A

Project Report

On

“MOVIE DATABASE”

Submitted in partial fulfillment of
the requirements for the 6th Semester Sessional Examination of

BACHELOR OF TECHNOLOGY
IN

COMPUTER SCIENCE & ENGINEERING

By

SHAMBHU SHAH	**Reg no: 1801210558**
SHRABAN SHAH	**Reg no: 1801210559**
SATISH PANDIT	**Reg no: 1801210556**
KRISHNA YADAV	**Reg no: 1801210554**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GANDHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

GUNUPUR – 765022

2020 - 2021

ACKNOWLEDGEMENT

A special gratitude to our HOD Dr. Sanjay Kumar Kunwar,

Thanks to Department Project Coordinator Mr.Bhavani Sankar Panda,

Thanks to class teacher mam miss Jemarani Jaypuria

Thanks to Department staff members

Thanks to friends

Introduction

Project overview: Project purpose and scope? Product features.

>> In the current scenario, a moviegoer (user) has to visit more than one website to get the following basic movie information

- ❖ List of movies playing in theaters, upcoming movies, DVD/Blu-ray movies.
- ❖ IMDB Rating; Rotten Tomatoes Rating.
- ❖ Simple Plot, Cast & Crew, Genre, Year Released, Runtime.
- ❖ Show Times.
- ❖ Links to stream/rent the movie online, buy DVD/Blu-ray.
- ❖ Similar Movie Suggestions.

There are websites like www.imdb.com and www.rottentomatoes.com with rich amount of the aforementioned data but the user has to open at least 3-4 websites to view all the data. So, this project was started with the intention of developing a one-stop destination for the user to view all the data. The data from these websites was fetched by calling the APIs and putting them together in a dynamic web application named. In addition to these, MD also suggests the similar movies that might interest the user. MD application has a rich, user-friendly Graphical

User Interface design developed using React Js. The movie data is obtained from available APIs provided by IMDB, Rotten Tomatoes and other official API providers. The data, which is static for a particular movie (Eg. Cast, Plot, Poster etc.), is fetched from the APIs and use directly. The data that may vary with time such as Ratings, Show times etc. are fetched in real time by calling the respective APIs.

The purpose of movie database:

- ❖ Help users to find interesting movies easier
- ❖ Allow the user to search movies by typing in the key word of the movie
- ❖ Allow the user to search movies by typing in the name of director
- ❖ Allow the user to search movies by typing in the name of an actor.
- ❖ Allow the user to search movies by typing in the movie's name

System Analysis

1. User Requirements (SRS)

1.1Functional Requirement:

Index.js

The index page will have:

- ● Web site Title in the header
- ● Search button the text box will be where users input the text that will be searched for.

The search will be constrained by what is chosen in the combo box. The input text box will accept movies, directors, actors, and keywords. The user will choose a constraint in the combo box. Once this is done they will input text into the text box. If no text is put in the text box and the search button is pushed all of the movies, directors and actors will show up for the user to choose. The search button will be the button user’s press when a search is to be done. If text is in the text box the button will execute an action method. The action method will search the database for data in the form of movies, actors, or directors that relate to the text.

App.js

The App results page will have:

- Title header
- Search results

The search results will be shown below the title header. These results will relate to what the user input on the index page. The results will depend on what the user searches for. When the user chooses to input a keyword multiple actors, directors, or movies will show up depending on relatedness to the keyword. If no results were found the user will be informed that no results have been found in the database. If results are found they will be shown in the form of links. The links, when clicked, will direct the user to the page corresponding to the link. If the link pertains to a movie and the link is clicked a page for that specific movie will be shown. If the link pertains to an actor or director the user will be directed to the person page.

Hardware Requirement:

- Pentium IV or higher, (PIV-300GHz recommended)
- 256 MB RAM.
- 1 Gb hard free drive space.

