

Instituto Tecnológico Colonia CTC

Obligatorio Diseño y Desarrollo de Aplicaciones
Obligatorio 2

Leandro Chelentano

Guadalupe Dovat

Profesor: Carlos Rodríguez

Año: 2022

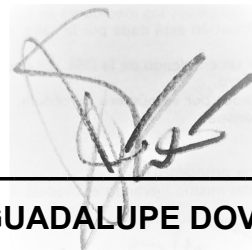
Declaración de autoría.

Nosotros, Leandro Chelentano y Guadalupe Dovat, declaramos que el trabajo que se presenta es de nuestra propia mano. Puedo asegurar que:

- La obra fue producida mientras cursaba la materia Diseño y Desarrollo de Aplicaciones con el profesor Carlos Rodriguez;
- He tenido en cuenta las clases dictadas por el Prof. Carlos Rodriguez, quien además proporcionó el material;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente mía;
- En la obra, he acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente qué fue construido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



LEANDRO CHELENTANO



GUADALUPE DOVAT

Obligatorio 2 DDA

Fecha: desde el 03 de noviembre al 30 de noviembre de 2022, a las 23:59.

Una empresa de turismo quiere manejar los planes para ofertar a sus clientes y luego poder asociar esos planes con los clientes que los adquieran.

Hay clientes estándar y clientes premium. Los clientes premium son aquellos que ya han adquirido 3 planes de viajes o más. Para ellos, el costo del cuarto plan en adelante tiene un 20% de descuento.

EL SISTEMA DEBE PERMITIR:

1) Mostrar un dashboard administrativo en ambiente Web o Mobile (sencillo) que permita trabajar con planes de viaje y clientes.

2) Dar de alta, eliminar y modificar planes de viaje. Cada uno de ellos tiene un solo **destino** (máx. 20 dígitos alfanuméricos), **fecha**, **modalidad** (solamente pueden ser aérea, marítima o terrestre), **precio** en USD, carrusel de fotos (opcional).

2) dar de alta, eliminar y modificar clientes.

Los clientes tienen **CI** (máx. 8 dígitos, mín. 7 dígitos, sin puntos ni guiones), **nombre** (máx. 30 dígitos alfanuméricos), **apellido** (máx. 30 dígitos alfanuméricos), **email** (máx. 30 dígitos alfanuméricos), **planes comprados** (si no tiene ninguno se deberá indicar con un mensaje que "no tiene planes comprados").

3) Contratar o borrarse de un plan (o más) de viaje.

4) Listar los planes (viajes) de un cliente mostrando todos los atributos de cada viaje.

5) Listar el primer viaje que tendrá un cliente después de una fecha especificada.

6) Controlar que no se permita ingresar viajes (planes) con fechas anteriores a la fecha actual.

7) Controles de errores para ingreso de datos.

PASOS A SEGUIR:

1) Analizar la realidad planteada y presentar una solución en Java.

2) Realizar el diagrama de clases con sus métodos y atributos.

3) Realizar el código que una vez ejecutado permita implementar la solución al problema.

A ENTREGAR:

1. Documentación en formato PDF que contenga:

- a. Portada del trabajo con los nombres de los integrantes del equipo.
- b. Letra del problema.

- c. Descripción del análisis y la solución.
 - d. Alcances obtenidos a consideración del equipo.
 - e. Diagrama de clases.
 - f. Código impreso de las clases.
2. El código de las clases implementadas:
- a. en el backend, en un archivo rar llamado
backendObligatorio2-ApellidoDeAlumno1-ApellidoDeAlumno2
 - b. en el frontend, en un archivo rar llamado
frontendObligatorio2-ApellidoDeAlumno1-ApellidoDeAlumno2
3. La base de datos (preferentemente en MySQL v 5.7.40 o SQL Server v 2012).

Descripción del problema

Se nos presenta una situación sumamente simple, desarrollar una REST API en Java haciendo uso del framework Spring Boot y una aplicación FrontEnd desarrollado en una tecnología de nuestra preferencia que sea capaz de consumir al API y sea nuestra capa de presentación.

