Project Part 4

Team:
Aditya Thyagarajan,
Prasanna Kumar Srinivasachar,
Vibhor Mishra

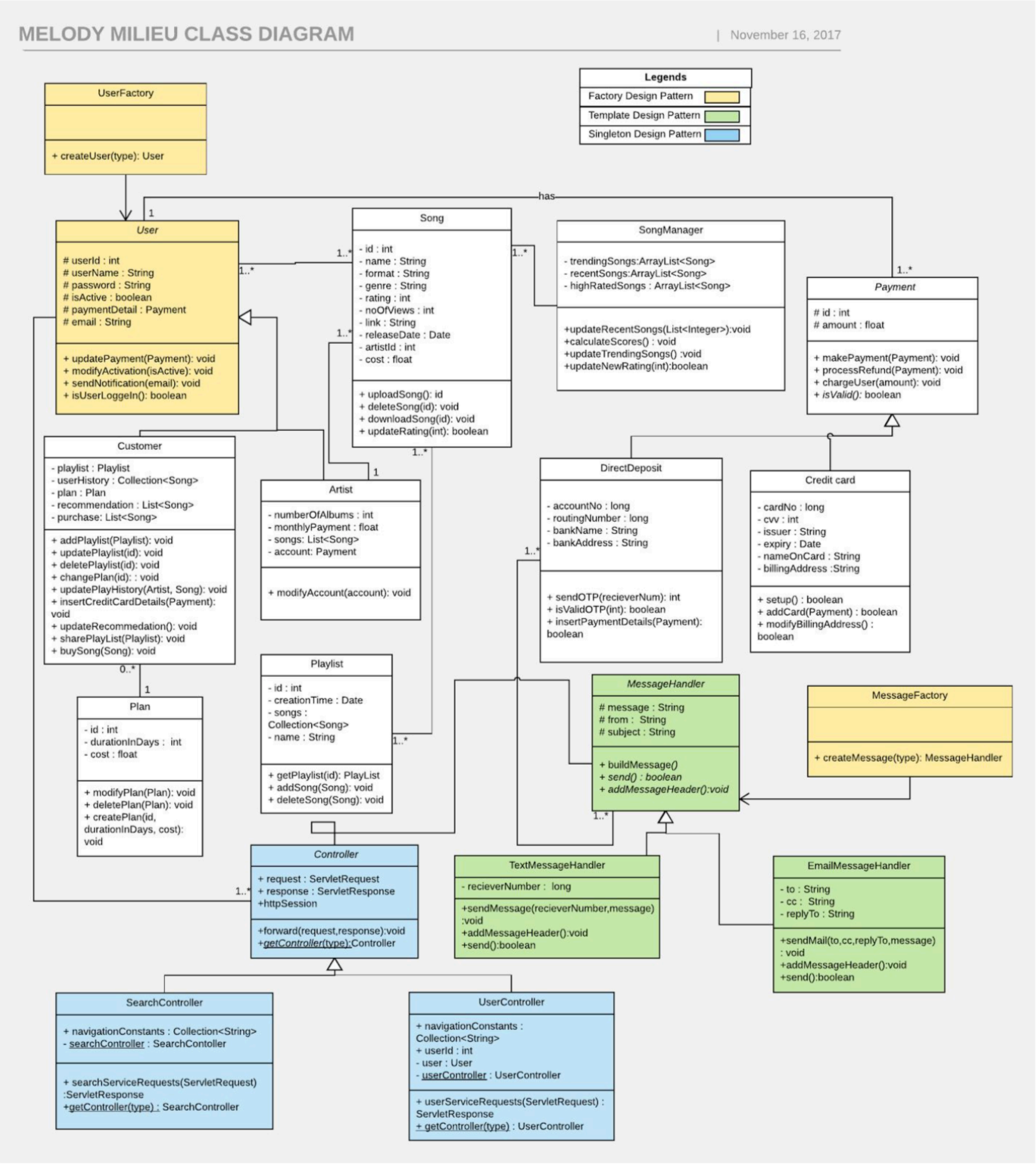
Title: Melody Milieu

Functions Implemented	
ID	**Title**
UR-01	As a new user, I want to sign up
UR-02	As a user, I want to log in
UR-03	As a user, I want to search songs
UR-04	As a user, I want to search artists
UR-05	As a user, I want to search songs by language
UR-06	As a user, I want to view my recent songs
UR-07	As a user, I want to create playlists
UR-08	As a user, I want to view recommendations
UR-09	As a user, I want to buy songs
UR-10	As a user, I want to choose/change my subscription plan
UR-14	As a user, I want to add credit/debit cards
UR-20	As an artist, I want to setup my direct deposit
UR-21	As an artist, I want to Create an account
UR-27	As an admin, I want to change activation status users/artists.

