Course Outline

Course Title: Front-End Web Development

Duration: 6 Months

Tech Stack: HTML, CSS, SASS/SCSS, JavaScript, Tailwind CSS, React, React Router, Redux, React Query, Git/GitHub, Hosting (Netlify, Vercel, GitHub Pages), Supabase, and More

---

Month 1: Web Foundations (HTML, CSS, Git/GitHub)

Week 1: Introduction to Web Development

Overview of the Web: How Websites Work, the Role of a Front-End Developer

- **Understanding Web Architecture**: Students will learn about the client-server model, which forms the backbone of how websites operate. This includes how browsers (clients) request resources from a server, and how those resources (HTML, CSS, JavaScript, and media files) are sent back and rendered as a website.
- **The Role of a Front-End Developer**: Front-end developers focus on the visible parts of a website or web application. This week, students will explore the responsibilities of a front-end developer, which include building user interfaces, creating responsive designs, ensuring website accessibility, and optimizing user experience (UX). They will also learn about how front-end integrates with back-end technologies through APIs and other data services.

Setting Up the Development Environment

- **VS Code Setup**: Visual Studio Code (VS Code) is a popular code editor among developers. Students will learn how to install and configure VS Code, along with helpful extensions (e.g., Live Server, Prettier, ESLint) that enhance the development process.
- **Browser Developer Tools**: Students will be introduced to the browser's Developer Tools, a powerful feature for inspecting and debugging HTML, CSS, and JavaScript directly in the browser. This will include learning how to examine the DOM, modify styles live, and check network requests.

Introduction to Version Control with Git and GitHub

- **Git Basics**: Git is essential for version control in software development. Students will learn how to install Git, initialize a Git repository, track changes, and create commits. They'll understand how Git helps in maintaining a history of code changes and facilitates collaboration in teams.

- **GitHub Overview**: GitHub is the most widely used platform for hosting and managing Git repositories. Students will create their GitHub accounts and learn how to create repositories, clone them locally, and push commits to GitHub.
- **Branches and Pull Requests**: They will explore the concept of branches, which allows developers to work on new features without affecting the main codebase. This week, students will learn how to create branches, make changes, and open pull requests to propose code changes.

HTML Basics: Structure, Semantic Elements, Headings, Paragraphs, Links, and Images

- **HTML Structure**: Students will start by learning the basics of HTML (Hypertext Markup Language), which forms the structure of web pages. Topics will include understanding tags, elements, and attributes, as well as how HTML documents are structured with `<!DOCTYPE html>`, `<html>`, `<head>`, and `<body>`.
- **Semantic Elements**: Semantic HTML elements like `<header>`, `<main>`, `<article>`, `<section>`, and `<footer>` will be introduced. Students will learn the importance of semantic HTML for accessibility and SEO.
- **Creating Links and Images**: The session will focus on adding links using `<a>` and inserting images using `<img>`, ensuring students understand relative and absolute URLs and the significance of the `alt` attribute for image accessibility.

Week 2: HTML and Git/GitHub

Lists, Tables, Forms, and Input Types

- **Creating Lists**: Students will learn how to create ordered lists (`<ol>`) and unordered lists (`<ul>`) with list items (`<li>`), understanding the different use cases for each type.
- **Tables**: This will cover creating basic tables using `<table>`, `<tr>`, `<th>`, and `<td>`, and students will build structured data tables to organize information visually.
- **Forms and Input Fields**: Forms are crucial for gathering user data. Students will explore form elements like `<form>`, `<input>`, `<textarea>`, `<select>`, and different input types such as `text`, `email`, `number`, `password`, `checkbox`, and `radio`.

Using GitHub Pages for Hosting Simple HTML Websites

- **GitHub Pages**: Students will learn how to publish their projects online using GitHub Pages, a free hosting service provided by GitHub. This includes pushing code to a GitHub repository, configuring the repository for Pages, and making a simple HTML site accessible via a public URL.

