

INSTITUTO POLITÉCNICO NACIONAL



ESCUELA SUPERIOR DE CÓMPUTO

INGENIERÍA EN SISTEMAS COMPUTACIONALES

Reporte de mini proyecto “Prevención de Fuerza Bruta con Rate Limiting”

Equipo 9 “Firewallers”

Alumnos:

Escárcega Hernández Steven Arturo 2022630499

García Mayorga Rodrigo

Paz Nieves José Antonio

6CV3

Profesor Ojeda Santillán Rodrigo

Índice general

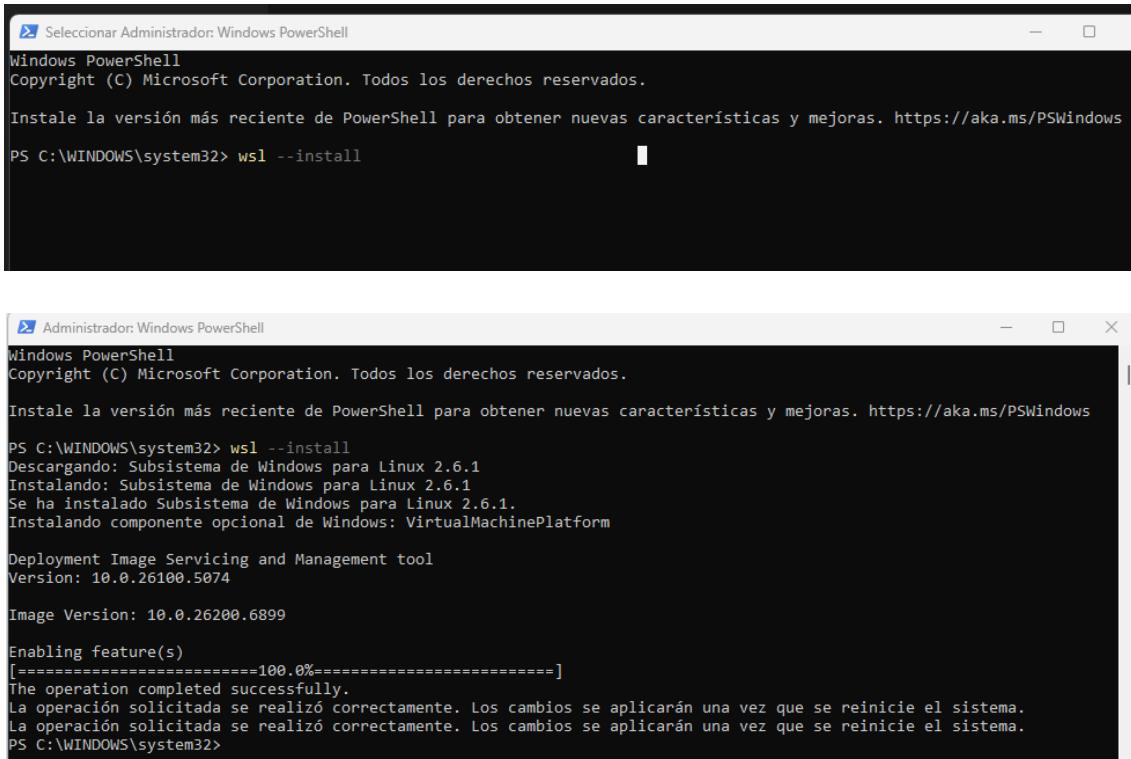
Instalar WSL2 (Windows PowerShell — como administrador)	4
Instalar Node.js + npm + herramientas.....	6
Instalación de Docker y habilitar WSL2	7
Abrir VS Code en WSL.....	9
Crear estructura del proyecto	12
Llenado de código	13
Prisma: migrate + generate	18
Rutas.....	18
Controladores.....	19
Vistas EJS.....	24
Middleware de Rate Limiting.....	26
Levantamos la app	26
Haciendo el proyecto más atractivo y mostramos el rol del usuario una vez iniciada la sesión.....	29
Pruebas de fuerza bruta	32

Índice de figuras

Ilustración 10. Configuramos la integración con WSL.....	8
Ilustración 12. Creación de directorio para el proyecto.....	9
Ilustración 13. VS Code con la carpeta del proyecto.....	10
Ilustración 4. Verificación de terminal con Ubuntu.....	10
Ilustración 15. Proyecto Node inicializado correctamente (npm init -y).....	11
Ilustración 16. Dependencias instaladas correctamente.....	11
Ilustración 17. Estructura del proyecto creada.....	12
Ilustración 18. Archivo prisma	13
Ilustración 19. Archivo .env	13
Ilustración 10. Archivo app.js	15
Ilustración 11. Instalación de nodemon	15
Ilustración 12. Archivo package.json.....	16
Ilustración 13. Archivo docker-compose.yml	17
Ilustración 14. Verificación del contenedor	17
Ilustración 15. Migraciones completas.....	18
Ilustración 16. Archivo authRoutes.js.....	18
Ilustración 17. Archivo adminRoutes.js	19

Ilustración 18. Archivo authController.js (Ver código completo)	21
Ilustración 19. Archivo adminController.js (Ver código completo)	23
Ilustración 20. Archivo login.ejs	24
Ilustración 21. Archivo register.ejs	24
Ilustración 22. Archivo.ejs	25
Ilustración 23. Archivo admin.ejs.....	25
Ilustración 24. Archivo rateLimiter.js	26
Ilustración 25. Aplicación escuchando en :3000.....	26
Ilustración 26. Vista de registro con usuario	27
Ilustración 27. Inicio de sesión inválido.....	27
Ilustración 28. Protección de rate Limiter	27
Ilustración 29. Login correcto	27
Ilustración 30. Introducimos en la consola el comando npx prisma studio	28
Ilustración 31. Nos abrira la UI de prisma	28
Ilustración 32. Confirmamos cambio de rol.....	28
Ilustración 33. Nueva vista de login.....	29
Ilustración 34. Nueva vista de register.....	29
Ilustración 35. Vista de usuario admin	29
Ilustración 36. Panel de administrador.....	30
Ilustración 37. CRUD de usuarios	30
Ilustración 38. Vista de usuario sin permisos.....	30
Ilustración 39. Acceso restringido a los menus de administrador	31
Ilustración 40. Directorio creado	32
Ilustración 41. Archivo de test de fuerza bruta	33
Ilustración 42. Pruebas completadas!.....	34

Instalar WSL2 (Windows PowerShell — como administrador)



The image contains two side-by-side screenshots of a Windows PowerShell window. Both windows have a title bar that says "Seleccionar Administrador: Windows PowerShell". The first window shows the initial state with the command "PS C:\WINDOWS\system32> wsl --install" entered. The second window shows the progress of the download and installation of the "Subsistema de Windows para Linux 2.6.1". It includes status messages like "Descargando:", "Instalando:", and "Se ha instalado Subsistema de Windows para Linux 2.6.1.". At the bottom, it shows the deployment tool version "Deployment Image Servicing and Management tool Version: 10.0.26100.5074" and the image version "Image Version: 10.0.26200.6899". The progress bar at the bottom of the second window indicates 100% completion.

Figure 1. Instalación y apertura de WSL (Ubuntu) en Windows.



Figure 2. Descargamos Ubuntu desde la Microsoft Store

```

Enter new UNIX username: steven
wsl: Failed to start the systemd user session for 'root'. See journalctl for more details.
New password:
Retype new password:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.6.87.2-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

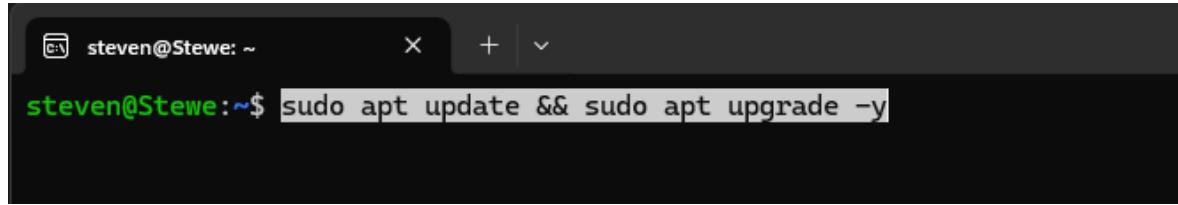
System information as of Fri Nov  7 19:38:33 CST 2025

System load: 0.79      Processes:          70
Usage of /: 0.1% of 1006.85GB   Users logged in:    0
Memory usage: 2%           IPv4 address for eth0: 172.24.102.57
Swap usage:  0%

This message is shown once a day. To disable it please create the
/home/steven/.hushlogin file.
steven@Stewe:~$ |

