*/\* eslint-disable no-undef \*/*

'use strict'

*// var bodyParser = require('body-parser');*

const express = require('express')

const fs = require('fs')

const path = require('path')

const server = express()

const jsonParser = express.json()

const fullPath = path.join(\_\_dirname, '/public')

server.use(express.json({ limit: '50mb' }));

server.use(express.urlencoded({ limit: '50mb', extended: true }));

*// const urlencodedParser = bodyParser.urlencoded({ extended: true })*

*// server.use(urlencodedParser)*

*// server.use(bodyParser.json())*

*// server.use(express.json({ limit: '5mb' }));*

*// server.use(express.urlencoded({ limit: '5mb' }));*

server.use(express.static(fullPath)) *// ))\_\_dirname + '/public'))*

const compression = require('compression')

*// const db = require('./db/index\_sqlite');*

const db = require('./db/index');

*// db.sequelize*

*//   .sync({ force: false })*

*//   .then(() => console.log('Ура! Связь с БД установлена.... '))*

db.sequelize.sync()

db.sequelize.authenticate().then(() => {

    console.log('Connection established successfully.');

}).catch(*err* => {

    console.error('Unable to connect to the database:', *err*);

}).finally(() => {

*// db.sequelize.close();*

});

server.use(compression())

const allowCrossDomain = function(*req*, *res*, *next*) {

*res*.header('Access-Control-Allow-Origin', '\*')

*res*.header('Access-Control-Allow-Headers', '\*')

    next()

}

server.use(allowCrossDomain)

*// tutDB - прописывается доступ к настройкам схемы tutorials в MySQL. Необходимо для работы с регламентами - старая версия.*

const tutDB = require('./tutorials/indextutorials');

*//Синхронизация и аутентификация к БД*

tutDB.sequelize.sync({ alter: true })

tutDB.sequelize.authenticate().then(() => {

    console.log('Connection with TUTORIAL database established successfully.');

}).catch(*err* => {

    console.error('Unable to connect to the TUTORIAL database:', *err*);

}).finally(() => {

*// tutDB.sequelize.close();*

});

*// springDB - прописывается доступ к настройкам схемы spring\_tutorials в MySQL. Необходимо для работы с регламентами - новая версия.*

const springDB = require('./spring\_tutorials/indextutorials');

*//Синхронизация и аутентификация к БД*

springDB.sequelize.sync({ alter: true })

springDB.sequelize.authenticate().then(() => {

    console.log('Connection with SPRING\_TUTORIAL database established successfully.');

}).catch(*err* => {

    console.error('Unable to connect to the SPRING\_TUTORIAL database:', *err*);

}).finally(() => {

*// tutDB.sequelize.close();*

});

function mySortObjOccsByNum(*a*, *b*) { *//функция сотрировки массива объектов.*

    var a1 = *a*.num;

    a1 = a1.split(".");

    var b1 = *b*.num;

    b1 = b1.split(".");

    for (var i = 0; i < a1.length; i++) {

        if (b1.length - 1 < i) {

            return 1;

        } *//b1 = 1.1. a1 =  1.1.1.*

        if (a1[i] === b1[i]) {

            continue;

        }

        return a1[i] - b1[i]; *//сортировка по возрастанию*

*//  return b1[i] - a1[i];  //сортировка по убыванию*

    }

    return 0;

}

server.get('/api/getEmployees', async function(*req*, *res*) {

    console.time('GetEmployees')

*//  await db.WORKMAN.sync({ force: false });*

    const users = await db.WORKMAN.findAll();

*res*.send(users)

*// db.Workman.authenticate().then(async() => {*

*/\* await db.WORKMAN.findAll().then((note) => {*

*if (!note) {*

*res.send("Ошибка соединения с БД ");*

*} else {*

*res.send(note)*

*}*

*}).finally(() => {})*

*/\* }).catch(err => {*

*console.error('Unable to connect to the database:', err);*

*}).finally(() => {*

*// db.sequelize.close();*

*});*

*\*/*

    console.timeEnd('GetEmployees')

});

server.get('/api/testAlert', async function(*req*, *res*) {

*res*.send("AHTUNG!")

})

server.get('/api/uploadfilesfromclient', async function(*req*, *res*) {

*res*.send("GET WORKS!")

})

server.post('/api/uploadfilesfromclient', async function(*req*, *res*) {

    console.time('uploadfilesfromclient')

*//const updateObj = JSON.parse(req.body)*

    let filecontent = *req*.body.content;

    let filename = *req*.body.name;

    const fileattr = *req*.body.lastModified;

    const filesize = *req*.body.size;

    let reportStr = '';

    filename = "c:\\aris\\sharefld\\" + filename;

    filecontent = Buffer.from(filecontent.slice(filecontent.indexOf(';base64,') + ';base64,'.length), 'base64');

    if (filecontent.length == filesize) {

        reportStr += 'Файл ' + *req*.body.name + ' принят успешно\n';

        console.log('Файл принят успешно.');

    }

    fs.writeFile(filename, filecontent, 'utf8', (*err*) => {

            if (*err*) {

                reportStr += 'Ошибка сохранения файла!!'

                throw *err*

            } else {

                reportStr += 'Файл ' + *req*.body.name + 'сохранен успешно'

                console.log('The file has been saved!');

            }

        })

*// console.log(errcode)*

*res*.send(reportStr)

    console.timeEnd('uploadfilesfromclient') *// время выполнения запроса*

*//fs.writeFileSync(filename, Buffer.from(filecontent.slice(filecontent.indexOf(';base64,') + ';base64,'.length), 'base64'), 'utf8')*

})

server.get('/api/getbpandppbyguids:id', async function(*req*, *res*) {

    console.time('GetBPForReglamentBuildByGUIDs')

    const numbp = *req*.params.id.slice(1).split("&")[0]

    const numpp = *req*.params.id.slice(1).split("&")[1]

    console.log(numpp);

    let bpObj = {};

    tutDB.sequelize.sync({ force: false }).then(async() => {

        await tutDB.tutorialBP.findOne({ where: { guid: numbp } }).then(async(*note*) => {

            if (!*note*) {

*res*.send("Запрашиваемый БП не найден.");

