

Overall Description of the System

WORKOUT is a small scaled responsive web app designed to provide information on variety of workout types, create personalized diet/workout plans and have room for expansion into further resources.

For further information on the **WORKOUT** including product summary, product purpose, target audience etc. please read project proposal.

The system was built using HTML, CSS, JavaScript and Vue.js. Changing the content of the page to suit the screen size (using meta tag and media tag), positioning the navigation bar on top of the page (fixed) in desktop view and the left side in mobile view (can be toggled by clicking on hamburger menu icon), providing response based on user input, (using form input bindings and rendering methods), to name a few.

Design Challenges

Using Vue CDN instead of Vue CLI.

```
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
```

I have chosen Vue 2 CDN (Content Distribution Network) to achieve what I need to do in my project. The reason I chose CDN over CLI (Command Line Interface) was, that I was not familiar with Node.js-based build tools. (In official Vue website it was recommended for beginners as well). I really thought it would be sufficient to implement all coding concepts and ideas but, in the end, I realized that I should have used Vue CLI from the beginning. Especially when I got stuck with reusability & composition tooling and scaling up. Unfortunately, a large portion of the project was already completed, and to transfer the project into the CLI proved complex, challenging and time consuming, and as such, this was abandoned.

Explanation of Features/Functions of the System (Concepts, Components and Coding Practices)

I have created 4 different Vue Instances/Apps throughout this project and have linked them to 4 different HTML files.

1. #create-diet-plans
2. #create-workout-plans
3. #workouts
4. #search

Creating and Rendering Customized Diet/Workout Plans. (1st & 2nd Vue Instance/App)

The main challenge I faced was creating the content of the diet/workout plans based on the user's inputs, which outlined as the main functionality of the system. I wanted to make sure that users could get the best diets and exercises into their plans according to their body type and lifestyle. To accomplish that task, I have created two Vue instances. One for creating diet plans and one for creating workout plans. Both are similar in behavior, but the output is bit different. These two pages were the most significant in terms of using Vue form Input Bindings.

Using Form Input Bindings.

- I have used Vue form input bindings to collect user inputs, which is essential to create diet/workout plans.

Using List Rendering.

- To render select options such as **'Select your Lifestyle'**. (i.e., v-for="item in lifestyleOptions")
- To return empty fields as error messages. (i.e., v-for="error in errors")
- To render foods. (i.e., v-for="food in monday.breakfast")
- To render workouts. (i.e., v-for="workout in day1")

Using conditional Rendering.

- If the user selects **'Female'** as their **Gender** the systems will display **'Enter Your Hip Measurement'** option. If the user selects **'Male'** as their **Gender** the system will stop rendering it. This condition is being controlled via event handling. (i.e., v-show="showHip")
- The system will stop rendering the user input form once the plan is generated.
- Users won't be able to create the plan without filling in all the required fields. (i.e., v-if="!errors.length = 0")
- Users won't be able to give feedback again once they already clicked on the like/dislike button. They will be disabled right after. (i.e., :disabled="disabled == 1")

Using Declarative Rendering.

- The system will render the content of plans using declarative rendering. (i.e., {{food}}, {{workout}})

Using Computed properties.

- In order to create the diet plan, first I needed to calculate their BMI (Body Mass Index) and decide what their BMI is. So, I have used computed catching to calculate this factor based on their height and weight. Whenever the user enters/changes their height and weight the system captures it in real time and computes the value when the plan is being created.
- As additional/useful information to show in the diet plan, I decided to add details of user's BMR (Base Metabolic Rate), Body Fat Percentage & Daily Calorie Intake.

Using Methods.

- Using methods, I have created three main functions that are responsible for validating user input form, creating plan, and displaying plan.
- **'checkForm'** function will check if all the required fields are completed by the users. If not, it will create a list named **'errors'** with empty fields.
- **'showDietPlan/showWorkoutPlan'** function will toggle the visibility of input and output (user input form and generated plan). When the user submits the form, it will stop rendering the form and will start rendering the plan.
- **'createDietPlan'** function will change the values of objects according to **BMI** and **Lifestyle**.
- **'createWorkoutPlan'** function will change the values of objects according to user's **Aim** and **Body Type**.
- Once the plan has been created users will be able to like/dislike them using **'likeFeedback'** and **'dislikeFeedback'** methods. They only have one chance at it. (I have used a validation method that will disable buttons from clicking again and again.)

Using Event Handling.

