# EECS 4413: Building Ecommerce Systems


EECS 4413

Final Project – Mezzo: Music Store

https://mezzo-4413.mybluemix.net/

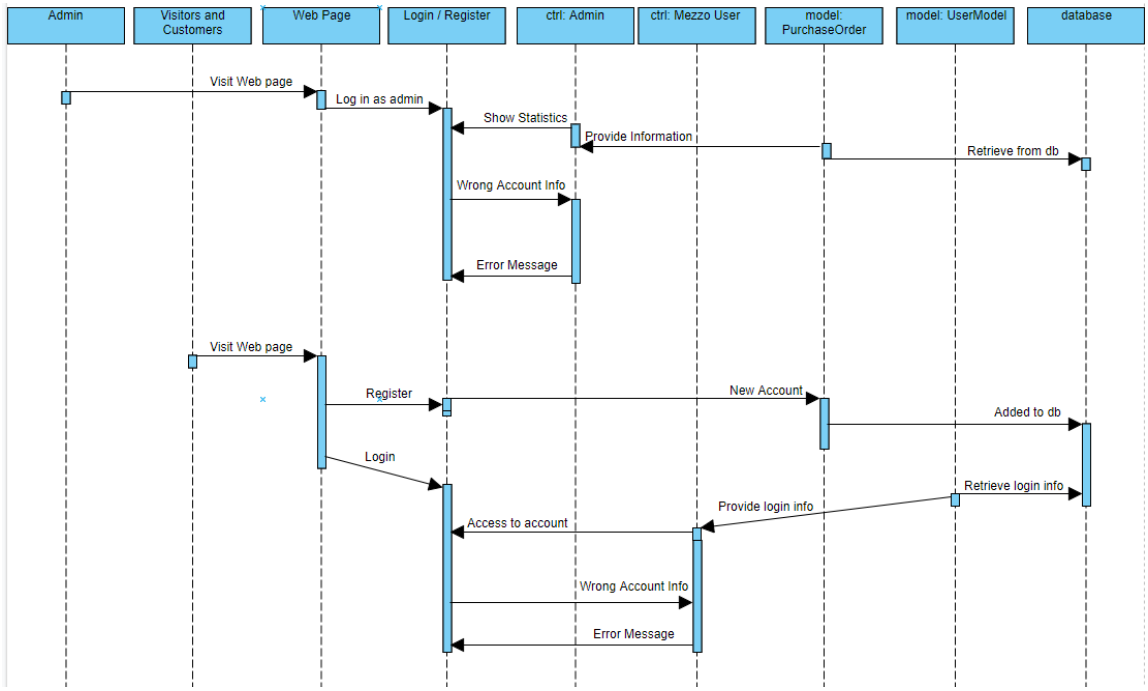
Design and Testing Document

Akinloluwa Adewale (215238231)
Aya Abu Allan (215250384)
Alan Tang (214717581)
Dong Jae Lee (214461560)
Professor: Marin Litou
Date: Monday, April 13th, 2020

