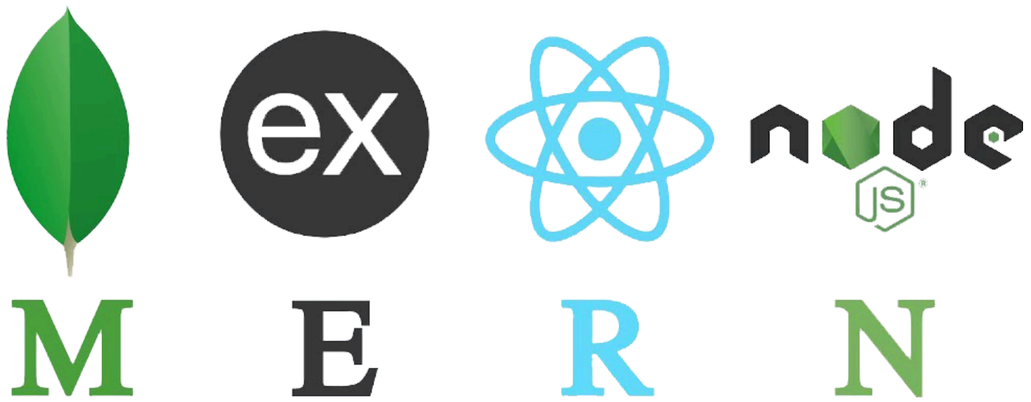
MERN LAB 3
AUTHENTICATION & STATE MANAGEMENT



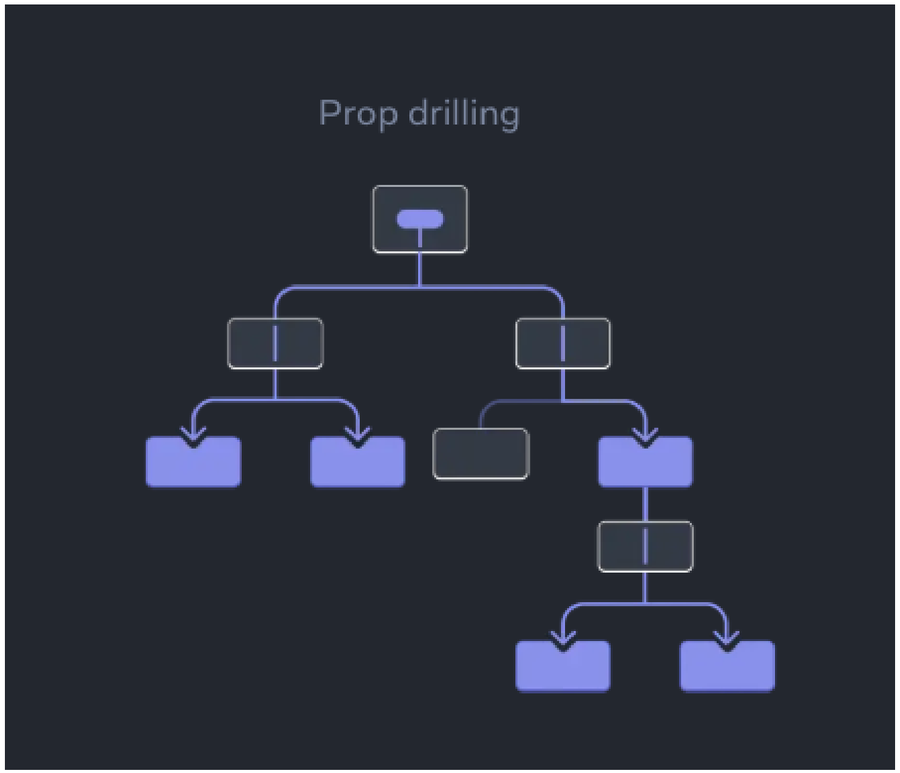
SOFTWARE SYSTEM DEVELOPMENT

20/09/2025

AGENDA

- PROP DRILLING
- CONTEXT API
- REDUX
- JWT
- AUTHENTICATION
- AUTHORIZATION

PROP DRILLING



- THE PROCESS OF PASSING DATA FROM A TOP-LEVEL COMPONENT DOWN TO DEEPER LEVELS
- HAPPENS WHEN A PIECE OF STATE NEEDS TO BE ACCESSIBLE BY A COMPONENT DEEP IN THE COMPONENT TREE
- GETS PASSED DOWN AS A PROP THROUGH ALL THE INTERMEDIATE COMPONENTS.

