Inter Institutional Computer Centre

(An Autonomous Department)

Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur

# *Certificate*

*This is to certify that the project report on* ***"Movix – Movie Application"*** *submitted by* ***Saurabh Mahendra Pakhale*** *has successfully completed the project in partial fulfillment of* ***"Master in Computer Application"*** *of Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur for the academic session 2023-24.*

*This is the original software project carried out by him/her under my supervision and guidance and has undergone the requisite duration prescribed by Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur for the project work.*

| | |
|---|---|
| *Project Guide* | *Director* |
| *Internal Examiner* | *External Examiner* |

*Date: -*

# Contents

# Company Profile

## Digitron Software’s & Technology

**Overview:** Digitron Software’s & Technology is an IT solutions provider based in Nagpur, Maharashtra, India. They offer a comprehensive range of services including software development, website development, digital marketing, mobile app development, and software testing.

**Mission and Vision:** Digitron Software’s & Technology aims to empower businesses through technological advancements. Their mission is to deliver tailored IT solutions that meet the unique needs of each client, fostering growth and efficiency. The vision is to be a global leader in the IT industry by continuously innovating and adapting to technological changes.

**Services Offered:**

1. **Software Development:**
    - Custom software solutions designed to address specific business requirements.
    - Integration of advanced technologies to enhance functionality and performance.
    - Full lifecycle development from concept to deployment and maintenance.
2. **Website Development:**
    - Creation of responsive and user-friendly websites.
    - Focus on design aesthetics, functionality, and SEO optimization.
    - Ongoing support and maintenance to ensure optimal performance.
3. **Digital Marketing:**
    - Comprehensive digital marketing strategies including SEO, SEM, SMM, and PPC.
    - Data-driven approach to enhance online presence and reach target audiences.
    - Analytics and reporting to measure effectiveness and ROI.
4. **Mobile App Development:**
    - Development of mobile applications for Android, iOS, and cross-platform.
    - User-centric design to enhance user experience and engagement.
    - Integration of the latest technologies and trends in mobile development.
5. **Software Testing:**
    - Rigorous testing processes to ensure software quality and reliability.
    - Functional, performance, security, and usability testing.
    - Automation and manual testing services to meet diverse client needs.

**Industry Focus:** Digitron Software’s & Technology serves a wide range of industries, including:

- **Education:** E-learning platforms, school management systems, and educational apps.
- **Travel:** Booking systems, travel management software, and mobile apps.
- **E-commerce:** E-commerce websites, mobile shopping apps, and inventory management systems.
- **Healthcare:** Healthcare management systems, telemedicine apps, and patient portals.
- **Others:** Tailored solutions for finance, real estate, manufacturing, and more.

**Technological Expertise:**

- Proficiency in programming languages like Java, Python, C#, PHP, and JavaScript.
- Expertise in frameworks and platforms such as Angular, React, Node.js, and Laravel.
- Knowledge of mobile development platforms including Android Studio, Xcode, and Flutter.
- Experience with cloud services like AWS, Azure, and Google Cloud.

**Client Approach:** Digitron Software's & Technology prides itself on a client-centric approach, emphasizing collaboration and transparency. The company follows a detailed project management process, ensuring timely delivery and client satisfaction. Regular updates and feedback loops are integral to their methodology, fostering strong client relationships.

**Achievements and Certifications:**

- Recognized for excellence in software development and innovation.
- Certifications in quality management and software development standards.
- Successful completion of numerous projects across various industries, showcasing a diverse portfolio.

**Contact Information:**

- **Office Address:** 10A, Shaktimata Nagar, Kharbi Rd, Nagpur, Maharashtra 440024, India
- **Phone:** +91 7709266577
- **Email:** info@digitronsoftwares.com
- **Website:** https://digitronsoftwares.com

# Project Profile

- **Project Name:** Movix
- **Project Description:** Movix is a dynamic web application that allows users to explore and discover a vast library of movies and TV shows. Content is fetched from The Movie Database (TMDB) API, providing an extensive and up-to-date catalogue. Movix features trending movies updated daily and weekly, offers search functionality, and provides detailed information including cast, ratings, and trailers. Users can enjoy this content without needing to log in or sign up.

- **Key Features:**
    - Trending Movies: Displays lists of trending movies updated both daily and weekly.
    - Search Functionality: Allows users to search for specific movies.
    - Detailed Movie View: Redirects to a dedicated page upon clicking a movie, showing detailed information such as cast, ratings, and trailers.
    - Movie Trailers: Integrates with YouTube to play movie trailers directly within the application.
    - Responsive Design: Ensures a seamless user experience across different devices and screen sizes.
    - 
- **Technologies Used:**
    - Frontend Framework: React.js (for building a component-based, efficient user interface)
    - Build Tool: Vite (provides a fast development server and optimized build process)
    - State Management: Redux Toolkit (simplifies state management and asynchronous logic)
    - HTTP Client: Axios (facilitates making API requests to TMDB)
    - Additional Libraries:
        - Day.js (efficient date manipulation and formatting)
        - React Circular Progressbar (circular progress indicators)
        - React Icons (variety of icons)
        - React Infinite Scroll Component (continuous content loading)
        - React Lazy Load Image Component (image loading optimization)
        - React Player (video playback functionality for trailers)
        - React Router Dom (client-side routing for smooth navigation)
        - React Select (customizable select menus)
        - Sass (styling with features like variables and mixins)
    - Development Tools:
        - ESLint (ensures code quality and consistency)
        - Vite Plugins (includes @vitejs/plugin-react for React integration)
- **Project Structure:**
    - **Main Directories:**
        - src (source code)
            - components (reusable components)
            - pages (different page components)
            - redux (Redux setup)
            - services (Axios configuration and API requests)

        - styles (Sass files)
        - utils (utility functions and constants)
    - **Main Files:**
        - App.js (root component defining structure and routing)
        - index.js (application entry point)
        - store.js (Redux store configuration)
        -
- **Installation and Setup:**
    - Clone the repository:

      ```
      git clone https://github.com/saurabhsp/movix.git
      ```

    - Navigate to the project directory:

      ```
      cd movix
      ```

    - Install dependencies:

      ```
      npm install
      ```

    - Start the development server:

      ```
      npm run saurya
      ```

- **Usage:**
    - Development Server: Use `npm run saurya` to start the development server.
    - Build: Use `npm run build` to build the project for production.
    - Lint: Use `npm run lint` to run ESLint and check for code issues.
    - Preview: Use `npm run preview` to preview the production build locally.
- **Conclusion**

Movix is a modern and efficient web application designed to provide users with an extensive and up-to-date catalog of movies and TV shows. It leverages the power of React, Vite, Redux, and Axios to offer a seamless and engaging user experience. The project's emphasis on performance, state management, and user experience makes it a standout example of a modern web application.

# Introduction

## Problem Definition

In the digital age, users crave easily accessible and up-to-date information on movies and TV shows. They seek platforms where they can effortlessly browse trending content, search for specific titles, and access detailed information like cast, ratings, and trailers. Existing platforms often require logins, hindering quick access. Additionally, users expect a seamless experience across all devices, which current solutions don't always guarantee.

## Present System In Use

Several platforms currently provide movie and TV show information, including popular streaming services and dedicated movie databases. These platforms often offer a wide range of features:

- Detailed movie and TV show information
- User reviews and ratings
- Personalized recommendations
- Trailers and promotional videos
- Cast and crew details

However, most require user authentication, deterring casual users who prefer quick access without account creation and management.

## Flaws In Present System

Despite the functionalities provided, existing platforms have several notable flaws:

1. **Mandatory Login/Signup:** Many platforms require account creation and login, inconvenient for quick access.
2. **Complex Interfaces:** Some platforms have overly complex interfaces that make navigation and information retrieval cumbersome.
3. **Performance Issues:** High latency and slow loading times can frustrate users, particularly on platforms not optimized for performance.
4. **Lack of Customization:** Users often have limited options to personalize their viewing experience.
5. **Inconsistent Experience:** User experience can be inconsistent across different devices, leading to usability issues on mobile or tablet devices.

## Need For New System

A new system is needed to address these flaws and enhance the user experience by providing:

1. **No Login/Signup Requirement:** Immediate access to content without user authentication.

2. **User-Friendly Interface:** A simple and intuitive interface that facilitates easy navigation and information retrieval.
3. **High Performance:** Fast loading times and a responsive design ensure a smooth user experience.
4. **Detailed Information:** Comprehensive details about movies and TV shows, including cast, ratings, and trailers.
5. **Consistency Across Devices:** A responsive design ensures a consistent and seamless experience on all devices.

## Proposed System

Movix is designed to meet these needs by offering a modern and efficient web application for browsing and discovering movies and TV shows. The proposed system will provide:

1. **No Login/Signup Requirement:** Users can access all features and information without needing to log in or create an account.
2. **Trending Content:** Daily and weekly updates of trending movies to keep users informed about the latest popular content.
3. **Search Functionality:** An efficient search feature allows users to find specific movies quickly and easily.
4. **Detailed Movie Views:** Dedicated pages for each movie provide detailed information, including cast, ratings, and trailers.
5. **YouTube Integration:** Embedded trailers from YouTube allow users to watch previews directly within the application.
6. **Responsive Design:** A design ensures a consistent and optimal experience across all devices, from desktops to mobile phones.

By addressing the flaws of current systems and focusing on user needs, Movix aims to provide a superior platform for movie and TV show enthusiasts. The use of modern technologies like React, Vite, Redux, and Axios ensures a robust, fast, and engaging application that meets the demands of today's users.

# Analysis

## Feasibility Study for a Movie/TV Show Explorer Application

A feasibility study assesses the practicality of building and maintaining a proposed system. In the context of a Movie/TV Show Explorer application, the study would typically encompass these key areas:

### 1. Technical Feasibility:

- **Infrastructure Requirements:** This involves evaluating the technological infrastructure needed to support the application, including servers, databases, and external APIs (like TMDb and YouTube).
- **Development Tools:** Here, we assess the feasibility of using specific technologies (e.g., React, Redux, Node.js) based on project requirements and the team's expertise.
- **Integration Feasibility:** This assesses the practicality of integrating various components like search functionality, video playback, and data retrieval from external APIs.

### 2. Operational Feasibility:

- **User Acceptance:** This involves evaluating how users (the target audience interested in movies/TV shows) would accept and use the application.
- **Maintenance:** We assess the feasibility of maintaining the application post-launch, including updates, bug fixes, and scaling to accommodate future growth.
- **Training Needs:** This involves identifying if there's a need for user training or documentation to effectively use the application.

### 3. Economic Feasibility:

- **Cost-Benefit Analysis:** This evaluates the costs associated with development, hosting, maintenance, and support against the expected benefits (such as increased user engagement, revenue from advertisements or subscriptions).
- **Return on Investment (ROI):** We assess the potential ROI based on revenue models like advertising, subscription fees, or affiliate marketing.

### 4. Legal and Compliance Feasibility:

- **Data Privacy:** This ensures compliance with data protection regulations (e.g., GDPR, CCPA) when handling user data and integrating third-party APIs.
- **Copyright and Licensing:** We ensure compliance with copyright laws when displaying movie/TV show content, posters, trailers, etc.
- **Terms of Service:** This involves reviewing and complying with the terms of service for APIs and data sources used in the application.

