EPIC STORE

MANUAL TECNICO

1 TABLA DE CONTENIDO

1. INTRODUCCION

*El sistema **EPIC STORE** ha sido analizado, desarrollado, sometido a pruebas de diferente tipo y al mismo tiempo dando soporte en cualquier fallo que se pudo haber presentado durante las mismas. Con el fin de presentar una práctica de la mejor manera y rendimiento posible a las personas encargadas de la calificación o a cualquier persona que pueda tener acceso a dicha práctica.*

Tratando de cumplir con los requerimientos establecidos en el documento proporcionado por el laboratorio del curso, así también implementando los conocimientos adquiridos durante las clases magistrales vistos hasta la fecha como lo son: Algoritmos, pseudocódigos, diagramas de clase, diagramas de entidad relación, diagrama de tablas, base de datos, programación estructurada modular y POO. Todo dentro del lenguaje de alto nivel Java, se presenta el siguiente manual técnico.

2 OBJETIVOS

2.1 OBJETIVO GENERAL

*Este documento tiene como objetivo y propósito principal presentar el proceso de desarrollo del sistema **epic store** . Al mismo tiempo dar referencias de como interactuar con el programa para futuras actualizaciones o darle un mantenimiento adecuado en caso de errores o caídas.*

2.2 OBJETIVOS ESPECIFICOS

1. *Mostrar el código fuente para una posible mejora en el sistema epic store.*
2. *Requisitos para una satisfactoria ejecución del programa.*
3. *Manifestar evidencias del diseño del sistema antes de implementarlo (Pseudocódigo, estados, etc.).*

3 ALCANCE

Este documento está especialmente diseñado para ser interpretado por estudiantes de programación o programadores, aunque también puede ser interpretado por personas que tengan conocimientos básicos en programación: Algoritmos, pseudocódigo, base de datos, NetBeans, Java, etc.

4 REQUERIMIENTOS TECNICOS

4.1 HARDWARE

4.1.1 REQUISITOS DEL SISTEMA

Mínimo: *Procesador de 500 MhZ, 96 MB de RAM, tarjeta de video de 16 MB, Windows XP o versiones posteriores.*

Recomendado: *Procesador de 800 MhZ, 128 MB de RAM, tarjeta de video de 32 MB, Windows XP o versiones posteriores.*

4.2 SOFTWARE

4.2.1 TECNOLOGIAS IMPLEMENTADAS

Java JDK: *Java™ Development Kit (JDK) es un software para los desarrolladores de Java. Incluye el intérprete Java, clases Java y herramientas de desarrollo Java (JDT): compilador, depurador, desensamblador, visor de applets, generador de archivos de apéndice y generador de documentación. El JDK le permite escribir aplicaciones que se desarrollan una sola vez y se ejecutan en cualquier lugar de cualquier máquina virtual Java. Las aplicaciones Java desarrolladas con el JDK en un sistema se pueden usar en otro sistema sin tener que cambiar ni recompilar el código. Los archivos de clase Java son portables a cualquier máquina virtual Java estándar.*

Recomiendo tener instalado en el ordenador la versión 21.0.4 de Java JDK.

Apache Maven: *Apache Maven es una herramienta que estandariza la configuración de un proyecto en todo su ciclo de vida, como por ejemplo en todas las fases de compilación y empaquetado y la instalación de mecanismos de distribución de librerías, para que puedan ser utilizadas por otros desarrolladores y equipos de desarrollo.*

También contempla temas relacionados con la integración continua, para poder realizar la ejecución de test unitarios y pruebas automatizadas, test de integración, etc.

Recomiendo tener instalado en el ordenador la versión 3.9.6 de Apache Maven.

Apache NetBeans: *NetBeans es un IDE o entorno de desarrollo integrado, basado en el lenguaje Java y ejecutado en Swing.*

De esta forma, NetBeans o Apache NetBeans es una aplicación de código abierto, que ha cobrado bastante popularidad en los últimos años.

Este IDE, orientado principalmente a las apps de Java, ofrece diferentes herramientas digitales como editor de texto, código, compilador, interfaz gráfica de usuario; además de un depurador.

Por otro lado, cabe destacar que NetBeans facilita la creación de aplicaciones estructuradas, ya que están basadas en un conjunto de módulos. Así, se favorece el desarrollo de las diversas funciones de una manera independiente y pudiendo también reutilizar los componentes.

Recomiendo tener en el ordenador la versión 21 de Apache NetBeans.

***MySQL:** MySQL es, como su nombre indica, un sistema de gestión de bases de datos relacionales o SGBD basado en SQL. En la actualidad, este **software de código abierto** forma parte de Oracle, la empresa que también desarrolló el lenguaje de programación Java.*

*MySQL almacena, gestiona y muestra datos en tablas. Funciona como un **sistema cliente-servidor**. Mientras que la base de datos actúa como un servidor en el que se almacena toda la información relevante, el software puede verse como un cliente. Con la ayuda del software, los usuarios de la base de datos relacional pueden formular diversas consultas, denominadas "**queries**", en el lenguaje de consulta SQL y enviarlas al sistema de base de datos. Estos son procesados por MySQL, por lo que el acceso a los datos es también una parte importante de MySQL.*

***Windows 10:** Windows 10 es uno de los sistemas operativos más utilizados en todo el mundo y su lanzamiento se produjo allá por julio de 2015. Los sistemas operativos como Windows se encargan de gestionar el hardware de nuestro ordenador para poder ejecutar aplicaciones y programas, ya sean propias o desarrolladas por terceros.*