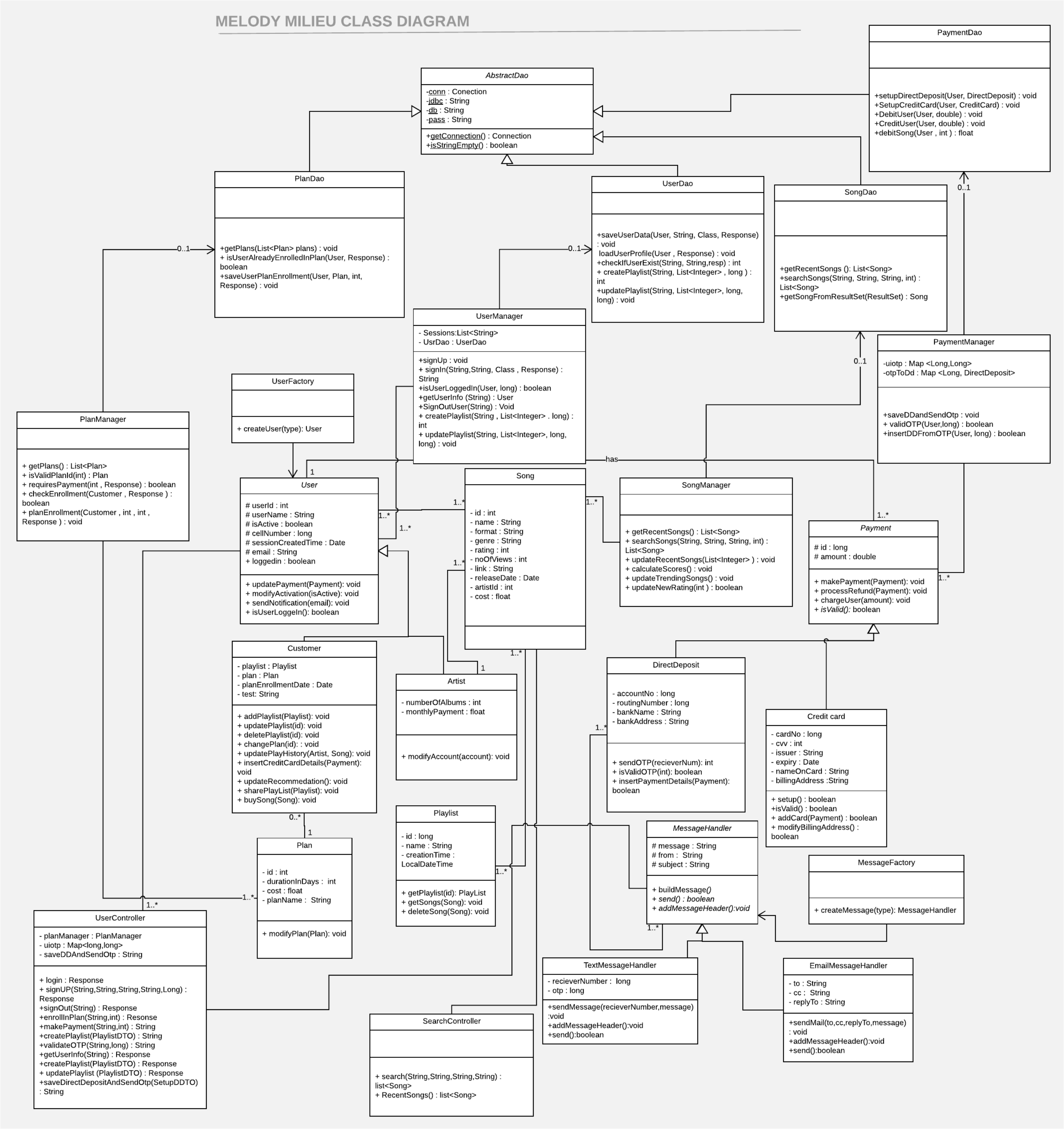
Functions not implemented

ID	Requirements
UR-11	As a user, I want to choose my payment methods
UR-12	As a user, I want to download songs
UR-13	As a user, I want to share playlists
UR-15	As a user, I want to rate a song
UR-16	As an artist, I want to upload songs
UR-17	As an artist, I want to view the popularity of my songs
UR-18	As an artist, I want to download my songs
UR-19	As an artist, I want to monitor my portfolio
UR-22	As an admin, I want to Create plan for users/artists
UR-23	As an admin, I want to modify plan for users/artists.
UR-24	As an admin, I want to view payment methods for users/artists.
UR-25	As an admin, I want to manage logs to check for suspicious activity
UR-26	As an admin, I want to perform health checks

Old Class Diagram


MELODY MILIEU CLASS DIAGRAM
| November 16, 2017
Legends
Factory Design Pattern
Template Design Pattern
Singleton Design Pattern
UserFactory
+ createUser(type): User
has
1
1..*
User
# userId : int
# userName : String
# password : String
# isActive : boolean
# paymentDetail : Payment
# email : String
+ updatePayment(Payment): void
+ modifyActivation(isActive): void
+ sendNotification(email): void
+ isUserLoggeIn(): boolean
Song
- id : int
- name : String
- format : String
- genre : String
- rating : int
- noOfViews : int
- link : String
- releaseDate : Date
- artistId : int
- cost : float
+ uploadSong(): id
+ deleteSong(id): void
+ downloadSong(id): void
+ updateRating(int): boolean
SongManager
- trendingSongs:ArrayList<Song>
- recentSongs:ArrayList<Song>
- highRatedSongs : ArrayList<Song>
+updateRecentSongs(List<Integer>):void
+calculateScores() : void
+updateTrendingSongs() :void
+updateNewRating(int):boolean
Payment
# id : int
# amount : float
+ makePayment(Payment): void
+ processRefund(Payment): void
+ chargeUser(amount): void
+ isValid(): boolean
Customer
- playlist : Playlist
- userHistory : Collection<Song>
- plan : Plan
- recommendation : List<Song>
- purchase: List<Song>
+ addPlaylist(Playlist): void
+ updatePlaylist(id): void
