```
//////////////////
// package.json //
//////////////////

{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "serve.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "mongoose-unique-validator": "^2.0.3"
  },
  "devDependencies": {
    "bcrypt": "^5.0.1",
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.19.0",
    "debug": "^2.6.9",
    "jsonwebtoken": "^8.5.1",
    "mongodb": "^4.0.1",
    "mongoose": "^5.13.5"
  }
}


///////////////
// serve.js //
/////////////

const app = require("./backend/app")
const debug = require("debug")("Server")
const http = require("http")

console.log("Hitting the deck!")

const normalizePort = val => {
    let port = parseInt(val, 10)

    // named pipe
    if (isNaN(port)) return val

    return port >= 0 ? port : false
}

const onError = error => {
    if (error.syscall !== "listen") throw error

    const bind = typeof addr === "string" ? "pipe " + addr : "port " + port
    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
    }
}

const onListening = () => {
    const addr = server.address()
    const bind = typeof addr === "string" ? "pipe " + addr : "port " + port
    debug("Listening on " + bind)
}

const port = normalizePort(process.env.PORT || "3000")
```

```
app.set("port", port)

const server = http.createServer(app)
server.on("error", onError)
server.on("listening", onListening)
server.listen(port)


//////////////////////
// backend > app.js //
//////////////////////

const express = require("express")
const mongoose = require("mongoose")
const app = express()

const mongo =
    "mongodb+srv://dbAdmin:ao32yCOOJajNhcR7@cluster0.kt8sb.mongodb.net/myFirstDatabase?retryWrites=true&w=majority"

mongoose.set("useNewUrlParser", true)
mongoose.set("useFindAndModify", false)
mongoose.set("useUnifiedTopology", true)
mongoose.set("useCreateIndex", true)
mongoose
    .connect(mongo)
    .then(() => console.log("Connected to db!"))
    .catch(err => console.log(err))

const bp = require("body-parser")
const bcrypt = require("bcryptjs")
const User = require("./models/user")
const adminRoute = require("./routes/admin")

app.use(bp.json())
app.use(bp.urlencoded({ extended: false }))

/* Set headers */
app.use((req, res, next) => {
    res.setHeader("Access-Control-Allow-Origin", "*")
    res.setHeader(
        "Access-Control-Allow-Headers",
        "Origin, X-Requested-With, Content-Type, Accept"
    )
    res.setHeader(
        "Access-Control-Allow-Methods",
        "GET, PUT, POST, PATCH, DELETE, OPTIONS"
    )
    next()
})

/* login */
app.post("/api/login", async (req, res) => {
    console.log("/api/login")

    const STATUS = ["ACTIVE", "INACTIVE"]
    let fetchedUser
    await User.findOne({ username: req.body.username }).exec(
        async (err, user) => {
            if (err) {
                res.statusMessage = "Unknown error"
                res.status(400).end()
            }
            if (!user) {
                res.statusMessage = "User not found"
                res.status(400).end()
            } else {
                fetchedUser = user
                await bcrypt
                    .compare(req.body.password, fetchedUser.hash)
                    .then(result => {
                        if (result) {
                            if (fetchedUser.status === STATUS[1]) {
                                console.log(fetchedUser.status)
                                console.log(STATUS[0])
                                res.statusMessage = "User is disabled. Contact Admin."
                                res.status(400).end()
```

```
                        } else if (
                            /* If a user is not an admin and has no assigned cage/box, do not log in */
                            !fetchedUser.admin &&
                            (!fetchedUser.cageId || !fetchedUser.boxId)
                        ) {
                            res.statusMessage =
                                "Non-admin user has no assigned box or cage. Contact Admin."
