Node.js and Secure RESTful API

CPE405 - Advanced Computer Engineering Technology

Dome Potikanond
Navadon Khunlertgit
Banana Software co., Ltd

Computer Engineering, Chiang Mai University

Topics

- Some more basic on Node.js and ExpressJS
  - exports and module.exports
  - Express Routing
  - Express Error Handling
- Token Based Authentication
  - JWT: Jason Web Token
- Hands on

What is a Module?

- A module encapsulates related code into a single unit
  - Done by moving all related properties and functions into a file
- Exporting a module

```
/* greetings.js */
exports.sayHello = function() {
   return "HELLO";
};
exports.sayHola = function() {
   return "Hola";
};
```

```
/* greetings.js */
module.exports = {
   sayHello: function() {
        return "HELLO";
   },
   sayHola: function() {
        return "Hola"; }
};
```

https://www.sitepoint.com/understanding-module-exports-exports-node-js/

What is a Module? (2)

- Importing a module (as an object)
  - The keyword **require** is used in Node.js to import modules

```
/* main.js - use greetings.js */
var greetings = require(“./greetings”);
greetings.sayHello();          // “Hello”
greetings.sayHola();           // “Hola”
```

  - Similar to the code below

```
/* main.js - define greetings in itself */
var greetings = {
   sayHello: function() { return "Hello"; },
   sayHola: function() { return "Hola"; }
};
…
```

Express Routing

- Routing refers to application end points (URIs)
  - https://expressjs.com/en/guide/routing.html
- Route paths, in combination with a request method define an endpoints
  - Can be strings or regular expressions
  - `'/ab?cd'` – match `/acd` and `/abcd`
  - `'/ab+cd'` – match `/abcd`, `/abbcd`, `/abbbcd`, ...
  - `'/ab*cd'` – match `/abcd`, `/abxcd`, `/abRANDOMcd`, ...
  - `'/ab(cd)?e'` – match `/abe` and `/abcde`
  - `'/.*fly$/'` - match `butterfly`, `dragonfly`, NOT `butterflyman`

Express Routing (2)

- Route parameters

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

```
Route path: /flights/:from-:to
Request URL: http://localhost:3000/flights/LAX-SFO
req.params: { "from": "LAX", "to": "SFO" }
```

```
Route path: /plantae/:genus.:species
Request URL: http://localhost:3000/plantae/Prunus.persica
req.params: { "genus": "Prunus", "species": "persica" }
```

Express Error Handling

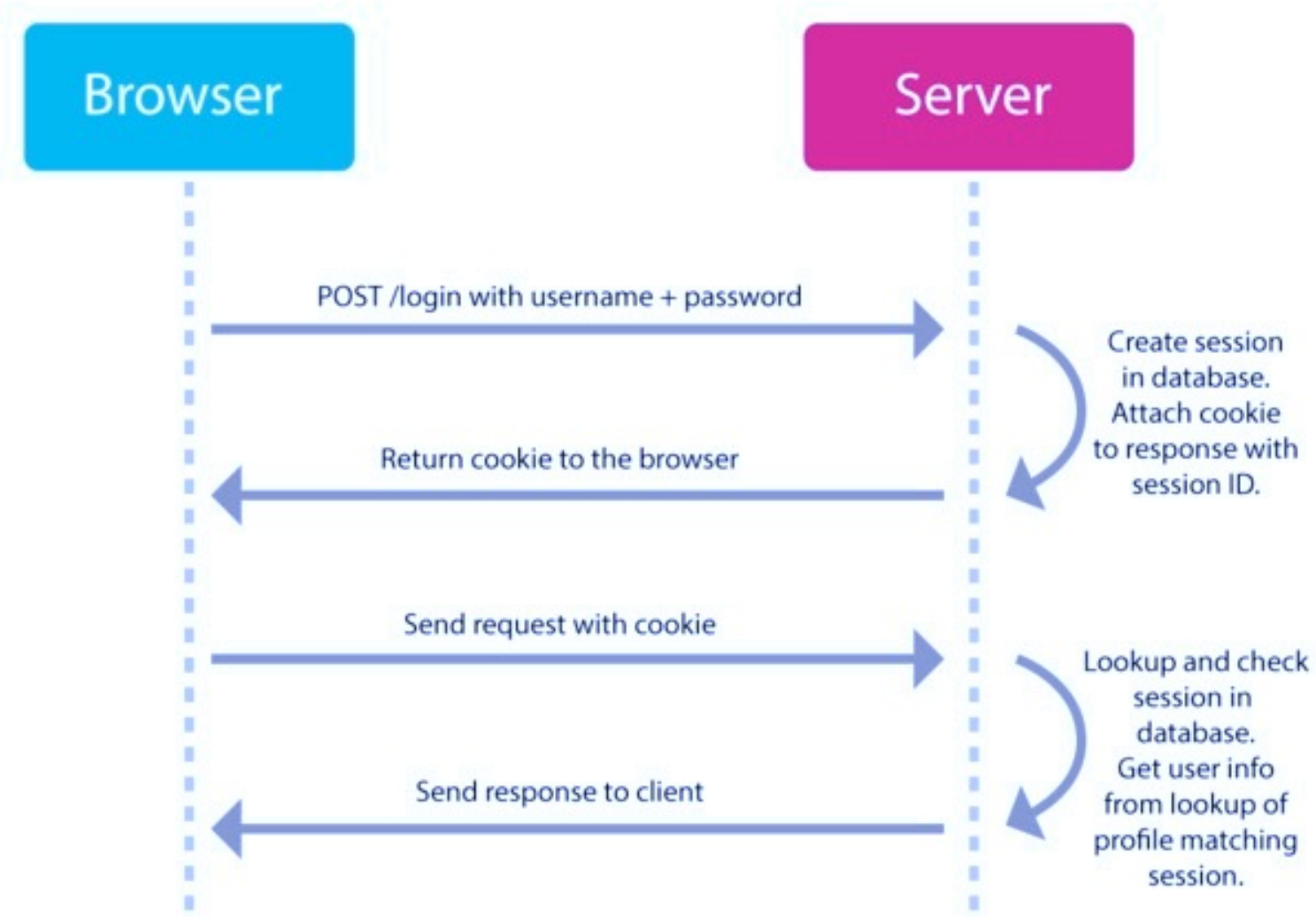
- Define error-handling middleware functions
  - http://expressjs.com/en/guide/error-handling.html
  - Error-handling functions have four arguments
    - `(err, req, res, next)`
  - Define last, after other `app.use()` and routes calls
- Response from within a middleware function can be
  - HTML error page, a simple message, a JSON string