```

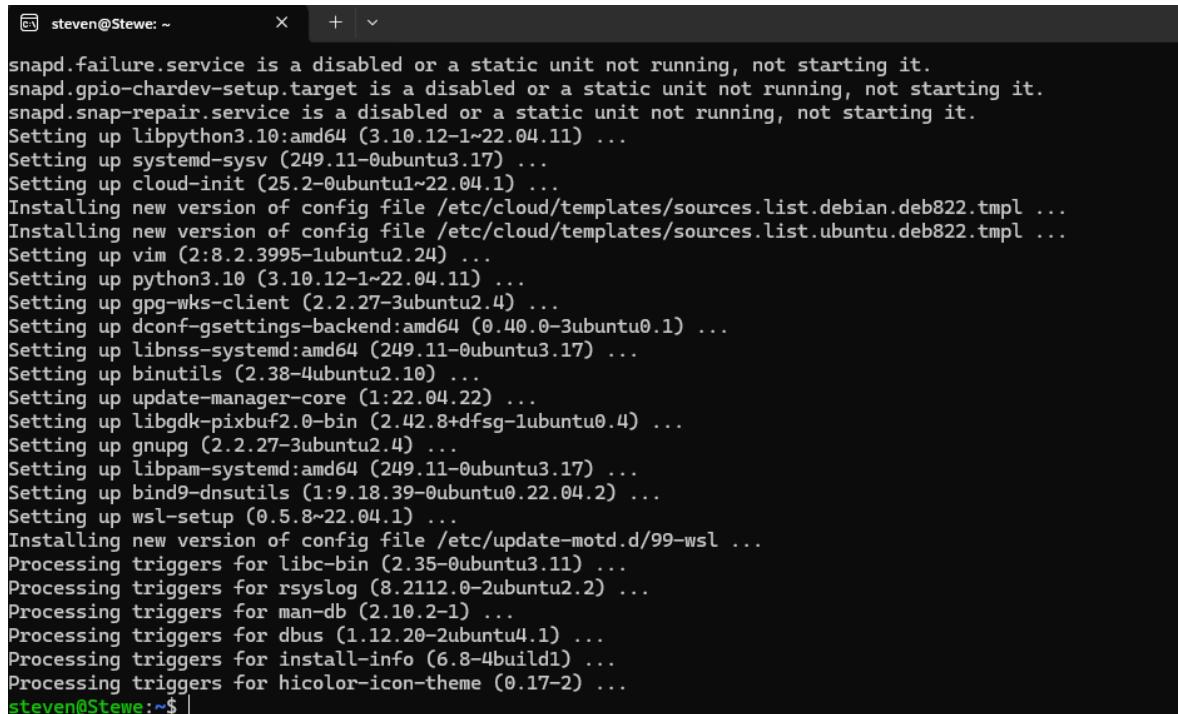
Figure 3. Ubuntu 22.04 LTS inicializado correctamente en WSL2



```

steven@Stewe:~$ sudo apt update && sudo apt upgrade -y

```



```

steven@Stewe:~$ 
snapd.failure.service is a disabled or a static unit not running, not starting it.
snapd.gpio-chardev-setup.target is a disabled or a static unit not running, not starting it.
snapd.snap-repair.service is a disabled or a static unit not running, not starting it.
Setting up libpython3.10:amd64 (3.10.12-1~22.04.11) ...
Setting up systemd-sysv (249.11-0ubuntu3.17) ...
Setting up cloud-init (25.2-0ubuntu1~22.04.1) ...
Installing new version of config file /etc/cloud/templates/sources.list.debian.deb822.tmpl ...
Installing new version of config file /etc/cloud/templates/sources.list.ubuntu.deb822.tmpl ...
Setting up vim (2:8.2.3995-1ubuntu2.24) ...
Setting up python3.10 (3.10.12-1~22.04.11) ...
Setting up gpg-wks-client (2.2.27-3ubuntu2.4) ...
Setting up dconf-gsettings-backend:amd64 (0.40.0-3ubuntu0.1) ...
Setting up libnss-systemd:amd64 (249.11-0ubuntu3.17) ...
Setting up binutils (2.38-4ubuntu2.10) ...
Setting up update-manager-core (1:22.04.22) ...
Setting up libgdk-pixbuf2.0-bin (2.42.8+dfsg-1ubuntu0.4) ...
Setting up gnupg (2.2.27-3ubuntu2.4) ...
Setting up libpam-systemd:amd64 (249.11-0ubuntu3.17) ...
Setting up bind9-dnsutils (1:9.18.39-0ubuntu0.22.04.2) ...
Setting up wsl-setup (0.5.8-22.04.1) ...
Installing new version of config file /etc/update-motd.d/99-wsl ...
Processing triggers for libc-bin (2.35-0ubuntu3.11) ...
Processing triggers for rsyslog (8.2112.0-2ubuntu2.2) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for dbus (1.12.20-2ubuntu4.1) ...
Processing triggers for install-info (6.8-4build1) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
steven@Stewe:~$ |

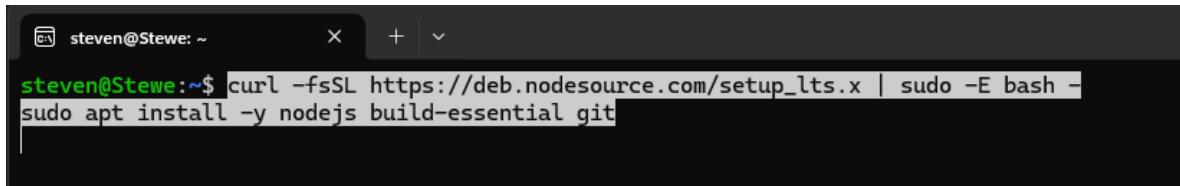
```

Figure 4. Revisión y actualización de paquetes

Instalar Node.js + npm + herramientas

Comandos:

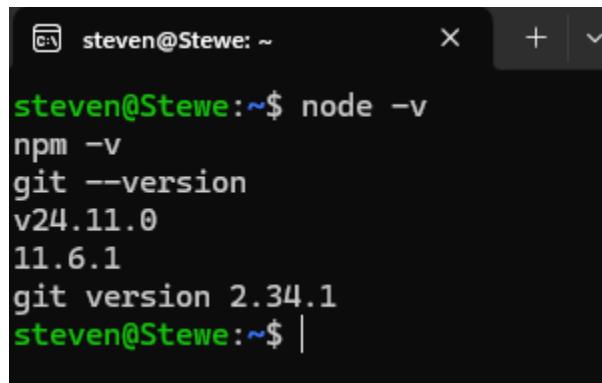
```
curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -  
sudo apt install -y nodejs build-essential git
```



A screenshot of a terminal window titled "steven@Stewe: ~". The window contains the following command:

```
steven@Stewe:~$ curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -  
sudo apt install -y nodejs build-essential git
```

Figure 5. Instalación de node.js y npm



A screenshot of a terminal window titled "steven@Stewe: ~". The window displays the following output of installed tools:

```
steven@Stewe:~$ node -v  
npm -v  
git --version  
v24.11.0  
11.6.1  
git version 2.34.1  
steven@Stewe:~$ |
```

