Project “Music Store”

Iteration 2: design, analysis, and inception of implementation

Members:

Karl Kangur

Lukáš Škuta

Adilzhan Shukenov

Anthony Lassoudière

Ilja Gužovski

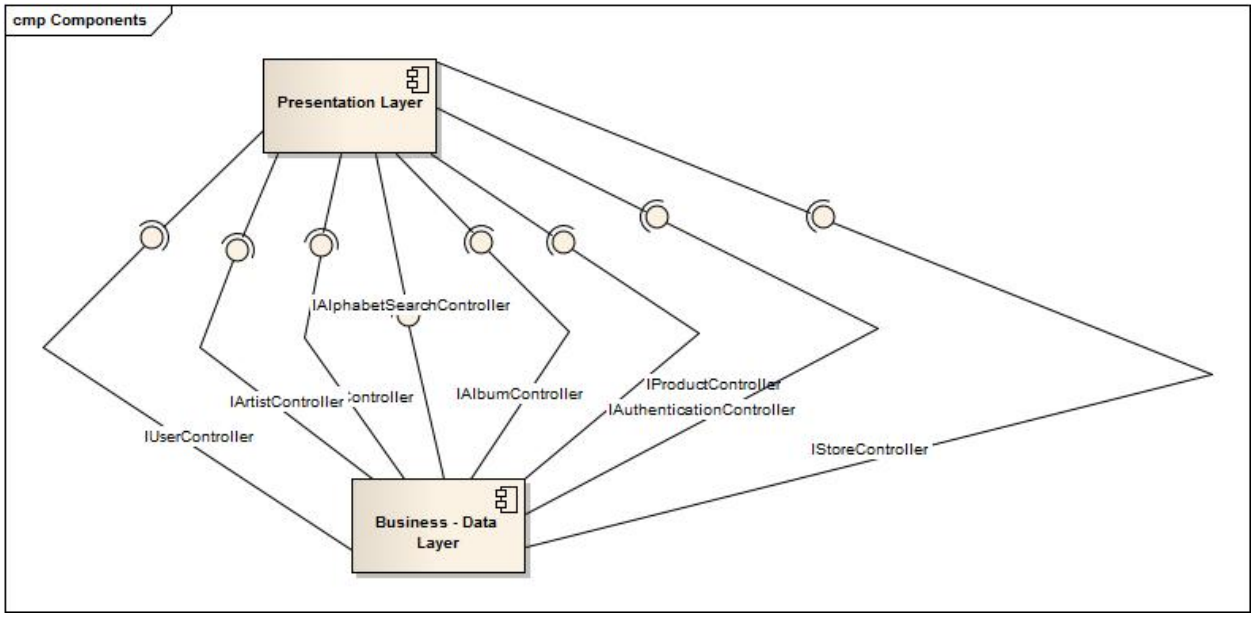
1. Architecture model

Application Architecture is designed as a three-layer architecture. The layers are the following:

1. Presentation layer.
2. Business-Data layer.

Each layer will provide maximum testability and maximize the loose coupling by using the dependency injection frameworks.

Application architecture is shown in the following figure:

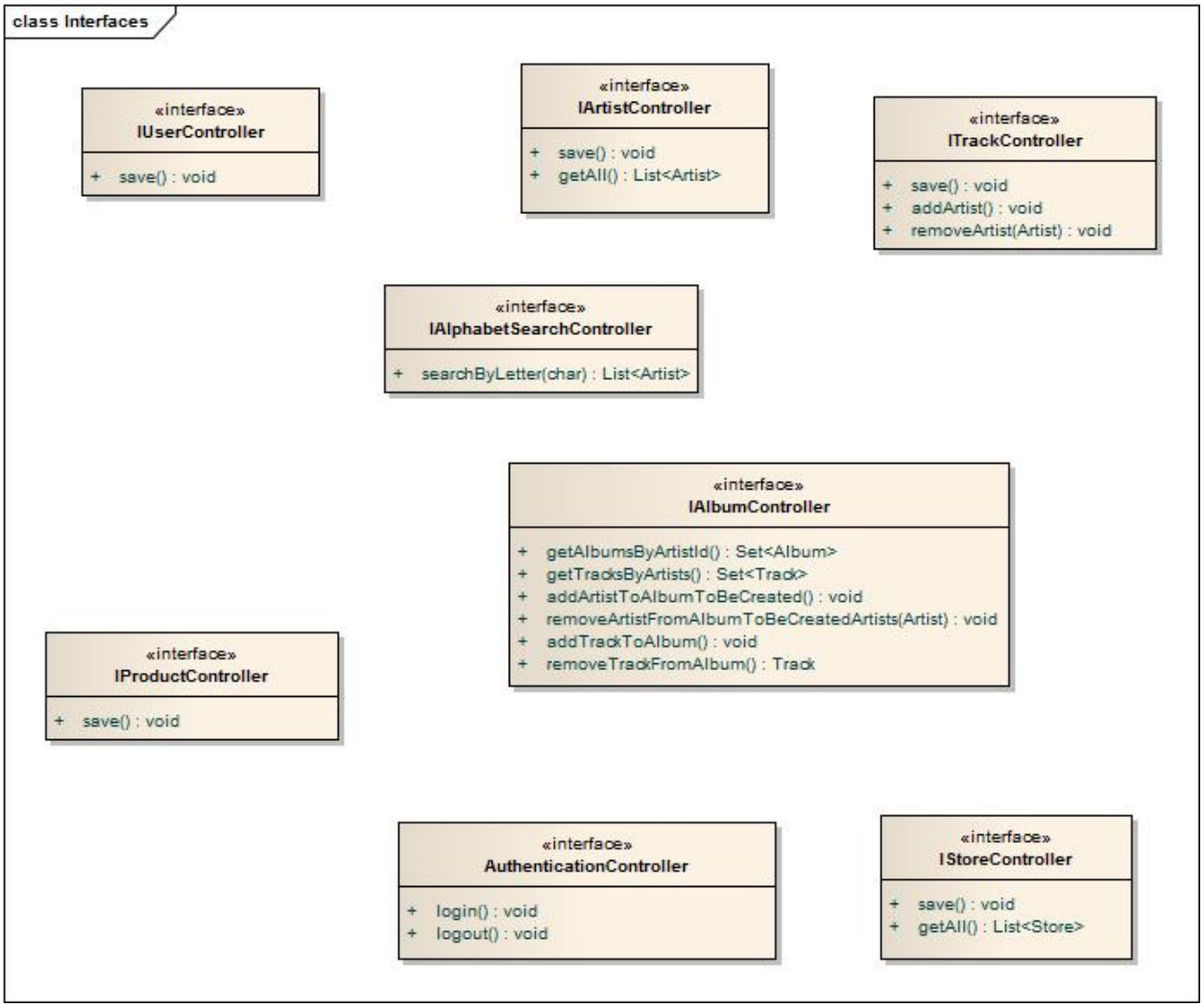


Application layers.

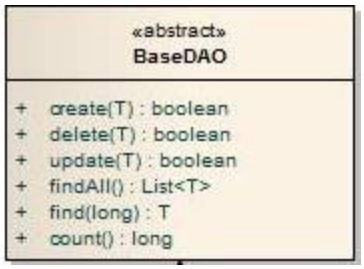
Presentation Layer

Presentation layer contains classes and components that are responsible for presentation of the informations to the user and takes from user requirements. These requirements are transformed in the way that remove dependency on the used technology and forwards them next to the data model. All the classes, which represent controllers are marked by @Named annotation and are named like <Some>Controller. All the controllers are built on top of the java EE ManagedBean technology.