Advanced Git Concepts: Branching, Merging, and Resolving Conflicts

- **Branching and Merging**: This week, students will dive deeper into Git’s branching model. They’ll create new branches to experiment with features, merge them back into the main branch, and handle merge conflicts if they arise.
- **Resolving Merge Conflicts**: When two branches modify the same part of the code, Git will require the user to manually resolve the conflict. Students will practice resolving these conflicts using VS Code’s built-in Git tools.

Collaborative Coding with GitHub

- **Cloning Repositories**: Students will learn how to clone repositories from GitHub, making it possible to contribute to open-source projects or collaborate on team projects.
- **Forking and Pull Requests**: Forking is a method of copying someone else's repository, making changes, and then proposing those changes back to the original project via a pull request. This will introduce students to open-source collaboration.

Week 3: Introduction to CSS

CSS Syntax, Selectors, Properties, and Values

- **CSS Syntax**: CSS (Cascading Style Sheets) defines the styles for a webpage. Students will learn the syntax of CSS, which includes selectors, properties, and values. They'll practice writing basic rules like changing colors, fonts, and spacing.
- **CSS Selectors**: This will cover the different types of CSS selectors, such as element selectors (`p`), class selectors (`.classname`), ID selectors (`#idname`), and attribute selectors (`input[type="text"]`). Students will learn how to use these to target specific elements in their HTML.

The Box Model: Margins, Padding, Borders

- **Understanding the Box Model**: The CSS box model is essential for controlling layout. Students will learn how every element on a webpage is considered a rectangular box, consisting of margins, padding, borders, and the content itself. Understanding this concept is key to manipulating layout.

Colors, Backgrounds, Borders, and Text Styling

- **Text Styling**: CSS allows developers to style text with properties like `color`, `font-size`, `font-family`, `font-weight`, `text-align`, and `line-height`. Students will practice making their text elements look polished and readable.
- **Backgrounds and Borders**: Students will learn to apply backgrounds (colors, images) and borders to elements using properties like `background-color`, `background-image`, `border-width`, and `border-radius`.

Using Git for Project Version Control

- **Committing and Pushing Changes**: This session will emphasize the use of Git for tracking changes during CSS styling. Students will make style changes, commit those changes with meaningful messages, and push their updates to GitHub.

Week 4: Layout Techniques and Responsive Design

CSS Positioning: Static, Relative, Absolute, Fixed

- **Positioning Concepts**: CSS positioning controls how elements are placed in relation to the browser window or other elements. Students will learn the four main positioning methods—static, relative, absolute, and fixed—and how to use them for specific layout needs.

- **Z-index and Stacking Context**: In addition, students will explore the `z-index` property to control how elements stack on top of each other, important for creating overlays and modals.

Flexbox Introduction: Creating Flexible, Responsive Layouts

- **Understanding Flexbox**: Flexbox is a modern CSS layout system designed to make it easier to create responsive layouts. Students will dive into the basic properties for the flex container (`display: flex`, `justify-content`, `align-items`, `flex-direction`) and the flex items (e.g., `flex-grow`, `flex-shrink`, `flex-basis`).

Media Queries for Responsive Design

- **Responsive Design**: Media queries allow developers to apply CSS based on the width of the browser window or device. Students will learn how to use media queries to create breakpoints in their CSS, making their web pages adapt seamlessly to different screen sizes, from mobile to desktop.

Deploying Static HTML/CSS Projects Using GitHub Pages

- **Deployment**: To wrap up, students will deploy their fully styled, responsive HTML/CSS projects using GitHub Pages, making their work live and accessible online. They'll also learn best practices for organizing their project files and preparing them for deployment.

Month 2: Advanced CSS, SASS/SCSS, Tailwind CSS, and Introduction to JavaScript

---

Week 5: Advanced Flexbox and Grid

Flexbox Deep Dive: Properties for Containers and Items

- **Flexbox Properties**: Students will go beyond the basics of Flexbox, exploring advanced properties like `align-content`, `flex-wrap`, and `align-self` for controlling how flex items behave within a flex container. Practical exercises will focus on building complex layouts like navigation bars, footers, and card layouts.

Introduction to CSS Grid: Creating Grid-Based Layouts

- **CSS Grid Layout**: Grid is a two-dimensional layout system that offers even more flexibility than Flexbox. Students will learn the fundamentals of CSS Grid, including `grid-template-columns`, `grid-template-rows`, `grid-area`, and `gap`. They'll practice building multi-column and multi-row layouts with ease.

