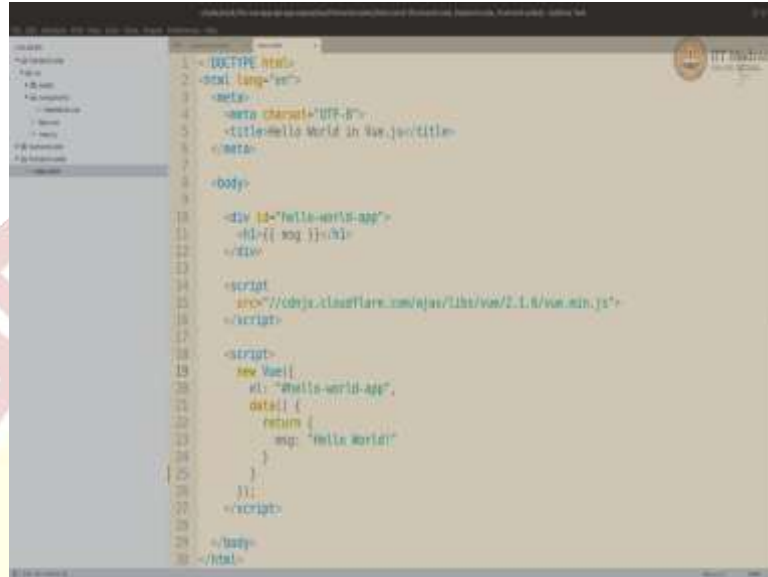




IIT Madras
ONLINE DEGREE

Modern Application Development – II
Professor Thejesh G. N
Software Consultant
Indian Institute of Technology, Madras
How to integrate Vue with Flask - Part - II

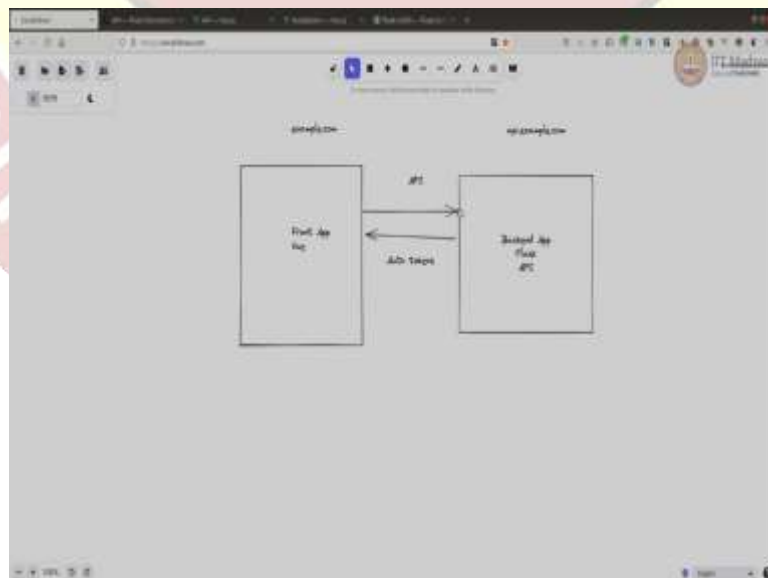
(Refer Slide Time: 00:14)



```
1 <!DOCTYPE html>
2 <html lang="en">
3   <meta>
4     <meta charset="UTF-8">
5     <title>Hello World in Vue.js</title>
6   </meta>
7
8   <body>
9
10    <div id="hello-world-app">
11      <h1 {{ msg }}</h1>
12    </div>
13
14    <script
15      src="//cdn.jsdelivr.net/npm/vue/dist/vue.min.js">
16    </script>
17
18    <script>
19      new Vue({
20        el: "#hello-world-app",
21        data: {
22          return {
23            msg: "Hello World!"
24          }
25        }
26      })
27    </script>
28
29  </body>
30 </html>
```

Welcome to the Modern Application Development –II screencast. In this screencast we will see, how to separate the frontend and backend application and how do we go about deploying it and what are the complexities involved and what are the advantages.

(Refer Slide Time: 00:31)



Let me just explain, how it is going to be set up. Assume, we have a front-end app; we have a back-end app, and let us say our front-end app is a flash cap and we just has APIs. Just a

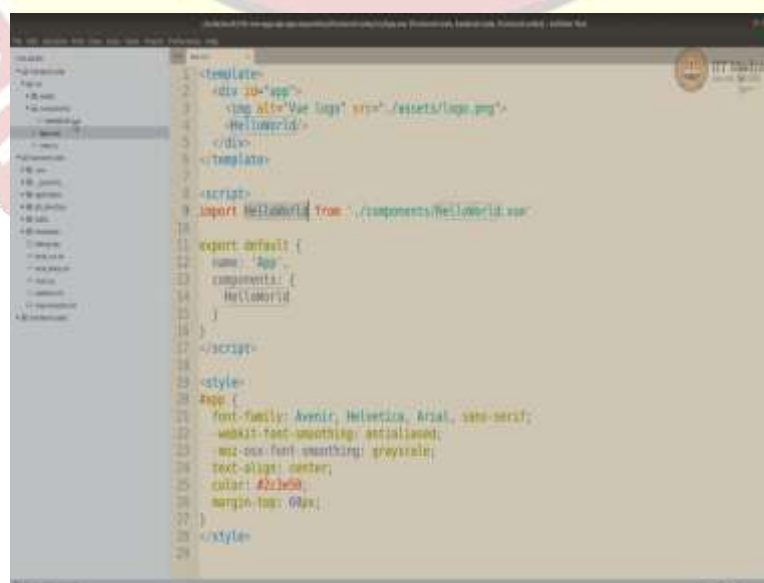
restful API. And this is a backend app, written in Vue and when it is deployed, let us, assume these two run on different domains or it could even be that you are building just a front-end app and someone else is built and delivering the back-end app.

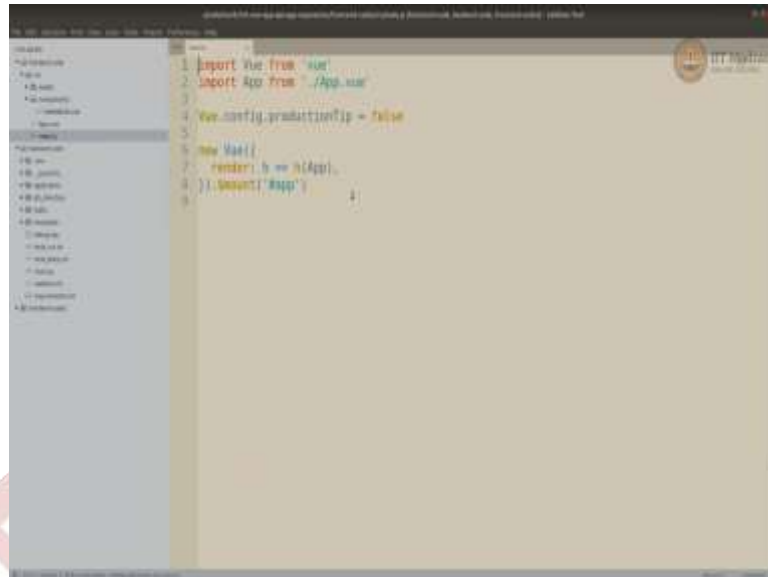
For example, you are using a third-party APIs then the back-end APIs becomes a completely different domain. It is similar to that in in terms of concept. For example, it could be, this could be on API dot example.com. And this could be at example.com. All practical purposes they are different domains. So, they will have different security settings in terms of cookie sharing, cost etc. We will see how to handle that.