Figure 6. Node, npm y Git funcionando correctamente en WSL

Instalación de Docker y habilitar WSL2

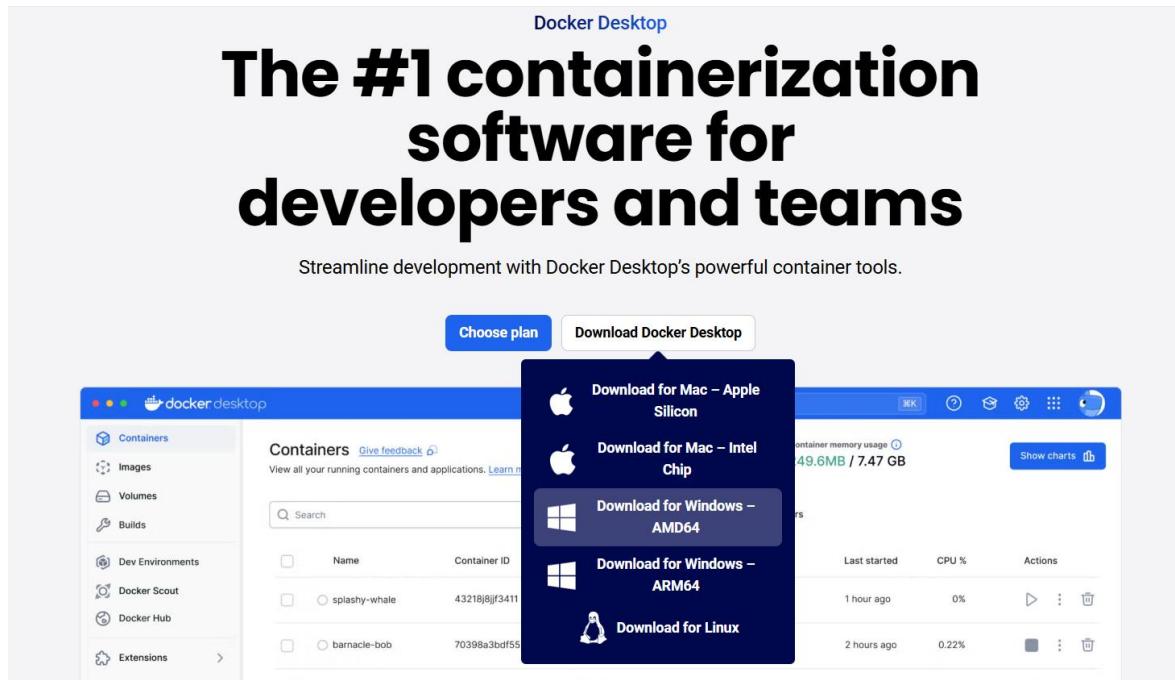


Figure 7. Descargamos la aplicación de docker desde su sitio oficial

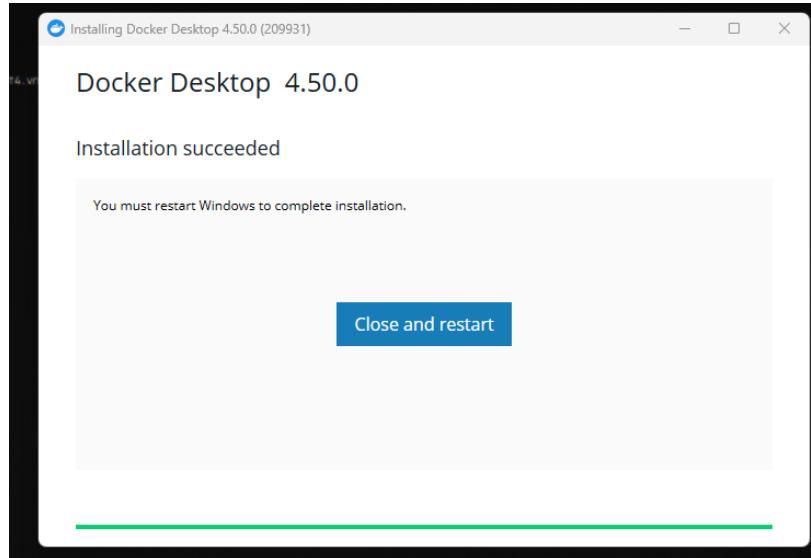


Figure 8. Instalación completa

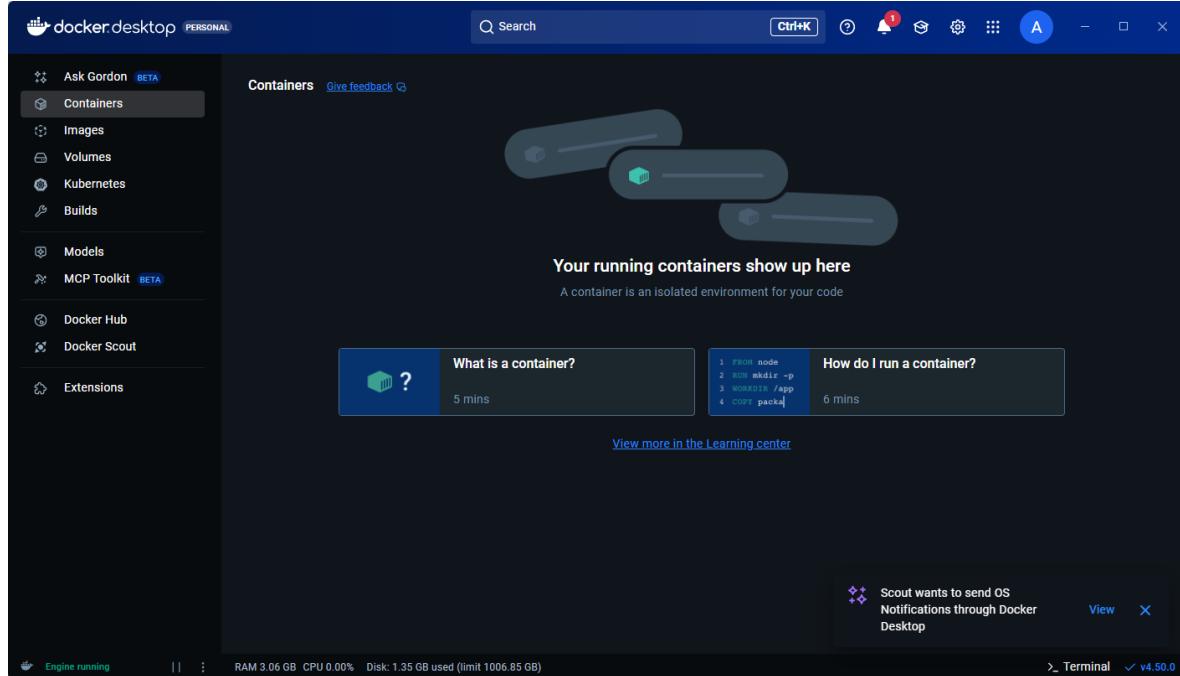


Ilustración 9. Interfaz de docker

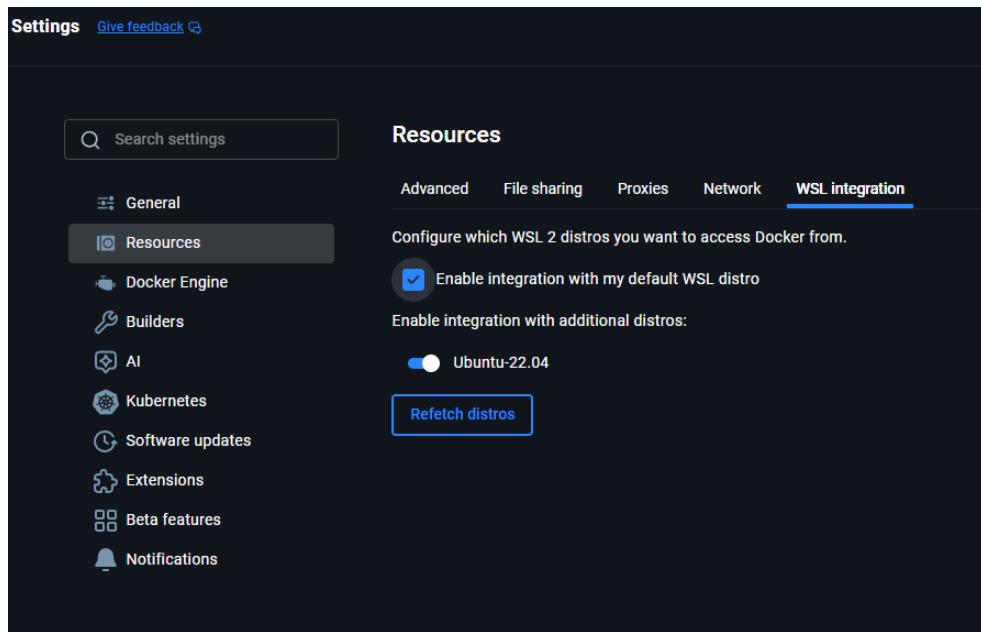
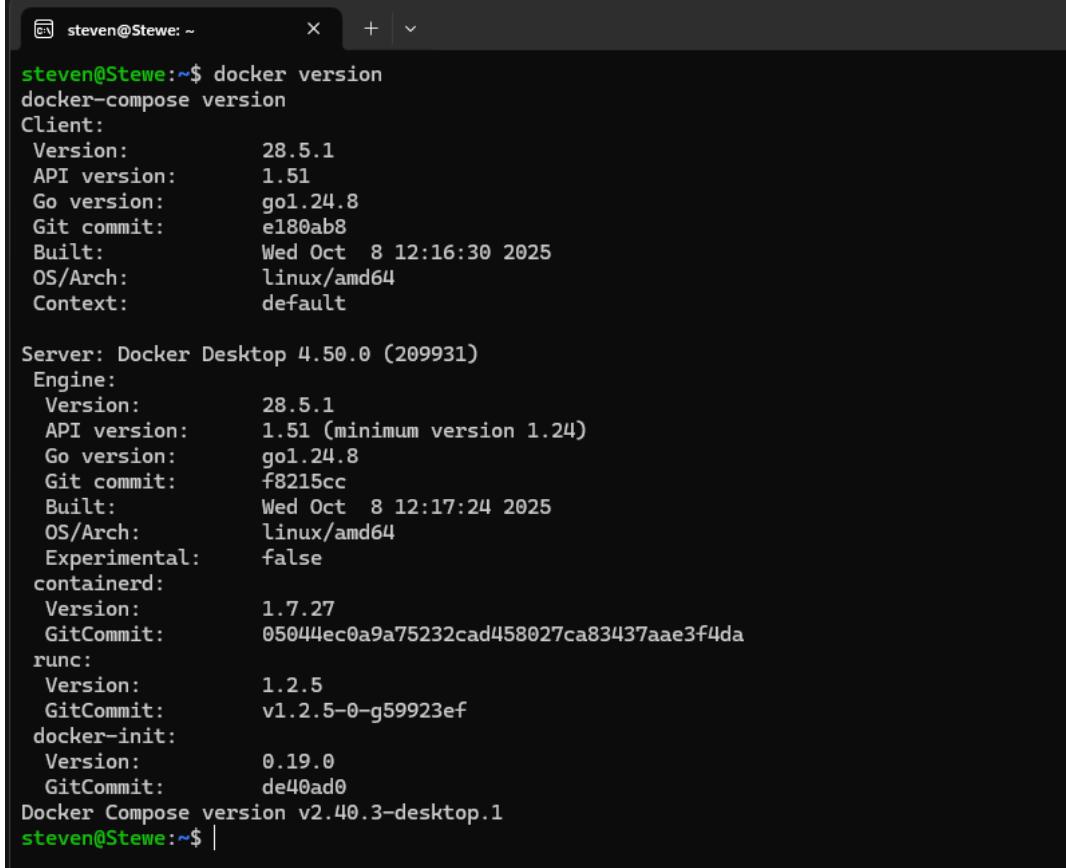


