**🧭 SPRINT 1 — Módulo de Matrícula**

**🎯 Objetivo general**

Implementar un sistema web que permita **automatizar el proceso de matrícula de estudiantes** en una academia preuniversitaria, reemplazando el procedimiento manual por un registro digital, seguro y validado por un administrador.

**⚙️ Arquitectura general del sistema**

| **Capa** | **Tecnología** | **Descripción** |
| --- | --- | --- |
| **Frontend** | React + Vite | Interfaz web con formularios, pasos guiados y panel administrativo. |
| **Backend** | Node.js + Express | API REST que gestiona autenticación, matrículas y validaciones. |
| **ORM / BD** | Prisma + MySQL (HeidiSQL) | Base de datos relacional administrada mediante Prisma ORM. |
| **Autenticación** | JWT + Cookies HTTPOnly | Control de sesión seguro y gestión de roles (ADMIN, ESTUDIANTE, etc.). |
| **Subida de archivos** | Multer | Manejo de comprobantes de pago en formato imagen/PDF. |

**🧩 Tablas utilizadas en el Sprint 1**

A continuación se describen las **tablas del modelo Prisma** usadas en este primer sprint y su rol dentro del flujo de matrícula.

**🧱 1️⃣ Usuario**

model Usuario {

id Int @id @default(autoincrement())

nombre String

apellidoP String

apellidoM String

correo String @unique

password String

rol Rol @default(USUARIO)

activo Boolean @default(true)

creadoEn DateTime @default(now())

estudiante Estudiante?

}

* Contiene la información básica de todos los usuarios del sistema (estudiantes, docentes, admin).
* Cada usuario se autentica mediante correo y contraseña encriptada (bcryptjs).
* Al aprobar una matrícula, el estudiante se vincula con su registro de usuario (usuarioId).

**🧱 2️⃣ Estudiante**

model Estudiante {

id Int @id @default(autoincrement())

usuarioId Int @unique

usuario Usuario @relation(fields: [usuarioId], references: [id])

nombre String

apellidoPaterno String

apellidoMaterno String

dni String @unique

fechaNacimiento DateTime

telefono String

nombreApoderado String?

telefonoApoderado String?

matriculas Matricula[]

createdAt DateTime @default(now())

}

* Se crea automáticamente **cuando el administrador aprueba una matrícula**.
* Almacena los datos personales definitivos del estudiante.
* Está relacionada 1:N con Matricula (un estudiante puede tener varias matrículas a lo largo del tiempo).

**🧱 3️⃣ Matricula**

model Matricula {

id Int @id @default(autoincrement())

estudianteId Int?

estudiante Estudiante? @relation(fields: [estudianteId], references: [id])

grupo Grupo @relation(fields: [grupoId], references: [id])

grupoId Int

modalidad Modalidad @relation(fields: [modalidadId], references: [id])

modalidadId Int

carreraPrincipal Carrera @relation("Principal", fields: [carreraPrincipalId], references: [id])

carreraPrincipalId Int

carreraSecundaria Carrera? @relation("Secundaria", fields: [carreraSecundariaId], references: [id])

carreraSecundariaId Int?

tipoPago String

comprobanteUrl String?

estado String @default("PENDIENTE")

createdAt DateTime @default(now())

// Datos temporales

nombre String?

apellidoPaterno String?

apellidoMaterno String?

dni String?

telefono String?

nombreApoderado String?

telefonoApoderado String?

}

* Es la **tabla principal del proceso de matrícula**.
* Al enviarse el formulario, se guarda con estado = "PENDIENTE".
* Al aprobarse, se genera un registro en Estudiante y se limpian los campos personales temporales.

**🧱 4️⃣ Tablas de referencia**

| **Tabla** | **Rol** |
| --- | --- |
| Modalidad | Define los tipos de ingreso: Ordinario, CEPRU, Primera Oportunidad, etc. |
| Grupo | Clasifica estudiantes por secciones (A, B, C, D). |
| Carrera | Asocia las carreras a cada grupo. |
| Asignatura | Define los cursos o materias dentro de un grupo. |

**🔁 Flujo funcional del módulo de matrícula**

**1️⃣ Registro e inicio de sesión**

* El usuario se registra o inicia sesión mediante /api/auth/register o /api/auth/login.
* El sistema genera un **token JWT** almacenado en una **cookie HTTPOnly**.
* El rol determina la vista a la que accede (usuario → matrícula, admin → dashboard).

**2️⃣ Llenado del formulario de matrícula**

En el frontend (Matricula.jsx) el estudiante completa un flujo de **4 pasos**:

1. **Datos personales**
2. **Datos académicos**
3. **Pago y comprobante**
4. **Confirmación final**

→ Todos los datos se agrupan y envían en un FormData (con el archivo de comprobante) al backend:

POST /api/matriculas

**3️⃣ Registro temporal en la base de datos**

En el backend (matriculas.controller.js):

* Se guarda la información con estado "PENDIENTE".
* El archivo se guarda con multer en /uploads/comprobantes/.
* Los datos personales quedan guardados **temporalmente** en la tabla Matricula.

**4️⃣ Revisión por el administrador**

El administrador entra a su panel (DashboardAdmin.jsx) y accede a:

/admin/validarMatricula

Donde ve la lista de matrículas **pendientes**.

**5️⃣ Aprobación o rechazo**

En la vista ValidarMatricula.jsx:

* Si aprueba → PUT /api/matriculas/:id/aprobar
  + Se crea un nuevo registro en Estudiante.
  + Se actualiza Matricula → estado = "APROBADA".
  + Se limpian los campos temporales (nombre, dni, etc.).
* Si rechaza → PUT /api/matriculas/:id/rechazar
  + Se marca la matrícula como RECHAZADA.

**6️⃣ Resultado final**

* El estudiante puede consultar su matrícula en su perfil (estado: “Pendiente”, “Aprobada”, “Rechazada”).
* El admin tiene control completo sobre el flujo, sin necesidad de papeleo o colas.

**🧠 Conclusiones del Sprint 1**

* Se implementó el **MVP (Producto Mínimo Viable)** del sistema académico.
* El flujo completo de matrícula está 100 % funcional y validado.
* Las relaciones entre tablas quedaron correctamente estructuradas.
* La arquitectura es modular, escalable y segura.