Daniel Wang
SpaceX Business Operations Intern 2015, Application

Attached is the Design and Technical Specifications of a computer science project I completed for Boston Apparel Company, an e-commerce business that I founded in the summer of 2014. The project involved implementing a crowd-sourced platform for the website, allowing customers to comment, bid, and help support custom apparel items. Boston Apparel Company is a subsidiary of Harvard Student Agencies.

A video describing the project can be viewed here:
https://www.youtube.com/watch?v=LvCRltobpz4

Jannie Reher, Daniel Wang
TF: Doug Lloyd

Documentation for:

Boston Apparel Company - Boutique Website

Our project is based on the already existing website http://www.bostonapparel.co, which is an e-commerce website for the student-founded *Boston Apparel Company* that has been selling Boston, Red Sox, and university apparel since November 2014. We augmented the website with an entirely new "BOUTIQUE" tab that provides the customer with several new interactive features. Firstly, the customer now has the opportunity to **place a bid** on a specific clothing item, which will only be produced and sold if enough customers have been found to back it. In case the funding has been secured, an email will automatically be sent to every customer who signed up to inform him or her that the item is now available online. A progress bar shows the stage at which the funding process is at the moment. Secondly, a **comment box** was installed to let staff and other customers know what they think about the products offered.

*Please note: Our project has been optimized for the **Google Chrome browser**[1], so please use this browser in order to enjoy the features on the website to the highest possible extent.*

Submission of form to back the production of a clothing item

In order to view the implementation of our project, please visit the website http://www.bostonapparel.co and click on the *BOUTIQUE menu* at the very right. You should now find yourself at http://www.bostonapparel.co/collections/boutique, viewing the latest item (a CS50 t-shirt), which will potentially be produced and sold through the website if enough customers will back it. We additionally implemented a progress bar and two lines of text showing the percentage of completion (based on the number of customers already backing the item relative to the total number of backers needed) and how many people are still needed to sign up in order for this item to be produced.

Now it's time to get this stylish piece of clothing into production! Please click on the image and have a closer look at the CS50 Custom T-Shirt. You may now declare your interest by clicking on the "Back This Item" Button in the right corner. Subsequently, you will find yourself confronted with a form that asks for your personal information – for security reasons, we are not asking the customer to provide his or her credit card number at this time. If you attempt to leave a field blank (apart from the additional information box), enter an email address without @ or try to enter more than 4 digits for the CVV code, you will not be able to submit this form. Also, if you don't provide digits but letters for your CVV code the data will

[1] Tested on version 39.0.2171.71 (64-bit) of the Google Chrome browser.

not be inserted into our database.

Please note: It might take half a minute or even up to a minute to submit the form, so please be patient!

Eventually – assuming you have by now tried to break this form in as many ways as possible – you will find yourself on a page that thanks you for your support. Furthermore, you should check your emails (in case you provided your actual email address), since you will have received an email confirming that you just backed the item.[2] In the meantime, we will have stored your data in our online database and we are now one step closer to actually producing the CS50 t-shirt.

If you click again on the BOUTIQUE tab on the right, you will notice that the progress bar has been updated and less people are still needed in order for the item to be produced. The color of the progress bar will change according to the number of backings that have been received – if the number of people who are backing this item is below 25% of the total number of backers needed, the bar will be red, if the number is between 21% and 70%, it will be yellow and from 71% onwards it will turn green. In the case that 5 customers in total have decided to back the item (this is currently the number of backers needed in order to break even), the text below the progress bar will change and our system will automatically send every backer an email greeting him or her in person and informing him or her that the item will soon be produced since enough backers have been found.

Comment Box

When clicking again on the image of the t-shirt, you should now find yourself at http://www.bostonapparel.co/products/cs50-custom-tshirt. Again, you will see the progress bar and some additional information. Please select now the menu "Suggestions/Comments For This Item" to the right of the t-shirt. A comment box will swing out and you can leave some suggestions for us on the homepage. Directly after entering, the comment will be stored in an online database and displayed on screen above the comment box. Upon reloading the entire page, the comment will be displayed above the thin grey line including the name and the timestamp.

[2] Emails will definitely be sent to addresses such @college.harvard.edu and @gmail.com, however, it did not work with @web.de. Do make sure to check your spam-folder since in our case gmail identified the emails as spam after numerous tryouts.

Beyond reaching our primary goal of creating a crowdsourced bidding platform – living on a third party server – that links to an online platform and a progress bar, we also implemented a comment box and several smaller features (client-side form checking, sending emails upon completion of backing process) that allow for a more interactive user experience.

