Authentication in NodeJS

Authentication

- The process of verifying the identity of a user
- Two methods for authentication in NodeJS:
  - Stateful (session-based)
  - Stateless (token-based)

Stateful Authentication

- Server creates a session for the user after successful login and stores session information on the server-side
- The server then sends a cookie containing the session ID to the client-side
  - Then it is stored and sent back with each subsequent request to the server

Stateful Authentication

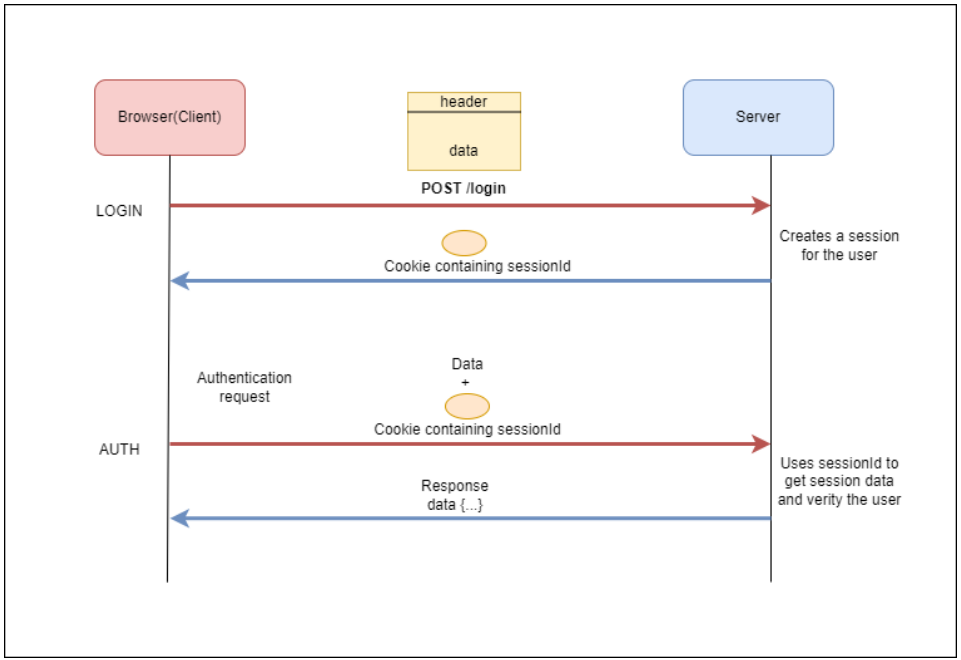
- Need some additional configurations for cookie be sent
  - From backend:

```
const corsOptions = {
  origin: true,
  credentials: true,
};
```

  - Fetch does not send cookies for cross-origin by default, need to add the fetch config `credentials: "include"`

```
const response = await fetch("http://localhost:8080/api",
{
   method: "post",
   credentials: "include",
});
```

Stateful Authentication



Stateful Authentication Example

- Set up session-based authentication in Node.js using the `express-session` middleware

```
const express = require("express");
const session = require("express-session");
const app = express();
// Middleware setup
app.use(
  session({
    secret: "your_secret_key",
    resave: false,
    saveUninitialized: false,
    cookie: {
      httpOnly: true,
      maxAge: 60 * 30, // In secs, Optional
    },
  })
);
```

Stateful Authentication Example

- Login Route

```
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  const user = users.find(u => u.username === username && u.password ===
  password);
  if (user) {
      req.session.userId = user.id; // Store user ID in session
      res.send('Login successful');
  } else {
      res.status(401).send('Invalid credentials');
  }
});
```

Stateful Authentication Example

- Protected Route

```
app.get('/home', (req, res) => {
   if (req.session.userId) {
      // User is authenticated
      res.send(`Welcome to the Home page, User ${req.session.userId}!`);
   } else {
      // User is not authenticated
      res.status(401).send('Unauthorized');
   }
});
```

Stateful Authentication Example

- Logout Route

```
app.get('/logout', (req, res) => {
    req.session.destroy(err => {
      if (err) {
        res.status(500).send('Error logging out');
      } else {
        res.redirect('/'); // Redirect to the home page after logout
      }
    });
});
```

