# MERN

## Table of Contents

# 1 Week 1

## 1.1 Intro to Git & Setup

### 1.1.1 Before we begin

- What is Git?
- Understanding version control or source control as a way of tracking file progress.
- Snapshots and branches to navigate versions.
- Backups to avoid data loss/ damage.
- Benefits of using Git for course control (Popularity, documentation and support, integration with other applications).
- Git vs GitHub
- Overview of GitHub
- Storing remote repositories.
- Collaborating on projects.
- Using a linked list to maintain structure

### 1.1.2 Setup

- Download and install VS Code
- Download and install relevant extensions:
    - Live Server
    - Prettier – Code formatter

### 1.1.3 Git

- Initializing a Git repo
- Stages of a file: untracked, unmodified, modified, staged
- Staging area: adding and removing files, reverting unstaged changes
- Status
- Committing changes
- Branching
- Git restore vs rm
- Navigating between commits and branches
- Resolving conflicts and merging branches

# 2 Week 2

## 2.1 Before we begin

### 2.1.1 Introduction to Web Development

- Building blocks: HTML, CSS, JavaScript
- Uniform Resource Locator (URL)
- Document Object Model (DOM)
- Developer tool

## 2.2 HTML

### 2.2.1 Intro to HTML

- Head, body tags
- Div/span
- Headings and paragraphs
- Images
- Classes and IDs
- Meta Elements: Name, Content
- Anchor tag, difference between a link and a hyperlink
- Lists
- Table
- Input and its types
- Buttons
- Adding video

## 2.3 CSS

### 2.3.1 Basic CSS and Box Model

- Applying CSS with class or ID
- Colors and background colors
- Different ways to represent colors
- Font-family, size, weight, line-height
- Box model: padding, border, outline, margin
- Border style, color, width, radius
- Maximum and minimum height/width
- Text-decoration, transform, align
- List-style
- Background image, position, repeat
- Hover, active, visited
- Box-sizing: border-box
- Position

- Z-index
- Object fit
- Overflow
- Pseudo-elements: first-child, nth child
- Optional exercise: CSS exercises from w3schools and https://github.com/TheOdinProject/css-exercises

### 2.3.2 Flexbox

- Flex container:
    - Direction
    - Justify-cintent
    - Align-items
    - Wrap
    - Align-content
- Flex items:
    - Align-self
    - Order
    - Grow, shrink, basis
- Optional exercise: https://flexboxfroggy.com/

### 2.3.3 Grid

- Grid container:
    - Rows and columns
    - Template
    - Gap
    - Justify-content and align-items
- Grid item:
    - Start/end
    - Align-self and justify-self
- Optional exercise: https://cssgridgarden.com/

# 3 Week 3

## 3.1 Javascript

### 3.1.1 Basics of Javascript

- Linking a JS file with HTML: script tag, external file
- Dynamically vs statically typed languages & ECMAScript
- Variables, constants and console logs: let, var, const
- Primitive data types in JS
- Arithmetic and logical operators
- Operator precedence
- Statement vs expression in JS
- Template strings
- if/else, switch statement, ternary operator
- Type conversion and coercion
- Truthy vs falsy values
- == vs ===
- prompt()
- Short circuiting
- Functions
    - Parameters vs arguments
    - Function declaration vs function expression
    - Arrow function, anonymous function, IIFE
    - Default parameters
    - Pass by copy, pass by reference
    - Higher-order functions
- Arrays
    - Storing and accessing values in arrays
    - Methods: push, pop, shift, unshift, indexOf, includes
    - De-structuring
- Objects
    - Notations
    - Methods: keys, values, entries
    - De-structuring
- Loops
    - for, while
    - for-of loop
    - break statement, continue statement
    - nested loops
- Spread and Rest Operator
- Optional Chaining

### 3.1.2 Intermediate Javascript

- The `this` keyword
    - this keyword in simple functions and arrow functions
- The call, apply, bind methods
- Arrays
    - slice, splice, reverse, concat, sort, join
    - forEach loop
    - map, filter, reduce, find, flat, flatMap methods
- Promises and HTTPS Requests
    - Promise lifecycle
    - fetch
    - async/await
    - Chaining calls and converting to JSON
    - Handling errors with catch block
    - Finally block
- Local Storage

## 3.2 Exercise

- Open the following link and signup for a free API key: http://www.omdbapi.com/
- Make a page with two fields: name for the movie, year launched
- Add a button to submit the search query
- On the click of the button, make the API call and retrieve search results.
- First, just map a single movie into a HTML element and insert it into a UL element as an LI element
- Then, map the entire array into HTML elements using the map function and insert into UL as LI elements
- By this point, you'll have a list of movies on your page
- Now, implement the year launched filter
- If a value is present in the year launched field, read that value and filter the movies array for movies that are launched in or after that year
- Then show the updated list of movies with the year launched filter applied
- If no movie is found against the name or year entered by the user, show a relevant message.
- Lastly, the movie data on the screen must not be lost when the page is refreshed. For this, store the data in local storage when the page is being unloaded, and retrieve the data from local storage when page is loaded. If no data is present in local storage, show a blank list.

