[TOC]

Day 1

Pre-requisite

Need to install Nodejs https://nodejs.org/en/download/

Need to install Visual Studio Code https://code.visualstudio.com/download#

Install Vue Cli

```
npm install -g @vue/cli
```

Create a new Project:

```
vue create <project-name>
```

```
vue create my-app

Select following optional

1. Manually select feature

2. ? **Please pick a preset:** Manually select features

  ? **Check the features needed for your project:** (Press **<space>** to select, **<a>** to tog

  ❯◉ Choose Vue version

  ? **Check the features needed for your project:**

  ◉ Choose Vue version

  ◉ Babel
```

```
  ◉ Progressive Web App (PWA) Support
  ◉ Router
  ❯◉ Vuex
  ◯ CSS Pre-processors
  ◯ Linter / Formatter
  ◯ Unit Testing
  ◯ E2E Testing
3. Use history mode for router? (Requires proper server setup for index fallback in production)
4. Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)> Select In packa
5. Preset yes
```

Installing important extensions in VS Code

- Vetur,
- debugger for Chrome,
- vue vscode snippet ,
- vue-format

Introduction to project filestructure

File / Folder	Description
public	
package.json	
src	
src/assets	
src/components	

src/views	
src/router	
src/App.vue	
src/main.js	

Introduction to Vuejs file

- Vuejs files get created with `.vue` extenstions
- Every file is mainly divided into 3 parts

Section	Description
template	All your html related code goes in this section
script	All your javascript related code goes in this section
style	All your styling related code goes here

- e.g

```js
<template>
  <div>
  </div>
</template>
<script>
  export  default {
  }
</script>
<style>
</style>
```

Important methods & blocks in script tag

Name	Type	Description
data	function	In this section you define your all variables which needs to be used in template/ html code

methods	block	In this section you define your all methods
computed	block	
mounted	function	
created	function	
destroyed	function	
components	block	

Welcome to Vuejs

```js
<template>
    <div  id="app">
        <h1>Welcome to Vuejs</h1>
    </div>
</template>
<script>
    export  default {
    }
</script>
<style>
</style>
```

Binding in Vuejs

One-way binding

One way binding is concept of declaring variable in javascript and **reading** those variables in `<template>` part i.e. in HTML code. Look at the following code.

```js
<template>
    <div  id="app">
        <h1>Welcome to Vuejs</h1>
        <h1>Hello {{name}}</h1>
    </div>
</template>
```

```
        return {
            name:  "Kapil"
        }
    },
}
</script>
```

In above code we have defined data method which returs all the variables. We defined a variable called `name` & assigned it a value.

Now to access this variable in html code we have to use in syntax: `` So in our code `` will be replaced by it's value.

Two-way binding

The concept of **reading** as well as **modifying** variables from html code. Look at the following code

```js
<template>
  <div id="app">
    <h1>Welcome to Vuejs</h1>
    <input type="text" v-model="name" placeholder="Enter your name">
    <h1>Hello {{name}}</h1>
  </div>
</template>
<script>
export default {
  data() {
    return {
      name: "Kapil"
    }
  },
}
</script>
```

In above code we we have inserted input tag to accept name from user. In this we have used vuejs attribute `v-model`, this is special attribute use to bind variable with input tag.

Assignment 1

Designe a small app which accepts user input dollar value from user & convert it into INR & vice a versa

Day 2

Defining Methods

You can define different methods in vuejs file using methods block. Following is the example of defining method.

```js
<template>
  <div>
    <button @click="increment()" @mouseover="increment()">Click Me</button>
    <h1>You have pressed button {{counter}} times</h1>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        counter: 0
      }
    },
    methods: {
      increment() {
        this.counter++
      },
      decrement(){}
    },
  }
</script>

<style lang="scss" scoped>

</style>
```

This example demonstrate how we have defined method named as *sayHello()* Also this method demonstrate how you can modify variable defined in data function. To access any variable in method you have to use `this` keyword

Assigment 2

Once user input value in any of the input box other input box value should be updated automatically.

e.g If i enter value 10 in DOLLAR input box INR input box should show 700.

