Welcome to Authentication and Authorization in Node.js

Estimated time needed: **20 minutes**

Objectives

In this reading, you will be able to:

- Define authentication
- Explain session-based, token-based, and passwordless authentication
- Compare and contrast different types of authentications, including session-based, token-based, and passwordless

Authentication

The authentication process confirms a user's identity using credentials by validating who they claim to be. Authentication assures an application's security by guaranteeing that only those with valid credentials can access the system. Authentication is the responsibility of an application's backend.

Three popular authentication methods in Node.js include:

1. Session-based
2. Token-based
3. Passwordless

Let's explain a little bit about each of these methods and compare them.

Session-based

Session-based authentication is the oldest form of authentication technology. Typically, the flow of a session is as follows.

1. The user uses their credentials to log in.
2. The login credentials are verified against the credentials in a database. The database is responsible for storing which resources can be accessed based on the session ID.
3. The server creates a session with a session ID that is a unique encrypted string. The session ID is stored in the database.
4. The session ID is also stored in the browser as a cookie.
5. When the user logs out or a specified amount of time has passed, the session ID is destroyed on both the browser and the database.

Below is a code snippet demonstrating session-based authentication in an Express application:

```
const express = require('express');
const session = require('express-session');
const app = express();
// Middleware to set up session management
app.use(session({
  secret: 'secret-key',      // Replace with a strong secret key
  resave: false,             // Whether to save the session data if there were no modifications
  saveUninitialized: true,   // Whether to save new but not modified sessions
  cookie: { secure: false }  // Set to true in production with HTTPS
}));
// POST endpoint for handling login
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  // Simulated user authentication (replace with actual logic)
  if (username === 'user' && password === 'password') {
    req.session.user = username;  // Store user information in session
    res.send('Logged in successfully');
  } else {
    res.send('Invalid credentials');
  }
});
// GET endpoint for accessing dashboard
app.get('/dashboard', (req, res) => {
  if (req.session.user) {
    res.send(`Welcome ${req.session.user}`);  // Display welcome message with user's name
  } else {
    res.send('Please log in first');
  }
});
// Start the server on port 3000
app.listen(3000, () => console.log('Server running on port 3000'));
```

Explanation:

- **Express Setup**: This code sets up an Express application and configures session management using express-session.
- **Session Configuration**: Express-session middleware is configured with a secret key (secret: 'secret-key') for encrypting the session data, and other options like resave and saveUninitialized.
- **Login Endpoint (/login)**: Handles POST requests for user login. If the provided username and password match, it stores the username (req.session.user) in the session.
- **Dashboard Endpoint (/dashboard)**: Checks if the user is authenticated (req.session.user exists). If authenticated, it welcomes the user; otherwise, it prompts them to log in.

Token-based

Token-based security entails two parts: authentication and authorization. Authentication is the process of providing credentials and obtaining a token that proves the user's credentials. Authorization refers to the process of using that token so the resource server knows which resources the user should have access to.