Actualmente, Windows 10 goza de mucha popularidad entre los usuarios de PC, incluso con la reciente salida de Windows 11 al mercado. Su gran rango

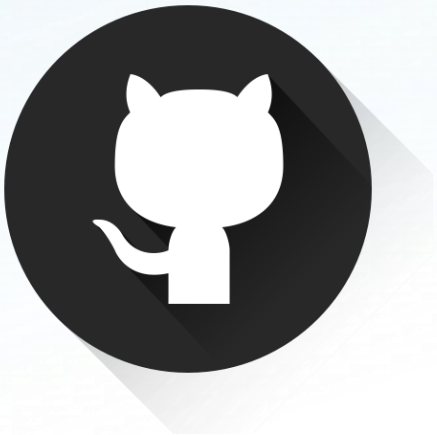
de adaptación y optimización lo hacen una gran opción incluso para ordenadores más austeros.

Para este proyecto recomiendo versiones de Windows posteriores a Windows 7, aunque queda a disposición del desarrollador escoger el OS que más le convenga.

Git: *es un sistema de control de versiones de código fuente. Es una herramienta que se utiliza para llevar un registro de los cambios en el código fuente de un proyecto y permitir a varios desarrolladores trabajar en el mismo proyecto de manera simultánea.*

Con Git, es posible crear un repositorio que contenga el código fuente de un proyecto y llevar un registro de todos los cambios realizados en el repositorio. Los desarrolladores pueden colaborar en el mismo proyecto al hacer «commits» (guardar cambios) en el repositorio y luego «pushear» (enviar) estos cambios a un servidor centralizado para que puedan ser utilizados por otros desarrolladores.

Git es una herramienta muy popular en la comunidad de desarrollo de software debido a su flexibilidad, velocidad y capacidad para manejar grandes proyectos. También es ampliamente utilizado en proyectos de código abierto, ya que permite a cualquier persona contribuir al proyecto y hacer seguimiento de los cambios realizados en el código fuente.



GitHub: *es una plataforma de desarrollo colaborativo que aloja proyectos en la nube utilizando el sistema de control de versiones llamado Git. Ayuda a los desarrolladores a almacenar y administrar el código llevando un registro de cambios. Generalmente el código es abierto, lo que permite realizar proyectos compartidos y mantener el seguimiento detallado de su progreso. La plataforma GitHub también funciona como red social conectando a los desarrolladores con los usuarios. Como usuario puedes descargar programas o aplicaciones, y de la misma manera puedes aportar a su desarrollo ofreciendo mejoras y discutir las cuestiones que te interesan en foros temáticos.*

5 Fases de desarrollo para sistema “Epic store de videojuegos”

Fase 0 — Base técnica

Objetivo: dejar el proyecto “bien armado” para crecer sin caos.

- *Estructura de paquetes (controllers / services / models / dtos / util)*
- *Convención JSON (siempre application/json, UTF-8)*
- *Manejo estándar de errores (JSON con mensaje, detalle, codigo)*
- *Endpoint base /api/health*

Entregable: plantilla lista + prueba de BD

Fase 1 — Catálogos y tienda pública

*Objetivo: que cualquiera pueda **ver** el catálogo (sin comprar aún).*
Endpoints sugeridos

- *GET /api/categorias*
- *GET /api/clasificaciones*
- *GET /api/empresas (listado/perfil)*
- *GET /api/videojuegos (listar + filtros: título, categoría, precio, empresa)*
- *GET /api/videojuegos?id=*
- *GET /api/banner (destacados activos)*

Tablas clave: CATEGORIA, CLASIFICACION_EDAD, EMPRESA, VIDEOJUEGO, VIDEOJUEGO_CATEGORIA, BANNER_DESTACADO

Entregable: Insomnia puede consultar todo el catálogo con filtros.

Fase 2 — Autenticación y roles

Objetivo: controlar permisos (admin / común / empresa).
Endpoints sugeridos

- *POST /api/auth/login (USUARIO)*
- *POST /api/auth/login-empresa (USUARIO_EMPRESA)*
- *GET /api/auth/me (datos del usuario logueado)*

Reglas:

- *USUARIO.tipo_usuario para ADMIN/COMUN*

- *usuario empresa siempre asociado a id_empresa*

Entregable: login funcional (aunque sea con token simple o sesión).

Fase 3 — Cartera digital (saldo y recargas)

Objetivo: cumplir cartera + dejar listo el flujo de compra.
Endpoints sugeridos

- *GET /api/cartera/saldo*
- *POST /api/cartera/recarga (monto + descripción)*
- *GET /api/cartera/movimientos*

Tablas: USUARIO, MOVIMIENTO_SALDO

Entregable: recargar y ver historial en JSON.

Fase 4 — Compras y biblioteca (núcleo del negocio)

Objetivo: comprar juegos y que queden en biblioteca correctamente.
Endpoints sugeridos

- *POST /api/compras*
  Body: idUsuario, idVideojuego, fechaCompra (manual)
  Reglas: saldo suficiente, venta_activa, edad mínima, calcular comisión, transacción.
- *GET /api/usuarios/biblioteca (propios + prestados)*
- *POST /api/biblioteca/estado (INSTALADO/NO_INSTALADO)*

Tablas: COMPRA, USUARIO, MOVIMIENTO_SALDO, BIBLIOTECA_JUEGO, VIDEOJUEGO, CLASIFICACION_EDAD

Entregable: compra transaccional + biblioteca.

