Informe Técnico Integral: Arquitectura, Diseño e Implementación del Sistema de Matriculación Vehicular

Institución: Escuela Politécnica Nacional (EPN)

Carrera: Tecnología en Desarrollo de Software

Autores:

- Michael Alexander Carrillo Mendez
- Kevin Alexander Amagua Valenzuela

---

1. Introducción y Contextualización del Proyecto

1.1 El Imperativo de la Transformación Digital

En el contexto académico de la **Escuela Politécnica Nacional**, este proyecto, alojado en el repositorio sistema-matriculacion-mvc, representa la culminación de competencias Programación Orientada a Objetos (). El sistema responde a la necesidad de modernizar la gestión de licencias de conducir, migrando de procesos manuales a una solución digital robusta.

La solución se ha diseñado no solo para cumplir una función administrativa, sino como un artefacto de ingeniería que demuestra el dominio de la Programación Orientada a Objetos (POO), la arquitectura en capas y la integración con servicios en la nube.

1.2 Objetivos Estratégicos

1. **Centralización del Dato:** Uso de una base de datos en la nube (Supabase/PostgreSQL) para garantizar que la información sea accesible en tiempo real, superando las limitaciones de los sistemas locales.
2. **Integridad y Seguridad:** Implementación de roles diferenciados (Admin/Analista) y hashing de contraseñas para proteger la información sensible del ciudadano.
3. **Arquitectura Profesional:** Estructuración del código bajo el patrón MVC (Modelo-Vista-Controlador) adaptado a aplicaciones de escritorio, facilitando el mantenimiento y la escalabilidad.

---

2. Estructura del Proyecto y Arquitectura de Software

La organización del código fuente es fundamental para la mantenibilidad. A continuación, se detalla la estructura real del proyecto tal como ha sido implementada por los autores.

2.1 Árbol de Directorios del Proyecto

El sistema se organiza bajo el paquete raíz com.licencias.sistemalicenciasfx, distribuyendo las responsabilidades de manera modular:

```
sistema-matriculacion-mvc/
├─────.idea/ # Configuraciones del IDE (IntelliJ)
├─────.mvn/ # Wrapper de Maven
├──── src/
│  └──── main/
│  ├──── java/
│  │  └──── com/
│  │  └──── licencias/
│  │  └──── sistemalicenciasfx/
│  │  ├──── config/
│  │  │  └──── DatabaseConfig.java # Configuración de conexión a BD (Singleton/Pool)
│  │  ├──── dao/ # Capa de Acceso a Datos
│  │  │  ├──── impl/
│  │  │  │  └──── UsuarioDAOImpl.java # Implementación concreta para Usuario
│  │  │  └──── interfaces/
│  │  │  ├──── IGenericDAO.java # Interfaz genérica (CRUD base)
│  │  │  ├──── ITramiteDAO.java # Contrato para operaciones de Trámites
│  │  │  └──── IUsuarioDAO.java # Contrato para operaciones de Usuarios
│  │  ├──── model/ # Capa de Modelo (Entidades)
│  │  │  ├──── entities/
│  │  │  │  ├──── Solicitante.java # POJO de la entidad Solicitante
│  │  │  │  └──── Usuario.java # POJO de la entidad Usuario
│  │  │  ├──── enums/
│  │  │  │  └──── Rol.java # Enumeración para roles (ADMIN, ANALISTA)
│  │  │  └──── exceptions/
│  │  │  └──── BaseDatosException.java # Manejo de errores personalizados
│  │  ├──── service/ # Capa de Lógica de Negocio
│  │  │  ├──── AuthService.java # Lógica de autenticación y sesiones
│  │  │  └──── SupabaseService.java # Lógica de comunicación con la nube
```

```
│   │   ├──── util/ # Utilitarios Transversales
│   │   │   ├──── PDFGenerator.java # Generación de reportes/licencias
│   │   │   └──── Sesion.java # Gestión del usuario logueado actualmente
│   │   ├──── view/ # Capa de Vista (Interfaz Gráfica)
│   │   │   ├──── DetalleTramite/ # Componentes para visualizar detalles
│   │   │   ├──── GenerarLicencia/ # Formulario de emisión final
│   │   │   ├──── GestionTramites/ # Tabla y filtros de trámites
│   │   │   ├──── GestionUsuarios/ # CRUD de usuarios (Admin)
│   │   │   ├──── Login/ # Ventana de acceso
│   │   │   ├──── MenuPrincipal/ # Dashboard principal
│   │   │   ├──── RegistroExamenes/ # Ingreso de notas
│   │   │   ├──── RegistroSolicitante/ # Alta de nuevos ciudadanos
│   │   │   ├──── Reportes/ # Módulo estadístico
│   │   │   └──── VerificacionRequisitos/ # Checklist de validación
│   │   └──── Main.java # Punto de entrada de la aplicación
│   └──── resources/ # Archivos estáticos (imágenes, configs)
├──── target/ # Salida de compilación (JARs)
├──── pom.xml # Dependencias Maven
└──── README.md # Documentación base
```

2.2 Análisis de Componentes Clave

Paquete config

- **DatabaseConfig.java**: Esta clase es responsable de establecer el puente con Supabase. Implementa el patrón **Singleton** para asegurar que no se creen múltiples instancias de conexión innecesarias, optimizando el consumo de recursos de red.