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

Token Based Authentication

- Probably the best way to handle authentication for multiple users
  - Major API or web application are most likely use tokens
- Server Based Authentication (traditional method)
  - HTTP protocol is stateless – HTTP does NOT remember params
    - every time a user is authenticated, the server will need to create a record on the server (sessions in memory)
  - When there are many users authenticating, the overhead on the server increases
  - Having vital info in session memory will limit ability to scale

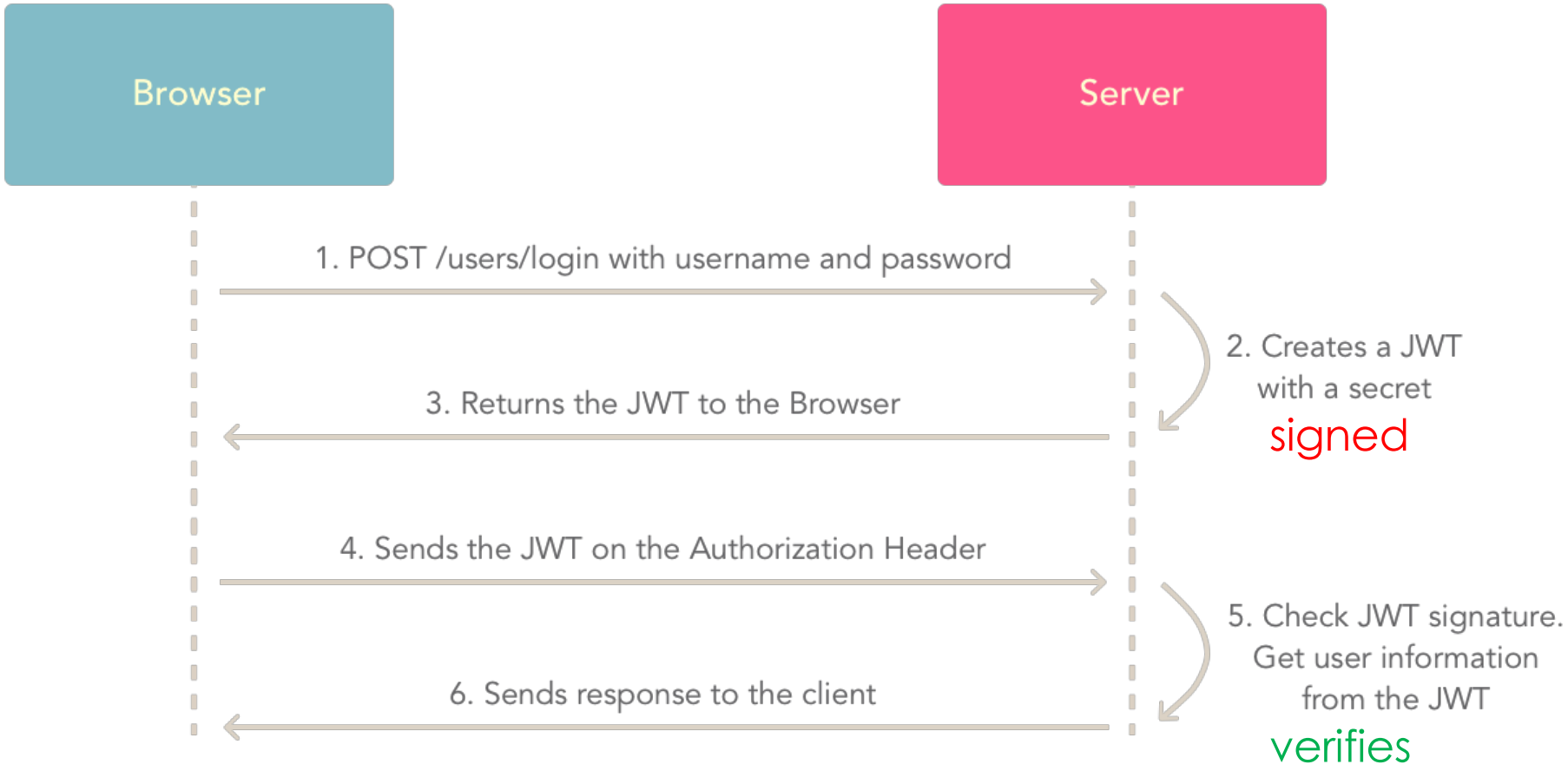
Server Based Authentication (traditional method)



