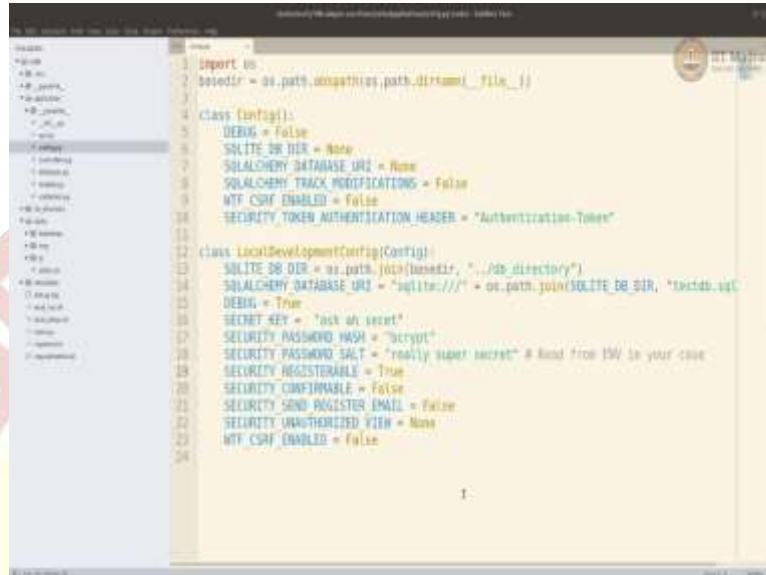




IIT Madras
ONLINE DEGREE

Modern Application Development – 2
Professor Thejesh G. N
Software Consultant
Indian Institute Technology, Madras
How to Integrate Vue with Flask – Part 1

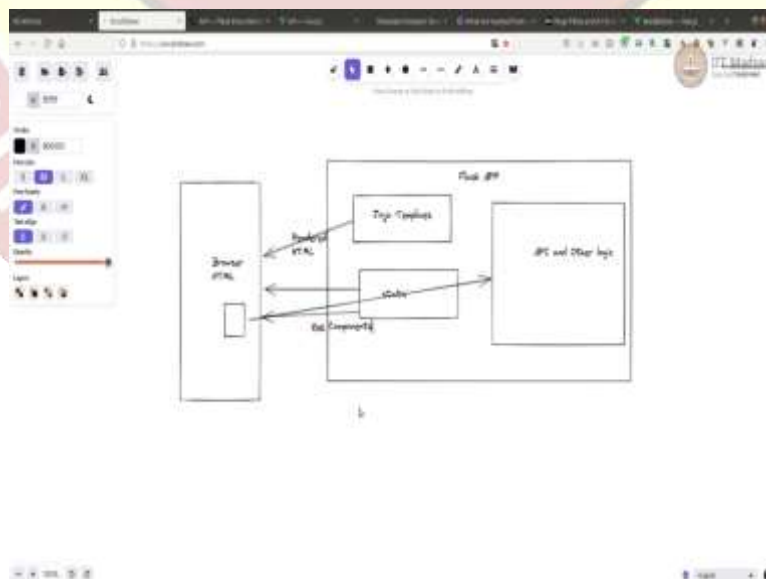
(Refer Slide Time: 00:14)



```
1 import os
2 basedir = os.path.abspath(os.path.dirname(__file__))
3
4 class Config:
5     DEBUG = False
6     SQLITE_DB_DIR = None
7     SQLALCHEMY_DATABASE_URI = None
8     SQLALCHEMY_TRACK_MODIFICATIONS = False
9     WTF_CSRF_ENABLED = False
10     SECURITY_TOKEN_AUTHENTICATION_HEADER = "Authentication-Token"
11
12 class LocalDevelopmentConfig(Config):
13     SQLITE_DB_DIR = os.path.join(basedir, "..db_directory")
14     SQLALCHEMY_DATABASE_URI = "sqlite:///{}{}".format(os.path.join(SQLITE_DB_DIR, "testdb.sql"),
15     DEBUG = True
16     SECRET_KEY = "ask oh secret"
17     SECURITY_PASSWORD_HASH = "bcrypt"
18     SECURITY_PASSWORD_SALT = "really super secret" # Read from ENV in your code
19     SECURITY_REGISTERABLE = True
20     SECURITY_CONFIRMABLE = False
21     SECURITY_SEND_REGISTER_EMAIL = False
22     SECURITY_UNAUTHORIZED_VIEW = None
23     WTF_CSRF_ENABLED = False
24
```

Welcome to the Modern Application Development 2 screencast. In this screencast, we will see how to set up Vue in a simple flask app. To follow along, you would need a terminal browser, editor and python setup or a flask Project Setup. We are going to use the same flask project that we used before, we are going to improve this same application.

(Refer Slide Time: 00:56)



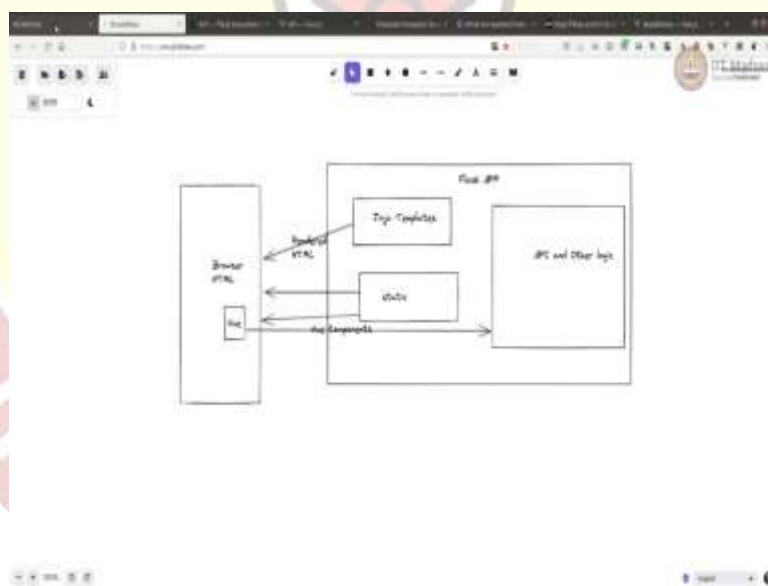
Now, this is a simple approach in which we are going to bring the Vue as a library just like any other library, and we are going to use it along with our HTML or Jinja template. Let me

just write a simple diagram to show. Let us say this is your whole flask app. And within that, you have your Jinja template. And you have your static files. And you have your API's and other things here.