Day 3

Defining Computed Property

Computed is block in which you define a method which can be accessed as a property in html code. Vue does provide a more generic way to observe and react to data changes on a Vue instance Following example shows how you can define a Computed Property

```js
<template>
<div class="home">
    <h1>Welcome to Home page</h1>
    <input type="text" v-model="fname" placeholder="Enter firstname">
    <input type="text" v-model="lname" placeholder="Enter lastname">
    <h1>Fullname: {{fullName}}</h1>
</div>
</template>

<script>
export default {
    data() {
        return {
            fname: "Kapil",
            lname: "Mundada"
        }
    },
    computed:{
      fullName(){
        return this.fname+" "+this.lname
      }
    }
}
</script>
```

In above example we have defined `fullName` as computed property, which concatinate variable value fname & lname.

Computed Property vs Method by example Following example demostrate what is difference between computer property & watcher

```js
<template>
<div class="home">
    <h1>Welcome to Home page</h1>
    <h1>Computed Property: {{random}}</h1>
    <h1>Method Call: {{mrandom()}}</h1>
    <h1>Method Call: {{mrandom()}}</h1>
    <h1>Computed Property: {{random}}</h1>
    <h1>Computed Property: {{random}}</h1>
    <h1>Method Call: {{mrandom()}}</h1>
</div>
</template>

<script>
export default {
    computed:{
      random(){
        return Math.random()
      }
    },methods: {
      mrandom() {
        return Math.random()
      }
    },
}
</script>
```

Output

```
Welcome to Home page
Computed Property: 0.49752439060313436
Method Call: 0.4572598657389706
Method Call: 0.6608244941907615
Computed Property: 0.49752439060313436
Computed Property: 0.49752439060313436
Method Call: 0.5214314876827064
```

You can see *computed property* always returns **same value** but *method returns* always **new value**.

- Computed property is by default a getter property
- You can define a computed proeperty as getter & setter as well. Following example demonstrate how you can define getter & setter in computed property.

```html
<template>
  <div>
    <h1>Computed Property Setter Example</h1>
    <input type="text" placeholder="Firstname" v-model="fname">
    <input type="text" placeholder="Lastname" v-model="lname">
    {{ fullName }}
    <br>
    <input type="text" placeholder="FullName" v-model="fullName">
  </div>
</template>

<script>
  export default {
    data() {
      return {
        fname: "",
        lname: ""
      }
    },
    computed: {
      /* fullName() {
        return this.fname+' '+this.lname
      }, */
      fullName:{
        get(){
          return this.fname+' '+this.lname
        },
        set(newValue){
          let name = newValue.split(' ')
          this.fname = name[0]
          this.lname = name[1]
        }
      }
    },
  }
</script>

<style lang="scss" scoped>
```

Watcher

While computed properties are more appropriate in most cases, there are times when a custom watcher is necessary. That's why Vue provides a more generic way to react to data changes through the `watch` option. This is most useful when you want to perform asynchronous or expensive operations in response to changing data.

Following example demonstrate how you can define watcher for different properties.

```js
<template>
  <div>
    <input type="text" placeholder="First name" v-model="fname">
    <input type="text" placeholder="First name" v-model="lname">
    <h1>Welcome {{fullName}}</h1>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        fname: '',
        lname: '',
        fullName: ''
      }
    },
    watch: {
      fname(newValue, oldValue) {
        this.fullName = newValue+' '+this.lname
      },
      lname(newValue, oldValue) {
        this.fullName = this.fname+' '+newValue
      },
    },
  }
</script>

<style lang="scss" scoped>

</style>
```

Assignment 3

Design a User Details form with following information.

- firstName, middleName, lastName (3 text boxes)
- dob (date)
- gender (redio button)
- address (text-area)
- Education (drop down): Under Gradualte / Graduate / Post Graduate

When user enters his name his fullname should be shown runtime in LastName+FirstName+MiddleName format

When user enters his birthdate, his age should be calculated and show on the screen.

