```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('costmangerlocal:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

/**
 * Normalize a port into a number, string, or false.
 */

function normalizePort(val) {
    var port = parseInt(val, 10);

    if (isNaN(port)) {
        // named pipe
        return val;
    }

    if (port >= 0) {
        // port number
        return port;
    }

    return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
    if (error.syscall !== 'listen') {
        throw error;
    }
```

```javascript

    var bind = typeof port === 'string'
        ? 'Pipe ' + port
        : 'Port ' + port;

    // handle specific listen errors with friendly messages
    switch (error.code) {
        case 'EACCES':
            console.error(bind + ' requires elevated privileges');
            process.exit(1);
            break;
        case 'EADDRINUSE':
            console.error(bind + ' is already in use');
            process.exit(1);
            break;
        default:
            throw error;
    }
}

/**
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
    var addr = server.address();
    var bind = typeof addr === 'string'
        ? 'pipe ' + addr
        : 'port ' + addr.port;
    debug('Listening on ' + bind);
}

```

```javascript
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
const ejsMate = require('ejs-mate');
var methodOverride = require('method-override');


var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var categoriesRouter = require('./routes/categories');
var costsRouter = require('./routes/costs');
var reportRouter = require('./routes/report');

// mongoose setup
const connectionURI =
'mongodb+srv://Natan:Gershbein@costmanager.dshwg1z.mongodb.net/Project'

const mongoose = require('mongoose')

mongoose.connect(connectionURI);

const db = mongoose.connection;

db.on('error', function () {
    console.log("Error");
});

db.once('open', () => {
    console.log("DB Connected");
});
// mongoose.connect('mongodb://localhost/CostMangerLocal');


var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({extended: false}));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use(methodOverride('_method'));

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/categories', categoriesRouter);
app.use('/costs', costsRouter);
app.use('/report', reportRouter);


app.engine('ejs', ejsMate);


```

```javascript
// catch 404 and forward to error handler
app.use(function (req, res, next) {
    next(createError(404));
});

// error handler
app.use(function (err, req, res, next) {
    // set locals, only providing error in development
    res.locals.message = err.message;
    res.locals.error = req.app.get('env') === 'development' ? err : {};

    // render the error page
    res.status(err.status || 500);
    res.render('error');
});

module.exports = app;

```

```javascript
const mongoose = require('mongoose');

const categorySchema = new mongoose.Schema(
    {
        name: String,
        date_created: {type: String, default: new Date().toLocaleDateString()},
    });

const Category = mongoose.model('Category', categorySchema);

module.exports = Category;
```

```javascript
const mongoose = require('mongoose');

const costSchema = new mongoose.Schema(
    {
        sum: {type: Number, default: 0},
        description: String,
        date: {type: String, default: new Date().toISOString()},
        category: {type: String, default: 'General'},
        userId: {
            type: mongoose.Schema.Types.Number,
            ref: 'User', required: 'need to be assign to specif user'
        }
    });

const Cost = mongoose.model("Cost", costSchema);

module.exports = Cost;
```

```javascript
const mongoose = require('mongoose');

const reportSchema = new mongoose.Schema(
    {
        date_created: {type: String, default: new Date().toLocaleDateString()},
        userId: {type: mongoose.Schema.Types.Number, ref: 'User'},
        costs: [{type: mongoose.Schema.Types.ObjectId, ref: 'Cost'}],
        totalSum: {type: Number, default: 0},
        title: {type: String, default: 'title'},
    });

const Report = mongoose.model('Report', reportSchema);

module.exports = Report;
```

```javascript
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema(
    {
        first_name: String,
        last_name: String,
        birthday: {type: String, default: new Date().toLocaleDateString()},
        marital_status: String,
        _id: Number,
        password: {type: String},
        userName: {type: String},
        costs: [{type: mongoose.Schema.Types.ObjectId, ref: 'Cost'}],
    });


const User = mongoose.model('User', userSchema);

module.exports = User;
```

```javascript
class ErrorObject {
    constructor(msg = 'err', backLink, backButtonText = 'Try Again') {
        this.msg = msg;
        this.backLink = backLink;
        this.backButtonText = backButtonText;
    };
}

exports.ErrorObject = ErrorObject;

```

```javascript
var express = require('express');
var router = express.Router();
const User = require('../models/User');
const {ErrorObject} = require("./ErrorObject");

let utilsFunctions = {};

/**
 * function for Login verification
 */
utilsFunctions.LoginValidation = async function (userName, password) {
    try {
        const result = await User.findOne({'userName': userName, 'password':
password});
        return result;
    } catch (error) {
        console.log(`Some Error was occurred: ${error}`);
    }
}

// rout for enter page (sign in or sign up)
router.get('/', function (req, res) {
    res.render('index');
});

// route for credits section
router.get('/credits', function (req, res) {
    res.render('credits');
});

// rout for retrieve login page
router.get('/login', function (req, res) {
    res.render('users/SignIn.ejs');
});

/**
 * rout that handle the login post request
 */
router.post('/login', async function (req, res, next) {

    // checks if the given username and password are match to some user in DB
    const LoginValidationResults = await utilsFunctions.
