**Express JS Cheat Sheet**

1. Simple **app** with express.js

const express = require('express');

const app = express();

app.use('/', (req, res, next) => {

  console.log('This always runs!');

  next();

});

app.use('/add-product', (req, res, next) => {

  res.send('<h1>Hello from Express!</h1>');

});

app.use('/', (req, res, next) => {

  res.send('<h1>Another middleware</h1>');

});

app.listen(3000);

1. app.js file

const path = require('path');

const express = require('express');

const bodyParser = require('body-parser');

const adminRoutes = require('./routes/admin');

const shopRoutes = require('./routes/shop');

const app = express();

app.use(bodyParser.urlencoded({extended: false}));

app.use(express.static(path.join(\_\_dirname, 'public')));

app.use('/admin', adminRoutes);

app.use(shopRoutes);

app.use((req, res, next) => {

    res.status(404).sendFile(path.join(\_\_dirname, 'views', '404.html'));

});

app.listen(3000);

1. admin.js file

const path = require('path');

const express = require('express');

const rootDir = require('../util/path');

const router = express.Router();

// /admin/add-product => GET

router.get('/add-product', (req, res, next) => {

  res.sendFile(path.join(rootDir, 'views', 'add-product.html'));

});

// /admin/add-product => POST

router.post('/add-product', (req, res, next) => {

  console.log(req.body); // fetch data by body-parser

  res.redirect('/');

});

module.exports = router;

1. shop.js file

const path = require('path');

const express = require('express');

const rootDir = require('../util/path');

const router = express.Router();

router.get('/', (req, res, next) => {

  res.sendFile(path.join(rootDir, 'views', 'shop.html'));

});

module.exports = router;

1. util/path.js

const path = require('path');

module.exports = path.dirname(process.mainModule.filename);

1. **Dynamic Routing fetch:** **req.query.edit** fetching the values from the URL like www.something.com**?edit=true&name=Sayan**

router.get('/products/:productId', (req, res, next) => {

const prodId = req.params.productId; // Fetch Id from url

  const editMode = req.query.edit; // Fetch query data from url

  if (!editMode) {

    res.redirect('/');

  }

  res.render('admin/edit-product', {

    pageTitle: 'Add Product',

    path: '/admin/add-product',

    editing: editMode

  });

});

1. When sending form inputs through the **POST method**, then in express code, the inputs can find in the request body. E.g., one form input has name like ***<input type = “text” name = “name”/>***. To fetch from the middleware, the code will be, ***req.body.name*** where name should be defined in the input attribute.
2. **Session & Cookies –** Cookies store inside the browser whether session store in system memory and also into the database. Also, session converts to the hash code values.
   1. **Set Cookie-**

exports.postLogin = (req, res, next) => {

res.setHeader(‘Set-Cookie’, ‘loggedIn=true; Expires=...; Max-Age=...; Secure; HttpOnly’)

}

* 1. **Extract the Cookie-**

exports.getLogin = (req, res, next) => {

const cookieData = req.get(‘Cookie’);

}

* 1. **Set Session –** In system memory

const session = require(‘express-session’); // npm install –save express-session

app.use(session({

secret: ‘my secret’, resave: false, saveUninitialized: false, cookie: {...}

}));

// use the session

exports.postLogin = (req, res, next) => {

req.session.isLoggedIn = true;

}

* 1. **Set Session in Database (MongoDB):** *npm install –save connect-mongodb-session*

const MongoDBStore = require(‘connect-mongodb-session’);

const store = new MongoDBStore({

uri: ‘database url’,

collection: ‘sessions’

});

app.use(session({

secret: ‘my secret’, resave: false, saveUninitialized: false, cookie: {...}, store: store

}));

// use the session

exports.postLogin = (req, res, next) => {

req.session.isLoggedIn = true;

}

* 1. **Deleteing Session:** It also delete from the database.

exports.postLogout = (req, res, next) => {

req.session.destroy((err) => {

console.log(err);

res.redirect(‘/’);

});

}

// Have to visit later again.

req.session.save(err => {

console.log(err);

res.redirect(‘/’);

