Web Engineering Project Team 30

s4668324 & s4753690 & s4698649

Web Engineering 2022-23

1 Overview/Introduction

For this project, our team developed a RESTful Web App to present the 1921-2020 Spotify dataset statistics. We divided the work into 4 milestones (M1, M2, M3 and M4), which correspond to the steps that needed to be followed in order to build such a web app; API design and specification, API implementation (back-end implementation), User Interface with some functionality (front-end implementation) and the final deployable Web app with all the functionality accompanied with a report.

2 API Design

The first thing we needed to do was to create the API design (the specification) in YAML format in order to represent the design of our endpoints. In total we have 13 endpoints. Below we will explain each endpoint and its functionalities (the bold text represents the type of endpoint and how to access it):

1. **GET: /songs** Retrieves a list of selected songs using name of a song.
This endpoint will retrieve results when the user inputs the name of the song he/she wishes to retrieve and then will give the result in either JSON or CSV format. It will return the name of the song, id of the song, artist name(s), artist ID(s), popularity and release date. The errors are 204 (when the list is empty or when we have no results), 400 (when the request is not well formed e.g. missing input) and 5xx (server error).

2. **POST: /songs** Creates a new song.
This endpoint will create a song when the user inputs the song name, artist name(s), artist ID(s), popularity and release date and will then give the result in JSON format. It will return whether or not the query was successful. The errors are 400 (when the request is not well formed e.g. missing input), 409 (a duplicate song exists) and 5xx (server error).

3. **GET: /songs/id** Retrieves a song using song ID.
This endpoint will retrieve results when the user inputs the song ID and will then give the result in either JSON or CSV format. It will return the name of the song, id of the song, artist name(s), artist ID(s), popularity and release date. The errors are 404 (when the list is empty or when we have no results), 400 (when the request is not well formed e.g. missing input) and 5xx (server error).

4. **PUT: /songs/id** Updates a song using song ID.
This endpoint will update a song when the user inputs the song ID, the song name, artist name(s), artist ID(s), popularity and release date. All these fields will get updated depending on what the user inputs and then the result will be given. The updated song state returns in JSON format. The errors are 400 (when the request is not well formed e.g. missing input), 404 (song wasn't found) and 5xx (server error).

5. **DELETE: /songs/id** Deletes a song using song ID.
This endpoint will delete a song when the user inputs the song ID and will then give a confirmation in JSON format. It will return whether or not the query was successful and if it was, it will also return the deleted song's name and id. The errors are 400 (when the request is not well formed e.g. missing input), 404 (song wasn't found) and 5xx (server error).

6. **DELETE: /songs/artist** Deletes songs by artist name.
This endpoint will delete the songs of a specific artist when the user inputs the artist name and will then give a confirmation in JSON format. It will return whether or not the query was successful and if it was, it will also return the number of deleted songs. The errors are 400 (when the request is not well formed e.g. missing input),

404 (artist wasn't found) and 5xx (server error).

7. **DELETE: /songs/artist/id** Deletes songs by artist ID.
This endpoint will delete all songs of a specific artist when the user inputs the artist ID and will then give a confirmation in JSON format. It will return whether or not the query was successful and if it was, it will also return the number of deleted songs. The errors are 400 (when the request is not well formed e.g. missing input), 404 (artist wasn't found) and 5xx (server error).

8. **GET: /songs/top** Retrieves top **n** songs.
This endpoint will retrieve the top N songs when the user inputs the number of top songs he/she wishes to get and will then give the result in either JSON or CSV format. It will return all the songs in the top N. The errors are 400 (when the request is not well formed e.g. missing input), 404 (songs weren't found) and 5xx (server error).

9. **GET: /artists/summary** Retrieves summary of artist using artist name.
This endpoint will retrieve the summary of artists when the user inputs the artist name and will then give the result in either JSON or CSV format. It will return, for each artist found, the artist ID, artist name, number of songs, most popular song, earliest song and latest song. The errors are 400 (when the request is not well formed e.g. missing input), 404 (artist not found) and 5xx (server error).