Now API's are accessible, but other logic is not directly accessible. Now, let us say if you have a browser here, when it accesses the URL, what usually happens is your Jinja templates get rendered and sent to front end. Get rendered, and sent to front end as HTML content. And these HTML content can also load some static files, for example, images, JS file, CSS files, etcetera, etcetera.

This is your browser and an HTML page. And now let us say there was a small part within which you are doing some Ajax calls to API's and stuff. You would call APIs and you would use it. This is what like a map. That is simple and we what we have been developing until now. Now, let us say we want to introduce Vue into it. In a simple approach, Vue sits along with your other static files, for example, we are going to load along with let us say, JS, and CSS, we are going to also load Vue specific JS and Vue components.

(Refer Slide Time: 03:00)



And you know I am going to remove this part. Let us say you wanted to add activity to this part of the application, then we could use this Vue component here. And then this can make actually call to API is that, we are talking. Now you see it is an hybrid approach, we have server-side rendering of Jinja templates and sending it and then there is a client-side rendering of this Vue component using data from APIs.

This has several advantages and several, some disadvantages or also there are good used cases where this could be a way to do it. For example, if we take the advantages, it is simple. We are going to use Vue components and Vue only where it is required. Otherwise, we will stick it to server side rendered HTML pages. The deployment is simple, because everything is together, you are going to deploy everything together.

And development is also straightforward, because we are going to hire all the code together and it is a single project. So, that if the team is smaller, it becomes easier to maintain and develop and deploy. Now, since all of it lives on the same server, and probably sought through the same URL. The Ajax calls or calls to API's, or fetch calls, can actually be based on cookie-based authentication.

We do not need to go for a token-based authentication. Cookie based authentication is much more simpler to manage than token based authentication. Also, this gives us an opportunity to mix and match. For example, in certain cases, we want to use server-side rendering and, in some cases, we want to use client-side rendering. This gives us an opportunity to mix the way we want to do that.

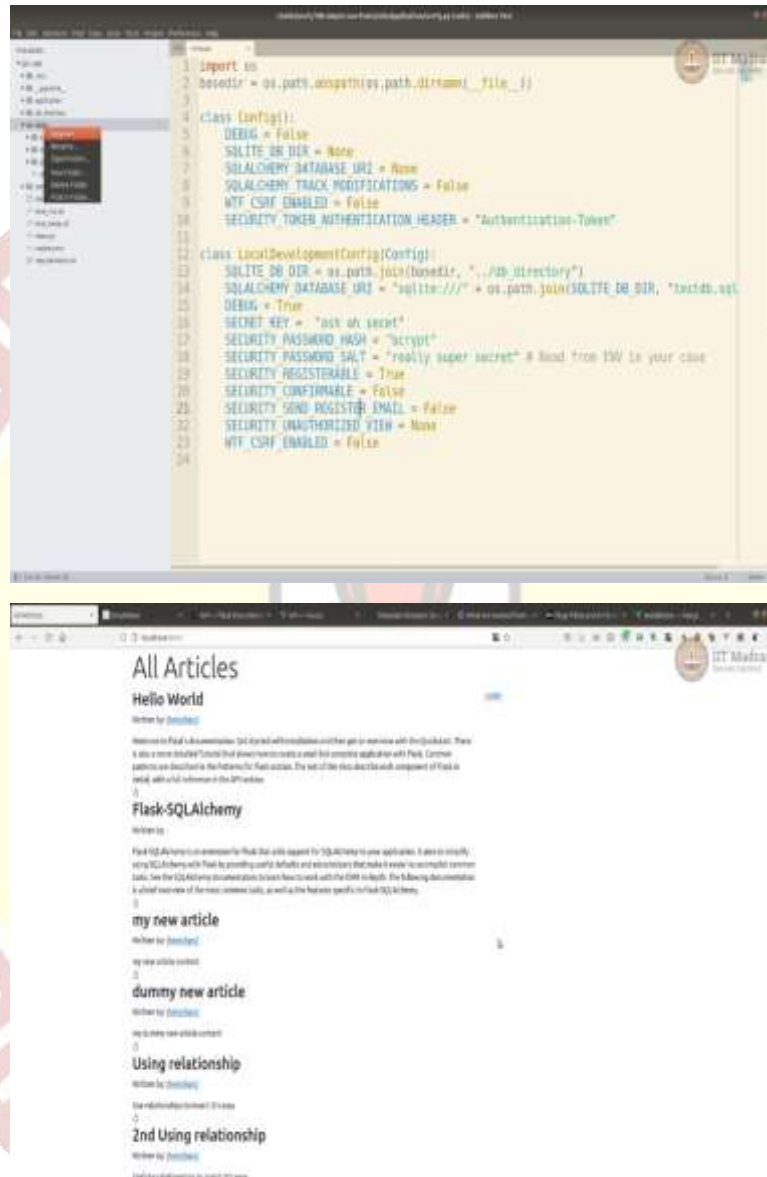
So that is the, those are the advantages. Now, cons is, you know you need to import Vue, in every Jinja template, wherever you want to use it, you have to manually import them, or you have to write a base template of where will you import and all the places where you want to use a Vue. Then all those Jinja templates helps to extend that base template. Some of the CLI tools of Vue, you might bring more optimization and ease of use or ease of development, that will not be available in this case, because we are not going to use CLI in this case.

