Authentication & Authorization

Today's topics

Authentication vs Authorization

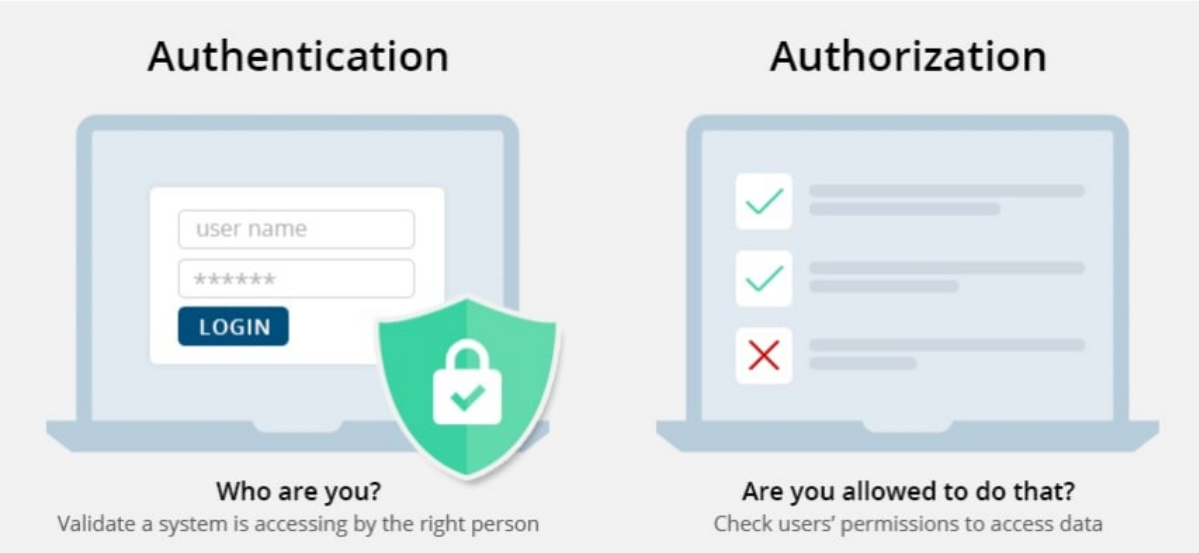
Express Middleware

Tutorial

- Logout/login
- isLoggedIn() middleware
- Vue router – navigation guard
- Roles & permissions - example