})

1. **Authentication:** 
   1. **Hash the password: *npm install –save bcryptjs***

const bcrypt = require(‘bcryptjs’);

exports.signup = (req, res, next) => {

const password = req.body.password;

User.findOne({email: email}).then(userDoc => {

if (userDoc) {

return res.redirect(‘/’);

}

return bcrypt.hash(password, 12);

}).then(hashPassword => {

// save into database

})

};

* 1. **Hash password to simple:**

bcrypt.compare(password, user.password)

.then(doMatch => {

if (doMatch) {

req.session.isLoggedIn = true;

req.session.user = user;

return res.redirect(‘/’);

}

}).catch(err => {redirect(‘/login’)});

* 1. **CSFR Attacks:** CSRF stands for Cross-Site Request Forgery. ***npm install –save csurf.***

initialize the csrf token into backend file

const csrf = require(‘csurf’);

const csrfProtection = csrf();

// initialize the session before

app.use(csrfProtection);

When rendering, pass the csrf token to the views html files that can pass for later uses

exports.getIndex = (req, res, next) => {

Product.find()

.then(products => {

Res.render(‘shop/index’, {

Prods: products,

csrfToken: req.cerfToken()

})

})

.catch()

};

 In template engine, store the csrf token

<form **action**=”/logout” **method** = “POST”>

<input type = “hidden” name=”\_csrf” value = “<% csrfToken %>”>

<button **type**=”submit”>Logout</button>

</form>

**For every render and managing separetlt, there is better solution –**

Main app file(entry file):

app.use((req, res, next) => {

res.locals.isAuthenticated = req.sessions.isLoggedIn;

res.locals.csrdToken = req.cerfToken();

next();

});

And then add this input field in every render files.

<input type = “hidden” name=”\_csrf” value = “<% csrfToken %>”>

* 1. **Pass Flash Message: *npm install --save connect-flash***

Define globally the flash 🡪

const flash = require(‘connect-flash’);

// after initialized the session

app.use(flash());

Define the flash message 🡪

exports.postLogin = (req, res, next) => {

User.findOne({email: email})

.then(user => {

if(!user){

req.flash(‘error’, ‘Invalid email or password ’);

return req.redirect(‘/login’);

}

})

.catch()

};

Use the flash message 🡪

exports.getLogin = (req, res, next) => {

res.render(‘auth/login’, {

path: ‘/login’,

errorMessage: req.flash(‘error’);

})

}

* 1. **Sending Mail(Send Grid): *npm install –save nodemailer nodemailer-sendgrid-transport***

const nodemailer = require(‘nodemailer’);

const sendgridTransport = require(‘nodemailer-sendgrid-transport’);

const transporter = nodemailer.createTransport({

sendgridTransport({

auth: {

api\_key: ‘your api key’

}

})

});

transporter.sendMail({

to: email,

from: ‘shop@node-complete.com’

subject: ‘Signup succeeded!’

html: ‘<h1>You successfully signup!</h1>’

}).catch(err => {

Console.log(err);

})

* 1. **Reset Password through mail:**

Create Token

const crypto = require(‘crypto’);

exports.postReset = (req, res, next) => {

crypto.randomBytes(32, (err, buffer) => {

if (err) {

console.log(err);

return res.redirect(‘/reset’);

}

const token = buffer.toString(‘hex’);

});

};

Send Mail

User.findOne({email: req.body.email})

.then(user => {

if (!user){

req.flash(‘error’, ‘No account found!’);

return res.redirect(‘/reset’);

}

user.resetToken = token;

user.resetTokenExpiration = Date.now() + 3600000;

user.save();

})

.then(result => {

res.redirect(‘/’);

transporter.sendMail({

to: req.body.email,

from: ‘shop@node.com’,

subject: ‘Reset Password’,

html: `<p>You requested a password reset</p>

<p>Click this <a href = “https://localhost:3000/reset/${token}”>link</a> to set a new password.</p>

`

})

})

.catch(err => {

console.log(err);

});

1. **Validation: *npm install –save express-validator.*** For more information, check the GitHub documentation. **Have to come later here.**
   1. **Email, Password & Confirm Password Validation**: Pass the middleware

const {check, body, params, cookie, …}= require(‘express-validator’/check);

// email is the name input field inside html code

router.post(

‘/signup’,

[

check(‘email’)

.isEmail()

.withMessage(‘Please enter a valid email’)

.custom((value, {req} => {

if (value === ‘test@test.com’){

throw new Error(‘This email address is forbidden’);

}

return true;

})

),

body

(

‘password’,

‘Please enter a password with only numbers and text and at least 5 charecters and maximum 15.’