Fase 5 — Comentarios y calificaciones con hilos

Objetivo: interacción social con reglas estrictas.
Endpoints sugeridos

- *GET /api/comentarios?idVideojuego=*
- *POST /api/comentarios (solo si compró)*
- *POST /api/comentarios/responder (id_comentario_padre)*
- *(Opcional) PUT /api/comentarios/visibilidad (moderación por comentario)*

Reglas:

- *solo quien compró puede comentar/calificar*
- *empresa puede ocultar comentarios (texto), pero calificación debe seguir visible*

Tablas: COMENTARIO, validación con COMPRA o BIBLIOTECA_JUEGO

Fase 6 — Grupos familiares y préstamos

Objetivo: biblioteca familiar con restricciones.
Endpoints sugeridos

- *POST /api/grupos (crear)*
- *POST /api/grupos/agregar-miembro*
- *GET /api/grupos?idGrupo=*
- *POST /api/grupos/prestar (registrar "prestado" en biblioteca del usuario)*
- *Validación fuerte: max 6 miembros, y max 1 prestado INSTALADO por usuario*

Tablas: GRUPO_FAMILIAR, MIEMBRO_GRUPO, BIBLIOTECA_JUEGO

Fase 7 — Gestión de empresa (catálogo + visibilidad)

Objetivo: que las empresas administren sus juegos.
Endpoints sugeridos

- *POST /api/empresa/videojuegos (crear)*
- *PUT /api/empresa/videojuegos (editar)*
- *PUT /api/empresa/videojuegos/venta (activar/desactivar)*
- *PUT /api/empresa/comentarios-visibles (global y por juego)*
- *POST /api/empresa/usuarios (crear usuarios internos)*

Tablas: EMPRESA, USUARIO_EMPRESA, VIDEOJUEGO, VIDEOJUEGO_CATEGORIA

Fase 8 — Admin (comisiones, categorías, banner)

Objetivo: control del sistema y monetización.
Endpoints sugeridos

- *POST /api/admin/categorias / PUT / DELETE*
- *POST /api/admin/banner / PUT / DELETE*
- *POST /api/admin/comision-global*
- *POST /api/admin/comision-empresa*

Reglas:

- *comisión empresa <= global vigente*
- *al cambiar global, ajustar específicas que queden arriba*

Tablas: CATEGORIA, BANNER_DESTACADO, COMISION_GLOBAL, COMISION_EMPRESA

Fase 9 — Reportes (JSON primero, luego PDF)

Objetivo: sacar reportes del enunciado sin complicarse.

1. *Primero endpoints que devuelven JSON (más rápido de validar)*
2. *Luego exportación a PDF (JasperReports)*

5.1 FASE 0

5.1.1 Endpoint de salud /api/health

*En **Source Packages** creamos:*

- *com.epicstore.epicstore.controllers*
- *com.epicstore.epicstore.services*
- *com.epicstore.epicstore.models*
- *com.epicstore.epicstore.dtos*

DTO: HealthDTO.java

***Paquete:** com.epicstore.epicstore.dtos.HealthDTO*

Model: HealthModel.java

***Paquete:** com.epicstore.epicstore.models.HealthModel*

Service: HealthService.java

***Paquete:** com.epicstore.epicstore.services.HealthService*

Controller (Servlet): HealthController.java

***Paquete:** com.epicstore.epicstore.controllers.HealthController*

5.2 Arranque formal — Fase 1: Catálogos y Tienda Pública

Objetivo de la fase

*Que **cualquier usuario** (sin login) pueda:*

- *Ver categorías*
- *Ver videojuegos*
- *Ver detalles de un videojuego*

DTO: CategoriaDTO.java

Paquete: com.epicstore.epicstore.dtos

Model: CategoriaModel.java

Paquete: *com.epicstore.epicstore.models*

Service: CategoriaService.java

Paquete: *com.epicstore.epicstore.services*

Controller: CategoriaController.java

Paquete: *com.epicstore.epicstore.controllers*

5.2.1 Siguiente paso (Fase 1): GET /api/videojuegos

Ahora vamos con el ***catálogo de videojuegos*** *(la tienda pública), que es el corazón del sistema.*

DTO: VideojuegoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: VideojuegoModel.java

Paquete: *com.epicstore.epicstore.models*

Service: VideojuegoService.java

Paquete: *com.epicstore.epicstore.services*

Controller: VideojuegoController.java

Paquete: *com.epicstore.epicstore.controllers*

5.2.2 Siguiente paso lógico (Fase 1, último endpoint)

GET /api/videojuegos?id=

Este endpoint es clave porque:

- *Lo necesita el frontend para la* ***pantalla de detalle***
- *Nos obliga a:*
    - *traer categorías del juego*
    - *validar si existe*
    - *preparar la base para comentarios*

DTO: VideojuegoDetalleDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: VideojuegoModel.java

Paquete: *com.epicstore.epicstore.models*

Service: VideojuegoService.java

Paquete: *com.epicstore.epicstore.services*

Controller: VideojuegoController.java

Paquete: *com.epicstore.epicstore.controllers*

5.3 Fase 2 — Autenticación y roles

Nuestra DB ya trae USUARIO.password y USUARIO_EMPRESA.password, así que creamos endpoints:

- *POST /api/auth/login (login usuario normal/admin)*
- *POST /api/auth/login-empresa (login de usuario de empresa)*
- *GET /api/auth/me (quién está logueado)*
- *POST /api/auth/logout*

