

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

```
1  require('dotenv').config()
2  const express = require('express')
3  const app = express()
4  const path = require('path')
5  const {logger} = require('./middleware/logger')
6  const errorHandler = require('./middleware/errorHandler')
7  const cookieParser = require('cookie-parser')
8  const cors = require('cors')
9  const corsOptions = require('./config/corsOptions')
10 const connectDB = require('./config/dbConn')
11 const mongoose = require('mongoose')
12 const { logEvents } = require('./middleware/logger')
13 const PORT = process.env.PORT || 3500
14
15 console.log(process.env.NODE_ENV)
16
17 connectDB()
18
19 app.use(logger)
20
21 app.use(cors(corsOptions))
22
23 app.use(express.json())
24
25 app.use(cookieParser())
26
27 app.use('/', express.static(path.join(__dirname, 'public')))
28
29 app.use('/', require('./routes/root'))
30 app.use('/songs', require('./routes/songRoutes'))
31 app.use('/artists', require('./routes/artistRoutes'))
32
33 app.all('*', (req, res) => {
34     res.status(404)
35     if (req.accepts('html')) {
36         res.sendFile(path.join(__dirname, 'views', '404.html'))
37     } else if (req.accepts('json')) {
38         res.json({message: '404 Not Found'})
39     } else {
40         res.type('txt').send('404 Not Found')
41     }
42 })
43
44 app.use(errorHandler)
45
46 mongoose.connection.once('open', () => {
47     console.log('Connected to MongoDB')
48     app.listen(PORT, () => console.log('Server running on port ${
```

```

    PORT}'))
49 })
50
51 mongoose.connection.on('error', err => {
52   console.log(err)
53   logEvents(`${err.no}: ${err.code}\t${err.syscall}\t${err.
     hostname}`,
54   'mongoErrLog.log')
55 })

```

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

```

1  const mongoose = require('mongoose')
2  const Song = require('../models/Song')
3
4  const artistSchema = new mongoose.Schema({
5    name: {
6      type: String,
7      required: true
8    },
9    popularity: {
10     type: Number,
11     required: true
12   }
13 })
14
15 module.exports = mongoose.model('Artist', artistSchema)

```

Song.js

```

1  const mongoose = require('mongoose')
2  const Artist = require('../models/Artist')
3
4  const songSchema = new mongoose.Schema(
5    {
6     name: {
7       type: String,
8       required: true
9     },
10    popularity: {
11      type: Number,
12      required: true
13    },
14    artists: [{
15      type: String,
16      required: true,
17      ref: 'Artist'
18    }],
19    artistIDs : [{

```

```

20     type: mongoose.Schema.Types.ObjectId,
21     required: true,
22     ref: 'Artist'
23   }],
24   date: {
25     type: Date,
26     required: true
27   }
28 }
29 )
30
31
32 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

```

1  const Artist = require('../models/Artist')
2  const Song = require('../models/Song')
3  const asyncHandler = require('express-async-handler')
4  const { areIntervalsOverlappingWithOptions } = require('date-fns/
    fp')
5
6  //converts JSON to CSV representation
7  function jsonToCsv(items) {
8     const header = Object.keys(items[0]);
9     const headerString = header.join(',');
10    const replacer = (key, value) => value ?? '';
11    const rowItems = items.map((row) =>
12      header
13        .map((fieldName) => JSON.stringify(row[fieldName],
14          replacer))
15        .join(',')
16    );
17    const csv = [headerString, ...rowItems].join('\r\n');
18    return csv;
19  }
20
21  const getAllArtists = asyncHandler( async (req, res) => {
22    const artists = await Artist.find().lean()
23    if (!artists?.length) {
24      return res.status(400).json({message: 'No artists found'})
25    }
26
27    const paramsString = req.url.split('?')[1];
28    const eachParamArray = paramsString.split('&');
29    const contentType = await (eachParamArray[0]).split('=')[1]
30
31    if (contentType == 'application/json') {

```

