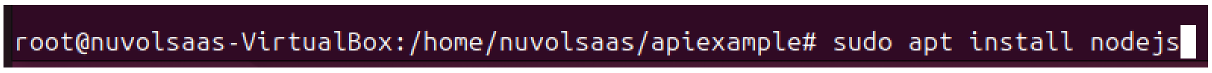
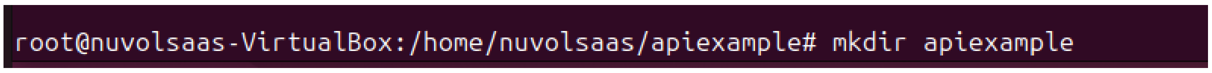
Exercici 3. Creació d'un servei web API REST.

1. En primer lloc, necessitareu instal·lar Node.js i npm (Node Package Manager) en el vostre sistema. Podeu descarregar Node.js i npm des de https://nodejs.org/.

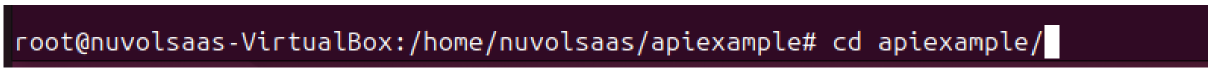


```
root@nuvolsaas-VirtualBox:/home/nuvolsaas/apiexample# sudo apt install nodejs
```

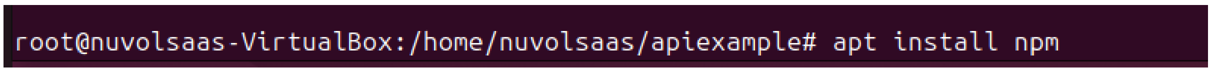
2. Un cop instal·lat Node.js i npm, creeu una nova carpeta per al vostre projecte i navegueu-hi des de la línia de comandes.



```
root@nuvolsaas-VirtualBox:/home/nuvolsaas/apiexample# mkdir apiexample
```

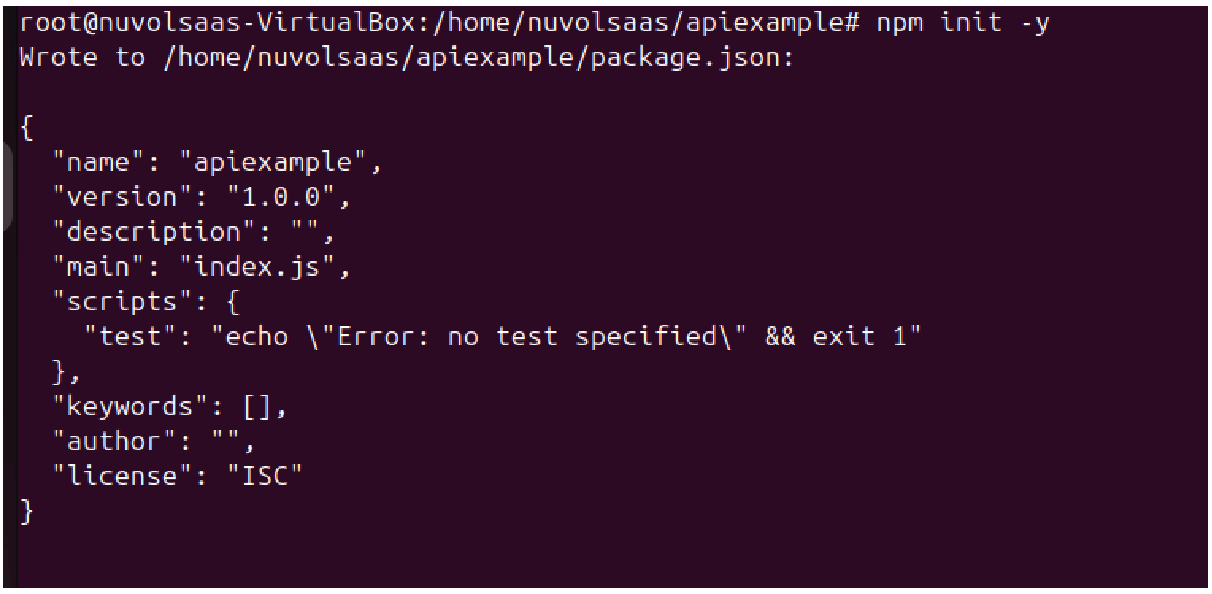


```
root@nuvolsaas-VirtualBox:/home/nuvolsaas/apiexample# cd apiexample/
```



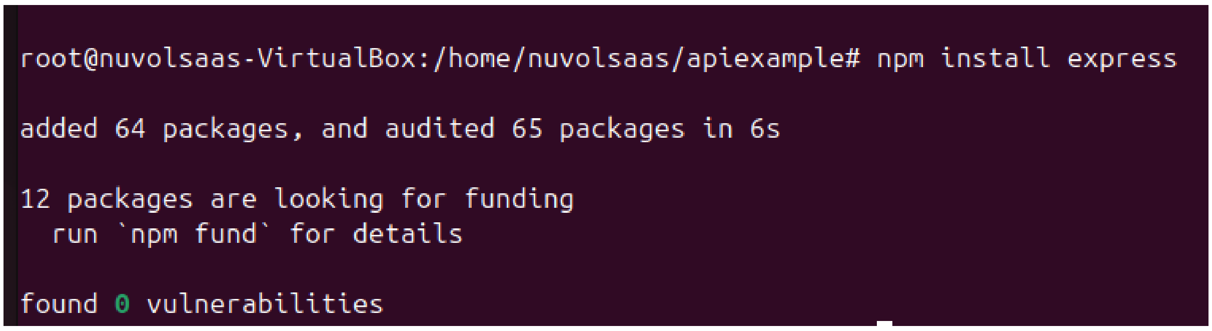
```
root@nuvolsaas-VirtualBox:/home/nuvolsaas/apiexample# apt install npm
```

3. A continuació, inicialitzeu un nou projecte Node.js executant `npm init -y`. Això crearà un nou fitxer `package.json` en la vostra carpeta.



