Josef Legal Coding Challenge

- **STAND DOWN NOTES**
  - Day 1 Thursday 1/2/2024:
    - Taken a Crash Course on Vue.js
      - How to Create a Vue App
        - {{ }} is the interpolation syntax for JavaScript inside of HTML
        - when referring to CSS classes
          - camelCase; or
          - 'kebab-case-with-quotes'
      - Attribute Binding
        - data() {
          - return {
            - productName: "Socks",
            - brand: "Vue Mastery",
            - cart: 0
          - }
        - }
      - Conditional Rendering
        - if-else statements but in Vue
        - `<v-if>`
        - `<v-else-if>`
        - `<v-else>`
      - List Rendering
        - for loops in Vue
        - `<p v-for="variant in variants"></p>`
      - Event Handling
        - using v-on directive
        - onclick => `v-on:click='addToCart'`
        - or simply just, @click
        - methods: {
          - addToCart() {
            - this.cart += 1
          - }
        - }
      - Class & Style Binding
        - use the `v-bind:<style>` directive
          - eg. `v-bind:<style>`="{ backGroundColor: blue }" or

        - `v-bind:<style>`="{ 'background-color: blue }"
    - or just simply, :style
    - same goes with :disabled, :class, etc
    - :class bindings also allow for Ternary Operators
        - e.g. `:class=" [ isActive ? activeClass : ' ' ] "`
- Computed Properties
    - computed: {
        - title() {
            - return this.brand + ' ' + this.productName
        - }
    - }
    - computed properties for the most part behave exactly like methods with the difference being that computed property values are cached and are only evaluated when one of the associated values are changed
    - with methods, these are run every single time, not great for performance if when something is updated very frequently
- Components & Props
    - Components are the equivalent of _*partials.rb*_ files in Ruby on Rails
        - Components are called by invoking the tags of the same name in HTML
        - eg. ProductDisplay.js is invoked by `<product-display>` tag, just make sure the naming everywhere is correct i.e. app.component('product-display', {})
    - Props is a way to pass variables that live in the parent's scope down to the child components. Kind of like in ruby we would have done like the following:
        - = render _*some_partial*, local: @some_variable
        - the convention is that you pass the key and variable to the relevant tags (or partials)
            - e.g. `<product-details :details="details"></product-details>`
            - inside of the ProductDetails.js files, make sure we expect to receive `details` as well by defining them as props
                - props: {
                    - details: {
                        - type: Array,
                        - required: true
                    - }

            - }
    - Communicating Events
        - This is akin to listeners, in particular, when it involves an event from within a child component **AND** needing to update a value that lives outside of the child scope (eg. in a parent component, main.js, etc)
        - To do this, inside of the relevant method, we specify the following:
            - (upon clicking the button) @click="updateCart" (we call the `updateCart()`)
            - `this.$emit('add-to-cart', this.variants[this.selectedVariant].id)`
            - Inside of the `addToCart()`, we `emit` a signal to the listener `@add-to-cart` that lives inside the relevant `<component>` tag i.e. `<product-display @add-to-cart="addToCart">`
            - which in turn invokes the `addToCart()` in `main.js` and updates the `cart` value residing in `main.js`
    - Forms & V-Model
        - when using v-bind: it only allows for data to flow in one direction
            - take `v-bind:src` for example, it allows for the image to be passed from the `data()` object to the relevant `:src` tag and that's it
            - but if we want to allow for two-way data flow - such as in the use case of getting user input in a form - then we have to find another way
        - introducing v-models, it allows for the data that is gotten from the form to be saved in the variables inside of the `data()` object
  - Extra reading on Vue.js and its components (i.e. computed properties, v-directives, etc)
  - Forked, cloned repo
  - Go through installation steps to make sure things are working
  - Get app up and running
- Day 2 Friday 2/2/2024
  - Look through the codebase to get a general understanding
  - Understanding how the Vue.js layout gets rendered
    - Understanding how the local API work
        - Local node server & client running in the background (`npm run server` & `npm run client`)
        - Server fetches data from `db/dev.json` and returns the response to the client which gets rendered on the `vue` page
        - The response is saved as `this.files` at `src/views/Layout.vue:38` which in turn gets rendered at