Esto usa ***cookies de sesión*** *(JSESSIONID).*

DTOs:

LoginUsuarioDTO.java

LoginEmpresaDTO.java

SesionDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: AuthModel.java

Paquete: *com.epicstore.epicstore.models*

Service: AuthService.java

Paquete: *com.epicstore.epicstore.services*

AuthController.java (usuario normal/admin)

AuthEmpresaController.java

AuthMeController.java

AuthLogoutController.java

Paquete: *com.epicstore.epicstore.controllers*

5.3.1 Registro de usuario común

Registro y perfil:

DTO: RegistroUsuarioDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: UsuarioModel.java

Paquete: *com.epicstore.epicstore.models*

Service: UsuarioService.java

Paquete: *com.epicstore.epicstore.services*

UsuarioRegistroController.java

Paquete: *com.epicstore.epicstore.controllers*

5.3.2 Perfil público de usuario común

DTOs: PerfilUsuarioDTO.java + JuegoPerfilDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: PerfilUsuarioModel.java

Paquete: *com.epicstore.epicstore.models*

Controller: PerfilUsuarioController.java

Paquete: *com.epicstore.epicstore.controllers*

5.3.3 Poder buscar usuario por lista

DTO: UsuarioBusquedaDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: UsuarioModel.java

Paquete: *com.epicstore.epicstore.models*

Service: UsuarioService.java

Paquete: *com.epicstore.epicstore.services*

Controller: UsuarioBuscarController.java

Paquete: *com.epicstore.epicstore.controllers*

5.3.4 AuthFilter

Con esto basta para el backend *y ya garantiza que:*

- *Cada rol solo puede pegarle a lo que le corresponde*
- *Si en Angular alguien intenta "entrar a una pestaña" y el frontend llama un endpoint protegido, el backend responde* ***401/403***

Creamos: AuthFilter.java

Paquete: *com.epicstore.epicstore.filters*

6 Fase 3 — Cartera digital (saldo y recargas)

6.1.1 Paso 1 — Historial de movimientos

Qué devuelve

- *Saldo actual*
- *Lista de movimientos ordenados por fecha*
- *Tipo: RECARGA / COMPRA*
- *Monto (+ o -)*
- *Fecha*
- *Descripción*

DTO: MovimientoSaldoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: CarteraModel.java

Paquete: *com.epicstore.epicstore.models*

Service: CarteraService.java

Paquete: *com.epicstore.epicstore.services*

Controller: CarteraMovimientosController.java

Paquete: *com.epicstore.epicstore.controllers*

6.1.2 Siguiente paso natural (para cerrar Cartera)

Implementamos:

POST /api/cartera/recarga

- *valida monto > 0*
- *inserta en MOVIMIENTO_SALDO*
- *actualiza USUARIO.saldo_actual*
- *transacción (como en compras)*

DTO: RecargaSaldoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: RecargaModel.java

Paquete: *com.epicstore.epicstore.models*

Service: RecargaService.java

Paquete: *com.epicstore.epicstore.services*

Controller: CarteraRecargaController.java

Paquete: *com.epicstore.epicstore.controllers*

6.2 Fase 4 — Compras y biblioteca (núcleo del negocio)

Ahora sí vamos con el endpoint más importante del sistema:

Reglas que se especifican (según requerimientos)

1. *El videojuego debe estar **en venta** (venta_activa = 'S')*
2. *El usuario debe tener **saldo suficiente** (USUARIO.saldo_actual)*
3. *Validar **edad mínima** comparando fecha nacimiento vs CLASIFICACION_EDAD.edad_minima*
4. *Registrar la compra con **fecha manual** (fecha_compra viene del request)*
5. *Calcular comisión y registrar:*
    - *COMPRA*
    - *MOVIMIENTO_SALDO (tipo COMPRA, monto negativo)*
    - *USUARIO.saldo_actual actualizado*
    - *BIBLIOTECA_JUEGO como PROPIO*
6. *Todo debe ser **transaccional** (si algo falla, no se guarda nada)*

CrearCompraDTO.java

Paquete: *com.epicstore.epicstore.dtos*

DTO: CompraResultadoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: CompraModel.java

Paquete: *com.epicstore.epicstore.models*

*Este model hace **toda la transacción** con commit/rollback*

Service: CompraService.java

Paquete: *com.epicstore.epicstore.services*

Controller: CompraController.java

Paquete: *com.epicstore.epicstore.controllers*

6.2.1 Siguiente paso natural (flujo lógico del sistema)

*Después de comprar, el usuario siempre quiere ver su **biblioteca:***

Qué debe devolver

- *Juegos **propios***
- *Juegos **prestados***
- *Estado de instalación*
- *Si es prestado: de qué grupo viene*

DTO: BibliotecaItemDTO.java

***Paquete:** com.epicstore.epicstore.dtos*

Model: BibliotecaModel.java

***Paquete:** com.epicstore.epicstore.models*

Service: BibliotecaService.java

***Paquete:** com.epicstore.epicstore.services*

Controller: BibliotecaController.java

***Paquete:** com.epicstore.epicstore.controllers*

6.2.2 Siguiente paso natural (para completar biblioteca)

*Después de listar, toca **actualizar estado de instalación** cumpliendo la regla:*

*un usuario solo puede tener **1 juego PRESTADO** en INSTALADO a la vez, los PROPIO no cuentan.*

