# COMP 4513 Assignment #1: React

*Due Saturday Nov 6th at midnight*
*Version 1.0 (Oct 18, 2021)*

## Overview

This assignment provides an opportunity for you to demonstrate your ability to generate a single-page application using React. **You can work alone or in groups of two or three on this assignment. Please don't ask for a group of four.** I have added a looking-for-a-group channel to the Discord for the course if you want to join a group or need additional group members.

## Beginning

Use the same techniques covered in the second React lab for this assignment. The githib repo for the assignment is: https://github.com/mru-comp4513-archive/comp4513-f2021-assign1.

## Submit

You must host your site on a working server platform that I can access. Since your React application is a static site, any host that supports static files will work. Heroku, GitHub Pages, FireBase Hosting, Netlify, Render, and Surge, all provide relatively trouble-free solutions for hosting React apps.

**You must upload the production build from create-react-app.** The `npm run build` command will create a production build of your application in a folder named `build`. You can find instructions for deploying to numerous environments at https://create-react-app.dev/docs/deployment/.

I will also need access to the source code, so you will have to make it available to me on GitHub. **The development version, not the production version, must be uploaded to GitHub.** If on a private repo, be sure to invite me.

So, to submit your assignment you must send me an email with three bits of information: 1) the URL of your React app, 2) the github repo URL, 3) the names of the people in your group.

## Grading

This assignment is worth 17% of the course grade. The grade for this assignment will be broken down as follows:

| | |
|---|---|
| Visual Design, Styling, and Usability | 20% |
| Programming Design + Documentation | 10% |
| Functionality (follows requirements) | 70% |

## Requirements

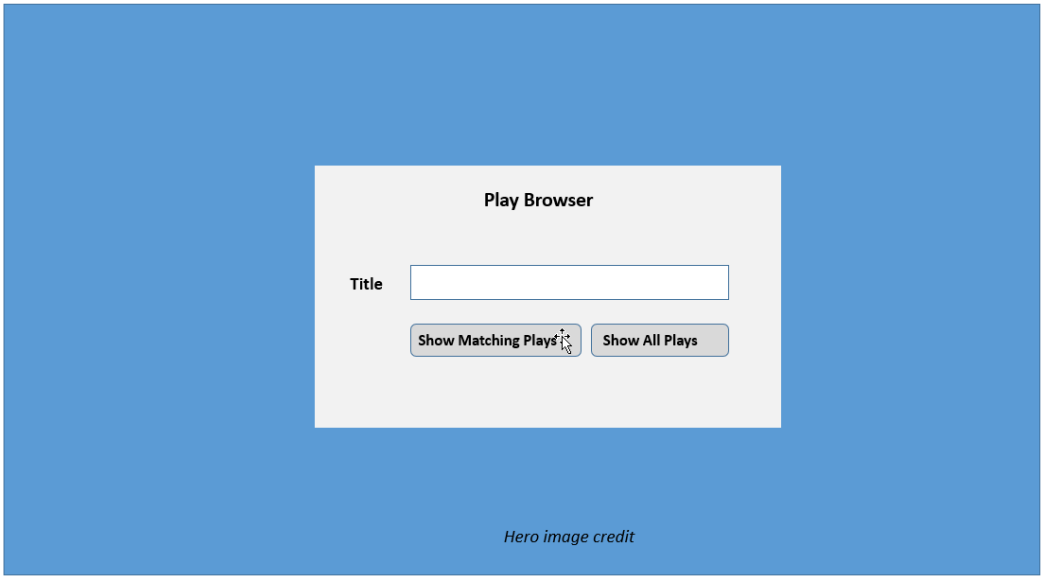
1. You must make use of the `create-react-app` starting files and structure. You must deploy a production build of the application to your live server; the development version must be on GitHub. The filename for your starting file should be `index.html` (`create-react-app` does this automatically).

2. This assignment consists of three main views (remember this is a single-page application, so it is best to think of the assignment consisting of different views). In the remainder of this assignment, I have provided a sketch of the basic layout of each view (some views have multiple subviews).