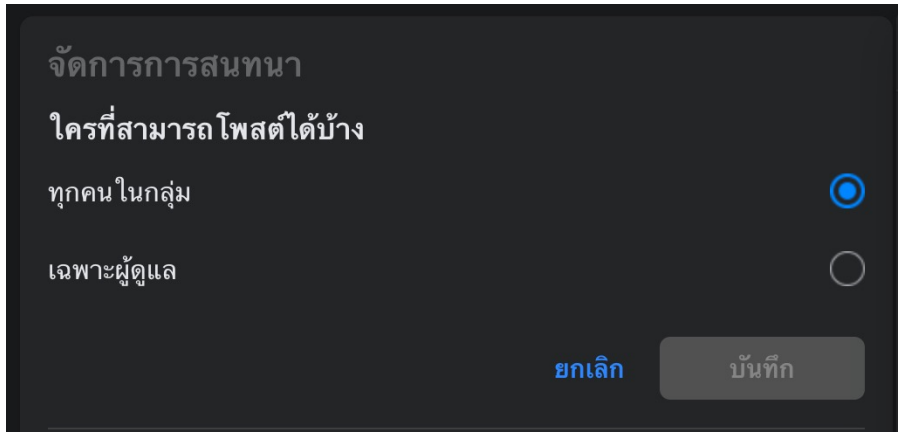
Authentication vs Authorization

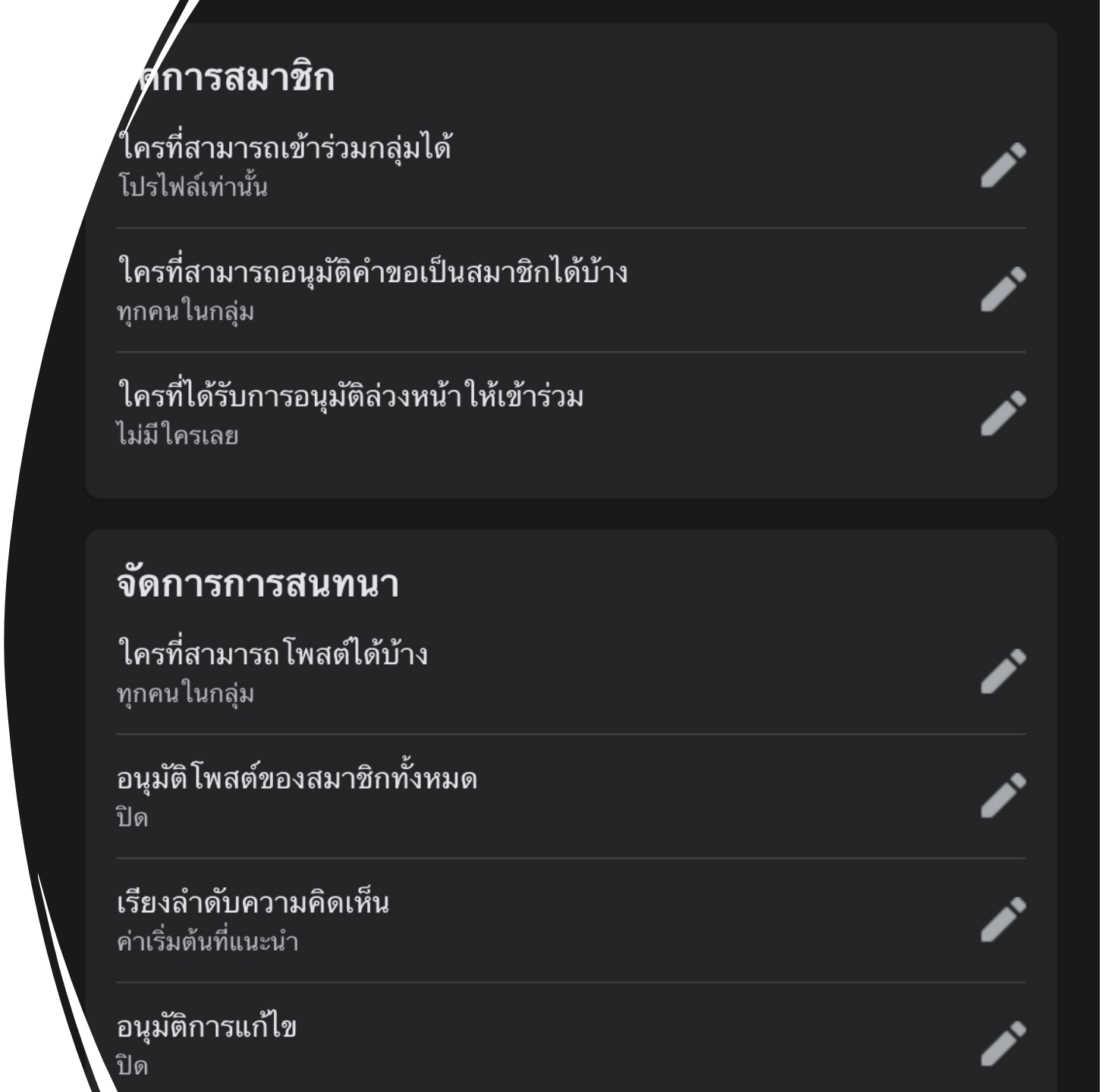
- **Authentication** confirms that users are who they say they are.
- **Authorization** Verifying user's permissions to access resources.