Now, where is it useful? Where would we go for this approach? This approach is useful for simple applications and where you want everything to live together, or a team, which is small and wants Vue to bit faster, without actually depending on many toolings. And then it is also good for apps, where you had actually at a traditional app, just standard Jinja base, and you want to migrate to a new app setup and you want to do not want to do big bang approach, you want to migrate parts of the app.

Vue also useful where you do not want the app to be activity in the whole app, you want to activity in certain parts of the app in certain widgets of the app. So hence, in that case, it, you can use this. It is also useful, where you do not want to specifically make single page apps. There are cases where single page apps are not ideal. For example, when high accessibility is required, it is preferred to use multiple page apps and add activity only when there is a quiet

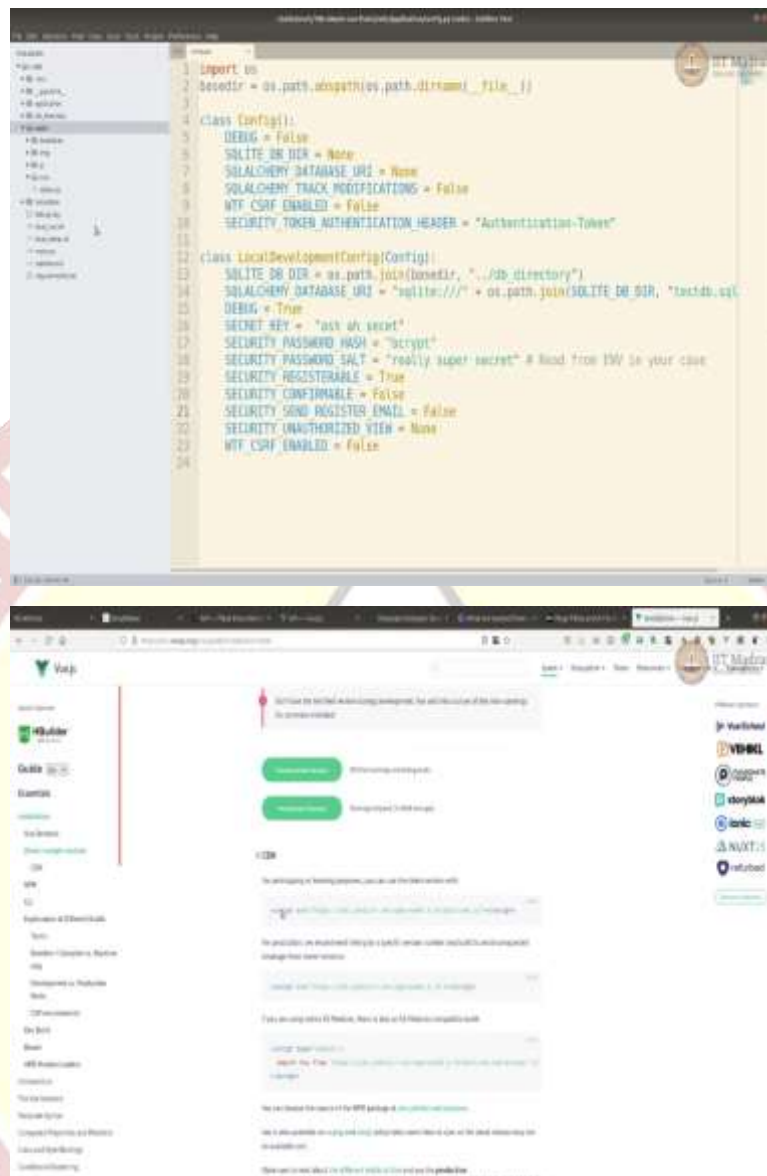
place in the quiet places instead of for the whole app. Maybe it is useful in those cases. So, let us try and do one simple example of wiping all of this together to see how it looks. Let us go back to the project I have already set up and running the project. So, if I go to 8080.

(Refer Slide Time: 07:23)



