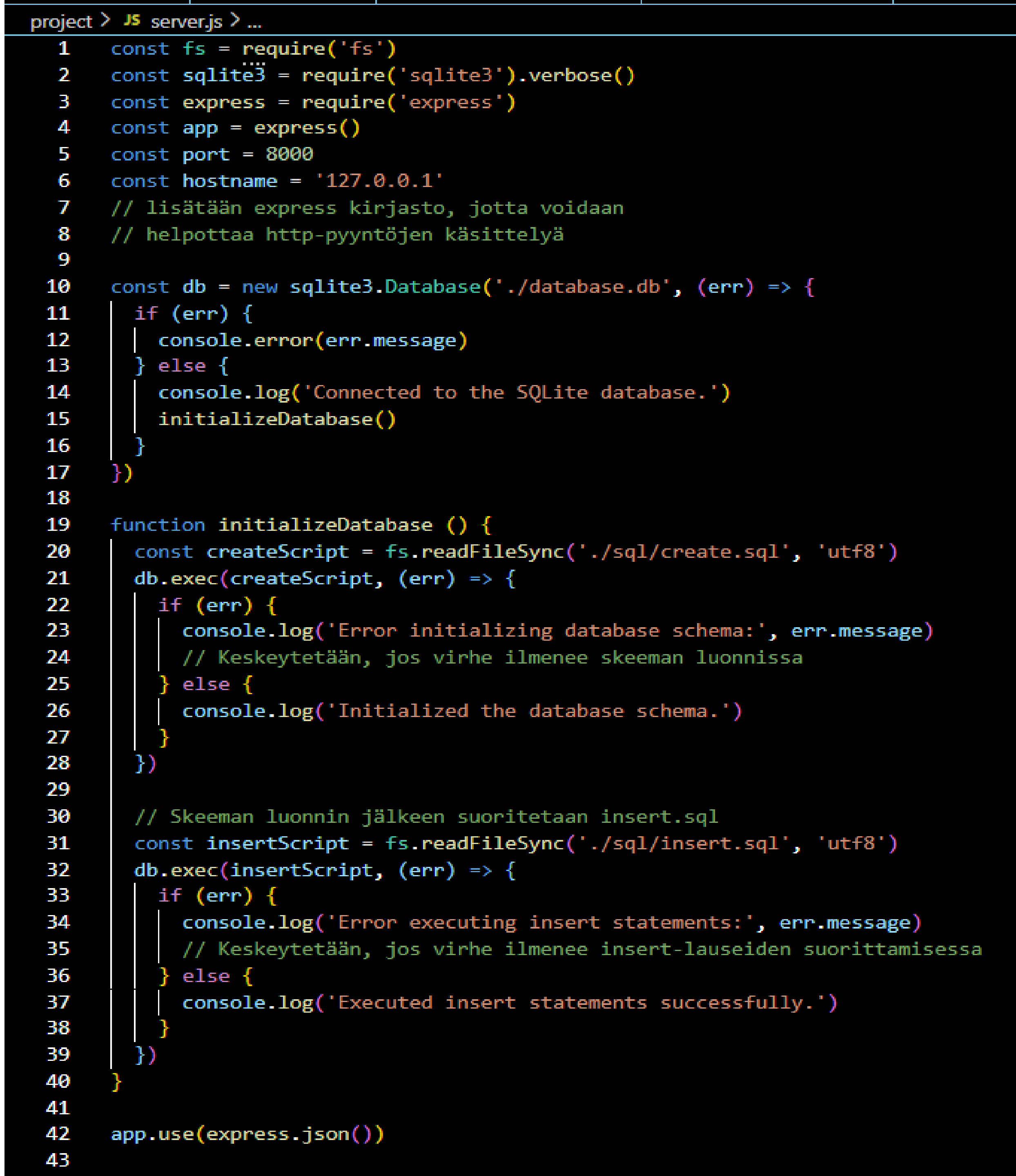
# API-project for course

Restful API using NodeJS and express-library.

# Server.js

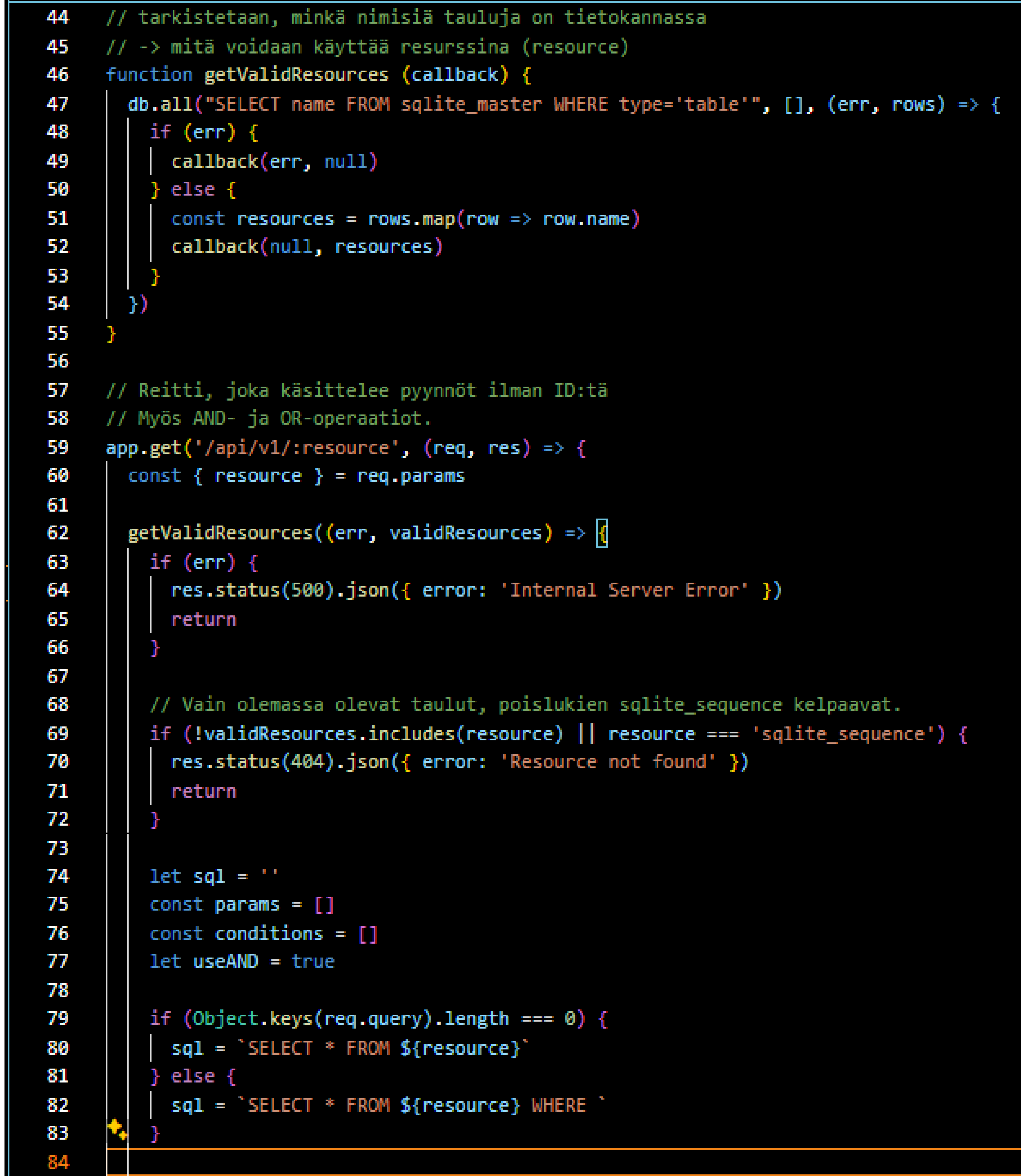
- 1-6 Constants of the program are declared.
- 10-17 Database is initialized to use ”database.db”.
- 19-40 Initialize-function for creating the tables for the database as well as to import the example data from insert.sql.



