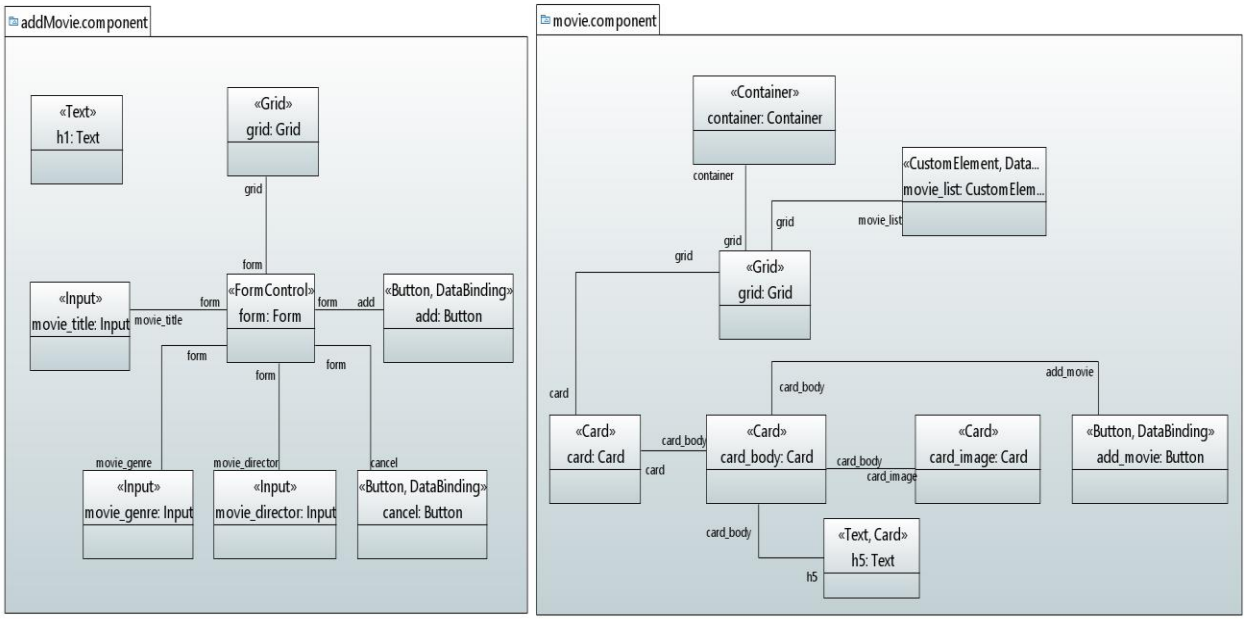
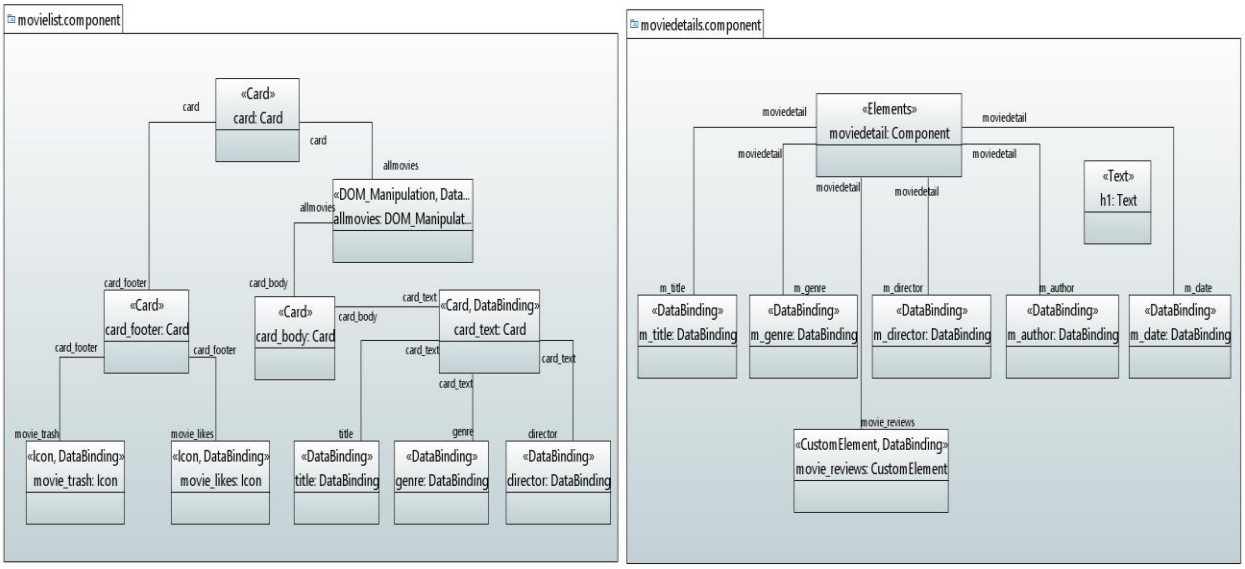
Movies Application Case Study

