Assignment CSS+Node

In the first part of this assignment, you will employ *CSS* to make your splash and game screen look good.

In the second part of this assignment, you will implement a number of smaller items, that each do not take a lot of time, but round off your application and showcase additional abilities of Node.js and the browser.

An automatically generated PDF of this assignment is available here.

0. Preliminaries

Remember that this is a group assignment! Work efficiently as a team! Both team members **must contribute to the code** and **both team members must understand the code**. The group interview will focus on having the required functionality and showing off your understanding of the code.

Overview of deliverables and upload procedure

Task	Deliverables
1.1	CSS
1.2	CSS
1.3	CSS
1.4	→→→ upload 1.1 / 1.2 to 💡 Brightspace forum
2	Source code
3	Source code
4	-
5 (optional)	→→→ upload URL to 💡 Brightspace forum

This assignment requires you to finalize your application's source code and thus there is **no** PDF to submit.

Submit your code in the form of a zipped folder. Make sure that your code contains the necessary `package.json` file to install/run the code, i.e. it should be sufficient to run `npm install` and `npm start` to start the server. Any specific configuration parameters your code requires should be described in an accompanying `README` file.

*Note: we expect one zipped code submission, we do **not** want one code submission per task!*

The PDF and code have to be uploaded by one of the team members to 💡 Brightspace under **CSE Web assessment** (find the category your group belongs too) before the assessment session with the teaching assistant and before the ultimate assessment deadline. This means that the outcomes of all web technology assignments are uploaded to the same directory!

To pass this assignment, you must have employed the necessary CSS, use templating and include at least one client-side cookie. You pass if your app can deal with players executing the game as intended (already achieved in the previous assignment) **AND with players trying to make invalid moves.**

1. CSS

Now that we finally covered CSS, you will continue to work on your splash and game screens and style them with CSS. Do **not** use external libraries or preprocessing tools. If you are unsure how much styling of your game is required, take a look at our demo game, it contains sufficient CSS styling to pass this requirement.

Style the game so that it looks and works well on a laptop/desktop device, i.e. we are considering screen resolutions of ~1366x768 or higher. We are **not** concerned about apps for mobile devices.

As you might have already guessed, your CSS should reside in `myapp/public/stylesheets`. That folder already contains a `style.css` boilerplate file. In order to keep your code maintainable, you can for instance place all CSS that applies to both the game and splash screens in `style.css` and in addition create separate `splash.css` and `game.css` stylesheets for CSS rules specific to each screen. Note that stylesheets can contain other stylesheets.

1.1)

First, work on your **splash screen** and style the page with CSS according to the design you produced in Assignment 4. You can deviate from your initial design. To ensure that everyone learns the basics of CSS, we provide a list of **must-have** CSS properties. Your code must include **at least** one instance of each of the following:

//TODO: upate the list (CSS grid instead of position attributes)

- Pseudo-classes `:hover` and `:active`
- Pseudo-elements `::after` and `::before`
- Box model: margin, padding, border
- At least two different position attributes, e.g. `position:relative` and `position:absolute`
- At least one CSS animation and one CSS transition.

👉 Hints:

- CSS examples are plentiful on the web; you can adapt them to your own needs. Make sure though, that you **understand** the different CSS rules you are adding and are able to **explain** their effect. If in doubt, prefer simpler CSS rules, often the same effect can be achieved in a myriad of ways.
- If you are looking at CSS examples from CodePen or similar platforms for inspiration, be aware of the fact that CSS extensions such as Sass exist. Do **not** use those, in this course we make use of *plain CSS*.
- If you are unsure about what combination of colors to use for your game, take a look at https://www.happyhues.co/ for inspiration.
- If you are in need of icons for your game (dice, chess figures, etc.) take a look at https://thenounproject.com.

1.2)

Next, tackle the CSS for your **game screen**. The look of the game screen should be coherent with the splash screen. The two are likely to share basic CSS settings (colors, fonts, etc.) - make sure that you are efficient and do not duplicate existing CSS code. Check out the CSS imports mentioned above.

Your code for the game screen must have at least one instance of each of the requirements listed in 1.1).

👉 Hint:

- In a board game, you may want to assign elements (tiles, cards, etc.) a different status depending on player actions, e.g. a game token moves from tile A to B. A simple but effect manner to achieve this effect is to assign different CSS `classes` depending on a tile's state. In JavaScript, `document.getElementById(id).className += " anotherClass";` adds a CSS `class` to an element.

1.3)

To ensure that your players are aware of the screen size limitations (i.e. the game works well on a larger screen), use media queries to **alert players** if their screen resolution is below a sufficiently large minimum. How exactly the alert looks like is up to you. What exactly the screen resolution minimum is, depends on the game you implemented and your specific CSS rules. There is no need to actually try your app on different physical devices, Firefox (other major browsers have similar tooling) has a Responsive Design Mode tool that provides good simulations of various devices.