+ deletePlaylist(id): void
+ changePlan(id): : void
+ updatePlayHistory(Artist, Song): void
+ insertCreditCardDetails(Payment): void
+ updateRecommedation(): void
+ sharePlayList(Playlist): void
+ buySong(Song): void
Artist
- numberOfAlbums : int
- monthlyPayment : float
- songs: List<Song>
- account: Payment
+ modifyAccount(account): void
DirectDeposit
- accountNo : long
- routingNumber : long
- bankName : String
- bankAddress : String
+ sendOTP(recieverNum): int
+ isValidOTP(int): boolean
+ insertPaymentDetails(Payment): boolean
Credit card
- cardNo : long
- cvv : int
- issuer : String
- expiry : Date
- nameOnCard : String
- billingAddress :String
+ setup() : boolean
+ addCard(Payment) : boolean
+ modifyBillingAddress() : boolean
0..*
Plan
- id : int
- durationInDays : int
- cost : float
+ modifyPlan(Plan): void
+ deletePlan(Plan): void
+ createPlan(id, durationInDays, cost): void
Playlist
- id : int
- creationTime : Date
- songs : Collection<Song>
- name : String
+ getPlaylist(id): PlayList
+ addSong(Song): void
+ deleteSong(Song): void
MessageHandler
# message : String
# from : String
# subject : String
+ buildMessage()
+ send() : boolean
+ addMessageHeader():void
MessageFactory
+ createMessage(type): MessageHandler
Controller
+ request : ServletRequest
+ response : ServletResponse
+httpSession
+forward(request,response):void
+getController(type):Controller
TextMessageHandler
- recieverNumber : long
+sendMessage(recieverNumber,message) :void
+addMessageHeader():void
+send():boolean
EmailMessageHandler
- to : String
- cc : String
- replyTo : String
+sendMail(to,cc,replyTo,message) : void
+addMessageHeader():void
+send():boolean
SearchController
+ navigationConstants : Collection<String>
- searchController : SearchContoller
+ searchServiceRequests(ServletRequest) :ServletResponse
+getController(type) : SearchController
UserController
+ navigationConstants : Collection<String>
+ userId : int
- user : User
- userController : UserController
+ userServiceRequests(ServletRequest) : ServletResponse
+ getController(type) : UserController