Token-based authentication

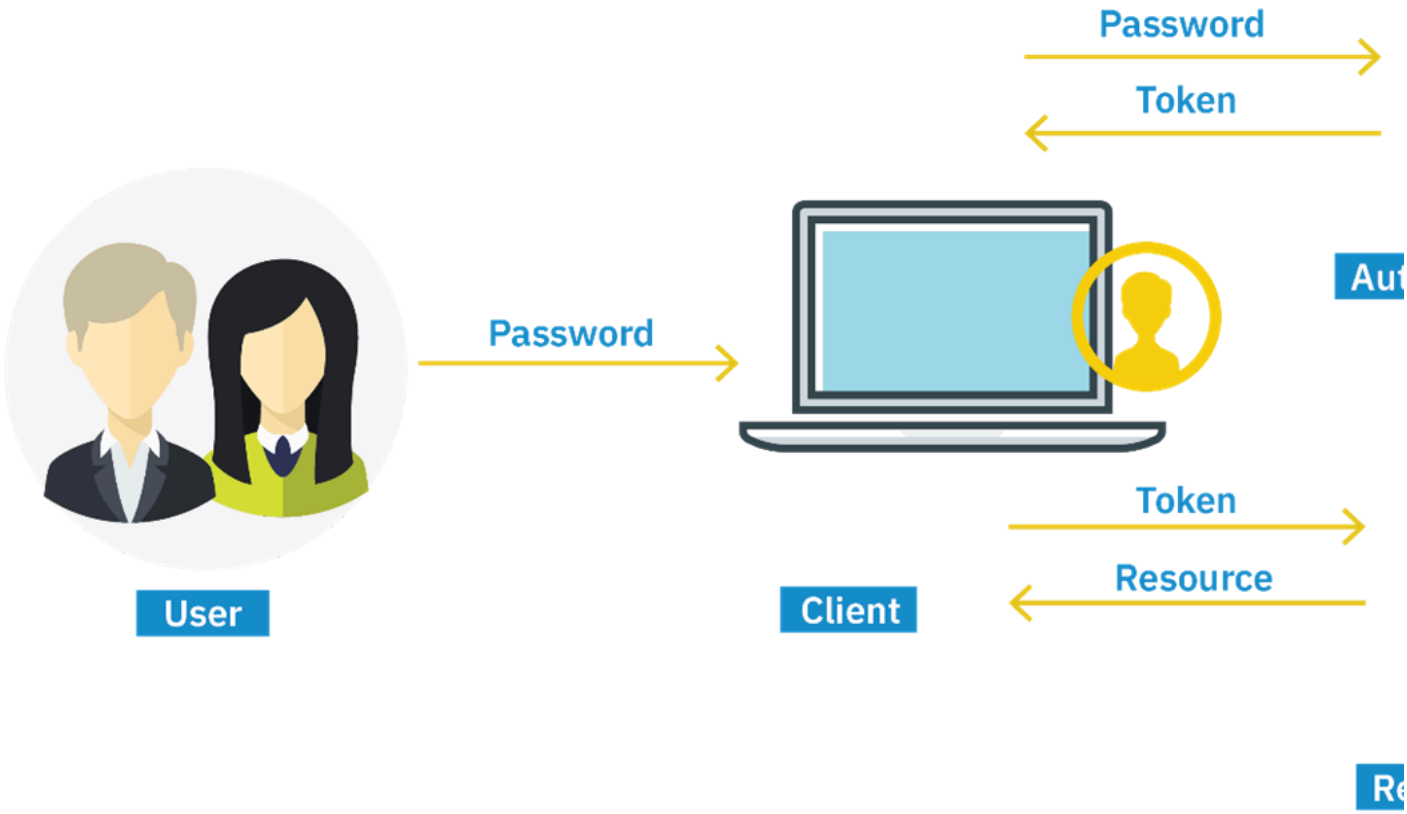
Token-based authentication uses access tokens to validate users. An access token is a small piece of code that contains information about the user, their permissions, groups, and expirations that get passed from a server to the client. An ID token is an artifact that proves that the user has been authenticated.

The token contains three parts: the header, the payload, and the signature. The header contains information about the type of token and the algorithm used to create it. The payload contains user attributes, called claims, such as permissions, groups, and expirations. The signature verifies the token's integrity, meaning that the token hasn't changed during transit. A JSON web token, pronounced "jot" but spelled JWT, is an internet standard for creating encrypted payload data in JSON format.

A user's browser makes a call to an authentication server and gets access to a web application. The authentication server then passes back an ID token which is stored by the client as an encrypted cookie. The ID token is then passed to the app on the web server as proof that the user has been authenticated.

Token-based authorization

This flowchart shows the workflow of a token through the authorization process.



The authorization process gets executed when the web application wants to access a resource, for example, an API that is protected from unauthorized access. The user authenticates against the Authorization server. The Authorization server creates an access token (note that the ID token and access token are two separate objects) and sends the access token back to the client, where the access token is stored. Then, when the user makes requests or resources, the token is passed to the resource, also called an API server. The token gets passed with every HTTP request. The token contains embedded information about the user's permissions without the need to access those permissions from the authorization server. Even if the token is stolen, the hacker doesn't have access to the user's credentials because the token is encrypted.

