Portfolio-2

Trackpack

Coms 319 - Spring 2016

Alex Berns and Ken Kohl

Package Tracker

Portfolio I

TABLE OF CONTENTS

Overview:

TrackPack is a mobile management software solution for customers to understand the movement of their valuable shipments. TrackPack also makes it easy for employees to submit package updates as they move around. This is all behind an account based system to keep the customers and employees separate on their tasks.

TrackPack has two different users it services. The employee and the customer. The employee is able to update packages locations. They have to provide the package ID, the new location of the package, the employee id for bookkeeping purposes. The customer on the other hand is able to view these updates to their package and submit their own. The front end does not yet allow for tracking of packages that you did not submit, but the PHP methods have been created.

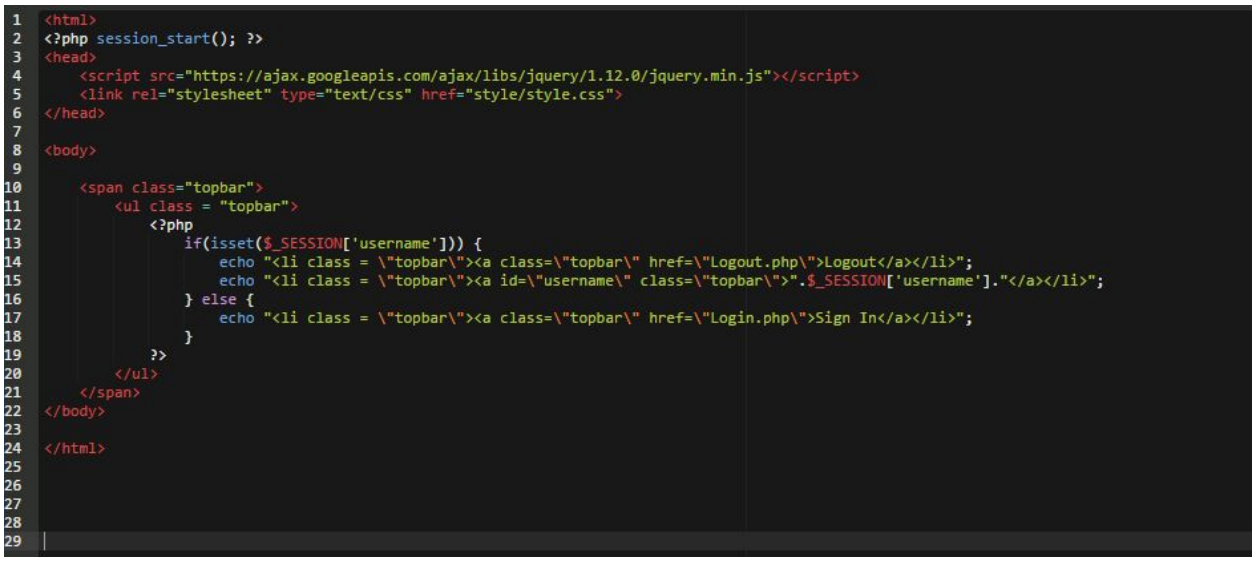
Another feature we had to create for this was user creation and login. We had to implement php code to check for login and check for valid inputs, along with return error messages appropriate to the circumstance.

For this software we learned lots of new techniques for creating websites with PHP/HTML/JS. Primarily we focused on making a more modular system of code and html. This lead to more writing of code as we designed and redesigned but less in the long run since we have learned it here.

Non-Class Materials and Issues

For this portfolio we decided to use three javascript libraries that were new to us. Vis.js allowed us to create timelines of a package movements. Moment.js was used to simplify javascripts date system. And Faker was used to create packages for testing, much easier than manual entry. These combined to make display and testing of the entire customer side of the application easier. Gone are the days of typing into 10+ fields random data values. Now with a click of a button, we could create new, real looking, information.