How to deploy that in a simpler way and how they interact. Now, this is not back end this is front end. And this is the back end. Just to be clear. So, and then they interact between those using APIs, and using Auth tokens, authentication using Auth tokens, because they are not on the same server, they cannot use cookie authentication.

And usually in such cases, there is no server-side rendering, all the rendering happens on the front side, front-end and the back-end is job is just to send the data to the front end like a JSON in our case and then the front end should take care of everything. Now, this front-end app in Vue could be based on regular simple HTML Vue or java script one or it could be an app built using CLI or Node dot js. It can be any of those. We will see both of them, just as an example and see how to go about doing it.

(Refer Slide Time: 03:19)

A screenshot of a code editor, likely VS Code, showing a Vue.js component file. The file is named 'App.vue' and is located in a directory named 'src'. The code is written in a light theme. The component has a template section with a div containing a logo and a 'HelloWorld' component. The script section imports 'HelloWorld' from './components/HelloWorld.vue' and exports a default object with a 'name' of 'App' and a 'components' object containing 'HelloWorld'. The style section defines a 'App' class with various CSS properties like font-family, text-align, and color. The editor's sidebar on the left shows a file explorer with a tree view of the project structure, including 'src' and 'components' folders. The status bar at the bottom indicates the file is 'App.vue' and is part of a 'workspace'.



```
1 import Vue from 'vue'
2 import App from './App.vue'
3
4 Vue.config.productionTip = false
5
6 new Vue({
7   render: h => h(App),
8 }).$mount('#app')
```

We are in the folder called source because it is in the source folder. Currently we are not going to build it; we are just going to run it. So, we are going to run it using Vue command line. To see what all the features available we can do Vue help we are going to use this, use serve main dot. Our main code is in main. So, we dot main. Vue serve, no, we are not going to build, we are just going to serve. So, let us start this let us see what port it starts. So, it is pulling out and building all the things that is required to show and it will start a dev server, might take couple of minutes.

(Refer Slide Time: 04:48)



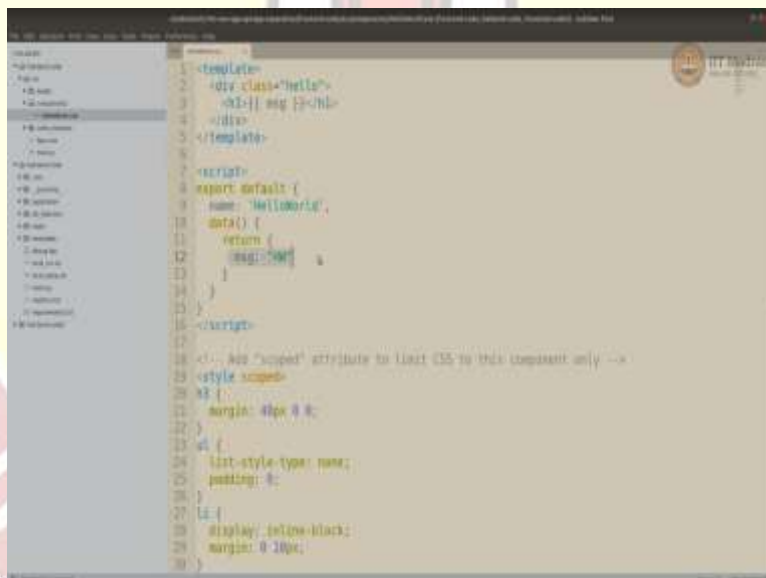
```
vue serve
  http://localhost:8080/
  https://192.168.1.100:8080/

Note that the development build is not optimized.
To create a production build, run 'vue build'.
```



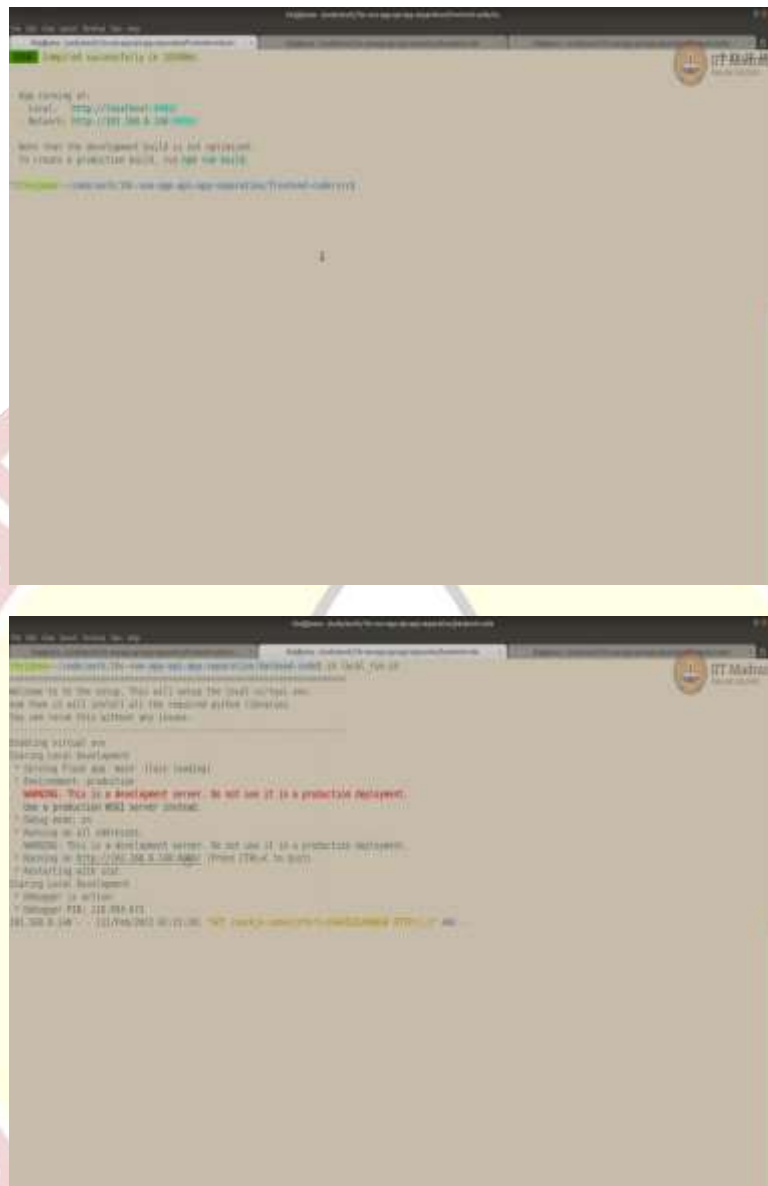
Because it first time it takes time to download the packages and build and all of those. So, let us go to 8080. Local host 8080. So, you can see that it is doing HW.

(Refer Slide Time: 05:13)



Now, we want to do something instead of printing the message hard coded message, want to pull this message from the back end. We will do that.

First, let us run the backend and see whether the API is returning the message. I am going to stop this. I am going to the back-end code. I have done already the local setup. So, sh local run, oh, this is also running 8080. So, we will run the front end in a different port, anyway.



