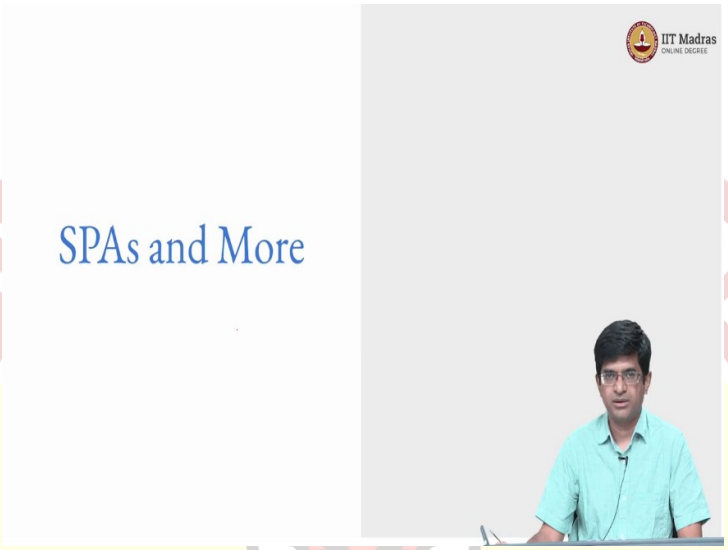
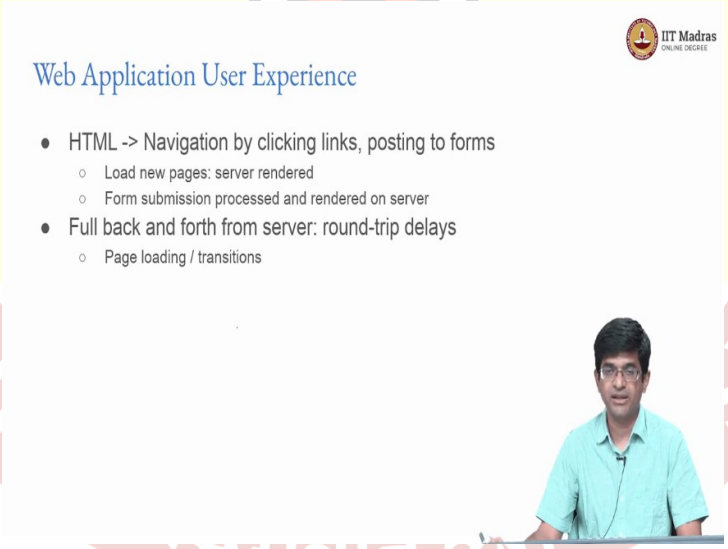
IIT Madras

ONLINE DEGREE

Modern Application Development – II
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
SPAs and More

(Refer Slide Time: 00:15)





Now we are going to look at a notion called a Single Page Application and related concept. So, there are a couple of different things over here, which have emerged as the result of all these new technologies in the web browsers.

So, let us first look at what the web application user experience used to look like, the original web applications. You would have HTML, which is received by the browser, it would be rendered, and all navigation from there on is handled by links or posting to forms. So,

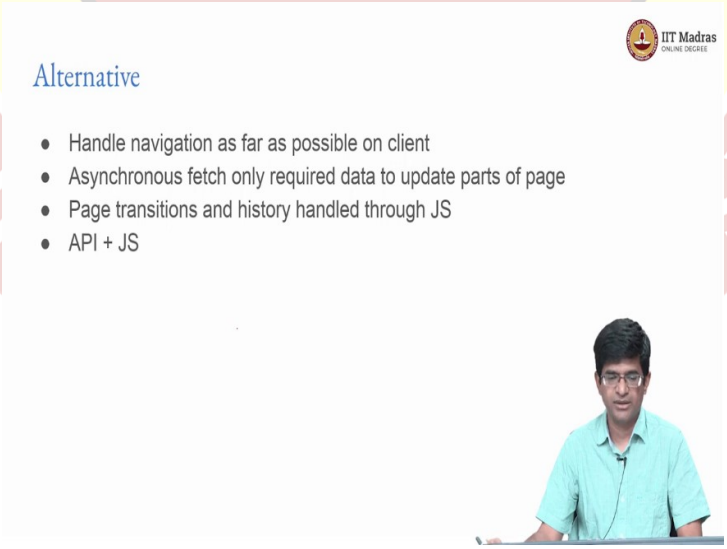
anytime you want to load a new page, the server gets a request, it renders the page if necessary or just picks up the HTML, if that is, if it is a static page, and sends it over to the browser.