We learned how to create a universal style guide with the use of a piece of html that exists at the top of every page. This top bar and the use of a css file created a universal style that was enjoyable to create. This also allowed us to create some code that would be run on every page to check if the user was logged in. If they were not, then they were redirected to the login page automatically. This was not taken as far as some other websites, but allowed us to exercise some ideas in this design.



```
<html>
<?php session_start(); ?>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style/style.css">
</head>

<body>

    <span class="topbar">
        <ul class = "topbar">
            <?php
                if(isset($_SESSION['username'])) {
                    echo "<li class = \"topbar\"><a class=\"topbar\" href=\"Logout.php\">Logout</a></li>";
                    echo "<li class = \"topbar\"><a id=\"username\" class=\"topbar\">".$_SESSION['username']."</a></li>";
                } else {
                    echo "<li class = \"topbar\"><a class=\"topbar\" href=\"Login.php\">Sign In</a></li>";
                }
            ?>
        </ul>
    </span>
</body>

</html>
```

Code for header.php

One of the issues we ran into was dates for Javascript. Dates are based on the time region you are in or are just based on unix time. We wanted to tag package activity with an absolute date, unix time, but in translating that back into human readable time we had to provide a timezone and some other information. Unix time also had issues with being used with Vis.js for the timeline. To fix this we decided to use Moment.js. This would in theory work to create a date

object that would be readable by the vis library. This also failed. Vis.js had old code that complained about methods that would soon be out of date. So we continued to use Moment for dates but changed it all to a date string in the ISO format. This fixed our issue and also still allowed Vis to interpret the date without any troubles and storage of the date became human readable so no parsing of it into one when displaying it.

Complex Topics

Universal Style:

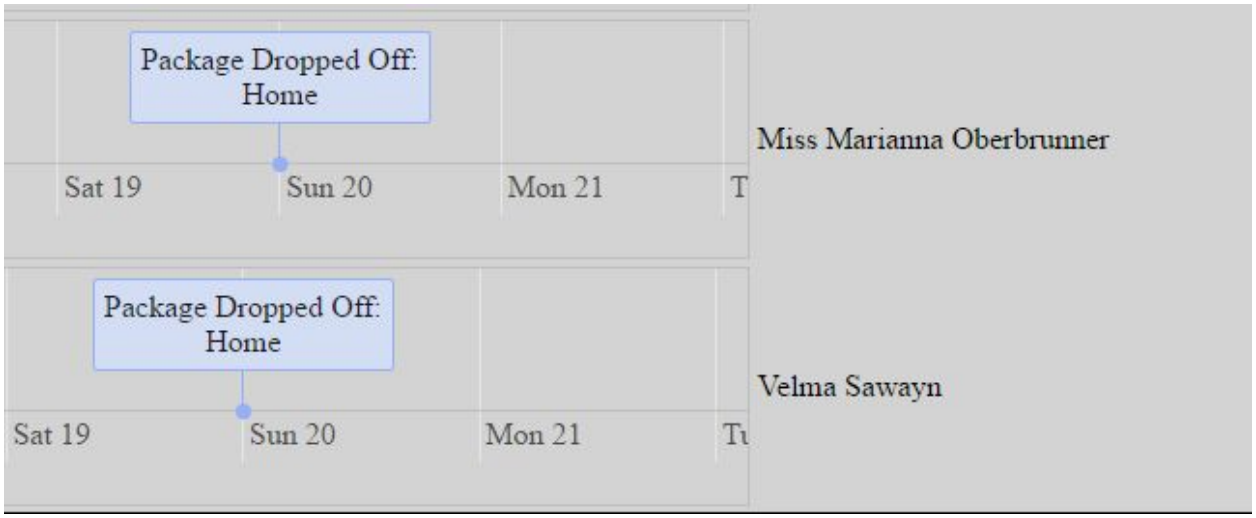
We added style to complex topics due to the way we approached it. To begin with, we learned how to use PHP's 'include' method. This allowed us to create a universal navigation bar at the top of the website. This bar contained the user's name and a logout button for when the user was logged in. But if the user was not logged in, it kicked them back to the login page, like many websites do. Since this was all stored in a single file, we could use one line of code in all the pages that needed a top bar and when we had to change something in that top bar, it was only a few clicks. This use of the 'include' method allowed us to create a universal style.css file. With this, we were able to create buttons and tables and inputs of all kinds to look the same. Even dynamically written DOM elements, like those from JS could be given a style with less JS lines. Both of these was something we had never strictly done. A similar idea was played around with before in our previous experience with Android but never went this far. This use of elements that must exist in every page really helped to reduce the amount of code we had to write.