Below is a code snippet demonstrating Token-based Authentication in an Express application:

```
const express = require('express');
const jwt = require('jsonwebtoken');
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.json());
const secretKey = 'your-secret-key'; // Replace with a strong secret key
// POST endpoint for user login and JWT generation
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  // Simulated user authentication
  if (username === 'user' && password === 'password') {
    // Generate JWT with username payload
    const token = jwt.sign({ username }, secretKey, { expiresIn: '1h' });
    res.json({ token }); // Send token as JSON response
  } else {
    res.send('Invalid credentials');
  }
});
// GET endpoint to access protected resource (dashboard)
app.get('/dashboard', (req, res) => {
  // Get token from Authorization header
  const token = req.headers['authorization'];
  if (token) {
    // Verify JWT token
    jwt.verify(token, secretKey, (err, decoded) => {
      if (err) {
        res.send('Invalid token');
      } else {
        // Token is valid, send welcome message with username
        res.send(`Welcome ${decoded.username}`);
      }
    });
```

```
  } else {
    res.send('Token missing');
  }
});
// Start server
app.listen(3000, () => console.log('Server running on port 3000'));
```

Explanation:

- **Express Setup**: Configures an Express application with middleware like body-parser to parse JSON requests.
- **JWT Generation (/login)**: Handles POST requests for user login. If credentials are valid, it generates a JWT (token) containing the username (jwt.sign({ username }, secretKey, { expiresIn: '1h' })).
- **JWT Verification (/dashboard)**: Checks for a JWT in the Authorization header of incoming requests (const token = req.headers['authorization']). If present, it verifies the token (jwt.verify(token, secretKey)) and extracts the username (decoded.username) to grant access.

