Developer Documentation
CS-F372 Final Project

Spencer Baysinger and Ivy Swenson

April 29, 2025

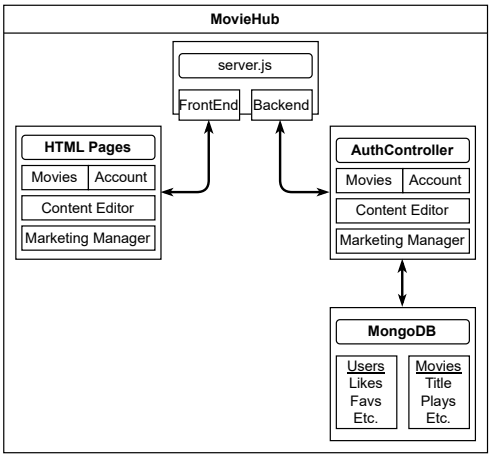
Contents

1 Introduction to Product Source Code

1.1 Overall System Architecture

The system is a Node.js and Express-based server that interacts with a MongoDB database and serves HTML/CSS/JS static files to the client. It supports user login, registration, movie management, and favorite/like functionality.

1.2 System Diagram



Description:

- Client Side: HTML, CSS, JavaScript (Frontend)
- Server Side: Express.js backend (Node.js)
- Database: MongoDB to store user accounts, movie information, user preferences.

1.3 Source Code Mapping

Module	**Source Code File**
All API GET/POST Requests	Javascript/authController.js
MongoDB Connection Setup	Javascript/db.js
Viewer User Scripts	Javascript/script_Account.js, Javascript/script_Login.js, Javascript/script_Home.js
Content Editor Scripts	Javascript/script_ContentEditor.js
Marketing Manager Scripts	Javascript/script_MarketingManager.js
Movie Importing	Javascript/script_ImportMovies.js
Video Player Functionality	Javascript/script_VideoPlayer.js
General Functions	Javascript/script_General.js
Webpage Files	html/index_*.html
Styling (CSS)	Styling/*.css
Server File (Express setup)	server.js

2 Major API Function Description

The following is a detailed list of the major functions and API's used in the codebase. Specifically in authController.js

2.1 API Registration: Session Authentication Functions

- **Function Name**: login
  POST: '/api/account/login', auth.login
  Parameters: req, res
  Result: Handles user login and returns success or failure message.
  Input: email, password
  Success Output: success: true, message: 'Login successful', redirect: redirectPage
  Fail Output: success: false, message: 'Wrong password'

- **Function Name**: register
  POST: '/api/account/register', auth.register
  Parameters: req, res
  Result: Registers a new user account.
  Input: username, email, password, roles = []
  Success Output: success: true, message: "Registration successful", redirect: redirectPage
  Fail Output: success: true, message: "Registration unsuccessful"

- **Function Name**: getSession
  GET: '/api/account/session', auth.getSession
  Parameters: req, res
  Result: Validates and retrieves active session data.
  Input: req.session and req.session.user
  Success Output [1]: success: true, email: req.session.user.email,
  Success Output [2]: username: req.session.user.username, roles: req.session.user.roles
  Fail Output: success: false, message: "No active session"

2.2 API Registration: User Account Management Functions

- **Function Name**: addFavorites
  POST: '/api/account/favorite/add', auth.addFavorites
  Parameters: req, res
  Result: Adds a movie to user's favorites list.
  Input: userId, movieTitle
  Success Output: success: true, message: "Favorite added"
  Fail Output: success: false, message: "Failed to add favorite"

- **Function Name**: removeFavorites
  POST: '/api/account/favorite/remove', auth.removeFavorites
  Parameters: req, res
  Result: Removes a movie from user's favorites list.
  Input: userId, movieTitle
  Success Output: success: true, message: "Favorite removed"
  Fail Output: success: false, message: "Failed to remove favorite"

- **Function Name**: getFavorites
  POST: '/api/account/favorite/get', auth.getFavorites
  Parameters: req, res
  Result: Retrieves user's list of favorite movies.
  Input: userId
  Success Output: success: true, favorites: favoritesList
  Fail Output: success: false, message: "Failed to retrieve favorites"