(Refer Slide Time: 05:43)

```
17 create_article_likes_parser = response.RequestParser()
18 create_article_likes_parser.add_argument('user_id')
19 create_article_likes_parser.add_argument('article_id')
20
21 class ArticleLikesAPI(Resource):
22     @auth_required('token')
23     def post(self):
24         args = create_article_likes_parser.parse_args()
25         article_id = args.get('article_id', None)
26         user_id = 1 # current user
27         # Do all the validation
28         new_like = ArticleLikes(user_id=user_id, article_id=article_id)
29         db.session.add(new_like)
30         db.session.commit()
31         return marshal(new_like, article_likes_resource_fields)
32
33 test_api_resource_fields = {
34     'msg': fields.String,
35 }
36
37 class TestAPI(Resource):
38     @auth_required('token')
39     def get(self):
40         return marshal({'msg': 'Hello World from TestAPI'}, test_api_resource_fields)
41
42
```

I am going to open 8080 and open our simple API that we had done earlier for testing called test API, which I am just going to initially comment out the authentication just to check whether it is working. So, it should print Hello World from test API if I access.

(Refer Slide Time: 06:02)

```
21 app.logger.info("App Setup complete")
22 # Setup Flask Security
23 user_datastore = SQLAlchemyUserDatabase(db.session, User, Role)
24 security = Security(app, user_datastore)
25 # user_datastore.create_user(username='theshiph', email='theshiph@com', password='password')
26 # db.session.commit()
27 api = Api(app)
28 app.add_url_rule('/api/test', view_func=test_api.get, methods=['GET'])
29 return app, api
30
31 app, api = create_app()
32
33 # Import all the controllers as they are named
34 from application.controllers import *
35
36 # Add all restful controllers
37 from application.api import ArticleLikesAPI
38 api.add_resource(ArticleLikesAPI, '/api/article_likes', '/api/article_likes/<int:article_id>')
39
40 from application.api import TestAPI
41 api.add_resource(TestAPI, '/api/test')
42
43 @app.errorhandler(404)
44 def page_not_found(e):
45     # note that we set the 404 status explicitly
46     return render_template("404.html"), 404
47
```

And the path would be, it will be slash API slash test.

(Refer Slide Time: 06:08)



Let us go here, local host slash API slash test, it is not 8000 it is 8080. It seems to be returning Hello World test API. I am going to close this and restart our front-end application.

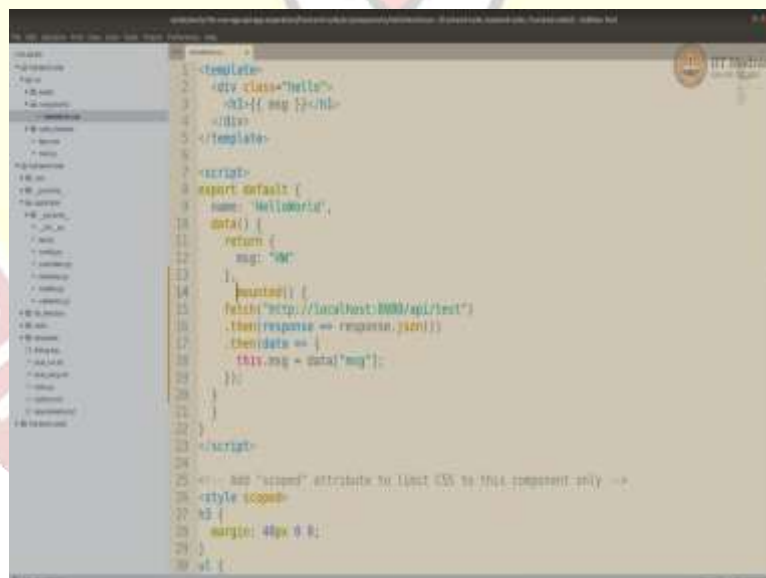
(Refer Slide Time: 06:29)





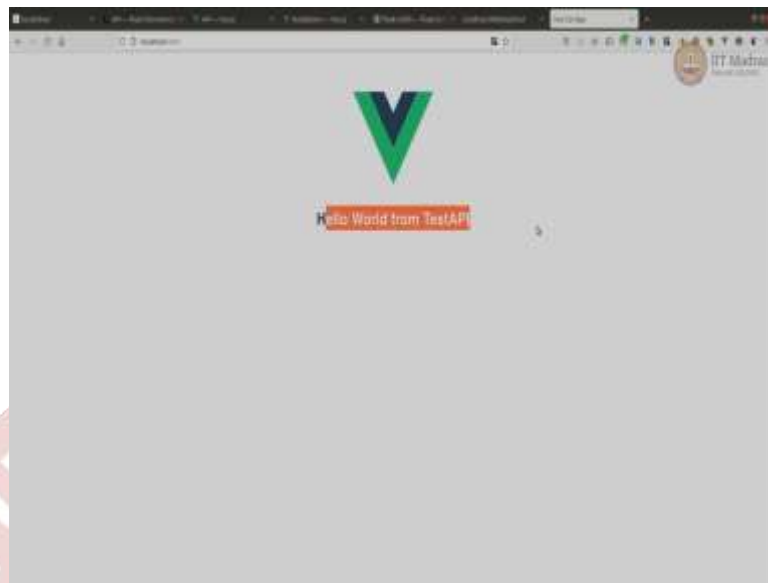
When I restart now it will start on a different port because port 8080 is already occupied by our backend. So, it is building give it a second, it is running on 8081. Copy link, now, I will paste this here. So, it is doing HW. Now, I want from this URL called this API to show this message on this. We know that is pretty straightforward.

(Refer Slide Time: 06:33)



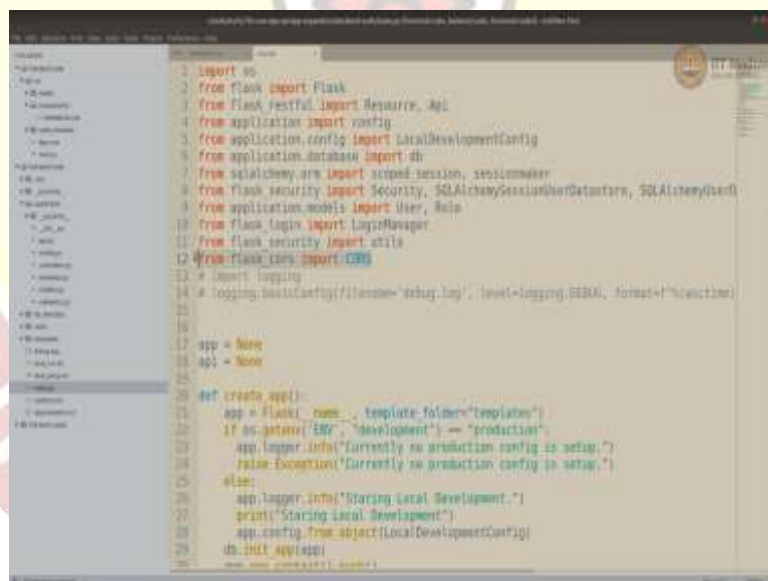
We can go to our app and go to this component and we can just unmount, we can call the API and set the data. If you see the API response, the message is inside a key message okay. So, if I do mounted on mount, this is a life cycle function, mount, it will call this API, which is localhost 8080 API test. When it returns data, I want this dot message to be the message. It is closed, closed, closed.