            }

            if (*note* == null) *res*.send("Somthing wrong :(");

            else {

                bpObj = *note*;

                let ppObj = await tutDB.tutorialPP.findOne({ where: { guid: numpp } })

                ppObj = ppObj.toJSON();

                let totalArrayList = ppObj.operations.split(",");

                let totaloperObjArray = [];

                for (var i = 0; i < totalArrayList.length; i++) {

                    let object = await tutDB.tutorialOperations.findOne({ where: { guid: totalArrayList[i] } })

                    if (object != null) {

                        totaloperObjArray.push(object);

*// console.log(object);*

                    }

                }

*// let totaloperObjArray = await tutDB.tutorialOperations.findAll({*

*//     where: {*

*//         guid: { in: totalArrayList }*

*//     }*

*// }, {*

*//     order: { num: "ASC" }*

*// })*

                if (totaloperObjArray.length == 0) { *res*.send({ message: "неправильно указан номер ПП" }); }

                totaloperObjArray = totaloperObjArray.sort(mySortObjOccsByNum)

                for (let p = 0; p < totaloperObjArray.length; p++) {

                    let oper = totaloperObjArray[p]

                    if (oper.isp != "") {

                        const oper\_isp = oper.isp.split(",")

                        let tt = await tutDB.tutorialIsp.findAll({ where: { guid: oper\_isp } }, {

                            order: { num: "ASC" }

                        })

                        oper.ispObj = tt;

                    }

                    if (oper.indocs != "") {

                        let oper\_docs = oper.indocs.split(",")

                        let tt = await tutDB.tutorialDocs.findAll({ where: { guid: oper\_docs } }, {

                            order: { num: "ASC" }

                        })

                        for (let dd = 0; dd < tt.length; dd++) {

                            let tempattr = tt[dd].attrs.split(",")

                            let tempconnector = tt[dd].connector.split(",")

                            tt[dd].attrObj = await tutDB.tutorialAttrDocs.findAll({ where: { guid: tempattr } }, { order: { num: "ASC" } })

                            tt[dd].connectorObj = await tutDB.tutorialConnector.findAll({ where: { guid: tempconnector } }, { order: { num: "ASC" } })

                        }

                        oper.indocsObj = tt;

                    } else {

                        oper.indocsObj = {};

                    }

                    if (oper.outdocs != "") {

                        let oper\_docs = oper.outdocs.split(",")

                        let tt = await tutDB.tutorialDocs.findAll({ where: { guid: oper\_docs } }, {

                            order: { num: "ASC" }

                        })

                        for (let dd = 0; dd < tt.length; dd++) {

                            let tempattr = tt[dd].attrs.split(",")

                            let tempconnector = tt[dd].connector.split(",")

                            tt[dd].attrObj = await tutDB.tutorialAttrDocs.findAll({ where: { guid: tempattr } }, { order: { num: "ASC" } })

                            tt[dd].connectorObj = await tutDB.tutorialConnector.findAll({ where: { guid: tempconnector } }, { order: { num: "ASC" } })

                        }

                        oper.outdocsObj = tt;

                    } else {

                        oper.outdocsObj = {};

                    }

                }

                bpObj.pp = ppObj;

                bpObj.pp.operations = totaloperObjArray.map((*oper*) => {

                    return {

                        guid: *oper*.guid,

                        name: *oper*.name,

                        isp: *oper*.ispObj,

                        indocs: *oper*.indocsObj,

                        ourdocs: *oper*.outdocsObj,

                        startEvent: *oper*.startEvent,

                        description: *oper*.description,

                        endEvent: *oper*.endEvent,

                        num: *oper*.num,

                        guidpp: *oper*.guidpp

                    }

                });

*res*.send(bpObj);

            }

        })

    }).catch(*err* => {

        console.error('при обращении за регламентом. Unable to connect to the database:', *err*);

    }).finally(() => {

*// tutDB.sequelize.close();*

    });

    console.timeEnd('GetBPForReglamentBuildByGUIDs')

});

server.get('/api/spring\_getbpandppbyguids:id', async function(*req*, *res*) {

    console.time('GetBPForReglamentBuildByGUIDs')

    const numbp = *req*.params.id.slice(1).split("&")[0]

    const numpp = *req*.params.id.slice(1).split("&")[1]

    console.log(numpp);

    let bpObj = {};

    springDB.sequelize.sync({ force: false }).then(async() => {

        await springDB.tutorialBP.findOne({ where: { guid: numbp } }).then(async(*note*) => {

            if (!*note*) {

*res*.send("Запрашиваемый БП не найден.");

            }

            if (*note* == null) *res*.send("Somthing wrong :(");

            else {

                bpObj = *note*;

                let ppObj = await springDB.tutorialPP.findOne({ where: { guid: numpp } })

                ppObj = ppObj.toJSON();

                let totalArrayList = ppObj.operations.split(",");

                let totaloperObjArray = [];

                for (var i = 0; i < totalArrayList.length; i++) {

                    let object = await springDB.tutorialOperations.findOne({ where: { guid: totalArrayList[i] } })

                    if (object != null) {

                        totaloperObjArray.push(object);

*// console.log(object);*

                    }

                }

*// let totaloperObjArray = await tutDB.tutorialOperations.findAll({*

*//     where: {*

*//         guid: { in: totalArrayList }*

*//     }*

*// }, {*

*//     order: { num: "ASC" }*

*// })*

                if (totaloperObjArray.length == 0) { *res*.send({ message: "неправильно указан номер ПП" }); }

                totaloperObjArray = totaloperObjArray.sort(mySortObjOccsByNum)

                for (let p = 0; p < totaloperObjArray.length; p++) {

                    let oper = totaloperObjArray[p]

                    if (oper.isp != "") {

                        const oper\_isp = oper.isp.split(",")

                        let tt = await springDB.tutorialIsp.findAll({ where: { guid: oper\_isp } }, {

                            order: { num: "ASC" }

                        })

                        oper.ispObj = tt;

                    }

                    if (oper.indocs != "") {

                        let oper\_docs = oper.indocs.split(",")

                        let tt = await springDB.tutorialDocs.findAll({ where: { guid: oper\_docs } }, {

                            order: { num: "ASC" }

                        })

                        for (let dd = 0; dd < tt.length; dd++) {

                            let tempattr = tt[dd].attrs.split(",")

                            let tempconnector = tt[dd].connector.split(",")

                            tt[dd].attrObj = await springDB.tutorialAttrDocs.findAll({ where: { guid: tempattr } }, { order: { num: "ASC" } })

