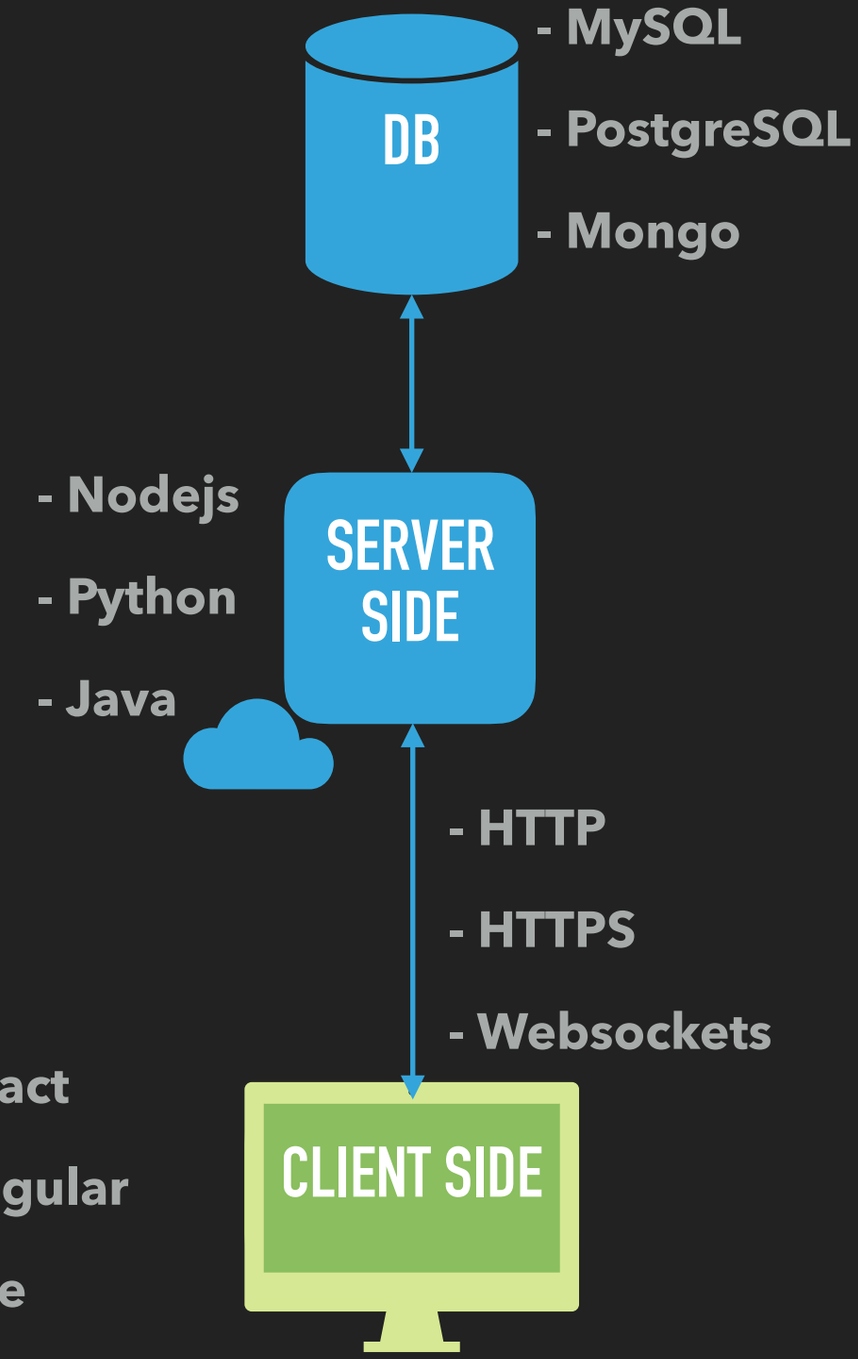
SERVER SIDE

AUTH & SECURITY

WEB DEVELOPMENT SPECTRUM

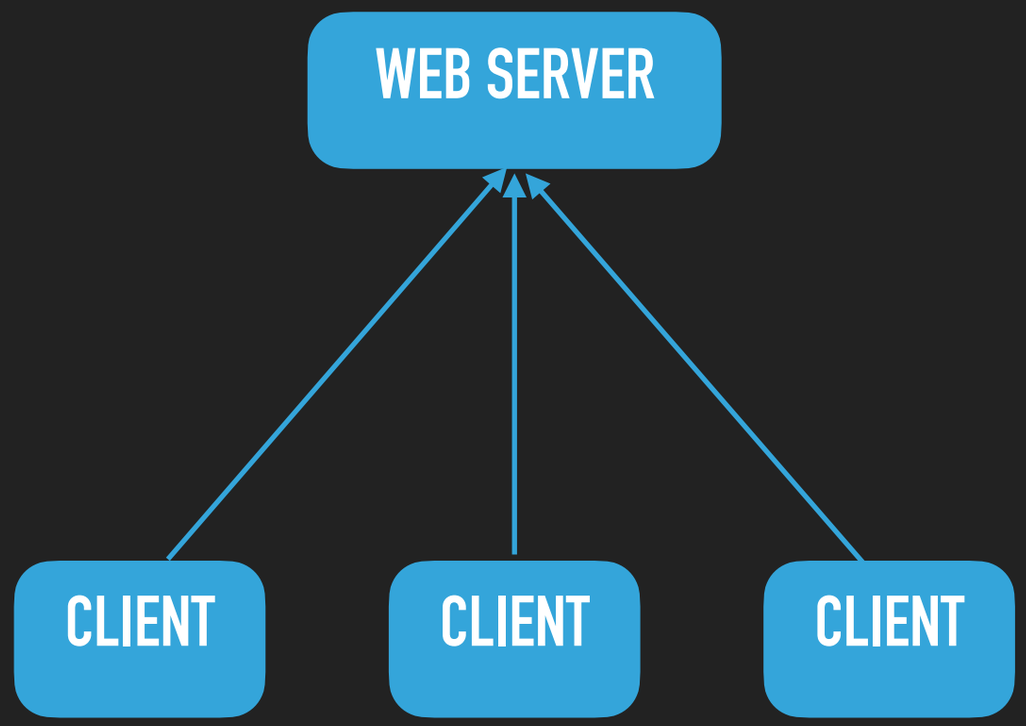
- Client side technologies
  - React, Angular, Vue, etc
- Server side
  - Node, Python, Java
- Database
  - MySQL, Mongo, etc
- Protocols
  - HTTP(S), WebSockets



LOGIN

WHAT DOES IT MEAN?