Paquete dao (Data Access Object)

Se aplica estrictamente el principio de inversión de dependencias:

- **interfaces/IGenericDAO.java**: Define operaciones comunes (create, read, update, delete) usando genéricos de Java, permitiendo reutilizar la firma de métodos para cualquier entidad.
- **impl/UsuarioDAOImpl.java**: Contiene el código SQL específico. Aquí se ejecutan las sentencias PreparedStatement para insertar o consultar usuarios de manera segura contra inyección SQL.

Paquete service

Aquí reside la inteligencia del sistema:

- **AuthService.java**: Gestiona el login. Verifica el hash de la contraseña y, si es correcto,

inicializa la clase Sesion.
- **SupabaseService.java**: Centraliza la lógica de negocio específica para interactuar con los servicios de Supabase, actuando como intermediario entre los controladores de la vista y los datos remotos.

Paquete view

A diferencia de proyectos monolíticos desordenados, este proyecto separa cada pantalla en su propio subpaquete (e.g., view/Login), conteniendo tanto la clase lógica .java como el diseño del formulario .form (si se usa el diseñador de IntelliJ). Esto facilita el trabajo colaborativo entre los dos desarrolladores, minimizando conflictos de fusión (merge conflicts).

---

3. Implementación de los Pilares de la POO

El proyecto demuestra la aplicación práctica de los conceptos teóricos enseñados en la EPN:

1. **Encapsulamiento:**
   - Clases como Usuario y Solicitante en el paquete model.entities protegen sus atributos (e.g., private String password;) y solo permiten el acceso mediante métodos getters y setters controlados.
2. **Abstracción:**
   - El uso de interfaces en dao.interfaces (ITramiteDAO, IUsuarioDAO) permite que el resto del sistema interactúe con "contratos" abstractos sin conocer los detalles de la implementación SQL subyacente.
3. **Herencia:**
   - Aunque no visible explícitamente en el árbol, es práctica común que los DAOs específicos (como UsuarioDAOImpl) extiendan de una clase base abstracta o implementen la interfaz genérica IGenericDAO, heredando comportamientos comunes.
4. **Polimorfismo:**
   - La capacidad de tratar a UsuarioDAOImpl como un IUsuarioDAO permite cambiar la implementación de la base de datos en el futuro sin romper el código de los servicios que la consumen.

---

4. Funcionalidades y Flujo de Usuario

4.1 Módulo de Seguridad (AuthService y Login)

- **Entrada:** Credenciales del usuario.

- **Proceso:** El AuthService consulta a través de UsuarioDAO si el usuario existe y está activo. Utiliza librerías de seguridad (BCrypt) para comparar hashes.
- **Sesión:** Si es exitoso, se instancia util.Sesion con los datos del usuario logueado (Nombre, Rol), permitiendo que el MenuPrincipal ajuste las opciones visibles (Admin vs. Analista).

4.2 Gestión de Trámites (GestionTramites, RegistroSolicitante)

- **Inicio:** El Analista usa RegistroSolicitante para crear la entidad Solicitante y abrir un trámite en estado PENDIENTE.
- **Validación:** En VerificacionRequisitos, se marcan los checkboxes. La lógica impide avanzar si no se cumplen las reglas de negocio.
- **Evaluación:** RegistroExamenes captura las notas. La lógica de aprobación (nota >= 14) reside probablemente en el controlador de esta vista o en un método de servicio invocado desde aquí.

4.3 Generación de Documentos (PDFGenerator)

- Una vez aprobado el trámite, la clase util.PDFGenerator toma los datos de las entidades y construye un documento final (Licencia) listo para imprimir o guardar, cerrando el ciclo del trámite.

---

5. Tecnologías y Herramientas

Componente	Tecnología Seleccionada	Justificación
Lenguaje	**Java (JDK)**	Estándar robusto, fuertemente tipado.
Framework UI	**Java Swing / FX**	Basado en la estructura sistemalicenciasfx, el uso de componentes visuales nativos.
Base de Datos	**Supabase (PostgreSQL)**	Plataforma PaaS moderna que ofrece BD relacional con alta disponibilidad.

Build Tool	**Maven**	Gestiona las dependencias (driver JDBC, librerías PDF, etc.) declaradas en pom.xml.
Control de Versiones	**Git**	Fundamental para el trabajo en equipo de Michael y Kevin.

---

6. Conclusión

El proyecto "Sistema de Matriculación MVC" desarrollado por **Michael Carrillo** y **Kevin Amagua** cumple con los estándares de un desarrollo de software profesional de nivel tecnológico. La estructura de carpetas denota una clara separación de preocupaciones (SoC), aislando la configuración de base de datos, la lógica de negocio y la interfaz de usuario. Esto asegura que el sistema sea escalable, auditable y mantenible a largo plazo.