`src/views/Layout.vue:7` {{files}}

    - Learn about Vue.js `props`
        - Used to pass `**data**` (String, Number, Array, etc) to a component
        - Learnt props in Day 1 Vue Mastery (https://www.youtube.com/watch?v=bzlFvd0b65c&t=2315s)
        - https://www.youtube.com/watch?v=koyrT34ffvU
        - https://www.youtube.com/watch?v=oQ6akOQKFtI
        - https://www.youtube.com/watch?v=te-lFidcrcM
    - Learn about Vue.js `slots`
        - Used to pass `**content**` (HTML, CSS, JS, video, image, etc) to a component
        - When using multiple slots, you have to name each slot (unnamed slots have the default name `default` but unsure what happens when you have more than 1 unnamed slot)
        - Good way to render default content on a page when the parent uses a component but doesn't provide the child component with certain values
        - Explains it very well (https://www.youtube.com/watch?v=orGcdmCRCc0)
        - This blog too (https://blog.logrocket.com/understanding-slots-vue-js/)
        - https://www.youtube.com/watch?v=emi436qg9mg
        - https://www.youtube.com/watch?v=8NvZeNgWaDU
    - Learn about basic Vue.js layout `<script> <template> <style> export default {}`
        - https://www.youtube.com/watch?v=ccsz9FRy-nk&list=PLC3y8-rFHvwgeQIfSDtEGVvvSEPDkL_1f
    - Create individual card components that will represent each file
- Day 3 Saturday 3/2/2024
    - Render each individual card
    - Styling the cards
    - Implement the filter by kitten function
    - Implement a button to trigger the filter switch
    - Temporarily finishing Front End Challenge, as what is left is to improve aesthetics, all required functionality has been implemented and working
    - Reading through Back End challenge (ponder ponder ponder...)
    - Attempting Back End Challenge
    - Refactor Back End Challenge
- Day 4 Sunday 4/2/2024 (Mostly a rest day to give the brain a bit of a distraction)
    - Continue trying to refactor the back end to be more efficient
    - Continue trying to refactor the back end to be more elegant

        - Mainly trying to figure out on my own to see if I can get the IDs to be sorted in ascending Question ID order
- Day 5 Monday 5/2/2024
    - More refactoring on back end component, successfully managed to sort rule ids in ascending order of question_id letter code
    - Attempting back end extra credit
        - Read up on what vuex module is
            - https://www.youtube.com/watch?v=nFh7-HfODYY
            - It is a state management library specifically for Vue applications
            - The problem it solves: having a central store where the data can be accessed by all of its components regardless of its scope. Otherwise, components will have to reach into its children / parent components via emits and and prop passing which is painful especially if there are many layers involved
            - Vuex is akin to having access to global variables for Vue apps
    - Moving data from Layout.vue to its own store
        - Create a file store
        - Instantiate the initial state (files = [])
        - Test that it works by just replacing instances of `this.files` with `this.$store.state["file"].files`
        - Set a getter method
        - Set a `dispatch action` (replacing the initial code in `Layout.vue:created()`), and then `commit mutation` to save the `response` as `state.files`
        - Set a `commit mutation` as outlined above
        - Doing all this should replace the need for using props. But doesn't that mean I don't get to showcase how props work?
- Day 6 Tuesday 6/2/2024
    - Refactor Card.vue and Rules.vue, cleaning up namespaces to improve readability
    - Moving data from Rules.vue to its own store
    - Get the damn checkGroup method to work again
    - Use getters to grab state values instead of accessing them directly for both Layout.vue and Rules.vue
    - Learn how to do unit testing with Jest for Vue components
        - https://www.youtube.com/watch?v=vHOMSq3-KMY
    - Mucking around with mocking Vuex store in Jest file and trying to get it to behave
- Day 7 Wednesday 7/2/2024
    - Got the mock store and mock dispatch functions to work but looks kind of weird (`store._modulesNamespaceMap["rule/"]`)

    - Spent a big chunk of time debugging and getting the test to run without problems
        - Setting mock getters
    - Attempted to move `results` hash to rule Vuex store
        - As a result, broke the unit tests and now trying to figure out how to wire everything back up again
    - Fixed tests after moving `results` hash to rule store
    - Writing more unit tests
    - Start to write README, probably unable to attempt QA section, will try to take a look tomorrow morning
- Day 8 Thursday 8/2/2024
    - Implement slot props for Front End component
    - API testing on `entity` object using Postman
    - Update README
    - Create PR and invite Marc