Chapter 7: Vue.js

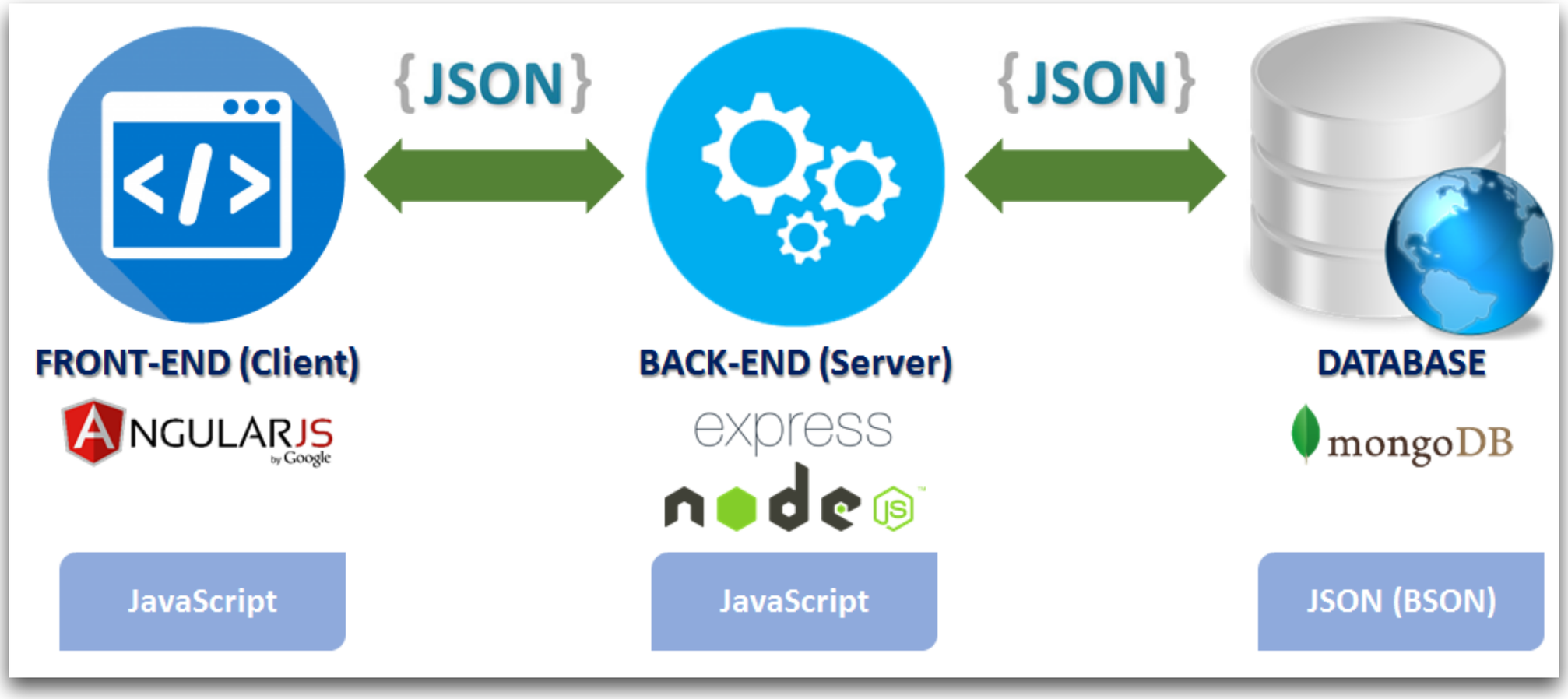
COMP7270 Web and Mobile Programming
& COMP7980 Dynamic Web and Mobile Programming