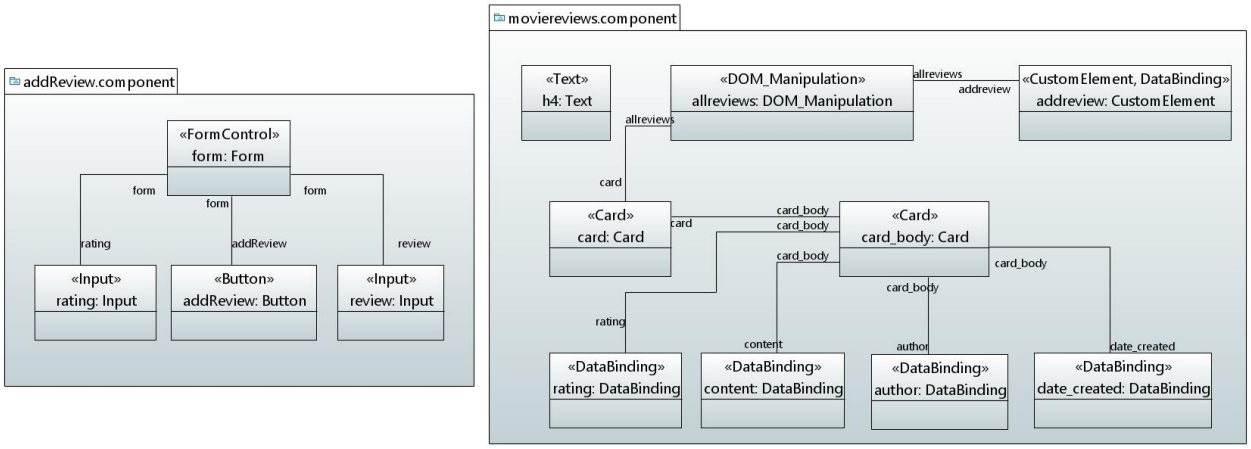
User Interface: The modeling of the Movies Application user interface is shown in the object diagram.



