Solución Prueba - Skate Park

- ***Paso 1:*** Creación y configuración del servidor, incluyendo las rutas que renderizan vistas con contenido dinámico. (Requerimientos abordados: 2)

index.js

```
// Importaciones
const express = require("express");
const app = express();
const exphbs = require("express-handlebars");
const expressFileUpload = require("express-fileupload");
const jwt = require("jsonwebtoken");
const secretKey = "Shhhh";
const {
    newSkater,
    getSkaters,
    getSkater,
    updateSkater,
    deleteSkater,
    setSkaterStatus,
} = require("./consultas");



// Server
app.listen(3000, () => console.log("Servidor encendido!"));

// Middlewares
app.use(express.urlencoded({ extended: false }));
app.use(express.json());
app.use(express.static(__dirname + "/public"));
app.use(
    expressFileUpload({
        limits: 5000000,
        abortOnLimit: true,
        responseOnLimit: "El tamaño de la imagen supera el límite
permitido",
    })
);
app.use("/css", express.static(__dirname +
```

```
"/node_modules/bootstrap/dist/css"));
app.engine(
    "handlebars",
    exphbs({
        defaultLayout: "main",
        layoutsDir: `${__dirname}/views/mainLayout`,
    })
);
app.set("view engine", "handlebars");

// Rutas
app.get("/", async (req, res) => {
    try {
        const skaters = await getSkaters()
        res.render("Home", { skaters });
    } catch (e) {
        res.status(500).send({
            error: `Algo salió mal... ${e}`,
            code: 500
        })
    };
});

app.get("/registro", (req, res) => {
    res.render("Registro");
});

app.get("/perfil", (req, res) => {
    const { token } = req.query
    jwt.verify(token, secretKey, (err, skater) => {
        if (err) {
            res.status(500).send({
                error: `Algo salió mal...`,
                message: err.message,
                code: 500
            })
        } else {
            res.render("Perfil", { skater });
        }
    })
});

app.get("/login", (req, res) => {
    res.render("Login");
```

```
});

app.post("/login", async (req, res) => {
    const { email, password } = req.body
    try {
        const skater = await getSkater(email, password)
        const token = jwt.sign(skater, secretKey)
        res.status(200).send(token)
    } catch (e) {
        console.log(e)
        res.status(500).send({
            error: `Algo salió mal... ${e}`,
            code: 500
        })
    };
});

app.get("/Admin", async (req, res) => {
    try {
        const skaters = await getSkaters();
        res.render("Admin", { skaters });
    } catch (e) {
        res.status(500).send({
            error: `Algo salió mal... ${e}`,
            code: 500
        })
    };
});
```

- ***Paso 2:*** Creación de la API REST de skaters. (Requerimientos abordados: 1)

index.js

```
// API REST de Skaters

app.get("/skaters", async (req, res) => {

    try {
        const skaters = await getSkaters()
        res.status(200).send(skaters);
    } catch (e) {
        res.status(500).send({
            error: `Algo salió mal... ${e}`,
            code: 500
        })
    };
});

app.post("/skaters", async (req, res) => {
    const skater = req.body;
    if (Object.keys(req.files).length == 0) {
        return res.status(400).send("No se encontro ningun archivo en la
consulta");
    }
    const { files } = req
    const { foto } = files;
    const { name } = foto;
    const pathPhoto = `/uploads/${name}`
    foto.mv(`${__dirname}/public${pathPhoto}`, async (err) => {
        try {
            if (err) throw err
            skater.foto = pathPhoto
            await newSkater(skater);
            res.status(201).redirect("/login");
        } catch (e) {
            console.log(e)
            res.status(500).send({
                error: `Algo salió mal... ${e}`,
                code: 500
            })
        };

    });
```

```
})

app.put("/skaters", async (req, res) => {
    const skater = req.body;
    try {
        await updateSkater(skater);
        res.status(200).send("Datos actualizados con éxito");
    } catch (e) {
        res.status(500).send({
            error: `Algo salió mal... ${e}`,
            code: 500
        })
    };
});

app.put("/skaters/status/:id", async (req, res) => {
    const { id } = req.params;
    const { estado } = req.body;
    try {
        await setSkaterStatus(id, estado);
        res.status(200).send("Estatus de skater cambiado con éxito");
    } catch (e) {
        res.status(500).send({
            error: `Algo salió mal... ${e}`,
            code: 500
        })
    };
});

app.delete("/skaters/:id", async (req, res) => {
    const { id } = req.params
    try {
        await deleteSkater(id)
        res.status(200).send();
    } catch (e) {
        res.status(500).send({
            error: `Algo salió mal... ${e}`,
            code: 500
        })
    };
});
```

consultas.js

```
const { Pool } = require("pg");
const pool = new Pool({
  user: "postgres",
  host: "localhost",
  password: "postgres",
  database: "skatepark",
  port: 5432,
});

const newSkater = async (skater) => {
  const values = Object.values(skater)
  const result = await pool.query(
    `INSERT INTO skaters  ( email  ,  nombre  ,  password  ,
anos_experiencia , especialidad  , foto  , estado ) values ($1,$2, $3,
$4 ,$5, $6, 'f') RETURNING *`
    , values);
  return result.rows[0];
}

const updateSkater = async (skater) => {
  const values = Object.values(skater)
  const result = await pool.query(
    `UPDATE skaters SET  nombre = $1, password = $2 , anos_experiencia =
$3 , especialidad = $4   RETURNING *`
    , values);
  return result.rows[0];
}


const getSkaters = async () => {
  const result = await pool.query(`SELECT * FROM skaters`);
  return result.rows;
}

const getSkater = async (email, password) => {
  const result = await pool.query(
    `SELECT * FROM skaters WHERE email = '${email}' AND password =
'${password}'`
  );

  return result.rows[0];
}
```

```
const setSkaterStatus = async (id, estado) => {
  const result = await pool.query(
    `UPDATE skaters SET estado = ${estado} WHERE id = ${id} RETURNING *`
  );
  const skater = result.rows[0];
  return skater;
}

const deleteSkater = async (id) => {
  const result = await pool.query(
    `DELETE FROM skaters WHERE id = ${id} RETURNING *`
  );
  const skater = result.rows[0];
  return skater;
}


module.exports = {
  newSkater,
  getSkaters,
  getSkater,
  setSkaterStatus,
  updateSkater,
  deleteSkater
};
```

- ***Paso 3:*** Inclusión de la funcionalidad Upload File en la ruta POST /skaters para subir la foto de perfil del skater. (Requerimientos abordados: 3)

index.js

```
if (Object.keys(req.files).length == 0) {
        return res.status(400).send("No se encontro ningun archivo en la
consulta");
    }
    const { files } = req
    const { foto } = files;
    const { name } = foto;
    const pathPhoto = `/uploads/${name}`
    foto.mv(`${__dirname}/public${pathPhoto}`, async (err) => {
        try {
            if (err) throw err
            skater.foto = pathPhoto
            await newSkater(skater);
            res.status(201).redirect("/login");
        } catch (e) {
            console.log(e)
            res.status(500).send({
                error: `Algo salió mal... ${e}`,
                code: 500
            })
        };

    });
```

- ***Paso 4:*** Restringir el contenido de la ruta /Perfil con JWT. (Requerimientos abordados: 4)

index.js

```
const { token } = req.query
jwt.verify(token, secretKey, (err, skater) => {
    if (err) {
        res.status(500).send({
            error: `Algo salió mal...`,
            message: err.message,
            code: 500
        })
    } else {
        res.render("Perfil", { skater });
    }
})
```