Business-Data Layer



Contains classes and components that implement required behaviour of application from the view of business logic. The Business logic in our application are built on top of the java EJB(Enterprise JavaBeans) technology, so called container of the business logic. Most of the EJB in our application deal with the persistence. That is why the most of them extend so called BaseDAO, in order to remove the code duplication and follow DRY(Do not Repeat Yourself) principle. Because most of the time you need DAOs only for CRUD operations, we decided to not to do the DAO layer, but instead of that, mix it with data layer.

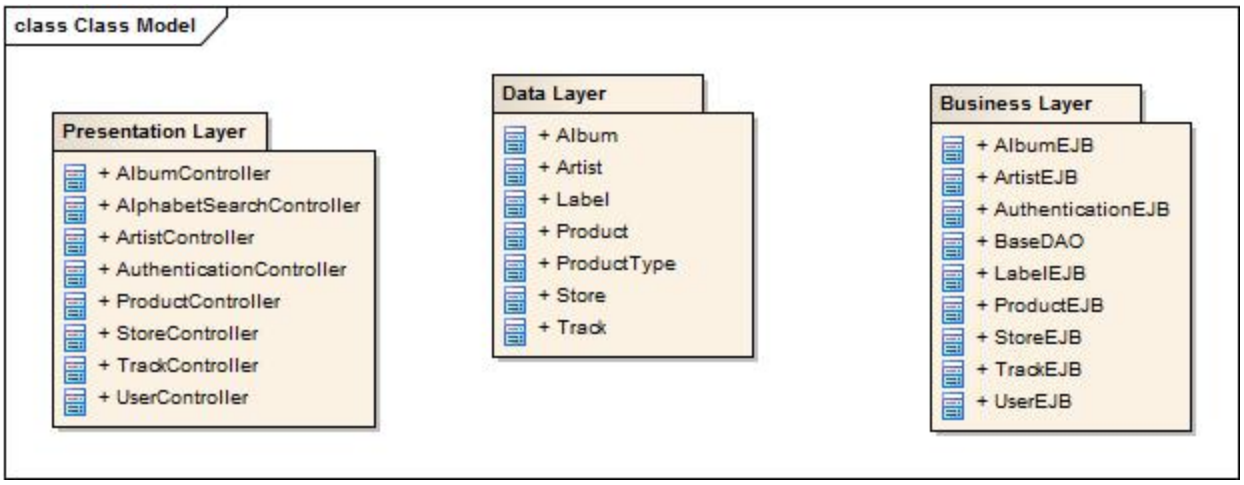


BaseDAO generic class.

2. Class diagram

Package “Class model” and review of the application layers

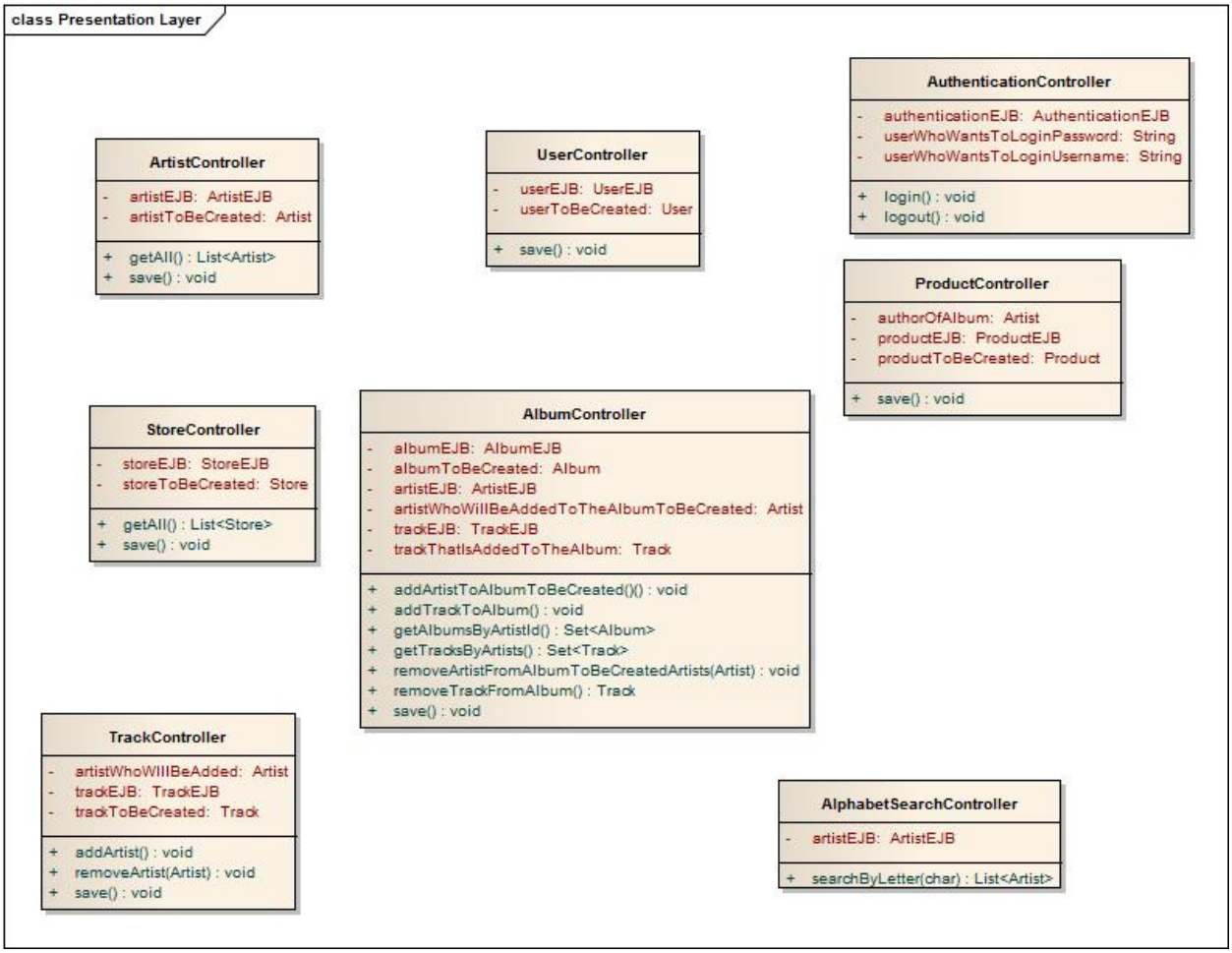
Package contains classes implementing project Music Store. Whole application is divided into three basic packages that correspond to existing layers three-layers architecture. All the dependencies and injection is managed by Java EE 7 CDI technology. This results in looser coupling and higher cohesion at the same time.