Dr Shichao MA, 5th March 2025

About me

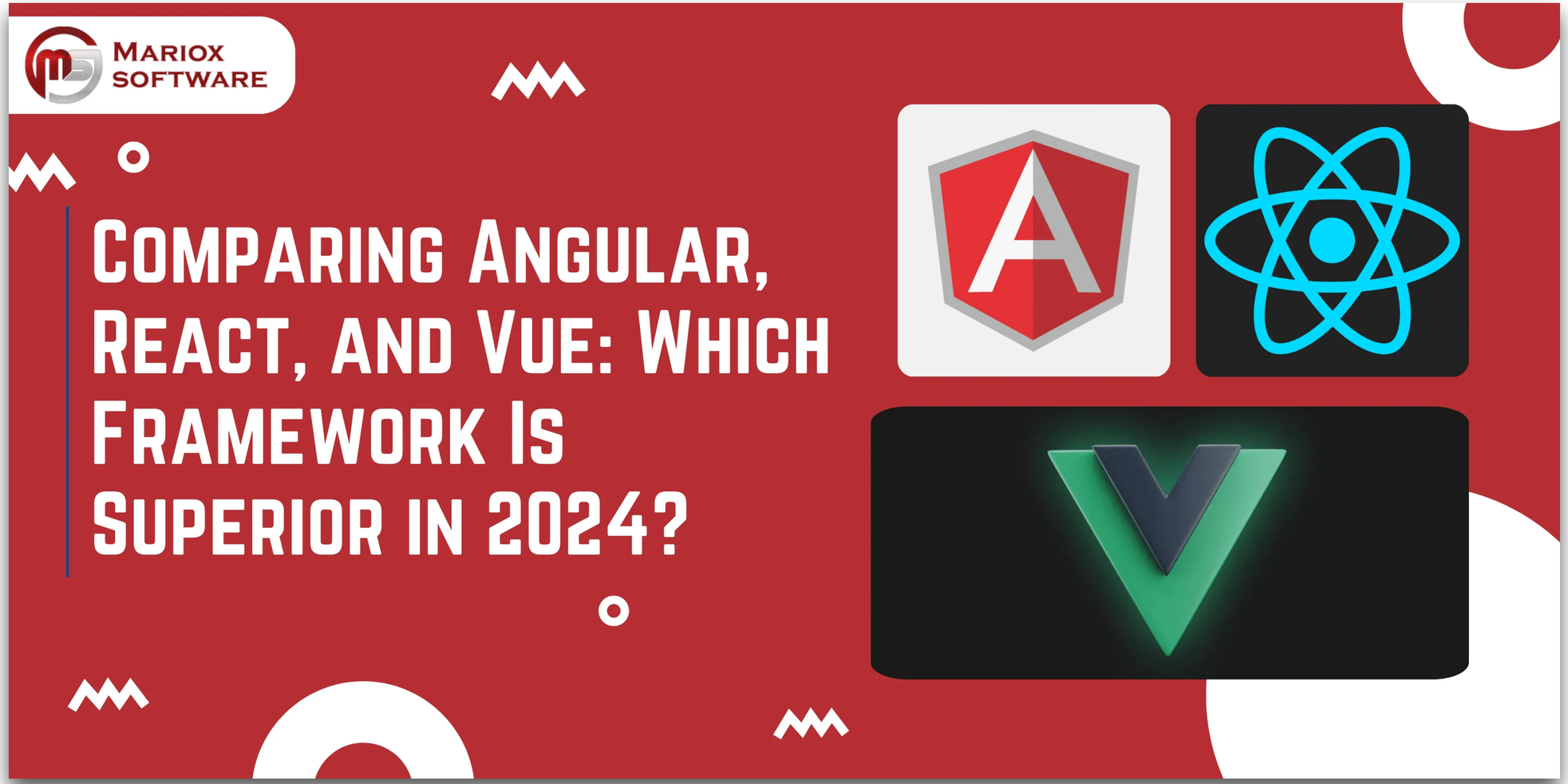
- Dr. Shichao MA
- Office: DLB 804
- Email: shichaoma@comp.hkbu.edu.hk