## Required Analysis

The required analysis for the Movie/TV Show Explorer project would involve these key steps:

### 1. User Requirements Analysis:

- **User Persona:** We create user personas (e.g., casual viewer, movie enthusiast, researcher) to understand user needs and behaviours. This helps tailor the application's features and functionalities to their preferences.
- **Use Cases:** Different use cases are documented (e.g., searching for movies by genre, watching trailers, exploring recommendations) to define the functional requirements for the application.

## 2. Functional Requirements:

- **Feature List:** This involves defining various features such as search functionality, video playback, user authentication (if applicable), genre filtering, and recommendation systems.
- **System Interfaces:** We describe how different modules (frontend, backend, third-party APIs) interact with each other to ensure a seamless user experience.
- **Performance Requirements:** Specific performance metrics (e.g., response time, number of concurrent users) that the system should meet are outlined.

## 3. Non-Functional Requirements:

- **Usability:** Here, we specify usability requirements, including accessibility standards, responsive design for different devices, and an intuitive user interface design to ensure a user-friendly experience.
- **Security:** Defining security requirements is crucial. This includes data encryption, secure authentication mechanisms, and protection against common web vulnerabilities like malicious code injection or unauthorized actions.
- **Scalability:** Outlining requirements for scaling the application as user traffic increases is important. This might involve load balancing and database scalability to handle a growing user base.

## 4. Constraints Analysis:

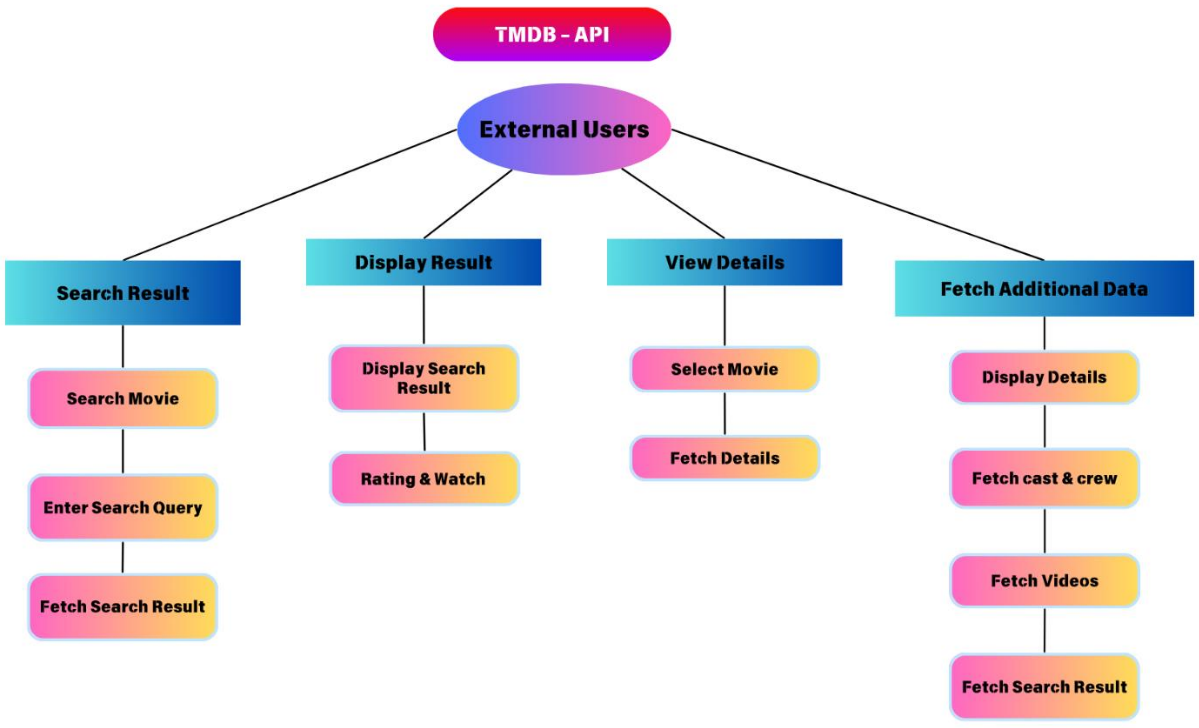
- **Technology Constraints:** This involves identifying limitations related to the chosen technologies for development (e.g., compatibility issues, learning curve for the development team).
- **Time Constraints:** Here, we assess project timelines and deadlines, including milestones for development, testing, and deployment phases, to ensure on-time delivery.
- **Resource Constraints:** This considers limitations related to budget, team size, expertise, and availability of third-party APIs or data sources.

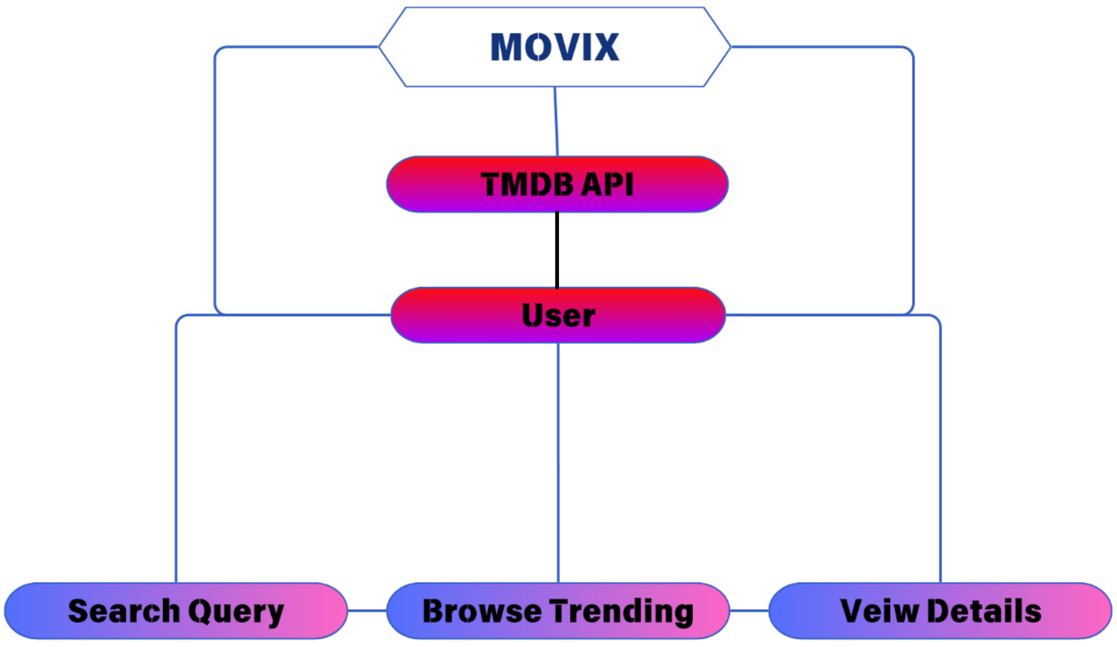
## 5. Risk Analysis:

- **Risk Identification:** This involves identifying potential risks (e.g., technical challenges, API reliability, data security breaches) that could negatively impact project success

## Context Flow Diagram (CFD)

A Context Flow Diagram (CFD) illustrates the interaction between the Movix system and external entities. Here's a basic representation:





# Design

## System Flow Diagram (Visual Representation)

```
Start SearchResult Component
    |--- Initialize state variables:
    |   - data: Holds fetched search results from the API.
    |   - pageNum: Tracks the current page number for pagination.
    |   - loading: Indicates if data is being fetched.
    |   - query: Stores the user-entered search query.
    |--- Fetch Initial Data (fetchInitialData function):
    |   |
    |   |--- Set loading state to true.
    |    |--- Fetch initial search results (/search/multi?query=${query}) using
    fetchDataFromApi utility function.
    |   |      |
    |   |      |--- Set loading state to false.
    |   |      |--- Update data state with fetched results.
    |   |      |--- Increment pageNum for pagination.
    |--- Handle User Interactions:
    |    - No specific user interactions are handled beyond the initial search
    query.
    |--- Render UI Components Based on State (data, loading):
    |   |
    |   |--- Display loading spinner if loading is true.
    |   |--- Render search results using InfiniteScroll component:
    |       |
    |       |--- If data.results.length > 0:
    |           |--- Render MovieCard components for each item in data.results.
    |         |--- Implement infinite scroll to load more data as user scrolls down.
    |       |
    |       |--- If data.results.length === 0:
    |         |--- Display "Results not found" message.
End SearchResult Component
```

## Data Dictionary

The data dictionary provides a detailed description of the main data entities used in the Explore and SearchResult components, including their attributes and usage contexts.

1. **Explore Component Data Dictionary**
    - data: Object containing fetched data from the API.
    - results: Array of objects representing movies or TV shows.
    - id: Unique identifier of the item.
    - title (or name for TV shows): Title or name of the item.
    - overview: Brief description of the plot or content.
    - poster_path: URL path to the poster image.
    - backdrop_path: URL path to the backdrop image.
    - vote_average: Average rating of the item.
    - release_date (or first_air_date for TV shows): Release date or first air date.
    - media_type: Type of media (movie, TV show).
    - pageNum: Integer indicating the current page number for pagination.
    - loading: Boolean indicating whether data is being fetched (true) or not (false).
    - genre: Array of selected genre objects for filtering.
        - id: Unique identifier of the genre.
        - name: Name of the genre.
    - sortby: Object representing the selected sorting criteria.
        - value: String representing the sorting criteria (e.g., "popularity.desc", "vote_average.asc").
        - label: Display label for the sorting option.

- genresData: Object containing genre data fetched from /genre/${mediaType}/list.
- genres: Array of genre objects.
- id: Unique identifier of the genre.
- name: Name of the genre.

2. **SearchResult Component Data Dictionary**
    - data: Object containing fetched search results from the API
    - results: Array of objects representing search results.
    - id: Unique identifier of the item.
    - title (or name for TV shows): Title or name of the item.
    - overview: Brief description of the plot or content.
    - poster_path: URL path to the poster image.
    - backdrop_path: URL path to the backdrop image.
    - vote_average: Average rating of the item.
    - release_date (or first_air_date for TV shows): Release date or first air date.
    - media_type: Type of media (movie, TV show).
    - pageNum: Integer indicating the current page number for pagination.
    - loading: Boolean indicating whether data is being fetched (true) or not (false).
    - query: String representing the user-entered search query.

## Explanation

- **Explore Component:** Manages browsing movies or TV shows with filters (genres) and sorting options (by popularity, rating, release date). Initial data fetch occurs on component mount and updates occur when users select filters or sorting criteria. The component utilizes infinite scrolling for pagination.
- **SearchResult Component:** Displays search results based on user-entered queries. It fetches data dynamically and updates the UI accordingly. Pagination is also implemented using infinite scrolling to load additional results as the user scrolls down.

These components utilize React hooks (useState, useEffect) for state management and external libraries (react-select, react-infinite-scroll-component) for enhanced UI functionality. The data structures (data, pageNum, loading, etc.) ensure efficient handling of API responses and seamless user experience during browsing and searching operations.

