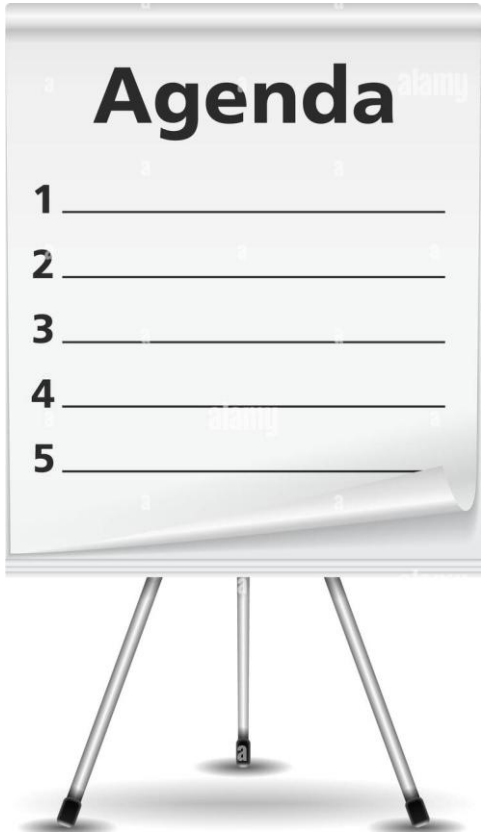
Welcome to the Apprenticeship Program – Session 33

Agenda

- Authentication vs Authorization
- Login Flow with JWT
- Storing & Using JWT Tokens
- Protecting Routes (Guards)
- Role-Based Access Control (RBAC)



Security Basics

- Authentication = Verifying who the user is (login)
- Authorization = Controlling what the user can access

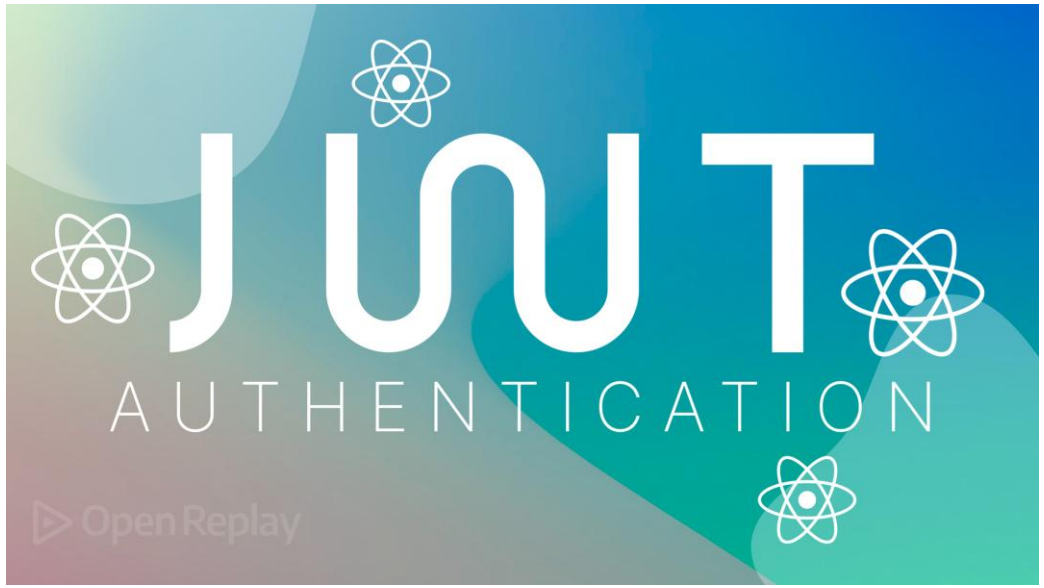
📦 Example:

  - Login = Authentication
  - Admin access = Authorization

JWT (JSON Web Token) Basics

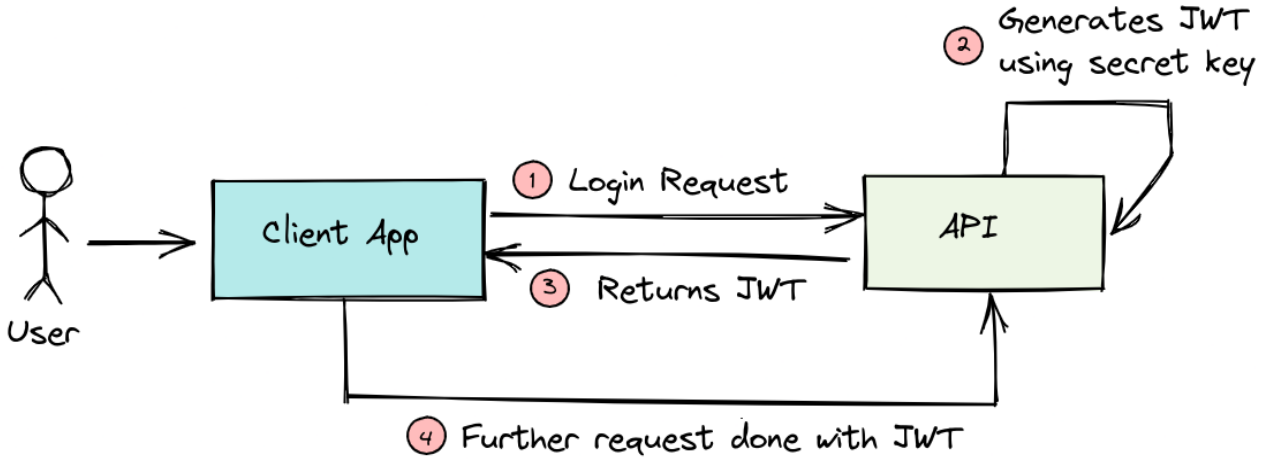
- JWT is a secure, compact token format used to pass authentication data
- Structure: Header.Payload.Signature
  - Example Payload:

```
{
  "sub": "123456",
  "role": "admin",
  "exp": 1712345678
}
```



🔑 Implementing Login with JWT

- User submits login form
- App sends credentials to backend
- Backend responds with JWT token
- Token is stored in localStorage or sessionStorage
- Token is sent with every API request for auth



Storing JWT Securely

- After successful login:

```
localStorage.setItem('token', jwtToken);
```

❗ Avoid storing sensitive data in the token

🔐 For better security, use HTTP-only cookies on backend for storage

Sending JWT in HTTP Requests

- Use an Angular HTTP Interceptor to attach the token to outgoing requests:

```
intercept(req: HttpRequest<any>, next: HttpHandler) {
  const token = localStorage.getItem('token');
  const cloned = req.clone({
    headers: req.headers.set('Authorization', `Bearer ${token}`)
  });
  return next.handle(cloned);
}
```

✅ Automatically adds JWT to each request

DEMO

🚦 Protecting Routes with Guards

- Using CanActivate for Auth Protection:
  - Guard Example:

```
canActivate(): boolean {
  return !!localStorage.getItem('token');
}
```

  - Route Example:

```
{ path: 'dashboard', canActivate: [AuthGuard], component: DashboardComponent }
```

✅ Only logged-in users can access the route

👥 Role-Based Access Control (RBAC)

- **RBAC :**
  - Check user role from token:

```
const token = localStorage.getItem('token');
const payload = JSON.parse(atob(token.split('.')[1]));
if (payload.role === 'admin') {
  // allow access
}
```

  - Usage in Guards:

```
canActivate(): boolean {
  return this.authService.hasRole('admin');
}
```

✅ Hide or show content based on user role

Show/Hide UI Based on Role

```
<div *ngIf="userRole === 'admin'">
  <button>Delete User</button>
</div>
```

🎯 Use Angular templates to enforce UI-level permissions in addition to route guards

DEMO

Conclusion and Q&A

- **Key Takeaways:**
  - Authentication: Login + JWT token.
  - Authorization: RBAC using token payload.
  - Use interceptors to send tokens automatically.
  - Use route guards to protect views.
  - Use template logic to control UI access.

Questions? 👀