# Encyclopedia Conway:
# A React Website


Olivia Larson

Hendrix College

A thesis submitted for the degree of

*Bachelor of Computer Science*

Hendrix College 2023

This thesis is dedicated to
my cat Meg.
She sat with me through it all

# Acknowledgements

I would like to acknowledge the Computer Science department at Hendrix College and their willingness to work with me through everything.

# Abstract

Becoming a new resident of a city is difficult, especially when it's one's goal to adapt to the city's culture. The best way to adapt is to have more information. For those moving to Conway, Arkansas both permanent residents and those here for college, this is a problem they face. How does one find out about events, thrift shops, local restaurants, road work, walk-in clinics and voting information? Through using React, a dynamic website framework, APIs, and a database the Encyclopedia Conway project is created. This website is for any Conway resident to use. This project also includes a tutorial for others to create their own website for their own city.

# Contents

# Chapter 1

# Introduction

How does one know that they have joined a community? Is it when they know which restaurants and businesses are available, or when they know what events are taking place, or maybe it's when they know how to navigate around? When one moves to a new place, how do they find out this information? I know for myself, I search for it online. However, it's not as simple as it sounds. Today the internet plays an important role in the dissemination of information; however, the amount of information it holds is almost to its detriment. It is on the user to decide how much effort they are willing to put in to gather information as well as determine if they think it is trustworthy.

Is it possible to foster community engagement using an online platform? Cities all over the United States are striving to adapt to online communication, so creating an accessible user-friendly website that consolidates information is a top priority. With Conway, Arkansas experiencing a growing residential and college student population, it is difficult to maintain a strong central community. That is why I have created a website that consolidates information into one location. This information includes upcoming events, road work, second-hand shops, walk-in clinics, voting info, and entertainment ideas.

So why Conway, Arkansas? Being a resident of Conway for my entire life I have truly seen the city grow in population, business, and opportunity. Conway is a city with a rich history and culture that every resident should know about. There is a lot of opportunity for more community growth but it's difficult to grow a community when residents don't know how to engage with it. Although there's information available, it is often hard to find or even know what to search for. After careful thought, I have

narrowed down certain aspects of city life that I believe to be useful both for new and old residents ranging from voting to supporting small locally owned businesses. The goal of this project is not merely to offer information but to catalyze community engagement, reinforce local governance, and drive economic growth.

# Chapter 2

# Background

## 2.1 An Overview

In this section, I will give you a general history of Conway. I hope that this reveals an interesting story about an interesting city. All cities have an origin story, many of them starting off in small rural areas. I also hope this history reveals how much Conway has grown and continues to grow. Next, I will talk about why community engagement is important and why this website is necessary. I will then talk about previous research that has been done about using technology to foster community engagement. Finally, I will talk about how my project fits into this scope and is necessary for a city like Conway, Arkansas.

## 2.2 Conway then and now; a brief history

Located in central Arkansas, Conway was founded in 1871 by Colonel Asa Robinson the chief engineer for Union Pacific in Little Rock[12]. The Union Pacific faced financial difficulty and in lieu of a paycheck, they offered Robinson a 640 acres of land[13]. A railroad soon came to the town which helped grow Conway Station, the name of the town before it became Conway[12]. Its starting population was only 5,700 (it's now grown to over 67,000)[9]. After the town started to grow, agriculture and colleges started to be established in the town. Starting with Hendrix College which moved into the area in 1890, followed by Central Baptist College in 1893, and finally, the University of Central Arkansas was founded in 1907[12]. A lot has changed in

the last 152 years for Conway. Business and population have boomed in Conway. The city is now known as the City of Colleges and more colloquially known as the City of Roundabouts. As of 2021, there were almost 24,000 college students in the city[14]. The biggest industries in the area are Health Care and Social Assistance, Retail Trade, and Accommodation and Food Services[14].

## 2.3 Why community engagement is important

### 2.3.1 Individual level

Community engagement is beneficial on a multitude of levels. On the individual level, it can foster a sense of belonging. People naturally want to fit in with their environment by being a part of a group. Group membership brings numerous benefits including reducing stress, providing resources, and social support as well as many other benefits.[4] Community engagement can also empower citizens to have a voice in decision-making processes ranging from their neighbourhood to their city. Empowerment often leads to civic engagement. This includes participation, such as voting, volunteering, and advocacy. All of which has been positively correlated with positive well-being.[3]

### 2.3.2 Community level

Community engagement also benefits the community. Through the collective, bigger impacts are made. A great example of this is voting, in which every vote dictates government and policies. Another example is mutual aid or collective action. This is when terrible events happen and a community comes together to help one another out either through monetary resources or time and effort. An engaged community is more likely to participate in local governance, which can lead to better policies and services. Community engagement also helps communities identify and address local issues. Working together as a large collective, practical solutions can be found that improve the quality of life for the community.

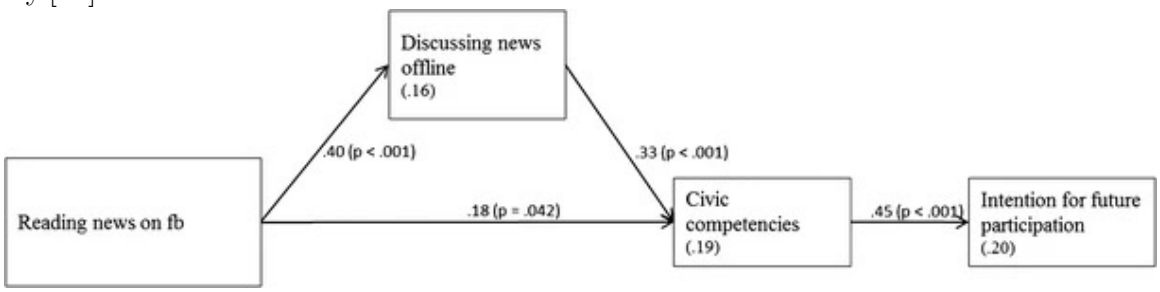
### 2.3.3 How we can grow community engagement with technology

It is important to note that I am not claiming that my website will make community engagement appear out of thin air. However, I am hoping that it will serve as a good first step for a resident of Conway to start learning about the community as well as having access to a variety of resources that are centrally located. I am hoping that knowledge of local events, entertainment spaces and opportunities, and voting will allow Conway residents to feel empowered and like like a true member of the group.

## 2.4 Previous Research

### 2.4.1 Social Networking Sites and Civic Engagement

Finding previous work that looked to address the same issues I was hoping to was difficult. Thus, I broadened my scope from informational websites to Social Networking Sites (SNS) being used for civic engagement. I found a study that looked into whether Facebook could foster civic engagement. In this study, the participants were Italian high school students (ages 14-17). Through several surveys, researchers asked participants if they read about news on Facebook, shared the news online, and had civic discussions offline with friends and family, and participant's competence for civic actions, and future intentions for civic engagement. Researchers found that there was a positive correlation between reading news on Facebook and civic competency. However, there was a key mediator, and that was discussing news offline with friends and family.[11]



#### 2.4.1.1 Figure Explained

This figure shows us that information can lead to more civic competencies and intention for future civic engagement. The mediator helps explain the pathway between the independent variable and the dependent variable. In this case, the independent variable is reading news on Facebook and the dependent variable is civic competencies. Discussing news offline explains this positive correlation.

#### 2.4.1.2 Facebook group vs Encyclopedia Conway

I believe the findings from this study are exactly what I hope to accomplish with my project. One of the best things people can have is information. So why not just use a Facebook group to foster community engagement?

Not everyone has a Facebook account and to exclude people from information based on if they want to be on social media is unfair. Also, while Facebook is great for many things, it can be difficult to navigate and find the information that you are looking for. It is my goal to have people use my website as a source of information. Sharing information about events happening in the area as well as voting information about what elections and policies are up for a vote is something that people want to know about.

## 2.5 My project

While there is a government website for the city of Conway it is a little lacking, especially with events that might be relevant to most residents. While there is information on the internet it is all scattered. That is why with this project I am bringing it all together into one location. Finally, because of the additional tutorial I am providing, anyone can create a React website for their city.