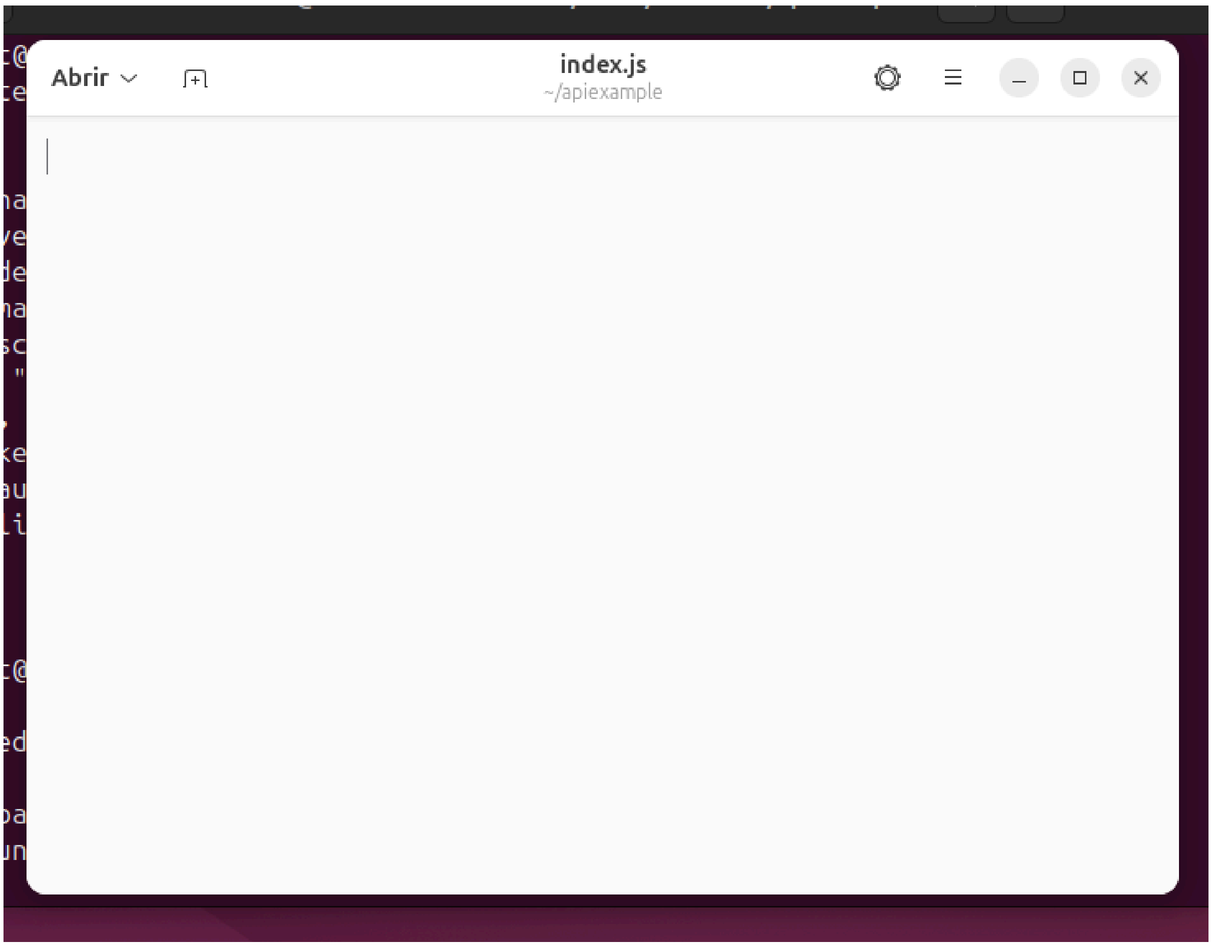
```
root@nuvolsaas-VirtualBox:/home/nuvolsaas/apiexample# npm init -y
Wrote to /home/nuvolsaas/apiexample/package.json:

{
  "name": "apiexample",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

4. Ara instal·leu Express.js en el vostre projecte amb `npm install express`.



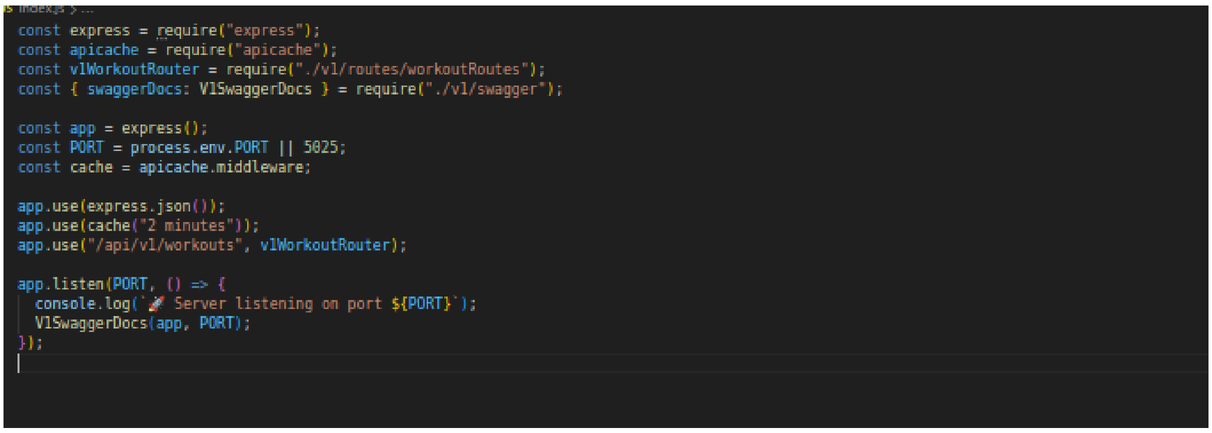
```
root@nuvolsaas-VirtualBox:/home/nuvolsaas/apiexample# npm install express

added 64 packages, and audited 65 packages in 6s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

5. Creeu un nou fitxer anomenat `index.js` i obriu-lo en el vostre editor de textos preferit.



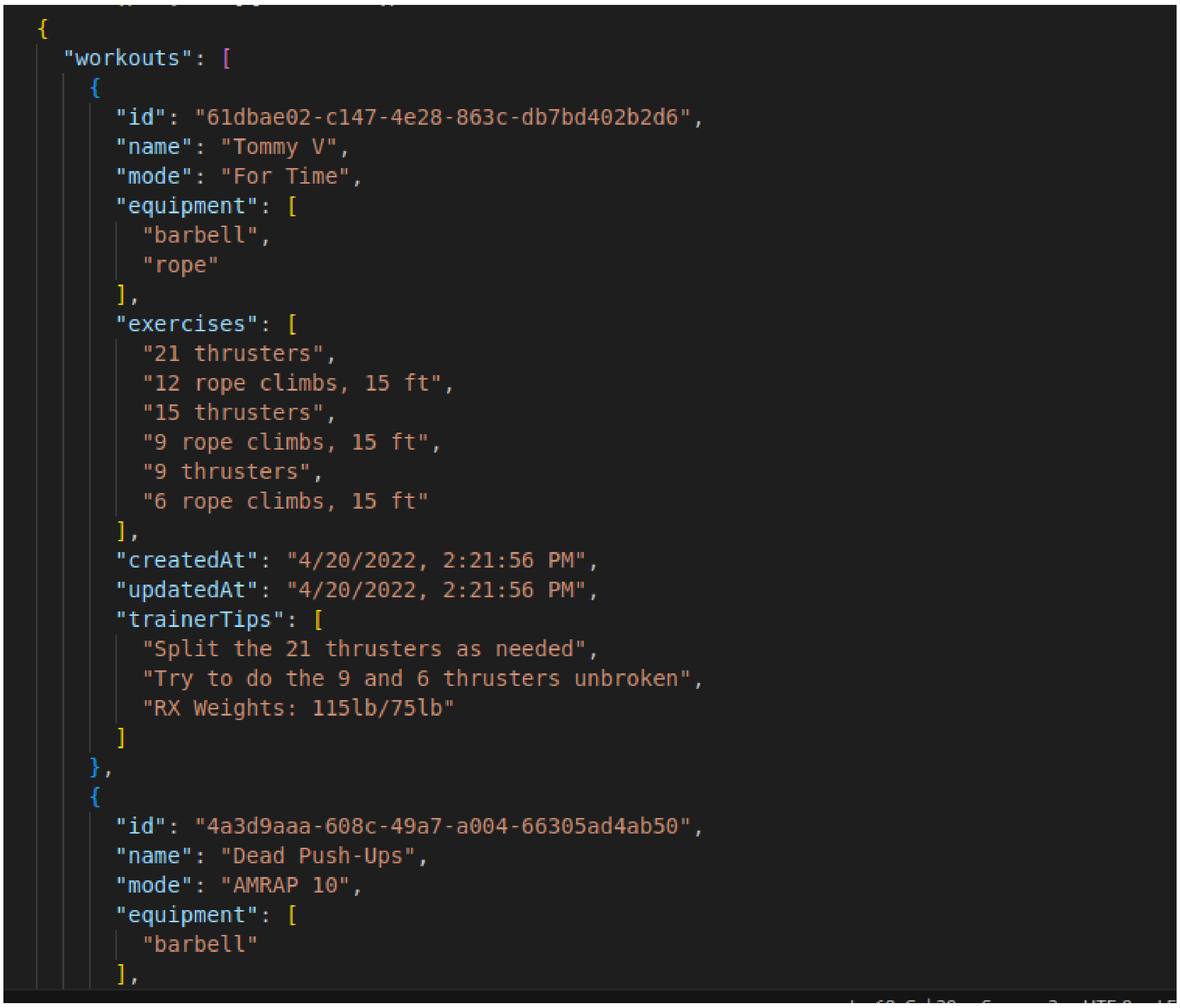
6. A continuació es mostra un exemple de codi per a un servei API REST senzill:

   Index.js



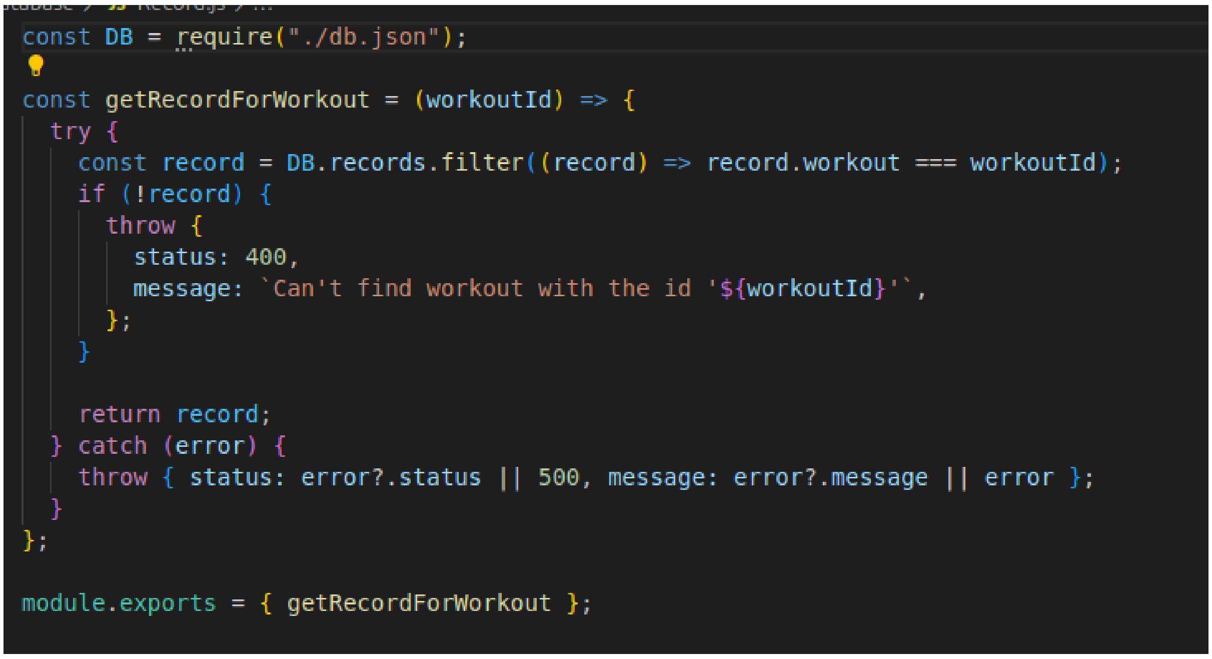
```
JS index.js > ...
const express = require("express");
const apicache = require("apicache");
const v1WorkoutRouter = require("./v1/routes/workoutRoutes");
const { swaggerDocs: V1SwaggerDocs } = require("./v1/swagger");

const app = express();
const PORT = process.env.PORT || 5025;
const cache = apicache.middleware;

app.use(express.json());
app.use(cache("2 minutes"));
app.use("/api/v1/workouts", v1WorkoutRouter);

app.listen(PORT, () => {
  console.log(`🚀 Server listening on port ${PORT}`);
  V1SwaggerDocs(app, PORT);
});
```

carpeta database:
db.json:



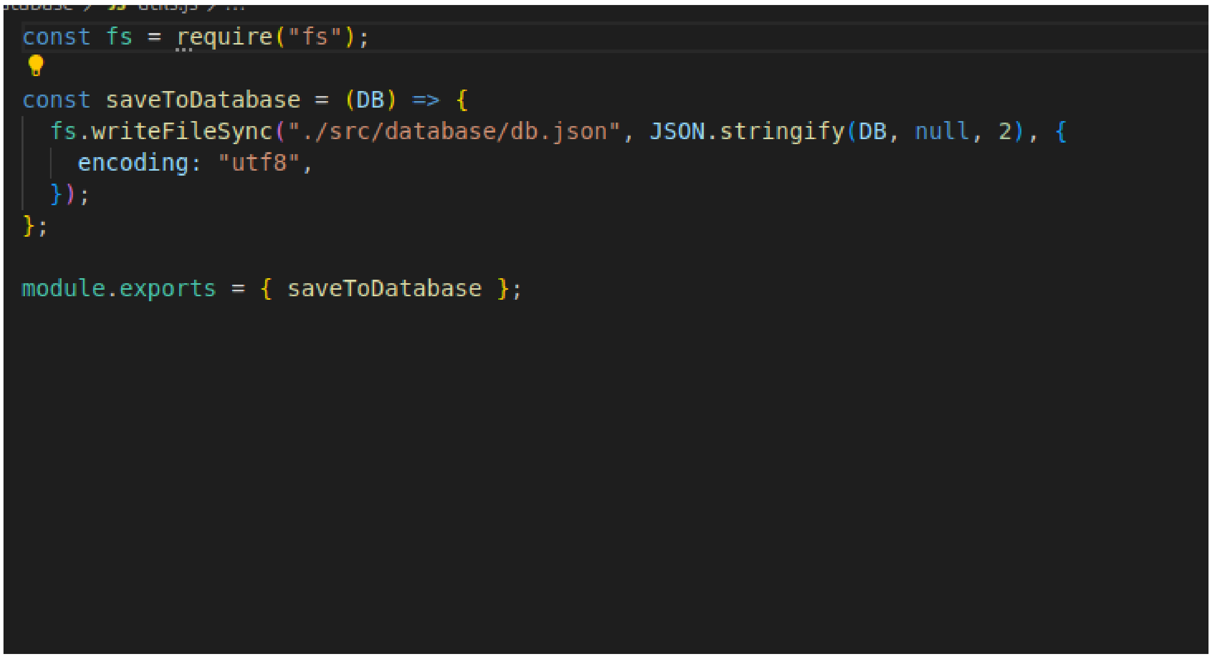
```
{
  "workouts": [
    {
      "id": "61dbae02-c147-4e28-863c-db7bd402b2d6",
      "name": "Tommy V",
      "mode": "For Time",
      "equipment": [
        "barbell",
        "rope"
      ],
      "exercises": [
        "21 thrusters",
        "12 rope climbs, 15 ft",
        "15 thrusters",
        "9 rope climbs, 15 ft",
        "9 thrusters",
        "6 rope climbs, 15 ft"
      ],
      "createdAt": "4/20/2022, 2:21:56 PM",
      "updatedAt": "4/20/2022, 2:21:56 PM",
      "trainerTips": [
        "Split the 21 thrusters as needed",
        "Try to do the 9 and 6 thrusters unbroken",
        "RX Weights: 115lb/75lb"
      ]
    },
    {
      "id": "4a3d9aaa-608c-49a7-a004-66305ad4ab50",
      "name": "Dead Push-Ups",
      "mode": "AMRAP 10",
      "equipment": [
        "barbell"
      ],
```