10. **GET: /artists/summary/id** Retrieves summary of artist using artist ID.
This endpoint will retrieve the summary of an artist when the user inputs the artist ID and will then give the result in either JSON or CSV format. It will return the artist ID, artist name, number of songs, most popular song, earliest song and latest song. The errors are 400 (when the request is not well formed e.g. missing input), 404 (artist not found) and 5xx (server error).

11. **GET: /artists/songs** Retrieves songs of artist using artist name.
This endpoint will retrieve the songs of an artist when the user inputs the artist name and will then give the result in either JSON or CSV format. It will return, for each artist, the artist information, and each song's information. The errors are 400 (when the request is not well formed e.g. missing input), 404 (artist not found) and 5xx (server error).

12. **GET: /artists/songs/id** Retrieves songs of artist using artist ID.
This endpoint will retrieve the songs of an artist when the user inputs the artist ID and will then give the result in either JSON or CSV format. It will return the artist's and each song's information. The errors are 400 (when the request is not well formed e.g. missing input), 404 (artist not found) and 5xx (server error).

13. **GET: /artists/top** Retrieves top **n** artists.
This endpoint will retrieve the top N artists when the user inputs the number of top artists he/she wants and will then give the result in either JSON or CSV format. The top N artists along with their information will then be returned. The errors are 400 (when the request is not well formed e.g. missing input), 404 (songs weren't found) and 5xx (server error).

3 API Implementation

For the API implementation, we were originally going to use Java and Springboot, however, we came across some difficulties, especially when trying to connect to an external database, and so we decided to switch to the MERN stack.
The MERN stack is a software stack, a collection of technologies, that is being used for a fast application development for dynamic web sites and web applications. It consists of 4 components: MongoDB used as an external online database, Express.js used as a web application framework, React.js used for building the user interface and Node.js used for server-side programming. This meant that we would be required to use Javascript for the back-end and the front-end framework instead of Java (our original plan).
In order to start creating the API, we needed to connect to a port and to a database. Since we are using MongoDB, we had to create a database that we can connect to over the internet. After doing that, we added the link needed to connect to it in our .env file.

```
NODE_ENV=development
DATABASE_URI=mongodb+srv://mongo:mongo@cluster0.uveog9h.mongodb.net/?retryWrites=true&w=majority
```

Afterwards, we needed to connect to a port and connect all the routes (done after creating routes which will be shown further below).

server.js

```
require('dotenv').config()
const express = require('express')
const app = express()
const path = require('path')
const {logger} = require('./middleware/logger')
const errorHandler = require('./middleware/errorHandler')
const cookieParser = require('cookie-parser')
const cors = require('cors')
const corsOptions = require('./config/corsOptions')
const connectDB = require('./config/dbConn')
const mongoose = require('mongoose')
const { logEvents } = require('./middleware/logger')
const PORT = process.env.PORT || 3500

console.log(process.env.NODE_ENV)

connectDB()

app.use(logger)

app.use(cors(corsOptions))

app.use(express.json())

app.use(cookieParser())

app.use('/', express.static(path.join(__dirname, 'public')))

app.use('/', require('./routes/root'))
app.use('/songs', require('./routes/songRoutes'))
app.use('/artists', require('./routes/artistRoutes'))

app.all('*', (req, res) => {
    res.status(404)
    if (req.accepts('html')) {
        res.sendFile(path.join(__dirname, 'views', '404.html'))
    } else if (req.accepts('json')) {
        res.json({message: '404 Not Found'})
    } else {
        res.type('txt').send('404 Not Found')
    }
})

app.use(errorHandler)

mongoose.connection.once('open', () => {
    console.log('Connected to MongoDB')
    app.listen(PORT, () => console.log('Server running on port ${
       PORT}'))
})

mongoose.connection.on('error', err => {
    console.log(err)
    logEvents(`${err.no}: ${err.code}\t${err.syscall}\t${err.
       hostname}`,
    'mongoErrLog.log')
})
```

We then had to create models (for artists and songs) so we could create objects and store them in the database. The use of mongoose.Schema means each object created would have its own personal ID, so the use of ID from the songs and artists data would be unnecessary.