# Chapter 3

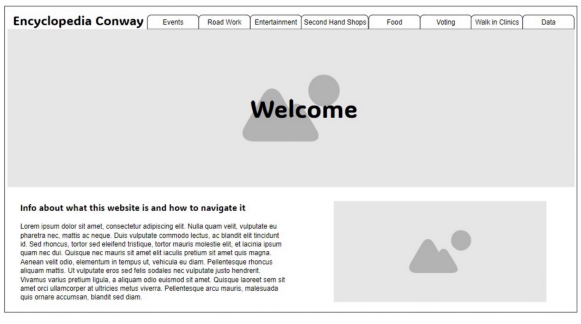
# Original Vision

## 3.1 The Original Vision

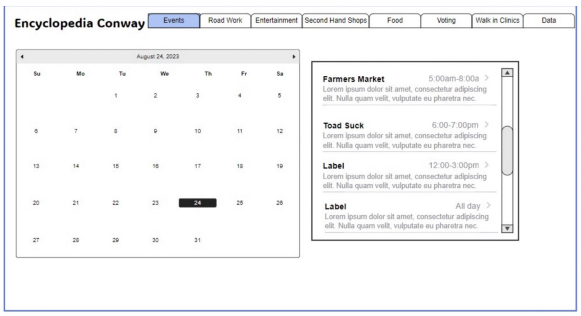
When planning out my project I dreamt big. I had grand ideas of using multiple APIs and open-source libraries to create my dream. My original goal was to create a website that was fully functional on its own and could update its own content without any help from me. In my original mockup of the website, I had 8 pages.

## 3.2 The Home Page

In my vision, the Home page would have been an opener for the website, describing what the project was and how to use the website. The Banner image would be rotating between different views of Conway. Some show off our more natural qualities and others show off the great businesses we have

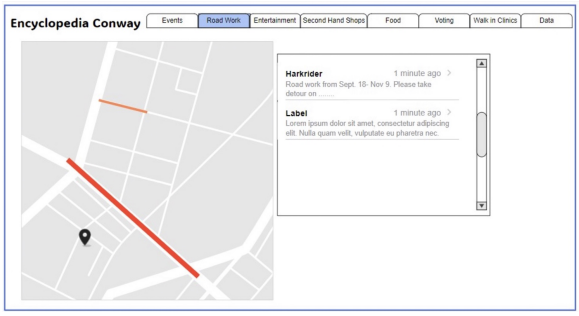


## 3.3 The Events Page




The Events page was the main inspiration for my website. I really wanted to know more about what was happening in Conway and miss fewer events. I was recently surprised to learn that the best place to hear about events was not the official City of Conway website but in fact social media like Facebook and Reddit. I have a plethora of information on events happening. This inspired me to consolidate this information onto a calendar instead of having to sort through all of the Facebook posts.

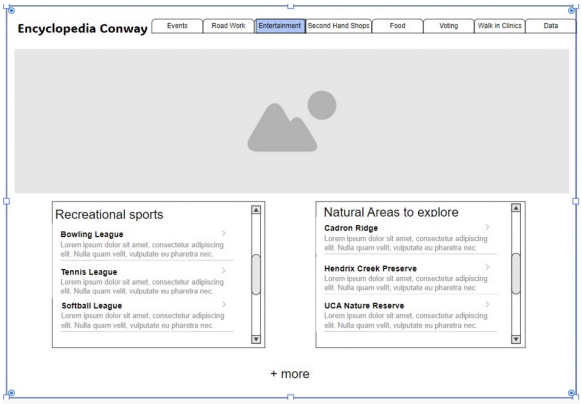
## 3.4 The Road Work Page




In the city of roundabouts, there's always new construction for new roundabouts. Many people like to know if their usual route will be impeded due to construction or are just curious about new road developments. This page consists of a map constructed from the Google Maps API. It includes a traffic layer that shows traffic and construction happening. This API allows the user to see up-to-date information about their city.

## 3.5 Entertainment Page

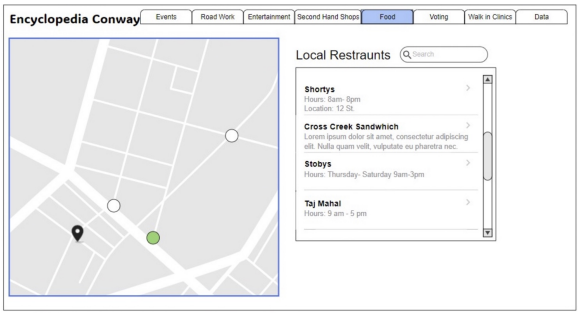
This page was inspired by "So, what do you guys do for fun around here?" I wanted to make a list of suggestions of activities or areas to visit that would hopefully make residents want to explore the area more. This page would hold information for natural areas near by, recreation leagues to join, and other miscellaneous ideas to do in Conway.




## 3.6 The Second Hand Shops Page

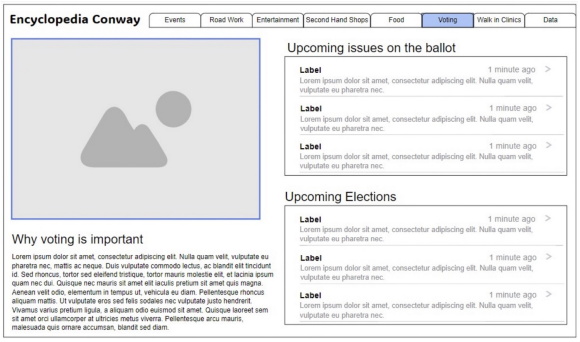
Second-hand shops are a crucial lifeline for many new residents. Most people cannot afford to buy brand new furniture or clothes whether they are college students or young adults with families. On this page, I included a map of all the Second-Hand shops in the Conway area along with the hours.

## 3.7 The Restaurants Page




It can be difficult to find small hole-in-the-wall restaurants when you first move to an area. This information is often reserved for longtime permanent residents. However, there is a true benefit to supporting local businesses. The tax revenue made through sale taxes from local businesses supports the local government which can then be used to reinvest in the community.[2]. This page consists of a map, made with Leaflet, with markers for all the local restaurants.

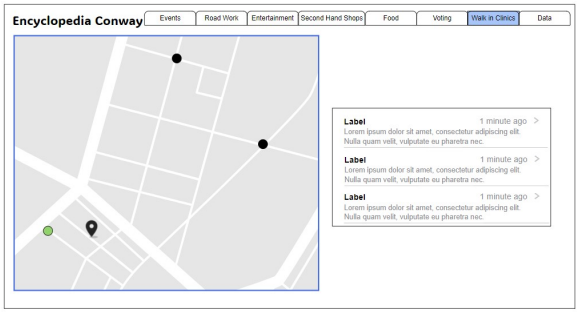
## 3.8 The Voting Page




Civic participation is paramount to being a resident in an area. Voting in local elections has large impacts on your community and can make important changes fast. This page consists of a list of upcoming voting opportunities, as well as resources that describe why voting is important and more information about voting in Conway.

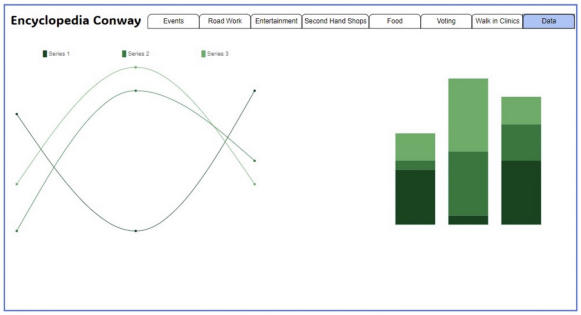
## 3.9 The Walk-In Clinics Page

It can be difficult to maintain a primary care physician. So, if you need to go to the doctor for cold or flu, you aren't going to the ER but a walk-in clinic. It's reassuring to know that Conway has many walk-in clinics available. This page consists of a map, made with Leaflet, with markers of all the Walk-in Clinics in Conway.




## 3.10 The Data Page

This is a page where I wanted to display some data from Conway. Originally, I was thinking it could display data from police reports, recycling reports, and just overall data about how the city was doing as a whole. I think it's important to know everything you can about your city and this can include the good, the bad, and the ugly.



## 3.11 Tools to make the website

I went through three different iterations of the Encyclopedia Conway website before settling on a framework. Each has its own positive and negative attributes. In this section, I will discuss why I considered each framework.

### 3.11.1 HTML

I started this project using HTML (Hypertext markup language). I had experience making web pages in HTML from my Databases and Web Systems class and it's pretty simple to code in. However, it's a static language which means the web page will stay fixed and not change unless the developer makes changes. For my project, I wanted a dynamic website that, through the use of APIs could update on its own.

### 3.11.2 Jekyll Blog

Next, I want to try using the Jekyll Blog framework. I also have some experience using this from my Databases and Web Systems class as well as my weekly Disco Tray Studios reflections. What's great about this framework is that it can easily be deployed on GitHub Pages. This means that GitHub hosts your blog online. Jekyll allows you to write in markdown, which is very simple and then generates HTML files from it using its selected templates. There are also a lot of templates, also known as themes, to choose from. However, I ended up not choosing Jekyll either for the same reason I chose not to use HTML, its static framework.