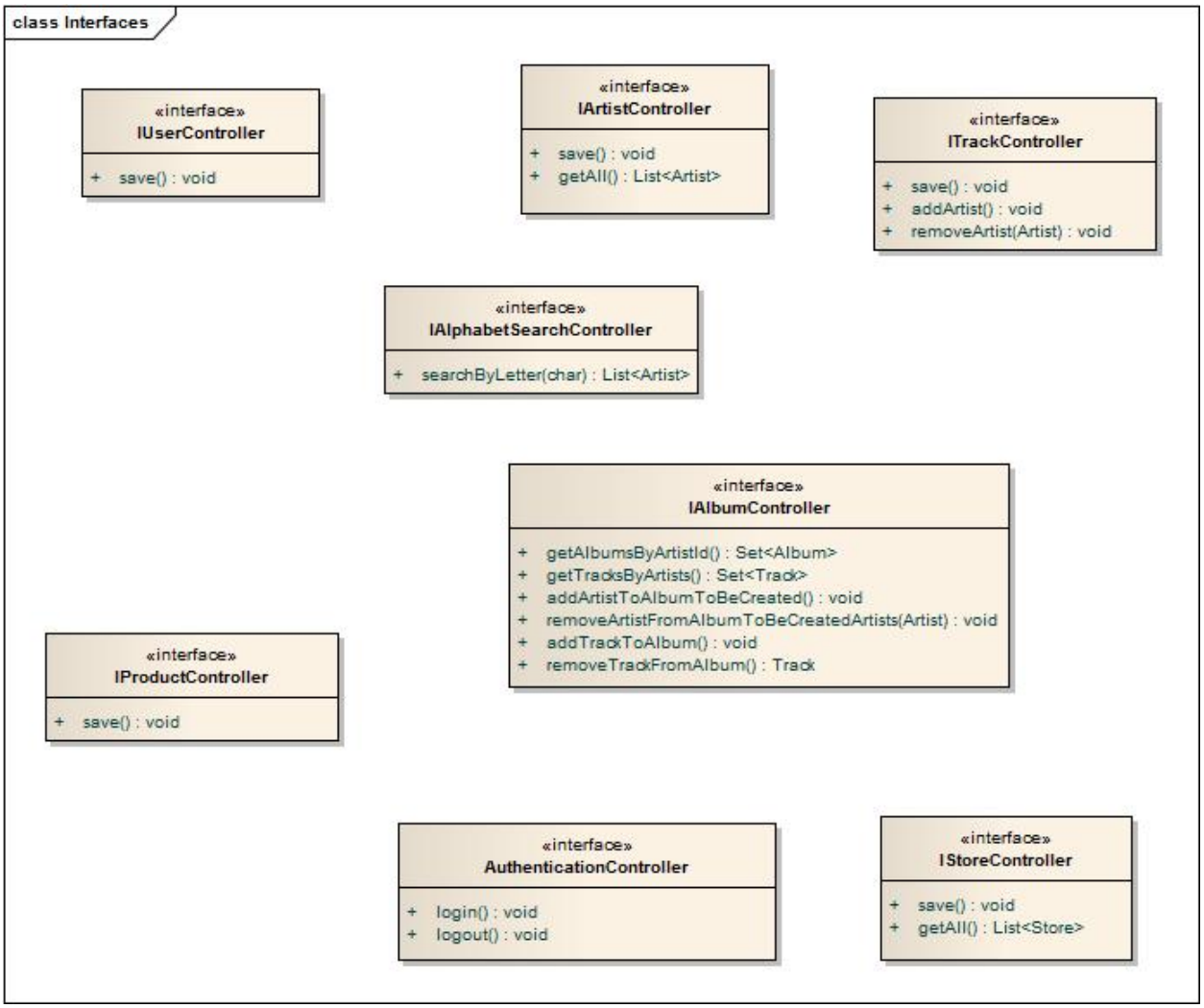


Root representation of the packages.

Package “Presentation layer”

Package contains classes of presentation layer that enable display the state of application to user and forward user’s requirements to business layer. The main goal of this project isn’t explaining principe user interface design, therefore the classes aren’t described in detail. In case of saving the space of the each diagram, we have not put here getters and setters for each attribute. Note, that because we use JSF(Java Server Faces), which is a lightweight component MVC framework, all the controllers have huge amount of different so called components (EJBs and POJOs). This MVC is not request oriented (as Spring MVC is for instance), so it is worse in terms of debugging, and it is harder to understand which methods are called by URL, because they are executed most of the time directly from JSF pages. But from the other point of view, it is much more easier to work with a session, for instance creating multi page (or page by page) processes, because current page remembers the information about the previous page.

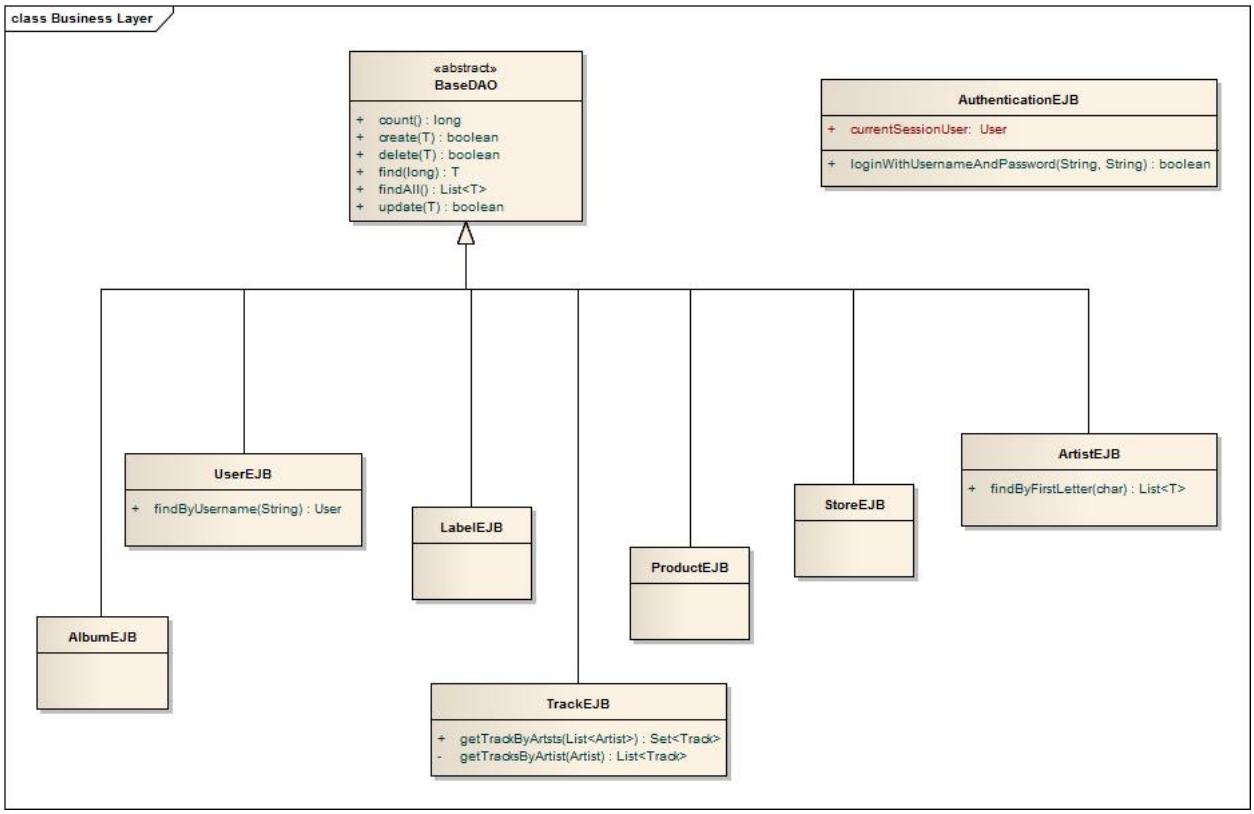




All controllers and their methods.