    LoginValidation(req.body.userName, req.body.password);

    // if its not null  -> user with given details was found
    if (LoginValidationResults !== null) {

        res.redirect(`/users/${LoginValidationResults._id.toString()}`);

    } // else -> show  error page with err info
    else {

        const errorObject = new ErrorObject
        ('User Not Exist Or you have enter wrong credentials',
        req.url);

        res.render('Errors/errorPage', {errorObject});
    }
});
```

```javascript

// rout for Sign-Up of new user page
router.get('/signUp', async function (req, res, next) {

    try {
        // retrieving the next id for new user
        const users = await User.find({}).sort({_id: -1});

        // In there are no users (first user)
        if (users.length === 0) {
            res.render('users/newUser.ejs', {'id': 0});
        }
        res.render('users/newUser.ejs', {'id': ++users[0]._id});
    } catch (error) {
        console.log(`Error: ${error}`);
        res.send(error);
    }

});

// rout for handling post request of Signing-Up new user page
// and saving in the DB
router.post('/signUp', async function (req, res) {

    try {
        const fittingUserToUserName = await User
        .findOne({'userName': req.body.userName});

        // user with given email(username) is not exist -> so create new user
        if (fittingUserToUserName === null) {
            let newUser = new User(req.body);
            let userBirthday = req.body.birthday;

            newUser.birthday = userBirthday === '' ?
                new Date().toISOString().split('T')[0] :
                new Date(userBirthday).toISOString().split('T')[0];

            newUser.save().then((createdUser) => {
                console.log(`newUser created: ${createdUser}`);
                res.redirect(`/users/${req.body._id}/`);

            }).catch((error) => {
                res.send(error);
                console.log(`Error: ${error}`);
            });
        } else // in case user with given username already exist
        {
            let errorObject = new ErrorObject
            (`User with the username: ${req.body.userName} already exist`, req.url);
            res.render('Errors/errorPage', {errorObject});
        }
    } catch (err) {
        res.send(err + " err");
    }
});

router.get('/signout', async function (req, res) {

    res.redirect('/');

});

module.exports = router;
```

```
const express = require('express');
const User = require('../models/User');
const Category = require('../models/Category');
const Cost = require('../models/Cost');

const router = express.Router();

// rout for main page of current user
router.get('/:id', async function (req, res, next) {
    try {
        const user = await User.findById(req.params.id);
        console.log(user);

        res.render('home', {'user': user});

    } catch (error) {
        res.send(`Error: ${error}`);
    }

});

// rout for  page that show a form to creation of new cost
router.get('/:id/costs/new', async function (req, res, next) {
    try {

        const categories = await Category.find({});
        res.render('costs/newCost.ejs',
            {'categories': categories, 'id': req.params.id, backLink: req.url});

    } catch (error) {

        res.send(`Error: ${error}`);
    }

});

module.exports = router;
```

```javascript
const express = require('express');
const Category = require('../models/Category');
const User = require('../models/User');
const Cost = require('../models/Cost');

const router = express.Router();


// rout for get all costs of current user
router.get('/:userId/', async function (req, res, next) {
    try {

        const costs = await Cost.find({userId: req.params.userId});
        console.log(costs);
        res.render('costs/allCosts', {'costs': costs, 'id': req.params.userId});
    } catch (error) {
        console.log(error);
        res.send(`Error: ${error}`);
    }
});


// rout for get all cost of current usr sorted by user choice( price or category)
router.get('/:userId/sort', async function (req, res) {

    let costs = [];

    try {

        if (req.query.sortBy === 'category') {
            costs = await Cost.find({userId: req.params.userId})
            .sort({category: 'asc'});
        }

        if (req.query.sortBy === 'price') {
            costs = await Cost.find({userId: req.params.userId})
            .sort({sum: 'asc'});
        }

        res.render('costs/allCosts', {'costs': costs, 'id': req.params.userId});
    } catch (error) {
        console.log(error);
        res.send(`Error: ${error}`);
    }
});

// rout for handling  cost creation request
router.post('/:userId/', async function (req, res) {

    const newCost = new Cost(req.body);

    newCost['date'] = req.body.date === "" ?
