Project Title: TradePizza

Group Members:

Jeremy Peter Borden Johnston, g4johnsu
Joseph Lee, g4joseph
Joshua Evan Aberback, g4aberba
Maja Gabrielle Kovats, g3kovats

Project Description:

Our project allows users to swap belongings they no longer need or want with one another, following the basic premise of "someone's trash is another person's treasure." This project is a very simple, but very straightforward, example of an online sharing economy, or more particularly a redistribution market. In redistribution markets used and pre-owned goods are passed from someone who doesn't want them to someone who does. Additionally, TradePizza encourages a no-money form of bartering to promote redistributive recycling instead of future consumption.

On TradePizza users sign-up, construct their user profile and begin posting "trades." Fundamentally, a trade contains an item you currently have, but no longer want, and a list of items you would be willing to trade your item for. TradePizza facilitates communication between users who want to trade their belongings, and it helps match users who have a complementary trade. Users are then free to contact each other and decide whether they would like to meet up and complete their transaction.

Project Architecture:

There are four fundamental sections in our software: Data Models, Views, Routes, and Middleware. Each are summarized below:

Data Models:

All document creation and retrieval from our MongoDb database is handled by our data models, which are implemented using Mongoose. Our software requires an Account model, for handling user and profile information, and a Trade model, for handling trade information. These files can be found in the '/models' folder. The Routes and Middleware sections of our app use these models, rather than the database directly, in order to access needed data.

Views:

All UI/UX considerations are addressed in our views, which are html pages generated using Jade. We made sure no data is processed within these templates. All templates can be found in the '/views' folder.

Our views incorporate multiple UI elements, including:

- Input controls: Buttons, list boxes, text fields, checkboxes;
- Navigational components: search bar, icons.

The pages included in our project consist of:

- Login, Registration;
- Homepage;
- Personal Profile, Profile Information Editing, List of User Profiles;
- Photo Upload for Display Pictures and pictures of Trade items;
- Posting a New Trade, List of All Existing Trades;
- Search, Search Results;
- General Administrative Controls, User Account Controls, Trade Controls.

Routes:

All routes served by our app are defined in routes/index.js. This file handles web requests, serves templates to the user and interacts with our models to retrieve and process data.

Middlewares:

Our Express middlewares are stored in the app.js file. This file loads and configures all modules required for our app including, for example, the Jade template engine, and Passport authentication. This file also takes care of error handling.

Security

We have addressed two security concerns during this assignment: authorization, and preventing DDoS attacks. For user authorization and authentication we used Passport, a middleware for Node.js. We used a username and password strategy as our first form of authentication. The user simply enter their username and password into an html form, their password is then hashed and stored in the database, as storing it in plain-text would result in a catastrophe if a hacker were to ever access the database. As a hashed item, it is essentially a one way function that turns the data into a unique string and it cannot be reversed. A single letter from the input returns a completely different hash so it is made safe that way. In addition to hashing, there is added salt to prevent two people from having the same password as it they would both be hashed the same way. With salt, we can have unique hashing where the passwords would be hashed twice by appending addition strings characters.

To prevent DDoS attacks the ‘ddos’ module for Node.js. Every request made by the same IP address is marked in an internal table using a ‘count’ parameter, like so:

```
{ host : <ip address>, count: 1, expiry: 1 }
```

If this count goes above a configurable ‘burst’ number then the expiry parameter doubles. If the count exceeds a pre-determined limit, then the request is denied. The only way for a user who has denied requests to continue is for them to let the expiration time pass, and when expiration hits 0, the entry is deleted from the table, and new requests are allowed like normal.

Optimization

Our code relies heavily on different node modules to do most of the work. Many of our routes simply invoke a lookup in the database, then render a jade template with the found document. Others invoke a lookup, then do a small modification to the found document and re-save it. Our static files are run through express.static, so caching is handled automatically. We can’t really think of much to do in the way of optimization.

Video:

https://youtu.be/ccrT2I3mEpg