

Patrons utilitzats

- Singleton

com/bibliosedaos/desktop/api/ApiFactory.java

com/bibliosedaos/desktop/security/SessionStore.java

- Factory

com/bibliosedaos/desktop/api/ApiFactory.java

- Injecció de dependències (via ControllerFactory)

com/bibliosedaos/desktop/MainApp.java;

com/bibliosedaos/desktop/controller/LoginController.java;

com/bibliosedaos/desktop/controller/DashboardController.java

- Capa de Serveis

com/bibliosedaos/desktop/service/AuthService.java

- MVC

com/bibliosedaos/desktop/login-view.fxml com/bibliosedaos/desktop/dashboard-view.fxml

com/bibliosedaos/desktop/controller/LoginController.java

com/bibliosedaos/desktop/controller/DashboardController.java

com/bibliosedaos/desktop/model/dto/LoginRequest.java;

com/bibliosedaos/desktop/model/dto/LoginResponse.java

- DTO

com/bibliosedaos/desktop/model/dto/LoginRequest.java

com/bibliosedaos/desktop/model/dto/LoginResponse.java

- Mock / Test double

com/bibliosedaos/desktop/api/mock/MockAuthApi.java

- Estratègia

com/bibliosedaos/desktop/api/AuthApi.java com/bibliosedaos/desktop/api/http/HttpAuthApi.java

com/bibliosedaos/desktop/api/mock/MockAuthApi.java

- Façana

com/bibliosedaos/desktop/api/ApiClient

- Observer

com/bibliosedaos/desktop/controller/BooksBrowseController.java

Conexió servidor-client

com/bibliosedaos/desktop/api/ApiClient.java

com/bibliosedaos/desktop/api/ApiFactory.java

com/bibliosedaos/desktop/api/AuthApi.java

com/bibliosedaos/desktop/api/http/HttpAuthApi.java

com/bibliosedaos/desktop/api/AutorApi.java

com/bibliosedaos/desktop/api/http/HttpAutorApi.java

com/bibliosedaos/desktop/api/ExemplarApi.java

com/bibliosedaos/desktop/api/http/ExemplarAuthApi.java

com/bibliosedaos/desktop/api/LibreApi.java

com/bibliosedaos/desktop/api/http/HttpLibreApi.java

com/bibliosedaos/desktop/api/UserApi.java

com/bibliosedaos/desktop/api/http/HttpUserApi.java

Sistema de xifratge

- Token JWT: L'aplicació client rep el token JWT del servidor en una resposta HTTP després del login (AuthService i HttpAuthApi). Aquest token s'emmagaçza en la SessionStore mentre dura la sessió de l'usuari i s'utilitza per autoritzar peticions posteriors al servidor.

Quan l'usuari fa logout, el mètode SessionStore.clear() elimina el token i les dades d'usuari de la memòria. També es realitza una petició al servidor per revocar el token (afegint-lo a una blacklist).

Observacions i coses a remarcar:

Repositori github:

[SergiSancho/BiblioSedaos-Client-Desktop: Client d'escriptori\(JavaFx\) del projecte BiblioSedaos](#)

Codi generat amb eines d'intel·ligència artificial

Assistents: ChatGPT-5 (versió web), DeepSeek AI V3.1(versió web)

MainApp.java

- Prompts utilitzats:

"Com puc implementar injecció de dependències en JavaFX sense Spring Framework?"

"Necessito un exemple de Composition Root per a una aplicació JavaFX"

"Com configurar una ControllerFactory per a injecció de dependències en FXML"

- Assistència rebuda:

Disseny del patró Composition Root per a la creació centralitzada de dependències

Implementació de ControllerFactory per a injecció manual de dependències

Estructura de configuració flexible amb mode mock/real

- Correccions realitzades:

Adaptació dels conceptes arquitectònics als requisits específics del projecte

Implementació del sistema de configuració amb fitxers properties

Integració amb el Navigator

AnimationUtils.java

- Prompts utilitzats:

"Com puc implementar efectes d'animació per a botons en JavaFX?"

"JavaFX CSS no suporta transicions d'animació, quina alternativa hi ha?"

"Necessito un efecte de pressió per a botons amb ScaleTransition"

- Assistència rebuda:

Recomanació d'usar ScaleTransition com a alternativa a CSS per a animacions

Implementació d'efecte de pressió amb interpolar EASE_BOTH

Gestió d'estats (PRESSED/RELEASED) per a l'animació

Sistema per evitar múltiples animacions simultànies

- Correccions realitzades:

Adaptació dels valors d'escala als requisits visuals del projecte

Implementació de gestió d'esdeveniments MOUSE_EXITED per a restauració

Optimització de la durada i suavitat de l'animació

Navigator.java

- Prompts utilitzats:

"Què és un Navigator en JavaFX i per què l'utilitza la gent?"

"Necessito un sistema per canviar entre pantalles en JavaFX sense crear noves instàncies cada vegada"

"Com gestionar la navegació entre diferents vistes FXML en una aplicació JavaFX?"

