Saxion university of applied sciences, Academy of creative technology

Web technology
Plants auction site – backend

Zuzana Botkova, 464686

Table of content

Assignment description

I built the auction site for plants. I designed and implemented the site using REST ASPI server using Express and all data are stored in MongoDB database. I used model files for auctions, bids, and users to make code more readable. All the requests are managed by controller classes, which do all the work. And routers route the requests which are exported to the server. Server then connects it all together. Requests testing is done by rest files. Explanation of the system follows.

Models

AuctionModel represents the auction. It requires auction data from the auctions collection from the database. Auction contains title, description, initial price, and end date.

BidModel represents the bid. It uses data from the bids collection from the database. Bid model contains name, bid price, created by (user) and time of creation.

UserModel represents the user and his/her data. It consists of username, email, password and repeatPassword which are both hashed by bcrypt and stored this way in database, and roles are also stored.

Controllers

The auction model from mongo database is required here in AuctionsController. All the methods are defined and exported to AuctionsRouter: getAllAuctions, getAuctionsByInitialPrice, getAuctionByName, addAuction, updateAuction and deleteAuction.

Bid model is required from the database to be used in BidController. It contains methods to manage the REST operations: getBids to get all the bids, getBidByName to get all bids for the same auction, addBid to create new bid, updateBid, and deleteBid. All these methods are then exported to the router, which can then manage the flow.

UserController reads the users data from the database. It uses bcrypt to hash the password when user registers or signs in, and jsonwebtoken when he/she signs in.

First password hashing comes to play when the user wants to register. The password entered by him/her is immediately hashed using 10 saltRounds. New user is then created with all the properties entered and is stored in the user's collection in database.

```
exports.register = function (req, res) {
    bcrypt.hash(req.body.password, 10, (err, hash) => {
        const newUser = new User({
            username: req.body.username,
            email: req.body.email,
            password: hash,
            repeatPassword: hash,
            roles: req.body.roles
        });
        newUser.save((err) => {
            if (err) {
                res.send('Username already in use.');
            } else {
                res.send('New user registered successfully.');
            }
        })
    });
};
```

Next use of bcrypt hashing happens when the user wants to sign in. The password entered by him/her is hashed and compared to the password store in database. If the password matches, token is generated and sent. If not, error message is sent.

```
exports.sign_in = function (req, res, next) {
    user.findOne({ username: req.body.username }, (err, user) => {
        if (err) {
            throw err;
        }
        if (!user || !user.comparePassword(req.body.password)) {
            return res.status(401).json({ message: 'Authentication failed. Incorrect username or password.'});
        }
        return res.json({ token: jwt.sign({ email: user.email, username: user.username, _id: user._id }, 'RESTFULAPIs) });
    });
};
```

Routers

AuctionsRouter handles requests regarding the auctions. auctionCotroller is created as the handler for the requests and routes are managed by this handler.

```
Module.exports = function (app) {
    const auctionHandler = require('../controllers/AuctionsController');
    app.route('/auctions')
        .get(auctionHandler.getAllAuctions);
    app.route('/auctions/:title')
        .get(auctionHandler.getAuctionByName);
    app.route('/auctions/:initialPrice')
        .get(auctionHandler.getAuctionsByInitialPrice);
    app.route('/auctions')
        .post(auctionHandler.addAuction);
    app.route('/auctions/:title')
        .patch(auctionHandler.updateAuction);
    app.route('/auctions/:title')
        .delete(auctionHandler.deleteAuction);
};
```

BidsRouter handles requests regarding the bid's requests throughout the bidsHandler. All the methods created in BidsController are passed to the router. The router then handles it.

```
Module.exports = function (app) {
    const bidsHandler = require('../controllers/BidController');
    app.route('/bids')
        .get(bidsHandler.getBids);
    app.route('/bids/addBid')
        .post(bidsHandler.addBid);
    app.route('/bids/:name')
        .patch(bidsHandler.updateBid);
    app.route('/bids/name')
        .delete(bidsHandler.deleteBid);
};
```

UsersRouter routes the users requests regarding the users. Methods created in UserController are passed to the router and handled.

```
module.exports = function (app) {
    const userHandler = require('../controllers/UserController');
    app.route('./auth/register')
        .post(userHandler.register);
    app.route('auth/sign_in')
        .post(userHandler.sign_in);
};
```

Server

User, Auction and Bid models are exported, as well as all the routers.

```
const User = require('./models/UserModel');
const Auction = require('./models/AuctionModel);
const Bid = require('./models/BidModel');

const userRoute = require('./routes/UserRouter');
userRoute(app);

const auctionRoute = require('./routes/AuctionsRouter');
auctionRoute(app);

const bidRoute = require('./routes/BidsRouter');
bidRoute(app);
```

Server then manages all the requests from rest files and forwards them to the right router which can then push it to the controller to perform the action required in the request.