New class diagram : Note : most of the attributes filled by lombok are not shown (finalclass.png in GitHub repo)

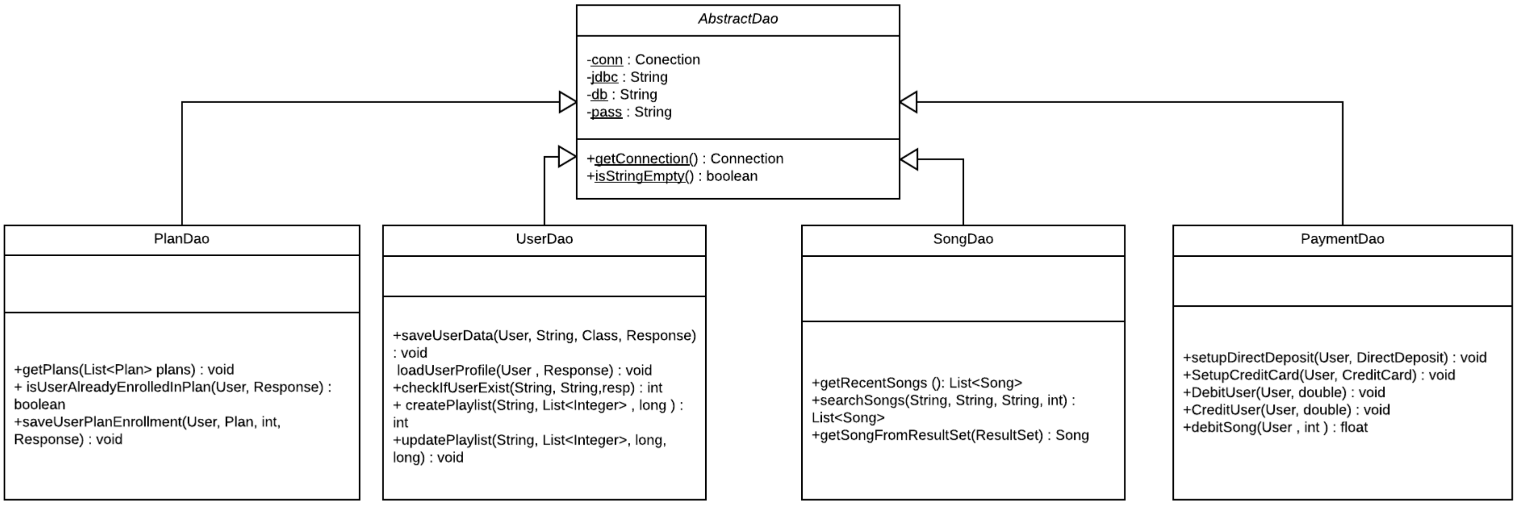

MELODY MILIEU CLASS DIAGRAM
AbstractDao
-conn : Conection
-jdbc : String
-db : String
-pass : String
+getConnection() : Connection
+isStringEmpty() : boolean
PaymentDao
+setupDirectDeposit(User, DirectDeposit) : void
+SetupCreditCard(User, CreditCard) : void
+DebitUser(User, double) : void
+CreditUser(User, double) : void
+debitSong(User , int) : float
PlanDao
+getPlans(List<Plan> plans) : void
+ isUserAlreadyEnrolledInPlan(User, Response) : boolean
+saveUserPlanEnrollment(User, Plan, int, Response) : void
UserDao
+saveUserData(User, String, Class, Response) : void
loadUserProfile(User , Response) : void
+checkIfUserExist(String, String,resp) : int
+ createPlaylist(String, List<Integer> , long) : int
+updatePlaylist(String, List<Integer>, long, long) : void
SongDao
+getRecentSongs (): List<Song>
+searchSongs(String, String, String, int) : List<Song>
+getSongFromResultSet(ResultSet) : Song
0..1
0..1
0..1
0..1
UserManager
- Sessions:List<String>
- UsrDao : UserDao
+signUp : void
+ signIn(String,String, Class , Response) : String
+isUserLoggedIn(User, long) : boolean
+getUserInfo (String) : User
+SignOutUser(String) : Void
+ createPlaylist(String , List<Integer> . long) : int
+ updatePlaylist(String, List<Integer>, long, long) : void
PaymentManager
-uiotp : Map <Long,Long>
-otpToDd : Map <Long, DirectDeposit>
+saveDDandSendOtp : void
+ validOTP(User,long) : boolean
+insertDDFromOTP(User, long) : boolean
UserFactory
+ createUser(type): User
PlanManager
+ getPlans() : List<Plan>
+ isValidPlanId(int) : Plan
+ requiresPayment(int , Response) : boolean
+ checkEnrollment(Customer , Response) : boolean
+ planEnrollment(Customer , int , int , Response) : void
has
1
User
# userId : int
# userName : String
# isActive : boolean
# cellNumber : long
# sessionCreatedTime : Date
# email : String
+ loggedin : boolean
+ updatePayment(Payment): void
+ modifyActivation(isActive): void
+ sendNotification(email): void
+ isUserLoggeIn(): boolean
1..*
1..*
1..*
Song
- id : int
- name : String
- format : String
- genre : String
- rating : int
- noOfViews : int
- link : String
- releaseDate : Date
- artistId : int
- cost : float
1..*
1..*
SongManager
+ getRecentSongs() : List<Song>
+ searchSongs(String, String, String, int) : List<Song>
+ updateRecentSongs(List<Integer>) : void
