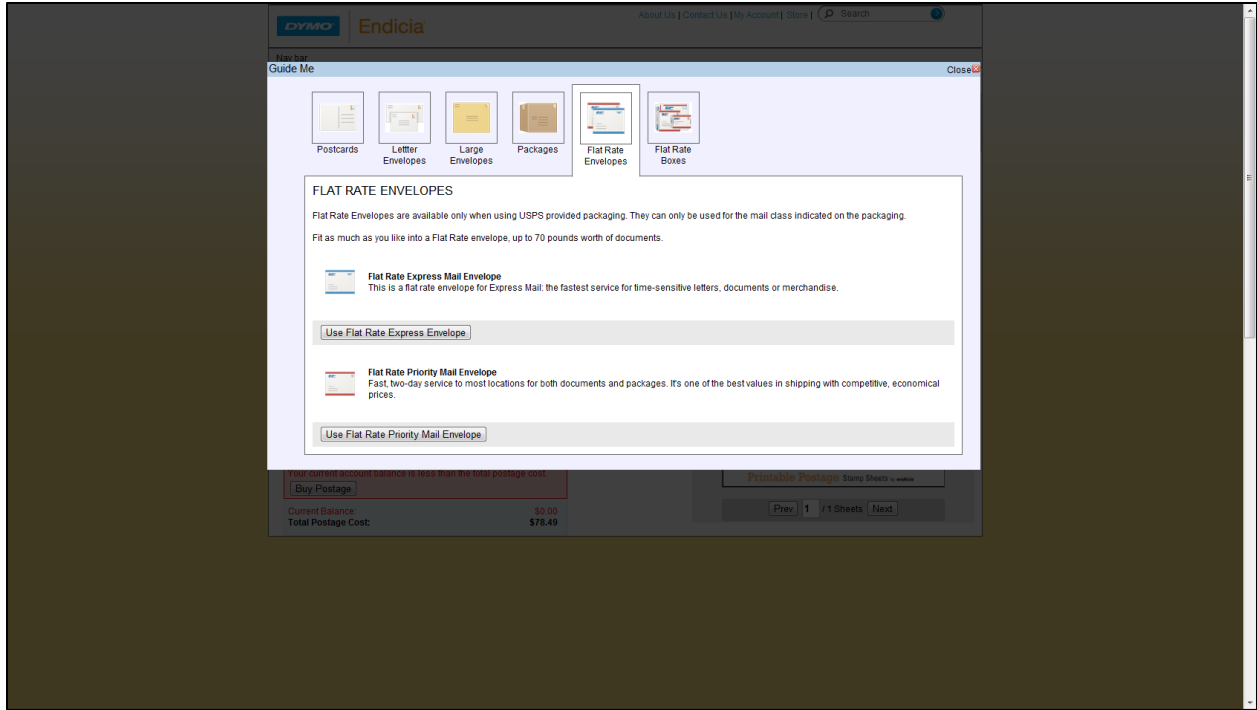


Revision 81: The Guide Me pop-up was created and the postage types can be selected but information is displayed only for letter envelopes as we only had that information at the time.



Revision 92: Here we have inserted the information for all postage types as well as buttons that link to the add stamps pop-up window so the appropriate fields are showed to the user.