Package ”Business-Data layer”

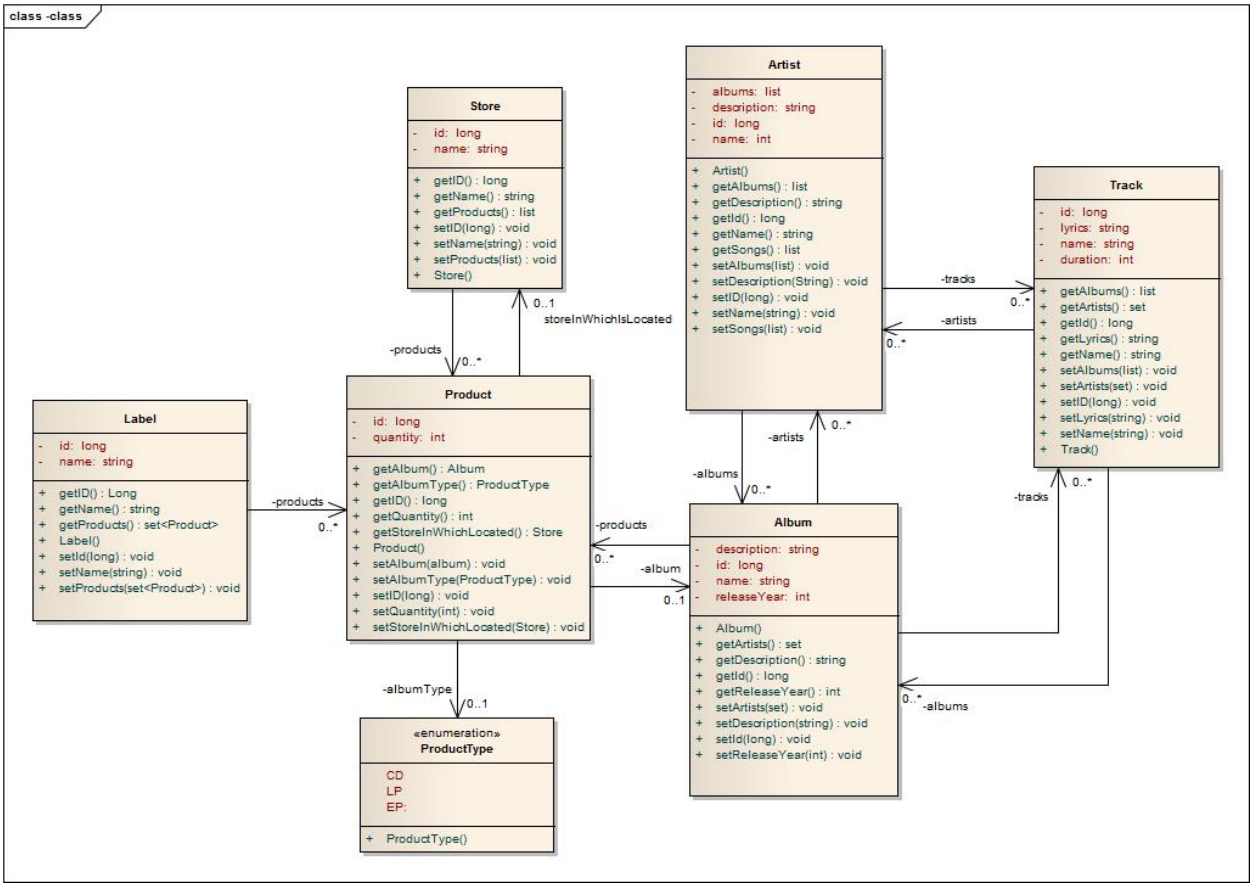
This package contains classes of business layer, that implement behaviour of whole application. These classes implement whole behaviour of application regarding business logic. Most of the business logic here is very clear and duplicated, that is why BaseDAO here is used. All EJB (except AuthenticationEJB, which holds logged in user) are stateless. From Java SE 6, we do not need to create a dao for each object anymore, because of generics introduced with that release.



Business layer classes.

Package “Data layer”

This package contains classes of data layer, that provide technical support for classes of business layer. This is an implementation of the domain model. All the classes here are POJOs. Some methods, like equals(), hashCode(), toString() are not shown. Note that all the relations between the classes are bidirectional. This could be easily done by using the hibernate or like in our case JPA (Java Persistence Api). So if from the business layer called create method, for example, then the associations are saved/persisted from both sides automatically.



POJOs of the data layer.

1) DATA LAYER CLASSES:

Album

An album is a collection of audio recordings (e.g., pieces of music). Album represent a piece of information, like what tracks it contains. It is not physical product.

Connections

Connector	Source	Target	Notes
Association -products Source -> Destination	Public Album	Public Product	
Association -album Source -> Destination	Public Product	Public Album	
Association -albums Source -> Destination	Public Artist	Public Album	
Association -artists Source -> Destination	Public Album	Public Artist	
Association -albums Source -> Destination	Public Track	Public Album	
Association -tracks Source -> Destination	Public Album	Public Track	

Attributes

Attribute	Notes	Constraints and tags
description string Private	description of the album	*Default:*
id long	unique identifier	*Default:*

Private		
name string Private	album name	*Default:*
releaseYear int Private	year, when the album was released.	*Default:*

Operations

Method	Notes	Parameters
Album() Public		
getArtists() set Public		
getDescription() string Public		
getId() long Public		
getReleaseYear() int Public		
setArtists() void Public		set [in] Artists
setDescription() void Public		string [in] description
setId() void Public		long [in] id
setReleaseYear() void Public		int [in] releaseYear

Artist

An artist takes part in the creation of a product by producing the tracks that composed it. Artist takes part in different ways depending if they are singer or compositor.

Connections

Connector	Source	Target	Notes
Association -tracks Source -> Destination	Public Artist	Public Track	
Association -artists Source -> Destination	Public Track	Public Artist	
Association -albums Source -> Destination	Public Artist	Public Album	
Association -artists Source -> Destination	Public Album	Public Artist	

Attributes

Attribute	Notes	Constraints and tags
albums list Private	albums that artist has done	*Default:*
description string Private	description of the artist	*Default:*
id long Private	unique identifier	*Default:*

name int Private	name of the artist	*Default:*

Operations

Method	Notes	Parameters
Artist() Public		
getAlbums() list Public		
getDescription() string Public		
getId() long Public		
getName() string Public		
getSongs() list Public		
setAlbums() void Public		list [in] albums
setDescription() void Public		String [in] description
setID() void Public		long [in] id
setName() void Public		string [in] name
setSongs() void Public		list [in] songs

Label

Record label of the Product. A record label is a brand and/or a trademark associated with the marketing of music recordings and music videos.

Connections

Connector	Source	Target	Notes
Association -products Source -> Destination	Public Label	Public Product	

Attributes

Attribute	Notes	Constraints and tags
id long Private	unique identifier	*Default:*
name string Private	record label's name	*Default:*

Operations

Method	Notes	Parameters
getID() Long Public		
getName() string Public		
getProducts() set<Product> Public		

Label() Public		
setId() void Public		long [in] id
setName() void Public		string [in] name
setProducts() void Public		set<Product> [in] products

Product

The product is the numerical element representing a physical product. For this store we have three different types of products : CD (Compact Disc), EP (Extended Play) and LP (Long Play). Each product is identified by it's name and release date, helping the searching of the customers.