+ calculateScores() : void
+ updateTrendingSongs() : void
+ updateNewRating(int) : boolean
1..*
Payment
# id : long
# amount : double
+ makePayment(Payment): void
+ processRefund(Payment): void
+ chargeUser(amount): void
+ isValid(): boolean
1..*
Customer
- playlist : Playlist
- plan : Plan
- planEnrollmentDate : Date
- test: String
+ addPlaylist(Playlist): void
+ updatePlaylist(id): void
+ deletePlaylist(id): void
+ changePlan(id): : void
+ updatePlayHistory(Artist, Song): void
+ insertCreditCardDetails(Payment): void
+ updateRecommedation(): void
+ sharePlayList(Playlist): void
+ buySong(Song): void
1..*
1
Artist
- numberOfAlbums : int
- monthlyPayment : float
+ modifyAccount(account): void
DirectDeposit
- accountNo : long
- routingNumber : long
- bankName : String
- bankAddress : String
+ sendOTP(recieverNum): int
+ isValidOTP(int): boolean
+ insertPaymentDetails(Payment): boolean
1..*
Credit card
- cardNo : long
- cvv : int
- issuer : String
- expiry : Date
- nameOnCard : String
- billingAddress :String
+ setup() : boolean
+isValid() : boolean
+ addCard(Payment) : boolean
+ modifyBillingAddress() : boolean
0..*
1
Plan
- id : int
- durationInDays : int
- cost : float
- planName : String
+ modifyPlan(Plan): void
Playlist
- id : long
- name : String
- creationTime : LocalDateTime
+ getPlaylist(id): PlayList
+ getSongs(Song): void
+ deleteSong(Song): void
1..*
MessageHandler
# message : String
# from : String
# subject : String
+ buildMessage()
+ send() : boolean
+ addMessageHeader():void
1..*
MessageFactory
+ createMessage(type): MessageHandler
1..*
1..*
UserController
- planManager : PlanManager
- uiotp : Map<long,long>
- saveDDAndSendOtp : String
+ login : Response
+ signUP(String,String,String,String,Long) : Response
+signOut(String) : Response
+enrollInPlan(String,int) : Resonse
+makePayment(String,int) : String
+createPlaylist(PlaylistDTO) : String
+validateOTP(String,long) : String
+getUserInfo(String) : Response
+createPlaylist(PlaylistDTO) : Response
+ updatePlaylist (PlaylistDTO) : Response
+saveDirectDepositAndSendOtp(SetupDDTO) : String
TextMessageHandler
- recieverNumber : long
- otp : long
+sendMessage(recieverNumber,message) :void
+addMessageHeader():void
+send():boolean
EmailMessageHandler
- to : String
- cc : String
- replyTo : String
+sendMail(to,cc,replyTo,message) : void
+addMessageHeader():void
+send():boolean
SearchController
+ search(String,String,String,String) : list<Song>
+ RecentSongs() : list<Song>