project > JS server.js > ...

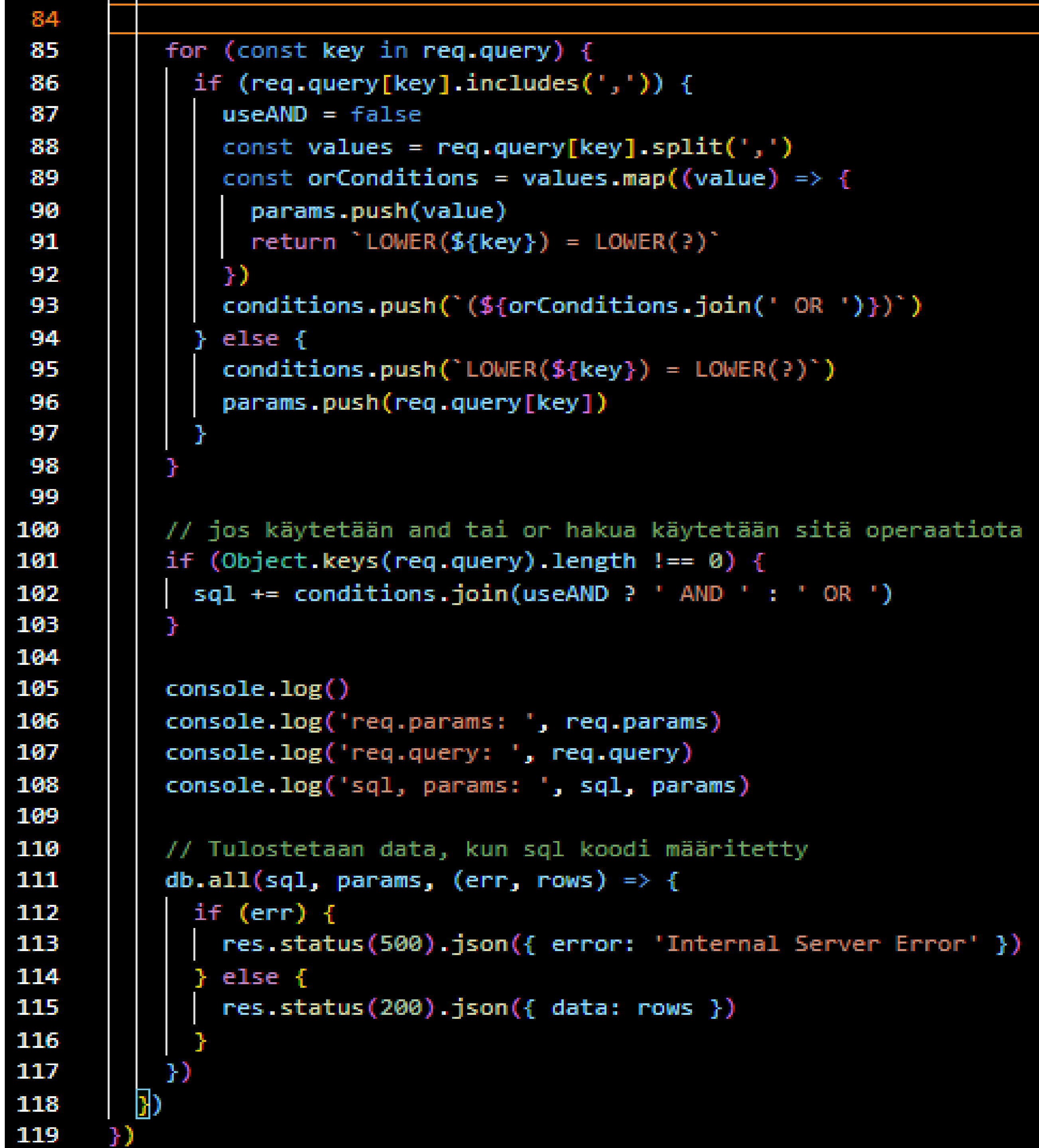
```javascript
1   const fs = require('fs')
2   const sqlite3 = require('sqlite3').verbose()
3   const express = require('express')
4   const app = express()
5   const port = 8000
6   const hostname = '127.0.0.1'
7   // lisätään express kirjasto, jotta voidaan
8   // helpottaa http-pyyntöjen käsittelyä
9
10  const db = new sqlite3.Database('./database.db', (err) => {
11    if (err) {
12      console.error(err.message)
13    } else {
14      console.log('Connected to the SQLite database.')
15      initializeDatabase()
16    }
17  })
18
19  function initializeDatabase () {
20    const createScript = fs.readFileSync('./sql/create.sql', 'utf8')
21    db.exec(createScript, (err) => {
22      if (err) {
23        console.log('Error initializing database schema:', err.message)
24        // Keskeytetään, jos virhe ilmenee skeeman luonnissa
25      } else {
26        console.log('Initialized the database schema.')
27      }
28    })
29
30    // Skeeman luonnin jälkeen suoritetaan insert.sql
31    const insertScript = fs.readFileSync('./sql/insert.sql', 'utf8')
32    db.exec(insertScript, (err) => {
33      if (err) {
34        console.log('Error executing insert statements:', err.message)
35        // Keskeytetään, jos virhe ilmenee insert-lauseiden suorittamisessa
36      } else {
37        console.log('Executed insert statements successfully.')
38      }
39    })
40  }
41
42  app.use(express.json())
43
```

