PICTURE PERFECT : AN ONLINE MOVIE TICKET BOOKING AND REVIEW SERVICE

SYSTEM DESIGN DOCUMENT

Prepared By : Aravind Ravikumar

REVISION HISTORY

Date	Version	Description
April 14th 2020	1.0	First Draft, for review
April 24th 2020	2.0	UX Mocks, ER Diagrams, File structure changes

TABLE OF CONTENTS

1. INTRODUCTION

Picture Perfect is an online movie ticket booking, review and rating service. The service helps users generate review and rating content for movies across the world. This section gives a brief overview of this design document.

1.1 Purpose

The purpose of this document is to present a detailed descriptions of the technical designs and system architecture of the Picture Perfect movie ticket booking and review system, which is intended as an individual ramp-up project to get familiarized with the tech-stack that is in use at Clumio, Ltd.

1.2 Scope

Picture Perfect is an online platform which facilitate ticket booking and reviewing of movies with user management functionalities. Anyone can register in the website, publish ratings and reviews on movies in a language of choice and book tickets for any show screened at given cities. It has a responsive UI with multiple web-pages each having different functionalities like catalogue, user management, ratings, reviews, booking shows. It is designed to be highly available, offer concurrent service to multiple users and be highly secure.

1.3 Overview

In the remainder of this document, we'll detail out the high-level design, the technology stack, technical approach, system architecture and deployment details of this system.

2. SYSTEM OVERVIEW

In this section we discuss the general system overview, such as the high-level description and the technologies utilized.

2.1 High Level Description

Picture Perfect system comprises of the following components

Desktop website for users to access functionalities provided by Picture Perfect
Desktop Web-based administration and management console for back-end operations
Public REST APIs for integration with other movie and media portals

2.2 Technology stack

Technology	Description
Golang	Server-side code
dep	Golang package manager
gorilla/mux	Web Framework in Golang
TypeScript	Client Side Language
ReactJS with Redux	Major Libraries utilized for development of User-Interface
PostgreSQL	Database Management
Jest, Enzyme	JavaScript testing framework for Front-end components
Selenium WebDriver	Cross-test against different browsers
GoConvoy	Go testing tool for Gophers
Jenkins	CI/CD environment

Golang
Golang is chosen as the back-end for the system because of its simplicity and performance. High performance backed by super efficient concurrency handling due to Goroutines and ease-of-code attributed to absence of any rigid programming paradigm constraint makes Golang standout among other server-side technologies.

gorilla/mux
Package gorilla/mux implements a request router and dispatcher for matching incoming requests to their respective handler. It implements the http.Handler interface so it is compatible with the standard http.ServeMux.

TypeScript
By definition, "TypeScript is JavaScript for application-scale development" which is a typed super-set of JavaScript which is compiled to JavaScript. It is JavaScript with additional features like compilation and error correction abilities, support of OOP paradigms, type definition support.

ReactJS
ReactJS offers better reusability of system components, better user experience because of faster rendering by use of virtual DOM, stable code due to downward data flow and integration possibility with Redux.

Redux
Redux is a state management tool for JavaScript applications. Redux makes the state predictable. This greatly simplifies the app and makes it easier to maintain and test.

PostgreSQL
We prefer a Relational Model for Database Management to a NoSQL database because we have rigid structure of tables, carrying out complex queries, more stability and data integrity during high transaction volumes and ACID compliance. We choose PostgreSQL as our Database Management System as it ensures swifter execution of complex queries as well as better data integrity all round compared to other RDBMS like MySQL.

3. SYSTEM ARCHITECTURE

In this section, we give a detailed description of the system components with focus on the REST API structure, the file architecture, back-end design, database design, interface design.

3.1 Components and REST API structure

The component services that constitute Picture Perfect and their functionalities with corresponding REST structure are as follows:

- **Catalogue** - This service is to retrieve and maintain the catalogue of movies, documentaries and television programs.
    - Typical user operations would include (both logged in or an unauthenticated user)
        - GET /movies/catalogue - Get a paginated list of movies, along with the associated media (links to the thumbnail pictures)
        - GET /movies/catalogue/{name} - Get a movie/documentary by name with detailed info and the media links images, videos

        - GET /movies/catalogue?{query} - Search a movie with filter and sort criteria
            - Filter - could be on any attribute name, language,
            - Sort - Sort the results in ascending or descending order
            - Paginate - To paginate the results to obtain the results in chunks
    - Typical backend admin operations would include
        - POST /movies/catalogue - Add a new item to the catalogue
        - PUT /movies/catalogue - Update an item in the catalogue
        - PATCH /movies/catalogue - Update a specific attribute to an item in the catalogue

- **IAM – Identity and Access management -** this is to authenticate a user, and identify if the user is general user or somebody who can manage the PicturePerfect operations based on a role and privilege
    - A generic user role - Should not have access to the backend console but only to the PicturePerfect website
        - POST /login - create a new token for a login session
        - POST /logout - Invalidate the session and logout
        - POST /reset - Reset the password to a new one
    - An admin user role - Should have access to both the backend console and the PicturePerfect website
        - POST /login - create a new token for a login session
        - POST /logout - Invalidate the session and logout
        - POST /reset - Reset the password to a new one

- **Reviews** - This service about the movies or shows
    - Users have
        - PUT /movies/review/{movie} - Add or update a new movie review
        - DELETE /movies/review/{movie} - Delete a movie review created by the user

- **Shows** - This service lists the cineplexes where the movie is being screened in a given city
    - User operations
        - GET /movies/shows/{city} - List all shows in all cineplexes in a city
            - This should have the ability to filter, paginate and sort the results
        - GET /movies/shows/{city}/{movie} - List the cineplexes screening a particular movie

- Admin operations - In addition to the user operations above, admins can do the following
  - POST - /movies/shows - Add a new show or add a new cineplex
  - PUT - /movies/shows - Update the show timings
  - DELETE - /movies/shows - Delete a show
  - DELETE - /movies/shows/{movie} - Remove a movie from all screens

3.2 File Architecture

We organize the files on the basis of domain instead of nature(function). The general file structure will be :

```
Containers
---------Home
-----------------HomeComponent.tsx
-----------------HomeContainer.ts
-----------------HomeStyle.css
---------Catalogue
-----------------CatalogueComponent.tsx
-----------------CatalogueContainer.ts
-----------------CatalogueStyle.css
---------...

Components
---------Header
-----------------Header,tsx
-----------------Header,css

Public
---------Images
---------Utilities

Reducers
---------rootReducer.ts
---------store.ts

Server
---------MiddleWare
```

------------------middleware.go
---------Handlers
------------------handlers.go
---------Database
------------------postgresql.go
---------Testing
------------------main_test.go
---------main.go

3.3 Back-end Design

We follow many clean architecture constraints while designing the back-end, which include

Independency of UI : User Interface independent of the back-end code, that is the interface can be changed easily without any modifications on the rest of the system.
Independency of Database: Server rules not dependent on any particular database. Hence we can swap database management systems easily.
Testability: Independent testing of server side elements possible.

We use dep as package manager and gorilla/mux as the web framework.

The route.go will have the following routes, serving the following files:

Route	Page Description
/home	Home Page
/login	Login Page
/catalogue	Movie Catalogue
/loginPage	Login Page
/cineplex?city=	View Cineplexes
/movie?movieTitle=	Details about a single Movie page
/reviews?movieTitle=	Reviews and Ratings of a Movie
/shows?city=&movieTitle=	All Shows of a Movie in a City

/profile?userId=	Profile of a User

3.4 Database Design

Out database schema consists of the following tables with the following attributes :

UserInfo :

Attribute	Data Type	Description
userId	VARCHAR(256) NOT NULL PRIMARY KEY	Unique UserId, maximum 256 char
name	VARCHAR(256) NOT NULL	Full name of the User
emailId	Custom domain ‘email’ with citext for case-insensitive text and check NOT NULL	Email Address of the user
city	VARCHAR(256) with citext for case-insensitive text NOT NULL	City the user resides, helps recommending shows nearby
phoneNo	VARCHAR(10) with check for numeric	Phone number if user requires notifications
address	VARCHAR(1024)	Address of the user for billing information
password	VARCHAR(256)	SHA256 Hashed passwords given by the user
role	ENUM	Privilege of the user. Default is user, can be changed by authorized admins or higher.

MovieCatalogue :

Attribute	Data Type	Description
movieId	SERIAL	Unique identifier for movie, with serial numbering

title	VARCHAR(256) NOT NULL	Title of the movie
language	VARCHAR(100) NOT NULL	Language of the movie
releaseDate	DATE	Date of release of the movie
duration	TIME	Duration of the movie
thumbnail	VARCHAR(256)	Link to the static file image
link	VARCHAR(256)	URL to the official page of the film

MovieGenre :

Attribute	Data Type	Description
movieId	INTEGER REFERENCES MovieCatalogue(movieId)	Movie Id of the movie for which the genre is specified.
genre	VARCHAR(256)	Genre of the movie

Cineplex :

Attributes	Data Type	Description
cineplexId	SERIAL	Unique identifier for cineplexes, with serial numbering
name	VARCHAR(256)	Name of the cineplex
city	VARCHAR(256)	City of the cineplex
address	VARCHAR(1024)	Address of the cineplex
phoneNo	VARCHAR(10) with check for numeric	Phone number of the cineplex

.

