workshop overview

# workshop goals

# Server description and flow

getting started

# create your Server

Open a command line window and CD to where you'd like to create your server.

Type:

npm init



Select sever.js as entry point and hit enter through everything.

Open up your package.json and add this line:

"scripts": {

"start": "node server.js"

},

# Server setup

Express

Express.js is a web application framework for Node.js. It provides various features that make web application development fast and easy which otherwise takes more time using only Node.js.

In order to install express type:

npm install express --save



[body-parser](https://www.npmjs.com/package/body-parser)

Node.js body parsing middleware.

Parse incoming request bodies in a middleware before your handlers, available under the req.bodyproperty.

In order to install body-parser type:

npm install body-parser --save



MongoDB

In order to install mongoDB type:

npm install mongodb --save



Logging

Generally, we need to use a combination of 2 or more modules to get a proper logging environment setup. In this example we'll be making use of *Morgan* and *Winston* for our logging needs.

In order to install *Morgan* type:

npm install morgan --save

In order to install *Winston* type:

npm install winston@next --save

After the installations your package.json should look like:

{

"name": "quiz-app",

"version": "1.0.0",

"description": "demo quiz app",

"main": "server.js",

"scripts": {

"start": "node server.js"

},

"author": "",

"license": "ISC",

"dependencies": {

"body-parser": "^1.18.2",

"express": "^4.16.2",

"mongodb": "^3.0.1",

"morgan": "^1.9.0",

"winston": "^3.0.0-rc1"

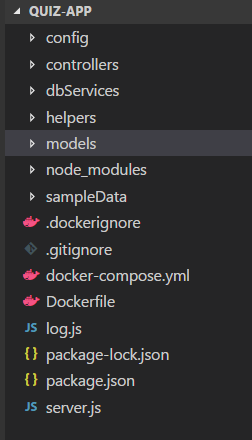
}

}

# Organize your app

##### Add some folders to your app folder, to be used later:

* config
* controllers
* dbServices
* helpers
* models
* sampleData



\*Some files will be added later.

We would like to have a file structure where different files and folders are responsible for different tasks.

Let’s see what files and folders there are at the root of our project with a brief explanation of what each of them is about:

* config/ – defines your app configurations
* controllers/ – defines your app routes and their logic
* dbServices/ – provides data base services
* helpers/ – code and functionality to be shared by different parts of the project
* models/ – represents data, implements business logic and handles storage
* server.js – initializes the app and glues everything together
* package.json – remembers all packages that your app depends on and their versions

# starting point of your application

##### Create a server.js file at the root of your project.

server.js is the starting point of your application.

It loads everything and it begins serving user requests.

Add the following code to server.js

const dataBaseService = require('./helpers/db') // we will define later

const logger = require('./log') // we will define later

const express = require('express')

, app = express()

, morgan = require('morgan')

, bodyParser = require('body-parser')

, winston = require('winston')

, port = process.env.PORT || 3001

Morgan and Winston used for logging, this lab will not cover the logging section.

You can read more in the following links:

<https://github.com/expressjs/morgan>

<https://github.com/winstonjs/winston>

Logging configurations - add the following code to server.js

const mainLogger = winston.createLogger({

level: 'info',

format: winston.format.json(),

transports: [

//

// - Write to all logs with level `info` and below to `combined.log`

// - Write all logs error (and below) to `error.log`.