## Source Code:

## Carousel

```
import React, { useRef } from "react";
import {
   BsFillArrowLeftCircleFill,
   BsFillArrowRightCircleFill,
} from "react-icons/bs";
import { useNavigate } from "react-router-dom";
import { useSelector } from "react-redux";
import dayjs from "dayjs";
import ContentWrapper from "../contentWrapper/ContentWrapper";
import Img from "../lazyLoadImage/Img";
import PosterFallback from "../../assets/no-poster.png";
import CircleRating from "../circleRating/CircleRating";
import Genres from "../genres/Genres";
import "./style.scss";
const Carousel = ({ data, loading, endpoint, title }) => {
   const carouselContainer = useRef();
   const { url } = useSelector((state) => state.home);
   const navigate = useNavigate();
   const navigation = (dir) => {
      const container = carouselContainer.current;

      const scrollAmount =
         dir === "left"
            ? container.scrollLeft - (container.offsetWidth + 20)
            : container.scrollLeft + (container.offsetWidth + 20);
      container.scrollTo({
         left: scrollAmount,
         behavior: "smooth",
      });
```

```
};
const skItem = () => {
    return (
        <div className="skeletonItem">
            <div className="posterBlock skeleton"></div>
            <div className="textBlock">
                <div className="title skeleton"></div>
                <div className="date skeleton"></div>
            </div>
        </div>
    );
};
return (
    <div className="carousel">
        <ContentWrapper>
            {title && <div className="carouselTitle">{title}</div>}
            <BsFillArrowLeftCircleFill
                className="carouselLeftNav arrow"
                onClick={() => navigation("left")}
            />
            <BsFillArrowRightCircleFill
                className="carouselRighttNav arrow"
                onClick={() => navigation("right")}
            />
            {!loading ? (
                <div className="carouselItems" ref={carouselContainer}>
                    {data?.map((item) => {
                        const posterUrl = item.poster_path
                            ? url.poster + item.poster_path
                            : PosterFallback;
```

```
return (
  <div
    key={item.id}
    className="carouselItem"
    onClick={() =>
      navigate(
        `/${item.media_type || endpoint}/${
          item.id
        }`
      )
    }
  >
    <div className="posterBlock">
      <Img src={posterUrl} />
      <CircleRating
        rating={item.vote_average.toFixed(
          1
        )}
      />
      <Genres
        data={item.genre_ids.slice(0, 2)}
      />
    </div>
    <div className="textBlock">
      <span className="title">
        {item.title || item.name}
      </span>
      <span className="date">
        {dayjs(item.release_date || item.first_air_date).format(
          "MMM D, YYYY"
```

```
                                        )}
                                    </span>
                                </div>
                            </div>
                        );
                    })}
                </div>
            ) : (
                <div className="loadingSkeleton">
                    {skItem()}
                    {skItem()}
                    {skItem()}
                    {skItem()}
                    {skItem()}
                </div>
            )}
        </ContentWrapper>
    </div>
  );
};
export default Carousel;
```

## circleRating.jsx

```
import React from "react";
import { CircularProgressbar, buildStyles } from "react-circular-progressbar";
import "react-circular-progressbar/dist/styles.css";
import "./style.scss";
const CircleRating = ({ rating }) => {
  return (
    <div className="circleRating">
```

```
            <CircularProgressbar
                value={rating}
                maxValue={10}
                text={rating}
                styles={buildStyles({
                    pathColor:
                        rating < 5 ? "red" : rating < 7 ? "orange" : "green",
                })}
            />
        </div>
    );
};
export default CircleRating;
```

## contentWrapper.jsx

```
import React from "react";
import "./style.scss";
const ContentWrapper = ({ children }) => {
    return <div className="contentWrapper">{children}</div>;
};
export default ContentWrapper;
```

## footer.jsx

```
import React from "react";
import {
    FaFacebookF,
    FaInstagram,
    FaTwitter,
    FaLinkedin,
    FaGithub,
    FaJsSquare,
    FaJs,
    FaCode,
} from "react-icons/fa";
import ContentWrapper from "../contentWrapper/ContentWrapper";
import "./style.scss";
const Footer = () => {
    return (
        <footer className="footer">
            <ContentWrapper>
                <ul className="menuItems">
                    <li className="menuItem" href="">Terms Of Use</li>
                    <li className="menuItem">Privacy-Policy</li>
                    <li className="menuItem">About</li>
                    <li className="menuItem">Blog</li>
                    <li className="menuItem">FAQ</li>
                </ul>
                <div className="infoText">
                    Explore endless movie magic with Movix: Your gateway to cinematic bliss! 🎬✨
#MovieMarathon #DiscoverMovix
                </div>
                <div className="socialIcons">
                    <a href="https://github.com/saurabhsp" target="_blank" className="icon">
                        <FaGithub />
                    </a>
                            <a href="https://replit.com/@SAURABHPAKHALE" target="_blank"
className="icon">
                        <FaCode />
                    </a>
                          <a href="https://www.instagram.com/_saurabh_pakhale_/" target="_blank"
className="icon">
```

```
                    <FaInstagram />
                </a>
                            <a  href="https://www.linkedin.com/in/saurabhsmp/"  target="_blank"
className="icon">
                    <FaLinkedin />
                </a>
            </div>
        </ContentWrapper>
    </footer>
  );
};
export default Footer;
```

## Genres

```
import React from "react";
import { useSelector } from "react-redux";
import "./style.scss";
const Genres = ({ data }) => {
  const { genres } = useSelector((state) => state.home);
  return (
    <div className="genres">
      {data?.map((g) => {
        if (!genres[g]?.name) return;
        return (
          <div key={g} className="genre">
            {genres[g]?.name}
          </div>
        );
      })}
    </div>
  );
};
export default Genres;
```

## Header.jsx

```
import React, { useState, useEffect } from "react";
import { HiOutlineSearch } from "react-icons/hi";
import { SlMenu } from "react-icons/sl";
import { VscChromeClose } from "react-icons/vsc";
import { useNavigate, useLocation } from "react-router-dom";
import "./style.scss";
import ContentWrapper from "../contentWrapper/ContentWrapper";
import logo from "../../assets/movix-logo.svg";
const Header = () => {
    const [show, setShow] = useState("top");
    const [lastScrollY, setLastScrollY] = useState(0);
    const [mobileMenu, setMobileMenu] = useState(false);
    const [query, setQuery] = useState("");
    const [showSearch, setShowSearch] = useState("");
    const navigate = useNavigate();
    const location = useLocation();
    useEffect(() => {
        window.scrollTo(0, 0);
    }, [location]);
    const controlNavbar = () => {
        if (window.scrollY > 200) {
            if (window.scrollY > lastScrollY && !mobileMenu) {
                setShow("hide");
            } else {
                setShow("show");
            }
        } else {
```

```
        setShow("top");
      }
      setLastScrollY(window.scrollY);
    };
    useEffect(() => {
      window.addEventListener("scroll", controlNavbar);
      return () => {
        window.removeEventListener("scroll", controlNavbar);
      };
    }, [lastScrollY]);
    const searchQueryHandler = (event) => {
      if (event.key === "Enter" && query.length > 0) {
        navigate(`/search/${query}`);
        setTimeout(() => {
          setShowSearch(false);
        }, 1000);
      }
    };
    const openSearch = () => {
      setMobileMenu(false);
      setShowSearch(true);
    };
    const openMobileMenu = () => {
      setMobileMenu(true);
      setShowSearch(false);
    };
    const navigationHandler = (type) => {
      if (type === "movie") {
        navigate("/explore/movie");
      } else {
        navigate("/explore/tv");
      }
      setMobileMenu(false);
    };
    return (
      <header className={`header ${mobileMenu ? "mobileView" : ""} ${show}`}>
        <ContentWrapper>
          <div className="logo" onClick={() => navigate("/")}>
            <img src={logo} alt="" />
          </div>
          <ul className="menuItems">
            <li
              className="menuItem"
              onClick={() => navigationHandler("movie")}
            >
              Movies
            </li>
            <li
              className="menuItem"
              onClick={() => navigationHandler("tv")}
            >
              TV Shows
            </li>
            <li className="menuItem">
              <HiOutlineSearch onClick={openSearch} />
            </li>
          </ul>

          <div className="mobileMenuItems">
            <HiOutlineSearch onClick={openSearch} />
```

```
                {mobileMenu ? (
                    <VscChromeClose onClick={() => setMobileMenu(false)} />
                ) : (
                    <SlMenu onClick={openMobileMenu} />
                )}
            </div>
        </ContentWrapper>
        {showSearch && (
            <div className="searchBar">
                <ContentWrapper>
                    <div className="searchInput">
                        <input
                            type="text"
                            placeholder="Search for a movie or tv show...."
                            onChange={(e) => setQuery(e.target.value)}
                            onKeyUp={searchQueryHandler}
                        />
                        <VscChromeClose
                            onClick={() => setShowSearch(false)}
                        />
                    </div>
                </ContentWrapper>
            </div>
        )}
    </header>
  );
};
export default Header;
```

- lazyLoadImage

```
import React from "react";
import { LazyLoadImage } from "react-lazy-load-image-component";
import "react-lazy-load-image-component/src/effects/blur.css";
const Img = ({ src, className }) => {
  return (
    <LazyLoadImage
      className={className || ""}
      alt=""
      effect="blur"
      src={src}
    />
  );
};
export default Img;
```

## movieCard.jsx

```
import React from "react";
import dayjs from "dayjs";
import { useNavigate } from "react-router-dom";
import { useSelector } from "react-redux";
import "./style.scss";
import Img from "../lazyLoadImage/Img";
import CircleRating from "../circleRating/CircleRating";
import Genres from "../genres/Genres";
import PosterFallback from "../../assets/no-poster.png";
const MovieCard = ({ data, fromSearch, mediaType }) => {
  const { url } = useSelector((state) => state.home);
  const navigate = useNavigate();
  const posterUrl = data.poster_path
    ? url.poster + data.poster_path
    : PosterFallback;
  return (
    <div
      className="movieCard"
      onClick={() =>
        navigate(`/${data.media_type || mediaType}/${data.id}`)
      }
    >
      <div className="posterBlock">
        <Img className="posterImg" src={posterUrl} />
        {!fromSearch && (
          <React.Fragment>
            <CircleRating rating={data.vote_average.toFixed(1)} />
            <Genres data={data.genre_ids.slice(0, 2)} />
          </React.Fragment>
        )}
      </div>
      <div className="textBlock">
        <span className="title">{data.title || data.name}</span>
        <span className="date">
          {dayjs(data.release_date).format("MMM D, YYYY")}
        </span>
      </div>
    </div>
  );
};
export default MovieCard;
```

## Spinner.jsx

```
import React from "react";
import "./style.scss";
```

```
const Spinner = ({ initial }) => {
    return (
        <div className={`loadingSpinner ${initial ? "initial" : ""}`}>
            <svg className="spinner" viewBox="0 0 50 50">
                <circle
                    className="path"
                    cx="25"
                    cy="25"
                    r="20"
                    fill="none"
                    strokeWidth="5"
                ></circle>
            </svg>
        </div>
    );
};
export default Spinner;
```