```

31     res.json(artists)
32   } else {
33     res.send(jsonToCsv(artists))
34   }
35 })
36
37 //POST: create artist
38 const createArtist = asyncHandler(async (req, res) => {
39   const { name, popularity } = req.body
40   if (!name || !popularity) {
41     return res.status(400).json({ message: 'All fields are
42       required'})
43   }
44   const duplicate = await Artist.findOne({name}).lean()
45   if (duplicate) {
46     return res.status(409).json({message: 'Duplicate artist'
47       })
48   }
49   const artist = await Artist.create({name, popularity})
50   if (artist) {
51     return res.status(201).json({ message: 'New artist
52       created'})
53   } else {
54     return res.status(400).json({ message: 'Invalid artist
55       data received'})
56   }
57 })
58
59 //GET: summary (name)
60 const getSummaryName = asyncHandler(async (req, res) => {
61   const { name } = req.body
62   if (!name) {
63     return res.status(400).json({ message: 'All fields are
64       required'})
65   }
66   const artists = await Artist.find({name}).lean()
67   if (!artists?.length) {
68     return res.status(400).json({message: 'No artists found'
69       })
70   }
71   const summaryList = []
72   for (var i = 0; i < artists.length; i++) {
73     const id = artists[i]._id
74     const sortedDates = await Song.find({artistIDs: id }).
75       sort({date: -1}).lean()
76     const artistName = artists[i].name
77     const popularity = artists[i].popularity
78     const numOfSongs = (await Song.find({artistIDs: id }).
79       exec()).length
80     const earliest = sortedDates[sortedDates.length - 1].date

```

```

75     const latest = sortedDates[0].date
76     const mostPopularSong = await Song.find({artistIDs: id}).
      sort({popularity: -1}).lean()
77     const mostPopular = mostPopularSong[0].name
78
79     const summary = {
80       "id": id,
81       "name": artistName,
82       "popularity": popularity,
83       "numOfSongs": numOfSongs,
84       "earliest": earliest,
85       "latest": latest,
86       "mostPopular": mostPopular
87     }
88
89     summaryList.push(summary)
90   }
91
92   const paramsString = req.url.split('?')[1];
93   const eachParamArray = paramsString.split('&');
94   const contentType = await (eachParamArray[0]).split('=')[1]
95
96   if (contentType == 'application/json') {
97     res.json(summaryList)
98   } else {
99     res.send(jsonToCsv(summaryList))
100   }
101 })
102
103 //GET: summary (id)
104 const getSummaryID = asyncHandler (async (req, res) => {
105   const { id } = req.body
106   if (!id) {
107     return res.status(400).json({ message: 'All fields are
      required'})
108   }
109   const artist = await Artist.findById(id).lean()
110   if (!artist) {
111     return res.status(400).json({message: 'No artists found'
      })
112   }
113
114   const sortedDates = await Song.find({artistIDs: id }).sort({
      date: -1}).lean()
115   const artistName = artist.name
116   const popularity = artist.popularity
117   const numOfSongs = (await Song.find({artistIDs: id }).exec())
      .length
118   const earliest = sortedDates[sortedDates.length - 1].date
119   const latest = sortedDates[0].date
120   const mostPopularSong = await Song.find({artistIDs: id}).sort
      ({popularity: -1}).lean()

```

```

121     const mostPopular = mostPopularSong[0].name
122
123     const summary = {
124         "id": id,
125         "name": artistName,
126         "popularity": popularity,
127         "numOfSongs": numOfSongs,
128         "earliest": earliest,
129         "latest": latest,
130         "mostPopular": mostPopular
131     }
132
133     const paramsString = req.url.split('?')[1];
134     const eachParamArray = paramsString.split('&');
135     const contentType = await (eachParamArray[0]).split('=')[1]
136
137     if (contentType == 'application/json') {
138         res.json(summary)
139     } else {
140         res.send(jsonToCsv(summary))
141     }
142 })
143
144 //GET: songs of artist (name)
145 const getSongsName = asyncHandler (async (req, res) => {
146     const { name } = req.body
147     if (!name) {
148         return res.status(400).json({ message: 'All fields are
149             required'})
150     }
151     const songs = await Song.find({artists: name}).lean().exec()
152     if (!songs) {
153         return res.status(400).json({message: 'No songs found'})
154     }
155
156     const paramsString = req.url.split('?')[1];
157     const eachParamArray = paramsString.split('&');
158     const contentType = await (eachParamArray[0]).split('=')[1]
159
160     if (contentType == 'application/json') {
161         res.json(songs)
162     } else {
163         res.send(jsonToCsv(songs))
164     }
165 })
166
167 //GET: songs of artist (ID)
168 const getSongsID = asyncHandler (async (req, res) => {
169     const { id } = req.body
170     if (!id) {
171         return res.status(400).json({ message: 'All fields are
172             required'})

```



