

Creació d'una API web amb Node.js

Aquesta pràctica consistirà en crear una petita API REST amb Node.js. S'avaluarà mitjançant un lliurament de la pràctica molt senzill. Servirà per puntuar la UF3 i la UF4.

Examen de la pràctica: dia 18/5 (5 punts preguntes + 5 punts programari)

Software que heu de tenir instal·lat:

- Visual Studio Code (o l'IDE que preferiu, però jo us recomano aquest)
- Git
- Node.js (des d'[aquest link](#))
- Postman o Insomnia per provar la API

IDE a utilitzar: Visual Studio Code amb els plugins de Node.js i Express

Tutorial i documentació que heu de seguir:

<https://docs.microsoft.com/ca-es/learn/modules/build-web-api-nodejs-express/>

Es tracta d'anar seguint aquest tutorial mitjançant les instruccions que hi ha a continuació, però procurant comprendre el codi i què fa cada fitxer, així com les comandes de node.js que emprarem via terminal. Ja que això serà important de cara a l'examen: comprendre què fa cada cosa i no simplement copiar el codi del tutorial.

PRIMERA PART: Instruccions

- Llegir i entendre les pàgines 1 i 2 del tutorial, que contenen teoria
- Instalar node.js
- Crear un nou repo a github per aquest projecte
- Seguir les instruccions de la pàgina 3 del tutorial: "Creación de una aplicación web básica con Express"
- Després del punt 4 de les instruccions de la pàgina 3, podeu fer el primer commit i push.
- En aquest moment, haurieu de tenir 3 fitxers: **app.js**, **package.json** i **package-lock.json**
- Investigueu què fan els fitxers **package.json** i **package-lock.json**.
 - package.json registra informació de configuració del projecte (nom, versió, llicència...etc) així com les dependències necessàries.
 - package-lock.json bloqueja les actualitzacions a les dependències del projecte perquè tots els desenvolupadors facin servir les mateixes versions d'aquestes.
- Investigueu què és la carpeta **node_modules**.

És una carpeta que es crea a la carpeta arrel del nostre projecte quan executem:

```
npm install express
```

Aquesta carpeta conté tots els paquets bàsics que necessitem pel nostre projecte de forma local.