MEAN Stack Revisited



Or other JS frontend frameworks like React and Vue.js

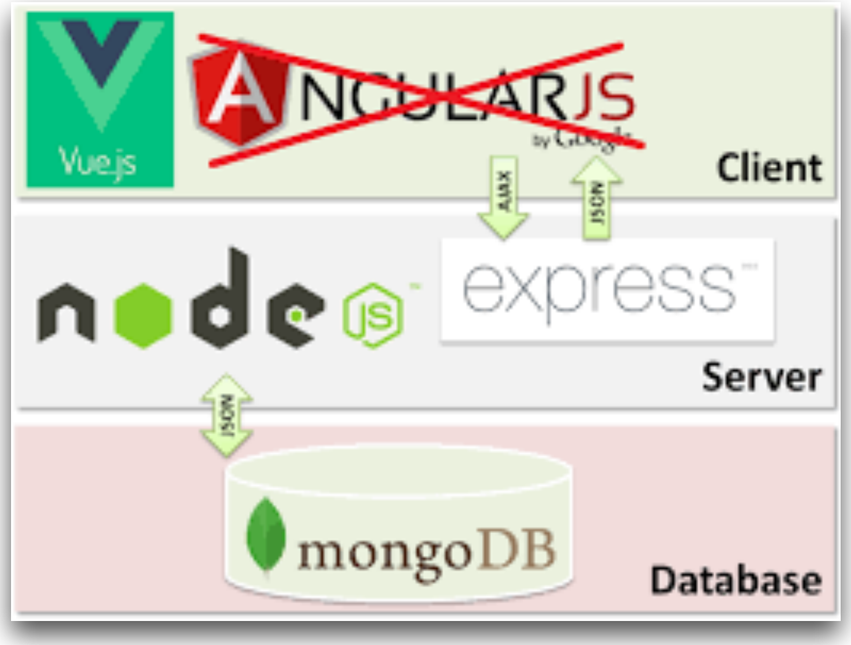
Frontend Frameworks



https://www.marioxsoftware.com/blog/comparing-angular-react-and-vue-which-framework-is-superior-in-2024/

AngularJS and Vue.js

- **AngularJS is migrating to Angular**, which is based on Typescript.
- On the other hand, Vue.js is very similar to AngularJS.
  - Vue.js is inspired by Angular.js, attempted to offer a significant improvement.
- As this course is centered around JavaScript, we will cover **Vue.js**.



Vue.js Version 3

- Vue3 provides two APIs:
  - Go with Options API if you are not using build tools, or plan to use Vue primarily in low-complexity scenarios, e.g. progressive enhancement.
  - Go with **Composition API** + Single-File Components if you plan to build full applications with Vue.

Vue Application Instance


```
<form class="container my-5" @submit.prevent="submitBooking">
```

- A Vue application instance is created and mounted to a DOM element.
  - It can then manage the elements under this DOM node.
- Instance properties are defined under **data**.
- Instance methods are defined under **methods**.

Option API

```
createApp({
    data() {
        return {
            message: 'Hello Vue!'
        }
    },
    methods: {
        submitBooking: async function (page) {}
    }
}).mount('#app')
```