Token Based Authentication (2)

- How Token Based Works
  - Server does not store any information about user on the server or in a session
  - No session information means your app can scale and add more machines as necessary
- Step-by-Step
  - User(or front-end app) requests access with username/password
  - API validates credentials and returns a signed token to the client
  - Client stores the token and sends it along with every request
  - API verifies token and responds with data

Token Based Authentication



Token Based Authentication (3)

- How to send a token from client to server
  - A token should be sent in the HTTP header
    - Keeping the idea of stateless HTTP requests
  - We can also use the token in a URL, a POST param as well
  - It is important to set our server to accept requests from all domains (not just localhost) using …

```
Access-Control-Allow-Origin: *
```

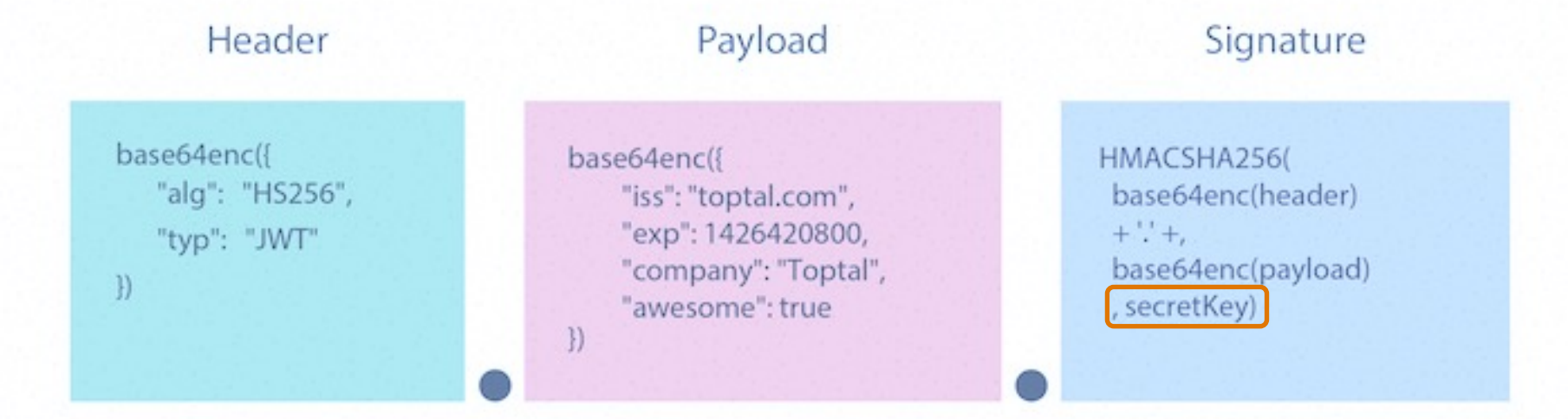
  - We can also pass (or delegate) the token to 3rd party apps
    - A permission based token (authorization)