                            tt[dd].connectorObj = await springDB.tutorialConnector.findAll({ where: { guid: tempconnector } }, { order: { num: "ASC" } })

                        }

                        oper.indocsObj = tt;

                    } else {

                        oper.indocsObj = {};

                    }

                    if (oper.outdocs != "") {

                        let oper\_docs = oper.outdocs.split(",")

                        let tt = await springDB.tutorialDocs.findAll({ where: { guid: oper\_docs } }, {

                            order: { num: "ASC" }

                        })

                        for (let dd = 0; dd < tt.length; dd++) {

                            let tempattr = tt[dd].attrs.split(",")

                            let tempconnector = tt[dd].connector.split(",")

                            tt[dd].attrObj = await springDB.tutorialAttrDocs.findAll({ where: { guid: tempattr } }, { order: { num: "ASC" } })

                            tt[dd].connectorObj = await springDB.tutorialConnector.findAll({ where: { guid: tempconnector } }, { order: { num: "ASC" } })

                        }

                        oper.outdocsObj = tt;

                    } else {

                        oper.outdocsObj = {};

                    }

                }

                bpObj.pp = ppObj;

                bpObj.pp.operations = totaloperObjArray.map((*oper*) => {

                    return {

                        guid: *oper*.guid,

                        name: *oper*.name,

                        isp: *oper*.ispObj,

                        indocs: *oper*.indocsObj,

                        ourdocs: *oper*.outdocsObj,

                        startEvent: *oper*.startEvent,

                        description: *oper*.description,

                        endEvent: *oper*.endEvent,

                        num: *oper*.num,

                        guidpp: *oper*.guidpp

                    }

                });

*res*.send(bpObj);

            }

        })

    }).catch(*err* => {

        console.error('при обращении за регламентом. Unable to connect to the database:', *err*);

    }).finally(() => {

*// tutDB.sequelize.close();*

    });

    console.timeEnd('GetBPForReglamentBuildByGUIDs')

});

server.get('/api/getAllBpForReglament', async function(*req*, *res*) { *//Получаем список БП утвержденных регламентов*

*//const numbp = req.params.id.slice(1);*

    let bpObj = {};

    tutDB.sequelize.sync({ force: false }).then(async() => {

        await tutDB.tutorialBP.findAll().then(async(*note*) => {

            if (!*note*) {

*res*.send("Запрашиваемый БП не найден.");

            } else {

                bpObj = *note*;

            }

        });

*res*.send(bpObj);

    }).catch(*err* => {

        console.error('при обращении за регламентом. Unable to connect to the database:', *err*);

    }).finally(() => {

*// tutDB.sequelize.close();*

    });

});

server.get('/api/getAllPpForReglament', async function(*req*, *res*) { *//Получаем список БП утвержденных регламентов*

*//const numbp = req.params.id.slice(1);*

    let bpObj = {};

    tutDB.sequelize.sync({ force: false }).then(async() => {

        await tutDB.tutorialPP.findAll().then(async(*note*) => {

            if (!*note*) {

*res*.send("Запрашиваемый БП не найден.");

            } else {

                bpObj = *note*;

            }

        });

*res*.send(bpObj);

    }).catch(*err* => {

        console.error('при обращении за регламентом. Unable to connect to the database:', *err*);

    }).finally(() => {

*// tutDB.sequelize.close();*

    });

});

server.get('/api/getbpandppbynums:id', async function(*req*, *res*) {

    console.time('GetBPForReglamentBuild2')

    const numbp = *req*.params.id.slice(1).split("&")[0]

    const numpp = *req*.params.id.slice(1).split("&")[1]

    let bpObj = {};

    const { Op } = require("sequelize");

    tutDB.sequelize.sync({ force: false }).then(async() => {

        await tutDB.tutorialBP.findOne({ where: { num: numbp } }).then(async(*note*) => {

            if (!*note*) {

*res*.send("Запрашиваемый БП не найден.");

            } else {

                bpObj = *note*.toJSON();

                let ppObj = await tutDB.tutorialPP.findOne({

                        where: {

                            [Op.and]: [

                                { num: numpp }, { guidbp: bpObj.guid }

                            ]

                        }

                    })

*//   let ppObj = await tutDB.tutorialPP.findAll({ where: { num: numpp, guidbp: bpObj.guid } })*

                if (ppObj == null) { *res*.send({ message: "неправильно указан номер ПП" }); }

                ppObj = ppObj.toJSON();

                let totalArrayList = ppObj.operations.split(",");

                let totaloperObjArray = await tutDB.tutorialOperations.findAll({

                    where: {

                        guid: { in: totalArrayList }

                    }

*// where: {*

*//     [Op.or]: [{ guidpp: ppObj.guid },*

*//         { guidpp: ppObj.guidmodel }*

*//     ]*

*// }*

                }, {

                    order: { num: "ASC" }

                })

                if (totaloperObjArray.length == 0) {

*res*.send({ message: "НЕ Нашлись операции по ПП. Возможно неправильно указан номер ПП" });

                }

*// totaloperObjArray = totaloperObjArray.toJSON();*

                totaloperObjArray = totaloperObjArray.sort(mySortObjOccsByNum)

                for (let p = 0; p < totaloperObjArray.length; p++) {

                    let oper = totaloperObjArray[p]

                    if (oper.isp != "") {

                        const oper\_isp = oper.isp.split(",")

                        let tt = await tutDB.tutorialIsp.findAll({ where: { guid: oper\_isp } }, {

                            order: { num: "ASC" }

                        })

                        oper.ispObj = tt;

                    }

                    if (oper.indocs != "") {

                        let oper\_docs = oper.indocs.split(",")

                        let tt = await tutDB.tutorialDocs.findAll({ where: { guid: oper\_docs } }, {

                            order: { num: "ASC" }

                        })

                        for (let dd = 0; dd < tt.length; dd++) {

                            let tempattr = tt[dd].attrs.split(",")

                            let tempconnector = tt[dd].connector.split(",")

                            tt[dd].attrObj = await tutDB.tutorialAttrDocs.findAll({ where: { guid: tempattr } }, { order: { num: "ASC" } })

                            tt[dd].connectorObj = await tutDB.tutorialConnector.findAll({ where: { guid: tempconnector } }, { order: { num: "ASC" } })