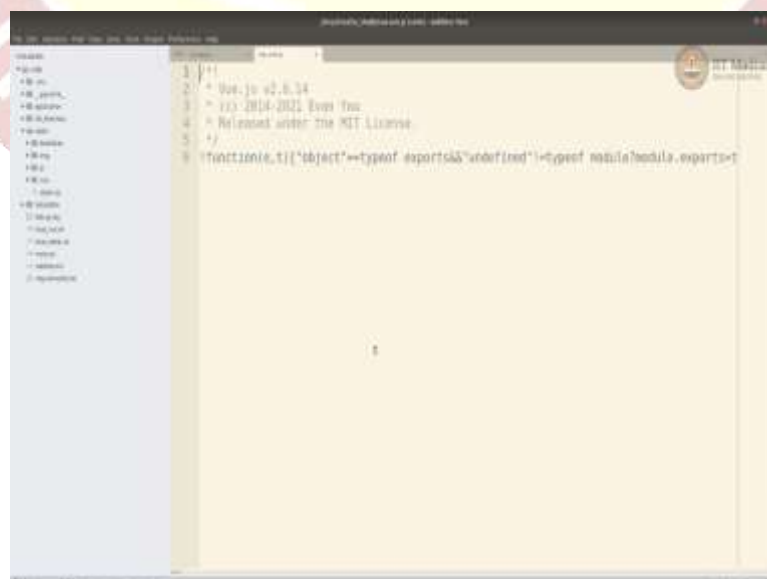
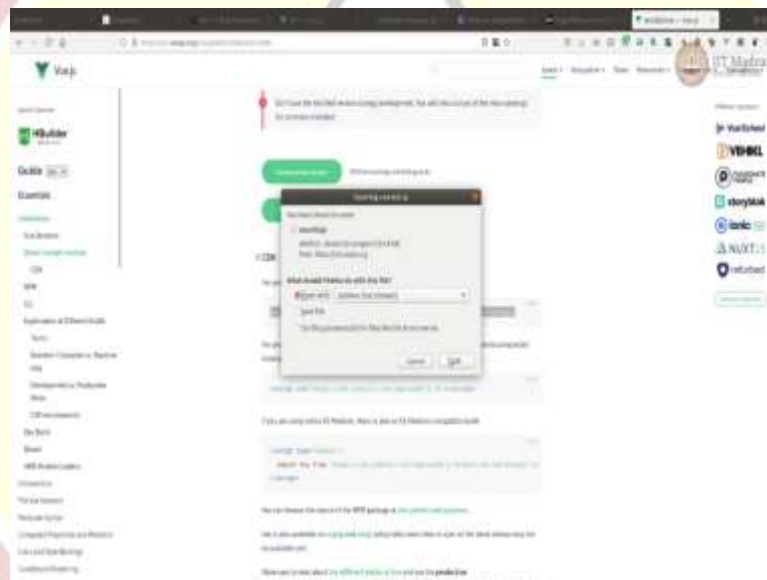
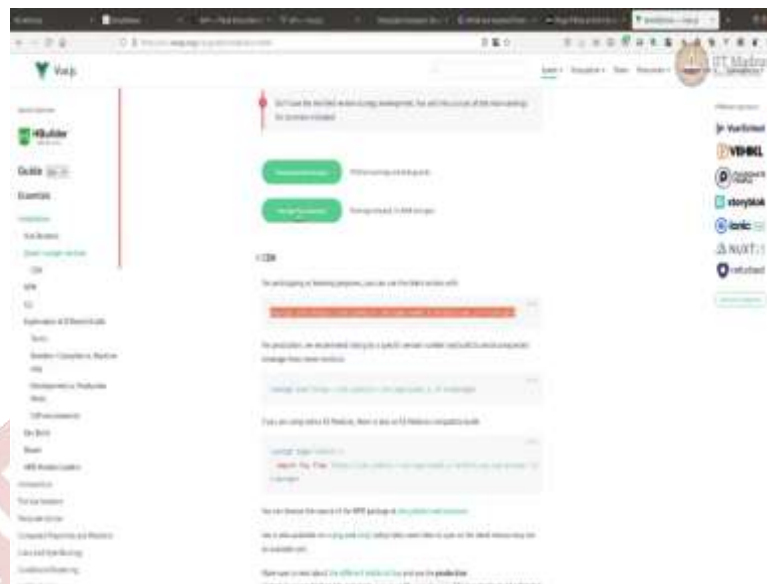
We have the project. Now, let us say we had a sidebar here. And we want to load a widget here, which does something based on Vue, I am not going to do anything special here, I am just going to display a message on the sidebar, within using Vue within using both Jinja template and Vue just to show how it can all be brought together. But then you can say for example, use a specific Vue component for this thumbs up icon only and then you have to code it once and you can just use it everywhere. Things like that, you can do that.

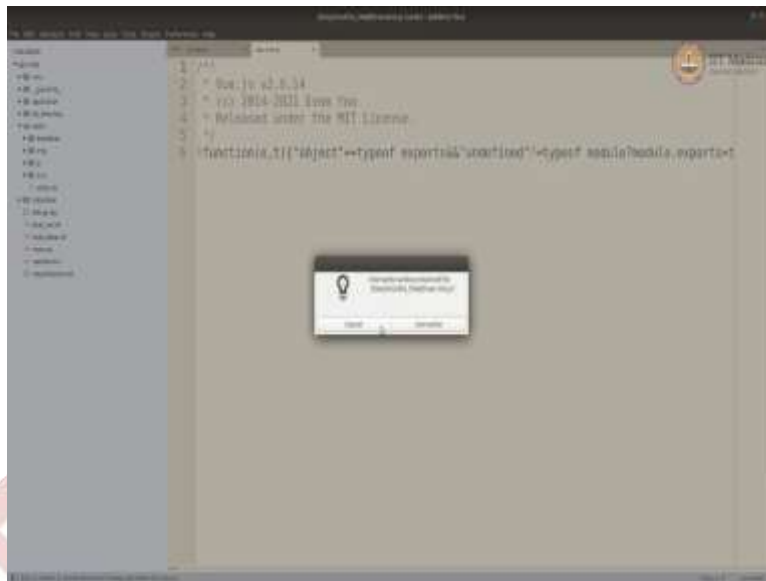
(Refer Slide Time: 08:03)



Let us start with creating a, going to our static folder, just like Bootstrap. Let us create a new folder called Vue. Now, you can also self use Vue from CDN or you can self force to you it's up to you. We can add it into a Vue folder and use it or you can actually use a CDN.

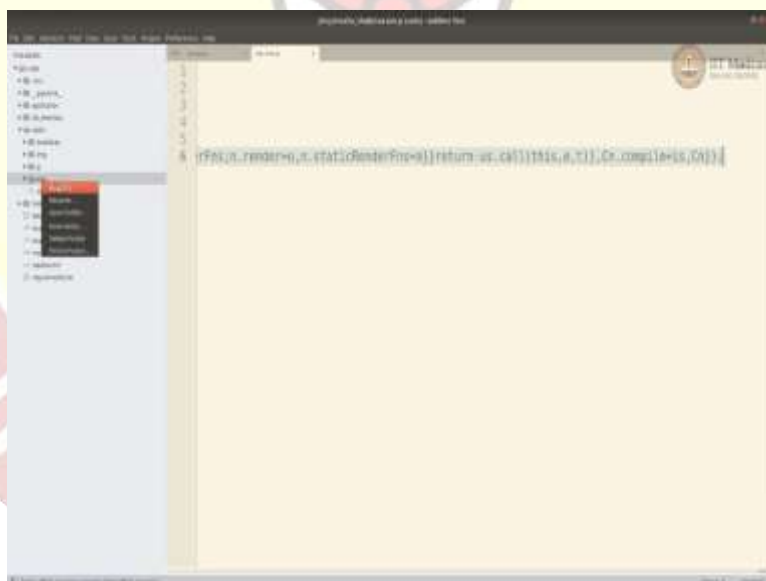
(Refer Slide Time: 08:28)

