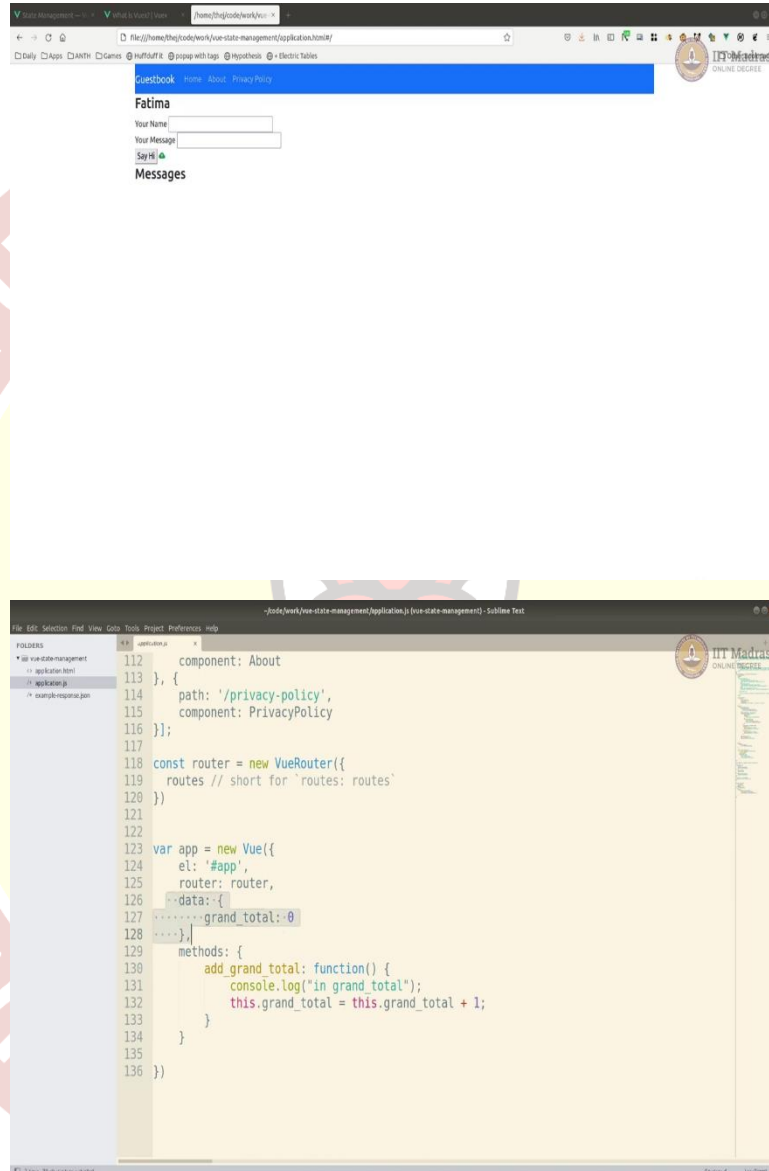


**IIT Madras**  
ONLINE DEGREE

**Modern Application Development – II**  
**Professor. Thejesh G N**  
**Software Consultant**  
**Indian Institute of Technology, Madras**  
**Statement Management using Vuex**

(Refer Slide Time: 00:14)



```
File Edit Selection Find View Goto Tools Project Preferences Help
-./code/work/vue-state-management/application.js (vue-state-management) - Sublime Text

73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101

    },
    .catch(error) => {
      console.error('Error:', error);
      this.savedIconClass = "text-danger";
    });

    this.vistor_name = "";
    this.vistor_message = "";
    this.$emit('add-to-grand-total');

  },
  computed: {
    count: function() {
      return this.messages.length;
    },
  },
  mounted: async function() {
    r = await fetch('example-response.json')
    d = await r.json()
    console.log(d)
    this.messages = [{
      "for": "fatima",
      "vistor_name": "Rajesh",
      "vistor_message": "Hello world"
    }]
  }
}
```

```
File Edit Selection Find View Goto Tools Project Preferences Help
-./code/work/vue-state-management/application.js (vue-state-management) - Sublime Text

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

const About = Vue.component('about', {
  template: `
    <div>
      <h3> About </h3>
      <p>Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque accumsan, tortor vestibulum venenatis placerat quam eu pharetra. Ut a mi imperdiet, efficitur nunc.
    </div>
  `
})

const PrivacyPolicy = Vue.component('privacy-policy', {
  template: `
    <div>
      <h3> Privacy Policy </h3>
      <p>Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque accumsan, tortor vestibulum venenatis placerat quam eu pharetra. Ut a mi imperdiet, efficitur nunc.
    </div>
  `
})

const MessageBoard = Vue.component('message-board', {
  props: ['title'],
  template: `
    <div>
      <h3> {{ title }} </h3>
      <div class="form-group">
        <label for="vistor_name">Your Name</label>
        <input type="text" id="vistor_name" v-model="vistor_name" />
      </div>
      <div class="form-group">

```

State Management - What is Vuex | Vuex | /notes/2020/06/01/

https://vuex.vuejs.org/en/index.html

Vue.js

Tooling

- Single File Components
- Testing
- TypeScript Support
- Production Deployment

Scaling Up

- Routing
- State Management
- Official Plugin Implementation
- Information for Next Developer
- Simple State Management from Scratch
- Server-Side Rendering
- Security

Internals

- Reactivity in Depth
- Migrating
- Migration from Vue 1.x
- Migration from Vue Router 1.x
- Migration from Vuex 0.x to 1.0

Meta

- Comparison with Other Frameworks
- Join the Vue.js Community!
- Meet the Team

Instance only provides access to it. Therefore, if you have a piece of state that should be shared by multiple instances, you can share it by identity.

```
var sourceOfTruth = {}

var vue = new Vue({
  data: sourceOfTruth
})

var web = new Vue({
  data: sourceOfTruth
})
```

Now whenever `sourceOfTruth` is mutated, both `vue` and `web` will update their views automatically. Subcomponents within each of these instances would also have access via `this.$store.state`. We have a single source of truth now, but debugging would be a nightmare. Any piece of data could be changed by any part of our app at any time, without leaving a trace.

To help solve this problem, we can adopt a **store pattern**:

```
var store = {
  state: {
    message: 'hello!'
  },
  setNewMessage(newValue) {
    if (this.debug) console.log('setNewMessage triggered with:', newValue)
    this.state.message = newValue
  },
  clearMessage() {
    if (this.debug) console.log('clearMessage triggered')
    this.state.message = ''
  }
}
```

Platform Sponsors

- Vue School
- VBKILL
- PASSIONATE PEOPLE
- storybook
- ionic
- NEXT.JS
- refurbed

Become a Sponsor



This brings us full circle back to [vues](#), so if you've read this far it's probably time to try it out!

- Routing
- Server-Side Rendering -

Caught a mistake or want to contribute to the documentation? [Edit this on GitHub](#)! Deployed on [Netlify](#)

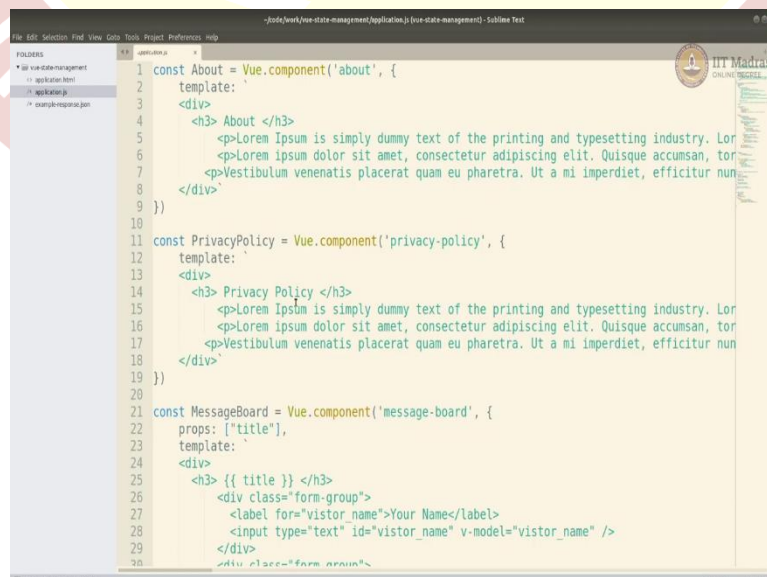
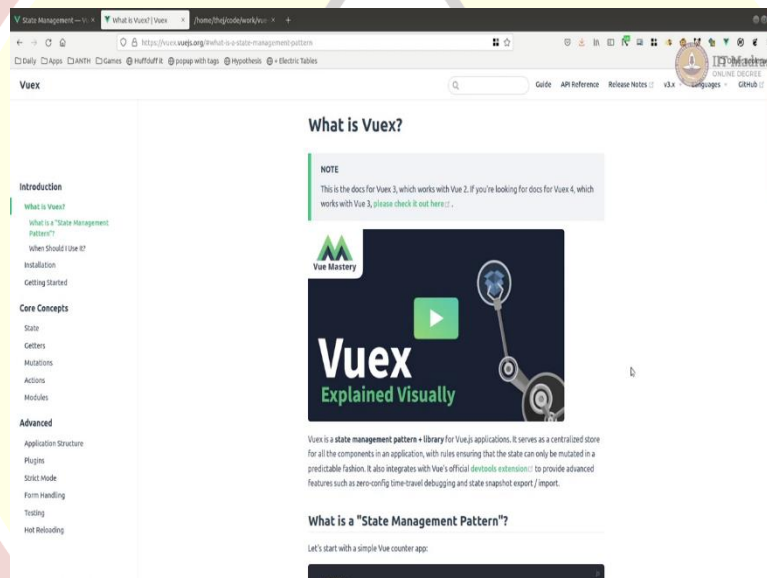
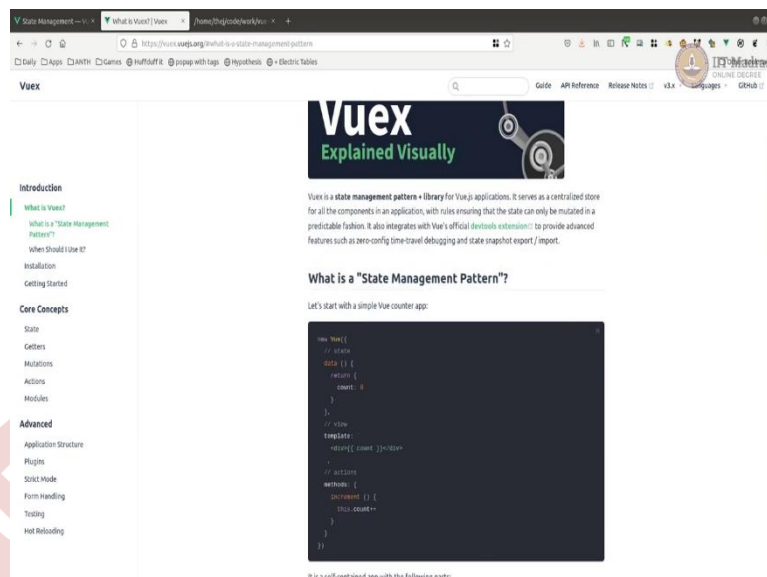
Welcome to the modern application development to screencast. In this screencast, we will see how to do advanced state management using Vuex. For this, you will need a browser, I am using Firefox, and you will need an editor, I am using Sublime Text here. One of the often overlooked parts of a Vue application is data. You have in your application here, the data. So, all your reactivity depends on that. It is the source of truth for the whole Vue application. And it is a raw object like a raw data object. And as the application grows, you will have many components.