## switchTabs.jsx

```
import React, { useState } from "react";
import "./style.scss";
const SwitchTabs = ({ data, onTabChange }) => {
    const [selectedTab, setSelectedTab] = useState(0);
    const [left, setLeft] = useState(0);
    const activeTab = (tab, index) => {
        setLeft(index * 100);
        setTimeout(() => {
            setSelectedTab(index);
        }, 300);
        onTabChange(tab, index);
    };
    return (
        <div className="switchingTabs">
            <div className="tabItems">
                {data.map((tab, index) => (
                    <span
                        key={index}
                        className={`tabItem ${
                            selectedTab === index ? "active" : ""
                        }`}
                        onClick={() => activeTab(tab, index)}
                    >
                        {tab}
                    </span>
                ))}
                <span className="movingBg" style={{ left }} />
            </div>
        </div>
    );
};
export default SwitchTabs;
```

## videoPopup.jsx

```
import React from "react";
import ReactPlayer from "react-player/youtube";
import "./style.scss";
const VideoPopup = ({ show, setShow, videoId, setVideoId }) => {
    const hidePopup = () => {
        setShow(false);
        setVideoId(null);
    };
    return (
        <div className={`videoPopup ${show ? "visible" : ""}`}>
            <div className="opacityLayer" onClick={hidePopup}></div>
            <div className="videoPlayer">
                <span className="closeBtn" onClick={hidePopup}>
                    Close
                </span>
                <ReactPlayer
                    url={`https://www.youtube.com/watch?v=${videoId}`}
                    controls
                    width="100%"
                    height="100%"
                    // playing={true}
                />
            </div>
        </div>
    );
};
export default VideoPopup;
```

- **hooks**
    - **useFetch**

```
import { useEffect, useState } from "react";
import { fetchDataFromApi } from "../utils/api";
const useFetch = (url) => {
    const [data, setData] = useState(null);
    const [loading, setLoading] = useState(null);
    const [error, setError] = useState(null);


    useEffect(() => {
        setLoading("loading...");
        setData(null);
        setError(null);


        fetchDataFromApi(url)
            .then((res) => {
                setLoading(false);
                setData(res);
            })
            .catch((err) => {
                setLoading(false);
                setError("Something went wrong!");
            });
    }, [url]);
    return { data, loading, error };
};
export default useFetch;
```

- Pages
    - 404
    - Details
        - Carousel
        - Cast
        - detailsBanner
        - videoSection
    - Explore
    - Home
    - searchResult

```
import React from "react";
import "./style.scss";
import ContentWrapper from "../../components/contentWrapper/ContentWrapper";
const PageNotFound = () => {
    return (
        <div className="pageNotFound">
            <ContentWrapper>
                <span className="bigText">404</span>
                <span className="smallText">Page not found!</span>
            </ContentWrapper>
        </div>
    );
};
export default PageNotFound;

recommendation.jsx
import React from "react";
import Carousel from "../../../components/carousel/Carousel";
import useFetch from "../../../hooks/useFetch";
const Recommendation = ({ mediaType, id }) => {
    const { data, loading, error } = useFetch(
        `/${mediaType}/${id}/recommendations`
    );
    return (
        <Carousel
            title="Recommendations"
            data={data?.results}
            loading={loading}
            endpoint={mediaType}
        />
    );
};
export default Recommendation;
```

## Similar.jsx

```
import React from "react";
import Carousel from "../../../components/carousel/Carousel";
import useFetch from "../../../hooks/useFetch";
const Recommendation = ({ mediaType, id }) => {
    const { data, loading, error } = useFetch(
        `/${mediaType}/${id}/recommendations`
    );
    return (
        <Carousel
            title="Recommendations"
            data={data?.results}
            loading={loading}
            endpoint={mediaType}
        />
    );
};
export default Recommendation;
```

## Cast

```
import React from "react";
import { useSelector } from "react-redux";
import "./style.scss";
import ContentWrapper from "../../../components/contentWrapper/ContentWrapper";
import Img from "../../../components/lazyLoadImage/Img";
import avatar from "../../../assets/avatar.png";
const Cast = ({ data, loading }) => {
    const { url } = useSelector((state) => state.home);
    const skeleton = () => {
        return (
            <div className="skItem">
                <div className="circle skeleton"></div>
                <div className="row skeleton"></div>
                <div className="row2 skeleton"></div>
            </div>
        );
    };
```

```
return (
    <div className="castSection">
        <ContentWrapper>
            <div className="sectionHeading">Top Cast</div>
            {!loading ? (
                <div className="listItems">
                    {data?.map((item) => {
                        let imgUrl = item.profile_path
                            ? url.profile + item.profile_path
                            : avatar;
                        return (
                            <div key={item.id} className="listItem">
                                <div className="profileImg">
                                    <Img src={imgUrl} />
                                </div>
                                <div className="name">{item.name}</div>
                                <div className="character">
                                    {item.character}
                                </div>
                            </div>
                        );
                    })}
                </div>
            ) : (
                <div className="castSkeleton">
                    {skeleton()}
                    {skeleton()}
                    {skeleton()}
                    {skeleton()}
                    {skeleton()}
                    {skeleton()}
                </div>
            )}
        </ContentWrapper>
    </div>
  );
};
export default Cast;
```

## DetailsBanner.jsx

```
import React, { useState } from "react";
import { useParams } from "react-router-dom";
import { useSelector } from "react-redux";
import dayjs from "dayjs";
import "./style.scss";
import ContentWrapper from "../../../components/contentWrapper/ContentWrapper";
import useFetch from "../../../hooks/useFetch";
import Genres from "../../../components/genres/Genres";
import CircleRating from "../../../components/circleRating/CircleRating";
import Img from "../../../components/lazyLoadImage/Img.jsx";
import PosterFallback from "../../../assets/no-poster.png";
import { PlayIcon } from "../Playbtn";
import VideoPopup from "../../../components/videoPopup/VideoPopup";
const DetailsBanner = ({ video, crew }) => {
  const [show, setShow] = useState(false);
  const [videoId, setVideoId] = useState(null);
  const { mediaType, id } = useParams();
  const { data, loading } = useFetch(`/${mediaType}/${id}`);
  const { url } = useSelector((state) => state.home);
  const _genres = data?.genres?.map((g) => g.id);
const director = crew?.filter((f) => f.job === "Director");
const writer = crew?.filter(
  (f) => f.job === "Screenplay" || f.job === "Story" || f.job === "Writer"
);
const toHoursAndMinutes = (totalMinutes) => {
  const hours = Math.floor(totalMinutes / 60);
  const minutes = totalMinutes % 60;
  return `${hours}h${minutes > 0 ? ` ${minutes}m` : ""}`;
};
return (
  <div className="detailsBanner">
    {!loading ? (
      <>
        {!!data && (
          <React.Fragment>
            <div className="backdrop-img">
              <Img src={url.backdrop + data.backdrop_path} />
            </div>
            <div className="opacity-layer"></div>
            <ContentWrapper>
              <div className="content">
                <div className="left">
                  {data.poster_path ? (
                    <Img
                      className="posterImg"
                      src={
                        url.backdrop +
                        data.poster_path
                      }
                    />
        ) : (
            <Img
                className="posterImg"
                src={PosterFallback}
            />
        )}
    </div>
    <div className="right">
        <div className="title">
            {`${
                data.name || data.title
            } (${dayjs(
                data?.release_date
            ).format("YYYY")})`}
        </div>
        <div className="subtitle">
            {data.tagline}
        </div>
        <Genres data={_genres} />
        <div className="row">
            <CircleRating
                rating={data.vote_average.toFixed(
                    1
                )}
            />
            <div
                className="playbtn"
                onClick={() => {
                    setShow(true);
                    setVideoId(video.key);
```

```
          }}
        >
          <PlayIcon />
          <span className="text">
            Watch Trailer
          </span>
        </div>
      </div>
      <div className="overview">
        <div className="heading">
          Overview
        </div>
        <div className="description">
          {data.overview}
        </div>
      </div>
      <div className="info">
        {data.status && (
          <div className="infoItem">
            <span className="text bold">
              Status:{" "}
            </span>
            <span className="text">
              {data.status}
            </span>
          </div>
        )}
        {data.release_date && (
          <div className="infoItem">
            <span className="text bold">
```

```
                    Release Date:{" "}
                </span>
                <span className="text">
                    {dayjs(
                        data.release_date
                    ).format("MMM D, YYYY")}
                </span>
            </div>
        )}
        {data.runtime && (
            <div className="infoItem">
                <span className="text bold">
                    Runtime:{" "}
                </span>
                <span className="text">
                    {toHoursAndMinutes(
                        data.runtime
                    )}
                </span>
            </div>
        )}
    </div>

    {director?.length > 0 && (
        <div className="info">
            <span className="text bold">
                Director:{" "}
            </span>
            <span className="text">
                {director?.map((d, i) => (
                <span key={i}>
                    {d.name}
                    {director.length -
                        1 !==
                        i && ", "}
                </span>
            ))}
        </span>
    </div>
)}

{writer?.length > 0 && (
    <div className="info">
        <span className="text bold">
            Writer:{" "}
        </span>
        <span className="text">
            {writer?.map((d, i) => (
                <span key={i}>
                    {d.name}
                    {writer.length -
                        1 !==
                        i && ", "}
                </span>
            ))}
        </span>
    </div>
)}

{data?.created_by?.length > 0 && (
                    <div className="info">
                        <span className="text bold">
                            Creator:{" "}
                        </span>
                        <span className="text">
                            {data?.created_by?.map(
                                (d, i) => (
                                    <span key={i}>
                                        {d.name}
                                        {data
                                            ?.created_by
                                            .length -
                                            1 !==
                                            i && ", "}
                                    </span>
                                )
                            )}
                        </span>
                    </div>
                )}
            </div>
        </div>
        <VideoPopup
            show={show}
            setShow={setShow}
            videoId={videoId}
            setVideoId={setVideoId}
        />
    </ContentWrapper>
</React.Fragment>
```

```
                )}
            </>
        ) : (
            <div className="detailsBannerSkeleton">
                <ContentWrapper>
                    <div className="left skeleton"></div>
                    <div className="right">
                        <div className="row skeleton"></div>
                        <div className="row skeleton"></div>
                        <div className="row skeleton"></div>
                        <div className="row skeleton"></div>
                        <div className="row skeleton"></div>
                        <div className="row skeleton"></div>
                        <div className="row skeleton"></div>
                    </div>
                </ContentWrapper>
            </div>
        )}
    </div>
  );
};
export default DetailsBanner;
```

## videoSection.jsx

- Details
- Playbtn

```
import React from "react";
import { useParams } from "react-router-dom";
import "./style.scss";

import useFetch from "../../hooks/useFetch";
import DetailsBanner from "./detailsBanner/DetailsBanner";
import Cast from "./cast/Cast";
import VideosSection from "./videosSection/VideosSection";
import Similar from "./carousels/Similar";
import Recommendation from "./carousels/Recommendation";

const Details = () => {
    const { mediaType, id } = useParams();
    const { data, loading } = useFetch(`/${mediaType}/${id}/videos`);
    const { data: credits, loading: creditsLoading } = useFetch(
        `/${mediaType}/${id}/credits`
    );

    return (
        <div>
            <DetailsBanner video={data?.results?.[0]} crew={credits?.crew} />
            <Cast data={credits?.cast} loading={creditsLoading} />
            <VideosSection data={data} loading={loading} />
            <Similar mediaType={mediaType} id={id} />
            <Recommendation mediaType={mediaType} id={id} />
        </div>
    );
};

export default Details;
```