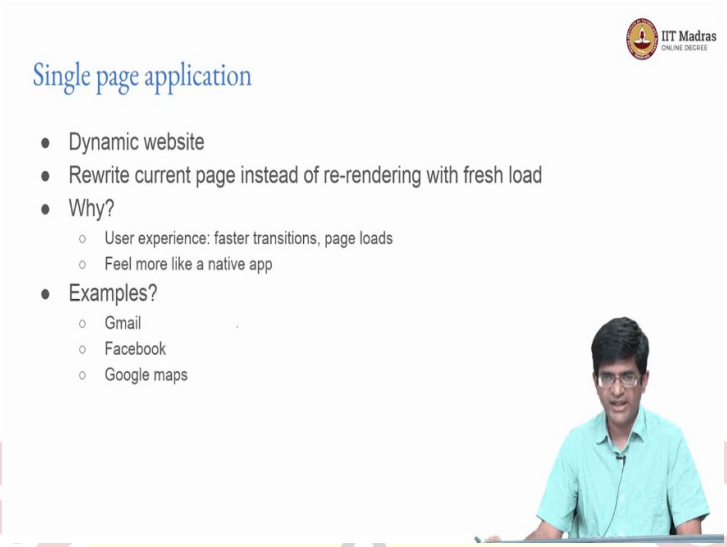
Now, form submissions, for example, are processed and any resulting output is rendered on the server. So, in some cases, the server would actually like have new HTML being generated or it might just have a sort of thank you for submitting kind of a standard template html page, which it sends back, once the form submission has been successful.

But every single such click, write any interaction is ultimately going back to the server and you have these round-trip delays, which sort of becomes very obvious when you are actually working with the system. It is very clear that every time that you click on a link, it is going back to the system and then coming back and, you have this page being refreshed completely from scratch.

And one of the questions that came up, as these technologies progress was, look, how do we make something like this more fluid, more sort of appealing to the end user. Something that looks more like an application that they are used to on their desktop, rather than something which is clearly a webpage, which sometimes goes into this loading mode, and takes a long time to refresh, and so on.

(Refer Slide Time: 02:16)





So, the alternative that slowly built up over time was, let us handle navigation as far as possible on the client. And asynchronously, fetch only whatever data is required to update parts of the page, rather than just getting the entire HTML back from the server. In, you can go one step further and say that look, if you are anyway handling the navigation on the browser, but you still want to give the user the feeling that they are using regular navigation through a webpage, maybe handle the page history and the transitions also through JavaScript.

So, basically, these you have several API's that are used through JavaScript in order to enhance the user experience, which finally sort of leads us to something that is known as a single page application. So, a single page application is typically a dynamic website, something which has to change. It is responding to the user's inputs, because otherwise, it is just a static HTML page, why bother calling it a single page application. That is not what we mean by a single page application.

So, in a single page application it is dynamic things do change over time depending on user input. But what we are going to do is, rewrite the current page as far as possible, instead of re rendering with a fresh load. Why do we do this? It is primarily for user experience. It feels faster, the page loads sort of occur instantaneously. It feels much more like a native application.

And these kinds of single page applications, there are any number of them that you probably use on a daily basis. The Gmail web interface is an example of a single page application. Pretty much everything out there is handled with JavaScript, but done in a clean way, so that for all practical purposes, it almost feels very close to a desktop application. Having said that,

it is pretty much completely in online mode, that is one of the things. There is an offline mode, which we will get to later that is a different story. How exactly is offline implemented?

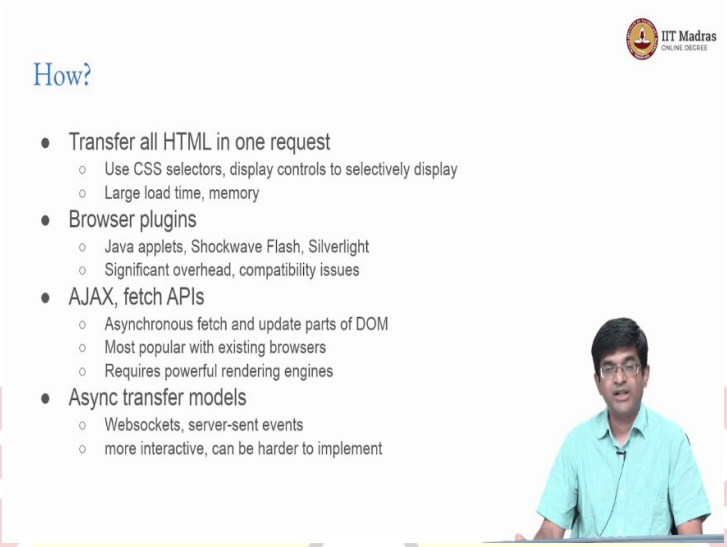
Facebook similarly is something which is completely online, but it does require the server. But all the navigation, all the updates and so on are handled on the client, which is why you have a nice smooth user interface when you use something of this sort. And Google maps, in fact, you might recall that the whole Ajax concept, one of the driving factors over there was this implementation of Google maps in the 2004, 2005 timeframe.

What exactly did they do? What they said was, as you sort of zoom into part of the map, rather than sort of going in and saying every time you click will go and load a new page, the loading of the image data was happening in the background. The system was basically picking up which way are you zooming which pages, which parts of the map do I need to load next.

And based on that, it will then go in, send continuously, it will be sending request to the server in the background, which is why you will actually know that, as you try zooming in, in a Google map, it takes a little bit of time before the thing sort of actually zooms in. Sometimes you see this slightly blocky looking thing, and then it gets overlaid with the actual data. What is happening is that a fetch occurred in the background.