### 3.11.3 React

React is a JavaScript library that was developed by Facebook. I did not have any prior experience with it; however, I ended up choosing it because it can generate dynamic web applications. There are many aspects of React that make it a great framework including components. Components allow for code to be broken up and they are reusable. This means you can create a navbar once and then it can be used on each web page without having to recode it on each page. React also has a very active community which creates open-source libraries to use.

## 3.12 Open Source Libraries and APIs

There are a number of open-source libraries that were made to work with React web applications. An open-source library is any library with an open-source license, which denotes software that is free to reuse, modify, and/or publish without permission.[10] For my project, I was able to utilize 2 open-source libraries that proved to be very useful.

#### 3.12.0.1 Tailwind

Tailwind is a CSS (Cascading Style Sheets) framework that allows the user to write less custom CSS. Tailwind offers many free templates for React components. On my website, I used a template for my navbar, footer, and clickable objects on the Entertainment page. Additionally, Tailwind CSS offers utility classes instead of fully stylized components like Bootstrap (another CSS framework). This means that the user has more freedom to customize their components.

#### 3.12.0.2 Leaflet

Leaflet is an open-source JavaScript library for interactive maps that works both on mobile and desktop platforms.[5] It's supported on many different browsers such as Chrome, Firefox, Safari 5+, Opera 12+, IE 9–11, and Edge. It also has many interactive features such as Drag panning with inertia, Scroll wheel zoom, Pinch-zoom on mobile, Double click zoom, Zoom to area (shift-drag), Keyboard navigation, Events: click, mouseover, etc., and Marker dragging. Additionally, it allows map layers and location markers. This was incredibly useful for my Second-Hand Shops, Walk-in clinics, and Restaurant pages.

### 3.12.1 Other Open Source Libraries

There are many other open-source libraries that are used for small chunks in this project. You will learn about them in the tutorial section of this paper.

### 3.12.2 APIs

The Encyclopedia Conway website also uses APIs (application programming interfaces). "An API is a set of protocols and instructions written in programming languages such as C++ or JavaScript that determine how two software components will communicate with each other. APIs work behind the scenes to allow users to locate and retrieve the requested information. Think of APIs like contracts that determine how two software systems will interact."[7]

## 3.13 Dependency issues and their headaches

Here's where my project started to turn into a headache. Every open-source library and API requires its own dependencies and sometimes those dependencies conflict. In this case, many of the libraries depended on different versions of React. This was not something I could change or work with, so I had to drop many of the complexities from my orignial project idea and focus on making a simplier, yet still very functional website. I found many work arounds that still display the information I had originally intended on displaying.

# Chapter 4

# Tutorial

## 4.1 A Tutorial

Want to create a React website for your city? Here's a tutorial of all the things that I learned while working on my project.

### 4.1.1 Installing React

For this project, I used Visual Studio Code (VSC) as my source code editor. You can use an equivalent editor. To find where to download this software, use this website: **https://code.visualstudio.com** If you decide to use VSC, install the extension **ES7+ React/Redux/React-Native snippets**. This will allow you to code faster by allowing the user to use shortcuts to produce React code snippets

#### 4.1.1.1 Installing Node.js

To install React you must have **Node.js** installed. You can check if you already have this installed by going into your terminal and type **node -v**. If you do not have Node.js installed, then go to `https://nodejs.org/en/` to download. For more documentation on how to install Node.js go to `https://nodejs.org/en/docs`.

#### 4.1.1.2 Installing React

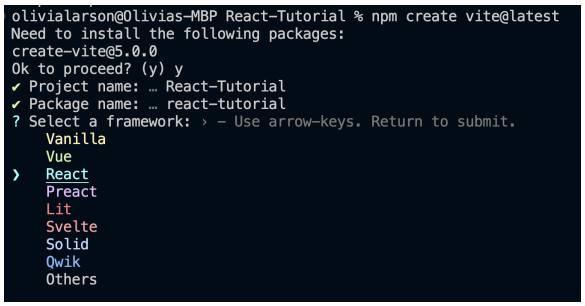
Now you can create your React project. Start by creating an empty folder for your project. For this Tutorial, I named my folder React-Tutorial. In your terminal

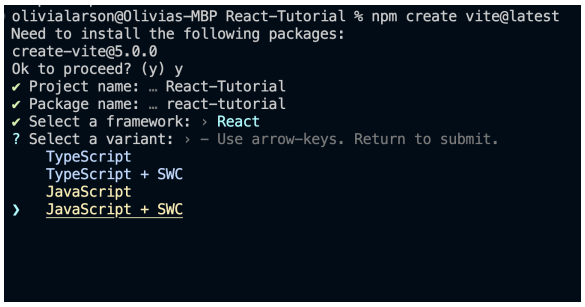
navigate to the directory of your folder. If you want to save your project on Github, log in to Github, make a new repository, clone it to your computer, and then open that folder in your terminal. Now you can run this command:

**npm create vite@latest vite-tw**.

Although you can create a React project using the command create-react-app (CRA), using Vite is better. Vite is a front-end tool that specializes in speed and performance. Unlike CRA it doesn't need to build the entire application before launching the website on the local server. This is because Vite divides the application modules into two categories: dependencies and source code. Dependencies are the plain Javascript libraries that do not change often. While the source code is the JSX and CSS files that are edited more.[1]

Next on the terminal screen, it will ask you to install the following packages. Type 'Y' and hit enter. Next, it will ask for a project name. My project name is React-Tutorial; you can name the project whatever you like. Next, the package name will appear and it should be the lowercase version of your project name. Hit enter. Next on the screen, it will ask you to select a framework. Hit your downward arrow button until React is underlined. When React is underlined then hit enter.

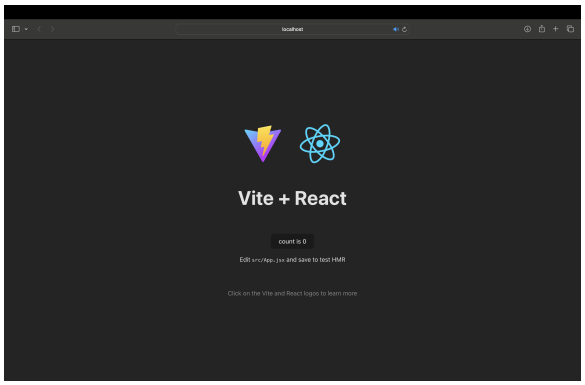



Next, it will ask you which variant. This is what file type your project will be in. For my project, I chose **Javascript + SWC**. This will create your project with JSX files rather than JS files. JSX is essentially javascript + HTML code which is easier for the user to understand. Use your downward arrow, until Javascript + SWC is underlined and then hit enter.

You will now see the project building in the original folder that you made.

Next open up the folder in VSC. You should now see your folder populated with folders and files. To run the website locally on your computer, open a terminal in VSC and run: **npm run dev**




You should now see the default website. If you get an error double-check that you are in the **vite-tw** folder. This will be important as we continue to install more libraries.

Congratulations! You now have a default React web page! Now let's look at making it functional.

### 4.1.2 Installing TailWinds

Before we start coding we are going to download a library to help make React functional React components. There are many options out there but for this project I used Tailwinds. Tailwinds is a super helpful open-source library with a lot of nice components. To install TailWinds: Go to your terminal in VSC and run these two commands:

**npm install -D tailwindcss postcss autoprefixer**

**npx tailwindcss init -p**

New folders and files will be imported into your project. Find the **tailwind.config.js**. Replace the code with

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Next, go to your index.css page and add this code at the top.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

You also need to install the correct dependency. In your terminal, type **npm install @headlessui/react @heroicons/react** For further documentation go to the Tailwinds website as listed here: `https://tailwindcss.com/docs/guides/vite`

### 4.1.3 Creating a simple website for your city

Continue to follow along with my tutorial, but feel free to add information that is more relevant to your city.

### 4.1.4 Beginning with a simple Home Page

Open up your project in a software like VSC. Notice all the folders that have been added. We are going to remove some of the default code so we can specialize it more to our liking. **First remove all of the code from the App.css, App.jsx, and the index.css** (except for the code we added for the tailwinds library). Your website should look completely blank. Next, we are going to replace some code in the App.jsx file. Add the code below and save it, it should reload on your local server.

```
import './App.css'

function App() {
    return(
    <h1>My City Encyclopedia</h1>
    )
}
export default App
```

Save your project. You can use the Command+S for Mac or Control+S for Windows. Use this each time you want to update the local version of your website.