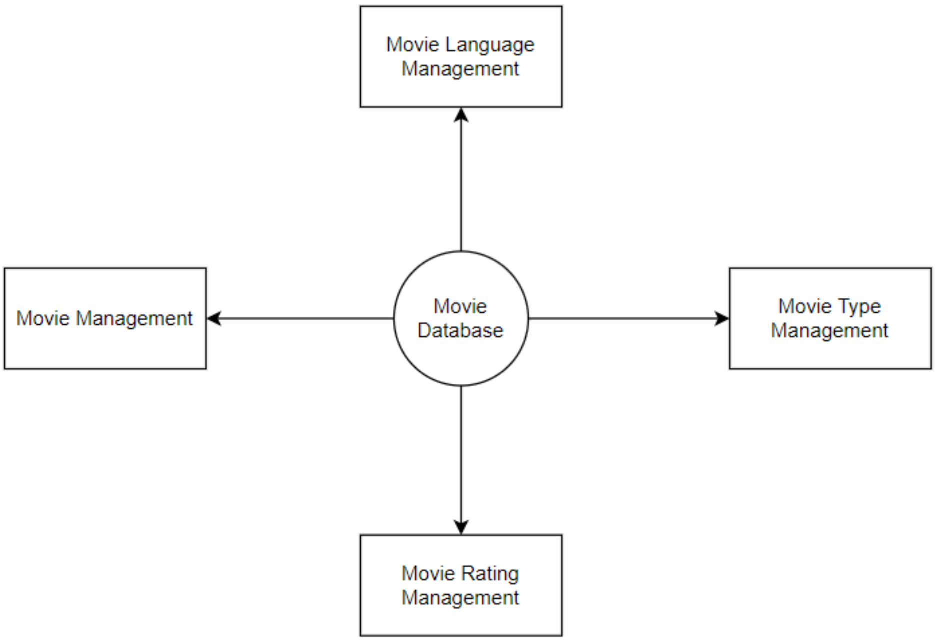
Software Requirement:

- Node. Js
- HTML, CSS
- JavaScript

System Design & Specification

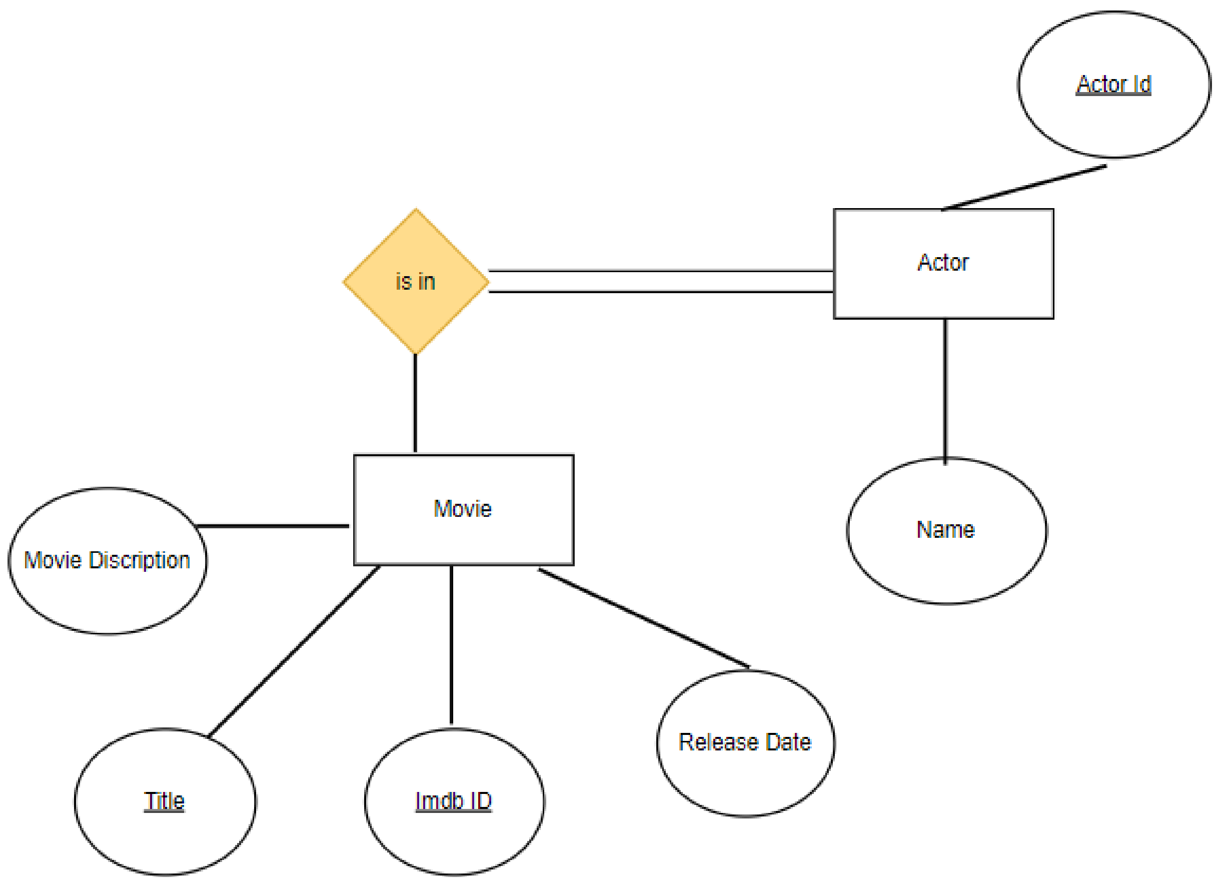
High Level Design(HLD)