- 46-55 Function will check what tables are used in the database and those are set to be on the ”white list” of our program.
- 59-119 Get-function for being able to access the data in json format.
- Function will check the input and return error, if something is wrong with the http-request.



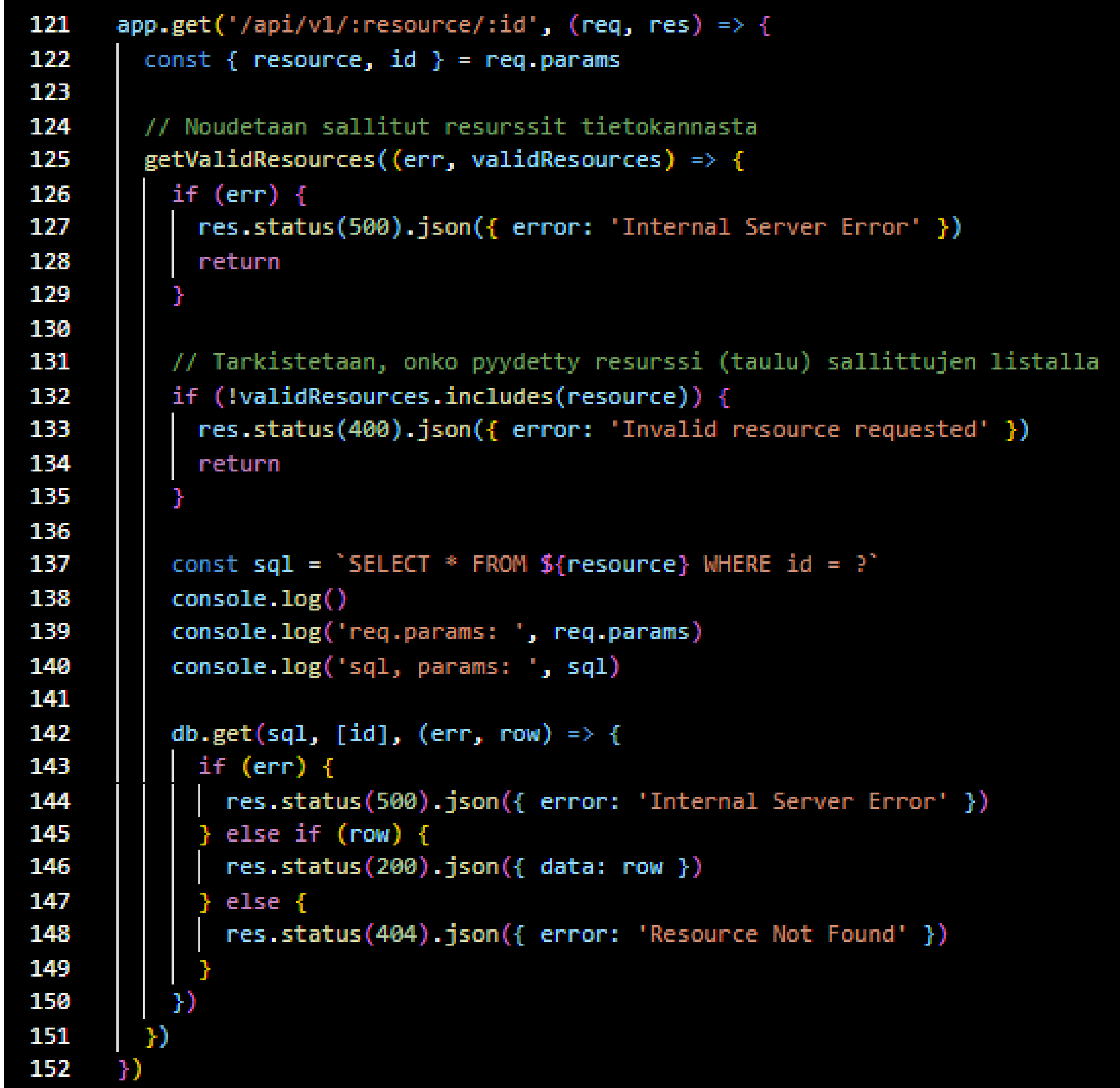
```
44  // tarkistetaan, minkä nimisiä tauluja on tietokannassa
45  // -> mitä voidaan käyttää resurssina (resource)
46  function getValidResources (callback) {
47    db.all("SELECT name FROM sqlite_master WHERE type='table'", [], (err, rows) => {
48      if (err) {
49        callback(err, null)
50      } else {
51        const resources = rows.map(row => row.name)
52        callback(null, resources)
53      }
54    })
55  }
56
57  // Reitti, joka käsittelee pyynnöt ilman ID:tä
58  // Myös AND- ja OR-operaatiot.
59  app.get('/api/v1/:resource', (req, res) => {
60    const { resource } = req.params
61
62    getValidResources((err, validResources) => {
63      if (err) {
64        res.status(500).json({ error: 'Internal Server Error' })
65        return
66      }
67
68      // Vain olemassa olevat taulut, poislukien sqlite_sequence kelpaavat.
69      if (!validResources.includes(resource) || resource === 'sqlite_sequence') {
70        res.status(404).json({ error: 'Resource not found' })
71        return
72      }
73
74      let sql = ''
75      const params = []
76      const conditions = []
77      let useAND = true
78
79      if (Object.keys(req.query).length === 0) {
80        sql = `SELECT * FROM ${resource}`
81      } else {
82        sql = `SELECT * FROM ${resource} WHERE `
83      }
84
```

- Ithere are no parameters given, function will return all the data from the table user gave in request.
- If there are given AND- or OR-operation the function will recognize it and build the sql-code accordingly.
- For debugging and to be able to follow the code flow, there are console.log() lines. With given parameters.