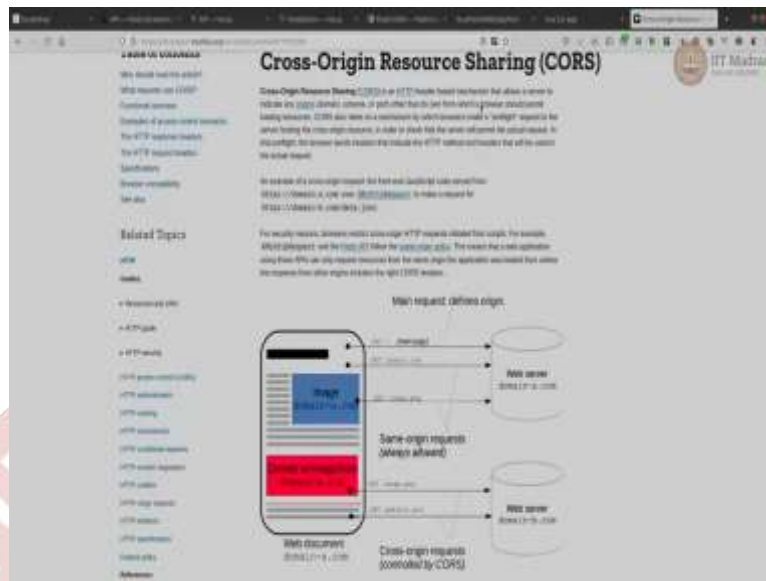
(Refer Slide Time: 07:55)



So, now, when I go refresh, it is just an extra bracket. Now, we can see it is come from Hello World from test API. So, it is calling localhost 8080 from 8081 and calling that.

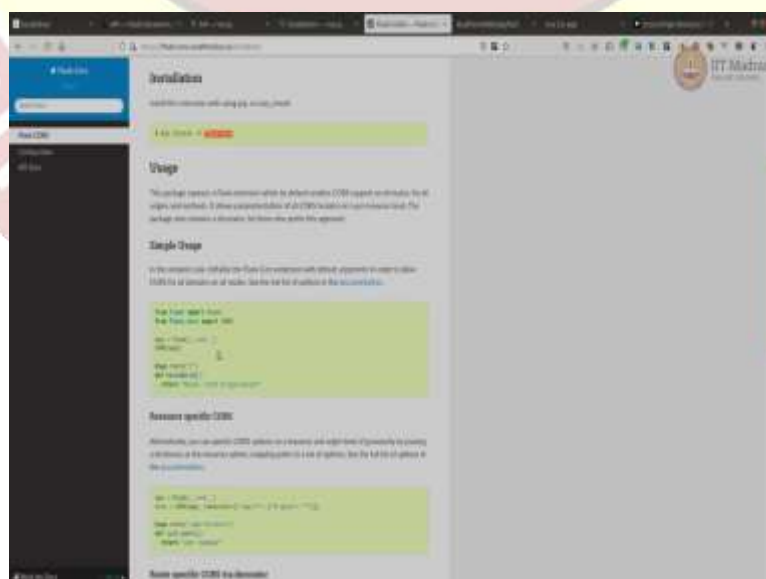
(Refer Slide Time: 08:20)





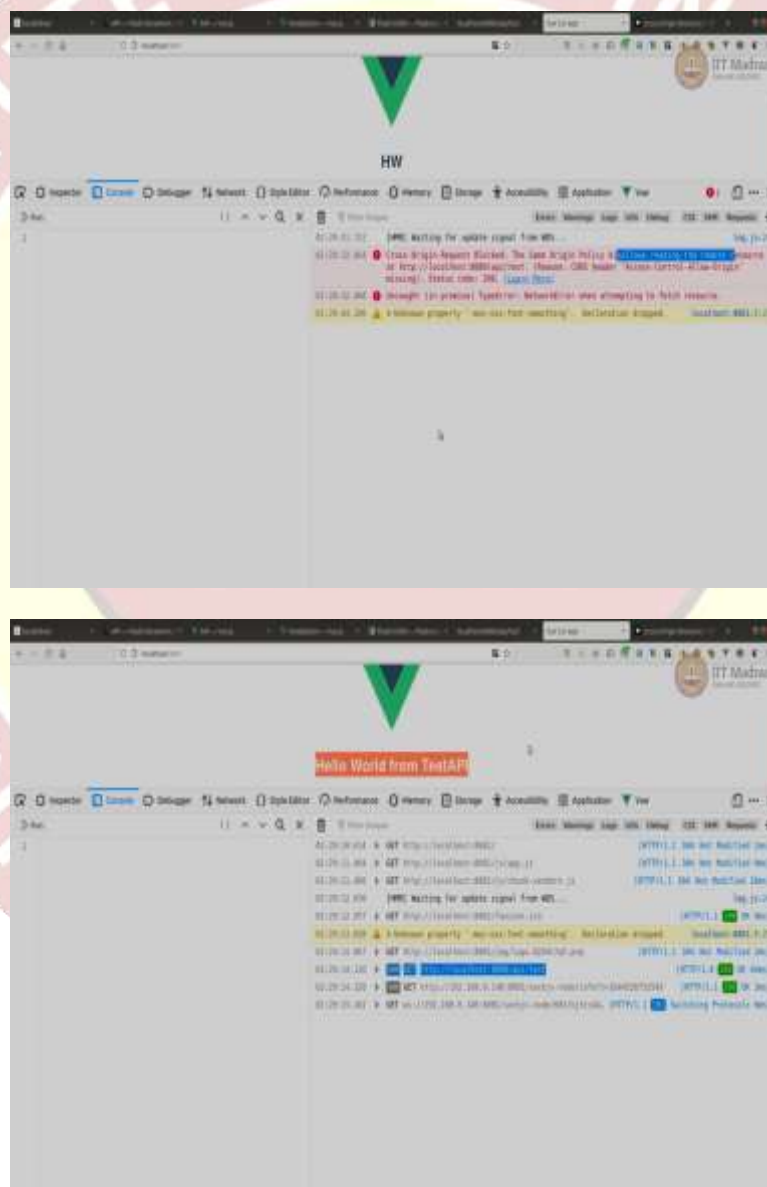
Let us see, what else we are done? We have done one more step here, to do something called CORS. CORS is you can search for CORS mdn CORS. So, it is a way of sharing the resources across origin or like across the domain, you can read about that. Now, origin in this webpage website is 8081 but it is trying to access from here, it is trying to call to 8080 API. So, that is across the origin or a different domain. So, it will throw an error if you do not allow on the server side. To allow on the server side, you have to input CORS and then add app to CORS.

(Refer Slide Time: 09:15)



You can read more about that here. You can actually enable it to specific domains. Now, currently we have by doing this, we have enabled from any other domain can I call this API through the browser without any restrictions, usually we do not do that it is not very secure. If you know that your app is running on a specific domain, your front-end app is running on a separate domain specific domain, we only allow request from that domain. So, we can do that specific domain allocation. You can do this origin instead of star you can tell specific. Now, what happens if I had not done this? Let us say I will mask this, let us see what happens?