## Playbtn.jsx

```
export const PlayIcon = () => {

    return (

        <svg

            version="1.1"

            xmlns="http://www.w3.org/2000/svg"

            xmlnsXlink="http://www.w3.org/1999/xlink"

            x="0px"

            y="0px"

            width="80px"

            height="80px"

            viewBox="0 0 213.7 213.7"

            enableBackground="new 0 0 213.7 213.7"
```

```
         xmlSpace="preserve"
      >
         <polygon
            className="triangle"
            fill="none"
            strokeWidth="7"
            strokeLinecap="round"
            strokeLinejoin="round"
            strokeMiterlimit="10"
            points="73.5,62.5 148.5,105.8 73.5,149.1 "
         ></polygon>
         <circle
            className="circle"
            fill="none"
            strokeWidth="7"
            strokeLinecap="round"
            strokeLinejoin="round"
            strokeMiterlimit="10"
            cx="106.8"
            cy="106.8"
            r="103.3"
         ></circle>
      </svg>
   );
};
```

## Explore.jsx

```
import React, { useState, useEffect } from "react";
import { useParams } from "react-router-dom";
import InfiniteScroll from "react-infinite-scroll-component";
import Select from "react-select";
import "./style.scss";
import useFetch from "../../hooks/useFetch";
import { fetchDataFromApi } from "../../utils/api";
import ContentWrapper from "../../components/contentWrapper/ContentWrapper";
import MovieCard from "../../components/movieCard/MovieCard";
import Spinner from "../../components/spinner/Spinner";
let filters = {};
const sortbyData = [
   { value: "popularity.desc", label: "Popularity Descending" },
   { value: "popularity.asc", label: "Popularity Ascending" },
   { value: "vote_average.desc", label: "Rating Descending" },
   { value: "vote_average.asc", label: "Rating Ascending" },
   {
      value: "primary_release_date.desc",
      label: "Release Date Descending",
   },
   { value: "primary_release_date.asc", label: "Release Date Ascending" },
   { value: "original_title.asc", label: "Title (A-Z)" },
];
const Explore = () => {
   const [data, setData] = useState(null);
   const [pageNum, setPageNum] = useState(1);
   const [loading, setLoading] = useState(false);
   const [genre, setGenre] = useState(null);
   const [sortby, setSortby] = useState(null);
   const { mediaType } = useParams();
   const { data: genresData } = useFetch(`/genre/${mediaType}/list`);
   const fetchInitialData = () => {
      setLoading(true);
```

```
        fetchDataFromApi(`/discover/${mediaType}`, filters).then((res) => {
            setData(res);
            setPageNum((prev) => prev + 1);
            setLoading(false);
        });
    };
    const fetchNextPageData = () => {
        fetchDataFromApi(
            `/discover/${mediaType}?page=${pageNum}`,
            filters
        ).then((res) => {
            if (data?.results) {
                setData({
                    ...data,
                    results: [...data?.results, ...res.results],
                });
            } else {
                setData(res);
            }
            setPageNum((prev) => prev + 1);
        });
    };
    useEffect(() => {
        filters = {};
        setData(null);
        setPageNum(1);
        setSortby(null);
        setGenre(null);
        fetchInitialData();
    }, [mediaType]);
```

```
const onChange = (selectedItems, action) => {
    if (action.name === "sortby") {
        setSortby(selectedItems);
        if (action.action !== "clear") {
            filters.sort_by = selectedItems.value;
        } else {
            delete filters.sort_by;
        }
    }
    if (action.name === "genres") {
        setGenre(selectedItems);
        if (action.action !== "clear") {
            let genreId = selectedItems.map((g) => g.id);
            genreId = JSON.stringify(genreId).slice(1, -1);
            filters.with_genres = genreId;
        } else {
            delete filters.with_genres;
        }
    }
    setPageNum(1);
    fetchInitialData();
};
return (
    <div className="explorePage">
        <ContentWrapper>
            <div className="pageHeader">
                <div className="pageTitle">
                    {mediaType === "tv"
                        ? "Explore TV Shows"
                        : "Explore Movies"}
                </div>
                <div className="filters">
                    <Select
                        isMulti
                        name="genres"
                        value={genre}
                        closeMenuOnSelect={false}
                        options={genresData?.genres}
                        getOptionLabel={(option) => option.name}
                        getOptionValue={(option) => option.id}
                        onChange={onChange}
                        placeholder="Select genres"
                        className="react-select-container genresDD"
                        classNamePrefix="react-select"
                    />
                    <Select
                        name="sortby"
                        value={sortby}
                        options={sortbyData}
                        onChange={onChange}
                        isClearable={true}
                        placeholder="Sort by"
                        className="react-select-container sortbyDD"
                        classNamePrefix="react-select"
                    />
                </div>
            </div>
            {loading && <Spinner initial={true} />}
            {!loading && (
                <>
```

```
                        {data?.results?.length > 0 ? (
                            <InfiniteScroll
                                className="content"
                                dataLength={data?.results?.length || []}
                                next={fetchNextPageData}
                                hasMore={pageNum <= data?.total_pages}
                                loader={<Spinner />}
                            >
                                {data?.results?.map((item, index) => {
                                    if (item.media_type === "person") return;
                                    return (
                                        <MovieCard
                                            key={index}
                                            data={item}
                                            mediaType={mediaType}
                                        />
                                    );
                                })}
                            </InfiniteScroll>
                        ) : (
                            <span className="resultNotFound">
                                Sorry, Results not found!
                            </span>
                        )}
                    </>
                )}
            </ContentWrapper>
        </div>
    );
};
```

```
export default Explore;
```

- **Home**
    - **heroBanner.jsx**
    - **popular.jsx**
    - **topRated.jsx**
    - **trending.jsx**

## heroBanner.jsx

```
import React, { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import { useSelector } from "react-redux";
import "./style.scss";

import useFetch from "../../../hooks/useFetch";

import Img from "../../../components/lazyLoadImage/Img";
import ContentWrapper from "../../../components/contentWrapper/ContentWrapper";

const HeroBanner = () => {
    const [background, setBackground] = useState("");
    const [query, setQuery] = useState("");
    const navigate = useNavigate();
    const { url } = useSelector((state) => state.home);
    const { data, loading } = useFetch("/movie/upcoming");

    useEffect(() => {
        const bg =
            url.backdrop +
        data?.results?.[Math.floor(Math.random() * 20)]?.backdrop_path;
    setBackground(bg);
}, [data]);

const search = () => {
    if (query.length > 0) {
        navigate(`/search/${query}`);
    }
};

const searchQueryHandler = (event) => {
    if (event.key === "Enter") {
        search();
    }
};

return (
    <div className="heroBanner">
        {!loading && (
            <div className="backdrop-img">
                <Img src={background} />
            </div>
        )}

        <div className="opacity-layer"></div>
        <ContentWrapper>
            <div className="heroBannerContent">
                <span className="title">Welcome </span>
                <span className="subTitle">
                    Millions of movies, TV shows and people to discover.
                    Explore now {">>"}
                </span>
                <div className="searchInput">
                    <input
                        type="text"
                        placeholder="Search for a movie or tv show...."
                        value={query}
                        onChange={(e) => setQuery(e.target.value)}
                        onKeyUp={searchQueryHandler}
                    />
                    <button onClick={search}>Search</button>
                </div>
            </div>
        </ContentWrapper>
    </div>
  );
};
export default HeroBanner;
```

## popular.jsx

```
import React, { useState } from "react";
import Carousel from "../../../components/carousel/Carousel";
import ContentWrapper from "../../../components/contentWrapper/ContentWrapper";
import SwitchTabs from "../../../components/switchTabs/SwitchTabs";

import useFetch from "../../../hooks/useFetch";
const Popular = () => {
    const [endpoint, setEndpoint] = useState("movie");
    const { data, loading } = useFetch(`/${endpoint}/popular`);
    const onTabChange = (tab) => {
        setEndpoint(tab === "Movies" ? "movie" : "tv");
    };
    return (
        <div className="carouselSection">
            <ContentWrapper>
                <span className="carouselTitle">What's Popular</span>
                <SwitchTabs
                    data={["Movies", "TV Shows"]}
                    onTabChange={onTabChange}
                />
            </ContentWrapper>
            <Carousel
                data={data?.results}
                loading={loading}
                endpoint={endpoint}
            />
        </div>
    );
};

export default Popular;
```

## topRated.jsx

```
import React, { useState } from "react";

import Carousel from "../../../components/carousel/Carousel";
import ContentWrapper from "../../../components/contentWrapper/ContentWrapper";
import SwitchTabs from "../../../components/switchTabs/SwitchTabs";

import useFetch from "../../../hooks/useFetch";
```

```
const TopRated = () => {
    const [endpoint, setEndpoint] = useState("movie");

    const { data, loading } = useFetch(`/${endpoint}/top_rated`);

    const onTabChange = (tab) => {
        setEndpoint(tab === "Movies" ? "movie" : "tv");
    };

    return (
        <div className="carouselSection">
            <ContentWrapper>
                <span className="carouselTitle">Top Rated</span>
                <SwitchTabs
                    data={["Movies", "TV Shows"]}
                    onTabChange={onTabChange}
                />
            </ContentWrapper>
            <Carousel
                data={data?.results}
                loading={loading}
                endpoint={endpoint}
            />
        </div>
    );
};
export default TopRated;
```

## trending.jsx

```
import React from "react";


import "./style.scss";
import HeroBanner from "./heroBanner/HeroBanner";
import Trending from "./trending/Trending";
import Popular from "./popular/Popular";
import TopRated from "./topRated/TopRated";
const Home = () => {
    return (
        <div className="homePage">
            <HeroBanner />
            <Trending />
            <Popular />
            <TopRated />
        </div>
    );
};
export default Home;
```

## searchResult.jsx

```
import React, { useState, useEffect } from "react";
import { useParams } from "react-router-dom";
import InfiniteScroll from "react-infinite-scroll-component";
import "./style.scss";
import { fetchDataFromApi } from "../../utils/api";
import ContentWrapper from "../../components/contentWrapper/ContentWrapper";
import MovieCard from "../../components/movieCard/MovieCard";
import Spinner from "../../components/spinner/Spinner";
import noResults from "../../assets/no-results.png";
const SearchResult = () => {
```

```
const [data, setData] = useState(null);
const [pageNum, setPageNum] = useState(1);
const [loading, setLoading] = useState(false);
const { query } = useParams();
const fetchInitialData = () => {
    setLoading(true);
    fetchDataFromApi(`/search/multi?query=${query}&page=${pageNum}`).then(
        (res) => {
            setData(res);
            setPageNum((prev) => prev + 1);
            setLoading(false);
        }
    );
};
const fetchNextPageData = () => {
    fetchDataFromApi(`/search/multi?query=${query}&page=${pageNum}`).then(
        (res) => {
            if (data?.results) {
                setData({
                    ...data,
                    results: [...data?.results, ...res.results],
                });
            } else {
                setData(res);
            }
            setPageNum((prev) => prev + 1);
        }
    );
};
useEffect(() => {
    setPageNum(1);
    fetchInitialData();
}, [query]);
return (
    <div className="searchResultsPage">
        {loading && <Spinner initial={true} />}
        {!loading && (
            <ContentWrapper>
                {data?.results?.length > 0 ? (
                    <>
                        <div className="pageTitle">
                            {`Search ${
                                data?.total_results > 1
                                    ? "results"
                                    : "result"
                            } of '${query}'`}
                        </div>
                        <InfiniteScroll
                            className="content"
                            dataLength={data?.results?.length || []}
                            next={fetchNextPageData}
                            hasMore={pageNum <= data?.total_pages}
                            loader={<Spinner />}
                        >
                            {data?.results.map((item, index) => {
                                if (item.media_type === "person") return;
                                return (
                                    <MovieCard
                                        key={index}
                                        data={item}
                                        fromSearch={true}
                                    />
                                );
                            })}
                        </InfiniteScroll>
                    </>
                ) : (
                    <span className="resultNotFound">
                        Sorry, Results not found!
                    </span>
                )}
            </ContentWrapper>
        )}
    </div>
  );
};
export default SearchResult;
```