DTO: ActualizarEstadoBibliotecaDTO.java

***Paquete:** com.epicstore.epicstore.dtos*

Model: BibliotecaEstadoModel.java

***Paquete:** com.epicstore.epicstore.models*

Service: BibliotecaEstadoService.java

***Paquete:** com.epicstore.epicstore.services*

Controller: BibliotecaEstadoController.java

***Paquete:** com.epicstore.epicstore.controllers*

6.3 Fase 5 — Comentarios y calificaciones con hilos

Este endpoint le sirve directo a la pantalla de detalle para mostrar:

- *hilos (idComentarioPadre)*
- *calificación (1–5)*
- *texto (solo si debe mostrarse)*
- *autor (nickname)*
- *fecha*

Y respeta la regla del proyecto:

- *si comentarios están deshabilitados (por empresa/juego)* ***se oculta el texto****, pero* ***se mantiene la calificación****.*

DTO: ComentarioDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: ComentarioModel.java

Paquete: *com.epicstore.epicstore.models*

Service: ComentarioService.java

Paquete: *com.epicstore.epicstore.services*

Controller: ComentarioController.java

Paquete: *com.epicstore.epicstore.controllers*

El siguiente paso natural (para cerrar comentarios completo) es:

POST /api/comentarios (crear comentario)
con validación ***"solo si compró"*** *usando COMPRA.*

DTO: CrearComentarioDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: ComentarioModel.java

Paquete: *com.epicstore.epicstore.models*

Service: ComentarioService.java

Paquete: *com.epicstore.epicstore.services*

Controller: ComentarioController.java (agregamos doPost)

Paquete: *com.epicstore.epicstore.controllers*

6.3.1 Visibilidad de comentarios (global y por juego)

Vamos por ***2 endpoints****:*

1. ***Global por cuenta empresa***
2. ***Específico por videojuego***

DTOs:
CambiarVisibilidadComentariosGlobalDTO.java

CambiarVisibilidadComentariosJuegoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: VideojuegoEmpresaModel

Paquete: *com.epicstore.epicstore.models*

Service: VideojuegoEmpresaService.java

Paquete: *com.epicstore.epicstore.services*

Controllers:

EmpresaComentariosGlobalController.java

EmpresaVideojuegosComentariosController.java

Paquete: *com.epicstore.epicstore.controllers*

6.4 Fase 6 — Grupos familiares y préstamos

En grupos familiares podemos:

1. ***Crear grupo familiar***
2. ***Agregar miembro al grupo***
3. ***Ver grupos del usuario / ver miembros***
4. ***Prestar un videojuego***
5. ***Quitar préstamo / expulsar miembro / eliminar grupo***

6.4.1 Crear grupo familiar

DTO: CrearGrupoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: GrupoModel.java

Paquete: *com.epicstore.epicstore.models*

Service: GrupoService.java

Paquete: *com.epicstore.epicstore.services*

Controller: GrupoController.java

Paquete: *com.epicstore.epicstore.controllers*

6.4.2 Agregar miembro al grupo

DTO: AgregarMiembroGrupoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: MiembroGrupoModel.java

Paquete: *com.epicstore.epicstore.models*

Service: MiembroGrupoService.java

Paquete: *com.epicstore.epicstore.services*

Controller: MiembroGrupoController.java

Paquete: *com.epicstore.epicstore.controllers*

6.4.3 Préstamo de videojuego

Objetivo

El dueño de un grupo presta un videojuego a un miembro. Esto se refleja creando una entrada en BIBLIOTECA_JUEGO del receptor como:

- *tipo_propiedad = 'PRESTADO'*
- *id_grupo_origen = (idGrupo)*
- *estado_instalacion = 'NO_INSTALADO'*
- *fecha_ultimo_cambio_estado = NOW()*

Reglas a cumplir según los requerimientos

1. *El grupo existe*
2. *El solicitante es **dueño (DUENIO)** del grupo*
3. *El receptor es **miembro** del grupo (rol MIEMBRO o DUENIO no aplica para receptor, lo controlamos)*
4. *El dueño debe tener el videojuego en su biblioteca como **PROPIO***
5. *El receptor **no debe tener ya el juego** (ni propio ni prestado)*
6. *El dueño **no puede prestarse a sí mismo***
7. *Se crea el préstamo (insert en BIBLIOTECA_JUEGO del receptor)*

DTO: CrearPrestamoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: PrestamoModel.java

Paquete: *com.epicstore.epicstore.models*

Service: PrestamoService.java

Paquete: *com.epicstore.epicstore.services*

Controller: PrestamoController.java

Paquete: *com.epicstore.epicstore.controllers*

6.4.4 Listar Grupos y Miembros

Esto le permite a Angular:

- *Mostrar **mis grupos***
- *Saber si soy **DUENIO o MIEMBRO***
- *Mostrar **miembros del grupo***

- *Desde ahí disparar **préstamos***

DTO: GrupoResumenDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: GrupoModel.java

Paquete: com.epicstore.epicstore.models

Service: GrupoService.java

Paquete: com.epicstore.epicstore.services

Controller: GrupoController.java

Paquete: com.epicstore.epicstore.controllers

6.4.5 Vamos con Listar miembros del grupo.

Endpoint

GET /api/grupos/miembros?idGrupo=1

Devuelve:

- ***idUsuario***
- ***nickname***
- ***rol (DUENIO / MIEMBRO)***

DTO: MiembroGrupoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: MiembroGrupoModel.java