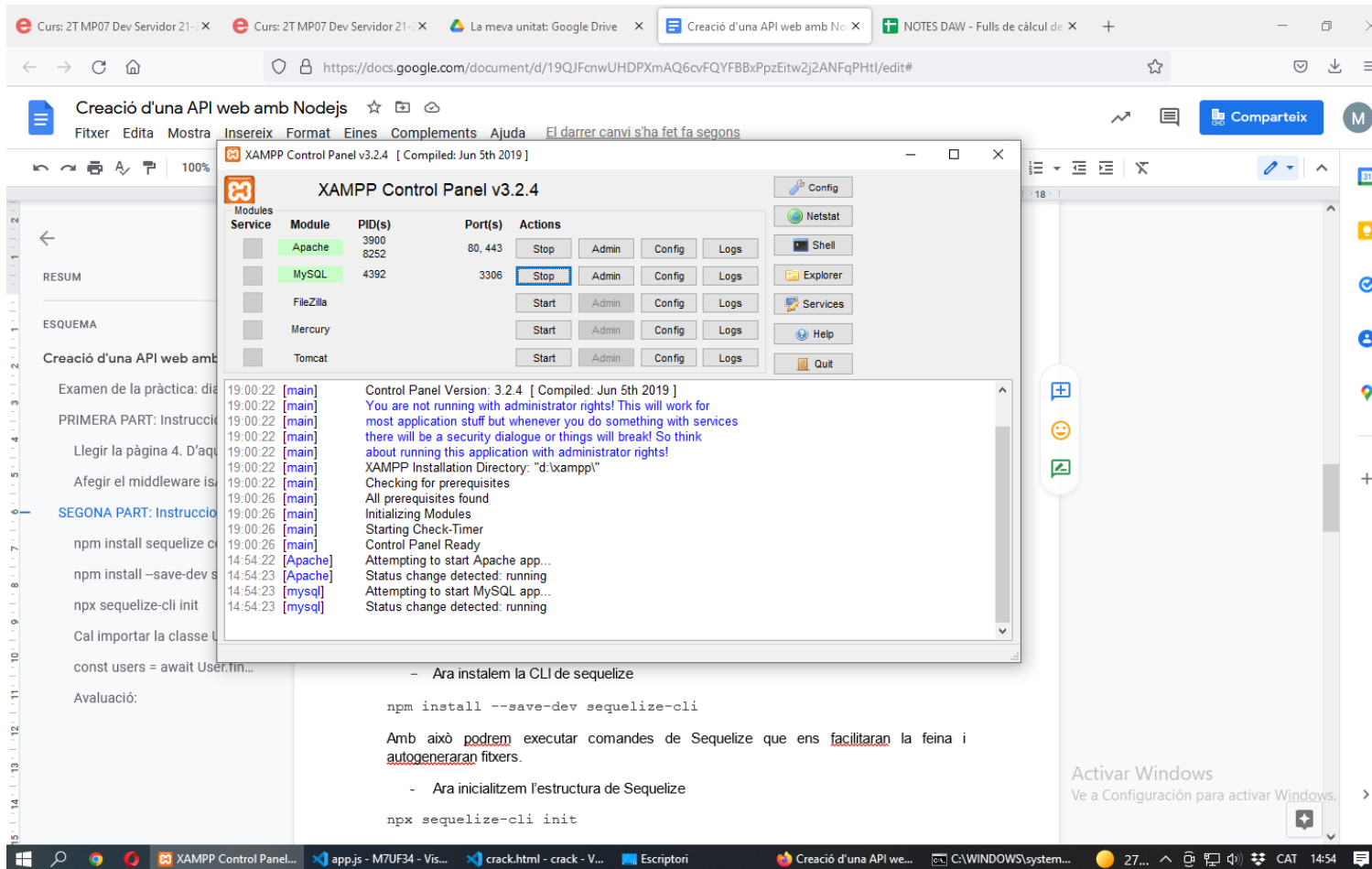
- Executar i veure que funciona. Acte seguit podem fer commit i push
- Seguir les instruccions de la pàgina 3 del tutorial: "Creación de una aplicación web que devuelve datos JSON"
- Executar i veure que funciona tal com es descriu al tutorial. Provar l'endpoint productes amb el navegador i també amb postman.
- Fer commit i push.
- Llegir la pàgina 4. D'aquí és important comprendre què són els paràmetres **req**, **res** i **next**.
 - req → request, sol·licitud entrant, encapçalats i adreça URL, pot contenir cos amb dades.
 - res → response, resposta, escriure info, encapçalats i dades a retornar.
 - next() → sol·licitud correcta, llesta per processar. Si no, indicar amb missatge.
- Aneu a la pàgina 5 però no us baixeu el repo que diu. Simplement modifiqueu el vostre **app.js** perquè sigui com el d'aquesta pàgina. També haureu de crear l'arxiu **client.js** i provar-lo tal com diu.
- Un cop hàgiu provat tot el que diu en aquesta pàgina, obriu el postman (o insomnia) i feu una crida al get de **/users**. Investigueu com fer-ho perquè funcioni i retorni el llistat d'usuaris.
- Afegir el middleware **isAuthorized** de la pàgina 5
- Finalment, contesteu el qüestionari de la pàgina 6
- Un cop hàgiu fet tot això, podeu fer un últim push. Recordeu penjar al moodle el link al vostre repo.

SEGONA PART: Instruccions

Volem aconseguir:

Que la informació de la resposta de **/users** surti d'una BD SQL que podeu instanciar amb phpmyadmin (necessitareu el port de la BD).

- Obriu el XAMPP i inicialitzeu l'Apache i el MySQL



Un cop tenim la BD encesa i funcionant, tornem al VS Code i seguim la part de programació. Utilitzarem la llibreria **Sequelize** per *express*. La podeu instal·lar com a paquet del projecte i definir els models pertinents, a més de la connexió amb la BD.