Connections

Connector	Source	Target	Notes
Association -albumType Source -> Destination	Public Product	Public ProductType	
Association -products Source -> Destination	Public Album	Public Product	
Association storeInWhichIsLocated Source -> Destination	Public Product	Public Store	
Association -album Source -> Destination	Public Product	Public Album	
Association -products Source -> Destination	Public Label	Public Product	
Association -products	Public	Public	

Source -> Destination	Store	Product	

Attributes

Attribute	Notes	Constraints and tags
id long Private	unique identifier	*Default:*
quantity int Private	quantity of the product	*Default:*

Operations

Method	Notes	Parameters
getAlbum() Album Public		
getAlbumType() ProductType Public		
getID() long Public		
getQuantity() int Public		
getStoreInWhichLocated() Store Public		
Product() Public		
setAlbum() void Public		album [in] album
setAlbumType() void		ProductType [in] albumtyoe

Public		
setID() void Public		long [in] id
setQuantity() void Public		int [in] quantity
setStoreInWhichLocated() void Public		Store [in] storeInWhichLocated

ProductType

Physical type of the product.

Connections

Connector	Source	Target	Notes
Association -albumType Source -> Destination	Public Product	Public ProductType	

Attributes

Attribute	Notes	Constraints and tags
CD ProductType Public «enum»	Compact disc, or CD for short, is a digital optical disc data storage format. Contains up to 80 minutes of play.	*Default:*
LP ProductType Public «enum»	The LP (Long Play), or vinyl record, is a format for phonograph (gramophone) records, an analog sound storage medium. Contains up to 45 minutes of play.	*Default:*

EP: ProductType Public «enum»	An extended play (EP) is a musical recording that contains more music than a single, but is too short to qualify as a full studio album or LP (Long Play). The EP is typically 3 or 4 tracks.	*Default:*

Operations

Method	Notes	Parameters
ProductType() Public		

Store

A store is the physical place where customers will be able to buy a product or pick it up if they bought it from the application.

Connections

Connector	Source	Target	Notes
Association storeInWhichIsLocated Source -> Destination	Public Product	Public Store	
Association -products Source -> Destination	Public Store	Public Product	

Attributes

Attribute	Notes	Constraints and tags
id long Private	unique identifier	*Default:*
name string	name of the store	*Default:*

Private		

Operations

Method	Notes	Parameters
getID() long Public		
getName() string Public		
getProducts() list Public		
setID() void Public		long [in] id
setName() void Public		string [in] name
setProducts() void Public		list [in] products
Store() Public		

Track

A track is a part of a product. It is one of the songs on the product. The track is identified by it's name. The duration and the genre are also indicated to help for searching. The lyrics are not always linked to the track.

Connections

Connector	Source	Target	Notes
Association -tracks Source -> Destination	Public Artist	Public Track	
Association -artists	Public	Public	

Source -> Destination	Track	Artist	
Association -albums Source -> Destination	Public Track	Public Album	
Association -tracks Source -> Destination	Public Album	Public Track	

Attributes

Attribute	Notes	Constraints and tags
id long Private	unique identifier	*Default:*
lyrics string Private	lyrics of the track (song)	*Default:*
name string Private	name of the track	*Default:*
duration int Private	duration of track like 2:32	*Default:*

Operations

Method	Notes	Parameters
getAlbums() list Public		
getArtists() set Public		
getId() long Public		

getLyrics() string Public		
getName() string Public		
setAlbums() void Public		list [in] albums
setArtists() void Public		set [in] artists
setID() void Public		long [in] id
setLyrics() void Public		string [in] lyrics
setName() void Public		string [in] name
Track() Public		

2) BUSINESS LAYER CLASSES:

AlbumEJB

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public AlbumEJB	Public BaseDAO	

ArtistEJB

Connections

Connector	Source	Target	Notes
Generalization	Public	Public	

Source -> Destination	ArtistEJB	BaseDAO	

Operations

Method	Notes	Parameters
findByFirstLetter() List<T> Public	Returns the list of artist by the beginning letter of their name.	char [in] letter

AuthenticationEJB

Attributes

Attribute	Notes	Constraints and tags
currentSessionUser User Public	user who logged in.	*Default:*

Operations

Method	Notes	Parameters
loginWithUsernameAndPassword() boolean Public	returns true if user was successfully logged in.	String [in] password String [in] username

BaseDAO

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public UserEJB	Public BaseDAO	

Generalization Source -> Destination	Public AlbumEJB	Public BaseDAO	
Generalization Source -> Destination	Public ArtistEJB	Public BaseDAO	
Generalization Source -> Destination	Public ProductEJB	Public BaseDAO	
Generalization Source -> Destination	Public StoreEJB	Public BaseDAO	
Generalization Source -> Destination	Public TrackEJB	Public BaseDAO	
Generalization Source -> Destination	Public LabelEJB	Public BaseDAO	

Operations

Method	**Notes**	**Parameters**
create() boolean Public	persists the entity to the database.	T [in] entity
delete() boolean Public	deletes the entity from database.	T [in] entity
update() boolean Public	updates entity from current memory to database.	T [in] entity
findAll() List<T> Public	finds all entities of proposed type.	
find() T Public	finds entity by primary key number.	long [in] id

count() long Public	counts all entities of given type T.	

LabelEJB

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public LabelEJB	Public BaseDAO	

ProductEJB

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ProductEJB	Public BaseDAO	

StoreEJB

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public StoreEJB	Public BaseDAO	

TrackEJB

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public TrackEJB	Public BaseDAO	

Operations

Method	Notes	Parameters
getTrackByArtsts() Set<Track> Public	return Set of track composed by different artists	List<Artist> [in] artsits
getTracksByArtist() List<Track> Private	gets all the track of artist.	Artist [in] artsits

UserEJB

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public UserEJB	Public BaseDAO	

Operations

Method	Notes	Parameters
findByUsername() User Public	finds User by username	String [in] useraname

3) PRESENTATION LAYER CLASSES:

AlbumController

Attributes

Attribute	Notes	Constraints and tags
albumToBeCreated Album Private		*Default:*
artistWhoWillBeAddedToTheAlbumToBeCreated Artist Private		*Default:*
trackThatIsAddedToTheAlbum Track Private		*Default:*
albumEJB AlbumEJB Private		*Default:*
trackEJB TrackEJB Private		*Default:*
artistEJB ArtistEJB		*Default:*

Private		

Operations

Method	Notes	Parameters
save() void Public	saves album to the database.	
getAlbumsByArtistId() Set<Album> Public	returns all artist albums.	
getTracksByArtists() Set<Track> Public	return all artist tracks.	
addArtistToAlbumToBeCreated() void Public	add some artist as author of some album.	
removeArtistFromAlbumToBeCreatedArtists() void Public	remove some artist as author of some album.	Artist [in] a
addTrackToAlbum() void Public	add some track to the album.	
removeTrackFromAlbum() Track Public	remove some track from the album.	

AlphabetSearchController

Attributes

Attribute	Notes	Constraints and tags
artistEJB ArtistEJB Private		*Default:*

Operations

Method	Notes	Parameters
searchByLetter() List<Artist> Public	returns the list of artists, whose name start with corresponding character.	char [in] c

ArtistController

Attributes

Attribute	Notes	Constraints and tags
artistToBeCreated Artist Private		*Default:*
artistEJB ArtistEJB Private		*Default:*

Operations

Method	Notes	Parameters
save() void Public	saves the artist to the database.	
getAll() List<Artist> Public	returns list of all Users created	

AuthenticationController

Attributes

Attribute	Notes	Constraints and tags
authenticationEJB AuthenticationEJB Private		*Default:*
userWhoWantsToLoginUsername String Private		*Default:*
userWhoWantsToLoginPassword String Private		*Default:*

Operations

Method	Notes	Parameters
login() void Public	performs user login.	
logout() void Public	destroys the session.	

ProductController

Attributes

Attribute	Notes	Constraints and tags
productToBeCreated Product Private		*Default:*
authorOfAlbum Artist Private		*Default:*

productEJB ProductEJB Private		*Default:*

Operations

Method	Notes	Parameters
save() void Public	saves Product to database.	