Artist.js

```
const mongoose = require('mongoose')
const Song = require('../models/Song')

const artistSchema = new mongoose.Schema({
    name: {
       type: String,
       required: true
    },
    popularity: {
        type: Number,
        required: true
    }
})

module.exports = mongoose.model('Artist', artistSchema)
```

Song.js

```
const mongoose = require('mongoose')
const Artist = require('../models/Artist')

const songSchema = new mongoose.Schema(
    {
     name: {
        type: String,
        required: true
     },
     popularity: {
        type: Number,
        required: true
     },
     artists: [{
        type: String,
        required: true,
        ref: 'Artist'
     }],
     artistIDs : [{
            type: mongoose.Schema.Types.ObjectId,
            required: true,
            ref: 'Artist'
        }],
        date: {
            type: Date,
            required: true
        }
    }
)


module.exports = mongoose.model('Song', songSchema)
```

We then created the controllers for the artists and songs, which are responsible for carrying out all the endpoints as necessary. For the GET requests the outputs are either in JSON or CSV format, specified by the response type.

artistsController.js

```
const Artist = require('../models/Artist')
const Song = require('../models/Song')
const asyncHandler = require('express-async-handler')
const { areIntervalsOverlappingWithOptions } = require('date-fns/
   fp')

//converts JSON to CSV representation
function jsonToCsv(items) {
    const header = Object.keys(items[0]);
    const headerString = header.join(',');
    const replacer = (key, value) => value ?? '';
    const rowItems = items.map((row) =>
      header
        .map((fieldName) => JSON.stringify(row[fieldName],
           replacer))
        .join(',')
    );
    const csv = [headerString, ...rowItems].join('\r\n');
    return csv;
}

const getAllArtists = asyncHandler( async (req, res) => {
    const artists = await Artist.find().lean()
    if (!artists?.length) {
        return res.status(400).json({message: 'No artists found'
           })
    }

    const paramsString = req.url.split('?')[1];
    const eachParamArray = paramsString.split('&');
    const contentType = await (eachParamArray[0]).split('=')[1]

    if (contentType == 'application/json') {
        res.json(artists)
    } else {
        res.send(jsonToCsv(artists))
    }
})

//POST: create artist
const createArtist = asyncHandler(async (req, res) => {
    const { name, popularity } = req.body
    if (!name || !popularity) {
        return res.status(400).json({ message: 'All fields are required'})
    }
    const duplicate = await Artist.findOne({name}).lean()
    if (duplicate) {
        return res.status(409).json({message: 'Duplicate artist'})
    }

    const artist = await Artist.create({name, popularity})
    if (artist) {
        return res.status(201).json({ message: 'New artist created'})
    } else {
        return res.status(400).json({ message: 'Invalid artist data received'})
    }
})

//GET: summary (name)
const getSummaryName = asyncHandler(async (req, res) => {
    const { name } = req.body
    if (!name) {
        return res.status(400).json({ message: 'All fields are required'})
    }
    const artists = await Artist.find({name}).lean()
    if (!artists?.length) {
        return res.status(400).json({message: 'No artists found'})
    }

    const summaryList = []
    for (var i = 0; i < artists.length; i++) {
        const id = artists[i]._id
        const sortedDates = await Song.find({artistIDs: id }).sort({date: -1}).lean()
        const artistName =  artists[i].name
        const popularity = artists[i].popularity
        const numOfSongs = (await Song.find({artistIDs: id }).exec()).length
        const earliest = sortedDates[sortedDates.length - 1].date
        const latest = sortedDates[0].date
        const mostPopularSong = await Song.find({artistIDs: id}).
           sort({popularity: -1}).lean()
        const mostPopular = mostPopularSong[0].name

        const summary = {
            "id": id,
            "name": artistName,