                        }

                        oper.indocsObj = tt;

                    } else {

                        oper.indocsObj = {};

                    }

                    if (oper.outdocs != "") {

                        let oper\_docs = oper.outdocs.split(",")

                        let tt = await tutDB.tutorialDocs.findAll({ where: { guid: oper\_docs } }, {

                            order: { num: "ASC" }

                        })

                        for (let dd = 0; dd < tt.length; dd++) {

                            let tempattr = tt[dd].attrs.split(",")

                            let tempconnector = tt[dd].connector.split(",")

                            tt[dd].attrObj = await tutDB.tutorialAttrDocs.findAll({ where: { guid: tempattr } }, { order: { num: "ASC" } })

                            tt[dd].connectorObj = await tutDB.tutorialConnector.findAll({ where: { guid: tempconnector } }, { order: { num: "ASC" } })

                        }

                        oper.outdocsObj = tt;

                    } else {

                        oper.outdocsObj = {};

                    }

                }

                bpObj.pp = ppObj;

                bpObj.pp.operations = totaloperObjArray.map((*oper*) => {

                    return {

                        guid: oper.guid,

                        name: oper.name,

                        isp: oper.ispObj,

                        indocs: oper.indocsObj,

                        ourdocs: oper.outdocsObj,

                        startEvent: oper.startEvent,

                        endEvent: oper.endEvent,

                        num: oper.num,

                        guidpp: oper.guidpp

                    }

                });

                res.send(bpObj);

            }

        })

    }).catch(*err* => {

        console.error('при обращении за регламентом. Unable to connect to the database:', err);

    }).finally(() => {

*// tutDB.sequelize.close();*

    });

    console.timeEnd('GetBPForReglamentBuild2')

});

server.post('/api/aris', async function(*req*, *res*) { *//эта апишка загружает БП из ариса для регламента*

    console.time('ApiArisStart');

    const utf8 = require('utf8');

    const bpObj = req.body;

    const bpObjOnly = {

        name: bpObj.name,

        guid: bpObj.guid,

        num: bpObj.num,

    };

    const ppArray = JSON.parse(bpObj.pp);

    bpObjOnly.pp = ppArray.map((*g*) => { return g.guid }).join(",")

    let ppObj = {};

    let ppObjArray = [];

    let operationsArray = [];

    let ppOperations = [];

    let indocsArray = [];

    let totalIndocsArray = [];

    let outdocsArray = [];

    let totalOutdocsArray = [];

    let ispArray = [];

    let totalExecutors = [];

    let totalAttrsArray = [];

    let totalConnectorsArray = [];

    for (let pp = 0; pp < ppArray.length; pp++) {

        ppOperations = JSON.parse(ppArray[pp].operations);

        ppObj = {

            name: ppArray[pp].name,

            num: ppArray[pp].num,

            guid: ppArray[pp].guid,

            guidbp: bpObj.guid,

            guidmodel: ppArray[pp].guidmodel,

            operations: ppOperations.map(function(*oper*) { return oper.guid }).join(",")

        };

        ppObjArray.push(ppObj)

        for (let o = 0; o < ppOperations.length; o++) {

            indocsArray = JSON.parse(ppOperations[o].indocs);

            outdocsArray = JSON.parse(ppOperations[o].outdocs);

            outdocsArray.forEach(*doc* => {

                totalOutdocsArray.push({

                    name: doc.name,

                    guid: doc.guid,

                    guidoper: ppOperations[o].guid

                });

                totalAttrsArray = totalAttrsArray.concat(JSON.parse(doc.attrs).map(function(*a*) {

                    return {

                        name: a.name,

                        guid: a.guid,

                        guiddoc: doc.guid

                    }

                }));

                totalConnectorsArray = totalConnectorsArray.concat(JSON.parse(doc.connector).map(function(*a*) {

                    return {

                        name: a.name,

                        guid: a.guid,

                        guiddoc: doc.guid

                    }

                }));

            });

            indocsArray.forEach(*doc* => {

                totalIndocsArray.push({

                    name: doc.name,

                    guid: doc.guid,

                    guidoper: ppOperations[o].guid

                });

                totalAttrsArray = totalAttrsArray.concat(JSON.parse(doc.attrs).map(function(*a*) {

                    return {

                        name: a.name,

                        guid: a.guid,

                        guiddoc: doc.guid

                    }

                }));

                totalConnectorsArray = totalConnectorsArray.concat(JSON.parse(doc.connector).map(function(*a*) {

                    return {

                        name: a.name,

                        guid: a.guid,

                        guiddoc: doc.guid

                    }

                }));

            });

            ispArray = JSON.parse(ppOperations[o].isp);

            ispArray.forEach(*isp* => {

                totalExecutors.push({

                    name: isp.name,

                    guid: isp.guid,

                    guidoper: ppOperations[o].guid

                })

            });

            operationsArray.push({

                name: ppOperations[o].name,

                num: ppOperations[o].num,

                guid: ppOperations[o].guid,

                guidpp: ppArray[pp].guid,

                startEvent: ppOperations[o].startEvent,

                endEvent: ppOperations[o].endEvent,

                description: ppOperations[o].description,

                indocs: indocsArray.map(function(*doc*) { return doc.guid }).join(","),

                outdocs: outdocsArray.map(function(*doc*) { return doc.guid }).join(","),

                isp: ispArray.map(function(*doc*) { return doc.guid }).join(",")

            })

        }

    } *// for (let pp = 0; pp < ppArray.length; pp++)*

    async function updateOrCreate(*model*, *where*, *newItem*) {

*// First try to find the record*

        const foundItem = await *model*.findOne({ where });

        if (!foundItem) {

*// Item not found, create a new one*

            const item = await *model*.create(*newItem*)

            return { item, created: true };

        }

*// Found an item, update it*

        const item = await *model*.update(*newItem*, { where });

        return { item, created: false };

    }

    await tutDB.sequelize.authenticate()

    tutDB.sequelize.sync({ force: false }).then(async() => {

        updateOrCreate(tutDB.tutorialBP, { guid: bpObjOnly.guid }, bpObjOnly)

*// const bpSTR = await tutDB.tutorialBP.create(bpObjOnly)*

*// await bpSTR.save()*

        await tutDB.tutorialPP.bulkCreate(ppObjArray, { individualHooks: true, updateOnDuplicate: ["guidbp", "name", "num", "operations"] }).then(() => {

            console.log('подпроцессы в БД записаны');