        // if the date is empty, fill today date
        newCost['date'] = new Date().toISOString().split('T')[0] :
        newCost['date'] = new Date(req.body.date).toISOString().split('T')[0];

    try {
        const user = await User.findById(req.params.userId);

        user.costs.push(newCost); // insert new cost to costs list of current user..
        newCost.userId = req.params.userId;

        await user.save();

        // saving new cost
        newCost.save().then(newCost => {
            console.log('new cost was created: ' + newCost);
            res.redirect(`/costs/${req.params.userId}`);
        }).catch((error) => {
            console.log(error);
            res.send(`Error: ${error}`);
        });
    } catch (err) {
        res.send(`Error: ${err}`);
    }
});

// rout for page that display the specific cost related to current login user.
router.get('/:userId/:costId', async function (req, res) {

    try {
        const targetCostToDisplay = await Cost.findById(req.params.costId);

        // if the targetCostToDisplay is empty
        //-> it means thar cost with given id is not exist
        if (targetCostToDisplay === null) {

            res.send("Cant find the requested cost");

        }

        res.render('costs/showCost', {
            'cost': targetCostToDisplay,
            'backLink': `/costs/${req.params.userId}` , 'id': req.params.userId
        });
    } catch (err) {
        res.send("Error accrued:" + err);
    }
});

// rout for get an edit page
router.get("/:userId/:costId/edit", async function (req, res) {

    try {
        const targetCostToUpdated = await Cost.findById(req.params.costId);
        const categories = await Category.find({});

        res.render("costs/editCost", {
            id: req.params.userId,
            cost: targetCostToUpdated,
            categories: categories,
            backLink: `/costs/${req.params.userId}/${req.params.costId}`
        });
    } catch (error) {
        res.send(error);
    }
});

// rout that handel the edit request (put request)
```

```javascript
router.put('/:userId/:costId', async function (req, res) {
    try {
        const updatedCost = req.body;
        const targetCostToUpdated = await Cost
        .findByIdAndUpdate(req.params.costId, updatedCost);

        if (targetCostToUpdated == null) {
            res.send("error did not find cost with given id");
        } else {
            res.redirect(`/costs/${req.params.userId.toString()}
            /${req.params.costId.toString()}`);
        }
    } catch (err) {
        res.send("error: " + err);
    }
});


// Deleting cost
router.delete('/:userId/:costId', async function (req, res) {

    try {
        const costToDelete = await Cost.findOneAndDelete({_id: req.params.costId});
        // successfully deleted
        //-> the return value is null if there is no cost with given id
        if (costToDelete !== null) {
            console.log("Cost to delete: " + costToDelete);

            Cost.find({userId: req.params.userId}, function (err, fittingCosts) {
                if (err) {
                    res.send("Error: " + err);
                } else {
                    User.findByIdAndUpdate(req.params.userId,
                         {"costs": fittingCosts}, function (err, result) {
                        if (err) {
                            res.send(err);
                        } else {
                            console.log("Updated user costs: " + fittingCosts);
                            res.redirect(`/costs/${req.params.userId}`);
                        }
                    });
                }
            });
        }
    } catch (err) {
        res.send("Error: " + err);
    }
});

module.exports = router;
```

```javascript
const express = require('express');
const Cost = require('../models/Cost');
const Report = require('../models/Report');
const router = express.Router();

let reportUtilsFunctions = {};

// function that creates title for report
reportUtilsFunctions.createReportTitle = function (reportDate) {

    // splitting date in  wed, 14 Jun 2022 07:00:00 GMT form to have only Jun, 2022
    // via Regex
    const dateInArrFormat = reportDate.split(/\W+/gm).slice(2, 4);
    return `Report for: ${dateInArrFormat[0]}, ${dateInArrFormat[1]}`;
};

// function for find fitting cost to asked report by date
reportUtilsFunctions.findFittingCost = function (costs, month, year) {
    return costs.filter(cost =>
        cost.date.split("-")[1] === month && cost.date.split("-")[0] === year
    );
};

// function which calc total sum of cost
reportUtilsFunctions.calcTotalSumOfReport = function (costsArray) {
    let sum = 0;
    costsArray.forEach((cost) => sum += cost.sum);
    console.log(sum);
    return sum;
};

// rout which shows all  report that related to current user
router.get('/:userId/allReports', async function (req, res) {