New data coming came back from the server, and that is what is being displayed on your screen now. Of course, it is also being cached efficiently and so on, but the fact was that the fetches happened in the background without needing to go to a new page, so Google maps, in that sense, was probably one of the best instances of what a single page application could look like. But nowadays, there are any number of them, that you could see around you.

(Refer Slide Time: 06:07)



So, there are many different ways in which something like a single page application can be implemented. So, we will look at some of the older techniques first, and some even newer techniques, before we sort of look at how you can do it in something like view.

Now, keep in mind that a single page application does not automatically mean that you need JavaScript. If all that you are looking for is something where what is displayed on the screen is something that changes with time and can be controlled through some kind of clicking or something of that sort you could actually do it completely with CSS. CSS allows you to selectively display certain document elements.

So, you could have one large HTML page where there are different sort of divs or something like that or containers that have specific IDs, and they get different classes in CSS associated with them. And depending on which class is there and which one is to be shown, you can change whether something is actually displayed on the screen or is it just sort of taken off the screen. So, you could transition between different things at one shot, it purely CSS.

Now, of course, what it means is that the entire page with all the information you need has to be loaded right up front. Usually, that means slower load times and using more memory. And at the end of the day, except for the fact that it allows you to show multiple different static pages and transition between them easily it still does not do much in terms of interaction.

Now for a long time the sort of dominant way by which you would have these kinds of applications inside a browser was using various kinds of browser plugins. Things like Java, applets, Shockwave Flash, Silverlight, from Microsoft, all of those, the idea of those things

was that you did not have enough power inside the browser to handle something of the sort, so let us put in a custom runtime environment that can handle things for you.

Now, usually what ended up was there was significant overhead. Even though Java applets sounded nice anybody who has used a Java applet would know that, I mean, the moment you hit a page with a Java applet, it is a bit scary. I mean, the whole thing slows down, the Java VM is going to start up, it needs to load the applet, it then shows loading, it was not a very pleasant experience. It does allow the web developer to give a very good sort of experience in certain ways if it is written well, but overall, the bottom line is that, it made usage sort of more complicated.

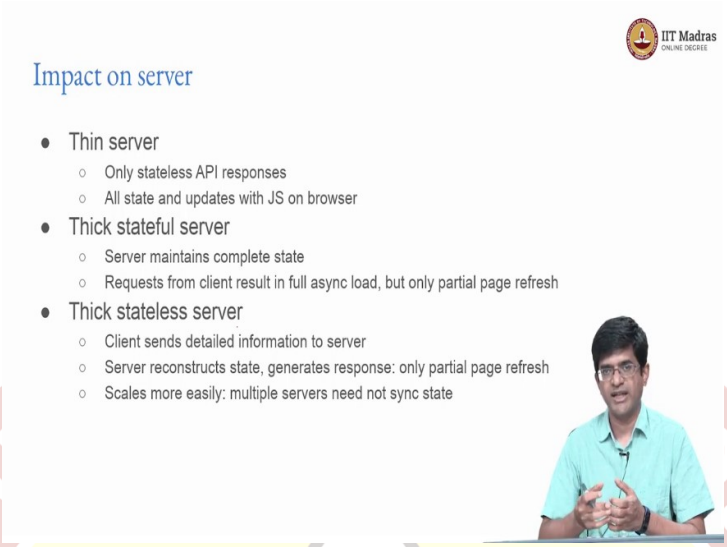
Now, of course, the most common one is AJAX, which has now developed into various kinds of fetch API's, where what you do is you asynchronously fetch and update just parts of the DOM and it is very popular with existing browsers. So, modern browsers are, have all these new JavaScript capabilities. And more importantly, modern hardware is sufficiently powerful that it can run it pretty much in real time and give you a very smooth experience when you are trying to run with something of this sort.

It does require powerful rendering engines. What I mean by rendering engines is, the part of the processing that actually converts the HTML into something that gets a set of pixels on the screen that is the rendering part. And because of the fact that things are changing so fast, all of this has to be done in the browser in a fairly efficient manner.

Now, there is one other approach also, which is even one step beyond what you would do with fetch API's, which is there are certain kinds of asynchronous transfer models that exist between browser and server, things like Websockets or server-sent events that actually allow for even better entry their activity than what you would normally see with frameworks that use JavaScript. In particular, the server can now push updates.

So, even things like for example, having real time chat or video conferencing or push notifications, all of those are handled very efficiently by means of the server-sent events and web sockets. But in general, those are a little bit harder to implement they are in terms of interactivity, and the kind of experience they give, they are probably the best in the line, but they are harder to implement than just slightly more simple framework based JavaScript.

(Refer Slide Time: 10:40)



Now, all of this has certain impacts on the server as well. In terms of how you go about designing and using the server, we can broadly sort of think of it in a few different classes. So, this is sort of the way it is described on Wikipedia, for example, which makes sense. I mean, this classification over here is a good way to think about the different ways in which servers can be thought of.

The first and most important thing is the moment you have this kind of separation out here, and you are trying to increase the load on the client, more of the processing load is going to be handled by the client, the navigation and deciding which component to load. Even most of showing the different parts of the components is all handled on the client side. The server actually becomes simpler and potentially thinner. Thinner, meaning that it does not really need to do very much.