- Assistència rebuda:

Concepte del Navigator: La IA va explicar que un Navigator és un patró comú en JavaFX per centralitzar la gestió de navegació entre pantalles, evitant la duplicació de codi i millorant la mantenibilitat

Arquitectura proposada: Sistema que gestiona càrrega de FXML, injecció de dependències, aplicació d'estils CSS i canvis d'escena

Implementació: Codi per a càrrega de vistes, gestió d'àrees de contingut, registre d'estils i control d'excepcions

- Correccions realitzades:

Conversió de Singleton a insància regular per millor testabilitat

Creació d'excepcions específiques per a errors de càrrega FXML

SessionStore.java

Prompts utilitzats:

"Com gestionar la sessió d'usuari en un client JavaFX de manera segura?"

"On és millor emmagatzemar el token JWT en una aplicació d'escriptori?"

"És necessari synchronized en un Singleton de sessió per JavaFX?"

Assistència rebuda:

Recomanació de seguretat: La IA va aconsellar emmagatzemar el token només en memòria (no en disc) i netejar-lo en fer logout

Patró Singleton: Va proposar aquest patró per assegurar una única instància global de les dades de sessió

Sincronització: La IA va recomanar synchronized per a entorns multi-fil, però després d'anàlisi es va determinar que al cas concret de l'aplicació JavaFX no és estrictament necessari

Correccions realitzades:

Revisió de sincronització: S'ha analitzat que el disseny actual de l'aplicació (UI single-threaded + Tasques sequencials) no requereix sincronització complexa

Neteja explícita de totes les variables en el mètode clear()

Documentació dels valors de rol (0=admin, 1=usuari)

Decisió final: S'han mantingut els synchronized com a mesura conservadora, tot i que l'anàlisi indica que l'aplicació actual no en requereix la protecció completa

AuthService.java

Prompts utilitzats:

"Com estructurar una capa de servei per a autenticació en JavaFX?"

"On hauria d'anar la lògica d'emmagatzematge de sessió: controller o servei?"

"És necessari implementar logout al client si el token s'elimina localment?"

Assistència rebuda:

Arquitectura en capes: La IA va recomanar separar la lògica de negoci (servei) de la UI (controller)

Injecció de dependències: Va proposar injectar AuthApi via constructor per a major testabilitat

Gestió de sessió: Va aconsellar gestionar el SessionStore des del servei, no des del controller

Logout simple: Va suggerir implementar un logout bàsic que netegi la sessió local, amb opció d'ampliar-ho al servidor en el futur

Correccions realitzades:

Validació de nulls al constructor

Simplificació dels tests

LoginController.java

Prompts utilitzats:

"Com estructurar un controlador JavaFX per a login amb validació i gestió d'errors?"

"Quines dependències he d'injectar en un controlador de login: Navigator, Service, tot?"

"Com utilitzar Tasks de JavaFX per a operacions en segon pla sense bloquejar la UI?"

- Assistència rebuda:

Arquitectura MVC: La IA va recomanar separar responsabilitats: Controller (UI), Service (lògica), Navigator (navegació)

Injecció de dependències: Va proposar injectar AuthService i Navigator via constructor per a major testabilitat

Gestió de fils: Va explicar com utilitzar Tasks per a operacions blocants i Platform.runLater() per a actualitzacions UI

Patró de disseny: Va suggerir l'ús de mètodes petits i específics per a cada responsabilitat

- Correccions realitzades:

Aplicació d'efectes visuals (AnimationUtils)

Ordenació i eliminació de mètodes innecessaris

DashboardController.java

- Prompts utilitzats:

"Com organitzar un controlador principal (dashboard) en JavaFX per a navegació centralitzada?"

"Com passar vistes dins d'un panell central (StackPane) i gestionar dependències sense Spring?"

"Quines bones pràctiques seguir per al maneig de sessió i navegació en controllers JavaFX?"

- Assistència rebuda:

Explicació i proposta d'un patró Navigator per gestionar la càrrega de vistes FXML dins d'un panell central (mainContentStack).

Recomanació d'injectar AuthService i Navigator via constructor per facilitar tests i separació de responsabilitats.

Aconsellada l'ús de Tasks per a operacions blocants i Platform.runLater() per a actualitzacions UI.

- Correccions realitzades:

S'ha decidit no usar controllerSetup com a obligatori: és una comoditat opcional que permet passar paràmetres o callbacks al controller carregat, però no és necessària si cada vista inicialitza les seves pròpies consultes en initialize().

ApiClient.java

Prompts utilitzats:

"Com implementar un client HTTP en Java per a consumir API REST?"

"Quina és la millor manera de gestionar fils en segon pla en JavaFX?"

"Com configurar HttpClient per a comunicacions amb servidor REST?"

"Com serialitzar i deserialitzar JSON en Java amb Jackson?"