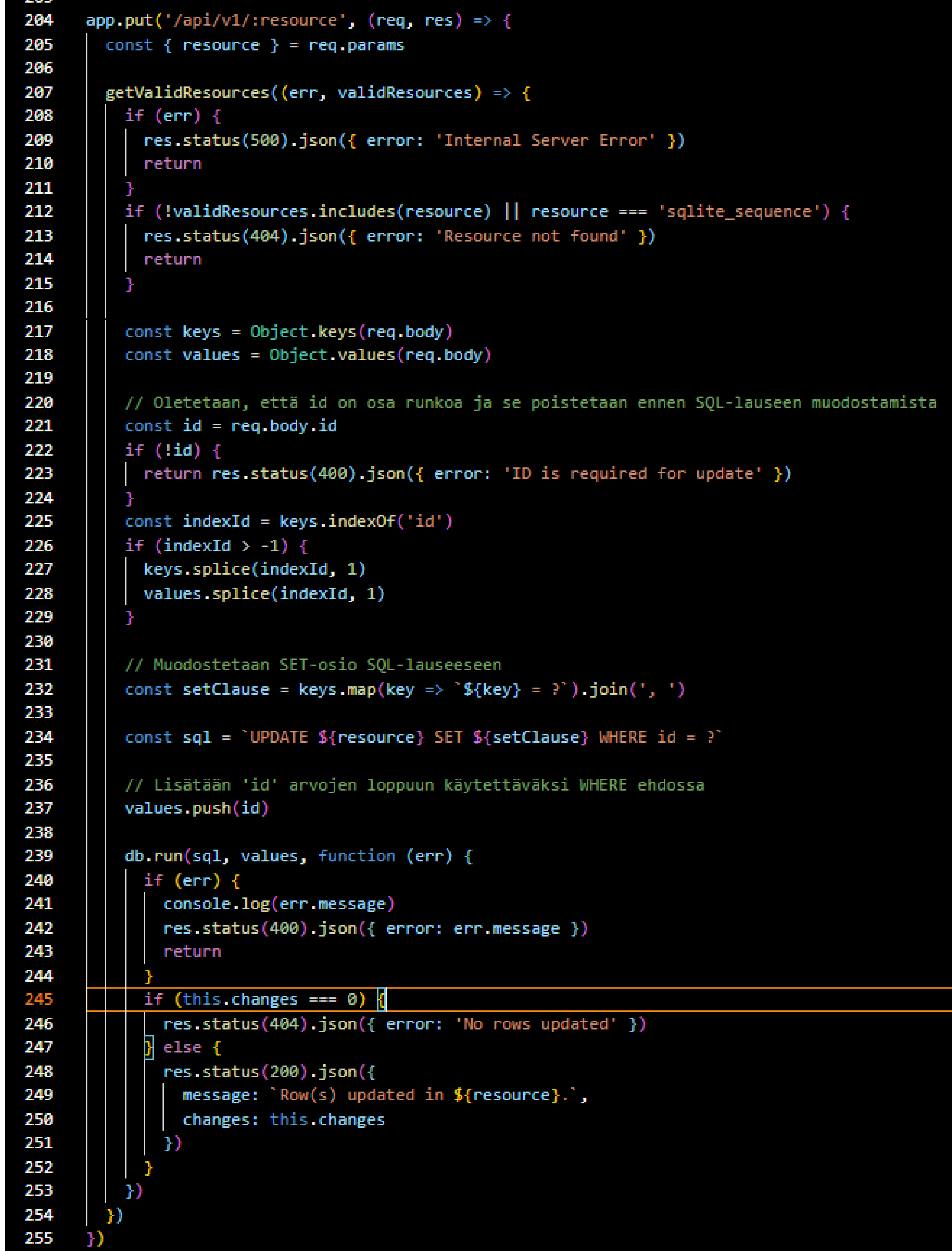
```

    for (const key in req.query) {
      if (req.query[key].includes(',')) {
        useAND = false
        const values = req.query[key].split(',')
        const orConditions = values.map((value) => {
          params.push(value)
          return `LOWER(${key}) = LOWER(?)`
        })
        conditions.push(`(${orConditions.join(' OR ')})`)
      } else {
        conditions.push(`LOWER(${key}) = LOWER(?)`)
        params.push(req.query[key])
      }
    }

    // jos käytetään and tai or hakua käytetään sitä operaatiota
    if (Object.keys(req.query).length !== 0) {
      sql += conditions.join(useAND ? ' AND ' : ' OR ')
    }

    console.log()
    console.log('req.params: ', req.params)
    console.log('req.query: ', req.query)
    console.log('sql, params: ', sql, params)

    // Tulostetaan data, kun sql koodi määritetty
    db.all(sql, params, (err, rows) => {
      if (err) {
        res.status(500).json({ error: 'Internal Server Error' })
      } else {
        res.status(200).json({ data: rows })
      }
    })
  })
})
```

