Arquitectura Hexagonal aplicada a tu proyecto (Spring + Gradle + SQL Server)

1. Mapa mental: dónde encaja Hexagonal en tu proyecto  
  
Núcleo (Dominio)  
- Entidades (Cliente), Interfaces (puertos) como ClienteRepositoryPort y casos de uso (ObtenerClienteUseCase).  
- Aquí sólo van reglas de negocio y contratos (interfaces).  
  
Adaptadores de entrada (Driving Adapters)  
- REST controllers (ClienteController) que exponen endpoints (Swagger).  
  
Adaptadores de salida (Driven Adapters)  
- Implementación del repositorio que conversa con SQL Server (Spring Data JPA / Repository implementation).  
- Implementación que llama al Stored Procedure (si decides usar JdbcTemplate).  
  
DTOs  
- Objetos de transferencia entre capa API y dominio (ClienteDTO).  
  
Servicios  
- Implementan casos de uso invocando puertos; validaciones (tipo documento C/P) y deciden códigos HTTP a devolver (vía excepciones manejadas por un handler global).  
  
Este esquema garantiza que si cambias DB o el canal (por ejemplo agregar gRPC), sólo tocas adaptadores, no el dominio.

2. Script SQL: crear BD, tabla y Stored Procedure  
-- 1. Crear base de datos (si no existe)  
CREATE DATABASE DBClientes;  
GO  
  
USE DBClientes;  
GO  
  
-- 2. Crear tabla Clientes  
CREATE TABLE Clientes (  
 Id INT IDENTITY(1,1) PRIMARY KEY,  
 TipoDocumento CHAR(1) NOT NULL, -- 'C' o 'P'  
 NumeroDocumento VARCHAR(50) NOT NULL,  
 PrimerNombre VARCHAR(100),  
 SegundoNombre VARCHAR(100),  
 PrimerApellido VARCHAR(100),  
 SegundoApellido VARCHAR(100),  
 Telefono VARCHAR(50),  
 Direccion VARCHAR(200),  
 CiudadResidencia VARCHAR(100),  
 CONSTRAINT UQ\_Clientes\_Tipo\_Num UNIQUE (TipoDocumento, NumeroDocumento)  
);  
GO  
  
-- 3. SP para obtener datos del cliente por tipo y número  
CREATE PROCEDURE ObtenerClientePorDocumento  
 @TipoDocumento CHAR(1),  
 @NumeroDocumento VARCHAR(50)  
AS  
BEGIN  
 SET NOCOUNT ON;  
  
 SELECT  
 PrimerNombre,  
 SegundoNombre,  
 PrimerApellido,  
 SegundoApellido,  
 Telefono,  
 Direccion,  
 CiudadResidencia  
 FROM Clientes  
 WHERE TipoDocumento = @TipoDocumento  
 AND NumeroDocumento = @NumeroDocumento;  
END  
GO

3. Cómo mapear requisitos a la Arquitectura Hexagonal  
  
Requisito: GET /api/clientes/{identificación} + tipo y número obligatorios →  
Propuesta práctica: usar ruta con ambos: GET /api/clientes/{tipo}/{numero}  
  
Códigos HTTP:  
- 200 → cliente encontrado, devuelve DTO.  
- 404 → no existe (lanza ClienteNotFoundException).  
- 400 → validación (tipo inválido, falta de parámetros).  
- 500 → error interno.  
  
Puerto por el cual inicia la app: server.port=8090 en application.properties.  
  
Swagger: usar springdoc-openapi para documentar y probar el endpoint.

