ACTIVIDAD 5 “Reservas de Pistas de Pádel” (MySQL + API)

0) Requisitos previos

- Node.js 18+ (recomendado)

- MySQL instalado (o XAMPP / Docker)

- VS Code

- Postman

- Un usuario MySQL con permisos

PARTE A — Base de datos con el SQL del enunciado

1) Crear la BBDD y tablas (OBLIGATORIO con el SQL dado)

Ejecuta exactamente el SQL del enunciado (incluye DROP DATABASE, creación de tablas e inserts).

Actividad_Reservas_Pistas_Padel…

PARTE B — Proyecto Node + TypeScript + Express (estructura limpia)

2) Crear el proyecto

En una carpeta vacía, abre terminal y ejecuta:

mkdir api-padel

cd api-padel

npm init -y

3) Instalar dependencias

Dependencias de ejecución

npm i express cors dotenv sequelize mysql2 express-validator

Dependencias de desarrollo (TypeScript)

npm i -D typescript ts-node-dev @types/express @types/cors @types/node

4) Crear tsconfig.json

npx tsc --init

Abre tsconfig.json y deja algo así:

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "CommonJS",
    "rootDir": "./src",
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true
  }
}
```

5) Scripts en package.json

Edita package.json:

```
"scripts": {
  "dev": "ts-node-dev --respawn --transpile-only src/server.ts",
  "build": "tsc",
  "start": "node dist/server.js"
}
```

PARTE C — Configuración: .env, servidor y middlewares

6) Crear archivo .env

Crea .env en la raíz:

PORT=3000

DB_HOST=localhost

DB_USER=root

DB_PASSWORD=

DB_NAME=club_padel_db

DB_DIALECT=mysql

FRONT_ORIGIN=http://localhost:5173

FRONT_ORIGIN lo usaremos en cors pensando en React/Vite (5173) o cámbialo a tu front cuando toque.

7) Estructura de carpetas recomendada

Crea esto:

src/

config/

db.ts

models/

Pista.ts

Reserva.ts

validators/

pistaValidators.ts

reservaValidators.ts

routes/

pistas.routes.ts

reservas.routes.ts

controllers/

pistas.controller.ts

reservas.controller.ts

server.ts

8) Conexión a MySQL con Sequelize

src/config/db.ts

```
import { Sequelize } from "sequelize";
import dotenv from "dotenv";

dotenv.config();

export const sequelize = new Sequelize(
  process.env.DB_NAME as string,
  process.env.DB_USER as string,
  process.env.DB_PASSWORD as string,
  {
    host: process.env.DB_HOST,
    dialect: "mysql",
    logging: false,
  }
);

export async function testDbConnection() {
  try {
    await sequelize.authenticate();
    console.log("Conexión a MySQL OK");
```

```
  } catch (error) {
    console.error("Error conectando a MySQL:", error);
    process.exit(1);
  }
}
```

9) Servidor Express + los 3 middlewares

src/server.ts

```
import express from "express";
import cors from "cors";
import dotenv from "dotenv";
import { testDbConnection } from "./config/db";

import pistasRoutes from "./routes/pistas.routes";
import reservasRoutes from "./routes/reservas.routes";

dotenv.config();

const app = express();
const port = process.env.PORT || 3000;

/**
 * MIDDLEWARE 1: express.json()
 * - Permite leer JSON en req.body
 * - Imprescindible para POST/PUT con body
 */
```

```
app.use(express.json());

/**
 * MIDDLEWARE 2: cors()
 * - Cuando conectemos un front (React/Vue), el navegador bloquea si el origen es distinto.
 * - Aquí permitimos SOLO el origen definido en FRONT_ORIGIN.
 */
app.use(
  cors({
    origin: process.env.FRONT_ORIGIN, // ej: http://localhost:5173
    methods: ["GET", "POST", "PUT", "DELETE"],
    allowedHeaders: ["Content-Type", "Authorization"],
  })
);

// Rutas
app.use("/pistas", pistasRoutes);
app.use("/reservas", reservasRoutes);