Record.js



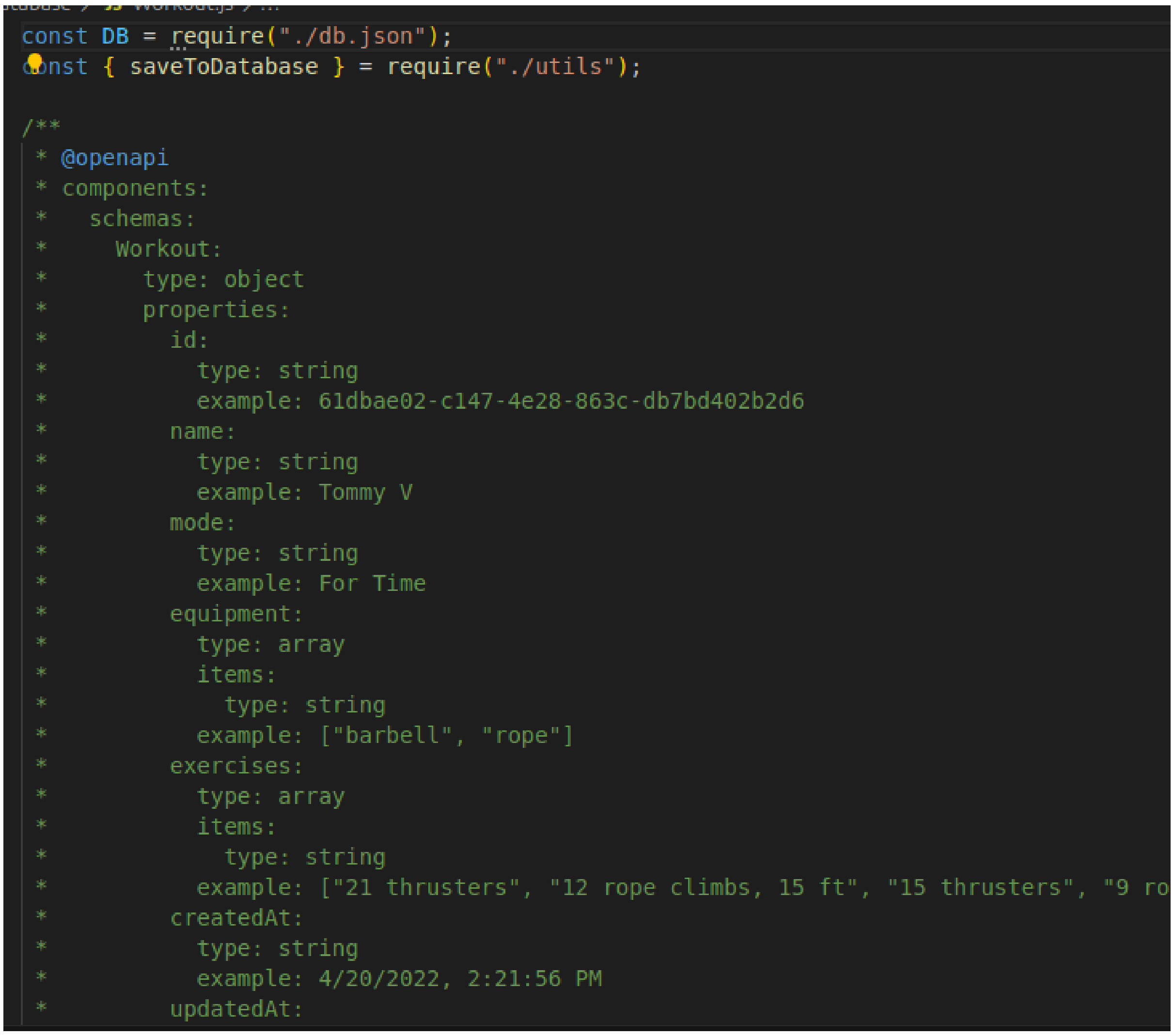
```
const DB = require("./db.json");

const getRecordForWorkout = (workoutId) => {
  try {
    const record = DB.records.filter((record) => record.workout === workoutId);
    if (!record) {
      throw {
        status: 400,
        message: `Can't find workout with the id '${workoutId}'`,
      };
    }

    return record;
  } catch (error) {
    throw { status: error?.status || 500, message: error?.message || error };
  }
};

module.exports = { getRecordForWorkout };
```

utils.js

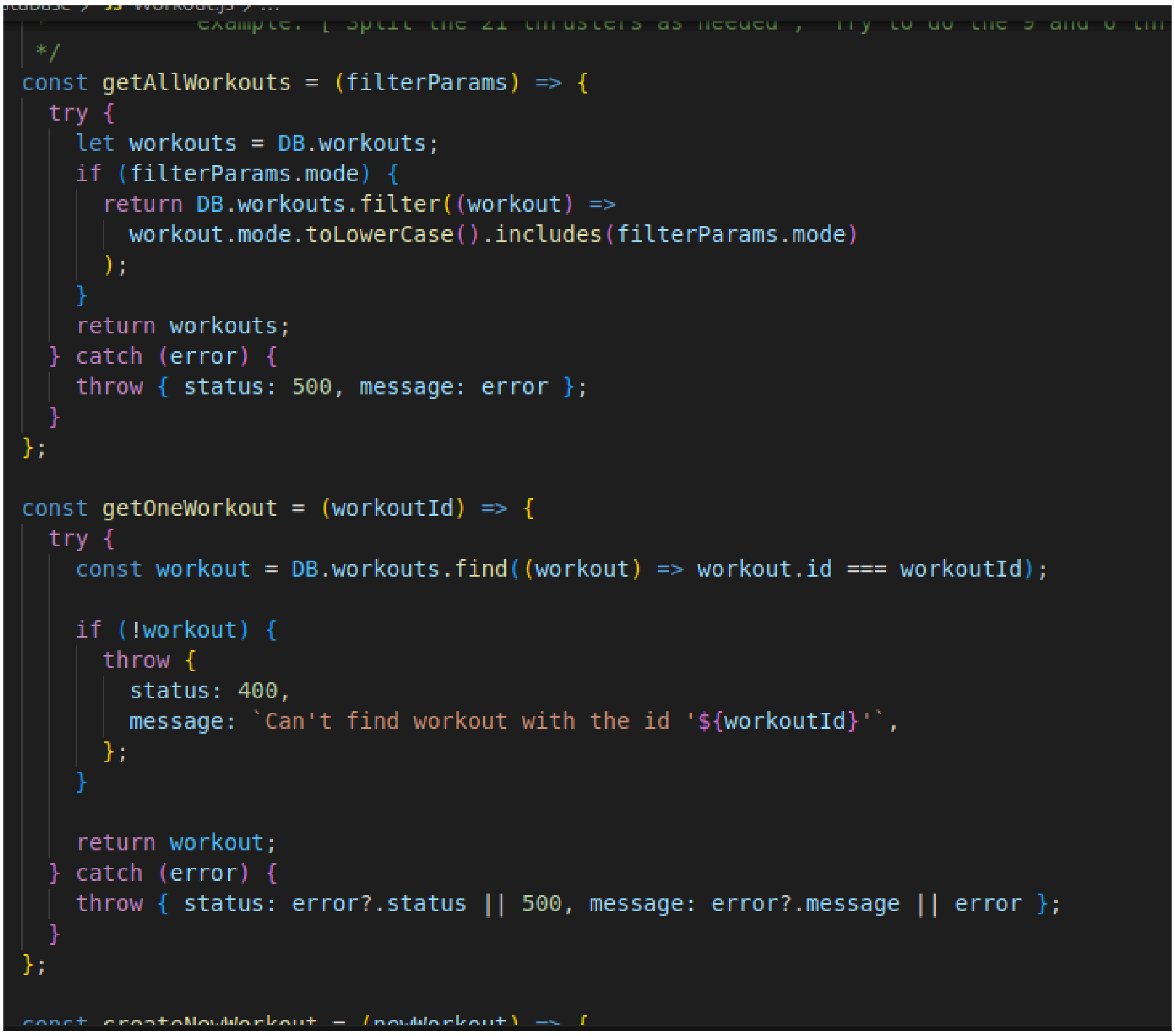


```
const fs = require("fs");

const saveToDatabase = (DB) => {
  fs.writeFileSync("./src/database/db.json", JSON.stringify(DB, null, 2), {
    encoding: "utf8",
  });
};

module.exports = { saveToDatabase };
```