    try {
        const allReports = await Report.find({ userId: req.params.userId })
        .populate('costs');
        res.render('report/allReports',
        { 'reports': allReports, "userId": req.params.userId });
    } catch (err) {
        res.send("Error" + err);
    }
});

// rout that give a form for choosing month & year  of report
router.get('/:userId/getReport', function (req, res) {
    res.render('report/getReport', {userId: req.params.userId});
});

router.get('/:userId/newReport', async function (req, res) {

    const fullDate = req.query.date;
    const reportDate = new Date(fullDate).toUTCString();
    const costs = await Cost.find({userId: req.params.userId});
    const yearAndMonth = fullDate.split("-");
    const year = yearAndMonth[0];
    const month = yearAndMonth[1];

    const reportTitle = reportUtilsFunctions.createReportTitle(reportDate);

    // user costs that fitting to desire month & year
    const costsArray = reportUtilsFunctions.findFittingCost(costs, month, year);
    let sum = -1;
    try {
        // find all the reports of the current user
        // -> each user can have maximum one report for
        // each month & year
        const reports = await Report.find
        ({userId: req.params.userId, date_created: fullDate});
        // case : creating new report
        if (reports.length === 0) {

            const totalSum = reportUtilsFunctions.calcTotalSumOfReport(costsArray);
            sum = totalSum;

            const generateNewReport = new Report({
                userId: req.params.userId,
                title: reportTitle,
                costs: costsArray,
                totalSum: totalSum,
                date_created: new Date(fullDate).toISOString()
                .split('-').slice(0, 2).join('-'),
            });

            generateNewReport.save().then((results)=>{
                console.log('new report was created '  + results);
            }).catch((err)=> {
                console.log('err' + err);
            });
            // await generateNewReport.save();

        }
        // checking existing report
        else {
            // case: report need get update (if cost was deleted or added)
            if (reports[0].costs.length !== costsArray.length) {
                console.log("updating existing report");

                const totalSum =
                reportUtilsFunctions.calcTotalSumOfReport(costsArray);
                sum = totalSum;

                await Report.findByIdAndUpdate(reports[0]._id,
                    {totalSum: totalSum, costs: costsArray});
            } else { // case: the report is up to date
                console.log("giving previous report");
            }
        }
        res.render('report/newReport',
            {
                'reportTitle': reportTitle,
                'costs': costsArray,
                'totalSum': sum === -1 ?
                reports[0].totalSum : sum,
                'userId': req.params.userId
            });

    } catch (err) {
        console.log(err + "error");
    }
```

```
});

module.exports = router;
```

```javascript
const express = require('express');
const Category = require('../models/Category');
const Cost = require('../models/Cost');

const router = express.Router();

/* get  all categories. */
router.get('/all', async function (req, res) {
    const allCategories = await Category.find({});
    res.render('Categories/allCategories.ejs', {"categories": allCategories});
});

/* rout for page which display form for creation of new category */
router.get('/new', function (req, res) {
    res.render('Categories/newCategory.ejs');
});

/* rout that handle with creation request of new category */
router.post('/new', async function (req, res) {

    const newCategory = new Category(req.body);

    await newCategory.save()
        .then((newCategory) => {
            console.log(`created: ${newCategory}`);
            res.redirect('/categories/all')
        }).catch((error) => {
            console.log(error);
        });
});

// rout that handel with deletion of category
router.delete('/:categoryId', async function (req, res) {

    const categoryToDelete = await Category.findById(req.params.categoryId);
    const CostWithTargetCategory = await Cost.find({category: categoryToDelete.name});

    // for each cost update his category to 'general' because his original category would deleted
    for (let cost of CostWithTargetCategory) {
        await Cost.findByIdAndUpdate(cost._id, {'category': 'General'})
            .then((results) => console.log(results))
            .catch((err) => console.log((err)));
    }

    Category.findByIdAndDelete(categoryToDelete._id)
    .then(() => {
        console.log("Category deleted");
        res.redirect('/categories/all');
    }).catch((err) => res.send(err));
});

module.exports = router;
```