# Table of Contents

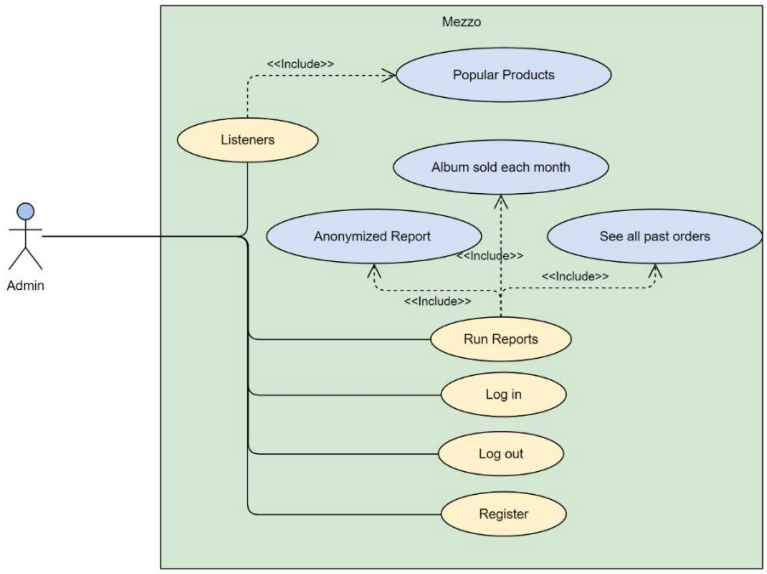
# Architecture

## Idealized Sequence Diagram

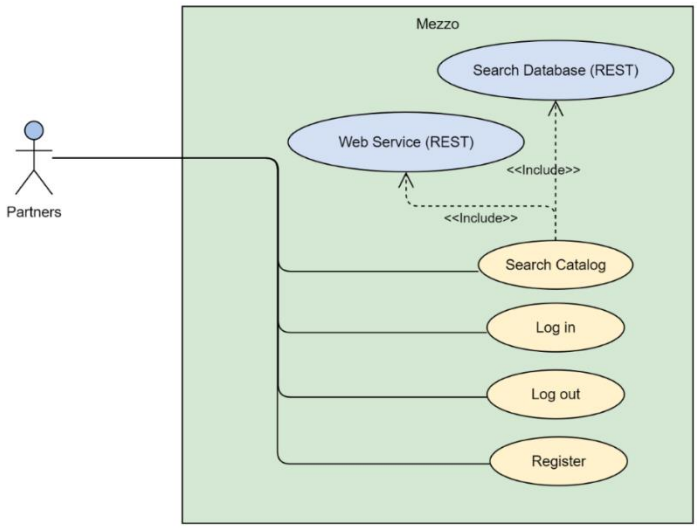



## Use Cases

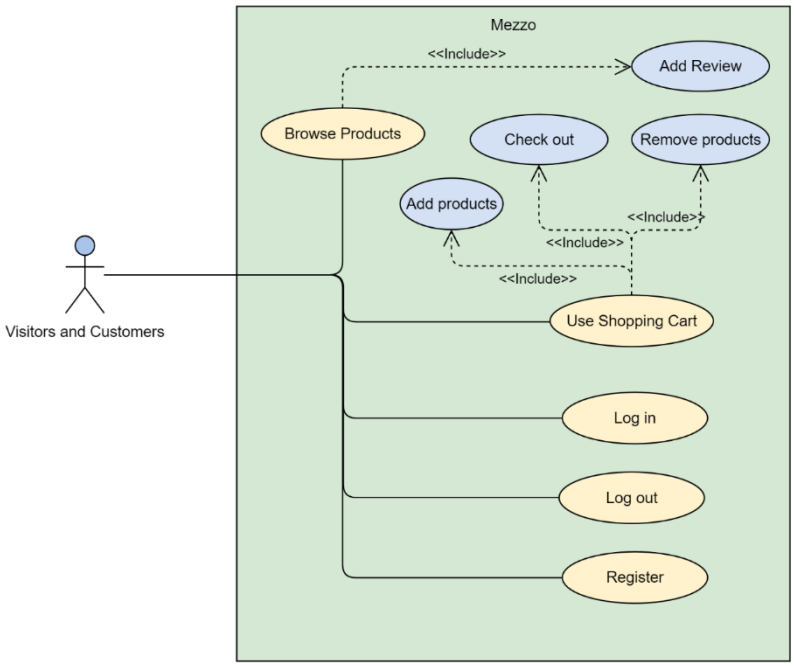
*Admin*



*Business Partners*




*Visitors and Customers*



# Implementation

A. Data Access Object

Our team opted for using a Data Access Object (DAO) design pattern integrated into MVC pattern for the project to facilitate communication between the application and the database. To realize this design pattern, we've created a number of DAO and Bean classes. The DAO classes are used in the model to extract information from the database that satisfies some required business logic. This information is returned to the model by the DAO using Bean objects. Using this structure that relies on persistence layering provides the model with the needed functionality without exposing the database. Here, it is evident that the DAO design pattern supports the single-responsibility principle as database navigation is no longer the model's responsibility which results in lower coupling. This also protects the model; if the database schema where to change, only the DAO's implementation would need to be adjusted. Our current database schema contains redundancies which is not good design. If we had more time to improve it, we could do so without having to alter the model. In that same vein, we would also finish parameterizing all our SQL queries to improve security which can be done in the DAOs. This use of the encapsulation design principle separates the persistence logic such that our application can safely react to any changes applied to the database.