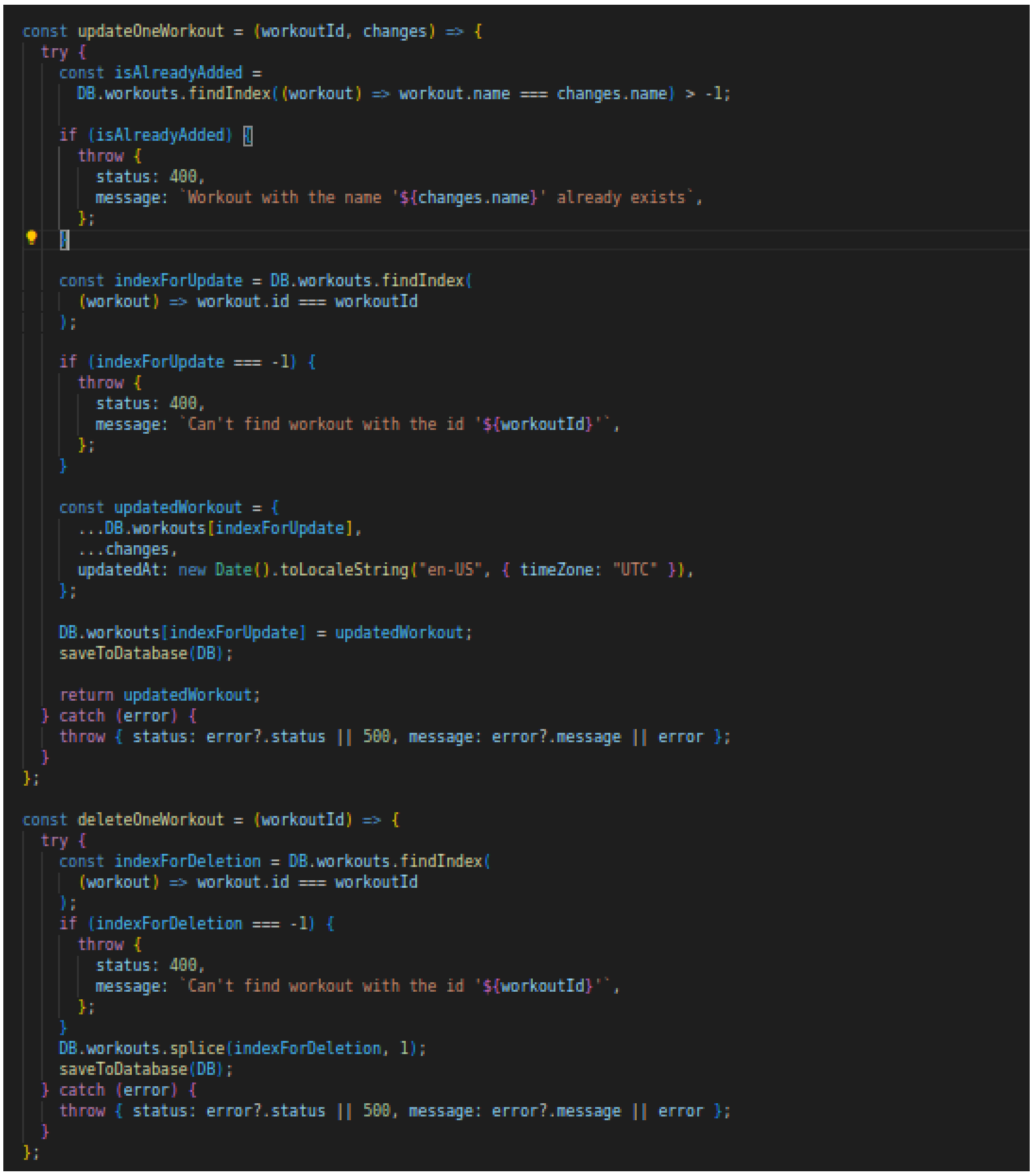
Workout.js



```
const DB = require("./db.json");
const { saveToDatabase } = require("./utils");

/**
 * @openapi
 * components:
 *   schemas:
 *     Workout:
 *       type: object
 *       properties:
 *         id:
 *           type: string
 *           example: 61dbae02-c147-4e28-863c-db7bd402b2d6
 *         name:
 *           type: string
 *           example: Tommy V
 *         mode:
 *           type: string
 *           example: For Time
 *         equipment:
 *           type: array
 *           items:
 *             type: string
 *           example: ["barbell", "rope"]
 *         exercises:
 *           type: array
 *           items:
 *             type: string
 *           example: ["21 thrusters", "12 rope climbs, 15 ft", "15 thrusters", "9 ro
 *         createdAt:
 *           type: string
 *           example: 4/20/2022, 2:21:56 PM
 *         updatedAt:
```

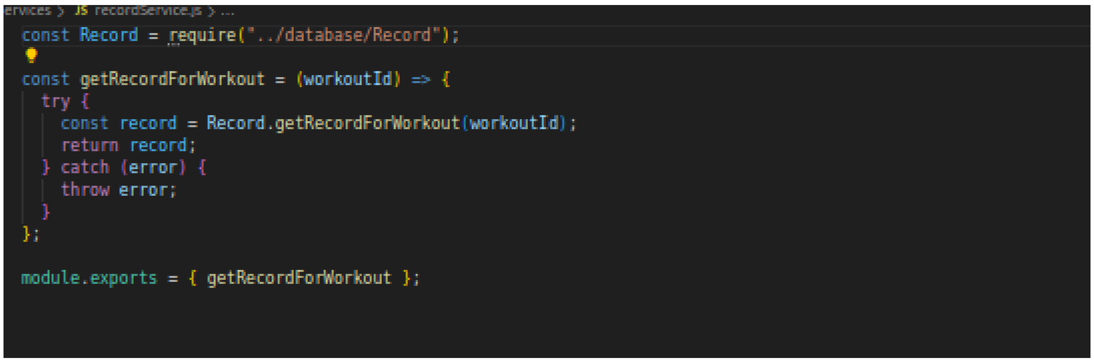


```
atabase > Workout.js > ...
 *              example: [ Split the 21 thrusters as needed ', 'Try to do the 9 and 6 th
 */
const getAllWorkouts = (filterParams) => {
  try {
    let workouts = DB.workouts;
    if (filterParams.mode) {
      return DB.workouts.filter((workout) =>
        workout.mode.toLowerCase().includes(filterParams.mode)
      );
    }
    return workouts;
  } catch (error) {
    throw { status: 500, message: error };
  }
};

const getOneWorkout = (workoutId) => {
  try {
    const workout = DB.workouts.find((workout) => workout.id === workoutId);

    if (!workout) {
      throw {
        status: 400,
        message: `Can't find workout with the id '${workoutId}'`,
      };
    }

    return workout;
  } catch (error) {
    throw { status: error?.status || 500, message: error?.message || error };
  }
};

const createNewWorkout = (newWorkout) => {
```



```javascript
const updateOneWorkout = (workoutId, changes) => {
  try {
    const isAlreadyAdded =
      DB.workouts.findIndex((workout) => workout.name === changes.name) > -1;

    if (isAlreadyAdded) {
      throw {
        status: 400,
        message: `Workout with the name '${changes.name}' already exists`,
      };
    }

    const indexForUpdate = DB.workouts.findIndex(
      (workout) => workout.id === workoutId
    );

    if (indexForUpdate === -1) {
      throw {
        status: 400,
        message: `Can't find workout with the id '${workoutId}'`,
      };
    }

    const updatedWorkout = {
      ...DB.workouts[indexForUpdate],
      ...changes,
      updatedAt: new Date().toLocaleString("en-US", { timeZone: "UTC" }),
    };

    DB.workouts[indexForUpdate] = updatedWorkout;
    saveToDatabase(DB);

    return updatedWorkout;
  } catch (error) {
    throw { status: error?.status || 500, message: error?.message || error };
  }
};

const deleteOneWorkout = (workoutId) => {
  try {
    const indexForDeletion = DB.workouts.findIndex(
      (workout) => workout.id === workoutId
    );
    if (indexForDeletion === -1) {
      throw {
        status: 400,
        message: `Can't find workout with the id '${workoutId}'`,
      };
    }
    DB.workouts.splice(indexForDeletion, 1);
    saveToDatabase(DB);
  } catch (error) {
    throw { status: error?.status || 500, message: error?.message || error };
  }
};
```