```

171     }
172     const songs = await Song.find({artistIDs: id}).lean().exec()
173     if (!songs) {
174         return res.status(400).json({message: 'No songs found'})
175     }
176
177     const paramsString = req.url.split('?')[1];
178     const eachParamArray = paramsString.split('&');
179     const contentType = await (eachParamArray[0]).split('=')[1]
180
181     if (contentType == 'application/json') {
182         res.json(songs)
183     } else {
184         res.send(jsonToCsv(songs))
185     }
186
187 })
188
189 //GET: top N artists
190 const getTopArtists = asyncHandler(async (req, res) => {
191     const { n } = req.body
192     if (!n) {
193         return res.status(400).json({message: 'All fields are
194             required'})
195     }
196     const artists = await Artist.find().sort({popularity: -1}).
197         limit(n).lean()
198
199     if(!artists?.length) {
200         return res.status(400).json({ message: 'No artists found'
201             })
202     }
203
204     const paramsString = req.url.split('?')[1];
205     const eachParamArray = paramsString.split('&');
206     const contentType = await (eachParamArray[0]).split('=')[1]
207
208     if (contentType == 'application/json') {
209         res.json(artists)
210     } else {
211         res.send(jsonToCsv(artists))
212     }
213 })
214
215 module.exports = {
216     createArtist,
217     getAllArtists,
218     getSummaryName,
219     getSummaryID,
220     getSongsName,
221     getSongsID,
222     getTopArtists

```

songsController.js

```

1  const Song = require('../models/Song')
2  const Artist = require('../models/Artist')
3  const asyncHandler = require('express-async-handler')
4
5  //converts JSON to CSV representation
6  function jsonToCsv(items) {
7      const header = Object.keys(items[0]);
8      const headerString = header.join(',');
9      const replacer = (key, value) => value ?? '';
10     const rowItems = items.map((row) =>
11         header
12             .map((fieldName) => JSON.stringify(row[fieldName],
13                 replacer))
14             .join(',')
15     );
16     const csv = [headerString, ...rowItems].join('\r\n');
17     return csv;
18 }
19
20 const getAllSongs = asyncHandler(async (req, res) => {
21     const songs = await Song.find().lean()
22     if (!songs?.length) {
23         return res.status(400).json({message: 'No songs found'})
24     }
25
26     const paramsString = req.url.split('?')[1];
27     const eachParamArray = paramsString.split('&');
28     const contentType = await (eachParamArray[0]).split('=')[1]
29
30     if (contentType == 'application/json') {
31         res.json(songs)
32     } else {
33         res.send(jsonToCsv(songs))
34     }
35 })
36
37 //GET: songs by name
38 const getSongs = asyncHandler(async (req, res) => {
39     const { name } = req.body
40
41     if (!name) {
42         return res.status(400).json({message: 'All fields are
43             required'})
44     }
45
46     const songs = await Song.find({ name }).lean()
47
48     if (!songs?.length) {
49         return res.status(400).json({ message: 'No songs found'})
50     }
51 })

```

```

48     }
49
50     const paramsString = req.url.split('?')[1];
51     const eachParamArray = paramsString.split('&');
52     const contentType = await (eachParamArray[0]).split('=')[1]
53
54     if (contentType == 'application/json') {
55         res.json(songs)
56     } else {
57         res.send(jsonToCsv(songs))
58     }
59 })
60
61 //GET: songs by ID
62 const getSong = asyncHandler(async (req, res) => {
63     const { id } = req.body
64
65     if (!id) {
66         return res.status(400).json({message: 'All fields are
67             required'})
68     }
69
70     const song = await Song.findById( id ).lean()
71
72     if (!song) {
73         return res.status(400).json({ message: 'No songs found'})
74     }
75
76     const paramsString = req.url.split('?')[1];
77     const eachParamArray = paramsString.split('&');
78     const contentType = await (eachParamArray[0]).split('=')[1]
79
80     if (contentType == 'application/json') {
81         res.json(song)
82     } else {
83         res.send(jsonToCsv(song))
84     }
85 })
86
87 //GET: top songs
88 const getTopSongs = asyncHandler(async (req, res) => {
89     const { n } = req.body
90     if (!n) {
91         return res.status(400).json({message: 'All fields are
92             required'})
93     }
94
95     const songs = await Song.find().sort({popularity: -1}).limit
96         (n).lean()
97
98     if (!songs?.length) {
99         return res.status(400).json({ message: 'No songs found'})
100     }

```

```

97     }
98
99     const paramsString = req.url.split('?')[1];
100    const eachParamArray = paramsString.split('&');
101    const contentType = await (eachParamArray[0]).split('=')[1]
102
103    if (contentType == 'application/json') {
104        res.json(songs)
105    } else {
106        res.send(jsonToCsv(songs))
107    }
108 })
109
110 //POST: create song
111 const createNewSong = asyncHandler(async (req, res) => {
112     const { name, popularity, artists, artistsID, date } = req.
        body
113     if (!name || !popularity || !artists?.length || !artistsID?.
        length || !date) {
114         return res.status(400).json({ message: 'All fields are
            required'})
115     }
116
117     const duplicate = await Song.findOne({ name }, { artists }).
        lean().exec()
118     if (duplicate) {
119         return res.status(409).json({message: 'Duplicate song'})
120     }
121
122     const song = await Song.create({ name, popularity, artists,
        date})
123     if (song) {
124         song.artistIDs = artistsID
125         const updatedSong = await song.save()
126         return res.status(201).json({ message: 'New song created'
            })
127     } else {
128         return res.status(400).json({ message: 'Invalid song data
            received'})
129     }
130 })
131
132 //PATCH: update song
133 const updateSong = asyncHandler(async (req, res) => {
134     const { id, name, popularity, artists, artistsID, date } =
        req.body
135     if (!id || !name || !popularity || !artists?.length || !
        artistsID?.length || !date) {
136         return res.status(400).json({ message: 'All fields are
            required'})
137     }
138

```

