Natflix-Backend

Table of Contents

1. About

Natflix is a subscription based streaming device that allows all members to watch Tv shows, series and movies . As the pandemic has led to an increase in the demand of our streaming services, we need to expand the range of our series, movies and documentaries , which means that we need a newer, more secure and scalable web based application. This application is developed by using java Spring boot and REST APIs to make it fully working website to stream media.

2. Project Overview

Natflix provides a frontend web page designed to meet our needs ,and we shall start from it. The project is developed by React Typescript. Here we should configure our backend to communicate with the frontend to get the webpage that offers a service to stream videos to the customers and manage the resources. The frontend also provides us the data about the videos. The data of each video is the title , URL, thumbnail, banner and the type of content like series, movies and documentaries.

In the backend project, there are 2 roles. They are:

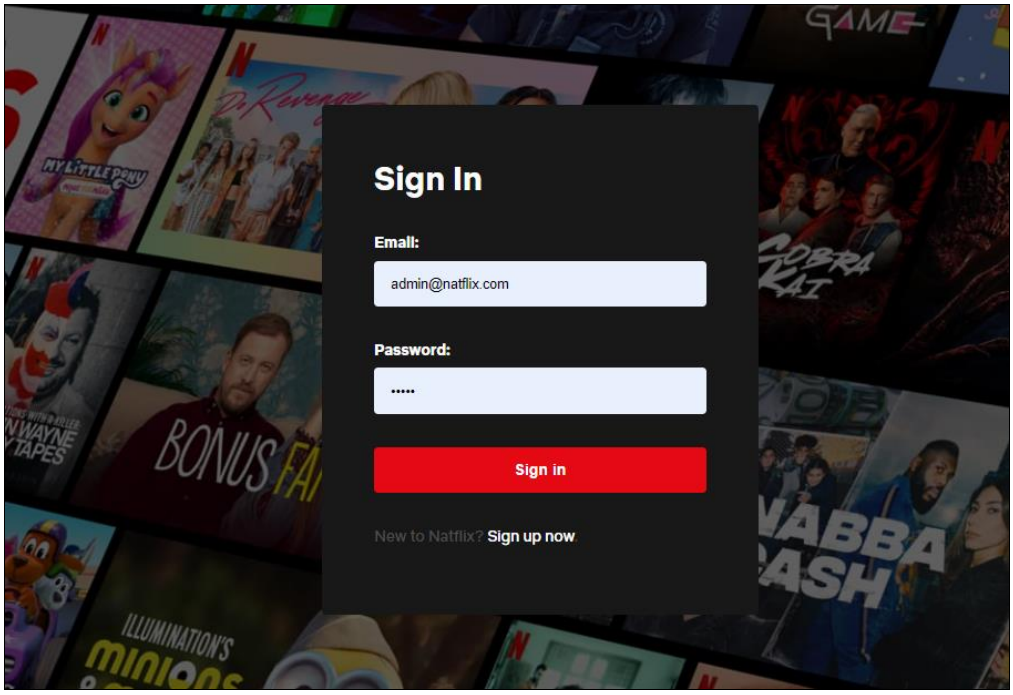
- Admin
- Customer

This application shall be containerized using Docker.

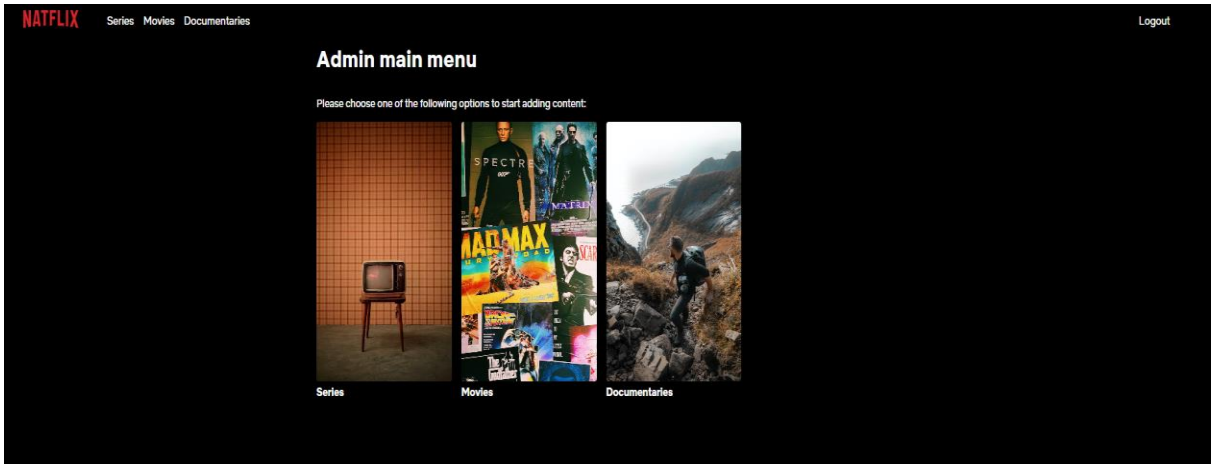
The data about the videos and customers are inserted in the database. The database we are using here is MySql. The data is inserted in the tables and is connected to the spring boot project in the backend.

