Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation

Sisyphus Table Website Documentation

Project Overview

This React application documents my process for building an actual Sisyphus Table. A beautiful eye-catching piece of functional art that combines my electronics, coding, woodworking, and mechanical systems skills into one. And now, my website design skills. The website serves as a portfolio piece showcasing the build process for the table through photos, descriptions, and technical details. And a portfolio piece showcasing my website design skills working within React. The site features four pages with routing, responsive design, and interactive components including a carousel, accordion, modal windows, and direct components for one time use to expand and showcase the site.

****For a look at all installs, plugins, and/or code in documentation, please refer to the README.md in GitHub Repo.**

Some Key Features

- Four distinct pages with React Router navigation
- Responsive design with mobile hamburger menu
- Interactive components (Carousel, Accordion, Modal, ImageModal, etc.)
- External data sources for content management
- Contact form with entry validation
- Photo gallery with built in modal viewer

Technology Used

- React - Component-based UI development
- React Router DOM - Client-side routing between pages
- Tailwind CSS 3.x - Utility styling from Tailwind directly with custom CSS for specific components
- JavaScript ES6+ - Modern JavaScript standards

Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation

How I Set Up Page Routing

I implemented page routing using React Router, which required installing the react-router-dom package. The routing structure uses a parent Layout component that contains common elements like the Header, Navbar, and Footer, with child routes for Home, About, Contact, and BlogPost pages that render their unique content in an Outlet location within the Layout.

For navigation between pages, I used React Router's Link component instead of traditional anchor tags. This prevents full page reloads and creates a smooth single-page application experience where only the content area changes while the header, navbar, and footer remain in place. The routing includes an index route for the home page and dedicated paths for each of the other pages.

How I Passed Data via Props

Throughout the application, I used props to pass data from parent components to child components, which makes components reusable. The Header component receives props for the background image and font family, the Carousel receives an array of photo objects from an external data file, and the Footer receives styling and link information. This approach means I can use the same component in different places with different data just by changing the props.

The Modal components use a special children prop that allows me to pass any content between its opening and closing tags. This makes the Modals extremely flexible, and I can use the same Modal component on four different pages with completely different content simply by passing different children. I also created separate data files for carousel photos and contact information that get imported and passed as props to their respective components. This use of Separation of Concern (SoC) allows me to easily alter content of one without affecting others downstream.

How I Established and Used React Hooks

I used the useState hook to manage component state throughout the application. The most common use was controlling modal visibility. Pages have a state that tracks whether the modal is open or closed, which changes when users click buttons to open the

modal and to close the modal. I also used useState for the hamburger menu's toggle state, the carousel's current slide tracking, and managing the contact form's input values.

For more complex interactions like the photo gallery on the BlogPost page, I combined multiple state variables. One state tracks whether the modal is open, while another stores which image's information should be displayed. When someone clicks a gallery photo, both states update simultaneously, with one receiving the image/data and the other opening the modal. I was able to use a single modal component to display different images and descriptions based on which photo was clicked.

How I Implemented the Hamburger Menu

The hamburger menu combines React state with CSS media queries to create responsive navigation. I added state in the Layout component to track whether the menu is open or closed, and a button that toggles this state when clicked. The navigation element receives a conditional class name that changes based on the state, which the CSS uses to show or hide the menu.

The styling uses a media query at 768 pixels to differentiate between desktop and mobile views. On desktop, the hamburger button is hidden and the menu displays horizontally. On mobile, the button appears and the menu is hidden by default until the state adds the "isOpen" class, at which point the menu (and content) is displayed vertically.

How My Components Are Related

The application follows a hierarchical structure with the App.js component managing all routing at the top and the Layout.js component being used to wrap all pages with consistent elements. Layout.js provides the Header, Navbar, and Footer, while an Outlet component determines where page-specific content renders. This creates consistency across the site with each page able to display its unique content.

Data flows into specific pages from external files, going through page components and then down to smaller components via props. For example, images for the carousel are stored in an array in a separate file, gets imported by the Home page, and are then finally passed to the Carousel component and displayed. But it allows for reusability, it’s not just one data set to one component. The Modal component is reused across multiple pages with different content, but components like Carousel and Accordion have their own

dataset. This aligns with SoC guidelines to keep code maintainable and makes it easy to update content without modifying component logic.

