Web Security

HTTPS / User Registration / Login

Registration Process

1. Connect to website over SSL / HTTP Port 443 - provides encryption between client (web browser) and server (web server).

2. Displays form and user creates account with some ID like an email address that must be unique and then the user types password and the password must be hashed. Hashing happens so that you can’t read a users password in the database in plain text

Login

1. Connect on SSL to Encrypt the network communication between client and server
2. User types their ID and Password.
3. Password is hashed again and compared to the store password hash created a registration
4. If this matches then a session will be created. Meaning the a session ID cookie is placed on the client and then the web server uses this to differentiate requests coming different browsers that “logged” in to a session.
5. The program you write has use the session ID to match to a a username. This is what essentially “Flask Login” is doing for you.

Web security relies mainly on…

Hashing

1 Way

Allows you to take a string of text characters that is human readable and then turn it into scrambled text that is difficult or impossible to turn back into the original text. There are various algorithms that have pros and cons; however, we will be using Bcrypt which is currently one of if not the most secure hashing algorithm used today.

Encryption

Encryption allows a sender and receiver to encode and decode messages sent across an insecure method of transport. It relies on a secret key shared between the sender and receiver. There are various encryption algorithms. The first known use of encryption was that it was used by Julius Caesar to communicate from the battle field to Rome.

Web Security basics - Data Security

1. HTTPS - Secures communication over the Internet using SSL which is a type of encryption
   a. Note: this make sure that as text travels back and forth between the client and server that it cannot be read by anyone that may be able to capture the network communication between the client and server
   b. HTTPS Uses the RSA algorithm which is a public private key.
2. Password Hashing
   a. Note: this make sure that if someone gains access to the database either as a hacker or even authorized user that they cannot easily determine any users password

SSL / TLS

Certificate - Used to generate the private / public key pair

Private Key - used on the server and is secret / hidden and used to encrypt outgoing message and decrypt incoming

Public Key - this is sent to the browser

The browser uses the public key to encrypt and decrypt the messages to/from the server, so that the information cannot be viewed by anyone intercepting the communication.

Web Security Basics - Authentication and Authorization

Two often confused but very important concepts to clearly understand are Authentication and Authorization.

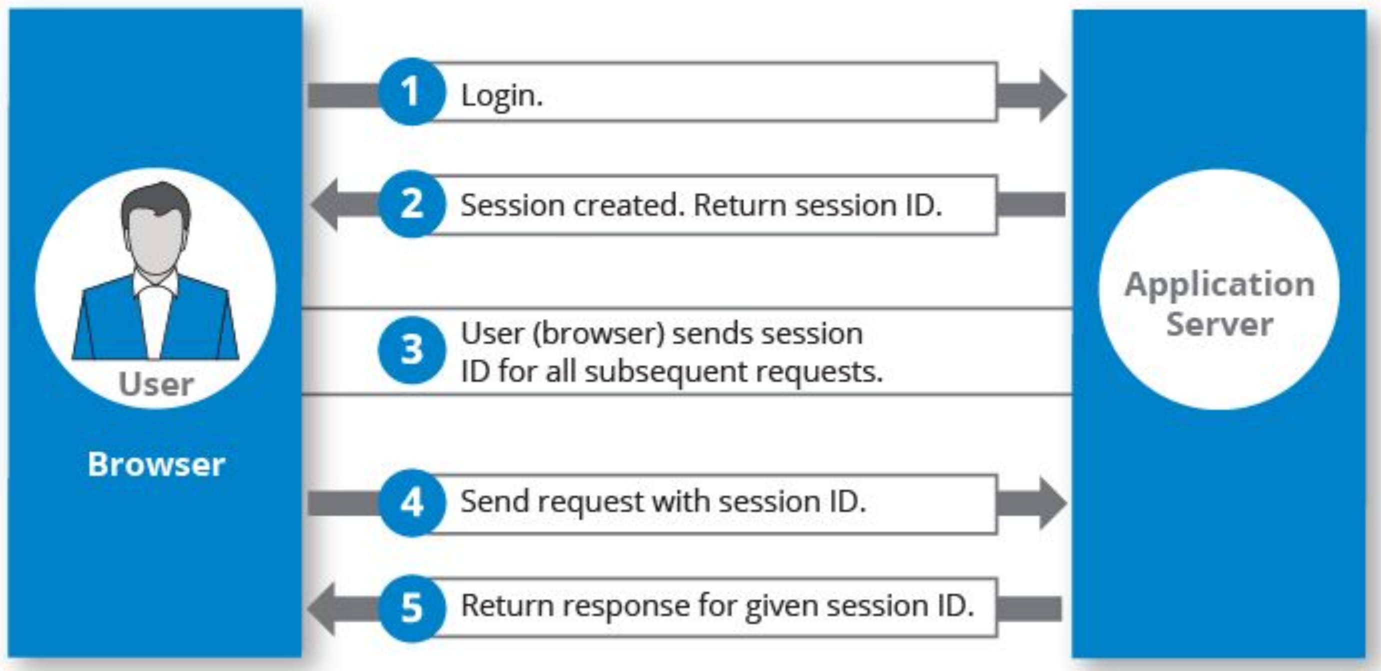
1. Authentication verifies that a user of a system is who they say they are.
2. Authorization verifies that a user of a system that has been authenticated can perform a given action such as delete a record or view a particular page.