        }).catch((*err*) => {

            console.log('не удалось записать подпроцессы');

            console.log(*err*);

        }).finally(() => {

*// tutDB.sequelize.close();*

        });

*// запись всех операций*

        await tutDB.tutorialOperations.bulkCreate(operationsArray, { individualHooks: true, updateOnDuplicate: ["guidpp", "name", "num", "startEvent", "endEvent", "indocs", "outdocs", "isp", "description", "updatedAt"] }).then(() => {

            console.log('операции  в БД записаны');

        }).catch((*err*) => {

            console.log('не удалось записать операции');

            console.log(*err*);

        });

*// запись всех документов*

        totalIndocsArray = totalIndocsArray.concat(totalOutdocsArray)

        totalIndocsArray = [...new Map(totalIndocsArray.map(*item* => [*item*['guid'], *item*])).values()];

        tutDB.tutorialDocs.bulkCreate(totalIndocsArray, { individualHooks: true, updateOnDuplicate: ["guidoper", "name", "attrs", "connecctor"] }).then(() => {

            console.log('документв  в БД записаны');

        }).catch((*err*) => {

            console.log('не удалось записать документы');

            console.log(*err*);

        });

*//запись всех исполнителей*

        totalExecutors = [...new Map(totalExecutors.map(*item* => [*item*['guid'], *item*])).values()];

        tutDB.tutorialIsp.bulkCreate(totalExecutors, { individualHooks: true, updateOnDuplicate: ["guidoper", "name", "attrs"] }).then(() => {

            console.log('исполнители  в БД записаны');

        }).catch((*err*) => {

            console.log('не удалось записать исполнителей');

            console.log(*err*);

        });

*//запись всех атрибутов документов*

        totalAttrsArray = [...new Map(totalAttrsArray.map(*item* => [*item*['guid'], *item*])).values()];

        tutDB.tutorialAttrDocs.bulkCreate(totalAttrsArray, { individualHooks: true, updateOnDuplicate: ["guiddoc", "name"] }).then(() => {

            console.log('атрибуты документов в БД записаны');

        }).catch((*err*) => {

            console.log('не удалось записать атрибуты документов');

            console.log(*err*);

        });

*//запись всех коннекторов*

        totalConnectorsArray = [...new Map(totalConnectorsArray.map(*item* => [*item*['guid'], *item*])).values()];

        tutDB.tutorialConnector.bulkCreate(totalConnectorsArray, { individualHooks: false, updateOnDuplicate: ["guiddoc", "name"] }).then(() => {

            console.log('соединиттели документов в БД записаны');

        }).catch((*err*) => {

            console.log('не удалось записать соединители документов');

            console.log(*err*);

        }).finally(() => {

*// tutDB.sequelize.close();*

        });

    })

    if (bpObj) {

*//return res.status(200)*

*//  res.send("Спасибо! \nВаша статья успешно добавлена!")*

*res*.send("Спасибо! " + utf8.encode(bpObj.name))

    }

    console.timeEnd('ApiArisStart')

})

*//const filePath = '/users.json'*

server.post('/api/setppbyid/:id', async function(*req*, *res*) {

    console.time('Update\_in\_to\_DB\_by\_PPID')

    const updateObj = JSON.parse(*req*.params.id)

*/\* образец данных на вход*

*{"bpppnum":"1.1.:1.","id":"1","idbp":"1676304","idpp":"2109445","bpnum":"1.1.","ppnum":"1.","expert":"null","ca":"","csbo":"","rDmin":"null","rDreg":"null","date":1624977882000,"user":"8\_Packevich\_032","edited":"null","createdAt":1624977882000,"updatedAt":1624977882000}*

*\*/*

    const resupdate = await db.PPO.update({

        expert: updateObj.expert,

        ca: updateObj.ca,

        csbo: updateObj.csbo,

        rDmin: updateObj.rDmin,

        rDreg: updateObj.rDreg,

        date: updateObj.date,

        user: updateObj.user,

        edited: updateObj.edited,

        createdAt: updateObj.createdAt,

        updatedAt: updateObj.updatedAt

    }, {

        where: {

            idbp: updateObj.idbp,

            idpp: updateObj.idpp

        }

    });

*res*.send(resupdate) *//возвращает кол-во обновленных строк*

    console.timeEnd('Update\_in\_to\_DB\_by\_PPID')

});

server.get('/api/getallknowledge', async function(*req*, *res*) {

    console.time('getKnowledge')

    await db.KNOWLEDGE.findAll().then((*note*) => {

        if (!*note*) {

*res*.send("Запрашиваемая таблица не найдена.");

        } else {

*note* = *note*.map(function(*r*) { return *r*.get({ plain: true }) })

*res*.send(*note*)

        }

    })

    console.timeEnd('getKnowledge')

});

server.post('/api/addknowledge/', async function(*req*, *res*) {

    console.time('addKnowledge')

    const data = *req*.body;

    console.log(data);

    async function updateOrCreate(*model*, *where*, *newItem*) {

*// First try to find the record*

        const foundItem = await *model*.findOne({ where });

        if (!foundItem) {

*// Item not found, create a new one*

            const item = await *model*.create(*newItem*)

            return { item, created: true };

        }

*// Found an item, update it*

        const item = await *model*.update(*newItem*, { where });

        return { item, created: false };

    }

    updateOrCreate(db.KNOWLEDGE, { id: data.id }, data);

*// const data = JSON.parse(req.params.id);*

*// let existed = await db.KNOWLEDGE.findOrCreate({*

*//     where: { theme: data.theme, textnode: data.textnode, title: data.title, users: data.users, data: data.data }*

*// });*

*// if (existed[1]) {*

*//     res.send("Спасибо! \nВаша статья успешно добавлена!")*

*// }*

    console.timeEnd('addKnowledge')

})

server.get('/api/getppbyid/:id', async function(*req*, *res*) {

    console.time('Begin\_get\_fromDB\_by\_PPID')

*// const bp = req.params.id.split(":")[0];*

*// const idpp = req.params.id.split(":")[1];*

    const bpnum = *req*.params.id.split(":")[1];

    const ppnum = *req*.params.id.split(":")[2];

    db.PPO.findOne({ where: { */\*idpp: idpp\*/* bpnum: bpnum, ppnum: ppnum } }).then((*note*) => {

        if (!*note*) {

*res*.send("Запрашиваемый идентификатор не найден");