            "popularity": popularity,
            "numOfSongs": numOfSongs,
            "eariest": earliest,
            "latest": latest,
            "mostPopular": mostPopular
        }

        summaryList.push(summary)
    }

    const paramsString = req.url.split('?')[1];
    const eachParamArray = paramsString.split('&');
    const contentType = await (eachParamArray[0]).split('=')[1]

    if (contentType == 'application/json') {
        res.json(summaryList)
    } else {
        res.send(jsonToCsv(summaryList))
    }
})

//GET: summary (id)
const getSummaryID = asyncHandler (async (req, res) => {
    const { id } = req.body
    if (!id) {
        return res.status(400).json({ message: 'All fields are
           required'})
    }
    const artist = await Artist.findById(id).lean()
    if (!artist) {
        return res.status(400).json({message: 'No artists found'
           })
    }

    const sortedDates = await Song.find({artistIDs: id }).sort({
       date: -1}).lean()
    const artistName =  artist.name
    const popularity = artist.popularity
    const numOfSongs = (await Song.find({artistIDs: id }).exec())
       .length
    const earliest = sortedDates[sortedDates.length - 1].date
    const latest = sortedDates[0].date
    const mostPopularSong = await Song.find({artistIDs: id}).sort
       ({popularity: -1}).lean()
    const mostPopular = mostPopularSong[0].name

    const summary = {
        "id": id,
        "name": artistName,
        "popularity": popularity,
        "numOfSongs": numOfSongs,
        "eariest": earliest,
        "latest": latest,
        "mostPopular": mostPopular
    }

    const paramsString = req.url.split('?')[1];
    const eachParamArray = paramsString.split('&');
    const contentType = await (eachParamArray[0]).split('=')[1]

    if (contentType == 'application/json') {
        res.json(summary)
    } else {
        res.send(jsonToCsv(summary))
    }
})

//GET: songs of artist (name)
const getSongsName = asyncHandler (async (req, res) => {
    const { name } = req.body
    if (!name) {
        return res.status(400).json({ message: 'All fields are
           required'})
    }
    const songs = await Song.find({artists: name}).lean().exec()
    if (!songs) {
        return res.status(400).json({message: 'No songs found'})
    }

    const paramsString = req.url.split('?')[1];
    const eachParamArray = paramsString.split('&');
    const contentType = await (eachParamArray[0]).split('=')[1]

    if (contentType == 'application/json') {
        res.json(songs)
    } else {
        res.send(jsonToCsv(songs))
    }
})

//GET: songs of artist (ID)
const getSongsID = asyncHandler (async (req, res) => {
    const { id } = req.body
    if (!id) {
        return res.status(400).json({ message: 'All fields are
           required'})
    }
    const songs = await Song.find({artistIDs: id}).lean().exec()
    if (!songs) {
        return res.status(400).json({message: 'No songs found'})
    }

    const paramsString = req.url.split('?')[1];
    const eachParamArray = paramsString.split('&');
    const contentType = await (eachParamArray[0]).split('=')[1]

    if (contentType == 'application/json') {
        res.json(songs)
    } else {
        res.send(jsonToCsv(songs))
    }

})

//GET: top N artists
const getTopArtists = asyncHandler(async (req, res) => {
    const { n } = req.body
    if (!n) {
        return res.status(400).json({message: 'All fields are
           required'})
    }
    const artists = await Artist.find().sort({popularity: -1}).
       limit(n).lean()

    if(!artists?.length) {
        return res.status(400).json({ message: 'No artists found'
           })
    }

    const paramsString = req.url.split('?')[1];
    const eachParamArray = paramsString.split('&');
    const contentType = await (eachParamArray[0]).split('=')[1]

    if (contentType == 'application/json') {
        res.json(artists)
    } else {
        res.send(jsonToCsv(artists))
    }
})

module.exports = {
    createArtist,
    getAllArtists,
    getSummaryName,
    getSummaryID,
    getSongsName,
    getSongsID,
    getTopArtists
```

```
}
```