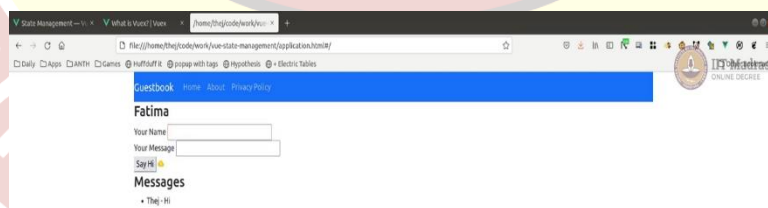
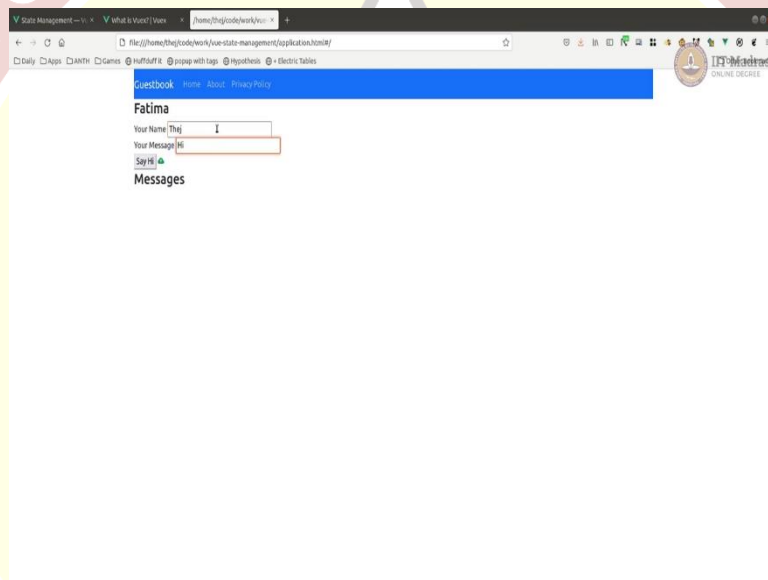
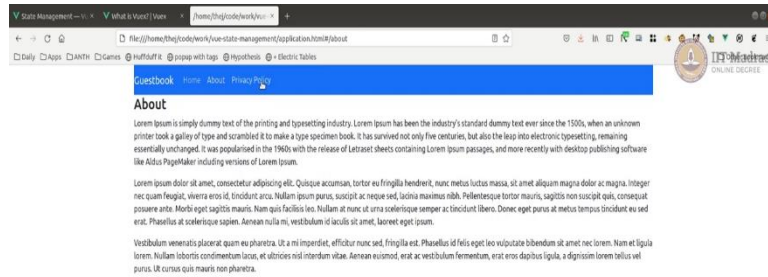
Like in this app, I mean, we have few components in this app. In a real application we will have many components. And the state is scattered across because even the components can have their own state, and the application can have its own state, so they are scattered across and managing them becomes difficult, because then you will have to see what change were flicked where, and what on the screen changed what, because of what, etc, etc. And then debugging becomes difficult.

Hence, we kind of tend to use a what we call a store pattern, where we like take out this data attribute or data from the components and stuff and like, make it at the application level and reuse it. For example, if you go to the Vue page, you can see it here I mean instead of using like a data of separate data between two, you can declare one at the top and reuse them one of the top and then reuse them.

In general, it would look like there is a store, and then use that store in each of your application or component. And basically, you declare the store as an object instead of just one attribute and then it can have its own functions that can change this. So, you have one central place, where you can change the values or clear the values, update the values, etcetera, so that you can use the functions, instead of directly changing the values. I mean, this is very basic and simple. To do anything better than Vue or versus something called Vuex.