// 404 básico
app.use((_req, res) => {
  res.status(404).json({ message: "Ruta no encontrada" });
});

async function start() {
  await testDbConnection();
```

```
  app.listen(port, () => console.log(` API escuchando en puerto ${port}`));
}

start();
```

PARTE D — Modelos Sequelize (adaptados al SQL del enunciado)

Ojo: el SQL usa created_at y updated_at. En Sequelize lo mapeamos para que cuadre 100% con las tablas del enunciado.

10) Modelo Pista

src/models/Pista.ts

```
import { DataTypes } from "sequelize";
import { sequelize } from "../config/db";

export const Pista = sequelize.define(
  "Pista",
  {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    nombre: { type: DataTypes.STRING(60), allowNull: false, unique: true },
    tipo: { type: DataTypes.ENUM("INDOOR", "OUTDOOR"), allowNull: false },
    precio_hora: { type: DataTypes.DECIMAL(7, 2), allowNull: false },
  },
  {
    tableName: "pistas",
    timestamps: true,
    createdAt: "created_at",
```

```
    updatedAt: "updated_at",
  }
);
```

11) Modelo Reserva

src/models/Reserva.ts

```
import { DataTypes } from "sequelize";
import { sequelize } from "../config/db";
import { Pista } from "./Pista";

export const Reserva = sequelize.define(
  "Reserva",
  {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    pista_id: { type: DataTypes.INTEGER, allowNull: false },
    fecha: { type: DataTypes.DATEONLY, allowNull: false },
    hora_inicio: { type: DataTypes.TIME, allowNull: false },
    hora_fin: { type: DataTypes.TIME, allowNull: false },
  },
  {
    tableName: "reservas",
    timestamps: true,
    createdAt: "created_at",
    updatedAt: "updated_at",
  }
);
```

```
// Relaciones (FK)
Pista.hasMany(Reserva, { foreignKey: "pista_id" });
Reserva.belongsTo(Pista, { foreignKey: "pista_id" });
```

PARTE E — Middleware 3: express-validator (validación body/params)

12) Validadores para Pistas

src/validators/pistaValidators.ts

```
import { body, param } from "express-validator";

export const validarIdParam = [
  param("id").isInt({ min: 1 }).withMessage("El id debe ser un entero >= 1"),
];

export const validarCrearPista = [
  body("nombre")
    .isString().withMessage("nombre debe ser texto")
    .notEmpty().withMessage("nombre es obligatorio")
    .isLength({ max: 60 }).withMessage("nombre máximo 60 caracteres"),

  body("tipo")
    .isIn(["INDOOR", "OUTDOOR"])
    .withMessage("tipo debe ser INDOOR u OUTDOOR"),

  body("precio_hora")
    .isFloat({ min: 0 })
```

```
  .withMessage("precio_hora debe ser un número >= 0"),
];

export const validarActualizarPista = [
  ...validarIdParam,
  body("nombre").optional().isString().notEmpty().isLength({ max: 60 }),
  body("tipo").optional().isIn(["INDOOR", "OUTDOOR"]),
  body("precio_hora").optional().isFloat({ min: 0 }),
];
```

13) Validadores para Reservas

src/validators/reservaValidators.ts

```
import { body, param } from "express-validator";

export const validarIdReservaParam = [
  param("id").isInt({ min: 1 }).withMessage("El id debe ser un entero >= 1"),
];

export const validarCrearReserva = [
  body("pista_id")
    .isInt({ min: 1 })
    .withMessage("pista_id debe ser entero >= 1"),

  body("fecha")
    .isISO8601()
    .withMessage("fecha debe tener formato YYYY-MM-DD"),
```

```
  body("hora_inicio")
    .matches(/^\d{2}:\d{2}:\d{2}$/)
    .withMessage("hora_inicio debe ser HH:MM:SS"),

  body("hora_fin")
    .matches(/^\d{2}:\d{2}:\d{2}$/)
    .withMessage("hora_fin debe ser HH:MM:SS"),