(Refer Slide Time: 10:08)

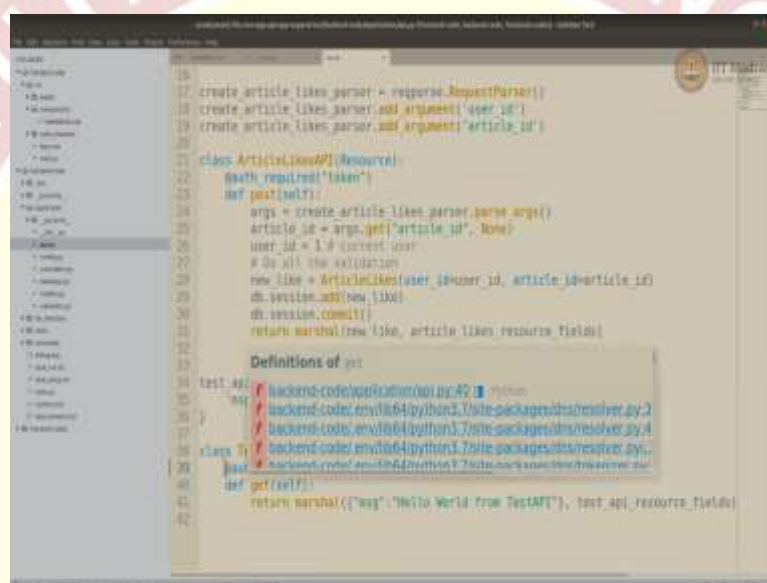


I will just restart if it is not. Let us go back to our and refresh this. So, it is not, it is not done the thing, got the message from the backend. You can check the console here to see what is happening. You can see that cross-region request blocked. Same urgent policy disallows

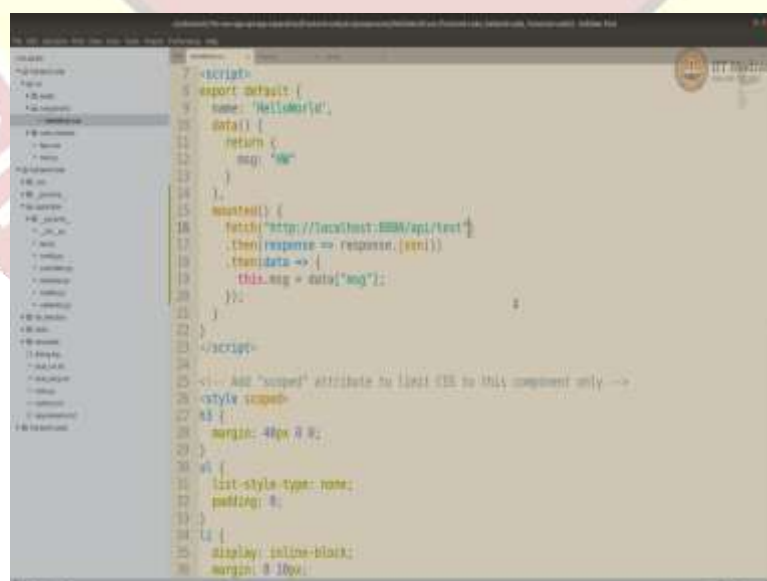
reading of the remote resource at that location from here, that is why it is not getting data and hence it is not refreshing.

Now, as soon as I add this back and then refresh this goes away. I have been able to access. You can see that I have been able to access that as I request and then I have been able to access. Get the message and show the message. This is very important to understand. Similarly, if authentication was enabled, you could not use the cookie you have to use the token. Let us check. So, let us go back to, where our API.

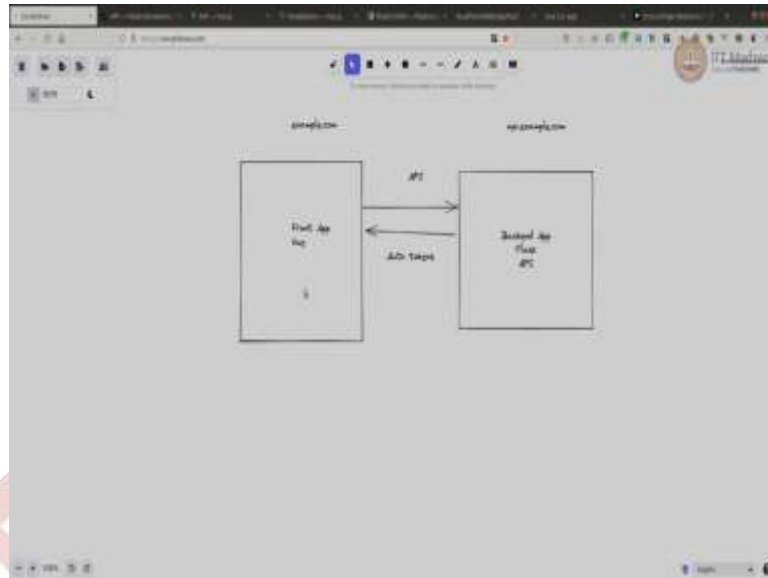
(Refer Slide Time: 11:38)



```
16 create_article_likes_parser = response.RequestParser()
17 create_article_likes_parser.add_argument('user_id')
18 create_article_likes_parser.add_argument('article_id')
19
20 class ArticleLikes(Resource):
21     @auth_required('token')
22     def post(self):
23         args = create_article_likes_parser.parse_args()
24         article_id = args.get('article_id', None)
25         user_id = 1 # current user
26         # Do all the validation
27         new_like = ArticleLikes(user_id=user_id, article_id=article_id)
28         db.session.add(new_like)
29         db.session.commit()
30         return marshal(new_like, article_likes_resource_fields)
31
32
33
34 test_api =
35
36
37
38 class TestAPI(unittest.TestCase):
39     def setUp(self):
40         self.app = create_app()
41         self.client = TestClient(self.app)
42         return marshal(("msg": "Hello World from TestAPI"), test_api_resource_fields)
```



```
7 <script>
8   export default {
9     name: 'HelloWorld',
10    data() {
11      return {
12        msg: 'Hi'
13      }
14    },
15    mounted() {
16      fetch('http://localhost:8080/api/test')
17        .then(response => response.json())
18        .then(data => {
19          this.msg = data['msg'];
20        });
21    }
22  }
23 </script>
24
25 <!-- Add "scoped" attribute to limit CSS to this component only -->
26 <style scoped>
27   # {
28     margin: 40px 0 0;
29   }
30   # {
31     list-style-type: none;
32     padding: 0;
33   }
34   # {
35     display: inline-block;
36     margin: 0 10px;
37   }
38 </style>
```

If I enable this auth token and then in my call to the my call to the API, I have to set up credential and send the auth token in the header etc. etc. I have shown that how to do that in a separate screencast. You can have a look at how to do that. So, this is how it is usually done. This has an advantage here like you have two separate projects some front-end team can work on the front-end, back-end team can work on the back-end. And it can be kept separate and the same back-end can be used by say front-end code, which is for web and also an android or an ios app.

So, the back-end same back-end code can be used or APIs can be used by both. So, it kind of keeps them separate, if the front-end code becomes big and large. So, it is a separate project it can be maintained well. Now, while deploying your halls it is not that you always have to deploy it separately, like the way with this saw here, it does not have to be always, you can always build this push it as part of a static here and serve it.