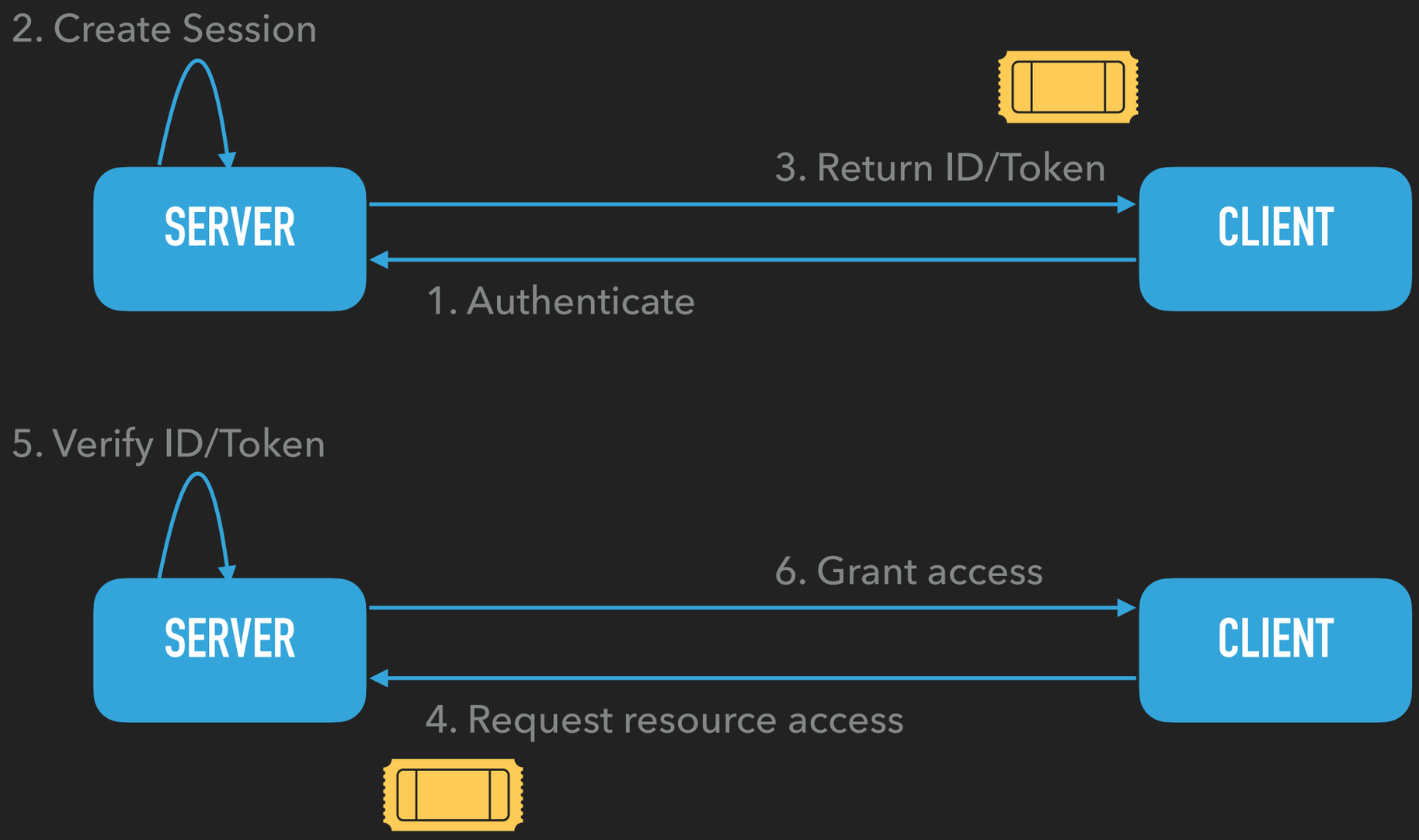
- For Server
  - login process is the way a user or system verifies their identity to gain access to server resources/services
  - Server knows how to identify a user/system which is requests for resources/services
- For Client
  - login process is the mechanism through which a user authenticates themselves to access resources or services
  - Client has a mechanism to prove identity and access resources/services
- Examples of resources/services
  - Bank account, banking services, FB account, Images, Videos, Email Inbox, Email service



WHEN A USER LOGS IN

- **Authentication**
  - The server verifies that the user is who they claim to be
    - username and password
    - two-factor authentication
    - biometrics
- **Authorization**
  - Once authenticated, the server checks what resources or actions the user is permitted to access
  - level of access or permissions the user has based on roles or specific rules set by the application
- **Session Creation**
  - The server establishes a session for the user
  - Unique session ID or token
  - This ID/token allows the server to track user activities without re-authenticating on every request
  - Sessions may expire after some time for security purposes
- **Resource Access**
  - Client stores the ID/token and passes it to the server to request for resources/services
  - Server identifies the user using the ID/token and gives access to resources/services