Assistència rebuda:

Arquitectura HTTP: La IA va explicar com utilitzar HttpClient per a crides REST

Gestió de fils: Va proposar l'ús d'ExecutorService amb fils dimoni per a no bloquejar la UI

Serialització JSON: Va recomanar ObjectMapper de Jackson per a la conversió d'objectes

Patrons de disseny: Va suggerir una classe amb mètodes estàtics com a punt centralitzat

Correccions realitzades:

Configuració específica d'ExecutorService per a JavaFX

Preparació per a autenticació amb tokens JWT

ProfileEditController.java

Prompts utilitzats:

"Com gestionar la càrrega de dades d'usuari des del servidor i SessionStore com a fallback?"

"Necessito implementar validació de camps amb missatges d'error específics per a cada camp"

"Com gestionar l'actualització de SessionStore després d'editar el perfil?"

Assistència rebuda:

Arquitectura de càrrega de dades

Separació de responsabilitats: mètodes per validació, creació d'objectes User i execució de tasques

Gestió de l'estat de la UI durant operacions asíncrones (deshabilitar botons, mostrar errors)

Correccions realitzades:

Implementació del mètode safeGet() per evitar NullPointerException

Millora de la gestió d'errors mostrant missatges específics per cada camp

Conversió del paràmetre currentUserID de String a Long per consistència

UsersListController.java

Prompts utilitzats:

"Com implementar una taula amb paginació i cerca avançada en JavaFX?"

"Necessito una columna d'accions amb botons personalitzats per a cada fila"

"Com validar cerca per ID i NIF amb expressions regulars?"

"Com gestionar l'eliminació amb confirmació i actualització automàtica de la llista?"

Assistència rebuda:

Patró de disseny per a cel·les d'accions amb classes internes (ActionsTableCell)

Sistema de paginació manual amb llistes observables i càcul de pàgines

Validació d'entrada amb regex per ID (només números) i NIF (8 números + 1 lletra)

Arquitectura de cerca amb múltiples criteris i combinació de filtres

Correccions realitzades:

Millora de l'experiència d'usuari amb scroll automàtic al canviar pàgina

Gestió d'errors específics per a cerca per ID/NIF

Optimització del rendiment amb applyFilterAndPagination()

UserFormController.java

Prompts utilitzats:

"Com implementar un formulari reutilitzable per a crear, editar i visualitzar usuaris?"

"Com canviar dinàmicament entre modes (VIEW/EDIT/CREATE) en un formulari JavaFX?"

"Com validar contrasenyes i formats específics com NIF i telèfon?"

Assistència rebuda:

Patró de disseny amb mode d'operació que controla visibilitat i comportament

Sistema de configuració dinàmica d'UI amb setVisibleManaged()

Validació completa de formulari amb verificació de camps obligatoris i formats

Gestió d'ícones i textos dinàmics segons el mode

Correccions realitzades:

Validació específica per a cada camp amb missatges d'error detallats

Gestió adequada de l'estat dels botons durant operacions de guardat

BooksListController.java

Prompts utilitzats:

"Com implementar gestió de llibres amb relacions Many-to-Many amb autors?"

"Necessito verificar existència d'exemplars abans d'eliminar un llibre"

"Com gestionar la cerca per múltiples camps amb un sol camp de text?"

Assistència rebuda:

Ús de `ReadOnlyStringWrapper` per mostrar noms d'autors en la taula

Sistema de verificació d'exemplars abans d'eliminació per evitar inconsistències

Arquitectura de cerca unificada que cerca en múltiples camps simultàniament

Gestió de columnes d'accions amb botons contextuais

Correccions realitzades:

Implementació de verificació de dependències abans d'eliminar llibres

Gestió d'errors específica per a conflictes d'integridat referencial

BookFormController.java

Prompts utilitzats:

"Com gestionar un formulari amb relacions Many-to-One (llibres-autors) en JavaFX?"

"Necessito un sistema per afegir autors nous des del mateix formulari de llibres"

"Com implementar gestió d'exemplars dins del formulari de llibres?"

Assistència rebuda:

Sistema de creació d'autors en temps real des del formulari

Gestió de taula d'exemplars integrada en el formulari de llibres

Arquitectura per a modes mixtes (crear llibre + afegir exemplars)

Correccions realitzades:

Implementació de `mapReservatToDisplay()` per a estats d'exemplars més llegibles

Gestió de visibilitat dinàmica segons el mode d'operació

Validació de l'existència de llibre abans d'afegir exemplars

BooksBrowseController.java

Prompts utilitzats:

"Com implementar una vista dual que mostri llibres i exemplars de manera intercanviable?"

"Necessito un sistema de cerca específic per a exemplars disponibles"

"Com gestionar columnes dinàmiques que canvien segons el tipus de dades mostrat?"

"Com implementar cerca per títol i autor en exemplars disponibles?"

Assistència rebuda:

Arquitectura de vista dual amb showingExemplars com a commutador

Sistema de cerca especialitzat per a exemplars lliures

Gestió de columnes dinàmiques que s'activen/desactiven segons el context

Patró de cel·les d'accions genèriques que funcionen amb Object

Correccions realitzades:

Implementació de readOnlyLongWrapperSafe() per a gestió segura d'IDs

Millora del sistema de commutació entre vistes amb actualització automàtica