## Discussion

This MovieZone System is divided into two part: General System and Admin System. Each system includes three components: controller, view and model. The view component display all of data in need to the interface. JavaScript will get the request and parameters from interface then post to controller component. The controller based on the user request invokes the appropriate functionality from the model class, which consist of a variety of method to provide different functionality. And model handles adapter class which is allowed to operate the database system. Ultimately, Ajax get request data from controller, and then update content in specified div. Some good working in this assignment are as follows.

* User’s request is caught by JavaScript, and then posted to server after data processing, which can reduce the burden on the server.
* AJax post the instructions and parameters to server, which is faster than submitting a whole form data.
* The controller uses switch to read the instructions and get parameters.
* The controller calls modules and view components to complete operation in need.
* The controller request services of the model class function to services the client request by invoking the function with appropriate parameters.
* The controller returns requested data to Ajax, which updates specifies container on page.
* The model class deal with the database adapter classes to retrieve the needed data and sends it back to the controller
* The controller class passed the retrieve data or error if no data available to the view class function
* The view class formats the data and display the results to the end user.

The link of my assignment is: http://infotech.scu.edu.au/~jtang13/assignment2

## Functional requirements

* Select list of all the movies from database system
* Select list of new released movies
* Filter movie by actor, by director, by genre, by classification
* Create new movie details
* Basic Information of new movie (Movie Title, Release Year, Tag line, Plot, Movie Poster, Director, Studio, Genre, Classification)
* Movie stars and co-stars(Stars List)
* Stock information(Rental Period, Rental Price, Purchased Price, in-stock items, rented)
* Edit movie details
* Movie Information
* Stock Information
* Delete movie details
* Select movie(s) to rent or purchase by user
* Create new member
* Edit member details
* Member ID: 34
* Contact detail
* Magazine
* Delete member
* Log in/out from admin panel
* Field validation of input data
* Display dynamical movie ads on the home page
* Contact page showing Google map

There are three groups of functions based on the functional requirement.

* Movie details processing functions:
  + Select all of new movie details
  + Filters movies detail based on different attribute
  + Edit movie details
  + Delete movie based on movie ID
* Member detail processing functions:
  + Add new member details
  + Search member detail based on user name and real name
  + Edit member details
  + Delete menber
* User verification functions
  + Admin login/out functions
  + Normal user login/out functions
  + Select movies to rent or purchase
  + Check out movies

## Non-functional requirements:

* Two column design at minimum
* Display new releases in front page
* Use separate CSS file for styling
* This website must have a good structure of header and footer sections
* Use JavaScript and Ajax to support the switching of html pages
* Movies detail are stored in a SQL database
* The external CSS file must be used to design the whole website
* This website need to support multiple screen sizes
* Set up two views for searching data in database system easily
* A side bar displaying new released movies
* The movie poster pictures are stored in a separate file
* The system uses Error Notice messages

## High-level design

### high_level_design The high-level design diagram

*Figure 1 high-level design diagram*

### The high-level design description

There are three types of visitors including administrator, member, normal User. They visit the site represent three different levels of permissions. User and administrator logging into the website will need to enter the request in the input/output interface component. The user’s request will be handled by JavaScript function and post request to the controller component. The controller examines the service required for the request and then invokes one or more components of model and view to provide the service. The model are allowed to handle the database adapter compomponent to operate database system. In database adapter, functions of connection, retrieves and insert data will satisfy the request. As a result, the result data will be package as an object and post to the controller. The view class will get the result date from controller and format them in html page in a user-friendly way.

## Low-level design

#### Information about modules:

Module 1: Show movies model - **selectAllMovies()**

|  |
| --- |
| * POST\_Input OUTPUT * GET\_Input |
| Movie detail output Module:   * movie\_ID: int(10) * title: String(45) * tagline: String(128) * plot: String(256) * thumb\_Path: String(40) * star1: String(128) * star2: String(128) * star3: String(128) * costar1: String(128) * costar2: String(128) * costar3: String(128) * director: String(128) * studio String(128) * genre: String(128) * classification: String(128) * rental\_period: String(128) * year: int(4) * DVD\_retal\_price: float * DVD\_purchase\_price: float * numDVD: int(3) * numDVDout: int(3) * BluRay\_rental\_price: float * BluRay\_purchase\_price: floar * numBluRay: int(3) * numBluRayOut: int(3) |
| * selectAllMovies() INPUT |