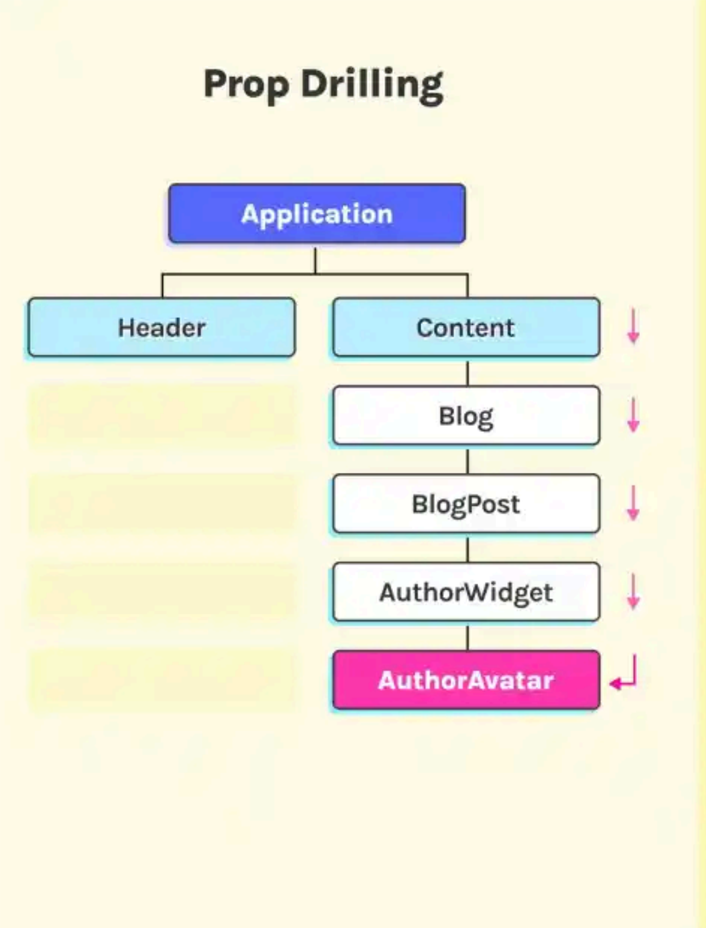
WHY PROP DRILLING?

- **State Management**:
  - Prop drilling is often used to manage state in a React application.
  - By passing state down through the component tree, you can share data between components.

- **Simplicity**:
  - Prop drilling keeps the application structure simple and makes it easier to understand the flow of data.
  - It's a straightforward way of handling data without introducing more complex tools.

```jsx
// Top-level component
function App() {
    const data = "Hello from App component";
    return <ChildComponent data={data} />;
}

// Intermediate component
function ChildComponent({ data }) {
    return <GrandchildComponent data={data} />;
}

// Deepest component
function GrandchildComponent({ data }) {
    return <p>{data}</p>;
}

```

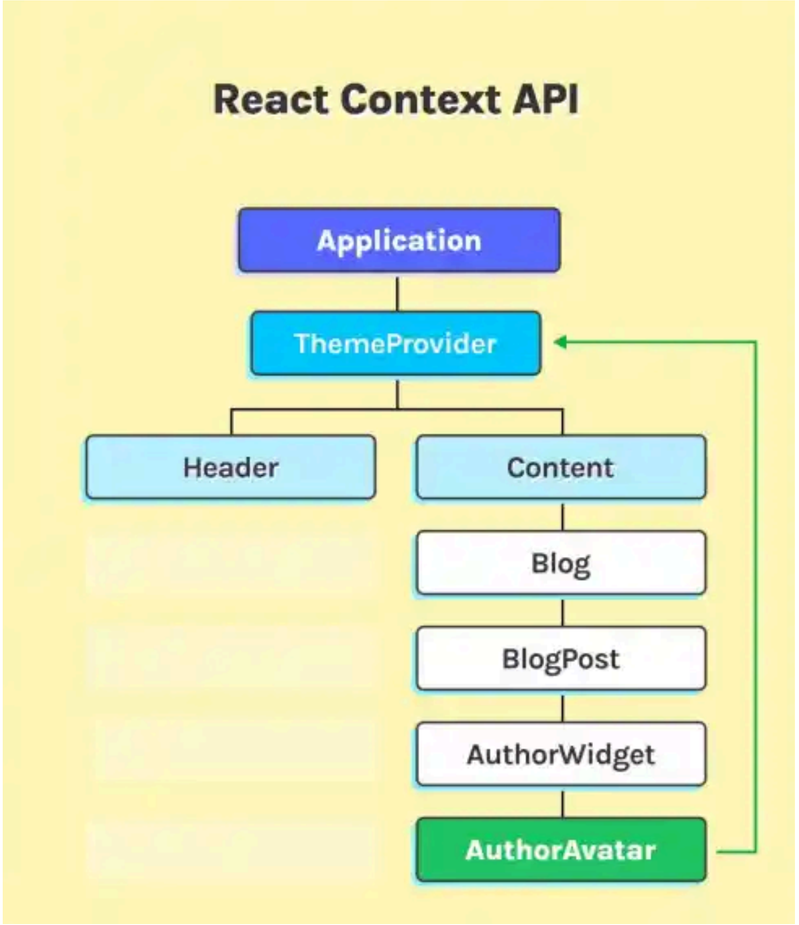
ISSUES WITH PROP DRILLING?