  // Regla lógica: hora_fin > hora_inicio (además del CHECK en SQL)
  body("hora_fin").custom((horaFin, { req }) => {
    const inicio = req.body.hora_inicio;
    if (inicio && horaFin <= inicio) {
      throw new Error("hora_fin debe ser mayor que hora_inicio");
    }
    return true;
  }),
];
```

14) Patrón común: función para devolver errores de validación

La usaremos en controladores para cortar si hay errores.

PARTE F — Controladores

15) Controlador de Pistas

src/controllers/pistas.controller.ts

```
import { Request, Response } from "express";
```

```
import { validationResult } from "express-validator";
import { Pista } from "../models/Pista";

function devolverErroresValidacion(req: Request, res: Response) {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    res.status(400).json({ errors: errors.array() });
    return true;
  }
  return false;
}

export async function crearPista(req: Request, res: Response) {
  if (devolverErroresValidacion(req, res)) return;

  try {
    const pista = await Pista.create(req.body);
    res.status(201).json(pista);
  } catch (e: any) {
    // nombre único -> error típico
    res.status(400).json({ message: "Error creando pista", detail: e.message });
  }
}

export async function listarPistas(_req: Request, res: Response) {
  const pistas = await Pista.findAll();
```

```
  res.json(pistas);
}

export async function obtenerPista(req: Request, res: Response) {
  if (devolverErroresValidacion(req, res)) return;

  const pista = await Pista.findByPk(req.params.id);
  if (!pista) return res.status(404).json({ message: "Pista no encontrada" });
  res.json(pista);
}

export async function actualizarPista(req: Request, res: Response) {
  if (devolverErroresValidacion(req, res)) return;

  const pista = await Pista.findByPk(req.params.id);
  if (!pista) return res.status(404).json({ message: "Pista no encontrada" });

  await pista.update(req.body);
  res.json(pista);
}

export async function eliminarPista(req: Request, res: Response) {
  if (devolverErroresValidacion(req, res)) return;

  const pista = await Pista.findByPk(req.params.id);
  if (!pista) return res.status(404).json({ message: "Pista no encontrada" });
```

```
  try {
    await pista.destroy();
    res.json({ message: "Pista eliminada" });
  } catch (e: any) {
    // si tiene reservas, la FK del SQL restringe el borrado
    res.status(409).json({
      message: "No se puede borrar la pista: tiene reservas asociadas",
      detail: e.message,
    });
  }
}
```

16) Controlador de Reservas (incluye solape obligatorio)

src/controllers/reservas.controller.ts

```
import { Request, Response } from "express";
import { validationResult } from "express-validator";
import { Op } from "sequelize";
import { Reserva } from "../models/Reserva";
import { Pista } from "../models/Pista";

function devolverErroresValidacion(req: Request, res: Response) {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    res.status(400).json({ errors: errors.array() });
    return true;
```

```
  }
  return false;
}

export async function crearReserva(req: Request, res: Response) {
  if (devolverErroresValidacion(req, res)) return;

  const { pista_id, fecha, hora_inicio, hora_fin } = req.body;

  // 1) Comprobar que la pista existe
  const pista = await Pista.findByPk(pista_id);
  if (!pista) return res.status(404).json({ message: "La pista no existe" });

  // 2) Comprobar SOLAPE (regla de negocio del enunciado)
  // Solape si: inicioNueva < finExistente AND finNueva > inicioExistente
  const solape = await Reserva.findOne({
    where: {
      pista_id,
      fecha,
      hora_inicio: { [Op.lt]: hora_fin },
      hora_fin: { [Op.gt]: hora_inicio },
    },
  });

  if (solape) {
    return res.status(409).json({
```

```
      message: "Reserva inválida: hay solape con otra reserva",
      conflicto: solape,
    });
  }