(Refer Slide Time: 03:07)







```
File Edit Selection Find View Goto Tools Project Preferences Help
-./code/week-two-state-management/application.html (vue-state-management) - Sublime Text

FOLDERS
* vue-state-management
  - application.html
  - application.js
  - example-response.json

7 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
  integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6au08tT94W/HrtjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
8
9 <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
10 <script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
11 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0qln4gmtz2MlQnikT1wXgys0gOMhU+PILRH9sENB00LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
12 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.7.2/font/bootstrap-icons.css">
13
14 </head>
15 <body>
16 <div class="container" id="app">
17 <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
18 <a class="navbar-brand" href="#">Guestbook</a>
19 <div class="collapse navbar-collapse" id="navbarNav">
20 <ul class="navbar-nav">
21 <li class="nav-item"> <router-link class="nav-link" to="/">Home</router-link> </li>
22 <li class="nav-item"> <router-link class="nav-link" to="/about">About</router-link> </li>
23 <li class="nav-item"> <router-link class="nav-link" to="/privacy-policy">Privacy Policy</router-link> </li>
24 </ul>
25 </div>
26 </nav>
27
28 <div class="row">
29 <div class="col">
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
File Edit Selection Find View Goto Tools Project Preferences Help
-./code/week-two-state-management/application.html (vue-state-management) - Sublime Text

FOLDERS
* vue-state-management
  - application.html
  - application.js
  - example-response.json

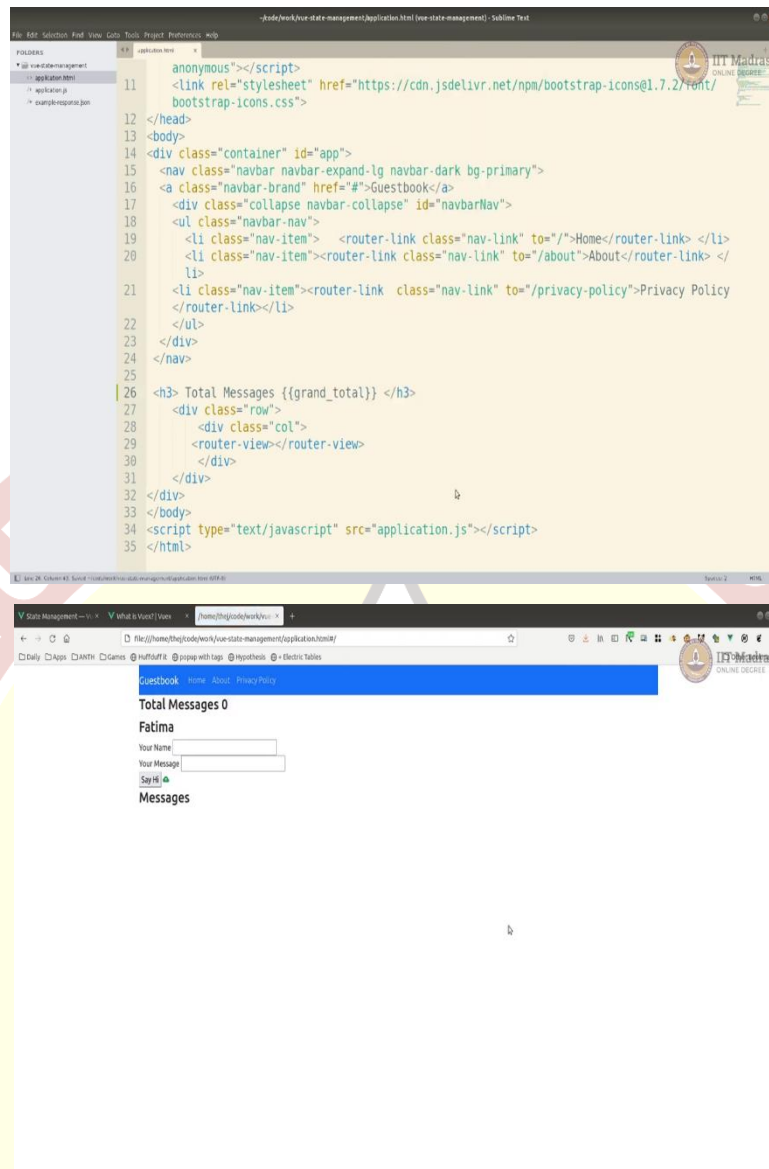
22 </ul>
23 </div>
24 </nav>
25
26 <div class="row">
27 <div class="col">
28 <router-view></router-view>
29 </div>
30 </div>
31 </body>
32 <script type="text/javascript" src="application.js"></script>
33 </html>
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
File Edit Selection Find View Goto Tools Project Preferences Help
-./code/week-two-state-management/application.html (vue-state-management) - Sublime Text

FOLDERS
* vue-state-management
  - application.html
  - application.js
  - example-response.json

11 <script></script>
12 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.7.2/font/bootstrap-icons.css">
13
14 </head>
15 <body>
16 <div class="container" id="app">
17 <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
18 <a class="navbar-brand" href="#">Guestbook</a>
19 <div class="collapse navbar-collapse" id="navbarNav">
20 <ul class="navbar-nav">
21 <li class="nav-item"> <router-link class="nav-link" to="/">Home</router-link> </li>
22 <li class="nav-item"> <router-link class="nav-link" to="/about">About</router-link> </li>
23 <li class="nav-item"> <router-link class="nav-link" to="/privacy-policy">Privacy Policy</router-link> </li>
24 </ul>
25 </div>
26 </nav>
27
28 <div class="row">
29 <div class="col">
30 <router-view></router-view>
31 </div>
32 </div>
33 </body>
34 <script type="text/javascript" src="application.js"></script>
35 </html>
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```





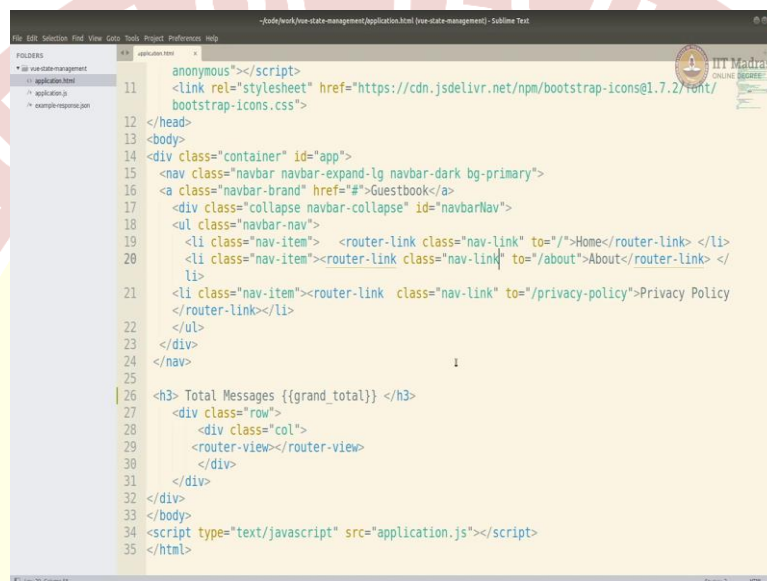
If you go to Vuex page, you can read about what it means. But in in small, or in short, Vuex offers a way to manage the state across the Vue application in a centralized way. It also integrates into Vue dev tools. I will show you how it integrates into Vue dev tools. So, what it can do is, with that integration, you can do travel debugging, time travel debugging, which means you can go back to the previous state hour, you can go forward to the previous state so that it helps you in debugging. So that since we are going back in time of the state, it is called, a time travel debugging. I will show an example as we go down.

Now let us start with the using it in our case, I have opened the same application that we had done last time with some routers and some data and this is how it looks. And you can go to a boat privacy etc, and then you have a home. And then you do give a name and say hi. It does certain things and then it shows you though, message here and name of the component or the title of the component view is showing here.

One of the thing that we missed, or we did not show his, you know, this grand total that we had before it still we still have it as part of the application. If we go to the application we still have it at the application, but we are not showing it as part of the application dot HTML. So, let us just put it there.

We had it here. We can just put it somewhere here, let us put here total. So, you can just refresh this, we have total messages, 0.

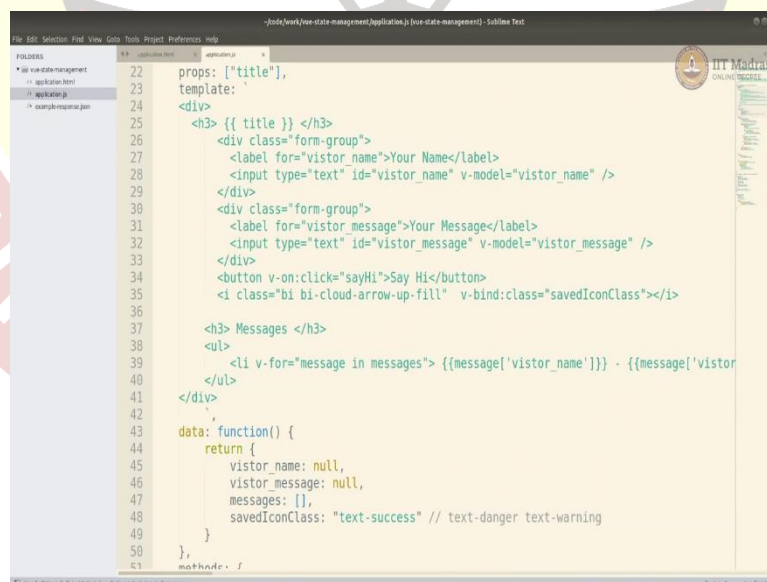
(Refer Slide Time: 05:26)



```
File Edit Selection Find View Goto Tools Project Preferences Help
-Code/Work/Non-state-management/Application.html (vue-state-management) - Sublime Text

FOLDERS
  * vue-state-management
    - application.html
    - application.js
    - example-response.json

11 anonymous"></script>
12 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.7.2/font/
13 bootstrap-icons.css">
14 </head>
15 <body>
16 <div class="container" id="app">
17 <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
18 <a class="navbar-brand" href="#">Guestbook</a>
19 <div class="collapse navbar-collapse" id="navbarNav">
20 <ul class="navbar-nav">
21 <li class="nav-item"><router-link class="nav-link" to="/">Home</router-link> </li>
22 <li class="nav-item"><router-link class="nav-link" to="/about">About</router-link> </li>
23 <li class="nav-item"><router-link class="nav-link" to="/privacy-policy">Privacy Policy
24 </router-link></li>
25 </ul>
26 </div>
27 </nav>
28
29 <h3> Total Messages {{grand_total}} </h3>
30 <div class="row">
31 <div class="col">
32 <router-view></router-view>
33 </div>
34 </div>
35 </body>
36 <script type="text/javascript" src="application.js"></script>
37 </html>
```



```
File Edit Selection Find View Goto Tools Project Preferences Help
-Code/Work/Non-state-management/Application.js (vue-state-management) - Sublime Text

FOLDERS
  * vue-state-management
    - application.html
    - application.js
    - example-response.json

22 props: ['title'],
23 template: `
24 <div>
25 <h3> {{ title }} </h3>
26 <div class="form-group">
27 <label for="vistor name">Your Name</label>
28 <input type="text" id="vistor_name" v-model="vistor_name" />
29 </div>
30 <div class="form-group">
31 <label for="vistor message">Your Message</label>
32 <input type="text" id="vistor_message" v-model="vistor_message" />
33 </div>
34 <button v-on:click="sayHi">Say Hi</button>
35 <i class="bi bi-cloud-arrow-up-fill" v-bind:class="savedIconClass"></i>
36
37 <h3> Messages </h3>
38 <ul>
39 <li v-for="message in messages"> {{message['vistor_name']}} - {{message['vistor
40 </li>
41 </ul>
42 </div>
43 `
44 data: function() {
45   return {
46     vistor_name: null,
47     vistor_message: null,
48     messages: [],
49     savedIconClass: "text-success" // text-danger text-warning
50   }
51 }
52 }
53 module.exports = {
54   props,
55   template,
56   data
57 };
```

```
File Edit Selection Find View Goto Tools Project Preferences Help
-:code\work\vue-state-management\application.js (vue-state-management) - Sublime Text

FOLDERS
  * vue-state-management
    - application.html
    - application.js
    - example-response.json

75     console.error('Error:', error);
76     this.savedIconClass = "text-danger";
77   });
78
79   this.vistor_name = "";
80   this.vistor message = "";
81   this.$emit('add-to-grand-total');
82
83   },
84
85   },
86   computed: {
87     count: function() {
88       return this.messages.length;
89     }
90   },
91
92   mounted: async function() {
93     r = await fetch("example-response.json")
94     d = await r.json()
95     console.log(d)
96     this.messages = [{
97       "for": "fatima",
98       "vistor_name": "Rajesh",
99       "vistor_message": "Hello world"
100     }]
101   }
102 })
103
```

```
File Edit Selection Find View Goto Tools Project Preferences Help
-:code\work\vue-state-management\application.html (vue-state-management) - Sublime Text

FOLDERS
  * vue-state-management
    - application.html
    - application.js
    - example-response.json

18 <ul class="navbar-nav">
19   <li class="nav-item"> <router-link class="nav-link" to="/">Home</router-link></li>
20   <li class="nav-item"><router-link class="nav-link" to="/about">About</router-link> </li>
21   <li class="nav-item"><router-link class="nav-link" to="/privacy-policy">Privacy Policy
22 </router-link></li>
23 </ul>
24 </nav>
25
26 <h3> Total Messages {{grand_total}} </h3>
27 <div class="row">
28   <div class="col">
29     <router-view></router-view>
30   </div>
31 </div>
32 </div>
33 </body>
34 <script type="text/javascript" src="application.js"></script>
35 </html>
```