Module 2: Filter movie by criteria model - **filterMovies()**

|  |
| --- |
| * POST\_Input OUTPUT * GET\_Input |
| MovieZoneModule:   * movie\_ID: int(10) * title: String(45) * tagline: String(128) * plot: String(256) * thumb\_Path: String(40) * star1: String(128) * star2: String(128) * star3: String(128) * costar1: String(128) * costar2: String(128) * costar3: String(128) * director: String(128) * studio String(128) * genre: String(128) * classification: String(128) * rental\_period: String(128) * year: int(4) * DVD\_retal\_price: float * DVD\_purchase\_price: float * numDVD: int(3) * numDVDout: int(3) * BluRay\_rental\_price: float * BluRay\_purchase\_price: floar * numBluRay: int(3) * numBluRayOut: int(3) |
| * filterMovies() INPUT |

Module 3: Add a movie model - addMovie()

|  |
| --- |
| * POST\_Input INPUT * GET\_Input |
| MovieZoneAdminModule:   * movie\_ID: int(10) * title: String(45) * tagline: String(128) * plot: String(256) * thumb\_Path: String(40) * star1: String(128) * star2: String(128) * star3: String(128) * costar1: String(128) * costar2: String(128) * costar3: String(128) * director: String(128) * studio String(128) * genre: String(128) * classification: String(128) * rental\_period: String(128) * year: int(4) * DVD\_retal\_price: float * DVD\_purchase\_price: float * numDVD: int(3) * numDVDout: int(3) * BluRay\_rental\_price: float * BluRay\_purchase\_price: floar * numBluRay: int(3) * numBluRayOut: int(3) |
| * addMovies() OUTPUT   Save to database |

Module 4: Edit movie detail model - editMovies()

|  |
| --- |
| * POST\_Input INPUT * GET\_Input |
| MovieZoneAdminModule:   * movie\_ID: int(10) * title: String(45) * tagline: String(128) * plot: String(256) * thumb\_Path: String(40) * star1: String(128) * star2: String(128) * star3: String(128) * costar1: String(128) * costar2: String(128) * costar3: String(128) * director: String(128) * studio String(128) * genre: String(128) * classification: String(128) * rental\_period: String(128) * year: int(4) * DVD\_retal\_price: float * DVD\_purchase\_price: float * numDVD: int(3) * numDVDout: int(3) * BluRay\_rental\_price: float * BluRay\_purchase\_price: floar * numBluRay: int(3) * numBluRayOut: int(3) |
| * editMovies() OUTPUT   Save to database |

Module 5: Delete movie model - deleteMovies()

|  |
| --- |
| * POST\_Input INPUT * GET\_Input |
| MovieZoneAdminModule:   * movie\_ID: int(10) |
| * deleteMovies() OUTPUT |

Module 6: Create member model- createMember() //Admin function

|  |
| --- |
| * POST\_Input INPUT * GET\_Input |
| MenberModule:   * menber\_ID: int(10) * surname: String(128) * other\_name: String(128) * contact\_method: String(10) * email: String(40) * mobile: String(40) * landline: String(40) * magazine: int(1) * street: String(40) * suburb: String(40) * postcode: int(4) * username: String(10) * password: String(10) * occupation: String(20) * join\_date: date |
| * createMenber() OUTPUT   SaveToDatabase |

Module 7: Edit member model - editMember() //Admin function

|  |
| --- |
| * POST\_Input INPUT * GET\_Input |
| MenberModule:   * menber\_ID: int(10) * surname: String(128) * other\_name: String(128) * contact\_method: String(10) * email: String(40) * mobile: String(40) * landline: String(40) * magazine: int(1) * street: String(40) * suburb: String(40) * postcode: int(4) * username: String(10) * password: String(10) * occupation: String(20) * join\_date: date |
| * updateMenber() OUTPUT   SaveToDatabase |

Module 8: Delete member model - deleteMember() //Admin function

|  |
| --- |
| * POST\_Input INPUT * GET\_Input |
| MenberModule:   * menber\_ID: int(10) |
| * deleteMenber() OUTPUT   Delete a member from Database |

Module 9: check out movies DVD model - checkoutItems() //user function

|  |
| --- |
| * POST\_Input OUTPUT * GET\_Input |
| MovieZoneAdminModule:   * movie\_ID: int(10) * title: String(45) * tagline: String(128) * year: int(4) * numDVD: int(3) |
| * showCheckoutItems() INPUT   Output the checkout items to webpages |

#### Information about controller

Controller 1: MovieZoneController - loadLeftNavPanel()

|  |
| --- |
| User accesses web page INPUT |
| Call leftNavpanel() method in View class |
| leftNavPanel( ) OUTPUT |

Controller 2: MovieZoneController - loadTopNavPanel()

|  |
| --- |
| User accesses web page INPUT |
| Call topNavpanel() method in View class |
| topNavPanel( ) OUTPUT |

If the result of actors, directors, genres, classification from model class are existed, the method will show the result in list through view class.

