

COMP 4513 Assignment #2: React

Due Monday April 7th at midnight

Version 1.0 (March 4, 2025)

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 on this assignment. Please don't ask for a group of three or four.**

Beginning

Use the same techniques covered in lab11b, lab11c, and lab14b for this assignment. You will either be consuming the web APIs from assignment 1 or using supabase's React API to retrieve the data, so you will still need your assign 1 database hosting to be working (i.e., remember to periodically re-awaken your supabase project and/or your server host from assign 1)

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. GitHub Pages, FireBase Hosting, Netlify, Render, and Surge, all provide relatively trouble-free solutions for hosting React apps. If DevOps is your thing, you could use Google Cloud Platform or AWS to create a virtual servers. Or you could use any third-party server that has ftp access.

You must upload the deploy version of Vite using the `vite build` command. You can find instructions for deploying to numerous environments at <https://vitejs.dev/guide/build>.

I will also need access to the source code, so you will have to make it available to me on GitHub.

If you are hosting on github pages, you will need two Github repos: one with the development version, and one with the production version! 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 development github repo URL, 3) the names of the people in your group.

Grading

This assignment is worth 18% of the course grade. The grade for this assignment will be broken down as follows:

Visual Design, Styling, and Usability	25%
Programming Design + Documentation	5%
Functionality (follows requirements)	70%
Extras	+10%

Requirements

1. You must make use of the Vite starting files and structure. The filename for your starting file should be `index.html` (Vite does this automatically).
2. You are encouraged to use a third-party CSS library. This can be Tailwind (it would be good to have that on your resume), but you can use any other CSS library (or component library) you'd like.
3. This assignment consists of two 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.

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 25% design mark will be based on my assessment of how much effort you made on the styling and design.

4. If you've used JS that you've found online, I expect you to indicate that in your documentation AND in your about dialog. 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!

5. **You have two choices when it comes to how you are going to retrieve the art data.** One way, is to simply consume (fetch) the APIs from your first assignment. This will mean you have to ensure your glitch/render server is up and running when you test. If your API isn't providing you with your expected data, you'll need to modify it on your server. The other way is to call the supabase methods directly in your React application (see Exercise 14b.9 which demonstrated how to access supabase in vanilla JS). Essentially, you would copy the relevant route code from your first assignment. The second option *might* be simpler from a hosting standpoint, but I could be wrong.

Example

```
// consuming assignment #1 APIs
useEffect( () => {
  url = "http://somecrazy.glichURL.com/api/artists";
  console.log('fetching ... here to check if I've gone infinite');
  fetch (url)
    .then( resp => resp.json() )
    .then( data => { setSeasons(data); }
}, [] );
```