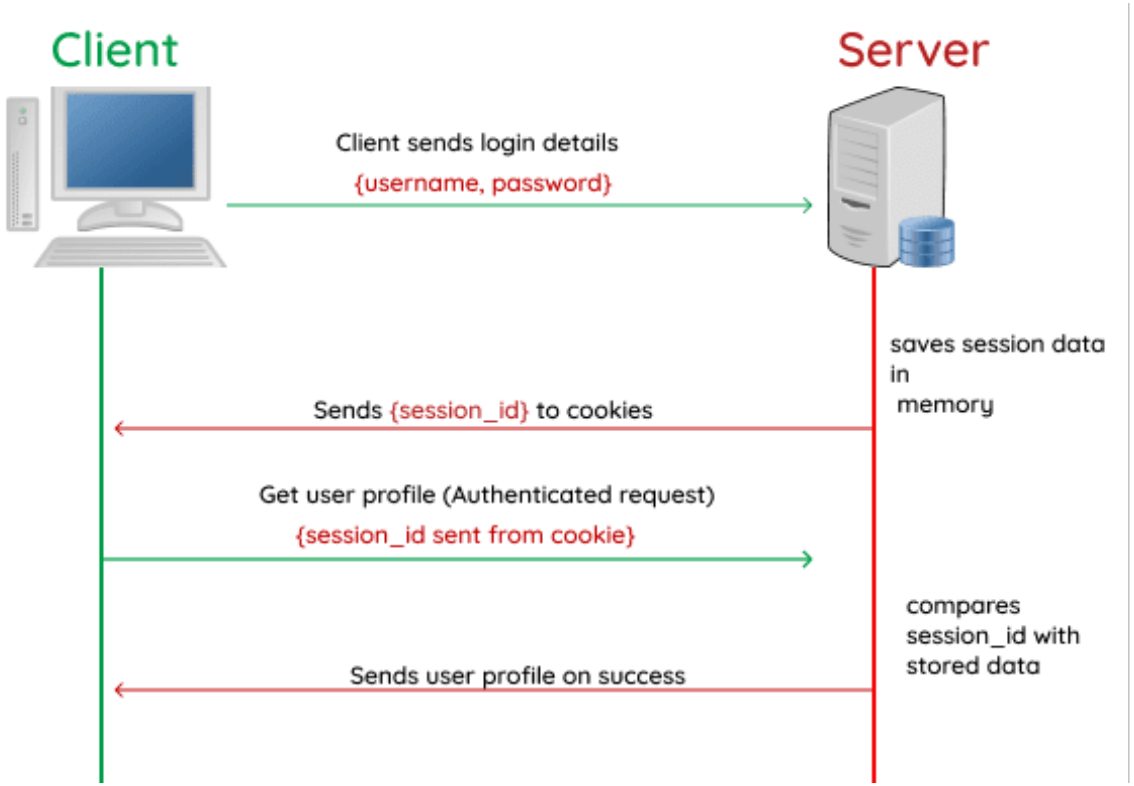
https://www.ssl2buy.com/wiki/authentication-vs-authorization-whats-the-difference