Changes made :

- Our previous class diagram didn’t have provision for DAO layer. We decided on having a layer dedicated for the database tasks, which solely handles all the database interaction.
- We also added Manager layer for validation, for any data transformation and caching of certain details required for some use cases. We also agreed that we can incorporate cache usage in the manager layer to reduce the database hits that will improve overall performance of our system.
- Spring Boot has its defined type of mapping for dispatcher servlet. We couldn’t wrap around the settings to have response forward to implement singleton pattern. Also we agreed given that its an established we can go along with what Spring boot has to offer.
- We implemented Singleton design pattern to obtain database connection, which worked well.

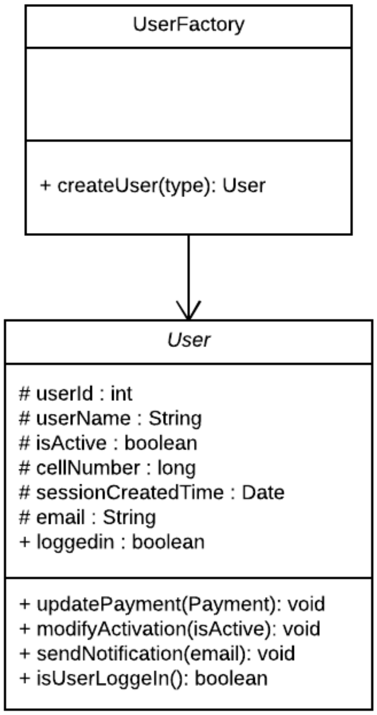
Design Patterns Used :