In fact, you could potentially have a server whose only job is to respond to API requests. And the rest of it, everything else is displayed on the browser using API's and JavaScript and whatever markup HTML or the equivalent that you have there. So, that essentially brings in the notion of a thin server.

Usually, we have what is called a thin client where everything is handled on the server, and the client is just sort of dummy front end display or here we are talking about the opposite, which is, the client takes care of a lot of the bulk processing and the server's job is simplified, it just needs to respond to API requests.

On the other hand, you could still go back and say, I still want to have a thick and stateful server. Thick, meaning that it has a lot of it has the entire system state associated with it. It also has the full state of whatever is currently being displayed on the screen. And if you think about it, a single page application, the point that we are trying to make is we do not want to be re-rendering the page each time, but if I do need to fetch information from the server, as long as it is done in the background, and sort of updated in place, I would still consider it a single page server, single page application.

So even in the context where, I have a server that is like, got complete information about the present state of the system and keeps updating what needs to get displayed on the screen based on any new sort of request update from the client, it would still be potentially a single page application.

So, when you look at this as when you realize that single page application can be a little bit shaky in terms of its definition. After all, if I am making all my fetch request to the server, and pretty much getting all the data that I need from there what is so great about the SPA. And sort of the main defining characteristic at that point is the fact that it is not explicitly rendering, re-rendering the page from scratch each time. Is that really so big deal?

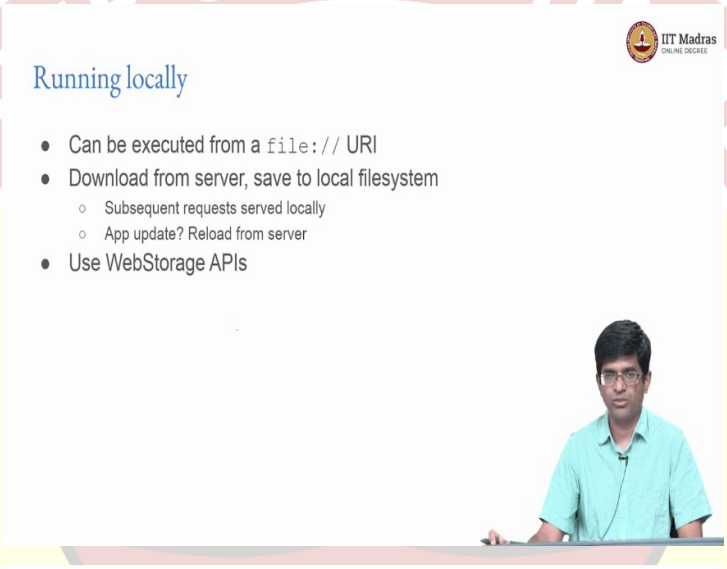
Well, it, that is where things get a little bit tricky in terms of whether you are defining something as an SPA or just as something which is corresponding to a thick server, which transfers all the data. You could also potentially have something in between these two, which is which we call a thick stateless server.

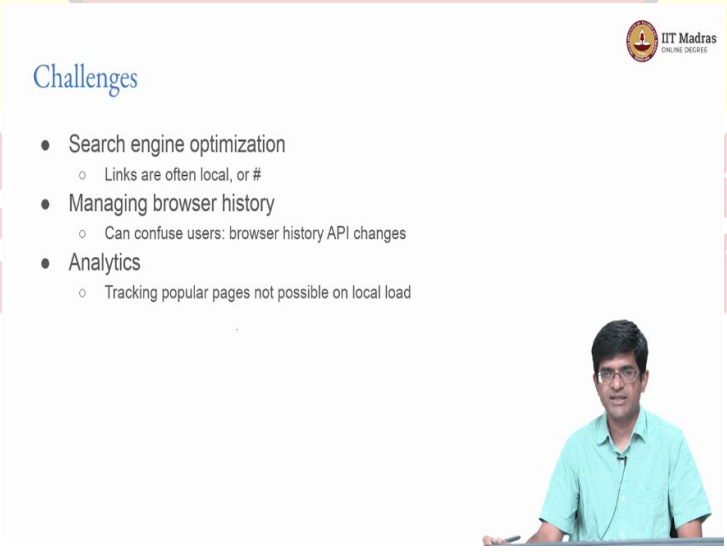
Now, this is interesting, because what it is saying is the server still handles most of the computation. It decides what needs to be rendered on the screen and so on, but it is stateless. Meaning that it is not actually keeping track of the present state of the client connected to it. So, then what is it doing? The client still has to send it a lot of data telling it what its current status, but based on that, it will look at its own internal state to take the entire client state, compute what it needs and send it back to the client.