RESOURCE ACCESS



SESSION

- **Server side sessions**
  - Session Id stored on the server
  - Database or in-memory
  - Session Id mapped to user Id and roles
  - Pros
    - Centralised control - Sessions can be centrally invalidated following a security event or logout
    - Enhanced Security - Sensitive info, like roles of a user, not exposed to client.
    - Smaller payloads
  - Cons
    - In a distributed system, session management becomes complex
    - Scalability challenges - Server-side session storage can become difficult to scale across multiple servers, requiring a shared session store or distributed cache
    - Resource-Intensive - Server memory or database storage is required to manage each active session

SESSION

- **Client side sessions**
  - Token based (JWT)
  - The server doesn’t need to store any session information
  - Session information is encoded in the token
  - Pros
    - Stateless Architecture - Server doesn’t need to store any session information, making it easier to scale
    - Reduced Server Load - Because sessions are managed client-side, the server doesn’t have to keep track of sessions, reducing memory and storage requirements.
    - Decentralised Verification - Any service with the JWT verification key can verify the token, simplifying authorization
  - Cons
    - Limited Control Over Sessions - JWTs cannot be easily invalidated or revoked on the server side, making logout or session expiration more challenging.
    - Potential for Larger Payloads - JWTs often carry encoded user information, which can result in larger payloads
    - Risk of Token Tampering or Leakage - Storing user data client-side increases exposure to potential token theft, or tampering. Strong token encryption and secure storage are essential.

SIMPLE LOGIN

- Server stores username, password
  - Passwords cannot be stored as plain text
    - Higher risk
    - Illegal in most countries
- Two common ways to store passwords
  - Hashing
  - Hashing + Salting

HASHING

- Hash of the password is stored
- When the user provides the password to login, the raw password is hashed and the hashes are compared
- MD5, SHA-3, etc
- Disadvantages
  - Same passwords map to same hash
  - Tables can be used to crack common passwords

SALTING

- **Salt** is a value generated by a cryptographically secure function
- Added to the input of hash functions to create unique hashes for every input
- The same salt has to be provided at the time of comparing the raw password with the hashed value

- BCrypt
  - Cryptographic hashing function designed specifically for securely hashing passwords.
  - Salt is automatically included as part of the hashed password.
  - The resulting hash contains information about the salt used, so there's no need to store the salt separately in the database
  - https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCrypt.html
- https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/

HTTPS

WHAT IS HTTPS?

- SSL/TLS layer over HTTP
- The SSL layer has 2 main purposes
  - Verifying that the client is talking directly to the server that it thinks it is talking to
  - Ensuring that only the server can read what the client sends it and only the client can read what it sends back
- Anyone can intercept every single one of the messages and still not be able to read any of the actual data
- https://robertheaton.com/2014/03/27/how-does-https-actually-work/

HOW IS THE CONNECTION ESTABLISHED?

- An SSL connection between a client and server is set up by a handshake
- 3 Steps
  - Hello
    - The handshake begins with the client sending a ClientHello message.
    - This contains all the information the server needs in order to connect to the client via SSL, including the various cipher suites and maximum SSL version that it supports.
    - The server responds with a ServerHello
    - a decision based on the client's preferences about which cipher suite and version of SSL will be used
  - Certificate Exchange
    - Now that contact has been established, the server has to prove its identity to the client
    - The server sends its SSL certificate. It contains the following
    - the name of the owner, the property (eg. domain) it is attached to, the certificate's public key, the digital signature and information about the certificate's validity dates
    - Client verifies the SSL certificate
      - The client uses the public key of the certificate authority (CA) that issued the certificate to verify the digital signature

HOW IS THE CONNECTION ESTABLISHED?

- Key Exchange
  - The encryption of the actual message data exchanged by the client and server will be done using a symmetric algorithm
  - The details of which was already agreed during the Hello phase
    - Both parties need to agree on this single, symmetric key, a process that is accomplished securely using asymmetric encryption and the server's public/private keys
    - The client generates a random key to be used for the main, symmetric algorithm
    - It encrypts it using an algorithm also agreed upon during the Hello phase, and the server's public key
    - Encrypted key is then sent it to the server
    - It is decrypted using the server's private key

OAUTH

WHAT IS OAUTH?