Ilustración 10. Configuramos la integración con WSL



```
steven@Stewe:~$ docker version
docker-compose version
Client:
  Version:          28.5.1
  API version:      1.51
  Go version:       go1.24.8
  Git commit:       e180ab8
  Built:            Wed Oct  8 12:16:30 2025
  OS/Arch:          linux/amd64
  Context:          default

Server: Docker Desktop 4.50.0 (209931)
Engine:
  Version:          28.5.1
  API version:      1.51 (minimum version 1.24)
  Go version:       go1.24.8
  Git commit:       f8215cc
  Built:            Wed Oct  8 12:17:24 2025
  OS/Arch:          linux/amd64
  Experimental:    false
containerd:
  Version:          1.7.27
  GitCommit:        05044ec0a9a75232cad458027ca83437aae3f4da
runc:
  Version:          1.2.5
  GitCommit:        v1.2.5-0-g59923ef
docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
Docker Compose version v2.40.3-desktop.1
steven@Stewe:~$ |
```

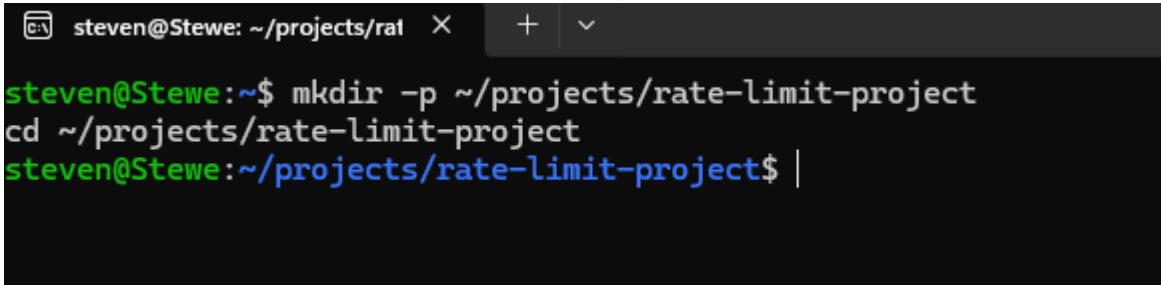
Ilustración 11. Verificamos desde la terminal de Ubuntu

Abrir VS Code en WSL

Creamos una carpeta para el proyecto:

```
mkdir -p ~/projects/rate-limit-project
```

```
cd ~/projects/rate-limit-project
```



```
steven@Stewe:~/projects/rat  X + ▾
steven@Stewe:~$ mkdir -p ~/projects/rate-limit-project
cd ~/projects/rate-limit-project
steven@Stewe:~/projects/rate-limit-project$ |
```

Ilustración 12. Creación de directorio para el proyecto

Abrimos VS Code con: **code .**

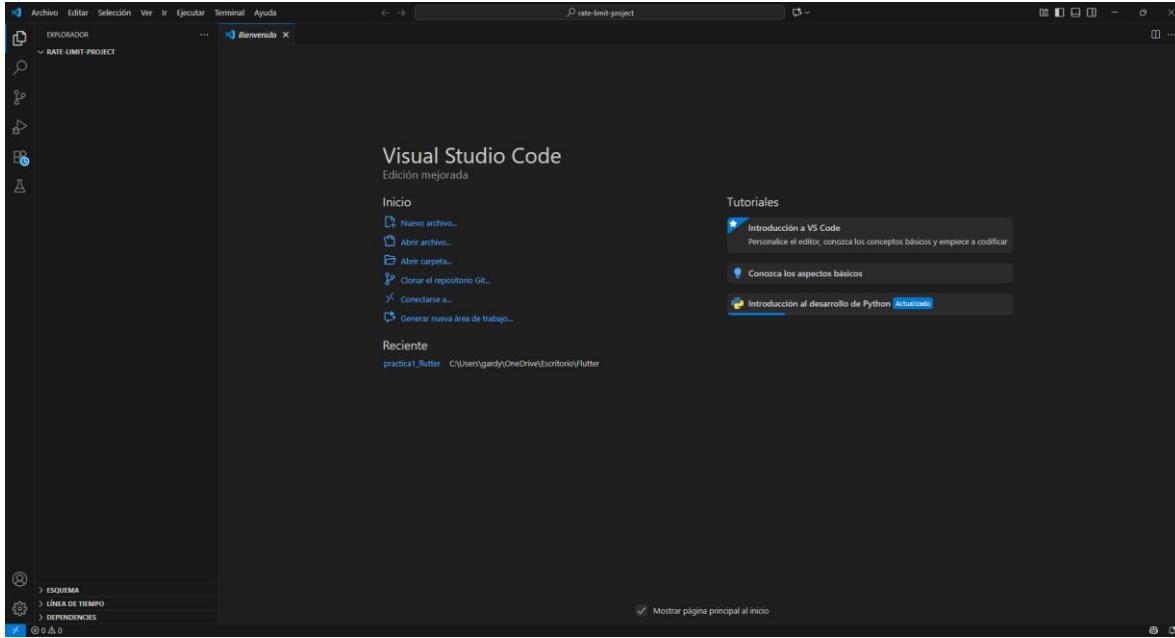


Ilustración 13. VS Code con la carpeta del proyecto

Abrimos una consola con **Control + N** y verificamos que estamos en la carpeta del proyecto:

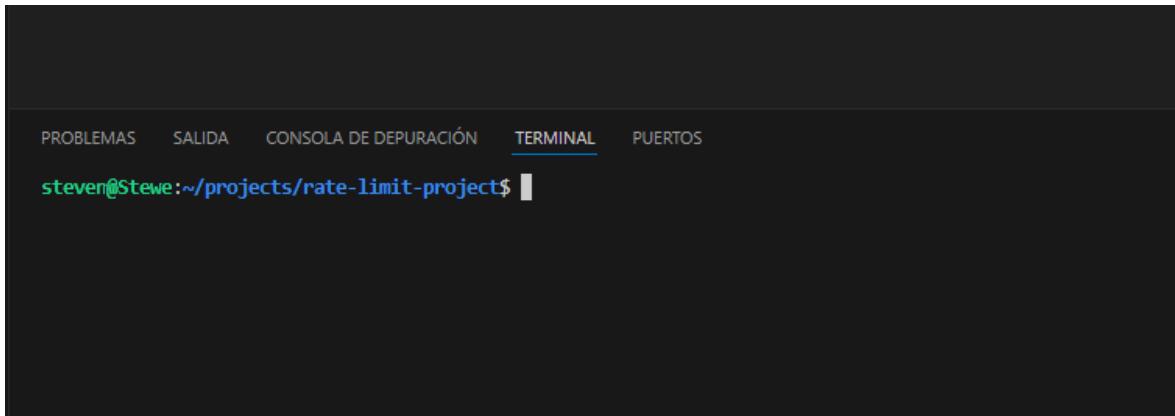


Ilustración 4. Verificación de terminal con Ubuntu

Pegamos dentro de la terminal: **npm init -y**

```
steven@Steve:~/projects/rate-limit-project$ npm init -y
Wrote to /home/steven/projects/rate-limit-project/package.json:

{
  "name": "rate-limit-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs"
}

steven@Steve:~/projects/rate-limit-project$ []
```

Ilustración 15. Proyecto Node inicializado correctamente (`npm init -y`)

Instalamos dependencias con:

```
npm install express express-session ejs bcrypt dotenv helmet express-rate-limit
@prisma/client
```

```
npm install prisma --save-dev
```

```
steven@Steve:~/projects/rate-limit-project$ npm install express express-session ejs bcrypt dotenv helmet express-rate-limit @prisma/client
npm install prisma --save-dev

added 91 packages, and audited 92 packages in 8s

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

found 0 vulnerabilities

added 33 packages, and audited 125 packages in 9s

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

found 0 vulnerabilities
steven@Steve:~/projects/rate-limit-project$ []
```

Ilustración 16. Dependencias instaladas correctamente

Crear estructura del proyecto

Pegamos en terminal para tener la estructura del proyecto:

```
mkdir -p prisma src/{routes,controllers,middleware,views}
touch prisma/schema.prisma
touch src/app.js
touch src/routes/authRoutes.js
touch src/routes/adminRoutes.js
touch src/controllers/authController.js
touch src/controllers/adminController.js
touch src/middleware/rateLimiter.js
touch src/views/login.ejs
touch src/views/register.ejs
touch src/views/user.ejs
touch src/views/admin.ejs
touch .env
touch README.md
```