songsController.js

```
const Song = require('../models/Song')
const Artist = require('../models/Artist')
const asyncHandler = require('express-async-handler')

//converts JSON to CSV representation
function jsonToCsv(items) {
    const header = Object.keys(items[0]);
    const headerString = header.join(',');
    const replacer = (key, value) => value ?? '';
    const rowItems = items.map((row) =>
      header
        .map((fieldName) => JSON.stringify(row[fieldName],
           replacer))
        .join(',')
    );
    const csv = [headerString, ...rowItems].join('\r\n');
    return csv;
}

const getAllSongs = asyncHandler(async (req, res) => {
    const songs = await Song.find().lean()
    if (!songs?.length) {
        return res.status(400).json({message: 'No songs found'})
    }

    const paramsString = req.url.split('?')[1];
    const eachParamArray = paramsString.split('&');
    const contentType = await (eachParamArray[0]).split('=')[1]

    if (contentType == 'application/json') {
        res.json(songs)
    } else {
        res.send(jsonToCsv(songs))
    }
})

//GET: songs by name
const getSongs = asyncHandler(async (req, res) => {
    const { name } = req.body

    if (!name) {
        return res.status(400).json({message: 'All fields are
           required'})
    }

    const songs = await Song.find({ name }).lean()

    if (!songs?.length) {
        return res.status(400).json({ message: 'No songs found'})
    }

    const paramsString = req.url.split('?')[1];
    const eachParamArray = paramsString.split('&');
    const contentType = await (eachParamArray[0]).split('=')[1]

    if (contentType == 'application/json') {
        res.json(songs)
    } else {
        res.send(jsonToCsv(songs))
    }
})

//GET: songs by ID
const getSong = asyncHandler(async (req, res) => {
    const { id } = req.body

    if (!id) {
        return res.status(400).json({message: 'All fields are
           required'})
    }

    const song = await Song.findById( id ).lean()

    if (!song) {
        return res.status(400).json({ message: 'No songs found'})
    }

    const paramsString = req.url.split('?')[1];
    const eachParamArray = paramsString.split('&');
    const contentType = await (eachParamArray[0]).split('=')[1]

    if (contentType == 'application/json') {
        res.json(song)
    } else {
        res.send(jsonToCsv(song))
    }
})

//GET: top songs
const getTopSongs = asyncHandler(async (req, res) => {
    const { n } = req.body
    if (!n) {
        return res.status(400).json({message: 'All fields are
           required'})
    }

    const songs = await Song.find().sort({popularity: -1}). limit
       (n).lean()

    if (!songs?.length) {
        return res.status(400).json({ message: 'No songs found'})
    }

    const paramsString = req.url.split('?')[1];
    const eachParamArray = paramsString.split('&');
    const contentType = await (eachParamArray[0]).split('=')[1]

    if (contentType == 'application/json') {
        res.json(songs)
    } else {
        res.send(jsonToCsv(songs))
    }
})

//POST: create song
const createNewSong = asyncHandler(async (req, res) => {
    const { name, popularity, artists, artistsID, date } = req.body
    if (!name || !popularity || !artists?.length || !artistsID?.length || !date) {
        return res.status(400).json({ message: 'All fields are required'})
    }

    const duplicate = await Song.findOne({ name }, { artists }).lean().exec()
    if (duplicate) {
        return res.status(409).json({message: 'Duplicate song'})
    }

    const song = await Song.create({ name, popularity, artists, date})
    if (song) {
        song.artistIDs = artistsID
        const updatedSong = await song.save()
        return res.status(201).json({ message: 'New song created'})
    } else {
        return res.status(400).json({ message: 'Invalid song data received'})
    }
})

//PATCH: update song
const updateSong = asyncHandler(async (req, res) => {
    const { id, name, popularity, artists, artistsID, date } = req.body
    if (!id || !name || !popularity || !artists?.length || !artistsID?.length || !date) {
        return res.status(400).json({ message: 'All fields are required'})
    }

    const song = await Song.findById(id).exec()
    if (!song) {
        return res.status(400).json({ message: 'Song not found'})
    }

    const duplicate = await Song.findOne({name}, {artists}).lean
        ().exec()
    if (duplicate && duplicate?._id.toString() !== id) {
        return res.status(409).json({ message: 'Duplicate song'})
    }

    song.name = name
    song.popularity = popularity
    song.artists = artists
    song.artistIDs = artistsID
    song.date = date

    const updatedSong = await song.save()

    res.json(`'${updatedSong.name}' updated`)
})

//DELETE: delete song by ID
const deleteSong = asyncHandler(async (req, res) => {
    const { id } = req.body
    if (!id) {
        return res.status(400).json({ message: 'Song ID required'
            })
    }

    const song = await Song.findById(id).exec()
    if (!song) {
        return res.status(400).json({message : 'Song not found'})
    }

    const result = await song.deleteOne()
    const reply = `Song '${result.name}' with ID ${result._id}
        deleted`
    res.json(reply)
})

//DELETE: delete songs by artist name
const deleteSongName = asyncHandler(async (req, res) => {
    const { artists } = req.body
    if (!artists) {
        return res.status(400).json({message: 'Artist name
            required'})
    }

    const songs = await Song.find({ artists }).exec()
    if (!songs?.length) {
        return res.status(400).json({message: 'No songs found to
        delete'})
    }
    const result = await Song.deleteMany({artists})
    const reply = `Deleted ${result.deletedCount} songs`
    res.json(reply)
})

//DELETE: delete songs by artist ID
const deleteSongID = asyncHandler(async (req, res) => {
    const { id } = req.body
    if (!id) {
        return res.status(400).json({message: 'Artist ID required
            '})
    }

    const songs = await Song.find({artistIDs: id}).exec()
    if (!songs?.length) {
        return res.status(400).json({message: 'No songs found to
            delete'})
    }
    const result = await Song.deleteMany({artistIDs: id})
    const reply = `Deleted ${result.deletedCount} songs`
    res.json(reply)
})


module.exports = {
    getAllSongs,
    getSongs,
    getSong,
    getTopSongs,
    createNewSong,
    updateSong,
    deleteSong,
    deleteSongName,
    deleteSongID
}
```