- Sequelize és un ORM (object relational mapper)
- Per començar a treballar amb la BD, executem dins del projecte:

```
npm install sequelize cors mysql2
```

npm install serveix per instal·lar paquets (llibreries) de node.js dins d'un projecte.

Mireu com han canviat els fitxers *package.json*, *package-lock.json*, i el directori *node_modules*. Haurien d'haver aparegut les 3 llibreries instal·lades.

- Ara instaleu la CLI de sequelize

```
npm install --save-dev sequelize-cli
```

Amb això podrem executar comandes de Sequelize que ens facilitaran la feina i autogeneraran fitxers.

- Ara inicialitzem l'estructura de Sequelize

```
npx sequelize-cli init
```

Haurien d'haver aparegut, dins del vostre projecte, els directoris models, seeders, migrations i config

- Crear el Model Usuari:

```
npx sequelize-cli model:generate --name User --attributes
username:string,password:string,email:string
```

Entre al fitxer que s'ha generat a */models* i fixeu-vos en l'estructura del fitxer.

- Executar migracions:

```
npx sequelize-cli db:migrate
```

Ara entreu al phpmyadmin i mireu l'estat de la vostra BD, hauria de tenir una nova taula Users amb els atributs username, password i email

- Generar un seeder d'usuaris de mentida:

```
npx sequelize-cli seed:generate --name demo-user
```

Modificar l'arxiu generat dins de seeders a la part await

```
module.exports = {
  async up (queryInterface, Sequelize) {
    /**
     * Add seed commands here.
     *
     * Example:*/
    await queryInterface.bulkInsert('users',
    [
      {
        username: "Jaime",
        password: "jamón",
        email: "jaime@gmail.com"
      },
      {
        username: "Erik",
        password: "jabugo",
        email: "erik@gmail.com"
      },
      {
        username: "Marcello",
        password: "krellar",
        email: "marcello@gmail.com"
      },
      {
        username: "Marc",
        password: "molinoNuevo",
        email: "marc@gmail.com"
      },
    ], {});
  },
};
```

Això genera un fitxer a /seeders amb el qual

- Executar el seeder

```
npx sequelize-cli db:seed:all
```

- Ara substituïm el codi de /users perquè retorni les dades reals de la BD

Cal importar la classe User de ./models/users.js i podem utilitzar el mètode *findAll()*

```
const { User } = require("../models");

app.get("/users", isAuthenticated, async (req, res) => {
  const users = await User.findAll();

  res.json(users);
});
```

Ara si executem un altre cop la crida GET /users des de POSTMAN, ens hauria de donar el JSON amb l'array d'usuaris que hi ha a la BD.

Proveu de modificar de nou la BD manualment, esborrar, afegir o modificar algun usuari. I torneu a fer la crida /users. Us hauria de donar sempre la informació de la BD.

Avaluació:

Pugeu a la tasca del moodle el link al vostre repo.

- La primera part de la pràctica són 5 punts, i la segona 5 punts més

De cara a l'examen:

- Les preguntes seran sobre:
 - Què és una API, per què serveixen, donar algun exemple
 - Què és el directori node_modules
 - De què serveix el package.json i quina diferència té amb el package-lock.json
 - Com s'instal·len paquets de l'entorn npm i quina utilitat tenen
 - Què són i què fan els paràmetres req, res i next d'una petició http en node.js
 - Què és un middleware i donar algun exemple.
 - La part de programar consistirà en fer el mateix que heu treballat a la pràctica:
 - Crear una app en Express amb Node.js.
 - Fer un endpoint que retorni informació inventada en format JSON
 - Afegir, a aquest endpoint, un middleware d'autenticació per contrasenya
 - Alguna cosa extra per a qui vulgui nota...