StoreController

Attributes

Attribute	Notes	Constraints and tags
storeToBeCreated Store Private		*Default:*
storeEJB StoreEJB Private		*Default:*

Operations

Method	Notes	Parameters
save() void Public		
getAll() List<Store> Public		

TrackController

Attributes

Attribute	Notes	Constraints and tags
trackToBeCreated Track Private		*Default:*
trackEJB TrackEJB Private		*Default:*
artistWhoWIllBeAdded Artist Private		*Default:*

Operations

Method	Notes	Parameters
save() void Public	saves track to database	
addArtist() void Public	adds some artist to the track authors	
removeArtist() void Public	removes some artist from the track authors	Artist [in] artist

UserController

Attributes

Attribute	Notes	Constraints and tags
userToBeCreated User		*Default:*

Private		
userEJB UserEJB Private		*Default:*

Operations

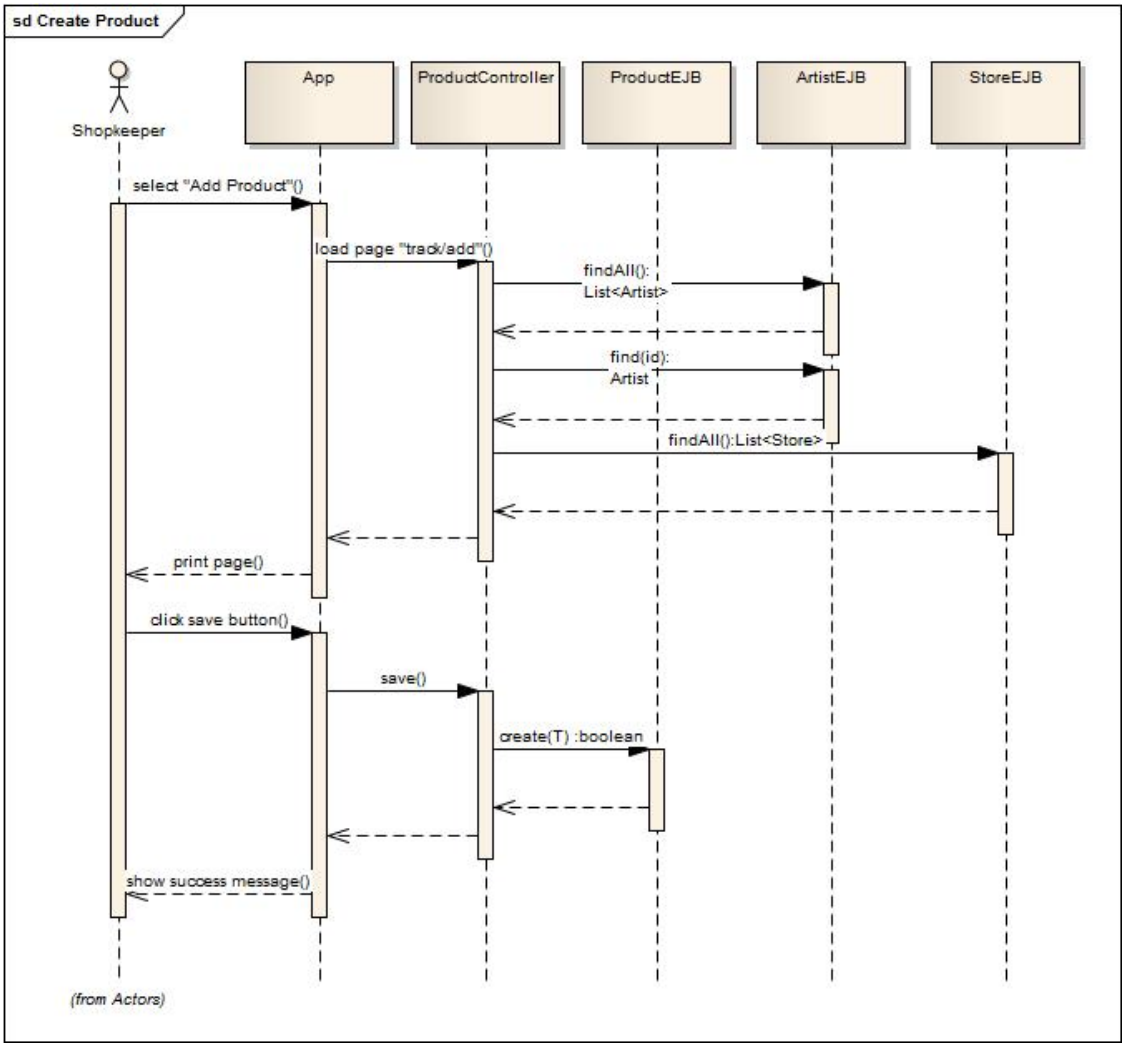
Method	**Notes**	**Parameters**
save() void Public	saves User to Database.	

3. Communication model.

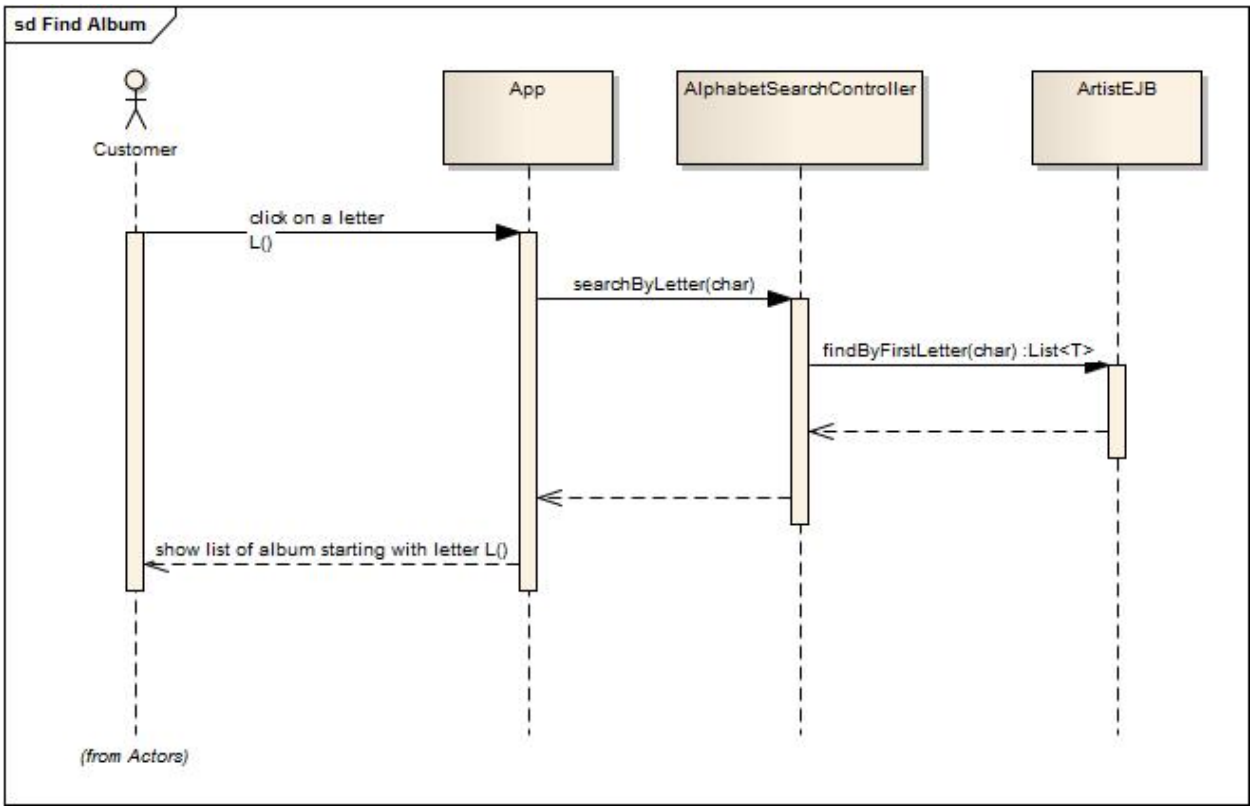
This chapter describe assignment of classes's responsibility during implementation of required functions. Existing classes were described in previous chapter. This chapter contains only description of classes's cooperation, that are interesting from the object oriented design's view.