Link to video on YouTube:
https://www.youtube.com/watch?v=LvCRltobpz4

In order to view the files in the *".liquid" format* in the *"Shopify" folder* please use the appliance's editor "gedit" or an equivalent. The files are not displayed online in the submission section.

Jannie Reher, Daniel Wang
TF: Doug Lloyd

Design Document for:

Boston Apparel Company - Boutique Website

Our website, http://www.bostonapparel.co, is hosted by Shopify.com, an e-commerce hosting platform. Our PHP webpages are hosted on 24hosting.com under the domain name www.bostonapparel.net. Overall, we used a combination of HTML, PHP, jQuery, CSS and MySQL coding.

Hosting the website online

We encountered two major challenges when implementing our project. Besides deciding on the layout of the different files and determining which languages we would use, hosting our website on a third-party server (24Hosting.com) and establishing a connection to the online MySQL database (which needed to be created on a paid server since most freely available hosts/servers would not allow us to establish a remote connection to that database) caused difficulties. In order to establish the connection to the database we ended up using the function mysql_select_db (instead of the already coded query function in pset7) and stored the information regarding the database, username, password and the server in the file constants.php (in $dbcon to be more precise), which is included at the beginning of the script.

Relevant files:

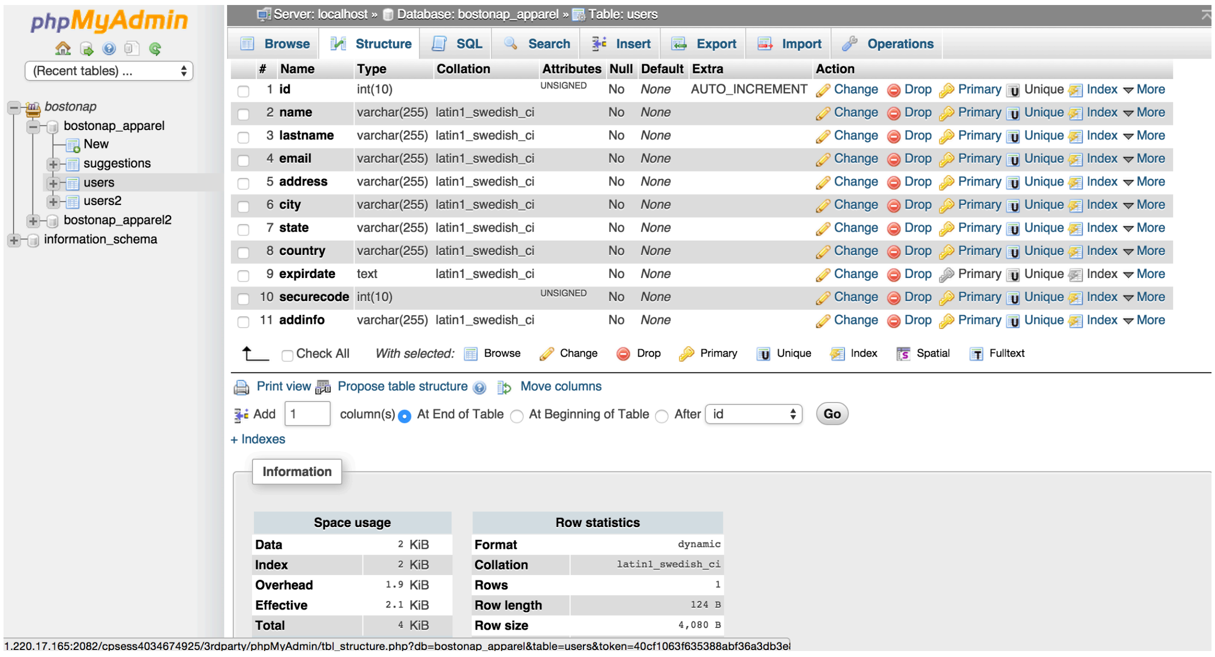
- includes/constants.php
- finalproject/BostonApparel.sql
- finalproject/table.sql

All files are world-readable and -executable for the purpose of making grading possible. Access was granted to the database via the "%-wildcard".

Databases

In our online-hosted myphpadmin software tool we created two tables in the database "bostonap_apparel": "users" (for storing the information from the backing form) and "suggestions" (for storing the comments submitted via the comment box). For the *"users"* table, the columns match the fieldnames from the kickstart_form.php form. Apart from the expirdate (which takes text as input since a "/" will be included in the submission) and securecode (which takes numbers as input) every variable is stored as varchar. id autoincrements in order to distinguish all the individual inputs.

