

IIT Madras
ONLINE DEGREE

Modern Application development – II
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
JAM Approach

Hello, everyone; welcome to modern application development part 2.

(Refer Slide Time: 00:14)



Now, I want to take a little while to talk briefly about JAM. You would probably have heard the term JAM stack, and it essentially refers to JavaScript, API's and markup. And give a few thoughts on why this is a good way in some sense of building apps, or at least a very well structured way that exists at present. Is it necessarily the best or the end of the road? Unlikely, but we will get to that later.

(Refer Slide Time: 00:43)

What does an app need?



- Data store: what the app is for
 - Access and retrieval: APIs
 - SQL, NoSQL, GraphQL, ...
- User Interface: to interact with the user
 - Vanilla HTML + forms: request/response
 - JavaScript: interactivity, closer to native
- Business logic: what should be done with the data?
 - Backend computation: Python, Go, NodeJS, ...
 - Frontend computation: JS, WASM



So, the first thing to understand, why would we even go for any kind of these kind of stacks is? What exactly does an app need? There are a few things. One is it needs a data store, some place where the data corresponding to the app is actually kept. So, I need to be able to access the data, retrieve the data, make modifications to the data; I can have API's for that. And how exactly is the data stored that could be done using SQL, NoSQL, GraphQL; we have seen all of that. Of course, GraphQL is not a replacement for SQL or NoSQL as we saw.

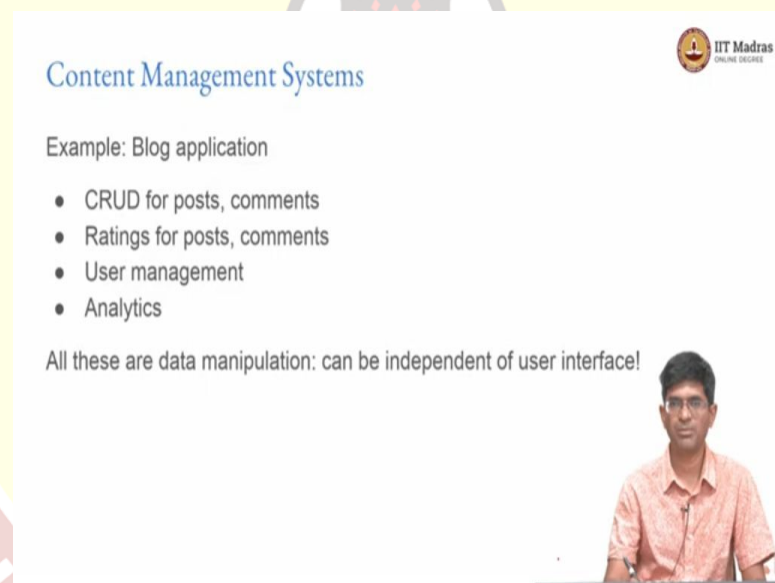
It is more of a way of interacting with the data. But the point is without the data, the app by itself is unlikely to be very useful. There has to be something that the app provides, which means that there is something that it stores and processes. Then of course, there is the user interface which is used to interact with the user. And the original HTML approaches was just Vanilla HTML plus forms. Every single interaction between the client and the server was you either click on a link or submit a form; the server looks at what you sent, and responds appropriately.

You can build a lot of very powerful functionality just using that. And what has been happening over the years is you have brought JavaScript in for the primary purpose that you want to make it more interactive, and closer to a native look and feel. So to say right, it should feel more like a native application, meaning that something that is directly running on your system; and not something that is going back to the server for every update on the screen. And of course, the other thing that an app needs is the business logic, what should be done with the data?

And you have once again you can break this up into certain amount of backend computation. What kind of processing of the data should I do, which can be done in many languages? It could be Python; it could be Go, NodeJS, all that depends on the person who is implementing the system. On the other hand, you have the frontend computation, which is usually handled with JavaScript; that is sort of the dominant, clearly dominant language on the frontend at this point. The only real alternative at this stage would be what is called WASM web assembly.

And the interesting thing is you actually have sort of converters from various other sources, including JavaScript to WASM. So, is WASM an equivalent of JavaScript? They serve slightly different purposes. JavaScript is easier to write and debug; WASM is more for performance, but they both run on the frontend.

(Refer Slide Time: 03:26)

The slide is titled "Content Management Systems" in blue text. Below the title, it says "Example: Blog application". There is a bulleted list of features: "• CRUD for posts, comments", "• Ratings for posts, comments", "• User management", and "• Analytics". At the bottom of the list, it says "All these are data manipulation: can be independent of user interface!". In the top right corner, there is a logo for "IIT Madras ONLINE DEGREE". In the bottom right corner, there is a small video feed of a man in a pink shirt speaking into a microphone. The entire slide is overlaid on a large, semi-transparent watermark of the IIT Madras logo, which includes the text "INDIAN INSTITUTE OF TECHNOLOGY MADRAS" and the Sanskrit motto "सिद्धिमेवाय कर्मणि".