RatingsAndReviews :

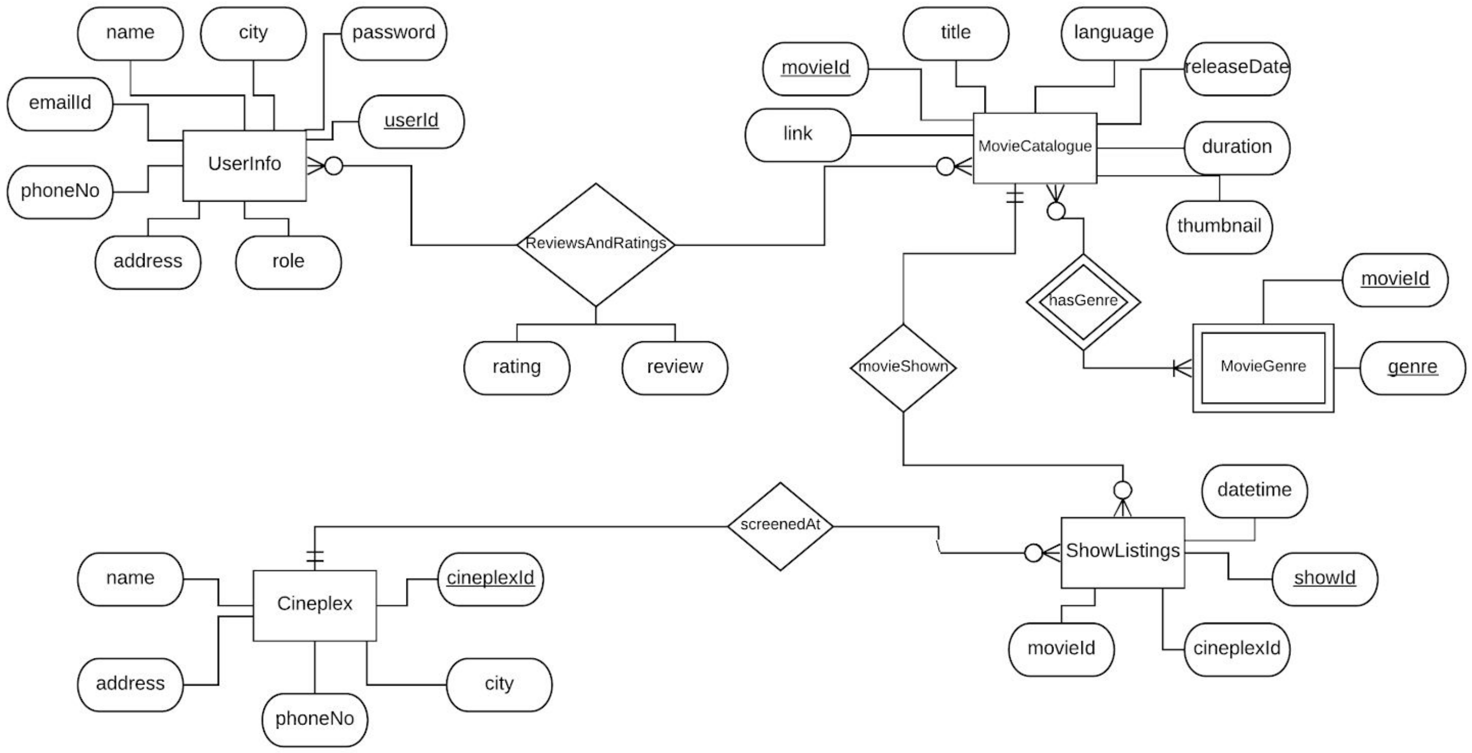
Attributes	Data Type	Description
userId	VARCHAR(256) REFERENCES	userId of the user who put up the rating/review.

	UserInfo(userId)	Foreign key of usedId in UserInfo.
movieId	INTEGER REFERENCES MovieCatalogue(movieId)	movieId of the movie this rating/review is for. Foreign key of movieId in MovieCatalogue
rating	INTEGER with check for greater than or equal to 0 and less than or equal to 10	Rating of the movie by the user
review	TEXT	Review of the movie by the user

ShowListing :

Attributes	**Data Type**	**Description**
showId	SERIAL	Unique identifier for cineplexes, with serial numbering
cineplexId	INTEGER REFERENCES Cineplex(cineplexId)	cineplexId of the Cineplex where the show is happening. Foreign key of cineplexId in Cineplex.
movieId	INTEGER REFERENCES MovieCatalogue(movieId)	movieId of the movie being screened. Foreign key of movieId in MovieCatalogue.
datetime	TIMESTAMP	Date and time of start of the show.

ER Diagram



3.5 Interface Design

The User-Interface consists of the following user components, accessibility depends on whether user is logged in or unauthenticated user:

- Home page - Listing of top movies across different categories
- Login page
- Search and listing of results, along with filters and pagination of results
- Ratings component - Adding and updating movie ratings from a rating widget
- Reviews component - Adding and updating reviews from a review widget

User-Interface also offers functionalities for authenticated administers through the same user components but with additional operations based on roles and privilege which are:

- Adding/Removing/Disabling cineplexes
- Updating movie catalogue
- Updating movie ratings
- Updating movie reviews
- Adding/Updating/Disabling promos and vouchers

The key requirements from the User-Interface are:

- Responsive UI that adapts to different screen sizes (mobile, tablet, desktop)
- Simple and easy navigation, via categories menu and bread crumbs
- i18n enabled –The reviews can be in any language
- Garbage collector ensures invalid, biased and reviews by automated bots are cleared up.

We use TypeScript as language for front-end with ReactJS library to build UI components with React-Redux as an application state management system.

We also use packages like Redux-Saga, React-Router, react-i18next and libraries like Material UI to develop the User-Interface.

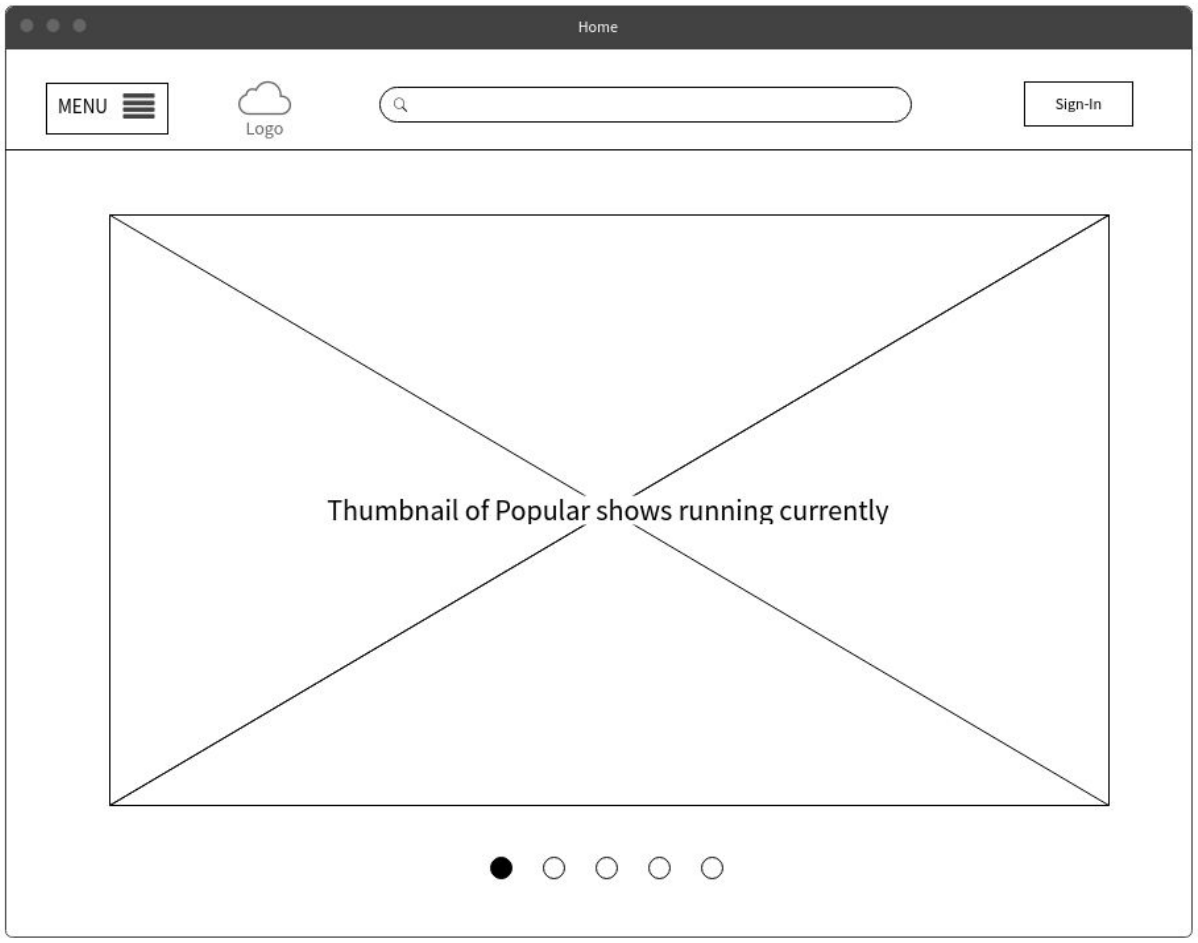
Redux-Saga
Redux-Saga is a library that aims to make application side effects (e.g., asynchronous actions such as fetching data) easier to handle and more efficient to execute.The idea is that a saga is similar to a separate thread in your application that's solely responsible for side effects. This can facilitate easier testing of asynchronous data flow and can avoid callback hell, hence preferred to redux-thunk.

Material-UI
Material UI library simplifies the development process and helps with designing an awesome front-end.