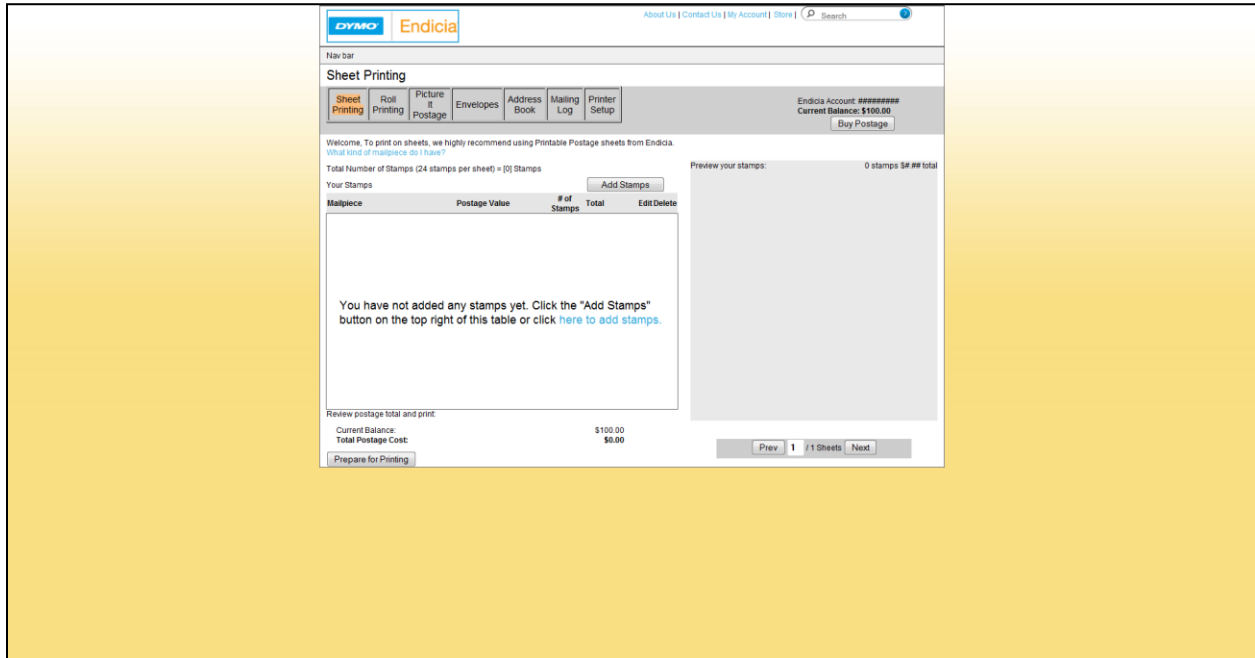




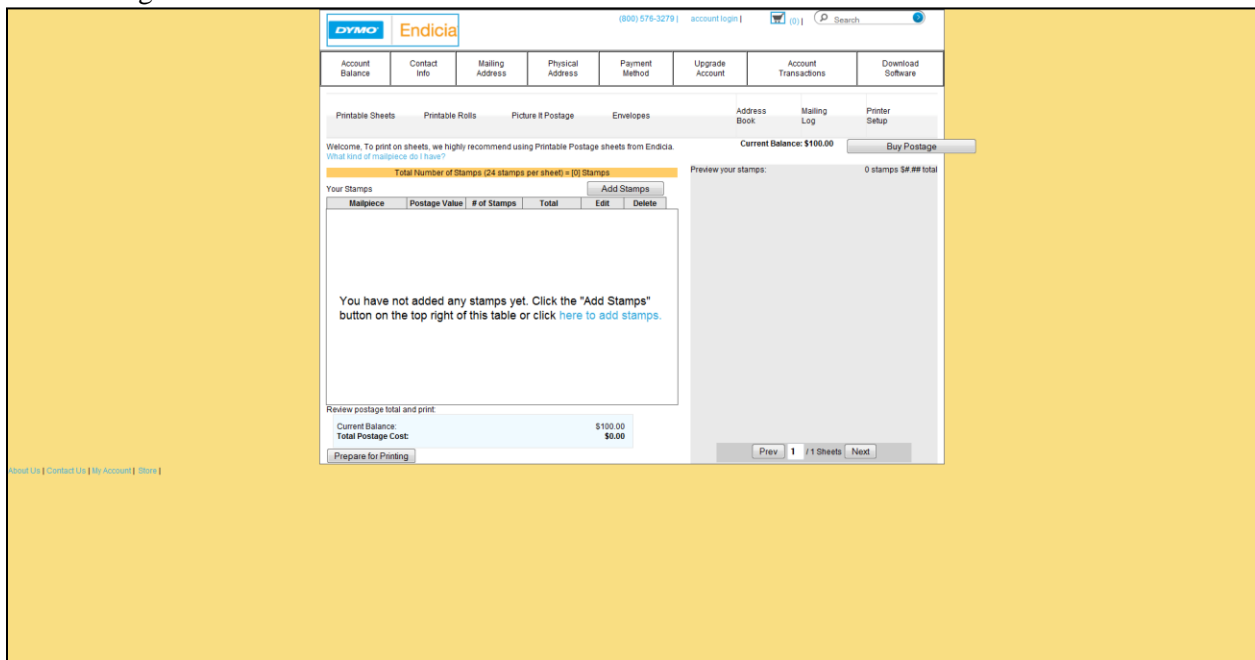
Revision 105: We have now added pictures as well as nicer formatting to help with overall presentation of the information to the users.

