Good afternoon, everyone! My name is Sid, I’m a developer on both the CYENSE team and the MAP team. And I'm thrilled to be here today to talk about React, a JavaScript library for building dynamic user interfaces.

(next slide)

In today's brown bag session, we will explore the fundamentals of React, we'll have the opportunity to dive into practical examples, examine code snippets, and discuss best practices.

By the end of this session, you should have a better understanding of React's core concepts. Whether you're a seasoned developer or new to the world of React, this session aims to provide valuable insights and practical knowledge.

So, let's embark on this exciting journey into the world of React and discover how it can empower us to create engaging, interactive, and user-friendly web applications.

(next slide)

so what is react

react is a javascript library for

building out user interfaces

when you look at websites like facebook, Instagram, netflix, github

and airbnb you're looking at uis built in react

react provides us with a set of

tools and structure for building out

these user interfaces and makes this

process much faster and easier

(next slide)

With react it's

very common to build out single page applications

(next slide)

so in traditional

websites we have a template for each

page on our website and return that

template back to the user whenever they

request it

(next slide)

with single page applications

however we are working with one single

template and are simply updating

3 slides, stop at product slide)

all the components within the dom

The term single page

application is a bit misleading as it

could make you think there is only one page in

the application when really we're just

using one single template and modifying

all the contents within it

(next slide)

components are what make up the visual

layer of our application

(next slide)

and let us split up our ui into independent

reusable pieces

while how you build and

structure your application is completely up to you

traditionally each part of our

ui would be built out separately as its

own component and then added into a

parent component making up the ui for a

specific page

(next slide)

a component is a

javascript class or function that

returns back some html

this is actually something called jsx and we will learn more about that

in the next section

(next slide)

one thing to note

about components is that they can be

nested as deep as you want

a component

can hold another component and that

component can hold more components

(next slide)

instead of writing traditional html

tags we're going to be using something

called jsx

(next slide)

which stands for javascript xml

(next slide)

jsx actually looks a lot like html

with some slight syntax differences and

also gives us some added functionality

take a look at this example and you'll

see how you can use the curly braces to

pass in variables and adding javascript

logic directly into the html

jsx tags are actually very similar to html tags

some notable differences are

things like class declarations which are

written as class name and how event

handlers are added

(next slide)

browsers can't

actually read jsx so this code will

be run through a compiler and

convert it into traditional html and

javascript code once it's output in the browser

(next slide, skip title page)

using a react router is how we can have

multiple pages in a single page application

with react we typically

handle url routing with something called

(next slide

a router that keeps our ui in sync with

a url

because we're not actually

changing pages the router will take care

of rendering out components into the dom

based on the current url

(next slide)

when you need to pass data down from one

component to another

you can pass it down as a prop

(next slide)

a prop can be passed down

like a function parameter

once a prop is passed down into a component

you can now use that prop anywhere in the child component

(next slide)

state is simply a javascript object used

to represent information in or about a

particular component

(next slide)

we can use react hooks like use state to

create a component state

so let's imagine for a second that we

have a list of notes that we want to

render out in our app

we can set an

initial state and then map through that

state and output all that data

(next slide)

Best Practices

Let’s look at some best practices when working with react

(next slide)

One thing you might run into when defining a brand new component is that

you get an error when trying to return two

sibling elements together, that's because

every component can have only one root element

(next slide)

so what you might do is wrap it with a div

that works perfectly fine but

it leads to a bunch of unnecessary divs in your markup

and that can cause issues with accessibility and css styling

(next slide)

to address this concern

react has a built-in fragment component

(next slide)

or better yet you can use the shorthand syntax of

the fragment component which is

an empty element which tells react to

render nothing as the parent element

(next slide)

second

The second best practice I want to share is about

conditional rendering in React.

Conditional rendering is when you use a

Boolean variable to control

whether some code should render or not. One wat to do that is by using the AND operator, but it often leads to UI bugs, which can be easily avoided, and it’s often not mentioned.

How logical AND operator Works

In case you are not familiar with how the logical AND operator works,

in our example,

if authenticated is a truthy value, <ConditionalComponent /> (Welcome) is rendered

if authenticated is a falsy value, <ConditionalComponent /> is not rendered

Why is that? It’s nothing React specific but rather a behavior of JavaScript and other programming languages called short-circuit evaluation

if the first operand (condition) is falsy, the AND operator (&&) stops and it does not evaluate the second condition (<ConditionalComponent/>).

Why Not To Use “&&”

The short syntax of AND operator is often preferred and it works. But! Just because you can doesn’t mean you should.

In our example, if condition evaluates to true or false, you get what you’d expect — <ConditionalComponent /> (Welcome) is or is not rendered respectively. All good so far.

However, if condition doesn’t evaluate to a boolean, it may cause trouble.

(next slide)

For example:

if condition is 0, 0 is displayed in the UI

To avoid showing something you don’t want in the UI, such as 0, that could mess up the layout,

(next slide)

use the ternary operator instead,

(quickly explain ternary operator)

(next slide)

So to prevent avoidable UI bugs

use the ternary operator for conditional rendering of React components

instead of the AND operator

(next slide)

Often there are scenarios where you pass boolean props to a component. I've seen a lot of people doing something like this

Where you pass the boolean props

But you don't need to do it necessarily like this because the occasion of the prop itself is either truthy (if the prop is passed) or falsy (if the prop is missing).

A cleaner approach would be:

To Just pass the variable

LAST PART

And that brings us to the end of our session on React,

a journey into the world of building dynamic user interfaces.

I hope you found today’s brown bag session insightful and gained a clearer understanding

of React's core concepts and best practices.

(next slide)

Before we conclude,

I would like to open the floor for any questions, comments, thoughts, or experiences you would like to share regarding React.

[Pause and allow for questions/comments]

(Thank you all for your active participation and engagement. I appreciate your valuable contributions to our discussion.)

(next slide)

If you have any further questions or thoughts that arise after the session,

please feel free to reach out to me or Ravi.

Thank you all for attending today's session, I hope you enjoy the rest of your day.