- If the url includes id-number the corresponding data will be returned from given table after validation cheks have been done.
- In case of an error as allways a descriptive message will be prompted.



```
app.get('/api/v1/:resource/:id', (req, res) => {
  const { resource, id } = req.params

  // Noudetaan sallitut resurssit tietokannasta
  getValidResources((err, validResources) => {
    if (err) {
      res.status(500).json({ error: 'Internal Server Error' })
      return
    }

    // Tarkistetaan, onko pyydetty resurssi (taulu) sallittujen listalla
    if (!validResources.includes(resource)) {
      res.status(400).json({ error: 'Invalid resource requested' })
      return
    }

    const sql = `SELECT * FROM ${resource} WHERE id = ?`
    console.log()
    console.log('req.params: ', req.params)
    console.log('sql, params: ', sql)

    db.get(sql, [id], (err, row) => {
      if (err) {
        res.status(500).json({ error: 'Internal Server Error' })
      } else if (row) {
        res.status(200).json({ data: row })
      } else {
        res.status(404).json({ error: 'Resource Not Found' })
      }
    })
  })
})
```

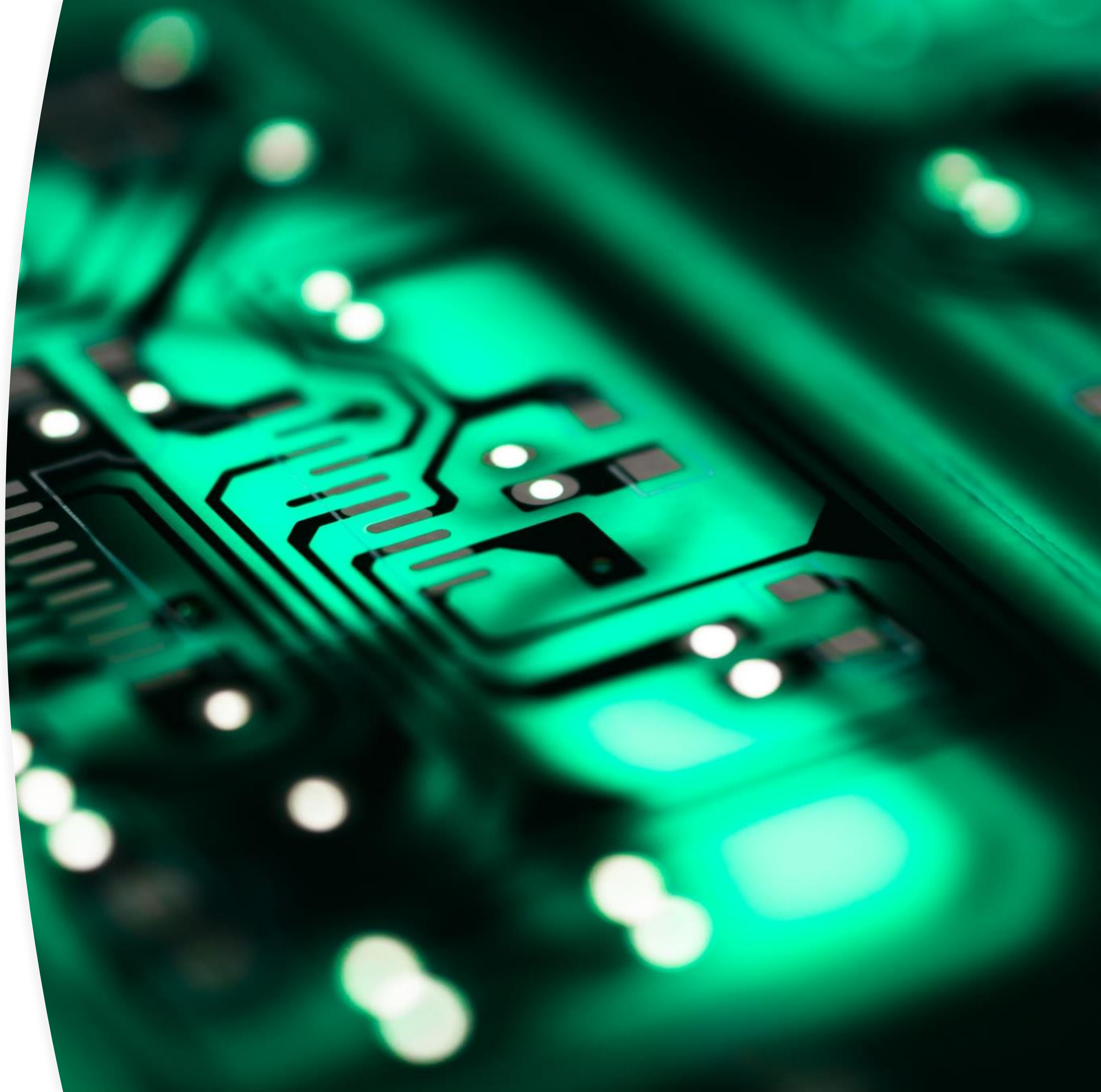
- The put/edit function has the same idea as all the others. The purpose is to be general and be able to intake data from any given table.
- Function will check if the given json works and update the given data.
- Id  of changed unit must be specified as the first parameter.



```
app.put('/api/v1/:resource', (req, res) => {
  const { resource } = req.params

  getValidResources((err, validResources) => {
    if (err) {
      res.status(500).json({ error: 'Internal Server Error' })
      return
    }
    if (!validResources.includes(resource) || resource === 'sqlite_sequence') {
      res.status(404).json({ error: 'Resource not found' })
      return
    }

    const keys = Object.keys(req.body)
    const values = Object.values(req.body)

    // Oletetaan, että id on osa runkoa ja se poistetaan ennen SQL-lauseen muodostamista
    const id = req.body.id
    if (!id) {
      return res.status(400).json({ error: 'ID is required for update' })
    }
    const indexId = keys.indexOf('id')
    if (indexId > -1) {
      keys.splice(indexId, 1)
      values.splice(indexId, 1)
    }

    // Muodostetaan SET-osio SQL-lauseeseen
    const setClause = keys.map(key => `${key} = ?`).join(', ')

    const sql = `UPDATE ${resource} SET ${setClause} WHERE id = ?`

    // Lisätään 'id' arvojen loppuun käytettäväksi WHERE ehdossa
    values.push(id)

    db.run(sql, values, function (err) {
      if (err) {
        console.log(err.message)
        res.status(400).json({ error: err.message })
        return
      }
      if (this.changes === 0) {
        res.status(404).json({ error: 'No rows updated' })
      } else {
        res.status(200).json({
          message: `Row(s) updated in ${resource}.`,
          changes: this.changes
        })
      }
    })
  })
})
```