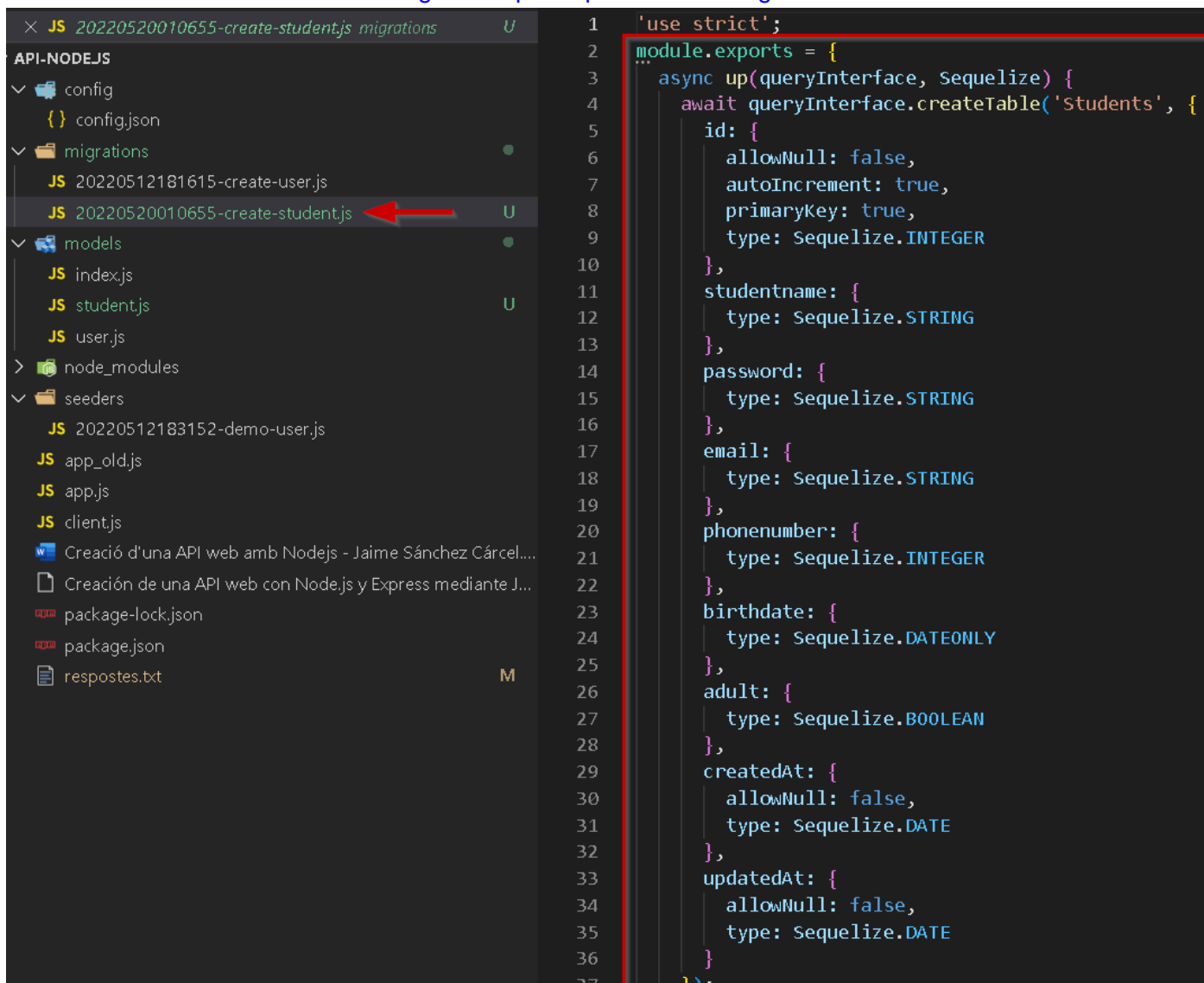
- Us heu d'inventar 1 model més que no sigui User, que tingui com a mínim 4 atributs (sense comptar id, createdAt, updatedAt) i que tingui com a mínim dos tipus d'atribut diferent (que no siguin tot string com a User...)