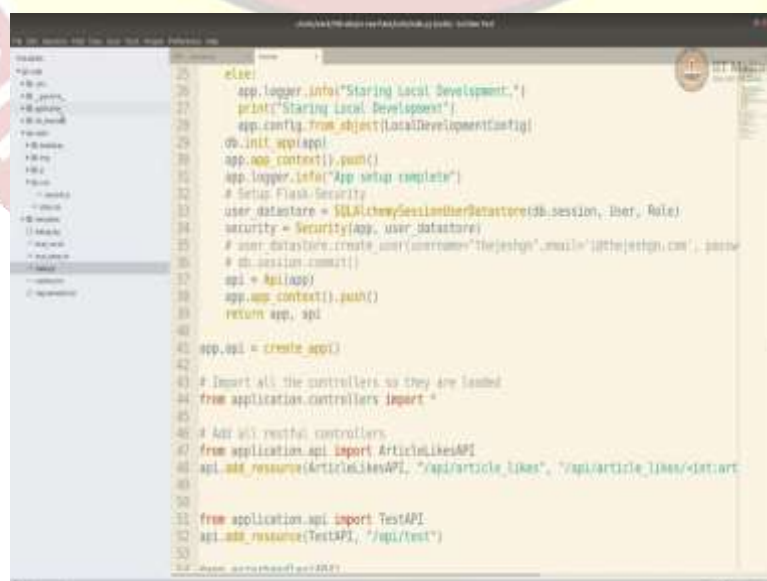
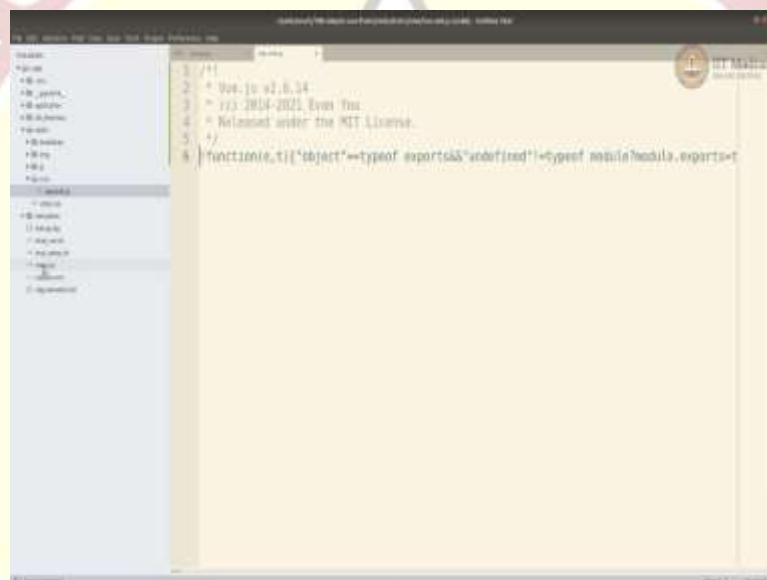
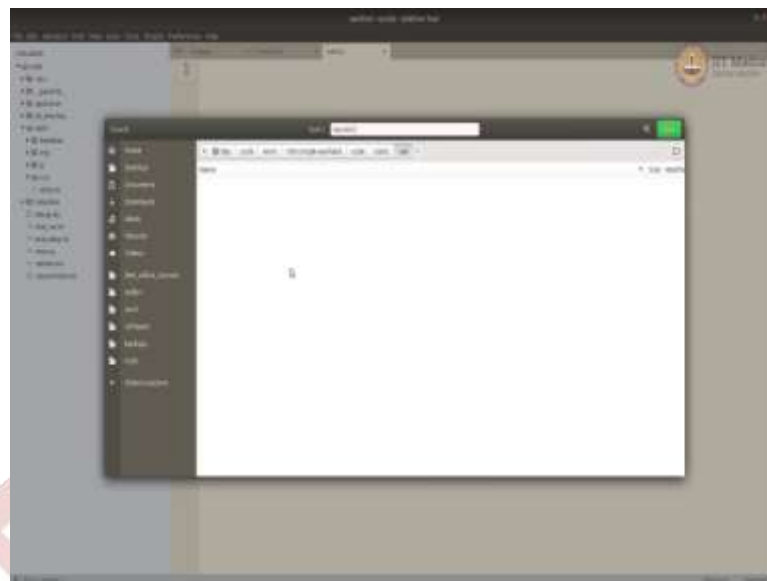




Let us go here, go to the Vue installation folder. If I was going to use CDN, I was going to use this or I can use a production version. Download it, I am just going to use the currently just CDN version. Otherwise, you could have download it here. For example, let us try that. Let us save it. So, we have Vue min.js.