- When the user selects '**Female**' as their **Gender** the system will alter the value of '**showHip**' into **true** which will render the '**Enter Your Hip Measurement Option**'. (i.e., `v-on:click="showHip = true"`)
- When the user selects '**Male**' as their **Gender** the system will alter the value of '**showHip**' into **false** which will stop rendering the '**Enter Your Hip Measurement Option**'. (i.e., `v-on:click="hideHip = false"`)
- When the user submits the form, the system will trigger the events that are related in creating plans. (i.e., `v-on:submit="createDietPlan()"`, `v-on:submit="createWorkoutPlan()"`)
- When user clicks on like/dislike button an appropriate feedback will display by triggering two functions. (`v-on:click="likeFeedback"`, `v-on:click="dislikeFeedback"`)
- Clicking on feedback buttons will trigger an event that will stop clicking them again.

Using Declarative Rendering.

- Using declarative rendering, the system will display what the user has entered in order to obtain the result. (Information such as their age, height , weight & etc. will include in their plan.)

Using Enter/Leave Transitions.

- If user attempted to create diet plan without filling out all the required fields, an error message will appear using a **bounce** transition, which will be caught user's attention immediately. (i.e., `<transition name="bounce">`)
- To make it a visually appealing website I have used transitions. The created plan will appear through a **slide-fade** transition. (i.e., `<transition name="slide-fade">`)

Using Filters.

- I wanted to display the user's name in uppercase when rendering the plan. I have defined a filter globally before creating the Vue instance, which fulfills that purpose. It will take user's input as a string and will return it in uppercase letters. (i.e., `{{userName|capitalize}}`)


Using Modifiers.

- I have used **.trim** modifier to get rid of unwanted spaces in user's name. (i.e., `v-model.trim="userName"`)
- I have used **.number** modifier to stop returning string values by HTML input elements. (i.e., `v-model.number="height"`)


Component Registration.

- I have created a globally registered component ('**like-plan**') to get feedback from users on the diet/workout plan created by the system.

PLEASE FILL IN THE FIELDS TO CREATE YOUR DIET PLAN




ENTER YOUR NAME (Optional)




SELECT YOUR GENDER

☐ Male ☐ Female




ENTER YOUR AGE

Years Old




ENTER YOUR HEIGHT

cm




ENTER YOUR WEIGHT

kg




ENTER YOUR NECK MEASUREMENT

cm



ENTER YOUR WAIST MEASUREMENT

cm



CHOOSE THE LIFESTYLE THAT BEST SUITS YOU

CREATE

Please input the following field(s):

- Age required.
- Select your gender.
- Enter your height.
- Enter your weight.
- Enter your neck measurement.
- Enter your waist measurement.
- Select your lifestyle.

Are you satisfied with the results?

Like



Dislike



file://

We appreciate your feedback to WORKOUT. What did you like? Let us know.

OK

Are you satisfied with the results?

Like



Dislike



PLEASE SELECT FROM THE OPTIONS BELOW

WHAT'S YOUR AIM?

I want to,

- ☐ Build Strength
- ☐ Build Muscles
- ☐ Build Endurance

WHAT'S YOUR BODY TYPE?



☐ Ectomorph

- Relative predominance of linearity and fragility.
- Greatest skin surface area relative to body mass causes greater sensory exposure.



☐ Mesomorph

- Relative predominance of muscle, bone, and connective tissue that dominates bodily economy.
- Heavy, hard, and rectangular in outline.