# APPENDIX C

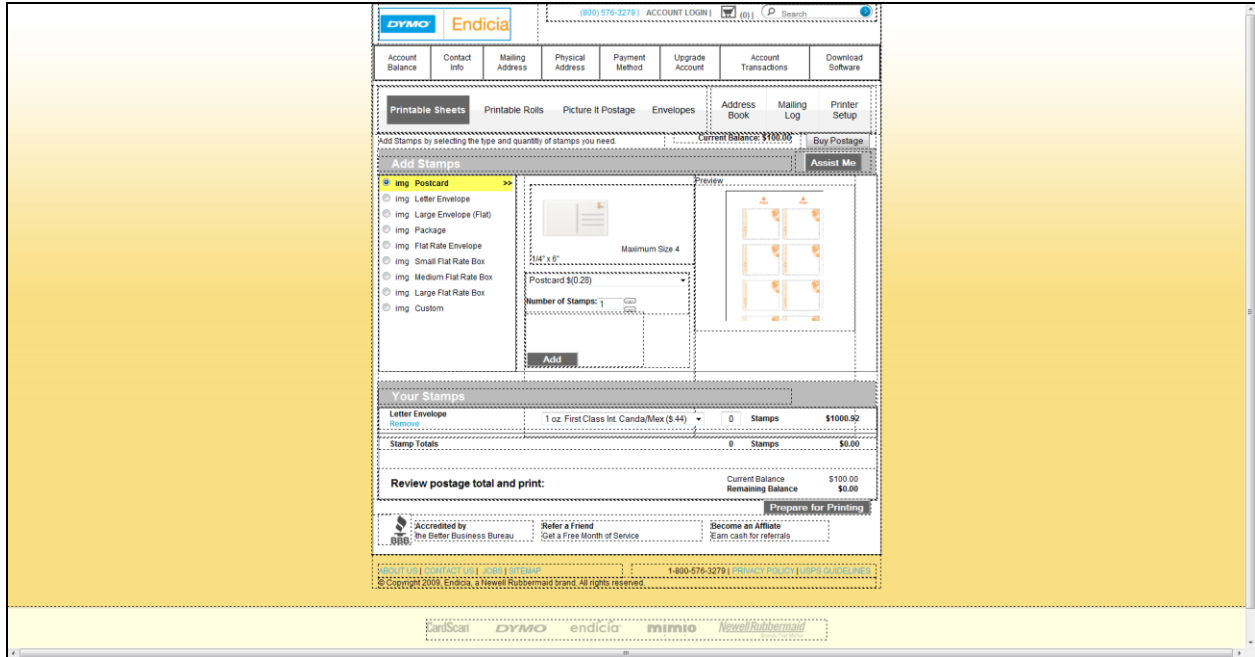
## Sheet Printing



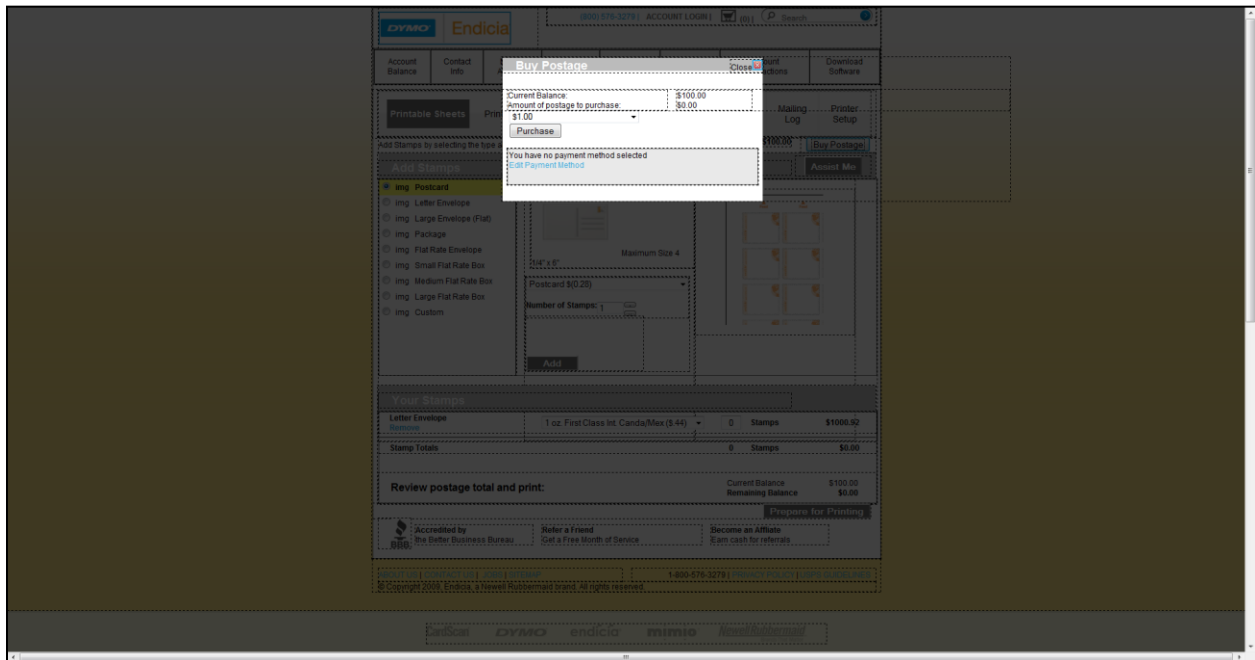
Revision 130, 1-27-11: The page is slowly converted to the new visual. The now old page was moved to a new folder and the images slowly migrate to the new folder with updated links in the HTML to clean up unused images.



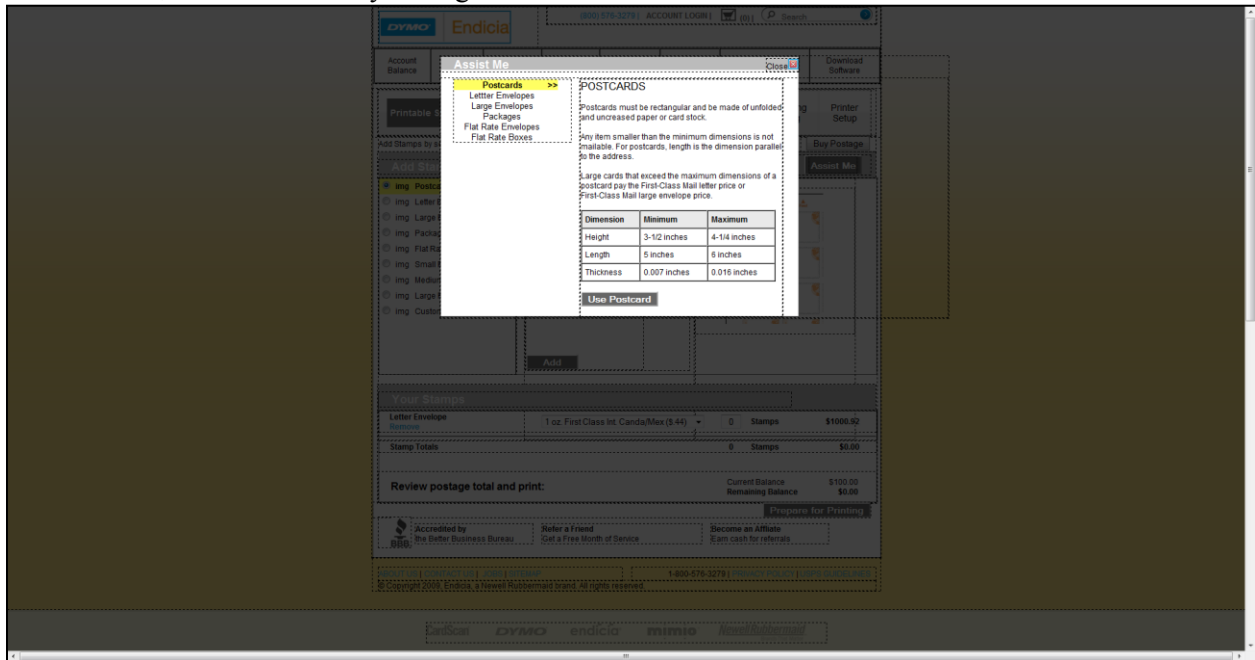
Revision 153, 1-28-11: The account navigation bar has been created and the page navigation bar is on its way to having the right look. Certain elements such as the "About US" link were moved to their appropriate places but are yet to be formatted.