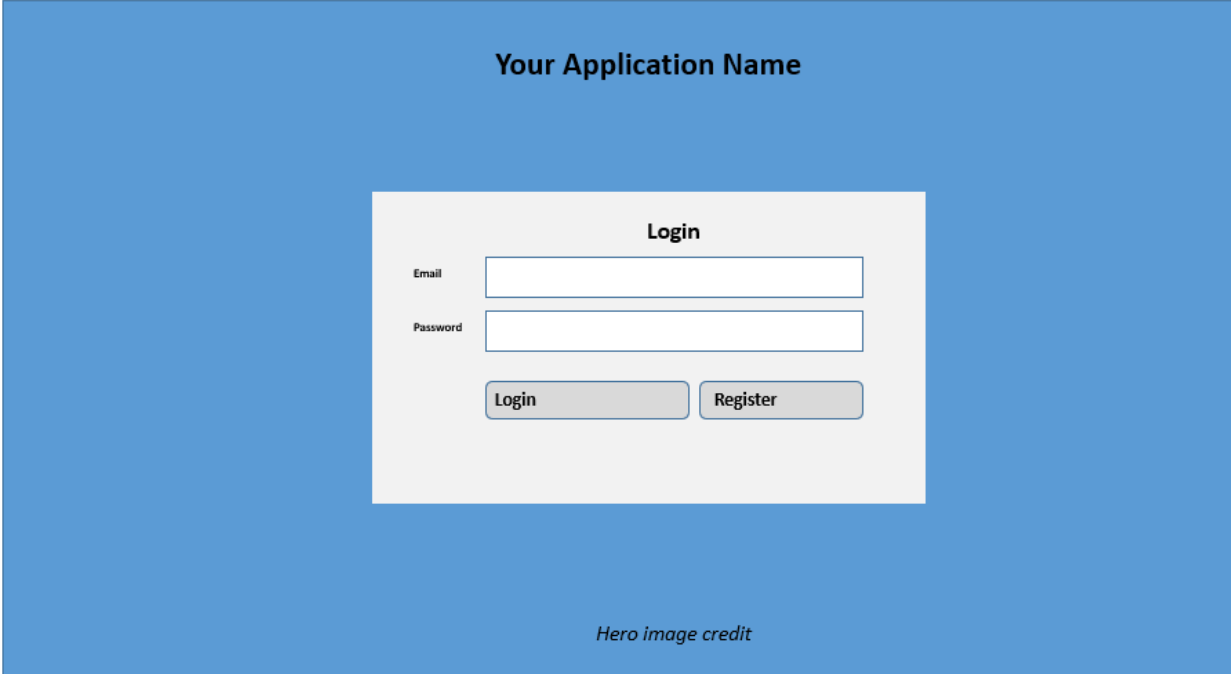
VERSUS

```
// accessing supabase directly in react
useEffect( () => {
  selectArtists();
}, []);

async function selectArtists() {
  console.log('getting from supabase ... here to check if I've gone infinite');
  // uses the same API as your assign 1 solutions
  const { data, error } = await db.from('artists')
    .select('*');
  if (error) {
    console.error('Error fetching artists:', error);
    return;
  }
  setArtists(data);
}
```

Be careful you don't get infinite loops with your requests! Use console.log messages and be sure to check them frequently!

Login View

A mockup of a login view. It features a solid blue background. At the top center, the text "Your Application Name" is displayed in a bold, black, sans-serif font. In the center of the page is a white rectangular box with a thin gray border. Inside this box, the word "Login" is centered at the top. Below it, there are two input fields: the first is labeled "Email" and the second is labeled "Password". At the bottom of the white box are two buttons: "Login" on the left and "Register" on the right. At the bottom center of the blue background, the text "Hero image credit" is written in a small, italicized font.

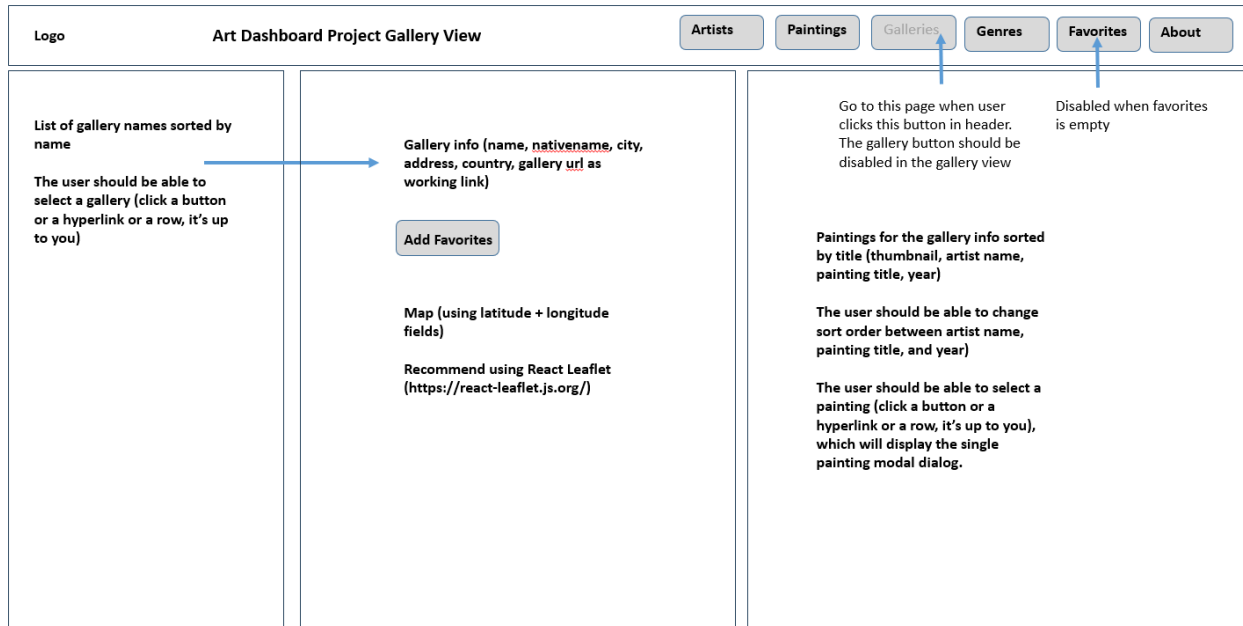
When your assignment is first viewed, it should display the Login View. It consists of a hero image (an image that fills the entire browser width or up to a specific very wide value, say 1800 or 2200 pixels). You could use unsplash.com as a source for your image, but be sure to provide credit information somewhere on this page. In the middle of the page should be another rectangle with a login form.