How I Styled My Components

I primarily used Tailwind CSS utility classes applied directly in the JSX for most styling needs on the components taking from or found in Material Tailwind, Foil, or Tailwind CSS. Tailwind provides layout utilities for flexbox and grid systems, spacing utilities for margins and padding, typography classes for text sizing and fonts, and color utilities for the indigo and purple theme. This approach allowed for rapid development getting to grab them directly from components defined in the many Tailwind documents. Felt like cheating because I didn't have to write much custom CSS, just had to tie it into my existing website.

For specific features like the header, navbar, footer, hamburger menu, and some custom content on each page outside of the components I created custom CSS. I created custom CSS files specific to each page based on the needs that arose for each. The navbar has a custom gradient background and hover effects, and the responsive behavior for the hamburger menu required media queries in CSS. I also created page-specific CSS files to handle layouts that need to switch from side-by-side on desktop to stacked on mobile. The combination of Tailwind utilities and targeted custom CSS provided both speed and flexibility.

How I Dealt with Difficulties and Resolved Them

The hamburger menu initially appeared on mobile but wouldn't toggle open when clicked. The state was updating correctly, and I saw the class was being added, but the menu still didn't show. The issue was from the CSS, the media query needed explicit instructions to hide the menu by default on mobile and only show it when the "isOpen" class was present. Restructuring the CSS with specific selectors resolved the problem.

Images weren't loading correctly and I was getting errors in the console. I referenced images with paths going through the public folder but React serves the public folder as the root directory, so those paths were incorrect. Removing the public folder reference from the paths fixed the issue. I also had trouble with props not reaching components, which came down to using incorrect relative paths in imports, using the right number of dots to navigate up directories solved it!

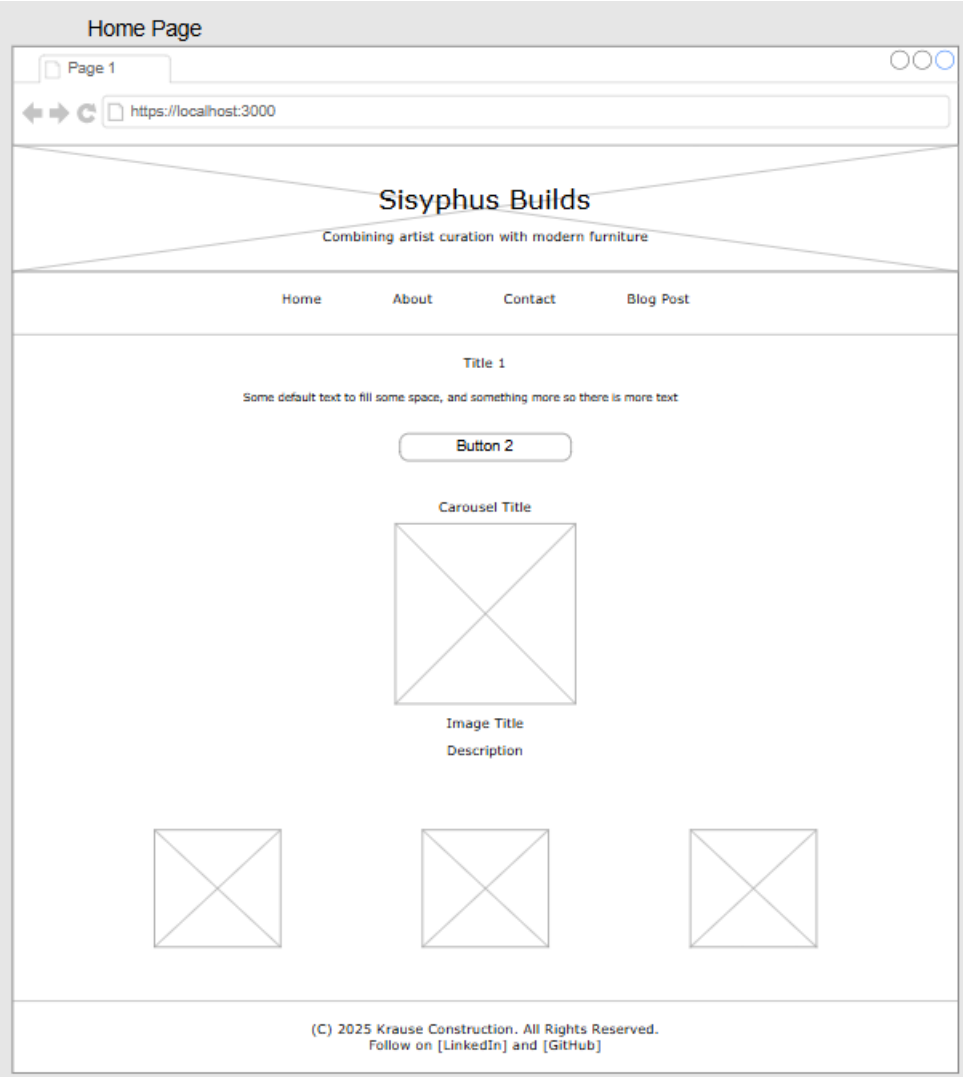
Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation

With each component brought in using Tailwind it did take time to work through what CSS I should use to link it to the site and get its placement and tie it into datasets to display data. For a few I had to do a full conversion from HTML to React that took a fair amount of time and testing. In the end it all started working as expected!