# 4 Week 4

## 4.1 React

### 4.1.1 Function-based Components

- VS Code Extension: Simple React Snippets by Burke Holland
- Transpiler: babeljs.io
- Virtual DOM and how React works
- Intro to JSX
- Project structure of create-react-app with vite
- Creating the first component
- Single parent in each component: div, fragment
- Exporting a component: default vs named
- Component's state : useState
- Styling a component: using className, using style object
- Passing props
- Rendering lists: map function
- Conditional rendering: short-circuiting, ternary operator, returning JSX from a function
- Rendering classes dynamically
- Types of event handlers
- Using arrow functions
- Passing event handlers as 'props'
- Passing parameters to event handlers
- Real-time searching with onChange event
- In-class exercise: Create two components, pass an event handler as a prop from one component to the other and simply add a log statement in the handler function.

### 4.1.2 Component Lifecycle and Uplifting the State

- UseEffect
- Unmounting
- Uplifting the state

### 4.1.3 API Calls with Axios and React-Query

- Making API calls with axios.
- Using async/await in useEffect
- Using then-catch-finally
- Using react-query with axios
- Caching in React-Query

### 4.1.4 Exercise

- Implement a to-do list using the concepts of components,
  state and props. Components to make:
- Task: Individual task entry.
- TaskList: The components that calls the map function on the entire task list and renders a Task component.
- AddTask: An input field and a button to add a new task.
- Create all these components in the ‘components’ directory in the ‘src’ folder.
- Also make a directory named ‘pages’ in ‘src’ folder and make a page (just like a .jsx component) named “Home”.
- Call the relevant components in the “Home” component and then call the “Home” component in App.js.
- In the end, store the current list in the local storage and on each refresh, retrieve this list. Use life-cycle methods to accomplish this task.
- Moreover, implement searching in the todo-list project

### 4.1.5 More hooks (Optional)

- useRef
- useMemo

# 5 Week 5

## 5.1 React Router

- Browser Router
- Routes
- Link
- Redirection using Navigate
- Route parameters: mandatory, optional
- Protected routes, maintaining past page state
- Outlets
- Nested routing

### 5.1.1 Exercise-1

- Initialize a new react app. This app will act as our complete React project for the entire 3 weeks.
  This will be an e-commerce app.

  For today, use axios to get data from the API at https://fakestoreapi.com. Make two folders in the src folder named components and pages. Make a page named Products and use your components (created in components folder here) and call this Products page in App.js.

  In the Products page, make a Card component for each product which will show its name, price, category and image. Make another component which will use this Card to render a list of products. This list component will also contain a search bar. The search bar functionality will be implemented as real-time searching i.e. change the displayed products as soon as something is typed in search bar.

  Use the correct method from react-query to fetch data from the API.

### 5.1.2 Exercise-2

- Implement routing for each of these pages in the following manner:
  - Home page must contain links to Categories and Products page.
  - In this case, Products page will list all the products from all the categories.
  - Categories page must lead to Products page.
  - In this case, the Products page will list products from the chosen category only
  - Products page will contain a list of products and the title of each of these products will be a clickable link which will lead to Product Details page. Use the Card component created earlier to list the products.
  - Product Details must contain detailed information for the selected product, a counter and an add to cart button
  - Each page will be located in src/pages folder and each component will be located in src/components folder.

    - After routing is implemented, you need to add filters to the Products page. These filters will be applied to the list of products on the Products page.
- These filters include:
    - Sort with price in ascending order
    - Sort with price in descending order
    - Sort in ascending alphabetical order based on title
    - Sort in descending alphabetical order based on title
    - Each of these filters must work fine in sync with the search functionality previously applied.

### 5.1.3 Exercise-3

- Use the concepts of Nested Routes and Outlets to achieve the following functionality:
    - Create a Navigation Bar and fix it on top of the page. Add links to All Products, Categories and Cart page in the nav bar. You can use the Navbar from react-bootstrap or material-ui. The Navigation Bar must be fixed and every page must render as an Outlet.
    - Create the Cart page. The parent-level URL for the Cart page will be /checkout. This page will show the list of components added in the cart.
    - Create two nested pages after /checkout. One of them will be the ‘user-details-form’ page and the other one will be ‘order-completed’ page.
    - The ‘/checkout’ page have a link to the ‘user-details-form’ and the ‘user-details-form’ will contain a link to the ‘confirmed-order’ page. These pages will be implemented later on. For now, just add simple headings and Link tags.