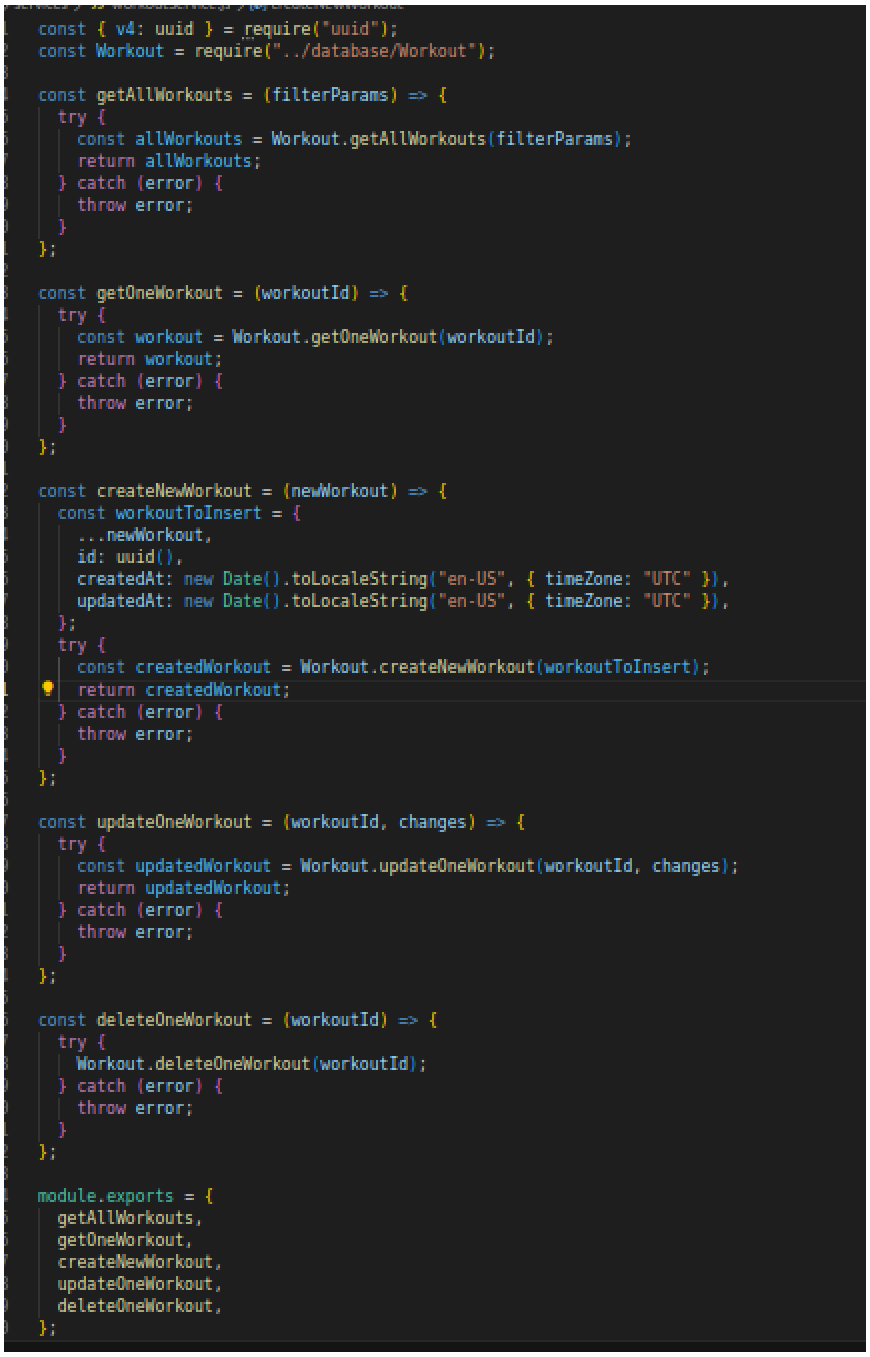
carpeta services:
recordService.js



```
ervices > JS recordService.js > ...
```

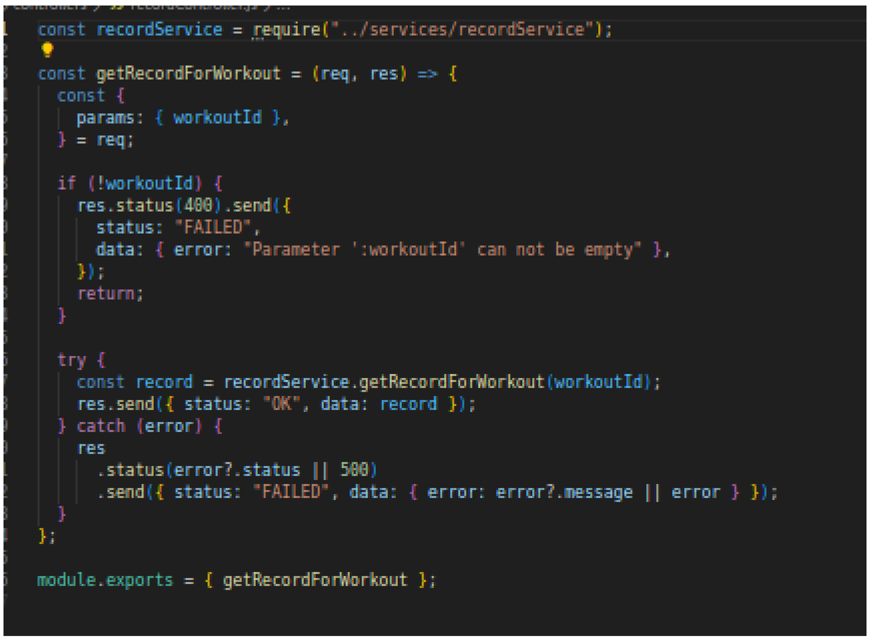
```javascript
const Record = require("../database/Record");

const getRecordForWorkout = (workoutId) => {
  try {
    const record = Record.getRecordForWorkout(workoutId);
    return record;
  } catch (error) {
    throw error;
  }
};

module.exports = { getRecordForWorkout };
```