Token Based Authentication (4)

- Benefits of Tokens:
  - Stateless and scalable servers
    - Do not need to keep sending the same user to the same server
  - Security
    - No cookie is sent, this helps to prevent CSRF
    - Token expires after a set amount of time
  - Pass authentication to other applications
    - Provide selective permissions to third-party application
    - e.g. allow other apps to post on Facebook or Twitter
  - Multiple platforms and Domain
    - Data and resources are available from any domain as long as a user has a valid token

JSON Web Tokens

- JWTs work across different programming languages
- JWTs are self-contained
  - They carry all the information necessary with in itself



eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ik
pvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

Hands on

- Create an API with Token based authentication using JWT
- Open API endpoints
  - Public API – unauthenticated users are allowed to access
- Authentication endpoint
  - For checking username and password against DB and return token
- Authenticated endpoints
  - Only for authenticated users
- Authenticated + Authorized endpoints
  - Only for authenticated users with admin privilege

Simple API for Web board app

Endpoints	HTTP Method	Description
/	GET	Welcome message
/api/login	POST	User Authentication using Token
/api/posts	GET	Get all posts
/api/posts	POST	Create a new post (*A)
/api/posts/:id	GET	Get a post by post_id
/api/posts/user/:id	GET	Get posts by user_id
/api/users	GET	Get all users
/api/users	POST	Create a new user (*AA)
/api/users/:id	GET	Get a user by user_id (*AA)
/api/users/:id	UPDATE	Update a user with user_id (*AA)
/api/users/:id	DELETE	Delete a user with user_id (*AA)

*A – for authenticated users, *AA – for authenticated users with admin authorization

Simple API - Dependencies

- Create a new project

```
# mkdir <project-dir>
# cd <project-dir>
# npm init
```

- Install all necessary dependencies

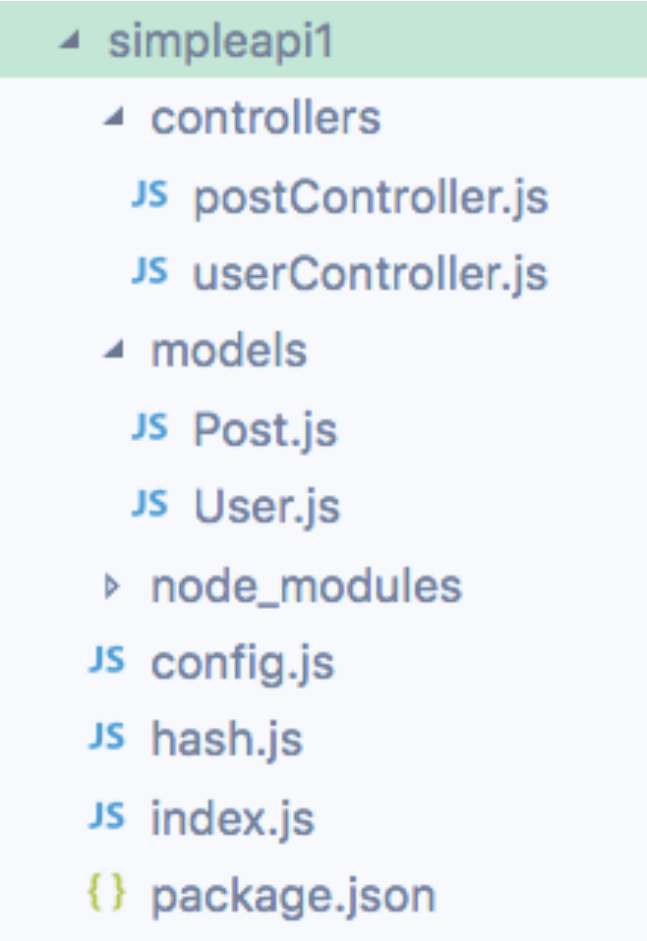
```
# npm install express body-parser morgan mongoose
   jsonwebtoken --save
```

- Optional: globally install nodemon for server live update

```
# npm install nodemon -g
```

Simple API – Project file Structure

- **simpleapi1** – project directory
  - **controllers**
    - Define control logic for user and post
    - Provide query methods
  - **models**
    - Define properties and constrains for user and post object
  - **config.js** : global configuration file
  - **hash.js** : a module for password hashing
  - **index.js** : entry points of the API app
    - Define routes



