Final Exam Study Guide

1. Vue

Consider the following code:

```
1  <template>
2  <div>
3    <h1>XKCD Browser</h1>
4    <div id="app">
5      <div v-if="loading">
6        <p>Loading...</p>
7      </div>
8      <div v-else>
9        <p>
10         <button v-on:click="firstComic">First</button>
11         <button v-on:click="previousComic">Previous</button>
12         <button v-on:click="nextComic">Next</button>
13         <button v-on:click="lastComic">Last</button>
14         <button v-on:click="randomComic">Random</button>
15       </p>
16       <h2>{{current.safe_title}}</h2>
17       <img v-bind:src="current.img" v-bind:alt="current.alt">
18       <p>{{current.alt}}</p>
19       <p><i>#{{current.num}}, drawn on {{current.day}} {{month}} {{current.year}}</i></p>
20       <h3>Add a Comment</h3>
21       <form v-on:submit.prevent="addComment">
22         <input v-model="addedName" placeholder="Name">
23         <p></p>
24         <textarea v-model="addedComment"></textarea>
25         <br />
26         <button type="submit">Comment</button>
27       </form>
28       <h3>Comments</h3>
29       <div v-for="comment in comments[number]">
30         <hr>
31         <p>{{comment.text}}</p>
32         <p><i>-- {{comment.author}}</i></p>
33       </div>
34     </div>
35   </div>
36 </div>
37 </template>
```

- What does the "v-if" do on line 5? And the "v-else" on line 8?
- Lines 10-14 setup an event handler. When are these triggered? Is there any difference between "v-on:click" and "@click"?
- Line 17 uses attribute binding twice. How is this different from data binding?
- What does the use of "prevent" do in line 21?
- Lines 22 and 24 use "v-model" to create data binding between JavaScript variables and DOM elements. What happens to the JavaScript variable when something is typed in the input? What happens to the input when some code changes the JavaScript variable?
- What does the "v-for" do in line 29? Normally we use attribute binding to ensure a "v-for" loop has a unique value for each item through the loop iteration, using ":key".

Consider the following code:

```
watch: {
    number(value, oldvalue) {
      if (oldvalue === '') {
        this.max = value;
      } else {
        this.xkcd();
      }
    },
  },
}
```

- What does the watch property do?
- When is the "number()" function called?
- What two parameters is it given?
- When would you use a watch function versus a computed property?

2. Vue CLI

Consider the following code:

```
1  <script>
2  import HomePage from ’@/components/HomePage.vue’;
3  import MyTickets from ’@/components/MyTickets.vue’;
4  import Admin from ’@/components/Admin.vue’;
5  import axios from ’axios’;
6  export default {
7    name: ’home’,
8    components: {
9      HomePage,
10     MyTickets,
11     Admin
12   },
13   async created() {
14     try {
15       let response = await axios.get(’/api/users’);
16       this.$root.$data.user = response.data.user;
17     } catch (error) {
18       this.$root.$data.user = null;
19     }
20   },
21   computed: {
22     user() {
23       return this.$root.$data.user;
24     }
25   }
26 }
27 </script>
```

- Lines 2–4 are importing Vue components and lines 8–12 list these components. What does this do?
- When is the function called ”created()” called on line 13?
- When we use a computed property, like in lines 21–25, when is this function recalculated?
- What configuration would be needed in ‘router/index.js‘ to use this view?

3. Node, Express, and Mongo/Mongoose

Now consider the following code:

```
1  app.get('/api/tickets', async (req, res) => {
2    try {
3      let tickets = await Ticket.find();
4      res.send({
5        tickets: tickets
6      });
7    } catch (error) {
8      console.log(error);
9      res.sendStatus(500);
10   }
11 });
12
13
14 app.post('/api/tickets', async (req, res) => {
15   const ticket = new Ticket({
16     name: req.body.name,
17     problem: req.body.problem
18   });
19   try {
20     await ticket.save();
21     res.send({
22       ticket: ticket
23     });
24   } catch (error) {
25     console.log(error);
26     res.sendStatus(500);
27   }
28 });
```

- On line 1 and line 14, why does these do endpoints use the same path, "/api/tickets"? Why doesn't Express get confused between the two?
- On line 1 (and line 14), what does "async" mean? What is this odd syntax with "(req, res) =>"? What does that mean? What is an asynchronous function? What is an anonymous function?
- On line 3, what does "Ticket.find()" do? Which tickets does it find?
- One lines 4–6, what status code (200? 403? 404? 500?) is sent back to the caller?
- What are lines 15–18 doing?
- In lines 16 and 17, where does "req.body.name" and "req.body.problem" come from?
- Why do we need to use "await" on line 3 and line 20?
- What is the difference between a GET, POST, PUT, or DELETE, if we are following REST API guidelines?