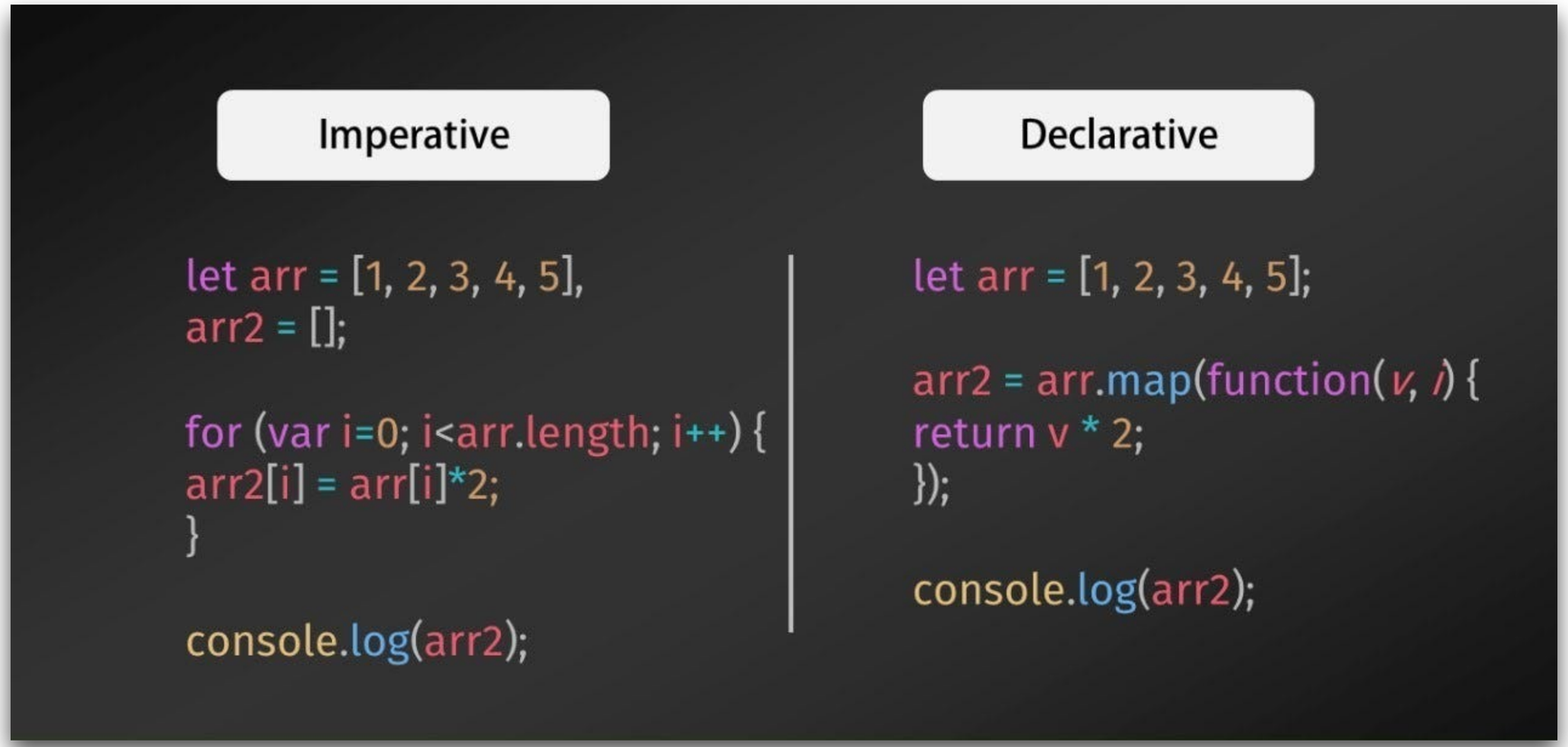
Declarative Rendering

- At the core of Vue.js is a system that enables us to declaratively render data to the DOM using straightforward template syntax
- The most basic form of **data binding** is data interpolation and rendering using the “Mustache” syntax (double curly braces):

```
<div v-if="route.name == 'view-booking'">
    <h1>{{ booking.email }}</h1>
    <p>Number of Tickets: {{ booking.numTickets }}</p>
    <p>Payment Method: {{ booking.payment }}</p>
    <p>Favourite Team: {{ booking.team }}</p>
    <p>Favourite Hero: {{ booking.superhero }}</p>
    <p>Terms and Conditions: {{ booking.terms }}</p>
</div>
```

```
booking: {
    email: '',
    numTickets: 1,
    payment: 'Credit Card',
    team: '',
    superhero: '',
    terms: false
}
```

Imperative VS Declarative



How?

What?

Directives

- At a high level, **directives are markers on a DOM element** that tell Vue's HTML compiler to attach a specified behavior to that DOM element, or even to transform the DOM element and its children.

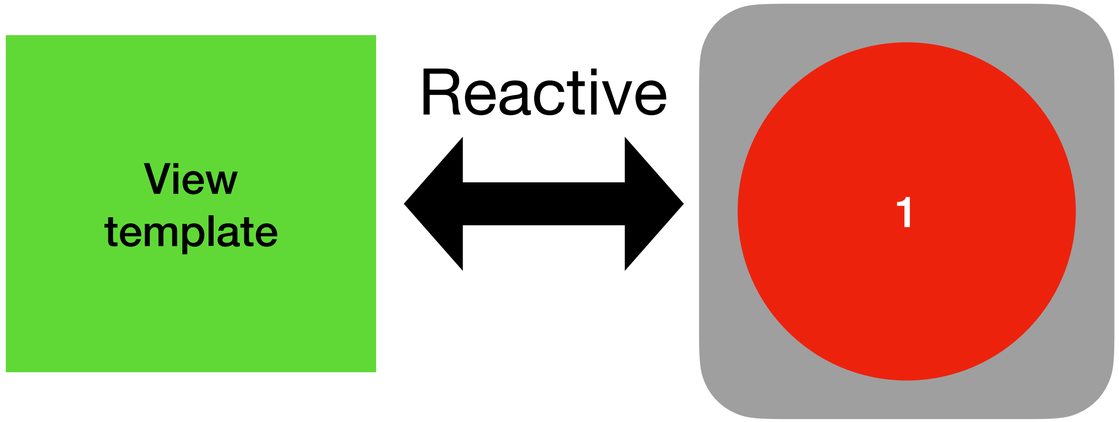
- Vue.js comes with a set of these directives built-in, like v-for, v-on, and, v-model etc…

Composition API

Data Properties

- The ref() takes an inner value and returns a reactive and mutable ref object, which has a single property **.value** that points to the inner value.