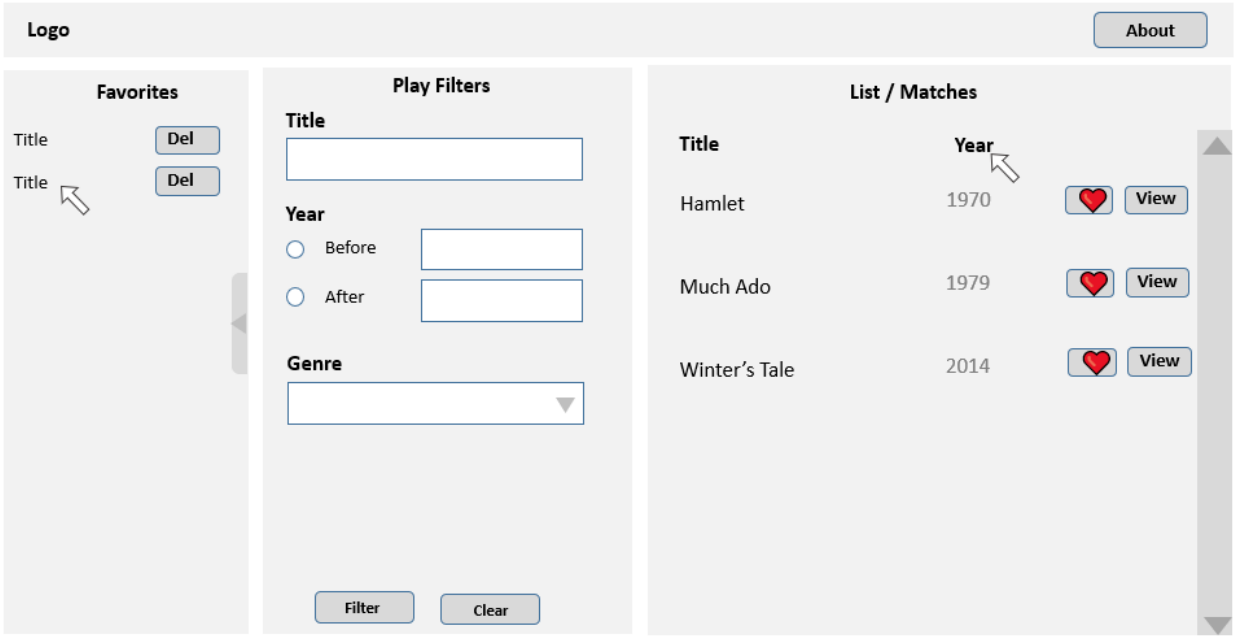
   These sketches do NOT show the precise styling (or even the required layout); rather they show functionality and content. I will be expecting your pages to look significantly more polished and attractive than these sketches! A lot of the design mark will be based on my assessment of how much effort you made on the styling and design.

   You can change the layout if you wish. If you want your play list to be horizontal on the bottom and the favorites to be vertical on the right, you can do so if you think that would be best … though I may not necessarily agree about the usability of your choices. I'd rather see you try something different in the layout/design than simply slap together something that is just a slightly improved version of the layout in my sketches.

3. If you've used JS or CSS that you've found online, I expect you to indicate that in your documentation and in the About view. Be sure to provide a URL of where you found it in the documentation. Failure to do so might result in a zero mark, so give credit for other people's work!




4. When your assignment is first viewed, it should display the Home View. It consists of a hero image (and image that fills the entire browser width or up to a specific very wide value, say 1800 or 2200 pixels). You could use unsplash.com as a source for your image, but be sure to provide credit information somewhere on this page. In the middle of the page should be another rectangle with a place where the user can enter a title search string, as well as two buttons. Both will take the user to the Default View. If the first is clicked, then the play list will be filtered on the play title; if the second is clicked all the plays will be displayed.
5. You must include some type of CSS transition and animation effect on this page. The more interesting it is, the more marks to be gained. I would also like you to use react-transition-group for this effect. See `https://reactjs.org/docs/animation.html` for more information.




This view should be broken down into a lot of different hierarchical components. Remember: the reason we are using React is that it allows us to break a complex application down into smaller components.

6. **Header**. The header should have a logo and a link/button to an About dialog. The logo should return to the Home View.

7. **Play List / Matches**. Displays a list of plays. The URL for the API is:

   ```
   https://www.randyconnolly.com//funwebdev/3rd/api/shakespeare/list.php
   ```

   This API returns an array of play objects with the following structure:


```
{
    "id": "hamlet",
    "filename": "hamlet.json",
    "title": "Hamlet",
    "likelyDate": "1602",
    "genre": "tragedy",
    "wiki": "https://en.wikipedia.org/wiki/Hamlet",
    "gutenberg": "https://gutenberg.org/ebooks/1524",
    "shakespeareOrg": "https://www.shakespeare.org.uk/e
    "synopsis": "Hamlet sees his dead dad's ghost, pret
    "desc": "Hamlet is considered among the most powerf
    adaptation by others'. It was one of Shakespeare's
    list of the Royal Shakespeare Company and its prede
    Goethe and Charles Dickens to James Joyce and Iris
},
```


   The play list should initially be sorted alphabetically on title. Initially, display all plays or filtered by name depending on what happened on the Home View. The Title and Year column headings should be clickable: when clicked they sort the plays by that field. The blue boxes represent small poster images (see below for more info).