                            res.status(400).end()
                        } else {
                            return res
                                .status(200)
                                .json({ message: "Successful login" })
                        }
                    } else {
                        res.statusMessage = "Incorrect password"
                        res.status(400).end()
                    }
                })
        }
    }
)
})

app.use("/api/admin", adminRoute)
module.exports = app


//////////////////////////////
//backend > routes > admin.js//
//////////////////////////////

const express = require("express")
const admin = express.Router()

const UserController = require("../controllers/admin/users")
const CageController = require("../controllers/admin/cages")
const BoxController = require("../controllers/admin/boxes")

/* Middle wear */

/* Get list of users */
/* Returns a list of users */
admin.get("/users", UserController.getUsers)
/* Add user */
/* Validates and adds user to db */
admin.post("/users/register", UserController.registerUser)
/* Modify user */
/* Validates and adds changes to user to db */
admin.put("/user/:id/modify", UserController.modifyUser)

/* Delete user */
/* Validates and deletes user in db */
admin.delete("/user/:id/delete", UserController.deleteUser)

/* Get list of cages */
/* Returns a list of cages */
admin.get("/cages", CageController.getCages)
/* Add cage */
/* Validates and adds cage to db */
admin.post("/cages/register", CageController.registerCage)
/* Update cage */
/* Validates and adds changes to cage in db */
admin.put("/cage/:id/update", CageController.updateCage)
/* Delete cage */
/* Validates and deletes cage in db */
admin.delete("/cage/:id/delete", CageController.deleteCage)

/* Get list of boxes in a cage */
/* Returns a list of boxes for specified cage */
admin.get("/cage/:id/boxes", BoxController.getBoxes)
/* Add box to a cage */
/* Validates and adds a box to specified cage */
admin.put("/cage/:cageId/box/:boxId", BoxController.addBox)
/* Delete box from a cage */
/* Validates and deletes box from specified cage */
admin.put("/cage/:cageId/box/:boxId/delete", BoxController.deleteBox)
```

```
module.exports = admin



////////////////////////////////
//backend > models > cage.js //
////////////////////////////////

const { Schema, model } = require('mongoose')
const uniqueValidator = require('mongoose-unique-validator')

const STATUS = ["ACTIVE", "INACTIVE"]
const UserSchema = new Schema({
    username: {
        type: Number,
        required: true,
        unique: true
    },
    hash: {
        type: String,
        required: true
    },
    admin: {
        type: Boolean,
        required: true
    },
    cageId: Number,
    boxId: {
        type: Number,
        trim: true,
        sparse: true
    },
    status: {
        type: String,
        enum: STATUS,
        default: STATUS[1]
    }
})

UserSchema.plugin(uniqueValidator)
module.exports = model("User", UserSchema)


////////////////////////////////
//backend > models > cage.js //
////////////////////////////////

const { Schema, model } = require("mongoose")

const uniqueValidator = require("mongoose-unique-validator")

const CageSchema = new Schema({
    id: {
        type: Number,
        required: true,
        unique: true,
    },
    status: {
        type: String,
        enum: ["ACTIVE", "INACTIVE", "ERROR"],
        default: "INACTIVE",
    },
    boxes: {
        type: [Number],
        unique: true,
        default: undefined,
    },
    lastUsed: {
        type: String,
        default: "N/A",
    },
})

CageSchema.plugin(uniqueValidator)
module.exports = model("Cage", CageSchema)
```

```
//////////////////////////////////////////
//backend > controllers > admin > users.js //
//////////////////////////////////////////
const User = require("../../models/user")
const bcrypt = require("bcryptjs")

exports.getUsers = async (req, res) => {
    console.log("/admin/users")
    /* Finds all users in db, includes all relevant fields except for hash */
    await User.find()
        .select("username admin cageId boxId status")
        .then(users => {
            return res.status(200).json({
                data: users,
            })
        })
        .catch(err => {
            res.statusMessage = err
            res.status(400).end()
        })
}

exports.registerUser = async (req, res) => {
    console.log("/admin/users/register")

    if (!req.body.admin && (!req.body.cageId || !req.body.boxId)) {
        res.statusMessage = "Non-admin user not assigned cage or box."
        res.status(400).end()
    } else if (req.body.admin && (req.body.cageId || req.body.boxId)) {
        res.statusMessage = "Admin can not be assigned a box or cage."
        res.status(400).end()
    }

    await bcrypt.hash(req.body.password, 10).then(hash => {
        const user = new User({
            username: req.body.username,
            hash: hash,
            admin: req.body.admin,
            cageId: req.body.cageId,
            boxId: req.body.boxId,
            status: req.body.status,
        })
        user
            .save()
            .then(result => {
                if (result) {
                    console.log("User created!")