2.1 Admin

The Administrator in this project has the information everything related to series, movies and documentaries. He shall be responsible to create, delete ,update and get the data (CRUD operations). He shall first login to the application and then he shall perform the operations .He initially has an account in database, so he shall directly log in to the application. You cannot create an admin account from the website. The screenshot of the login is shown below:



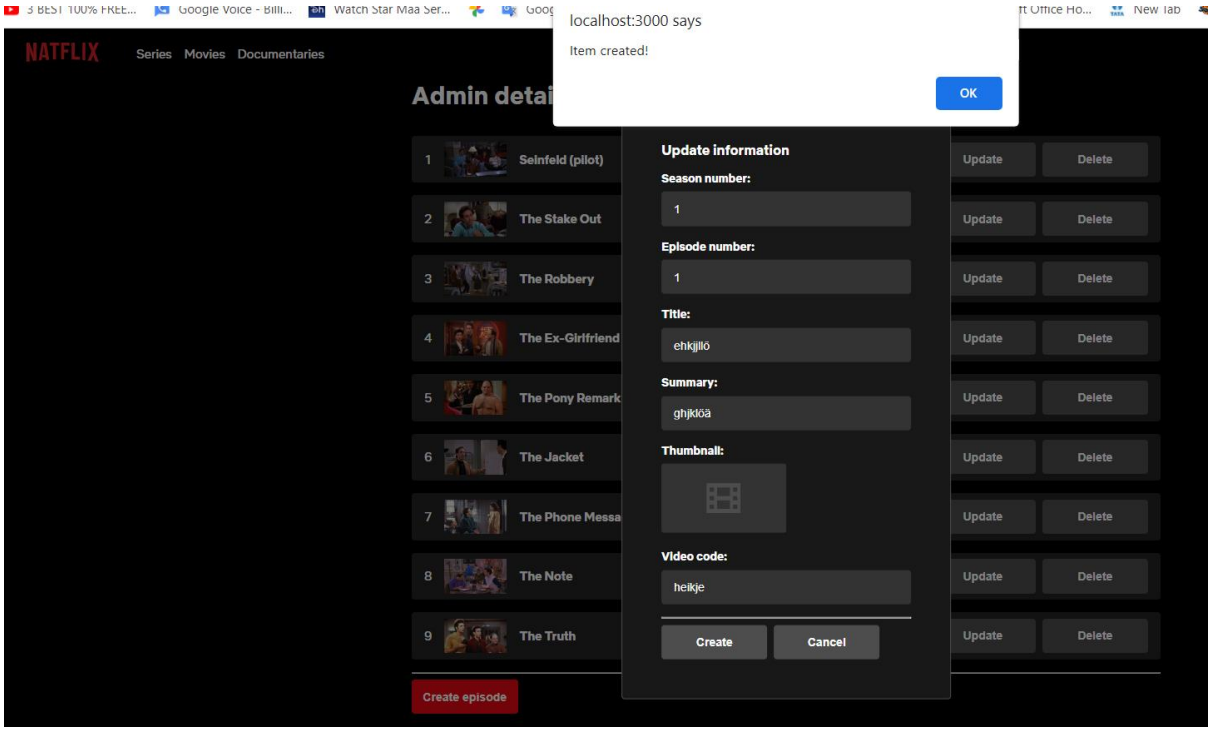
After logging into the natflix, the homepage of the admin is shown below:



The Admin shall have the option to Create. Delete and Update the series, movies and documentaries.

2.1.1 Create Content

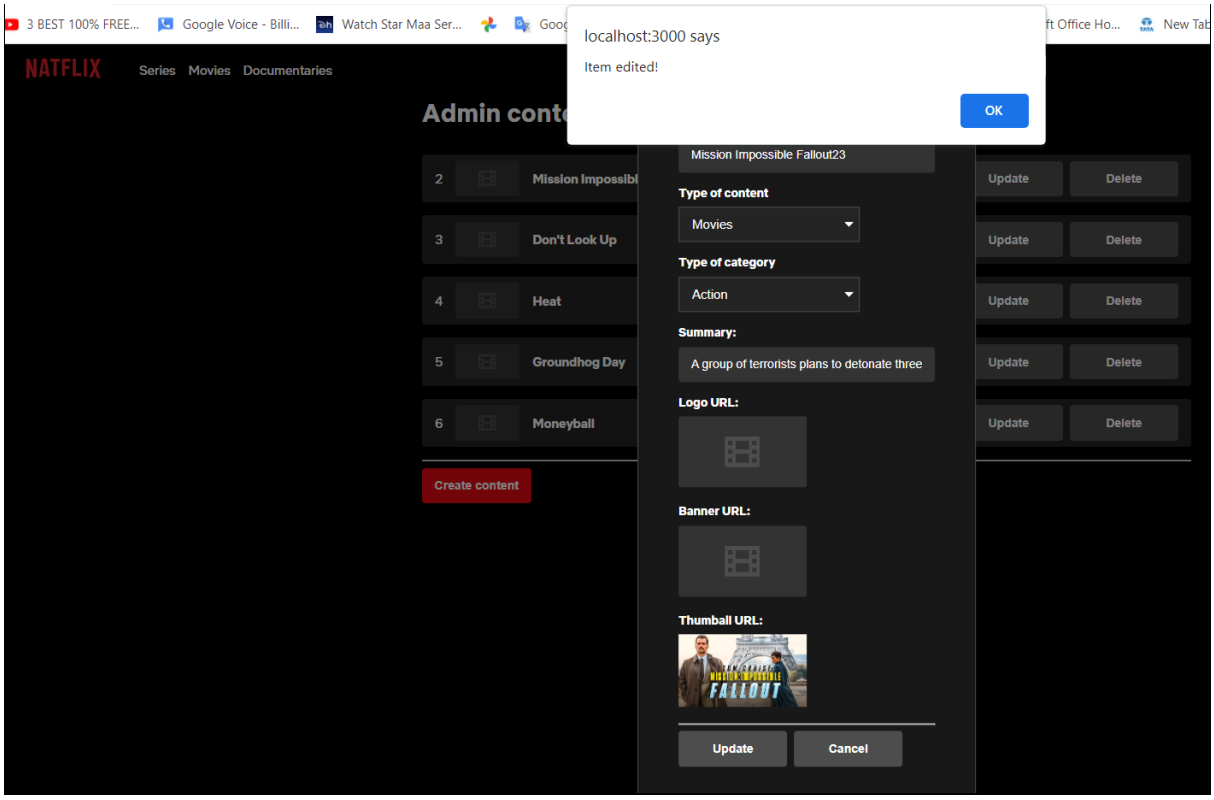
The Administrator shall have the option to create a Series , Movie and Documentaries. He shall create a new Series and post it in the Natflix ,so that the customer can view the Series. The screenshot of this is shown below.