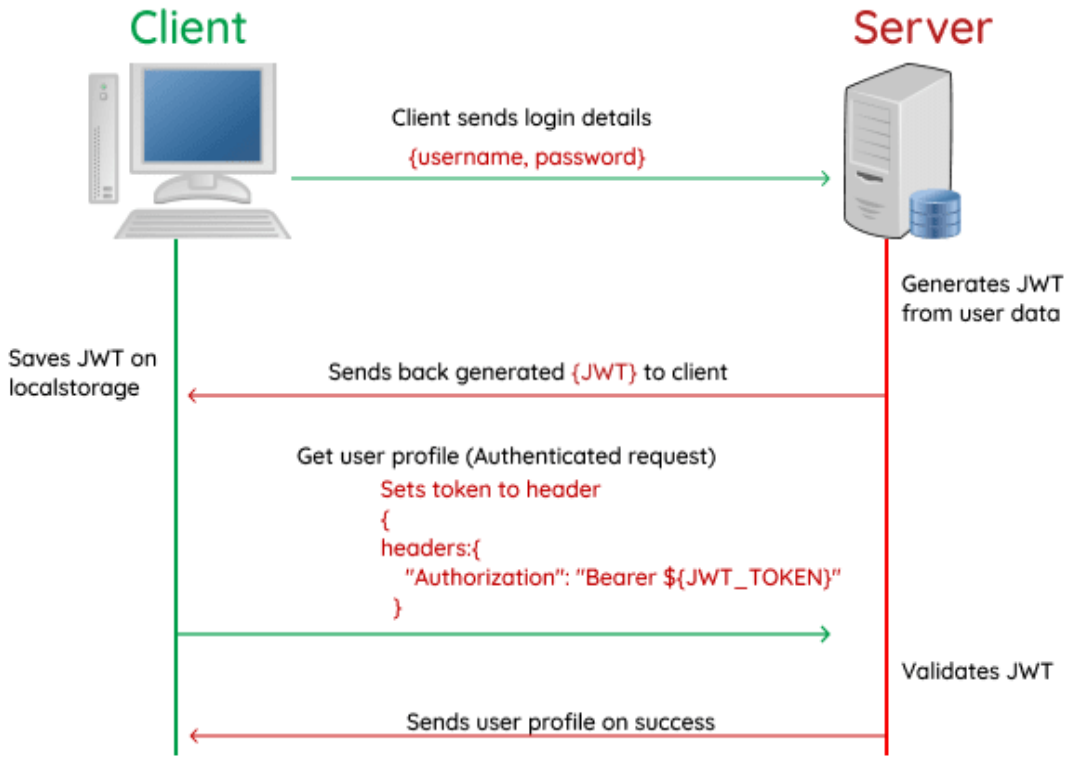
Facebook Group Authorization





Session Based vs Token Based Authentication





Session Based vs Token Based Authentication

Session Based