(Refer Slide Time: 09:00)





```
from flask import Flask, request
from flask import render_template
from flask import current_app as app
from application.models import Article
from flask_security import login_required, roles_accepted, roles_required

app.route('/', methods=['GET', 'POST'])
def articles():
    app.logger.info("Inside get all articles using info")
    articles = Article.query.all()
    app.logger.debug("Inside get all articles using debug")
    return render_template("articles.html", articles=articles)

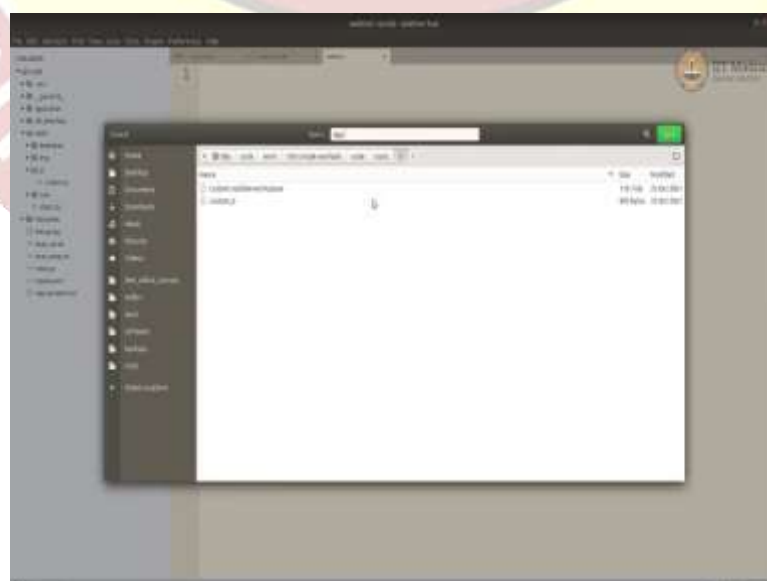
app.route("/articles by/author name", methods=['GET', 'POST'])
@login_required
def articles_by_author(user_name):
    articles = Article.query.filter(Article.authors.any(username=user_name))
    return render_template("articles by author.html", articles=articles, username=user_name)

app.route("/article like/article id", methods=['GET', 'POST'])
def like(article_id):
    print(article_id)
    return "OK", 200

app.route("/feedback", methods=['GET', 'POST'])
def feedback():
    if request.method == "GET":
        return render_template("pages/feedback.html", username=None)
```

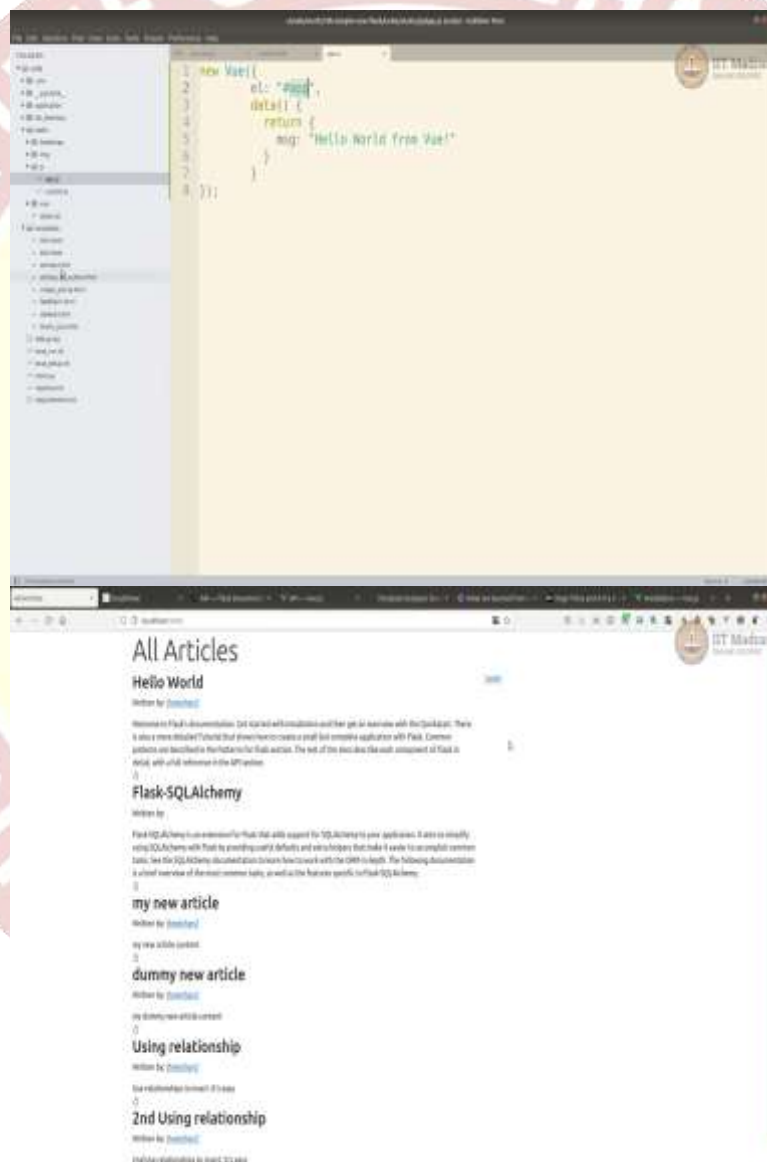
```
<meta charset="utf-8">
<title>All Articles</title>
<link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='bootstrap/css/bootstrap.min.css')}}" />
<link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css')}}" />
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.0/font/bootstrap-icons.css">
<script type="text/javascript" src="{{ url_for('static', filename='bootstrap/js/bootstrap.min.js')}}"></script>
<script type="text/javascript" src="{{ url_for('static', filename='js/custom.js')}}"></script>
<script type="text/javascript" src="{{ url_for('static', filename='vue/vue.min.js')}}"></script>
</head>
<body>
<div>

<div class="container">
<div class="jumbotron">
<h1 class="display-4">All Articles</h1>
</div>
<div class="row">
<div class="col-md-8">
<div>
<h2>{{ article["title"]}}</h2>
<p>Written by:
{{ article["author"]}}</p>
</div>
</div>
</div>
```



Now, in our main, we are loading the index file from controllers. So, if we go to controllers the index page is article dot HTML. So, let us modify articles dot HTML, which is a Jinja template, in our case. Template, articles dot HTML. So, we just need to import our Vue that file but I am just going to import it into copy pasting the whole thing. From the static instead of from the JS, it'd Vue, and Vue dot main.js. Now the other thing that we need to do is to create an app dot JS. All our Vue app is in the app dot js. You can call it anything. But to keep it simple, I am just going to call it app dot js.