4. Código de ejemplo (Spring Boot + Gradle) — piezas clave  
  
build.gradle (dependencias principales)  
--------------------------------------  
dependencies {  
 implementation 'org.springframework.boot:spring-boot-starter-web'  
 implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
 implementation 'com.microsoft.sqlserver:mssql-jdbc:11.2.2.jre17'  
 implementation 'org.springframework.boot:spring-boot-starter-validation'  
 implementation 'org.springdoc:springdoc-openapi-ui:2.1.0'  
 testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}  
  
application.properties  
----------------------  
server.port=8090  
  
spring.datasource.url=jdbc:sqlserver://<HOST>:1433;databaseName=DBClientes  
spring.datasource.username=<DB\_USER>  
spring.datasource.password=<DB\_PASS>  
spring.datasource.driver-class-name=com.microsoft.sqlserver.jdbc.SQLServerDriver  
  
spring.jpa.hibernate.ddl-auto=none  
spring.jpa.show-sql=true  
  
Dominio: Cliente (Entity)  
-------------------------  
@Entity  
@Table(name = "Clientes")  
public class Cliente {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Integer id;  
 @Column(name = "TipoDocumento", length = 1)  
 private String tipoDocumento;  
 @Column(name = "NumeroDocumento")  
 private String numeroDocumento;  
 private String primerNombre;  
 private String segundoNombre;  
 private String primerApellido;  
 private String segundoApellido;  
 private String telefono;  
 private String direccion;  
 private String ciudadResidencia;  
 // getters y setters  
}  
  
DTO: ClienteDTO  
----------------  
public class ClienteDTO {  
 private String primerNombre;  
 private String segundoNombre;  
 private String primerApellido;  
 private String segundoApellido;  
 private String telefono;  
 private String direccion;  
 private String ciudadResidencia;  
 // constructor, getters, setters  
}  
  
Puerto (Interfaz)  
-----------------  
public interface ClienteRepositoryPort {  
 Optional<ClienteDTO> obtenerPorTipoYNumero(String tipo, String numero);  
}  
  
Adaptador de salida: JdbcTemplate  
---------------------------------  
@Repository  
public class ClienteRepositoryJdbc implements ClienteRepositoryPort {  
 private final JdbcTemplate jdbcTemplate;  
 public ClienteRepositoryJdbc(JdbcTemplate jdbcTemplate) {  
 this.jdbcTemplate = jdbcTemplate;  
 }  
 @Override  
 public Optional<ClienteDTO> obtenerPorTipoYNumero(String tipo, String numero) {  
 String sql = "EXEC ObtenerClientePorDocumento @TipoDocumento = ?, @NumeroDocumento = ?";  
 try {  
 return Optional.ofNullable(jdbcTemplate.queryForObject(  
 sql,  
 new Object[]{tipo, numero},  
 (rs, rowNum) -> {  
 ClienteDTO dto = new ClienteDTO();  
 dto.setPrimerNombre(rs.getString("PrimerNombre"));  
 dto.setSegundoNombre(rs.getString("SegundoNombre"));  
 dto.setPrimerApellido(rs.getString("PrimerApellido"));  
 dto.setSegundoApellido(rs.getString("SegundoApellido"));  
 dto.setTelefono(rs.getString("Telefono"));  
 dto.setDireccion(rs.getString("Direccion"));  
 dto.setCiudadResidencia(rs.getString("CiudadResidencia"));  
 return dto;  
 }  
 ));  
 } catch (EmptyResultDataAccessException ex) {  
 return Optional.empty();  
 }  
 }  
}  
  
Servicio (caso de uso)  
-----------------------  
@Service  
public class ClienteService {  
 private final ClienteRepositoryPort clienteRepository;  
 public ClienteService(ClienteRepositoryPort clienteRepository) {  
 this.clienteRepository = clienteRepository;  
 }  
 public ClienteDTO obtenerCliente(String tipo, String numero) {  
 if (tipo == null || (!tipo.equalsIgnoreCase("C") && !tipo.equalsIgnoreCase("P"))) {  
 throw new BadRequestException("Tipo de documento inválido. Debe ser 'C' o 'P'.");  
 }  
 return clienteRepository.obtenerPorTipoYNumero(tipo.toUpperCase(), numero)  
 .orElseThrow(() -> new ClienteNotFoundException("Cliente no encontrado"));  
 }  
}  
  
