Project Part 3 : Refactoring

Team : Prasanna Srinivasachar
Vibhor Mishra
Aditya Thyagarajan

Title : Melody Milieu

Design Patterns Applied :

1. **Factory Design Pattern:**
   a. In our design, we have three types of users: Admin, Customer and Artist. We introduced a UserFactory class which controls the instantiation of these users and refers to the newly created object using a common interface. For instance, in the future if we wish to extend the type of users and want to introduce a primeCustomer/primeArtist class, then the code modification will be easier because the object creation is being handled by one centralized class i.e. the UserFactory class.
   b. Similarly, we have two types of messages that are handled in the system: text and email. We introduced MessageFactory class which controls the instantiation of these messages and refers to the newly created message object using common interface. It'll be helpful when we add new message types like messages on social media (FB/Twitter) etc.
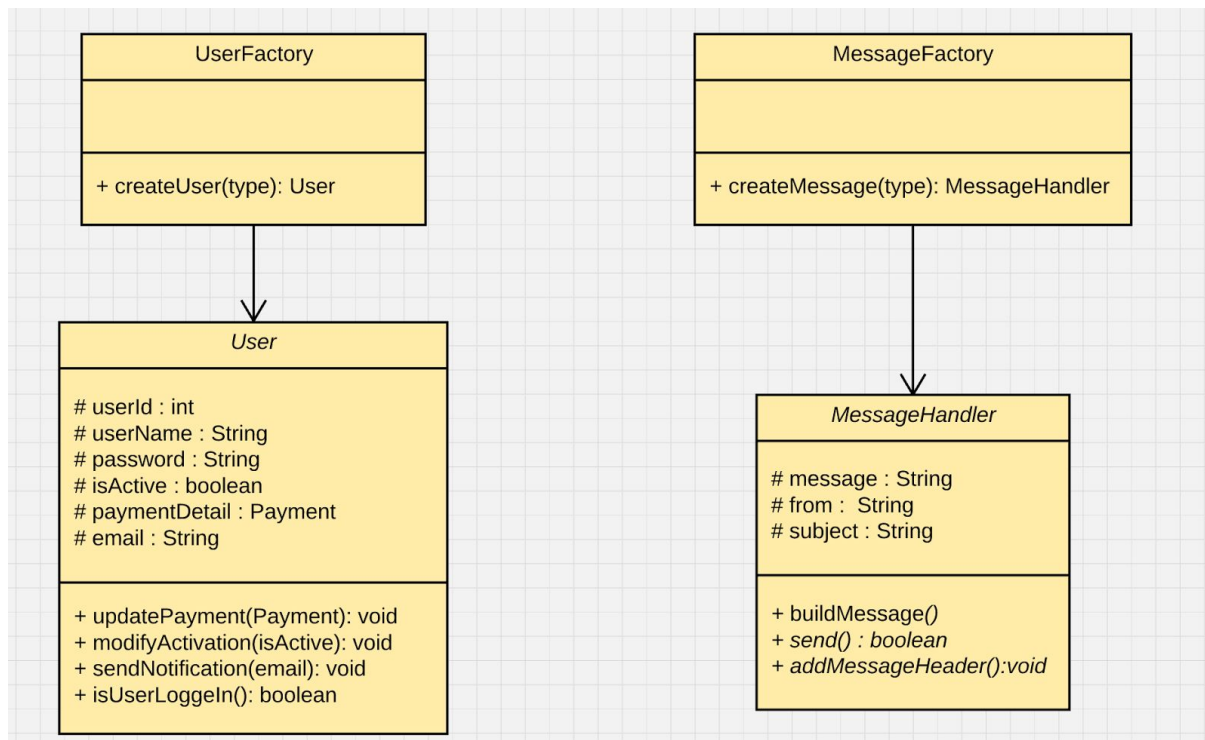
2. Template : We have currently two types of messages being handled through the MessageHandler class. The steps involving the creation of a message packet is similar except for the message header which contains information about the message grid or the mail grid to be used and the way it's sent . Other type of message notification like handling social media aspect can be added here with appropriate integration.
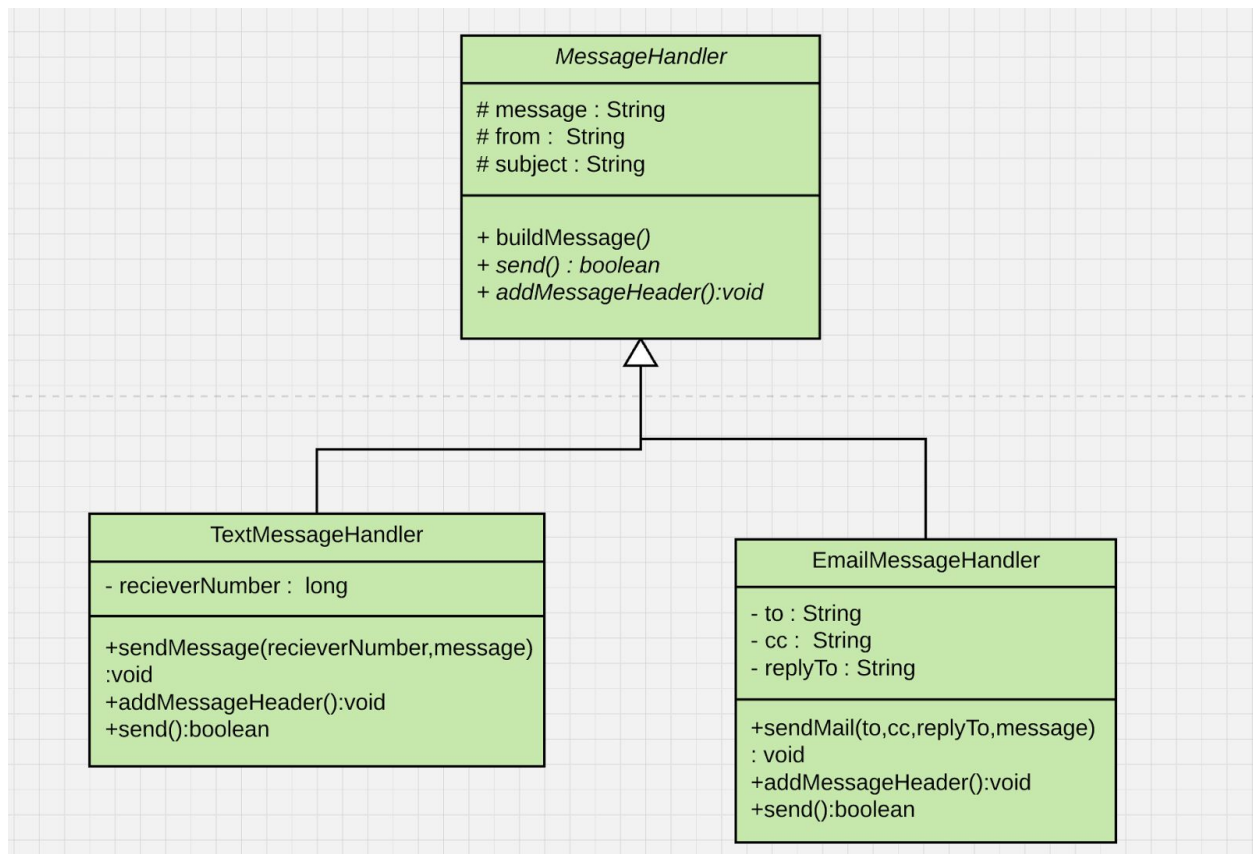
3. Singleton