Nuestra solución debería ser capaz de hacer operaciones de alta, baja y modificación sobre clientes y viajes, quienes tienen entre sí una relación N a N, representando esta la compra de viajes por parte de los clientes.

Se nos pactan determinados atributos que determinadas clases deben contener, además se nos introducen diferentes tipos de clientes, concretamente son solo dos, clientes estándar y VIP, siendo estos últimos los clientes que han comprado más de tres viajes, tendiendo estos la ventaja de tener un 20% de descuento en sus compras.

Se nos pide que el usuario reciba 'feedback' del programa en caso de errores en cualquier caso.

Por último se nos presenta una funcionalidad añadida, a partir de un cliente y una fecha, debemos decir cual es a futuro el viaje más próximo a esa fecha.

Nuestra solución;

En cuanto a las tecnologías, haremos uso de todo lo requerido en la parte del BackEnd. Debido a que se nos recomienda el uso de MySQL como motor de base de datos y el hecho de que es el motor de más fácil acceso de uno de nosotros, decidimos hacer uso de este.

Mientras tanto, en el FrontEnd hemos optado por el uso de ReactJS, empleando el fetching nativo de JavaScript para emplear la menor cantidad de dependencias posibles y no hacer que el proyecto sea demasiado pesado para un obligatorio tan simple.

Referente al código, empleamos los atributos listados en la letra. Para diferenciar entre clientes estándar y VIP optamos por utilizar un atributo de tipo Boolean puesto que una herencia nos parece en exceso desmedido a fines prácticos.

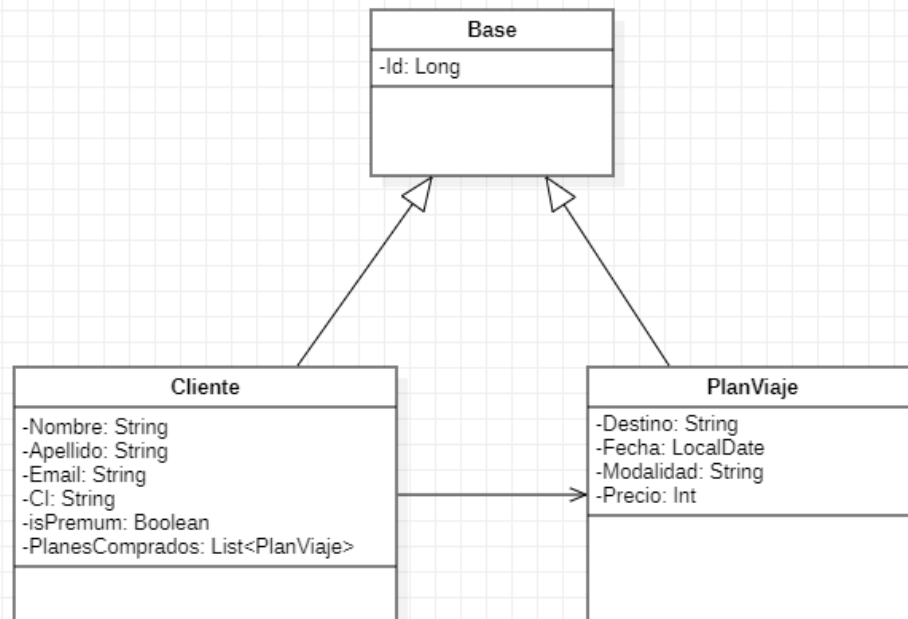
En las operaciones comunes (GET, PUT, POST, etc.) a las entidades Clientes y PlanViaje implementamos una interfaz para simplicidad y reutilización de código. Por su lado, la relación ManyToMany tiene su propia clase controladora con solo dos métodos, POST y DELETE, puesto que no necesitamos operaciones extras como lectura, puesto que esta información viaja propiamente dentro del objeto cliente en el body de la response en forma de JSON.

Debemos hacer una aclaración, hemos decidido no permitir al usuario dar de baja un plan de viaje que ya ha sido comprado por algún cliente para evitar posibles errores.

Alcances obtenidos

Quienes formamos parte de este grupo consideramos que los objetivos planteados por el docente fueron plenamente cumplidos a excepción del slider de imágenes, puesto que consideramos, no es lo suficientemente beneficioso como para compensar los problemas de cohesión que generaría en lo que respecta a la estética de nuestro Front End.