Authentication - Managing identity on the web

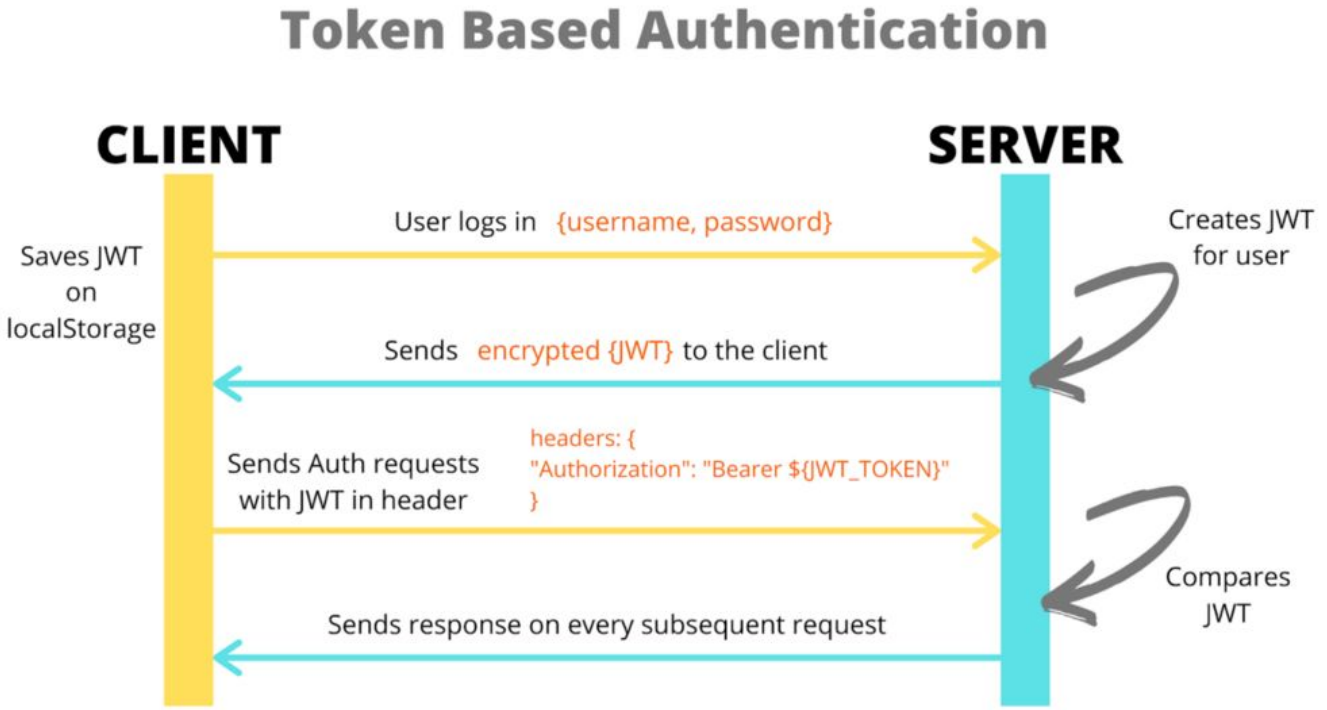
There are a few ways to manage authentication and this is not a complete list, this represents two of the major methods

a. Cookie-Based authentication
b. Token-Based authentication
c. Third party access(OAuth, API-token)
d. OpenId
e. SAML

