Lucas Kitaev

Mr. Combs

Senior Projects

8 April 2018

Journal 6

Goals

- Research the Bootstrap framework and use it to style pages
- Research Font Awesome icons and add them to pages
- Set up database tables for users
- Create login page for users
- Create signup page for users
- Add ability to store JSON data in database
- Add ability to retrieve JSON data from database

Research

As final presentations are coming up, I wanted to focus on polishing the look and feel of the application. Since I lack an artistic eye, I opted to use a design framework for giving my website a professional appearance. The most popular design framework is Bootstrap, and is what I chose, although I had initially been apprehensive to doing so, since I wanted a better learning experience by minimizing my use of frameworks. Bootstrap makes it incredibly simple to make clean user interfaces for web applications, using a combination of SCSS (an extension of regular

CSS) and JavaScript. Bootstrap provides classes for styling just about every possible UI element one could want on their website. These classes are added to HTML elements, which allows the Bootstrap stylesheet to target them. Bootstrap styles can be overridden by one's own styles either by loading a stylesheet after Bootstrap, or by using the `!important` declaration after certain styles one wants to keep. Some Bootstrap components I used include alerts, buttons, collapsible menus, forms, input groups, list groups, and navigation bars (Otto and Thornton). The overall result is very pleasing, in my opinion. Bootstrap is automatically responsive, and will look good on both mobile and desktop.

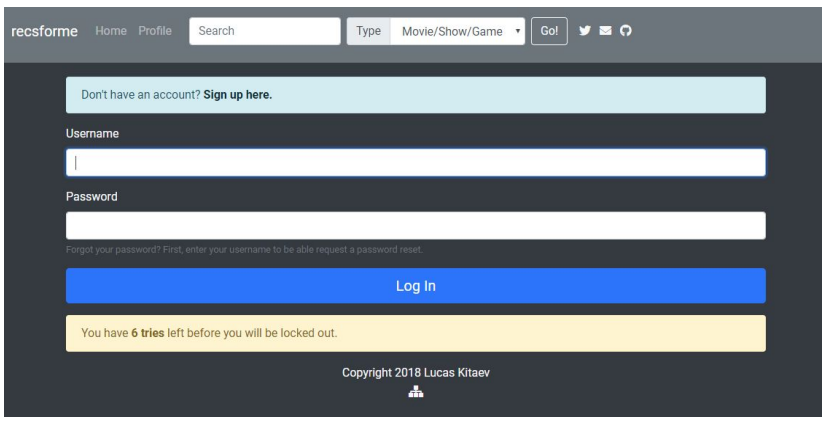
The other component I used to improve the look of my website, while not entirely necessary, was icons. Icons differ from regular images because they are loaded as a part of a stylesheet, or script. Icons made in SVG (Scalable Vector Graphics) format are able to, as the name implies, scale as vectors, meaning that no quality is lost even when upscaling to many times their original size. The icon library I went with was the Font Awesome free icons collection, including over a thousand icons. Icons are traditionally added as either `<span>` or `<i>` tags. The Font Awesome stylesheet will insert an icon into the element based on what classes it specifies. Font Awesome has four main classes for fonts: `fas` for solid icons, `far` for regular icons, `fal` for light icons, and `fab` for brand icons. For example, `fas fa-ambulance` will insert a solid ambulance icon ("Font Awesome 5 How to Use"). Together, I think Bootstrap and Font Awesome have made an improvement on my user interface over what I had previously.

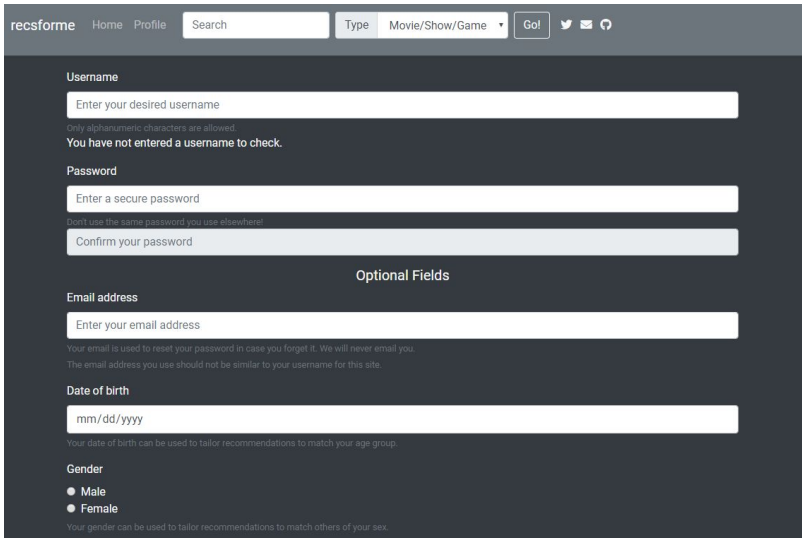
I also had to do some research on securing user passwords. Initially, I had hoped to not have to deal with this, since I could use OAuth 2.0 services that many popular websites offer,