If we had enough time available, I would have asked you to implement the login authentication using supabase auth. But as there won't be enough time, so you will not need to implement actual authentication, but will just simulate it by jumping to the correct View when the login button is clicked.

If you decide to expand on this project during the spring or summer, implementing this functionality will make your assignment a fantastic portfolio piece!

After the user clicks "login", go to either Artist View, Gallery View, or Painting View (it's up to you)

Gallery View



When first displayed, display a list of galleries.

Until there is something in the favorites collection (see below), the Favorites button should be disabled. If there are favorites, then enable the Favorites button.

While I used text buttons in the image, feel free to use icons instead.

The user should be able to select a gallery (click a button or a hyperlink or a row, it's up to you). When they do, then display information about the gallery, a way to add a gallery to the favorites, and a map of the gallery (Recommend using React Leaflet via <https://react-leaflet.js.org/>).

Don't just dump the gallery info fields on the screen. Look at how information is presented in other web sites: notice that not everything is label:value ... instead, fields that don't need a label, won't have a label; when labels are needed, look at how they are typically formatted differently than the values. A lot of the visual design marks will be decided by how much effort you take here.

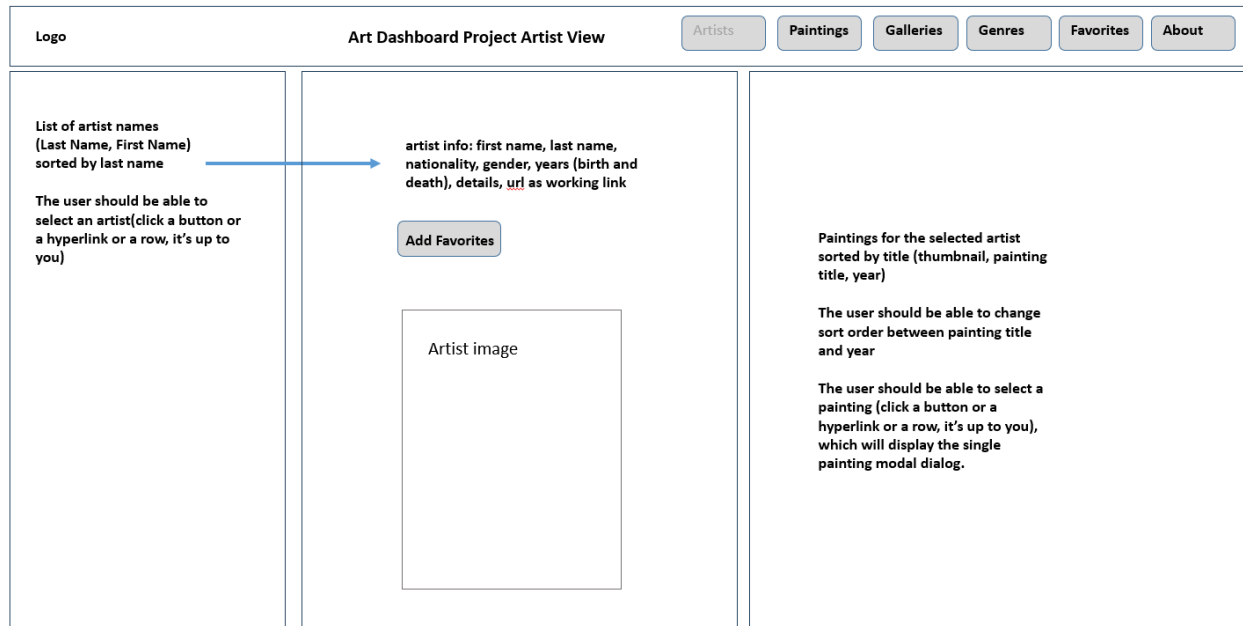
Also display a list of paintings (thumbnail, artist name, painting title, year) in that gallery, sorted by title.