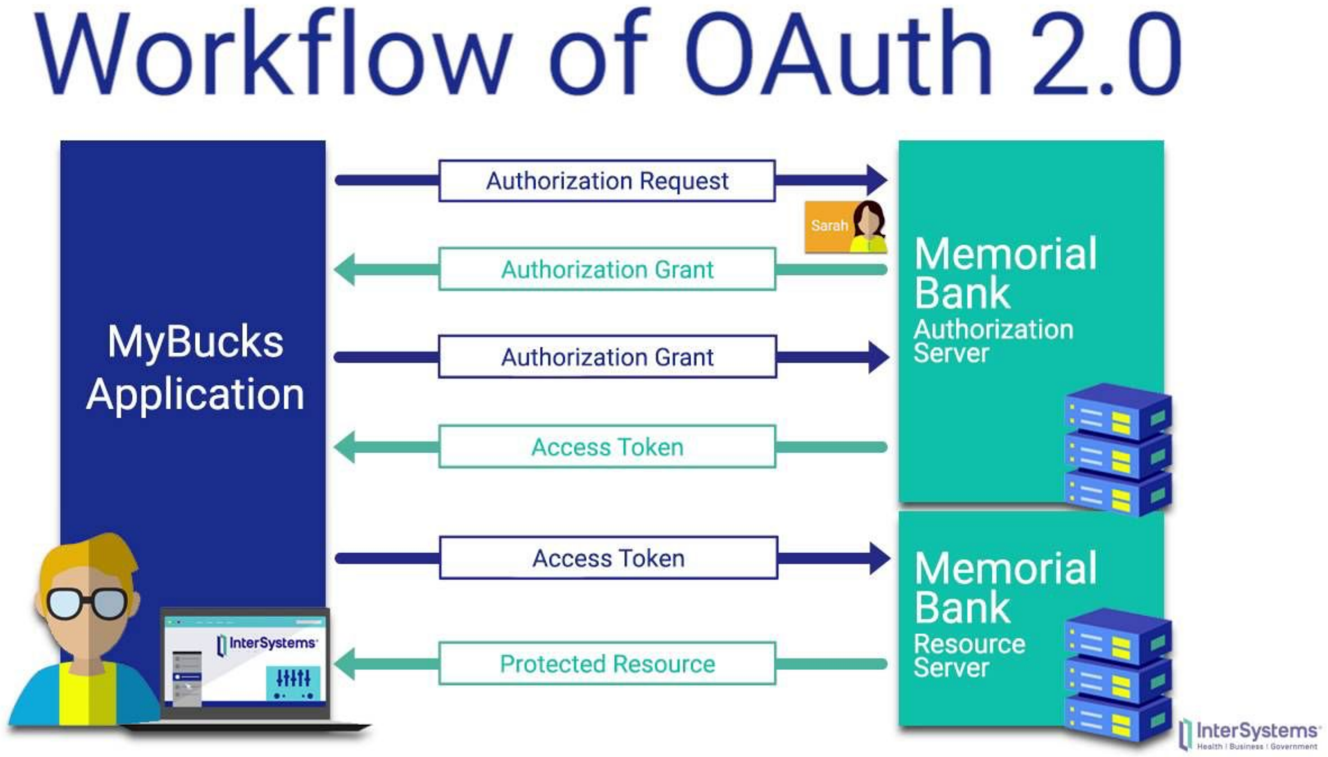
Session Based with Cookies


User
Browser
1 Login.
2 Session created. Return session ID.
3 User (browser) sends session ID for all subsequent requests.
4 Send request with session ID.
5 Return response for given session ID.
Application Server

JWT Tokens - Replacing cookie sessions and needed for mobile and clients that are not web browsers