We will be using the singleton design pattern for our Controller class as there should be only one instance of the controller subclasses which control the navigation aspect of the project.
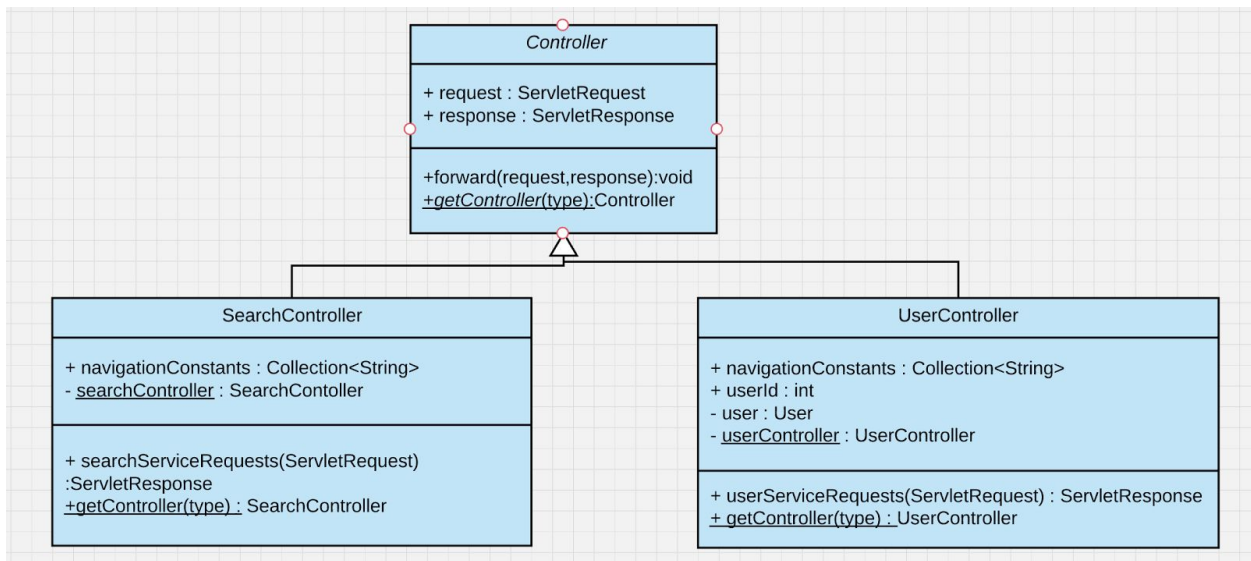
Steps of refactoring :

- Added static variables and methods to get the controller instance so that lazy initialization is supported.
- Send and addMessageHeader methods are added to support the template pattern
- Added two factory classes to support user and messageHandlers.