- Open Authorization
- Framework that enables secure third-party access to a user's data without the need for the user's credentials to be shared directly.
- Uses authorization tokens to prove an identity between consumers and service providers
- User logs into one platform and uses tokens generated by that platform to grant access to data and perform actions in one or more other applications
- There are different modes of OAuth
- https://fusionauth.io/articles/oauth/modern-guide-to-oauth

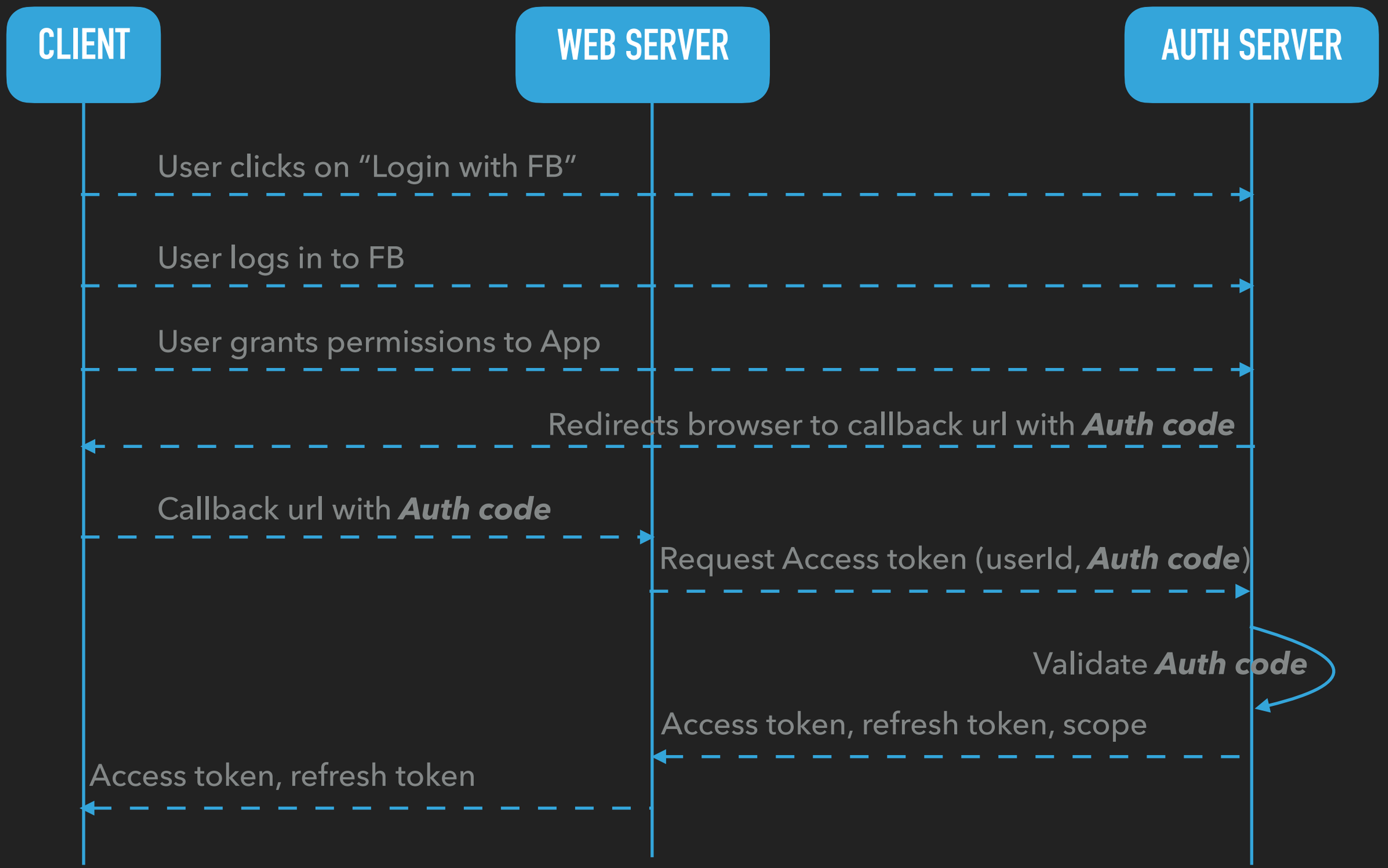
THIRD PARTY LOGIN

- We will look at third party login and registration mode
- Avoid storing any credentials
- Use Authentication of a third party like Google, Facebook, etc
- “Login with …”
- your application will need to use one or more APIs from the OAuth provider in order to retrieve information about the user
- or do things on behalf of the user (Like sending a message on behalf of the user)
- the user has to grant your application permissions
- the third-party service usually shows the user a screen that asks for certain permissions.

FLOW

1. A user visits TWGTL (The world's greatest TODO List) and wants to sign up and manage their ToDos.
2. They click the "Sign Up" button on the homepage.
3. On the login and registration screen, the user clicks the "Login with Facebook" button.
4. This button takes them over to Facebook's OAuth server.
5. They log in to Facebook (if they aren't already logged in).
6. Facebook presents the user with the permission grant screen based on the permissions TWGTL needs. This may not be done every time the user logs in.
7. Facebook redirects the browser back to TWGTL, which logs the user in.
8. TWGTL server also calls Facebook APIs to retrieve the user's information and validate.
9. The user begins using TWGTL and adds their current ToDos.
10. The user stops using TWGTL; they head off and do some ToDos.
11. Later, the user comes back to TWGTL and needs to log in to check off some of their ToDos. They click the My Account link at the top of the page.
12. This takes the user to the TWGTL login screen that contains the "Login with Facebook" button.
13. Clicking this takes the user back to Facebook and they repeat the same process as above.

FLOW


CLIENT
WEB SERVER
AUTH SERVER
User clicks on “Login with FB”
User logs in to FB
User grants permissions to App