)

.isLength({min: 5, max: 15})

.withMessage(‘some error message’)

.isAlphanumeric(),

body(‘confirmPassword’)

.custom((value, {req}) => {

if (value !== req.body.password) {

throw new Error(‘Password have to match’);

}

return true;

})

],

authController.postSignup);

Controller Code

const {validationResult} = require(‘express-validator/check’);

exports.postSignup = (req, res, next) => {

const email = req.body.email;

const error = validationResult(req);

if (!error.isEmpty()) {

return res.status(422).render();

}

};

1. **Files Handling-**
   1. **Upload Files-** Bodyparser can’t parse the file data. It’s only for handling text input data. For that, have to install new package. ***npm install --save multer.***

**HTML File** 🡪

<form **action** = “/admin/product” **method** = “POST” **enctype** = “multipart/form-data”>

<input **type** = “file” **name** = “image”/>

</form>

**Express Code**(main file like app.js) 🡪 Define

const multer = require(‘multer’);

const fileStorage = multer.diskStorage({

destination: (req, file, callback) => {

callback(null, ‘images’);

},

filename: (req, file, callback) => {

callback(null, new Date().toISOString() + ‘-’ + file.originalname);

}

});

const fileFilter = (req, file, callback) => {

if (file.mimetype === ‘image/png’ || file.mimetype === ‘image/jpg’) {

callback(null, true);

} else {

callback(null, false);

}

};

app.use(bodyParser.urlencoded({extended: false}));

// for one file & input field name is image

app.use(multer(

{

storage: fileStorage,

fileFilter: fileFilter

}).single(‘image’));

**Use 🡪**

exports.formHandling = (req, res, next) => {

const image = req.file;

console.log(image);

};

* 1. **Serve the files (images):**
     1. Statically serve files

// serve css files

app.use(express.static(path.join(\_\_dirname, ‘public’)));

// serve image files

app.use(express.static(path.join(\_\_dirname, ‘images’)));

* + 1. **Serve File through filesystem-**

const fs = require(‘fs’);

const path = require(‘path’);

exports.getInvoice = (req, res, next) => {

const orderId = req.params.orderId;

const invoiceName = ‘invoice-’ + orderId + ’.pdf’;

const invoicePath = path.join(‘data’, ‘invoice’, invoiceName);

fs.readFile(invoicePath, (err, data) => {

if (err){

next();

}

res.setHeader(‘Content-Type’, ‘application/pdf’);

res.setHeader(‘Content-Disposition’, ‘attachment; filename=”’+invoiceName+‘”’);

res.send(data);

});

}

* + 1. **Another Method 🡪** Recommended for bigger files

const fs = require(‘fs’);

const path = require(‘path’);

exports.getInvoice = (req, res, next) => {

const orderId = req.params.orderId;

const invoiceName = ...

const invoicePath = ...

const file = fs.createReadStream(invoicePath);

res.setHeader(‘Content-Type’, ‘application/pdf’);

res.setHeader(...);

file.pipe(res);

};

* + 1. **Generate File & Serve 🡪** ***npm install --save pdfkit***

const PDFDocument = require(‘pdfkit’);

exports.getInvoice = (req, res, next) => {

const orderId = req.params.orderId;

Order.findById(orderId)

.then(order => {

if (!order){

return next(new Error(‘No order found’));

}

if (order.user.userId.toString() !== req.user.\_id.toString()) {

return next(new Error(‘Unauthorized’));

}

const invoiceName = ...