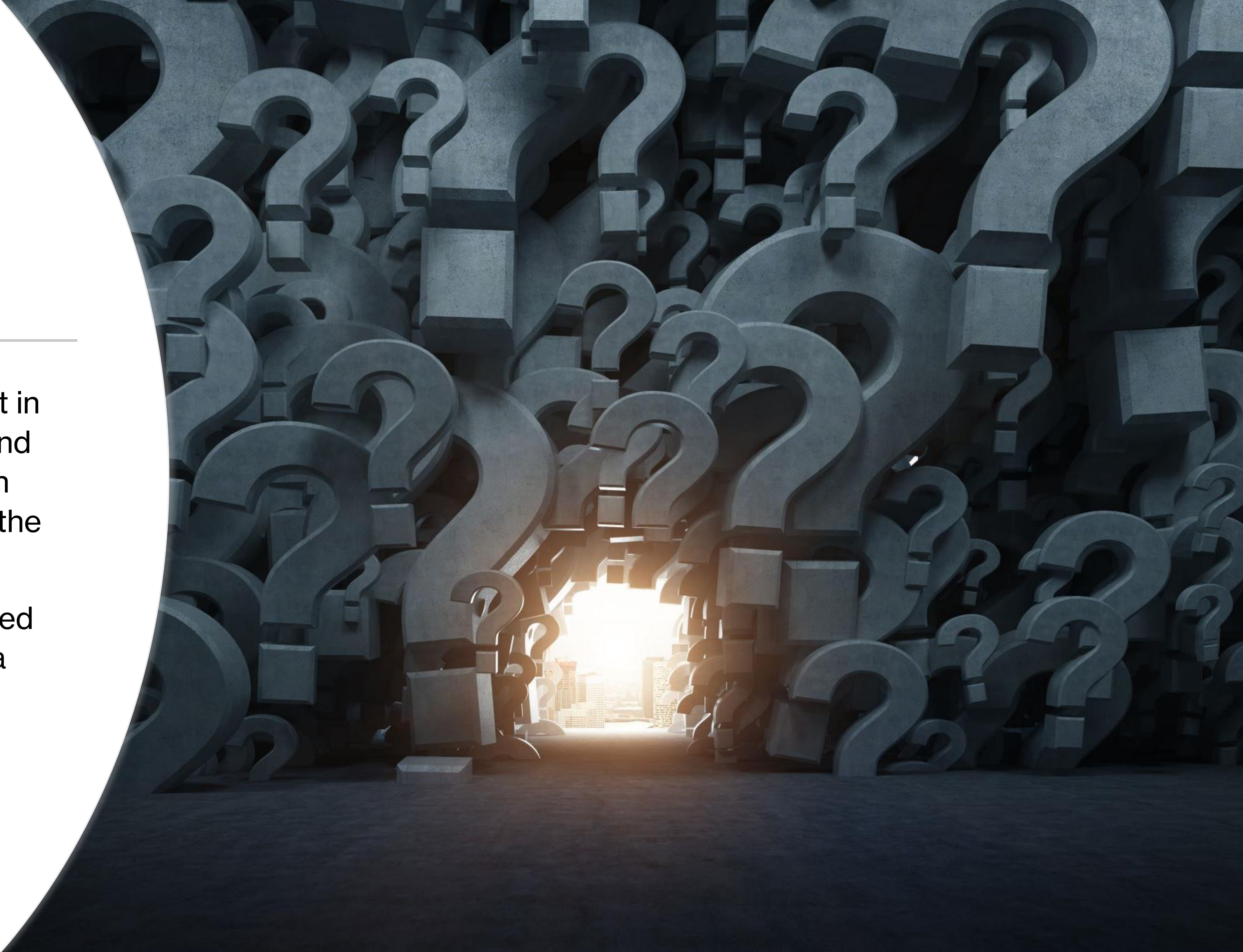
## Technologies

- NodeJS and express-library for better handling of http-requests.
- Sqlite used for database.



## Challenges

- The most challenging part in the project was to build and use javascript project with almost no experience on the language.
- Whole course concentrated on SQL-code which had a minimal role in the final project.



## Solutions

- Necessery information for building the project could be found on the internet.
- What comes to code, I wanted to keep it short and as geneeric as possible.
- So I made the functions to take parameters as variables and use those variables when building the sql-code to run on the database. So all the requests of the same type could be handled on same piece of code

## scalability

- The core idea in my .js-script has been that you can edit the database tables and still be able to use the same unchanged .js file because it will read the tables from the database.
- Project itself is scalable to handle larger set of data and javascript code's geneeric functionality quarantees good results.

## Summary

Overall the API building project was intresting concept.

Even thought we did practilly no javascript on the course I was able to build a working concept on application programming interface.

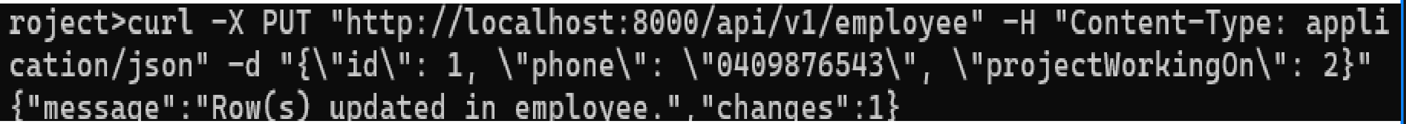
Time consumed by the project was a little bit over one week.