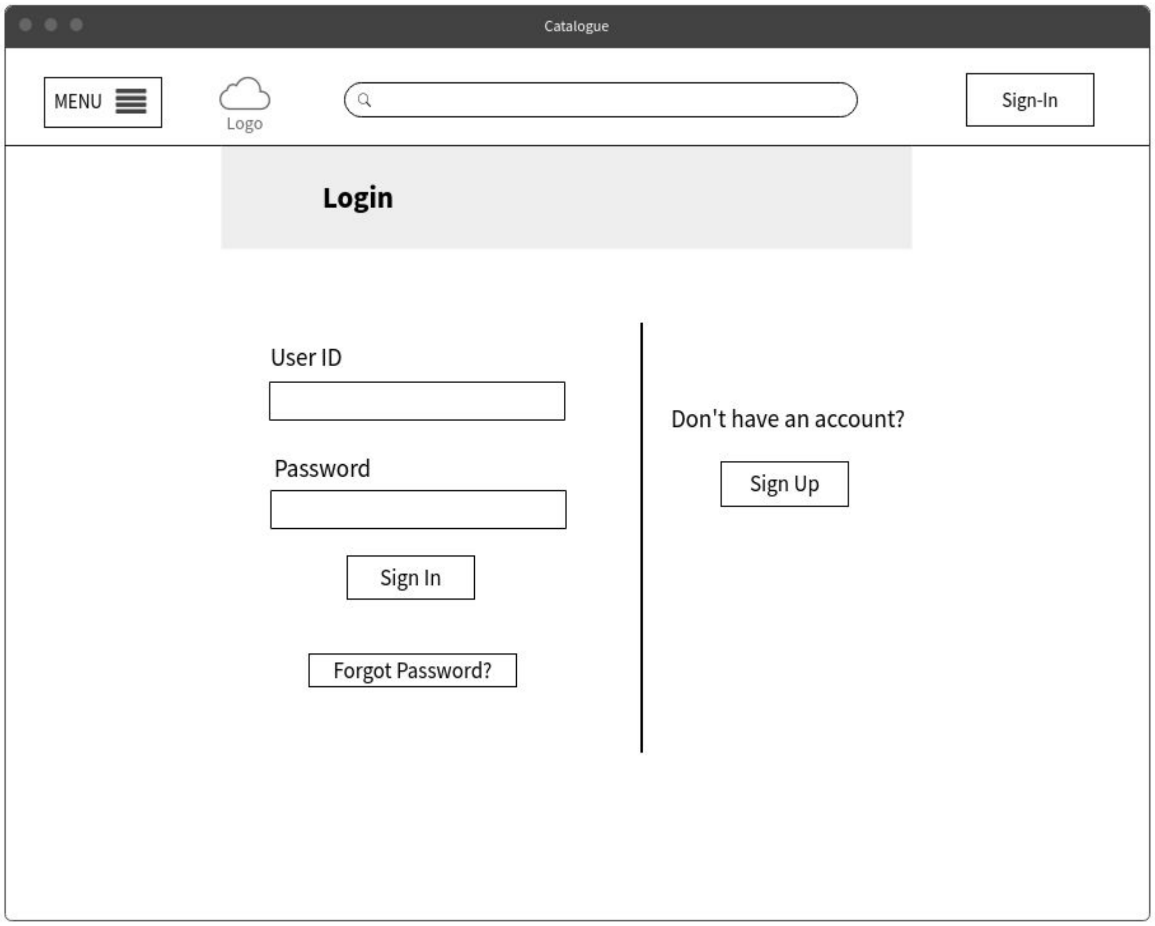
React-Router
With React-Router, the routes are considered as components. When the app is running, the routes are rendered to the screen.

react-i18next
react-i18next is an internationalization framework for React based on i18next.