Now you should have a blank website with the plain text that you typed between the `<h1></h1>`. This is where you can see the HTML in the jsx file. Also, this is

important to note that this is the layout for functions in React. So any time that you are typing in the jsx file you will be recreating this layout.

Let's look at the style sheet. Go to the index.css file in your project. This is basic css and can be edited as such.

### 4.1.5 Adding some simple components

Components are the driving force of React. Let's make a functional navbar and start adding multiple pages to our website using the Tailwinds library.

#### 4.1.5.1 Creating functional navbar and pages

First, in your **src folder**, you want to add 2 folders. One should be called "pages" and the other "components". In the pages folder, you want to add a new folder for every page you want to include on your website. For me, I added a folder for the 1.home, 2.events, 3.roadWork, 4.entertainment, 5.secondHandShops, 6.restaurants, 7.walkInClinics, and 8.voting. Adding the numbers in front keeps the folders in the correct order. Next, in the component folder you want to add 2 new folders. One is called "header" and the other is called "footer." In the header folder, we want to add a file called Navbar.jsx. In this file type add a new function following the code layout above. Or if you installed the React code snippets extension, you can type rafce. This will also create the layout for the code. Your file should look like this:

```
import React from 'react'

const Navbar = () => {
    return (
        <div>Navbar</div>
    )
}

export default Navbar
```

Now go through each of the pages folders and add a jsx file for the corresponding folder. For example, in the 1.home folder add a Home.jsx file. It's important that you capitalize the file name for React. In each of these new files either use **rafce**

command or type out the basic function layout, making sure to name the function with the same name as the file.

Now that we have all the necessary folders and files, we can grab a premade navbar component from Tailwinds. Go to the website: `https://tailwindui.com/components` and find the Navbar. Once you click on it, it will show you many different kinds of navbars. I used the first option. Click on the `<Code>` button and make sure you have React selected instead of HTML. You can now copy this code into your Navbar.jsx file. Make sure you replace the name of the function with Navbar.


Now go back to your App.jsx file and add `<Navbar/>` inside of the function. This is how you call and utilize components. You will also need to import the component into your App.jsx file. You can do this by **import Navbar from './components/Header/Navbar'** at the top of the file. Your App.jsx file should look like this. If done successfully you should have a blank page with a Navbar at the top.

Now let's update the Navbar.jsx file to have the correct pages. At the top of the file find const navigation. Change the names to be the names of the pages. My const navigation looks like this:


Now your website should have updated with the names of your pages in the Navbar. However, there are a couple more steps before the navbar is functional. Go back to your App. jsx file. Type in the import file name for each of the pages at the top of the App.jsx file. For example, importing your Home page should look like this:

**import Home from './pages/1.home/Home';**

Remember to do this for each of your pages. Next, we will install a Routing library that will allow us to navigate between the pages on the navbar. Use this command in

your terminal to do so. Make sure you are in the correct folder vite-tw. If you need to navigate into that folder simply type in cd vite-tw before typing in the following command:

**npm i react-router-dom@latest**

Next, we will be adding in the import for the Router Library. In your App.jsx file add this import at the top of the file next to the others we just put in there for the pages.

```
import {BrowserRouter as Router, Routes, Route} from
'react-router-dom';
```

In the App function type `<Routes> </Routes>`. This is a common syntax in React files. I will be referring to it as wrapping because it goes at the beginning and the end with other bits of code in the middle. The Routes wrapper allows us to define possible page routes for the navbar and it then chooses the correct one to send the user to. Then within the Routes wrapper, we are going to define the routes of the pages. To make these routes we are going to type in the path to the page like this: `<Route path="/" element ={} />` The path is the path to the file for the page and the element is the name of the component (which in this case is the page). So for one route we will type: `<Route path "/home element={<home/>} />` Do this for each of the pages. Once you're done, we are going to change the Route for the home page to `<Routes path="\" element ={\Home} />` This sets your home page as the home page of your website.

Your routes should look like this:

```
function App() {
  return (
    <div>
      <Router>
      <NavBar />
      <Routes>
        <Route exact path='/' element={<Home />} />
        <Route  path='/events' element={<Events />} />
        <Route  path='/roadwork' element={<RoadWork />} />
        <Route  path='/entertainment' element={<Entertainment />} />
        <Route  path='/secondhandshops' element={<SecondHandShops />} />
        <Route  path='/restaurants' element={<Restaurants />} />
        <Route  path='/voting' element={<Voting />} />
        <Route  path='/walkinclinics' element={<WalkInClinics />} />
      </Routes>
      </Router>
    </div>
  )
}
export default App
```

You should now have a functional navigation bar for your website! Now let's go in and start making the pages!

To remove the extra profile picture of the man on the right side of the navbar, remove all the code inside the `<Menu> </Menu>` wrapper including the wrapper itself. To remove the alarm bell icon do the same thing except this time we are looking for the code in between

the `<button> </button>` wrapper.

**OPTIONAL STYLE EDITS:**

Here are some quick style edits you can make for your navbar. In the Navbar.jsx file search for the ¡img/¿ wrapper. In this wrapper, you will see "src." This stands for the image source. You can change the image source to whatever you like. For this, I used Canva to create my EC logo and then removed the background before I added it to my project. To add images to your project, save them to the public/assets folder. This means find the public folder, click on it, find the assets folder and then save the image in there. You can then set images by declaring them in the source as "/assets/whatEverYouNamedYourImage.jpg"

To change the color of your navbar go to the index.css file. Create a new class. It should look like this:

```
.navbar_color{

}
```

This is common css syntax. If you would like to learn more about css here is documentation that you can refer to: `https://devdocs.io/css/`

In this class type background: green. It should now look like this:

```
.navbar_color{
    background: green;
}
```

Now go back to the Navbar.jsx file and change

`<Disclosure as="nav" className="bg-gray-800">`

to

`<Disclosure as="nav" className="navbar_color">`

Save the project and you should see that the navbar has changed its color to green.

Make adjustments to the logo in the navbar and the color as needed! A great resource I used to find colors and color pallets is Coolers.co it is a color pallet generator and super useful.

#### 4.1.5.2 Creating a footer

Go back to the components folder and find the folder called footer. In that folder add a file called "Footer.jsx". Either use the rafce shortcut or make a standard function.

In the `<div> </div>` wrapper add a class name to the opening (the first div). Make the class name `"fixed bottom-0 w-full bg-black text-white px-4 p-4"`. This is a built in css style.

Make sure to go back to the App.jsx file and add the footer element beneath the `<Routes> <\Routes>` wrapper. You should now have a footer on your website.

Go back to the Footer.jsx file. You can put any information you would like between the `<div> </div>` wrapper. For my project, I put contact information.

## 4.2 Creating the real Home page

Go to Home.jsx. Let's add some information about our website and what we want the user to expect. Between the `<div> <\div>` wrapper add `<h1> </h1>` wrapper (this is also called a tag in HTML). This will create the correct size font for a title. This is a common html language. If you would like to learn more, visit this documentation: URL: `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements`

Inside the `<h1> </h1>` wrapper type Welcome. You should now see Welcome on your home page. If 'Welcome' doesn't look like a title, that's fine we can adjust it in the index.css file. Go to the index.css file and create a h1 class. Within the class type **font-size:20px;** . This will make the font size bigger. To bold the text, add **font-weight: bold;** to the class. If you want to change the font type, add **font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif;**. There are other built-in font options available. Just type font-family: and a list should appear that you can select from. Your h1 class should now look like this:

```
h1{
    font-size:30px;
    font-weight: bold;
    font-family: 'Gill Sans', 'Gill Sans MT', Calibri,
    'Trebuchet MS', sans-serif;
}
```

Now go back to the Home.jsx file. Underneath the `<h1>` tag add in `<p> </p>`. Similar to the h1 tag p is a common tag. In this wrapper, you can add a welcome introduction to your website. If you want to change the size or font for the words in the p tag, you can go to the index.css file and do exactly what we did for the h1 tag and make adjustments.

For my website, I added another section for how to navigate the website. You can copy the two steps above by making a header and then adding information.

If you want to add some padding around your text, go to your index.css file and make a new class called .container. In this class add padding:10px. Then back in your Home.jsx file in the outermost `<div>` wrapper give it the class name container. You can mess around with the padding. You can even only have padding on certain sides by using padding-left or padding-top.