const invoicePath = ...

const pdfDoc = new PDFDocument();

res.setHeader(‘Content-Type’, ‘application/pdf’);

res.setHeader(‘Content-Disposition’, ‘inline; filename ...’);

pdfDoc.pipe(fs.createWriteStream(invoicePath));

pdfDoc.pipe(res);

pdfDoc.text(‘Hello world!’);

pdfDoc.fontSize(26).text(‘Invoice’, {

underline: true

});

pdfDoc.end();

const file = fs

});

};

1. **Pagination:**
   1. **Skip already loaded products to go to different page:**

Product.find()

.skip((page – 1) \* ITEMS\_PER\_PAGE)

.limit(ITEMS\_PER\_PAGE)

.then(products => {

res.render(‘shop/index’, {

prods: products,

path: ‘/’

})

})

* 1. **Preparing pagination data on the server:**

let totalItems;

Product

.find()

.countDocuments()

.then(numProducts => {

totalItems = numProducts;

return Product

.find()

.skip((page - 1)\*ITEMS\_PER\_PAGE)

.limit(ITEMS\_PER\_PAGE);

})

.then(products => {

res.render(‘shop/index’, {

prods: products,

path: ‘/’,

currentPage: page,

hasNextPage: ITEMS\_PER\_PAGE \* page < totalProducts,

hasPreviousPage: page > 1,

nextPage: Page + 1,

previousPage: page – 1,

lastPage: Math.ceil(totalItems / ITEMS\_PER\_PAGE)

})

})

* 1. **In HTML Templates:**

<section>

<% if (currentPage != 1 && previousPage !== 1) { %>

<a **href**=”/?page=1”>1</a>

<% } %>

<a href=”/?page=<% currentPage %>” class=”active”><% currentPage %></a>

  <% if (hasPreviousPage) { %>

<a href=”/?page=<% previousPage %>” class=”active”><% previousPage %></a>

<% } %>

<% if (hasNextPage) { %>

<a href=”/?page=<% nextPage %>” class=”active”><% nextPage %></a>

<% } %>

<% if (lastPage !== currentPage && nextPage !== lastPage) { %>

<a href=”/?page=<% lastPage %>” class=”active”><% lastPage %></a>

<% } %>

</section>

1. **Asynchronous Request:**

<div **class**=”card\_actions”>

<a **href**=”/admin/edit-product”>Link</a>

<input type=”hidden” value=”<%= product.\_id %>” name = “productId”>

<input type=”hidden” name=”\_csrf” value = “<%= csrdToken %>”>

<button **class**=”btn” **type**=”button” **onClick** = “deleteProduct(this)”>Delete</button>

<div>

const deleteProduct = (btn) => {

const prodId = btn.parentNode.querySelector(‘[name=productId]’).value;

const csrf = btn.parentNode.querySelector(‘[name=\_csrf]’).value;

const productElement = btn.closest(‘article’);

fetch(‘/admin/product/’ + prodId, {

method: ‘DELETE’,

headers: {

‘csrf-token’: csrf

}

})

.then(result => {

return result.json();

})

.then(data => {

console.log(data);

productElement.remove();

})

.catch(err => {

console.log(err);

});

};

router.delete(‘/product:productId’, isAuth, adminController.deleteProduct);

exports.deleteProduct = (req, res, next) => {

const prodId = req.params.productId;

Product.findById(prodId)

.then(product => {

if(!product){

return next(new Error(‘Product not found’));

}

fileHelper.deleteFile(product.imageUrl);

return Product.deleteOne({\_id: prodId, userId: req.user.\_id});

})

.then(() => {

console.log(‘DESTROYED PrRODUCT’);

res.status(200).json({message: “Success!”});

})

.catch(err => {

res.status(500).json({message: “Deleting product failed”});

})

};

1. **Payments (Stripe 3rd Party Provider): Have to come later here.**
   1. **Codes🡪**

**Router**

route.get(‘/checkout’, isAuth, shopController.getCheckout);

**Controller**

exports.getCheckout = (req, res, next) => {

req.user

.populate(‘cart.items.productId’)