He shold be submit form if he is atleast graduate

On form submission, form should not be visible only user information shold be visible and a link to go to add new User button

On click of add new User button form should be visible with blank values.

Day 4

Defining Lifecycle hooks

To read more about lifecycle methods please refer this **link** .

Following code demonstrate 2 lifecycle methods

```js
created() {
    console.log("Hi from created");
},
mounted() {
    console.log("Hi from mounted");
},
destroyed() {
```

There are **8 lifecyle methods** executes in following order

1. beforeCreate
2. created
3. beforeMount
4. mounted
5. beforeUpdate
6. updated
7. beforeDestroy
8. destroyed

Calling method in lifecycle hooks

```js
created() {
    this.init()
},
methods:{
    init(){
        console.log("This is init method")
    }
}
```

List Rendering

We can use the `v-for` directive to render a list of items based on an array

```html
<template>
    <div>
      <ul>
        <li v-for="(str,index) in frameworks" :key="index">{{index}} - {{str}}</li>
      </ul>
      <h1>Object Array</h1>
      <ul>
        <li v-for="item in items" :key="item.id">
          {{ item.name }} Rs.{{ item.price }}/-
        </li>
      </ul>
    </div>
```

```
<script>
  export default {
    data() {
      return {
        frameworks : ['Angular', 'React', 'Vue', 'Expressjs', 'Vue'],
        items: [
                { id: 1, name: "Orange", price: 24 },
                { id: 2, name: "Banana", price: 12 },
                { id: 3, name: "Apple", price: 15 },
            ]
      }
    },
  }
</script>

<style lang="scss" scoped>

</style>
```

- In above example for loop is represented by `<li v-for="item in items" :key="item.id">` & **:key** indicates unique key by which every item is distinguished.
- You can also use `of` as the delimiter instead of `in`, so that it is closer to JavaScript's syntax for iterators.
- `v-for` also supports an optional second argument for the index of the current item.

```
<li  v-for="(item, index) in items" :key="item.id">
  {{ index }} - {{ item.name }}
</li>
```
js

`v-for` with an Object

You can also use `v-for` to iterate through the properties of an object.

```
<template>
<div class="home">
  <h1>Welcome to Home page</h1>
  <ul id="v-for-object" class="demo">
      <li  v-for="value  in  object"  :key="value"> {{ value }} </li>
```
html

```
#### Output
- How to do lists in Vue
- Jane Doe
- 2016-04-10
```

And another for the index:

```html
<div v-for="(value, name, index) in object" :key="index">
  {{ index }}.{{ name }}: {{ value }}
</div>
```

Output

0. title: How to do lists in Vue
1. author: Jane Doe
2. publishedAt: 2016-04-10

v-for on a `<template>`

`<template>` tag with `v-for` to render a block of multiple elements. For example:

```js
<ul>
    <template v-for="item in items">
        <li :key="item.id">{{ item.name }}</li>
    </template>
</ul>
```

Assignment 4

Book Information Page

List Books, Each book will have (id, book name, author name, number of pages, issued/available)

List Books table will show id, name, author name, page count, status, operations [edit, delete]

Create form will have Add Book button & update book form will have Update Book button. Update Book will show status drop-down/ radio button, this controle will not be visible in Create Book.

Add Book : will create new entry of the book in the table

Update Book: Will update the details of the book if any changed

In table operation column if you click on edit, then that book information should be shown in update book form automatically.

In table operation column if you click on delete, then book should get deteted from the table.

When you load page first time it should have atleast 3 books in the list.

Day 5

Components

Components are reusable Vue instances with a name Following is and example of simple component number-counter Create new file in `components/ButtonCounter.vue` & paste following code

```js
<template>
    <div>
        <button class="orange" @click="counter++">You clicked {{counter}}</button>
    </div>
</template>

<script>
    export default {
        data() {
            return {
                counter: 0
            }
        },
    }
</script>

<style scoped>
.orange{
    background-color: orange !important;
```

Above code just rendering a button with counter 0. & when you click button it will increment a count.

`<style scoped>` here this scoped indicates styling will be applicable to this component/file only.