Paquete: *com.epicstore.epicstore.models*

Service: MiembroGrupoService.java

Paquete: com.epicstore.epicstore.services

Controller: MiembroGrupoListarController.java

Paquete: *com.epicstore.epicstore.controllers*

6.4.6 Endpoint: POST /api/prestamos/devolver

Qué hace

- *Verifica que **exista** el préstamo en la biblioteca del receptor como PRESTADO*
- *Verifica que ese préstamo pertenezca al idGrupo (por id_grupo_origen)*
- *Verifica que el solicitante sea **dueño del grupo***
- *Si el juego prestado está **INSTALADO**, lo pasamos a NO_INSTALADO antes de devolver (para no dejar estados raros)*

- *Elimina el registro BIBLIOTECA_JUEGO del receptor (solo el PRESTADO)*

DTO: DevolverPrestamoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: DevolucionPrestamoModel.java

Paquete: *com.epicstore.epicstore.models*

Service: DevolucionPrestamoService.java

Paquete: *com.epicstore.epicstore.services*

Controller: DevolucionPrestamoController.java

Paquete: *com.epicstore.epicstore.controllers*

6.4.7 Eliminar miembro de grupo familiar

Transacciones*, validando dueño, y limpiando* ***préstamos activos*** *antes de borrar para evitar inconsistencias / errores de FK.*

Importante: en el modelo de nuestra DB, el "dueño" está en MIEMBRO_GRUPO.rol='DUENIO' (no en GRUPO_FAMILIAR). Así que todo se valida por ahí.

Endpoint

DELETE /api/grupos/miembros

DTO: EliminarMiembroDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: GrupoAdminModel.java

Paquete: *com.epicstore.epicstore.models*

Service: GrupoAdminService.java

Paquete: *com.epicstore.epicstore.services*

Controller: MiembroGrupoController

Paquete: *com.epicstore.epicstore.controllers*

6.4.8 Eliminar grupo familiar

DTO: EliminarGrupoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Controller: GrupoController.java

Paquete: com.epicstore.epicstore.controllers

6.5 Fase 7 — Gestión de empresa (catálogo + visibilidad)

Existe la base perfecta para arrancar "Usuario Empresa"*:*

*Se tiene **AuthEmpresaController** en **/api/auth/login-empresa**, **AuthMeController** ya soporta sesiones USUARIO y EMPRESA, **AuthLogoutController** ya cierra sesión para ambos y la BD ya tiene EMPRESA, USUARIO_EMPRESA, VIDEOJUEGO(venta_activa, comentarios_visibles), EMPRESA(comentarios_visibles_global)*

6.5.1 Crear o eliminar un usuario de empresa

DTOs:

CrearUsuarioEmpresaDTO.java

UsuarioEmpresaDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: UsuarioEmpresaModel.java

Paquete: *com.epicstore.epicstore.models*

Services: UsuarioEmpresaService.java

Paquete: *com.epicstore.epicstore.services*

Controller: EmpresaUsuariosController.java

Paquete: *com.epicstore.epicstore.controllers*

6.5.2 El paso natural ahora es el CATÁLOGO DE LA EMPRESA:

Módulo siguiente

1. ***Listar mis videojuegos***
2. ***Publicar videojuego***
3. ***Editar videojuego***
4. ***Activar / desactivar venta***
5. ***Ver ventas propias***

DTO: VideojuegoEmpresaDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: VideojuegoEmpresaModel.java

Paquete: *com.epicstore.epicstore.models*

Services: VideojuegoEmpresaService.java

Paquete: *com.epicstore.epicstore.services*

Controller: EmpresaVideojuegosController.java

Paquete: *com.epicstore.epicstore.controllers*

6.5.3 Siguiente paso PUBLICAR VIDEOJUEGO (Empresa)

Ahora lo natural es:

1. ***Publicar videojuego*** *(POST /api/empresa/videojuegos)*
2. ***Editar videojuego*** *(PUT /api/empresa/videojuegos?id=)*
3. ***Activar/Desactivar venta*** *(PUT /api/empresa/videojuegos/venta)*

DTO: PublicarVideojuegoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: VideojuegoEmpresaModel.java

Paquete: *com.epicstore.epicstore.models*

Services: VideojuegoEmpresaService.java

Paquete: *com.epicstore.epicstore.services*

Controller: EmpresaVideojuegosController.java

Paquete: *com.epicstore.epicstore.controllers*

*6.5.4 **Editar videojuego** es el siguiente paso natural*

Ejemplo:
PUT http://localhost:8080/EpicStore-1.0-SNAPSHOT/api/empresa/videojuegos?id=12

1. *Solo si tipoSesion = "EMPRESA"*
2. *Solo puede editar **videojuegos de SU empresa***
3. *Validar campos básicos:*

- *titulo no vacío*
- *descripcion no vacía*
- *precio > 0*
- *idClasificacion válida*

4. ***Evitar duplicado de título*** *dentro de la misma empresa (excluyendo el mismo juego)*

DTO: EditarVideojuegoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: VideojuegoEmpresaModel.java

Paquete: *com.epicstore.epicstore.models*

Services: VideojuegoEmpresaService.java

Paquete: *com.epicstore.epicstore.services*

Controller: EmpresaVideojuegosController.java

Paquete: *com.epicstore.epicstore.controllers*

6.5.5 ACTIVAR / DESACTIVAR VENTA (Empresa)

*Esto cumple el requerimiento de **"suspender venta"** sin borrar el juego ni afectar bibliotecas/compras anteriores*