Token Based Authentication
CLIENT
SERVER
User logs in {username, password}
Creates JWT for user
Saves JWT on localStorage
Sends encrypted {JWT} to the client
Sends Auth requests with JWT in header
headers: {
"Authorization": "Bearer ${JWT_TOKEN}"
}
Compares JWT
Sends response on every subsequent request

Oauth - WHen you sign in with Google



JWT vs Oauth vs OpenID

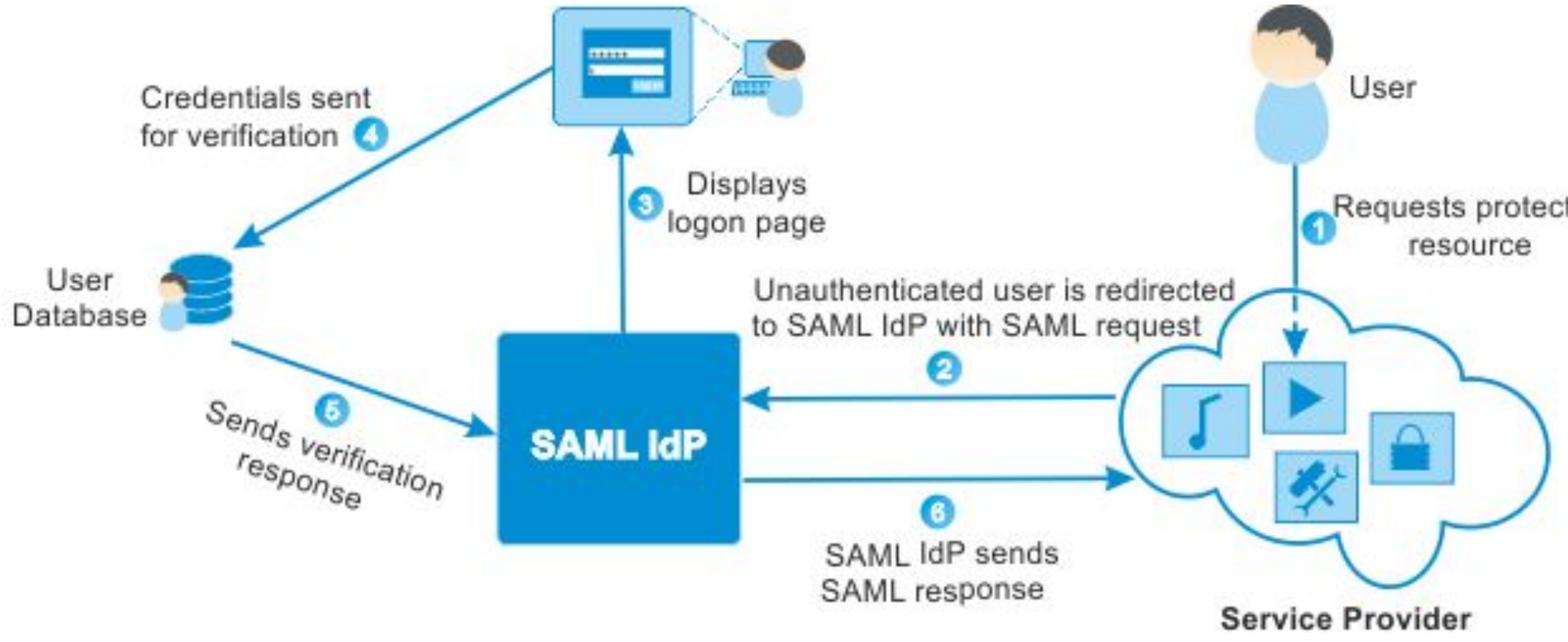
Basically, JWT is a token format.

OAuth is an standardised **authorization** protocol that can use JWT as a token. Oauth says basically that a user is allowed to do something.

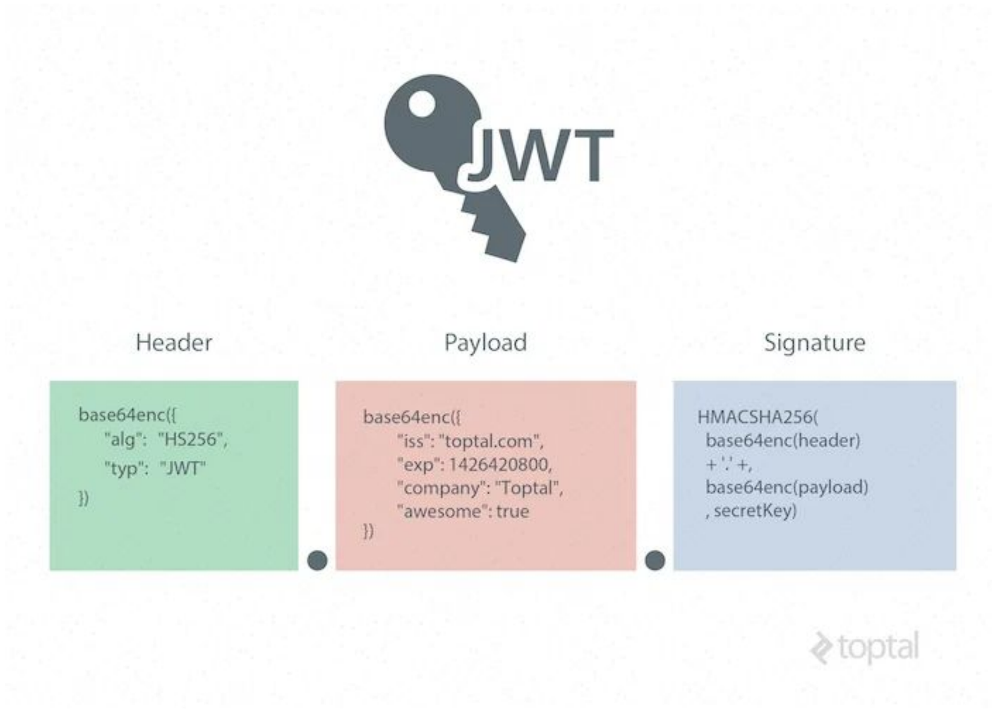
OpenID adds on top of OAUTH to provide a standard for authentication / identity verification.

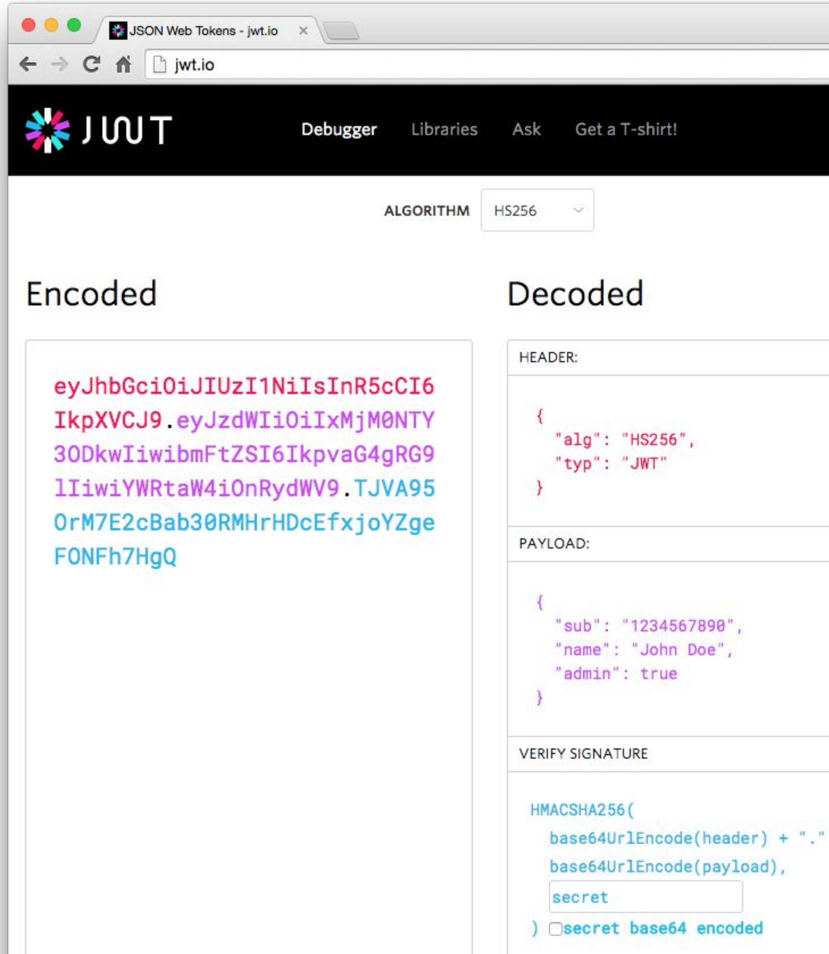
Both use JWT tokens as part of their process

SAML - You use this with Canvas


Credentials sent for verification 4
Displays logon page 3
User
Requests protected resource 1
User Database
Unauthenticated user is redirected to SAML IdP with SAML request
2
5
Sends verification response
SAML IdP
6
SAML IdP sends SAML response
Service Provider

JWT Tokens





ALGORITHM HS256

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6IkpvaG4gRG9
lIiwiYWRtaW4iOnRydWV9.TJVA95
OrM7E2cBab30RMHrHDcEfxjoYZge
FONFh7HgQ
```

Decoded

HEADER:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "."
  base64UrlEncode(payload),
  secret
) ☐secret base64 encoded
```

Summary of how this all fits together

HTTPS uses encryption to secure 2 way communication between the client and server so messages that are sent over the internet cannot be intercepted. Authentication uses hashing to obfuscate a users password so that it cannot be read by simply looking at the database record. Once authentication is complete. I.e. someone “logs in” to a system then their identity must be persisted between requests, which is where JWT or other methods of persisting identity take over. JWT and other authentication strategies would not be secure without HTTPS. JWT is important because it does not rely on the storage of the identity on the server and so it is more scalable. Session cookies are stored on the server and complicate scaling. With JWT when a token is sent it can be decoded to determine the requestor's identity by any application that has the same secret. This is needed in high scalability applications because they are often made of many “micro services” that are basically mini apps that perform a single function. Also JWT is needed for mobile applications / clients that are not web browsers, since session cookies rely on data specifically stored in a web browser.

CSRF Protection

Essentially the attack is based on the idea of somehow impersonating a user by accessing their SessionID cookie and then using that to make requests on the users behalf. CSRF tokens are used to secure applications by generating a unique token like a hash and then storing it in the users session on the server. The generated token is added to a form as a hidden field and that token is compared to the one stored on the server. Each time a form is generated a fresh unique token is generated and the users session is updated on the server.   This way even if someone gains access to the sessionID they will not be able to send form data to the server because any form they create will not have the session id. This works because of the same origin policy that prevents web clients from making requests to any other address than the one that the current page was loaded.

Security

1. Hashed password can be vulnerable if not using a good algo, so there has been an evolution in hashing. Currently Bcrypt is the standard.
2. There is a policy called the “same origin” policy - it requires that an http client i.e. browser can only make requests to the same address that sent the response / web page.

Theoretical Attack and Prevention

1. SOmeone gains access to your computer and is able to get your sessionID.
   a. THey can then impersonate, so that they can make requests to the webserver and the that web server will not know that the requests are not coming from you.
2. CSRF tokens prevent this to a degree because what happens is that every time a server sends a form to the client, the form will have a CSRF token embedded into the form. When the form is submitted the server looks at both the session ID and the CSRF token of that form to verify that not only is the form sent by the user but that this form is unmodified and being sent by the specifec browser that received the form data in the HTTP response.

Web security things to check

1. YOu have to use HTTPS to secure the network communication
2. You have to use CSRF tokens to provide an additional layer of security on top of forms because there are a variety of CSRF attack that can take advantage of insecure sessions
3. Database - Prevent SQL injection attack
    a. Basically its possible to send SQL database or types of database commands to a web server through insecure form fields and processing of data
    b. We use parameterized queries and never allow any data from a form to ever reach a database
    c. We also implement validation on fields to make sure they contain the correct data
    d. ORM SQLALchemy
4. Validation MUST be done on the server and it should be done on client because if you don’t validate on the client, your user will be upset when they submit a form and it is rejected for validation issues and then they need to fill the form out again.

Don’t store peoples info

Option 1:

Use Oauth to get a grant so you don’t have their password or other private info you don’t absolutely need for your application.

Option 2:

1. Use identity management service like Auth0 or OKTA