Now you can use this file in any `.vue` file as a component in 2 ways.

- Local Component
- Global Component

Local Component When you want to use component only in your file you can import in your loca file with following 2 steps

1. Import component file with some name
2. Declare imported component in `components` block
3. Use in kebab case

```js
<template>
<div class="home">
  <h1>Welcome to Home page</h1>
  <button-counter></button-counter>
  <ButtonCounter />
</div>
</template>
<script>
import ButtonCounter from "@/components/ButtonCounter"
export default {
  components: {
      ButtonCounter
  }
}
</script>
```

Global Component When you want to load component globally so it's accessible to entire application you can define in following way.

1. Go to main.js
2. import your component with some name
3. define Vue.component(`<component-name>` , `<component-imported-in-step2>`)

```js
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import ButtonCounter from '@/components/ButtonCounter'

Vue.config.productionTip = false
Vue.component('button-counter', ButtonCounter)
new Vue({
  router,
  render: h => h(App)
}).$mount('#app')
```

Accepting Inputs from user Lets understand by example. Consider following code. Here is list of items displayed on web page.

```js
<template>
<div>
    <template v-for="(item, index) in items">
        <div :key="index">
            <div style="float:right">Rs. {{item.price}}/-</div>
            <h1>{{item.name}}</h1>
            <h5>{{item.id}}</h5>
            <hr>
        </div>
    </template>
</div>
</template>

<script>
export default {
    data() {
        return {
            items: [{ id: 1, name: "Apple", price: 12 },
                { id: 2, name: "Banana", price: 13 },
                { id: 3, name: "Mango", price: 15 },
                { id: 4, name: "Orange", price: 10 },
                { id: 5, name: "Watermelon", price: 16 },
                { id: 6, name: "Grapes", price: 18 },
                { id: 7, name: "Sweet Lime", price: 19 },
            ]
        }
    },
```

```
</style>
```

Here we can define a component `/components/ItemCard.vue` which will take item as an input and render it.

```js
<template>
    <div>
        <div style="float:right">Rs. {{item.price}}/-</div>
        <h1>{{item.name}}</h1>
        <h5>{{item.id}}</h5>
        <hr>
    </div>
</template>

<script>
    export default {
        props:['item']
    }
</script>
<style scoped>
h1{
    font-size: 16px;
}
h5{
    font-size: 12px;
}
</style>
```

In above code `props` is an array of inputs component can take. Here we have define only ne input `item`.

- value defined in props array is accessible in the component as variable. So you can access it as `this.item` or
- It's always adviced that you should not manipulate props valus directly.

Now we can use this component as as

```js
<template  v-for="(item, index) in  items">
    <item-card  :item="item"  :key="index"></item-card>
```

Here we are passing every item to ItemCard component to render.

Day 6

Event Handling

Lets consider above example. We need to add new items in an array. So let's define a new component `/components/EditItem.vue`

```html
<template>
    <div>
        <input type="number" v-model="id">
        <input type="text" v-model="name">
        <input type="number" v-model="price">
        <button @click="saveItem()">Save</button>
        <button>Cancel</button>
    </div>
</template>

<script>
    export default {
        data() {
            return {
                id: 0,
                name: "",
                price: 0
            }
        },
        methods: {
            saveItem() {
                console.log("Sending Event")
                this.$emit('save', {"id":this.id, "name":this.name, "price":this.price})
            }
        },
    }
</script>
<style scoped>
</style>
```

In above code,

- Every input control is binded with 1 variable.
- On click of Save button we are calling method.
- In saveItem we call `this.$emit` method. It has following syntax

```js
$emit(eventName) // Using this syntax you can fire only event without data
$emit(eventName, data) // Using this syntax you can fire event with some data
```

- `$emit` actially send this event to listning component. Now we can add this component in ProductList page as below and listen to event using `@eventName`

```html
<div align="center" style="background-color:orange; padding:4%">
  <edit-item @save="saveItem($event)"></edit-item>
</div>
```

Here we are listing event `save` using @save and passing the received data to one of the method which will add this item into products list.

```js
saveItem($event){
  console.log("Recieved Event as: "+JSON.stringify($event))
  this.items.push($event)
}
```

Using above technique we can do the event handling.