```
File Edit Selection Find View Goto Tools Project Preferences Help
-:code\work\vue-state-management\application.js (vue-state-management) - Sublime Text

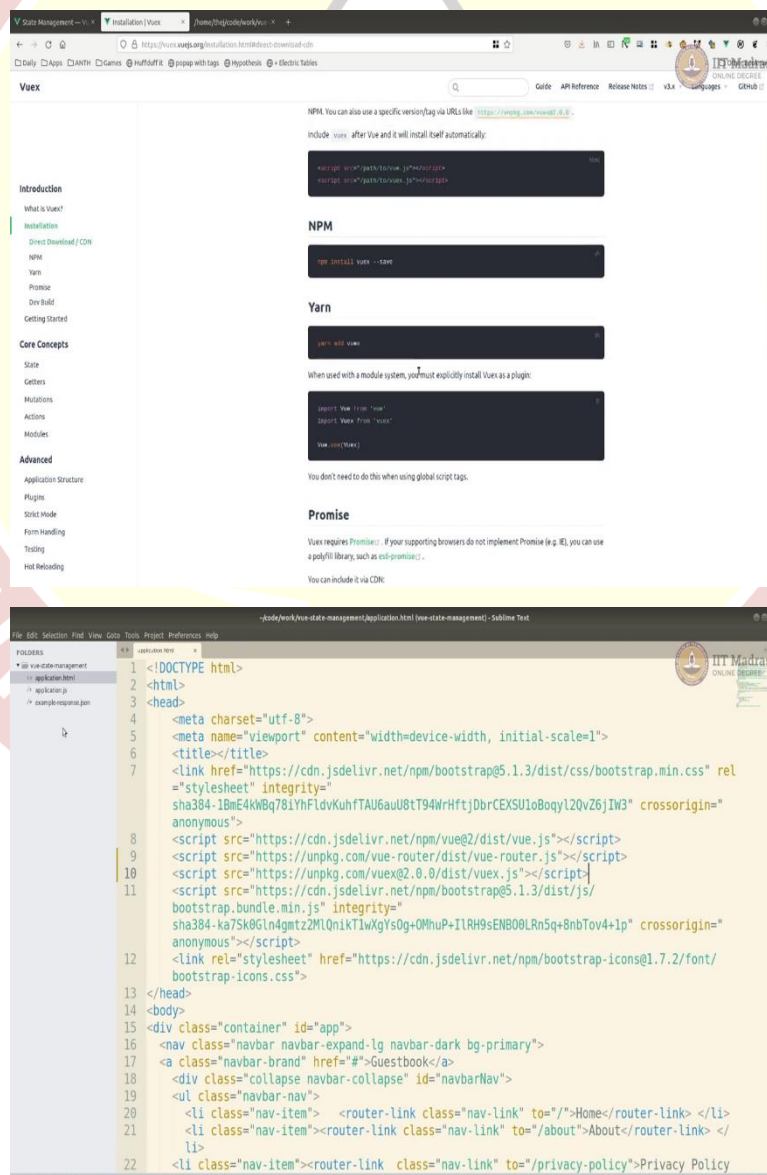
FOLDERS
  * vue-state-management
    - application.html
    - application.js
    - example-response.json

106 const routes = [
107   path: '/',
108   component: MessageBoard,
109   props: { title: 'Fatima' }
110 ], {
111   path: '/about',
112   component: About
113 }, {
114   path: '/privacy-policy',
115   component: PrivacyPolicy
116 }];
117
118 const router = new VueRouter({
119   routes // short for 'routes: routes'
120 })
121
122
123 var app = new Vue({
124   el: '#app',
125   router: router,
126   data: {
127     grand_total: 0
128   },
129   methods: {
130     addGrandTotal: function(){
131       console.log("in-grand total");
132       this.grand_total = this.grand_total + 1;
133     }
134   }
135 })
```

Now let us assume this, we had many components. And that would, they are all trying to access this total messages. And we are also have, we also have tried to omit this add two grand total, which we had it in as a V on here now we do not have it because we cannot, like, I do not want to have it here, I want it to manage as part of the state. Because ultimately, we are changing the state. Instead of emitting here, I really want to change the state of the application this directly instead of calling this and then doing that state change.

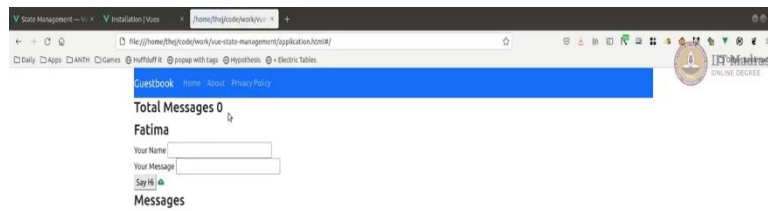
So, what I will do is, I will probably move this data to a centralized store using Vue store, and then I will try to handle this, instead of emitting the event and handling it, I will directly use the Vue store or Vuex rather to do that change, and we will see how to do that. So, first of all, let us remove this from here and move to a separate store called Vuex store.

(Refer Slide Time: 06:44)









Now, before we do that, we have to include another file, we have to install the Vuex. And that is a straightforward install. Since we are using JavaScript, you can install it directly as by adding the `or` and script importing the JS file. Let us go back and add here our Vuex after the Vuex route. This is where, this is the file that is required to do this. I am adding it here. Now I am going to declare a Vuex store.

You can declare it at the top, but I am just going, yeah, let us declare it at our top. And that can be done by doing this and here, here. So, we are declaring a constant called `store`, which is a Vuex store and that has a state and the grand total as 0. Now we have to use this store in our Vue app. So, that can be done by.

I will, first of all, I will remove this. We do not need it because we are going to use a centralized data now, `store` colon `store`. So that will get used now instead of the data. Just like how we included router before I am including the store now. So, we have done one step now and we using this.

Now, if I refresh the page, it is not going to show because now, actually, this grand total or the data that we are trying to access here is not part of the Vue app, it is part of the store, so we should be able to pretend that. That can be done in many ways in the. I will go with the easier one which we have learned using the computed method. Computed method, we can call it the same grand total so that it kind of matches.

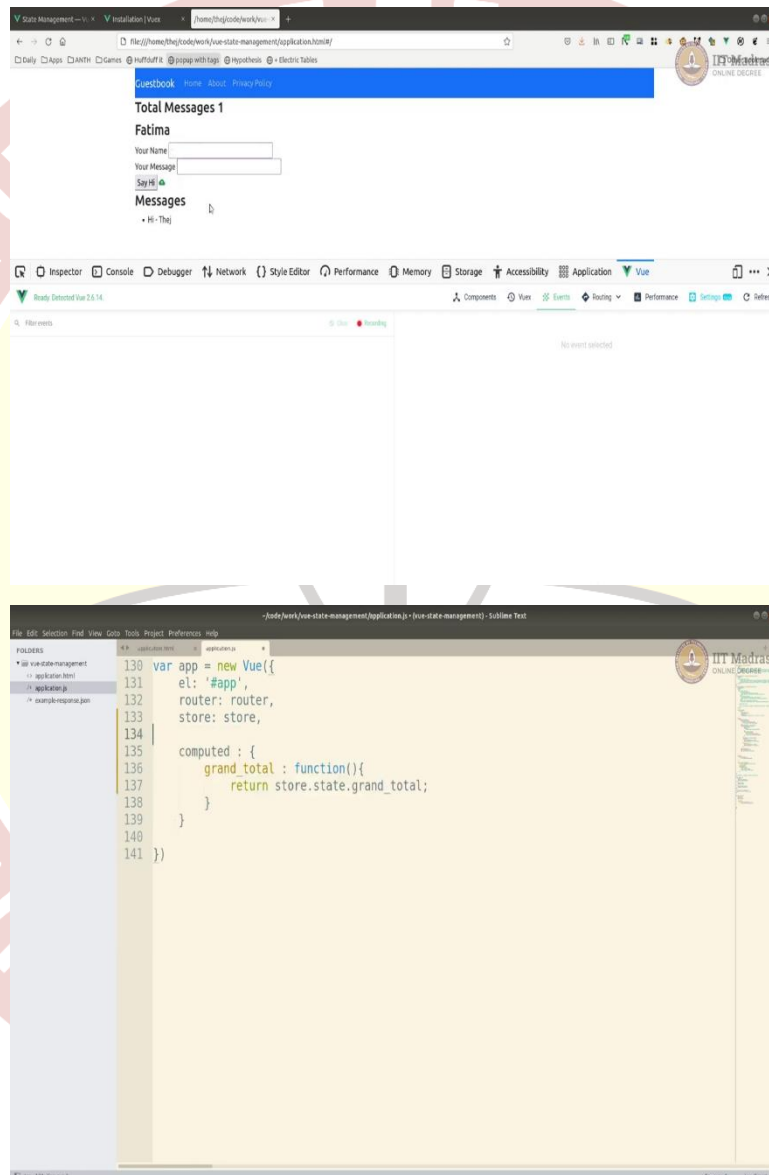
Now we have to return from the store. Like not from the `this` dot grand total because there it is not there in this, it has to come from the store. So, you can just since we have access global



access, we can just call this. I am going to remove this because this would not. I will show it later.

Anyway, you have computed here you can do grand total, and then function and then use that computed call here. Now at least it should choose 0. It should show 0. Now if I add let us do inspect, and then say, Thej, Hi, say hi.

(Refer Slide Time: 10:16)



```
File Edit Selection Find View Tools Project Preferences Help
-:/code/work/vue-state-management/application.js (vue-state-management) - Sublime Text