workoutService.js



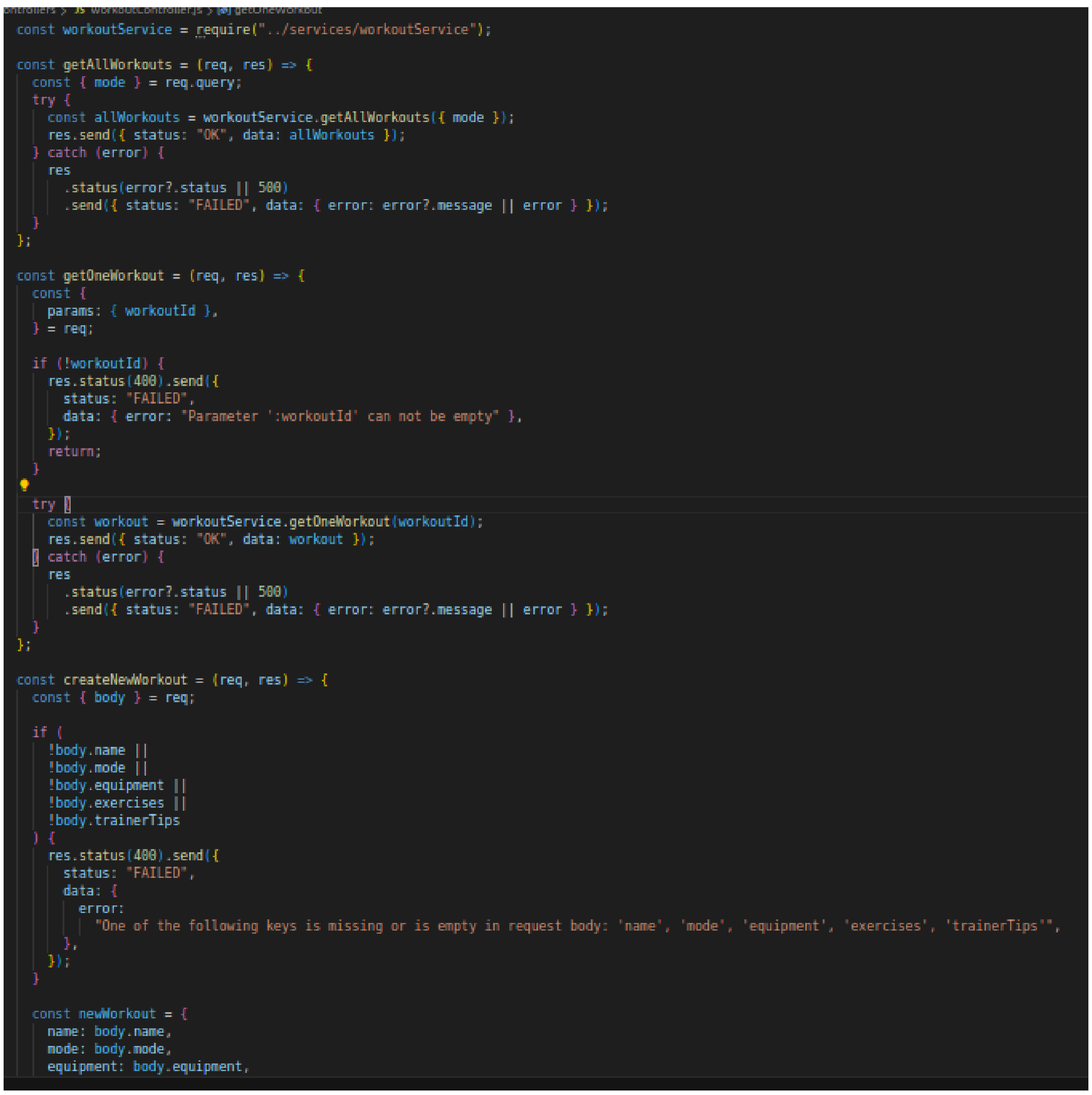
```javascript
const { v4: uuid } = require("uuid");
const Workout = require("../database/Workout");

const getAllWorkouts = (filterParams) => {
  try {
    const allWorkouts = Workout.getAllWorkouts(filterParams);
    return allWorkouts;
  } catch (error) {
    throw error;
  }
};

const getOneWorkout = (workoutId) => {
  try {
    const workout = Workout.getOneWorkout(workoutId);
    return workout;
  } catch (error) {
    throw error;
  }
};

const createNewWorkout = (newWorkout) => {
  const workoutToInsert = {
    ...newWorkout,
    id: uuid(),
    createdAt: new Date().toLocaleString("en-US", { timeZone: "UTC" }),
    updatedAt: new Date().toLocaleString("en-US", { timeZone: "UTC" }),
  };
  try {
    const createdWorkout = Workout.createNewWorkout(workoutToInsert);
    return createdWorkout;
  } catch (error) {
    throw error;
  }
};

const updateOneWorkout = (workoutId, changes) => {
  try {
    const updatedWorkout = Workout.updateOneWorkout(workoutId, changes);
    return updatedWorkout;
  } catch (error) {
    throw error;
  }
};

const deleteOneWorkout = (workoutId) => {
  try {
    Workout.deleteOneWorkout(workoutId);
  } catch (error) {
    throw error;
  }
};

module.exports = {
  getAllWorkouts,
  getOneWorkout,
  createNewWorkout,
  updateOneWorkout,
  deleteOneWorkout,
};
```

carpeta controllers:
recordControllers.js



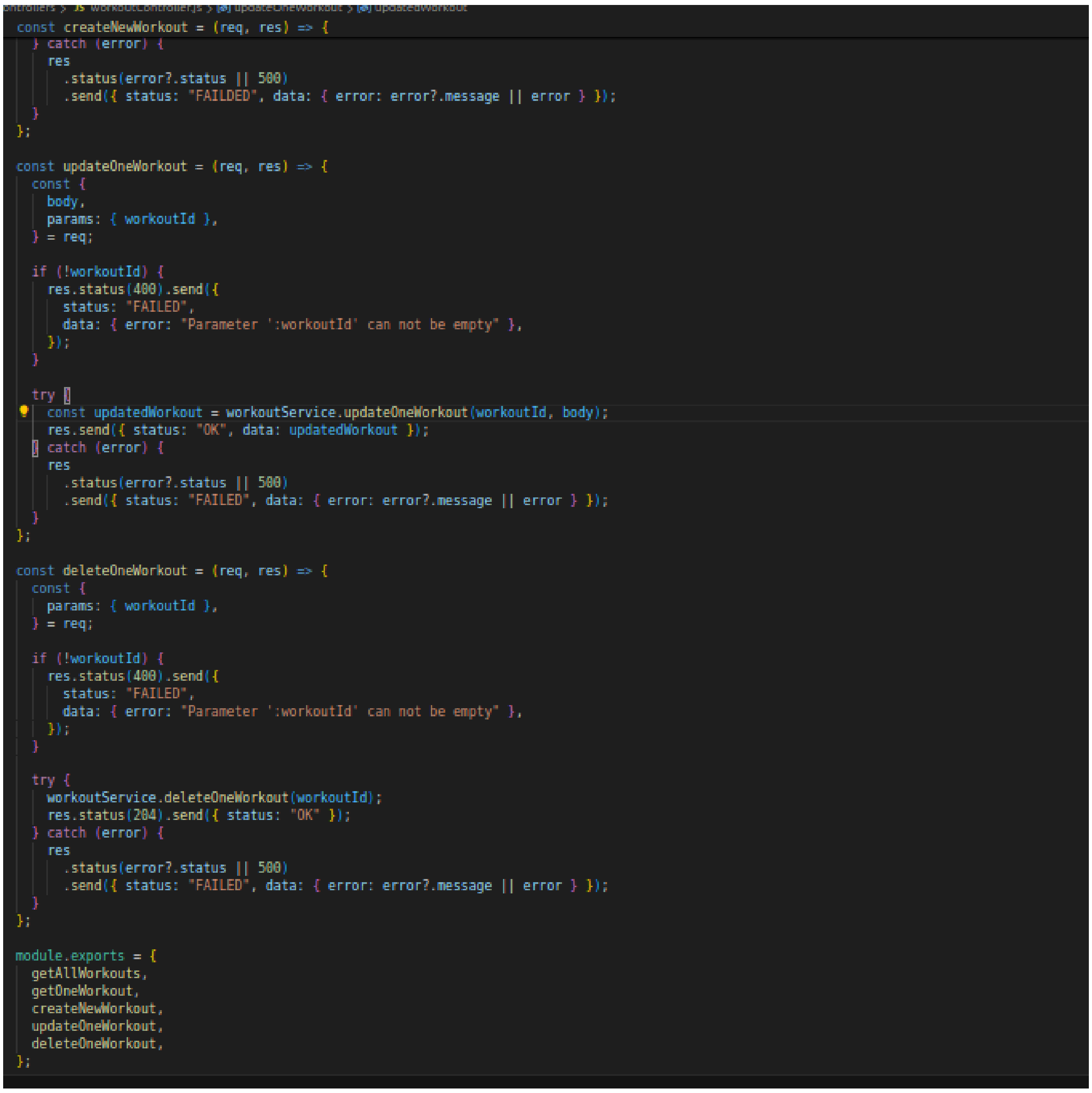
```javascript
const recordService = require("../services/recordService");

const getRecordForWorkout = (req, res) => {
  const {
    params: { workoutId },
  } = req;

  if (!workoutId) {
    res.status(400).send({
      status: "FAILED",
      data: { error: "Parameter ':workoutId' can not be empty" },
    });
    return;
  }

  try {
    const record = recordService.getRecordForWorkout(workoutId);
    res.send({ status: "OK", data: record });
  } catch (error) {
    res
      .status(error?.status || 500)
      .send({ status: "FAILED", data: { error: error?.message || error } });
  }
};

module.exports = { getRecordForWorkout };
```