Data flow diagram (DFD):



DFD of Movie Database

E-R Diagram:



UML

Use case:

: Search a movie title

1. On our Index page user enters desired movie title into text area, select ‘movie’ in drop down box. Program brings you to our search results page.

2. Once user has located correct movie, click the movie link. Program brings you to movie details page.

3. Movie details have all necessary movie information including directors, actors, and a synopsis. From movie details page, you can click on any link to relevant directors and actors for their personal page.

Coding

App.js :

```
import React, { useState } from 'react'
import axios from 'axios'

import Search from './components/Search'
import Results from './components/Results'
import Popup from './components/Popup'

function App() {
    const [state, setState] = useState({
        s: "",
        results: [],
        selected: {}
    });
    const apiurl = "http://www.omdbapi.com/?apikey=de4ddc6c";

    const search = (e) => {
        if (e.key === "Enter") {
            axios(apiurl + "&s=" + state.s).then(({ data }) => {
                let results = data.Search;

                setState(prevState => {
                    return {...prevState, results: results }
                })
            });
        }
    }

    const handleInput = (e) => {
        let s = e.target.value;

        setState(prevState => {
            return {...prevState, s: s }
        });
    }

    const openPopup = id => {
        axios(apiurl + "&i=" + id).then(({ data }) => {
            let result = data;

            console.log(result);

            setState(prevState => {
                return {...prevState, selected: result }
            });
```

```
       });
   }

   const closePopup = () => {
       setState(prevState => {
           return {...prevState, selected: {} }
       });
   }

   return (

   <div className = "App" >
       <header >
       <h1 > Movie Database < /h1> </header > <main >
       <
       Search handleInput = { handleInput }
       search = { search }
       />

       <
       Results results = { state.results }
       openPopup = { openPopup }
       />

       {
           (typeof state.selected.Title != "undefined") ? < Popup selected = {
state.selected }
           closePopup = { closePopup }
           /> : false} </main> </div>

       );
   }

   export default App;
```

```
import React from 'react'

import Result from './Result'

function Results({ results, openPopup }) {
    return (
        <section className = "results" > {
            results.map(result => ( <
                Result key = { result.imdbID }
                result = { result }
                openPopup = { openPopup }
                />
            ))
        }
        </section>
    )
}

export default Results
```

Conclusions & Limitation

The cost for making this site is totally zero. But it takes lots of hard work and dedication. We are working on this and making this site more efficient.

Challenges and Drawbacks:

- Fetching the data through API and putting them together is one of the biggest challenges faced in this project.
- Since the dynamic data is coming from the API in real time, the application is running a bit slow.

References:

1. http://www.omdbapi.com/
2. Chris Blakely YouTube