- **store**
    - **homeSlice.jsx**
    - **store.jsx**

```
import { createSlice } from "@reduxjs/toolkit";
export const homeSlice = createSlice({
  name: "home",
  initialState: {
    url: {},
    genres: {},
  },
  reducers: {
    getApiConfiguration: (state, action) => {
            state.url = action.payload;
        },
        getGenres: (state, action) => {
            state.genres = action.payload;
        },
    },
});
// Action creators are generated for each case reducer function
export const { getApiConfiguration, getGenres } = homeSlice.actions;
export default homeSlice.reducer;
```

store

```
import { configureStore } from "@reduxjs/toolkit";
import homeSlice from "./homeSlice";
export const store = configureStore({
    reducer: {
        home: homeSlice,
    },
});
```

- **utils**
    - **api.jsx**

```
import axios from "axios";
const BASE_URL = "https://api.themoviedb.org/3";
const TMDB_TOKEN = import.meta.env.VITE_APP_TMDB_TOKEN;
const headers = {
    Authorization: "bearer " + TMDB_TOKEN,
};
export const fetchDataFromApi = async (url, params) => {
    try {
        const { data } = await axios.get(BASE_URL + url, {
            headers,
            params,
        });
        return data;
    } catch (err) {
        console.log(err);
        return err;
    }
};
```

## App.jsx

```
import { useState, useEffect } from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import { fetchDataFromApi } from "./utils/api";
import { useSelector, useDispatch } from "react-redux";
import { getApiConfiguration, getGenres } from "./store/homeSlice";
import Header from "./components/header/Header";
import Footer from "./components/footer/Footer";
import Home from "./pages/home/Home";
import Details from "./pages/details/Details";
import SearchResult from "./pages/searchResult/SearchResult";
import Explore from "./pages/explore/Explore";
import PageNotFound from "./pages/404/PageNotFound";
function App() {
    const dispatch = useDispatch();
    const { url } = useSelector((state) => state.home);
    console.log(url);
    useEffect(() => {
        fetchApiConfig();
        genresCall();
}, []);
const fetchApiConfig = () => {
  fetchDataFromApi("/configuration").then((res) => {
    console.log(res);

    const url = {
      backdrop: res.images.secure_base_url + "original",
      poster: res.images.secure_base_url + "original",
      profile: res.images.secure_base_url + "original",
    };
    dispatch(getApiConfiguration(url));
  });
};
const genresCall = async () => {
  let promises = [];
  let endPoints = ["tv", "movie"];
  let allGenres = {};

  endPoints.forEach((url) => {
    promises.push(fetchDataFromApi(`/genre/${url}/list`));
  });

  const data = await Promise.all(promises);
  console.log(data);
  data.map(({ genres }) => {
    return genres.map((item) => (allGenres[item.id] = item));
  });
  dispatch(getGenres(allGenres));
};
return (
        <BrowserRouter>
            <Header />
            <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/:mediaType/:id" element={<Details />} />
                <Route path="/search/:query" element={<SearchResult />} />
                <Route path="/explore/:mediaType" element={<Explore />} />
                <Route path="*" element={<PageNotFound />} />
            </Routes>
            <Footer />
        </BrowserRouter>
    );
}
export default App;
```

## main.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import "./index.scss";
import { store } from "./store/store";
import { Provider } from "react-redux";
ReactDOM.createRoot(document.getElementById("root")).render(
    <Provider store={store}>
        <App />
    </Provider>
);
```

## Index.html

```
<!DOCTYPE html>
```

```html
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/movix-logo.png" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Movix</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

## Package.json

```json
{
  "name": "movix",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "saurya": "vite",
    "build": "vite build",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-
warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
```

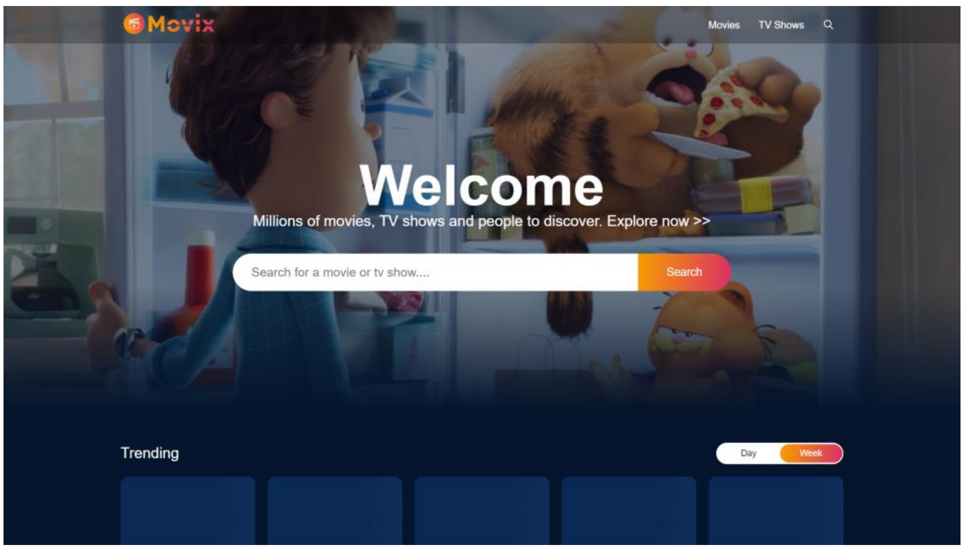
```
  "@reduxjs/toolkit": "^1.9.1",
  "axios": "^1.2.2",
  "dayjs": "^1.11.7",
  "react": "^18.2.0",
  "react-circular-progressbar": "^2.1.0",
  "react-dom": "^18.2.0",
  "react-icons": "^4.7.1",
  "react-infinite-scroll-component": "^6.1.0",
  "react-lazy-load-image-component": "^1.5.6",
  "react-player": "^2.11.0",
  "react-redux": "^8.0.5",
  "react-router-dom": "^6.6.2",
  "react-select": "^5.7.0",
  "sass": "^1.57.1"
},
"devDependencies": {
  "@types/react": "^18.2.66",
  "@types/react-dom": "^18.2.22",
  "@vitejs/plugin-react": "^4.2.1",
  "eslint": "^8.57.0",
  "eslint-plugin-react": "^7.34.1",
  "eslint-plugin-react-hooks": "^4.6.0",
  "eslint-plugin-react-refresh": "^0.4.6",
  "vite": "^5.2.0"
}
```

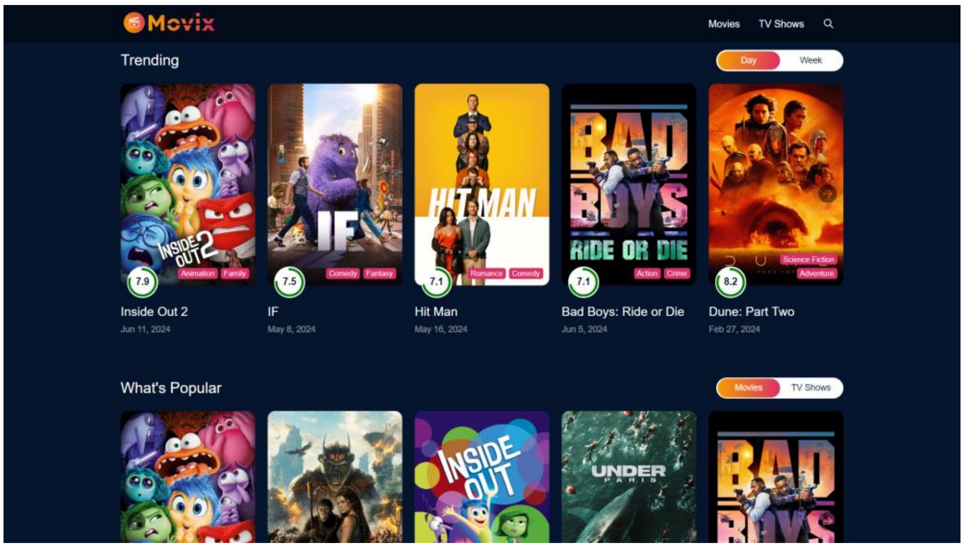
}

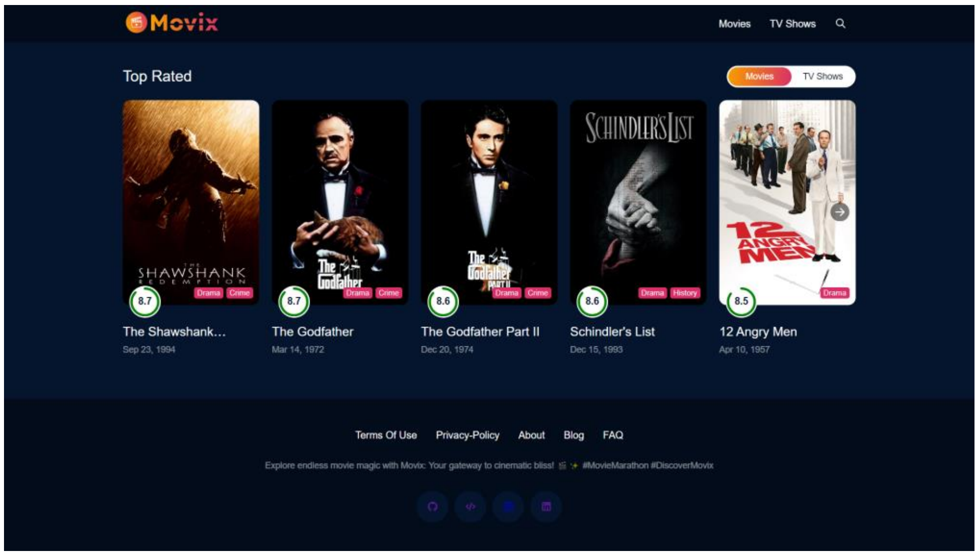
## Screen / Reports

- ### Input Output Screens
- **Home Page**
    - Landing page with popular movies/TV shows.
    - Header: Logo, navigation menu.
    - Main Section: Carousel, trending section.
    - Footer: Social media links, contact information.
- **Movie/TV Show Details Page**
    - Detailed view of a chosen movie/TV show.