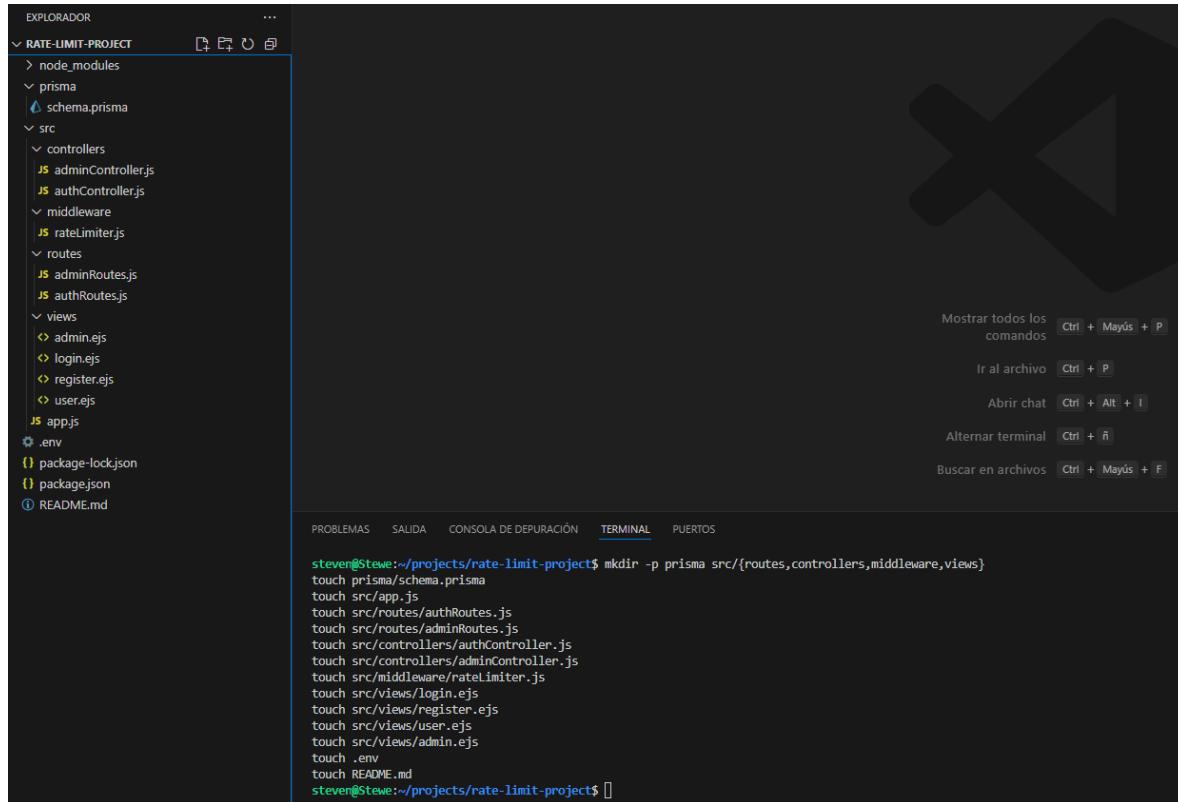


Ilustración 17. Estructura del proyecto creada

Llenado de código

Empezamos por lo más importante: **Prisma (schema.prisma)**

```

prisma > schema.prisma
  1 generator client {
  2   provider = "prisma-client-js"
  3 }
  4
  5 datasource db {
  6   provider = "postgresql"
  7   url      = env("DATABASE_URL")
  8 }
  9
 10 model User {
 11   id      Int      @id @default(autoincrement())
 12   email   String   @unique
 13   password String
 14   role    String   @default("user")
 15   createdAt DateTime @default(now())
 16 }
 17
 18

```

Ilustración 18. Archivo prisma

Seguimos con variables de entorno:

```

.env
  1 DATABASE_URL="postgresql://postgres:admin123@localhost:5432/ratelimit_db?schema=public"
  2 SESSION_SECRET="clave_segura_para_sesiones"
  3 PORT=3000
  4

```

Ilustración 19. Archivo .env

Configurar servidor Express:

Abrimos **app.js** y configuramos el servidor express con el siguiente código:

```

import express from "express";
import session from "express-session";
import helmet from "helmet";
import dotenv from "dotenv";
import rateLimit from "express-rate-limit";
import path from "path";
import { PrismaClient } from "@prisma/client";
import authRoutes from "./routes/authRoutes.js";
import adminRoutes from "./routes/adminRoutes.js";

```

```
dotenv.config();
const app = express();
const prisma = new PrismaClient();

app.use(helmet());
app.use(express.urlencoded({ extended: true }));

// Sesiones seguras
app.use(
  session({
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: false,
    cookie: { httpOnly: true, sameSite: "strict" },
  })
);

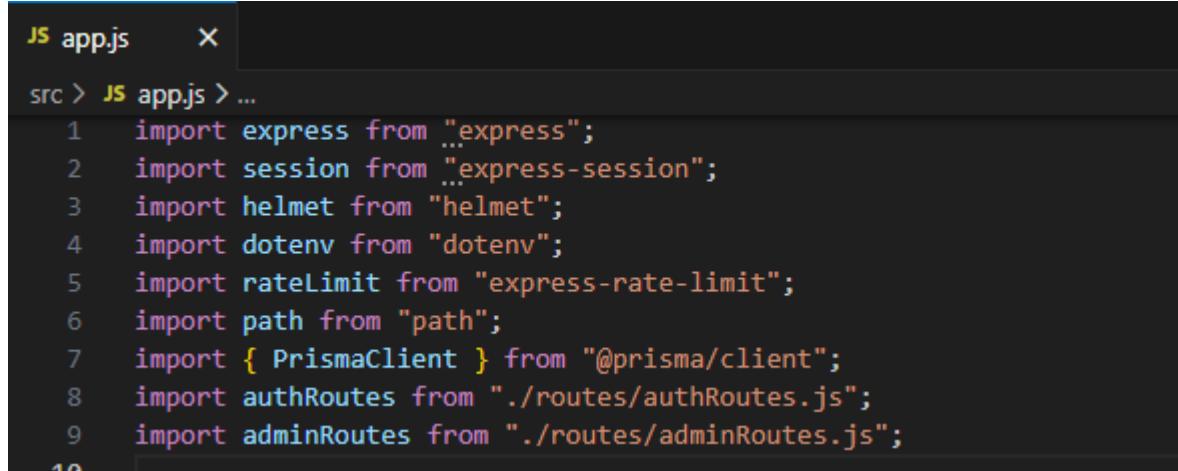
// Configuración del Rate Limiter
const loginLimiter = rateLimit({
  windowMs: 5 * 60 * 1000, // 5 minutos
  max: 5, // máximo 5 intentos
  message: "Demasiados intentos fallidos. Intenta de nuevo en 5 minutos.",
  standardHeaders: true,
  legacyHeaders: false,
});

app.use("/login", loginLimiter);

// Configurar vistas
app.set("view engine", "ejs");
app.set("views", path.join(process.cwd(), "src/views"));

app.use("/", authRoutes);
app.use("/admin", adminRoutes);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () =>
  console.log(`Servidor en http://localhost:${PORT}`)
);
```

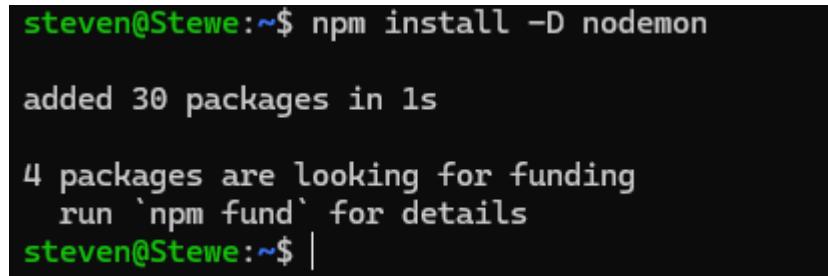


The screenshot shows a code editor window with the file 'app.js' open. The code is a Node.js application using Express. It imports various middleware and route handlers from other files. The imports include express, express-session, helmet, dotenv, rateLimit, path, PrismaClient, authRoutes, and adminRoutes.

```
JS app.js    X
src > JS app.js > ...
1 import express from "express";
2 import session from "express-session";
3 import helmet from "helmet";
4 import dotenv from "dotenv";
5 import rateLimit from "express-rate-limit";
6 import path from "path";
7 import { PrismaClient } from "@prisma/client";
8 import authRoutes from "./routes/authRoutes.js";
9 import adminRoutes from "./routes/adminRoutes.js";
```

Ilustración 10. Archivo app.js

Instalamos nodemon en WSL



```
steven@Stewe:~$ npm install -D nodemon
added 30 packages in 1s
4 packages are looking for funding
  run `npm fund` for details
steven@Stewe:~$ |
```

Ilustración 11. Instalación de nodemon

Cambiamos el package.json con las nuevas dependencias:

```
{ package.json X
  package.json > {} devDependencies > prisma
  1  {
  2    "name": "rate-limit-project",
  3    "version": "1.0.0",
  4    "description": "",
  5    "type": "module",
  6    "main": "src/app.js",
  7    "scripts": {
  8      "dev": "nodemon src/app.js",
  9      "start": "node src/app.js"
 10    },
 11    "dependencies": {
 12      "@prisma/client": "^5.0.0",
 13      "bcrypt": "^5.1.1",
 14      "dotenv": "^16.4.5",
 15      "ejs": "^3.1.10",
 16      "express": "^4.19.2",
 17      "express-rate-limit": "^7.0.0",
 18      "express-session": "^1.17.3",
 19      "helmet": "^7.1.0"
 20    },
 21    "devDependencies": [
 22      "nodemon": "^3.1.4",
 23      "prisma": "^5.18.0"
 24    ]
 25  }
 26 }
```

Ilustración 12. Archivo package.json

Creamos el docker compose dentro de la raíz del proyecto: **docker-compose.yml**

```

version: "3.8"
services:
  postgres:
    image: postgres:15
    container_name: rl_postgres
    restart: unless-stopped
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: admin123
      POSTGRES_DB: ratelimit_db
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data
volumes:
  pgdata:

```

Ilustración 13. Archivo docker-compose.yml

Levantamos el contenedor con:

docker-compose up -d

docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0fe8c5c9cb	postgres:15	"docker-entrypoint.s"	1 second ago	Up Less than a second	0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp	rl_postgres