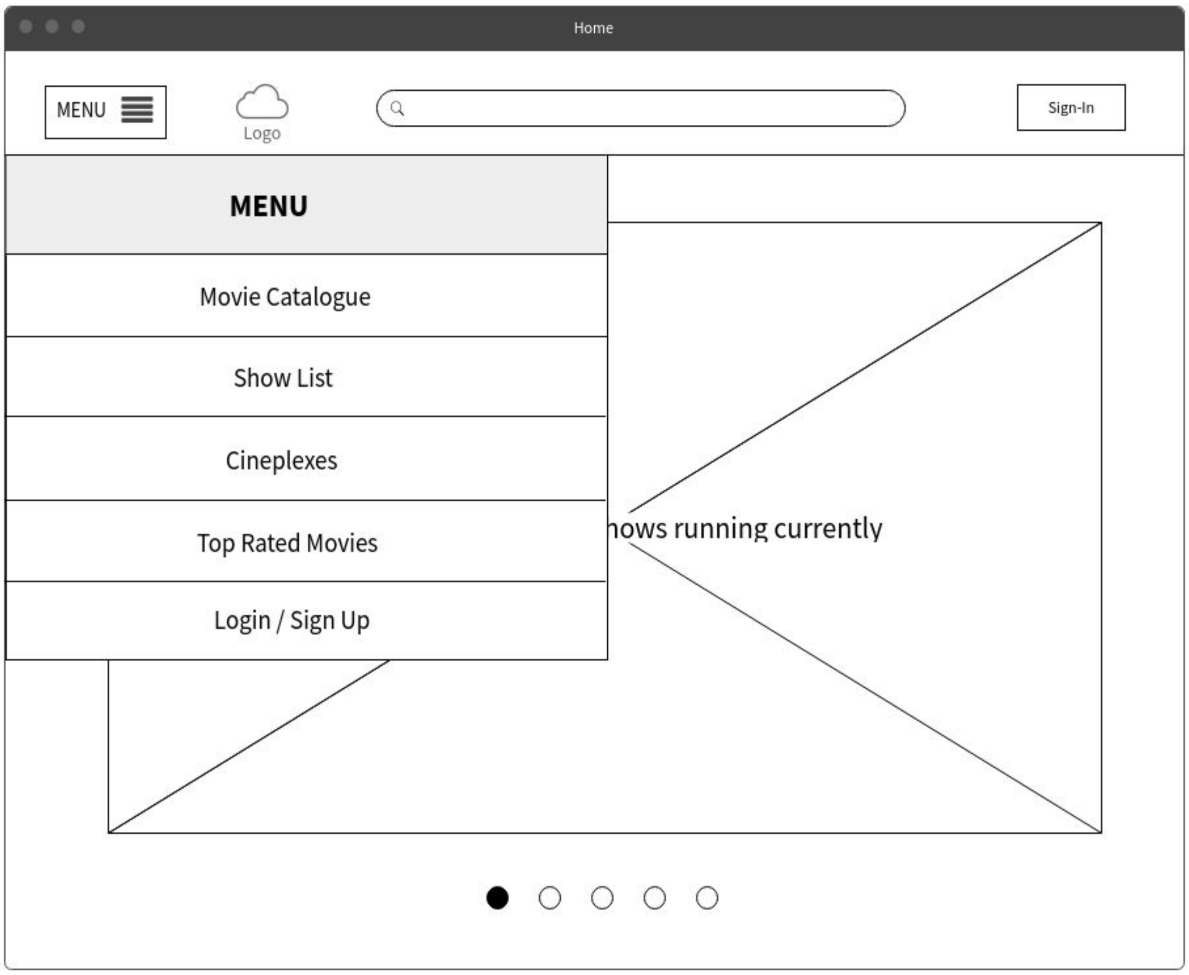
3.6 Mock UX Diagrams


Home
MENU
Logo
Sign-In
Thumbnail of Popular shows running currently

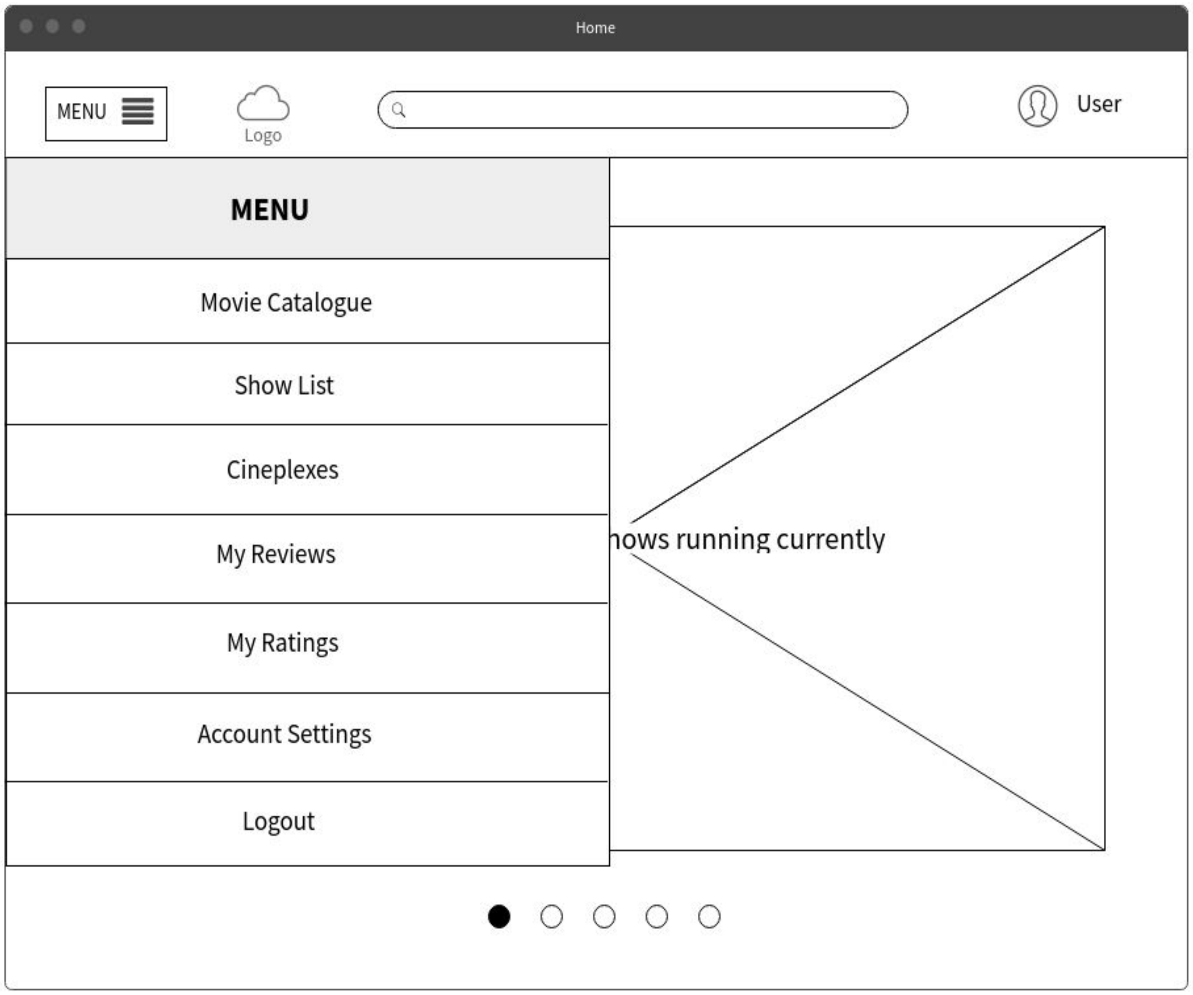
Home Page


Catalogue
MENU
Logo
Sign-In
Login
User ID
Password
Sign In
Forgot Password?
Don't have an account?
Sign Up