For the *"suggestions"* table we included the variables id (again auto incrementing), name, email, body (to store the actual comments and suggestions) and date. We defined timestamp as the type of variable 'date' as this will store the specific date and time of submission of the comment.



Server: localhost » Database: bostonap_apparel » Table: users

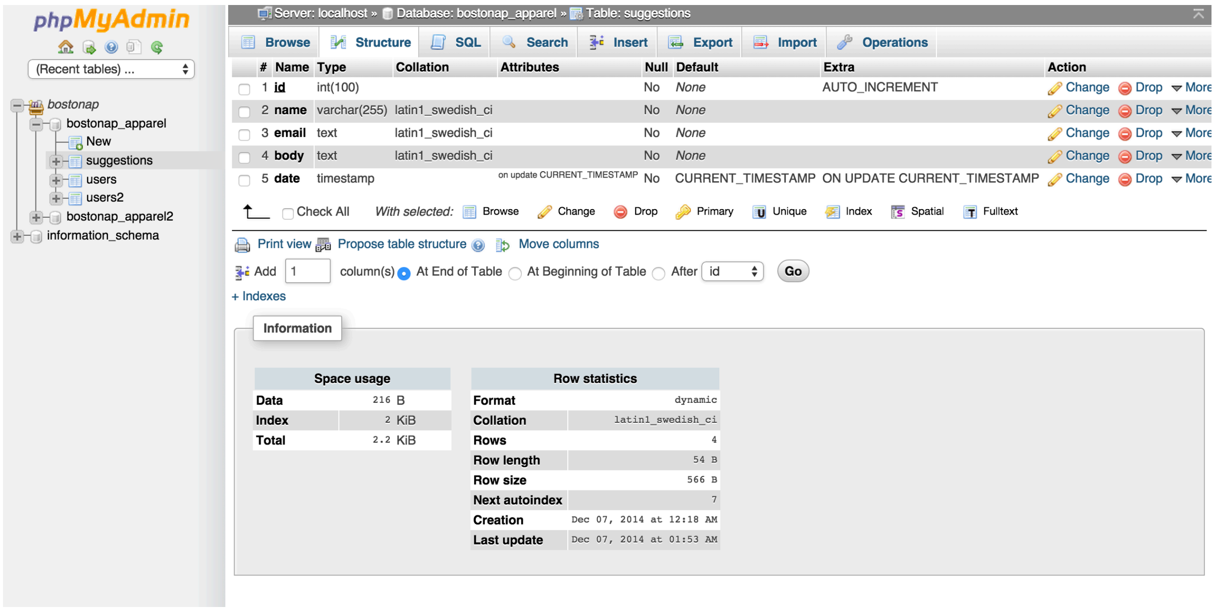
#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	id	int(10)		UNSIGNED	No	None	AUTO_INCREMENT	Change Drop Primary Unique Index More
2	name	varchar(255)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index More
3	lastname	varchar(255)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index More
4	email	varchar(255)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index More
5	address	varchar(255)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index More
6	city	varchar(255)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index More
7	state	varchar(255)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index More
8	country	varchar(255)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index More
9	expirdate	text	latin1_swedish_ci		No	None		Change Drop Primary Unique Index More
10	securecode	int(10)		UNSIGNED	No	None		Change Drop Primary Unique Index More
11	addinfo	varchar(255)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index More

Information

Space usage		
Data	2	KiB
Index	2	KiB
Overhead	1.9	KiB
Effective	2.1	KiB
Total	4	KiB

Row statistics	
Format	dynamic
Collation	latin1_swedish_ci
Rows	1
Row length	124 B
Row size	4,080 B

1.220.17.165:2082/cpsess4034674925/3rdparty/phpMyAdmin/tbl_structure.php?db=bostonap_apparel&table=users&token=40cf1063f635388abf36a3db3el



Server: localhost » Database: bostonap_apparel » Table: suggestions

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	id	int(100)			No	None	AUTO_INCREMENT	Change Drop More
2	name	varchar(255)	latin1_swedish_ci		No	None		Change Drop More
3	email	text	latin1_swedish_ci		No	None		Change Drop More
4	body	text	latin1_swedish_ci		No	None		Change Drop More
5	date	timestamp		on update CURRENT_TIMESTAMP	No	CURRENT_TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP	Change Drop More