- **Readability**: Prop drilling can make the code less readable, especially when you have many levels of components.
- It’s **difficult to trace** where a particular prop is coming from.
- **Maintenance**: If the structure of the component tree changes, and the prop needs to be passed through additional components, it requires modifications in multiple places.

CONTEXT API IN REACT ?

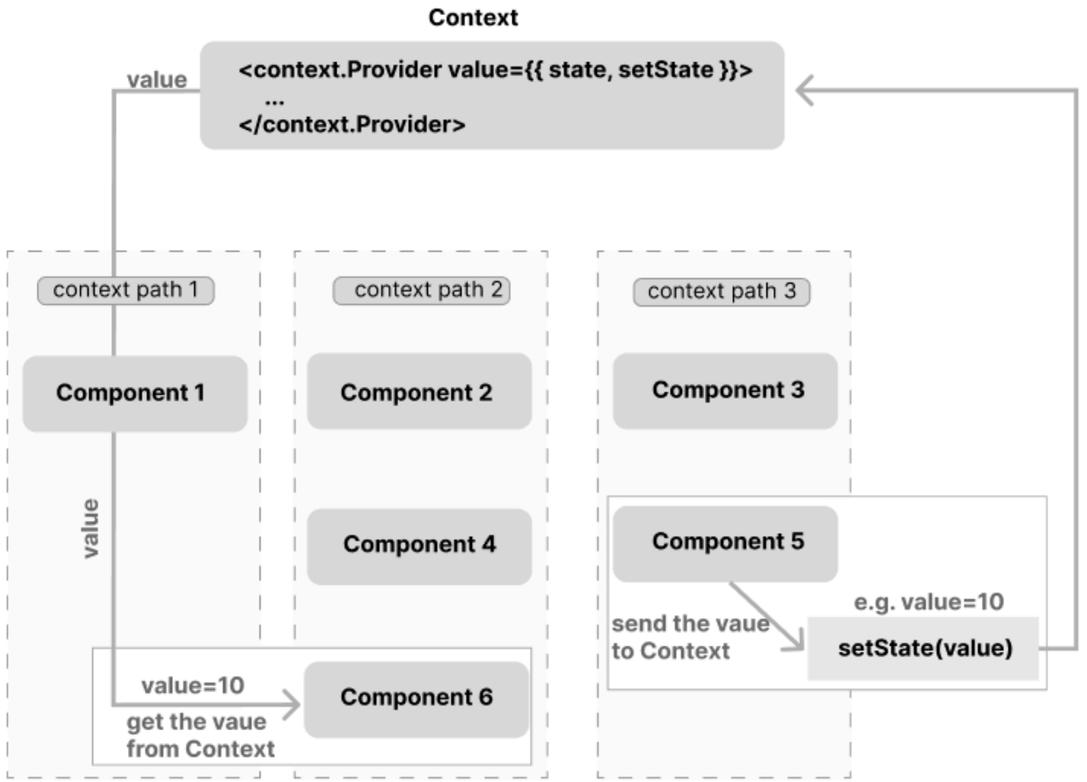
- The Context API is a form of Dependency Injection.
- A feature in React that provides a way to share values like props between components without explicitly passing them through each level of the component tree.
- Solves the prop drilling problem by allowing data to be accessed by components at any level without the need to pass it through intermediate components.