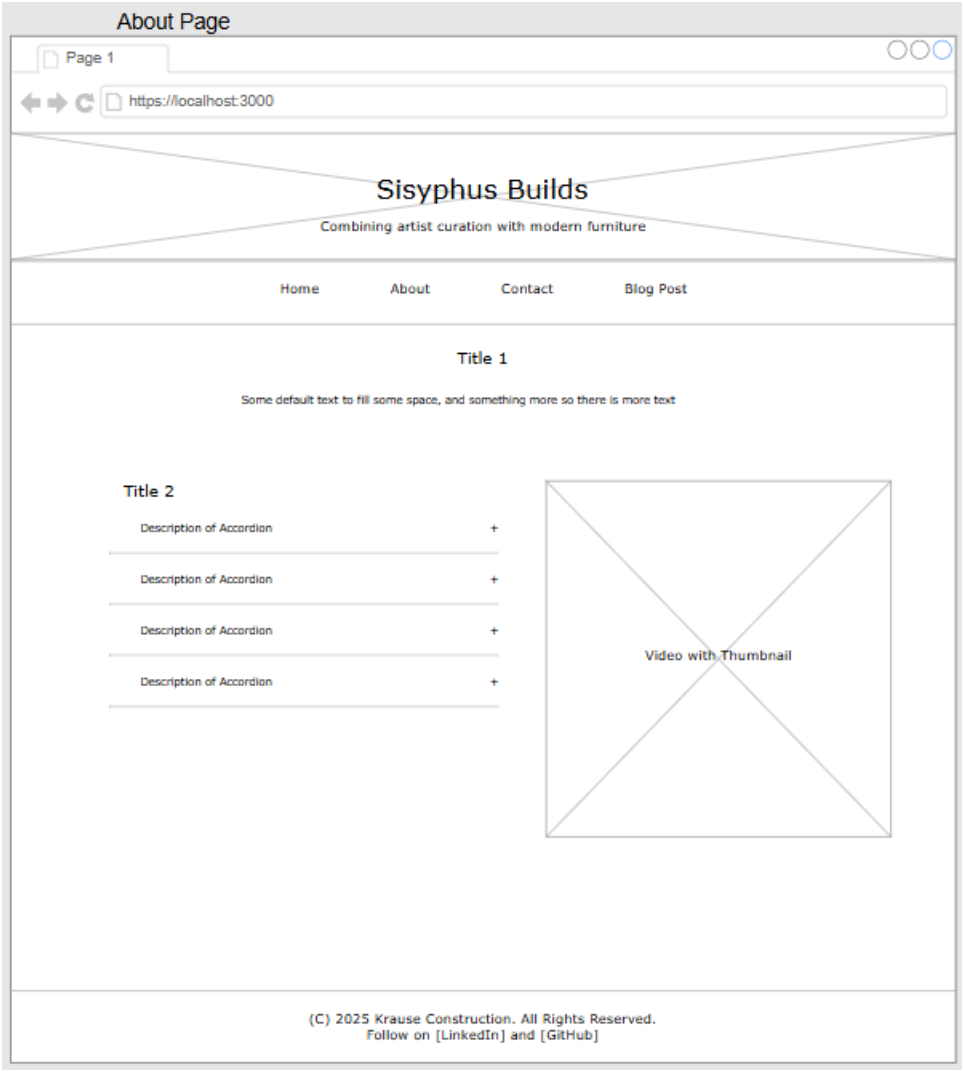
Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation

Wireframe

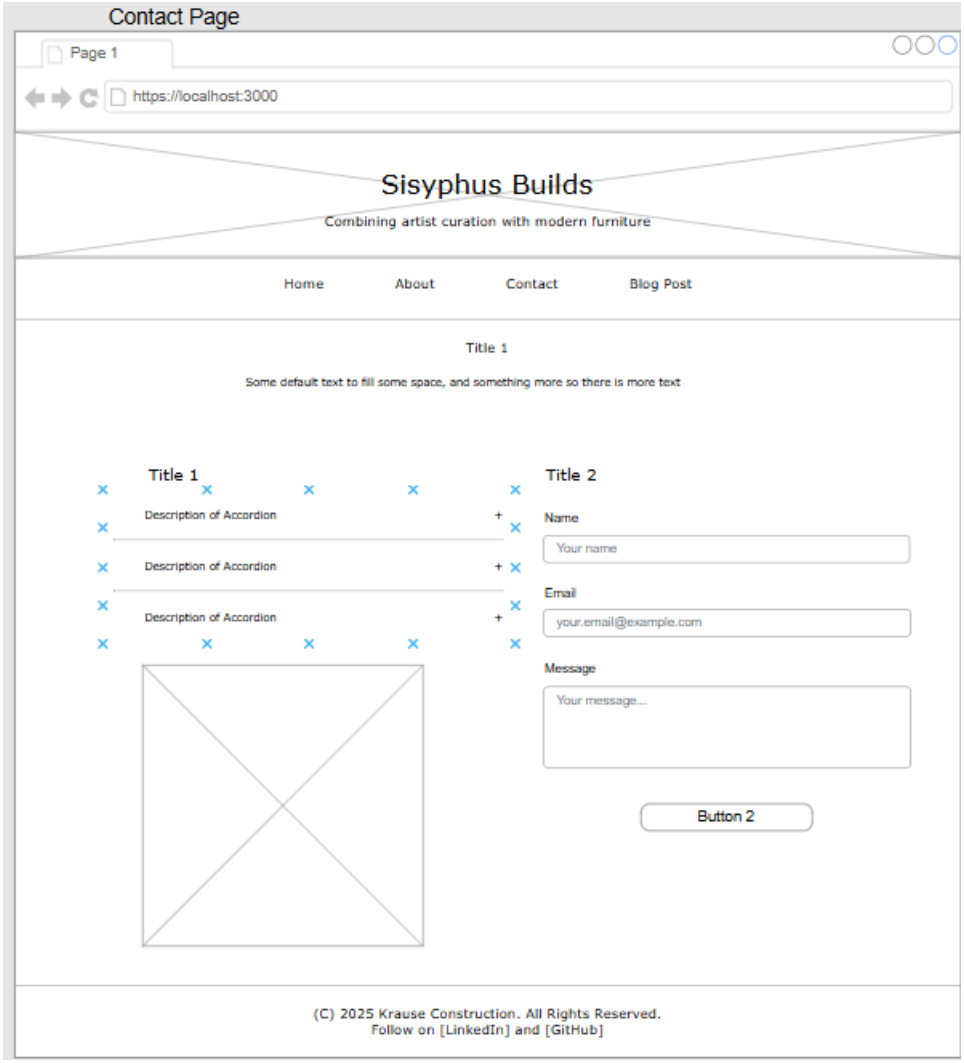
Desktop View >= 768px



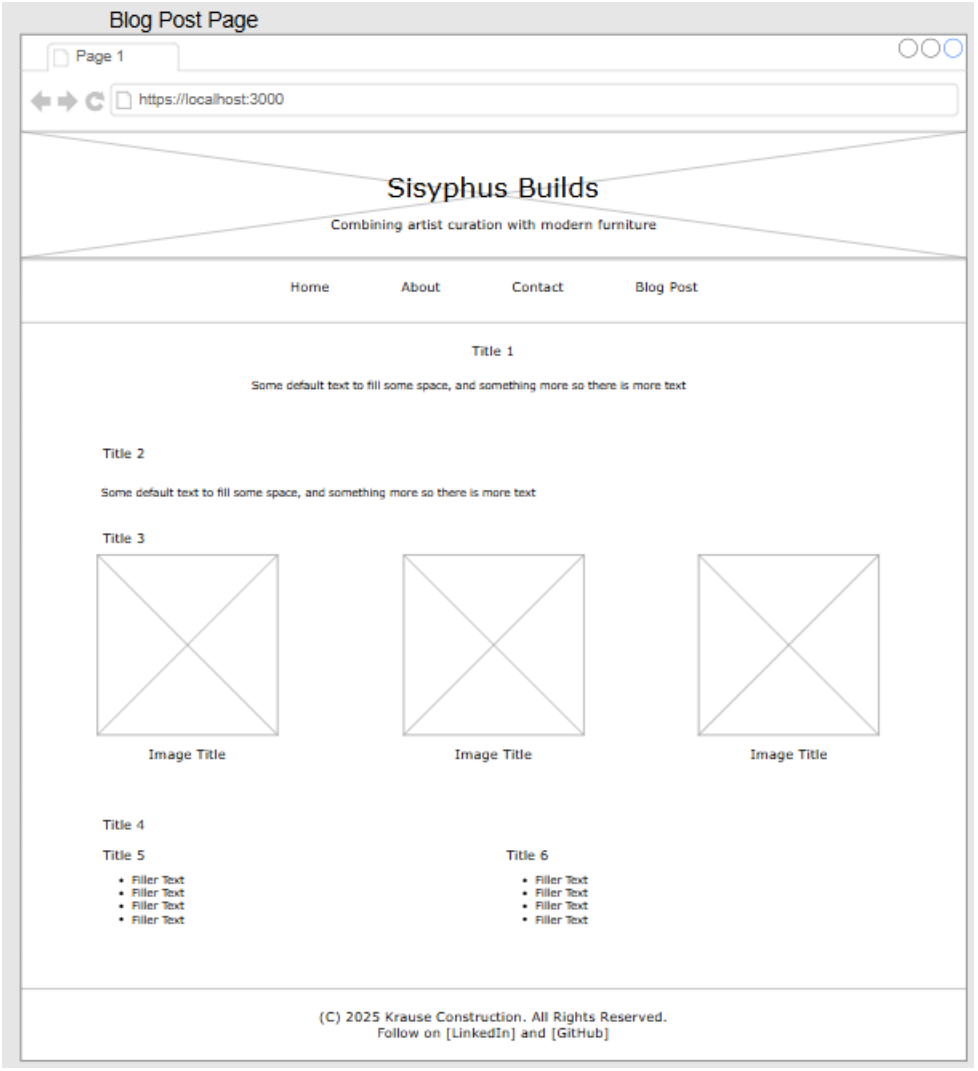
Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation



Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation

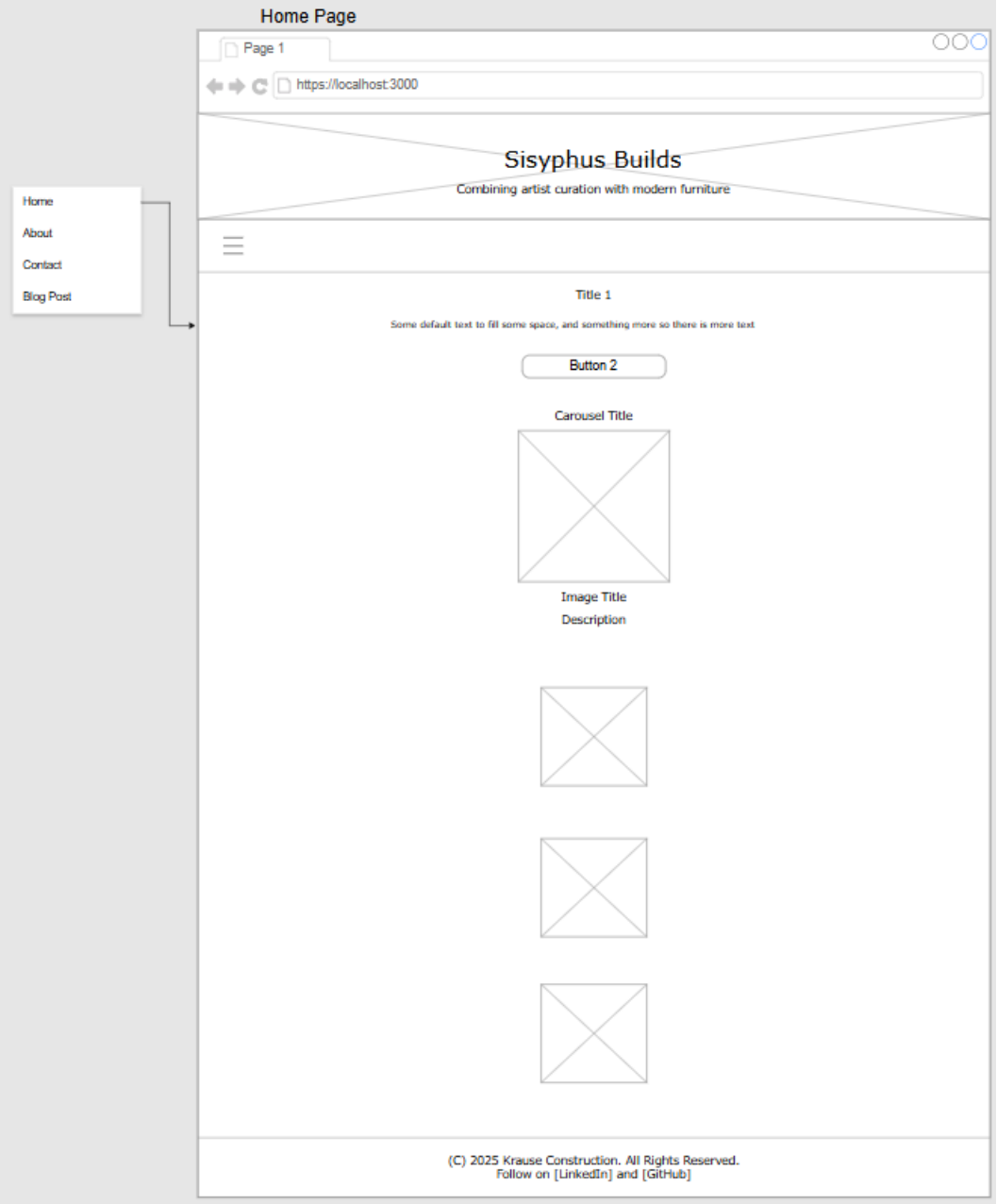


Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation

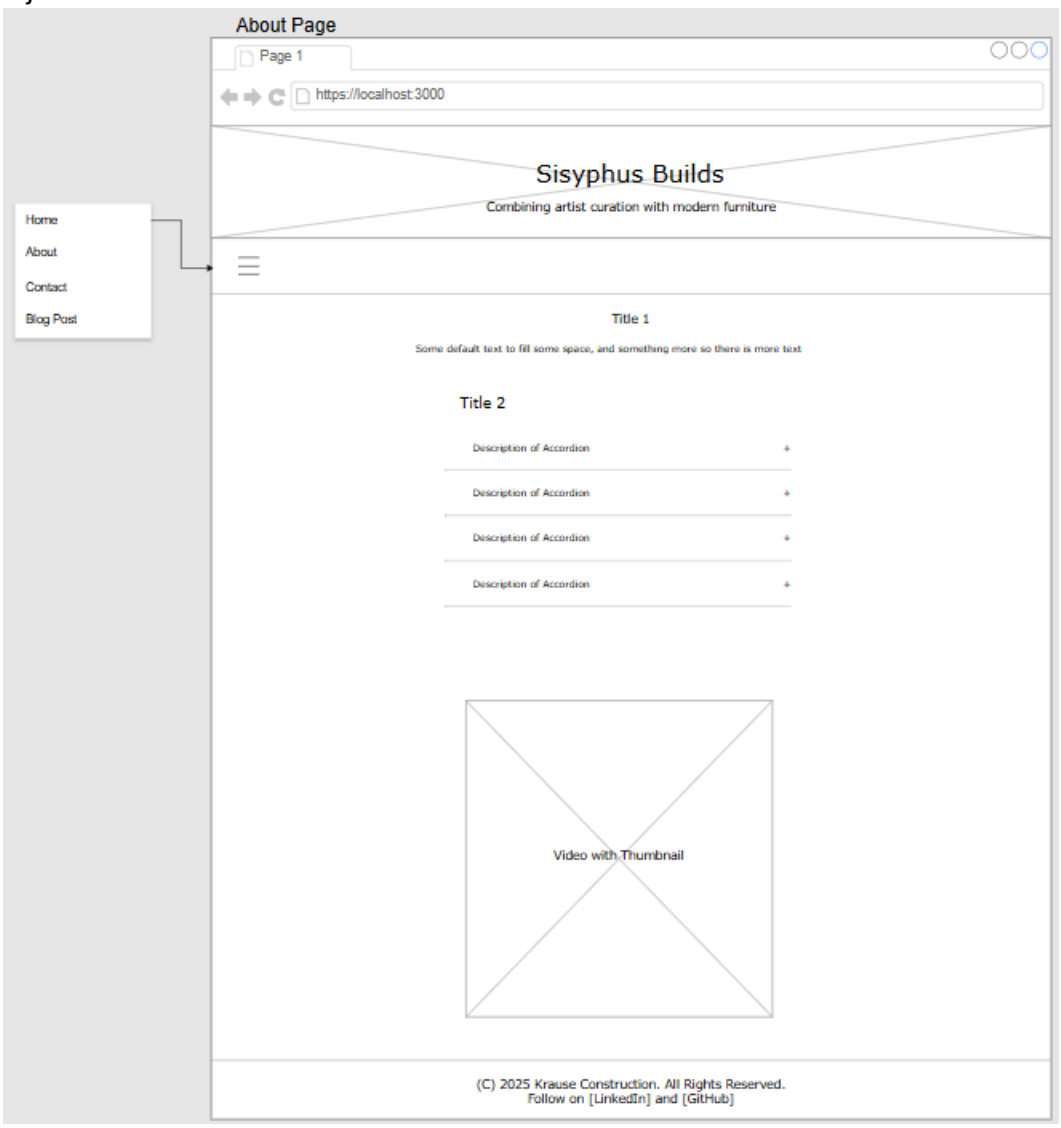


Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation

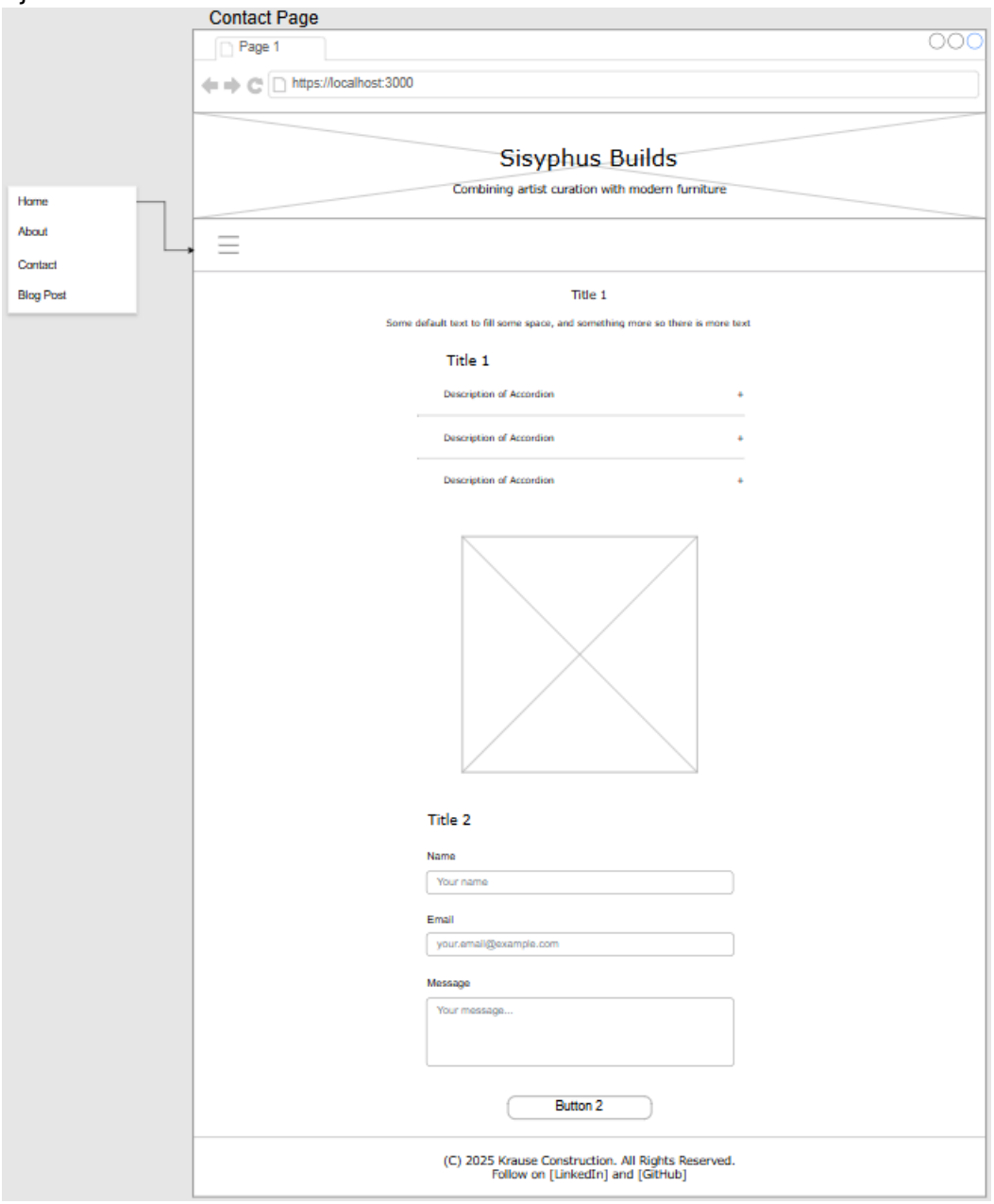
Mobile View < 768px



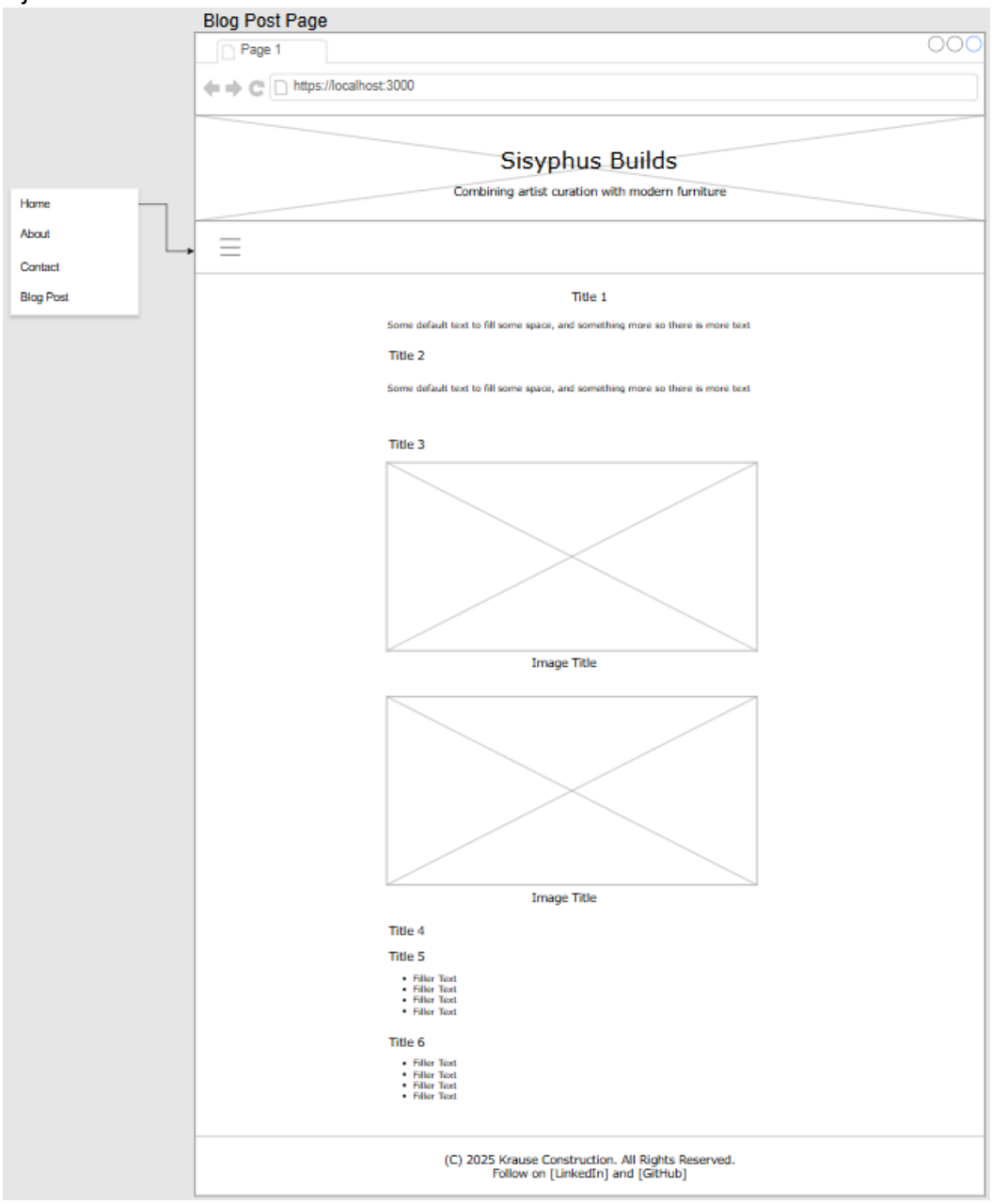
Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation



Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation



Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation



Michael Krause
CIT 313 – Commercial Website Design
Final Project – React Documentation

Demo Video Presentation

https://1drv.ms/v/c/adb938b61298731d/EU8fUEcyLDlNqitf-opbsVwBcpL4KzGyNhSuG0_u0JCKHg?e=ghugeP

Video Demo.mp4

Video Demo.zip