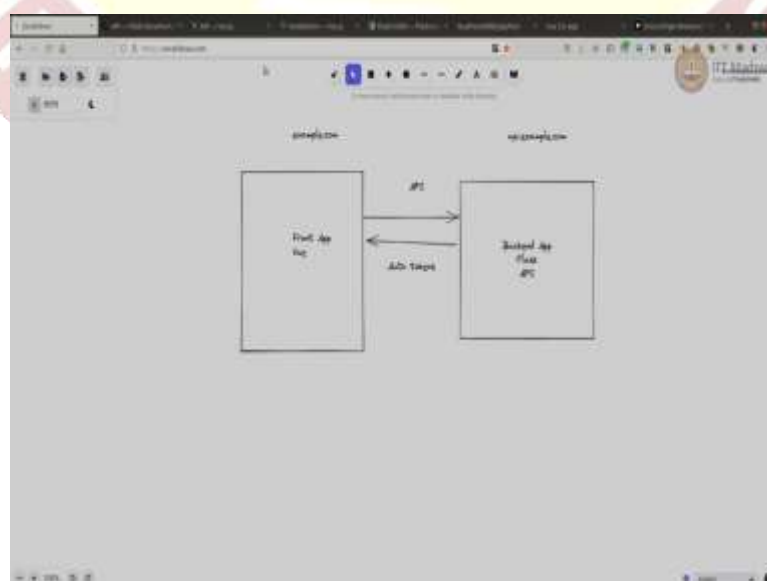
(Refer Slide Time: 13:06)

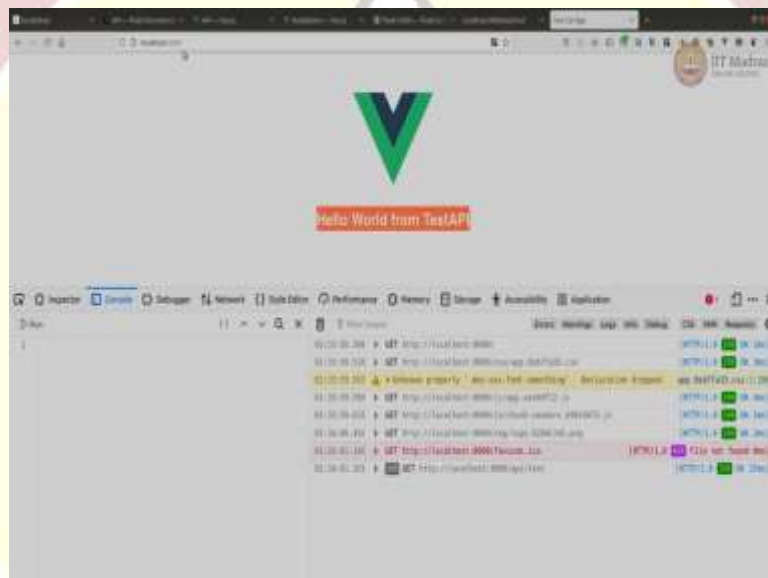
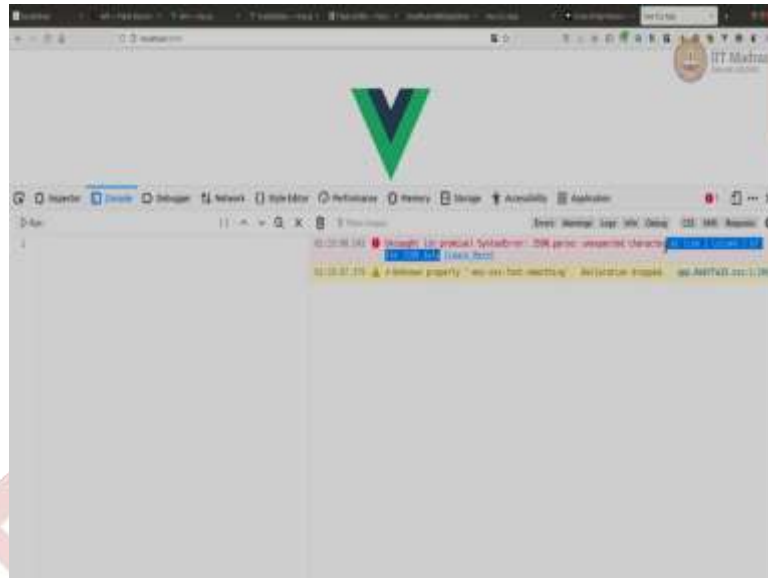
```
nodejs -v
v10.16.0

npm run build
node build/index.js

node build/index.js --help
node build/index.js --help
```

```
node build/index.js --help
node build/index.js --help
```





So, let us see that, let us build it. Let us say `Vue help` and then there will be `Vue build`, `Vue build` and `main dot js`. So, now it is going to build and create a distribution folder for the production deployment, usually this is how you do, you do not run it that was a development server. It might take couple of minutes. So, when you build it for the first time. It is done. So, everything is inside the `dist` folder.

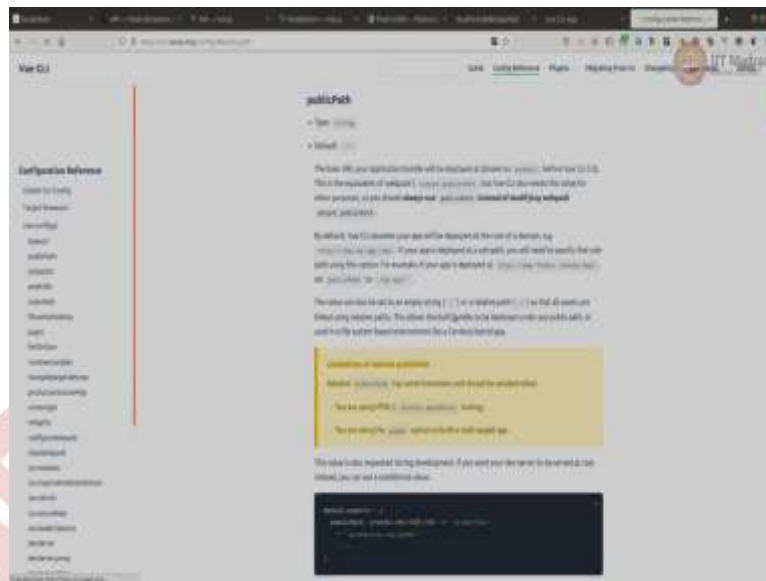
Now, when you are deploying it if it was a separate setup in the sense if you are going for the same thing there would have been a server running here you would have been Python3 I am going to start simple http server minus it `http dot server`. So, it is running at 8000. So, copy link, oh, it is not localhost, should be fine. Let us see.

(Refer Slide Time: 15:54)





(Refer Slide Time: 19:06)



And there is one more thing that you might have to do in if you change that, which is setting the path. Let me just get you the name of it. It is called public path. So, this is what you have to do, while building. If you are going to use a separate path which is not a dist path. Because it assumes that, you are going to be serving from here, but if you are not then you will have to give the specific path here. We are actually serving from dist. So, that is why. But you could make it also simpler. Let us do that. Let us say in the source, we can change the config to do dist path and let us build that again.