☐ Endomorph

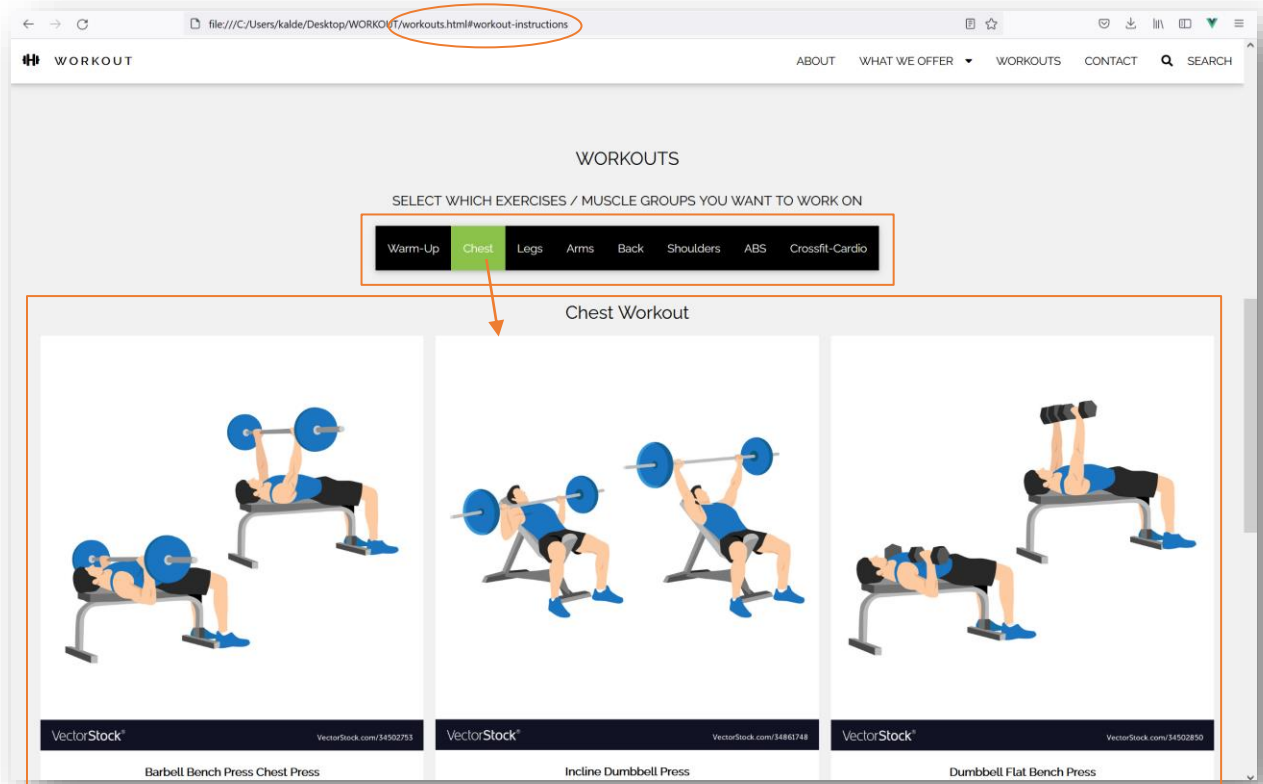
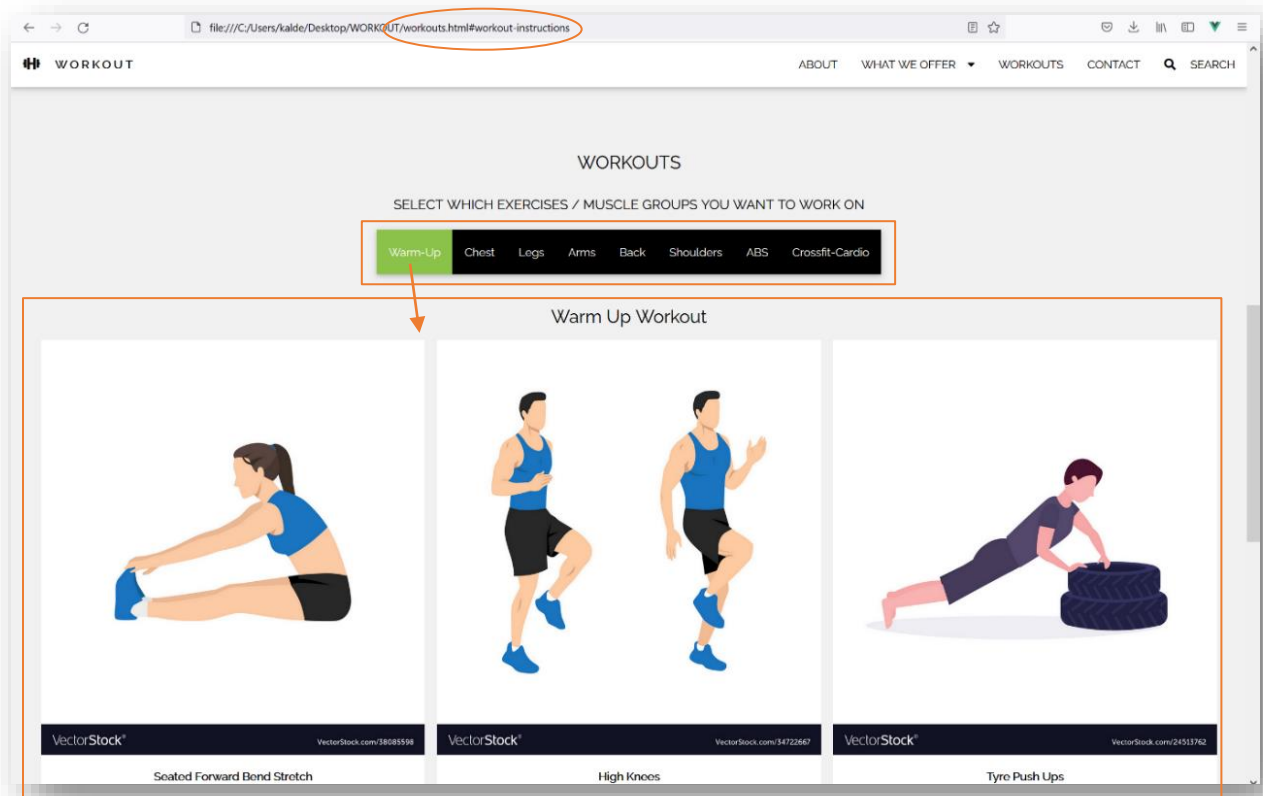
- Relative predominance of soft roundness throughout various regions of the body.
- Digestive viscera are more massive and relatively dominate bodily economy.

