SUPERVISOR: AHMED HANIF

NARWAL AUTH API DELIVERABLE 8

MUHAMMAD NASEEM
SAMI NAEEM
KYNAT MANSHA

Narwal Auth Library

Successfully integrated NarwalAuth Into the Registration process of our website:

```
const handleSignup = async () => {
```

```
try {
      const narwalAuth = await NarwalAuth();
      password = await narwalAuth.GetPublicKey(password, email);
      console.log("Signup:", name, email, password);
      axios
        .post("http://localhost:8081/api/signup", { name, email, password })
        .then((response) => {
          // Ensure the response has data
          if (response && response.data) {
            const { user } = response.data;
            // Store the user data in the local storage
            localStorage.setItem("user", JSON.stringify(user));
            console.log("Signup successful:", response.data);
          } else {
            // Handle cases where response does not have expected data
            console.error("Signup failed: No data in response");
          }
        })
        .catch((error) => {
          // Handle any errors from the request
          console.error("Signup failed:", error);
        });
    } catch (error) {
      console.error("An error occurred during signup:", error);
    }
```

Stored in the database as a Username and Public key. (logs will be shared in the next deliverable)

Narwal Auth Api

Setup and Dependencies

The API requires the following Node.js modules

- express
- body-parser
- crypto

  To set up the server:

```
const express = require("express");
const bodyParser = require("body-parser");
const crypto = require("crypto");

const app = express();
const port = 3001;

app.use(bodyParser.json());
```

API Endpoints

1. Generate Challenge

- **URL:** `/generate-challenge`
- **Method:** POST
- **Body:** `{ username: string }`
- **Response:** `{ challenge: number }`

Generates a random challenge for a given username.

```
app.post("/generate-challenge", (req, res) => {
  const { username } = req.body;
  if (!username) {
    return res.status(400).json({ error: "Username is required" });
  }

  // Generate a random challenge
  const challenge = crypto.randomInt(1, Number.MAX_SAFE_INTEGER);
  challenges.set(username, challenge); //why this ?

  res.json({ challenge });
});
```

2. Verify Authentication

- **URL:** `/verify`
- **Method:** POST
- **Body:** `{ username: string, publicKey: string, c: string, z: string }`
- **Response:** `{ success: boolean }`

Verifies the authentication data provided by the client.

```
app.post("/verify", async (req, res) => {
  const { username, publicKey, c, z } = req.body;

  try {
    const a = challenges.get(username);
    if (!a) {
      return res.status(400).json({ error: "Challenge not found" });
    }

    // Convert inputs to BigInt
    const Y = BigInt(publicKey);
    const cInt = BigInt(`0x${c}`);
    const zInt = BigInt(z);
    const g = BigInt(2); // Replace with actual generator value

    // Compute T' = Y^c * g^z
    const TPrime = Y ** cInt * g ** zInt;

    // Compute hash(Y || T' || a)
    const hashInput = `${Y.toString()}||${TPrime.toString()}||${a}`;
    const computedHash = await hash(hashInput);

    // Verify if hash(Y || T' || a) == c
    const isSuccess = computedHash === c;
    // Send the result back to the client
    res.json({ success: isSuccess });
  } catch (error) {
    console.error("Error verifying authentication:", error);
    res.status(500).json({ error: "Internal server error" });
  }
});
```

3. Receive Challenge Request

- **URL:** `/receive-challenge-request`
- **Method:** POST
- **Body:** `{ challengeRequest: { username: string } }`
- **Response:** `{ challenge: string }`

Handles challenge requests from the server.

```
app.post("/receive-challenge-request", (req, res) => {
  // Example implementation, replace with actual logic
  const { challengeRequest } = req.body;
  console.log("Received challenge request:", challengeRequest);

  // Generate a challenge for the requested username
  const challenge = crypto.randomInt(1, Number.MAX_SAFE_INTEGER).toString();
  challenges.set(challengeRequest.username, challenge);

  res.json({ challenge });
});
```

4. Receive Data from Server

- **URL:** `/receive-data-from-server`
- **Method:** POST
- **Body:** `{ username: string, publicKey: string, c: string, z: string }`
- **Response:** `{ success: boolean }`

Processes authentication data received from the server.

```javascript
app.post("/receive-data-from-server", async (req, res) => {
  const { username, publicKey, c, z } = req.body;

  try {
    const a = challenges.get(username);
    if (!a) {
      return res.status(400).json({ error: "Challenge not found" });
    }

    // Convert inputs to BigInt
    const Y = BigInt(publicKey);
    const cInt = BigInt(`0x${c}`);
    const zInt = BigInt(z);
    const g = BigInt(2); // Replace with actual generator value

    // Compute T' = Y^c * g^z
    const TPrime = Y ** cInt * g ** zInt;

    // Compute hash(Y || T' || a)
    const hashInput = `${Y.toString()}||${TPrime.toString()}||${a}`;
    const computedHash = await hash(hashInput);

    // Verify if hash(Y || T' || a) == c
    const isSuccess = computedHash === c;

    // Send the result back to the server
    sendResultToServer(isSuccess); // Assuming you have a function to send the
result

    res.json({ success: isSuccess });
  } catch (error) {
    console.error("Error verifying authentication:", error);
    res.status(500).json({ error: "Internal server error" });
  }
});
```

Key Functions

- **hash(data):** Computes a SHA-256 hash of the input data.
- **sendResultToServer(success)**: Sends the authentication result back to the server

```
const axios = require('axios');

// Function to send result to the server
async function sendResultToServer(success) {
  try {
    const serverUrl = 'http://main-server-url.com/authentication-result';
// Replace with actual server URL
    const response = await axios.post(serverUrl, {
      success: success,
      timestamp: new Date().toISOString()
    });

    if (response.status === 200) {
      console.log('Result sent successfully to server');
    } else {
      console.error('Failed to send result to server. Status:',
response.status);
    }
  } catch (error) {
    console.error('Error sending result to server:', error.message);
  }
}
```

Data Structures

- **Challenges:** A ‘`Map`’ object storing challenges indexed by username.

Running the Server

To start the server:

Javascript Copy

```javascript
app.listen(port, () => {
  console.log(`API running at http://localhost:${port}`);
});
```

The server will run on `http://localhost:3001`.

Security Considerations

- The API uses cryptographic operations for secure authentication.
- Challenges are randomly generated for each authentication attempt.
- The verification process uses BigInt for large number operations.
- SHA-256 is used for hashing operations.