//

new winston.transports.File({ filename: 'error.log', level: 'error' }),

new winston.transports.File({ filename: 'combined.log' })

]

});

//

// If we're not in production then log to the `console` with the format:

// `${info.level}: ${info.message} JSON.stringify({ ...rest }) `

//

if (process.env.NODE\_ENV !== 'production') {

mainLogger.add(new winston.transports.Console({

format: winston.format.simple()

}));

}

logger.log = mainLogger;

const httpLog = morgan(

'combined',

{

"stream": {

write: (str) => { mainLogger.info(str); }

}

});

Application-level middleware – bind the following middleware to express app

Add the following code to server.js

app.use(httpLog)

app.use(bodyParser.json())

app.use(bodyParser.urlencoded({ extended: true })) // extended: true – allows to post nested object

app.use(require('./controllers')) // we will define later

Connecting to the db and listening to port 3001

Add the following code to server.js

dataBaseService.connect()

.then(() => {

app.listen(port, () => {

mainLogger.info('Listening on port ' + port)

})

})

.catch((err => {

mainLogger.error(err)

process.exit(1);

}))

# Logger

##### Create a log.js file at the root of your project.

Add the following code to log.js

const logger = {log: undefined};

module.exports = logger; // export the logger object

The module.exports or exports is a special object which is included in every JS file in the Node.js application by default.

module is a variable that represents current module and exports is an object that will be exposed as a module.

So, whatever you assign to module.exports or exports, will be exposed as a module.

# Config

##### Create a index.js file in config folder.

Add the following code to index.js

const config = {

database: {

url: 'mongodb://localhost:27017',

dbName: 'quizdb'

}

};

module.exports = config;

# Controllers

This is the folder where you will be defining all the routes that your app will serve. Your controllers will handle web requests and interact with your models to process and retrieve data.

Usually you will have at least one file for each logical part of your application. For example, one file to handle quizzes action, another file to handle requests about questions and so on.

##### Create a controllerHandler.js in controllers folder.

Add the following code to controllerHandler.js

const ApiError = require('../helpers/apiError') // we will define later

const logger = require('../log');

const controllerHandler = (promise, params) => (req, res, next) => {

const boundParams = params ? params(req, res, next) : [];

promise(...boundParams)

.then( result => (res.status(200).json(result || { message: 'OK' })))

.catch(error => (next(Object.prototype.isPrototypeOf.call(ApiError.prototype, error) ? error : new ApiError(error))))

};

module.exports = controllerHandler;

@param promise - model promise. e.g. Quiz.getById.  
@param params - a function e.g. (req, res, next) => [req.params.id]

In case of no error controllerHandler set status to 200 and return the result to the client.

In case of error we are simply redirecting any captured error to Express error handler.

##### Create a quizzes.js in controllers folder.

The following creates a router as a module, defines some routes, exposes the router object, and handles all quizzes CRUD operations.

Add the following code to quizzes.js

const controllerHandler = require('./controllerHandler')

const express = require('express')

, router = express.Router()

, Quiz = require('../models/quiz') // we will define later

, Question = require('../models/question') // we will define later

// define the /quizzes route

router.get('/', controllerHandler(Quiz.all, (req, res, next) => []))

// define the /quizzes/id route

router.get('/:id', controllerHandler(Quiz.getById, (req, res, next) => [req.params.id]))

router.patch('/:id', controllerHandler(Quiz.updateQuizResults, (req, res, next) => [req.params.id, req.body.score]))

router.get('/:id/questions', controllerHandler(Question.getByQuizId, (req, res, next) => [req.params.id]))

router.post('/', controllerHandler(Quiz.create, (req, res, next) => [req.body]))

module.exports = router

##### Create a questions.js in controllers folder.

questions.js will handle in all questions CRUD operations.

Add the following code to questions.js

const controllerHandler = require('./controllerHandler')

const express = require('express')

, router = express.Router()

, Question = require('../models/question')

router.get('/:id', controllerHandler(Question.getById, (req, res, next) => [req.params.id]))

router.post('/', controllerHandler(Question.create, (req, res, next) => [req.body]))

module.exports = router

##### Create a index.js in controllers folder.

This file’s router holds all your routes. This is the only router that your application has to load at startup.

Add the following code to index.js

const ApiError = require ('../helpers/apiError') // we will define later

const logger = require('../log');

const express = require('express')

, router = express.Router()

router.use('/setup', require('./setup')) // we will define later

router.use('/quizzes', require('./quizzes'))

router.use('/questions', require('./questions'))

Express does have a default error handler which you should be using to capture at least the most unexpected errors.

