Assignment 1: Theatre Blocking

Assignment Handout

Due date: Thursday, February 28th by 3pm.
This assignment is done **individually.**

Overview

In this assignment, you will be combining Product Management skills with some web programming to create a small web app.

Although you will be doing some coding and using web technologies, the focus is not on how well you can code, or on how fancy you can make your project look.
It will be on how you **define your users**, work out a **plan** with them in mind, and then develop a solution based on your **well-thought out assumptions**.

In the sections that follow, we will give you a specification of a particular app.
Most of your grade (80%) will come from following our specifications. However, to potentially receive full marks, you will have to add some **enhancements** on top of the work you do. These must also follow some rules we give you and should tie in to the course material.
This is in line with the Arts and Science guidelines for A+: "*Strong evidence of original thinking; good organization; capacity to analyze and synthesize; superior grasp of subject matter with sound critical evaluations; evidence of extensive knowledge base.*"

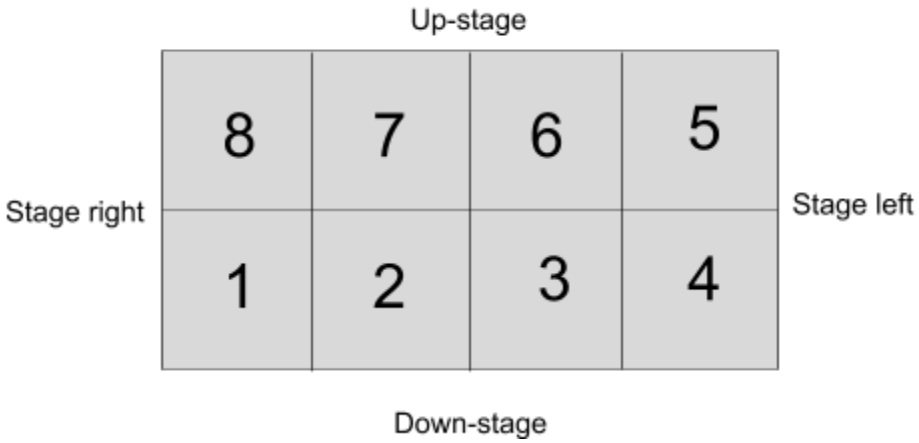
Starter Files/GitHub Repo

You will be submitting your assignment entirely on GitHub.
Private GitHub repos will be created in our course GitHub Org, and you will be responsible for ensuring that your repo is up to date (with your submission on the **master** branch) at the time the assignment is due. You will be responsible for any delays introduced if you do not join the course GitHub org in time.
We have provided you with a bunch of starter files on the master branch of the repo to help you get started, with instructions on how to get things to work (particularly helpful if you haven't done any web programming before).

Specification: Theatre Blocking

When putting on a play in a theatre, it is important that everyone involved knows where the actors are at any given point in the script.
Theatre Blocking is the concept of providing information (text and/or pictures) about where the actors in the cast are supposed to be on stage during a particular part of the script.
Every theatre production has their own method for blocking, but there are some common elements that we'll apply for this assignment.
In order to properly ***block*** a scene, every **line** in the script should be mapped to the **location** of the actors at that point.

For the purposes of this assignment, we will assume that actors can only be in one of 8 positions on the stage, and that **at most one actor** can take up a particular spot on stage:



Suppose we have two actors, actor A and B, and the following script:

"See you later A!I wonder where B is going.."

The blocking for these separate parts of the script are highlighted and indicated below:

A in position **2**, **B** in position **6**
A in position **3**

Note that actor B is not on stage in the second script part (green) - this means that they have left the stage between these two parts of the script - we do not need to specify the details of what happened between parts for the purposes of blocking the scene.

Also, note that the blocking doesn't say *who* is saying the line - just where the actors are on stage.

The ability for the cast and the crew to have every scene blocked allows for a much easier time during rehearsal - everyone knows the overall structure of the stage positions during any particular point in the script.

You are tasked with creating a prototype **web app** for a theatre company that would like to be able to block their production and share their blocking among the different members of the production team.
As a ***smart*** **software engineer**, you aren't just going to start coding right away and hoping that they like what you have. You are going to use your Product Management skills to ensure that you know **what** you are building, **who** you are building it for, and **why** you are building it.

There are many different people involved in a production team, and you can't build a web app that works for all of them right away. You need to be selective, and define a Minimum Viable Product (MVP) - a product that has just enough functionality to be accepted by the user.
For the baseline requirements of this assignment, you will focus on two groups of users: **Actors** and **Directors**.