Please input the following field(s):

- Aim required.
- Select your bodytype.

FIND

View Workout Information. (3rd Vue Instance/App)



As an addition to project proposal, I thought it would be great to have a workout instruction section in my web application. Arrays are used to store workout information, so making changes or entering new details can be done effortlessly. I have rendered the plans by iterating through the arrays as required. I wanted to let users select the workout type they need and view instructions on how to do it without switching to a different page. Rendering information based on the user's selection within the same page was really challenging. I couldn't see a proper way of doing it using condition-based rendering or any other coding concepts I knew. In week 8 we were taught how to use single file component in Vue. And it turns out the best way to overcome this challenge was using Vue single file component.

Using Single File Components.

- I wanted to display each workout in each different section but not in a different page. So, I have chosen single file component. Each workout section will house different information. Users may view this information by switching between tabs within the same page.

```
<component v-bind:is="currentTabComponent" class="container"></component>
```

Component registration.

- I have created eight different globally registered components. Each contains different workout types. (i.e., 'tab-warm-up', 'tab-chest, tab-legs'...)

Using List Rendering.

- To render the tabs, I have used Vue list rendering. (i.e., v-for="tab in tabs")

Using Declarative Rendering.

- To render workout title and its description I have used declarative rendering.

Using Computed Properties.

- I have created a computed property that returns the current tab. This will update the DOM structure and will render the content in active component.

Using Class and Style Bindings.

- By using Vue class binding I have assigned a class type for active tab component. When the user selects a tab, it will trigger the styles and will make it look different from other inactive tabs.

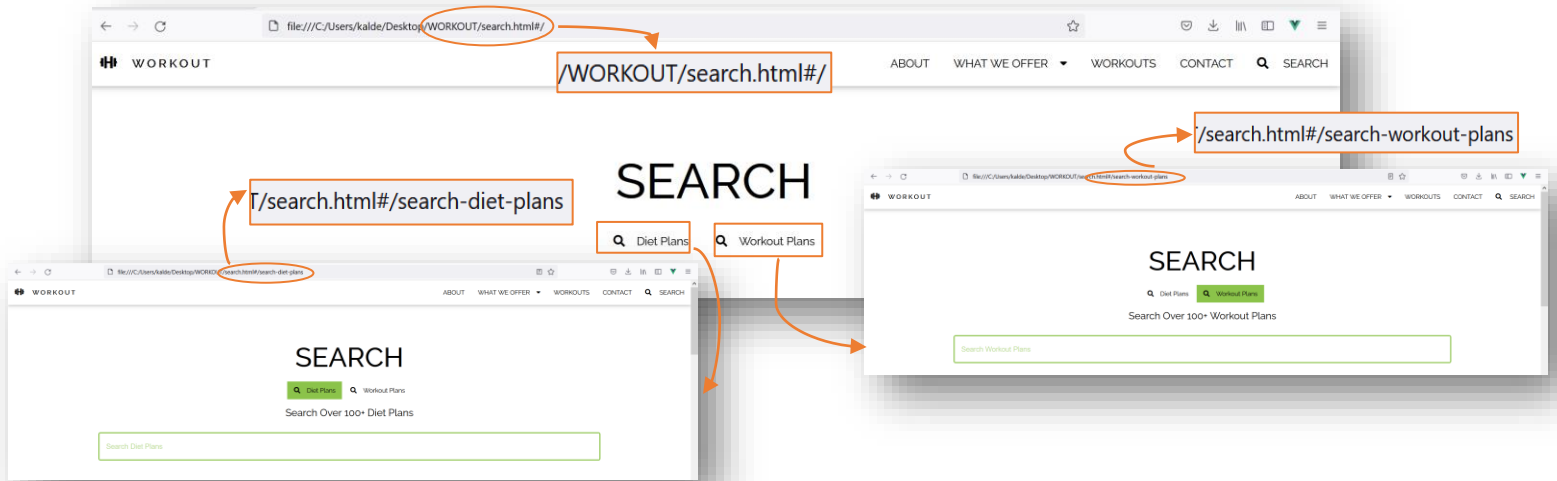
Using Event Handling.