DTO: CambiarVentaVideojuegoDTO.java

***Paquete:** com.epicstore.epicstore.dtos*

Model: VideojuegoEmpresaModel.java

***Paquete:** com.epicstore.epicstore.models*

Services: VideojuegoEmpresaService.java

***Paquete:** com.epicstore.epicstore.services*

Controller: EmpresaVideojuegoVentaController.java

***Paquete:** com.epicstore.epicstore.controllers*

Perfil público con catálogo de creaciones

(1 endpoint público que lista empresa + videojuegos de esa empresa)

DTOs:

EmpresaPublicaDTO.java

VideojuegoPublicoDTO.java

***Paquete:** com.epicstore.epicstore.dtos*

Model: EmpresaPublicaModel.java

***Paquete:** com.epicstore.epicstore.models*

Services: VideojuegoEmpresaService.java

***Paquete:** com.epicstore.epicstore.services*

Controller: EmpresaPerfilPublicoController.java

***Paquete:** com.epicstore.epicstore.controllers*

6.6 Fase 9 — Reportes (JSON primero, luego PDF)

6.6.1 Empresa — Reporte de Ventas Propias (JSON)

Endpoint

GET /api/empresa/reportes/ventas-propias?desde=YYYY-MM-DD&hasta=YYYY-MM-DD

DTO: ReporteVentaPropiaDTO.java

***Paquete:** com.epicstore.epicstore.dtos*

Model: ReportesEmpresaModel.java

***Paquete:** com.epicstore.epicstore.models*

Services: ReportesEmpresaService.java

Paquete: *com.epicstore.epicstore.services*

Controller: EmpresaReporteVentasPropiasController.java

Paquete: *com.epicstore.epicstore.controllers*

6.6.2 EMPRESA — Reporte de Feedback (3 endpoints)

6.6.2.1 Calificación promedio por juego

DTO: ReporteCalificacionJuegoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: ReportesEmpresaModel.java

Paquete: *com.epicstore.epicstore.models*

Services: ReportesEmpresaService.java

Paquete: *com.epicstore.epicstore.services*

Controller: EmpresaReporteFeedbackCalificacionesController.java

Paquete: *com.epicstore.epicstore.controllers*

6.6.2.2 Mejores comentarios (más respuestas)

DTO: ReporteMejorComentarioDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: ReportesEmpresaModel.java

Paquete: *com.epicstore.epicstore.models*

Services: ReportesEmpresaService.java

Paquete: *com.epicstore.epicstore.services*

Controller: EmpresaReporteFeedbackMejoresComentariosController.java

Paquete: *com.epicstore.epicstore.controllers*

6.6.2.3 Peores calificaciones

DTO: ReportePeorCalificacionDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: ReportesEmpresaModel.java

Paquete: *com.epicstore.epicstore.models*

Services: ReportesEmpresaService.java

Paquete: *com.epicstore.epicstore.services*

Controller: EmpresaReporteFeedbackPeoresCalificacionesController.java

Paquete: *com.epicstore.epicstore.controllers*

6.6.2.4 Top 5 juegos más vendidos

DTO: ReporteTopJuegoEmpresaDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: ReportesEmpresaModel.java

Paquete: *com.epicstore.epicstore.models*

Services: ReportesEmpresaService.java

Paquete: *com.epicstore.epicstore.services*

Controller: EmpresaReporteTop5JuegosController.java

Paquete: *com.epicstore.epicstore.controllers*

6.6.3 Usuario común — Historial de Gastos (JSON)

Endpoint

GET /api/reportes/historial-gastos?desde=YYYY-MM-DD&hasta=YYYY-MM-DD

- *Requiere sesión usuario común (tipoSesion=USUARIO)*
- *Si no hay intervalo: histórico*

DTO: HistorialGastoDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: ReportesUsuarioModel.java

Paquete: *com.epicstore.epicstore.models*

Services: ReportesUsuarioService.java

Paquete: *com.epicstore.epicstore.services*

Controller: UsuarioReporteHistorialGastosController.java

Paquete: *com.epicstore.epicstore.controllers*

6.6.4 Reporte: Análisis de Biblioteca (Usuario)

6.6.4.1 Comparación de valoraciones (personal vs comunidad) y categorías favoritas

1) Endpoints (JSON)

A) Valoración personal vs comunidad

GET /api/reportes/analisis-biblioteca/valoraciones

Devuelve por cada juego en la biblioteca:

- *promedio comunidad*
- *última valoración personal del usuario (si existe)*

B) Categorías favoritas

GET /api/reportes/analisis-biblioteca/categorias-favoritas?limit=10

Devuelve:

- *categoría*
- *total compras*

DTOs:
ReporteValoracionBibliotecaDTO.java

ReporteCategoriaFavoritaDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: ReportesUsuarioModel.java

Paquete: *com.epicstore.epicstore.models*

Services: ReportesUsuarioService.java

Paquete: *com.epicstore.epicstore.services*

Controllers:

UsuarioReporteAnalisisValoracionesController.java

UsuarioReporteAnalisisCategoriasFavoritasController.java

Paquete: *com.epicstore.epicstore.controllers*

6.6.4.2 Uso Biblioteca Familiar (Préstamos)

Endpoint (JSON)