Practical Project: Building Responsive Layouts with Flexbox and Grid

- **Project**: Students will build a responsive web

layout using both Flexbox and CSS Grid, applying their knowledge to create a site that adapts to different screen sizes while maintaining a clean, organized structure.

Week 6: SASS/SCSS

Introduction to SASS/SCSS: Variables, Nesting, and Mixins

- **What is SASS/SCSS?**: SASS (Syntactically Awesome Style Sheets) and its more modern version SCSS are CSS preprocessors that make CSS more powerful and maintainable. Students will learn how to install and configure SASS in their projects, using variables for reusable values (e.g., colors, font sizes), nesting to organize CSS more logically, and mixins for reusing blocks of styles.

Using Partials and Imports for Organizing Styles

- **Partials and Modularity**: One of the key benefits of SASS/SCSS is the ability to split styles into smaller files (partials) and then import them into a main stylesheet. Students will learn how to break down a large CSS file into manageable pieces, promoting cleaner and more maintainable code.

Building a Simple Design System with SASS

- **Design Systems**: Students will use SASS to build a simple design system. They'll define a consistent color palette, typography, spacing, and reusable UI components like buttons and cards, ensuring their designs are cohesive across pages.

Week 7: Tailwind CSS Basics

Introduction to Tailwind CSS: Utility-First CSS Framework

- **Utility-First CSS**: Tailwind CSS is a utility-first CSS framework that provides low-level utility classes for building custom designs without writing custom CSS. Students will explore the philosophy behind utility-first frameworks and how they differ from traditional approaches.

Setting Up Tailwind in Projects

- **Installation**: Students will learn how to install Tailwind CSS in their projects, configure Tailwind's `tailwind.config.js` file, and compile the necessary styles using PostCSS or a build tool like Vite or Webpack.

Common Utilities in Tailwind: Typography, Spacing, and Grid

- **Utility Classes**: Students will dive into Tailwind's rich set of utility classes for managing typography (`text-lg`, `font-bold`), spacing (`mt-4`, `p-2`), layout (`grid`, `flex`), and more. They'll learn how to combine utilities to create complex, responsive layouts quickly.

Responsive Design and Customization with Tailwind

- **Responsive Utilities**: Tailwind makes it easy to implement responsive designs with its mobile-first utilities (`md:`, `lg:`). Students will also learn how to customize Tailwind's configuration file to suit their project needs, overriding default values for colors, spacing, or breakpoints.

Week 8: JavaScript Fundamentals

Introduction to JavaScript: Variables, Data Types, Operators

- **JavaScript Basics**: Students will be introduced to JavaScript, the programming language of the web. They'll learn about the basic building blocks of JavaScript, including variables (`let`, `const`), data types (strings, numbers, booleans, arrays, objects), and operators (arithmetic, comparison, and logical operators).

Functions, Control Flow (if/else, loops)

- **Functions**: Students will learn how to define and invoke functions, passing arguments and returning values. This will cover both traditional function syntax and arrow functions.
- **Control Flow**: Control structures like `if/else` statements and loops (`for`, `while`, `forEach`) will be introduced to handle decision-making and repeating tasks in JavaScript code.

Arrays and Objects in JavaScript

- **Data Structures**: Arrays and objects are foundational data structures in JavaScript. Students will learn how to create, manipulate, and loop through arrays (`push`, `pop`, `map`) and work with objects by accessing and modifying their properties.

Introduction to External Hosting: Setting Up a Tailwind Project on Netlify

- **Netlify Hosting**: To close the month, students will deploy their first Tailwind CSS project on Netlify, a modern platform that makes it easy to deploy static websites. This will introduce them to the deployment process, ensuring their projects are live on the web.

