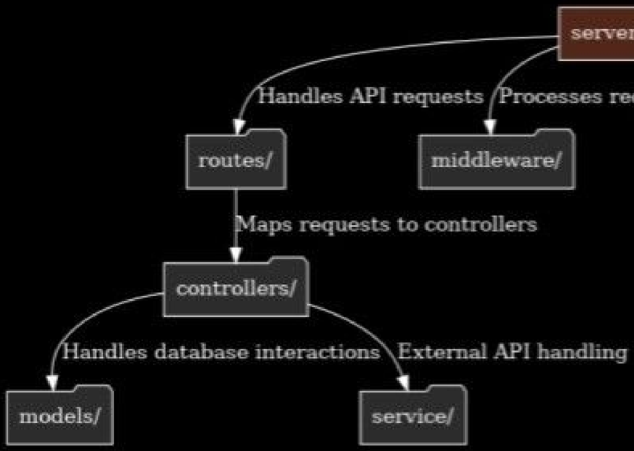
1. Overview

WATCHit TM is a full-stack video streaming application with a React.js frontend and a Node.js/Express backend. It interacts with a database to manage users and fetches movie/TV data from TMDb API.

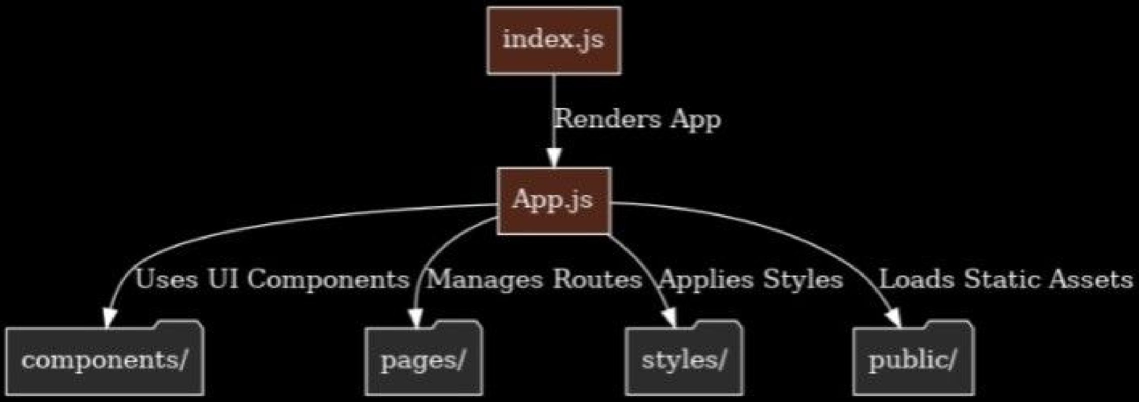
Backend Architecture (Node.js & Express)

The backend processes API requests, handles authentication, and communicates with the database.



Frontend Architecture (React.js)

The frontend is built with React.js, managing user interaction and communicating with the backend.



Installed Packages

Frontend-packages:- npm i

```
npm create vite@latest .

npm i axios lucide-react swiper
react-player react-hot-toast
react-router-dom zustand

npm run dev
```

backend packages:-

```
npm init -y

npm install express jsonwebtoken
moongose cookie-parser dotenv axios
bcryptjs
```

2. Backend Architecture (Node.js & Express)

The backend is responsible for handling requests from the frontend, processing data, and sending responses.

Backend Directory Structure (Key Files)

Copy code

```
├── config
│   ├── db.js          # Database connection setup
│   └── envVars.js      # Environment variables (API keys, DB
credentials)
├── controllers
│   ├── auth.controller.js  # Handles user login & signup
│   ├── movie.controller.js # Fetches movies from TMDb API
│   ├── search.controller.js # Handles search functionality
│   └── tv.controller.js    # Fetches TV shows from TMDb API
├── middleware
│   └── protectRoute.js # Middleware for authentication
(JWT-based)
├── models
│   └── user.model.js   # User schema (MongoDB or other DB)
├── routes
│   ├── auth.route.js   # User authentication routes (login,
signup)
│   ├── movie.route.js  # Movie-related API routes
│   ├── search.route.js # Search API routes
│   └── tv.route.js     # TV-related API routes
├── server.js          # Main entry point, initializes Express app
├── service
│   └── tmdb.server.js # Interacts with TMDb API to fetch
movie/TV data
└── utils
    └── generateToken.js # JWT token generation for
authentication
```

How Backend Works

1. **User Requests** → Frontend sends API requests to the backend.
2. **Routing & Middleware** → The request passes through Express routes and middleware.
3. **Controllers Handle Logic** → The controllers process the request, fetch data, or interact with the database.
4. **Database & External APIs** → If required, data is fetched from a database or the TMDb API.
5. **Response Sent** → Processed data is sent back to the frontend in JSON format.