SOURCE: MARK'S DEV BLOG

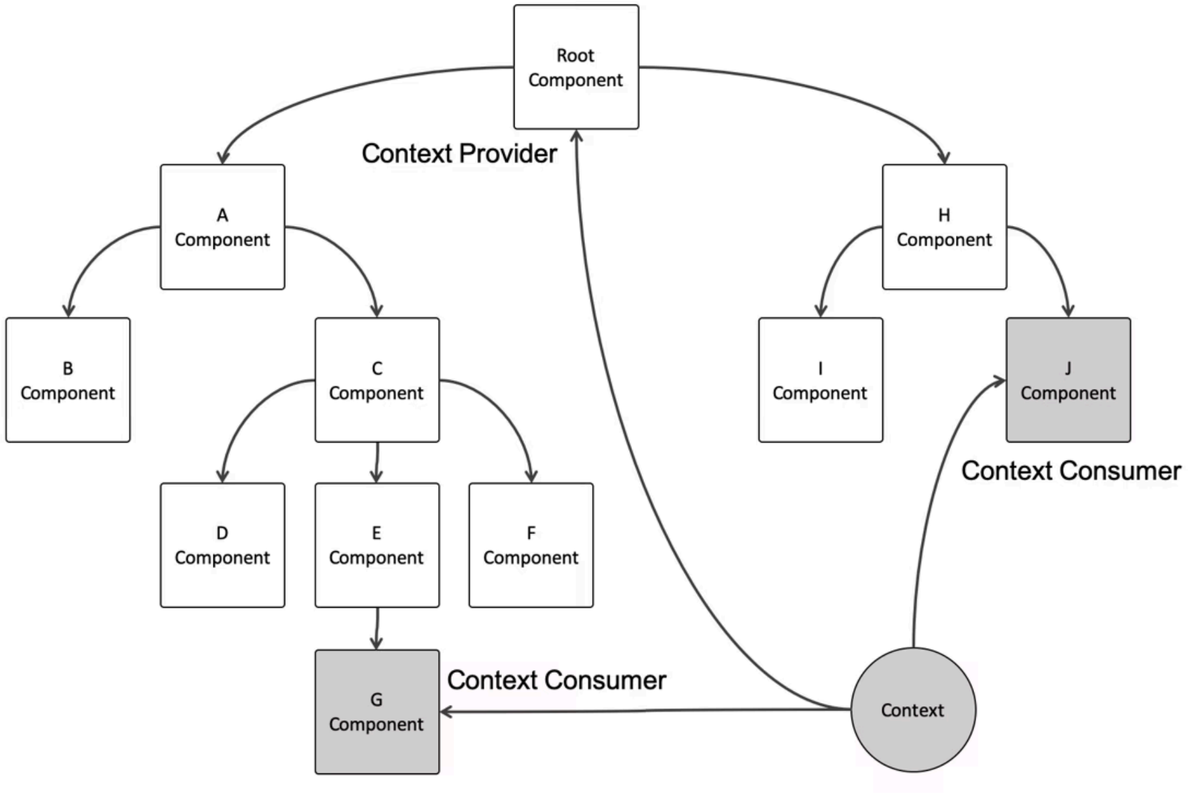
COMPONENTS OF CONTEXT API

- **createContext**(): The function is used to create a context. Returns an object with two components - Provider and Consumer.
- **Provider:** Responsible for providing the context value to its descendants. It is placed at the top of the component tree.
- **Consumer (or useContext hook):** Allows components to consume the context value. Alternatively, the useContext hook can be used for a more concise syntax.