Content Management Systems

Example: Blog application

- CRUD for posts, comments
- Ratings for posts, comments
- User management
- Analytics

All these are data manipulation: can be independent of user interface!

So, with all of this, the one interesting thing sort of moving forward from the regular apps that we have been looking at, where everything is designed from scratch is to say that look, can we sort of think of a different kind of approach. And there we see that there are these things called Content Management Systems.

And let us, if you are working with let us say an example of a blog, where you want to create a blog. You would have the standard CRUD approach for creating posts and comments. You might have ratings that you want to associate with posts and comments; you will need to have some kind of user management, certain analytics.

But, you look closely at this and you realize that, the content management systems that we are used to are always something web based, they have a front end, they have a specific way, they have teams and so on. But, all this functionality is actually data manipulation; and it can be done independent of the user interface.

(Refer Slide Time: 04:25)

Wordpress

- One of the oldest and most popular
- Handles both data storage (backend) and templating (frontend)
- Also provides API: <https://developer.wordpress.org/rest-api/>

API can be used to build a CMS without frontend!



One of the oldest and most popular of such content management systems is what is called Wordpress. You are very likely have heard the term, you might even have used it at some point. It handles both the data storage as well as the frontend; in fact, if you are look at it Wordpress is implemented in PHP.

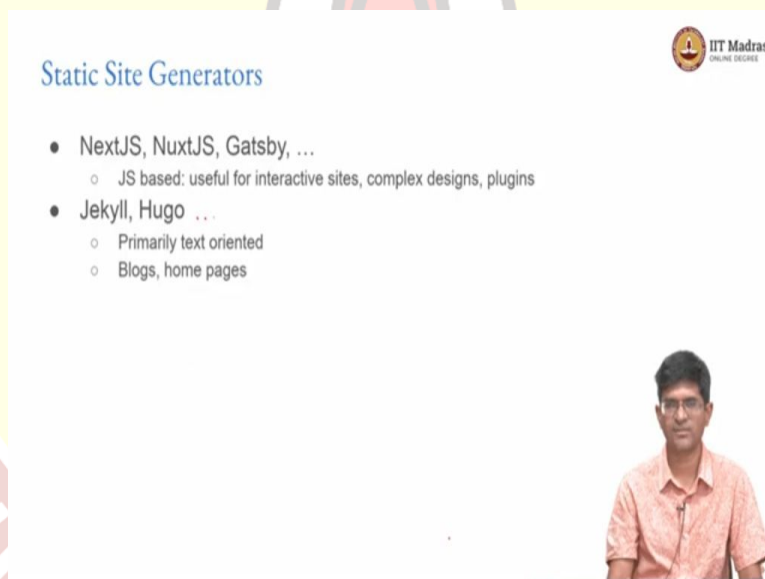
PHP is primarily meant for creating the front end. It turns out; it can also have the logic corresponding to the back end in it. But, the interesting thing about Wordpress and you can look at this document on their own website. It provides an API and this is part of the evolution of Wordpress.

Essentially, what it is saying is they have recognized the fact that people might have reasons for trying to use the data present in their CMS, without necessarily wanting the same frontend interface. So, an API in other words can be used to build a CMS without a frontend, which means that you can now provide your own frontend. You can create your own frontend using view or any other kind of technique that interfaces with this API. So, Wordpress is not the only one of course.

There are in fact, nowadays a large number of different such systems that are effectively saying, look, we are not even going to look at the front end. But, they still call themselves content management systems. And effectively what they say is they are what are called headless content management systems. So, a head headless content management system is something that just takes care of the data storage aspects, provide some kind of basic interface for managing the data storage.

But, the actual display and how you go about constructing the blog page or whatever the application that you want, is left up to the end user. And this is in some sense, a powerful abstraction; there are people who are good at managing the backend side. Let them focus on constructing the CMS in a headless manner and somebody else can then take care of specializing the frontend.

(Refer Slide Time: 06:31)



There are a number of such things that can be used with API's like this. You might already have come across a number of so called static site generators. There is NextJS, there is NuxtJS which is like next, but with view instead of react, there is Gatsby; most of these are JavaScript based. And essentially what they are trying to do is that they are saying that you can build an app actually; not just a website.

Why? Because NextJS for example, works with react; it allows you to not only have content, but also have JavaScript associated with it. Which means that you could create something with an

NextJS frontend that then interfaces with let us say, a Wordpress, API at the backend. But, now provides you with all the functionality, you want to create your own unique kind of blog application.

On the other hand, there are static site generators that are primarily text oriented; they just generate pages, not necessarily interactive. Jekyll and Hugo are examples; there are of course many others, I think. The last time I checked there is, there was in fact a web page called static site generators dot Net or something like that, which had more than 300 such examples. So, pretty much anyone can create their own static site generator, but these are the more popular ones these days.

And what is the idea behind it right? Once again, we are sort of going full circle, we started with plain HTML, went to introducing JavaScript, went overboard with JavaScript where everything had to have JavaScript and everything was dynamic. And then people so he came back and said, look that is making things too complicated and overloading your clients, and your servers too much.