- When the user clicks on a tab, an event will trigger that turns it into the current tab component. (i.e., v-on:click="currentTab = tab")

Using Enter/Leave Transitions.

- I added transitions (**slide-fade**) as users switch between tabs to make it a more visually appealing website. (i.e., <transition name="slide-fade">)

Search/Filter Diet or Workout Plans. (4th Vue Instance/App)



I wanted to implement a search/filter function that could easily find plans for users. Plans are stored using two arrays ('dietPlanList', 'workoutPlanList'), so it is easy to modify or add more details later. When the need arises, I have iterated through the arrays to render the plans. It was a bit difficult to implement a perfect search function that gives the most accurate search results. The purpose is to provide the user with the most relevant search results according to the keywords entered. I thought it was more practical to have two different search functions. One to search for diet plans and one to search for workout plans. In week 9 we were taught how to implement a similar concept to my idea using Vue-router. It makes possible to view two different components without navigating to a new page or refreshing the current page. It simply changes the path of the current page and renders new content.

Using Plugins - Vue Router (Routing).

```
<script src="https://unpkg.com/vue-router@3.5.2/dist/vue-router.js"></script>
```

- I have created **two paths** ('/search-diet-plans', '/search-workout-plans') with two different components ('searchDietPlans', 'searchWorkoutPlans'). One path is responsible for **retrieving information about available diet plans** and the other one is responsible for **retrieving information about available workout plans**.

```
<router-view></router-view>
```

Component registration.

- I have created **two separate components** for each search filter. One is **capable of searching diet plans** and the other one is **capable of searching workout plans**. Each component is linked to a different routing path.

Using Form Input Bindings.

- I have used form input bindings to get the user input as search keywords. (i.e., v-model.trim="searchTextDietPlan", v-model.trim="searchTextWorkoutPlan")

Using Modifiers.

- It is likely to enter unwanted spaces while typing keywords. If that happens, the search function will respond with zero results, even if the results match the keyword. So, I have used **.trim** modifiers to ignore unwanted spaces.

Using Computed Properties.

- The filter/search function converts the entered keywords into lower case letters and creates a list (**'filteredDietPlanList'**, **'filteredWorkoutPlanList'**) of plans if a plan includes a title that matches the entered keyword.
- I have created a function (**'currentRouteName'**) to retrieve the current pathname so that a particular style class can be activated.

Using List Rendering.

- Once the list (**'filteredDietPlanList'**) has been created the system will render the items (plans that match searched keyword) in the list using Vue list rendering. (i.e., `v-for="plan in filteredDietPlanList"`, `v-for="plan in filteredWorkoutPlanList"`)

Using Conditional Rendering.

- If there are no search results to show the system will render an error message. (i.e., `v-if="filteredDietPlanList.length == 0"`, `v-if="filteredWorkoutPlanList.length == 0"`)
- Content of the plan will be not rendered if it is empty in the data object.

Using Class and Style Bindings.

- To outline which path/search option the users are using (Search Diet Plans or Search Workout Plans) I have used Vue class bindings. Once the user has selected either option the button will change its style. (i.e., `v-bind:class="['btn', { active: currentRouteName === '/search-diet-plans' }]"`, `v-bind:class="['btn', { active: currentRouteName === '/search-workout-plans' }]"`)

Using Enter/Leave Transitions.

- Once the user has selected the search functionality they want to use, the system will render it through a fading transition. (and the same will happen when users switch between routing paths) (i.e., `<transition name="fade">`)