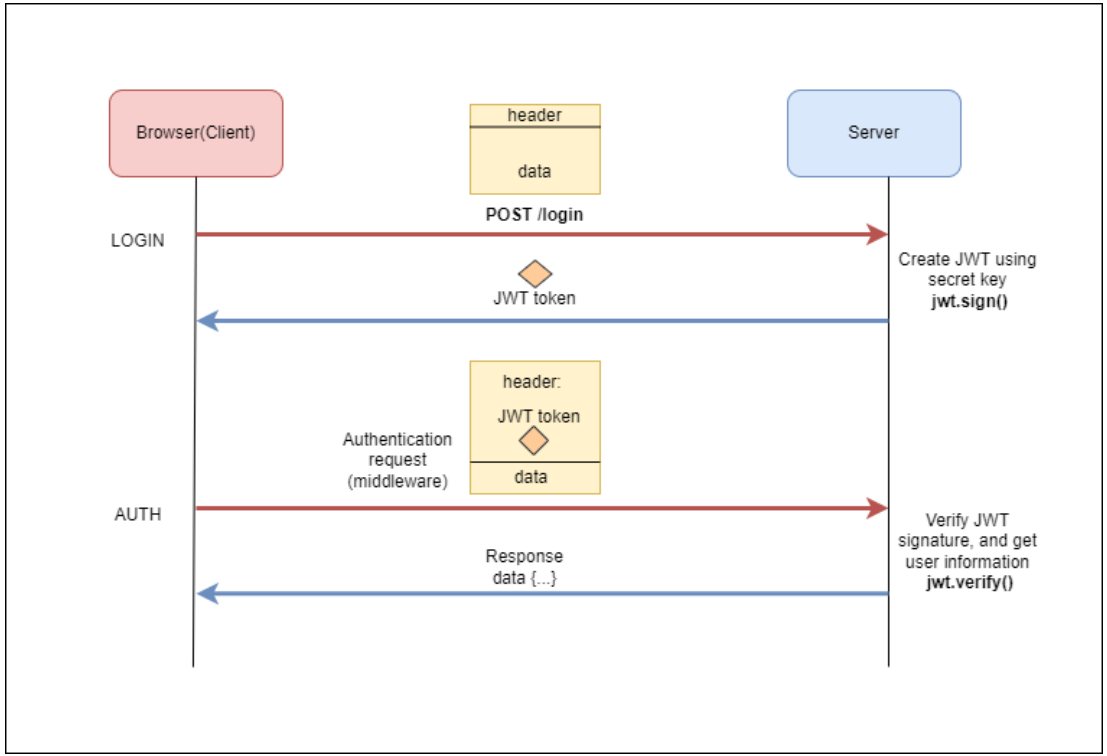
Stateful Authentication

- Advantages
  - Better performance due to stored information
  - Extra security layer
  - Intuitive due to ‘memory’
- Disadvantages
  - Requires memory for data storage
  - Server bears a significant burden
  - Performance depends on network memory efficiency

Stateless Authentication

- The server doesn’t store any session data
  - It uses tokens (like JWT – Json Web Token) which are sent to the client after successful login
- The client then sends this token in the Authorization header for each subsequent request
  - `const headers = { 'Authorization': 'Bearer my-token' };`
  - `fetch('https://www.example.com/api', { headers })`

Stateless Authentication



Stateless Authentication Example

- Login Route

```
app.post("/login", (req, res) => {
  const { username, password } = req.body;
  const user = users.find((u) => u.username === username && u.password
  === password);
  jwt.sign({ user }, secretKey, { expiresIn: "1h" }, (err, token) => {
     if (err) {
        res.status(500).send("Error generating token");
     } else {
        res.json({ token });
     }
  });
});
```

Stateless Authentication Example

- Protected Route

```
app.get('/dashboard', verifyToken, (req, res) => {
    res.send('Welcome to the Home page');
});
// Verify token middleware
function verifyToken(req, res, next) {
    const token = req.headers['authorization'];
    if (typeof token !== 'undefined') {
      jwt.verify(token.split(' ')[1], secretKey, (err, decoded) => {
        if (err) {
          res.status(403).send('Invalid token');
        } else {
          req.user = decoded.user;
          next();
        }
      });
    } else {res.status(401).send('Unauthorized');}
  }
```

Stateless Authentication

- Advantages
  - Easier to scale
  - Offers fault tolerance
  - Components are easily reusable
- Disadvantages
  - No built-in concept of past or future requests
  - May require more processing power and bandwidth
  - Implementing stateless authentication can be complex