### 4.2.1 Creating the photo carousel

On my website, I have an autoplay photo carousel displaying pictures of Conway. If you would like to add that to your website you should follow this tutorial on Youtube: `https://youtu.be/SAWQ_LmyY2Q?si=RS46ECGB9-Hbh4bw`. First, create a Carousel.jsx file. I put my Carousel.jsx file in the components folder.

For your images, make a new folder in your assets folder (which is in your public folder) called Carousel. Save all the images you want in your Carousel in there.

The in your 1.home folder (the folder for your home page), add a new file called Photos.jsx. This is where we will keep the path to the photos. We are going to create an array of image paths. It will look something like this. Change the image path as needed.

```
export const photos = [
    {
        image: "./assets/carousel/bridge.png",

    },
    {
        image: "./assets/carousel/downtown1.jpeg",

    },
```

```
    {
        image: "./assets/carousel/downtown2.jpeg",

    },
];
```

I took out the titles for the photos as it was originally done in the tutorial but you can keep them in. All you have to do is add the title: "Title of photo" underneath the image: "imagepath".

Now create another file called CarouselOnHome.jsx. Use the rafce shortcut or create the standard function. Import the photos.jsx file and the Carousel component file. It should look like this:

```
import Carousel from '../../components/carousel/Carousel'
import {photos} from './Photos'
```

The reason photos are in brackets is that it is a list that we are importing and not a function. Now in the CarouselOnHome function, type `<Carousel images={photos}/>` This will give the list of image paths to the Carousel component.

On the Home.jsx file, make sure to import the CarouselOnHome file and component, similar to how we imported the Navbar. Then above the Welcome header, insert the Carousel element `<CarouselonHome/>`. You should now have an autoplay rotating carousel.

***** Warning: I will now be showing you how to make these pages in order. Some pages are simpler to make than others, so, we are starting with the easy ones and then going to the more difficult ones *****

## 4.3 Creating Second Hand Shops Page

This page will be pretty simple. All I wanted to do was to have an interactive map with the locations and names of second-hand shops in the area. To do this we are going to import a library called Leaflet.

#### 4.3.0.1 Leaflet

Type in the **npm install leaflet react-leaflet** into your terminal. Make sure you are in the correct folder (vite-tw). For more installation help go to `https:`

//react-leaflet.js.org/docs/start-installation/

In your SecondHandShops folder add a new file called Map.jsx. In this file add this code to create your map

```
import React from 'react';
import {MapContainer, TileLayer, Marker, Popup} from 'react-leaflet';

function Map() {
  const cityPosition = [LATITUDE, LONGITUDE];

  return (
    <MapContainer center={cityPosition} zoom={13} style=
    {{ height: '500px', width: '100%' }}>
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution='&copy; <a href="https://www.openstreetmap.org
        /copyright">
        OpenStreetMap</a> contributors'
      />

    </MapContainer>
  );
}

export default Map;
```

Add the coordinates for your city in the cityPosition and then on the SecondHandShops.jsx import the Map file and type in the element in the return. It may look a little wonky so we need to add some css. Instead of adding it to our regular index.css, we will add a Map.css file to our secondHandShops folder. In the Map.css file copy and paste the css from this website `https://unpkg.com/leaflet@1.9.4/dist/leaflet.css`. This is the official leaflet style sheet. You should now have a map of your city! Now let's add the important information to it.

In your SecondHandShops folder make a new file called ShopsList.jsx. This file is going to hold all the information for the name and location of the second-hand shops. In this file type `export const secondHandShops = [];`. In the [] brackets we will

need to have a name, coordinate, address, and locationLink for each shop. It will look something like this:

```
export const secondHandShops = [
    { name: 'Goodwill- Oak Street',
    coordinates: [35.091186798743955, -92.42957560152153],
    address:'245 Oak St, Conway, AR 72032',
    locLink:'https://maps.google.com/?
    ll=35.091186798743955,-92.42957560152153' },

    { name: 'Dads and Lads Consignment',
    coordinates:[35.11062146097442, -92.44557790039295],
    address:'2326 Washington Ave, Conway, AR 72032',
    locLink:'https://maps.google.com/?
    ll=35.11062146097442,-92.44557790039295'},

    { name: 'Goodwill- Sanders Street',
    coordinates:[35.1106380472001, -92.43662483141217],
    address:'2425 Sanders St, Conway, AR 72032',
    locLink:'https://maps.google.com/?
    ll=35.1106380472001,-92.43662483141217'},
];
```

The location link is a link to the Google Maps page for the second-hand shop. You might notice that the coordinates of the shop are in the link. You could also go to the Google Maps page for the shop and steal the link from there. Also, it's important to note that you can have as many shops on this list as you would like!

Now go back to the Map.jsx file we are going to create markers on the map for all the second-hand shops. To do this we are going to use the react function called .map(). This takes all of the information stored in our array and will make an individual marker for it. It can also do a lot more. To read more about the .map() go to this documentation: `https://legacy.reactjs.org/docs/lists-and-keys.html`. In the Map.jsx file add this import **import Link from 'react-router-dom'** and code below. It creates a marker for each of the shops we listed in the list.

```
{secondHandShops.map((shop, index) => (
  <Marker key={index} position={shop.coordinates}>
    <Popup>
      {shop.name}
```

```
        <div className='small-empty-space' />
        <Link to={shop.locLink}> {shop.address} </Link>
        </Popup>
    </Marker>
  ))}
```

This code should go underneath the `<TileLayer/>` wrapper.

You may notice my `<div classname= "small-empty-space">` I made this in the index.css. It essentially gives me a line of space between elements. Now we must go to the SecondHandShops.jsx file, and import the list we made for shops at the top of the file. Find the `<Map/>` element and add `<Map secondHandShops= {secondHandShops}/>`. This is going to give the map element our list of shops for it to render with markers. Before saving the project, go back to the Map.jsx file and in the function add `{secondHandShops}` it should look like this: `function Map({secondHandShops}) {`. Now save your project. You should see a marker for every shop you listed! And when you click on it you should see a popup with the name and address link.

The setup for the Second Hand Shops page is very similar to to Restuarants and Walk-in Clinics page. If you wish you can skip ahead to those pages in this tutorial.

## 4.4 Creating a Restaurants Page

This page has a similar setup to the Second Hand Shops page. In fact, you can copy the Map. jsx file and the Map.css file into this folder. Create a RestaurantsList.jsx file too. We will create a similar list of restaurant names, coordinates, addresses, and locationLinks, but this time we are adding URLs for the restaurant website page. Your list will look something like this:

```
export const restaurantsList= [
    { name: 'Shortys Bar-B-Q ',
    coordinates:[35.09376770122701, -92.43720266788422],
    address:'1101 Harkrider St, Conway, AR 72032',
    url:'https://www.facebook.com/ShortysinConway/',
    locLink:'https://maps.google.com/?
    ll=35.09376770122701,-92.43720266788422'},
    { name: 'Verona Italian Restaurant ',
    coordinates:[35.113072847886656, -92.42945097456317],
```

```
    address:'190 Skyline Dr, Conway, AR 72032',
    url:'https://www.facebook.com/VeronaConwayAr/',
    locLink:'https://maps.google.com/?
    ll=35.113072847886656,-92.42945097456317'},
    { name: 'The Rogue Roundabout ',
    coordinates:[35.09021098245866, -92.43927327641352],
    address: '804 Chestnut St, Conway, AR 72032',
    url:'https://www.theroguеroundabout.com',
    locLink: 'https://maps.google.com/?
    ll=35.09021098245866,-92.43927327641352'}
  ];
```

You can have as many Restaurants as you would like. For my website, I focused on local restaurants, but you can put any restaurant in that you'd like. Now go back to your Map.jsx page and change every secondhandshops to restaurantsList. The `<Marker>` will look like this:

```
{restaurantsList.map((shop, index) => (
    <Marker key={index} position={shop.coordinates}>
      <Popup>
        {shop.name}
        <div className='small-empty-space' />
        <Link to= {shop.locLink} > {shop.address}</Link>
        <div className='small-empty-space' />
        <Link to={shop.url}>{shop.name} Restaurant link</Link>
      </Popup>
    </Marker>
))}
```

Go to your Restaurants.jsx file and add in the import for the restuarantsList at the top of the file. Next, add the `<Map/>` element in the function. Similar to the Second Hand Shops page we are going to pass in the list of restaurant information to the map. It will look like this `<Map restaurantsList = {restuarantsList/>}`. Now you should have a fully functioning map of all the local restaurants in your area!