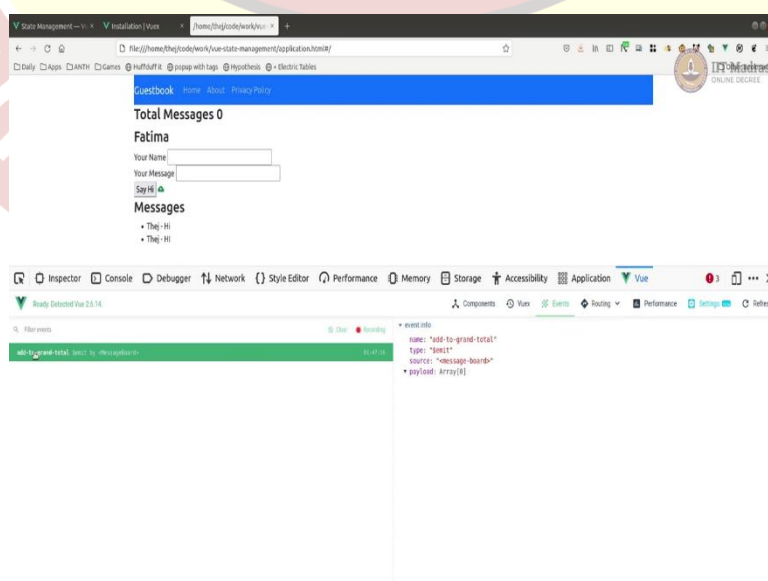
FOLDERS
  • vue-state-management
    • application.html
    • application.js
    • example-response.json

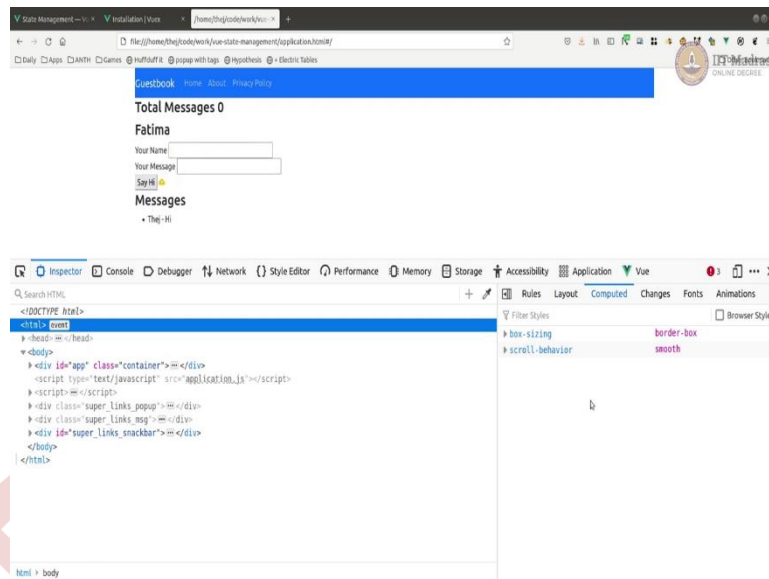
130 var app = new Vue({
131   el: '#app',
132   router: router,
133   store: store,
134   methods: {
135     add_grand_total: function() {
136       console.log('in-grand-total');
137       this.grand_total = this.grand_total + 1;
138     },
139   },
140   computed: {
141     grand_total: function() {
142       return store.state.grand_total;
143     }
144   }
145 })
146
```

```
File Edit Selection Find View Tools Project Preferences Help
-:/code/work/vue-state-management/application.js (vue-state-management) - Sublime Text

FOLDERS
  • vue-state-management
    • application.html
    • application.js
    • example-response.json