Simple API – Mongodb

- This project uses Mongodb for database and requires the collection for users and posts

- **User schema:**

(12) ObjectId("5a021fe22...	{ 10 fields }	Object
_id	ObjectId("5a021fe22ccd03606...	ObjectId
id	102	Int32
name	Jan van Holland	String
age	28	Int32
email	jholland@gmail.com	String
salt	e50555a71465c71f	String
passwdhash	8bf58f80a102384d590bd879a...	String
created	2017-11-07 21:04:34.921Z	Date
admin	false	Boolean

Simple API – Mongodb (2)

- **Post schema:**

(1) ObjectId("5a024d117...	{ 5 fields }	Object
_id	ObjectId("5a024d1172425ac9e...	ObjectId
userId	101	Int32
id	101	Int32
title	Dome - at nam consequatur ea l...	String
body	cupiditate quo est a modi nesci...	String
(2) ObjectId("5a024d9d7...	{ 5 fields }	Object
_id	ObjectId("5a024d9d72425ac9e...	ObjectId
userId	101	Int32
id	102	Int32
title	Dome2 - at nam consequatur e...	String
body	cupiditate2 quo est a modi nesc...	String

- Collection name:
  - Users or users
  - Posts or posts

Simple API – config.js

- Create a file to store a list of configurations
  - Mongodb connection string
  - JSON Web Token's secret key
  - Listening port, hostname, …

```
/* config.js */

module.exports = {
    'port': 3000,
    'hostname': 'localhost',
    'secret': 'myawesomeapi',
    'database': 'mongodb://<dbuser>:<dbpasswd><server>:<port>/<db>'
};
```

Simple API – Import dependencies

- Import all dependencies and global configuration into the entry point file

```
/* index.js - application entry point */
var express = require('express');
var app = express();

var bodyParser = require('body-parser'); // handling HTML body
var morgan = require('morgan');          // logging
var mongoose = require('mongoose');      // Mongodb library
var jwt = require('jsonwebtoken');       // token authentication

var config = require('./config');        // global config
var hash   = require('./hash');          // passwd hashing module
```

Simple API – configure app

- Configure middlewares, mongoose, server

```
/* index.js - application entry point */

// import dependencies
…
var port = process.env.PORT || config.port; // load port config
var hostname = config.hostname;             // load hostname config

mongoose.connect(config.database);          // setup mongoose

// use body parser so we can get info from POST and/or URL params
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

// use morgan to log requests to the console
app.use(morgan('dev'));
…
```

Simple API – add welcome route

- Add '/' as welcome route and start API server

```
/* index.js - application entry point */

// configure middlewares, mongoose and server
…
app.get('/', function(req, res) {
   res.send('Hello! The API is at http://localhost:'+port+'/api');
});

app.listen(port, hostname, () => {
   console.log('Simple API started at http://localhost:' + port);
});

…
```

Simple API – create User model

- Define **User** model in **models/User.js** using mongoose

```
/* User.js */
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

module.exports = mongoose.model('User', new Schema({
      id: { type: Number, required: true, unique: true },
    name: { type: String, required: true, trim: true },
     age: { type: Number, min: 13, max: 99 },
   email: { type: String, required: true, unique: true,
                match: /<regular expression>/ },
    salt: String,
   passwordhash: String,
   admin: { type: Boolean, default: false },
   created: { type: Date, default: Date.now }
}));
```

Simple API – create User model

- Define **`Post`** model in **`models/Post.js`** using mongoose

```
/* Post.js */
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

module.exports = mongoose.model('Post', new Schema({
      id: { type: Number, required: true, unique: true },
  userId: { type: Number, required: true },
   title: { type: String, required: true },
    body: { type: String, required: true },
  created: { type: Date, default: Date.now }
}));
```

Mongoose produces a collection name by pluralized the model's name.
We can manually specify a collection name as 3rd argument. See more at
http://mongoosejs.com/docs/api.html#index_Mongoose-model

Simple API – /api/users [GET]

- To get a list of all users, a route '/api/users' is created
  - To handle HTTP GET request, we use the `app.get(…)` method
  - This route will activate the **getUsers** function defined as User's controller, located in './controllers/userController.js'

```
/* index.js */

// import functions defined as User's controller
var Users = required('./controllers/userController.js');
…
app.get('/api/users', function(req,res) {
  Users.getUsers(req,res);    // passing request and respond objs.
});
…
```

```
http://localhost:3000/api/users (GET)
```