SOURCE: HTTPS://REACT.DEV/LEARN/PASSING-DATA-DEEPLY-WITH-CONTEXT

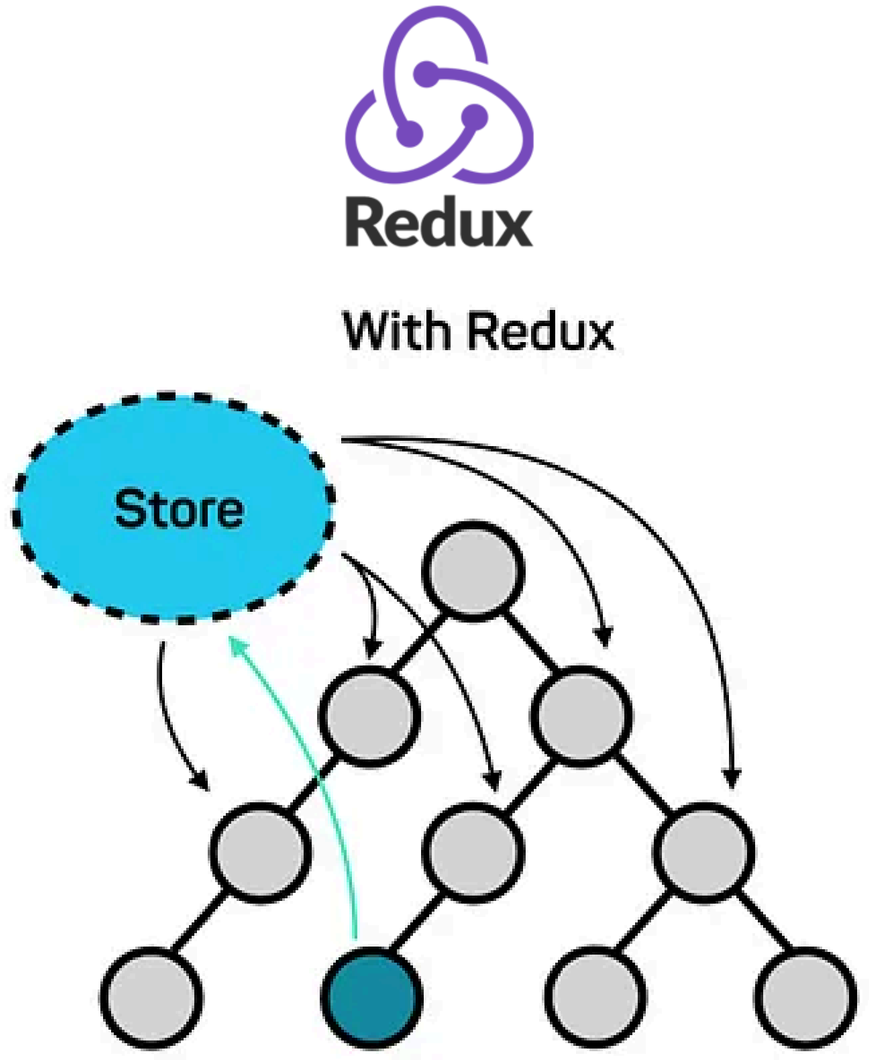
PROS OF CONTEXT API



- **Avoids Prop Drilling**: Context API eliminates the need for passing props through intermediate components, making the code cleaner and more maintainable.
- **Global State**: It allows you to manage global state that can be accessed by components across the application.
- For complex state management needs, additional tools like **Redux** might be more suitable.

WHAT'S REDUX?

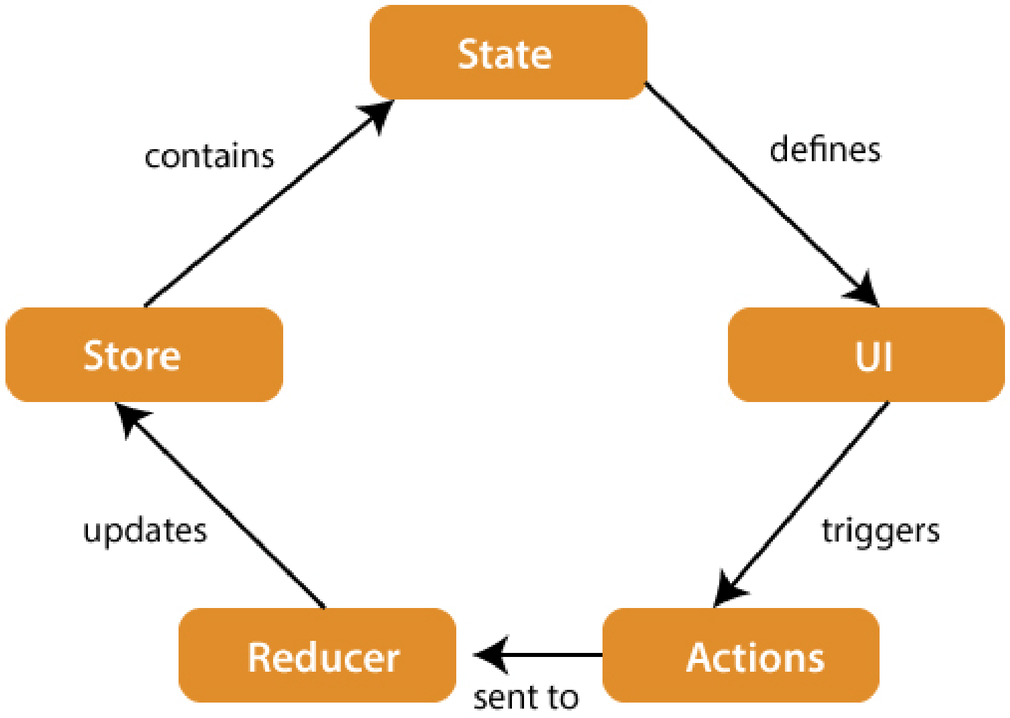
- Redux is a powerful state management library often used with React.
- It introduces a global store and follows a unidirectional data flow.
- Redux provides more features than Context API
- Redux comes with additional concepts and boilerplate.
- Middleware support, time-travel debugging, broader ecosystem.