Let us simplify and have the main page as just HTML, and use the JavaScript for additional interactivity and functionality as needed. So, it is sort of all these developments happen in cycles. And so once again, we are at this point where the static sites are the ones that really makes sense, in terms of handling large loads in an efficient manner.

(Refer Slide Time: 08:36)

Why SSGs?



- Servers can focus on delivering content
 - Static files faster to fetch
 - "compile time" optimization to reduce file transfer

"First Contentful Paint"

- Pure HTML allows easy transfer and parsing, can be displayed quickly



So, the server's can focus on delivering content and there is this term that is used which is called first contentful paint. These are things that we will be looking at in a little bit more detail when we look at measures of performance. And the idea behind first contentful paint is that when I go to a website, how quickly does it actually display something on the screen? And how do I know when there is something on the page for me to read?

And if I am transferring pure HTML, the chances are that I will be able to sort of very easily transfer it, parse it and display it without any complications; it can be displayed very quickly. So, static files generally tend to be faster to fetch, and I can do lots of compile time optimizations to reduce even what is being transferred. Even if there is JavaScript, I can strip it out and specify only some parts of it to be sent.

So, static site generators, in other words are a good way of approaching the construction of large sites. Are they necessarily the best way of constructing an app that depends on how you go about it? You could sort of use the static site generator to construct all the text and the documentation of the site. The actual core application would certainly need something like JavaScript.

(Refer Slide Time: 09:52)

JS hydration



- Static HTML transferred from server - no interactivity
- "Hydrate" the HTML with event handlers
 - Inject interactivity after initial rendering complete
- Delayed, but still fast enough
 - Good combination of speed and interactivity



And there is an interesting sort of take on that which is called hydration; its just terminology. But, it is an interesting way that of looking at it, which is to say that you first just send static HTML from the server. There is no interactivity but something gets displayed on the screen very fast, and then subsequently transferred the JavaScript that actually hydrates the page.

Meaning that there is some bootstrap JavaScript or some core JavaScript that is already there in the page, whose job is to pull in the other modules, take its own time over it. But, then make the page interactive. And all of this is a question of how you can sort of more efficiently build up a user experience that is closer to what would happen, if everything was running on your own local machine; and give you a better user interactivity and performance.

(Refer Slide Time: 10:50)



JAMStack: pinnacle of web app development?

- Takes care of storage + logic + presentation type apps
 - APIs flexible enough to handle any backend
 - Markup easy to change or compile
 - JS powerful enough to emulate any other behaviour
- Other developments?
 - Real-time communication
 - New interface devices, displays
- JAM approach general enough to extend with APIs
 - Until hit by performance issues
 - Wait for the Next Big Thing!



All of this ultimately is what you find when you look at what is called the JAM stack; the JavaScript, API's and markup which is what we were looking at all along. The JavaScript part we have looked at in a lot of detail and views what we sort of saw as a good frontend. API's: there is rest, there is GraphQL.

There are different ways by which you can retrieve data. And markup: there is of course the standard HTML markup, or you could use markdown and similar text based markup languages. But, all of them put together take care of all your problems; there is storage, logic and presentation.

So, the storage is the API, the logic is the JavaScript, and the presentation is the markup. All of these together are taking care of any app. In other words, that requires these three components to be present can be built up using this some variant of the JAM stack. And the thing to keep in mind is the JAM stack is not a single unique set of components; it is just anything that uses this approach.

And the, is this sufficient? Is that all you can ever really expect or require from an application that is going to be running on your system? The short answer is no. For two things, one is of course, what we have not looked at so far, but which can be handled to some extent is things like real time communication.

Actual chat, or even video or audio conversations other kinds of things, where you actually need to be able to directly interact with the remote system. There are ways by which it can be handled in JavaScript. There is in fact something called web RTCs, a web RTC, there are web sockets for directly pushing information from the server to the client and so on. So, there are a few more things which can be handled even with the existing JavaScript. But, what that is sort of telling you is that not everything is going to be a storage plus logic plus presentation type of app.

And more importantly, what happens as we come up with new interfaces? Holographic displays, or some kind of haptic display, haptic interfaces, where there is also a touch interface associated with how you use it. The vibrate mode on the phone is just a most basic form of a haptic interface that can be much more advanced versions of this. Three dimensional displays things that are immersive, virtual reality, all of those are these kind of techniques sufficient to take care of all of that, is of course the question that we have.

And the point is that the JAM approach at present is general enough that it can be extended in most cases through the use of APIs for interfacing. But, why was even JAM, why did JAM even come into existence? Because people felt that this whole idea of going back and forth between the server and the client was too slow, and did not give a good user experience. Which means that even though in principle JAM stack is capable of handling all of these different functionalities; there, this will only be as good as it works until you are hit by performance issues.

At which point it is entirely possible that people will come up with the next big thing? What is the next evolution going to be? We don't know, it is hard to predict at the moment. There might be things that sort of try to build up on the existing technologies or something fundamentally that disrupt what we have. At the moment though, this is a very good sort of set of technologies that are available for building web applications.