  // 3) Crear reserva
  try {
    const reserva = await Reserva.create({ pista_id, fecha, hora_inicio, hora_fin });
    res.status(201).json(reserva);
  } catch (e: any) {
    res.status(400).json({ message: "Error creando reserva", detail: e.message });
  }
}

export async function listarReservas(req: Request, res: Response) {
  // filtros opcionales (enunciado lo permite)
  const { fecha, pista_id } = req.query;

  const where: any = {};
  if (fecha) where.fecha = fecha;
  if (pista_id) where.pista_id = pista_id;

  const reservas = await Reserva.findAll({ where });
  res.json(reservas);
}
```

```
export async function eliminarReserva(req: Request, res: Response) {
  if (devolverErroresValidacion(req, res)) return;

  const reserva = await Reserva.findByPk(req.params.id);
  if (!reserva) return res.status(404).json({ message: "Reserva no encontrada" });

  await reserva.destroy();
  res.json({ message: "Reserva eliminada" });
}
```

PARTE G — Rutas (dónde se “cuelgan” los validadores)

17) Rutas de Pistas (con express-validator)

src/routes/pistas.routes.ts

```
import { Router } from "express";
import {
  crearPista,
  listarPistas,
  obtenerPista,
  actualizarPista,
  eliminarPista,
} from "../controllers/pistas.controller";

import {
  validarCrearPista,
  validarActualizarPista,
  validarIdParam,
```

```
} from "../validators/pistaValidators";

const router = Router();

// Endpoints del enunciado :contentReference[oaicite:7]{index=7}
router.post("/", validarCrearPista, crearPista);
router.get("/", listarPistas);
router.get("/:id", validarIdParam, obtenerPista);
router.put("/:id", validarActualizarPista, actualizarPista);
router.delete("/:id", validarIdParam, eliminarPista);

export default router;
```

18) Rutas de Reservas

src/routes/reservas.routes.ts

```
import { Router } from "express";
import {
  crearReserva,
  listarReservas,
  eliminarReserva,
} from "../controllers/reservas.controller";

import {
  validarCrearReserva,
  validarIdReservaParam,
} from "../validators/reservaValidators";
```

const router = Router();

// Endpoints del enunciado :contentReference[oaicite:8]{index=8}

router.post("/", validarCrearReserva, crearReserva);

router.get("/", listarReservas);

router.delete("/:id", validarIdReservaParam, eliminarReserva);

export default router;

PARTE H — Arranque y pruebas con Postman

19) Arrancar la API

npm run dev

Debes ver:

Conexión a MySQL OK

API escuchando en puerto 3000

Evidencia para PDF: captura consola.

20) Pruebas Postman

Base URL: http://localhost:3000

A) Crear pista (POST /pistas)

POST http://localhost:3000/pistas

{

"nombre": "Pista 3 - Indoor",

"tipo": "INDOOR",

"precio_hora": 18.0

}

Captura respuesta 201.

Prueba “mala” para ver express-validator

Envía:

{

"nombre": "",

"tipo": "CUBIERTA",

"precio_hora": -5

}

Debe devolver 400 con lista de errores.

B) Listar pistas (GET /pistas)

GET http://localhost:3000/pistas

Captura response.

C) Crear reserva válida (POST /reservas)

El SQL ya inserta una reserva:

pista 1, fecha 2026-01-12, 18:00–19:00

Crea una no solapada a continuación:

POST http://localhost:3000/reservas

{

"pista_id": 1,

"fecha": "2026-01-12",

"hora_inicio": "19:00:00",

"hora_fin": "20:00:00"

}

Debe devolver 201.

D) Intento de reserva inválida por solape (obligatorio)

Ahora fuerza solape con 18:00–19:00:

POST http://localhost:3000/reservas

{

"pista_id": 1,

"fecha": "2026-01-12",

"hora_inicio": "18:30:00",

"hora_fin": "19:30:00"

}

Debe devolver 409 Conflict con mensaje:

“Reserva inválida: hay solape…”

Captura del error 409 .

E) Listar reservas (GET /reservas)

GET http://localhost:3000/reservas

Opcional:

GET /reservas?fecha=2026-01-12

GET /reservas?pista_id=1

Captura response.

F) Cancelar/eliminar reserva (DELETE /reservas/:id)

Copia un id real de la lista.

Haz:

DELETE http://localhost:3000/reservas/2

Debe devolver { "message": "Reserva eliminada" }