Information

Space usage		
Data	216	B
Index	2	KiB
Total	2.2	KiB

Row statistics	
Format	dynamic
Collation	latin1_swedish_ci
Rows	4
Row length	54 B
Row size	566 B
Next autoindex	7
Creation	Dec 07, 2014 at 12:18 AM
Last update	Dec 07, 2014 at 01:53 AM

Relevant files:

- finalproject/BostonApparel.sql
- finalproject/table.sql

Shopify

Our website www.bostonapparel.co is hosted on Shopify, a paid e-commerce hosting platform. While Shopify has preconfigured themes with HTML/CSS, we had to make edits and additions in several files to make changes to the display and format of the Boutique webpages. There was a learning curve involved in using Shopify, since the entire platform is coded in Liquid, a proprietary HTML-like markup language that is used solely for editing Shopify themes. Namely, Liquid is an open-source, Ruby-based template language created by Shopify. It is the backbone of Shopify themes and is used to load dynamic content on storefronts. Thus, it required us some time to learn how to edit and write in the Liquid environment.

It is important to note that the Liquid environment did not support the dynamic-content features we wished to include, such as a progress bar, form submission, and comment box. Thus, we decided to write the code for these features mainly in PHP and host the webpages on a separate server. We then embedded these PHP pages in the Shopify webpage. While not the most appealing solution, this was the quickest and most reliable method of keeping the website live on the Shopify platform while also introducing the dynamic-content features that we wished to include.

The first file we created was "collection.boutique.liquid," which created the webpage showing the exclusive item on sale. More specifically, the image for the item is hosted through Shopify's product upload functions. The rest of the code, including the progress bar and automatically updating counter, were embedded into the HTML document as an object. The objects were then cropped to remove white space and to only show the content that was important to the webpage.

The second file we created was "product.boutique.liquid," which includes all of the details about the item. Again, most of the page was created by embedding PHP forms and content into the HTML frame of the Shopify page. The title, price, and details of the item are displayed through Shopify's own product upload functionality. The progress bar and counter are implemented in a way similar to those in "collection.boutique.liquid". The accordion display used to hold the product content was part of the original Shopify theme, and was slightly altered in order to display the content we wished to include. Under the "Suggestions/Comments For This Item," there are two separate PHP pages that are embedded. The "Make A Suggestion for This Boutique Item" comment box was implemented using adopted code. The form checks for user input and then displays the comment upon user submission. This interaction is described in further detail in the Comment Box section. In addition to displaying the comment to the user, the comment box also queries the comment and user information to the “suggestions” table under the “bostonap_apparel” SQL database hosted under bostonapparel.net (the domain name we bought out to host all of the PHP files). The PHP page above the comment box, also embedded into the Shopify page as an object, queries data out of the “suggestions” table, effectively displaying all of the past comments along with the user’s name and timestamp. We did not want the page to take up too much room on the webpage as more suggestions were added, so we decided to limit the size of the embedded page and allow the user to scroll through the suggestions.

When the user clicks the “Back This Item!” button, the third Shopify page we edited, “page.boutique_checkout.liquid”, is loaded. This page simply contains a PHP form that requests the user’s contact information. Note that we intentionally did not include a field for a credit card number. Since the webpage is accessible to people around the world and is visited on a daily basis, we did not want visitors to enter their true credentials through the form since we did not perform rigorous security checks. Once the user information is submitted, it is queried to the “users” table in the “bostonap_apparel” database.

Relevant files:

- shopify/product.boutique.liquid
- shopify/collection.boutique.liquid
- shopify/page.boutique_checkout.liquid

Progress Bar and Remaining Counters

The progress bar is displayed via the html <meter> tag which changes colors automatically based on the value inputted in relation to the pre-determined min, max, low, high and optimum values. All these numbers are dependent on the global variable $total, that has been defined by hardcoding the total number of items needed in order for the t-shirt to be produced (5 items). Should this number change (e.g. due to a change in the production process), the number need only be modified in constants.php and it impacts all other calculations such as the progress bar and the number of people that remains to back this project successfully. The progress bar’s value is determined by querying the current number of rows in the MySQL table “users” via the mysql_query("SELECT * FROM users", $dbcon); function. This number is then converted into a percentage value in the <meter> tag.

All these calculations are dynamic since they depend on the current number of rows in the “users” table and on the global value $total. This required us to write the script in PHP instead of using pure HTML, which is why we had to create a separate file that is linked into the Liquid code on Shopify. If all backers have been found, the statement changes to say that the item will be produced shortly.