B. Store

To start off, we will discuss the construction of our Mezzo shop which starts with the Shop controller. This class provides the user with album(s) information based on whether they searched up some parameter or if they chose a category. Upon selecting an album, the application will pass them to the product page that is observed by the ProductPage controller. Here, the selected album's info is displayed along with its rating and an option to add a rating given a user. Both of these controllers have a MusicStore model object instantiated. The Shop controller uses the MusicStore object to retrieve albums relevant to the user's search whereas the ProductPage controller uses it to retrieve a single album's information along with review options. This is possible due to the modular design of the model, it has different DAOs available to it that are relevant to the function it aims to provide. The MusicStore uses the AlbumDAO and ReviewDAO to access different parts of the database. Both DAOs build and return database information in their respective bean object or an encapsulated version. For example, the retrieveAlbumByCat(cat) method in the AlbumDAO returns a map object with the catergory as the key and the AlbumBean as the value. This is then used by the MusicStore model to push the results to the view in navigable format (after being called from the Shop controller).

C. User account

The user element is obviously crucial to an application like ours. We first define a user in the UserModel fundamental functionality is defined such as registerUser(..) and loginUser(..). This model class is often called in the MezzoUser controller. This controller is utilized at a logout, login, or register pages at the view. If all 3 flags are null, the controller redirects to a LoginReg page where the user is asked to log in or register. If the logout flag is on, MezzoUser redirects to the homepage. If the register flag is on, the controller calls on its UserModel object to invoke its registerUser(..) method. The UserModel checks if the parameters passed to the method are valid before calling on its UserDAO to enroll the new user into the database. To ensure success, we added a check in the controller to verify registration was successful by, again, calling on the UserModel to query the database using a UserDOA. If the signin flag was on instead,

the controller would behave similarly, calling the appropriate UserModel method. We've implemented a filter AuthFilter to ensure this login step is secure. In case of failure in either login or registration, the user is redirected to the LoginReg page. However, in case of success, we have defined a method populateUser(data) to parse the info returned from the model calls into the view. This data is in the form of a map with the user's last name as key and the associated ProfileBean as the value. This bean is also used in setting up the profile page in the Profile controller. That controller also has a retrieveProfileBilling(..) method used to add the users billing address to their profile page, considering the address schema is decoupled from the profile schema.

As mentioned in the last section, when at the product page, a user may add a review of the album. In our implementation, we did not instantiate a UserModel in here, instead we provided user data indirectly via the SessionManagement controller; see section x for more. The ProductPage uses its MusicStore object to call the putReview(aidInt, username, ratingInt, review) method which sanitizes the username since it wasn't obtained from a ProfileBean. It also sanitizes the review object passed to it and checks if all the passed values are valid. Lastly, the MusicStore object uses the ReviewDAO to inject the whole review safely into the database.

D. Order Process

Another function vital to the user other than logging in and registration is the process of shopping. The Checkout controller first retrieves the current user's username using the SessionManagement controller (more on this later). Then passes that username to a UserModel object and stores it in CheckoutProfileBean. This bean is used to retrieve the user's billing address and profile to and set them as attributes. The next stage is handled by the Pay controller. It first retrieves the current user and cart from the SessionManagement controller and check if the cart is valid. Upon success, the controller adds a new purchase order to the database via the PurchuseOrder model.

