PV Challenge - Manual Técnico (Versión 1.0.0)

Autor: Carlos Fecha de preparación: 2025-08-28

1) Estructura del Proyecto

```
com.carlos.challenge
|-- PvChallengeApplication (main)
|-- config
|   `-- SecurityConfig
|-- controller
|   |-- PointOfSaleController  (Rutas /api/pos ...)
|   |-- AccreditationController (Rutas /api/acreditaciones ...)
|   `-- CostController         (Rutas /api/costs ...)
|-- dto
|   |-- CreatePointRequest
|   |-- UpdatePointRequest
|   |-- CreateAccreditationRequest
|   |-- AccreditationResponse
|   |-- EdgeRequest
|   |-- MinPathResponse
|   `-- NeighborResponse
|-- model
|   |-- PuntoVenta
|   `-- Acreditacion
|-- repository
|   `-- AcreditacionRepository
|-- service
|   |-- PointCacheService
|   |-- AccreditationService
|   `-- GraphService
|-- service.impl
|   |-- PointCacheServiceImpl
|   |-- AccreditationServiceImpl
|   `-- GraphServiceImpl
`-- tests
    |-- unit.controller
    |   |-- PointOfSaleControllerTest
    |   |-- AccreditationControllerTest
    |   `-- CostControllerTest
    |-- unit.service
    |   |-- PointCacheServiceImplTest
    |   `-- GraphServiceImplTest
    `-- integration
        `-- AcreditacionIntegrationTest
```

2) Diagrama de Clases (texto)

Dependencias (conceptual):

- PointOfSaleController -> PointCacheService
- AccreditationController -> AccreditationService
- CostController -> GraphService
- AccreditationServiceImpl -> AcreditacionRepository (+ PointCacheService para enriquecer nombre PV)

```
- GraphServiceImpl          -> PointCacheService (para validar/nombrar
nodos)
- PointCacheServiceImpl     -> ConcurrentHashMap<Integer, String>
```

3) Detalle de Clases y Métodos

```
Classes y métodos principales (resumen):

1) config.SecurityConfig

   - SecurityFilterChain securityFilterChain(HttpSecurity): configura Basic
Auth, roles USER/ADMIN, paths protegidos.

   - (Opcional) In-memory users para desarrollo.

2) controller.PointOfSaleController (/api/pos)

   - ResponseEntity<List<PuntoVenta>> list()
   - ResponseEntity<PuntoVenta> create(@RequestBody @Valid
CreatePointRequest)
   - ResponseEntity<PuntoVenta> update(@PathVariable id, @RequestBody @Valid
UpdatePointRequest)
   - ResponseEntity<Void> delete(@PathVariable id)
   - ResponseEntity<PuntoVenta> findById(@PathVariable id)

3) controller.AccreditationController (/api/acreditaciones)
   - ResponseEntity<AccreditationResponse> crear(@RequestBody @Valid
CreateAccreditationRequest)
   - ResponseEntity<Page<AccreditationResponse>> listar(...
Optional<Integer> idPuntoVenta, Optional<Instant> from, to, Pageable)

4) controller.CostController (/api/costs)
   - ResponseEntity<Void> upsert(@RequestBody @Valid EdgeRequest)
[ADMIN]
   - ResponseEntity<Void> delete(@RequestBody @Valid EdgeRequest)
[ADMIN]
   - ResponseEntity<List<NeighborResponse>> neighbors(@PathVariable fromId)
[USER/ADMIN]
   - ResponseEntity<MinPathResponse> minPath(@RequestParam from, to)
[USER/ADMIN]

5) dto.*
   - CreatePointRequest { Integer id, String nombre }
   - UpdatePointRequest { String nombre }
   - CreateAccreditationRequest { BigDecimal amount, Integer idPuntoVenta }
   - AccreditationResponse { String id, BigDecimal importe, Integer
idPuntoVenta, String nombrePuntoVenta, Instant fechaRecepcion }
   - EdgeRequest { Integer fromId, Integer toId, int costo }
   - MinPathResponse { int costoTotal, List<Integer> rutaIds, List<String>
rutaNombres }
   - NeighborResponse { Integer id, String nombre, int costo }

6) model.PuntoVenta (record) { Integer id, String nombre }
   model.Acreditacion (@Document(collection = "acreditacionesV2"))
      - id (String), importe (BigDecimal), idPuntoVenta (Integer),
nombrePuntoVenta (String), fechaRecepcion (Instant)
```

- índices recomendados: {idPuntoVenta, fechaRecepcion} (compuesto) para filtros.