Next, we had to create the routes for the endpoints, how they are going to be accessed from the URL, as well as which route corresponds to which controller function.

artistRoutes.js

```
const { Router } = require('express')
const express = require('express')
const router = express.Router()
const artistsController = require('../controllers/
   artistsController')

router.route('/')
    .post(artistsController.createArtist)
    .get(artistsController.getAllArtists)

```

```
router.route('/summary')
    .get(artistsController.getSummaryName)

router.route('/summary/id')
    .get(artistsController.getSummaryID)

router.route('/songs')
    .get(artistsController.getSongsName)

router.route('/songs/id')
    .get(artistsController.getSongsID)

router.route('/top')
    .get(artistsController.getTopArtists)

module.exports = router
```

songRoutes.js

```
const express = require('express')
const router = express.Router()
const songsController = require('../controllers/songsController')

router.route('/all')
    .get(songsController.getAllSongs)

router.route('/')
    .get(songsController.getSongs)
    .post(songsController.createNewSong)


router.route('/id')
    .get(songsController.getSong)
    .patch(songsController.updateSong)
    .delete(songsController.deleteSong)

router.route('/top')
    .get(songsController.getTopSongs)

router.route('/artist')
    .delete(songsController.deleteSongName)

router.route('/artist/id')
    .delete(songsController.deleteSongID)

module.exports = router
```

After putting everything together in the server.js file, we were able to run it on port: 3500.

4 Running

1 Navigate to the **2022-Group-30** directory

2 Build and run the project by typing in the terminal:

```
npm run dev
npm start
```

3 Open a web browser and go to: `http://localhost:3500`