77
78 .then(data => {
79   console.log('Success:', data);
80   this.savedIconClass = "text-success";
81 })
82 .catch(error => {
83   console.error('Error:', error);
84   this.savedIconClass = "text-danger";
85 });
86
87 this.vistor_name = "";
88 this.vistor message = "";
89 this.$emit('add-to-grand-total');
90
91 }
92 },
93 computed: {
94   count: function() {
95     return this.messages.length;
96   }
97 },
98
99 mounted: async function() {
100   r = await fetch("example-response.json")
101   d = await r.json()
102   console.log(d)
103   this.messages = [{
104     "for": "fatima",
105     "vistor_name": "Rajesh",
106   }]
```





Now if we go to the Vue we can just click once more. So, you can see that it has raised in event, but we are not handling this event, basically. So, nothing is getting done because we are not handling that event to update this attribute, we are just raising the event, that is it, not handling the event. So, basically this is useless. Instead of that we can do something that is done by actually changing this value here, instead of raising the event we can actually change that value.

We will see how to do that with respect to a store, which we used to do here, like before. So, since we are going to use this now, I am going to remove this just to reduce the confusion. And we will have to do something here. To change the value we can do store dot grand total plus, plus let us assume it will be store dot state dot I think.

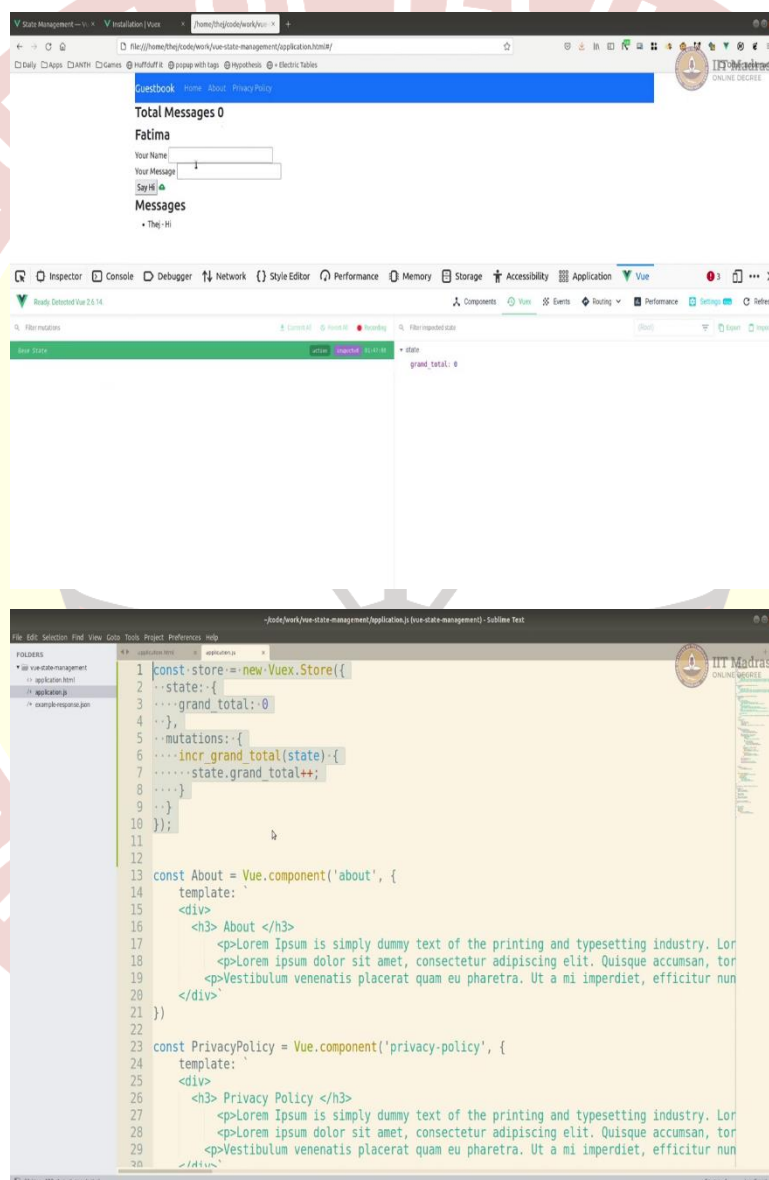
Let us check, what happened. I am going to comment in this. We are going to clear the console. Go to Vue, say, hi, Thej. This is changing, but ideally, we should not be doing it because then again we will laid into the same problem as to who is changing what, then it becomes much more difficult also because we have access to store object directly. Sometimes you may not have access to store objects directly here we have. But sometimes you may not have access to that.

Anyway, we should not be doing this. Instead, we should be calling something called mutation Vuex offers like another type of functions called mutations, which are generally used to change the values in the store. So, like, by the meaning, by definition, a mutation is the one which makes a change, mutates, updates or increments, decrements and stuff like that. So, you can add a mutation here to change it. You can say mutations I am just want to copy paste say, increment the grand total.

Now mutations will have will get state as the input, and then we update the state or increment the state. Now, you can do something called all mutations needs to be committed to make the change to this. So, instead of doing this, we will actually do commit, that can be done by state dot commit by doing this. Here, we are getting access to the store by dollar store.

Like we have we used to get access to data, and then, we are doing a commit, not changing it directly we are doing a commit.

(Refer Slide Time: 14:17)



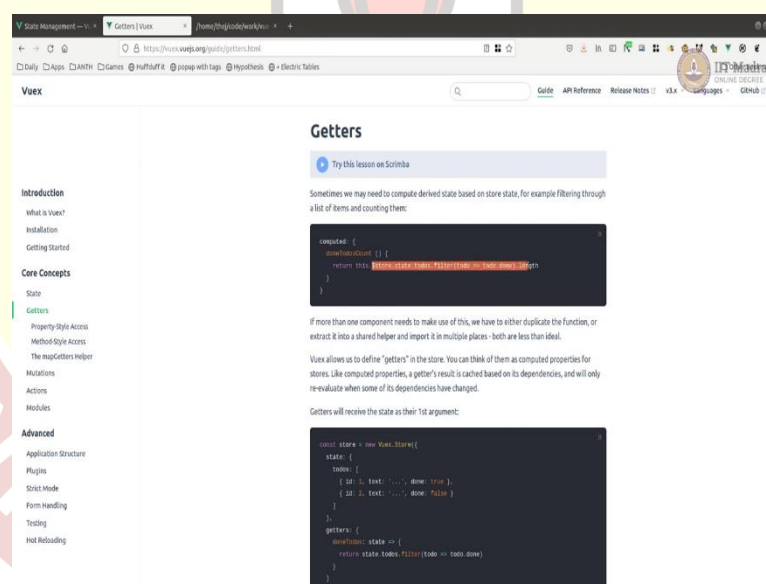
Now this has two advantages. I will show it to you. Let us go over here and go to Vuex and it says base state. Now, let us say Thej, Hi. Now we go on back you can see here, the there is a commit increment grand total was committed. Now we can reward this now it goes back to 0.

Now you can see that the messages is not reverting. That is because our messages are still part of the component. We are still keeping it as part of the component.

We can move this also here to have a global set of messages, if you want to maintain it like that. But this also shows that even if you have a global Vuex, your component can still have its own data, its own specific private data. And you can see, you can store the application level data in a common place, that way you can separate out some of it, like to manage the data where it belongs, rather. You can do a step of moving this to actually the global store and using them. I am not going to do that, but you can do that.

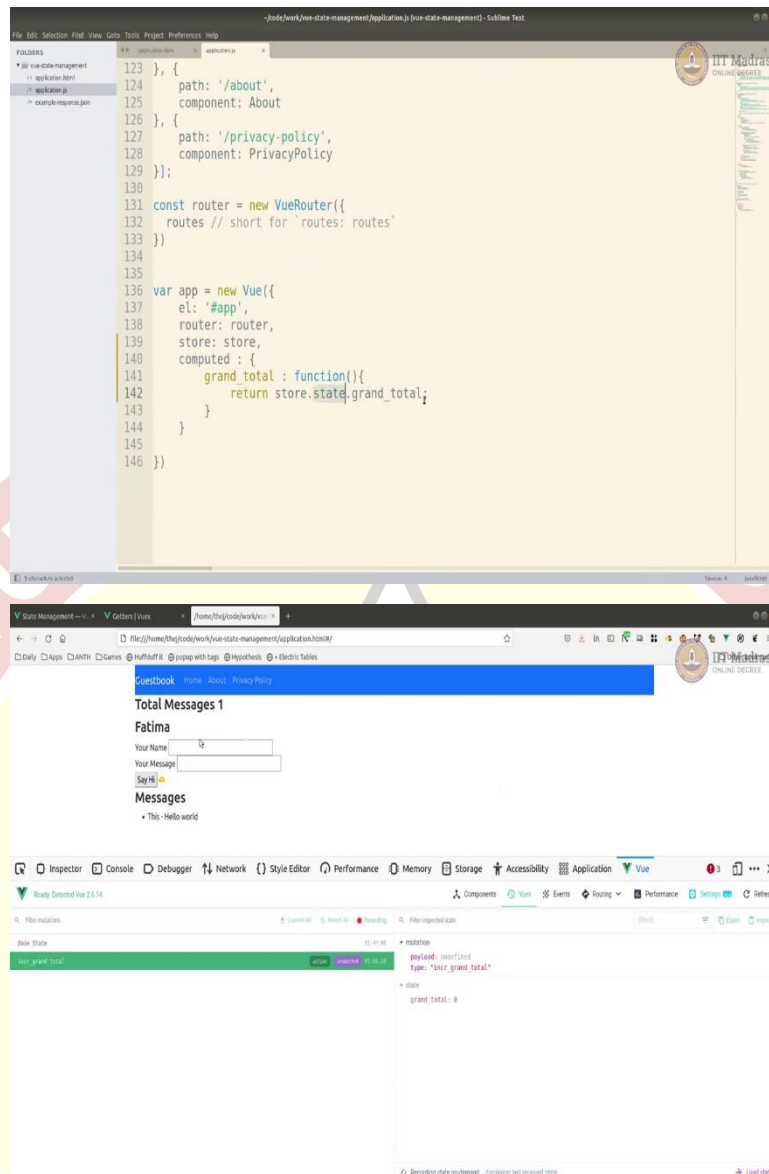
Now we saw that you can update or mutate using commit. You actually commit a mutation by sending the name of the mutation and you can also pass some values if that mutation or that function takes values instead of changing it directly. And that helps you to do this history or time travel, like we spoke before. You can do this time travel using because we are doing commit. Now, let us go back and see what other things your Vuex offers.

(Refer Slide Time: 16:30)









Just like mutations, it also offers getters. Now, getters are you, there are two way you can access the data inside store. We already saw one way which is direct attribute access like store dot state dot grand total. You can also do this dot dollar store, you can also do that. But there is a better way to access it using getters.

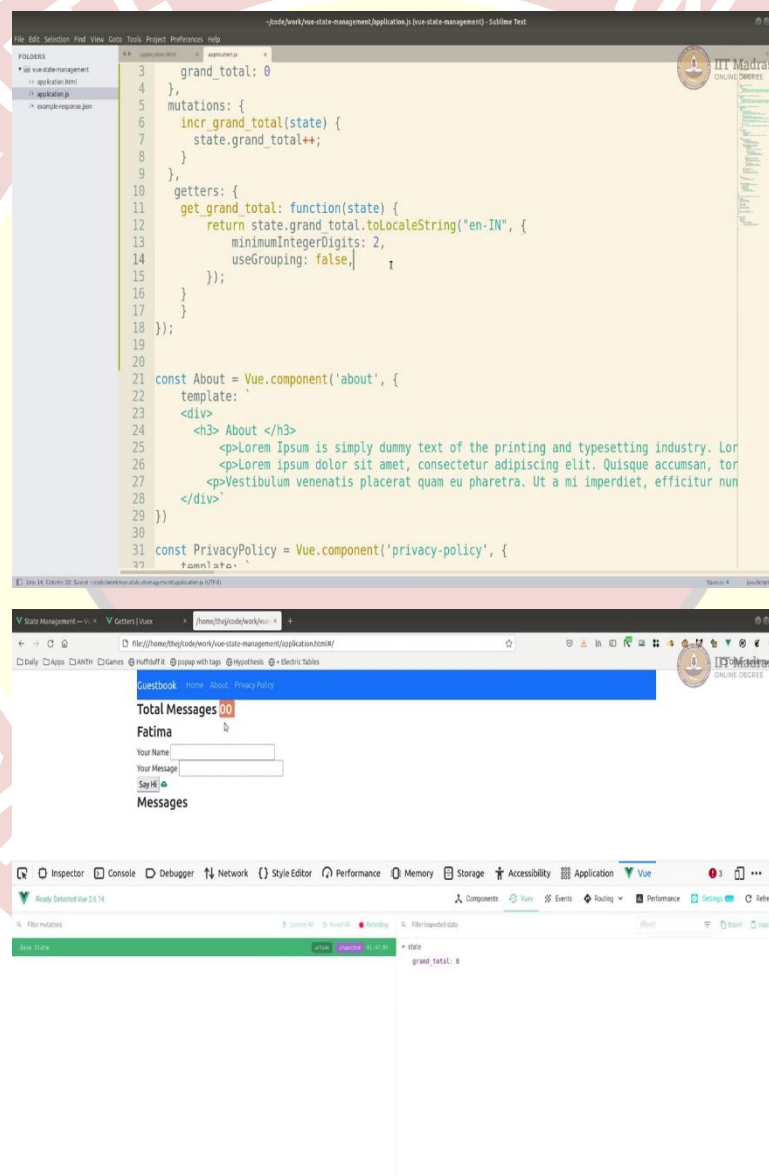
Now getters are the functions that will return the value. You can also do some modifications or you can do plain simple return. Let us do like a simple one first and then I will see like a better usage of getter. Like just like mutations, you have to define getters as function and then here, here.

And then in this you can for simple purpose you can just return if it is a simple thing, you can just do state dot grand total your do not need to do anything else. It is actually written. Now, let me just check if it is all closed. Yeah, now you can use this getter instead of directly using

the attribute name. Let us go down and use that instead of the attribute way of accessing it, you can do this. Now here I am from the store I am getting letters I am doing get grand total.

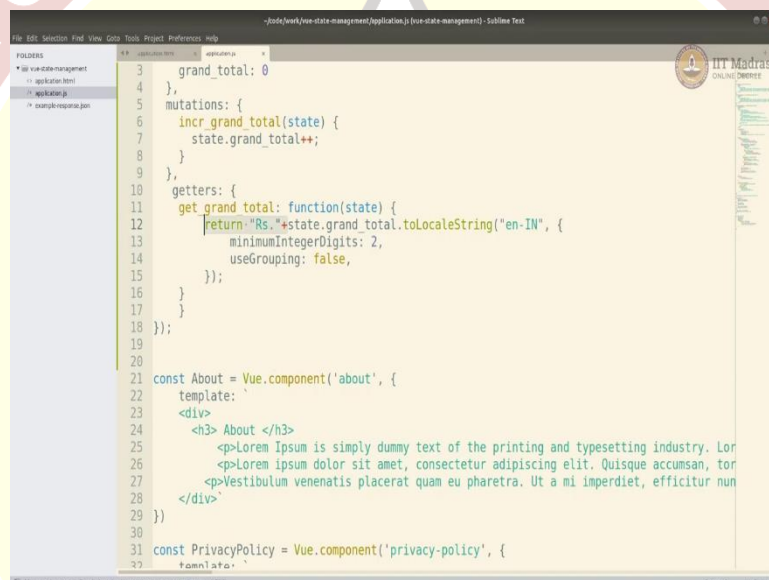
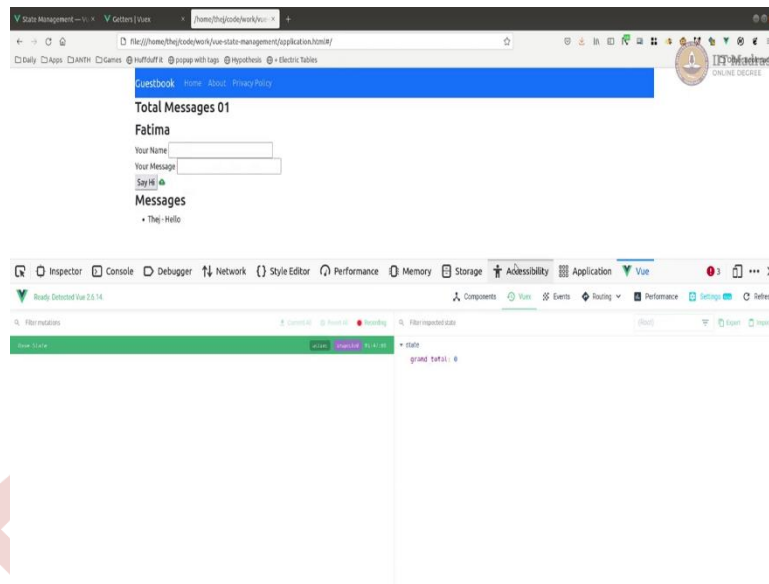
Now if I when I refresh, you can also do this dot dollar store, if you want to access from inside the component. Now we were accessing from outside. Also, our store object is directly accessible. Let us go back here and refresh this. We have refreshed it, then if I do this dot HelloWorld, as message then you get this. It is going through the getters going to get grand total and doing it.

(Refer Slide Time: 19:09)