AddMovie and Movie Component



MovieList and MovieDetails Component



AddReview and MovieReviews Component

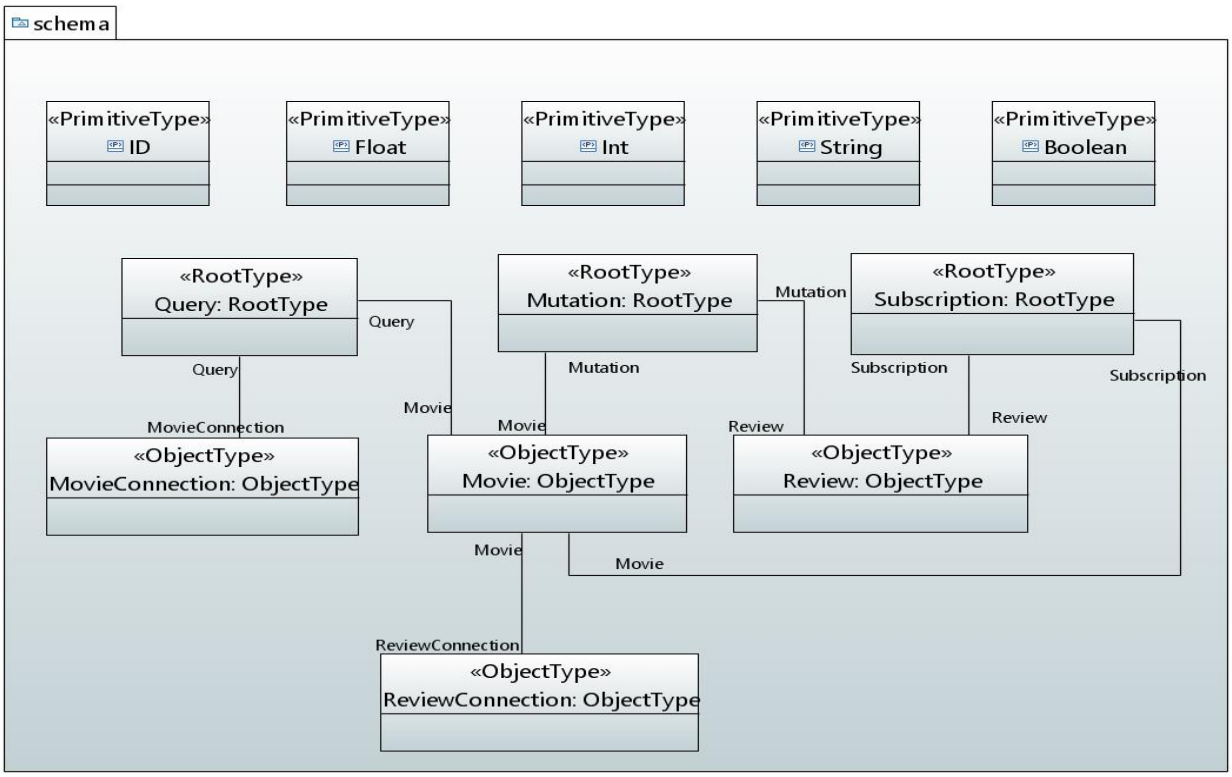
The AddMovie component contains container, grid, form, input, text and button elements, therefore Grid, FormContol, Input, Text and Button stereotypes are mapped to them for adding the elements with layout and styling. DataBinding stereotype is applied to bind event to the button. The Movie Component contains different elements such as card, text, button and custom element. Card stereotype is mapped to the card element that displays heading and image on card. Text stereotype is applied to heading of the text that take values for the heading. Button stereotype is applied to create an event button and DataBinding stereotype is applied to bind event to the button.

The MovieList component contains card that displays data taken from the AddMovie component. Therefore, Card stereotype is applied to card element that adds card to the user interface. The values of data on the card is retrieved through DOM_Manipulation and DataBinding stereotypes. Icon stereotype is mapped to the icon element that display icon along with data.

The MovieDetails Component displays the data of the created movie. Elements stereotype is applied to add some styling to component. DataBinding stereotype is applied to take values for different binding types. CustomEement stereotype is applied to add the custom element and attach data to it through DataBinding stereotype. Text stereotype is applied to take the heading text.

The AddReview component contains form, Input and button elements. So, FormControl, Input and Button stereotypes are applied to these elements that take values to create a user interface. Similarly, MovieReviews Component display reviews and rating on the movie by taking values through Text, DOM_Manipulation, Card and DataBinding stereotypes. CustomElement stereotype is applied to add custom element to movie reviews component.

Backend GraphQL API Schema: The model of the backend GraphQL API schema for Movies Application is shown below.