Create product

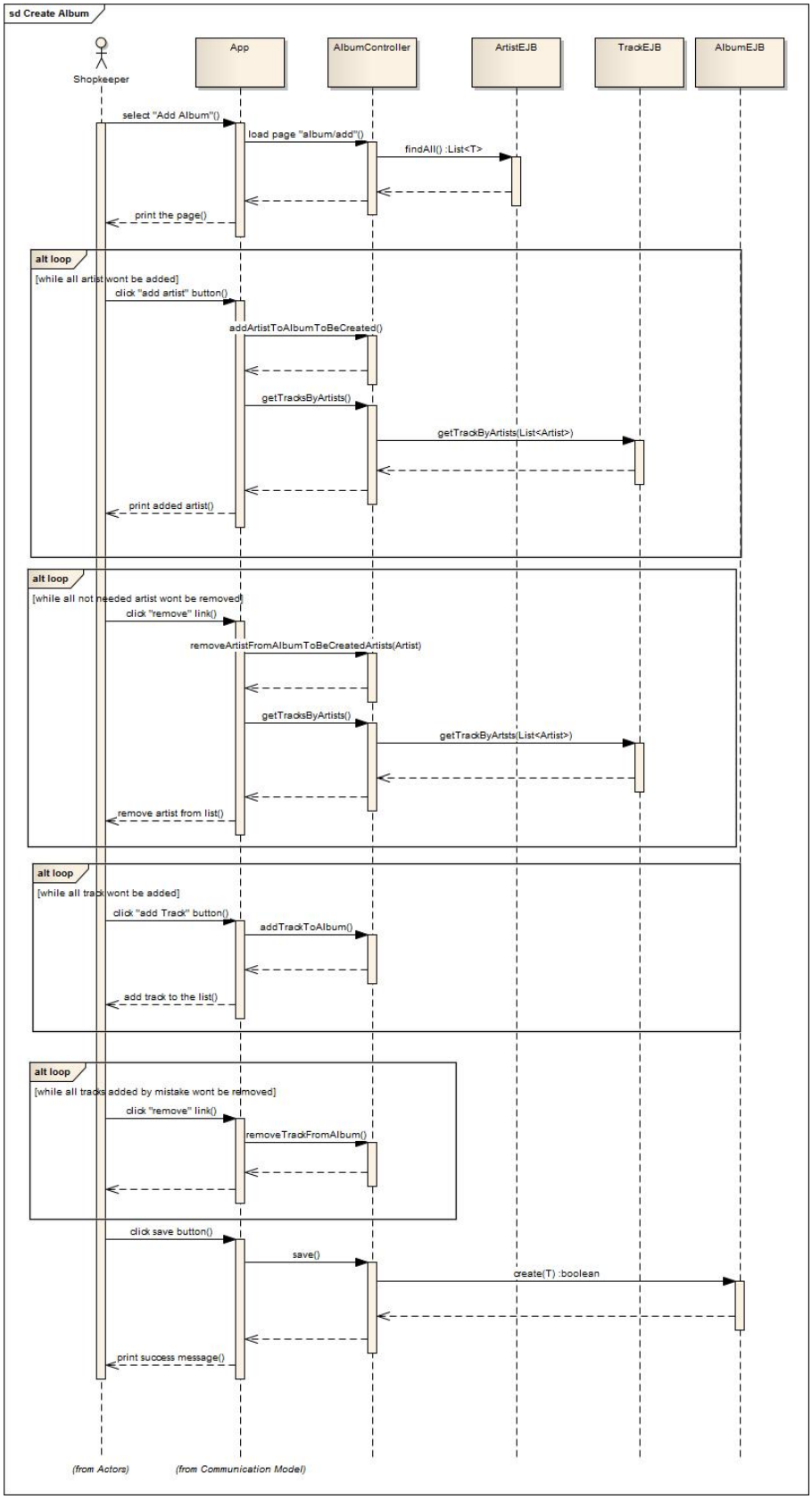
This chapter contains description of classes's cooperation for implementation of functionality related with creating of product. At the picture is captured cooperation of classes during creating of product.