## 4.5 Creating Walk-in Clinics Page

This page is also very similar to the Second Hand Shops page and the Restaurants page. Copy the Map.jsx and the Map.css files from either the Secondhandshops folder

or the Restaurant folder. In the Map.jsx file replace any mention of secondHandShops or restaurantsList with clinicsList. Similar to what we did before, make a new file called ClinicsList.jsx. This is going to be the list of names of clinics, coordinates, addresses, and locationlinks. You know the drill! It should look something like this:

```
export const clinicsList = [
    { name: 'Medcare Urgent Express',
    coordinates: [35.09176934180663, -92.43467766052645],
    locLink: 'https://maps.google.com/?
    ll=35.09176934180663,-92.43467766052645',
    address:'805 Oak St, Conway, AR 72032' },
    { name: 'Primecare Medical Clinic',
    coordinates: [35.09220827463228, -92.43614751101053],
    locLink: 'https://maps.google.com/?
    ll=35.09220827463228,-92.43614751101053',
    address:'812 Oak St, Conway, AR 72032' },
    { name: 'Conway Family Practice Clinic',
    coordinates: [35.09087158632688, -92.44303638483913],
    locLink: 'https://maps.google.com/?
    ll=35.09087158632688,-92.44303638483913',
    address:'919 Locust St, Conway, AR 72034' },
];
```

We next need to add the markers for the map in the Map.jsx file. Go to the Map.jsx file and add this code for the markers:

```
{clinicsList.map((shop, index) => (
    <Marker key={index} position={shop.coordinates}>
      <Popup>
        {shop.name}
        <div className='small-empty-space' />
        <Link to={shop.locLink}> {shop.address} </Link>
      </Popup>
    </Marker>
  ))}
```

Go to your WalkInClinics.jsx file and add the import for the clinicsList at the top of the file. Next, add the `<Map/>` element in the function. Similar to the Second Hand Shops page we are going to pass in the list of clinic information to the map. It will look like this `<Map clinicsList = {clinicsList/>}`. Now you should have a

map of walk-in clinics in your area!

## 4.6 Creating Entertainment Page

The Entertainment page is a little different from the other pages. For this, I used a tailwinds component. Go to the `TailwindUI.com` and brows the components. Look for the **Blog Sections** component. Click on it and make sure you are looking at the React code. Create a new file called **EntertainmentSection.jsx** in your Entertainment folder. Copy the code from Tailwinds and paste it into your **EntertainmentSection.jsx** page. Look for "From the Blog" and change it to **Things to do near you!** or whatever you want it to be. Look for "Learn how to grow your business with our expert advice." and change it to "Here are some ideas for things you can do in your area" or whatever works for you.

At the top of the file, you will see **const posts=[]**. This is what we are going to edit to be entertainment information. It's pretty straightforward. Here is how I edited mine to have the information I wanted.

```
{
id: 1,
title: 'Outdoor activities',
href: '#',
description:
  'This is a list that includes
  some nearby outdoor areas that
  you should explore!',
date: '',
datetime: '',
category: { title: 'Outdoor', href: '#' },
author: {},
image: './assets/Cadron.jpeg',
list: <ul>
  <ul>Cadron Settlement Park</ul>
  <ul>Beaver Fork Park</ul>
  <ul> Hendrix Creek Preserve </ul>
  <ul>Don Owen Sport Complex</ul>
  <ul>Laurel Park</ul>
  <ul>Tucker Creek Walking Trail</ul>
```

```
      <ul>Jewel Moore Nature Reserve</ul>
      <ul>Martin Luther King Jr. Square</ul>
    </ul>
  },
```

To make another "post" copy the format above and change **id:1 to id:2** and so on and so forth. If you want an image associated with the post, add them to the public/assets folder and then reference them in the post.

Now we need to make this post clickable. We will need to install the Reactjs-Popup library. To do this type this command in the terminal (make sure you are in the vite-tw folder) **npm install reactjs-popup**. Now in your EntertainmentSection.jsx file add these imports:

```
import React from 'react';
import Popup from 'reactjs-popup';
```

If you want to learn more about this library here is the url: `https://react-popup.elazizi.com/getting-started`

Then towards the bottom of the code, find the closing `<article/>` wrapper and above it put this code below.

```
<Popup trigger=
  {<div className='button-border'>
    <div className='h2'>
      <button> Click to see full list </button>
    </div>
  </div>}
  modal nested>
  {
    close => (
      <div className='modal'>
        <div className='content'>
          <h2>{post.title}</h2>
          <div className='small-empty-space' />
          <div className"container">
            <p>{post.list}</p>
          </div>
        </div>
        <div>
```

```
            <button onClick=
              {() => close()}>
            </button>
          </div>
        </div>
      )
    }
</Popup>
```

This will make a button that when clicked makes a popup for your list of activities. This is a great opportunity to mess around with the index.css and add some style to this page. I put a border around my posts and changed the background color to make it pop.

In one of my posts, I included links for registration sign-up. To do this add this import at the top of the file: **import Link from 'react-router-dom'** and then you can add a link by using this format:

```
<Link to='https://conwayarkansas.gov/
parks/programs/adult-softball/' >
Adult Softball league </Link>
```

Include this in the list of the post. Now you have an Entertainment page!

## 4.7 Creating the Road Work Page

This page is going to use the Google Maps API. Theoretically, you should be able to add a traffic layer on top of the map. This layer would include construction going on in the area. This was one aspect of the project that I wasn't able to figure out. Try your hand at it if you want! Or you can follow along with my workaround!

### 4.7.1 React Google Maps API

Insert this command in your terminal: **npm i -S @react-google-maps/api**. Make sure you are in the correct folder vite-tw. Now go to your Roadwork folder. Make a new file called GoogleMaps.jsx. In this file insert the code from the library found here: `https://www.npmjs.com/package/@react-google-maps/api`. This code will make a map for you. Go to your Roadwork.jsx file, type **import MyComponent**

**from "./GoogleMaps";** at the top of the file. Then add `<MyComponent/>` in the function and save the page. You might notice that this doesn't work yet. That is because we have to get an API key from Google Maps. An API key is used for project identification and authorization [8].

Go to a different tab and search Google Maps API it should be the first result: Google Maps Platform. If you don't have a Google account you'll need to make one. Once you are logged in click on Get Started. Find the New Project button and click on it. Name the project and click Create. On the side menu find Keys and Credentials and click on it. Once you do, it'll give you your API key. Copy the key and go back to the GoogleMap.jsx file in your Roadwork file. Find `googleMapsApiKey: "YOUR_API_KEY"` and paste your API key there. Now save your project. Your Map should now work! Now we just need to make some adjustments to the size of the map and center it on the correct city.

You can change where the map is centred by changing the latitude and longitude here:

```
const center = {
  lat: -3.745,
  lng: -38.523
};
```

You can change the size of the map by adjusting the **const containerstyle**. I changed mine to:

```
const containerStyle = {
  width: '700px',
  height: '700px'
};
```

Both of these are in the GoogleMaps.jsx file.

### 4.7.2 Creating a list for Construction

Now add in a list of construction events happening. Create a new file called ConstructionList.jsx in the Roadwork folder. Here we are going to make a list similar to what we did with the SecondHandshops pages. In our **export Constructionlist =[]** we want a **where:** and **description:** . This is what my list looks like:

```
export const ConstructionList=[
    { where: "Corner of Front Street and Mill Street",
    description: "Filling in a pothole"},
    { where: "Tyler/Salem roundabout",
    description:"Routine Maintenance"},
    { where: "Front Street at the intersection with Washington Ave.",
    description: "Paving roadway"},
    { where: "Mill Street at the intersection with Front Street",
    description: "Cleaning up debris from the construction site"},
    ];
```

You can put as many construction incidents as you would like!

Now Create a Construction.jsx file in your Roadwork folder. In this file use the rafce command or make a simple function. We want the

`function Construction({ConstructionList}`. Now within this function, we are going to use the .map() that we used in the previous pages. In the return, type:

```
{ConstructionList.map((incident, index) => (
    <div className='container'>
        <div className='title'>{incident.where} </div>
        <p> {incident.description}</p>
    </div>
))
}
```

Now go back to the Roadwork.jsx file and import Construction from './Construction' and `import{ConstructionList} from './ConstructionList'`. Similar to what we did before, we are going to put:
`<Construction ConstructionList ={ConstructionList}/>` underneath `<MyComponenet/>`.

This is a great opportunity to go into the index.css file and make a new class called `.row{}`. In this class put:

```
display:flex;
flex-direction:row;
```

Now go back to your Roadwork.jsx file and wrap `<MyComponent/>` and
`<Construction ConstructionList ={ConstructionList}/>` in
`<div className='row'> </div>`. This will put the map side by side with the construction list.