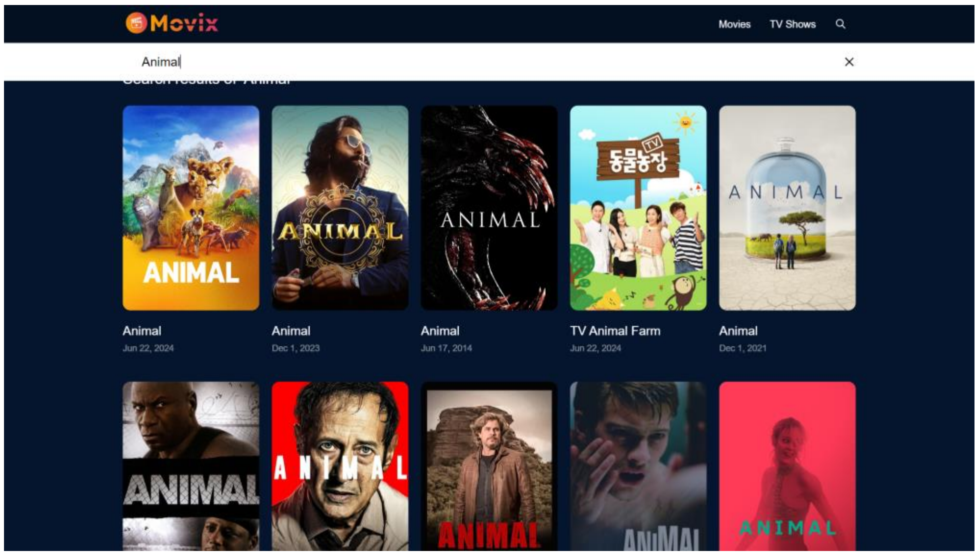
  - Banner: Background image, title, rating.
  - Synopsis: Description, genres, release date, runtime.
  - Cast: List of main cast members with characters.
  - Trailer: Button to play the trailer in a popup.
  - Recommendations: Similar movies/TV shows.
- **Explore Page**
  - Filter and discover movies/TV shows based on criteria.
  - Filters: Dropdowns for genres, sort options, ratings.
  - Movie/TV Show Grid: Results displayed based on filters.
  - Pagination: Infinite scroll to load more results.
- **Search Results Page**
  - Results of a search query.
  - Search Bar: Input field for search queries.
  - Results Grid: Search results displayed in a grid layout.
  - Pagination: Infinite scroll to load more search results.
- **Watch Trailer Popup**
  - Modal popup to play the trailer.
  - Video Player: Embedded YouTube video player.
  - Close Button: Button to close the popup.

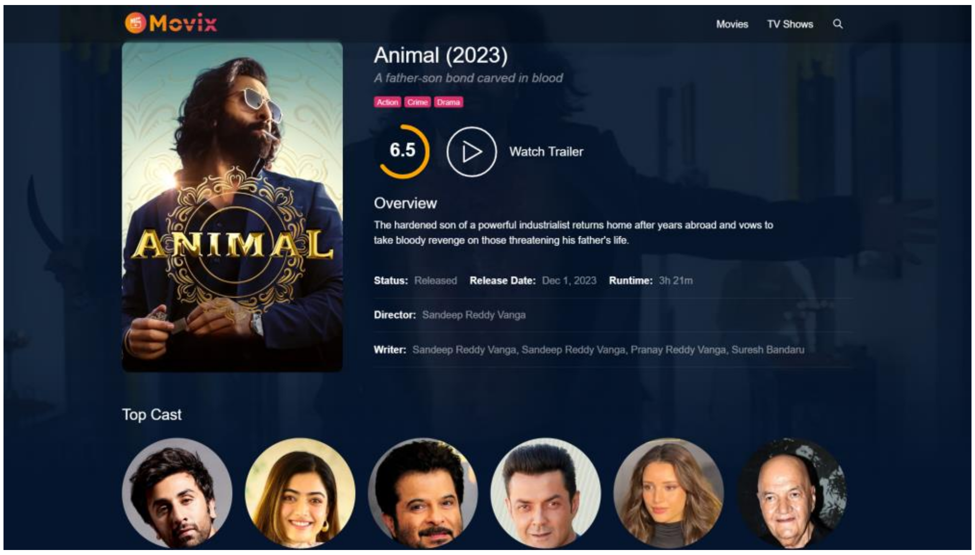

Movix
Movies
TV Shows
Welcome
Millions of movies, TV shows and people to discover. Explore now >>
Search for a movie or tv show....
Search
Trending
Day
Week





Explore endless movie magic with Movix: Your gateway to cinematic bliss! 🎬 ✨ #MovieMarathon #DiscoverMovix





Animal (2023)

*A father-son bond carved in blood*

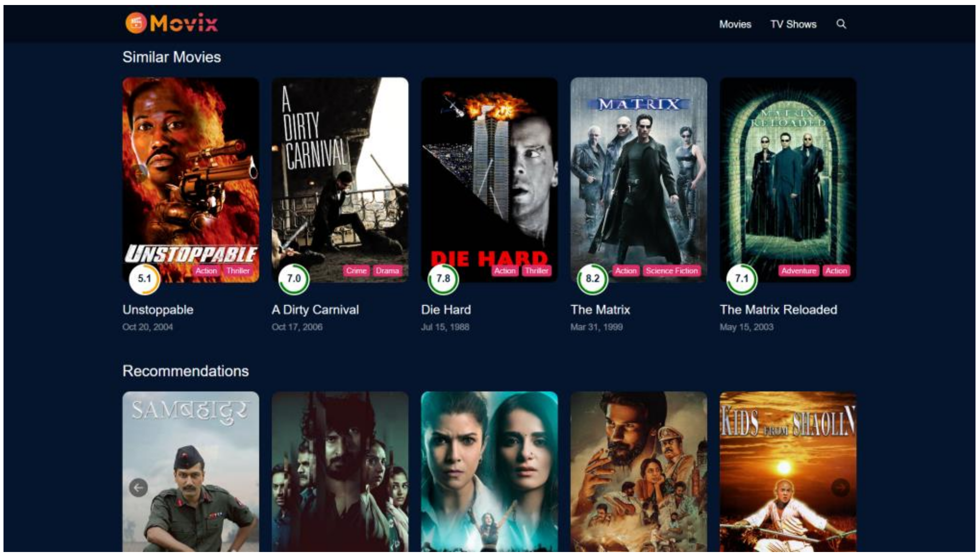
Overview

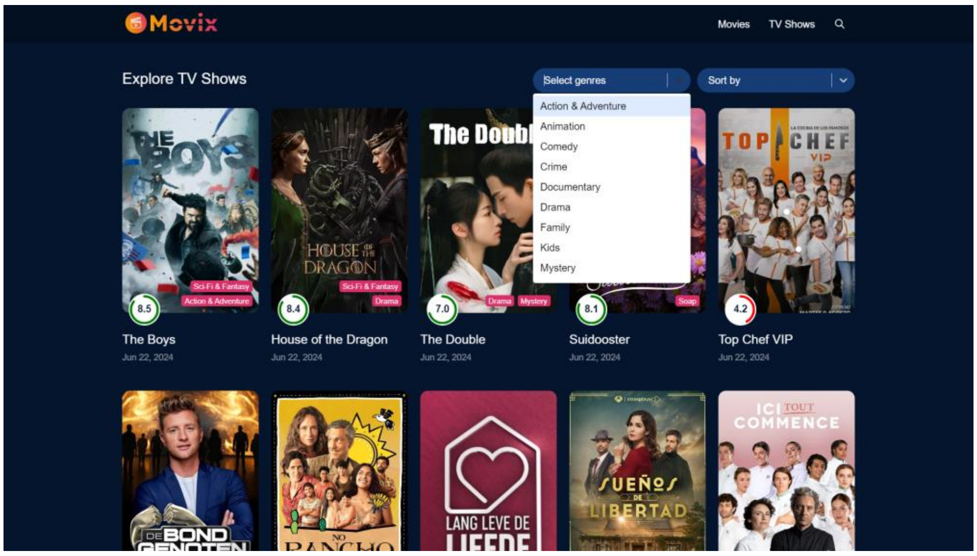
The hardened son of a powerful industrialist returns home after years abroad and vows to take bloody revenge on those threatening his father's life.

**Status:** Released **Release Date:** Dec 1, 2023 **Runtime:** 3h 21m

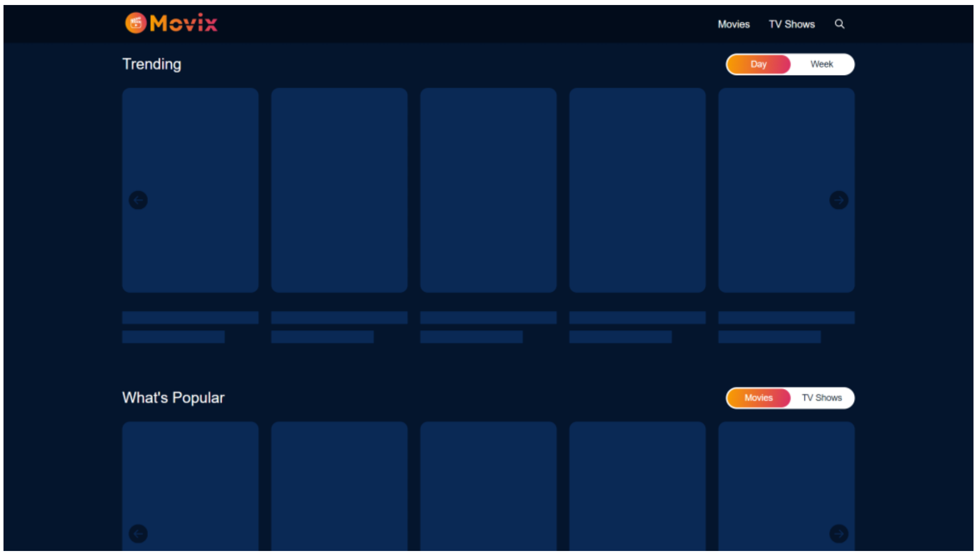
**Director:** Sandeep Reddy Vanga

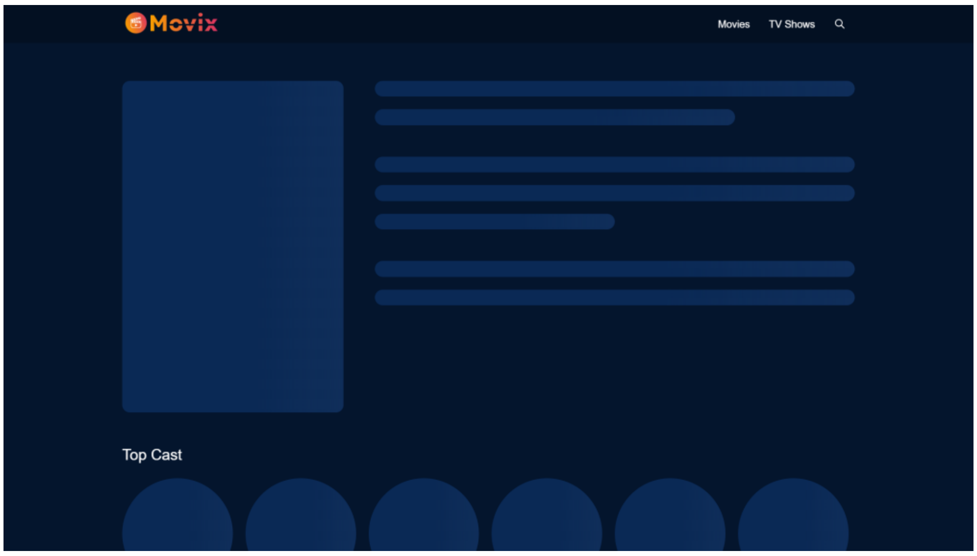
**Writer:** Sandeep Reddy Vanga, Sandeep Reddy Vanga, Pranay Reddy Vanga, Suresh Bandaru





- **Loading Screen Animation using JS Library**





## Sample Reports

- **Popular Movies Report**
  - Lists most popular movies in a timeframe.
  - Title: "Popular Movies Report"
  - Date Range: "Start Date - End Date"
  - Table: Rank, Movie Title, Release Date, Popularity Score.
  - Summary: Explanation of the report data.
- **User Engagement Report**
  - Shows user engagement metrics.
  - Title: "User Engagement Report"

  - Date Range: "Start Date - End Date"
  - Metrics: Total Users, Active Users, Average Session Duration, Most Viewed.
  - Graphs: Daily Active Users, Session Duration.
- **Genre Popularity Report**
  - Analyzes popularity of different genres.
  - Title: "Genre Popularity Report"
  - Date Range: "Start Date - End Date"
  - Table: Genre, Number of Views, Percentage.
  - Pie Chart: Genre viewership percentages.
  - Summary: Explanation of the report data.