   The API will take some time to retrieve this data. Display a loading animation (there are many free animated GIF available) until the data is retrieved. Simply display the image just before the fetch and then hide it after fetch is successful.

Clicking on the View button or the title will take the user to the **Play Details** view. Be sure to set the mouse cursor to indicate they are clickable. As you will see below, the Play Details view will potentially allow user to view the play text. However, only a few plays have text available. Indicate this status visually, perhaps by styling the title differently, or add an icon (probably best).

To improve the performance of your assignment, you **MUST** store the content retrieved from the play list API in local storage after you fetch it (see Exercise 10.11 in Lab 10). Your page should thus check if play data from this API is already saved in local storage: if it is then use it, otherwise fetch it and store it in local storage. This approach improves initial performance by eliminating an early fetch in future uses of the application. Be sure to test that your application works when local storage is empty before submitting (you can empty local storage in for instance Chrome via the Application tab in DevTools).

8. Clicking on the Heart button will add the play to the Favorites list. When a play is added to the favorites list, its title should be added to the list. If the user clicks on the title in the favorites list, it should switch to **Play Details** view for that play.

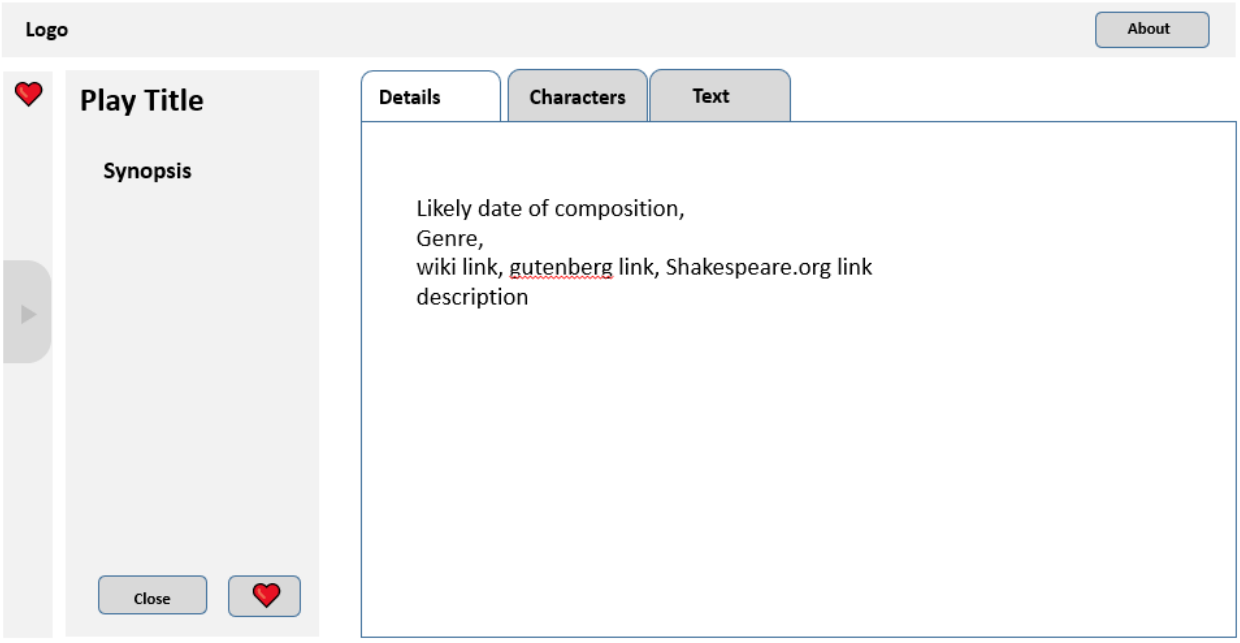
   The user should be able to toggle the visibility of the favorites panel (though initially should be visible). Don't be scared of using icons instead of text. There must be some type of transition effect (e.g., animation or fade), rather than just instantaneous visible/invisible.