                    return res.status(201).json({
                        data: "User successfully created",
                    })
                } else {
                    res.statusMessage = "Could not register user"
                    res.status(400).end()
                }
            })
            .catch(err => {
                res.statusMessage = err
                res.status(400).end()
            })
    })
}

exports.modifyUser = async (req, res) => {
    console.log("/admin/user/id/modify")
    /* checks if a password change was requested. */
    /* If so, hash the new password */
    let hashed = ""
    if (req.body.password) {
        await bcrypt.hash(req.body.password, 10).then(hash => {
            hashed = hash
        })
    }

    let updates = {
```

```
        hash: hashed,
        admin: req.body.admin,
        cageId: req.body.cageId,
        boxId: req.body.boxId,
        status: req.body.status,
    }
    console.log(Object.keys(updates))
    console.log(updates[0])
    console.log(Object.values(updates))

    for (let key in updates) {
        if (
            updates[key] === null ||
            updates[key] === undefined ||
            updates[key] === ""
        ) {
            console.log(`deleted ${updates[key]}`)
            delete updates[key]
        }
    }

    await User.updateOne(
        { username: req.params.id },
        { $set: { ...updates } },
        {
            new: true,
        }
    ).then(update => {
        if (!update) {
            res.statusMessage = "Error updating user"
            res.status(400).end()
        }

        return res
            .status(201)
            .json({ data: `Successfully updated user ${req.params.id}` })
    })
}

exports.deleteUser = async (req, res) => {
    console.log("/user/:id/delete")
    await User.deleteOne({ username: req.params.id })
        .then(err => {
            if (!err.deletedCount) {
                res.statusMessage = "Error deleting user"
                res.status(400).end()
            }
            return res.status(200).json({
                data: `User ${req.params.id} deleted successfully`,
            })
        })
        .catch(err => {
            res.statusMessage = err
            res.status(400).end()
        })
}


//////////////////////////////////////////
//backend > controllers > admin > cages.js //
//////////////////////////////////////////

const Cage = require("../../models/cage")

exports.getCages = async (req, res) => {
    await Cage.find()
        .then(cages => {
            return res.status(200).json({
                data: cages,
            })
        })
        .catch(err => {
            res.statusMessage = err
            res.status(400).end()
        })
}
```

```
exports.registerCage = async (req, res) => {
    const cage = new Cage({
        id: req.body.id,
    })
    cage
        .save()
        .then(cage => {
            if (cage) {
                return res.status(200).json({ data: `Cage successfully added` })
            } else {
                res.statusMessage = "Error adding cage"
                res.status(400).end()
            }
        })
        .catch(err => {
            res.statusMessage = err
            res.status(400).end()
        })
}

exports.updateCage = (req, res) => {
    Cage.where({ id: req.params.id })
        .findOne()
        .then(cage => {
            const statusChange = cage.get("status") !== req.body.status

            if (!cage || !statusChange) {
                res.statusMessage = "Error updating cage"
                res.status(400).end()
            } else {
                let updates = {
                    status: req.body.status,
                    lastUsed: Date.now(),
                }

                cage
                    .updateOne(
                        { id: req.params.id },
                        { $set: { ...updates } },
                        { new: true }
                    )
                    .then(() => {
                        return res.status(200).json({
                            data: "successfully updated box ${req.params.id} to ${req.body.status",
                        })
                    })
                    .catch(err => {
                        res.statusMessage = err
                        res.status(400).end()
                    })
            }
        })
}

exports.deleteCage = async (req, res) => {
    await Cage.deleteOne({ id: req.params.id })
        .then(del => {
            if (del) {
                return res.status(200).json({
                    data: "Cage deleted successfully",
                })
            } else {
                res.statusMessage = "Error deleting cage"
                res.status(400).end()
            }
        })
        .catch(err => {
            return res.status(404).json({
                data: err.message,
            })
        })
}
```

```
//////////////////////////////////////////
//backend > controllers > admin > boxes.js //
//////////////////////////////////////////
const Cage = require("../../models/cage")

exports.getBoxes = async (req, res) => {
    await Cage.findOne({ id: req.params.id })
        .then(cage => {
            if (cage) {
                return res.status(200).json({
                    data: cage.boxes,
                })
            } else {
                res.statusMessage = "Error fetching cages"
                res.status(400).end()
            }
        })
        .catch(err => {
            res.statusMessage = err
            res.status(400).end()
        })
}

exports.addBox = async (req, res) => {
    await Cage.where({ id: req.params.id })
        .update({
            $push: {
                boxes: req.body.box,
            },
        })
        .then(() => {
            return res.status(200).json({
                data: `Successfully added box ${req.body.box} to ${req.params.id}`,
            })
        })
        .catch(err => {
            res.statusMessage = err
            res.status(400).end()
        })
}

exports.deleteBox = async (req, res) => {
    await Cage.updateOne(
        { id: req.params.id },
        { $pull: { boxes: req.params.boxId } }
    )
        .then(() => {
            return res.status(200).json({
                data: `Box ${req.params.boxId} successfully deleted from cage ${req.params.cageId}`,
            })
        })
        .catch(err => {
            res.statusMessage = err
            res.status(400).end()
        })
}
```