7) repository.AcreditacionRepository (Spring Data Mongo)
- Page<Acreditacion> findByIdPuntoVenta(Integer, Pageable)
- Page<Acreditacion> findByFechaRecepcionBetween(Instant, Instant, Pageable)

- Page<Acreditacion> findByIdPuntoVentaAndFechaRecepcionBetween(Integer, Instant, Instant, Pageable)

service.PointCacheService

- List<PuntoVenta> findAll()
- PuntoVenta create(Integer id, String nombre)
- PuntoVenta update(Integer id, String nombre)
- void delete(Integer id)
- PuntoVenta findById(Integer id)

impl.PointCacheServiceImpl
- Map<Integer, String> points = new ConcurrentHashMap<>()
- Validaciones y ResponseStatusException con 404/409

service.AccreditationService

- AccreditationResponse create(CreateAccreditationRequest)
- Page<AccreditationResponse> list(Optional<Integer> idPuntoVenta, Optional<Instant> from, Optional<Instant> to, Pageable)

impl.AccreditationServiceImpl

- Usa AcreditacionRepository y mapea a DTO. Enriquezco nombre PV pidiendo a PointCacheService.
- PaginaciÃ³n con PageRequest y filtros combinables.

service.GraphService

- void upsertEdge(int fromId, int toId, int costo)
- void removeEdge(int fromId, int toId)
- List<NeighborResponse> neighborsOf(int fromId)
- MinPathResponse shortestPath(int fromId, int toId)

impl.GraphServiceImpl
- adj: ConcurrentMap<Integer, ConcurrentMap<Integer, Integer>> (lista de adyacencia en memoria)
- Dijkstra con PriorityQueue<int[]>; dist, prev en HashMap; reconstrucciÃ³n de ruta.
- Valida nodos vÃ-a PointCacheService; simetrÃ-a de aristas A<->B.
- (Plan) SincronizaciÃ³n fina para escrituras usando StampedLock en upsert/delete.

4) Decisiones Técnicas

- Elegí- Basic Auth + roles en memoria para moverme rápido en desarrollo y concentrarme en la funcionalidad. En próximas versiones migro a JWT

- Para los Puntos de Venta, uso ConcurrentHashMap: la latencia es bajísima y es suficiente para CRUD en memoria del challenge. Si escala o necesito persistir, lo paso a Mongo o a una cache externa (Redis).

- En el grafo uso una lista de adyacencia con ConcurrentHashMap: la lectura (consultas de vecinos y Dijkstra) es dominante y barata. Las escrituras se hacen con computeIfAbsent y put (atómicas). Para evitar condiciones de carrera entre escrituras concurrentes propongo StampedLock alrededor de secciones críticas de upsert/remove (lecturas seguras lock-free).

- Las Acreditaciones van a Mongo para que el listado paginado con filtros sea eficiente y perdurable. Defino índices por fecha y por punto de venta para performance.
- La API es REST, consistente y predecible; el paginado devuelve Page<T> estÃ¡ndar de Spring para integrarse fÃ¡cil con front.
- Tests: tengo unit tests de controllers con MockMvc, unit de services y un test de integraciÃ³n end-to-end con Testcontainers + Mongo 7.

5) Que entrego en esta versión

Lo entregado en esta versión (que valor aporta):
- API completa y segura: CRUD de Puntos de Venta, altas y listados de Acreditaciones, y endpoints para grafo de costos (vecinos, camino mínimo).
- Paginación real: evito traer todo; página y tamaño configurables, listo para UIs.
- Código claro y probado: controllers con slice tests, servicios con unit tests, y flujo de integración real con contenedor de Mongo.
- Observabilidad lista para crecer: Actuator incluido y estructura para sumar métricas/logs estructurados.
- Entrega dockerizable: Dockerfile y docker-compose para correr todo con un solo comando.

