COMS32500 Web Technologies

Luke Storry (LS14172)

August 2018

Introduction

I have developed a simple web store, selling windsurfing equipment.

There are a range of products dynamically loaded from a database, a user accounts system, a shopping cart stored in cookies, a search feature, and a design-your-own-sail app.

I've submitted the source-code in a zip folder, unzipping it and running `npm i; npm start` should get it running at `http://localhost:3000`. See the README for more usage instructions.

The source-code is also on Github at `https://github.com/LukeStorry/windsurf-shop`, and the website is deployed on Heroku at `https://windsurf-shop.herokuapp.com`.

HTML - A

I originally wrote the website in XHTML (verified with `validator.w3.org`), occasionally using adaptations of Bootstrap snippets - eg for the nav-bar and product-card elements.

When it came to dynamic page generation there wasn't the flexibility I wanted, and to keep my codebase clean and DRY I opted to use the Pug's HTML-generation templating framework, allowing simple python-like code to take variables passed in from the Express server, and produce fully-validated XHTML.

Re-writing all of my HTML and the various Bootstrap snippets in such a way as to ensure Pug produced good clean XHTML (and that it correctly interfaced with the data objects provided by the server) required me to develop a much deeper understanding of the different issues surrounding HTML delivery, and the structure of HTML pages with various `divs`, `containers`, `rows`, and responsively-expanding grid systems.

However, to show my ability to write pure XHTML, and link it with client-side JavaScript, I left the "Design your own sail" page as a statically-served XHTML file.

I have investigated a variety of different issues, and gained a general high level of confidence with both the structure of HTML pages and with generating HTML via a framework.

CSS - A

The majority of my site uses the Bootstrap framework's CSS package, which provided me with a good base of well-written CSS to read through and select classes from, to use in my HTML. For places where I wanted extra customization not provided by the Bootstrap CSS, I either overrode the particular CSS tag (eg with the `card` class), or added my own (eg with `bg-blue`), to keep all style out of my HTML files and in a separate CSS directory.

I have investigated a variety of different issues and gained a general high level of confidence with CSS style, and using/adapting CSS via a framework.

JS - B

Most of my time was spent writing Javascript on the server-side, but I did also experiment a lot with client-side JS.

Getting various client-side scripts to work robustly over the Express server, especially as I moved to cloud hosting, meant that for many things (such as cart quantity updating) it worked a lot better when interaction was done via GET/POST calls handled server-side instead of statically serving JS files via Express.

The client-side JS I included for this coursework is a touch-compatible drawing app, where users can draw their own sail design. Getting correct coordinates for both mouse and touch, as well as smoothing out the drawing when the mouse is dragged quickly, and including an overlay with the translucent sail, were the main challenges.

I've written a script with a number of functions with different issues involved, and have gained a high level of understanding of how client-side JavaScript works.

PNG - A

For all of the images (and descriptions) used in the shop, I used products from Boardwise. I used GIMP to clean up and remove the backgrounds of all the product images, to look nicer on the coloured card backgrounds in the store.

As there were a variety of different formats of image on that site, with some low-resolution jpgs and gifs, I had to employ a range of techniques, but my usual workflow was something like:

- converting the colour-space to RGBA
- splitting the image into multiple layers
- selecting the background with the magic wand tool
- growing the selection to account for aliasing on the lower quality images
- editing the selection with the lasso tool
- using a bucket-fill with colour-erase mode to remove the background with partial transparency, keeping shadows and other effects (like translucent sails) where possible
- some airbrushing, hand-erasing or selective darkening was applied to clear up the image where needed
- Export as PNG

I have sorted out basic skills such as converting images to PNG, gained experience with basic tools such as changing colours and filling, and gained experience with more sophisticated tools such as handling layers and transparency, and airbrushing

SVG - B

The site logo was created using Inkscape, with text, shape tools, grouping, and path editing. This .svg was then exported to a png, to be used as both the favicon and the navbar branding.