Source: https://redux.js.org/

WHY REDUX?

- **Complexity**: Context API is simple and built into React, Redux is powerful but comes with additional complexity.
- **Scalability**: Often preferred for larger applications due to their ability to manage complex state logic.
- **Community Support**: Redux has a large and established community with a wide range of middleware and tools.
- **Context API for Simplicity; Redux for Scalability**



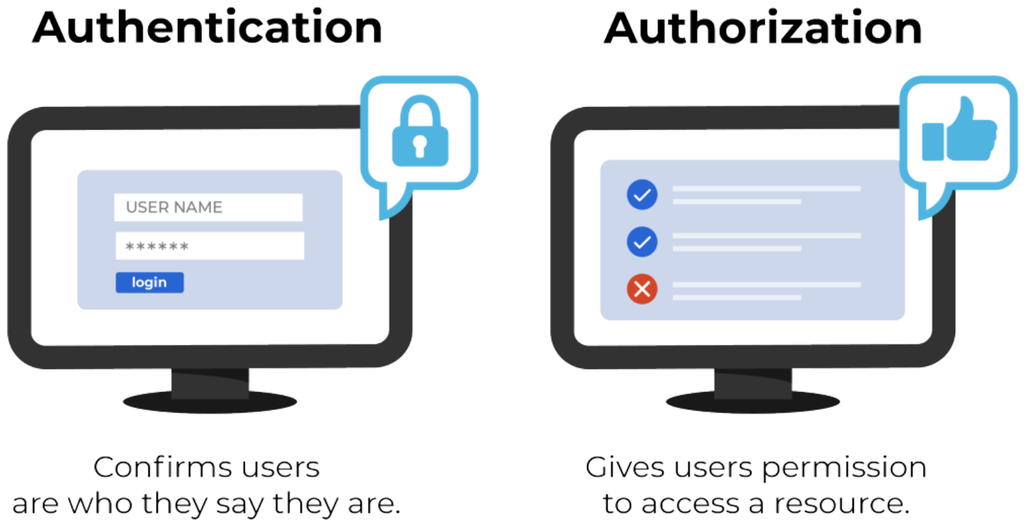
EXAMPLE - REDUX IMPLEMENTATION

```jsx
// Store creation
import { createStore } from 'redux';

const initialState = { user: { name: 'John' } };

const rootReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'CHANGE_NAME':
      return { ...state, user: { name: 'Jane' } };
    default:
      return state;
  }
};

const store = createStore(rootReducer);

// Accessing Redux state in a component
function Profile() {
  const user = useSelector((state) => state.user);
  const dispatch = useDispatch();

  return (
    <div>
      <p>Welcome, {user.name}</p>
      <button onClick={() => dispatch({ type: 'CHANGE_NAME' })}>Change Name</button>
    </div>
  );
}
```

AUTHENTICATION & AUTHORIZATION

- The process of confirming that a user is who they claim to be.

