# Package Class Diagram

This file contains the whole class diagram split in packages. In this description there is the analysis of the central `DisplayController` and `ProxyDBController` which connect the packages with each other.

## DisplayController

The `DisplayController` handles all user interface navigation in the application. It is responsible for rendering pages, switching views, and displaying popup dialogs based on user actions and system flow. It connects UI actions with the appropriate views and supports both user and admin perspectives.

### Methods

- `DisplayController()` — Initializes the display controller.
- `displaySignInPage():void` — Shows the sign-in screen for user authentication.
- `displayProfilePage(account: AccountEntity):void` — Displays the profile page of the given account.
- `displayRequestVerificationPage(account: AccountEntity):void` — Opens the verification request page for the specified user.
- `displayMapView():void` — Switches to the map view interface showing event locations.
- `displayListView():void` — Displays events in a scrollable list layout.
- `displayEventTabList():void` — Populates the Listview with the event UI card elements accordingly.
- `displayFilterTabsPopup():void` — Opens a popup to apply filters to event tabs.
- `displayCreateEventPage():void` — Loads the form interface to create a new event.
- `displayOrganizedEventsPage(account:AccountEntity):void` — Shows all events organized by the given user account.
- `displayEventTabManage():void` — Loads the interface for managing an event.
- `displayEditEventPage(event:EventEntity):void` — Displays the edit form pre-filled with the selected event's data.
- `displayEventDetailsPage(event: EventEntity):void` — Opens the page of the detailed view of a specific event.
- `displayReviewPage(event: EventEntity):void` — Loads the review page for a specific event.
- `displayPopupDialog(message: String, buttonLabels: String[]):void` — Pushes a dialog popup with a message and configurable buttons.
- `displayVerificationRequestsForAdmin():void` — Displays all pending account verification requests for administrative review.

# ProxyDBController

The `ProxyDBController` acts as the main interface to the database. It handles all data operations related to events, accounts, reviews, and verification processes.

## Attributes

- `id: int` — Unique identifier for the controller instance.
- `dbConnection: DatabaseConnection` — Reference to the database connection used for executing queries.
- `cache: Map<String,Object>` — In-memory cache used to store frequently accessed data.
- `lastFetched: LocalDateTime` — Timestamp of the last data retrieval operation from the database.

## Methods

- `ProxyDBController()`
- `getAccountEvents(accountId: int):List<EventEntity>` — Returns all events associated with a specific account.
- `getVenueEvents(venueId: int):List<EventEntity>` — Returns all events organized by a specific venue.
- `getEventFeed(accountId: int):EventEntity` — Returns the event feed recommended to a specific account.
- `getNearbyEvents(location: String):List<EventEntity>` — Returns events close to a given location.
- `getEventsByFilters(filters: String[]):List<EventEntity>` — Filters and returns events according to criteria.
- `getEventsByKeywords(keywords: String[]):List<EventEntity>` — Returns events containing given keywords.
- `getLikedEvents(accountId: int):List<EventEntity>` — Returns events liked by the user.
- `getAllEvents():List<EventEntity>` — Returns all registered public events.
- `getNotifiedEvents(accountId: int):List<EventEntity>` — Returns events where user has notifications enabled.
- `deleteEvent(eventId: int):void` — Removes the specified event from the database.
- `createEvent(event: EventEntity):void` — Saves a new event to the event table.
- `editEvent(event: EventEntity):void` — Updates existing event data.
- `getAccountById(accountId: int):AccountEntity` — Retrieves account information by ID.
- `getFollowingEntities(accountId: int):Map<String,AccountEntity/VenueEntity>` — Returns every account and venue the user follows.
- `getEventVenue(eventId: int):VenueEntity` — Gets venue information for this event.

- `getEventOwner(eventId: int):AccountEntity` — Gets the creator (account) of an event.
- `getEventDetails(eventId: int):EventEntity` — Retrieves full details of specified event.
- `searchVenuesByName(name: String):List<VenueEntity>` — Returns matching venues by name.
- `searchAccountsByName(name: String):List<AccountEntity>` — Returns matching accounts by name.
- `rejectAccountVerification(accountId: int, message: String):void` — Denies verification request with reason.
- `verifyAccount(accountId: int):void` — Marks account as verified.
- `isAccountVerified(accountId: int):bool` — Checks whether account is verified.
- `getVerificationStatus(accountId: int):VerificationStatus` — Returns account's verification state.
- `updateVerificationRequest(accountId: int, files: String[]):void` — Uploads new verification documents.
- `getVerificationRequests():List<VerificationEntity>` — Returns pending verifications for review.
- `getAccountReviews(accountId: int):List<ReviewEntity>` — Returns all reviews written by the user.
- `createReview(review: ReviewEntity):void` — Adds a new review to the review table.
- `editReview(review: ReviewEntity):void` — Updates an existing review.
- `deleteReview(reviewId: int):void` — Deletes review by ID.
- `getEventReview(eventId: int):List<ReviewEntity>` — Returns all reviews for a given event.
- `updateCache():void` — Refreshes internal cache with latest data.