# 6 Week 6

## 6.1 React Formik

### 6.1.1 Creating forms

- Creating a form with formik
- Validation schema using yup

### 6.1.2 Exercise

- Implement Login, Signup and Checkout form for your e-commerce store. Backend for these forms will not be implemented for now. If a form submission passes all validation checks, redirect to the next page.
- In case of a successful signup, navigate to the login page. In case of successful login, navigate to the Home page.

## 6.2 React Context

### 6.2.1 Creating context

- createContext function
- Default values
- useContext hook

### 6.2.2 Exercise

- Upon successful login, generate a random number and store it in the local storage as ‘token’, along with the email/username of the user. When the user signs out, remove these items from the local storage.
  Create a User context and whenever the app is refreshed, check for this information in the local storage. If the information exists, the user must be logged-in by default. Retrieve and store this information into the User context and use this context to access this information throughout the app. If this information is not present in the local storage, the app must open with the user logged-out.

# 7 Week 7

## 7.1 Redux

- Three core concepts: action, reducer, store
- Three principles:
    - The state of entire application is maintained in a single store
    - The only way to change state is to emit/dispatch an action
    - Reducers will return new objects with updated state values
- Creating an action creator, reducer, store
- useSelector, useDispatch hooks
- Redux Logger
- Redux DevTools extension
- Redux Persistor
- Redux Toolkit (optional)

### 7.1.1 Exercise

- Implement the ‘Add to Cart’ button for each product and the cart functionality using Redux.

### 7.1.2 Miscellaneous Topics (Optional)

- Redux Toolkit
- React Router v6
- CRUD with Firebase
- Modular SASS
- Lottie Animations

# 8 Week 8

## 8.1 Node

### 8.1.1 Overview of Node.js and its architecture

- What is Node.js
- Event-Driven and Non-Blocking I/O
- Single-Threaded Event Loop
- CommonJS Modules
- Web Server Capabilities
- Scalability and Real-time Applications

### 8.1.2 Basic Concepts of Asynchronous programming in Node.js

- Callbacks
- Event Emitters
- Promises
- Async/Await

## 8.2 Express Framework

### 8.2.1 Installing Express and creating an Express application

- Set up a project directory
- Initialize a new Express project:
    - npx express-generator
    - Install node packages
    - Npm Install/ npm  i
- Create the project structure
- Implement the controller
    - (GET, POST, PUT, DELETE)
- Create the service
- Routing
- HTTP requests
    - (GET, POST, PUT, DELETE)
- Middleware
- Logging, parsing request bodies, handling authentication, and more.
- Controllers
- Services
- Request and Response Objects
    - BodyParams
    - QueryParams
- Validation

- Error Handling

# 9 Week 9

## 9.1 Database and ORM

### 9.1.1 Database Connection

- Set up environment
- Install prostgresql and PgAdmin
- How to define models
    - Attributes
    - Constraints

### 9.1.2 Database Relations

- Database Keys
    - Primary Key, Foreign Key , Compound Key, Composite Key, Unique Key
- Relations
    - 1:1, 1:M, M:M
    - Cascade
    - Paranoid

### 9.1.3 Database CRUD’ and ORM

- What is ORM?
- Sequelize
- CRUD operations
- Sorting, Filters, Clauses, pagination
- Include, exclude
- Joints
- Hooks
- Seeders
- Transactions

# 10 Week 10

## 10.1 REST API's, Postman

### 10.1.1 REST API's

- What are REST API's
    - Integrations
    - Expansion
    - Ease of maintenance
- Types of API's
    - Private, Public, Partner, Composite
- How to create end points
- API endpoints
    - Security
    - Performance

### 10.1.2 Postman

- Set up environment
- What is postman?
- How does it work?
- Testing API's

## 10.2 Authentication, Validation and Error Handling

### 10.2.1 Authentication

- Username and Password
    - Unique username
    - Encrypted password
- Bcrypt to encrypt password
- Token-Based Authentication
    - JWT token
- Implementing authentication
- Login, Logout and Reset Password API's
    - OTP

### 10.2.2 Validation and Error Handling

- What is validation?
    - How does it work?
    - How to handle error

- Throwing Custom errors

# 11 Week 11,12

## 11.1 Final Project

- Trainees will form groups and decide their roles and  projects