Controller (adaptador de entrada)  
----------------------------------  
@RestController  
@RequestMapping("/api/clientes")  
@Tag(name = "Clientes", description = "Operaciones para consultar clientes")  
public class ClienteController {  
 private final ClienteService clienteService;  
 public ClienteController(ClienteService clienteService) {  
 this.clienteService = clienteService;  
 }  
 @Operation(summary = "Obtener cliente por tipo y número de documento")  
 @ApiResponses(value = {  
 @ApiResponse(responseCode = "200", description = "Cliente encontrado"),  
 @ApiResponse(responseCode = "400", description = "Solicitud inválida"),  
 @ApiResponse(responseCode = "404", description = "Cliente no encontrado"),  
 @ApiResponse(responseCode = "500", description = "Error interno")  
 })  
 @GetMapping("/{tipo}/{numero}")  
 public ResponseEntity<ClienteDTO> obtenerCliente(  
 @PathVariable("tipo") String tipo,  
 @PathVariable("numero") String numero) {  
 ClienteDTO cliente = clienteService.obtenerCliente(tipo, numero);  
 return ResponseEntity.ok(cliente);  
 }  
}  
  
Manejo de errores global  
-------------------------  
@RestControllerAdvice  
public class GlobalExceptionHandler {  
 @ExceptionHandler(BadRequestException.class)  
 public ResponseEntity<Object> handleBadRequest(BadRequestException ex) {  
 return ResponseEntity.status(HttpStatus.BAD\_REQUEST)  
 .body(Map.of("error", ex.getMessage()));  
 }  
 @ExceptionHandler(ClienteNotFoundException.class)  
 public ResponseEntity<Object> handleNotFound(ClienteNotFoundException ex) {  
 return ResponseEntity.status(HttpStatus.NOT\_FOUND)  
 .body(Map.of("error", ex.getMessage()));  
 }  
 @ExceptionHandler(Exception.class)  
 public ResponseEntity<Object> handleAll(Exception ex) {  
 return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)  
 .body(Map.of("error", "Error interno del servidor"));  
 }  
}  
  
Excepciones personalizadas  
---------------------------  
public class BadRequestException extends RuntimeException {  
 public BadRequestException(String msg){ super(msg); }  
}  
public class ClienteNotFoundException extends RuntimeException {  
 public ClienteNotFoundException(String msg){ super(msg); }  
}

5. Swagger / OpenAPI  
Con la dependencia springdoc-openapi-ui activa, accedes a:  
http://localhost:8090/swagger-ui.html

6. Validaciones y manejo de HTTP codes  
- 400 Bad Request → tipo inválido o parámetros faltantes.  
- 404 Not Found → cliente no existe.  
- 500 Internal Server Error → error interno DB.  
- 200 OK → cliente encontrado.

7. Pruebas manuales  
INSERT INTO Clientes (TipoDocumento, NumeroDocumento, PrimerNombre, SegundoNombre, PrimerApellido, SegundoApellido, Telefono, Direccion, CiudadResidencia)  
VALUES ('C', '12345678', 'Aura', 'Cristina', 'Garzon', 'Rodriguez', '3001234567', 'Calle Falsa 123', 'Bogotá');

8. Cómo explicar en la sustentación  
- Hexagonal desacopla negocio de infraestructura.  
- Puertos y adaptadores claros (entrada: Controller, salida: RepositoryJdbc).  
- Fácil de testear con mocks.  
- Manejo de errores centralizado.  
- Todos los requisitos cumplidos con Spring + Gradle.

9. Diagrama conceptual (texto)  
 +----------------------+  
 -> | REST Controller |  
 | +----------------------+   
 | |  
 | v  
+----------+---------+ +----------------+  
| Dominio Core |<->| DTOs |  
| - ClienteService | | ClienteDTO |  
| - Cliente Port(s) | +----------------+  
+--------------------+  
 |  
 v  
 +---------------------+  
 | ClienteRepository |  
 +---------------------+  
 |  
 v  
 +---------------------+  
 | SQL Server (SP / DB)|  
 +---------------------+