When working with clients, you may often receive vague explanations of your project requirements, and this should not surprise you! This case is also no different, the only information you are given to start from the users is:

"Directors should be able to modify the blocking for all parts of the script, and actors should be able to see the blocking for their parts."

With this small (but powerful) piece of information in mind, it's time to outline your tasks for this assignment.

Your Tasks

Part 1: Product Management and User Analysis

Given the user statement above, you will go through the initial stages of Product Management, including **Goals** and **Requirements.** (you should review the lecture notes and worksheets from class to remind yourself of what is required as you go through the tasks).
Remember that you only need to consider the Actor and Director users and creating the parts below.

1. Create an **objective statement** for your app.
2. Create **three personas** of potential users of the app.
3. Create **three user stories** for the app.
4. Write down the **acceptance criteria** for the user stories.

Add these elements to the README.md file under the headings:
objective statement
personas
user stories
acceptance criteria

In part 2, you will work with an example implementation of this app. Although you can see the implementation, your user stories and acceptance criteria can still go beyond what you see in the web browser - it is up to you to decide what the actor and director users will accept, including features that may not currently show up in the web browser.

Part 2: Building a prototype Web App

Note: *Do not start Part 2 until you have Part 1 done! You need to know what you are building before you start building it.*

You will create a prototype web app using **Python Flask** and **Docker**, and deployed to the web using **Heroku**.
Note that we will be testing using **Google Chrome**, so please ensure you are using that browser when making your assignment.

Instructions to get started

1. Clone the starter code from your repo. Make sure that you can run the Docker container and that the site loads on **localhost** in Chrome.
    - Note that any time you make a change to your code, you will have to rebuild the Docker container for those changes to take effect.
    - If you only open the HTML files in your browser **without opening it through localhost**, the functionality will not work - it needs the server to run properly.
2. There are two html files (actor.html and director.html) in the **app/static** folder that each represent the actor and director views, respectively. You can access them as files of a static directory, using a URL such as **localhost/actor.html**
3. Have a look at the overall structure of the **app** folder:
    - **static** contains the HTML, CSS, and JavaScript files. You will only have to edit the Javascript files.
    - **main.py** is the Python Flask server file. You will edit this file on the server side.
    - **script_data** is the folder where the scripts and their blocking **persist** on the server. You can think of this as the 'database' of the scripts.
    - **actors.csv** A mapping of actor numbers to actor names

Chrome JavaScript Console

To test the JavaScript functions found in the file, open up the JavaScript Console in Google Chrome (View -> Developer -> JavaScript Console).
If you want to test functions for actor.js, navigate to the actor.html page first - same with director.js to director.html

In the next section, we will go over the files and how they are used in the app.

File descriptions

If you are new to web programming, you don't have to worry - there is not too much code in the JavaScript files, and we have included a lot of comments to help you out.
Most importantly, we have provided any functions that have to do with **User Interface (UI)**. For example, adding a block to the page is as simple as calling the `addScriptPart()` function.
Note: You cannot change these UI functions in the JavaScript functions. - you will lose a lot of marks if you do.

actor.js is linked to actor.html, and there are functions that allow you add and remove script parts on that page. There is also a function that shows an example of a call to a server - a **'fetch'** call. The one in `getExampleBlock()` shows what happens when you click the Get Example Block button in the HTML.
This fetch call is a **GET** request to the '/example' route in the Python Flask server. Look at **main.py** to see what that route returns to the client (the web browser). It is just an array of information to place into the example block.
The **fetch** call might look a little strange if you haven't seen it before - the overall structure (the '*then()*' keyword) isn't something you have to worry about (it's a JavaScript Promise, if curious). The only part of a GET fetch call you should be thinking about is the part with **`jsonResult`,** which holds the JSON body of the response from the server. You can now access it in JavaScript (in the example call, it is simply an array that is indexed to add the script part to the browser window
For the director.html view, a **director.js** is also linked, with some similar but also some different elements. In this case, there are also some example blocks, which also contain textboxes to *change* the blocking of a script. You must fill in both the getBlocking and changeBlocking functions with **fetch** calls to the server. changeBlocking requires a slightly different request - it is a **POST** request, that will send along with it some data - the entire blocking of the script, which is sent to the server so that it may update the text files properly. It will be your job to fill in the part of the request that provides specific data to the server. How will you decide on the data to send back and forth between the server and client? Read on..

Serializing using JSON data
As mentioned in class, data can be **serialized** using JSON, which is a standard in web development because it can easily be converted between JSON string and JavaScript object (or Python dictionary in the case of Flask).