Month 3: JavaScript, DOM Manipulation, and External APIs

Week 9: DOM Manipulation and Events

Selecting and Modifying DOM Elements

- **The DOM (Document Object Model)**: Students will explore how the DOM represents a webpage's structure as a tree of elements. They'll learn how to interact with the DOM using JavaScript by selecting elements with methods like `document.getElementById()`, `querySelector()`, and `querySelectorAll()`.
- **Modifying Elements**: Students will practice changing the content and styles of DOM elements dynamically, using properties like `innerHTML`, `textContent`, and `style`. They'll also learn to add and remove classes with `classList`.

Handling Events: Click, Input, Submit

- **Event Listeners**: Events are how JavaScript responds to user interactions on the webpage. Students will learn to attach event listeners (`addEventListener()`) to elements for common

events like clicks (`click`), input changes (`input`), and form submissions (`submit`). They will explore event objects to understand how to capture event-specific details.
- **Preventing Default Behavior**: Students will learn how to prevent default browser actions (such as form submission or link redirection) using `event.preventDefault()`.

Building Interactive Elements: Tabs, Modals, and Form Validation

- **Practical Projects**: By the end of the week, students will build interactive components like tabs (content that switches based on user clicks), modals (pop-up windows), and simple client-side form validation (e.g., checking if fields are filled before submission). These small projects will emphasize how JavaScript can enhance UX.

Week 10: JavaScript ES6+ and Promises

Modern JavaScript Features: let/const, Arrow Functions, Destructuring

- **ES6+ Syntax**: ES6 (ECMAScript 2015) introduced many new features to JavaScript. Students will learn the difference between `let` and `const` for variable declarations, how to write more concise code using arrow functions, and how destructuring can simplify working with arrays and objects.
- **Template Literals**: They will also learn about template literals, which make string interpolation and multi-line strings easier, replacing traditional concatenation.

Introduction to Promises and Async/Await

- **Promises**: JavaScript promises are used to handle asynchronous operations (like API requests). Students will understand how promises work, the concepts of `resolve` and `reject`, and how to use `then()` and `catch()` to manage promise outcomes.
- **Async/Await**: Async/await is a more modern way to write asynchronous code in JavaScript. Students will explore how to use `async` functions and the `await` keyword to write cleaner, more readable asynchronous code.

Fetching Data from APIs: GET Requests, Working with JSON

- **APIs (Application Programming Interfaces)**: APIs allow front-end applications to fetch data from external sources. Students will learn how to use the Fetch API to make GET requests to a public API, handling the JSON responses with `response.json()` and rendering the data on the webpage.

Week 11: Project: JavaScript Interactive Website

Creating an Interactive JavaScript-Driven Website

- **Project Overview**: This week, students will apply everything they've learned about JavaScript to build a full-featured interactive website. This project will incorporate HTML, CSS, and JavaScript to create a website with dynamic content and user interaction.

Form Handling, Data Validation, Dynamic DOM Updates

- **Form Validation**: Students will implement client-side form validation, ensuring that inputs meet specified criteria (e.g., correct email format, required fields) before submission. They will use JavaScript to provide real-time feedback to the user.
- **Dynamic Content**: Students will build features like dynamically updating sections of the page without a full reload (e.g., loading new content into a div based on user actions), improving overall user experience.

Fetching Data from a Public API and Rendering it on the Page

- **Integrating an API**: The project will include fetching data from a public API (like a weather or movie API) and displaying that data dynamically on the webpage. This teaches students how to integrate external data sources into their applications, a critical skill for modern web development.

Week 12: Git/GitHub and JavaScript Project Deployment

Collaborating on Projects with GitHub (Teamwork Simulation)

- **Simulating Team Collaboration**: This week, students will simulate a professional environment by working in teams on GitHub. They'll experience real-world collaboration, creating branches for different features, submitting pull requests, reviewing code, and resolving merge conflicts. They'll also practice good Git commit practices, using descriptive commit messages and ensuring meaningful code history.

Hosting JavaScript Projects on Netlify or Vercel

- **Project Deployment**: Students will learn how to deploy their JavaScript projects to a live environment using services like Netlify or Vercel. These platforms allow for easy, continuous deployment from GitHub, so every push to the main branch updates the live project. They'll also learn how to configure build settings, environment variables, and custom domain names.