GET /api/reportes/biblioteca-familiar/uso

- ***Requiere sesión usuario (tipoSesion=USUARIO)***
- ***Devuelve solo juegos PRESTADOS del usuario, con:***
    - ***estado instalación + fecha último cambio***
    - ***tiempo instalado simulado***
    - ***valoración comunidad (AVG comentarios visibles)***

DTO: ReporteUsoBibliotecaFamiliarDTO.java

Paquete: *com.epicstore.epicstore.dtos*

Model: ReportesUsuarioModel.java

Paquete: *com.epicstore.epicstore.models*

Services: ReportesUsuarioService.java

Paquete: *com.epicstore.epicstore.services*

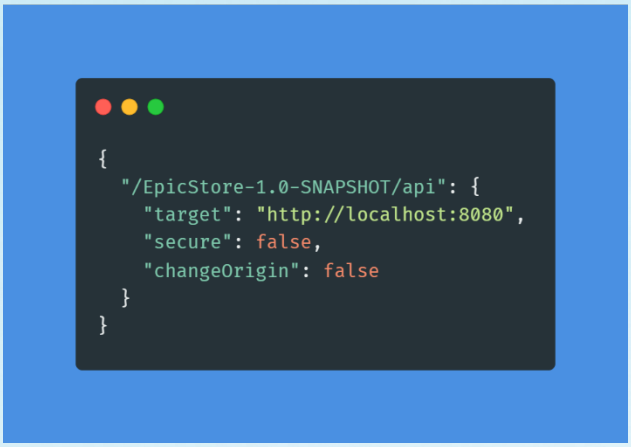
Controllers: UsuarioReporteUsoBibliotecaFamiliarController.java

Paquete: *com.epicstore.epicstore.controllers*

6.7 saltamos a Angular

6.7.1 Configurar el proxy para Tomcat (lo clave)

*Creamos este archivo en la **raíz del proyecto**:*

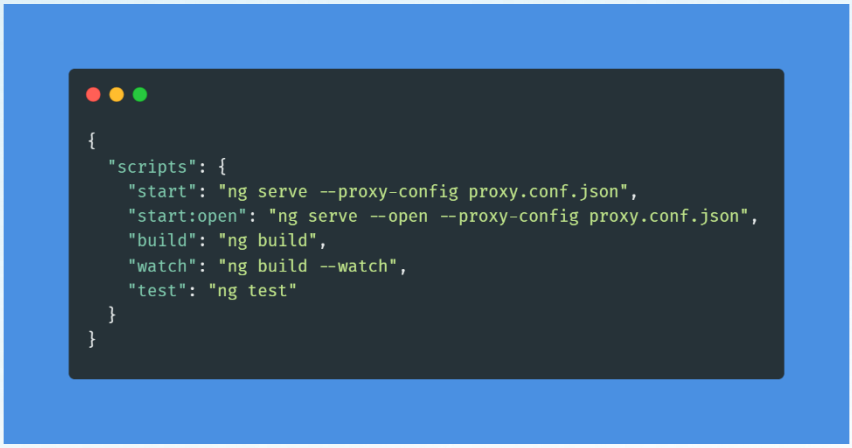


```
{
  "/EpicStore-1.0-SNAPSHOT/api": {
    "target": "http://localhost:8080",
    "secure": false,
    "changeOrigin": false
  }
}
```

Con esto, el frontend llama a:
/EpicStore-1.0-SNAPSHOT/api/...
y el navegador maneja la sesión como si fuera el mismo origen, evitando el infierno CORS.

6.7.2 Ajustamos el script de arranque (para no olvidar el proxy)

En package.json:



```
{
  "scripts": {
    "start": "ng serve --proxy-config proxy.conf.json",
    "start:open": "ng serve --open --proxy-config proxy.conf.json",
    "build": "ng build",
    "watch": "ng build --watch",
    "test": "ng test"
  }
}
```

6.7.3 Definimos el API_BASE (para que todo sea consistente)

Creamos:

src/app/core/config/api.config.ts

```
export const API_BASE = '/EpicStore-1.0-SNAPSHOT/api';
```

(Esto evita “strings mágicos” por todo el proyecto.

6.7.4 Prueba rápida (smoke test)

Arrancamos Angular y probamos en el navegador:

- *Front: http://localhost:4200/*
- *Backend health: http://localhost:4200/EpicStore-1.0-SNAPSHOT/api/health*

Si health responde 200 → proxy perfecto.

6.7.5 Asegurar provideHttpClient() en app.config.ts

En src/app/app.config.ts agregamos provideHttpClient() dentro de providers

```
import { ApplicationConfig } from '@angular/core';
import { provideRouter } from '@angular/router';
import { provideHttpClient } from
import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(routes),
    provideHttpClient()
  ]
};
```

CONCLUSIONES

Este manual proporciona la documentación necesaria para que un desarrollador pueda instalar, compilar, ejecutar y mantener el sistema epic store. La arquitectura modular facilita la ***extensión y evolución del sistema****, asegurando un ciclo de vida de software ordenado y profesional.*

DESPEDIDA

ESPERO QUE ESTE MANUAL SEA DE UTILIDAD PARA QUE PUEDAS COMPRENDER DE UNA MEJOR MANERA EL DESARROLLO DEL PROYECTO Y QUE PUEDAS AUMENTAR TUS HABILIDADES EN EL LENGUAJE DE PROGRAMACION JAVA. ¡GRACIAS POR LEER!

“RECUERDA, SI PUEDES IMAGINARLO, PUEDES PROGRAMARLO” (Programación ATS, 2018).