Add the following code to index.js

router.use((err, req, res, next) => {

// Expected errors always throw Error.

// Unexpected errors will either throw unexpected stuff or crash the application.

if (Object.prototype.isPrototypeOf.call(ApiError.prototype, err)) {

return res.status(err.status).json({ error: err.message });

}

logger.log.error(`Unexpected error exception: ${err}`);

return res.status(500).json({ error: 'Unexpected Error' });

});

module.exports = router

# Helpers

##### Create a apiError.js in helpers folder.

Add the following code to apiError.js

class ApiError{

constructor(message, status = 500){

this.message = this.parsMessage(message);

this.status = status;

}

parsMessage(message){

if(typeof message === "object"){

return message.message;

}

return message;

}

}

module.exports = ApiError;

##### Create a db.js in helpers folder.

In db.js we will connect to mongoDb and expose the db client.

Add the following code to db.js

const MongoClient = require('mongodb').MongoClient;

const config = require('../config')

class DbClient {

constructor() {

this.\_db = null

}

connect() {

if (this.\_db !== null)

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

resolve();

})

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

MongoClient.connect(config.database.url, (err, client) => {

if (err) {

reject(err);

return;

}

this.\_db = client.db(config.database.dbName);

resolve();

});

})

}

get db(){

return this.\_db;

}

}

const databaseService = new DbClient();

module.exports = databaseService;

##### Create a dbUtils.js in helpers folder.

Add the following code to dbUtils.js

const ObjectId = require('mongodb').ObjectId;

exports.stringToObjectId = (id) => {

return new ObjectId(id);

}

exports.stringIDToObjectId = (arr, filedName) => {

return arr.map(o => {

if(o.hasOwnProperty(filedName)){

o[filedName] = new ObjectId(o[filedName]);

return o;

}

return o;

})

}

# Db Services

This is the folder where you will be defining all the services that interact with MongoDB

##### Create a mongoService.js in dbServices folder.

Add the following code to mongoService.js

const ApiError = require('../helpers/apiError')

const mongoDB = require('../helpers/db')

const dbUtils = require('../helpers/dbUtils')

exports.findAll = (collectionName) => {

const collection = mongoDB.db.collection(collectionName);

return collection.find({}).toArray();

}

The [find()](https://docs.mongodb.com/manual/reference/method/db.collection.find/#db.collection.find) method with no parameters returns all documents from a collection and returns all fields for the documents.

Add the following code to mongoService.js

exports.findByConditions = (collectionName, conditions) => {

const collection = mongoDB.db.collection(collectionName);

return collection.find(conditions).toArray();

}

To find documents that match a set of selection criteria, call find() with the < conditions > parameter.

Add the following code to mongoService.js

exports.findOneAndUpdate = (collectionName, filter, update, options) => {

const collection = mongoDB.db.collection(collectionName);

return collection.findOneAndUpdate(filter, update, options);

}

[findOneAndUpdate()](https://docs.mongodb.com/manual/reference/method/db.collection.findOneAndUpdate/#db.collection.findOneAndUpdate) updates the first matching document in the collection that matches the filter.

Add the following code to mongoService.js

exports.insertMany = (collectionName, docs) => {

const collection = mongoDB.db.collection(collectionName);

return collection.insertMany(docs);

}

Given an array of documents, [insertMany()](https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/" \l "db.collection.insertMany" \o "db.collection.insertMany()) inserts each document in the array into the collection.