The user should be able to change sort order between artist name, painting title, and year, probably by clicking a column heading or some type of icon.

The user should be able to select a painting (click a button or icon or a hyperlink or a row, it's up to you), which will display the single painting modal dialog.

The user should be able to add the current gallery to the favorites list. Provide some type of feedback, that the favorites list has been modified.

Artist View



When first displayed, display a list of artists.

The user should be able to select an artist (click a button or a hyperlink or a row, it's up to you). When they do, then display information about the artist and a way to add the artist to the favorites.

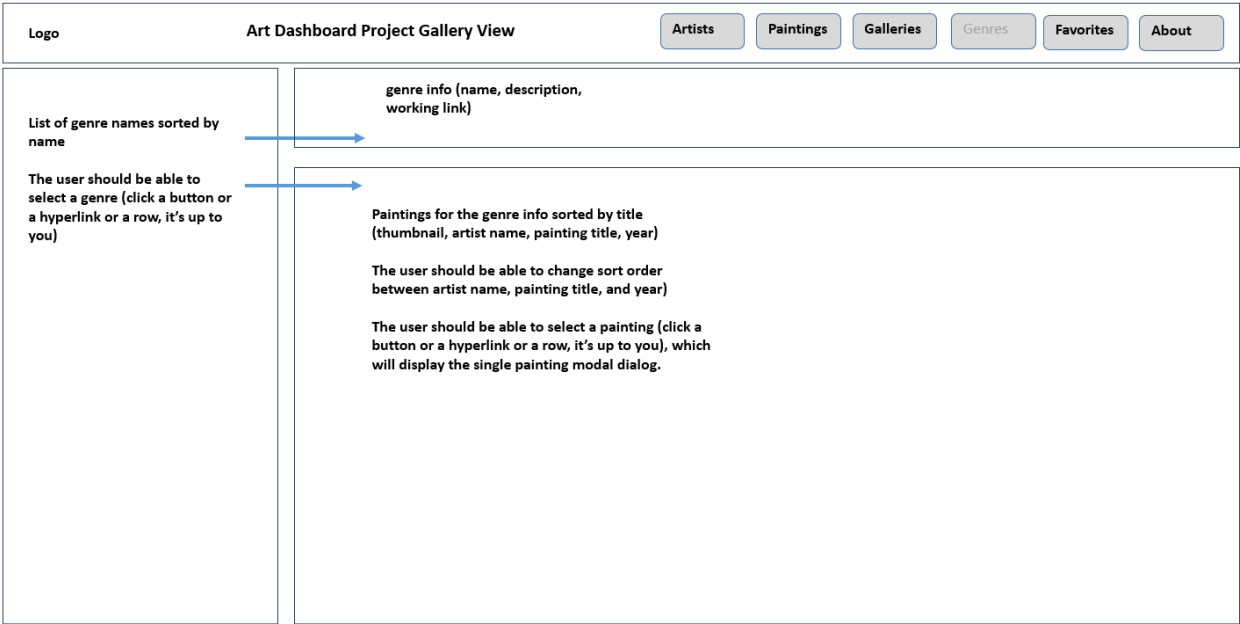
Also display a list of paintings (thumbnail, painting title, year) for that artist, sorted by title.

The user should be able to change sort order between painting title and year, probably by clicking a column heading or some type of icon.