Finalizing and Deploying the Interactive JavaScript Project

- **Final Project Showcase**: By the end of the week, students will have a fully deployed, interactive JavaScript project. This project will serve as a capstone to the first three months of training, showcasing the student's understanding of web development basics, JavaScript, DOM manipulation, and API integration.

Summary for Months 1-3

By the end of the first three months of this training program, students will have developed a strong foundation in:

- **HTML & CSS**: Understanding web page structure, styling, and responsive design principles.
- **Git & GitHub**: Mastering version control, collaboration, and deployment workflows.
- **JavaScript Fundamentals**: Writing JavaScript code to manipulate the DOM, handle events, and interact with external APIs.

- **SASS & Tailwind CSS**: Enhancing their styling capabilities with CSS preprocessors and utility-first frameworks.

Students will have completed multiple projects, including a final interactive JavaScript-driven website, preparing them for the more advanced topics to come in the next phase of the course.

In **Months 4-6**, students will dive deeper into modern JavaScript, React, and state management techniques, moving from foundational skills to advanced front-end development practices.

Month 4: Introduction to React and State Management

Week 13: React Basics

- **Introduction to React: Component-Based Architecture** React is built around the concept of components—independent, reusable bits of code that represent parts of the user interface. Students will learn how to think in components by breaking down complex UIs into smaller, manageable pieces. This includes understanding how each component has its own logic, state, and rendering, making the development of complex user interfaces more maintainable and scalable.
- **Setting Up a React Project with `create-react-app`** The `create-react-app` CLI is an essential tool for bootstrapping React projects. Students will learn how to use this tool to create a new React project from scratch, configuring it to automatically handle modern JavaScript, JSX, and CSS. This also includes a hands-on demo where students will create and run their first React project locally.
- **JSX Syntax, Functional vs. Class Components** JSX (JavaScript XML) allows developers to write HTML-like code within JavaScript. Students will explore JSX syntax and the differences between React's functional components and class components. Emphasis will be placed on functional components and their simplicity and performance benefits, especially with the introduction of hooks in React 16.8.
- **Introduction to Props: Passing Data Between Components** Props (short for properties) allow data to be passed from one component to another. Students will learn how to pass and use props to create dynamic components that can reuse logic while rendering different outputs. This forms the foundation for building larger, interconnected components.

Week 14: State and Event Handling in React

- **Introduction to React State: `useState` Hook** State in React determines how a component behaves and renders. Students will be introduced to the `useState` hook, which is used to create and manage state in functional components. They will learn how to declare state variables, update them, and observe how state changes trigger re-renders in the component.
- **Handling Events and User Inputs** React's event handling system is similar to traditional DOM event handling, but with slight differences. Students will practice handling events like button clicks, form submissions, and input changes. They will also explore synthetic events and how to prevent default behavior in forms or anchor tags.

- **Building Simple Interactive Components** Using state and event handlers, students will build interactive components such as a to-do list or a counter. These exercises will reinforce their understanding of React's reactive nature—how components automatically update when the state changes.

Week 15: React Router

- **Introduction to React Router: Setting Up Routes and Navigation** React Router is essential for building multi-page applications in React. Students will learn how to set up client-side routing, creating routes that map to different components (pages). They'll explore how to link between pages and manage URL-based navigation without a page reload.
- **Dynamic Routing, Route Parameters** Dynamic routing is a powerful feature that allows the creation of flexible routes. Students will learn to handle route parameters (e.g., `/products/:id`) to pass dynamic data through the URL and render appropriate components based on the route.
- **Building a Multi-Page React App** Students will build a multi-page app, incorporating dynamic routing and navigation. This project will involve components like a home page, about page, product page, and a dynamic details page based on route parameters. Emphasis will be placed on the seamless user experience of React Router's navigation.