(Refer Slide Time: 19:58)

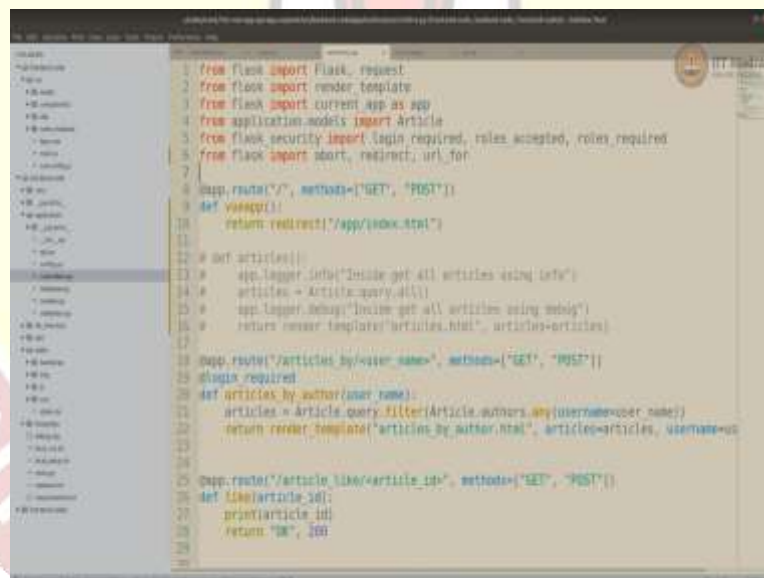


We can just add a Vue config, new file and then we can just. So, let us not call it dist in our thing, let us call it app, just to be a nicer. It is not for production we will just remove this for now, will be app. So, I will make a similar change in the main, even though it is dist folder we will call it app.

Now, let us stop this. I am going to delete this old list. I am going to rebuild this, that settings change, rebuilding. Now, it will build with this setting of Vue config where we are telling it that we are serving from slash app. It is built. Now, I am going to go back from front end. So, source and test.

I am going to go to backend, paste it here. So, that I can serve it as part of flask. Now I am going to start the flask app again. Now, let us see localhost 8000 app. Is it 8000 or 8080. Let us see this app is correct. Here app is correct here this dist it has to be index. So, it is working. Otherwise you could also do like a forwarding, using the main dot controller.

(Refer Slide Time: 22:57)

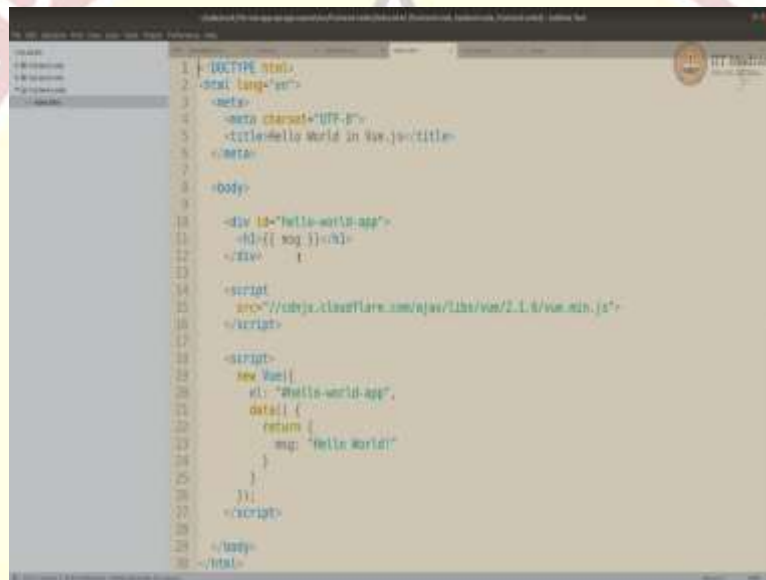
A screenshot of a code editor window displaying Python code for a Flask application. The code is organized into several sections: imports, a root route, an 'articles' endpoint, an 'articles by author' endpoint, and an 'article like' endpoint. The code uses Flask's request and render_template functions, and interacts with a database using SQLAlchemy's query and filter methods. The editor has a dark theme and a sidebar on the left showing a file explorer.

```
1 from flask import Flask, request
2 from flask import render_template
3 from flask import current_app as app
4 from application.models import Article
5 from flask_security import login_required, roles_accepted, roles_required
6 from flask import abort, redirect, url_for
7
8 app.route("/", methods=["GET", "POST"])
9 def vindex():
10     return redirect("/app/index.html")
11
12 # def articles():
13 #     app.logger.info("Inside get all articles using info")
14 #     articles = Article.query.all()
15 #     app.logger.debug("Inside get all articles using debug")
16 #     return render_template("articles.html", articles=articles)
17
18 app.route("/articles by/author-name", methods=["GET", "POST"])
19 @login_required
20 def articles_by_author(user_name):
21     articles = Article.query.filter(Article.authors.any(username=user_name))
22     return render_template("articles by author.html", articles=articles, username=user_name)
23
24 app.route("/article like/article_id", methods=["GET", "POST"])
25 def like_article_id():
26     print(article_id)
27     return "OK", 200
```


If I had was running separately it gives an opportunity to not only code by separate, but also running and deployment separate. So, you one or like since Vue app front-end app is completely static, you can deploy it on a cdn which is super-fast and you can run the back end on a flask on a separate server. So, code is separate and deployment is also separate.

In the second case, code is separate you are coding separately but at the time of deployment, what we will do is we will deploy only one, which will copy from dist into the flask and then use it. In this case it gives an advantage of it is on the same server. So, cookies can be used but if you deploy separately you can use it. You have to use auth tokens.

(Refer Slide Time: 26:40)



```
1 <!DOCTYPE html>
2 <html lang="en">
3   <meta>
4     <meta charset="UTF-8">
5     <title>Hello World in Vue.js</title>
6   </meta>
7
8   <body>
9
10    <div id="hello-world-app">
11      <h1 {{ msg }}</h1>
12    </div>
13
14    <script
15      src="//cdn.jsdelivr.net/npm/vue/dist/vue.min.js">
16    </script>
17
18    <script>
19      new Vue({
20        el: "#hello-world-app",
21        data: {
22          return {
23            msg: "Hello World!"
24          }
25        }
26      })
27    </script>
28
29  </body>
30 </html>
```

There is also a piece of code that I had written in the standard way, you do not have to use CLI to do all of that, you could also simply write like a html and you can start a server and call it. And even that should work.