6) Mejoras Planeadas

Mejoras planificadas (siguientes iteraciones):
- Seguridad JWT (stateless), roles via claims, expiración/refresh, y autorización por método.
- Arquitectura Hexagonal: separar puertos (services) de adaptadores (web/mongo/cache) para testear aún más fácil y poder cambiar infra sin tocar dominio.
- Carga de grafos por CSV/JSON y valor por defecto desde application.yml; endpoint de import para operaciones.
- Strategy para algoritmos de caminos: si aparecieran aristas negativas o necesidades distintas, poder conmutar entre Dijkstra, Bellman-Ford o A*.
- Índices compuestos en Mongo (idPuntoVenta, fechaRecepcion) y compound shard key si fuese necesario.
- Telemetria: trazas, métricas con Micrometer, dashboard en Dynatrace.

7) omo usarlo (Docker / local)

Como usarlo (local y Docker):
1) Requisitos: Docker Desktop y Docker Compose.
2) Build local del jar (opcional si se usa compose con build):
   mvn -DskipTests package

```
3) Levantar todo con Docker Compose:
   docker compose up --build -d
4) Ver logs de la app:
   docker compose logs -f app
5) Probar con curl (usuario por defecto):
   curl -i -u user:user http://localhost:8080/api/pos
6) Parar y limpiar:
   docker compose down
   docker volume ls | grep pv-challenge | xargs -I {} docker volume rm {}
7) Variables (compose):
   - SPRING_DATA_MONGODB_URI=mongodb://mongo:27017/pv
   - Puerto app: 8080 (host)
```

8) cURL de referencia

```
Puntos de Venta (ADMIN para escribir):

- Crear PV:
  curl -i -u admin:admin -H "Content-Type: application/json" -d
'{"id":1,"nombre":"Sucursal Centro"}' http://localhost:8080/api/pos

- Listar PVs:
  curl -i -u user:user http://localhost:8080/api/pos

- Obtener PV por id:
  curl -i -u user:user http://localhost:8080/api/pos/1

- Actualizar PV (ADMIN):
  curl -i -u admin:admin -X PUT -H "Content-Type: application/json" -d
'{"nombre":"Sucursal Centro (Nueva)"}' http://localhost:8080/api/pos/1

- Borrar PV (ADMIN):
  curl -i -u admin:admin -X DELETE http://localhost:8080/api/pos/1

Acreditaciones:

- Crear acreditaciÃ³n:
  curl -i -u user:user -H "Content-Type: application/json" -d
'{"amount":1234.56,"idPuntoVenta":1}'
http://localhost:8080/api/acreditaciones

- Listar (sin filtros, page/size):
  curl -i -u user:user
"http://localhost:8080/api/acreditaciones?page=0&size=20"

- Listar por PV:
  curl -i -u user:user
"http://localhost:8080/api/acreditaciones?idPuntoVenta=1&page=0&size=10"

- Listar por rango de fechas (ISO-8601):
  curl -i -u user:user
"http://localhost:8080/api/acreditaciones?from=2025-08-26T00:00:00Z&to=20
25-08-27T00:00:00Z&page=0&size=10"

Grafo de costos (/api/costs):

- Upsert arista (ADMIN):
```

```
  curl -i -u admin:admin -H "Content-Type: application/json" -d
'{"fromId":1,"toId":2,"costo":5}' http://localhost:8080/api/costs

- Eliminar arista (ADMIN):
  curl -i -u admin:admin -X DELETE -H "Content-Type: application/json" -d
'{"fromId":1,"toId":2,"costo":0}' http://localhost:8080/api/costs

- Vecinos:
  curl -i -u user:user http://localhost:8080/api/costs/neighbors/1

- Camino mínimo:
  curl -i -u user:user
"http://localhost:8080/api/costs/min-path?from=1&to=4"
```

9) Como lo entrego

- Subo el repo (código fuente) y dejo en la raiz:
  - Dockerfile
  - docker-compose.yml
  - pv-challenge.postman_collection.json
  - Este manual (.rtf) y un README.md con el resumen
- Instrucciones claras en el README: docker compose up --build -d
- Verifico que /swagger-ui/index.html abra correctamente (si springdoc está activo)
- Incluyo usuarios demo (user/user y admin/admin) en configuración de seguridad.

Si no llegara a incorporar el StampedLock aún:
- Problema: múltiples escrituras simultaneas podrían pisarse si intentan modificar el mismo mapa de adyacencia al mismo tiempo.
- Plan: envolveria upsertEdge/removeEdge con un StampedLock writeLock; reads (neighborsOf/shortestPath) con tryOptimisticRead y, si falla, con readLock. Esto mantiene lecturas rápidas y asegura coherencia en modificaciones.