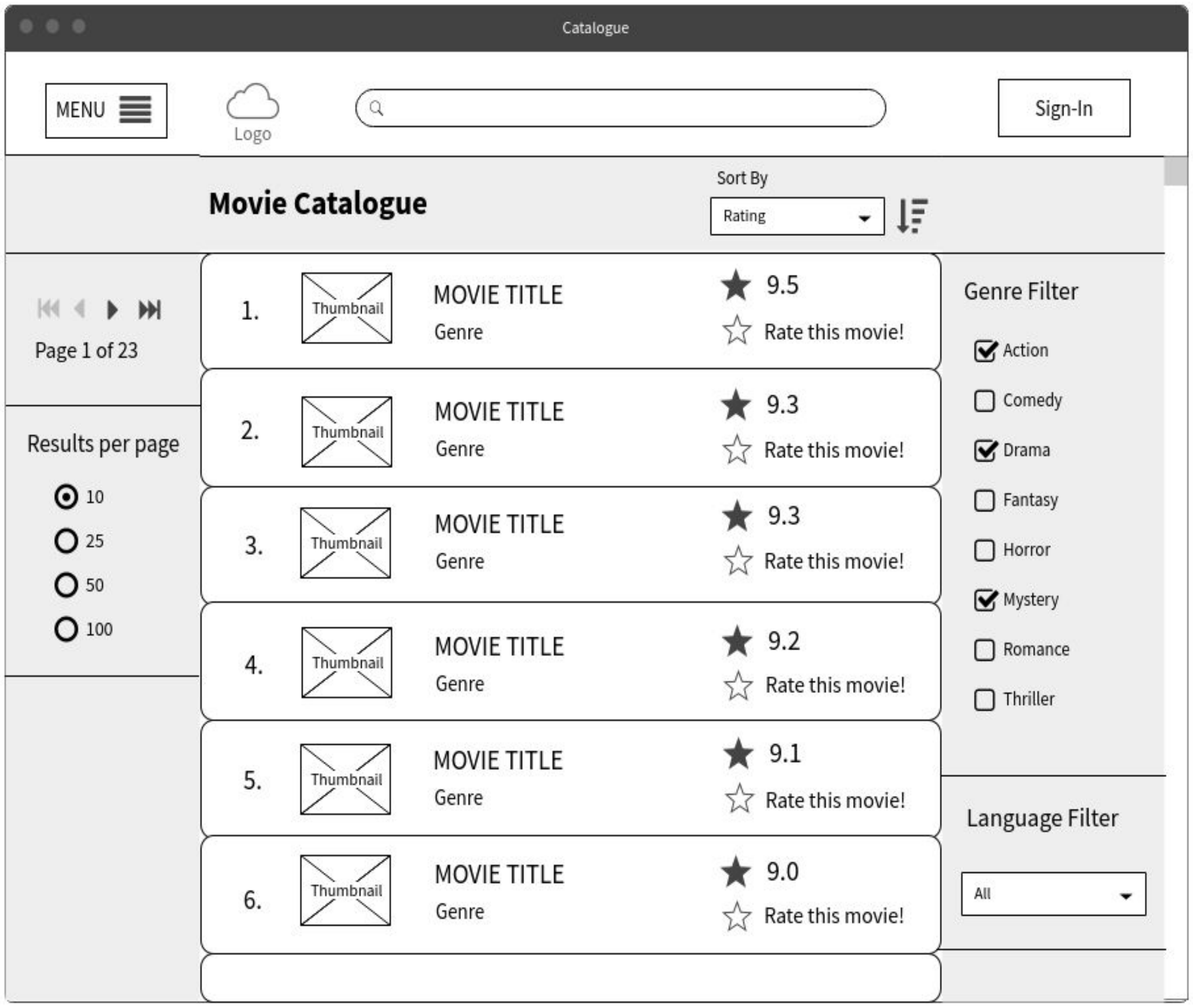
Login Page



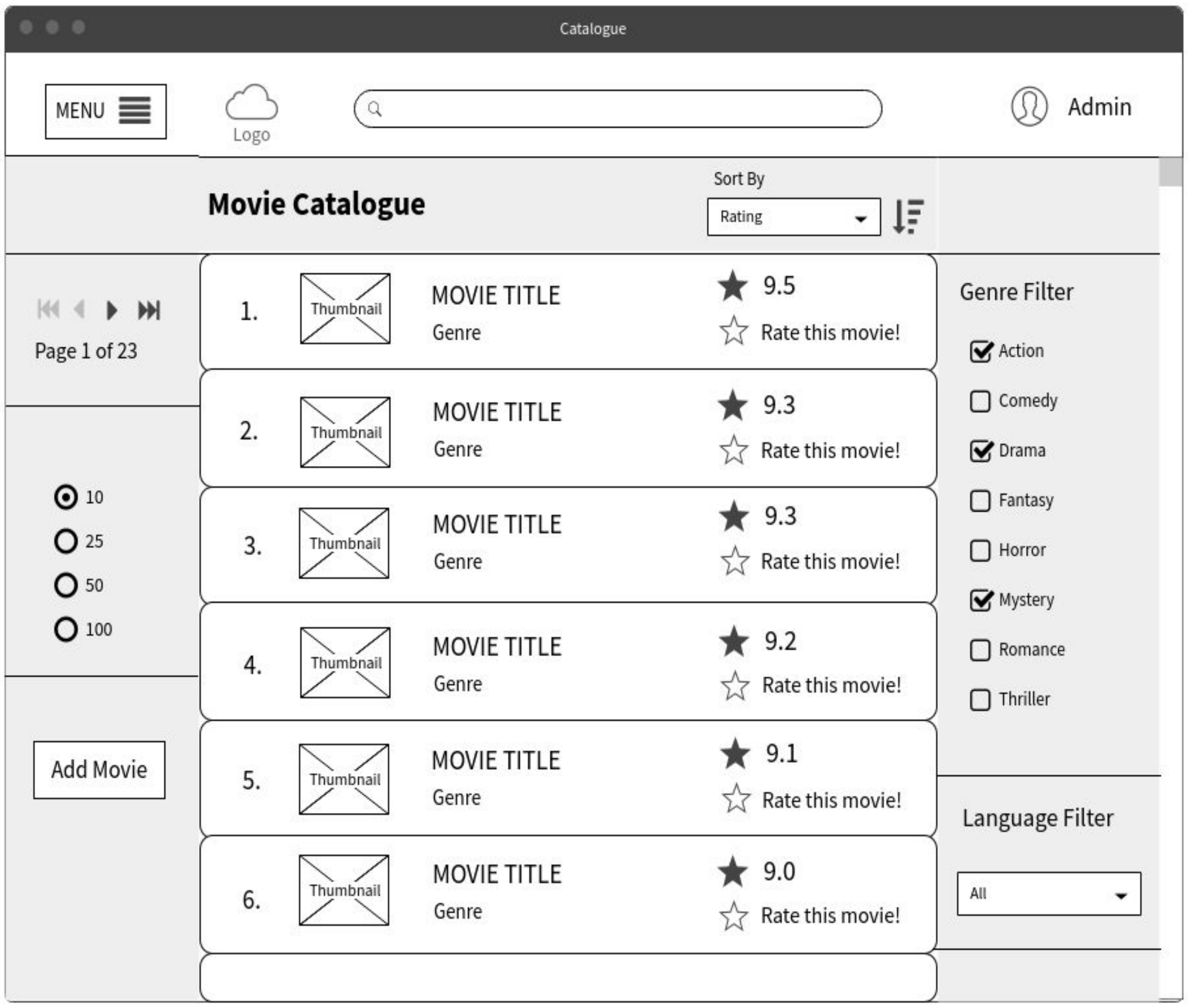
Menu For Non-Users



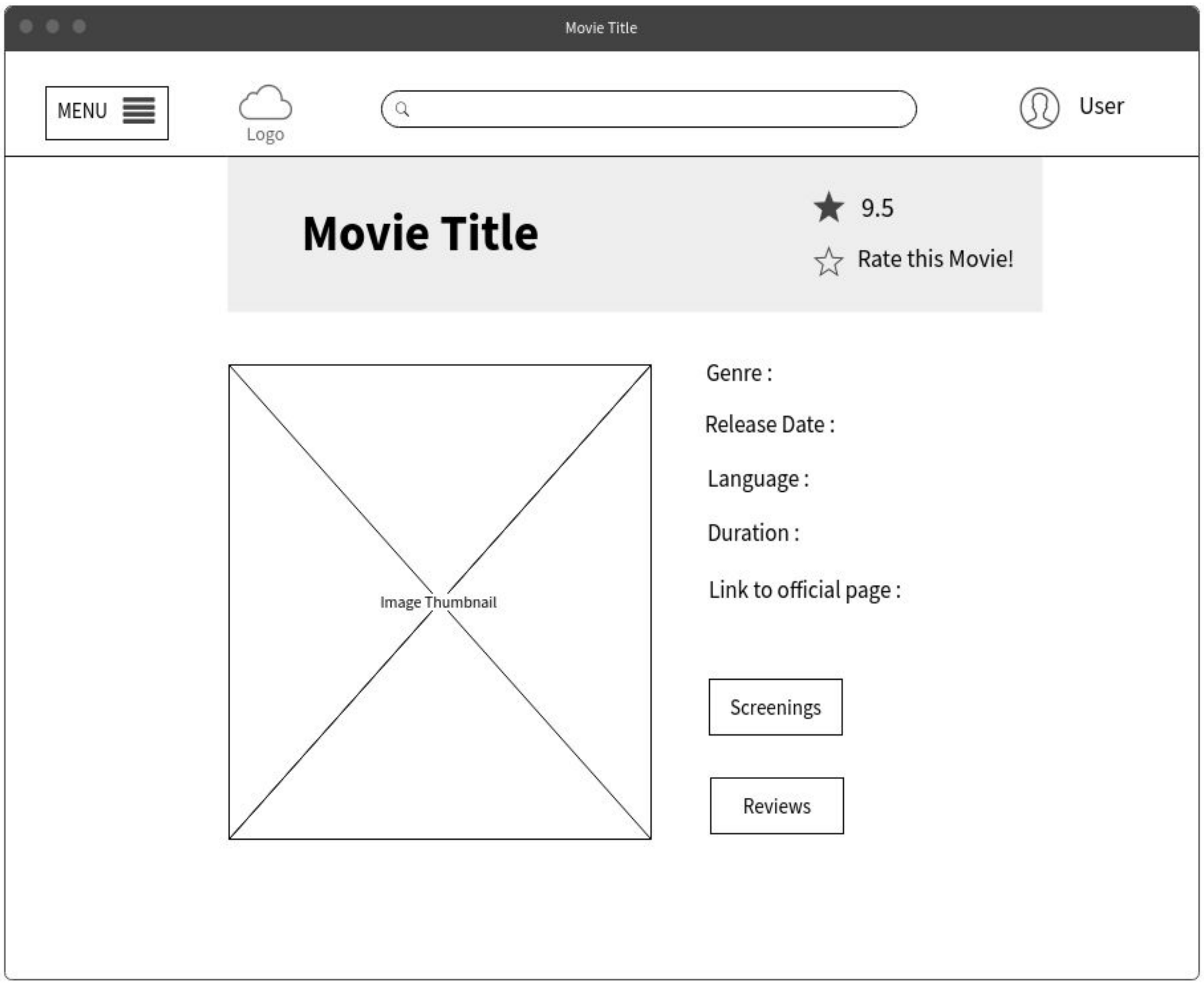
Menu For Authenticated Users & Admins


Catalogue
MENU
Logo
Sign-In
Movie Catalogue
Sort By
Rating
Page 1 of 23
Results per page
● 10
○ 25
○ 50
○ 100
1. Thumbnail MOVIE TITLE Genre 9.5 Rate this movie!
2. Thumbnail MOVIE TITLE Genre 9.3 Rate this movie!
3. Thumbnail MOVIE TITLE Genre 9.3 Rate this movie!
4. Thumbnail MOVIE TITLE Genre 9.2 Rate this movie!
5. Thumbnail MOVIE TITLE Genre 9.1 Rate this movie!
6. Thumbnail MOVIE TITLE Genre 9.0 Rate this movie!
Genre Filter
✓ Action
☐ Comedy
✓ Drama
☐ Fantasy
☐ Horror
✓ Mystery
☐ Romance
☐ Thriller
Language Filter
All

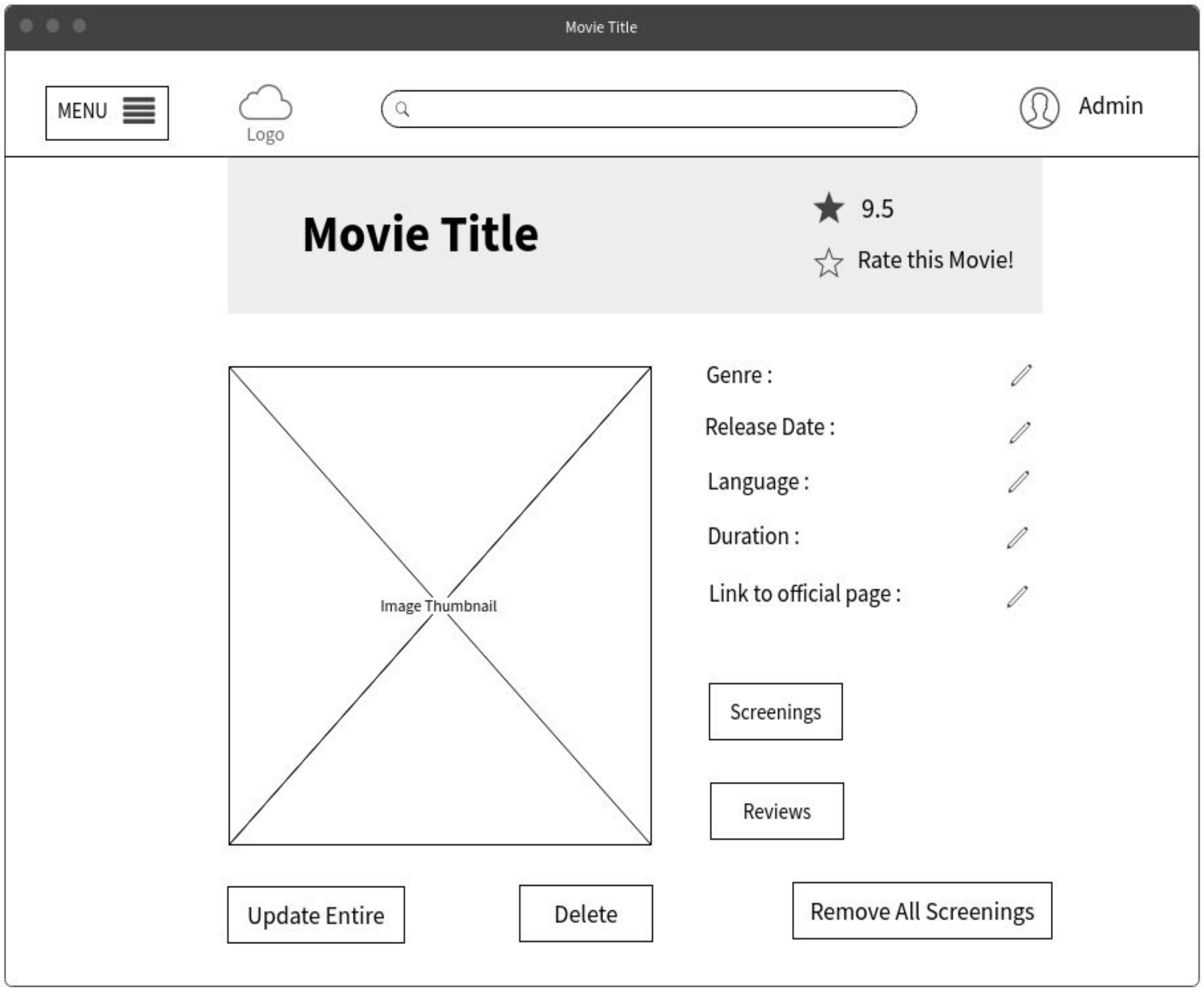
Movie Catalogue for Non Admins


Catalogue
MENU
Logo
Admin
Movie Catalogue
Sort By
Rating
Page 1 of 23
[✓ 10]
25
50
100
Add Movie
1. Thumbnail MOVIE TITLE Genre ★ 9.5 ☆ Rate this movie!
2. Thumbnail MOVIE TITLE Genre ★ 9.3 ☆ Rate this movie!
3. Thumbnail MOVIE TITLE Genre ★ 9.3 ☆ Rate this movie!
4. Thumbnail MOVIE TITLE Genre ★ 9.2 ☆ Rate this movie!
5. Thumbnail MOVIE TITLE Genre ★ 9.1 ☆ Rate this movie!
6. Thumbnail MOVIE TITLE Genre ★ 9.0 ☆ Rate this movie!
Genre Filter
✓ Action
Comedy
✓ Drama
Fantasy
Horror
✓ Mystery
Romance
Thriller
Language Filter
All