Ilustración 14. Verificación del contenedor

Prisma: migrate + generate

En terminal (WSL):

```
npx prisma migrate dev --name init
```

```
npx prisma generate
```

```
steven@Stewe:~/projects/rate-limit-project$ npx prisma migrate dev --name init
npx prisma generate
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": PostgreSQL database "ratelimit_db", schema "public" at "localhost:5432"

Applying migration `20251108023015_init`

The following migration(s) have been created and applied from new schema changes:

prisma/migrations/
  └── 20251108023015_init/
    └── migration.sql

Your database is now in sync with your schema.

✓ Generated Prisma Client (v6.19.0) to ./node_modules/@prisma/client in 91ms

Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma

✓ Generated Prisma Client (v6.19.0) to ./node_modules/@prisma/client in 77ms

Start by importing your Prisma Client (See: https://pris.ly/d/importing-client)
Tip: Need your database queries to be 1000x faster? Accelerate offers you that and more: https://pris.ly/tip-2-accelerate

steven@Stewe:~/projects/rate-limit-project$
```

Ilustración 15. Migraciones completas

Rutas

Archivo authRoutes.js

```
src > routes > js authRoutes.js > ...
1 import express from "express";
2 import { showLogin, showRegister, registerUser, loginUser, logoutUser, userPage } from "../controllers/authController.js";
3 import { loginLimiter } from "../middleware/rateLimiter.js";
4
5 const router = express.Router();
6
7 router.get("/login", showLogin);
8 router.post("/login", loginLimiter, loginUser);
9 router.get("/register", showRegister);
10 router.post("/register", registerUser);
11 router.get("/user", userPage);
12 router.get("/logout", logoutUser);
13
14 export default router;
```

Ilustración 16. Archivo authRoutes.js

Archivo adminRoutes.js

```
JS adminRoutes.js X
src > routes > JS adminRoutes.js > ...
1  import express from "express";
2  import { showAdmin, listUsers, changeRole, deleteUser } from "../controllers/adminController.js";
3
4  const router = express.Router();
5
6  // Middleware simple de protección para rutas admin
7  function requireAdmin(req, res, next) {
8    if (!req.session.userId || req.session.role !== "admin") {
9      return res.status(403).send("Acceso restringido");
10    }
11    next();
12  }
13
14  router.get("/", requireAdmin, showAdmin);
15  router.get("/users", requireAdmin, listUsers);
16  router.post("/users/:id/role", requireAdmin, changeRole);
17  router.post("/users/:id/delete", requireAdmin, deleteUser);
18
19  export default router;
20 |
```

Ilustración 17. Archivo adminRoutes.js

Controladores

Archivo authController.js:

```
import bcrypt from "bcrypt";
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();
const SALT_ROUNDS = 10;

export function showLogin(req, res) {
  res.render("login", { error: null, ok: null });
}

export function showRegister(req, res) {
  res.render("register", { error: null });
}

export async function registerUser(req, res) {
  try {
    const { email, password } = req.body;
    if (!email || !password) return res.render("register", { error: "Completa los campos." });
  }
}
```

```
const exists = await prisma.user.findUnique({ where: { email: email.toLowerCase() } });
if (exists) return res.render("register", { error: "Ese correo ya está registrado." });

const hash = await bcrypt.hash(password, SALT_ROUNDS);
await prisma.user.create({ data: { email: email.toLowerCase(), password: hash } });

res.redirect("/login");
} catch (e) {
  console.error(e);
  res.render("register", { error: "Error al registrar." });
}
}

export async function loginUser(req, res) {
  try {
    const { email, password } = req.body;
    const user = await prisma.user.findUnique({ where: { email: (email || "").toLowerCase() } });
    const bad = () => res.status(401).render("login", { error: "Credenciales inválidas", ok: null });

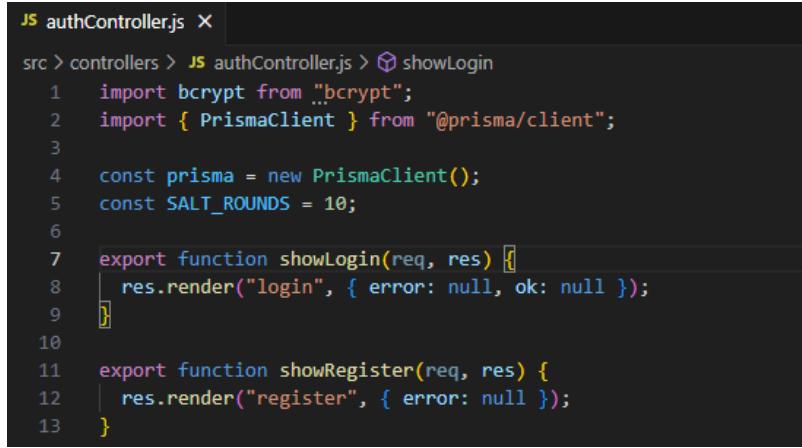
    if (!user) return bad();
    const ok = await bcrypt.compare(password, user.password);
    if (!ok) return bad();

    req.session.userId = user.id;
    req.session.email = user.email;
    req.session.role = user.role;

    res.render("login", { error: null, ok: "Login OK" });
  } catch (e) {
    console.error(e);
    res.status(500).render("login", { error: "Error interno", ok: null });
  }
}

export function logoutUser(req, res) {
  req.session.destroy(() => res.redirect("/login"));
}

export async function userPage(req, res) {
  if (!req.session.userId) return res.redirect("/login");
  res.render("user", { email: req.session.email });
}
```



A screenshot of a code editor showing the `authController.js` file. The code defines two export functions: `showLogin` and `showRegister`. Both functions render templates named "login" and "register" respectively, passing an object with `error: null` and `ok: null`.

```
JS authController.js X
src > controllers > JS authController.js > ⚡ showLogin
1 import bcrypt from "bcrypt";
2 import { PrismaClient } from "@prisma/client";
3
4 const prisma = new PrismaClient();
5 const SALT_ROUNDS = 10;
6
7 export function showLogin(req, res) {
8   res.render("login", { error: null, ok: null });
9 }
10
11 export function showRegister(req, res) {
12   res.render("register", { error: null });
13 }
```

Ilustración 18. Archivo `authController.js` (Ver código completo)

Archivo `adminController.js`:

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

/**
 * showAdmin - renderiza el panel admin (resumen)
 */
export async function showAdmin(req, res) {
  if (!req.session.userId || req.session.role !== "admin") {
    return res.status(403).send("Acceso restringido");
  }

  // resumen simple: cantidad de usuarios
  const usersCount = await prisma.user.count();
  res.render("admin", { email: req.session.email, usersCount });
}

/**
 * listUsers - lista usuarios (sin passwords)
 */
export async function listUsers(req, res) {
  if (!req.session.userId || req.session.role !== "admin") {
    return res.status(403).send("Acceso restringido");
  }

  const users = await prisma.user.findMany({
```

```
    select: {
      id: true,
      email: true,
      role: true,
      createdAt: true
    },
    orderBy: { createdAt: "desc" },
    take: 100
  });

  res.render("adminUsers", { email: req.session.email, users });
}

/** 
 * changeRole - promover/demover usuario (body: role)
 */
export async function changeRole(req, res) {
  if (!req.session.userId || req.session.role !== "admin") {
    return res.status(403).send("Acceso restringido");
  }

  const { id } = req.params;
  const { role } = req.body;
  if (!["admin", "user"].includes(role)) {
    return res.status(400).send("Rol inválido");
  }

  await prisma.user.update({
    where: { id: Number(id) },
    data: { role }
  });

  res.redirect("/admin/users");
}

/** 
 * deleteUser - elimina usuario por id
 */
export async function deleteUser(req, res) {
  if (!req.session.userId || req.session.role !== "admin") {
    return res.status(403).send("Acceso restringido");
  }

  const { id } = req.params;
  await prisma.user.delete({ where: { id: Number(id) } });
}
```

```
    res.redirect("/admin/users");
}

JS adminController.js X

src > controllers > JS adminController.js > ...
1  import { PrismaClient } from "@prisma/client";
2
3  const prisma = new PrismaClient();
4
5  /**
6   * showAdmin - renderiza el panel admin (resumen)
7   */
8  export async function showAdmin(req, res) {
9    if (!req.session.userId || req.session.role !== "admin") {
10      return res.status(403).send("Acceso restringido");
11    }
12
13    // resumen simple: cantidad de usuarios
14    const usersCount = await prisma.user.count();
15    res.render("admin", { email: req.session.email, usersCount });
16 }
```

Ilustración 19. Archivo adminController.js (Ver código completo)

Vistas EJS

Vista login.ejs:

```
login.ejs X
src > views > login.ejs > ...
1  <!doctype html>
2  <html>
3  | <body>
4  | | <h2>Inicio de Sesión</h2>
5  | | <% if (error) { %><p style="color: red"><%= error %></p><% } %>
6  | | <% if (ok) { %><p style="color: green"><%= ok %></p><% } %>
7  | | <form method="POST" action="/login">
8  | | | <input type="email" name="email" placeholder="Correo" required /><br/>
9  | | | <input type="password" name="password" placeholder="Contraseña" required /><br/>
10 | | | <button type="submit">Entrar</button>
11 | | </form>
12 | | <p><a href="/register">Registrarse</a></p>
13 | </body>
14 </html>
```

Ilustración 20. Archivo login.ejs

Vista register.ejs:

```
register.ejs X
src > views > register.ejs > ...
1  <!doctype html>
2  <html>
3  | <body>
4  | | <h2>Registro</h2>
5  | | <% if (error) { %><p style="color: red"><%= error %></p><% } %>
6  | | <form method="POST" action="/register">
7  | | | <input type="email" name="email" required /><br/>
8  | | | <input type="password" name="password" required /><br/>
9  | | | <button type="submit">Registrar</button>
10 | | </form>
11 | | <p><a href="/login">Iniciar sesión</a></p>
12 | </body>
13 </html>
```

Ilustración 21. Archivo register.ejs

Vista user.ejs:

```
user.ejs  X

src > views > user.ejs > ...
1  <!doctype html>
2  <html>
3  |  <body>
4  |    <h2>Panel de Usuario</h2>
5  |    <p>Bienvenido: <%= email %></p>
6  |    <a href="/logout">Cerrar sesión</a>
7  |  </body>
8  </html>
9  |
```

Ilustración 22. Archivo.ejs

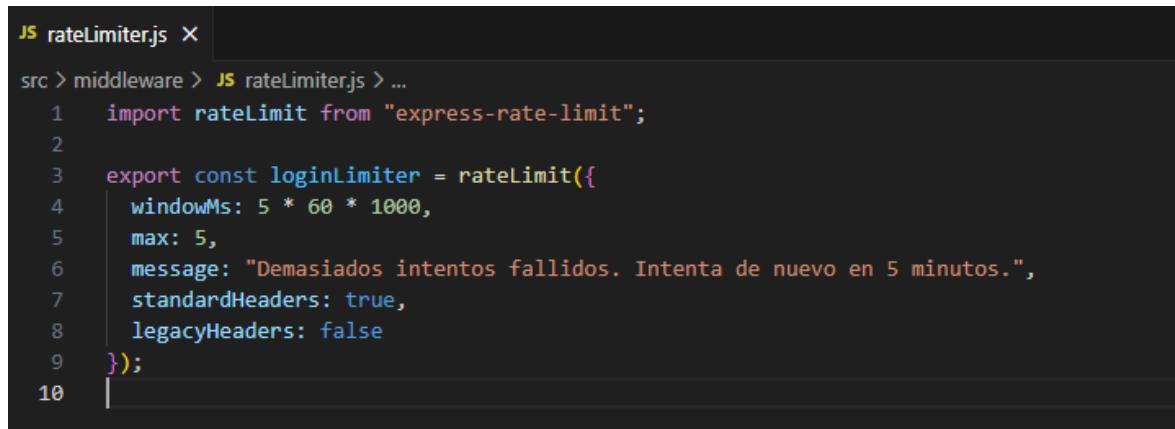
Vista admin.ejs:

```
admin.ejs  X

src > views > admin.ejs > ...
1  <!doctype html>
2  <html>
3  |  <body>
4  |    <h2>Panel Admin</h2>
5  |    <p>Admin: <%= email %></p>
6  |    <a href="/logout">Cerrar sesión</a>
7  |  </body>
8  </html>
9  |
```

Ilustración 23. Archivo admin.ejs

Middleware de Rate Limiting

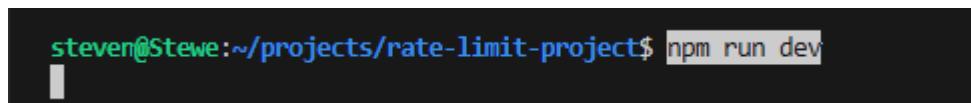


```
JS rateLimiter.js X
src > middleware > JS rateLimiter.js > ...
1 import rateLimit from "express-rate-limit";
2
3 export const loginLimiter = rateLimit({
4   windowMs: 5 * 60 * 1000,
5   max: 5,
6   message: "Demasiados intentos fallidos. Intenta de nuevo en 5 minutos.",
7   standardHeaders: true,
8   legacyHeaders: false
9 });
10 |
```

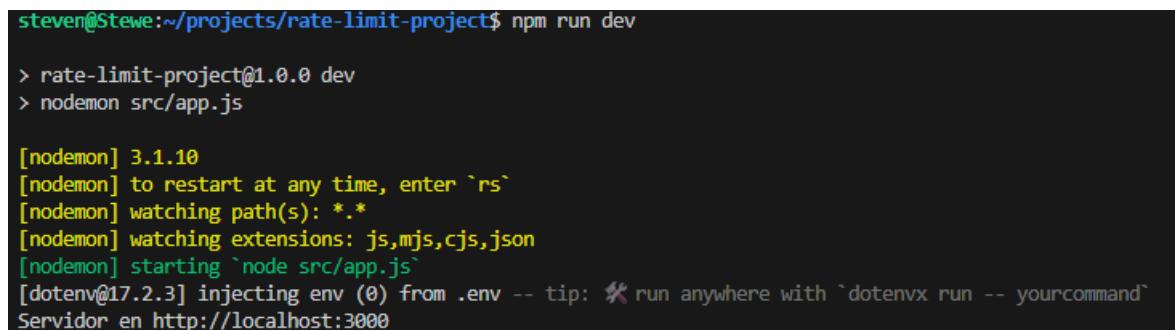
Ilustración 24. Archivo rateLimiter.js

Levantamos la app

Con **npm run dev**



```
steven@Stewe:~/projects/rate-limit-project$ npm run dev
```



```
steven@Stewe:~/projects/rate-limit-project$ npm run dev

> rate-limit-project@1.0.0 dev
> nodemon src/app.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/app.js`
[dotenv@17.2.3] injecting env (0) from .env -- tip: ⚡ run anywhere with `dotenvx run -- yourcommand`
Servidor en http://localhost:3000
```

Ilustración 25. Aplicación escuchando en :3000

Registro

steven@test.com
.....
Registrar

[Iniciar sesión](#)

Ilustración 26. Vista de registro con usuario

Inicio de Sesión

Credenciales inválidas

Correo
Contraseña
Entrar

[Registrarse](#)

Ilustración 27. Inicio de sesión inválido

Demasiados intentos fallidos. Intenta de nuevo en 5 minutos.

Ilustración 28. Protección de rate Limiter

Inicio de Sesión

Login OK

Correo
Contraseña
Entrar

[Registrarse](#)

Ilustración 29. Login correcto

PARA CONVERTIR A UN USUARIO NORMAL EN ADMIN:**npx prisma studio**