Relevant files:

- public/itemstogo.php
- public/new_progressbar.php

Submission of form to back the production of a clothing item

The form (kickstart_form.php) that is filled out by the customer in order to declare his or her interest for the item has been written in HTML. There are two ways in which user input is being checked: i) client-side check with jQuery and ii) server-side check with PHP. In kickstart_form.php every time the submit button is clicked, jQuery will remind the user to fill out any field that has been left empty (except for the additional info field, which is optional) via the adopted $("#backitem_form").validate function. This will prevent the user from submitting an incomplete form. After the submit button is pressed, there is another check performed by the server via several if(empty) and a if(!is_numeric) statements that will stop the data from being entered into the MySQL database if it does not fulfill the necessary requirements.

On kickstart.php we did not check separately for the request method, e.g. by $method= $_SERVER["REQUEST_METHOD"] (which could either result in “GET” or “POST”). It is rather difficult for a user to reach the kickstart.php page via GET, i.e. through a link (because we have not established one) or by searching for it (because the actual page is embedded in Shopify’s Liquid code). This is why we did not make the execution of kickstart.php dependent on the specific request method.

Submission of the actual form takes quite some time for several reasons. We presume that it takes quite some time for the server to process and send out the confirmation email, especially since we are hosting via a cheap server. Furthermore, a new connection must be established to our online database every time, which will also take some time. Lastly, our code could have possibly been written more efficiently or simply in a different way in order to accelerate the submission process.

What should be noted is that the MySQL table "users" will in fact accept more backings than the $total (in this case 5) determined in the constants.php file. This means that users can still fill out the form although enough backers have been found already. However, we will inform the user of the finishing of the backing process on the one hand via email and one the other hand via a new message on the screen below the progress bar so that he or she will probably not see the need to fill out the form.

Relevant files:

- public/kickstart.php (controller)
- templates/kickstart_form.php (HTML form)
- templates/kickstart_form.css (css stylesheet)

Email

Personalized emails are sent to visitors when they submit the form to back the item and when the item has reached the minimum number of backers to reach production. All of the code for this is contained within kickstart.php. Specifically, an email thanking the customer for backing the item is first sent to the provided email address with the 'mail' PHP function. The email is obtained via $_POST['email'] and the email is personalized in the subject line and body with the first name of the customer via $_POST['name']. The script then checks to see if the minimum number of backers has been reached with the most recent submission. This is validated using the 'mysql_query' function to select all of the rows in the table, and then the 'mysql_num_rows' function to record the number of rows as an integer value. If this integer value is equal or greater than the globally defined '$total' value, then an 'if loop' is executed. We originally wanted to insert all of the email address into the '$to' field as a string delineated by commas, but we realized this would send out one email to all of the email addresses at once with all of the email addresses visible. Instead, we decided to implement a loop, choosing security over speed. The loop selects the email addresses from the table one at a time and sends the customer an email notifying him/her that the item has been moved to the production phase. We chose to implement this in a loop format so that customers would not see the email addresses of other backers. The emails are also sent using the 'mail' PHP function, with the '$to' variable being filled in with the email addresses in the table one at a time.

Relevant file:

- public/kickstart.php

Comment Box

The comment box is written using adopted code from http://tutorialzine.com/2010/06/simple-ajax-commenting-system/. The code utilizes HTML, PHP, CSS, and jQuery to create a form that dynamically updates and shows a user his/her suggestion without reloading the entire page. Specifically, the submit.php first validates that all of the fields have been completed. The file also utilizes a JSON object to keep the data in the fields if an error is encountered. All of the error-checking functions are contained within scripts.js. Since the adopted code only displayed the suggestions in the present page (i.e. the comments would be deleted once the page was reloaded), we decided to store all of the suggestions in a SQL database so that they could all be displayed to visitors. In addition, we chose to store them in a database so that unwanted submissions could be easily deleted by the site administrators. The code specifying the connection details is contained in connect.php. We also added code to submit.php to query the data immediately to the database. We then created submissionlog.php to display all of the past suggestions. We decided to only display, via the query statements, the name and timestamp of the visitors to protect the user identities.

Relevant files:

- suggestionbox/comment.class.php
- suggestionbox/connect.php
- suggestionbox/demo.php
- suggestionbox/script.js
- suggestionbox/styles.css
- suggestionbox/submit.php
- suggestionbox/suggestionlog.php
- suggestionbox/table.sql