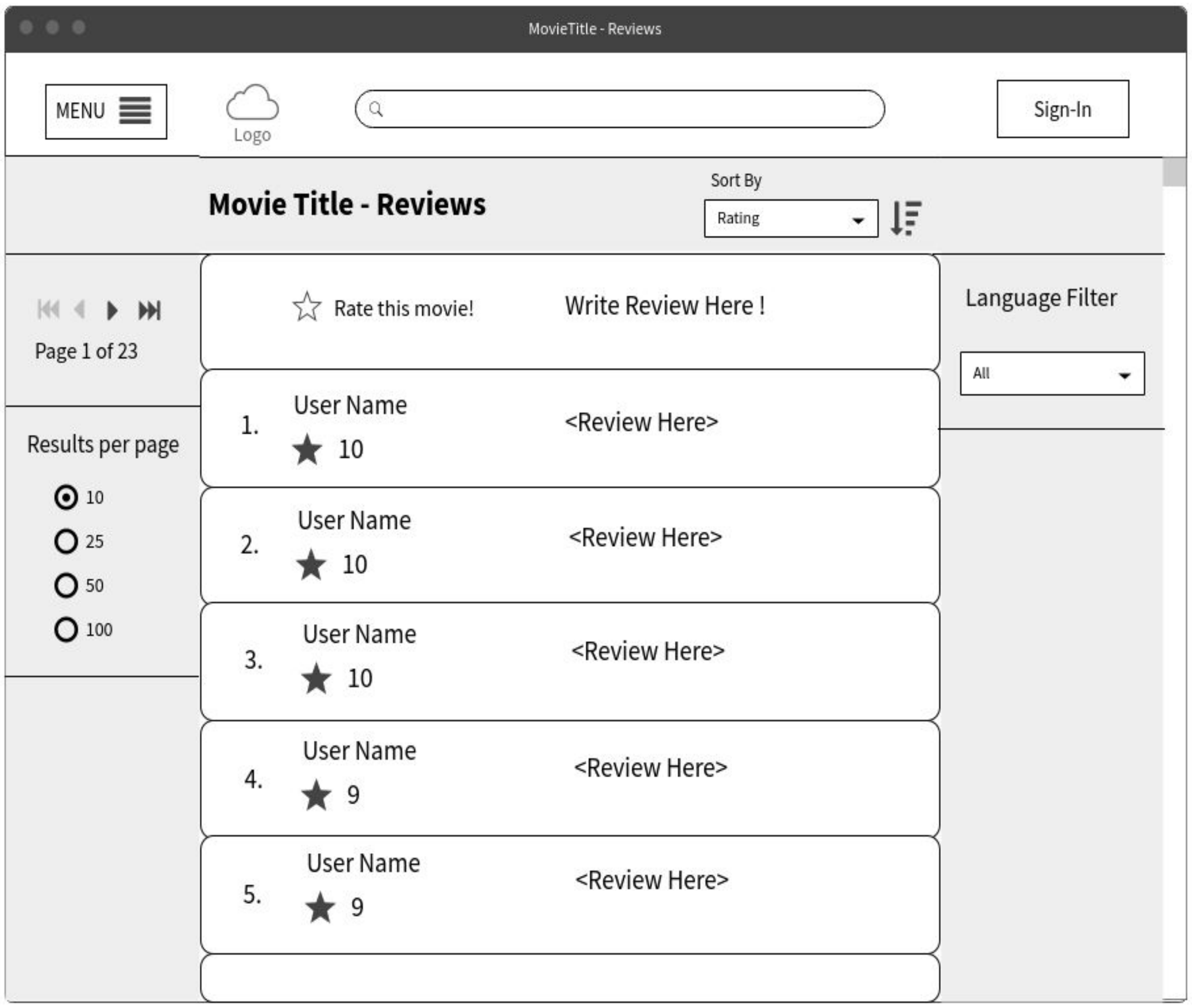
Movie Catalogue for Admins



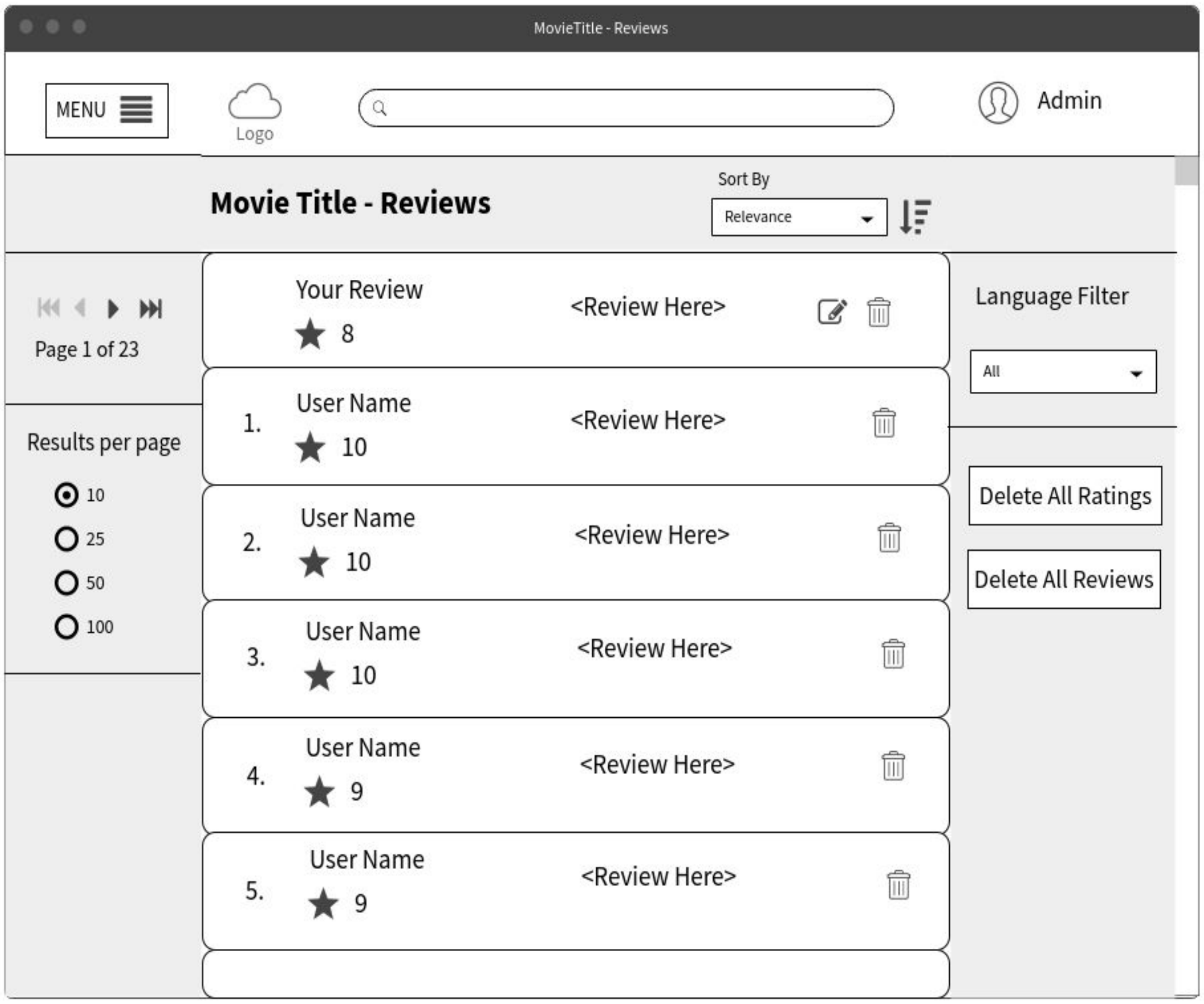
Movie View for Non Admins



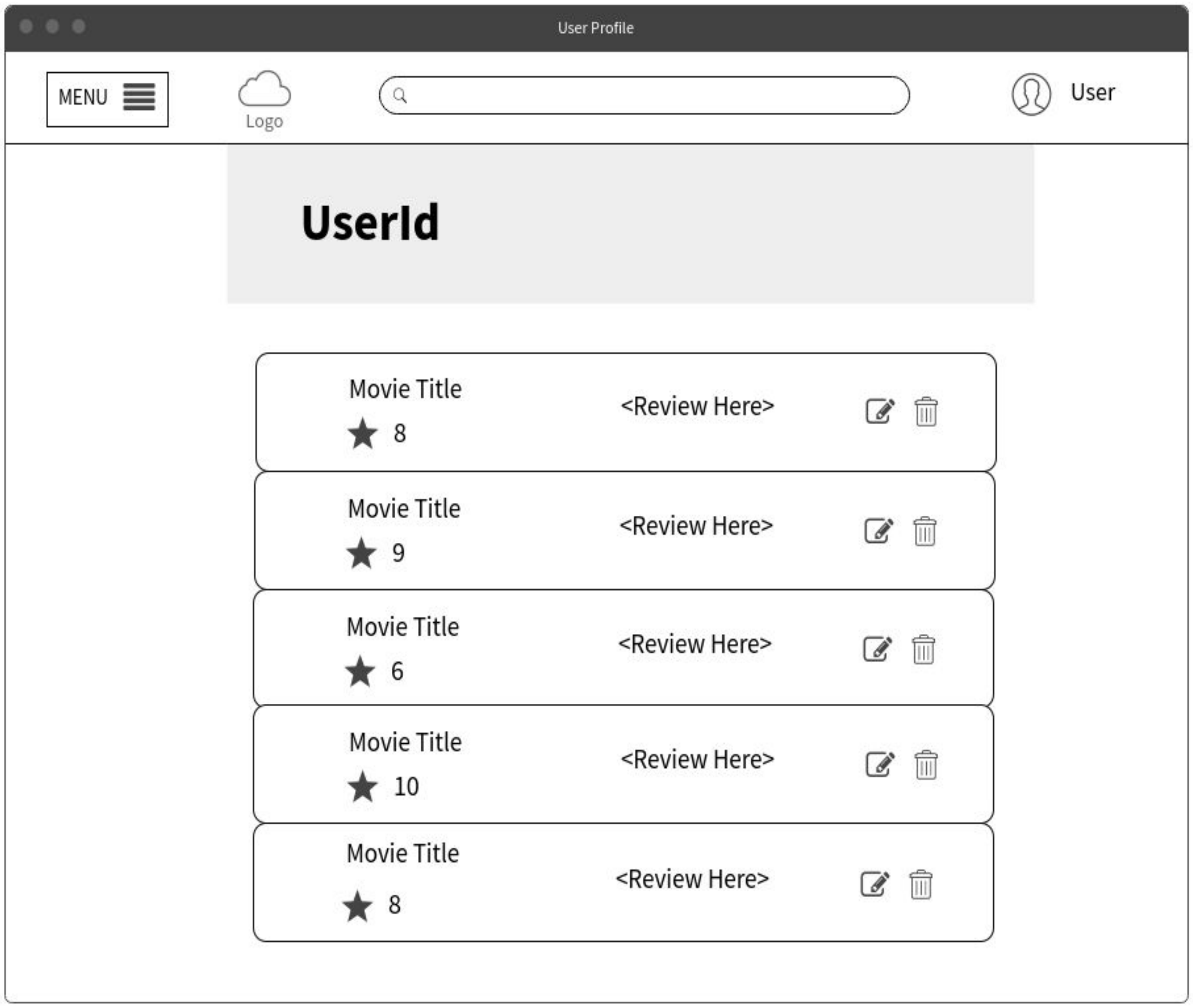
Movie View for Admins



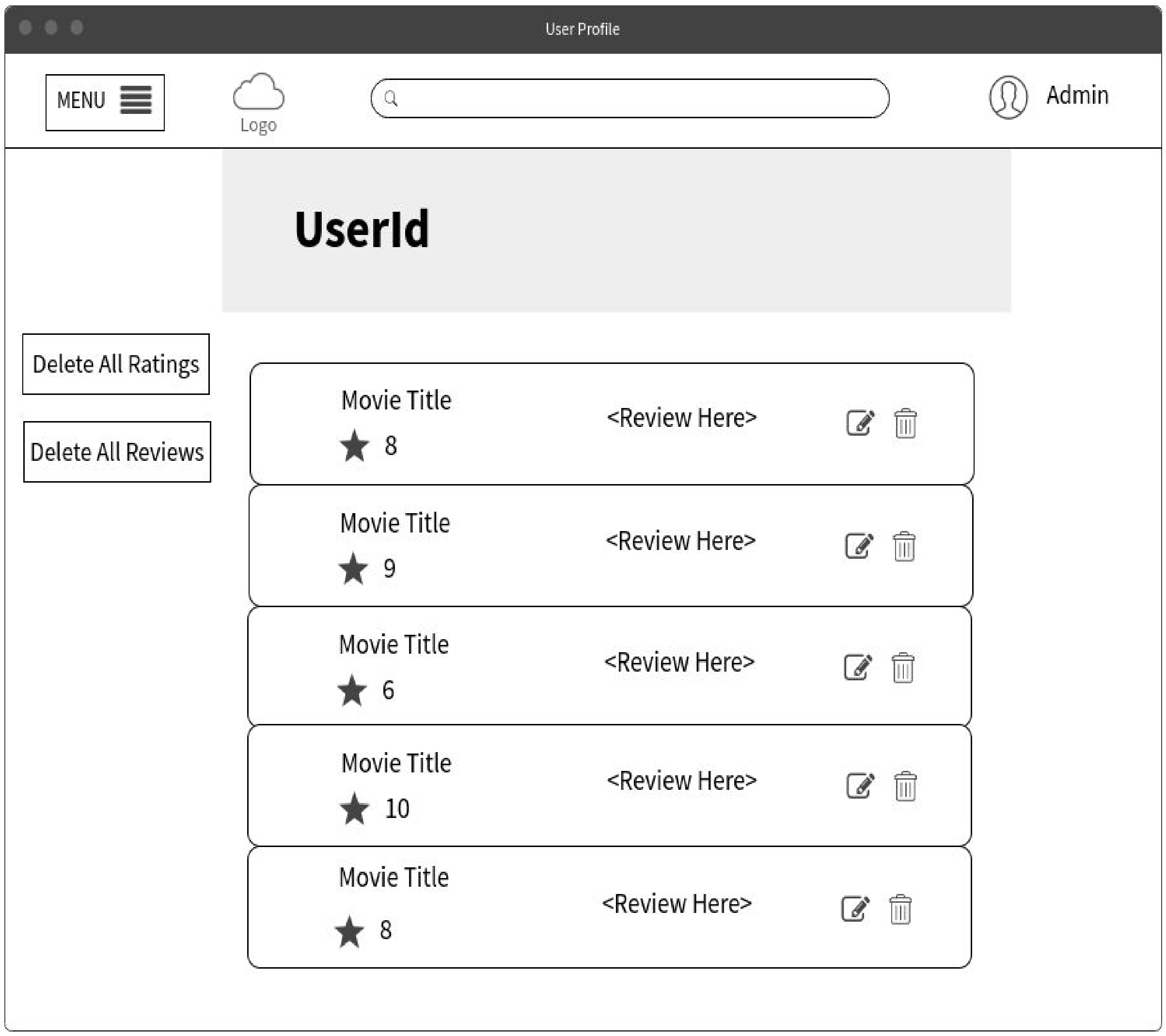
Reviews View for Non Admins



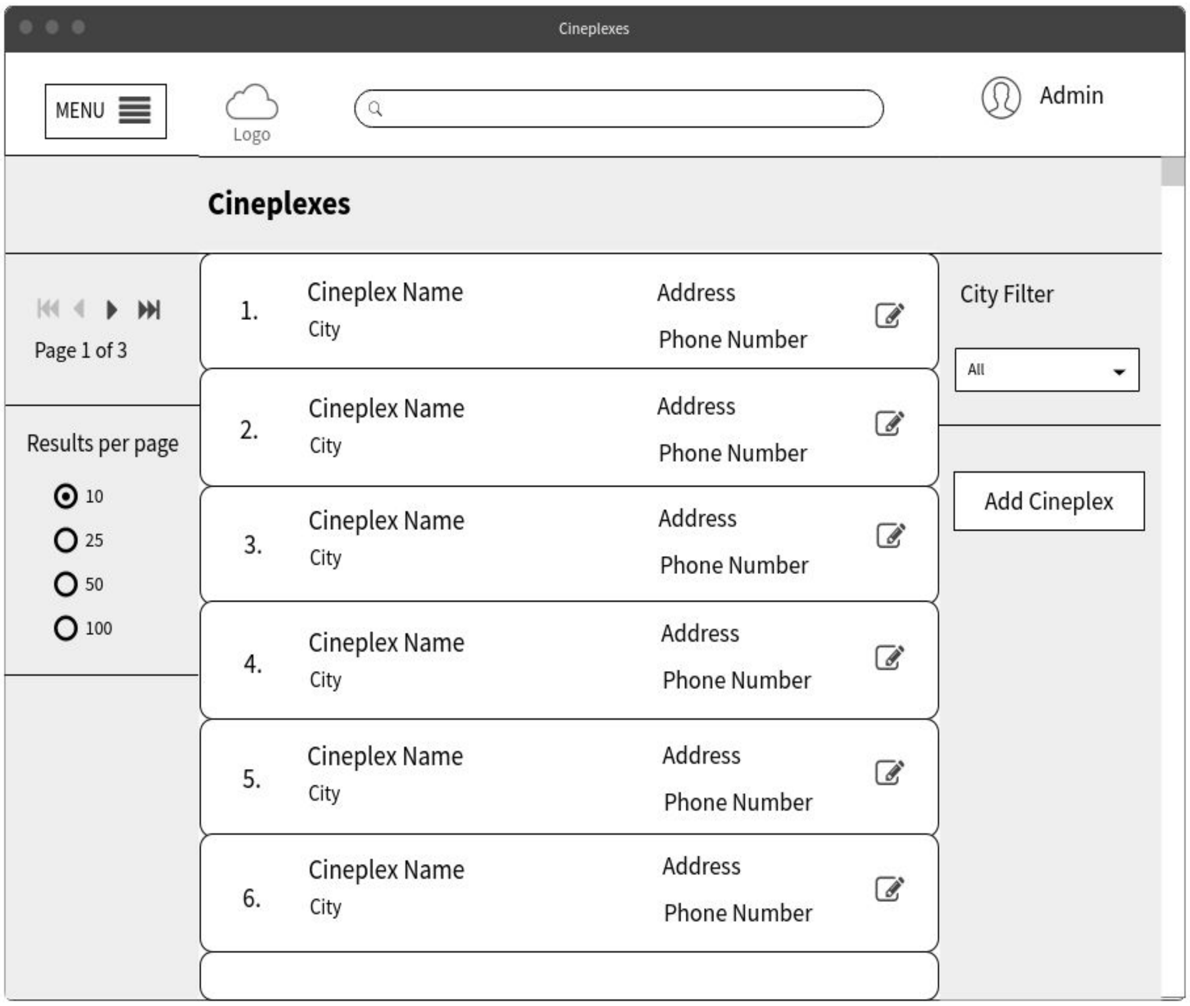
Reviews View For Admins



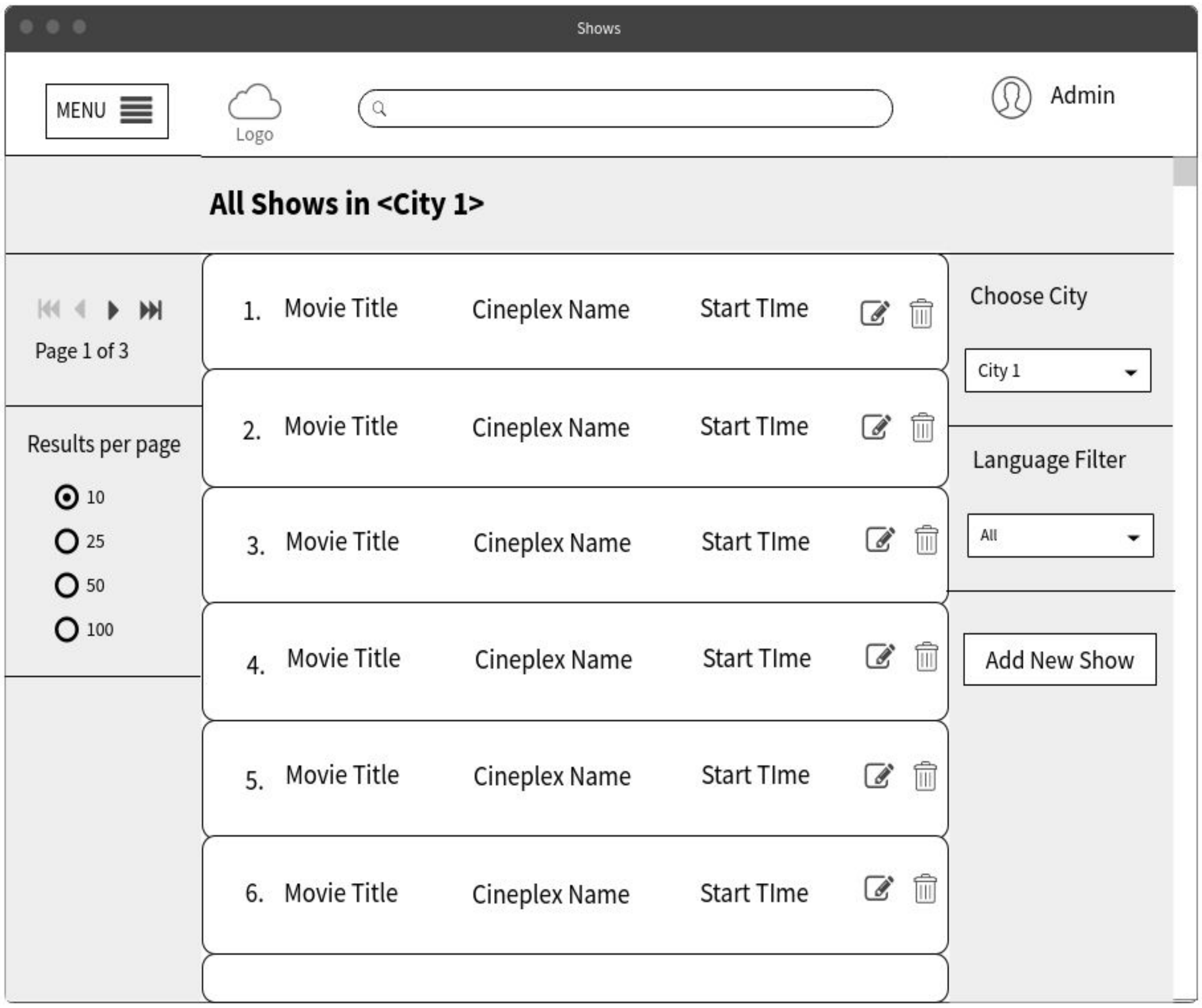
User Profile for the User



User Profile For Admins


Cineplexes
MENU
Logo
Admin
Cineplexes
Page 1 of 3
Results per page
10
25
50
100
1. Cineplex Name City Address Phone Number
2. Cineplex Name City Address Phone Number
3. Cineplex Name City Address Phone Number
4. Cineplex Name City Address Phone Number
5. Cineplex Name City Address Phone Number
6. Cineplex Name City Address Phone Number
City Filter
All
Add Cineplex

Cineplex View for Admins



All Shows in <City 1>

Page 1 of 3

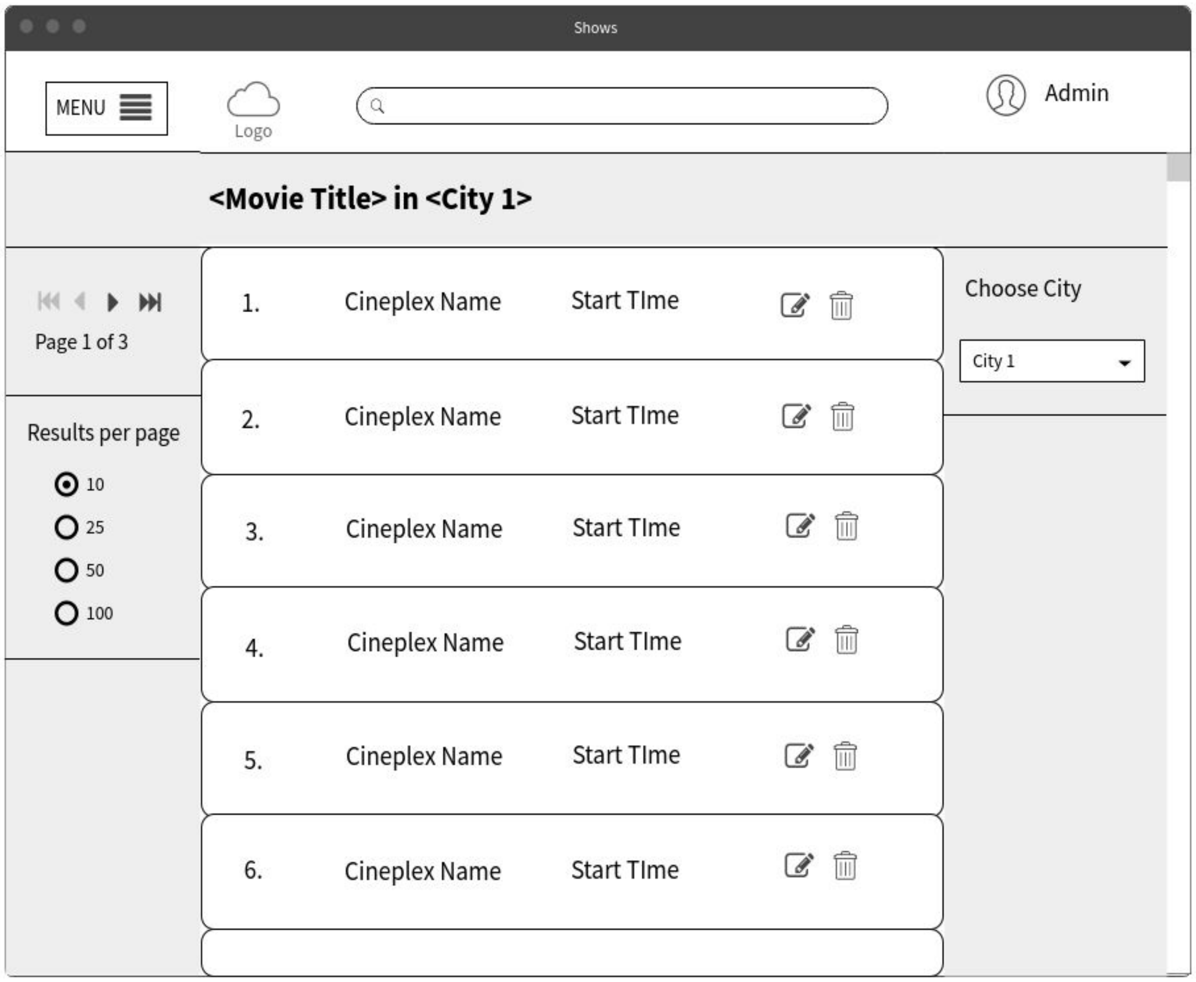
Results per page
[✓ 10] 25 50 100

Choose City: City 1

Language Filter: All

1.	Movie Title	Cineplex Name	Start TIme