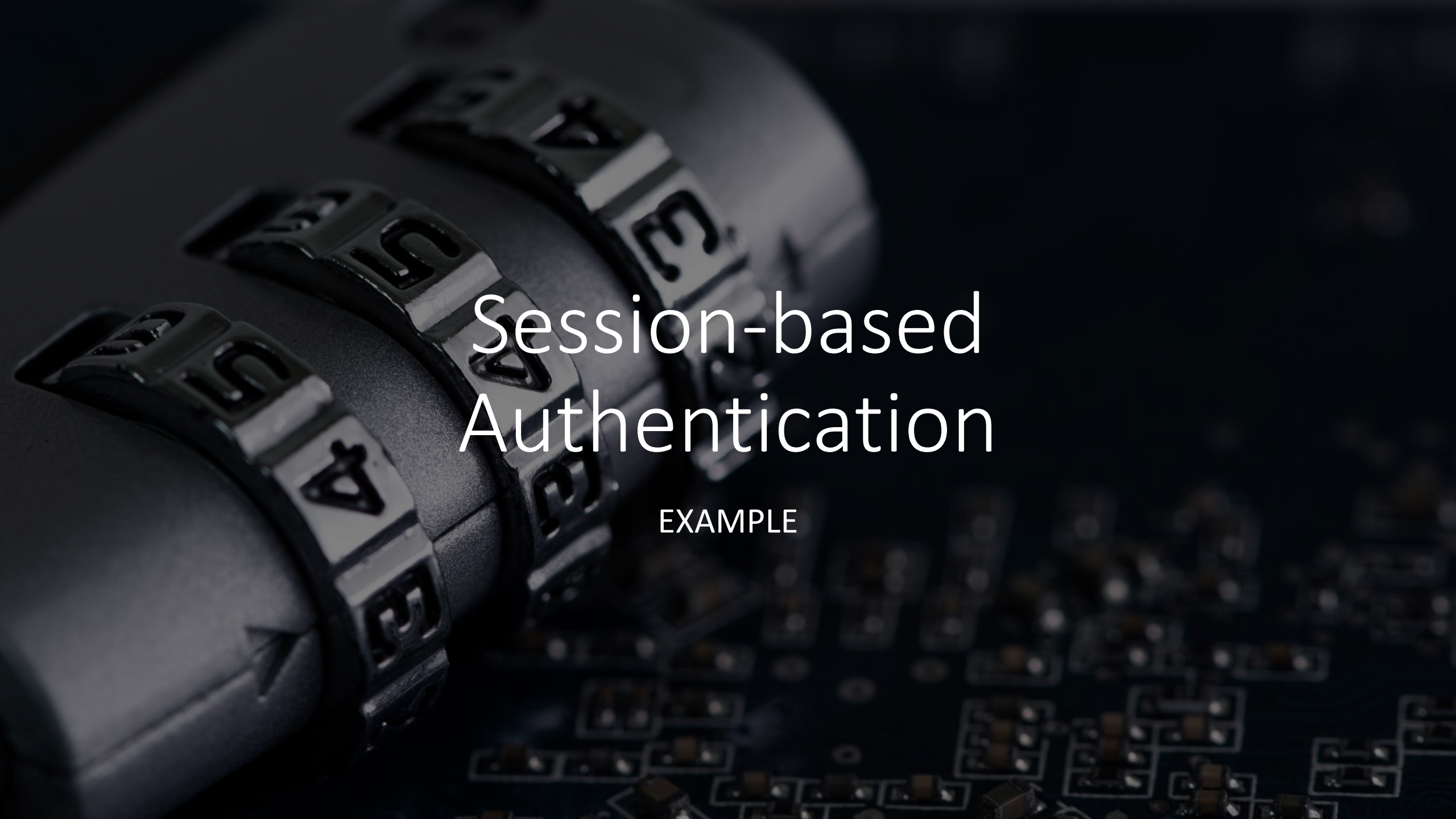
- User state is store in server (memory/database)
- Use Cookie to store session-id

Token Based

- User state is stored on client (localStorage)
- Attach token inside request header