```
3 grand_total: 0
4 },
5 mutations: {
6   incr_grand_total(state) {
7     state.grand_total++;
8   }
9 },
10 getters: {
11   get_grand_total: function(state) {
12     return state.grand_total.toLocaleString("en-IN", {
13       minimumIntegerDigits: 2,
14       useGrouping: false,
15     });
16   }
17 }
18 });
19
20
21 const About = Vue.component('about', {
22   template: `
23     <div>
24       <h3> About </h3>
25       <p>Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and composed it to create a sample book.
26       <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque accumsan, tortor ac vestibulum venenatis placerat quam eu pharetra. Ut a mi imperdiet, efficitur nunc.
27     </div>`
28 })
29
30
31 const PrivacyPolicy = Vue.component('privacy-policy', {
32   template: `
33     <div>
34       <h3> Privacy Policy </h3>
35       <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque accumsan, tortor ac vestibulum venenatis placerat quam eu pharetra. Ut a mi imperdiet, efficitur nunc.
36     </div>`
37 })
```

The screenshot shows a web browser displaying a Vue.js application. The application has a header with links: Home, About, Privacy Policy. The main content area shows 'Total Messages 00' in a blue box, followed by 'Fatima' and a form with 'Your Name' and 'Your Message' input fields, a 'Say Hi' button, and a 'Messages' section. The bottom of the browser shows the Vue.js DevTools component inspector, displaying the state object with 'grand\_total: 0'.



Let us say you want to do some formatting, you want to show it and let go like in some way. Let us say you want to format it as a two-digit number all the time or three-digit number all the time. I mean, it makes sense sometimes if you are doing a decimal point 00 you want to show two decimal points or you want to show rupee symbol and things like that, but here I am just going to make it like a formatted into two digits.

So, I am just going to use to grant local string and like have two digits as a minimum grouping. Just making sure it is all closed. Now if you go here, you can see it has become 00 to two minimum digits. Then if I do Thej Hello World then it becomes 01. Now if it was like a rupee or something you could have done RS or actually rupee symbol and some things like that.

So, you can use it for like doing formatting and many other things. You can have as many getters as you want. You could have different getters for different formatting and stuff. So we saw two types, one is mutations, and one is getters.

(Refer Slide Time: 20:34)

The top screenshot shows the Vue.js documentation page for 'State Management'. It features a diagram with two green circles labeled 'Component' connected by a dashed line with an arrow pointing from one to the other, labeled 'Trigger Updates'. Below the diagram, there is a text box stating: 'It's important to note that you should never replace the original state object in your actions - the components and the store need to share reference to the same object in order for mutations to be observed.' Below this, another text box explains: 'As we continue developing the convention where components are never allowed to directly mutate state that belongs to a store, but should instead dispatch events that notify the store to perform actions, we eventually arrive at the flux architecture. The benefit of this convention is we can record all state mutations happening to the store and implement advanced debugging helpers such as mutation logs, snapshots, and history re-rolls / time travel. This brings us full circle back to vuex, so if you've read this for it's probably time to try it out!' There are links for 'Routing' and 'Server-Side Rendering'. At the bottom, it says 'Caught a mistake or want to contribute to the documentation? Edit this on GitHub. Deployed on Netlify.'

The bottom screenshot shows the Vue.js documentation page for 'Actions'. It has a section titled 'Actions' with a sub-section 'Try this lesson on Scrimba'. Below this, it states: 'Actions are similar to mutations, the differences being that: • Instead of mutating the state, actions commit mutations. • Actions can contain arbitrary asynchronous operations.' It then says 'Let's register a simple action.' and shows a code example for a Vuex store:

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    decrement (state) {
      state.count--
    }
  },
  actions: {
    decrement (context) {
      context.commit('decrement')
    }
  }
})
```

Below the code, it explains: 'Action handlers receive a context object which exposes the same set of methods/properties on the store instance, so you can call `context.commit` to commit a mutation, or access the state and getters via `context.state` and `context.getters`. We can even call other actions with `context.dispatch`. We will see why this context object is not the store instance itself when we introduce `Modules` later. In practice, we often use ES2015 `argument destructuring` to simplify the code a bit (especially when we need to call `context` multiple times):'

```
File Edit Selection Find View Goto Tools Project Preferences Help
-:/code/work/vue-state-management/application.js - (vue-state-management) - Sublime Text

FOLDERS
  * vue-state-management
    - application.html
    - application.js
    - example-response.json

1 const store = new Vuex.Store({
2   state: {
3     grand_total: 0
4   },
5   mutations: {
6     incr_grand_total(state) {
7       state.grand_total++;
8     }
9   },
10  getters: {
11    get_grand_total: function(state) {
12      return state.grand_total.toLocaleString("en-IN", {
13        minimumIntegerDigits: 2,
14        useGrouping: false,
15      });
16    }
17  },
18  actions: {
19    send_first_and_then_incr(context) {
20      //fetch or ajax call
21
22      //sucess you can call commit
23
24      //failure
25    }
26  }
27 });
28
29 const About = Vue.component('about', {
30
```

```
File Edit Selection Find View Goto Tools Project Preferences Help
-:/code/work/vue-state-management/application.js - (vue-state-management) - Sublime Text

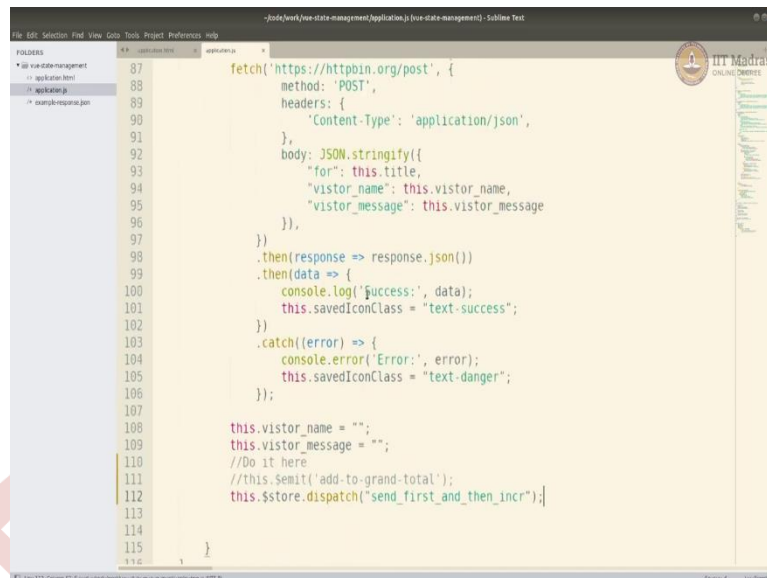
FOLDERS
  * vue-state-management
    - application.html
    - application.js
    - example-response.json

98
99
100 .then(response => response.json())
101 .then(data => {
102   console.log('Success:', data);
103   this.savedIconClass = "text-success";
104 })
105 .catch(error => {
106   console.error('Error:', error);
107   this.savedIconClass = "text-danger";
108 })
109
110 this.vistor name = "";
111 this.vistor_message = "";
112 //Do it here
113 //this.$emit('add-to-grand-total');
114 this.$store.commit('incr_grand_total')
115
116 },
117 computed: {
118   count: function() {
119     return this.messages.length;
120   }
121 },
122
123 mounted: async function() {
124   r = await fetch("example-response.json")
125   d = await r.json()
126   console.log(d)
127   this.messages = [{
128
```