## Project Grade

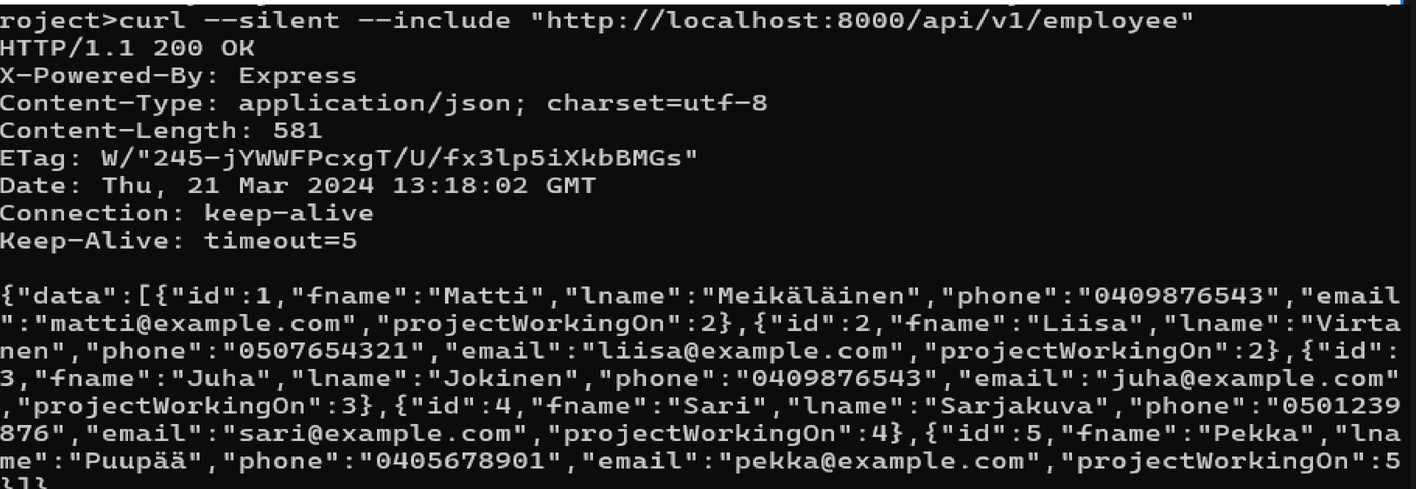
- With this project I aim for grade +2.

# Example output

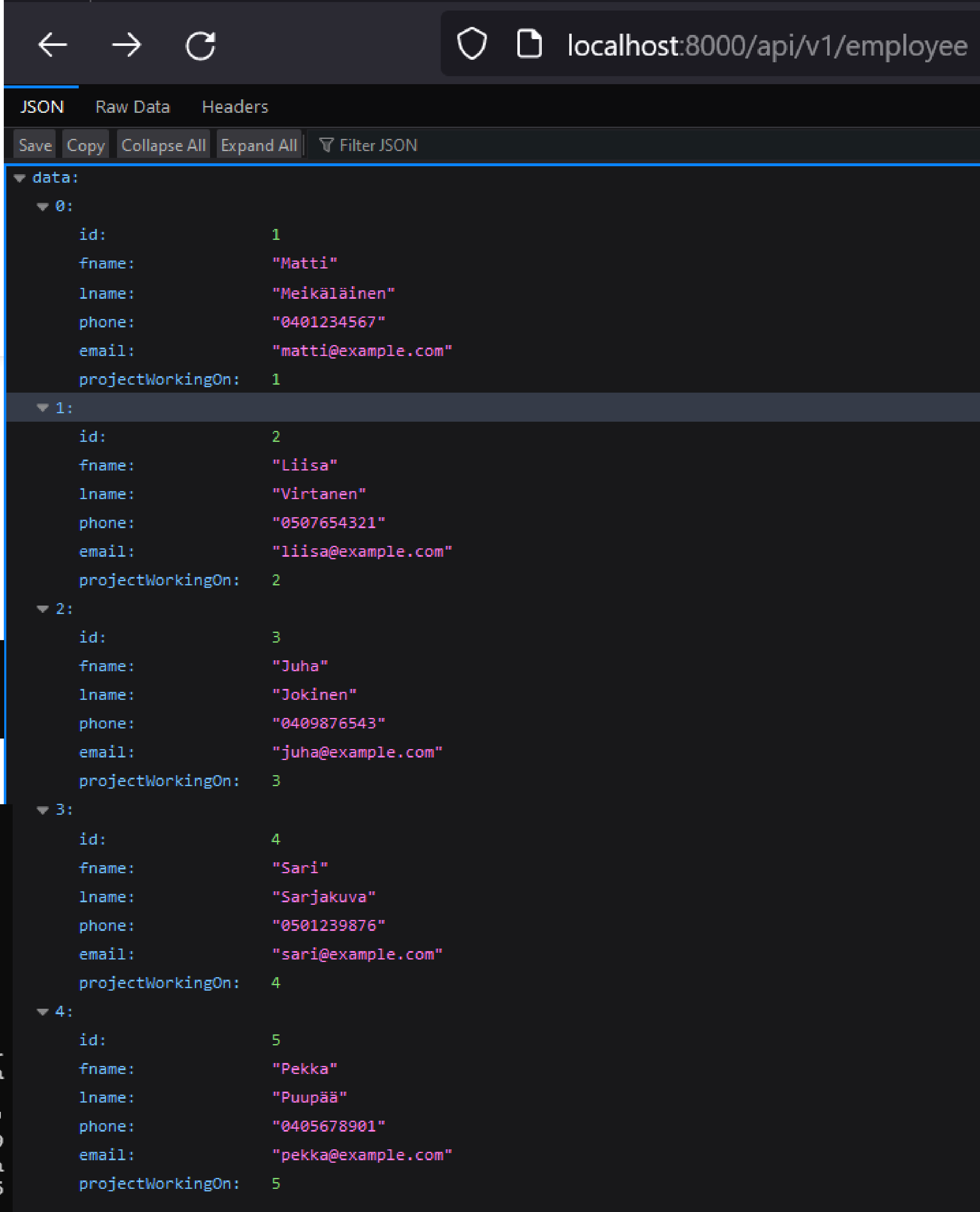
- Successfull requests on terminal and browser.



```
roject>curl -X PUT "http://localhost:8000/api/v1/employee" -H "Content-Type: appli
cation/json" -d "{\"id\": 1, \"phone\": \"0409876543\", \"projectWorkingOn\": 2}"
{"message":"Row(s) updated in employee.","changes":1}
```



```
roject>curl --silent --include "http://localhost:8000/api/v1/employee"
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 581
ETag: W/"245-jYWWFPcxgT/U/fx3lp5iXkbBMGs"
Date: Thu, 21 Mar 2024 13:18:02 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"data":[{"id":1,"fname":"Matti","lname":"Meikäläinen","phone":"0409876543","email
":"matti@example.com","projectWorkingOn":2},{"id":2,"fname":"Liisa","lname":"Virta
nen","phone":"0507654321","email":"liisa@example.com","projectWorkingOn":2},{"id":
3,"fname":"Juha","lname":"Jokinen","phone":"0409876543","email":"juha@example.com"
,"projectWorkingOn":3},{"id":4,"fname":"Sari","lname":"Sarjakuva","phone":"0501239
876","email":"sari@example.com","projectWorkingOn":4},{"id":5,"fname":"Pekka","lna
me":"Puupää","phone":"0405678901","email":"pekka@example.com","projectWorkingOn":5
}]}
```