# Testing

A robust testing strategy is essential to ensure the Movie/TV Show Explorer application functions flawlessly, delivers a reliable experience, and meets user expectations. Here's how we can approach testing:

### 1. Unit Testing:

- **Focus:** Isolate and test individual components, functions, and modules to verify their correctness.
- **Tools:** Utilize testing libraries like Jest and React Testing Library for React components.
- **Examples:**
  - Test hooks like useFetch to guarantee data fetching and state management work as intended.
  - Validate utility functions that manipulate data or handle API interactions.
  - Verify rendering and behavior of components like MovieCard, Spinner, and Select dropdowns.

### 2. Integration Testing:

- **Objective:** Ensure seamless interaction between different components or modules.
- **Approach:** Render larger sections of the application and simulate user interactions.
- **Examples:**
  - Test interactions between Explore page components (filters, pagination, and movie cards) to guarantee filtering and pagination work as expected.
  - Verify search functionality and pagination by testing SearchResult page components and the InfiniteScroll behavior.

### 3. End-to-End Testing:

- **Goal:** Test the entire application flow from start to finish, mimicking real user scenarios.
- **Tools:** Utilize tools like Cypress for end-to-end testing.

- **Examples:**
    - Simulate user actions such as selecting genres, applying sorting criteria, scrolling through results, and verify expected responses.
    - Ensure navigation works correctly between pages (Explore, SearchResult, Details).
    - Test edge cases (handling empty data, network errors, or unexpected API responses).

### 4. Performance Testing:

- **Objective:** Measure application performance under various conditions (data size, network latency).
- **Tools:** Utilize tools like Lighthouse for web performance audits, or create custom scripts to simulate load and stress testing.
- **Examples:**
    - Measure and optimize loading times for data-heavy components (e.g., Explore page with large results sets).
    - Ensure smooth scrolling and responsiveness during pagination in InfiniteScroll components.
    - Test API response times and optimize data fetching strategies for enhanced performance.

### 5. User Acceptance Testing (UAT):

- **Objective:** Validate the application against user expectations and requirements.
- **Approach:** Involve stakeholders or end-users to perform tests in a real-world environment.
- **Examples:**
    - Conduct usability tests to ensure intuitive navigation and user-friendliness when exploring movies and TV shows.
    - Gather feedback on the effectiveness of search features, filtering options, and overall user experience.
    - Validate that the application meets functional requirements specified in user stories or use cases.

## Conclusion of Project

The Movie/TV Show Explorer project has been successfully developed and implemented, offering users a comprehensive platform for exploring and discovering movies and TV shows. During development, several key components and features were built, contributing to a reliable and user-friendly application.

### Key Achievements

1. **User-Centric Design:**
    - **Responsive Design**: Ensures consistent user experience across various devices.
    - **Intuitive Navigation** (using React Router): Enables seamless navigation between sections (Explore, Search, Details).
2. **Rich Functionality:**
    - **Explore Section**: Users can browse movies/TV shows by genres and sort them using various criteria.

   - **Search Functionality**: A robust search feature allows users to find titles using keywords, with real-time results and pagination for large datasets.
   - **Detailed Information**: Provides comprehensive information for each movie/TV show.
   - **Interactive Features**: Integrates features like video trailers, loading indicators (Spinner), and infinite scrolling (InfiniteScroll) to enhance user engagement.
3. **Solid Technical Foundation:**
   - **API Integration**: Utilizes The Movie Database (TMDb) API for up-to-date movie and TV show data, genres, and search results.
   - **Efficient State Management**: Employs React hooks and custom hooks (like useFetch) to manage application state and handle asynchronous data fetching effectively.
   - **Reusable Component Architecture**: Develops reusable components (e.g., MovieCard, Spinner, Select) for code maintainability and reusability.
4. **Rigorous Testing:**
   - **Unit and Integration Testing**: Implemented using Jest and React Testing Library to verify component behavior, API interactions, and application flows.
   - **Performance Optimization**: Optimizes application performance with lazy loading, efficient data fetching, and minimized rendering bottlenecks.
5. **Deployment and Maintainability:**
   - **Deployment**: Deployed to a scalable hosting platform (e.g., Netlify, AWS) with continuous integration and deployment pipelines for streamlined updates and reliability.
   - **Comprehensive Documentation**: Provides thorough documentation including project structure, setup instructions, API usage, and component documentation to facilitate future maintenance and onboarding.

## Future Enhancements

Moving forward, several enhancements could further enrich the Movie/TV Show Explorer application:

- **Personalized Experience**: Implement user accounts and profiles to enable features like watchlists, recommendations, and user preferences.
- **Advanced Search and Filtering**: Expand search capabilities with advanced filters (e.g., year range, language) and predictive search suggestions.
- **Community Engagement**: Introduce user reviews, ratings, and community-driven features to enhance user interaction and engagement.
- **Accessibility Focus**: Prioritize accessibility improvements (e.g., screen reader support, keyboard navigation) for inclusivity.
- **Global Reach**: Implement Internationalization (i18n) to support multiple languages and cater to a broader audience.

### Overall Conclusion

The Movie/TV Show Explorer project has been a rewarding journey, combining technical expertise with a focus on user experience. By leveraging modern frontend technologies, API integrations, and best practices in software development, the application successfully fulfills its objective of providing a rich and intuitive platform for exploring the world of entertainment.

Ongoing maintenance and future enhancements will ensure the application continues to evolve and cater to the needs of its growing user base in the dynamic media landscape.

## Limitations

Despite the successful implementation of the Movie/TV Show Explorer project, there are several limitations and areas for improvement worth considering for future development:

1. **API Rate Limits and Data Management**:
    - The application relies on external APIs (e.g., TMDb) with rate limits. This could impact responsiveness during high traffic.
    - Data accuracy and timeliness depend on external sources. Implementing mechanisms to handle potential discrepancies and outdated content would enhance data reliability.
2. **Third-Party Service Dependencies**:
    - The application heavily relies on external services like TMDb and YouTube. Disruptions in these services could affect functionality.
    - Exploring alternative data sources or fallback mechanisms could improve resilience.
3. **Offline Functionality**:
    - The application currently requires an internet connection.
    - Implementing offline caching or fallback content would enhance user experience in low-connectivity scenarios (**High Impact**).
4. **Performance Optimization**:
    - While performance optimization efforts exist, bottlenecks might arise on low-end devices or slow connections.
    - Further optimizations in rendering, data caching, and network requests could be explored (**Medium Impact**).
5. **Security Considerations**:
    - Currently, user authentication and authorization are not implemented.
    - Adding authentication would be crucial for future personalized features and protecting user information (**Medium Impact**).
6. **Internationalization and Localization**:
    - The application is primarily designed in English without built-in support for multiple languages (i18n).
    - Implementing i18n features would broaden its appeal and accessibility to a global audience (**High Impact**).
7. **Browser Compatibility**:
    - Although efforts were made for compatibility across major browsers, differences might exist.
    - Regular testing across various browsers and versions is necessary to maintain a consistent user experience (**Medium Impact**).
8. **Scalability**:
    - The current infrastructure might require adjustments to accommodate a significantly larger user base or increased traffic.
    - Exploring load balancing, server-side optimizations, and database scaling would ensure smooth operation at scale (**High Impact for future growth**).
9. **Ongoing Maintenance and Documentation**:

- Comprehensive documentation exists, but ongoing maintenance and updates are crucial for future feature additions, bug fixes, and team collaboration.
- Maintaining and enhancing documentation will streamline development and ensure long-term project sustainability.

## References / Bibliography

1. **React Official Documentation**
   - Source: https://legacy.reactjs.org/docs/getting-started.html
   - Description: Comprehensive guide and documentation on React, a JavaScript library for building user interfaces.
2. **Redux Official Documentation**
   - Source: https://redux.js.org/introduction/getting-started
   - Description: Detailed information on Redux, a predictable state container for JavaScript apps.
3. **React Router Documentation**
   - Source: https://reactrouter.com/
   - Description: Official documentation for React Router, a collection of navigational components for React applications.
4. **dayjs Documentation**
   - Source: https://day.js.org/
   - Description: Documentation for dayjs, a minimalist JavaScript library for date manipulation.
5. **The Movie Database (TMDb) API Documentation**
   - Source: https://developer.themoviedb.org/docs/getting-started
   - Description: Official documentation for the TMDb API, which provides access to a large database of movies, TV shows, and actors.
6. **React Select Documentation**
   - Source: https://react-select.com/
   - Description: Documentation for React Select, a flexible and beautiful Select Input control for React.
7. **Infinite Scroll Documentation**
   - Source: https://www.npmjs.com/package/react-infinite-scroll-component

- Description: Guide and documentation for using the Infinite Scroll component in React applications.

8. **Sass Documentation**
    - Source: https://sass-lang.com/documentation/
    - Description: Official documentation for Sass, a powerful CSS extension language.
9. **Babel Documentation**
    - Source: https://babeljs.io/docs/usage
    - Description: Comprehensive guide on Babel, a JavaScript compiler.
10. **Webpack Documentation**
    - Source: https://webpack.js.org/concepts/
    - Description: Documentation for Webpack, a module bundler for JavaScript applications.
11. **Axios Documentation**
    - Source: https://axios-http.com/
    - Description: Official documentation for Axios, a promise-based HTTP client for the browser and Node.js.
12. **MDN Web Docs**
    - Source: https://developer.mozilla.org/en-US/
    - Description: Mozilla's official documentation and learning resource for web technologies, including HTML, CSS, and JavaScript.
13. **Visual Studio Code Documentation**
    - Source: https://code.visualstudio.com/docs
    - Description: Documentation for Visual Studio Code, a powerful and lightweight code editor.
14. **ESLint Documentation**
    - Source: https://eslint.org/docs/latest/use/getting-started
    - Description: Official documentation for ESLint, a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code.
15. **Jest Documentation**
    - Source: https://jestjs.io/
    - Description: Official documentation for Jest, a delightful JavaScript testing framework.

16. **Postman Documentation**

- Source: https://learning.postman.com/
- Description: Guide and documentation for using Postman, a collaboration platform for API development.