   When the user clicks on the Del button or icon, remove the play from the list (and from state). Clicking on the Del button must remove the play from favorites in state. Ideally, there will be some type of animation/transition on the play title to provide visual feedback that the play is being deleted. Also, a play can only be added once to favorites.
9. When the user selects the About link/button, it should display some type of modal dialog/window with information about assignment. Include group members, github link, technology used, any third-party source code, etc. Sometimes it makes sense to make use of existing components rather than re-invent the wheel. It is important to know how to integrate existing components. Thus here you **must** make use of an existing React modal-dialog component (e.g., react-modal or react-modal-dialog).
10. **Play Filters**. Allow the user to easily filter the list of plays. User should be able to find plays whose title contains anywhere within it whatever was entered into the title input box. The user should be able to filter the play list by the likely date field. They should be able to specify either: plays before a year, after a year, or between two years. Similarly, the user should be able to filter the play list by the genre (either comedy, tragedy, or history). Also provide way to remove filters (that is, return to all plays being displayed). When the user clicks the filter button, the play list should update.

    If no matching plays are found, notify the user within the Play List/Matches area.

    The list of plays should scroll while the rest of the page stays. You can do this easily with the CSS `overflow` property.

    These filters are not mutually exclusive. For instance, a user should be able to search for plays with the word "hen" in it with a likely year between 1990 and 2000 and a genre of comedy.




11. **Play Details**. Displays detailed information for the selected play. There will be three tabs: one for the play details (already retrieved in the play list API), one for the play, one for the characters in the play, and one for the text of the play.

    Most of the play details are found within the list API from page 4. The characters and play text come from another API, whose URL is (where xxxx is the id of the play):

    `https://www.randyconnolly.com//funwebdev/3rd/api/shakespeare/play.php?name=XXXX`

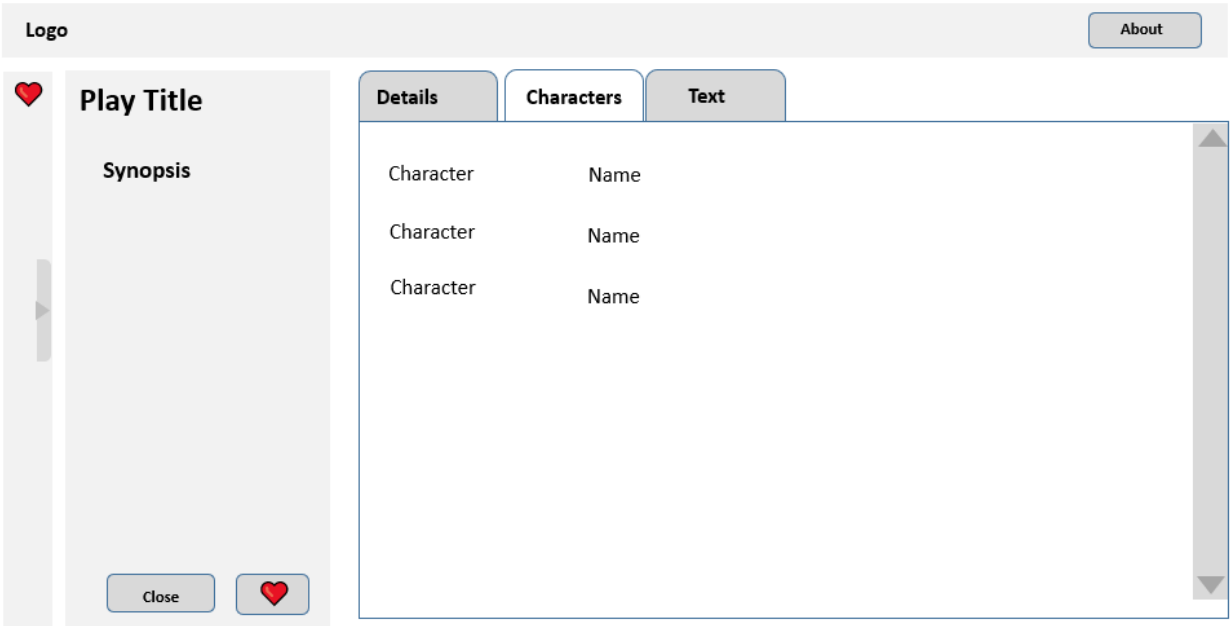
    Only some plays are available from this API. If the `filename` field from the list API (see page 4) is not empty, then the play's characters and text are available. If the `filename` field is blank then disable the Characters and Text tabs.

    The API will take some time to retrieve this data. Display a loading animation until the data is retrieved. Simply display the image just before the fetch and then hide it after fetch is successful.