Example API Flow (Fetching Movies)

1. Frontend sends a request to `/api/movies/popular`.
2. `movie.route.js` routes this request to `movie.controller.js`.
3. `movie.controller.js` calls `tmdb.server.js` to fetch movie data.
4. TMDb API returns the data.
5. Backend sends this data as JSON to the frontend.

Frontend Directory Structure (Key Files)

Copy code

```
├── src
│   ├── components    # Reusable UI components (Navbar, Buttons, etc.)
│   ├── pages         # Pages (Home, Login, Movie Details, etc.)
│   ├── styles        # CSS or styled-components for styling
│   ├── App.js        # Main component, handles routing
│   ├── index.js      # Entry point, renders App.js
├── public
│   ├── favicon.ico   # App icon
│   ├── index.html    # HTML template where React renders
├── package.json      # Lists dependencies (React, Axios, etc.)
```

How Frontend Works

1. **User Navigates to the Website** → React loads the application.
2. **React Router Loads Pages** → App.js handles routing (/, /login, /movies/:id).
3. **API Calls to Backend** → React fetches data using **Axios** or **Fetch API**.
4. **State Management** → Data is stored in React state (useState or Redux if used).
5. **Components Render Data** → Components update dynamically based on API responses.

Example Flow (User Searching for a Movie)

1. User types a movie name in the search bar.
2. SearchComponent.js calls fetchMovies(query).
3. fetchMovies() sends a request to /api/search?query=<movie_name>.
4. Backend (search.controller.js) calls TMDb API and returns results.
5. Frontend updates state and renders movie search results.

4. Connecting Frontend & Backend

The frontend and backend communicate via HTTP requests using the Fetch API or Axios.

How Data Flows Between Them

1. **User interacts with frontend** (e.g., clicks "Get Popular Movies").
2. **Frontend makes an API request** → `GET /api/movies/popular`.
3. **Backend processes the request** → Calls TMDb API and returns results.
4. **Frontend receives JSON response** → Updates state and displays movies.

Example API Call from Frontend (Using Axios)

Javascript Copy code

```javascript
import axios from "axios";

const fetchMovies = async () => {
 const response = await
axios.get("http://localhost:5000/api/movies/popular");
 console.log(response.data); // Logs movie data
};
```

5. Authentication System (JWT-Based Login)

- **User enters email & password** on the login page.
- **Frontend sends credentials** to `/api/auth/login`.
- **Backend verifies credentials** with `auth.controller.js`.
- **If correct, backend generates a JWT token** (`generateToken.js`).
- **Frontend stores JWT in localStorage** and includes it in future API requests.
- **Backend middleware (`protectRoute.js`) checks JWT** before granting access.

6. External API Integration (TMDb)

The app fetches movie & TV show data from **TMDb API** using tmdb.server.js.

- Requests are made with an **API key** (config/envVars.js).
- Example **TMDb API request**:

Javascript ⎘ Copy code

```javascript
const response = await axios.get(

`https://api.themoviedb.org/3/movie/popular?api_key=YOUR_API_KEY`
);
```

- The backend **proxies these requests** to keep the API key secure.

7. Database Management (User Data)

- Likely **MongoDB or SQL database** (db.js file found in config/).
- user.model.js defines user schema (ID, email, password, etc.).
- **Example MongoDB schema (if used)**:

Javascript　　Copy code

```javascript
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
  email: String,
  password: String
});

module.exports = mongoose.model("User", UserSchema);
```

- **Authentication uses hashed passwords** (bcrypt) for security.

8. Summary of Technologies Used

Category	Technology
Frontend	React.js, Axios
Backend	Node.js, Express.js
Database	Likely MongoDB or MySQL
Authentication	JWT (JSON Web Token)
External API	TMDb (The Movie Database API)

9. Final Explanation

- The **frontend** (React.js) sends API requests to the **backend** (Node.js/Express).
- The **backend processes requests** using Express routes & controllers.
- **User data is stored in a database** (MongoDB/ MySQL).
- **Movie/TV data is fetched** from TMDb API via the backend.
- The **frontend updates dynamically**, providing an interactive streaming experience.