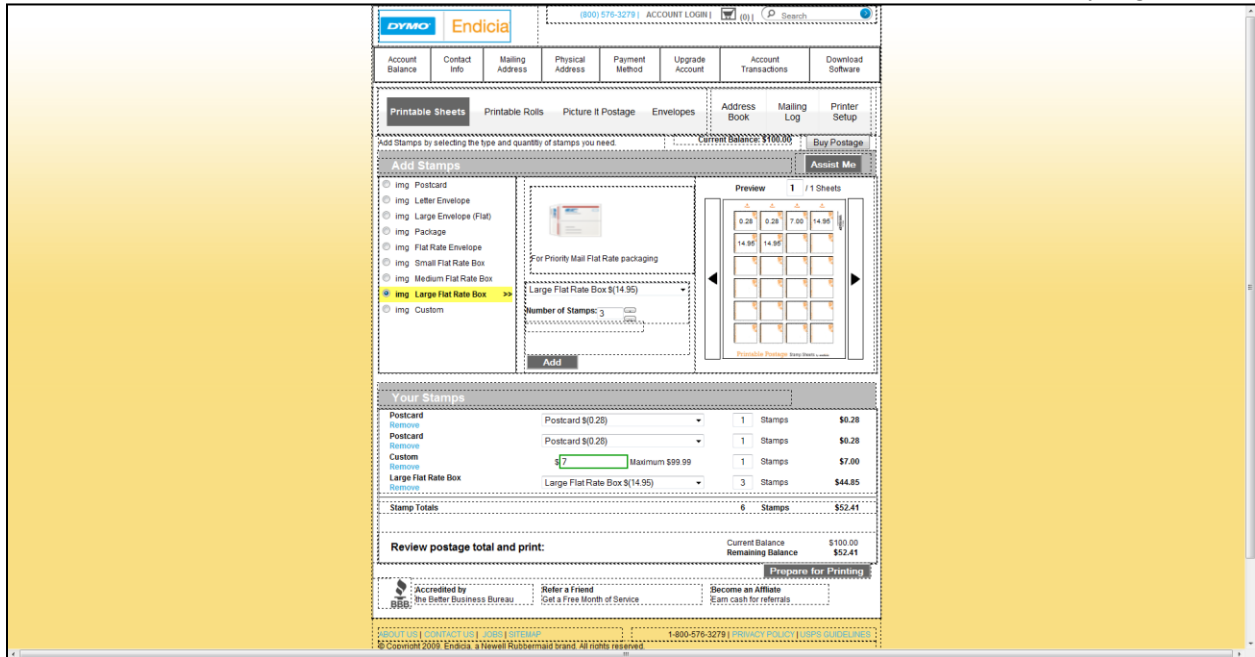
Revision 157, 1-31-11: The grid has been converted from a 960 pixels width to a width of 790 pixels in order to mesh with the structural format of the current Endicia website. We have converted the page into the new visual's layout. Links to associated companies and other part of Endicia's website have been added. A template of the order table was in place but there are no functioning JavaScript at this point in time.



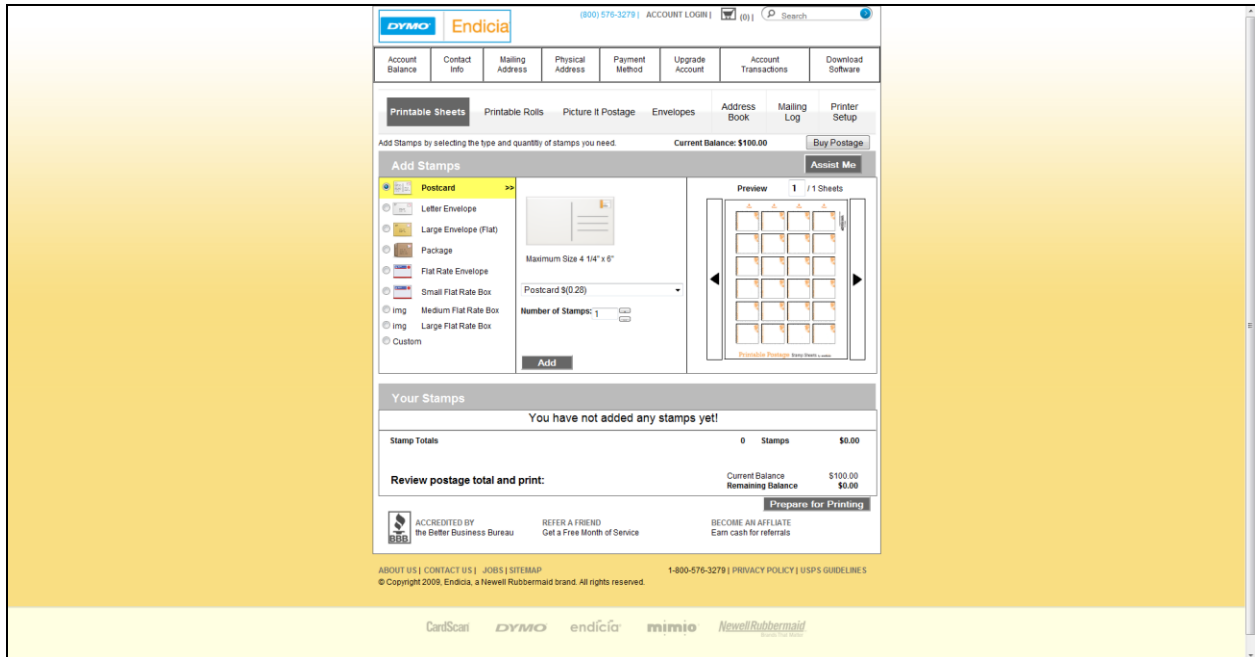
Revision 157, 1-31-11: The Buy Postage box has been converted to the new visual's look and feel.



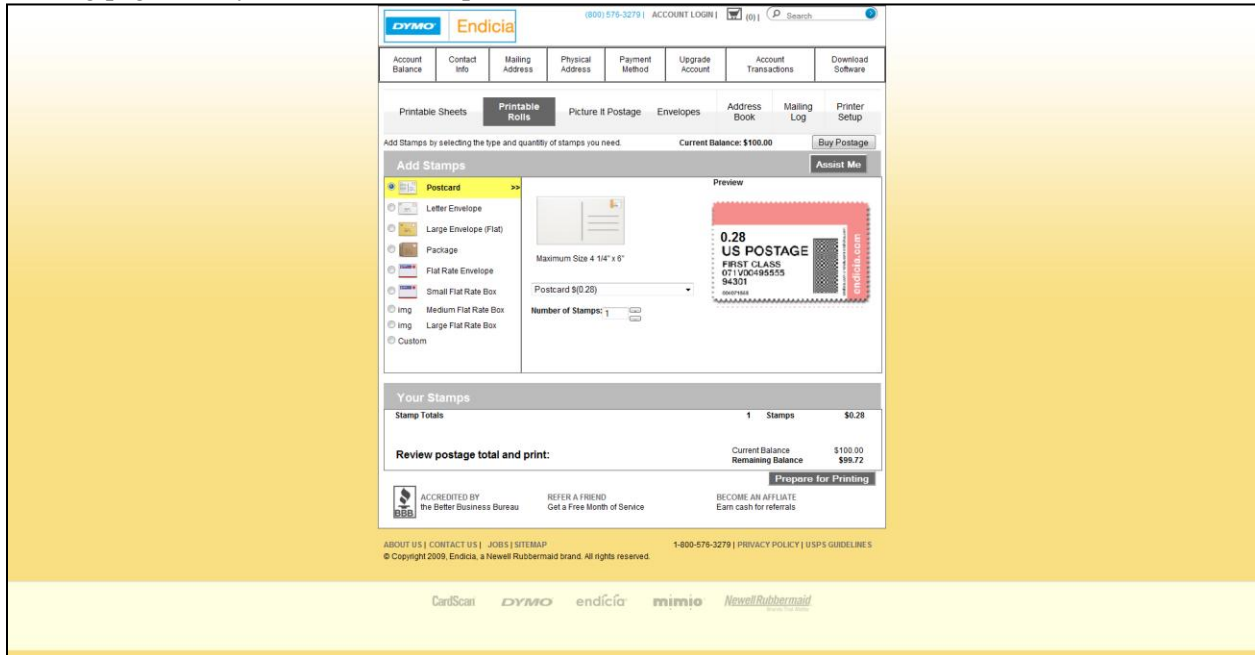
Revision 157, 1-31-11: The Assist Me box has been converted to the new visual's vertical styling.