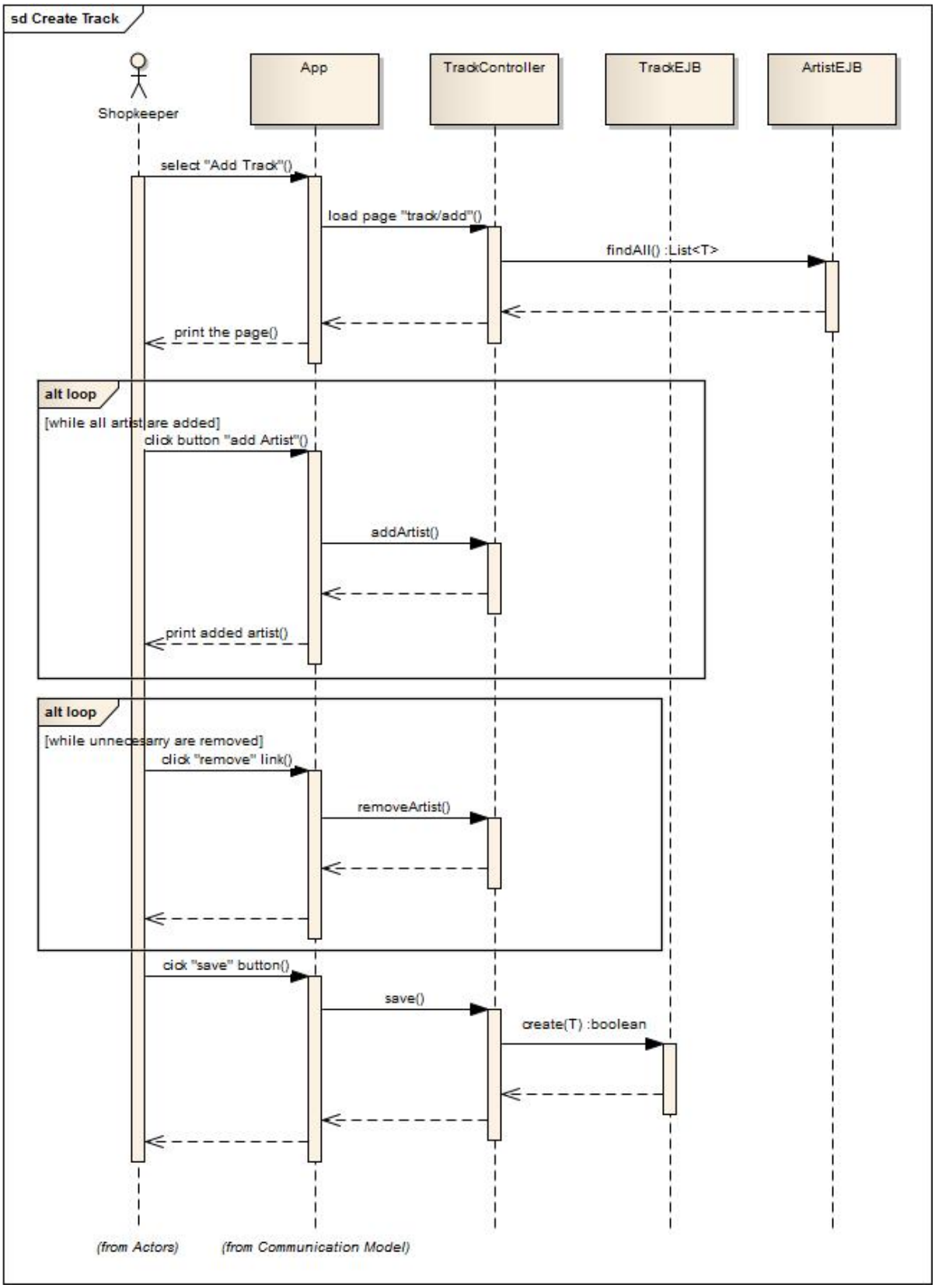
Find Album


sd Find Album
Customer
App
AlphabetSearchController
ArtistEJB
click on a letter
L()
searchByLetter(char)
findByFirstLetter(char) :List<T>
show list of album starting with letter L()
(from Actors)

Create Album



Create Track

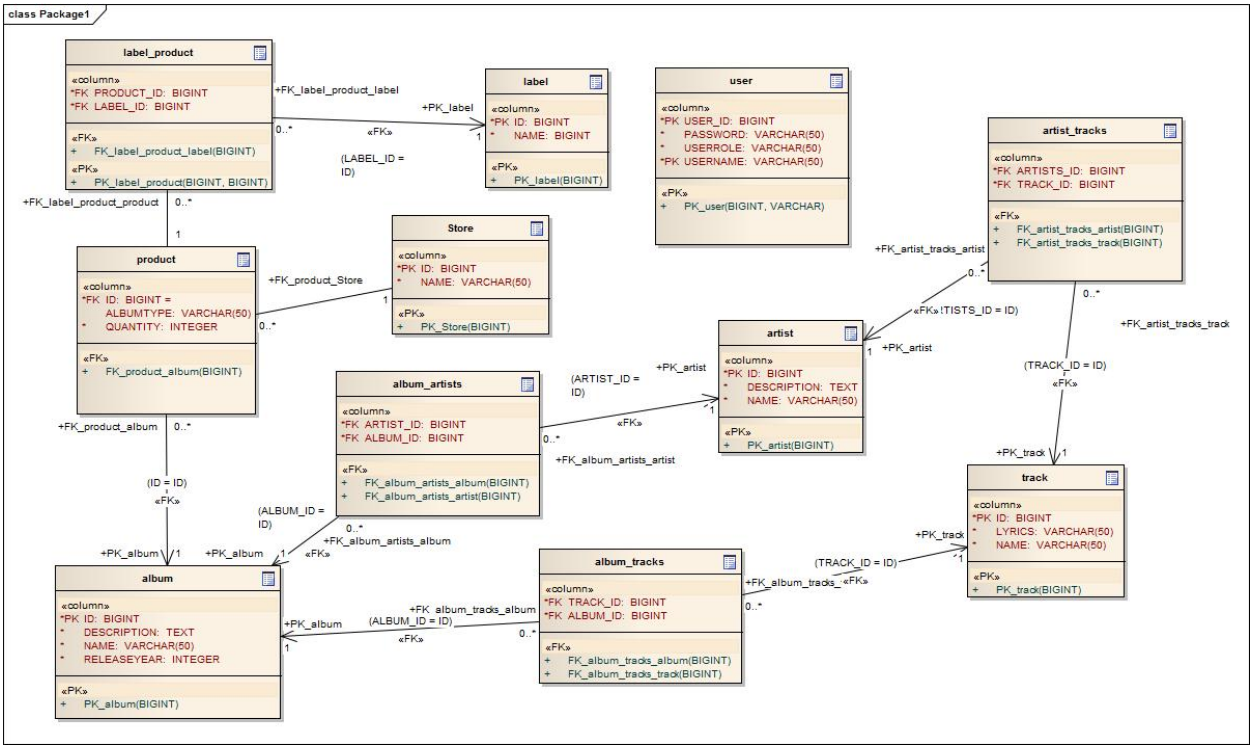


4. Database model

The chapter contains the proposed method of storing data in a relational database.

Tables of Database.

Music Store Database view



Relational model of the database.

Store

Table of stores(ID, NAME)

Attribute	Notes	Constraints and tags
ID BIGINT Public «column»		*Default:*
NAME VARCHAR Public «column»	Name of Store	*Default:* Less than 255 letters.

Album

Table of albums (ID,DESCRIPTION,NAME,RELEASEYEAR)

Attribute	Notes	Constraints and tags
ID BIGINT Public «column»		*Default:*
DESCRIPTION TEXT Public «column»	Description of album	*Default:* Must be BLOB.
NAME VARCHAR Public «column»	Name of album	*Default:*
RELEASEYEAR INTEGER Public «column»	Release Year of an Album	*Default:* Minimum is 1500

album_artists

Attribute	Notes	Constraints and tags
ARTIST_ID BIGINT Public «column»		*Default:*
ALBUM_ID BIGINT Public «column»		*Default:*

album_tracks

Attribute	Notes	Constraints and tags
TRACK_ID BIGINT Public «column»		*Default:*
ALBUM_ID BIGINT Public «column»		*Default:*

artist

Table, which contains an information about artists(ID,DESCRIPTION,NAME)

Attribute	Notes	Constraints and tags
ID BIGINT Public «column»		*Default:*
DESCRIPTION TEXT Public «column»	Description about an artist	*Default:* Must be BLOB.
NAME VARCHAR Public «column»	Name of an artist	*Default:* Less than 255 letters.

artist_tracks

Attribute	Notes	Constraints and tags
ARTISTS_ID BIGINT Public «column»		*Default:*
TRACK_ID BIGINT Public «column»		*Default:*

label

Table of labels

Attribute	Notes	Constraints and tags
ID BIGINT Public «column»		*Default:*
NAME BIGINT Public «column»	Name of Label	*Default:* Less than 255 letters.

label_product

Attribute	Notes	Constraints and tags
PRODUCT_ID BIGINT Public «column»		*Default:*
LABEL_ID BIGINT Public «column»		*Default:*

product

Table of products

Attribute	Notes	Constraints and tags
ID BIGINT Public «column»		*Default:*
ALBUMTYPE VARCHAR Public	Type of album(EP,LP,CD)	*Default:*

«column»		
QUANTITY INTEGER Public «column»	Quantity of product	*Default:* Must be positive and greatet than zero.

track

Attribute	Notes	Constraints and tags
ID BIGINT Public «column»		*Default:*
LYRICS VARCHAR Public «column»	Lyrics of a track	*Default:*
NAME VARCHAR Public «column»	Name of a track	*Default:* Less than 255 letters.

user

Attribute	Notes	Constraints and tags
USER_ID BIGINT Public «column»		*Default:*
PASSWORD VARCHAR Public «column»	Password of User	*Default:* More than 6 letters. Crypted
USERROLE VARCHAR Public «column»	Role of User	*Default:* Default role of user is USER.
USERNAME VARCHAR Public «column»	Name of User	*Default:* Unique.