COMS32500: Web Technologies

Elis Jones and Sonny Peng

- A for HTML
- A for CSS
- B for JS
- C for PNG
- C for SVG
- B/A for Server
- B/A for Database
- C/B for Dynamic pages
- for Depth (out of 40)

1 HTML

The site is delivered as XHTML and thus all HTML is forced to be structured correctly, i.e. all tags are closed. A variety of tags have been used to style our page differently according to different sections, but more special tags are used to complement the style further, such as `<video>`. Classes are used frequently in order to ease CSS styling.

2 CSS

Heavy use of the recently implemented: `display: grid;` option. This leads to far simpler positioning of elements within the site. It also leads to far more easily implemented responsive layouts. This done by simply changed the grid template option. i.e. :

```
@media (max-width: 900px) {
  body {
      grid-template-columns: 1fr;
    }
}
```

Although, the current support for this feature in CSS3 is at 87.4% of browsers in use. The trade-off for ease-of-use, particularly in responsive designs, is well worth it.

3 JS

There are extensive client side scripts. These scripts are split into multiple files, then using `module.exports` and `require` along with browserify are bundled into bundle.js. Allowing for more organised code, and enabling the `require` functionality. At the moment, there is ES6 syntax inconsistently sprinkled in, if necessary this may need to be changed to be consistent throughout the project. Google map key is requested from the Google API, user can use the map and the marker to navigate to the destination with ease. When a user clicks read-more the `src` of that element is changed to be the original higher resolution image, such that the cropped image is not used.

4 PNG

Simple image cropping has been implemented, in order to crop each of the images on the homepage to be identical.

5 SVG

The logo was constructed using Inkscape, with very basic techniques. There are also SVG illustrations of the boats and some wildlife which will be implemented for the final product.

6 Server

The server utilises Express, and many libraries which it offers passport, nodemailer, and crypto.

Passport allows the server to authenticate users who are logging in. This is done using the Local Strategy defined by:

```
passport.use(new LocalStrategy(..));
```

Such that when the server calls:

```
passport.authenticate('local',...);
```

The server authenticates a user. This works in tandem with `express-session` which allows the server to establish a user session. Once a user has been authenticated, all subsequent requests will be accompanied with an appended attribute `passport`, such that

```
if(req.session.passport){..};
```

Determines whether a user has logged in or not.

The nodemailer library allows the server to send automatic emails. For example, when a user visits the route `/verify` after they attempt to register, `transporter.sendMail(...);` will construct and send an email according to the options passed to it. The transport is configured in `nodemailer.createTransport(...);`.

The bcrypt library allows the server to hash and salt user passwords, as well as compare plaintext passwords with hashes. This is done using `bcrypt.hash(...);` and `bcrypt.compare(...);`.

The crypto library allows the server to generate random tokens, which are used for both verification, and password resetting, using `crypto.randomBytes(...);`

For server page delivery and security, we approach them with native express functionalities. Many basic security issues are dealt with by express automatically, such as invalid url prevention by redirect or ignore character: `"/."`, such functionality removed the threat from accessing security file located at server side or files those are suppose to be hidden.

All valid page is delivered by using express static file method:

```
app.use(checkdoubleslash, express.static(path.join(__dirname, '/public')));
```

Such approach eliminate the complication of deliverying correct and precise page, while providing extra url validations.

the nature of express is from top to bottom, thus the order of our url validation is crucial. However, whenever a request is sent, url must be valid before proceeding to extra security. Although express can handle all the universal url security validation, url containning // is still valid, a middleware is used:

```
var checkdoubleslash = function(req, res, next)
```

This is applied to all incoming URL request by injecting this to our static file provider.

For other url that does not violate this rule but can't be processed, we then passed on to our redirection and security middleware:

```
app.get("/:id", function(req, res, next)
```

Such method catches url with missing characters or symbols, although this may cause some security problem, but everything that is under `public` should be visible to any audiance, except url containing `/admin`. Strictly speaking, if all middleware fails then the url should not be considered as a healthy url, thus, all url reaching this point will be banned:

```
app.get("*", function(req, res, next) {
  res.send('Page not found', 404)
});
```

Porting has been done as well by using a small utility to check availability of the port, as for priority the port number should be 80, but if taken, the server will automatically listen to port 8080. One of the great functionality of Express is that it runs everything under `text/html` form with character-set to UTF-8. But in case anything can not be provided by static file, we added an extra insurance before sending redirection:

```
res.setHeader('content-type', 'text/html; charset=utf-8');
```

7 Database

`sql.js` deals with all sql queries, using `module.exports` to export functions to `server.js`. This utilises an embedded sqlite3 database, the structure of this database is illustrated in Figure 1

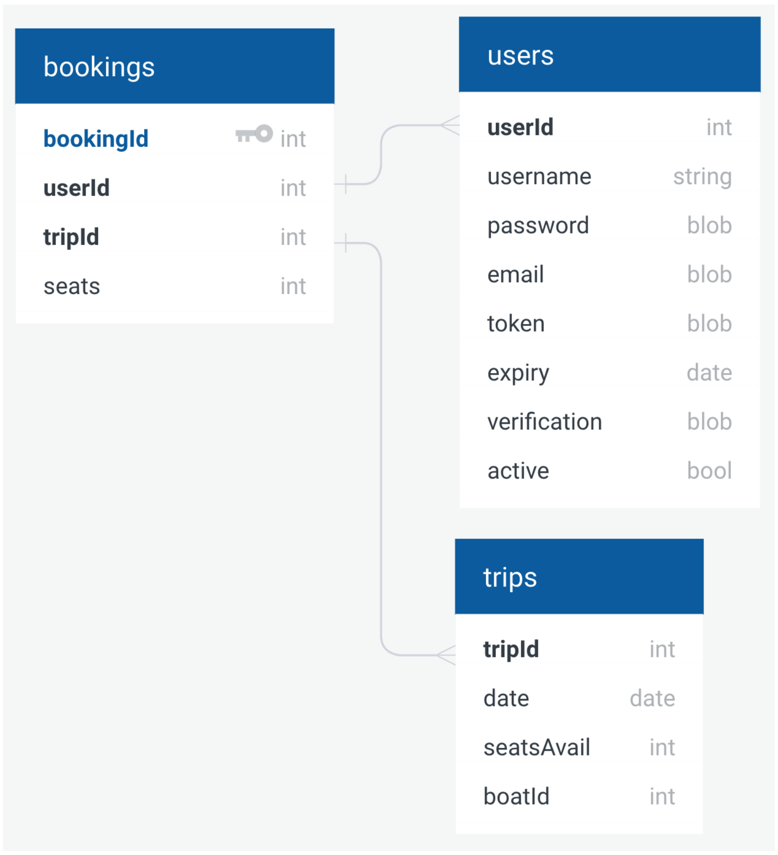

Figure 1: Diagram of database

8 Dynamic Pages

Templating is used for the my bookings page. Once a user is logged in, when the route to the my bookings page, the server queries their UserID with the database, information from the trips table is concatenated with a query to the bookings table, this information is returned to the client side in a JSON, which is inserted into a template in the `views.js` module, which injects new sections into the html. There is also an instance of templating in `server.js`, in the `/verify` and `/newpassword` routes an entire html file is templated, so that the query parameters from the get request, created by a user clicking the link from the email, are stored in hidden elements.