Diagrama de Clases



Código

entity/Base.java

```
@MappedSuperclass
public class Base implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long Id;

    //#region Getters-Setters
    public Long getId() {
        return Id;
    }

    public void setId(Long id) {
        Id = id;
    }
    //#endregion
}
```

entity/Cliente.java

```
@Entity
@Table (name = "clientes")
public class Cliente extends Base {
    @Column (nullable = false, length = 30)
    private String Nombre;

    @Column (nullable = false, length = 30)
    private String Apellido;

    @Column (nullable = false, length = 30, unique = true)
    private String Email;

    @Column (nullable = false, unique = true, length = 8)
    private String CI;

    @Column (columnDefinition = "bool default(false)", nullable = false)
    private boolean isPremium;

    @ManyToMany(cascade = CascadeType.REFRESH)
    private List<PlanViaje> PlanesComprados = new ArrayList<PlanViaje>();

    //#region Getters-Setters
    public String getNombre() {
```



```
    return this.Nombre;
}

public void setNombre(String nombre) {
    this.Nombre = nombre;
}

public String getApellido() {
    return this.Apellido;
}

public void setApellido(String apellido) {
    this.Apellido = apellido;
}

public String getCI() {
    return this.CI;
}

public void setCI(String CI) {
    this.CI = CI;
}

public String getEmail() {
    return this.Email;
}

public void setEmail(String email) {
    this.Email = email;
}

public boolean isPremium() {
    return this.isPremium;
}

public void setIsPremium(boolean isPremium) {
    this.isPremium = isPremium;
}

public List<PlanViaje> getViajes() {
    return this.PlanesComprados;
}

public void setViajes(List<PlanViaje> PlanesComprados) {
    this.PlanesComprados = PlanesComprados;
}
//#endregion

public Cliente() { }
```

```
public Cliente(String Nombre, String Apellido, String Email, boolean isPremium, String
CI) {
    this.Nombre = Nombre;
    this.Apellido = Apellido;
    this.Email = Email;
    this.CI = CI;
    this.isPremium = isPremium;
}
}
```

entity/PlanViaje.java

```
@Entity
@Table (name = "planes")
public class PlanViaje extends Base {
    @Column (length = 20)
    private String Destino;

    @Column (name="Fecha")
    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
    private LocalDate Fecha;

    @Column (nullable = false, length = 20)
    private String Modalidad;

    @Column (nullable = false)
    private int Precio;

    ##region Getters-Setters
    public String getDestino() {
        return this.Destino;
    }

    public void setDestino(String Destino) {
        this.Destino = Destino;
    }

    public String getModalidad() {
        return this.Modalidad;
    }

    public void setModalidad(String Modalidad) {
        this.Modalidad = Modalidad;
    }

    public LocalDate getFecha() {
        return this.Fecha;
    }
}
```

```

public void setFecha(LocalDate Fecha) {
    this.Fecha = Fecha;
}

public int getPrecio() {
    return this.Precio;
}

public void setPrecio(int Precio) {
    this.Precio = Precio;
}
//#endregion

public PlanViaje() {}

public PlanViaje(String Destino, LocalDate Fecha, String Modalidad, int Precio) {
    this.Destino = Destino;
    this.Fecha = Fecha;
    this.Modalidad = Modalidad;
    this.Precio = Precio;
}
}

```

controller/BaseController.java

```

public interface BaseController<E extends Base, ID extends Serializable> {
    public ResponseEntity<?> create (@RequestBody E entity) throws Exception;

    public ResponseEntity<?> read (@PathVariable ID id) throws Exception;

    public ResponseEntity <?> update (@RequestBody E entity, @PathVariable ID id) throws
Exception;

    public List<E> readAll() throws Exception;

    public ResponseEntity <?> delete (@PathVariable ID id) throws Exception;
}