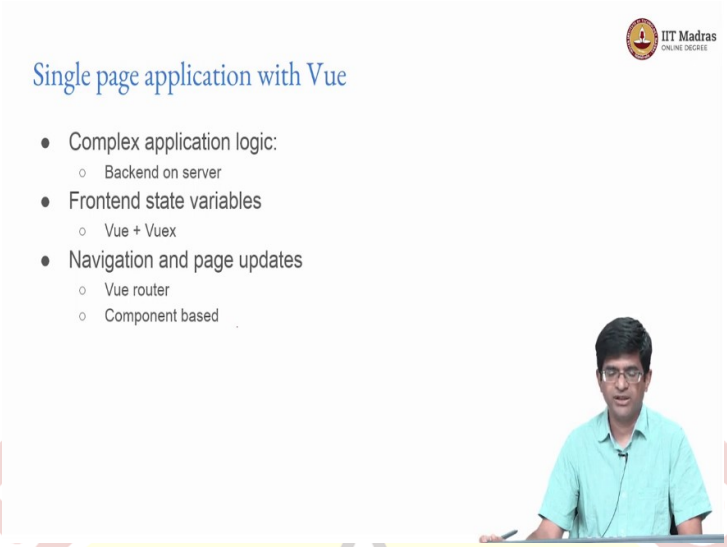
Under what circumstances is this good? When you have a strong network connection between client and server. A client that by itself is not very powerful, but a server that is reasonably powerful or more importantly, maybe a large sort of group of servers where you can distribute the load because each individual server is stateless. It does not need to know in which state the client was.

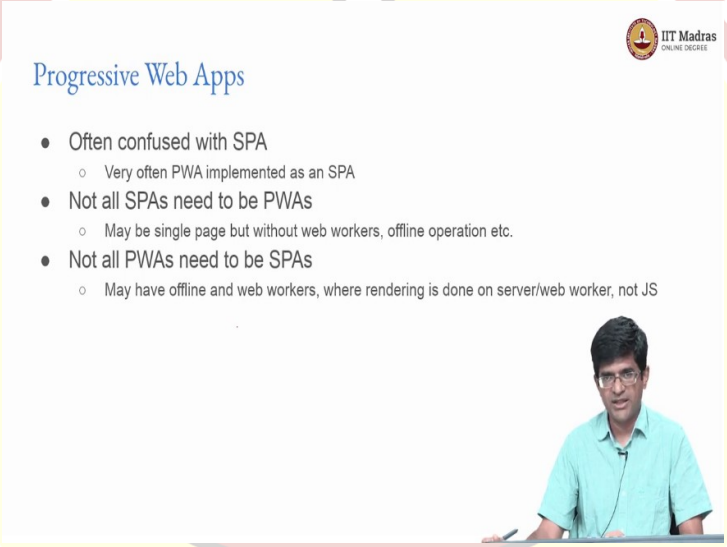
It gets all that information, performs the computation and sends the data back. So, from that point of view, a thick stateless server can also be A good way of implementing something like this. These are potential ways of looking at your architecture. The common one, of course, at least that we would normally see in a JavaScript and API-based approach would be tending more towards the thin server. Meaning that the front end the JavaScript on the browser takes care of most of the work for you, and just makes API request in order to decide what needs to get updated on screen.

(Refer Slide Time: 15:28)









The interesting thing is, you can also take these kinds of SPAs, and in a lot of cases, run them locally. You can save them as a file, which means that they will now be accessible under a file colon URI. And you, the first time around you download it from the server, you save it to the local file system, but subsequent requests are then served locally.

And the interesting thing here is you want to update the app, just reload from the server. So, the person who is maintaining the app also just needs to update whatever they are serving, and that is it that is like an app update. You do not need to download a lot of information for that. Now, when you run locally, of course, there are questions such as where do you store information and so on, but there are API's. The web storage API that we have already seen, can be used in order to store user information at the browser itself.

Now, there are several challenges associated with single page application. One of them is, how do you optimize for search engine visibility. Because after all, what is happening is previously, you would have a specific URL that corresponds to each different part of what gets displayed on the screen. But with routers, for example, you are no longer able to have that kind of control or you can, but it is not sort of straightforward.

Especially when a person clicks through right or a web crawler is looking at your page, it is not really seeing all the different links, because crawlers usually do not run JavaScript. They are just looking at the HTML of the page and deciding where to go next. So, it is quite likely, they will not go through all the router links and decide which ones to sort of look at one after the other.

Managing the browser history, like I said, has to be handled with care so that it gives a good user experience at the end of the day. And that can also be problems with analytics, because after all, you are loading components internal to the system over there, and not all of them are generating hits on to the main server.

And more importantly, things like Google Analytics, for example, would not even get triggered because those happen only when you explicitly have some kind of HTML page load under page refresh. If just a component or an API call goes out, that does not automatically trigger analytics update.

Now, obviously, since we have been looking at Vue as a framework, a single page application with Vue, what would it look like? Usually, you would have something like, the all the application logic, the complex part of it would be on the backend on a server, possibly implemented using flask or some other kind of platform that, you know, works in terms of implementing API's, for example.

The front-end state variables, it would be a combination of Vue and possibly Vuex assuming that you have sufficiently complicated state that you want to use the state management library. And the Vue router would then take care of navigation and page updates, which would now be completely component based rather than individually HTML pages.

So, that is basically where Vue as library helps you to construct single page applications. You can do it with other frameworks with React, Angular, there are whole bunch of different frameworks that can be used in order to construct a single page applications Vue happens to be one that can be done with sort of a minimum of friction, so to say.