The user should be able to select a painting (click a button or a hyperlink or a row, it's up to you), which will display the single painting modal dialog.

The user should be able to add the current artist to the favorites list. Provide some type of feedback, that the favorites list has been modified.

Genre View



When first displayed, display a list of genres. Selecting a genre will display info about it as well as a list of paintings that have that genre.

The user should be able to change sort order between artist name, painting title and year, probably by clicking a column heading or some type of icon.

The user should be able to select a painting (click a button or a hyperlink or a row, it's up to you), which will display the single painting modal dialog.

Painting View

Logo

Art Dashboard Project Paintings View

Artists

Paintings

Galleries

Genres

Favorites

About

Painting Filters

Title

Artist

Gallery

Year

Less

Greater

2017

Clear

Filter

Paintings that match filter, initially all paintings sorted by year
(thumbnail, artist name, painting title, year, gallery name, medium, width and height)

The user should be able to change sort order between artist name, painting title, gallery name, and year)

The user should be able to select a painting (click a button or a hyperlink or a row, it's up to you), which will display the single painting modal dialog.

The paintings view allows user to filter paintings by a variety of criteria. To make coding easier, assume these criteria are mutually exclusive.

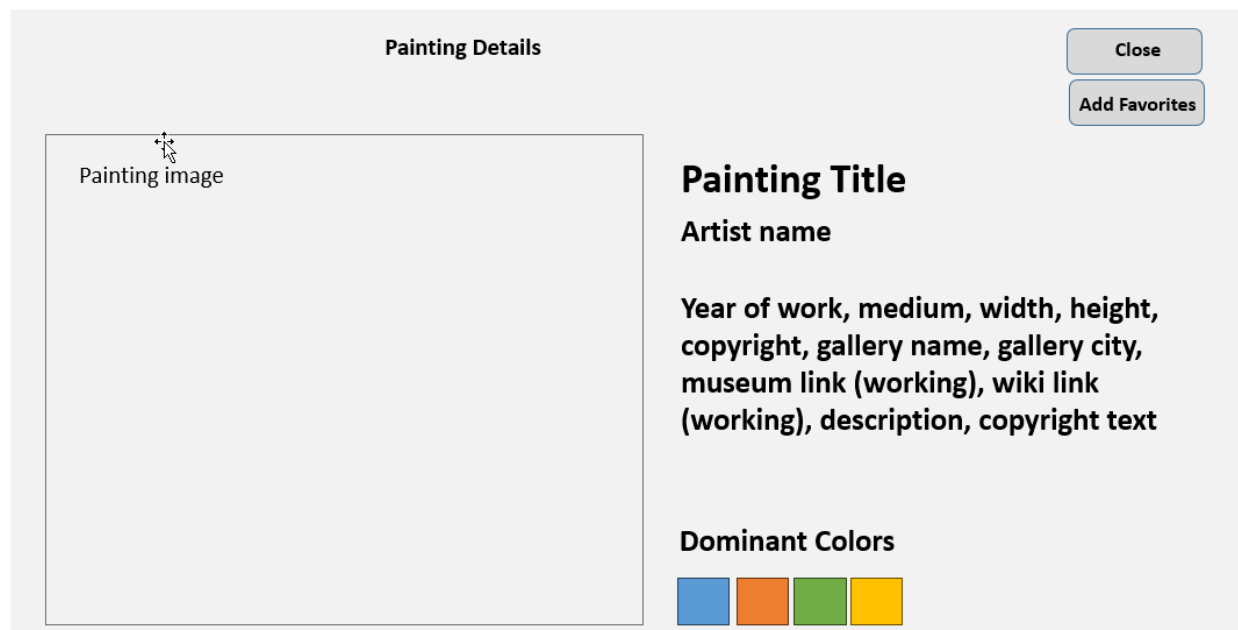
The artist, genre, and gallery filters should be select lists with sensible sorted content.

Just like the gallery painting view, you should be able to change sort order and to select a painting that displays the single painting modal dialog.

Note that changing the sort order must preserve the current filter.

Clicking clear will remove any filters (go back to displaying all the paintings) and reset the filter screen.