Controller 3: MovieZoneController - handleSelectAllMovieRequest( )

|  |
| --- |
| POST\_Input INPUT  GET\_Input |
| Call selectAllMovies() in Model  IF the result of movie is not null THEN  showMovies( )  ELSE  showError( ) |
| handleSelectAllMovieRequest() OUTPUT |

If the result of movie detail is existed, the method will show the result in list through view class. Otherwise, the class will show the error message.

Controller 3: MovieZoneController - handleFilterMovieRequest( )

|  |
| --- |
| REQUEST\_Input INPUT |
| IF input not empty  Assign criteria id of corresponding type to variable  Call filterMovies() in the Model  IF movies not empty  Call showMovies( ) in View  ELSE  Call showError( ) in View |
| showMovies( ) OUTPUT |

Controller 4: MovieZoneController - handleAddMovieRequest( )//for admin

|  |
| --- |
| REQUEST\_Input INPUT |
| Create an array that contains data of a movie : keys[]  Create a null array to contain keys[]: moviedata[]  FOREACH data attribute in keys[]  IF the keys[] array not null  Assign keys to moviedata[ ]  ELSE  Call showError( )  Call addMovie( ) in Model  IF output null  Call getError( ) in Model  IF error not empty  Call showError( ) in View |
| Save to the database OUTPUT |

Controller 5: MovieZoneController - handleEditMovieRequest( )//for admin

|  |
| --- |
| REQUEST\_Input INPUT |
| Create an array that contains data of a movie : keys[]  Create a null array to contain keys[]: moviedata[]  FOREACH data attribute in keys[]  IF the keys[] array not null  Assign keys to moviedata[ ]  ELSE  Call showError( )  Call addMovie( ) in Model  IF output null  Call getError( ) in Model  IF error not empty  Call showError( ) in View |
| * SaveToLocalDatabase * SaveToInfoTechServer OUTPUT |

Controller 6: MovieZoneController - handleDeleteMovieRequest( )

|  |
| --- |
| POST\_Input  GET\_Input INPUT |
| Create an array that contains data of a movie : keys[]  Create a null array to contain keys[]: moviedata[]  FOREACH data attribute in keys[]  IF the keys[] array not null  Assign keys to moviedata[ ]  ELSE  Call showError( )  Call deleteMovie( ) in Model  IF output null  Call getError( ) in Model  IF error not empty  Call showError( ) in View |
| * SaveToLocalDatabase * SaveToInfoTechServer OUTPUT |

Controller 7: MovieZoneController - handleCreateMemberRequest( )

|  |
| --- |
| POST\_Input  GET\_Input INPUT |
| Create an array that contains data of a member: keys[]  Create a null array to contain keys[]: memberdata[]  FOREACH data attribute in keys[]  IF the keys[] array not null  Assign keys to memberdata[ ]  ELSE  Call showError( )  Call createMember( ) in Model  IF output null  Call getError( ) in Model  IF error not empty  Call showError( ) in View |
| * SaveToLocalDatabase * SaveToInfoTechServer OUTPUT |

Controller 8: MovieZoneController - handleEditMemberRequest( )

|  |
| --- |
| POST\_Input  GET\_Input INPUT |
| Create an array that contains data of a member: keys[]  Create a null array to contain keys[]: memberdata[]  FOREACH data attribute in keys[]  IF the keys[] array not null  Assign keys to memberdata[ ]  ELSE  Call showError( )  Call editMember( ) in Model  IF output null  Call getError( ) in Model  IF error not empty  Call showError( ) in View |
| * SaveToLocalDatabase * SaveToInfoTechServer OUTPUT |

Controller 9: MovieZoneController - handleDeleteMemberRequest( )

|  |
| --- |
| POST\_Input  GET\_Input INPUT |
| Create an array that contains data of a member: keys[]  Create a null array to contain keys[]: memberdata[]  FOREACH data attribute in keys[]  IF the keys[] array not null  Assign keys to memberdata[ ]  ELSE  Call showError( )  Call deleteMember( ) in Model  IF output null  Call getError( ) in Model  IF error not empty  Call showError( ) in View |
| * SaveToLocalDatabase * SaveToInfoTechServer OUTPUT |