Week 16: Introduction to Redux

- **Introduction to Redux: State Management Across Components** Redux is a predictable state container for JavaScript apps. Students will learn the principles of Redux, including the unidirectional data flow and the concept of a single source of truth (the Redux store). They will understand how Redux helps manage application-wide state in complex React apps where multiple components need to share and manipulate state.
- **Setting Up Redux in a React Project** This includes installing the necessary Redux libraries and setting up the Redux store. Students will integrate Redux with their React app, configuring the `Provider` to make the store available throughout the component tree.
- **Creating Reducers, Actions, and Using the Redux Store** Students will write reducers, which are pure functions that take the current state and an action and return the next state. They will also define actions (JavaScript objects describing what happened) and learn how to dispatch these actions to update the Redux store. A focus will be on understanding how React components can access the Redux store's state via the `useSelector` and dispatch actions using `useDispatch`.
- **Project: Implementing Global State with Redux** In this hands-on project, students will build an app (e.g., a simple shopping cart) where state is shared across multiple components. They'll implement features like adding/removing items from the cart, and track the cart's total value using Redux.

Month 5: Advanced React, React Query, and Supabase

Week 17: External Data Fetching in React

- **Fetching Data with `fetch` and `axios` in React** Data fetching is a core requirement in most React apps. Students will learn to use `fetch` and `axios` to make HTTP requests to external APIs. They'll implement GET requests and handle the response, practicing asynchronous programming using JavaScript promises and `async/await`.
- **Handling Loading States, Errors, and Displaying Fetched Data** Students will build components that handle the full lifecycle of an API request, including loading states (e.g., showing a spinner while data is being fetched) and error handling (e.g., displaying an error message if the request fails). They'll learn how to structure React apps to keep the UI responsive and user-friendly during data fetching.
- **Introduction to React Query** React Query simplifies the management of server-side state by providing hooks that handle fetching, caching, and synchronizing data with external APIs. Students will learn the basics of React Query, including how to fetch data, refetch it automatically, and manage background synchronization.

Week 18: React Query and API Integration

- **Using React Query for Advanced Data Fetching** This week dives deeper into React Query, where students learn more advanced concepts like pagination, infinite scrolling, and automatic refetching on certain conditions (e.g., when the window refocuses). They will see how React Query optimizes API requests and reduces unnecessary re-fetching.
- **Caching, Synchronization, and Background Data Fetching** React Query provides built-in caching mechanisms, which can significantly improve performance. Students will explore how React Query caches data, updates it in the background, and keeps the UI synchronized with server-side changes.
- **Building a CRUD App with React Query** Students will implement a simple CRUD (Create, Read, Update, Delete) application that interacts with an external API. This will showcase how to manage data-fetching operations and handle form submissions and updates using React Query.

Week 19: Introduction to Supabase

- **What is Supabase? Overview of Backend-as-a-Service (BaaS)** Supabase is an open-source alternative to Firebase, providing an easy-to-use backend for applications. Students will get an overview of Supabase's key features, such as authentication, real-time databases, and API handling.
- **Setting Up a Supabase Project and Database** Students will sign up for a Supabase account, create a new project, and set up a PostgreSQL database. They will learn how to define tables, create relationships between them, and manage data through Supabase's GUI and RESTful APIs.
- **Integrating Supabase with a React App for Authentication and Data Storage** This session covers how to integrate Supabase with a React app to handle user authentication (sign-up, login) and store data in the database. Students will configure authentication methods (email/password, third-party providers) and set up CRUD operations through Supabase's API.

Week 20: Supabase Project

- **Building a Full-Stack React App Using Supabase** This project will involve building a fully functional front-end application powered by Supabase's backend. Students will implement user authentication and integrate Supabase for data storage, fetching, and updating data in real-time.
- **CRUD Operations with Supabase and React Query** Students will combine Supabase with React Query to create a smooth, optimized front-end experience that interacts with Supabase's API. They will implement all CRUD operations and use React Query's caching and synchronization features to manage the app's state effectively.

Month 6: Advanced Topics and Final Projects

Week 21: Advanced React: Context API and State Management

- **Introduction to the React Context API for State Management** The Context API allows state to be shared across components without the need for props drilling. Students will learn when and how to use the Context API, creating a global state that can be accessed by any component in the React tree.
- **Comparing Context API with Redux and React Query** Context API is best suited for certain scenarios, while Redux and React Query are better for others. Students will compare these tools in terms of scalability, ease of use, and performance, learning how to make informed decisions on which tool to use for different projects.
- **Refactoring an Existing App to Use the Context API** In this session, students will refactor an app that uses props drilling for state management to instead use the Context API. They will learn how to set up a global state, access it, and manage updates throughout the component tree.

