Entrega 1 — VisiControl (Gestión de visitas penitenciarias)

Integrantes:

- Kevin Adrián Gil Soto (@kevingS0712)
- Boris Guiillermo Briones Dupla
- Fernando Moreno Mora
- Rafael Alejandro Ajila Gallegos

Carrera: Ingeniería de Software
Fecha: [13/10/2025]
Repositorio Git: https://github.com/kevings0712/VisiControl

---

1. Introducción

VisiControl es un sistema para **solicitar, aprobar/programar y controlar** visitas a personas privadas de libertad (PPL) en centros penitenciarios.
Esta entrega documenta los cimientos del proyecto: actores, funcionalidades, **diagramas UML** (casos de uso, clases, componentes y secuencia), **arquitectura en capas**, aplicación de **principios SOLID**, **patrones de diseño** y consideraciones de **seguridad y escalabilidad**.

---

2. Justificación del proyecto

- El **visitante** necesita un proceso claro para **solicitar una visita** y conocer su **estado** sin trámites presenciales.
- El **funcionario/organizador** requiere **revisar**, **validar** y **programar** visitas respetando **cupos y horarios**.
- El **guardia** debe **verificar el pase/QR** y **registrar asistencia** en el punto de acceso.

Objetivo: digitalizar y trazar el flujo de visitas, reducir tiempos y mejorar la experiencia de todos los actores.

---

3. Alcance del sistema

El sistema permitirá:

- Registro/inicio de sesión para **staff** (autenticación).
- **Solicitud de visita** por visitante y **carga de documentos** (DNI/antecedentes).
- **Revisión/validación** por parte del staff y **aprobación/rechazo**.

- **Agendamiento** con reglas de **cupos** y **horarios**.
- Emisión de **pase/QR** y **consulta de estado** de la solicitud.
- **Registro de asistencia** en puerta.

No incluye todavía:

- Integraciones con RENIEC/biometría.
- Notificaciones push en tiempo real (se prevén **colas + correo**).
- Reportería avanzada.

---

4. Historias de Usuario

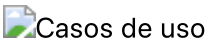
1. Como **visitante**, quiero **solicitar una visita** para poder ver a un interno.
2. Como **visitante**, quiero **subir documentos** para que validen mi identidad.
3. Como **visitante**, quiero **consultar el estado** (pendiente, aprobada, rechazada) para saber qué esperar.
4. Como **staff**, quiero **revisar/validar** solicitudes para **aprobar/rechazar**.
5. Como **staff**, quiero **configurar horarios/cupos** por centro para controlar aforos.
6. Como **staff**, quiero **programar fecha/hora** y **emitir un pase/QR**.
7. Como **guardia**, quiero **escanear el pase/QR** y **registrar asistencia**.
8. Como **sistema**, quiero **notificar por correo** cambios de estado de la visita.
9. Como **staff**, quiero **reprogramar** visitas cuando sea necesario.
10. Como **staff**, quiero **buscar/filtrar** solicitudes por fecha/estado/centro.

---

5. Diagramas UML y explicaciones

5.1. Caso de Uso

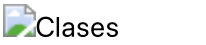
Representa las principales funcionalidades y la interacción de **Visitante**, **Staff** y **Guardia**.


Casos de uso

Explicación:
El visitante solicita y consulta; el staff valida, decide y programa; el guardia controla acceso y asistencia. Se incluyen notificaciones por correo y generación de pase/QR.

5.2. Clases de Dominio

Modelo conceptual del dominio penitenciario.


Clases

Explicación (entidades clave):

- **Visitante / Funcionario / Interno**: actores del proceso.
- **Visita**: solicitud programable con estados (PENDIENTE, APROBADA, …).
- **Documento**: adjuntos del visitante.
- **Centro, Horario, Cupo**: reglas operativas para agenda.

5.3. Componentes

Organización del sistema por módulos.