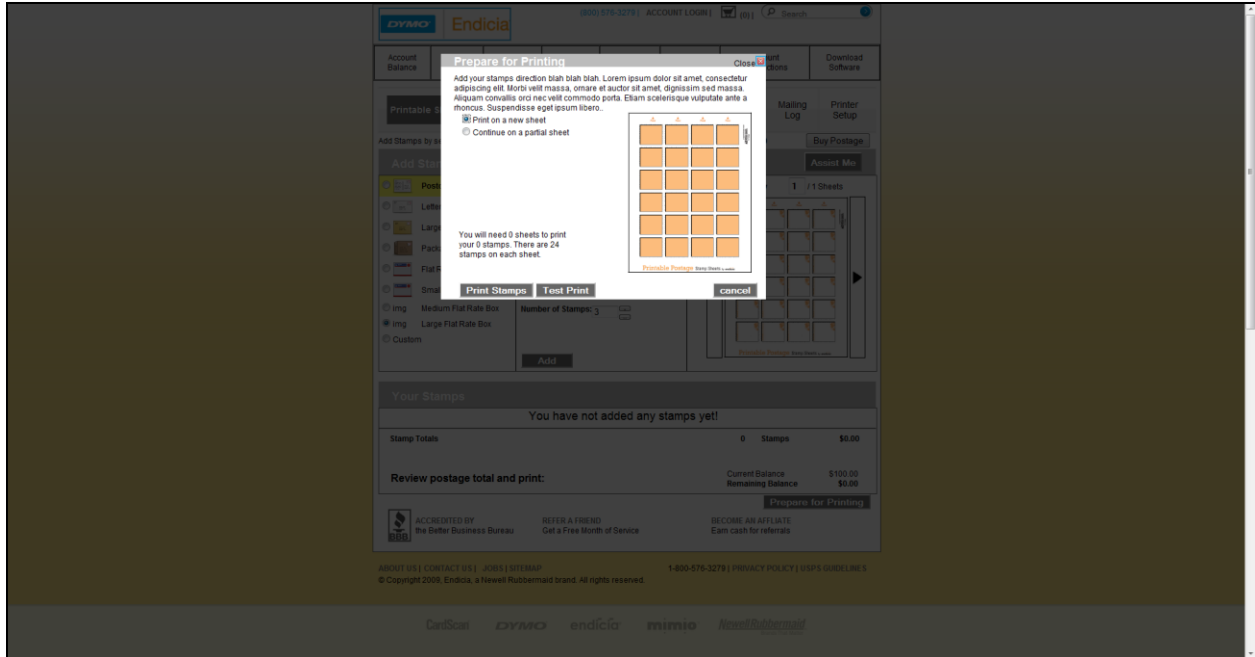
Revision 163, 2-1-11: The JavaScript has been converted to work with the new visual. The Add button adds a new row to the order table. Preview has working spinner buttons and the stamp order shows up accordingly in the stamp sheet preview area. All input checking has also been migrated to this version.



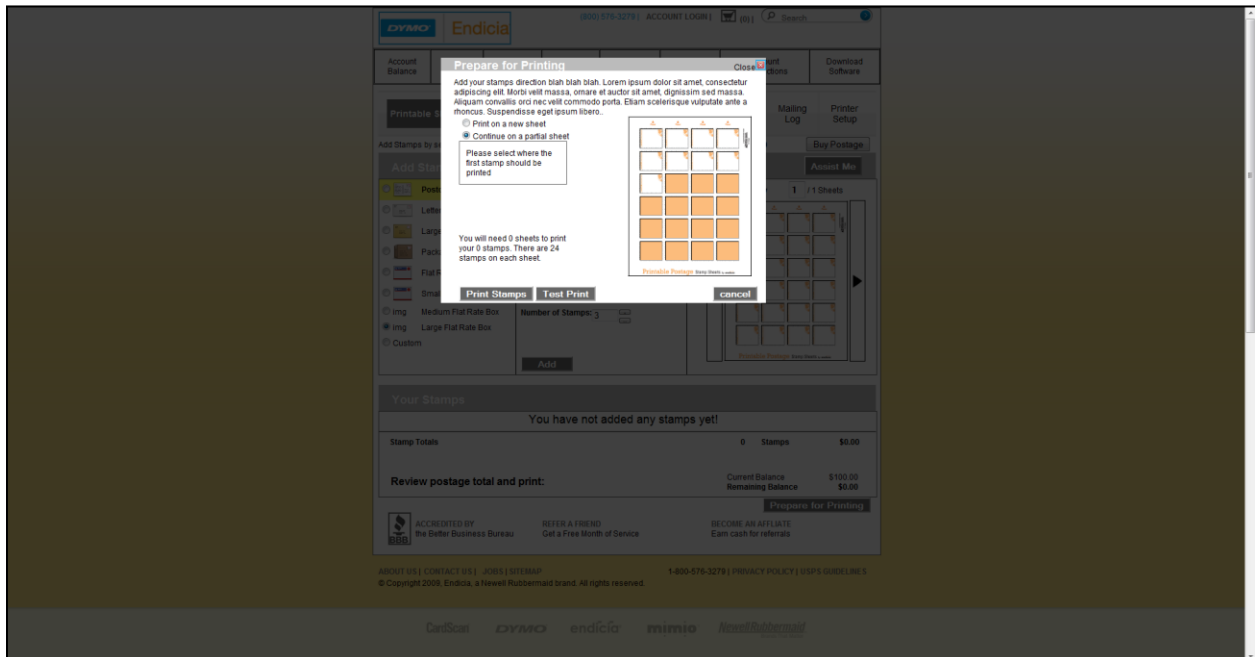
Revision 178, 2-2-11: The images for each mail piece type have been migrated into the page. The Sheet Printing page is fully functional at this point.