Controller 10: MovieZoneController - handleCheckoutRequest( )

|  |
| --- |
| REQUEST\_Input INPUT |
| Create an array for selected movies: selected\_movies  Or retrieve it from session storage  Get movie id by REQUEST method  Convert selected\_movies to string  IF selected\_movies not empty  Set movie id into array selected\_movies and assign it the value of 1  ELSE  Unselect the movie by removing the id  Save to SESSION for next time access |
| showMovies( ) OUTPUT |

#### Information about view

Module 1: View - leftNavPanel( )

|  |
| --- |
| POST\_Input  GET\_Input INPUT |
| Print leftnav html file |
| showMovies( ) OUTPUT |

Module 2: View - showMovies( )

|  |
| --- |
| POST\_Input  GET\_Input INPUT |
| IF an array of movies not empty  printMovieHtml(movie) |
| showMovies( ) OUTPUT |

Module 3: View - showError( )

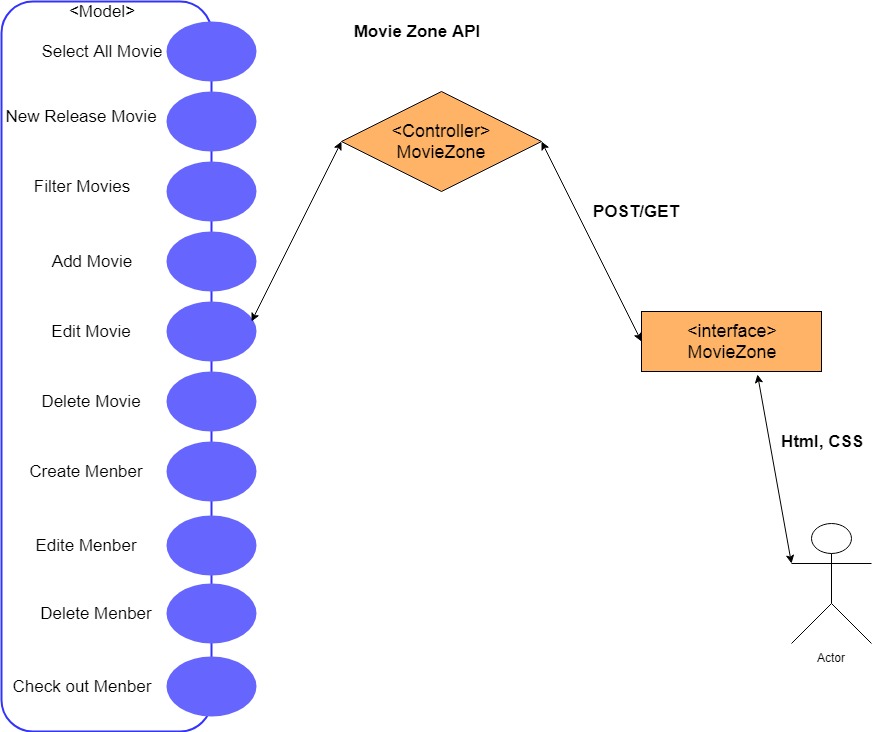
|  |
| --- |
| POST\_Input  GET\_Input INPUT |
| Print Error: |
| showMovies( ) OUTPUT |

### The low-level design description

The movie zone website includes three components: controller, view and model. The view component display all of data in need to the interface. JavaScript will get the request and parameters from interface then post to controller component. The controller based on the user request invokes the appropriate functionality from the model class, which consist of a variety of method to provide different functionality. And model handles adapter class which is allowed to operate the database system. The controller component can request to create different view. The general working flows are as follows.

1. User send a request using browser to the server
2. JavaScript post the request and parameters to controller
3. The controller reads the request and parameters
4. The controller detects the request command
5. The controller request services of the model class function to services the client request by invoking the function with appropriate parameters
6. The model class deal with the database adapter classes to retrieve the needed data and sends it back to the controller
7. The controller class passed the retrieve data or error if no data available to the view class function
8. The view class formats the data and display the results to the end user.

## User case description



*Figure 2 Use Case Diagram*

As above, the use case diagram in figure 2 describes all of requirements of movie zone website, including searching particular movies by different movie detail, edit movie detail, initiating member accessing the website. And then, member can check out the movies they selected. For instance, in the actor filter, users are allowed to select any actors in the list and get the result in need. In output processing, data retrieval are managed by the controller, model and view components.