```
steven@Stewe:~/projects/rate-limit-project$ npx prisma studio
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Prisma Studio is up on http://localhost:5555
Prisma schema loaded from prisma/schema.prisma
[ ]
```

Ilustración 30. Introducimos en la consola el comando npx prisma studio

	id #	email A	password A	role A	createdAt ☰
	1	steven@test.com	\$2b\$10\$bP/sSHyvVoCyBN.cpwiSwujvaAPr.pqsBvAQwUN10w6RF1K56.V7a	user	2025-11-08T02:49:53,493Z

Ilustración 31. Nos abre la UI de prisma

Y aquí podremos observar todos nuestros usuarios registrados, sus correos, sus contraseñas hasheadas y su rol. Podemos cambiar el rol a admin:

	id #	email A	password A	role A
	1	steven@test.com	\$2b\$10\$bP/sSHyvVoCyBN.cpwiSwujvaAPr.pqsBvAQwUN10w6RF1K56.V7a	admin

Ilustración 32. Confirmamos cambio de rol

Haciendo el proyecto más atractivo y mostramos el rol del usuario una vez iniciada la sesión



Ilustración 33. Nueva vista de login



Ilustración 34. Nueva vista de register

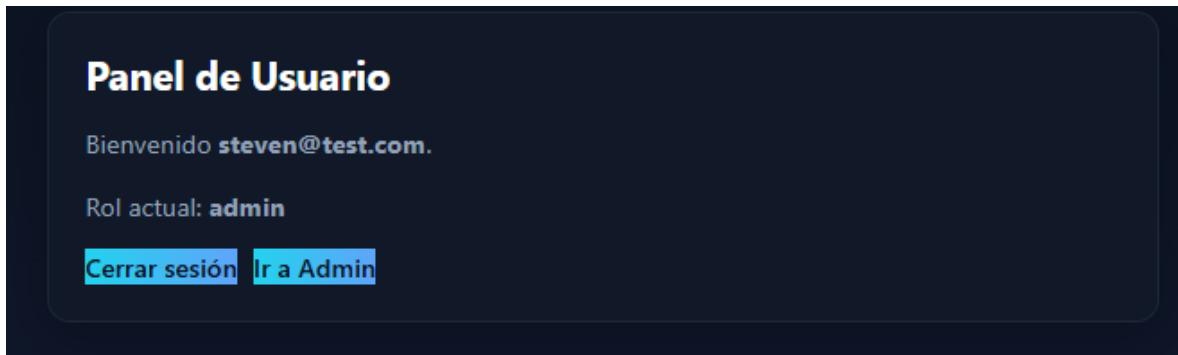


Ilustración 35. Vista de usuario admin

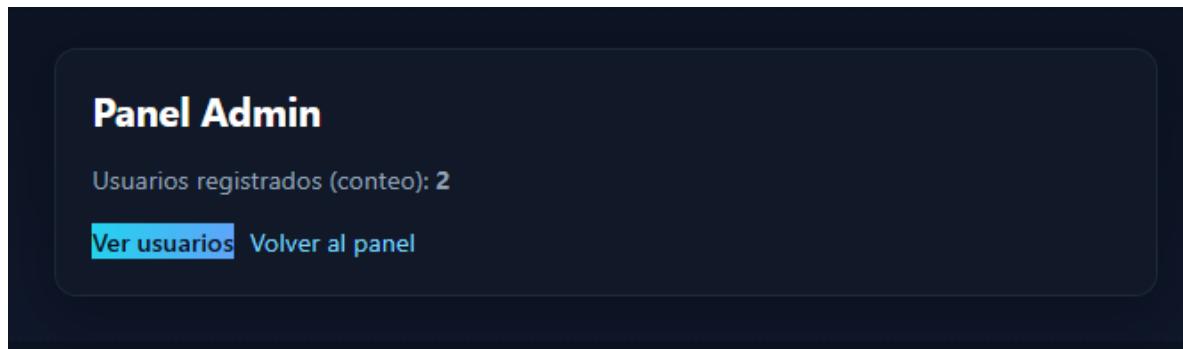


Ilustración 36. Panel de administrador

ID	Email	Rol	Creado	Acciones
2	test1@test.com	user	11/7/2025, 9:05:50 PM	<button>user ▾ Cambiar</button> <button>Eliminar</button>
1	steven@test.com	admin	11/7/2025, 8:49:53 PM	<button>admin ▾ Cambiar</button> <button>Eliminar</button>

Ilustración 37. CRUD de usuarios

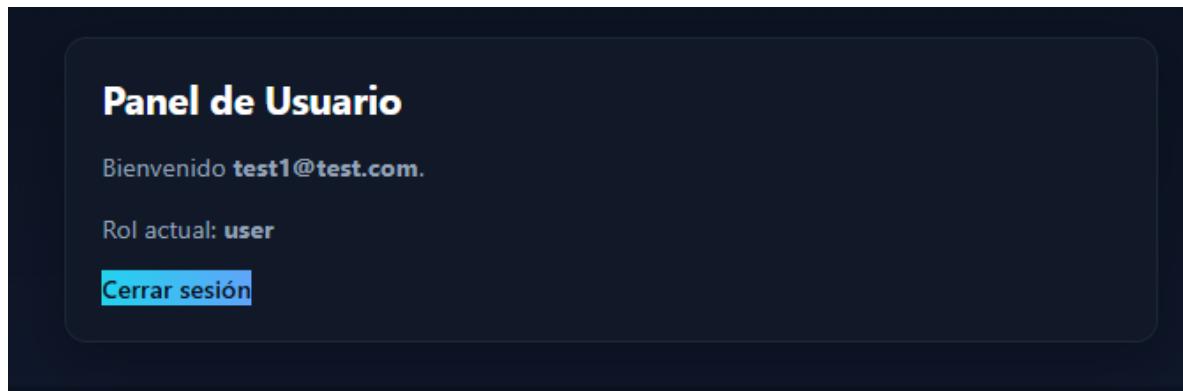


Ilustración 38. Vista de usuario sin permisos

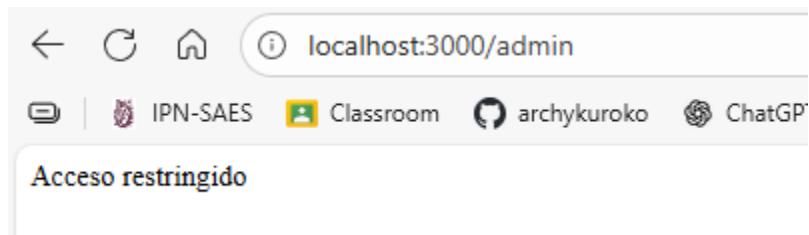


Ilustración 39. Acceso restringido a los menus de administrador

Pruebas de fuerza bruta

Prueba con Script Node (**brute_force.js**)

Objetivo: simular varios intentos de login fallido y observar cuándo aparece **HTTP 429**.

Creamos el archivo con:

```
mkdir -p test  
cat > test/brute_force.js <<'EOF'  
  
const url = "http://localhost:3000/login";  
  
// Credenciales falsas  
  
const body = "email=fakeuser@example.com&password=fakepass";  
  
async function main(){  
  
    for (let i = 1; i <= 10; i++){  
  
        const res = await fetch(url, {  
  
            method: "POST",  
  
            headers: { "Content-Type": "application/x-www-form-urlencoded" },  
  
            body  
        });  
  
        console.log(`Intento ${i}: HTTP ${res.status}`);  
  
        if (res.status === 429) {  
  
            console.log(">> Rate limit activado (Too Many Requests)\n");  
        }  
    }  
}  
  
main().catch(console.error);  
  
EOF
```

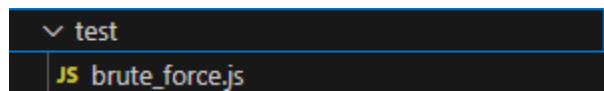


Ilustración 40. Directorio creado

```
JS brute_force.js ×
test > JS brute_forcejs > ...
1  const url = "http://localhost:3000/login";
2
3 // Credenciales falsas
4 const body = "email=fakeuser@example.com&password=fakepass";
5
6 async function main(){
7   for (let i = 1; i <= 10; i++) {
8     const res = await fetch(url, {
9       method: "POST",
10      headers: { "Content-Type": "application/x-www-form-urlencoded" },
11      body
12    });
13
14    console.log(`Intento ${i}: HTTP ${res.status}`);
15
16    if (res.status === 429) {
17      console.log("» Rate limit activado (Too Many Requests)\n");
18    }
19  }
20}
21main().catch(console.error);
22|
```

Ilustración 41. Archivo de test de fuerza bruta

Lo ejecutamos con: **node test/brute_force.js**

Resultado esperado

- Intentos 1–5 → 401 (o similar)
- Intentos $\geq 6 \rightarrow \textbf{429 Too Many Requests}$

```
steven@Stewe:~/projects/rate-limit-project$ node test/brute_force.js
Intento 1: HTTP 401
Intento 2: HTTP 401
Intento 3: HTTP 401
Intento 4: HTTP 401
Intento 5: HTTP 401
Intento 6: HTTP 429
>> Rate limit activado (Too Many Requests)

Intento 7: HTTP 429
>> Rate limit activado (Too Many Requests)

Intento 8: HTTP 429
>> Rate limit activado (Too Many Requests)

Intento 9: HTTP 429
>> Rate limit activado (Too Many Requests)

Intento 10: HTTP 429
>> Rate limit activado (Too Many Requests)

steven@Stewe:~/projects/rate-limit-project$ █
```

Ilustración 42. Pruebas completadas!

Resultados:

Durante las pruebas se ejecutaron 10 intentos de autenticación contra el endpoint `/login` usando credenciales inválidas. Los primeros 5 intentos fueron procesados normalmente (HTTP 401 – **Unauthorized**). A partir del sexto intento el servidor respondió **HTTP 429 – Too Many Requests**, con el mensaje configurado en el limitador. Esto confirma que el mecanismo de **Rate Limiting** se activó correctamente y bloqueó temporalmente nuevas solicitudes desde la misma clave de identificación (IP/cliente).

Resultado observado:

- Intentos 1–5 → HTTP 401
- Intentos ≥ 6 → HTTP 429 (Rate limit activado)

Conclusiones

1.

La implementación del **Rate Limiting** demostró ser una defensa efectiva contra ataques de fuerza bruta dirigidos al sistema de autenticación. Al limitar la cantidad de intentos permitidos por dirección IP durante una ventana de tiempo definida, se logró mitigar el riesgo de automatización maliciosa sin afectar de manera significativa la experiencia de los usuarios legítimos.

2.

El uso conjunto de herramientas como **express-rate-limit**, **bcrypt**, **helmet**, y sesiones seguras permitió construir una arquitectura sólida que cumple con principios básicos de seguridad web: protección de credenciales, reducción de superficie de ataque, y defensa frente a automatizaciones agresivas.

3.

La integración de **Prisma + PostgreSQL** facilitó un manejo ordenado y consistente de la información de usuarios. Su modelo de datos sencillo y migraciones automatizadas permitieron escalar y mantener el sistema sin complicaciones durante el desarrollo.

4.

El sistema diferenció adecuadamente entre roles (**admin / user**), lo que favorece la aplicación del principio de **menor privilegio**, permitiendo limitar accesos según el perfil del usuario. Esto evitó que cualquier persona autenticada accediera a funciones administrativas.

5.

Las pruebas de fuerza bruta simuladas mostraron que, tras superar el umbral de intentos permitido, el servidor respondía con código **HTTP 429 ("Too Many Requests")**, confirmando que el mecanismo de rate limiting se ejecutó correctamente y bloqueó temporalmente nuevas solicitudes maliciosas.

6.

Desde la perspectiva operativa, se comprobó que los mecanismos aplicados pueden integrarse fácilmente en sistemas web reales por su bajo costo computacional, facilidad de configuración y mantenimiento. Además, su estructura modular permite incorporar reglas adicionales o expandir el sistema sin afectar el funcionamiento actual.

7.

Finalmente, este miniproyecto evidencia que la seguridad en aplicaciones web no depende de un solo control, sino de la combinación de varias prácticas: almacenamiento seguro de credenciales, configuración de cabeceras, validación de entradas, sesiones aisladas y limitación de frecuencia. Adoptarlas de manera conjunta fortalece la postura de seguridad general del sistema y reduce significativamente la probabilidad de compromiso.