Simple API – Users.getUsers

- The **getUser** method use the **User** model to query mongodb with mongoose's **find** method
  - The callback function defines what to do with the results
  - An array of User objects is returned and stored in **users** variable

```
/* userController.js */
var mongoose = require('mongoose');
var User = require('../models/User');     // Import User model.

exports.getUsers = function(req, res) {
   User.find( (err, users) => {           // Define what to do
     if(err) throw err;                   // when query finished.
     res.json(users);                     // Using respond obj to
   });                                    // return users as JSON.
};
…
```

```
{
    "_id": "5a021fe22ccd0360675fd0c5",
    "id": 102,
    "name": "Jan van Holland",
    "age": 28,
    "email": "jholland@gmail.com",
    "salt": "e50555a71465c71f",
    "passwdhash": "8bf58f80a102384d590bd879a033a918579c6a6f0dea7791cc666
        01e749d415a3b6aebb3b7e8e41053dbbd1b6058712",
    "__v": 0,
    "created": "2017-11-07T21:04:34.921Z",
    "admin": false
},
{
    "_id": "5a044c1357a62f878d758a32",
    "id": 0,
    "name": "Hugo Boss",
    "age": 33,
    "email": "hboss@gmail.com",
    "salt": "7e1954f3608b16f2",
    "passwdhash": "b963824b36561401883d2208e247475433edde019acb227f164cf
        db575ba4604a8aa41e1b7852068b85f150c2d02bc9",
    "__v": 0,
    "created": "2017-11-09T12:37:39.358Z",
    "admin": false
}
```


http://localhost:3000/api/users (GET)

Simple API – /api/users/id/:id [GET]

- Get a user data with specified `id`
  - This route will activate the **`getUserById`** function defined in the user controller

```
/* index.js */

…
app.get('/api/users/id/:id', function(req,res) {
  Users.getUserById(req,res); // passing request and respond objs.
});
…
```

```
http://localhost:3000/api/users/id/101 (GET)
```

Simple API – Users.getUserById

- The `getUserById` method
  - A User objects is returned and stored in **`user`** variable

```
/* userController.js */
…
exports.getUserById = function(req, res) {
  User.find({id: req.params.id}, (err, user) => {   // req.params.id = 101
    if(err) throw err;
    if(user && user.length != 0)            // check a user is found
      res.json(user);
    else
      res.status(404).json({                // if not found, return
        success: false,                     // an error message
        message: 'user not found!'
      });
  });
};
…
```

```
{
    "_id": "5a01e782cdcf8e52efffe3ff",
    "id": 101,
    "name": "Dome Potikanond",
    "age": 40,
    "email": "dome@gmail.com",
    "salt": "9e2013204ee708c0",
    "passwdhash": "508ab3b92a71a0c1211c24d1202049ab6493ce6ae1c7956dca9
        9143d9941be0bdd066a3000ea0786b2cbc41f40",
    "__v": 0,
    "created": "2017-11-07T17:04:02.201Z",
    "admin": true
}
```


http://localhost:3000/api/users/id/101 (GET)

Simple API – /api/users/oid/:_id [GET]

- Get a user data with specified ObjectId: `_id`
  - This route will activate the **`getUserByOId`** function in the user controller

```
/* index.js */

…
app.get('/api/users/:_id', function(req,res) {
  Users.getUserByOId(req,res);
});
…
```

```
http://localhost:3000/api/users/5a024d1172425ac9e6b62c78 (GET)
```

Simple API – Users.getUserByOId

- The `getUserByOId` method
  - A User objects is returned and stored in `user` variable

```
/* userController.js */
…
exports.getUserByOId = function(req, res) {
  User.findById( req.params._id, (err, user) => {
    if(err) throw err;
    if(user && user.length != 0)            // check a user is found
      res.json(user);
    else
      res.status(404).json({                // if not found, return
        success: false,                     // an error message
        message: 'user not found!'
      });
  });
};
…
```

```
{
    "_id": "5a01e782cdcf8e52efffe3ff",
    "id": 101,
    "name": "Dome Potikanond",
    "age": 40,
    "email": "dome@gmail.com",
    "salt": "9e2013204ee708c0",
    "passwdhash": "508ab3b92a71a0c1211c24d1202049ab6493ce6ae1c7956dca9
        9143d9941be0bdd066a3000ea0786b2cbc41f40",
    "__v": 0,
    "created": "2017-11-07T17:04:02.201Z",
    "admin": true
}
```


http://localhost:3000/api/users/5a024d1172425ac9e6b62c78 (GET)

Simple API – /api/users/ [POST]