Revision 178, 2-2-11: The Roll Printing has been updated with all the layout changes in the Sheet Printing page. The Roll Printing page is fully functional at this point in time.

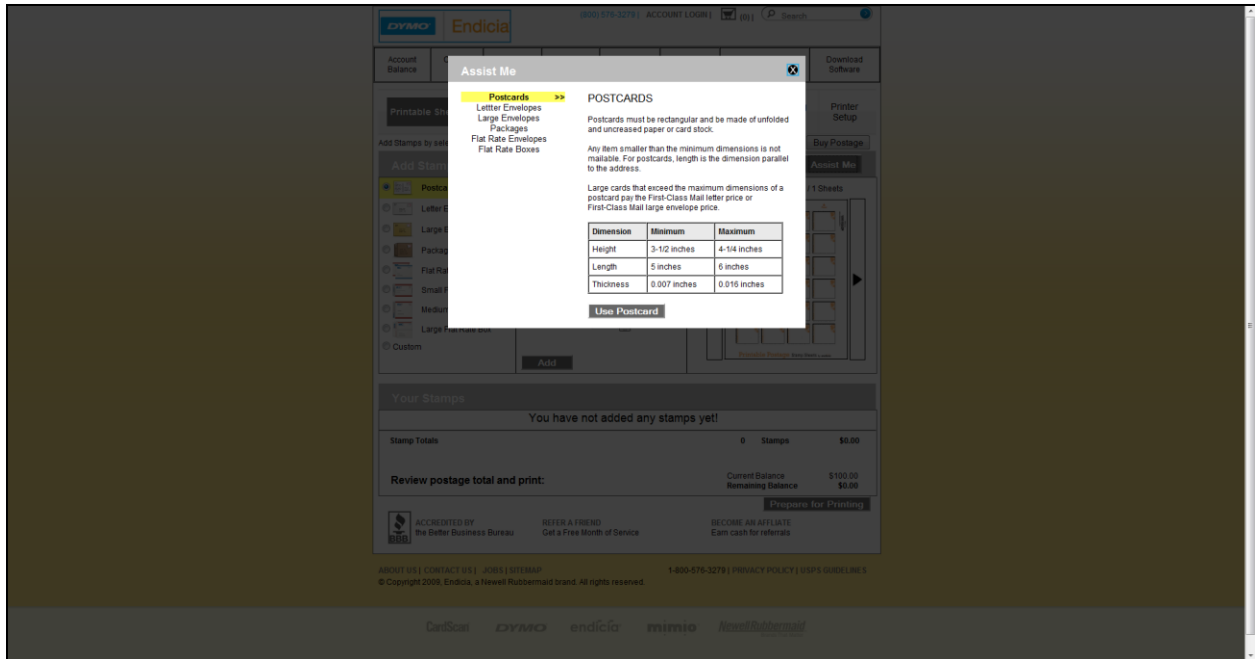


Print page with a new sheet.

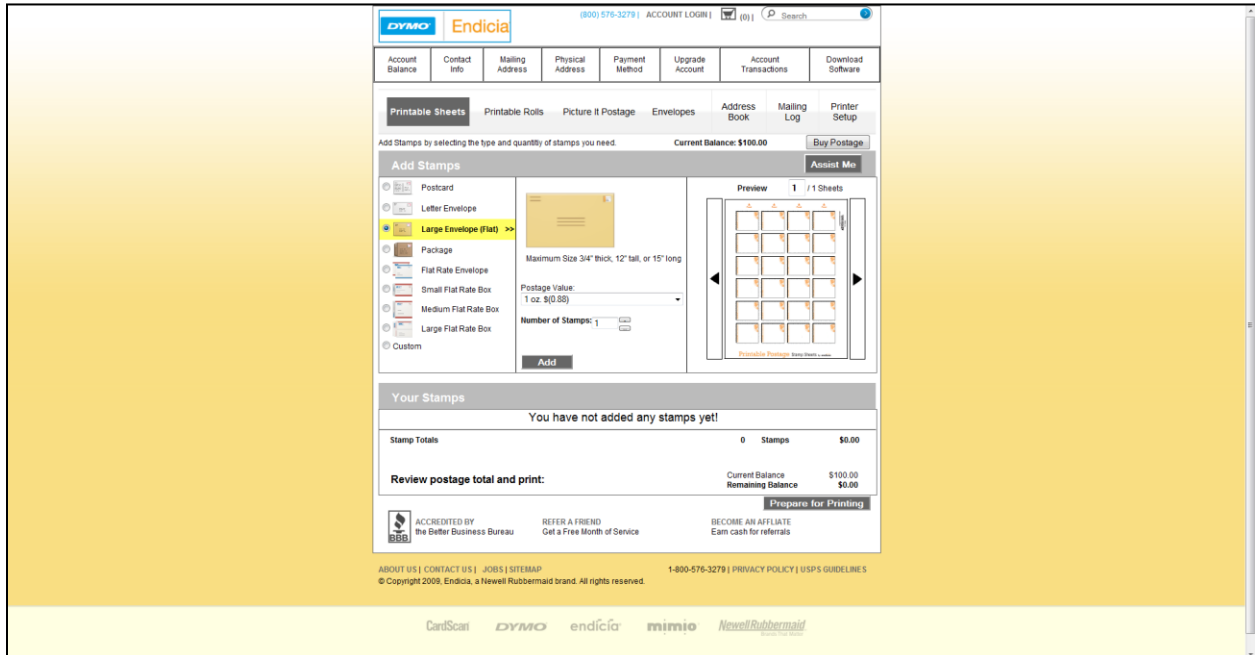


Print page with a partial sheet.