## 4.8 Creating Events Page

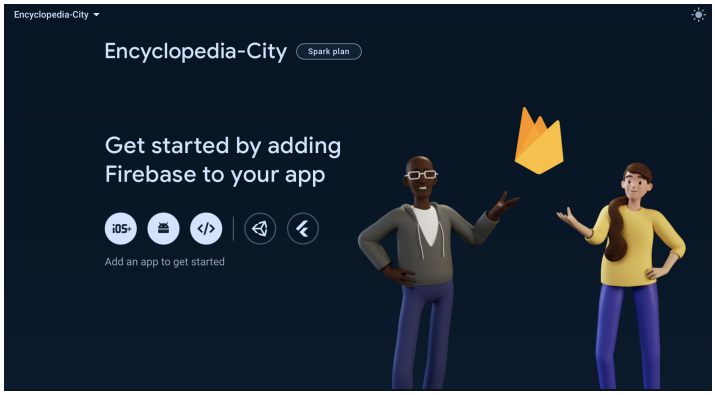
This page will display events happening in your city! Unfortunately, I was not able to get the functional calendar that I wanted due to the library dependency problems. So the workaround I used was making a list of events that could be updated through Firebase. Feel free to make any style selections you want for your website. I kept this page very simple. Now let's connect Firebase to our project. We will first deploy our website onto Firebase and then connect the database.

### 4.8.1 Hosting Website on Firebase

Let's deploy our website to a server so anyone can see it. While there are many options on how to do this, I decided to host my website on Firebase. My reasoning for this? It's a very simple setup and has an easy command that you can put in the terminal to deploy your website.

Go to `firebase.google.com` and log into your Google account and then go to the console. Click Create a project and give your project a name. Click continue. You can decide if you want analytics or not, but I turned off this feature for my website.

Once the project is created click on the web icon. It looks like this:




On the next screen, it will ask you to give your webapp a name. Most importantly you must click the check box for Firebase hosting. Then click Register App.

Next, go back to your project's terminal and in the vite-tw folder run this command:

**npm install firebase**

Then copy the code it gives you and paste it beneath the imports in your main.jsx file. Go back to Firebase and click next. It will then tell you to run a command to install Firebase CLI. Go to your project terminal and run:

**npm install -g firebase-tools**

If you get an error try:

**npm install firebase-tools**

This just removes the global aspect. It's not necessary for just this project.

Then copy the commands from the Firebase website into your terminal.

**firebase login** login to your firebase account.

**firebase init** to initiate your project. It will ask you to select "Which Firebase features do you want to set up for this directory?". Use the arrow keys and click the spacebar when the **Hosting: Configure Files for Firebase Hosting and (optionally) Set Up Github Action Deploys** is highlighted. Then click enter. It will then ask you to set up a default project. Use your arrow keys to highlight **Use an existing project** and select your project name and click enter.

It will then ask what you want to use as your public directory. Type **dist** this is very important.[6] It will then ask you the following questions:

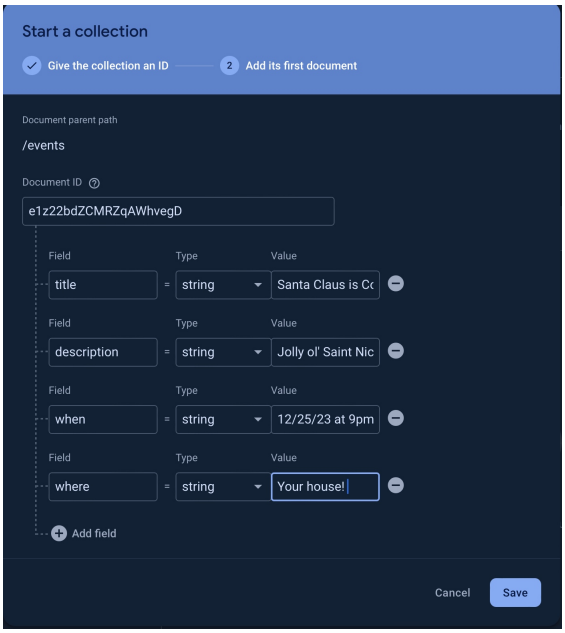
1. Configure as a single-page app (rewrite all urls to /index.html)? **Select Yes**
2. Set up automatic builds and deploys with GitHub? **Select No**

Then in your terminal run the command **npm run build**. This builds your project. You can then run the command **firebase deploy** and this will deploy your website on the firebase server. Copy the link into your internet browser and there it is!

### 4.8.2 Creating a Database

Go to the Firebase console and click on the menu. Click Build and then select **Firestore Database**. Click the **Create Database** button. Click next. Select **Start in test mode**. This is an option that can be changed later. It can be dangerous to keep your data while the database is in test mode but it will make it easier to set up the project like this for now. Click **Enable** and now we have our database!

### 4.8.3 Adding events to Firebase




Since we are starting with our events page, we must think about what type of data we want to store. For instance, for an event, you might want to know the name/title of the event, a description, what day/time it's at, and where it's at.

Click **Start Collection** and name the collection **events**. Click next. On the next page, you will be creating your first document. Documents are stored in collections. Think of collections as the category and the document as the things that belong in that category. On this page, select **Auto-ID** to give the document its unique ID. Then in the fields, we are going to create the things that need to go in the event. Here we are going to create a field for title, description, when, and where. For now, you can put some fake data in the value. We can change this later when we are ready. When you're ready click save.

### 4.8.4 Making events from Firebase display on website

Now let's make this event show up on our website. First, In your Events folder, create a new file called EventsList.jsx and either use the rafce command or make a basic function.

Then, create a new folder in your src folder called **services**. This folder is going to hold files that talk to Firebase. In this folder make a file called **firestoreEventService.jsx**. In that file copy this code:

```
import { getApp } from "firebase/app";
import { collection, getDocs, getFirestore } from "firebase/firestore"
export async function getEvents() {
    const db = getFirestore(getApp())
```

```
    const eventsCollection = collection(db, 'events')
    const events = await getDocs(eventsCollection);

    return events.docs.map(doc => ({
        id: doc.id,
        title: doc.data()["title"],
        description: doc.data()["description"],
        when: doc.data()["when"],
        where: doc.data()["where"]
    }));

}
```

Go back to your EventsList.jsx file and add:
**import getEvents from '../../services/firestoreEventService'**
at the top of the file. Now we have to make a function on how to render the events. Before function EventsList, make a new function called **renderEvent(event)**. We want this function to take the data from Firestore and make the event. In the return(), type `<div key={event.id}> </div>`. This will keep event information separate. Now, within the `<div key={event.id}></div>` wrapper you are going to type:

```
<div className='title'> {event.title} </div>
<p> {event.description}</p>
<p> when: {event.when} </p>
<p> where: {event.where}</p>
```

Because event.title is in curly braces the code knows that it's the event title and not actually `{event.title}` that we want to see rendered. This goes for the rest of the information.

Your entire function should look like this:

```
function renderEvent(event) {
    return (
        <div key={event.id}>
            <h1> {event.title} </h1>
            <p> {event.description}</p>
            <p>when: {event.when} </p>
            <p>where: {event.where}</p>
        </div>
    )
```

}

Now move down to the function EventsList. Before we return() we need to add a state. We need to know that we are waiting for the events to be sent from Firebase. Once it's been delivered by the API then we need to render it to our website. To do this we are going to use the React Hook SetState. To read more about React Hooks go this this documentation: `https://legacy.reactjs.org/docs/hooks-intro.html`. At the top of the file type: `import {useEffect, useState} from 'react';`

Then type **const [events, setEvents] = useState(null)** before the return(). This is very specific syntax and you can learn more about it here: `https://legacy.reactjs.org/docs/hooks-state.html`. What this says, is that you have a state of events that cannot be changed so the way you change it is through this function called setEvents.

Underneath **const [events, setEvents] = useState(null)** we are going to type:

```
useEffect(() => {
    async function loadEvents() {
        const firestoreEvents = await getEvents()
        setEvents(firestoreEvents)
    }
    loadEvents()
    }, [])

    if (!events) {
        return <div>Loading...</div>
    }
```

useEffect is another React Hook that has its own specific syntax. If you want to learn more about useEffect you can read more about it here: `https://legacy.reactjs.org/docs/hooks-effect.html`. This function is calling on the getEvents() function that we made in the firestoreEventService file that makes calls to our database. Once it gets the events it sets them to events. We will be able to render these events in the return now.

If there are no events (hence !events) it will return Loading... until there are events to load.