Circuit Details Pop-up / Modal Dialog



When the user selects a painting, they should see in a modal dialog / pop-up, additional information about the painting.

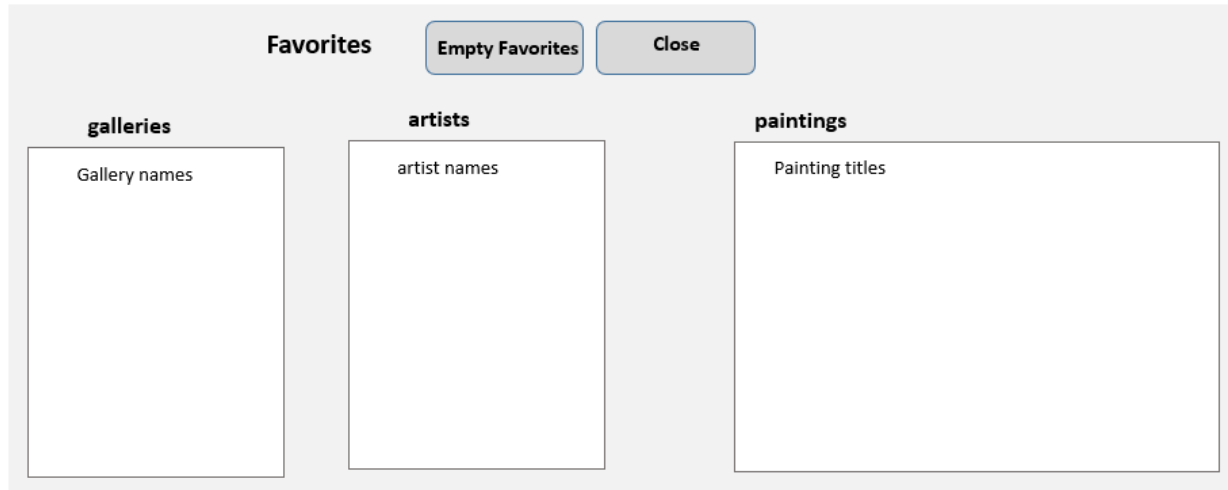
The Close button will close the dialog, while the add to favorites adds the painting to the favorites.

Sometimes it makes sense to make use of existing components rather than re-invent the wheel. You may decide to use a third-party modal/dialog component (e.g., react-modal or react-modal-dialog) or use one within a CSS library you are using (e.g., such as ant, bootstrap, etc). Alternately, if you look on youtube, you can find lots of examples of how to implement a modal dialog in React and CSS.

Some component libraries provide alternatives to modal dialogs that you can use instead. For instance, the Ant Design component library has a Drawer component, which is like a modal dialog, but is displayed from an edge rather than in the middle (see <https://ant.design/components/drawer>).

The JsonAnnotations field has a series of dominant color definitions; display these as colored boxes with the appropriate color; set the title attribute of the box to the color name.

Favorites Pop-up / Modal Dialog



Simply display the three lists. The dialog can be closed or the favorites can be emptied. Ideally, provide a way to remove individual items from the favorites.

Other Guidance

1. To make your application feel more responsive, retrieve the paintings, genres, galleries, and artists for the respective lists just once.
2. The API will take some time to retrieve this data the first time. 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.
3. Painting images are on the Cloudinary CDN. The URL for a given painting is as follows:

`https://res.cloudinary.com/funwebdev/image/upload/SIZE/art/paintings/FILENAME`

e.g.,

`https://res.cloudinary.com/funwebdev/image/upload/w_200/art/paintings/square/001020.jpg`

where **SIZE** is **w_###** or **h_###** where **###** is the width or height in pixels; **FILENAME** is the value of the `ImageFileName` property in the painting object array. If you don't provide a size, then you will get the original image, which is very large (often 2000 pixels wide or more).

The URL for a square version of the painting is as follows:

`https://res.cloudinary.com/funwebdev/image/upload/SIZE/art/paintings/square/FILENAME`

e.g.,

`https://res.cloudinary.com/funwebdev/image/upload/w_75/art/paintings/square/001020.jpg`