2.	Movie Title	Cineplex Name	Start TIme
3.	Movie Title	Cineplex Name	Start TIme
4.	Movie Title	Cineplex Name	Start TIme
5.	Movie Title	Cineplex Name	Start TIme
6.	Movie Title	Cineplex Name	Start TIme

All Shows in City View for Admins


Shows
MENU
Logo
Admin
<Movie Title> in <City 1>
Page 1 of 3
Results per page
10
25
50
100
1. Cineplex Name Start TIme
2. Cineplex Name Start TIme
3. Cineplex Name Start TIme
4. Cineplex Name Start TIme
5. Cineplex Name Start TIme
6. Cineplex Name Start TIme
Choose City
City 1

All Screenings of a Movie View for Admins

4. TESTING AND AUTOMATION

For testing our UI ReactJS components, we use Jest and Enzyme.

Jest
Jest is a JavaScript Testing Framework compatible with TypeScript and acts as a test runner, assertion library, and mocking library.

Enzyme
Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output. Enzyme provides additional testing utilities

for testing.

Selenium WebDriver is a web automation framework that allows you to cross-test against various browsers. It is one of the most preferred testing tool-suite for automating web applications as it provides support for popular web browsers which makes it very powerful.

We use GoConvey for testing APIs for all the microservices in Golang. It has direct integration with ‘go test’ and provides a fully-automatic web UI test report.

We incorporate Continuous Integration and Continuous Deployment (CI/CD) by :

Implementing a Jenkins job for running unit tests for each commit
Implementing a Jenkins pipeline for functional testing
Implementing a Jenkins pipeline for deployment

5. DEPLOYMENT STRATEGY

For deploying our webapp, we make use of Amazon Elastic Compute Cloud. AWS EC2 supports Go and provides simple deployment with optimized computation and memory.

In case of high database requirements, we can shift our PostgreSQL database to more dedicated Amazon Relational Database Service.

We deploy the Go application as a binary as it will make deployment faster, especially if dependencies in consideration are large.