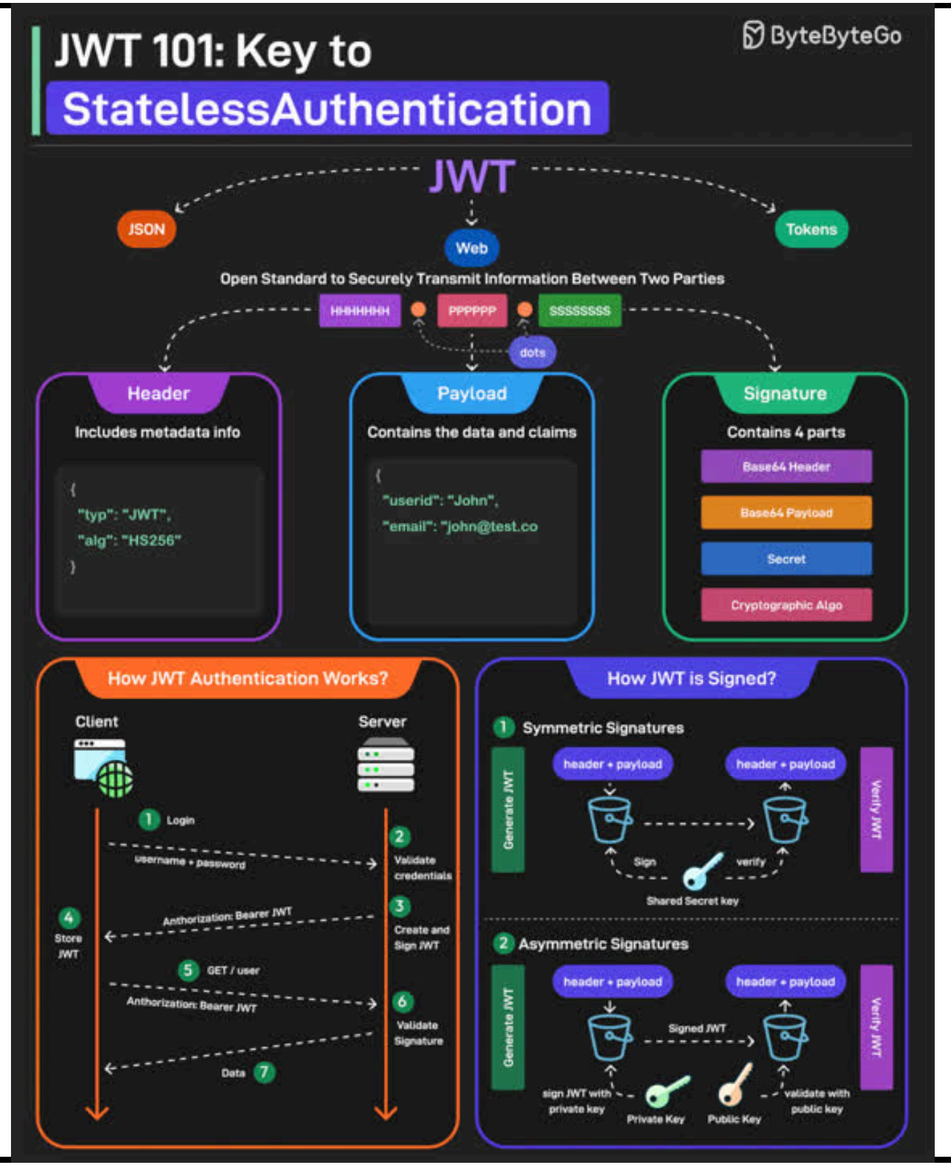
- **Identification types**:
  - Username+Password
  - Single Sign-on (SSO)
  - Multi-Factor Authentication (MFA)
  - Passwordless Authentication (biometric, security keys)



okta

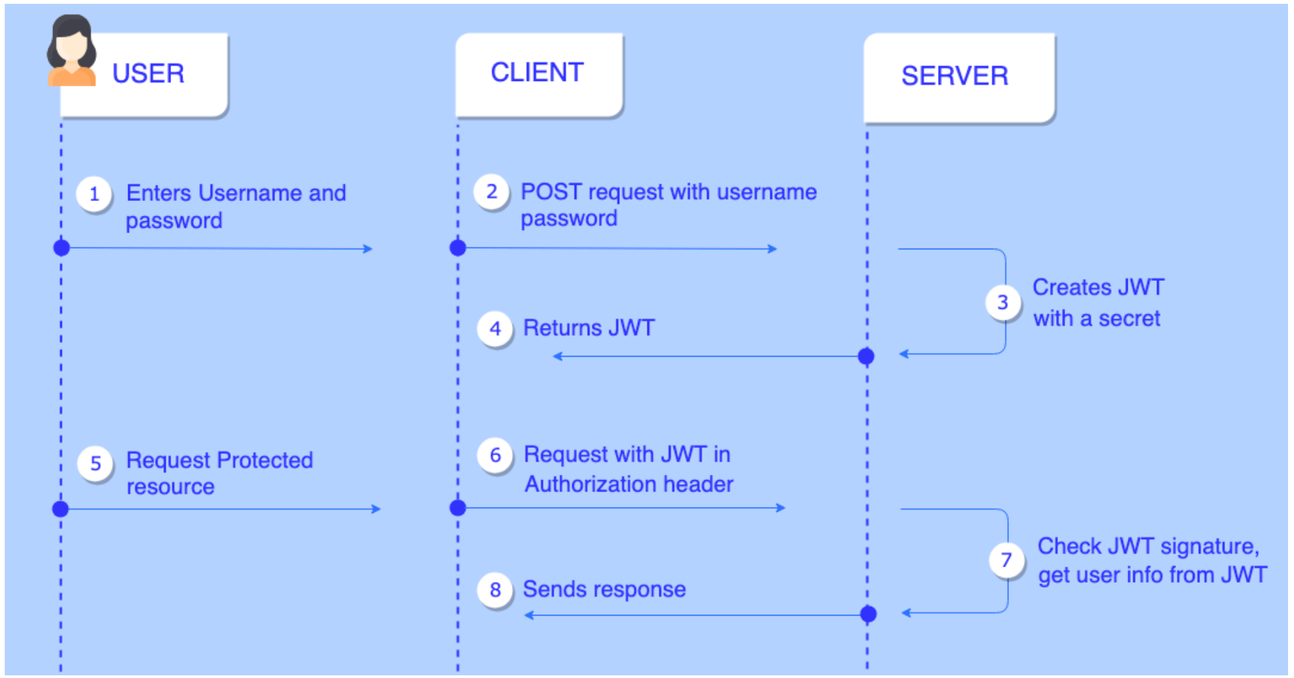
WHAT IS JWT?