(Snapshot of universal header/style)

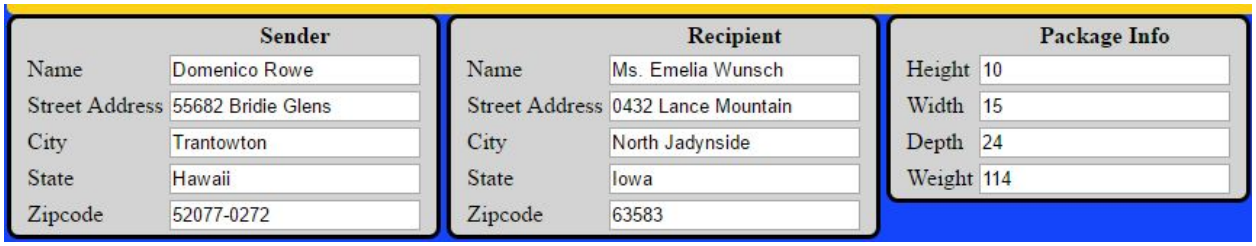
Timeline and Dates:

We used Vis.js, a robust JS library for displaying many kinds of graphical information, allowed us to create a timeline for the movement of a package. Now this was added into the complex section, along with Dates, because of the timelines use of dates and the need for us to randomly generate them. JS and PHP do not clean date generation methods, or at least none that we found, instead we used Moment.js. This library had some native integration with Vis.js as some point, which is what pointed us to it. Unfortunately, this integration was being taken away in coming updates for Vis. This meant we got lots of console warnings and issues. But, we already had Moment learned. So instead, we opted to use Moment to print a nice looking Date String to use in the Timeline. This string was read and interpreted by Vis into a real date all the same. We choose to use the ISO date format for storage just to make it look more consistent.

Another major feature of Moment that we used was it ability to generate a random date within a range. For testing, the package creation is set to generate a timestamp anywhere from the current time to 2 weeks back. This allowed us to display the events on the timeline without having to modify the raw data.



(Snap shot of timeline with dates)



(Example of creating package with auto generated fields)

Blooms Taxonomy

Analysis

TrackPack was built with the idea of expandability. We wrote the PHP code then wrote the Website since we knew what functions and features the backend would need to provide. This also allowed us to use the MVC pattern since storage and backend code where the model and controller parts, dealing very little with the front end unless called.
Model

- user.json
- package.json

Controller

- All the server side PHP Code

View

- PHP/HTML

Evaluating

For this project we chose not to use a database for our data storage. We believe that everything we wanted to accomplish would be just as feasible without a database. Instead we used a json string stored inside of a text file. The biggest reason for this was to try out the unique nature of PHP's and JS' hashing of variables. This intrigued us and we wanted to play with this more than a database. So when creating the system to store packages, we stored them with the key of the package id. This allowed us to write checks to see if the package existed and check its information by only providing a package id to the php code. The php code then just opened the json, checked the hash, and returned a value. The idea that all parameters of an object are just stored as a hash is not something we have encountered before and we wanted to see how far we could take this idea. Databases are databases, they are universally used. But learning to use JS and PHP in a fun way to access and store data is fun new challenge.

Creating

The creation process for this software was fun. During its creation we learned many new things about JavaScript and PHP. Our favorite was the fact that JS object properties are actually just hash maps. We took this idea and ran with it when developing the package lookup and storage. By storing the packages as just part of this hash with the key being the package id, we could

quickly load and modify packages with minimal effort. We had thought about doing a database but we valued learning the hash more.
It was also fun to further develop the login system. We had made the login system for labs 4 and 8 but with this project we were able to develop them further by allowing the email to have any number of characters a typical email would rather than what lab 4 required. We wanted to develop the security further by hashing the information sent between the server and the files stored within but we decided that our time was better spent features that would better demonstrate what we learned in class.

The complexity of this project came from writing a reusable and extendable system. The php code was created before the html. Then the html was created before the JS that added more things. We created it piece by piece and this made it modular in design. We also wanted to keep things modular in design, allowing us to reuse the look of buttons and graphs and tables by only labeling the DOM elements with a class.

Lastly was the creation of two different users. Initially the capability of the employee and the customer were in a single page. But since we made each part in its own HTML, we only had to move the import statement. This allowed us to debug very easily.