1. Crear el model de dades Student:

`npx sequelize-cli model:generate --name Student --attributes`

`studentname:string,password:string,email:string,phonenumber:integer,birthdate:dateonly,adult:boolean`

2. Crear la taula executant la migració: `npx sequelize-cli db:migrate`

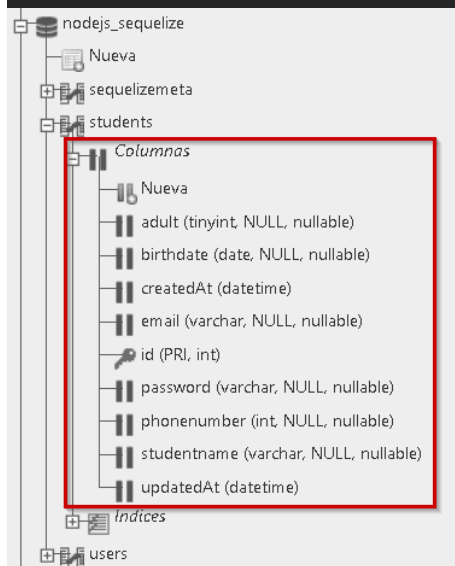
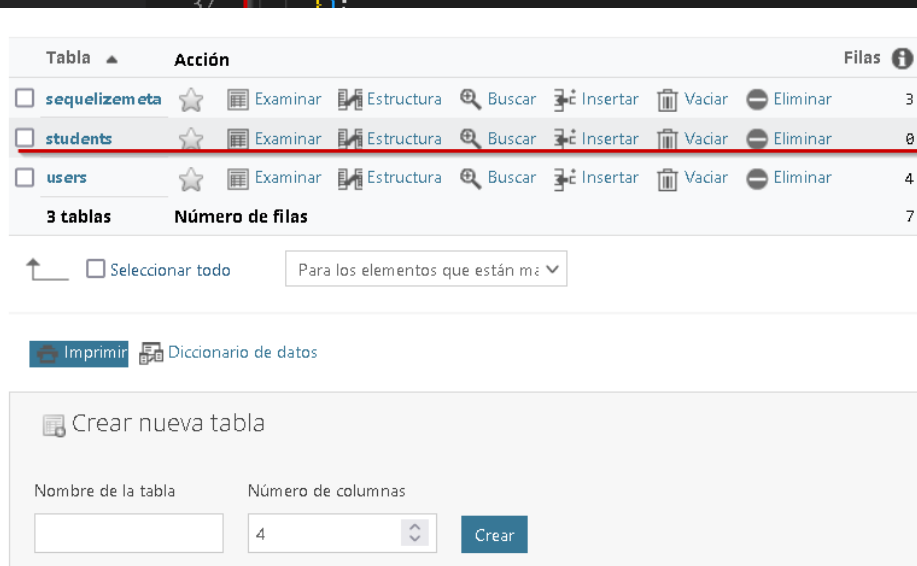


```

'use strict';

module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.createTable('Students', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      studentname: {
        type: Sequelize.STRING
      },
      password: {
        type: Sequelize.STRING
      },
      email: {
        type: Sequelize.STRING
      },
      phonenumber: {
        type: Sequelize.INTEGER
      },
      birthdate: {
        type: Sequelize.DATEONLY
      },
      adult: {
        type: Sequelize.BOOLEAN
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE
      }
    });
  },
  async down(queryInterface, Sequelize) {
    // ...
  }
};

```













3. Generar el seeder (sembrador) dels estudiants, modificar i ficar dades de prova:

`npx sequelize-cli seed:generate --name demo-student`

```
await queryInterface.bulkInsert('students', [
  {
    studentname: "Jaime Sánchez",
    password: "123456",
    email: "jaime.sc@gmail.com",
    phonenumber: 666111222,
    birthdate: "1975-05-10",
    adult: true,
  },
  {
    studentname: "Erk Tingsvall",
    password: "654321",
    email: "erik.tingsvall@gmail.com",
    phonenumber: 666999888,
    birthdate: "1994-01-01",
    adult: true,
  },
  {
    studentname: "Marcello Krell",
    password: "456789",
    email: "marcelo.krell@gmail.com",
    phonenumber: 777444999,
    birthdate: "2006-10-10",
    adult: false,
  },
  {
    studentname: "Marc Molinuevo",
    password: "987654",
    email: "marc.molinuevo@gmail.com",
    phonenumber: 777111555,
    birthdate: "1996-07-07",
    adult: true,
  },
], {});
```