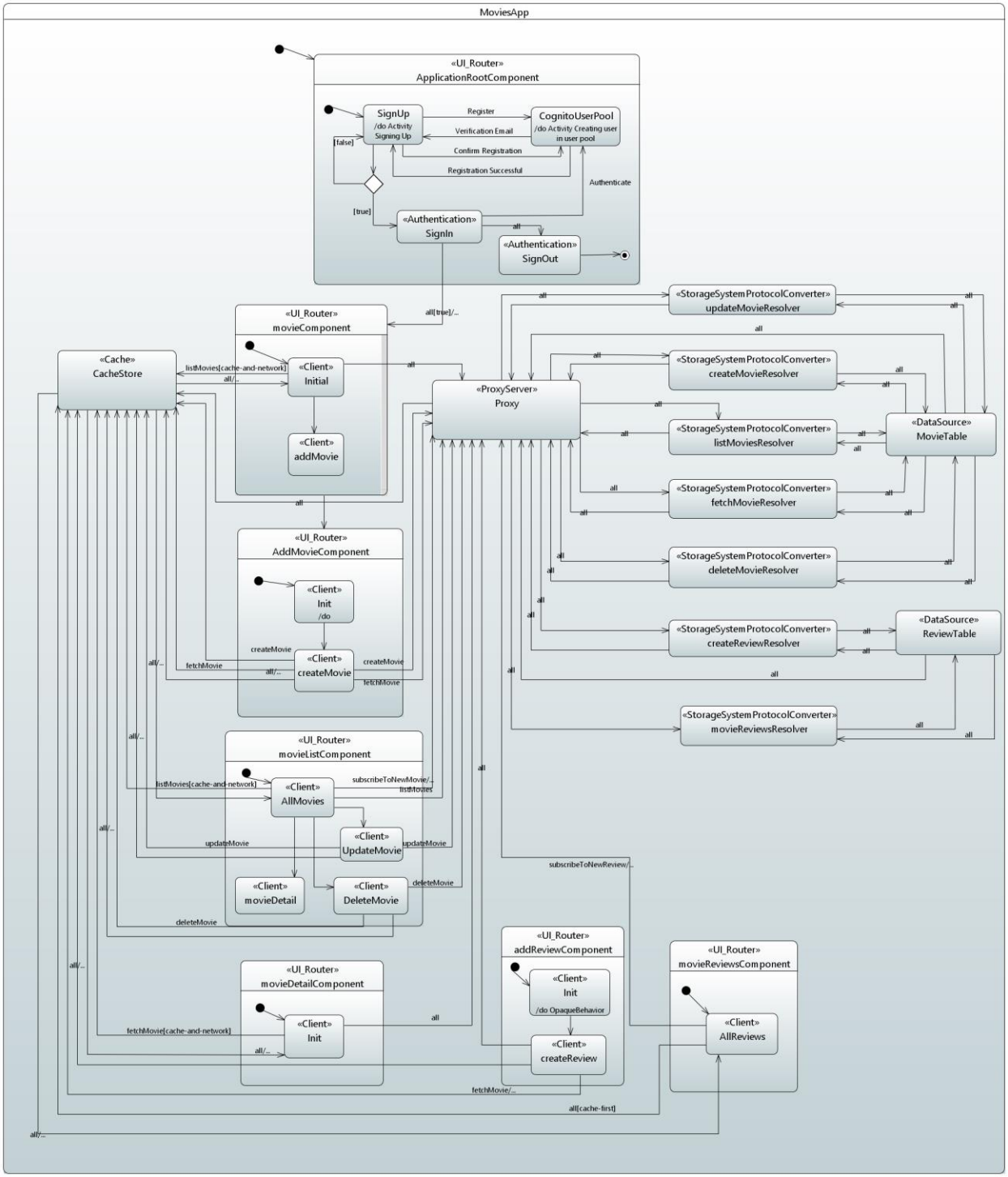
API Schema for Movies Application

RootType stereotype is applied to define the Root Operations. For this case study, the Operations include createMovie, createReview, updateMovie, listMovies etc. ObjectType stereotype is applied to take the field values for the Movie, MovieConnection, Review and ReviewConnection object types. TypeModifier stereotype is applied to the fields defined in Movie, MovieConnection, Review and ReviewConnection types and to the operations defined in the RootType.