Singleton for database connection

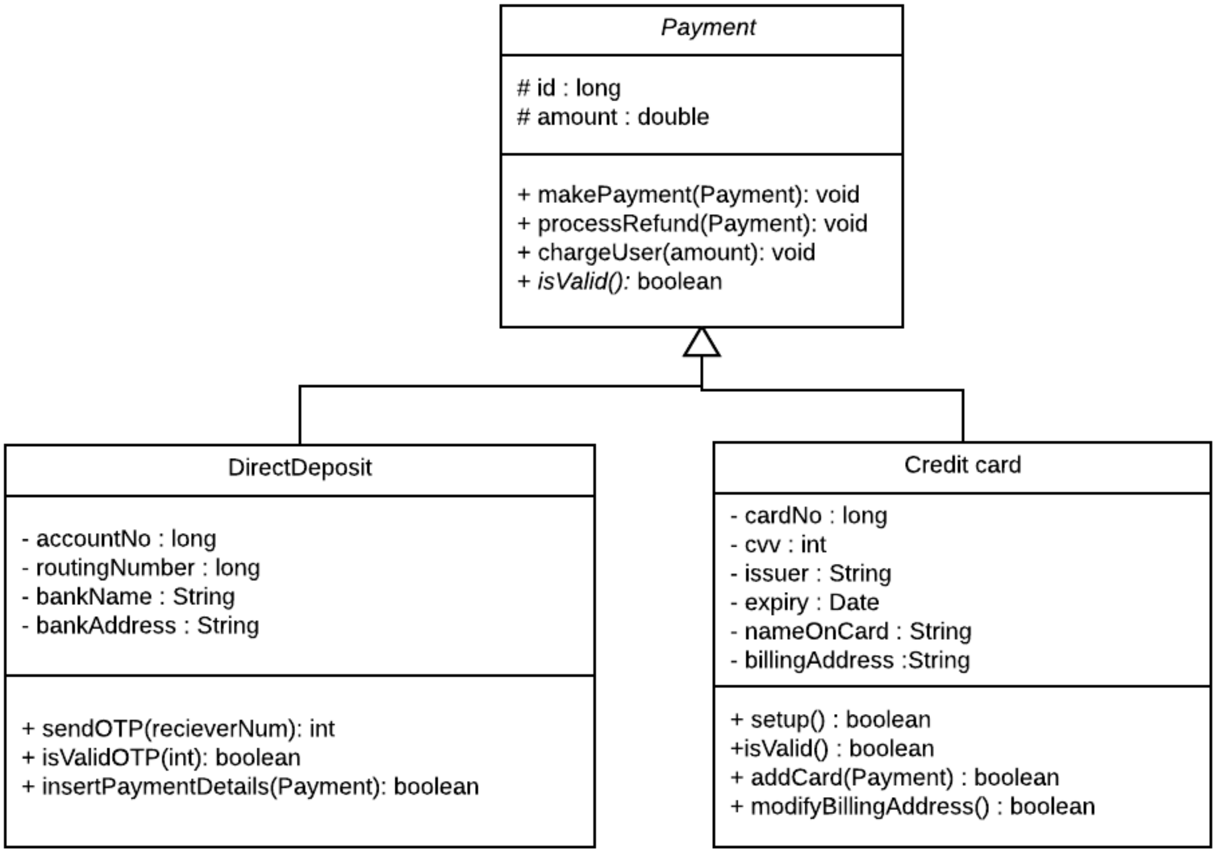


AbstractDao
-conn : Conection
-jdbc : String
-db : String
-pass : String
+getConnection() : Connection
+isStringEmpty() : boolean
PlanDao
+getPlans(List<Plan> plans) : void
+ isUserAlreadyEnrolledInPlan(User, Response) : boolean
+saveUserPlanEnrollment(User, Plan, int, Response) : void
UserDao
+saveUserData(User, String, Class, Response) : void
loadUserProfile(User , Response) : void
+checkIfUserExist(String, String,resp) : int
+ createPlaylist(String, List<Integer> , long) : int
+updatePlaylist(String, List<Integer>, long, long) : void
SongDao
+getRecentSongs (): List<Song>
+searchSongs(String, String, String, int) : List<Song>
+getSongFromResultSet(ResultSet) : Song
PaymentDao
+setupDirectDeposit(User, DirectDeposit) : void
+SetupCreditCard(User, CreditCard) : void
+DebitUser(User, double) : void
+CreditUser(User, double) : void
+debitSong(User , int) : float

Factory Pattern to get the different user instances.


UserFactory
+ createUser(type): User
User
# userId : int
# userName : String
# isActive : boolean
# cellNumber : long
# sessionCreatedTime : Date
# email : String
+ loggedin : boolean
+ updatePayment(Payment): void
+ modifyActivation(isActive): void
+ sendNotification(email): void
+ isUserLoggeIn(): boolean

We can make use of strategy design pattern here and use makePayment as abstract method which can handle all the payments. Couldn't do it for current project.



What we learnt from the process :

Main takeaways :

- Coming up with the design for a project is quite an involved task which has to account for all the variations, efficient reuse of Classes, support for scalability, provision for probable replacement of the tech stack. In our honest opinion we all agree that having a well defined design for classes will solve majority of the project work.
- Albeit good design once we start implementing there may be technical difficulties which will make us reconsider our design.
- We all had a major learning curve while working with Spring Boot and developing a web service using it. It’s quite convenient to get a project started using it.
- Design patterns are really helpful. Even with prior knowledge of a design pattern, accommodating it in the code could be challenging.
- Team work.
- As and when we went about implementing functionalities we left it was quiet useful to take a break and update each other of progress. It gave us an idea about things which we can use, for instance usage of lombok annotations , user session etc.
- We have a sense of what all practices goes into developing a well designed application.