4. Executar el seeder perque faci el INSERT INTO de students:

`npx sequelize-cli db:seed:all`

←T→											
			id	studentname	password	email	phonenumber	birthdate	adult	createdAt	updatedAt
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	1	Jaime Sánchez	123456	jaime.sc@gmail.com	666111222	1975-05-10	1	0000-00-00 00:00:00 0000-00-00 00:00:00
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	2	Erk Tingsvall	654321	erik.tingsvall@gmail.com	666999888	1994-01-01	1	0000-00-00 00:00:00 0000-00-00 00:00:00
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	3	Marcello Krell	456789	marcelo.krell@gmail.com	777444999	2006-10-10	0	0000-00-00 00:00:00 0000-00-00 00:00:00
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	4	Marc Molinuevo	987654	marc.molinuevo@gmail.com	777111555	1996-07-07	1	0000-00-00 00:00:00 0000-00-00 00:00:00

- Heu d'afegir una ruta GET que retorni totes les instàncies del nou model que heu creat. Per exemple si heu creat el model Estudiant, heu de fer un GET /estudiants (o /students) que retorni tots els estudiants que hi hagi en aquell moment a la BD.
- Afegiu, a aquesta nova ruta, un middleware que faci el següent: Que imprimeixi, per consola, un missatge cada cop que li arriba una petició amb la data i moment en què s'ha fet. Per exemple "S'han demanat els estudiants el 19/05/2022 a les 13:59".

```
// Script de l'exercici studentsApp.js
const express = require("express"); // Importació de la llibreria d'express
const app = express(); // Instància de la classe express
const intPort = 3000; // Port de les comunicacions
const { Student } = require("./models"); // Instància del model

/* Amb el verb "get" fem una petició per demanar les dades,
   abans executarà la funció que fa de middleware (intermediari) */

app.get("/students", printDateTimeRequest, async (req, res) => {
  const students = await Student.findAll();

  res.json(students);
});

/* Imprimeix per consola el missatge amb la data i
   l'hora i continua amb l'execució amb el mètode next() */
function printDateTimeRequest(req, res, next) {
  let dateNow = new Date();

  dateNowFormatted = "" +
    dateNow.getDate().toString().padStart(2, '0') + "/" +
    (dateNow.getMonth()+1).toString().padStart(2, '0') + "/" +
    dateNow.getFullYear().toString().padStart(4, '0') + " a les " +
    dateNow.getHours().toString().padStart(2, '0') + ":" +
    dateNow.getMinutes().toString().padStart(2, '0');

  console.log(`S'han demanat els estudiants el ${dateNowFormatted}`);

  next();
}

app.listen(intPort, () => console.log(`Example app listening on port ${intPort}`));
```


The image shows a web browser window with a REST client interface. The URL bar shows `localhost:3000/students` with a red circle 1 next to it. The `Send` button has a red circle 3 next to it. The `Body` tab is selected, showing a JSON array of three student objects. A red box highlights the first object, and a red circle 5 is next to the `Visualize` button. The JSON data is as follows:

```
[
  {
    "id": 1,
    "studentname": "Jaime Sánchez",
    "password": "123456",
    "email": "jaime.sc@gmail.com",
    "phonenumber": 666111222,
    "birthdate": "1975-05-10",
    "adult": true,
    "createdAt": null,
    "updatedAt": null
  },
  {
    "id": 2,
    "studentname": "Erk Tingsvall",
    "password": "654321",
    "email": "erik.tingsvall@gmail.com",
    "phonenumber": 666999888,
    "birthdate": "1994-01-01",
    "adult": true,
    "createdAt": null,
    "updatedAt": null
  },
  {
    "id": 3,
    "studentname": "Marcello Krell",
    "password": "456789",
    "email": "marcelo.krell@gmail.com",
  }
]
```

Below the browser window is a terminal window showing the execution of the `node studentsApp.js` command. The terminal output shows the application starting and listening on port 3000. A red circle 2 is next to the command, and a red circle 4 is next to the timestamp `022 a les 04:31`.

- Afegiu en un fitxer `trespuntsextra.txt` una explicació amb les vostres paraules de com heu aconseguit fer els 3 punts anteriors.