E. Analytics Page

An administrator is a type of user with special privileges. To maximize decoupling, we created a controller that specifically caters to the needs of such user. Some administrator privileges include adding a new album to the store, upgrading a user's privileges, and perform some interesting analytics. To add an album, the controller waits for the "add album" button to be engages and it simply calls on or MusicStore's addAlbum(..), it will check for valid parameters and pass it to the AlbumDao. Notice how we always pass a "0" to the album ID? This doesn't actually assign the album's ID to 0, it just always us to pass the other parameters on to the AlbumDAO which assigns the ID dynamically. If the "updateRoll" parameter isn't null, the admin controller forwards the username and desired privilege status to the UserModel object that passes it to the UserDAO as always. Both of the discussed functions forward the user to the "final" page.

A more interesting function is the analytics that administrators can generate. The first one displays the albums sold per month. We added a retrieveAlbumsPerMonth() method to the PODAO that queries all the items in the Purchase Order table and parses through each items' "PO_DATE" value to retrieve the month value using substring(4,6) function. This is an efficient and simple way of recovering the month value from the string-typed date. This method is called from the controller using the PurchaseOrder object and it returns an array of size 12, an index for each month.

The second analytical calculation our admin can perform is maintain a top 3 list. Typically, our model classes passes data on from a DAO as is; however, here we require some manipulation to be done to the data before it's pushed back to the controller to render the view. In the PurchaseOrder model, our getTopThree() calls the PODAO only to retrieve an album count in the form of a map with the album ID as key and quantity sold as value. The method then checks if the size of this map is less than 3, in which case, it calls on another method to return the most popular album. If not, then it creates 2 array lists, one containing the map's keyset (album ID), the other contains the values (quantity). A topThree string array is also created to store the results where the smaller the index, the more popular the item. To find the most popular item, we simply find the max in the second array list (most quantity sold), get its album ID and use that to retrieve the album name. The name is saved to the 0 index of topThree and its quantity is removed from the array list only to repeat the process twice more.

Lastly, an administrator can produce an anonymized report of users addresses and total purchases so far. This is another situation where the model had to perform more labour than typical. In the PurchaseOrder model, we first retrieve a map of usernames and their zipcodes from the AddressDAO. Then, we extract the keyset of usernames and zipcodes into their own arrays list as before. Next, we'll iterate through the usernames and retrieve their purchase count using PODAO. Finally, we create a hashmap and add the zipcode as key and purchase count as value as we loop through it.

F. Session Controller

The SessionManagement controller is a utility class that abstracts away the handling of data stored in the session scope. This allows for seamless bind and unbind operations on objects to session. Many contollers rely on SessionManagement such as Admin, Checkout, MezzoUser, Pay, ProductPage, and Profile. It is also useful in other circumstances such as when using AuthFilter and CartFilter. This controller provides those filters with bounded sessional data such as current user but also initializes needed objects if none are bounded as seen in the MezzoSessionListener.

G. Frontend

Frontend design and development was done with vanilla HTML. We made use of bootstrap libraries, jQuery and much more to ensure a uniform frontend experience across different viewing devices. Unfortunately, we were not able to investigate other frontend frameworks that may have been better in our E-Commerce store as it would have required us to learn new technologies in time we unfortunately did not have so we opted for the familiar and functional.

Api is the controller for the REST API that responsible for parsing results retrieved by model objects MusicStore and PurchaseOrder and converts them into various JSON files as needed (e.g. album document, error document). We were unable to investigate other web service APIs such as JAX-RS and stuck to a servlet based one due to being short on time.

Lastly, validation was intentionally implemented with HTML5 over JavaScript to reduce complexity and increase compatibility for users browsing without it. This reduces privacy issues while promoting standardized HTML5 features.

H. Security

As mentioned earlier, our security features are incomplete due to lack of time. To mitigate SQL injection attacks, we developed an authorization filter, AuthFilter, to limit access to privileges. We opted for this type of role-based access control despite it violating the principle of fail-safe default because we ran out of time and couldn't investigate other alternatives. This has also resulted in not all DAOs were parameterized. Had we had more time, we would've also parameterized the DAOs with complex query calls.