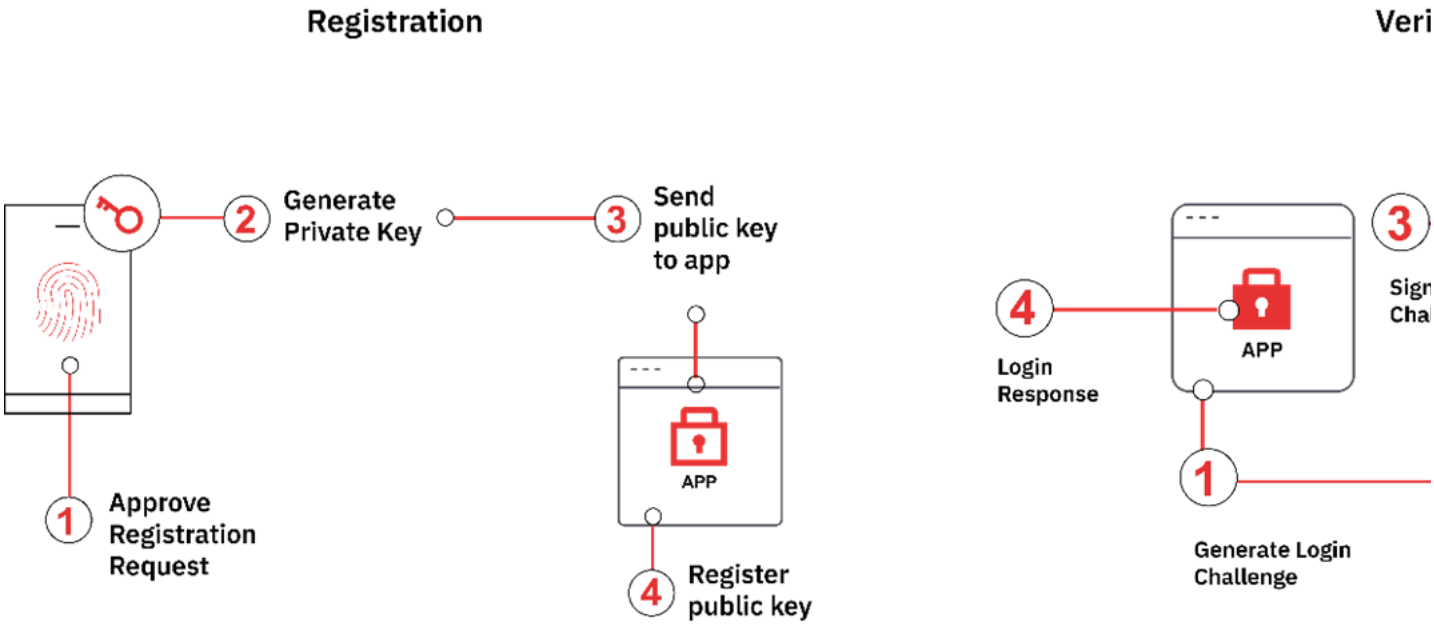
Passwordless

With passwordless authentication, the user does not need login credentials, but rather, they gain access to the system by demonstrating they possess a factor that proves their identity. Common factors include biometrics such as a fingerprint, a "magic link" sent to their email address, or a one-time passcode sent to a mobile device. Password recovery systems now commonly use passwordless authentication.

Passwordless authentication is achieved using Public Key and Private Key Encryption. In this method, when a user registers for the app, the user's device generates a private key/public key pair that utilizes a factor that proves their identity, as noted above.

The public key is used to encrypt messages, and the private key is used to decrypt them. The private key is stored on the user's device, and the public key is stored with the application and registered with a registration service.

Anyone may access the public key, but the private key is only known to the client. When the user signs into the application, the application generates a login challenge, such as requesting biometrics, sending a "magic link," or sending a special code via SMS, encrypting it with the public key. The private key allows the message to be decrypted. The app then verifies the sign-in challenge and accepts the response to authorize the user.



Below is a code snippet demonstrating Passwordless-based Authentication in an Express application:

```
const express = require('express');
const bodyParser = require('body-parser');
const nodemailer = require('nodemailer');
const app = express();
app.use(bodyParser.json());
const users = {}; // In-memory storage for demo purposes
// Endpoint to request access and send verification code via email
app.post('/request-access', (req, res) => {
  const { email } = req.body;
  // Generate a 6-digit verification code
  const code = Math.floor(100000 + Math.random() * 900000).toString();

  // Store the code in memory (users object)
  users[email] = code;
  // Simulated email sending (for demonstration)
  console.log(`Sending code ${code} to ${email}`);
  res.send('Code sent to your email');
});
// Endpoint to verify the received code
app.post('/verify-code', (req, res) => {
  const { email, code } = req.body;
  // Compare the received code with stored code for the email
  if (users[email] === code) {
    // Code matches, access granted
    res.send('Access granted');
  } else {
    // Code does not match, access denied
    res.send('Invalid code');
  }
});
// Start the Express server
app.listen(3000, () => console.log('Server running on port 3000'));
```

Explanation:

- **Express Setup**: Sets up an Express application with middleware to parse JSON requests (body-parser).
- **Request Access (/request-access)**: Handles POST requests where users provide their email to request access. Generates a 6-digit verification code (code) and stores it in an in-memory object (users[email] = code).
- **Verify Code (/verify-code)**: Handles POST requests to verify the received code against the stored code (if (users[email] === code)). If matched, it grants access; otherwise, it denies access.

Summary

In this reading, you learned that:

- Authentication is the process of confirming a user's identity using credentials by validating who they claim to be.
- Session-based authentication uses credentials to create a session ID stored in a database and the client's browser. When the user logs out, the session ID is destroyed.
- Token-based authentication uses access tokens, often JWTs, that get passed between server and client with the data that is passed between the two.
- Passwordless authentication uses public/private key pairs to encrypt and decrypt data passed between client and server without the need for a password.

Author(s)

Rajashree Patil
Sapthashree K S