- JSON Web Token is an open standard (RFC 7519)
- JWT defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- Signed tokens can verify the integrity of the claims contained within it.
- Single Sign On (SSO) is a feature that widely uses JWT.
- JWT is used at the Internet scale



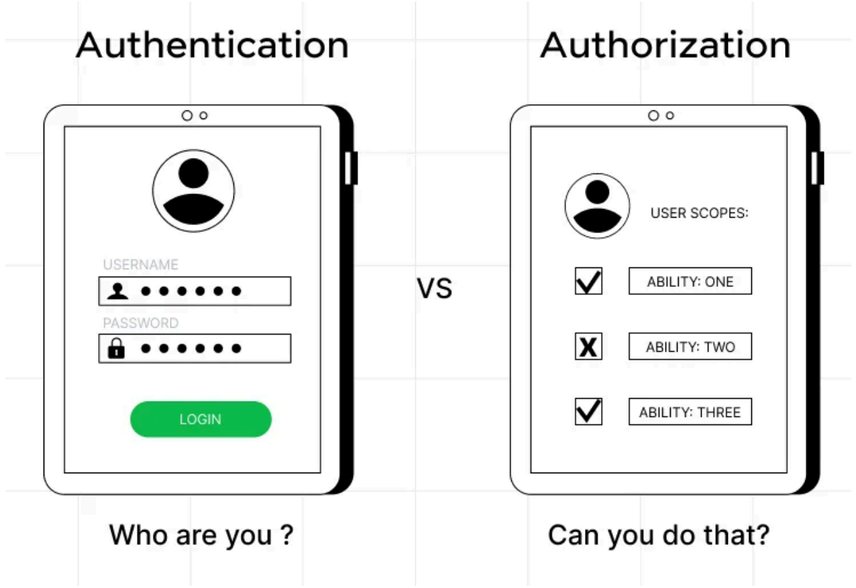
Source: https://bytebytego.com/guides/jwt-101-key-to-stateless-authentication/

AUTH USING JWT & LOCALSTORAGE



1. Client sends a POST request to the server with the user's details.
2. The server validates the provided information.
3. The server generates a JWT containing the user's information.
4. The server sends the generated JWT back to the client.
5. The client stores the received JWT in the browser's localStorage.

AUTHORIZATION



The process of determining what actions a user is allowed to perform after authentication.

- **Role-Based Access Control (RBAC)**

(e.g., admin, teacher, student, parent)

- **Permission-Based Access Control**

(granular feature-level permissions)

- **Attribute-Based Access Control (ABAC)**

(decisions based on user attributes, context, policies)

Implemented via:

- Protected Routes (React Router)
- Conditional Rendering (UI based on roles/permissions)
- Backend APIs enforcing policies

FURTHER READING

- **STATE MANAGEMENT:**
  - https://dev.to/chintanonweb/say-goodbye-to-prop-drilling-learn-usecontext-in-react-like-a-pro-451e
  - https://www.frontendeng.dev/blog/49-what-is-prop-drilling-in-react
  - https://react.dev/learn/managing-state
  - https://react.dev/learn/passing-data-deeply-with-context
  - https://redux.js.org/
  - https://www.freecodecamp.org/news/what-is-redux-store-actions-reducers-explained/
- **AXIOS**:
  - https://axios-http.com/docs/intro
  - https://medium.com/@theNewGenCoder/welcome-to-axios-101-mastering-api-requests-eb45638363d1
- **AUTHENTICATION & AUTHORIZATION**
  - https://www.jwt.io/introduction#what-is-json-web-token
  - https://auth0.com/docs/secure/tokens/json-web-tokens
  - https://www.descope.com/blog/post/oauth2-react-authentication-authorization
  - https://clerk.com/blog/oauth2-react-user-authorization
  - https://learn.microsoft.com/en-us/entra/identity-platform/tutorial-single-page-app-react-prepare-app?tabs=workforce-tenant

THANK YOU!