I've gained experience with some of Inkscape's features.

Server - A

I opted to use Express and the range of middleware add-ons it provides, to be able to focus more on adding wider range of functionality to my server.

To generate the initial file-structure and backbone connections of my express server, I used the `express-generator` package.

The modules I used are:

`bcrypt`	To handle the hashing and comparison of stored passwords.
`connect-flash`	To display error messages produced by form-validation to the user.
`connect-sqlite3`	To connect the session with the backup session database.
`cookie-parser`	To get user's id from cookies, enabling reloading the cart from the database.
`csurf`	To protect user pages against Cross-Site Request Forgery.
`express-session`	To save the cart in the local session, and back up cookies in a database.
`express-validator`	To validate form input.
`passport`	A modular Authentication platform, to secure the sign-in / out.
`passport-local`	A Strategy for the Passport module, authenticating using username/password.
`pug`	A templating engine, producing validated XHTML from pug scripts.
`serve-favicon`	To provide the favicon to browsers with the best content-type
`sqlite3`	For loading and editing the database.

I used LetsEncrypt's free SSL certificate-issuance client, `certbot` to provide HTTPS security, with express's https server module overlaying the original http one on a different port.

However, I then looked into cloud hosting, and used Heroku's HTTPs certificates and port options for my final deployment. This deployed set up is visible at `https://windsurf-shop.herokuapp.com`.

Apart from the docs and guides provided by the modules themselves, I also followed part of a Youtube tutorial to help with the Passport integration for logging in and out and storing sessions. Although the tutorial series offered other advice on setting up an online store, much of it was not useful, as the aims of that site were different from mine, plus it was using MongoDB and Handlebars, so the amount I could take from it was very limited.

I created a server by loosely following a tutorial to set up express, then dealt with many other things, like port numbers, URL validation, mixed-content, sending redirections to browsers, https and certificates, cloud hosting, security issues beyond URL validation, and cookies

Database - A/B

I used Node's `SQLite3` package to access a database on the Express server. The main database contains a `products` table with all the information about various products, and a `users` table, which contains names, email addresses and hashed passwords for any user that sign up.

The `products` table is simply read into memory when the server is turned on. To more easily update the products in the database, I wrote a short SQL script attached to '`npm run update-products`', which loads a `products.csv` file into the database.

As well as `npm run update-products`, I have written SQL scripts attached to both `npm run reset-database` and `npm run check-database`, which respectively clears or prints the schema of the database.

The `users` table is initially empty, but when a user signs up on the website, their password is hashed and a row added, along with their email and a unique id, to the table. This hashed password is later checked (using the `bcrypt` package), to allow users to later sign back in.

There is also a `sessions` database, which uses the user's unique id (stored as a cookie, or looked-up upon sign-in) to store and re-produce the user's cart.

I've gained lots of experience with SQL, and have put effort into database design and of handling data.

Dynamic pages - A

As mentioned above in the HTML section, I opted to use Pug (formerly Jade) as my templating system, and linked with Express for dynamic delivery of XHTML. I chose Pug as I liked its clean look, and I've wanted to learn how to use it for a while, plus it allows a nice separation of code into importable `mixins`, and pythonic `for` loops.

Using Bootstrap's CSS for my styling meant I had to completely re-write their snippets of HTML on its in Pug. I went further, splitting up the various items (eg `navbar` and `cards`) into a variety of `mixins`, to simplify my code massively and keep it very DRY.

When the Express Router receives a GET request for a page, a JS callback then renders the applicable a Pug template, passing in Javascript data objects as parameters. My Pug code then unpacks these parameters and checks their validity, before looping through given arrays and using the mixins to produce XHTML.

I have put in a lot of programming effort and become very fluent in using my chosen framework.

Depth

I've covered most of my "depth" in the sections above, but what took the most time (and I'm particularly proud of) is the highly-protected accounts system, alongside the cart system with the session storage and database.

Also some of the translucency on the sail images are nice.