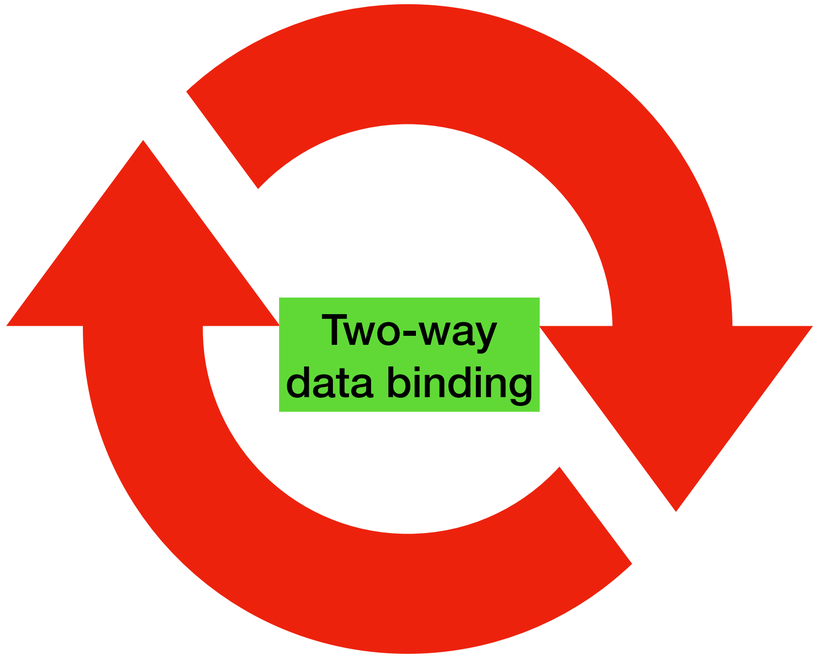
```
const booking = ref({
    email: '',
    numTickets: 1,
    payment: 'Credit Card',
    team: '',
    superhero: '',
    terms: false
})
```



Form Input Bindings with v-model

- We can use the v-model directive to create **two-way data bindings** on form input and select elements, etc.

```
<input type="email" class="form-control" v-model="booking.email" required>
```




```
const booking = ref({
    email: '',
    numTickets: 1,
    payment: 'Credit Card',
    team: '',
    superhero: '',
    terms: false
})
```

Event Handling with v-on

- We can use the v-on directive to listen to DOM events and run some JavaScript when they're triggered.
- v-on accepts the name of an **instance method** that you want to call

```
<form class="container my-5" v-
on:submit.prevent="submitBooking" >
```

- v-on:submit could be shortened as @submit

```
<form class="container my-5"
@submit.prevent="submitBooking" >
```

```
const submitBooking = async function () {
    // post the booking to the backend
    const response = await fetch('/api/
bookings', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/
json'
        },
        body:
JSON.stringify(booking.value)
    });
    // convert the response to json
    const json = await response.json();
    // log the json
    console.log(json);
    // alert the user
    alert(JSON.stringify(json));
}
```

Instance Method

- Notice the use of .value to **unwrap** the inner value.
- JSON.stringify() converts a JavaScript value to JSON text string

```
const submitBooking = async function () {
    // post the booking to the backend
    const response = await fetch('/api/bookings', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(booking.value)
    });
    // convert the response to json
    const json = await response.json();
    // log the json
    console.log(json);
    // alert the user
    alert(JSON.stringify(json));
}
```

Computed Property

- Computed Properties are provided with custom getter functions.
- They will be re-computed **whenever the data properties got updated**. They could be used as normal data properties.

```
// Use computed property to get the superheroes
const superheroes = computed(() => {
    if (booking.value.team == 'Avengers') {
        return ["Captain America", "Iron Man",
"Thor", "Hulk", "Black Widow", "Hawkeye"];
    } else if (booking.value.team == "JLA") {
        return ["Superman", "Batman", "Wonder
Woman", "Flash", "Green Lantern”, "Aquaman"];
    } else {
        return [];
    }
});
```