Issue:

This option will be visible when status is `AVAILABLE`. When user clicks on Issue books status should change from `AVAILABLE` to `ISSUED`

Return:

This option will be visible when status is `ISSUED`. When user clicks on Issue books status should change from `ISSUED` to `AVAILABLE`

Component with v-model binding

due to `v-model`

So we need to define a component where we can bind varibale from parent to child component. Lets define a component `/components/MyTextBox.vue`

```html
<template>
    <div>
        <input type="text" v-model="textValue" @change="validate()">
        <label class="hint-text" v-if="hint">{{hint}}</label>
    </div>
</template>

<script>
    export default {
        data() {
            return {
                textValue: null,
                hint:""
            }
        },
        methods:{
            validate(){
                if(this.textValue.length <= 0){
                    this.hint="Value cannot be empty"
                }else{
                    this.$emit('input', this.textValue)
                }
            }
        }
    }
</script>

<style scoped>
.hint-text{
    color: red;
    font-size: 10px;
}
</style>
```

In above code we have defined a special component which takes input from user, performs validation on user-input & if input is valid sends to parent otherwise shows message in `red` as `Value cannot be empty`

So when i am using this component as follows

```html
<my-text-box v-model="name"></my-text-box>
```

input value is get automatically bind with variable `name`

Day 7

Routing

Vuejs used `vue-router` module for routing

https://router.vuejs.org/

Day 8

Dynamic Routes

Let's consider a scenario where we have StudentsList displayed on the page. Also it has 2 operations

- Add new Student
- Edit existing Student

When user click on a any of the operation you have to navigate to StudentDetails page.

- If user clicks on Add Student operation StudentDetails page should show blank fields to add new student.
- If user clicks on Edit Student operation Existing student details should be populated in text box.
- After Save from StudentDetails operation page the updated data i.e. either new student details or modified student details should be reflected on the StudentsList page.

In this example we will read the list of students from external .json file.

- Create new folder `date` in src --> `/src/data`
- Create new file student.json under `/src/data/`

```js
[
    {"id":"1", "name": "Student 1", "grade":"A"},
    {"id":"2", "name": "Student 2", "grade":"B"},
    {"id":"3", "name": "Student 3", "grade":"A"},
    {"id":"4", "name": "Student 4", "grade":"C"},
    {"id":"5", "name": "Student 5", "grade":"A"}
]
```

StudentCard.vue

```html
<template>
    <div>
        <div style="float:right; font-size:24px">{{item.grade}}</div>
        <h1>{{item.name}}</h1>
        <h5>{{item.id}}</h5>
        <div align="right">
            <button @click="$router.push('/editStudent/'+item.id)">Edit</button>
            <button @click="$emit('delete')">Delete</button>
        </div>
        <hr>
    </div>
</template>

<script>
    export default {
        props:['item']
    }
</script>

<style scoped>
h1{
    font-size: 16px;
}
h5{
    font-size: 12px;
}
</style>
```

Consider following as StudentsList page.

```
        <span style="float:right"><router-link to="/editStudent/0">Add</router-link></span>
        <h1>Student List</h1>
        <div align="center" v-if="msg" style="background-color:lightgreen; padding:2%">{{msg}}</
        <template v-for="student in students">
            <student-card :item="student" :key="student.id"></student-card>
        </template>
    </div>
</template>

<script>
import StudentCard from "@/components/StudentCard.vue"
import studentList from "@/data/students.json"
export default {
        components: {
            StudentCard,
        },
        data() {
            return {
                students: studentList,
                msg: ""
            }
        },
        mounted(){
            this.initComponent()
        }
    }
</script>
```