Redirects browser to callback url with Auth code
Callback url with Auth code
Request Access token (userId, Auth code)
Validate Auth code
Access token, refresh token, scope
Access token, refresh token

ACCESS TOKEN

- Access tokens are used in token-based authentication to allow an application to access an API
- Access tokens are issued by an authorization server after a successful authentication and authorization process
- Access tokens have a limited lifespan, and their duration is determined by the authorization server during issuance
- The client application receives an access token
- The client application passes the access token as a credential when it calls the target API
- The passed token informs the API that the bearer of the token has been authorized to access the API
- Perform specific actions specified by the Scope that was granted during authorization.
- https://auth0.com/docs/secure/tokens/access-tokens

REFRESH TOKEN

- A refresh token is a credential used to obtain a new access token
- It is part of the OAuth 2.0 authorization framework and is commonly used to extend the validity of an access token without requiring the user to reauthenticate.
- Refresh tokens are issued by the authorization server along with the access token
- Refresh tokens typically have a longer lifespan compared to access tokens
- When an access token expires, the client application can use the refresh token to request a new access token from the authorization server without requiring the user to re-enter their credentials
- https://www.loginradius.com/blog/engineering/guest-post/what-are-refresh-tokens-and-when-to-use-them/

JWT

- JSON Web Token (JWT) access tokens conform to the JWT standard
- Compact
- A valid JWT confirms that the user/entity has been authenticated and has specific permissions or claims
- Secure
  - Server can easily verify if the token has been tampered with
- JWT has 3 parts
  - The ***header*** typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA
  - The ***payload*** contains the claims. Claims are statements about an entity (typically, the user) and additional data
  - The ***signature*** is created by combining the encoded header, encoded payload, and a secret key. The signature ensures the integrity of the token.
- https://jwt.io/
- https://auth0.com/docs/secure/tokens/json-web-tokens