Add the following code to mongoService.js

exports.getById = (collectionName, id) => {

const collection = mongoDB.db.collection(collectionName);

return new Promise(function (resolve, reject) {

collection.findOne({ \_id: dbUtils.stringToObjectId(id) }, (err, doc) => {

if(err){

reject(err);

}

else{

if(doc === null){

reject(new ApiError('The server can not find requested resource', 404));

}

else{

resolve(doc);

}

}

});

})

}

exports.drop = (collectionName) => {

const collection = mongoDB.db.collection(collectionName);

return collection.drop();

}

##### Create a questionService.js in dbServices folder.

Add the following code to questionService.js

const mongoService = require('./mongoService')

const dbUtils = require('../helpers/dbUtils')

const collection = "questions";

exports.getByQuizId = (quizId) => {

return mongoService.findByConditions(collection, {quiz\_id: dbUtils.stringToObjectId(quizId)});

}

getByQuizId() – get all the questions where question.quiz\_id = quizId

Add the following code to questionService.js

exports.getById = (id) => {

return mongoService.getById(collection, id);

}

exports.create = (docs) => {

return mongoService.insertMany(collection, dbUtils.stringIDToObjectId(docs, 'quiz\_id'));

}

exports.all = () => {

return mongoService.findAll(collection);

}

exports.drop = () => {

return mongoService.drop(collection);

}

##### Create a quizService.js in dbServices folder.

Add the following code to quizService.js

const mongoService = require('./mongoService')

const dbUtils = require('../helpers/dbUtils')

const collection = "quizzes";

exports.all = () => {

return mongoService.findAll(collection);

}

exports.getById = (id) => {

return mongoService.getById(collection, id);

}

exports.updateQuizResults = (id, score) => {

const filter = { \_id: dbUtils.stringToObjectId(id) };

const update = { $inc: { aggregatedScore: score, gameCounter: 1 } };

return mongoService.findOneAndUpdate(collection, filter, update, {});

}

exports.create = (docs) => {

return mongoService.insertMany(collection, docs);

}

exports.drop = () => {

return mongoService.drop(collection);

}

# Models

Models are the files where you interact with your data.

They contain all the methods and functions which will handle your data.

This includes not only the methods for creating, reading, updating and deleting items, but also any additional business logic.

##### Create a question.js in models folder.

Add the following code to question.js

const questionService = require('../dbServices/questionService')

exports.getByQuizId = (quizId) => {

return questionService.getByQuizId(quizId);

}

exports.getById = (id) => {

return questionService.getById(id);

}

exports.create = (docs) => {

return questionService.create(docs);

}

##### Create a quiz.js in models folder.

Add the following code to quiz.js

const quizService = require('../dbServices/quizService')

exports.all = () => {

return quizService.all();

}

exports.getById = (id) => {

return quizService.getById(id);

}

exports.updateQuizResults = (id, score) => {

return quizService.updateQuizResults(id, score);

}

exports.create = (docs) => {

return quizService.create(docs);

}

# sample data

In order to start the server with some data we will create setup api that insert the sample data into mongoDB.

##### Create a data.json in sampleData folder.

Add the following data to data.json

[

{

"quiz": {

"name": "History",

"image": "https://i.imgur.com/BEZRovh.png"

},

"questions": [

{

"questionText": "When did Christopher Columbus arrive in the 'New World'?",

"quiz\_id": "5a4cd2aae7a4152314648676",

"answers": [

"1211",

"1492",

"1650",

"1962"

],

"correctAnswerIndex": 1

},

{

"questionText": "When did India declare its independence from Britain?",

"quiz\_id": "5a4cd2aae7a4152314648676",

"answers": [

"1947",

"1957",

"1967",

"1977"

],

"correctAnswerIndex": 0

}

]

},

{

"quiz": {

"name": "Music",

"image": "https://i.imgur.com/SWbgdNB.png"

},

"questions": [

{

"questionText": "John Lennon was part of which band?",

"quiz\_id": "5a4cd2c0e7a4152314648677",

"answers": [

"The Rolling Stones",

"The Monkeys",

"The Beatles",

"R.E.M."