NPM **express-session**
https://www.npmjs.com/package/express-session

NPM **express-jwt**
https://www.npmjs.com/package/express-jwt



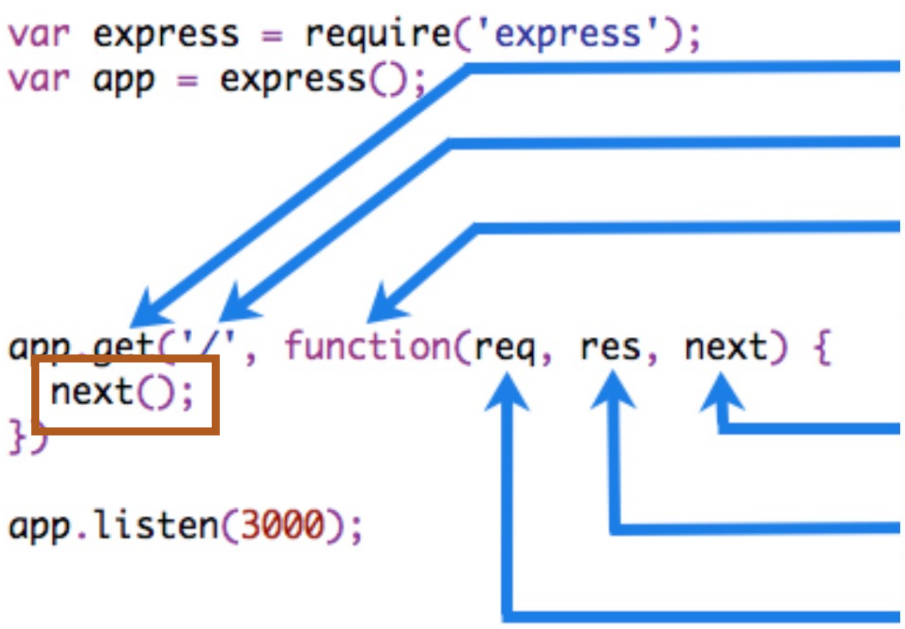
Session-based Authentication

EXAMPLE

Express Middleware

Middleware in Express JS

- **Middleware** functions are functions that have access to the request object (req), the response object (res), and the ***next*** function in the application's request-response cycle.
- The ***next*** function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware.
- **Middleware** functions can perform the following tasks:
  - Execute any code.
  - Make changes to the request and the response objects.
  - End the request-response cycle.
  - Call the next middleware in the stack.



HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

HTTP response argument to the middleware function, called "res" by convention.

HTTP request argument to the middleware function, called "req" by convention.

Let’s see some examples

Express Middleware

Types of middleware

- An Express application can use the following types of middleware:
  - Application-level middleware
  - Router-level middleware
  - Error-handling middleware
  - Built-in middleware
  - Third-party middleware

Application-level middleware

- Bind application-level middleware to an instance of the app object by using the **app.use()** and **app.METHOD()** functions, where METHOD is the HTTP method of the request that the middleware function handles (such as GET, PUT, or POST) in lowercase.

```
const express = require('express')
const app = express()

app.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
})
```

```
app.get('/user/:id', (req, res, next) => {
  res.send('USER')
})
```

Router-level middleware

- Router-level middleware works in the same way as application-level middleware, except it is bound to an instance of express.Router().

- Let’s see some code

Error-handling middleware

Error-handling middleware always takes ***four*** arguments. You must provide four arguments to identify it as an error-handling middleware function. Even if you don't need to use the next object, you must specify it to maintain the signature. Otherwise, the next object will be interpreted as regular middleware and will fail to handle errors.

```
app.use((err, req, res, next) => {
  console.error(err.stack)
  res.status(500).send('Something broke!')
})
```

Error Handling

- ***Error Handling*** refers to how Express catches and processes errors that occur both synchronously and asynchronously. Express comes with a default error handler so you don't need to write your own to get started.
  - If synchronous code throws an error, then Express will catch and process it. For example:

```
app.get('/', (req, res) => {
  throw new Error('BROKEN') // Express will catch this on its own.
})
```

- For errors returned from asynchronous functions invoked by route handlers and middleware, you must pass them to the next() function, where Express will catch and process them. For example:

```
app.get('/', (req, res, next) => {
  fs.readFile('/file-does-not-exist', (err, data) => {
    if (err) {
      next(err) // Pass errors to Express.
    } else {
      res.send(data)
    }
  })
})
```

Built-in middleware

- Express has the following built-in middleware functions:
  - express.static serves static assets such as HTML files, images, and so on.
  - express.json parses incoming requests with JSON payloads. **NOTE: Available with Express 4.16.0+**
  - express.urlencoded parses incoming requests with URL-encoded payloads. **NOTE: Available with Express 4.16.0+**

```
app.use(express.static(path.join(__dirname, 'static')))
app.use(express.json()) // for parsing application/json
app.use(express.urlencoded({ extended: true })) // for parsing application/x-www-form-urlencoded
```

Third-party middleware

- Use third-party middleware to add functionality to Express apps.
- Install the Node.js module for the required functionality, then load it in your app at the application level or at the router level.

```
const express = require('express')
const app = express()
const cookieParser = require('cookie-parser')

// load the cookie-parsing middleware
app.use(cookieParser())
```


Simple Token Authentication
Let’s re-invent the wheel

Example (Simple Token Authentication)

Tutorial: https://github.com/it-web-pro/WEEK12-TUTORIAL-EXERCISE