Now, we have talked so far about single page applications. We can go one step further from there and talk about something called a progressive web application or a PWA. This is a closely related and often confused term. I mean, people generally tend to confuse PWA's with SPAs, and so on. And it is understandable because there are a lot of similarities between the two.

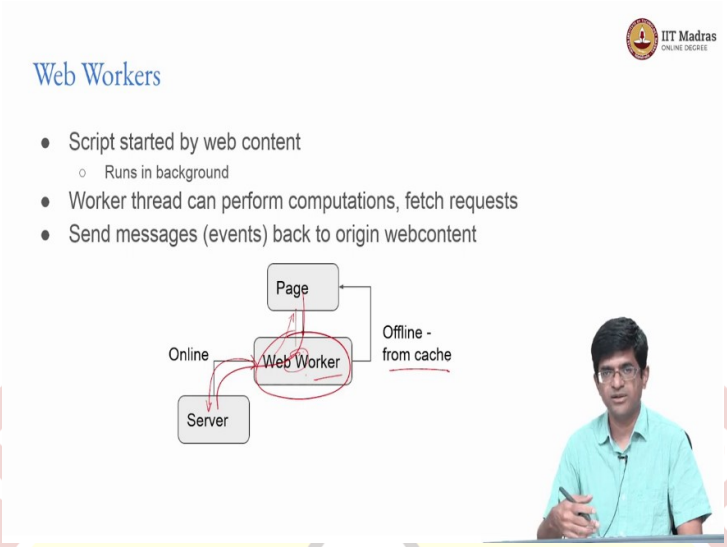
And the point is that one of the reasons for that confusion is the fact that very often, PWA's are implemented as SPA. But there are certain things that people think of when they hear the term PWA, which sort of extend a little bit beyond what a plain SPA by itself is required. So, the point is that not all single page applications would be considered progressive web applications.

You could have even some something as simple as a blog website being a single page application, but nobody is really going to call that a PWA. Because it is, it does not have any significant dynamic content. There is no notion of sort of offline usage and things of that sort, which you need to worry about in a simple SPA.

On the other hand, it is possible to have something which is actually a PWA, a progressive web application, which may not necessarily be implemented as a SPA. You could have the notion of offline and web workers where, which we will get to in a moment. But where the rendering is done explicitly, and you do have new requests being sent to some server and then coming back from that server with a response.

But like I said, the most common scenario is that a PWA, a progressive web app is implemented in the form of an SPA, which means that I get to benefit from all the front end advances of SPAs. So, then comes a question, what exactly is it that a PWA provides, in addition to that?

(Refer Slide Time: 21:11)



So, to understand that, we also need to think a little bit about this notion of what is called Web Workers. So, a web worker is something it is a script, you can think of it as a script, probably a JavaScript function or set of functions that is started by the web content. What is the web content, it is basically the HTML page that gets displayed. And the idea is that it runs in the background.

And a worker thread, can perform computations. It can also perform fetch request. So, in other words, it can perform a whole bunch of asynchronous operations. But the interesting thing is, if a worker thread was started or a web worker was started by some web page, it means that that page can communicate with a web worker. It can make a request to the web worker, and more importantly, that worker can then also push events back to the page.

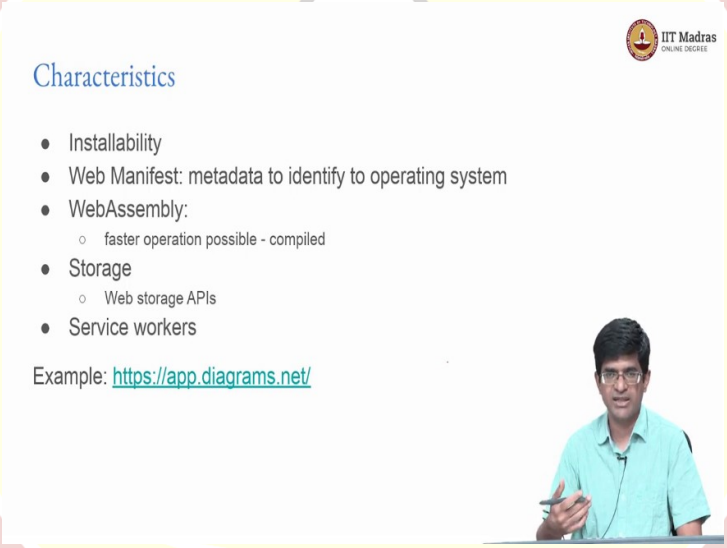
So you could have something where the entire sort of interaction between the page and the web worker is a nice sort of back and forth of messages that are going on over there. Now, what this means in turn, is that by having this web worker sitting over here you effectively make it a proxy. I mean, I am using the term proxy loosely, this is not exactly the term in which sense in which it is implemented over here.