Class Diagram:

Old Part 2 Class Diagram :

MELODY MILIEU CLASS DIAGRAM | October 26, 2017

New class diagram:

MELODY MILIEU CLASS DIAGRAM | November 16, 2017

Legends
Factory Design Pattern	
Template Design Pattern	
Singleton Design Pattern	

UserFactory

+ createUser(type): User

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
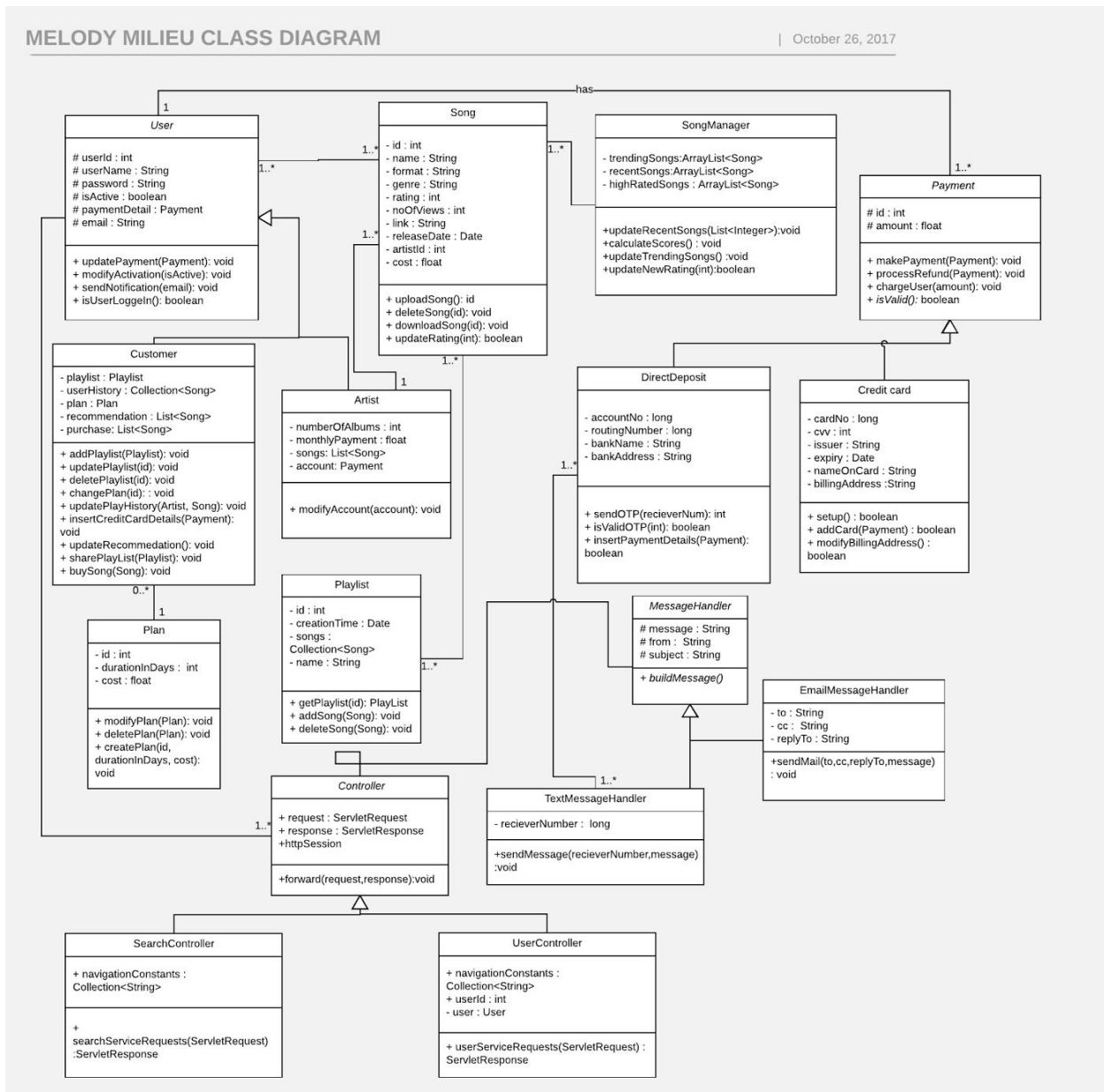- trendingSongs:ArrayList<Song>
- recentSongs:ArrayList<Song>
- highRatedSongs : ArrayList<Song>

+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

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

has
1
1..*
1..*
1..*
1..*
1
1..*
1
0..*
1
1..*
1..*
1..*