**Challenge back:**

el challenge que se trata de generar los siguientes 3 endpoints:

- 1.Uno que traiga las empresas que hicieron transferencias el último mes

- 2. Otro que traiga las empresas que se adhirieron el último mes.

-3 El último que haga la adhesión de una empresa.

Deseable: usar arquitectura hexagonal**(o utilizar servicio con el que te sientas más sólido/seguro)**

Base: puede usarse **relacional** o no relacional

**Datos de la empresa: CUIT, Razón Social, Fecha Adhesión**

**Datos de la transferencia: Importe, Id Empresa, Cuenta Débito, Cuenta Crédito**

**Datos de apitrasaccionDetalle CUIT, Razón Social, Fecha Adhesión,Importe, Id Empresa, Cuenta Débito, Cuenta Crédito,** **fechaTransaccion**

**Realizar test unitarios.**

Las dudas en el desarrollo asumirlas y ponerlas en un archivo de aclaracion.

**Aclarcaiones**:

**Agrege al modelo una clase ApiTransaccionDetalle** , con el fin de tener los dato de La empresa adherida con la transferencia en una misma tabla **ApiTransaccionDetalle** y así mejorar los tiempo de respesta de las consulta al realizar un select a la misma.Esta tabla que expone la información detalladas de las transferecias de las empresa , se cargaria con un storeprocedure /dtsx .

**Campos de apitrasaccionDetalle CUIT, Razón Social, Fecha Adhesión,Importe, Id Empresa, Cuenta Débito, Cuenta Crédito,** **fechaTransaccion.**

**TEST JUNIT**

**Los tests junit** los implemente sobre los **controles** , el obejitvo fue realizar pruebas de integración

**VALIDACIONES**

La separe en dos packetes a nivel de los DTOs y del MODELO. (No necesariamente son iguales los dtos y las entidades del modelos).

**TODO**

Falto implementar los test de dependencia; para ver si existe depencia circular , jars deprecados.El objetivo es que las capas no esten acopladas con Archunit ; para estar alineado, con los criterios de arquitectura limpia.Como complemento al junit ,se le puede agregar un test simple d carga utilizando rest assured.

**IMPLEMENTACIÓN DE LA SOLUCION**

**El Punto 1 y 2 se resuelve implementado las consutas utilizando @Query**

**Punto 1** Uno que traiga las empresas que hicieron transferencias el último mes

@Query(value = "SELECT DISTINCT e.\* FROM Empresa e " + "JOIN ApiTransaccionDetalle t ON e.cuit = t.cuitEmpresa " + "WHERE t.fechaTransaccion >= DATE\_SUB(CURDATE(), INTERVAL 1 MONTH)", nativeQuery = true) List<Empresa> findEmpresasConTransferenciasUltimoMes();

2. Otro que traiga las empresas que se adhirieron el último mes.

@Query("SELECT e FROM Empresa e WHERE e.fechaAdhesion >= :fechaInicioMesAnterior") List<Empresa> findEmpresasAdheridasUltimoMes(@Param("fechaInicioMesAnterior") LocalDate fechaInicioMesAnterior);

//3 Metodao para adherir una empresa  
public Empresa adherirEmpresa(Empresa empresa) {  
 empresa.setFechaAdhesion(LocalDate.*now*());  
 return empresaRepository.save(empresa);  
}

**Detalle de la solución**

Los tres endpoints

**// 1-get Trealas empresas que hicieron transferencias el último mes**  
@GetMapping("/findEmpresasConTransferenciasUltimoMes")  
public List<Empresa> findEmpresasConTransferenciasUltimoMes() {  
 return empresaService.findEmpresasConTransferenciasUltimoMes();  
}  
**//2. Otro que traiga las empresas que se adhirieron el último mes.**  
@GetMapping("/adheridas-ultimo-mes")  
public List<Empresa> obtenerEmpresasAdheridasUltimoMes() {  
 return empresaService.obtenerEmpresasAdheridasUltimoMes();  
}  
**//3 Metodao para adherir una empresa**  
@PostMapping("/adherir")  
public ResponseEntity<Empresa> adherirEmpresa(@RequestBody Empresa empresa) {  
 Empresa empresaAdherida = empresaService.adherirEmpresa(empresa);  
 return new ResponseEntity<>(empresaAdherida, HttpStatus.*CREATED*);  
}

DO

Se implemento un controller de prueba de las validaciones, las validaciones las aplico en los mapper y no antes @RequestBody

@Mapper  
public interface DatosMapper {  
  
  
 DatosMapper *INSTANCE* = Mappers.*getMapper*(DatosMapper.class);  
  
 @Mapping(source = "id", target = "id")  
 @Valid  
 Empresa empresaDTOaEmpresa(EmpresaDTO empresaDTO);  
  
 @Valid  
 @Mapping(source = "id", target = "id")  
 Transferencia transferenciaDTOaTransferencia(TransferenciaDTO transferenciaDTO);  
  
 @Valid  
 @Mapping(source = "id", target = "id")  
 EmpresaDTO empresaAempresaDTO(Empresa empresa);  
  
 @Valid  
 @Mapping(source = "id", target = "id")  
 TransferenciaDTO transferenciaAtransferenciaDTO(Transferencia transferencia);  
}

**CONTROLLER**

@RestController  
@RequestMapping("api/validacion/")  
public class ValidadorControlle {  
  
 private DatosMapper datosMapper;  
 @Autowired  
 private EmpresaService empresaService;  
 @Autowired  
 private TransferenciaService transferenciaService;  
  
 public ValidadorControlle(DatosMapper datosMapper) {  
 this.datosMapper = datosMapper;  
 }  
  
 @PostMapping("/empresas")  
 public ResponseEntity<String> crearEmpresa( @RequestBody EmpresaDTO empresaDTO) {  
 //*TODO Lógica para crear la empresa* empresaService.save(datosMapper.empresaDTOaEmpresa(empresaDTO));  
 return ResponseEntity.*ok*("Empresa creada con éxito");  
 }  
  
 @PostMapping("/transferencias")  
 public ResponseEntity<String> crearTransferencia(@Valid @RequestBody DatosTransferenciaDTO transferenciaDTO) {  
 // TODO Lógica para crear la transferenciatransferenciaService.save(datosMapper.empresaDTOaEmpresa(transferenciaDTO));  
 return ResponseEntity.*ok*("Transferencia creada con éxito");  
 }  
  
  
}