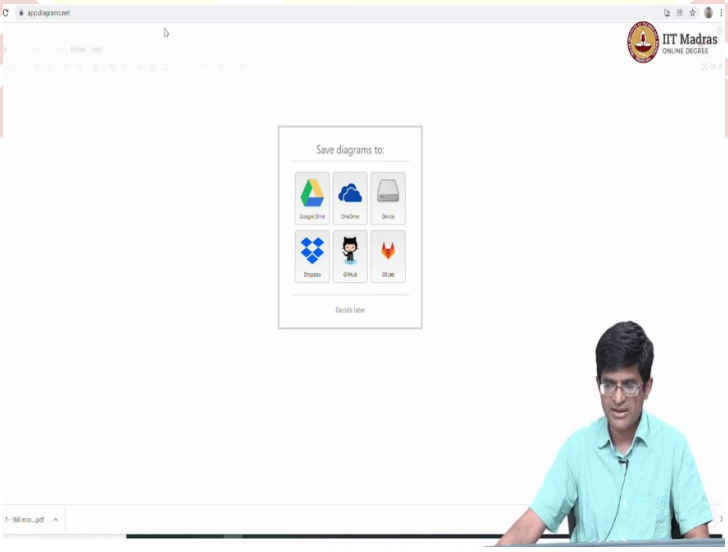
What happens is that this page now sort of connects to the web worker, and makes requests to the web worker. And what the web worker can do is, if it is online, and has access to the server, usually it would then connect to the server get the information back and then resend it back to the page. So that is a sense in which it looks like a proxy. It is not absolutely required to function exactly this way, it will probably do some processing on it before sending it back to the page.

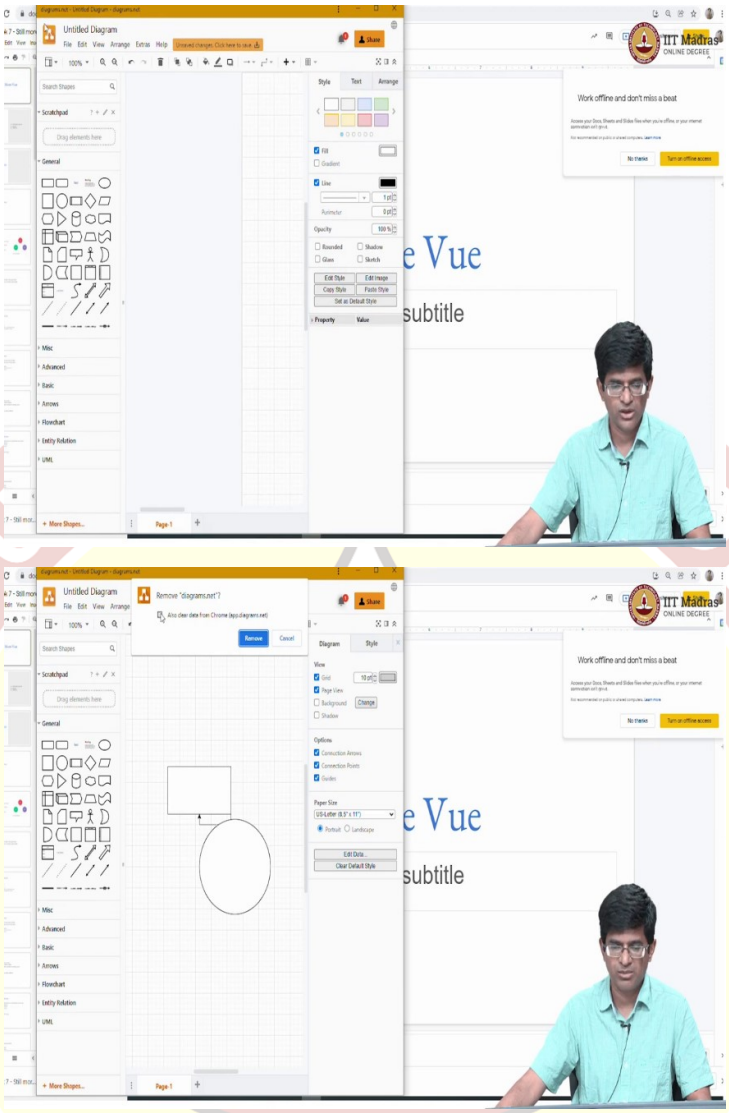
But the important point is, because this web worker is running on your not. Well, it is essentially part of the browser that is there on your system. What it means is that even if you are offline, your network connectivity is disrupted, the web worker would still have access to some information. Maybe previous information or maybe whatever the developer decided should be there in an offline mode, which it can now deliver from the cache.

And this is one of the crucial aspects that sort of makes a PWA an actual web application. The fact that it continues to operate and can exist, even without the network connectivity, and can behave in a reasonable manner, even though you do not have the full access to the Internet. That is where it starts behaving like an application rather than just a connection between you and the server.

(Refer Slide Time: 24:02)







So, there are a number of characteristics that define PWA. One of them is the fact that it can be installed and I will show you an example of that in a moment. There is something called a web manifest that needs to be provided with each PWA. And this is basically meta data that helps to identify the operator to the operating system, some details about the application.

In a lot of cases, these PWA's nowadays, especially are beginning to use web assembly. Web assembly being a sort of compiled version of JavaScript that can execute faster. It executes in some kind of a virtual machine inside the browser and you have this byte code that gets generated and can execute over there potentially faster than vanilla JavaScript.