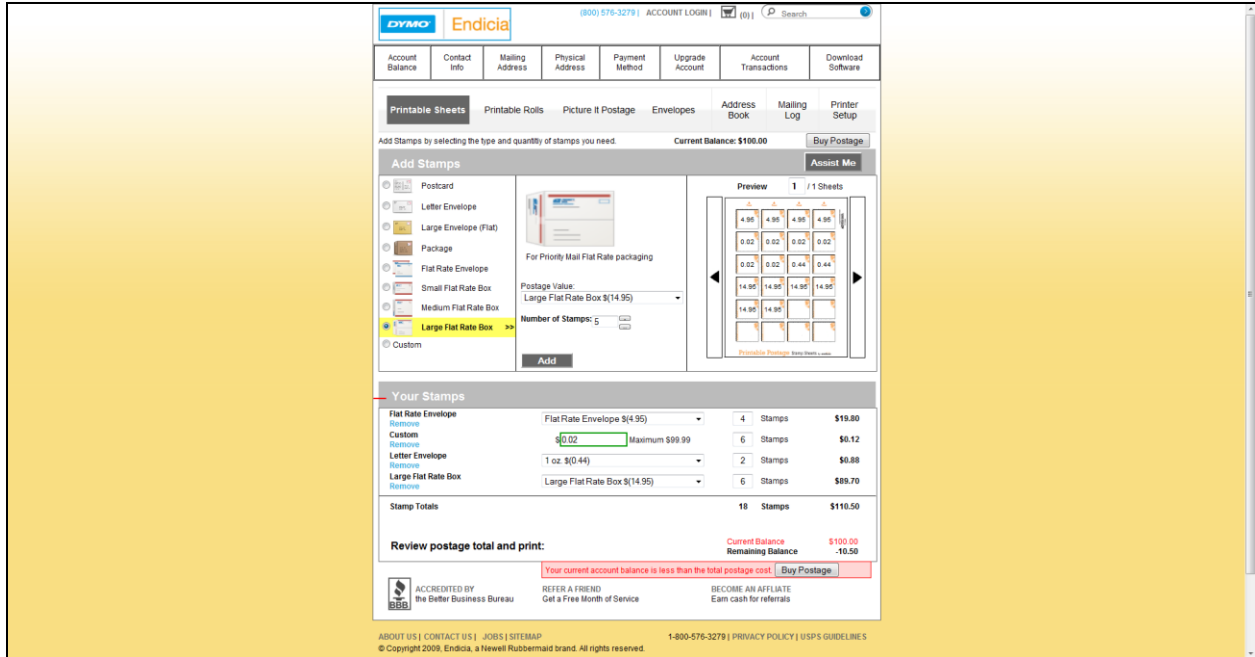
Revision 178, 2-2-11: The Prepare for Printing pop-up is now following the new visual style and design. The new sheet option fills the preview sheet with orange squares while the partial sheet starts empty and allows the user to fill the sheet in from the chosen position of the first unused stamp square.



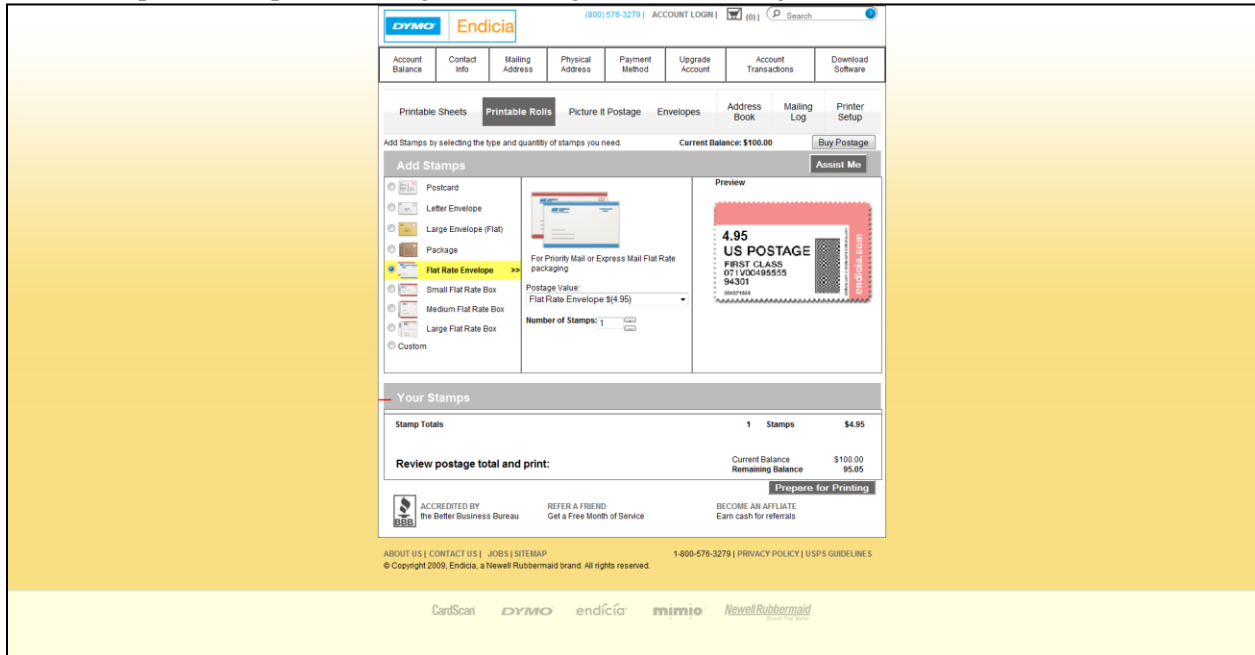
Revision 218, 2-4-11: All pop ups now have a customized close button that is bigger than original version.