],

"correctAnswerIndex": 2

},

{

"questionText": "Which band is the most commercially successful band in the history of popular music?",

"quiz\_id": "5a4cd2c0e7a4152314648677",

"answers": [

"Coldplay",

"The Monkeys",

"The Beatles",

"R.E.M."

],

"correctAnswerIndex": 2

},

{

"questionText": "When did The Rollings Stones band formed?",

"quiz\_id": "5a4cd2c0e7a4152314648677",

"answers": [

"1949",

"1962",

"1972",

"1980"

],

"correctAnswerIndex": 1

}

]

},

{

"quiz": {

"name": "Geography",

"image": "https://i.imgur.com/E0xbmeA.png"

},

"questions": [

{

"questionText": "Where is France?",

"quiz\_id": "5a4cee07e7a415231464867e",

"locationAnswer": {

"latitude": 47,

"longitude": 3.6

},

"errorMarginRadius": 5,

"mapSettings": {

"latitude": 30,

"longitude": 0,

"zoom": 2

}

},

{

"questionText": "Where is Tokyo?",

"quiz\_id": "5a4cee07e7a415231464867e",

"locationAnswer": {

"latitude": 35.43,

"longitude": 139.6

},

"errorMarginRadius": 15,

"mapSettings": {

"latitude": 0,

"longitude": 180,

"zoom": 1

}

}

]

},

{

"quiz": {

"name": "Sports"

},

"questions": [

{

"questionText": "In the Summer Olympic Games of 2016, who won the men's 100 meters run gold medal?",

"quiz\_id": "5a4e8f282fddcd0768fed70a",

"answers": [

"Kirani James",

"Usain Bolt",

"Andre De Grasse",

"Taoufik Makhloufi"

],

"correctAnswerIndex": 1

},

{

"questionText": "For which sport is Serena Williams famous for?",

"quiz\_id": "5a4e8f282fddcd0768fed70a",

"answers": [

"Tennis",

"Boxing",

"Skiing",

"Baseball"

],

"correctAnswerIndex": 0

}

]

},

{

"quiz": {

"name": "Nature",

"image": "https://i.imgur.com/ooSE2VK.png"

},

"questions": [

{

"questionText": "This is a...",

"questionImage": "https://i.imgur.com/ooSE2VK.png",

"quiz\_id": "5a5347eedca8bd4da0715362",

"answers": [

"lion",

"spider",

"flower",

"beetle"

],

"correctAnswerIndex": 3

}

]

},

{

"quiz" : {"name": "Movies", "image": "https://i.imgur.com/bbLkMuJ.png"}

}

]

##### Create a setup.js in models folder.

Add the following code to setup.js

const ApiError = require('../helpers/apiError')

const fs = require('fs');

const jsonData = JSON.parse(fs.readFileSync('./sampleData/data.json', 'utf8'));

const quizService = require('../dbServices/quizService')

const questionService = require('../dbServices/questionService')

let isInitStart = false;

insertQuizId = (questions, id) => {

return questions.map(q => {

q['quiz\_id'] = id;

return q;

})

}

getQuizIdByName = (quizzes, quizName) => {

for (let quiz of quizzes) {

if (quiz.name === quizName)

return quiz.\_id.toJSON();

}

return null;

}

isQuizExist = (quizzes, name) => {

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

if (quizzes[i].name === name)

return true;

}

return false;

}

isQuestionExist = (questions, questionText) => {

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

if (questions[i].questionText === questionText)

return true;

}

return false;

}

initDBData = (clearData) => {

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

try {

clearDbData(clearData)

.then(res => {

quizService.all()

.then(quizzesFromDb => {

const quizzesFromJson = jsonData.map(data => {

return data.quiz;

})

const filteredQuizzes = quizzesFromJson.filter(q => !isQuizExist(quizzesFromDb, q.name));

if (filteredQuizzes.length === 0) {

resolve('there aren\'t new quizzes to insert')

return;

}

quizService.create(filteredQuizzes)