- Add a new user with specified information
  - This route will activate the **signup** function in the user controller

```
/* index.js */
…
app.post('/api/users', function(req,res) {
  Users.signup(req,res);
});
…
```

```
http://localhost:3000/api/users (POST)
Passing parameters in the body (x-www-form-urlencoded):
    name: Hugo Boss
     age: 33
   email: hboss@gmail.com
Password: goodpassword
```

Simple API – Users.signup

```
/* userController.js */
var hash = require('../hash');     // import salt and hash functions
…
exports.signup = function(req, res) {
  var salt = hash.genRandomString(16);
  var pwd_data = hash.sha512(req.body.password, salt);
  // find a user with maximum id: find all users and sort by id max-to-min
  User.find({}).sort({id: -1}).limit(1).exec( (err, users) => {
     if(err) throw err;
     if(users && users.length != 0) {
       var newUser = new User({
          id: users[0].id + 1,     // users is an array of User objects
          name: req.body.name,
          age: parseInt(req.body.age),
          email: req.body.email,
          salt: pwd_data.salt,
          passwdhash: pwd_data.passwordHash,
          admin: req.body.admin?req.body.admin:false
       });
       … // continue on next slide
```

Simple API – Users.signup (2)

```
/* userController.js */
     … // continue from previous slide
        newUser.save( function(err, user) {
           if(err) {
             return res.json({
               success: false,
               message: 'Unable to add new user!',
             });
           } else {
             return res.json({
               success: true,
               message: 'New user has been created',
               user: {
                 name: newUser.name,
                 email: newUser.email,
                 admin: newUser.admin
               }
             });
           }
        });        // continue on next slide
```

Simple API – Users.signup (3)

```
/* userController.js */
     … // continue from previous
     } else {
       res.json({
         success: false,
         message: 'User cannot be added!'
       });
     }
  });
};
```

```
{
    "success": true,
    "message": "New user has been created",
    "user": {
        "name": "Hugo Boss",
        "email": "hboss@gmail.com",
        "admin": false
    }
}
```

_id	ObjectId("5a04585a170c8b896...
id	103
name	Hugo Boss
age	33
email	hboss@gmail.com
salt	80bd2b900218a535
passwdhash	83a09e91a446dd004d67ce04e...
created	2017-11-09 13:30:02.063Z
admin	false

Simple API – Salt and Hash

- Salting and hashing a password before saving to the database is a standard procedure
  - Salting : create a fixed length random string (salt)
  - Hashing : calculate a fixed length encoded string from plaintext
- Hashed password:
  - Hashedpassword = **Hashing(plaintext password + salt)**

```
…
  var salt = hash.genRandomString(16);     // a salt of length 16
  var passwordData = hash.sha512(password, salt);   // return obj.
…
```

Simple API – Salt and Hash (2)

- Salt and Hash functions are defined in hash.js

```
/* hash.js */
var crypto = require('crypto');    // import built-in cryptographic library

exports.genRandomString = function(length){
  return crypto.randomBytes(Math.ceil(length/2))
          .toString('hex')  /* convert to hexadecimal format */
          .slice(0,length); /* return required number of characters */
};

exports.sha512 = function(password, salt){
  var hash = crypto.createHmac('sha512', salt); /* Hashing with sha512 */
  hash.update(password);
  var value = hash.digest('hex');
  return {
    salt:salt,
    passwordHash:value
  };
};
```

Simple API – /api/login [POST]

- Login a user with specified email and password
  - Typically, the connection between client and API server should be secured by **HTTPS**

```
/* index.js */
…
app.post('/api/login', function(req,res) {
  Users.login(req,res);
});
…
```

```
http://localhost:3000/api/login (POST)
Passing parameters in the body (x-www-form-urlencoded):
   email: dome@gmail.com
Password: mypassword
```

Simple API – Users.login

- The `login` method

```
/* userController.js */
exports.login = function(req, res) { ... What to do ... }
```

  - Find a user with an **email** = `req.body.email`
    - If found, return an object of user and store in user variable
  - Calculate **hash1** = `sha512(req.body.password, user.salt)`
  - Compare **hash1** to `user.passwdhash`
    - If not equal, return error message
  - Create a payload object: `user.id, user.email, user.admin`
  - Sign a token containing the payload with the secret key
    - Include the expiration time (e.g. 86400 for a day)
  - Send the **signed token** back to the client

Simple API – Users.login (2)