Arrow Function Expression

- An unnamed function

Input

```
() => {
    if (booking.value.team == 'Avengers') {
        return ["Captain America", "Iron Man", "Thor", "Hulk", "Black Widow",
"Hawkeye"];
    } else if (booking.value.team == "JLA") {
        return ["Superman", "Batman", "Wonder Woman", "Flash", "Green Lantern",
"Aquaman"];
    } else {
        return [];
    }
});
```

Output

Function body

Computed Property

- The arrow function expression is the **getter** function (retrieve value) provided as an argument of the computed function call

```
// Use computed property to get the superheroes
const superheroes = computed(() => {
    if (booking.value.team == 'Avengers') {
        return ["Captain America", "Iron Man", "Thor", "Hulk", "Black Widow",
"Hawkeye"];
    } else if (booking.value.team == "JLA") {
        return ["Superman", "Batman", "Wonder Woman", "Flash", "Green Lantern”,
"Aquaman"];
    } else {
        return [];
    }
});
```

Attribute Bindings with v-bind

```
<option value="" selected>Open this select menu</option>
<option value="Avengers">Avengers</option>
<option value="JLA">Justice League</option>
```

- Use a v-bind directive to develop dynamic attribute value.

```
<select class="form-select" aria-label="Default select example"
v-model="booking.superhero" v-bind:disabled="!booking.team">
</select>
```

- Vue also provides a special shorthand for v-bind, which is simply :

```
<select class="form-select" aria-label="Default select example"
v-model="booking.superhero" :disabled="!booking.team">
</select>
```

Watcher

```
watch(() => booking.value.team, (newValue) => {
    // if the new value is not empty
    if (newValue) {
        booking.value.superhero = superheroes.value[0];
    } else {
        booking.value.superhero = "";
    }
});
```

- In Vue.js, a watcher is a feature that allows you to reactively watch for changes in a specific property or expression and perform a corresponding action when that change occurs.

- Both **watchers** and **computed properties** are used to reactively update values based on changes in data.

List Rendering with v-for

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

superhero is an alias for the array element being iterated on

:key to provide an unique identifier for each array element

```
<option v-for="superhero in superheroes" :key="superhero" :value="superhero">
    {{ superhero }}
</option>
```