.execPopulate()

.then(user => {

const products = user.cart.items;

let total = 0;

products.forEach(p => {

total += p.quantity \* p.productId.price;

});

res.render(‘shop/checkout, {

path: ‘/checkout’,

products: products,

totalSum: total

})

})

.catch(err => {

const error = new Error(err);

error.httpStatusCode = 500;

return next(error);

})

};

1. **Rest API & JWT:**
   1. **GET:** Get a resource from the server
   2. **POST:** Post a resource to the server i.e. create or appebd Resource
   3. **PUT:** Put a resource onto the server i.e. create or overwrite a resource
   4. **PATCH:** Update parts of an existing Resource on the server
   5. **DELETE:** Delete a resource on the server
   6. **Options:** Determine whether follo-up Request is allowed (sent automatically)
   7. **JWT:** JavaScript Web Token.

**Define JWT authentication-**

const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {

  const authHeader = req.get('Authorization');

  if (!authHeader) {

    const error = new Error('Not authenticated.');

    error.statusCode = 401;

    throw error;

  }

  const token = authHeader.split(' ')[1];

  let decodedToken;

  try {

    decodedToken = jwt.verify(token, 'somesupersecretsecret');

  } catch (err) {

    err.statusCode = 500;

    throw err;

  }

  if (!decodedToken) {

    const error = new Error('Not authenticated.');

    error.statusCode = 401;

    throw error;

  }

  req.userId = decodedToken.userId;

  next();

};

**Pass this auth through Api:**

const express = require('express');

const { body } = require('express-validator');

// PUT Update Post

router.put(

  '/post/:postId',

  isAuth,

  [

    body('title').trim().isLength({ min: 5 }),

    body('content').trim().isLength({ min: 5 }),

  ],

  feedController.updatePost

);

**In Controller:**

const jwt = require('jsonwebtoken');

exports.login = (req, res, next) => {

  const email = req.body.email;

  const password = req.body.password;

  let loadedUser;

  User.findOne({ email: email })

    .then((user) => {

      if (!user) {

        const error = new Error('A user with this email could not be found');

        error.statusCode = 401;

        throw error;

      }

      loadedUser = user;

      return bcrypt.compare(password, user.password);

    })

    .then((isEqual) => {

      if (!isEqual) {

        const error = new Error('Wrong Password!');

        error.statusCode = 401;

        throw error;

      }

      // generate json web token - npm install --save jsonwebtoken

      const token = jwt.sign(

        {

          email: loadedUser.email,

          userId: loadedUser.\_id.toString(),

        },

        'somesupersecretsecret',

        { expiresIn: '1h' }

      );

      res.status(200).json({ token: token, userId: loadedUser.\_id.toString() });

    })

    .catch((err) => {

      if (!err.statusCode) {

        err.statusCode = 500;

      }

      next(err);

    });

};

**In fornt-end:**

fetch('http://localhost:8080/feed/post/' + postId, {

      method: 'DELETE',

      headers: {

        Authorization: 'Bearer ' + this.props.token

      }

    })

1. **WebSocket & Socket.io:** 
   1. **Create server-side Connection: *npm install –save socket.io***

mongoose

  .connect('mongodb://localhost/restApiPost', {

    useNewUrlParser: true,

    useUnifiedTopology: true,

  })

  .then((result) => {

    const server = app.listen(8080);

    const io = require('socket.io')(server, {

      cors: {

        origin: 'http://localhost:3000',

        credentials: true,

      },

    });

    io.on('connection', (socket) => {

      console.log('Client connected');

    });

  })

  .catch((err) => {

    console.log(err);

  });

* 1. **Create client side Connection:** ***npm install –save socket.io-client***

import openSocket from 'socket.io-client';

componentDidMount() {

    fetch('URL')

      .then(res => {

        if (res.status !== 200) {

          throw new Error('Failed to fetch user status.');

        }

        return res.json();

      })

      .then(resData => {

        this.setState({ status: resData.status });

      })

      .catch(this.catchError);

    this.loadPosts();

    openSocket('http://localhost:8080');

  }