```

139     const song = await Song.findById(id).exec()
140     if (!song) {
141         return res.status(400).json({ message: 'Song not found'})
142     }
143
144     const duplicate = await Song.findOne({name}, {artists}).lean
145         ().exec()
146     if (duplicate && duplicate?._id.toString() !== id) {
147         return res.status(409).json({ message: 'Duplicate song'})
148     }
149
150     song.name = name
151     song.popularity = popularity
152     song.artists = artists
153     song.artistIDs = artistsID
154     song.date = date
155
156     const updatedSong = await song.save()
157
158     res.json(`'${updatedSong.name}' updated`)
159 })
160
161 //DELETE: delete song by ID
162 const deleteSong = asyncHandler(async (req, res) => {
163     const { id } = req.body
164     if (!id) {
165         return res.status(400).json({ message: 'Song ID required'
166             })
167     }
168
169     const song = await Song.findById(id).exec()
170     if (!song) {
171         return res.status(400).json({message : 'Song not found'})
172     }
173
174     const result = await song.deleteOne()
175     const reply = `Song '${result.name}' with ID ${result._id}
176         deleted`
177     res.json(reply)
178 })
179
180 //DELETE: delete songs by artist name
181 const deleteSongName = asyncHandler(async (req, res) => {
182     const { artists } = req.body
183     if (!artists) {
184         return res.status(400).json({message: 'Artist name
185             required'})
186     }
187
188     const songs = await Song.find({ artists }).exec()
189     if (!songs?.length) {
190         return res.status(400).json({message: 'No songs found to

```

```

        delete'})
187     }
188     const result = await Song.deleteMany({artists})
189     const reply = 'Deleted ${result.deletedCount} songs '
190     res.json(reply)
191 })
192
193 //DELETE: delete songs by artist ID
194 const deleteSongID = asyncHandler(async (req, res) => {
195     const { id } = req.body
196     if (!id) {
197         return res.status(400).json({message: 'Artist ID required'})
198     }
199
200     const songs = await Song.find({artistIDs: id}).exec()
201     if (!songs?.length) {
202         return res.status(400).json({message: 'No songs found to delete'})
203     }
204     const result = await Song.deleteMany({artistIDs: id})
205     const reply = 'Deleted ${result.deletedCount} songs '
206     res.json(reply)
207 })
208
209
210 module.exports = {
211     getAllSongs,
212     getSongs,
213     getSong,
214     getTopSongs,
215     createNewSong,
216     updateSong,
217     deleteSong,
218     deleteSongName,
219     deleteSongID
220 }

```

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

```

1  const { Router } = require('express')
2  const express = require('express')
3  const router = express.Router()
4  const artistsController = require('../controllers/
    artistsController')
5
6  router.route('/')
7      .post(artistsController.createArtist)
8      .get(artistsController.getAllArtists)
9

```

```
10 router.route('/summary')
11     .get(artistsController.getSummaryName)
12
13 router.route('/summary/id')
14     .get(artistsController.getSummaryID)
15
16 router.route('/songs')
17     .get(artistsController.getSongsName)
18
19 router.route('/songs/id')
20     .get(artistsController.getSongsID)
21
22 router.route('/top')
23     .get(artistsController.getTopArtists)
24
25 module.exports = router
```

songRoutes.js

```
1  const express = require('express')
2  const router = express.Router()
3  const songsController = require('../controllers/songsController')
4
5  router.route('/all')
6      .get(songsController.getAllSongs)
7
8  router.route('/')
9      .get(songsController.getSongs)
10     .post(songsController.createNewSong)
11
12
13 router.route('/id')
14     .get(songsController.getSong)
15     .patch(songsController.updateSong)
16     .delete(songsController.deleteSong)
17
18 router.route('/top')
19     .get(songsController.getTopSongs)
20
21 router.route('/artist')
22     .delete(songsController.deleteSongName)
23
24 router.route('/artist/id')
25     .delete(songsController.deleteSongID)
26
27 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`