The functions in the JavaScript files are given to you as is, and since the functions that do UI all have specific parameters, you will have to ensure that your JSON objects have the right properties to make your app work (this is similar to what happens in real life - you may get an API that you can't (or don't have time to) change, so you have to work with what you're given and change only what you can change.
Your JSON must also **limit redundant information** (i.e. duplicate data over multiple properties). You will be assessed on how you make your JSON objects.

JSON Task:
As part of your submission, provide **examples** of JSON objects you will use to serialize your the data (you will have to keep reading this handout to see what other information you may need to include in your JSON objects).
One for the GET requests that provide script blocking information, and one for the POST requests (the JSON you are sending to the server). Save them as files in:

- `script_get_data.json`
- `script_post_data.json`

You should properly format the JSON and make sure it is valid:
https://jsonformatter.curiousconcept.com/

In your `README.md` under a `#JSON files` header, explain your choice for your JSON serialization and why you chose it.

JSON API and Script files
Now that you've decided on how you're going to serialize your data, you should create a way to move this data around. There are two things to think about: **requesting** some JSON from the browser, and **sending** it from the server.
In the Python Flask server, we can send a JSON object by first making a Python dictionary or array, and calling **jsonify()** to make a JSON representation. Since you've decided on how your JSON format looks, you should be able to send it back to the request origin (the user's browser)
A request from the browser is made in javascript that lives on the browser side from the fetch() calls explained above.

The script files persist in the **script_data** folder contain text files that the theatre has given you (again, another example of having to work with what the user gives you). They contain the script ID number, a newline, the script text in its entirety on **one line**, followed by a newline, and then followed by the blocking for each part in the format:

<part_number>. <char_start>, <char_end>, <list of actors-position...>

Each line has the number of the part of the script, the start character and end character of the full script text where the part is contained, and a list actor names and their positions from 1 to 8 (or **0** if the actor does not appear in that part).
In actor.html, you will notice that an *Actor Number* is required to see the blocking for a particular actor. These can be found in the **actors.csv** file.

You can use Python file I/O functions to read and write to these files on the server side (file explained below). If you mess up the script files, you can always re-download them (you shouldn't commit messed up files to the repo).
Just note that you must access the files from the '/app' directory. (i.e. /app/actors.csv).

Flask Server (main.py)
On the server side of things, all routes for the app can be found in the **main.py** file.
You are responsible for modifying the **/script** GET and POST routes. These will include things like sending and receiving JSON, and reading and writing script files. Make sure you understand the purpose of the routes before you start programming.
Test your routes often as you working and build them up to ensure they're working correctly every time you add a feature.

Postman
If you want to test that your routes work, you can use an app called Postman. It lets you make calls to your JSON APIs without using a browser so you can test them more easily without worrying about the frontend.

Enhancements!

20% of your grade will come from enhancements that you will create on top of the baseline requirements and specification. This can be anything that adds to the assignment, and could involve product management, serialization, or anything else we've talked about in class.
You can potentially look at different user groups that work in theatres to help start you off:

- https://en.wikipedia.org/wiki/List_of_theatre_personnel

Notes, this is not a web dev class, so the following will not count as an enhancement

- Authentication will not count as an enhancement
- UI will generally fanciness will not count for enhancements - no React or other frontend templating engines should be used. If you do, small UI elements that create value for your users are much better than fancy elements

Important Information - Please Read!

- Do not change the headers/body of any of the given JavaScript UI functions
    - If you want to add some UI elements, make separate functions that add them. Regardless, we are not grading you on UI, so minimal additions are encouraged - this is an MVP, after all.

Handing in your work

For the TAs to grade the functionality of your app, it **must** be deployed to heroku. Get a free account on http://heroku.com and follow the instructions in our starter README.md to create an app and push your Docker container to it. You will lose many marks if you do not deploy your app.

Your utorid should appear in the app name. You **must include** the URL to your heroku app in your README.md.

The code in your Github **master branch must match the deployed app** - you will lose a lot of marks if this is not the case.

Grading

You will be graded on the following criteria.

- **User stories** (25 marks): You need to clearly write down the user stories you have decided to implement.
- **JSON example files & explanation** (10 marks): You should
- **Code and Functionality** (40 marks): Your code must work and be deployed on Heroku. We will test it and look at your code to see your implementation of the requirements.
- **README.md** (5 marks): Your README should be properly formatted and organized.
- **Enhancements.md** (20 marks): You need to clearly explain
    - What enhancements you have implemented
    - How to use them.
    - Why you have decided to add them, and how they relate to the course material