```

controller/BaseControllerImpl.java

```
public abstract class BaseControllerImpl<E extends Base, S extends
BaseServiceImpl<E, Long>> implements BaseController<E, Long> {
    @Autowired
    protected S servicio;

    @PostMapping
    public ResponseEntity<?> create(@RequestBody E entity) throws Exception {
        return ResponseEntity.status(HttpStatus.CREATED).body(servicio.save(entity));
    }

    @GetMapping("/{id}")
    public ResponseEntity<?> read(@PathVariable Long id) throws Exception {
        Optional<E> oE = servicio.findById(id);
        if (!oE.isPresent())
            return ResponseEntity.notFound().build();

        return ResponseEntity.ok(oE);
    }

    @PutMapping("/{id}")
    public ResponseEntity<?> update (@RequestBody E newData, @PathVariable Long
id) throws Exception{
        Optional<E> existingEntity = servicio.findById(id);
        if (!existingEntity.isPresent())
            return ResponseEntity.notFound().build();

        return ResponseEntity.status(HttpStatus.OK).body(servicio.update(id, newData));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> delete (@PathVariable Long id) throws Exception{
        if (!servicio.findById(id).isPresent())
            return ResponseEntity.notFound().build();

        servicio.deleteById(id);
        return ResponseEntity.ok().build();
    }

    @GetMapping
    public List<E> readAll() throws Exception{
        return (List<E>) StreamSupport
            .stream(servicio.findAll().spliterator(), false)
            .collect(Collectors.toList());
    }
}
```

controller/CientesController.java

```
@RestController
@RequestMapping("/clientes")
@CrossOrigin(origins = "*", allowedHeaders = "*")
public class CientesController extends BaseControllerImpl<Cliente, ClienteService> { }
```

controller/PlanViajeController.java

```
@CrossOrigin(origins = "*", allowedHeaders = "*")
@RestController
@RequestMapping("/planes")
public class PlanViajeController extends BaseControllerImpl<PlanViaje,
PlanViajeService> {
    @Override
    @PostMapping
    public ResponseEntity<?> create (@RequestBody PlanViaje plan) throws Exception {
        if (plan.getFecha().isBefore(LocalDate.now())) {
            return ResponseEntity.badRequest().build();
        }

        return ResponseEntity.status(HttpStatus.CREATED).body(servicio.save(plan));
    }

    @PutMapping("/{id}")
    public ResponseEntity<?> update (@RequestBody PlanViaje plan, @PathVariable
    Long id) throws Exception {
        Optional<PlanViaje> existingEntity = servicio.findById(id);
        if (!existingEntity.isPresent())
            return ResponseEntity.notFound().build();

        if (plan.getFecha().isBefore(LocalDate.now()))
            return ResponseEntity.badRequest().build();

        return ResponseEntity.status(HttpStatus.OK).body(servicio.update(id, plan));
    }
}
```

controller/RelationController.java

```
@CrossOrigin(origins = "*", allowedHeaders = "*")
@RestController
@RequestMapping("/relation")
public class RelationController {
    @Autowired
    protected ClienteService clienteService;

    @Autowired
    protected PlanViajeService planViajeService;

    @PostMapping
    public ResponseEntity<?> create(@RequestBody String body) throws Exception {
        Object dataO = JSONValue.parse(body);
        JSONObject data = (JSONObject) dataO;

        Long clienteld = (Long) data.get("cliente");
        Long viajeld = (Long) data.get("viaje");

        Optional<PlanViaje> viajeO = planViajeService.findByld(viajeld);
        Optional<Cliente> clienteO = clienteService.findByld(clienteld);

        if (!viajeO.isPresent() || !clienteO.isPresent())
            return ResponseEntity.notFound().build();

        Cliente cliente = clienteO.get();
        PlanViaje viaje = viajeO.get();

        List<PlanViaje> listaViajes = cliente.getViajes();
        Boolean founded = false;
        for (PlanViaje PV : listaViajes) {
            if (PV.getId() == viaje.getId()) {
                founded = true;
            }
        }

        if (founded || viaje.getFecha().isBefore(LocalDate.now())) {
            return ResponseEntity.badRequest().build();
        }

        listaViajes.add(viaje);
        cliente.setViajes(listaViajes);

        if (listaViajes.size() > 2) {
            cliente.setIsPremium(true);
        }

        return
```

```

ResponseEntity.status(HttpStatus.CREATED).body(clienteService.save(cliente));
}

@DeleteMapping
public ResponseEntity<?> delete(@RequestParam String cli, @RequestParam String
plan) throws Exception {
    Long clienteld = Long.valueOf(cli);
    Long viajeld = Long.valueOf(plan);

    Optional<PlanViaje> viajeO = planViajeService.findById(viajeld);
    Optional<Cliente> clienteO = clienteService.findById(clienteld);

    if (!viajeO.isPresent() || !clienteO.isPresent())
        return ResponseEntity.notFound().build();

    Cliente cliente = clienteO.get();
    PlanViaje viaje = viajeO.get();

    List<PlanViaje> listaViajes = cliente.getViajes();
    Boolean founded = false;
    for (PlanViaje PV : listaViajes) {
        if (PV.getId() == viaje.getId()) {
            founded = true;
        }
    }

    if (!founded) {
        return ResponseEntity.badRequest().build();
    }

    listaViajes.remove(viaje);
    cliente.setViajes(listaViajes);

    return
ResponseEntity.status(HttpStatus.CREATED).body(clienteService.save(cliente));
}
}