XSS protection was not implemented; and example of such attach is demonstrated on the Michael Jackson Thriller album review. CSRF protections were also not implemented leaving our callback parameter on login and forms vulnerable.

Below, we provide some test cases to our use cases:

Test case 1: *Admin* -

| *Test ID: 10* | *Step Detail* | *Expected Result* | *Actual Result* | *Pass/ Fail* |
|---|---|---|---|---|
| Step # | | | | |
| *1* | Navigate to https://mezzo-4413.mybluemix.net/shop | Site should open | Site opens | Pass |
| *2* | Click on Admin tab at the top | Site redirects to Register/ Login page | Site redirects to Register/ Login page | Pass |
| *3* | Login with client credentials | Site does not redirect and clears login fields | Site does not redirect and clears login fields | Pass |

Test case 2: *Business Partners*-

| *Test ID: 20* | *Step Detail* | *Expected Result* | *Actual Result* | *Pass/ Fail* |
|---|---|---|---|---|
| Step # | | | | |
| *1* | Navigate to https://mezzo-4413.mybluemix.net/shop | Site should open | Site opens | Pass |
| *2* | Click on Partner tab at the top | Site should redirects to Api tutorial page | Site redirects to Api tutorial page | Pass |
| *3* | Edit URL such that it resembles https://mezzo-4413.mybluemix.net/api/orders/?aid=1 | Should redirect to page showing: {"totalOrders":3,"orders":["20190607154533 2","201912041403527","202004010203634"]} | Redirects to page showing: {"totalOrders":3,"orders":["2019060715453 32","201912041403527","202004010203634"] } | Pass |

Test case 2: *Visitors and Customers*-

| *Test ID: 30* | *Step Detail* | *Expected Result* | *Actual Result* | *Pass/ Fail* |
|---|---|---|---|---|

| Step # | | | | |
|---|---|---|---|---|
| *1* | Navigate to https://mezzo-4413.mybluemix.net/shop | Site should open | Site opens | Pass |
| *2* | Select “Hip-Hop” category on the left-hand side | Should show Hip-Hop albums on same page | Shows Hip-Hop albums on same page | Pass |
| *3* | Hover over “Damn” by Kendrick Lamar and click “add to cart” | Should add album to cart and returns to home | Adds album to cart and returns to home | Pass |
| *4* | Click on cart icon the on “Proceed to checkout” | Should redirect to checkout page | Redirects to checkout page | Pass |
| *5* | Fill in shipping info and click “Place Order” | Should redirect to “Order Successfully Complete” page | Should redirect to “Order Successfully Complete” page | Pass |

# Contributions

### Group Work

With the COVID-19 pandemic beginning in the last month of the semester, group members were challenged by very difficult circumstances and living conditions. A few in-person meetings were conducted prior to the closure of the university which only provided general direction. After the closure of the university, our group members faced very difficult circumstances. One group member had a close family member hospitalized. Another was evicted. Despite these challenges, which caused significant lost work time, group members made every effort to collaborate on the project. Voice calls and screen-sharing sessions using Discord were performed often to assist other members. Impromptu pair-programming occurred during these sessions. Text-based standup meetings like those seen in a variety of development methodologies were conducted on Discord. A primitive ad-hoc version of Scrum was exercised, with the “Scrum-board" (to do list) posted on Discord, then on GitHub as issues. Version control was facilitated by GitHub. GitHub was used for communication alongside Discord, a popular gaming chat platform which is very similar in functionality to workplace communications platforms such as Slack and Microsoft Teams. Major changes were tested on a live database hosted on IBM Cloud before being pushed to the Git repository.

### Individual Contributions

## Akin

Akinloluwa Adewale’s contribution to the project include but are not limited to :

- Controllers
    - Admin (Author and contributor)
    - Mezzo (Author - Main controller)
    - Profile (Author)
    - MezzoUser (Author)

    - Profile (Author)
    - Shop (Author)
- Beans
    - AccountBean
    - AlbumBean
    - ProfileBean
- Filters
    - CartFilter (Contributor)
- DAO
    - AlbumDAO (Author)
    - UserDAO (Author)
- Models
    - MusicStore (Author)
    - UserModel (Author)
    - ShoppingCart (Contributor)
- Frontend
    - Creation, Design decisions and parsing of **ALL** site pages (Kinny - me)
        - Initial site template procured from colorlib, as well as codepen.io, w3schools and more and altered to current state by Akin
        - With Alan frequently handling parsing as well
    - CSS, JS etc. etc.