Week 22: Deployment and Hosting

- **Hosting React Apps on Vercel, Netlify, and GitHub Pages** Students will explore different platforms for

hosting React apps, such as Vercel, Netlify, and GitHub Pages. They will learn to configure their apps for deployment, understanding how build processes work and ensuring their React apps are production-ready.

- **Environment Variables, Configuring DNS, and Handling SSL** Proper handling of environment variables is crucial for deploying secure and scalable applications. Students will learn how to use environment variables in React for things like API keys. They'll also configure custom domains and manage SSL certificates for securing their apps.
- **Project: Deploying React/Supabase Applications** In this project, students will deploy the full-stack app they built earlier (integrating React with Supabase) onto one of the hosting platforms. They'll set up environment variables, configure domains, and make their apps live on the internet.

Week 23: Final Project Preparation

- **Scoping and Planning the Final Project** Students will plan their final projects, defining the scope of what they'll build. They'll break down the project into manageable tasks, decide on which tools and technologies to use (React, Tailwind CSS, Redux/React Query, Supabase), and begin setting up the project structure.
- **Structuring the Application** A significant part of this preparation will involve setting up the foundational structure for the application. This includes organizing components, setting up state management (Redux/React Query/Context API), and integrating Tailwind CSS for styling.
- **Guidance and Mentoring on Advanced Project Topics** During this week, instructors will provide personalized guidance to students on advanced topics that they may wish to incorporate into their final projects, such as integrating third-party libraries, optimizing performance, or implementing advanced UI/UX patterns.

Week 24: Final Project and Assessment

- **Final Project: Building a Complete Front-End Application** In the final week, students will build a complete, production-ready front-end application. This project will showcase their ability to integrate everything they've learned, from React and Tailwind to Redux/React Query and Supabase. They'll implement full-stack functionality, including authentication, data fetching, and state management.
- **Final Assessment: Students Present Their Projects** Students will present their final projects to their peers and instructors, receiving constructive feedback on both their code quality and the overall user experience. This will simulate real-world scenarios where developers present their work to stakeholders or teams.
- **Course Wrap-Up: Portfolio Building and Career Readiness** After completing their projects, students will learn how to package their work into a professional portfolio. They'll receive guidance on job-seeking strategies, interview preparation, and how to present themselves as industry-ready developers.

Summary of Months 4-6

At the end of the final three months, students will:

- Have a strong understanding of **React**, from basic concepts like components and state to advanced topics like React Router and state management with **Redux** and the **Context API**.
- Be able to manage server-side state and API interactions using **React Query**, improving the performance and user experience of their apps.
- Have practical experience with **Supabase**, building full-stack applications with authentication and data handling.
- Understand how to **deploy** and host their projects using modern platforms like Vercel and Netlify.
- Complete a comprehensive **final project**, demonstrating their ability to build and deploy a real-world web application from start to finish.

By the end of the course, students will have the knowledge and experience to confidently pursue roles as front-end web developers and present their work in a professional portfolio.

Optional Extras/Extensions (Time Permitting)

- **Advanced Animations with Tailwind CSS and React** Students will explore advanced animation techniques using Tailwind CSS's utility classes combined with React. They'll build animated components, such as modals, accordions, and loaders, adding interactive and polished UI elements to their apps.
- **TypeScript with React** TypeScript is a superset of JavaScript that adds static typing to the language, reducing bugs and improving code quality. In this section, students will learn how to integrate TypeScript with React, using basic types, interfaces, and type annotations to make their applications more maintainable.
- **Next.js Overview** Next.js is a powerful React framework that adds features like server-side rendering (SSR), static site generation (SSG), and built-in routing. This section will introduce students to Next.js, comparing it to traditional React apps and exploring the additional flexibility it provides for SEO and performance optimization.
- **Deploying Next.js Projects with Vercel** Vercel, the company behind Next.js, makes it easy to deploy Next.js applications. Students will deploy a Next.js app, learning about features like incremental static regeneration, environment variables, and automatic SSL certificates for secure, scalable deployments.