    To improve the performance of your assignment (and reduce the number of requests on my server), you must store each retrieved play text data in local storage after you fetch it from the API. Your page should thus check if this play data is already saved in local storage: if it is then use local data, otherwise fetch it and store it in local storage. This approach improves initial performance by eliminating this first fetch in future uses of the application. Be sure to test that your application works when local storage is empty before submitting (you can empty local storage in Chrome via the Application tab in DevTools).

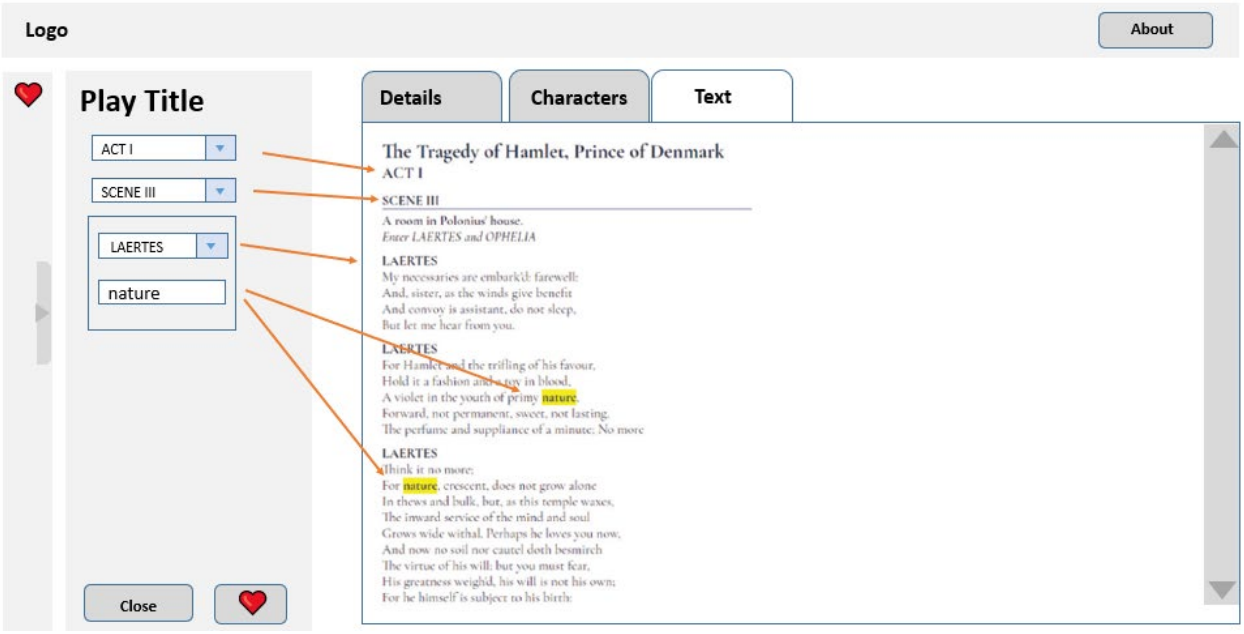
    I expect the play detail data to be nicely formatted and laid out sensibly.

    The Close button will return user to the Default View. The Add to Favorites will add play to favorites list. The Favorites panel is still available here but should default to closed.




Raw Parsed


```
{
  "title": "The Tragedy of Hamlet, Prince of Denmark",
  "short": "Hamlet",
  "persona": [
    {
      "position": 1,
      "player": "CLAUDIUS",
      "desc": "king of Denmark"
    },
    { … }, // 3 items
    { … }, // 3 items
  ],
  "acts": [
    {
      "name": "ACT I",
      "scenes": [
        {
          "name": "SCENE I",
          "title": "Elsinore. A platform before the castle",
          "stageDirection": "FRANCISCO at his post. Enter to him BERNARDO",
          "speeches": [
            {
              "speaker": "BERNARDO",
              "lines": [
                "Who's there?"
              ]
            },
            { … }, // 2 items
            { … }, // 2 items
```




**Text Tab**. If the play has this information, the Text tab will display the play's text. A Shakespeare play contains multiple acts; each act contains multiple scenes. (To reduce the size of the downloaded files, not all acts and scenes have been included).

Populate the three `<select>` elements from this data. Whenever the user changes one of the select list choices or enters text in the filter textbox, update the display of the text. Add an ALL PLAYERS option to the top of the character `<select>` list.

The filter button will highlight all occurrences of the user-entered text in the play and only show the speeches from the specified player. This will require wrapping the text in, for instance, `<b>` or `<span>` elements. You might find using `RegExp` helpful here for the text-matching.

I have provided you with some sample markup that you can generate along with styling for that markup. You can of course go your own way with the styling, but the client (me) expects the font of the play text to be a serif font.