1.4)

Once you have completed the CSS of your app, head over to CSE1500's 💡 Brightspace, go to *Discussions* and then once more the forum **BOARD GAME APP DESIGNS**. **Find the thread you uploaded your wireframes too.** Add your implemented design screenshots to your 💡 Brightspace discussion forum thread. *Does your implementation deviate significantly from your initial design?* If so, write a paragraph comparing the two.

Feel free to browse your colleagues' designs and implementations and comment on them!

2. Templating

When you started out and generated the boilerplate code (previous assignment, task 1), we asked you to set as *view engine* `ejs`. Let's now make use of that view engine for the **splash screen**. If you followed our assignment instructions, so far, not a lot is going on with it; it should visually look appealing and have a *Play* button, but that's about it.

One last requirement for the splash screen is to show a number of statistics about the games played, games completed, etc. Templates offer us a simple way to *inject* those numbers into our splash screen.

There are many view engines available, `ejs` is one of the more common ones. All we have to do now is to create a `splash.ejs` file in folder `myapp/views/` as this folder is the default one the view engine is looking for content. Initially, `splash.ejs` is simply a copy of your `splash.html` file. Add a few template tags for the game statistics in the right place. If you are unsure what exactly to do, take a look at our demo game's splash.ejs file. Then, rewrite a few lines of `app.js` to incorporate the view engine and alter the route to `/`; it should now use the `res.render` call. This code snippet can help you:

```
//code snippet, not a complete piece of code!
var app = express();
...
//only the route to / needs to be changed
app.set('view engine', 'ejs')
app.get('/', function(req, res) {
    //example of data to render; here gameStatus is an object holding this
information
    res.render('splash.ejs', { gamesInitialized:
gameStatus.gamesInitialized, gamesCompleted: gameStatus.gamesCompleted });
})
```

Note: in the lecture we covered Ajax, which is an alternative approach to templating for our use case of presenting game statistics. Both technologies are relevant in practice, it is important to understand both of them.

3. Cookies

Let's quickly practice cookies. Use a **client-side cookie** to log how often a user has accessed your game. Report this information to the user on either the splash or game screen.

4. Cleaning up

Ensure that your app works as intended in two major browsers.

Check the ESLint (or any other linter) output: it should help you to spot easy-to-make mistakes, which in turn should help you write better code.

Ensure that your code contains the necessary `package.json` file content to install/run the code, i.e. it should be sufficient to take your `myapp` folder, and run `npm install` and `npm start` to start the server. Any specific configuration parameters your code requires should be described in an accompanying `README` file.

🎇 🎇 🎇 🎇 🎇 **Well done!** If you have reached this point, you have a fully functioning game.

You now have three options:

- Be glad you made it and call it a day!
- Deploy your game on a publicly accessible server and let others play it!
- Make a screencast showing off your game!

5. Optional: game deployment / videotaping

Short-term deployment

To make your game accessible to the public, it needs to be deployed somewhere - `locahost:3000` only goes so far.

For short-term deployments (a few minutes/hours), e.g. to show off your game quickly, ngrok is an easy solution. You can run your server on localhost and expose it through a public URL. *Please do **NOT** run any*

privacy-sensitive apps this way, as all traffic passes through ngrok servers!

Production deployment

ngrok is not a solution for an actual deployment. Heroku (among other platforms) is. Heroku is a cloud platform that has a free account tier, suitable for web applications without a lot of traffic. You can deploy a Node.js application following their instructions.

//TODO: check whether Heroku instructions still hold Deploy your app in Heroku according to the instructions linked above, but first make these changes:

1. In `app.js`, replace `http.createServer(app).listen(port);` by `http.createServer(app).listen(process.env.PORT || 3000);`
2. In `myapp` folder, create a new File called `Procfile` with this line: `web: node app.js`
3. In the instructions you will see that to *Prepare the app* they clone an existing app to test. Instead of doing that, in that step you have to make sure that you have a local `git` repository in `myapp`. If that is not the case then install `git`, go to the `myapp` folder and type the following from the terminal:

```
git init
git add .
git commit –n "First commit"
```

If you want to deploy your game elsewhere, feel free to do so! MDN has a good article on deploying a Node.js/Express application to production.

Videotaping

As an alternative, make a screencast of your game and upload it to a video hosting platform such as vimeo or YouTube.

Once you have deployed or videotaped your app, head over to CSE1500's 💡 Brightspace, go to *Discussions* and the forum **DEPLOYED/VIDEOTAPED GAMES**. Add the URL of your deployed app or video.

🎮 👾 And now, it's time to play some games of your fellow study colleagues! 👾 🎮