He shall also do the same for Movies and Documentaries.

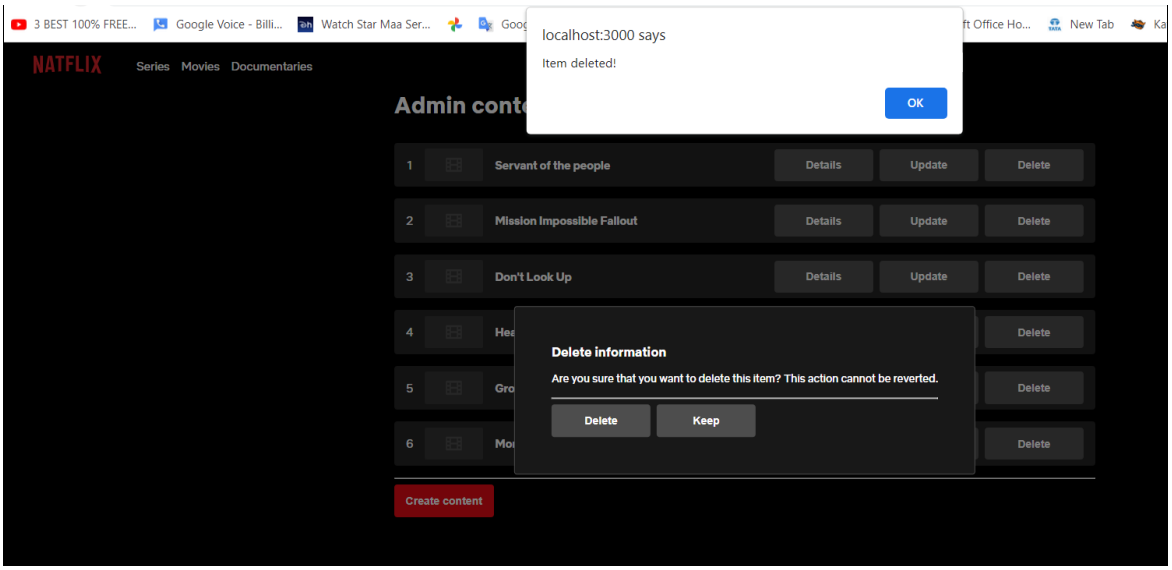
2.1.2 Update Content

The Administrator shall also update content for the Series, Movies and Documentaries, so that the customer shall have access to the content. The screenshot of this is shown below:



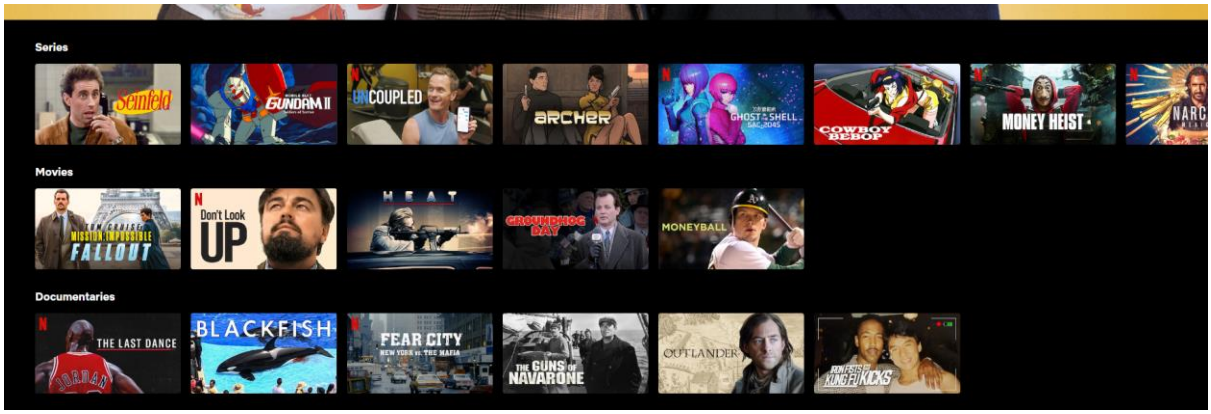
2.1.3 Delete Content

The Administrator shall have an option to delete the unwanted content from the natflix. He shall delete the unwanted Series, Episode, Movie or documentary. The screenshot of this is shown below:



2.2 Customer

The Customers shall have free accounts and login using the login page on the website . Authentication is more important here . A user to login to the system, authentication is more needed. The customers home page shall be shown below:



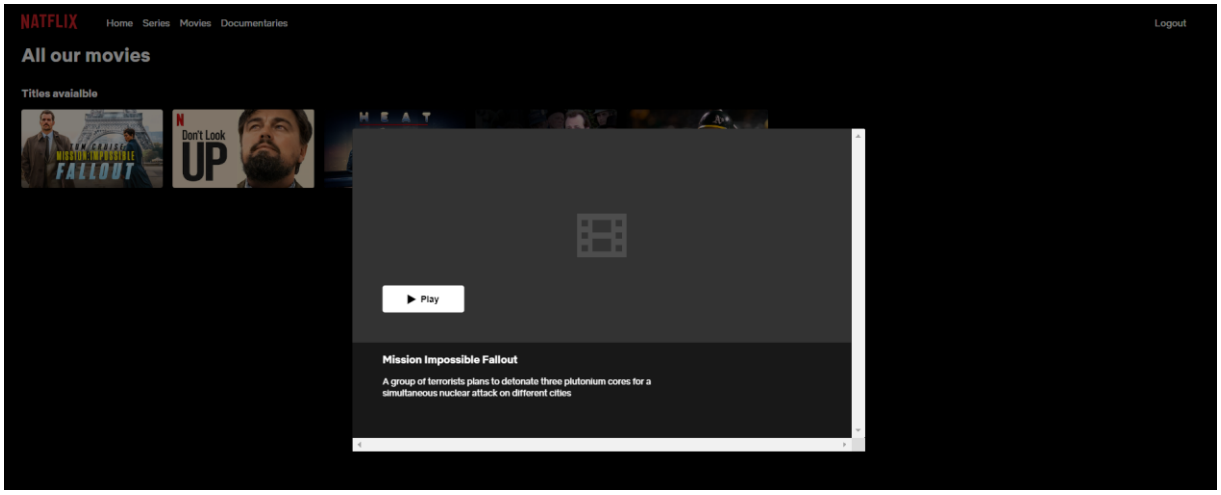
He shall have the access to view the Series, Movies and Documentaries.

2.2.1 View Series

The customer shall have access to view all the Series. He shall open the series and view the number of seasons and episodes. He can also play a song by selecting particular Series. The screenshot is shown below:

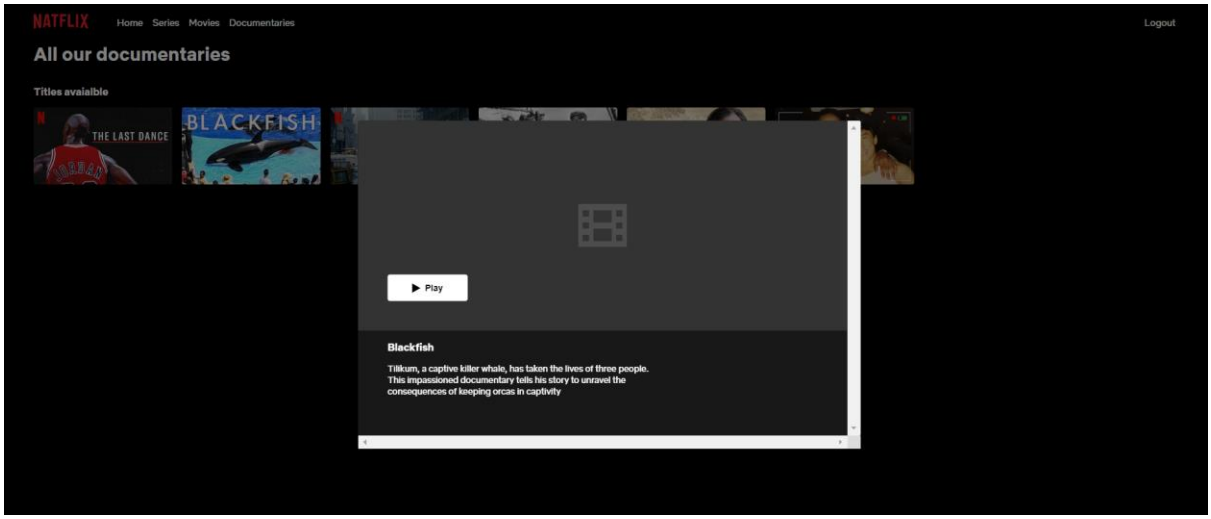
2.2.2 View Movies

The customer shall have access to view all the Movies in the natflix. He shall open the Movie and view the Movie. He shall also play a song by selecting particular Movies. The screenshot is shown below:



2.2.3 View Documentaries

The customer shall have access to view all the Documentaries in the Natflix. He shall open the particular documentary and view it. He shall also play a song by selecting particular Documentary. The screenshot is shown below:



2.3 Logout

The Customer and Administrator shall have the option of logout. They shall logout of the application by using logout button in the navigation bar.