(Refer Slide Time: 10:32)



```

1  <!-- @author: Iqbal Ahmad, Amir Hossain, Fahim Reza, Fahim Reza -->
2
3  <!-- CSS -->
4  <meta charset="utf-8">
5  <title>All Articles</title>
6  <link rel="stylesheet" type="text/css" href="{% url for 'static',
7    filename='bootstrap/css/bootstrap.min.css' %}">
8  <link rel="stylesheet" type="text/css" href="{% url for 'static',
9    filename='style.css' %}">
10 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/
11 bootstrap@5.1.1/fonts/bootstrap-icons.css">
12 <script type="text/javascript" src="{% url for 'static', filename='bootstrap
13 /js/bootstrap.min.js' %}"></script>
14 <script type="text/javascript" src="{% url for 'static', filename='js/
15 custom.js' %}"></script>
16
17 <script type="text/javascript" src="{% url for 'static', filename='vue/
18 vue.min.js' %}"></script>
19
20 </head>
21 <body>
22
23 <div class="container">
24 <div class="jumbotron">
25 <h1 class="display-4">All Articles</h1>
26 </div>
27 <div class="row">
28 <div class="col-md-8">
29 {% for article in articles %}
30 <div>
31 <h2>{{ article['title'] }}</h2>
32 <p>Written by:
33 {% for author in article['author'] %}

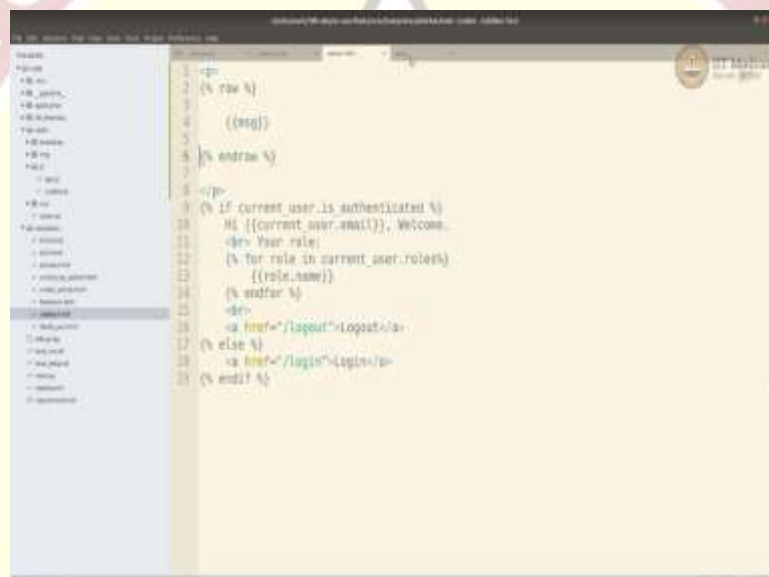
```

[illegible]

```

1 <script type="text/javascript" src="{ if uri for 'static', filename=
2 //s/bootstrap.min.js } }"></script>
3 <script type="text/javascript" src="{ if uri for 'static', filename='js/
4 custom.js } }"></script>
5
6 <script type="text/javascript" src="{ if uri for 'static', filename='vue
7 bootstrap.min.js } }"></script>
8
9 </head>
10 <body>
11
12 <div>
13
14 <div class="container">
15 <div class="jumbotron">
16 <div class="display-4">All Articles</div>
17 </div>
18 <div class="row">
19 <div class="col-md-8">
20 { % for article in articles %}
21 <div>
22 <h2>{{ article["title"] }}</h2>
23 <p>Written by:
24 { % for author in authors %}
25 <a href="/articles-by/{{ author['username'] }}">
26 {{ author["username"] }}</a>{ % if not loop.last
27 %}, { % endl %}
28 { % endl %}
29 </p>
30 <div>
31 {{ article["content"] }}
32 </div>

```

New file, this file is not required. App dot js, we are going to write simple Vue app. That is going to just print hello world from you, and is going to use a div id, this app. Now, we wanted to put it in the sidebar. Here, the sidebar. So, let us open up the sidebar. See where the sidebar comes from? If you see our article, sidebar comes from side by theme. Let us go edit that sidebar dot HTML.

So ideally, what we would do, we would put msg. we all know how in this is our standard Vue template. And then in our main articles, we need to import or include our app dot js. There is no Vue inside, it is inside js and app dot js. Now, let us see what happens if we run this. So, the one thing that we are not done is setting a powerful app. Again, set it up at the highest level, I am just going to make container is equal to app.

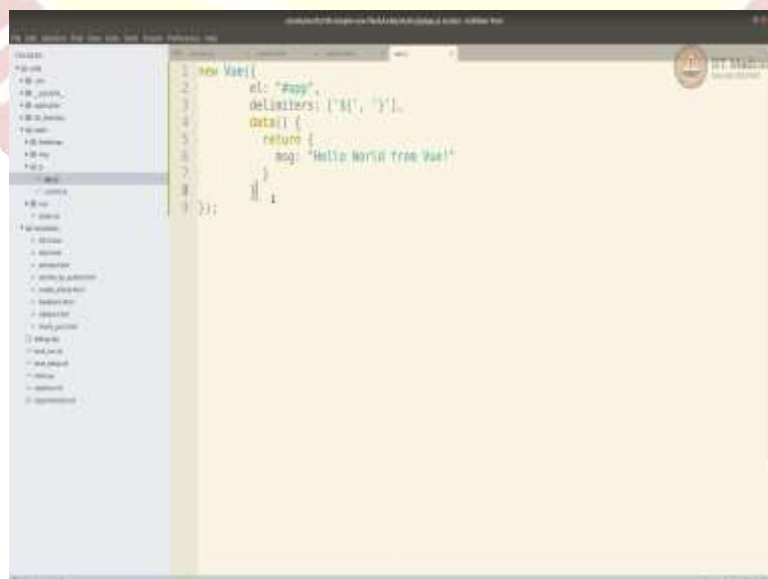
That is the other thing that we need to do. It still will not load, it still will not show. Why is that? Why is that happening? That is because here in our sidebar, HTML, we are actually trying to the template and we are assuming this is going to be a template for our Vue. But we also know that even Jinja uses the same format for printing of the variables.