        } else {

            console.log(note.get({ plain: true }));

            console.log('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

            console.log(`Выполнено idpp: ${*note*.idpp}, ppnum: ${*note*.ppnum}`);

*res*.send({

                bpppnum: `${*note*.bpnum}:${*note*.ppnum}`,

                id: `${*note*.id}`,

                idbp: `${*note*.idbp}`,

                idpp: `${*note*.idpp}`,

                bpnum: `${*note*.bpnum}`,

                ppnum: `${*note*.ppnum}`,

                expert: `${*note*.expert}`,

                ca: `${*note*.ca}`,

                csbo: `${*note*.csbo}`,

                rDmin: `${*note*.rDmin}`,

                rDreg: `${*note*.rDreg}`,

                date: new Date(`${*note*.date}`).getTime(),

                user: `${*note*.user}`,

                edited: `${*note*.edited}`,

                createdAt: new Date(`${*note*.createdAt}`).getTime(),

                updatedAt: new Date(`${note.updatedAt}`).getTime(),

            });

        }

    }).finally(() => {})

    console.timeEnd('Begin\_get\_fromDB\_by\_PPID')

})

server.get('/api/getparticipants:id', async function(*req*, *res*) {

    console.time('GetParticipantsAndExperts')

    const idbp = *req*.params.id.slice(1);

*// const numpp = req.params.id.slice(1).split("&")[1]*

    let resp = null;

*// await db.PPO.sequelize.authenticate().then(() => {*

    await db.PPO.findAll({ where: { idbp: idbp } }).then((*note*) => {

        if (!*note*) {

*res*.send("Запрашиваемая таблица не найдена.");

        } else {

*// note = note.map(function(r) { return r.get({ plain: true }) })*

            resp = note.map((*r*) => { return r.dataValues })

            resp = resp.map((*el*) => {

                return {

                    bpnum: *el*.bpnum,

                    ca: "" + *el*.ca,

                    createdAt: "" + *el*.createdAt,

                    csbo: '' + *el*.csbo,

                    date: "" + *el*.date,

                    edited: (*el*.edited == null) ? "" : *el*.edited,

                    expert: *el*.expert, *// '8000003200,8000003000,8000001500,8000004900'*

                    id: "" + *el*.id,

                    idbp: "" + *el*.idbp, *// 1676304*

                    idpp: "" + *el*.idpp, *//2109445*

                    ppnum: *el*.ppnum, *//'1.'*

                    rDmin: *el*.rDmin,

                    rDreg: *el*.rDreg,

                    updatedAt: *el*.updatedAt, *//Fri Dec 17 2021 15:30:09 GMT+0300 (Москва, стандартное время)*

                    user: *el*.user *// '8\_Packevich\_032'*

                }

            });

*res*.send(resp)

        }

    }).finally(() => {})

    console.timeEnd('GetParticipantsAndExperts')

});

server.get('/uploadfiles', async function(*req*, *res*) {

        console.time('Begintime\_API\_uploadfiles')

        const files = ['C:\\aris\\sharefld\\DataTreeBP.json', 'C:\\aris\\sharefld\\DataTreeSubBP.json', 'C:\\aris\\sharefld\\BpOwnExps.json']

        const getFile = (*fileName*) => {

            return new Promise((*resolve*, *reject*) => {

                fs.readFile(*fileName*, 'utf8', (*err*, *data*) => {

                    if (*err*) {

                        reject(*err*) *// calling `reject` will cause the promise to fail with or without the error passed as an argument*

                        return *// and we don't want to go any further*

                    }

                    resolve(JSON.parse(*data*))

                })

            })

        }

        let bpcontent = null

        let ppcontent = null

*// let analitica = null*

*// let BpOwnExps = null*

        const filesContentArray = await Promise.all([getFile(files[0]), getFile(files[1]), getFile(files[2])]).catch(function(*err*) {

*res*.send(*err*.message)

        })

        bpcontent = filesContentArray[0]

        ppcontent = filesContentArray[1]

        let experts = filesContentArray[2]

        let data = [];

        let bp = null

        let pptemp = null

        let expertsTMP = null;

        const datestamp = new Date().getTime();

*//  db.PPO.sync({ force: true })*

        for (let i = 0; i < bpcontent.length; i++) {

            bp = bpcontent[i];

            await db.PPO.destroy({

                    where: {

                        idbp: bp.bpid *//this will be your id that you want to delete*

                    }

                }) *// rowDeleted will return number of rows deleted*

            pptemp = ppcontent.filter((*pp*) => {

                return *pp*.bpid == bp.bpid

            })

            expertsTMP = experts.filter(function(*exp*) { return *exp*.bpid === bp.bpid }).map((*r*) => { return *r*.experts.map((*t*) => { return *t*.expert10code }) }).join(',')

            if (!expertsTMP) expertsTMP = ""

            if (pptemp.length == 0) {

                data.push({

                    idbp: bp.bpid,

                    idpp: '',

                    bpnum: bp.bpnum,

                    ppnum: '',

                    expert: expertsTMP,

*//"",*

                    ca: "",

                    csbo: "",

                    rDmin: "",

                    rDreg: "",

                    date: datestamp,

                    user: "8\_Packevich\_032"

                })

            } else {

                for (let j = 0; j < pptemp.length; j++) {

                    data.push({

                        idbp: bp.bpid,

                        idpp: pptemp[j].subbpcode,

                        bpnum: bp.bpnum,

                        ppnum: pptemp[j].ppnum,

                        expert: expertsTMP,

*//"",*

                        ca: "",

                        csbo: "",

                        rDmin: "",

                        rDreg: "",

*// edited: "0",*

                        date: datestamp,

                        user: "8\_Packevich\_032"

                    })

                }

            }

*/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*

        }

        db.sequelize.sync({ force: false }).then(() => {

            db.PPO.bulkCreate(data, { individualHooks: true }).then(() => {

                console.log('notes created');

            }).catch((*err*) => {

                console.log('failed to create notes');

                console.log(*err*);

            }).finally(() => {

*//sequelize.close();*

            });

        });

*res*.send(data)

        console.timeEnd('Begintime\_API\_uploadfiles')

    })