however this turned out to be more complicated than I thought, so I went old school. The key to making user passwords secure is by encrypting them in a manner such that the original password cannot be recovered, even if attackers gain control of the database. Password encryption contains two pieces: hash and salt. The password hash is a sequence of bits that are output by a hashing algorithm. The most popular hashing algorithm is SHA (Secure Hashing Algorithm), however SHA on its own is not enough to protect against attacks that are increasingly easier, due to faster computer hardware. To solve this problem, PBE (password-based encryption) was created. PBE applies a number of other techniques in order to secure password, and to make the resulting hash even harder to crack, the algorithm is applied over many iterations (1000 is the absolute minimum recommended number). The PBE algorithm I chose is PBKDF2, which I configured to generate a key using SHA, and iterate 100000 times, which results in almost no speed loss ("Parameter Choice for PBKDF2"). Salt is a randomly generated sequence of bits added to a password before hashing. Upon account creation, each user is given a salt that will be unique to them. This salt is then used to ensure that any two users will never have the same password hash (this is known as a hash collision). Depending on the strength of the password, it could take thousands of years to crack even one, with these methods applied (Gupta). In the unlikely event my database is seized, I feel confident the stored passwords will be safe.

Accomplishments

- Login and signup pages with Bootstrap and icons





- Password encryption functionality

```
private String generateHash() throws
NoSuchAlgorithmException, InvalidKeySpecException,
DecoderException {
  char[] charPass = pass.toCharArray();
  PBEKeySpec spec = new PBEKeySpec(charPass,
Hex.decodeHex(salt.toCharArray()), iterations, 256);
  SecretKeyFactory key =
SecretKeyFactory.getInstance(HASH_ALGO);
  byte[] enc =
key.generateSecret(spec).getEncoded();
  return Hex.encodeHexString(enc);
 }
```

```
private String generateSalt() throws
NoSuchAlgorithmException {
  SecureRandom rand =
SecureRandom.getInstance(SALT_ALGO);
  byte[] randSalt = new byte[16];
  rand.nextBytes(randSalt);
  return Hex.encodeHexString(randSalt);
}
```

- JSON storage to/from database

```
public static ListData mapData(String json) throws
IOException {
  ObjectMapper mapper = new ObjectMapper();
  ListData data;
  try {
    data = mapper.readValue(json, ListData.class);
  } catch (JsonMappingException | JsonParseException
e) {
    data = new ListData();

System.err.println(Arrays.toString(e.getStackTrace()));
  }
  return data;
 }
```

```
public static String generateItem(String name, String id,
String type) {
  name = name.replace("'", "''");
  ListModel item = new ListModel(name, id, type);
  String ret;
  ObjectMapper mapper = new ObjectMapper();
  try {
    ret = mapper.writeValueAsString(item);
  } catch (JsonProcessingException e) {
    ret = e.getMessage();

System.err.println(Arrays.toString(e.getStackTrace()));
  }
  return ret;
 }
```

Reflection

As the date of final presentations approaches, it is becoming apparent that I will not have time to implement the actual recommendation part of the recommendation web app. I neglected the fact that I would have to have a user accounts system before I could even think of making a recommendation system, and along with that came way more work than I was expecting. I am pretty pleased with what I have though, and the main thing left to implement is the ability to remove groups from a user's database list; however, there are some miscellaneous tasks that I have made note of in the unlikely event that I find myself with extra time to fix things. I can also try to write more tests to ensure that the application behaves as expected, but more important is that I am able to deploy the app to the Microsoft Azure cloud (I've opted not to use Amazon Web Services, since I could not get it working on their platform). With that being said, I think my project will be in good shape for final presentations in two weeks.

Works Cited

"Font Awesome 5 How to Use." *Font Awesome*, Fonticons, Inc., fontawesome.com/how-to-use/web-fonts-with-css.

Gupta, Lokesh. "Generate Secure Password Hash : MD5, SHA, PBKDF2, BCrypt Examples." *HowToDoInJava*, 4 July 2016, howtodoinjava.com/security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/.

Otto, Mark, and Jacob Thornton. "Bootstrap Documentation." *GetBootstrap*, Twitter, getbootstrap.com/docs/4.0/getting-started/introduction/.

"Parameter Choice for PBKDF2." *Cryptosense*, 22 Mar. 2018, cryptosense.com/blog/parameter-choice-for-pbkdf2/.