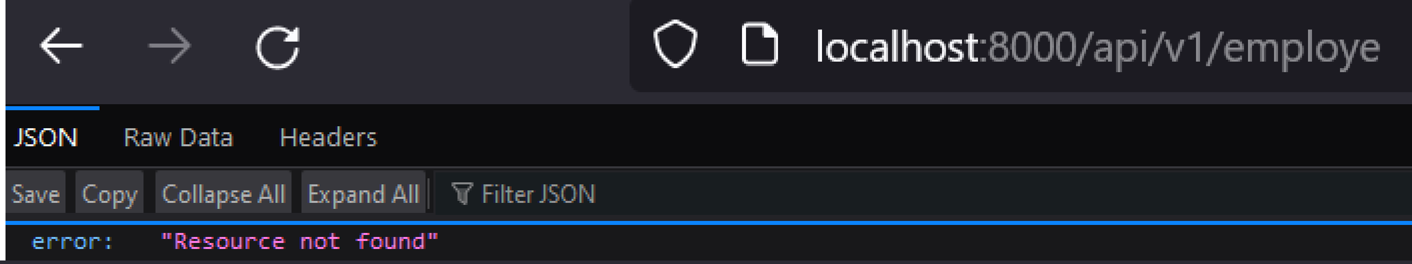


localhost:8000/api/v1/employee

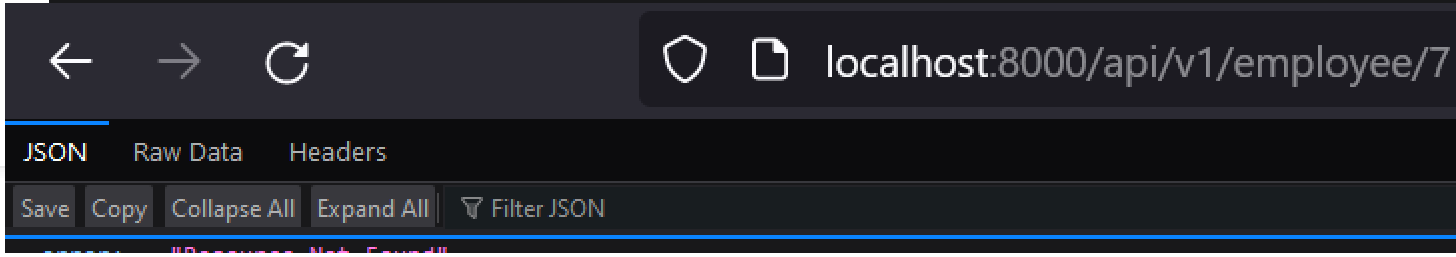
```
data:
  0:
    id:                1
    fname:             "Matti"
    lname:             "Meikäläinen"
    phone:             "0401234567"
    email:             "matti@example.com"
    projectWorkingOn:  1
  1:
    id:                2
    fname:             "Liisa"
    lname:             "Virtanen"
    phone:             "0507654321"
    email:             "liisa@example.com"
    projectWorkingOn:  2
  2:
    id:                3
    fname:             "Juha"
    lname:             "Jokinen"
    phone:             "0409876543"
    email:             "juha@example.com"
    projectWorkingOn:  3
  3:
    id:                4
    fname:             "Sari"
    lname:             "Sarjakuva"
    phone:             "0501239876"
    email:             "sari@example.com"
    projectWorkingOn:  4
  4:
    id:                5
    fname:             "Pekka"
    lname:             "Puupää"
    phone:             "0405678901"
    email:             "pekka@example.com"
    projectWorkingOn:  5
```

## Example errors

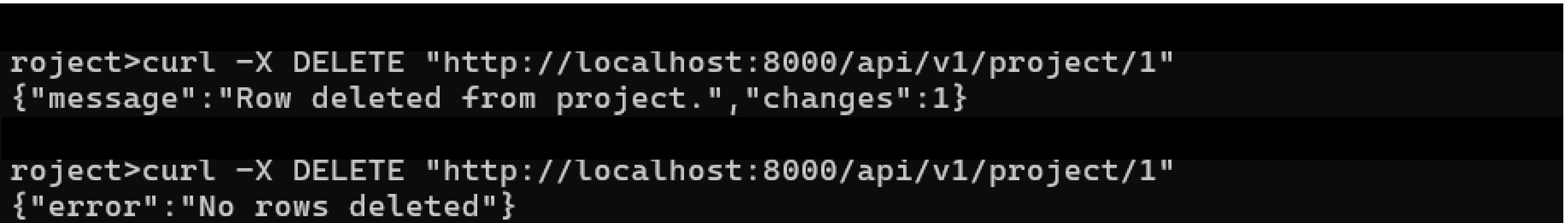
- Output of requests that respond with error messages.



```
error:   "Resource not found"
```



```
error:   "Resource Not Found"
```



```
roject>curl -X DELETE "http://localhost:8000/api/v1/project/1"
{"message":"Row deleted from project.","changes":1}

roject>curl -X DELETE "http://localhost:8000/api/v1/project/1"
{"error":"No rows deleted"}
```

## Thank you!