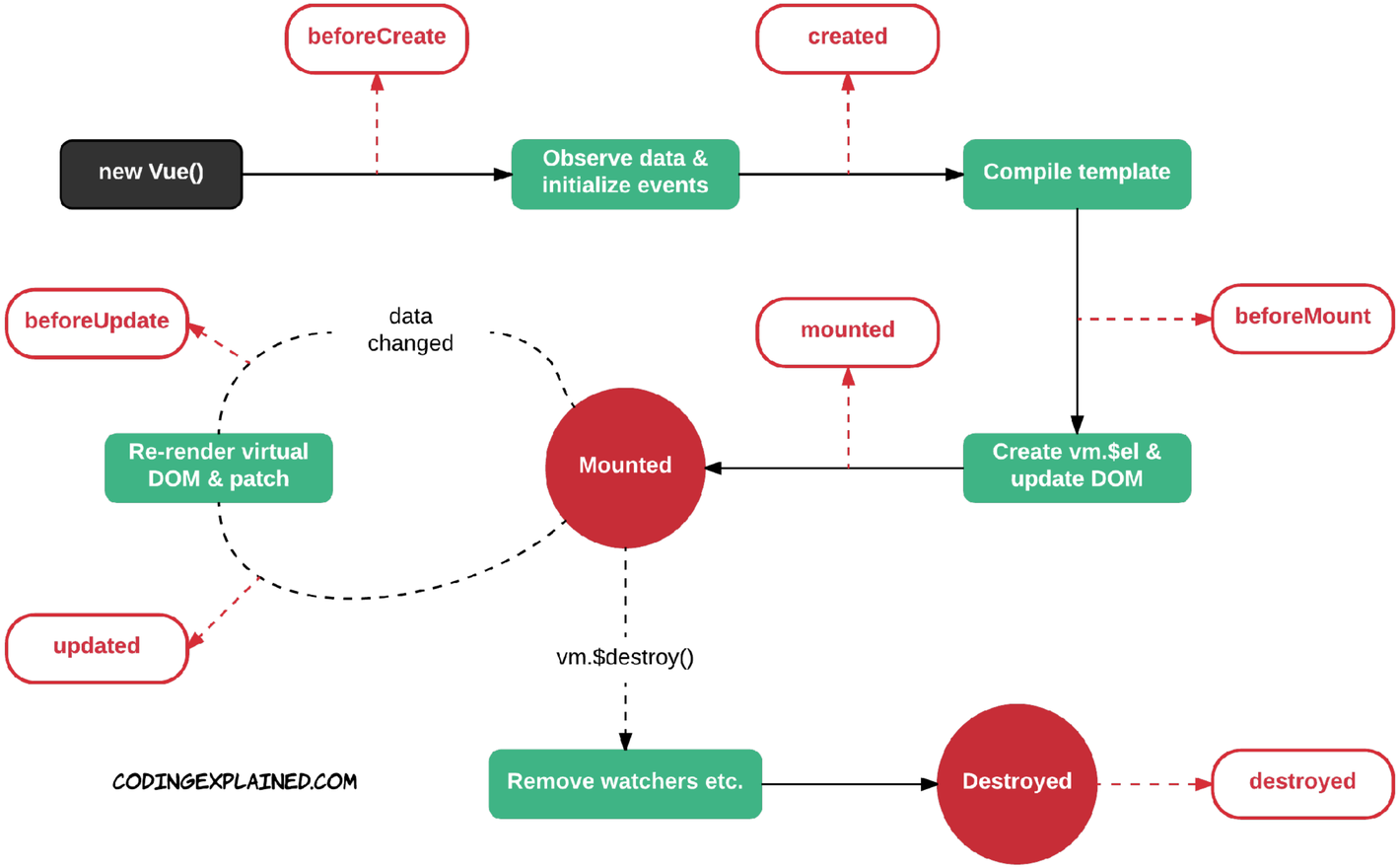
Data binding specified in double braces

```
const superheroes = computed(() => {
    if (booking.value.team == 'Avengers') {
        return ["Captain America", "Iron Man", "Thor", "Hulk", "Black
Widow", "Hawkeye"];
    } else if (booking.value.team == "JLA") {
        return ["Superman", "Batman", "Wonder Woman", "Flash", "Green
Lantern”, "Aquaman"];
    } else {
        return [];
    }
});
```

LifeCycle Hooks

- Giving users the opportunity to add their own code at specific stages

```
onMounted(async () => {
    // if there is an id in the route
    if (route.params.id) {
        await getBooking();
    }
});
```



- onMounted — Execute after the instance is mounted to the DOM element.

onMounted

- onMounted() expects an argument, in the form of a function or an **arrow function expression**.
- The argument cannot be a function call.

```
onMounted(async () => {
    // if there is an id in the route
    if (route.params.id) {
        await getBooking();
    }

    watch(() => booking.value.team, (newValue) => {
        // if the new value is not empty
        if (newValue) {
            booking.value.superhero = superheroes.value[0];
        } else {
            booking.value.superhero = "";
        }
    });
});
```

Conditional Rendering with v-if

```
<div v-if="route.name == 'view-booking'">
    <h1>{{ booking.email }}</h1>
    <p>Number of Tickets: {{ booking.numTickets }}</p>
    <p>Payment Method: {{ booking.payment }}</p>
    <p>Favourite Team: {{ booking.team }}</p>
    <p>Favourite Hero: {{ booking.superhero }}</p>
    <p>Terms and Conditions: {{ booking.terms }}</p>
</div>
```

- The directive v-if is used to conditionally render a block.
- The block will only be rendered if the directive's expression returns a **truthy** value.

```
import { useRoute } from "vue-router";

const route = useRoute();
```

Vue Router

- We can define a new route on the frontend with the following:

```
{
    path: '/booking/:id',
    name: 'view-booking',
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js) for this route
    // which is lazy-loaded when the route is visited.
    component: () => import('../views/BookingView.vue')
},
```

- In the script, we can access the dynamic path parameters via

```
route.params.id
```