This member created the initial project and set the groundwork for the project structure, creating the java project, git repository and setting up the IBM cloud project, With the most design experience, this member took the responsibility of managing all front-end creation, parsing and design decisions. This member created the database file (see Proj.sql) and set up DB2 to be ready for use in the application. Member Alan also made several future changes to the database. This member was also consistently keeping team mates on track so as to reach the project goal on time and exceed project requirements.

## Aya

- Controllers
    - Admin (Contributor)
- DAO
    - AddressDAO
    - PODAO (Contributor)
    - POItemDAO (Contributor)
- Models
    - PurchaseOrder (Contributor)
- Report
    - Implementation sec. A-H

My initial contribution was meant to be the Admin controller and its associated elements. However, I ran into a persistant issue where any call I made to the PurchaseOrder model returned a Null Pointer Exception. I spent copious amount of time attempting to resolve the bug to no avail. Meanwhile, my teammates were able to progress to new tasks. I've discussed it with my teammates and they all weighed in equally to help me debug via calls, discord chat, and by pushing my work to a new branch for them to try out. We didn't have much luck with it until Dong ran it again and noticed a pool

exhaustion error. I never got that error for some reason and at this point we didn't have time to find out why and try to produce the error. Therefore, he took over and completed the implementation as moved on to the report. It turns out the DAO was broken and I was debugging in my model. To compensate I took on a decent chunk of the report material.

## Dong Jae

Dong Jae Lee's Contribution

- Beans (bean) – edited accordingly and added documentation
    - AccountBean
    - AddressBean
    - AlbumBean
    - POBean
    - POItemBean
    - ProfileBean
- Controllers (ctrl)
    - Admin (DiffAdmin) – Backend part of Analytics
- Data Access Object (DAO) – Implementation / bug fixes / documentation
    - AddressDAO
    - AlbumDAO
    - PODAO
    - POItemDAO
- Models (model) – Implementation / bug fixes / documentation
    - PurchaseOrder
    - ShoppingCart

Contribution to the beans and models - ensured that they were properly set up / necessary accessors were available for usage and proper documentation was added. Created multiple DAO methods for proper data retrieval from the database and data relay to the controllers. Many bug fixes in the data access objects. Responsible for the admin data analytics on the admin page. Completed diagrams and other parts on the report.

## Alan

The work performed by group member Alan Tang was as follows:

- Controllers
    - SessionManagement
    - Checkout
    - Pay
    - MezzoUser (contributor)
        - Integration with AuthFilter
        - Bug fixes
    - Profile
    - ProductPage
    - Api (REST endpoints)
- Beans
    - ReviewBean

    - CartItemBean
    - CheckoutProfileBean
- Filters
    - CartFilter
    - AuthFilter
- Listeners
    - MezzoSessionListener
- DAO
    - ReviewDAO
    - UserDAO (contributor)
        - Bug fixes
- Models
    - MusicStore (contributor)
        - Review functionality
    - UserModel (contributor)
        - Cart functionality
        - Checkout bug fixes
    - ShoppingCart (contributor)
        - Integration with CartFilter
- Frontend
    - Integration with ProductPage controller
    - Minor controller integration bug fixes
    - General suggestions of interface between controllers and frontend
- Other
    - Conversion of project to Maven dependency management

Direct communication via Discord message or voice call via Discord was performed between this member and the rest of the group to maintain awareness of the each other's work. Careful documentation and reading of others' code, facilitated by commits to GitHub, was essential to maintaining awareness when communication was not possible.

## Signatures

Akinloluwa Adewale

Aya

Dong Jae Lee

Alan