Revision 218, 2-4-11: Both Sheet Printing and Roll Printing have the new images from the current DYMO Stamps application.

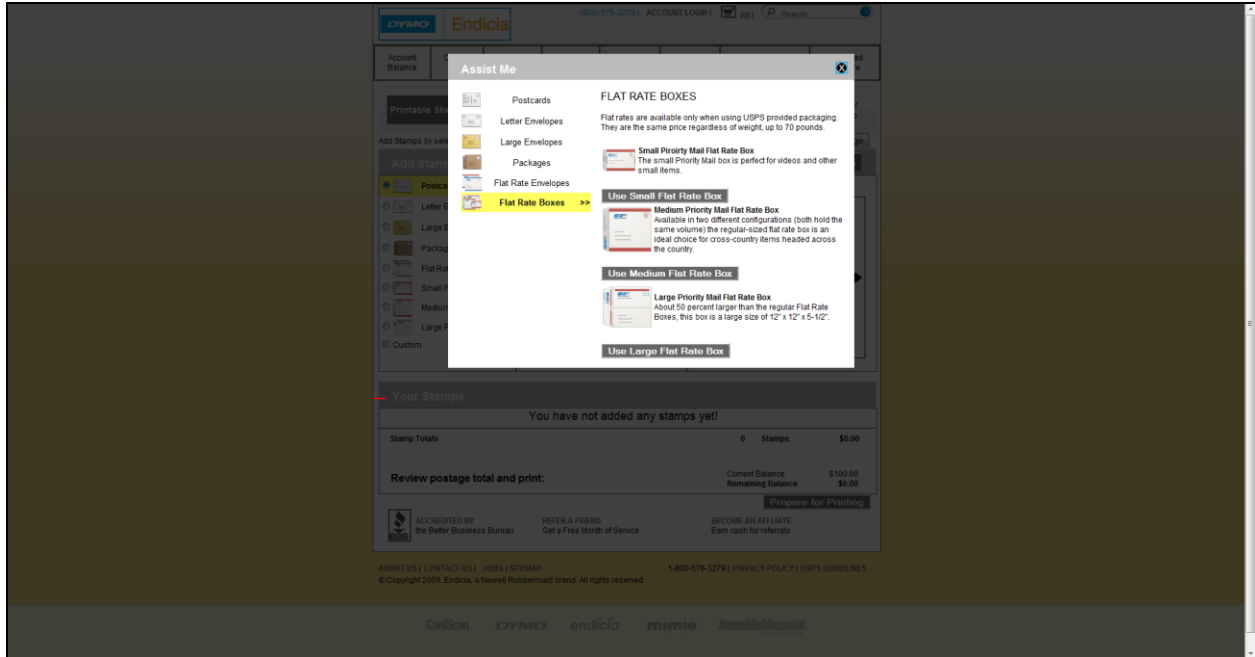


Revision 224, 2-7-11: Final version of Sheet Printing. The red line near “Your Stamps” shows the page fold at 600 pixels. We performed rigorous testing and fixed all bugs we found.

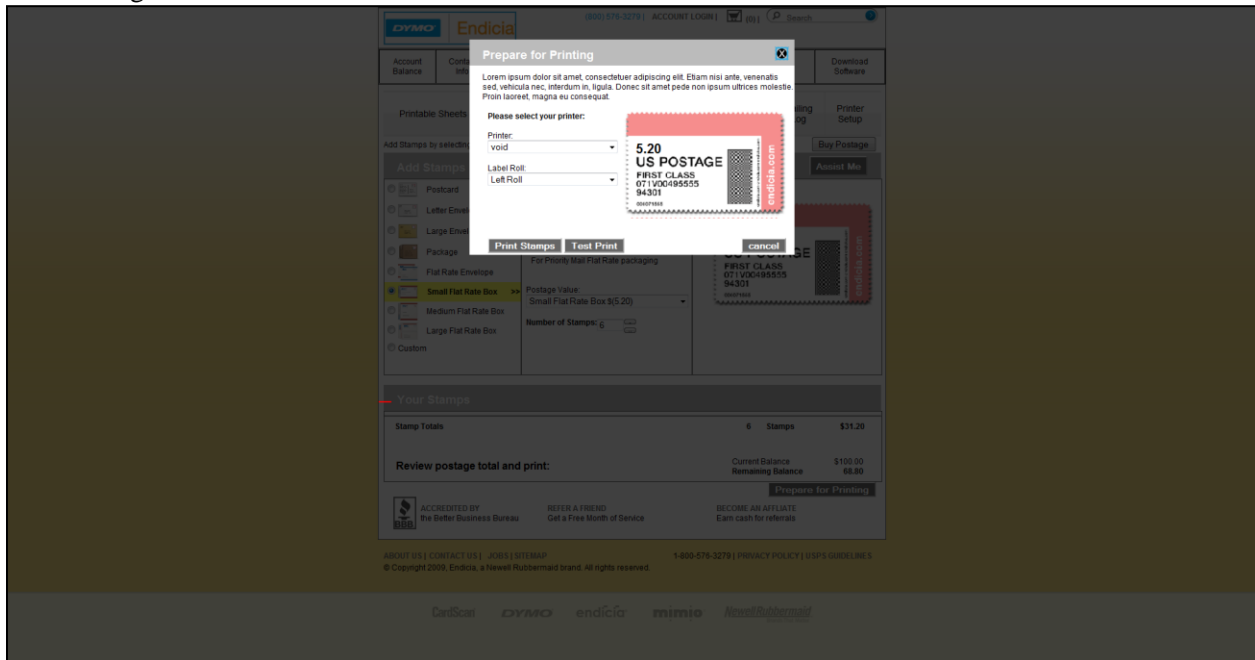


Revision 224, 2-7-11: The Final version of Roll Printing. Rigorous testing was performed and all the bugs we found were fixed.





Revision 224, 2-7-11: The final version of the Guide Me box has been updated with images and the navigation bar on the left has uniform width so the information will not move around as the mail piece name changes to bold font.



Revision 224, 2-7-11: Prepare for Printing page for the roll has the new look of the visual.