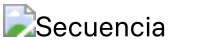
 Componentes

Explicación:

- **Controladores** (HTTP) → **Servicios** (reglas) → **Repositorios** (DB).
- **Auth/JWT**, **Eventos (Observer)** para notificaciones, **DB Pool (Singleton)** en PostgreSQL.
- **Frontend** (React) consume la API vía cliente HTTP.

5.4. Secuencia

Flujo de **solicitar** y **aprobar** una visita.

 Secuencia

Explicación (resumen):
El visitante envía la solicitud → se valida y persiste → se emite evento de confirmación → el staff aprueba y genera pase/QR → el sistema notifica al visitante.

---

6. Arquitectura Propuesta

Arquitectura en capas por separación de responsabilidades y mantenibilidad:

- **Presentación (React/Vite):** captura de datos y visualización del estado.
- **Aplicación (Node/Express):** reglas (cupos/horarios, estados), autenticación y orquestación.
- **Datos (PostgreSQL):** persistencia relacional, índices en campos de alta consulta.

---

7. Principios SOLID aplicados

1. **SRP (Responsabilidad Única):**
    - Controladores coordinan I/O HTTP.
    - Servicios encapsulan reglas y casos de uso.
    - Repositorios aíslan el acceso a datos.
2. **DIP (Inversión de Dependencias):**
    - Servicios dependen de **repositorios** (interfaces), no del driver `pg`.
    - Facilita pruebas y posibles cambios de motor de DB.
3. **OCP (Abierto/Cerrado):**
    - Nuevas reglas/casos (p. ej., reprogramación con validaciones extra) se añaden sin modificar lo existente, extendiendo rutas/controladores/servicios.

---

8. Patrones de diseño utilizados

- **Singleton:** conexión a la base de datos (Pool) para una sola instancia compartida.
- **Factory:** creación de DTO/entidades a partir de payloads (normalización).
- **Observer:** eventos (“VisitaCreada”, “VisitaAprobada”) para correos/colas sin acoplar el núcleo.

---

9. Seguridad y Escalabilidad

Seguridad

- **JWT** para staff; **Helmet + CORS**; variables en **.env**.
- **Hash** de contraseñas (bcrypt) y **validación** de entradas (zod/express-validator).
- Reglas de **transiciones de estado** bien definidas (pendiente → aprobada → completada).

Escalabilidad

- API **stateless** (lista para escalar horizontalmente).
- **Índices**: `visits(date)`, `visits(inmate_id)`, `documents(visitor_id)`.
- **Colas** (BullMQ/Redis) para notificaciones masivas y trabajos diferidos.

---

10. Evidencia del código

Repositorio con **monorepo** y código mínimo funcional (healthcheck y conexión a DB): visicontrol/ ├─ docs/ | └─ uml/ | ├─ visicontrol_caso_uso.png | ├─ visicontrol_clases_dominio.png | ├─ visicontrol_componentes.png | └─ visicontrol_secuencia.png └─ src/ ├─ backend/ (Node/Express/TS) | ├─ src/{config,routes,controllers,services,repositories,...} | └─ sql/init_db.sql └─ frontend/ (React/Vite/TS) └─ src/{App.tsx, api/client.ts, main.tsx}

- **API viva (local):** `GET http://localhost:4000/api/health`
- **(Opcional) Evidencias gráficas:**
   API viva
   Salud BD

---

11. Conclusiones

Se definieron actores y funcionalidades, se documentó con **UML**, se propuso una **arquitectura en capas** alineada a **SOLID** y se eligieron **patrones** adecuados. Existe un **repo funcional** (frontend + backend + DB) listo para la siguiente fase: autenticación real, CRUD completo y eventos/colas.

---

12. Bibliografía

- Apuntes de clase (Unidades 1.2, 1.3, 1.4).
- Sommerville, I. *Ingeniería de Software*. Pearson.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. *Design Patterns*. Addison-Wesley.
- Documentación oficial de **Express**: https://expressjs.com/
- Documentación oficial de **PostgreSQL**: https://www.postgresql.org/