

**Instructor Notes:**

Add instructor notes here.



**Instructor Notes:**

Add instructor notes here.

## Lesson Objectives



Introduction  
Session Handling in Node.js  
Storing session information in MongoDB  
Cryptography  
Introduction  
Hashing data using bcryptjs



**Instructor Notes:**

Add instructor notes here.

**4.1: Session Handling**  
**Introduction**

- HTTP is a stateless protocol, server cannot differentiate between different connections of different users.
- One of the most common requirements for any web application are sessions.
- Using sessions to keep track of users as they browse through the site, becomes an important aspect for web application.
- Typically, the process of managing the state of a web-based client is through the use of session IDs. Session IDs are used by the application to uniquely identify a client browser, while background (server-side) processes are used to associate the session ID with a level of access.
- There are three methods available to allocate and receive session ID information.
  - URL Based Session ID's
  - Hidden Post Fields
  - Cookies

July 31, 2018

Proprietary and Confidential

- 3 -

Session cookies are temporary cookie files, which are erased when browser is closed. When browser restarted a new session cookie will be generated, which will store browsing information and will be active until the browser is closed.

Each time a client web browser accesses content from a particular domain or URL, if a cookie exists, the client browser is expected to submit any relevant cookie information as part of the HTTP request. Thus cookies can be used to preserve knowledge of the client browser across many pages of the site.

**Instructor Notes:**

Add instructor notes here.

## 4.1: Session Handling

## Session Handling in Node.js



- Express with Node application makes it very easy to get sessions up and running
- Since sessions use cookies to keep track users we need both the cookie-parser and express-session modules.
- cookieParser need to be used before the session. This order is required for sessions to work. Create package.json using *npm init* command as given below

**Instructor Notes:**

Add instructor notes here.

#### 4.1: Session Handling

### Session Handling in Node.js



```
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');

var app = express();

app.use(cookieParser());
app.use(session({secret: 'IGATEKarthik', resave: true, saveUninitialized: true}));

app.get('/', function(req, res) {
  console.log('home : '+req.sessionID);
  if(req.session.lastPage) {
    res.write('Last page was: ' + req.session.lastPage + '\n');
  }
  req.session.lastPage = 'Home';
  res.end('Home Page');
});

app.get('/about', function(req, res) {
  console.log('about : '+req.sessionID);
  if(req.session.lastPage) {
    res.write('Last page was: ' + req.session.lastPage + '\n');
  }
  req.session.lastPage = 'AboutUs';
  res.end('About us');
});

app.listen(process.env.PORT || 3000);
console.log('Server listening on the port 3000');
```

July 31, 2018

Proprietary and Confidential

- 5 -

/\*Deny Directory Listing\*/

```
app.use('/', express.static(__dirname + '/'));
app.get('*', function(request, response, next) {
  response.sendFile(__dirname + '/index.html');
});
```

**Instructor Notes:**

Add instructor notes here.

## 4.1: Session Handling

## Storing session information in MongoDB



- Anytime we restart our app, all of the user sessions are lost, which gives a problem with our updating process.
- There are several ways to solve this problem, but one of the best way is to use an external store for the session data.
- More specifically, we can use MongoDB to store your session data. This way the session data is completely separate from our running application.
- To set up MongoDB with Express sessions, we need an extra module called *connect-mongo*.

```
{
  "name": "SessionDemo",
  "version": "1.0.0",
  "dependencies": {
    "connect-mongo": "0.5.3",
    "cookie-parser": "1.3.3",
    "express": "4.11.2",
    "express-session": "1.10.2",
    "nodemon": "1.2.1"
  }
}
```

July 31, 2018

Proprietary and Confidential

- 6 -

## Instructor Notes:

Add instructor notes here.

### 4.1: Session Handling Storing session information in MongoDB



```
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');
var MongoStore = require('connect-mongo')(session);

var app = express();

app.use(cookieParser());
app.use(session({
  store: new MongoStore({db: "SessionDB", host: "localhost", port: "27017", collection: "sessions"}),
  cookie: { maxAge: 50 * 1000 }, /* 5 min */
  secret: 'IGATEKarthik',
  resave: true,
  saveUninitialized: true
}));

app.get('/', function(req, res) {
  console.log('home : '+req.sessionID);
  if(req.session.lastPage) {
    res.write('Last page was: ' + req.session.lastPage + '\n');
  }
  req.session.lastPage = 'Home';
  res.end('Home Page');
});

app.get('/logout', function(req, res) {
  req.session.destroy();
  res.redirect("/");
});

app.listen(process.env.PORT || 3000);
console.log('Server listening on the port 3000');
```

July 31, 2018

Proprietary and Confidential

- 7 -


Failure to set "saveUninitialized" and "resave" will generate two warnings:

This is simply saying the default values will change so they want to ensure that by setting the values explicitly now. By doing this you won't run into unexpected behavior if the defaults do change in the near future.

Instructor Notes:

Demo

Session Handling Demo



July 31, 2018

Proprietary and Confidential

- 8 -



**Instructor Notes:**

Add instructor notes here.

**4.2: Cryptography  
Introduction**

- Cryptography is a method of storing and transmitting data in a particular form so that only those for whom it is intended can read and process it.
- As a best practice, any private information (like password) stored on the server should be encrypted.
- Hashing is the greatest way for protecting passwords and considered to be pretty safe for ensuring the integrity of data or password.
- Bcrypt hashing algorithm is a good option for secured password hashing. It is a one way hash i.e. It cannot be decrypted.
- bcryptjs* is the popular node module for hashing private information.

## Instructor Notes:

Add instructor notes here.

### 4.2: Cryptography Hashing data using bcryptjs



```
var bcrypt = require('bcryptjs');

/*Synchronous Approach */
var hash = bcrypt.hashSync("Karthik",bcrypt.genSaltSync(10));
bcrypt.compareSync("Karthik", hash); // returns true
bcrypt.compareSync("karthik", hash)); // returns false

/*Asynchronous Approach*/
bcrypt.hash("Karthik", bcrypt.genSaltSync(10), function(err, hash) {
  bcrypt.compare("Karthik", hash, function(err, res) {
    //res returns true
  });
});
```

**hashSync (data, salt)** : where data [required] is the data to be encrypted and salt [required] is the salt to be used in encryption.

**genSaltSync (rounds)** : where rounds - [OPTIONAL] - the number of rounds to process the data for. (default - 10)

Instructor Notes:

Demo

Hashing



July 31, 2018

Proprietary and Confidential

- 11 -