*// server.get('/firstSQL', function (req, res) {*

*//   console.time('BegintimeFirstSQL request')*

*//   /\*const sqlstring = 'SELECT \* FROM PPO.test'*

*//     connection.query(sqlstring, function (error, results, fields) {*

*//     if (error) throw error*

*//     console.log(new Date() + ' Отправляем: ' + results + new Date())*

*//     res.send(results)*

*//   })\*/*

*//   console.timeEnd('BegintimeFirstSQL request')*

*// })*

server.get('/', function(*req*, *res*) {

*res*.sendFile('index.html', { root: \_\_dirname })

})

server.get('/api/one', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\analitica.json')

})

server.get('/api/for\_pzk', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\for\_PZK.json')

})

server.get('/api/relations', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\relations.json')

})

server.get('/analitics.html', function(*req*, *res*) {

*res*.sendFile('analitics.html', { root: \_\_dirname })

})

server.get('/lpa.html', function(*req*, *res*) {

*res*.sendFile('lpa.html', { root: \_\_dirname })

})

server.get('/reg1', function(*req*, *res*) {

*res*.sendFile('3c458750-1c6e-11e8-1e27-0017a4772806.html', { root: \_\_dirname.concat('\\public') })

})

server.get('/localknowledge.html', function(*req*, *res*) {

*res*.sendFile('localknowledge.html', { root: \_\_dirname })

})

server.get('/form.html', function(*req*, *res*) {

*res*.sendFile('form.html', { root: \_\_dirname })

})

server.get('/filer.html', function(*req*, *res*) {

*res*.sendFile('filer.html', { root: \_\_dirname })

})

server.get('/tree.html', function(*req*, *res*) {

*res*.sendFile('tree.html', { root: \_\_dirname })

})

server.get('/knowledge.html', function(*req*, *res*) {

*res*.sendFile('knowledge.html', { root: \_\_dirname })

})

server.get('/api/bpcount', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\bpcount.txt')

})

server.get('api/getallppid/', function(*req*, *res*) {

    var fs = require('fs')

    var contents = fs.readFileSync('C:\\aris\\sharefld\\DataTreeSubBP.json')

    var jsonContent = JSON.parse(contents, function(*k*, *v*) {

        console.log(*k*) *// пишем имя текущего свойства, последним именем будет ""*

        return *v* *// возвращаем неизменённое значение свойства*

    })

*res*.send(jsonContent)

})

server.get('/api/getallbp', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\DataTreeBP.json')

})

server.get('/api/getMD', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\ModelTree.json')

})

server.get('/api/pasports', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\pasports.json')

})

server.get('/api/orgStrGPS', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\orgStrGPS.json')

})

server.get('/api/orgStrCBURD', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\orgStrCBURD.json')

})

server.get('/api/orgStrCSBO', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\orgStrCSBO.json')

})

server.get('/api/orgStrCA', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\orgStrCA.json')

})

server.get('/api/getallpp', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\DataTreeSubBP.json')

})

server.get('/api/BpOwnExps', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\BpOwnExps.json')

})

server.get('/api/PZK\_TREE', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\ArisApi\\public\\BIGtree.json')

})

server.get('/api/BP\_PP\_TREE', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\ArisApi\\public\\BIGsubBBandPP.json')

})

server.get('/api/PPExps', function(*req*, *res*) {

*res*.sendFile('C:\\aris\\sharefld\\PPExps.json')

})

server.get('/api/getbpandpp2', async function(*req*, *res*) {

    console.time('Begintime\_API\_getbpandpp2')

    const files = ['C:\\aris\\sharefld\\DataTreeBP.json',

        'C:\\aris\\sharefld\\DataTreeSubBP.json',

        'C:\\aris\\sharefld\\analitica.json',

        'C:\\aris\\sharefld\\BpOwnExps.json'

    ]

    const getFile = (*fileName*) => {

        return new Promise((*resolve*, *reject*) => {

            fs.readFile(*fileName*, 'utf8', (*err*, *data*) => {

                if (*err*) {

                    reject(err) *// calling `reject` will cause the promise to fail with or without the error passed as an argument*

                    return *// and we don't want to go any further*

                }

                resolve(JSON.parse(*data*))

            })

        })

    }

    let bpcontent = null

    let ppcontent = null

    let analitica = null

    let BpOwnExps = null

    const filesContentArray = await Promise.all([getFile(files[0]), getFile(files[1]), getFile(files[2]), getFile(files[3])]).catch(function(*err*) {

*res*.send(err.message)

    })

    bpcontent = filesContentArray[0]

    ppcontent = filesContentArray[1]

    analitica = filesContentArray[2]

    BpOwnExps = filesContentArray[3]

    let analitic = ''

    let ownerpos = null

    let bp = null

    let pptemp = null

    let iso = null

    let experts = null

    for (let i = 0; i < bpcontent.length; i++) {

        bp = bpcontent[i]

        analitic = analitica.filter(*an* => {

            return *an*.bpid === bp.bpid

        }).map(function(*obj*) { return *obj*.analitic })

        iso = analitica.filter(*an* => {

            return *an*.bpid === bp.bpid

        }).map(function(*obj*) { return *obj*.iso })

        ownerpos = analitica.filter(*an* => {

            return *an*.bpid === bp.bpid *//an.ownercode === bp.bpowner*

        }).map(function(*obj*) { return *obj*.owner })

        experts = BpOwnExps.filter(*exp* => {

            return *exp*.bpid === bp.bpid

        }).map(function(*obj*) { return *obj*.experts })

        bp.bpownerpos = ownerpos[0]

        bp.analitic = analitic[0]

        bp.iso = iso[0]

        bp.experts = experts[0]

        pptemp = ppcontent.filter((*pp*) => {

            return *pp*.bpid == bp.bpid

        })

        for (let j = 0; j < pptemp.length; j++) {

            pptemp[j].bpownerpos = bp.ownerpos

            pptemp[j].analitic = bp.analitic

            pptemp[j].bpnum = bp.bpnum

        }

    }

*res*.send([bpcontent, ppcontent])

    console.timeEnd('Begintime\_API\_getbpandpp2')

})

server.get('/api/getbpandpp', async function(*req*, *res*) {

    console.time('Begintime\_API\_getbpandpp')

    const files = ['C:\\aris\\sharefld\\DataTreeBP.json',

        'C:\\aris\\sharefld\\DataTreeSubBP.json',

        'C:\\aris\\sharefld\\BpOwnExps.json'