Consider the following code:

```
1  const express = require('express');
2  const bodyParser = require("body-parser");
3  const mongoose = require('mongoose');
4
5  // setup express
6  const app = express();
7
8  // setup body parser middleware to conver to JSON and handle URL encoded forms
9  app.use(bodyParser.json());
10 app.use(bodyParser.urlencoded({
11   extended: false
12 }));
13
14 // connect to the mongodb database
15 mongoose.connect('mongodb://localhost:27017/pagliaccio', {
16   useUnifiedTopology: true,
17   useNewUrlParser: true
18 });
19
20 const cookieParser = require("cookie-parser");
21 app.use(cookieParser());
22
23 const cookieSession = require('cookie-session');
24 app.use(cookieSession({
25   name: 'session',
26   keys: [
27     'secretValue'
28   ],
29   cookie: {
30     maxAge: 24 * 60 * 60 * 1000 // 24 hours
31   }
32 }));
```

- When we use "require" in lines 1–3 what is this doing?
- We're using a lot of middleware here. See for example lines 9–12, 20-21, 23-32. How does middleware work? What is it doing?
- In the code above, when we connect to the MongoDB database in lines 15–18, what do we give it a host name and port? Where is Mongo running? What is "pagliaccio" here?

Now consider the following code:

```
 1 const mongoose = require('mongoose');
 2 const express = require("express");
 3 const router = express.Router();
 4 const users = require("./users.js");
 5 
 6 //
 7 // Tickets
 8 //
 9 
10 const User = users.model;
11 const validUser = users.valid;
12 
13 // This is the schema for a ticket
14 const ticketSchema = new mongoose.Schema({
15   user: {
16     type: mongoose.Schema.ObjectId,
17     ref: "User"
18   },
19   problem: String,
20   status: {
21     type: String,
22     default: "open"
23   },
24   response: {
25     type: String,
26     default: ""
27   },
28   created: {
29     type: Date,
30     default: Date.now
31   },
32 });
33 
34 // The model for a ticket
35 const Ticket = mongoose.model('Ticket', ticketSchema);
```

- What is the ticket schema defining in lines 14–32?
- What sorts of things can Mongoose specify in a schema?
- When a property in the schema has a type of ObjectId, what is an Object ID? And what does the ”ref” mean?
- Can you define methods on schemas that operate on the documents in the database? (Hint, see the user schema for the ”Authenticating Users” activity or for Lab 5).

Now consider the following code:

```
1  // get tickets -- will list tickets that a user has submitted
2  router.get('/', validUser, async (req, res) => {
3    let tickets = [];
4    try {
5      if (req.user.role === "admin") {
6        tickets = await Ticket.find().sort({
7          created: -1
8        });
9      } else {
10       tickets = await Ticket.find({
11         user: req.user
12       }).sort({
13         created: -1
14       });
15     }
16     return res.send({
17       tickets: tickets
18     });
19   } catch (error) {
20     console.log(error);
21     return res.sendStatus(500);
22   }
23 });
```

- Why is this post request using the path ”/”? How does it eventually get the path ”/api/tickets”?
- What is ”validUser” doing?
- How are we able to access ”req.user” in line 5 without having to first find the user in the database?
- What does ”.sort” do on lines 12–14?

5. Authentication

Now consider the following code:

```
// generate a hash. argon2 does the salting and hashing for us
const hash = await argon2.hash(this.password);
```

- This code uses the argon2 library to salt and hash a password. What is a salt?
- What is a cryptographic hash function?
- Why do we salt and hash passwords?
- If an attacker is able to steal a plaintext password file, what can they do?
- If an attacker is able to steal a password file in which the passwords are hashed but not salted, what can they do?
- If an attacker is able to steal a password file in which the passwords are both salted and hashed, what can they do?
- We used cookies to keep a user logged in. What is a cookie? What is a session? How does this help a user to stay logged in?