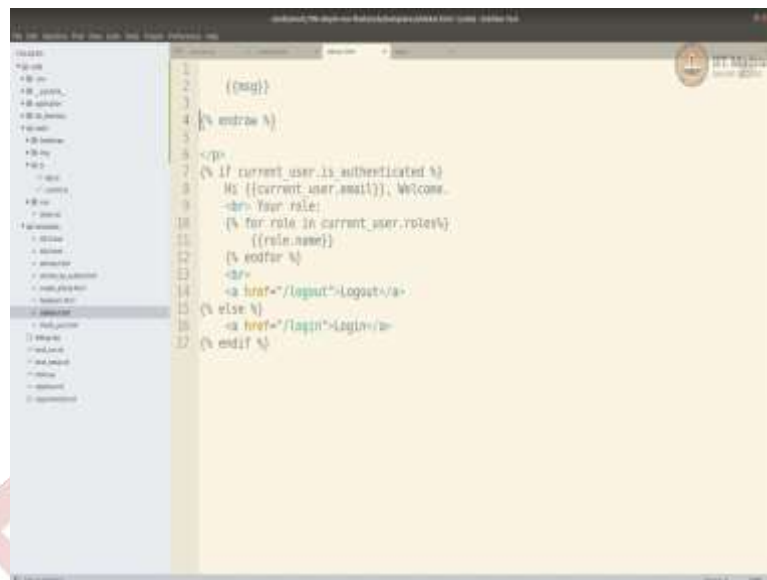
So, it is actually here, when it first gets rendered on the server, even before it is sent to the client, it kind of already tries to render message, which is null. So, it will not do anything, we can just do to do actual rendering of this to client. So, it has to have this exact value sent to the client as HTML, and then it should get rendered at the client side, which means it has to be sent raw, without this part should not be rendered on the server side, only this part should be rendered on the server side, how do you do that?

Jinja actually gives you that, that exact feature called raw and endraw using these strikes, whatever that confuses Jinja will be sent raw to the front end and that gets rendered, as like text by the front end. And if it is a Jinja, Vue template it gets rendered as Vue that is called actually escaping so we are going to add this tag, so it gets escaped as Jinja from the Jinja and then gets rendered only in the front end.

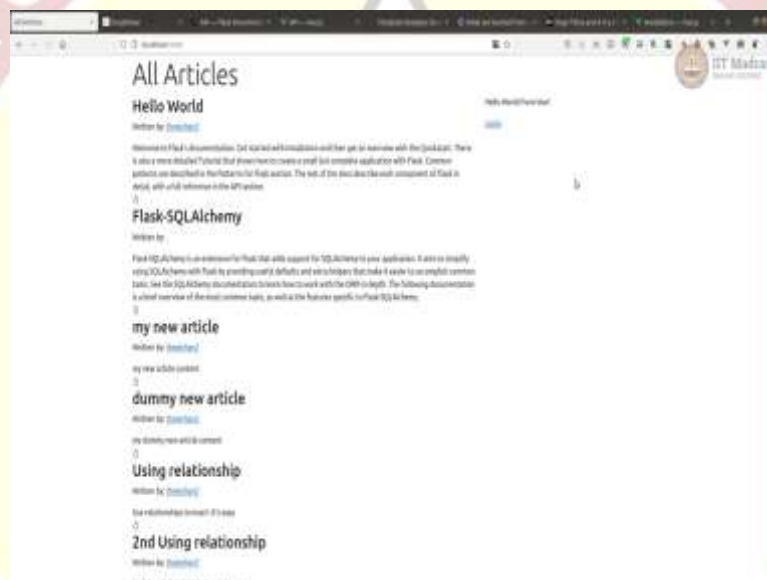
Let us put endraw. I am just going to add it to inside a P or paragraph so it is clearer. Now, let us refresh our page. There you go. Now it is actually rendered from Vue. But when the HTML is sent, HTML actually sends the actual unescaped or rather escaped raw. This part, raw message variable, Now, this is a lot of work to do this all the time. And then there is a good chance that you will miss out as well. The luck part is.

(Refer Slide Time: 15:14)





```
1 {{ user }}
2
3
4 {% endraw %}
5
6
7 {% if current_user.is_authenticated %}
8   Hi {{current_user.email}}, Welcome.
9   -> Your role:
10   {% for role in current_user.roles %}
11     {{role.name}}
12   {% endfor %}
13 {% else %}
14   <a href="/logout">Logout</a>
15 {% else %}
16   <a href="/login">Login</a>
17 {% endif %}
```



Vue allows you to set up your own delimiters for the variables. We can set up one which is almost like ES6 templating like this, instead of double quote, double flower bracket, you can use dollar and flower bracket. Once you do these delimiters and since this does not have any specific meaning in Jinja it should be fine without the raw.

So, I will remove this remove this now Jinja template variable will look like this. correct. Now, if I refresh, it will still stay. If I remove this it will go. And this will come back. So, either you can use escaping in jinja or use delimiters. I usually use delimiters because it makes, whenever you look at Jinja template, you clearly know that this is client side rendered, dollar, starts with dollar and whatever with regular Jinja template actually rendered on the server side.

So, it makes it very clear and easy to code. Now we got reactivity to this, this part of page, it could do something similar for other parts of the page or whole page. You could, like we have learned before you can go ahead and improve this and add a fetch request to it all of that, you can different components and in use component inside. You can mix and match all of it and it is quite powerful when you mix both server-side rendering and client-side rendering. That is it. Thank you for watching.