    ]

    const getFile = (*fileName*) => {

        return new Promise((*resolve*, *reject*) => {

            fs.readFile(*fileName*, 'utf8', (*err*, *data*) => {

                if (*err*) {

                    reject(*err*) *// calling `reject` will cause the promise to fail with or without the error passed as an argument*

                    return *// and we don't want to go any further*

                }

                resolve(JSON.parse(*data*))

            })

        })

    }

    let bpcontent = null

    let ppcontent = null

    let bpOwnerContent = null

    const filesContentArray = await Promise.all([getFile(files[0]), getFile(files[1]), getFile(files[2])]).catch(function(*err*) {

*// dispatch a failure and throw error*

*// throw err*

*res*.send(*err*.message)

    })

    bpcontent = filesContentArray[0]

    console.log(filesContentArray[0])

    ppcontent = filesContentArray[1]

    console.log(filesContentArray[1])

    bpOwnerContent = filesContentArray[2]

    console.log(filesContentArray[2])

*// const bpshort = bpcontent.map(function (bp) { return { bpid: bp.bpid, bpnum: bp.bpnum, bpname: bp.bpname, bpowner: bp.bpowner } })*

    function TSH(*s*) {

*//  let s = obj.bpid;*

        for (var i = 0, h = 9; i < *s*.length;) { h = Math.imul(h ^ *s*.charCodeAt(i++), 9 \*\* 9) }

        return h ^ h >>> 9

    }

    function mySortObjOccsByName(*a*, *b*) {

        var a1 = *a*.ppnum

        a1 = a1.split('.')

        var b1 = *b*.ppnum

        b1 = b1.split('.')

        for (var i = 0; i < a1.length; i++) {

            if (b1.length - 1 < i) {

                return 1

            } *// b1 = 1.1. a1 =  1.1.1.*

            if (a1[i] === b1[i]) {

                continue

            }

            return a1[i] - b1[i]

        }

        return 0

    }

    const bphash = {}

    console.time('Hashtime')

    const bppparray = []

    let ppsubarray = []

    let bpOwnername = []

    bpcontent.forEach(function(*bp*) {

        bphash[*bp*.bpid] = TSH(*bp*.bpid)

        ppsubarray = ppcontent.filter(function(*pp*) {

            return *pp*.bpid === *bp*.bpid

        })

        bpOwnername = bpOwnerContent.filter(function(*owner*) {

            if (*owner*.bpid === *bp*.bpid) return *owner*

        })

        ppsubarray = ppsubarray.sort(mySortObjOccsByName)

        bppparray.push({

                bpid: *bp*.bpid,

                bpnum: *bp*.bpnum,

                bpname: *bp*.bpname,

                bpownerName: bpOwnername[0],

                pparray: ppsubarray

            }

        )

    })

    console.timeEnd('Hashtime')

*res*.send(bppparray)

    console.timeEnd('Begintime\_API\_getbpandpp')

})

server.get('/api/test', async function(*req*, *res*) {

    console.time('test')

    const files = ['C:\\aris\\sharefld\\analitica.json',

        'C:\\aris\\sharefld\\DataTreeSubBP.json'

    ]

    const getFile = (*fileName*) => {

        return new Promise((*resolve*, *reject*) => {

            fs.readFile(*fileName*, 'utf8', (*err*, *data*) => {

                if (*err*) {

                    reject(*err*) *// calling `reject` will cause the promise to fail with or without the error passed as an argument*

                    return *// and we don't want to go any further*

                }

                resolve(JSON.parse(*data*))

            })

        })

    }

    let bpcontent = null

    let ppcontent = null

    const filesContentArray = await Promise.all([getFile(files[0]), getFile(files[1])]).catch(function(*err*) {

*// dispatch a failure and throw error*

*// throw err*

*res*.send(err.message)

    })

    bpcontent = filesContentArray[0]

    console.log(filesContentArray[0])

    ppcontent = filesContentArray[1]

    console.log(filesContentArray[1])

*// const bpshort = bpcontent.map(function (bp) { return { bpid: bp.bpid, bpnum: bp.bpnum, bpname: bp.bpname, bpowner: bp.bpowner } })*

    function TSH(*s*) {

*//  let s = obj.bpid;*

        for (var i = 0, h = 9; i < *s*.length;) { h = Math.imul(h ^ *s*.charCodeAt(i++), 9 \*\* 9) }

        return h ^ h >>> 9

    }

    function mySortObjOccsByName(*a*, *b*) {

        var a1 = *a*.ppnum

        a1 = a1.split('.')

        var b1 = *b*.ppnum

        b1 = b1.split('.')

        for (var i = 0; i < a1.length; i++) {

            if (b1.length - 1 < i) {

                return 1

            } *// b1 = 1.1. a1 =  1.1.1.*

            if (a1[i] === b1[i]) {

                continue

            }

            return a1[i] - b1[i]

        }

        return 0

    }

    const bphash = {}

    console.time('Hashtime')

    const bppparray = []

    let ppsubarray = []

    bpcontent.forEach(function(*bp*) {

        bphash[*bp*.bpid] = TSH(*bp*.bpid)

        ppsubarray = ppcontent.filter(function(*pp*) {

            return *pp*.bpid === *bp*.bpid

        })

        ppsubarray = ppsubarray.sort(mySortObjOccsByName)

        bppparray.push({

                bpid: *bp*.bpid,

                bpname: *bp*.bp,

                analitic: *bp*.analitic,

                owner: *bp*.owner,

                pparray: ppsubarray

            }

        )

    })

    console.timeEnd('Hashtime')

*res*.send(bppparray)

    console.timeEnd('Begintime\_API\_getbpandpp')

})

*// const DBSOURCE = \_\_dirname + '/' + 'ArisSQLITE.db'*

*/\* const db = new sqlite3.Database(DBSOURCE, sqlite3.OPEN\_READWRITE, (err) => {*

*if (err) {*

*console.error('Erro opening database ' + err.message)*

*} else {*

*// db.req()*

*console.log('Server running on port %PORT%'.replace('%PORT%', HTTP\_PORT))*

*console.log('db opened....')*

*}*

*}) \*/*

server.listen(8888, function() {

    console.log('Сервер ожидает подключения.on port 8888..' + new Date())

})