.then(res => {

const insertedArray = res.ops;

questionService.all()

.then(questionsFromDb => {

let filteredQuestionsWithQuizId = [];

for (let quizData of jsonData) {

if (quizData.questions) {

let filteredQuestions = quizData.questions.filter(q => !isQuestionExist(questionsFromDb, q.questionText));

filteredQuestionsWithQuizId.push(...insertQuizId(filteredQuestions, getQuizIdByName([...insertedArray, ...quizzesFromDb], quizData.quiz.name)));

}

}

questionService.create(filteredQuestionsWithQuizId)

.then(res => {

resolve(res)

})

.catch(error => {

reject(error)

})

})

.catch(error => {

reject(error)

})

})

.catch(error => {

reject(error);

})

})

.catch(error => {

reject(error);

})

})

.catch(error => {

reject(error);

})

}

catch (e) {

reject(e);

}

}

)

}

clearDbData = (clearData) => {

if (clearData === 'true')

return Promise.all([quizService.drop(), questionService.drop()])

return Promise.resolve();

}

exports.init = (clearData) => {

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

if (isInitStart) {

reject('setup already started')

return;

}

isInitStart = true;

initDBData(clearData)

.then(res => {

resolve(res);

})

.catch(error => {

reject(error)

})

.then(() => {

isInitStart = false;

})

})

}

##### Create a setup.js in controllers folder.

Add the following code to setup.js

const controllerHandler = require('./controllerHandler')

const express = require('express')

, router = express.Router()

, Setup = require('../models/setup')

router.get('/', controllerHandler(Setup.init, (req, res, next) => [req.query.clear]))

module.exports = router

# mongodb

You can install mongoDB on your local machine or use mongo on a Docker container.

If you prefer to use mongo on docker container, jump to Dockerize your project section.

Install mongoDB from:

https://docs.mongodb.com/manual/administration/install-community/

# Running the server

Navigate to the project folder and type:

npm start

# Populate the db with sample data

Open the following url in your browser:

<http://localhost:3001/setup>

In case you want to clear the data first:

<http://localhost:3001/setup?clear=true>

# dockerize your project

Install Docker from - <https://docs.docker.com/engine/installation>

##### Docker file

##### Create a Dockerfile file at the root of your project.

Add the following commands to Dockerfile.

FROM node:carbon

RUN mkdir /app

WORKDIR /app

COPY package\*.json /app/

RUN npm install

COPY . /app

EXPOSE 3001

CMD ["npm", "start"]

##### docker compose

##### Create a docker-compose.yml file at the root of your project.

Add the following commands to docker-compose.

version: '3'

services:

node:

image: quiz-app

ports:

- "3001:3001"

links:

- mongo

mongo:

image: mongo

ports:

- "27017:27017"

##### Building your image

Navigate to /config/index.js –

Change

From url: 'mongodb://localhost:27017'

To url: 'mongodb://mongo:27017'

Go to the directory that has your Dockerfile and run the following command to build the Docker image.

The -t flag lets you tag your image so it's easier to find later using the docker images command.

Type:

docker build –t quiz-app .

##### Running the server

Go to the directory that has your docker-compose.yml and run the following command:

Type:

docker-compose up

# Node container and local mongodb

In case you want to run your node on docker container with local mongoDB

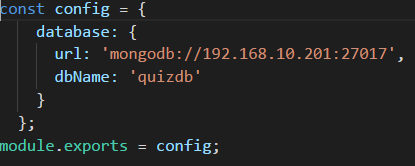
Create a Dockerfile file as described [here](#_Docker_file)

Build your image as described [here](#_Building_your_image)

The data base url in this case should be:

url: 'mongodb://Your\_Local\_IP:27017'

For example:



Run your local mongoDB from the mongoDB’s folder by typing:

mongod --bind\_ip localhost,Your\_Local\_IP

For example:



Run the node container by typing:

docker run -p 3001:3001 quiz-app

For example:

