# Design Workflow

## Preflight

Upon landing on the website, the client performs a preflight request to the server. The server consequently evaluates the following:

* Did the client provide a web session ID (cookie = webSessionId)?
  + Yes: perform a query on the web\_session table to determine whether an existing session using the provided ID exists with a future-dated expiry. Did the query return results?
    - Yes: is the client from both a trusted origin and trusted IP address (are the request origin and request IP address equal to those stored in the metadata of the web session stored in the database)?
      * Yes: return success (HTTP code 200 OK) to the client.
      * No: return failure (HTTP code 401 unauthorized) to the client.
    - No: return failure (HTTP code 401 unauthorized) to the client – session expired.
  + No: is the client from both a trusted origin and trusted IP address (are the request origin <http://localhost:8000> and IP address ::1)?
    - Yes: create both a cookie (webSessionId) and a record in the web\_session table using a 32-byte hexadecimal-encoded string unique identifier (sessionId) generated using a PSRNG, with a lifetime of 12 hours. In the metadata field of the database record, store the client origin and client IP address as a JSON string encoded object. Return success (HTTP code 200 OK) to the client.
    - No: return failure (HTTP code 403 forbidden) to the client – client does not have permission.

## Sign Up

For user registration, the client evaluates the additional criteria before the request is made to the server:

* Do the passwords field contents match?
* Does the password meet the following criteria?
  + At least 8 characters in length
  + At least 1 uppercase letter
  + At least 1 lowercase letter
  + At least 1 digit
  + At least 1 special character

If the above criteria are met, the request is made to the server, where the following is evaluated:

* Do the email or username already exist?
  + Yes: return failure (HTTP code 409 conflict) to the client.
  + No: generate a 32-byte hexadecimal-encoded salt using a PSRNG. Concatenate the client password with the salt and hash the result. Create a record in the user table with the password field containing the hashed and salted password. Create a record in the user\_login table to store the salt. Return success (HTTP code 201 created) to the client.

## Authentication

The login workflow is broken down into a two-stage process: Identify and Login. This procedure implements the challenge-response authentication protocol.

### Identify

In the identification stage, the client disregards the plaintext password and sends only the username to the server. The server consequently evaluates the following:

* Query the user\_login table with the username. Is a salt returned?
  + Yes: generate a 64-byte hexadecimal-encoded challenge using a PSRNG. Store the challenge in the corresponding record with a lifetime of 10 seconds. Return the salt and the challenge to the client. Return success (HTTP code 201 created) to the client.
  + No: an account with that username does not exist. Return failure (HTTP code 404 not found) to the client.

### Login

In the login stage, the client will perform the following:

* Generate a 16-byte hexadecimal-encoded initialization vector using a PSRNG.
* Concatenate the password with the salt and hash the result.
* Encrypt (AES-256-CBC) the challenge using the hashed and salted password, as well as the initialization vector to produce a ciphertext.
* The payload of the request to the server will consist of the username, ciphertext, and initialization vector.

The server consequently performs the following:

* Retrieve the hashed and salted password as well as the challenge from the joined user x user\_login table using the username.
* Encrypt (AES-256-CBC) the challenge using the hashed and salted password, as well as the initialization vector supplied by the client to produce a ciphertext.
* Compare the client-generated ciphertext with the server-generated ciphertext. Are they equivalent? And was the comparison performed before the challenge expired?
  + Yes: is there an existing record in the user\_session table with the corresponding username?
    - Yes: update the existing record’s by extending the expiry by 15 minutes from the current datetime. Create a cookie using the existing user session ID. Return success (HTTP code 200 OK) to the client.
    - No: create both a cookie (userSessionId) and a record in the user\_session table using a 32-byte hexadecimal-encoded string unique identifier generated using a PSRNG, with a lifetime of 15 minutes. Return success (HTTP code 200 OK) to the client.
  + No: return failure (HTTP code 401 unauthorized) to the client denoting that the username or password was incorrect.

Once authenticated, the client will store the plaintext password (called master key) in the browser’s session storage.

## Save

To add a site, the client performs the following before sending the request to the server:

* Generate a 16-byte hexadecimal-encoded initialization vector using a PSRNG.
* Hash the master key.
* Encrypt (AES-256-CBC) the site password using the hashed master key, as well as the initialization vector to the encrypted site password.

Upon receiving the request, the server will perform the following:

* Is the user session valid?
  + Yes: does the username and user site combination exist in the table?
    - Yes: perform an update on the existing record in the user\_safe table.
    - No: insert a new record in the user\_safe table.
    - In addition to the site credentials, store the site initialization vector. Return success (HTTP code 200 OK) to the client.
  + No: delete the user session record from the user\_session table and expire the corresponding cookie. Return failure (HTTP code 401 unauthorized) to the client – session expired.

## Update

To update an existing site on the add new site page, select an existing site using the dropdown. Doing so will perform a request similar to that in the Save section of this document, including the site ID as an additional parameter.

## Sites

When the client makes a request to fetch the list of sites that the user has stored credentials for, the server performs the following:

* Is the user session valid?
  + Yes: Retrieve the list of site names and corresponding list of site IDs from the joined user\_session x user\_safe table using the user session ID. Return the aforementioned lists. Return success (HTTP code 200 OK) to the client.
  + No: delete the user session record from the user\_session table and expire the corresponding cookie. Return failure (HTTP code 401 unauthorized) to the client – session expired.

## Load

To load a site, the client will request the site name, site username, site password, and the site’s corresponding initialization vector by sending the site ID to the server. The server will then perform the following:

* Is the user session valid?
  + Yes: Retrieve the corresponding data from the user\_safe table using the user session ID. Return success (HTTP code 200 OK) to the client.
  + No: delete the user session record from the user\_session table and expire the corresponding cookie. Return failure (HTTP code 401 unauthorized) to the client – session expired.

Considering the site password was encrypted before it was stored in the user\_safe table, the client is responsible for decrypting the site password before the credentials are displayed on the webpage. The client therefore decrypts (AES-256-CBC) the site password using the hashed master key, as well as the initialization vector to the encrypted site password.

## Logout

The server will perform the following:

* Does cookie containing user session ID exist?
  + Yes: Delete any existing records in the user\_session table containing the user session ID. Return success (HTTP code 200 OK) to the client.
  + No: Return failure (HTTP code 401 unauthorized) to the client; do not return error message, simply notify that the client successfully logged out.

# Security Vulnerabilities

## SQL Injection

Using PHP prepared statements protect against SQL injection by providing a parameterized and reusable SQL querying mechanism that forces developers to write the desired SQL command and the user-provided data separately. Using bind variables, the database engine compiles the query using placeholders, and the user-supplied data is added later.

## Site Request Forgery (CSRF)

Addressed by preflight