Behavioral Model of Movies Application: The behavioral model of the Movies Application is shown in a state machine diagram. The application starts with user authentication. After authentication, the router navigates to the MovieComponent state and activates the initial state. Client stereotype is mapped to define the variables and operations used in this state. After defining the operations and variables, Request_Data stereotype is applied to listMovies request to fetch all the movies data from the cache and network. If the movies data is already in cache, the user interface is updated with the data. ListmoviesResolver state fetch all the movies from the MoviesTable through StorageSystemProtocolConverter stereotype.

In addMovie state, Client stereotype is mapped to navigate the router to the AddMovieComponent state. In AddMovieComponent state, the initial state is activated that calls the current session of authentication through Client stereotype. When the createMovie state is active, Client stereotype specifies the data fields for creating a new movie. Moreover, Request_Data stereotype is applied to take data fields for createMovie request. Subsequently, the proxy validates the request and moves to the createMovieResolver state that add movie details in the MoviesTable through StorageSystemProtocolConverter stereotype. Once a new movie is created in the MoviesTable,

new movie data is pushed to the clients that ultimately updates the user interface. While offline, an instant UI is created, and the request is saved. The request is automatically sent to the proxy when the network is available.



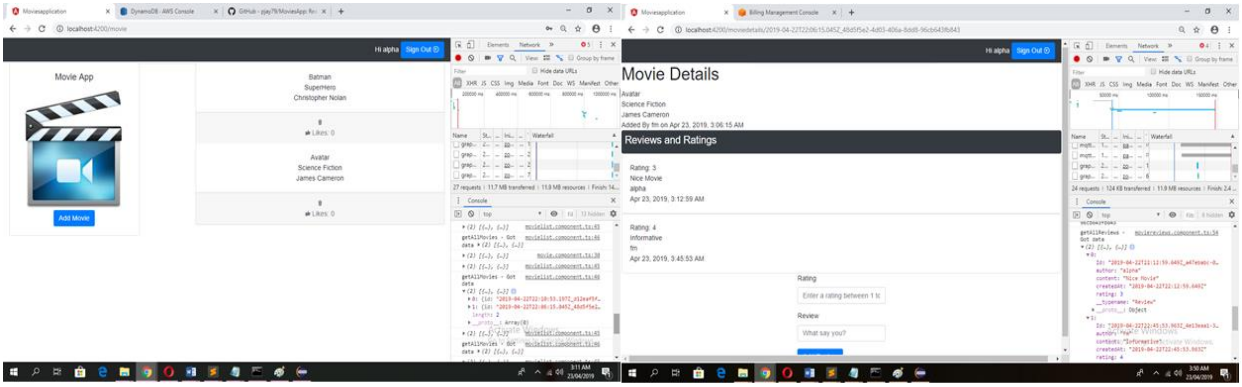
Behavioral Model of Movies Application

In movieListComponent state, the listMovies state get all the movies from cache and network. The proxy validates the request and listMovieResolver state fetch the data from MoviesTable through StorageSystemProtocolConverter stereotype. To update the movie, the proxy receives the request and updateMovieResolver state updates the movie data in the MoviesTable through StorageSystemProtocolConverter stereotype. Similarly, the deleteMovie state deletes the selected movie from the list of movies.

In MovieDetailsComponent state, Client stereotype is applied to the init state that takes the values to fetch the selected movie. A fetchMovie request is made to the proxy. After request validation, the fetchMovieResolver state is activated that fetch movie data from the MoviesTable through StorageSystemProtocolConverter stereotype. The response is then returned, and cache store is updated.

In AddReviewComponent state, upon activating the addReview state, addReview request is made to the proxy. Request_Data stereotype takes the data fields for addReview request. Then the proxy validates and processes the request and activates the addReviewResolver state to add review to the movie through StorageTransitionProtocolConverter stereotype. As a result, the movie review is created in the ReviewTable. Once a new review is created, a subscription is invoked that subscribe reviews to all the subscribers and updates the cache store. In the movieReviewsComponent state, the getAllReviews state is activated and all the reviews are fetched. Similarly, the movieReviewsResolver state return all the reviews associated with an individual movie.

The movies application after deployment is shown below. The data source used in this case study is Amazon DynamoDB.



Movies Application After Deployment