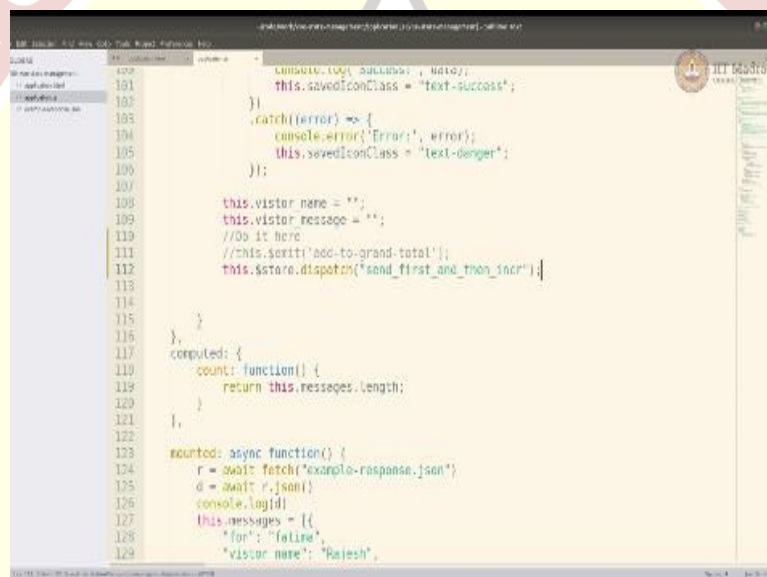
```
File Edit Selection Find View Goto Tools Project Preferences Help
-:/code/work/vue-state-management/application.js - (vue-state-management) - Sublime Text

FOLDERS
  * vue-state-management
    - application.html
    - application.js
    - example-response.json

2   state: {
3     grand_total: 0
4   },
5   mutations: {
6     incr_grand_total(state) {
7       state.grand_total++;
8     }
9   },
10  getters: {
11    get_grand_total: function(state) {
12      return state.grand_total.toLocaleString("en-IN", {
13        minimumIntegerDigits: 2,
14        useGrouping: false,
15      });
16    }
17  },
18  actions: {
19    send_first_and_then_incr(context) {
20      //fetch or ajax call
21
22      //sucess you can call commit
23      store.state.commit('incr_grand_total')
24      //failure
25    }
26  }
27 });
28
29
30 const About = Vue.component('about', {
31   template: `
32
```



```
87 fetch('https://httpbin.org/post', {
88   method: 'POST',
89   headers: {
90     'Content-Type': 'application/json',
91   },
92   body: JSON.stringify({
93     "for": this.title,
94     "vistor_name": this.vistor_name,
95     "vistor_message": this.vistor_message
96   }),
97 })
98 .then(response => response.json())
99 .then(data => {
100   console.log('Success:', data);
101   this.savedIconClass = "text-success";
102 })
103 .catch(error => {
104   console.error('Error:', error);
105   this.savedIconClass = "text-danger";
106 });
107
108 this.vistor_name = "";
109 this.vistor_message = "";
110 //Do it here
111 //this.$emit('add-to-grand-total');
112 this.$store.dispatch("send_first_and_then_incr");
113
114 }
115 }
```



```
101 console.log('Success:', data);
102 this.savedIconClass = "text-success";
103 })
104 .catch(error => {
105   console.error('Error:', error);
106   this.savedIconClass = "text-danger";
107 });
108
109 this.vistor_name = "";
110 this.vistor_message = "";
111 //Do it here
112 //this.$emit('add-to-grand-total');
113 this.$store.dispatch("send_first_and_then_incr");
114
115 },
116 computed: {
117   count: function() {
118     return this.messages.length;
119   }
120 },
121 mounted: async function() {
122   r = await fetch("example-response.json")
123   d = await r.json()
124   console.log(d)
125   this.messages = [{
126     "for": "felix",
127     "vistor name": "Rajesh",
128   }]
```

Now, there is another type called actions. When you want to perform certain actions on the on this state in the store, you can see them here. Now there is one major difference between actions and mutations. The difference is everything inside the mutation has to be synchronous, it cannot be a synchronous. You cannot make it like a promise and wait and then commit or then change the value.

Because if you do that, then we would not be able to use this history management in the thing. And it also, it is a state that you want to be sure that it is when it is committed, it means it is changed. So, whatever you write inside the mutation has to be synchronous, but whatever you write inside the action does not have to be synchronous. You can do some Ajax calls. And as part of that, at the end of it, you can call a mutation and commit. So that where actions are useful.



So, we can see and read about that in the page. Basically, that is the difference. We could probably write an action called similar to this increment grand total. Let us say you wanted to also do some background update. Let us say increment send first and then incr. I am going to pick the state here. I think, you can get state and context rather.

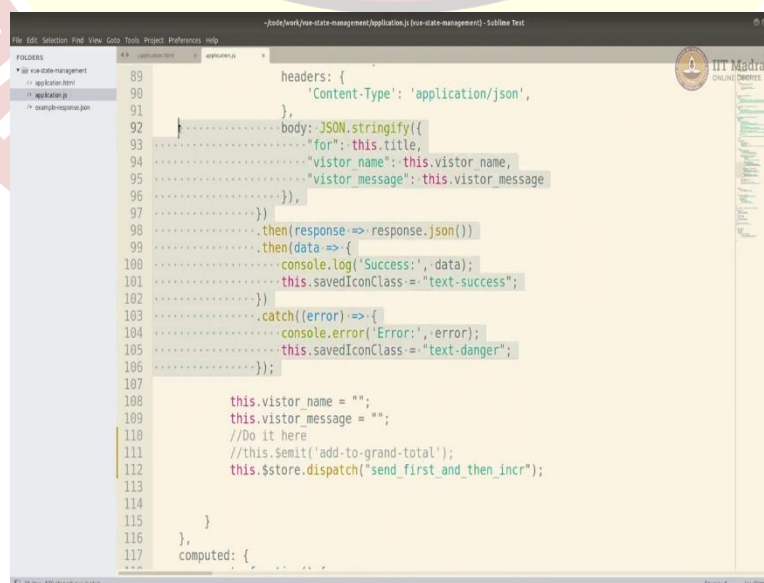
Now, you could do here, for example, first to fetch or an Ajax call. On success, you can call commit. Failure, you do something else. Here also your to actually call, a mutation you cannot ,it should not be directly changing the state, you should actually call the mutation and commit the mutation and then do.

And can use this send first and then increment, while when we are calling in mutation we call it committing, committing the mutation. Similarly, for actions we call it it dispatch, dispatch an action we commit a mutation.

So, you can from somewhere you could probably like we could move this here. For example, I am just this is simple case. I could do this and then you could do some asynchronous stuff, whatever you want to do. That is how you do and then you could, rather you could do context dot commit, because you are getting a context not store.

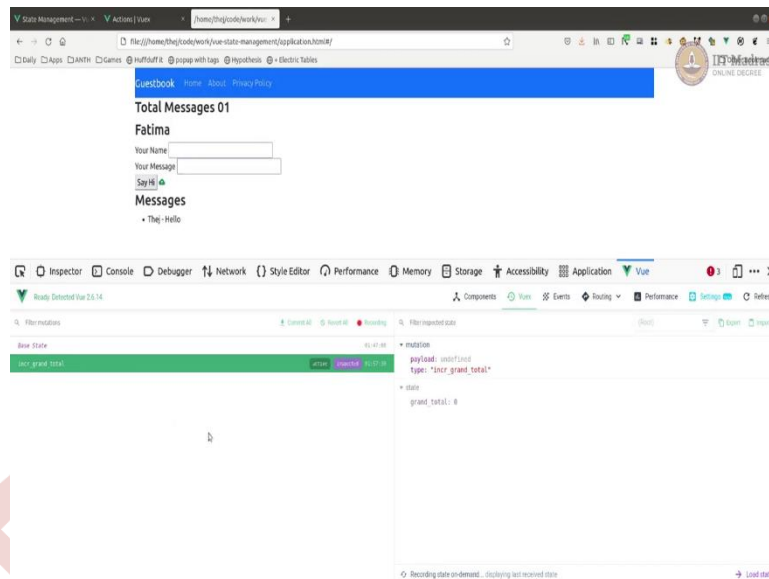
And then instead of calling this as a thing, we could call this by dispatching instead of we were doing here commit we can do a dispatch. You can do this dot dollar store dot dispatch. To dispatch an action and the action can do many things, and then actually commit the grand total.

(Refer Slide Time: 24:57)



```
89     headers: {
90       'Content-Type': 'application/json',
91     },
92     body: JSON.stringify({
93       "for": this.title,
94       "vistor name": this.vistor name,
95       "vistor message": this.vistor message
96     }),
97   },
98   .then(response => response.json())
99   .then(data => {
100     console.log('Success:', data);
101     this.savedIconClass = "text-success";
102   })
103   .catch(error => {
104     console.error('Error:', error);
105     this.savedIconClass = "text-danger";
106   });
107
108   this.vistor name = "";
109   this.vistor message = "";
110   //Do it here
111   //this.$emit('add-to-grand-total');
112   this.$store.dispatch("send_first_and_then_incr");
113
114   }
115 },
116 computed: {
117
```





Now for example, you could have done all of this asynchronous things if required within this action, and then do the commit at the end of it. Like, if you did not want to do it here, you could have just done everything as part of the store. But ideally I would this is like a data. Like, so what I want keeping this is the state the functions that changes the state, the functions that get the state, then those are the functions that can actually do something, and then commit the state or do something with that state.

That is what those are the things, the functions of the things that are very close to the data, I would keep it here. And which are at the high level, probably at the application level. I try not to keep anything that is very specific to a component, that you can still use the component level data to keep it. This will be at application level, so the application you can use or the any of the components can use it is like a shared data. So, that is the end of it.

You can go ahead and check the website and they have some more examples. We have covered most of it, we have covered state, getters, mutations and actions, you know all of it now to use. Now, do you, are you forced to use all the time? Maybe not, but any decent sized application will have a complex data or complex state. And in that case, it is always good to use Vuex store to manage it, which is centralized, and gives you a centralized place where you can write all those functions that change the data.

And also, it gives you a very good debugging tool in the form of Vuex time travel, which will allow you to commit, which allows you to revert and do many things. That will really help you if your application is very complex, and if you have lots of data and playing around with changing the states. Yeah, that is it. Thanks for watching.