3. Implementation and Design

3.1 Technologies

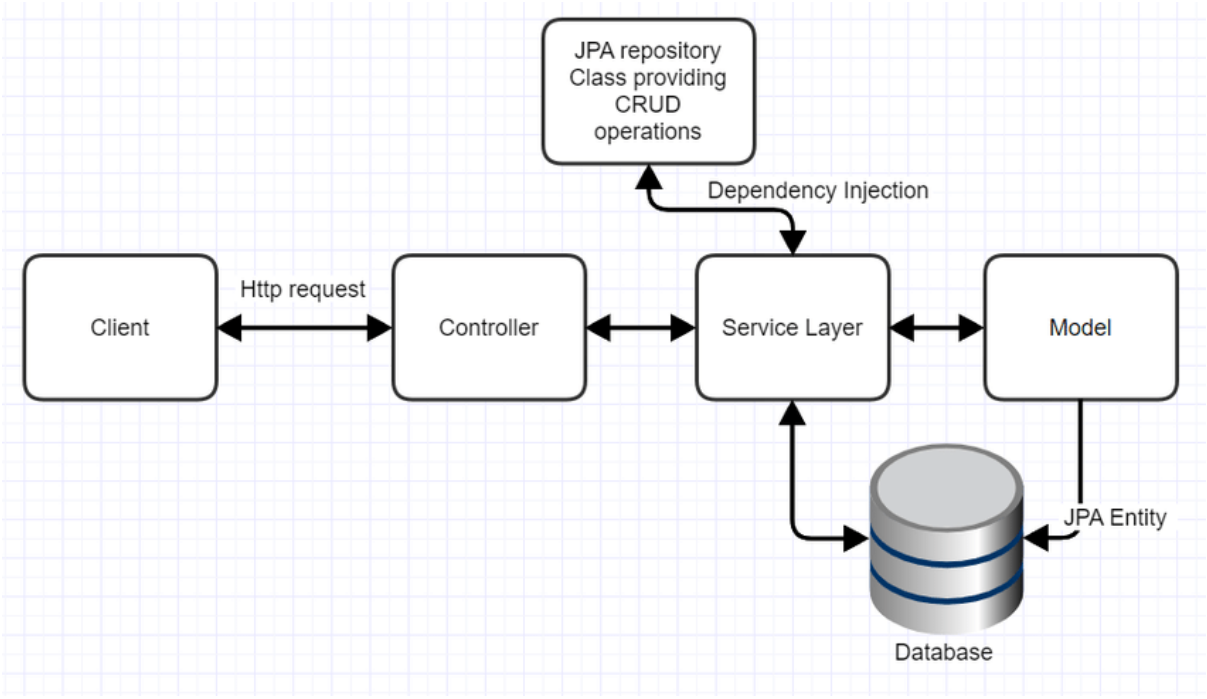
The following technologies are used in designing and developing the Natflix application.

1. TypeScript – Coding language for implementing the Natflix Frontend application
2. React -Framework for building the Graphical User Interface
3. Eclipse – IDE for backend development
4. Spring boot- Opensource Framework in Java
5. Visual Studio - IDE used for developing Frontend
6. Microsoft word – Document preparation

7. Postman- Postman is an API platform to design, build, test and iterate API's.

3.2 Architecture and Design

Spring boot MVC shall be used in designing the Natflix application, see below



Helps in building flexible and loosely coupled web applications.

With **M**-Model, **V**-View & **C**-Controller – there is a clear segregation of code for different logics