(Refer Slide Time: 27:00)



```
index.js
// REST client for REST API
const axios = require('axios');
const url = 'http://localhost:8080/api/v1/users';

// GET method
const get = async () => {
  try {
    const response = await axios.get(url);
    console.log(response);
  } catch (error) {
    console.log(error);
  }
};

// POST method
const post = async (data) => {
  try {
    const response = await axios.post(url, data);
    console.log(response);
  } catch (error) {
    console.log(error);
  }
};

// PUT method
const put = async (data) => {
  try {
    const response = await axios.put(url, data);
    console.log(response);
  } catch (error) {
    console.log(error);
  }
};

// DELETE method
const deleteMethod = async (id) => {
  try {
    const response = await axios.delete(`${url}/${id}`);
    console.log(response);
  } catch (error) {
    console.log(error);
  }
};

// Main function
const main = async () => {
  // GET method
  get();

  // POST method
  post({name: 'John', age: 20});

  // PUT method
  put({name: 'John', age: 20});

  // DELETE method
  deleteMethod(1);
};

main();
```

Starting local development
Debugger PID: 128, 861-472

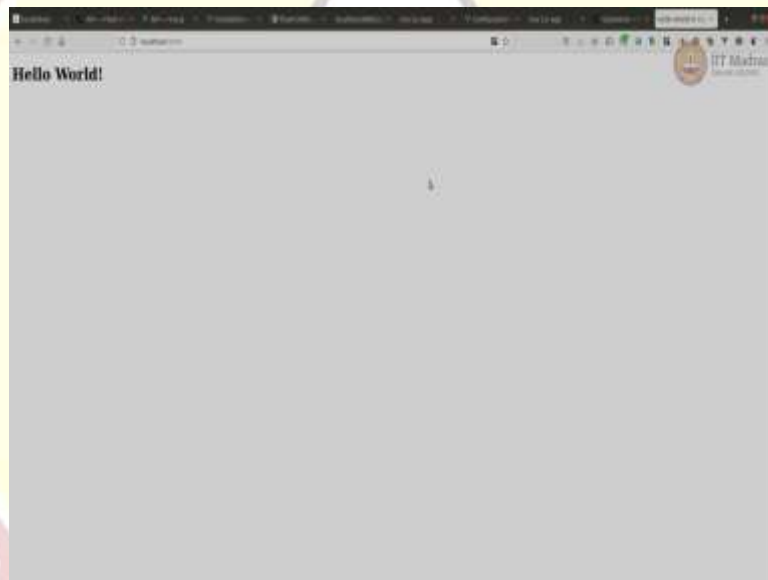
127.0.0.1 - [11/May/2022:02:40:33] "GET / HTTP/1.1" 200 -
127.0.0.1 - [11/May/2022:02:40:33] "GET / HTTP/1.1" 200 -
127.0.0.1 - [11/May/2022:02:40:34] "GET / HTTP/1.1" 200 -
127.0.0.1 - [11/May/2022:02:40:34] "GET / HTTP/1.1" 200 -
127.0.0.1 - [11/May/2022:02:40:34] "GET / HTTP/1.1" 200 -
127.0.0.1 - [11/May/2022:02:40:34] "GET / HTTP/1.1" 200 -
127.0.0.1 - [11/May/2022:02:40:34] "GET / HTTP/1.1" 200 -

Browser: Chrome 102.0.5009.102
URL: http://localhost:8080/


Hello World from TestAPI

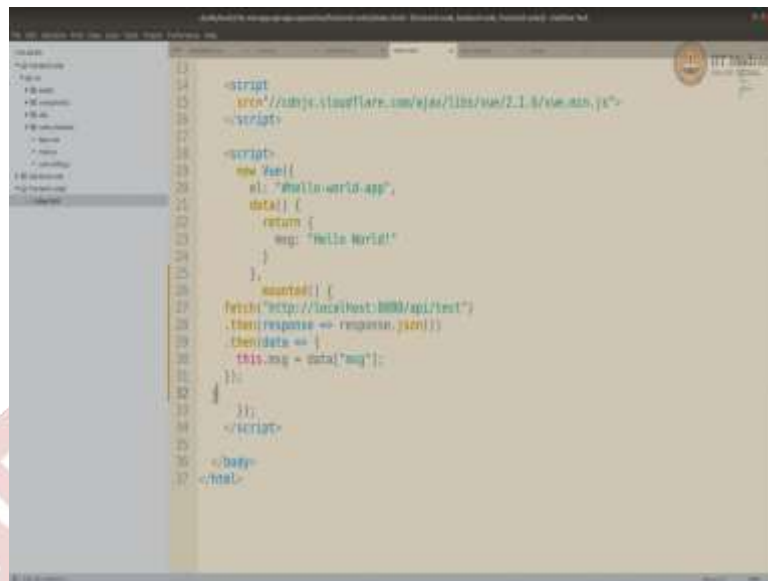
Let me just show that also. Let me just stop, this is stopped, okay. This let this run because we want the API. API is accessible. Now, instead of calling 8080, I will call I will start another server in the code part two.

(Refer Slide Time: 27:20)



This is all simple static server. Python 3 minus m http dot server. Now, we do not have any build process anything here, it is just a simple html and we are just running it. Call host 8000. It just shows Hello World nothing special, because that is what we have written here. Just Hello World.

(Refer Slide Time: 28:02)



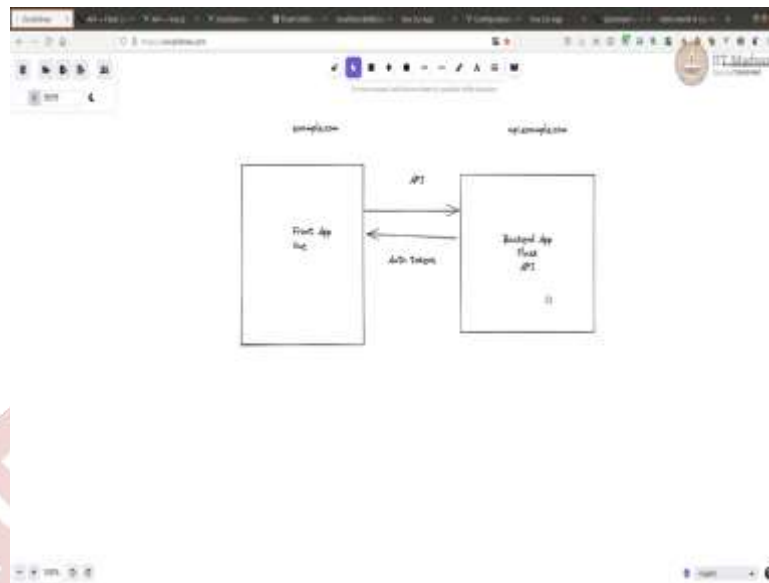
```
13  
14 <script  
15   src="//cdn.jsdelivr.net/npm/vue/dist/vue.min.js">  
16 </script>  
17  
18 <script>  
19   new Vue({  
20     el: '#hello-world-app',  
21     data() {  
22       return {  
23         msg: 'Hello World!'  
24       };  
25     },  
26     mounted() {  
27       fetch('http://localhost:8080/api/test')  
28       .then(response => response.json())  
29       .then(data => {  
30         this.msg = data['msg'];  
31       });  
32     },  
33   });  
34 </script>  
35 </body>  
36 </html>
```



Now, you can go ahead and attach the same similar mounted thing to load it from the remote. This ends here. And now, I just have to refresh and I should get the message from the remote server and I should be able to load it and run it. Here the difference is, this is like does not use any java script or like a node dot js thing or CLI or anything.

This can be still put into a separate project and can be actually deployed separately, just like how we deployed the other one. So, both, regular simple apps with using just html and java script can also be deployed in a similar way. Or if you use Vue CLI, also, you can deploy it that way or you can combine too.

(Refer Slide Time: 29:04)



So, that is all actually, we have seen all methods now, in a complex way of deployment structures but in either really big applications and big teams usually they try to keep both deployment and code separate and based on the complexity of the front end, they could use CLI or just use html and java script that is again left developer is choice. So, these are the other set of architecting your app, structuring your code and deploying your code. Thank you for watching.