In return of the function EventsList, type: `{events.map(renderEvent)}`

Now go to your Events.jsx file and import the EventsList component. Make sure to return `<EventsList/>`. Save your project. You should now have a working connection between your website and Firebase! Now you can go into Firebase and add, change, or delete events.

This is also a great time to go in and add some style elements to your index.css file. I added an underline class that adds puts a line under each event to add more separation. Or you could even have a new font for titles of events. The world is your oyster.

#### 4.8.4.1 For more help with Firebase

Connecting Firebase to this project was very difficult for me, but I was able to find a tutorial that was very helpful. Here is the link to their video. I highly recommend watching their Coding Monday series. `https://youtu.be/u25_Fd5xNkc?si=Yq3HCPhpTUtyMrLF`

### 4.8.5 Photo carousel for Events Page

I added a photo carousel to my events page to make it more interesting. This will be the same process you did for the home page. Add a new folder in **public/assets called EventPhotos** and save the photos you want for this page there. Then go back to your Events folder, add a file called **EventPhotosList.jsx**, and create a list of paths to your **EventPhotos**.

Then create another file called **CarouselOnEvents.jsx**. Use the rafce shortcut or create the standard function. Import the photos.jsx file and the Carousel component file. It should look like this:

```
import Carousel from '../../components/carousel/Carousel'
import {EventPhotosList} from './EventPhotosList'
```

Now with in the CarouselOnEvents function, type `<Carousel images={photos}/>`

On the Events.jsx file, make sure to import the CarouselOnEvents file and component, similar to how we imported the CarouselOnHome. Then above the `<Events/>`, insert the Carousel element `<CarouselOnEvents/>`. You should now have an autoplay rotating carousel.

## 4.9 Creating the Voting Page

This page is going to be very similar to the events page. Start by going into Firebase and creating a new collection called **voting**. Think about what information should be relevant for voters. For me, I included a **title, description, date, and additionalInfo** (for links). For now, you can just load in fake information and then go back later to update it with real information.

Go back to your project, and create a new file in the **services** folder called **firestoreVotingServices**. Copy the code we put in the **fireStoreEventServices**.

```
import { getApp } from "firebase/app";
import { collection, getDocs, getFirestore } from "firebase/firestore"

export async function getVotingDoc() {
    const db = getFirestore(getApp())
    const votingCollection = collection(db, 'voting')
    const voting = await getDocs(votingCollection);

    return voting.docs.map(doc => ({
        id: doc.id,
        title: doc.data()["title"],
        description: doc.data()["description"],
        date: doc.data()["date"],
        addInfo: doc.data()["addInfo"]
    }));

}
```

Change the code so it resembles the code above. Now go back to your Voting folder and create a file called **VotingList.jsx**. Create a function or use the rafce command. Above the **function VotingList()** create a new function called **renderVoting(voting)**. This function is very similar to the one we made for the renderEvent().

```
function renderVoting(voting) {
    return (
        <div key={voting.id} className='votingPadding' >
            <div className='title'> {voting.title} </div>
```

```
            <p> {voting.description}</p>
            <p>{voting.date} </p>
            <p>{voting.addInfo}</p>
            <div className='underline' />
        </div>
    )
}
```

Your code will look something like this. Remember to import the **firestoreVotingServices** and **useEffect, useState**. Then, above the return for the VotingList function type:

```
const [voting, setVoting] = useState(null)
    useEffect(() => {
        async function loadVoting() {
            const firestoreVoting = await getVoting()
            setVoting(firestoreVoting)
        }
        loadVoting()
    }, [])

    if (!voting) {
        return <div>Loading...</div>
    }
```

This is very similar to what we did before with our events page. Then in the return, type `{voting.map(renderVoting)}`. Go back to your Voting.jsx file and type **import VotingList from './VotingList'** and in the return for the Voting function type **¡VotingList/¿**. You have now connected Firebase to your Voting page!

### 4.9.1 Style

To make my voting page more engaging I added an image, a description of why voting is important, and links to resources about why you should vote and voting information for my county. You can do something similar if you like.

## 4.10 Completed website

You have now finished your website! Run **npm run build** and then **firebase deploy** and now your website will be out on the internet for anyone to see.

### 4.10.1 Firebase deploy not building entire website

If you experience this error here is a quick fix that I found. Go to your **vite.config.js** file and replace this **defineConfig** function with this code:

```
export default defineConfig({
  plugins: [react()],
  build: {
    chunkSizeWarningLimit: 1600,
  },
})
```

You may also have to run the **npm run build** command a couple of times before running the **firebase deploy** command. If you are still finding problems you can find more information by searching **chunkSizeWarningLimit: 1600,**

# Chapter 5

# Conclusions

This project came with its learning curve and at times it was more overwhelming than I anticipated. Learning a new framework in a semester and completing a substantial project, took many more hours to complete than I would care to admit. However, I am proud that I powered through and found workarounds when needed. I think this project is a great beginning for an even bigger project. I have demonstrated that Conway, along with other communities, needs a website that consolidates many different types of information. Which can be useful for permanent residents, temporary residents (e.g. college students), and those looking to move into the area.

In the future, if I decide to continue working on my project or if someone else makes their own using my tutorial, there are a couple of additions/changes that should be made. Creating a functional calendar for the events page that loads in event data from Firebase is at the top of the list. This was one of the problems I ran into with my dependency issues; however, it is possible to create a React calendar from scratch rather than using an open source library.

Another aspect that would be nice to incorporate is adding a Traffic layer on top of the Google Map. It is possible, but the amount of work and research that I had to do just to get the a maps API to display on the page took most of my time. There are also a couple of other maps APIs that I could see being used instead of Google maps. This includes HERE maps API and the MapQuest API.

Finally, getting community input for the website and making the changes based on their feedback would be another important step. This was part of my plan; however, it got derailed when it took me longer to make the website.

I also hope that if anyone decides to use my tutorial to build a website for their community, they learn enough from my tutorial to make new and better changes to their website. I think my tutorial is a great jumping-off point to learning the world of React.

# Bibliography


[1] 4 Reasons Why You Should Prefer Vite Over Create-React-App (CRA). https://semaphoreci.com/blog/vite#:~:text=Vite%20is%20a%20next%2Dgeneration,%2C%20jsx%2C%20and%20dynamic%20import. Accessed: 2023-11-21.

[2] 7 Reasons To Shop Local And Support Small Businesses. https://www.forbes.com/sites/forbesfinancecouncil/2022/06/28/7-reasons-to-shop-local-and-support-small-businesses/?sh=64caa7ea50d1. Accessed: 2023-11-9.

[3] Civic Participation. https://health.gov/healthypeople/priority-areas/social-determinants-health/literature-summaries/civic-participation#:~:text=Volunteering%20is%20a%20common%20form,that%20can%20yield%20health%20benefits.&text=Studies%20show%20that%20volunteers%20enjoy,and%20more%20positive%20emotional%20health. Accessed: 2023-12-11.

[4] Having a Lot of a Good Thing: Multiple Important Group Memberships as a Source of Self-Esteem. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4446320/. Accessed: 2023-12-11.

[5] Leaflet. https://leafletjs.com. Accessed: 2023-11-8.

[6] Vite + Firebase : How to deploy React App. https://medium.com/@rajdeepmallick999/vite-firebase-how-to-deploy-react-app-5e5090730147. Accessed: 2023-12-10.

[7] What is an API? https://www.coursera.org/articles/what-is-an-api. Accessed: 2023-11-8.

[8] Why and when to use API keys. https://cloud.google.com/endpoints/docs/openapi/when-why-api-key. Accessed:2023-12-10.

[9] United States Census Bureau. QuickFacts Conway city, Arkansas. https://www.census.gov/quickfacts/conwaycityarkansas. Accessed: 2023-10-30.

[10] Heavy.AI. Open source library definition. https://www.heavy.ai/technical-glossary/open-source-library#:~:text=An%20open%20source%20library%20is,and%2For%20publish%20without%20permission. Accessed: 2023-11-8.

[11] Michela Lenzi, Alessio Vieno, Gianmarco Altoè, Luca Scacchi, Douglas D. Perkins, Rita Zukauskiene, and Massimo Santinello. Can Facebook Informational Use Foster Adolescent Civic Engagement? *American Journal of Community Psychology*, 2015.

[12] Advisory Council on Historic Preservation. Conway, Arkansas. https://www.achp.gov/preserve-america/community/conway-arkansas. Accessed: 2023-10-30.

[13] Corrine Robinson. Conway. https://www.faulknerhistory.org/communities/conway/. Accessed: 2023-10-30.

[14] Data USA. Conway, AR. https://datausa.io/profile/geo/conway-ar. Accessed: 2023-10-30.