1. Model – normally represented as POJO & encapsulates the application data

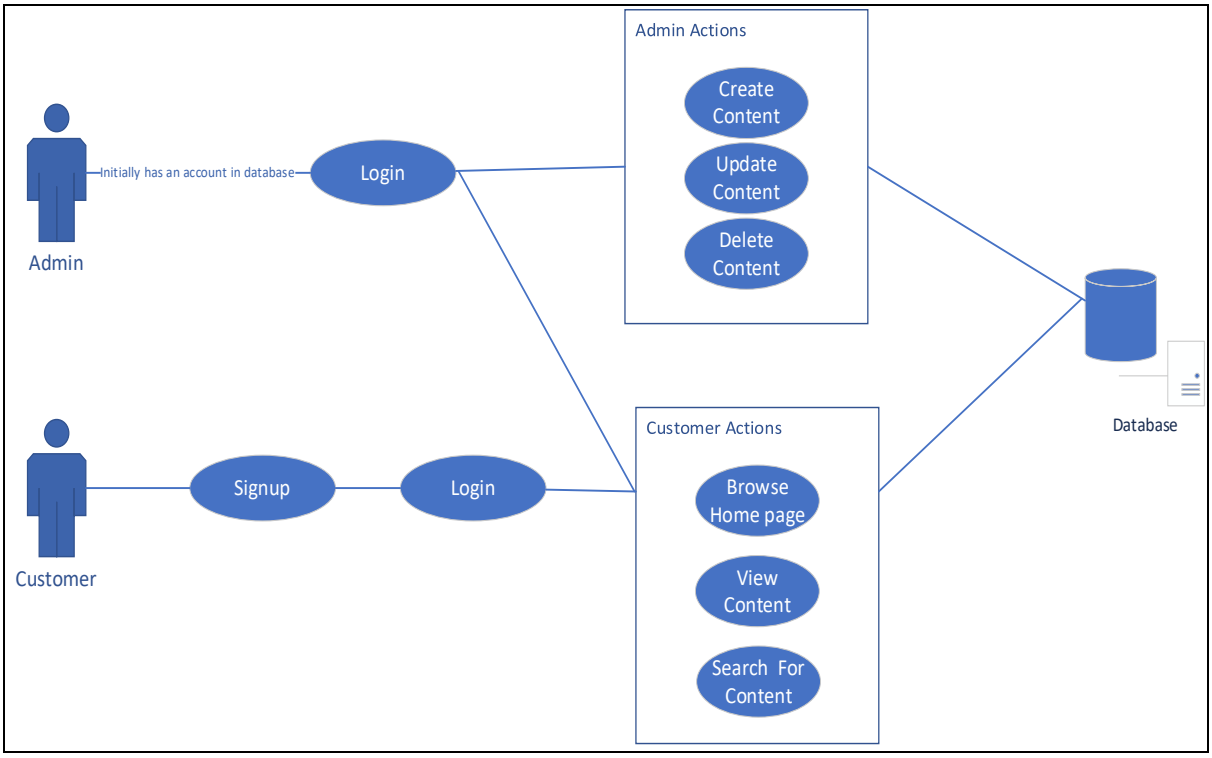
2. View – renders the response to the front-end user for the request handled

3. Controller – takes care of request processing on behalf of end user and constructs appropriate model to reply back to the calling user (requested user).

This Framework is designed for handling HTTP requests/responses. It also supports REST web services.

3.3 Use case Diagram

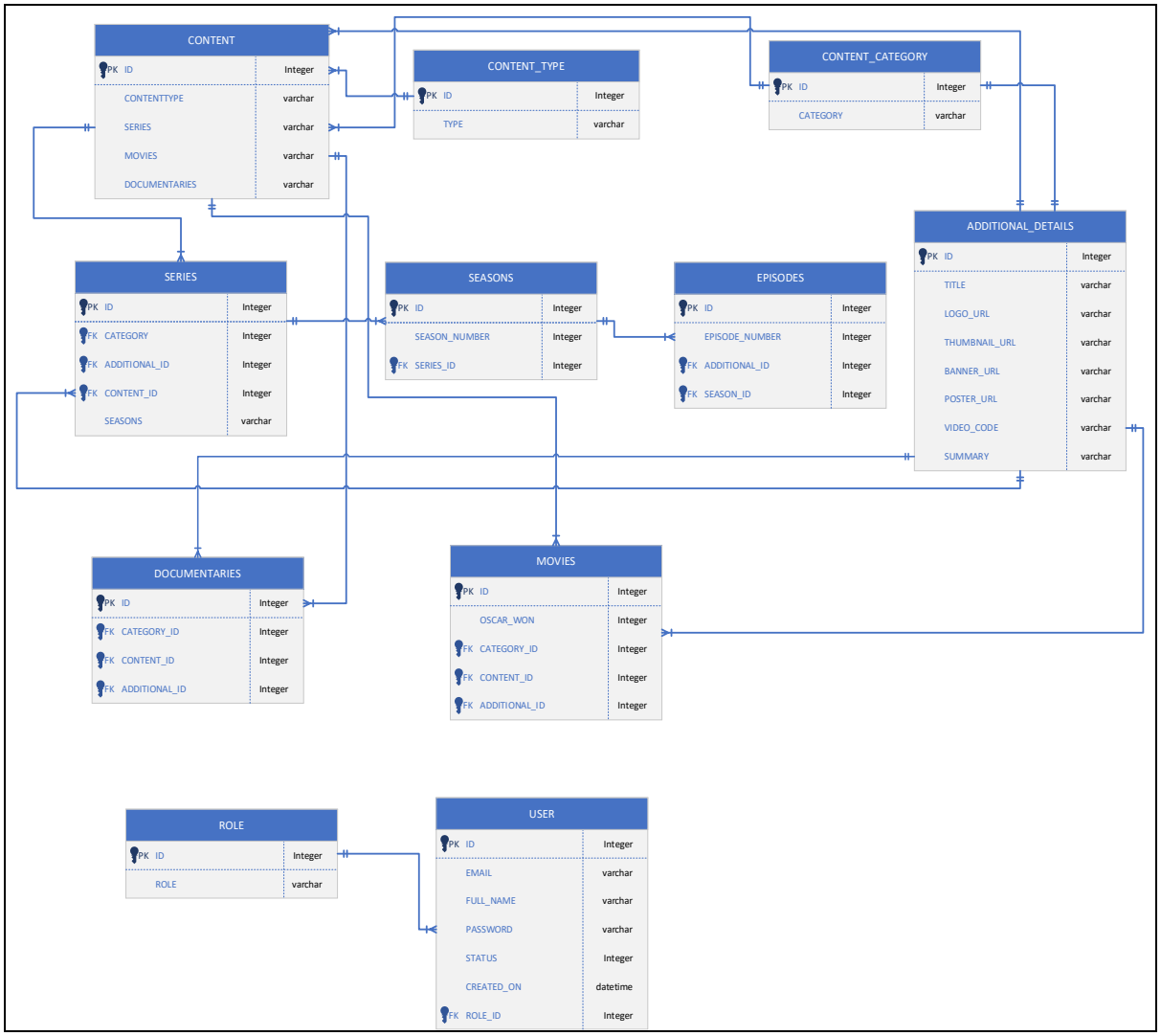
A use case diagram is a graphical depiction of a user's possible interactions with a system. It illustrates and defines the context and requirements of entire system. The usecase diagram for the natflix application is shown below:



3.4 Model Diagram

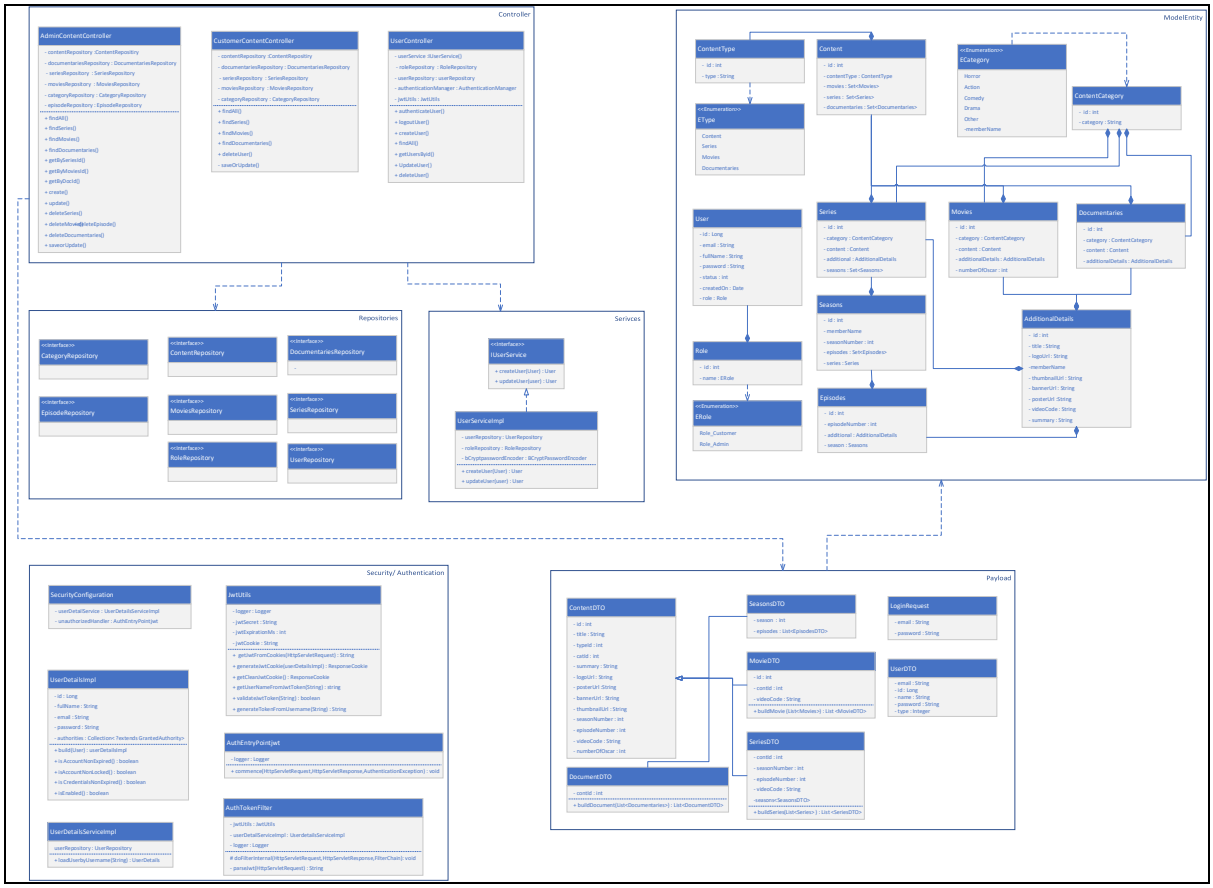
The model diagram represents the abstract view of the system. ER diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases.

ER diagrams are created based on three basic concepts: entities, attributes and relationships. The Model diagram is shown below:



3.5 Class Diagram

Class diagrams are blue prints of our system. We use class diagrams to model the objects that make up the system, to display the relationships between the objects, and to describe what those objects do and the services that they provide. The class diagram is shown below:


Controller
AdminContentController
- contentRepository : ContentRepository
- documentariesRepository : DocumentariesRepository
- seriesRepository : SeriesRepository
- moviesRepository : MoviesRepository
- categoryRepository : CategoryRepository
- episodeRepository : EpisodeRepository
+ findAll()
+ findSeries()
+ findMovies()
+ findDocumentaries()
+ getById(id)
+ getByMovies(id)
+ getByDocId()
+ create()
+ update()
+ deleteSeries()
+ deleteMovies(deleteEpisode)
+ deleteDocumentaries()
+ saveorUpdate()
CustomerContentController
- contentRepository :ContentRepository
- documentariesRepository : DocumentariesRepository
- seriesRepository : SeriesRepository
- moviesRepository : MoviesRepository
- categoryRepository : CategoryRepository
+ findAll()
+ findSeries()
+ findMovies()
+ findDocumentaries()
+ deleteUser()
+ saveOrUpdate()
UserController
- userService :IUserService()
- roleRepository : RoleRepository
- userRepository : userRepository
- authenticationManager : AuthenticationManager
- jwtUtils: JwtUtils
+ authenticateUser()
+ logoutUser()
+ createUser()
+ findAll()
+ getUsersById()
+ UpdateUser()
+ deleteUser()
Repositories
<<interface>>
CategoryRepository
<<interface>>
ContentRepository
<<interface>>
DocumentariesRepository
<<interface>>
EpisodeRepository
<<interface>>
MoviesRepository
<<interface>>
SeriesRepository
<<interface>>
RoleRepository
<<interface>>
UserRepository
Services
<<interface>>
IUserService
+ createUser(User) : User
+ updateUser(user) : User
UserServiceImpl
- userRepository : UserRepository
- roleRepository : RoleRepository
- bCryptpasswordEncoder : BCryptPasswordEncoder
+ createUser(User) : User
+ updateUser(user) : User
ModelEntity
ContentType
- id : int
- type : String
Content
- id : int
- contentType : ContentType
- movies : Set<Movies>
- series : Set<Series>
- documentaries : Set<Documentaries>
<<Enumeration>>
ECategory
Horror
Action
Comedy
Drama
Other
-memberName
ContentCatagory
- id : int
- category : String
<<Enumeration>>
EType
Content
Series
Movies
Documentaries
User
- id : Long
- email : String
- fullName : String
- password : String
- status : int
- createdOn : Date
- role : Role
Series
- id : int
- category : ContentCategory
- content : Content
- additional : AdditionalDetails
- seasons : Set<Seasons>
Movies
- id : int
- category : ContentCategory
- content : Content
- additionalDetails : AdditionalDetails
- numberOfOscar : int
Documentaries
- id : int
- category : ContentCategory
- content : Content
- additionalDetails : AdditionalDetails
Seasons
- id : int
- memberName
- seasonNumber : int
- episodes : Set<Episodes>
- series : Series
AdditionalDetails
- id : int
- title : String
- logoUrl : String
- memberName
- thumbnailUrl : String
- bannerUrl : String
- posterUrl :String
- videoCode : String
- summary : String
Role
- id : int
- name : ERole
<<Enumeration>>
ERole
Role_Customer
Role_Admin
Episodes
- id : int
- episodeNumber : int
- additional : AdditionalDetails
- season : Seasons
Security/ Authentication
SecurityConfiguration
- userDetailService : UserDetailsServiceImpl
- unauthorizedHandler : AuthEntryPointJwt
JwtUtils
- logger : Logger
- jwtSecret : String
- jwtExpirationMs : int
- jwtCookie : String
+ getJwtFromCookies(HttpServletRequest) : String
+ generateJwtCookie(userDetailsImpl) : ResponseCookie
+ getCleanJwtCookie() : ResponseCookie
+ getUserNameFromJwtToken(String) : string
+ validateJwtToken(String) : boolean
+ generateTokenFromUsername(String) : String
UserDetailsImpl
- id : Long
- fullName : String
- email : String
- password : String
- authorities : Collection< ?extends GrantedAuthority>
+ build(User) : userDetailsImpl
+ isAccountNonExpired() : boolean
+ isAccountNonLocked() : boolean
+ isCredentialsNonExpired() : boolean
+ isEnabled() : boolean
AuthEntryPointJwt
- logger : Logger
+ commence(HttpServletRequest,HttpServletResponse,AuthenticationException) : void
UserDetailsServiceImpl
- userRepository : UserRepository
+ loadUserByUsername(String) : UserDetails
AuthTokenFilter
- jwtUtils : JwtUtils
- userDetailsServiceImpl : UserdetailsServiceImpl
- logger : Logger
doFilterInternal(HttpServletRequest, HttpServletResponse, FilterChain) : void
- parseJwt(HttpServletRequest) : String
Payload
ContentDTO
- id : int
- title : String
- typeId : int
- cntId : int
- summary : String
- logoUrl : String
- posterUrl :String
- bannerUrl : String
- thumbnailUrl : String
- seasonNumber : int
- episodeNumber : int
- videoCode : String
- numberOfOscar : int
SeasonsDTO
- season : Int
- episodes : List<EpisodesDTO>
LoginRequest
- email : String
- password : String
MovieDTO
- id : int
- cntId : int
- videoCode : String
+ buildMovie (List<Movies>) : List <MovieDTO>
UserDTO
- email : String
- id : Long
- name : String
- password : String
- type : Integer
DocumentDTO
- cntId : int
+ buildDocument(List<Documentaries>) : List <DocumentDTO>
SeriesDTO
- cntId : int
- seasonNumber : int
- episodeNumber : int
- videoCode : String
-seasonsrSeasonsDTO>
+ buildSeries(List<Series>) : List <SeriesDTO>