- Find a user with an **email** = `req.body.email`
- Calculate **hash1** = `sha512(req.body.password, user.salt)`

```
/* userController.js */
// This code is inside exports.login = function(res,req) {…}
User.findOne({ email: req.body.email }, function(err, user) {
  if (err) throw err;
  if (!user) {
    res.status(401).json({
      success: false,
      message: 'Authentication failed. User not found.'
    });
  } else if(user) {
    var passwdData = hash.sha512(req.body.password, user.salt);
    …
    // continue on next slide
```

Simple API – Users.login (3)

- Compare **hash1** to **`user.passwdhash`**
- Create a **payload** object: **`user.id, user.email, user.admin`**

```
/* userController.js */
    …
    if (user.passwdhash != passwdData.passwordHash) {
      return res.json({
        success: false,
        message: 'Authentication failed. Wrong password.' });
    } else {
      const payload = {
        id: user.id,
        email: user.email,
        admin: user.admin
      };
      …
```

Simple API – Users.login (4)

- Add payload, sign the token with secret key and send a response

```
/* userController.js */
      …
      var token = jwt.sign(payload, config.secret, {
        expiresIn: 86400 // expires in 24 hours
      });
      return res.json({
        success: true,
        message: 'Enjoy your token!',
        token: token
      });
    }
  } // end of else if(user)
}); // end of the callback function
```

```
http://localhost:3000/api/login (POST)
Passing parameters in the body (x-www-form-urlencoded):
   email: dome@gmail.com
Password: mypassword
```


```
{
    "success": true,
    "message": "Enjoy your token!",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
        .eyJpZCI6MTAxLCJlbWFpbCI6ImRvbWVAZZ1haWwuY29tIiwiYWRtaW4iOnRydWUsImlh
        NTQxfQ.TAI84z9TODDXcvl7KeuISkQIyK0hWTIvtvRk0KQAsWg"
}
```


Client has to store the returned token and sent it with requests when necessary

Simple API – Token Verification

- Create a middleware to verify token for sensitive data
  - Place the non-sensitive routes before the middleware
  - Place the sensitive routes after the middleware

```
/* index.js */

/* non-sensitive routes */
// e.g. /api/login, /api/posts/, /api/posts/:_id, …

app.user( function(req, res, next) {
  // code for token verification - continue on next slides
  // if token is valid, continue to the specified sensitive route
  // if token is NOT valid, return error message
});


/* all sensitive routes */
// e.g. /api/users (admin-only), /api/posts [post], …
```

Simple API – Token Verification (2)

```
/* index.js */
// read a token from body or urlencoded or header (key = x-access-token)
var token = req.body.token || req.query.token || req.headers['x-access-token'];

if (token) {
  jwt.verify(token, config.secret, function(err, decoded) {
    if (err) {
      return res.json({ success: false, message: 'Invalid token.' });
    } else {
      req.decoded = decoded;       // add decoded token to request obj.
      next();                      // continue to the sensitive route
    }
  });
} else {
  return res.status(403).send({
    success: false,
    message: 'No token provided.'
  });
}
```

```
{
  "id": 101,
  "email": "dome@gmail.com",
  "admin": true,
  "iat": 1510231033,
  "exp": 1510317433
}
```

Simple API – Token Verification (3)

- From now on, to access sensitive routes

```
/* index.js */

app.user( function(req, res, next) {
  // Token verification middleware
});


/* all sensitive routes */
// e.g. /api/users (admin-only), /api/posts [post], …
```

- Each request must include a valid token

```
http://localhost:3000/api/users (GET)
Passing the token in the header:
   x-access-token: yJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI
                   6MTAxLCJlbWFpbCI6ImRvbWVAZ21haWwuY29tIiw...
```

Simple API – Token Verification (3)

- Some of the sensitive routes may also required admin privilege (admin = true) – authorization
  - we can add a logic to check for admin authorization

```
/* index.js */
…
app.get('/api/users', function(req,res) {
  if (req.decoded.admin)       // check admin authorization
    users.getUsers(req,res);
  else {
    res.status(401).json( {    // if not an admin user, return
      success: false,          // an error message
      message: 'Unauthorized Access'
    });
});
…
```

References

- The ins and outs of token based authentication
  https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication
- JSON Web Token
  https://scotch.io/tutorials/the-anatomy-of-a-json-web-token
- Node.js Crypto Module (w3schools.com)
  https://www.w3schools.com/nodejs/ref_crypto.asp
- bcrypt (npmjs.com)
  https://www.npmjs.com/package/bcrypt
- Node.js Tutorial For Absolute Beginners (Traversy Media)
  - https://www.youtube.com/watch?v=U8XF6AFGqlc&index=20&list=WL