```

repository/BaseRepository.java

```
@NoRepositoryBean
public interface BaseRepository<E extends Base, ID extends Serializable> extends
JpaRepository<E, ID> {
    E save(Optional<E> entity);
}
```

repository/PlanViajeRepository.java

```
@Repository
public interface PlanViajeRepository extends BaseRepository<PlanViaje, Long> { }
```

repository/UserRepository.java

```
@Repository
public interface UserRepository extends BaseRepository<Cliente, Long> { }
```

service/BaseService.java

```
public interface BaseService<E extends Base, ID> extends Serializable {
    public Iterable<E> findAll() throws Exception;

    public Page<E> findAll(Pageable pageable) throws Exception;

    public Optional<E> findById(ID Id) throws Exception;

    public E save(E entity) throws Exception;

    public void deleteById (ID Id) throws Exception;

    public E update(Long id, E entity) throws Exception;
}
```


service/BaseServiceImpl.java

```
public abstract class BaseServiceImpl<E extends Base, ID extends Serializable>
implements BaseService<E, ID> {
    protected BaseRepository<E, ID> baseRepository;

    public BaseServiceImpl(BaseRepository<E, ID> baseRepository) {
        super();
        this.baseRepository = baseRepository;
    }

    @Override
    @Transactional(readOnly = true)
    public Iterable<E> findAll() throws Exception {
        try {
            return baseRepository.findAll();
        } catch (Exception e) {
            throw e;
        }
    }

    @Override
    @Transactional(readOnly = true)
    public Page<E> findAll(Pageable pageable) throws Exception {
        try {
            return baseRepository.findAll(pageable);
        } catch (Exception e) {
            throw e;
        }
    }

    @Override
    @Transactional(readOnly = true)
    public Optional<E> findById(ID id) throws Exception {
        try {
            return baseRepository.findById(id);
        } catch (Exception e) {
            throw e;
        }
    }

    @Override
    @Transactional
    public E save(E entity) throws Exception {
        try {
            return baseRepository.save((E) entity);
        } catch (Exception e) {
            throw e;
        }
    }
}
```

```

    }

    @Override
    @Transactional
    public void deleteById(ID Id) throws Exception {
        try {
            baseRepository.deleteById(Id);
        } catch (Exception e) {
            throw e;
        }
    }

    @Override
    @Transactional
    public E update(Long Id, E newData) throws Exception {
        try {
            newData.setId(Id);
            return baseRepository.save(newData);
        } catch (Exception e) {
            throw e;
        }
    }
}

```

service/ClienteService.java

```

@Service
public class ClienteService extends BaseServiceImpl<Cliente, Long>{
    public ClienteService(BaseRepository<Cliente, Long> baseRepository) {
        super(baseRepository);
    }
}

```

service/PlanViajeService.java

```

@Service
public class PlanViajeService extends BaseServiceImpl<PlanViaje, Long>{
    public PlanViajeService(BaseRepository<PlanViaje, Long> baseRepository) {
        super(baseRepository);
    }
}

```