It also has or makes use of the storage API's, typically web storage APIs and the worker API's, the service workers. All of these are sort of common characteristics of PWAs. It is not that every one of these has to be present, but most of them are going to be found in various

PWAs. So as an example of a PWA, I am opening this website called app dot diagrams, dotnet. It is a website that can be used in order to draw various kinds of vector graphics and can be useful for lots of flowcharts and various other diagrams that you might potentially find useful.

It is a regular website. And it asked me where I want to save the diagrams I am just going to say decide later for now. It brings up this menu. It is clearly inside a browser, because I can see the back button, the refresh button. I can see the URL bar up on the top, but otherwise, everything else, it allows me to basically draw things over here and change things right on it, and so on.

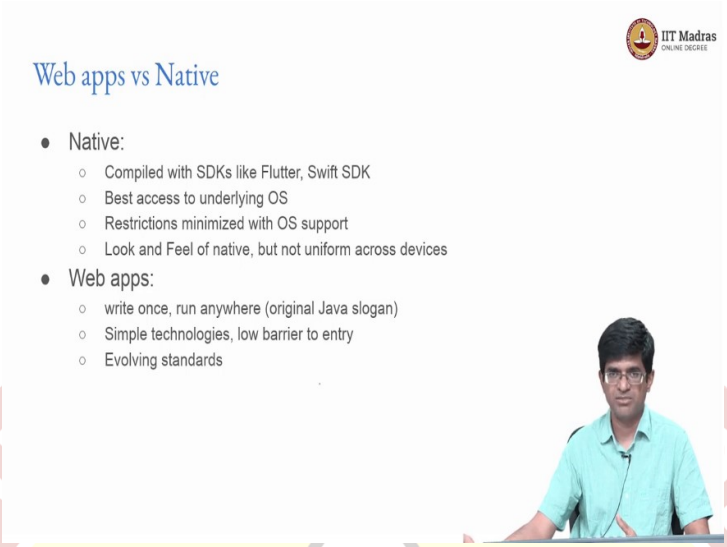
The interesting thing is in the URL bar on the top right, you will see this new icon that says, install diagrams dotnet. When I click on it, it basically says, should I install the app, and I can click on install. And the moment I do that, it now opens up in a separate window of its own. So, you will notice that it has now opened up in a separate window, that window does not have the rest of the URL bar, the back button or any of those things that indicate that this is a browser, it almost looks like a native application.

And in fact, I can use it that way. I can continue doing the same thing. Of course, it still has the diagram that I left off over there. I can add more to it, I can make connections and so on. Use it just the way that I would use it in the regular app. And finally, if I do not want it, I can just go there, I can go and click on app info, find out more about it.

It tells me a little bit about the connection, and so on, the site settings, cookies and so on are required in order for this to operate, but the interesting thing is, I can also either open it in Chrome, in the browser or I can uninstall the entire app, just by clicking on this remove button, at which point, it is basically going away.

So, app dot diagrams dot net is an example of a PWA, that can be used in order to install something locally onto your system that you can then use almost like a regular application that you would use on the system. It, of course, brings up a restricted version of the browser that only has access to that particular website and allows you to work. It also installs in offline mode, which means, that even if you are not connected to the Internet you should still be able to work on the system. So that is essentially sort of the notion of a progressive web app.

(Refer Slide Time: 27:52)



And finally, a word about web apps versus native applications. So, ultimately, the reason behind going for a web app is simply because of the ease of development in some sense. You have a very low barrier to entry. But on the other hand native applications they are typically compiled with a software development kit, there are many of them. There is Flutter, there is the Swift SDK for iOS and Mac applications.

Several other SDKs that allow you to compile and directly create an executable that will run on a particular platform. Now, the native apps therefore have the best access to the underlying operating system, because they are usually integrating tightly with the API's provided by the operating system itself, and not going through an extra browser interface in between.

They will generally look and feel like the native application. So, on a Mac, they would look exactly like a Mac on Linux, they would look like a Linux application on Windows, they would look like a Windows application and so on. But the problem is, it may not look uniform across devices.

On the other hand, a web app, the original Java slogan, the Java language slogan that Sun Microsystems came up with in around 1995 or 96, was write once, run anywhere. And the idea was simply that, that you would write in Java, compile it into bytecode. And then it did not matter on which platform you were running, whether it was Linux or Windows, at the time.

You had, as long as you had a Java bytecode interpreter, a Java virtual machine, you would be able to run the program there. Now, of course, Java ran into other problems, and ultimately made it big on the server scene. But in terms of taking our apps and applets, it was not really very successful.

On the other hand, web apps have been perfect because they started incrementally. They started with just cleaning up the user interface, HTML and CSS became better. And along the way, JavaScript developed more and more powerful capabilities to the point where with a very low barrier to entry with very low requirements from the developer, you can actually construct fairly complex applications.

One of the concerns over here, of course, is these are continuously evolving standards. So, you need to sort of keep track of what is the latest and greatest and in order to make the best use of the facilities that you have. Even in terms of optimizing the size of the applications, minimizing them getting like efficient network transfer, and so on all of those.

But at the end of the day, we are now at a stage where web applications can come very close to native in terms of the look feel and performance. And this, SPA, PWA's and so on, are one of the most interesting ways in which we can get to that kind of behavior.