Consider following EditStudent.vue

```
<template>
<div align="center" style="padding:10%">
    <input type="number" v-model="student.id" placeholder="Enter id"><br>
    <input type="text" v-model="student.name" placeholder="Enter name"><br>
    <input type="text" v-model="student.grade" placeholder="Enter grade"><br>
    <br>
    <button @click="saveStudent()">Save</button>
    <button @click="$router.go(-1)">Cancel</button>
</div>
</template>

<script>
```

```
        return {
            students: studentsList,
            student: { id: 0, name: "", grade: "C" },
            id: 0
        }
    },
    mounted(){
        this.id = this.$route.params.id
        if(this.id>0){
            this.student = this.students.find(rec => rec.id == this.id)
        }
    },
    methods: {
        saveStudent() {
            //this.$router.push({name:"Students", params:{entity:this.student, add:this.id > 0 ?
            this.$router.push({path:"/students", query:{entity:this.student, add:this.id > 0 ? f
        }
    },
}
</script>
```

In above example `saveStudent` method you can see how you can pass the data between routes using two ways

1. Using query
2. Using params

- When you use `path` in your route it ignores params attribute
- so when you use `path` you should use with query to send data.
- You can use `named` paths in the router.push method as `name` (Make sure u have provided a name to path in router configurations)

Navigation Gaurds

As the name suggests, the navigation guards provided by `vue-router` are primarily used to guard navigations either by redirecting it or canceling it.

```js
router.beforeEach((to, from, next) => {
  console.log("To: "+to.name)
```

```
  }else{
    next()
  }
})
```

Global before guards are called in creation order, whenever a navigation is triggered. Guards may be resolved asynchronously, and the navigation is considered **pending** before all hooks have been resolved.

Every guard function receives three arguments:

- `to: Route` : the target Route Object being navigated to.
- `from: Route` : the current route being navigated away from.
- `next: Function` : this function must be called to **resolve** the hook. The action depends on the arguments provided to `next` :
  - `next()` : move on to the next hook in the pipeline. If no hooks are left, the navigation is **confirmed**.
  - `next(false)` : abort the current navigation. If the browser URL was changed (either manually by the user or via back button), it will be reset to that of the `from` route.
  - `next('/')` or `next({ path: '/' })` : redirect to a different location. The current navigation will be aborted and a new one will be started. You can pass any location object to `next` , which allows you to specify options like `replace: true` , `name: 'home'` and any option used in `router-link` 's `to` prop or `router.push`
  - `next(error)` : (2.4.0+) if the argument passed to `next` is an instance of `Error` , the navigation will be aborted and the error will be passed to callbacks registered via `router.onError()` .

Make sure that the `next` function is called exactly once in any given pass through the navigation guard. It can appear more than once, but only if the logical paths have no overlap, otherwise the hook will never be resolved or produce errors.

Day 9

Mixins

Mixins are a flexible way to distribute reusable functionalities for Vue components. A mixin object can contain any component options. When a component uses a mixin, all options in the mixin will be “mixed” into the component’s own options.

Local Mixins

Crate following file in project for common functionality

`/src/mixins/common-functions.vue`

```html
<script>
export default {
    methods: {
        printGreeting(msg) {
            alert("Message from mixins\n"+msg)
        }
    },
}
</script>
```

Accessing this common functions

```js
<template>
  <div>
    <button @click="printGreeting('Hello')">Click Me</button>
  </div>
</template>

<script>
import commonMixins from '@/mixins/common-functions.vue'
export default {
  mixins: [commonMixins],
}
</script>

<style lang="scss" scoped></style>
```

When a mixin and the component itself contain overlapping options, they will be “merged” using appropriate strategies. Normally the component’s options will take priority when there are conflicting keys in these objects:

Globlal mixins

Use with caution! Once you apply a mixin globally, it will affect **every** Vue instance created afterwards. When used properly, this can be used to inject processing logic for custom options:

Add following code in `main.js`

```js
import globalMixins from '@/mixins/common-functions.vue'

/**
 * Way to define global mixins
 */
 Vue.mixin(globalMixins)
```

> Use global mixins sparsely and carefully, because it affects every single Vue instance created, including third party components. In most cases, you should only use it for custom option handling like demonstrated in the example above. It's also a good idea to ship them as **Plugins** to avoid duplicate application.

Day 10

Slots