workoutControllers.js



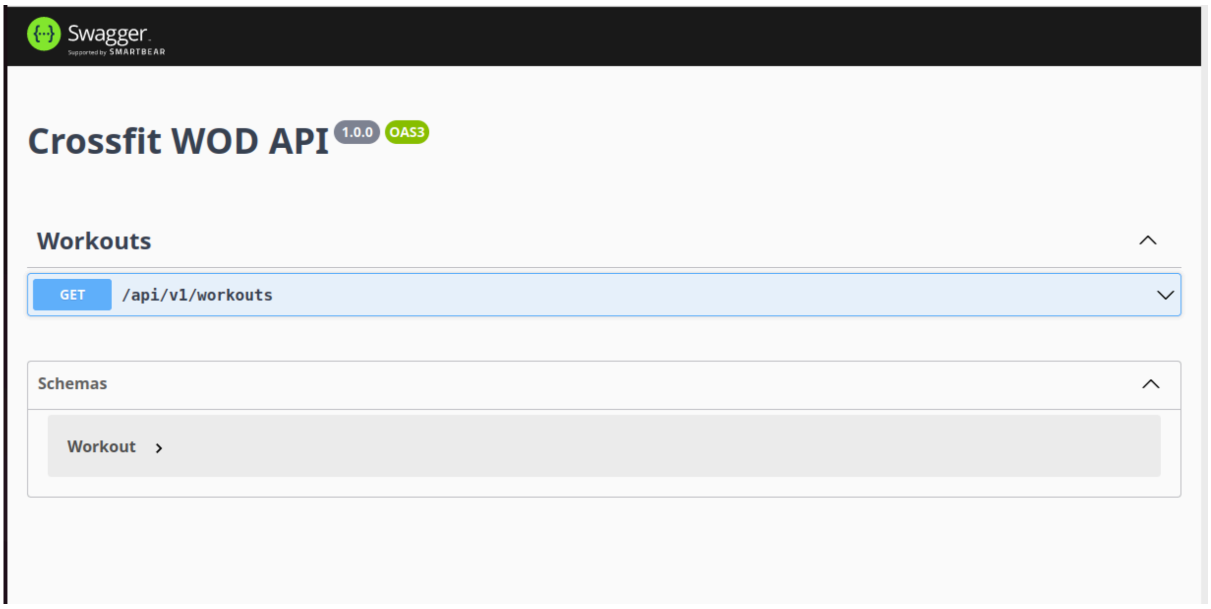
ontrollers > JS workoutController.js > [ø] getOneWorkout

```javascript
const workoutService = require("../services/workoutService");

const getAllWorkouts = (req, res) => {
  const { mode } = req.query;
  try {
    const allWorkouts = workoutService.getAllWorkouts({ mode });
    res.send({ status: "OK", data: allWorkouts });
  } catch (error) {
    res
      .status(error?.status || 500)
      .send({ status: "FAILED", data: { error: error?.message || error } });
  }
};

const getOneWorkout = (req, res) => {
  const {
    params: { workoutId },
  } = req;

  if (!workoutId) {
    res.status(400).send({
      status: "FAILED",
      data: { error: "Parameter ':workoutId' can not be empty" },
    });
    return;
  }

  try {
    const workout = workoutService.getOneWorkout(workoutId);
    res.send({ status: "OK", data: workout });
  } catch (error) {
    res
      .status(error?.status || 500)
      .send({ status: "FAILED", data: { error: error?.message || error } });
  }
};

const createNewWorkout = (req, res) => {
  const { body } = req;

  if (
    !body.name ||
    !body.mode ||
    !body.equipment ||
    !body.exercises ||
    !body.trainerTips
  ) {
    res.status(400).send({
      status: "FAILED",
      data: {
        error:
          "One of the following keys is missing or is empty in request body: 'name', 'mode', 'equipment', 'exercises', 'trainerTips'",
      },
    });
  }

  const newWorkout = {
    name: body.name,
    mode: body.mode,
    equipment: body.equipment,
```



```
ontrollers > JS workoutController.js > [@] updateOneWorkout > [@] updatedWorkout
const createNewWorkout = (req, res) => {
  } catch (error) {
    res
      .status(error?.status || 500)
      .send({ status: "FAILDED", data: { error: error?.message || error } });
  }
};

const updateOneWorkout = (req, res) => {
  const {
    body,
    params: { workoutId },
  } = req;

  if (!workoutId) {
    res.status(400).send({
      status: "FAILED",
      data: { error: "Parameter ':workoutId' can not be empty" },
    });
  }

  try {
    const updatedWorkout = workoutService.updateOneWorkout(workoutId, body);
    res.send({ status: "OK", data: updatedWorkout });
  } catch (error) {
    res
      .status(error?.status || 500)
      .send({ status: "FAILED", data: { error: error?.message || error } });
  }
};

const deleteOneWorkout = (req, res) => {
  const {
    params: { workoutId },
  } = req;

  if (!workoutId) {
    res.status(400).send({
      status: "FAILED",
      data: { error: "Parameter ':workoutId' can not be empty" },
    });
  }

  try {
    workoutService.deleteOneWorkout(workoutId);
    res.status(204).send({ status: "OK" });
  } catch (error) {
    res
      .status(error?.status || 500)
      .send({ status: "FAILED", data: { error: error?.message || error } });
  }
};

module.exports = {
  getAllWorkouts,
  getOneWorkout,
  createNewWorkout,
  updateOneWorkout,
  deleteOneWorkout,
};
```

Resultat final:



Swagger

Crossfit WOD API 1.0.0 OAS3

Workouts

GET /api/v1/workouts

Schemas

Workout