- **Function Name**: likeDislikeMovie
  POST: '/api/account/like-dislike', auth.likeDislikeMovie
  Parameters: req, res
  Result: Handles liking or disliking a movie.
  Input: userId, movieTitle, reaction
  Success Output: success: true, message: "Reaction updated"
  Fail Output: success: false, message: "Failed to update reaction"

- **Function Name**: getLikedMovies
  POST: '/api/account/like/get', auth.getLikedMovies
  Parameters: req, res
  Result: Retrieves movies the user has liked.
  Input: userId
  Success Output: success: true, likedMovies: likedMoviesList
  Fail Output: success: false, message: "Failed to retrieve liked movies"

- **Function Name**: getDislikedMovies
  POST: '/api/account/dislike/get', auth.getDislikedMovies
  Parameters: req, res
  Result: Retrieves movies the user has disliked.
  Input: userId
  Success Output: success: true, dislikedMovies: dislikedMoviesList
  Fail Output: success: false, message: "Failed to retrieve disliked movies"

- **Function Name**: searchMovies
  POST: '/api/movies/search', auth.searchMovies
  Parameters: req, res
  Result: Searches movies by title or genre.
  Input: searchQuery
  Success Output: success: true, results: movieResults
  Fail Output: success: false, message: "Movie search failed"

- **Function Name**: logout
  POST: '/api/account/logout', auth.logout
  Parameters: req, res
  Result: Logs user out and destroys session.
  Input: session
  Success Output: success: true, message: "Logged out successfully"
  Fail Output: success: false, message: "Failed to logout"

- **Function Name**: updateWatchHistory
  POST: '/api/account/watchHistory', auth.updateWatchHistory
  Parameters: req, res
  Result: Updates user's watch history.
  Input: userId, movieTitle
  Success Output: success: true, message: "Watch history updated"
  Fail Output: success: false, message: "Failed to update watch history"

- **Function Name**: getWatchHistory
  POST: '/api/account/watchHistory/get', auth.getWatchHistory
  Parameters: req, res
  Result: Retrieves user's watch history.
  Input: userId
  Success Output: success: true, history: watchHistoryList
  Fail Output: success: false, message: "Failed to retrieve watch history"

- **Function Name**: getUserReactions
  POST: '/api/account/reactions', auth.getUserReactions
  Parameters: req, res
  Result: Retrieves all like/dislike reactions for the user.
  Input: userId
  Success Output: success: true, reactions: userReactions
  Fail Output: success: false, message: "Failed to retrieve reactions"

2.3 API Registration: Content Editor Functions

- **Function Name**: addMovie
  POST: '/api/editor/add-movie', auth.addMovie
  Parameters: req, res
  Result: Adds a new movie to the database.
  Input: movieData
  Success Output: success: true, message: "Movie added successfully"
  Fail Output: success: false, message: "Failed to add movie"

- **Function Name**: updateMovie
  POST: '/api/editor/update-movie', auth.updateMovie
  Parameters: req, res
  Result: Updates an existing movie's information.
  Input: movieId, updatedFields
  Success Output: success: true, message: "Movie updated successfully"
  Fail Output: success: false, message: "Failed to update movie"

- **Function Name**: deleteMovie
  POST: '/api/editor/delete-movie', auth.deleteMovie
  Parameters: req, res
  Result: Deletes a movie from the database.
  Input: movieId
  Success Output: success: true, message: "Movie deleted successfully"
  Fail Output: success: false, message: "Failed to delete movie"

- **Function Name**: importMovies
  POST: '/api/movies/import', auth.importMovies
  Parameters: req, res
  Result: Imports movie data in bulk from a JSON file.
  Input: moviesJSON
  Success Output: success: true, message: "Movies imported successfully"
  Fail Output: success: false, message: "Failed to import movies"

2.4 API Registration: Marketing Manager Functions

- **Function Name**: updatePlayCount
  POST: '/api/marketing/play-count', auth.updatePlayCount
  Parameters: req, res
  Result: Updates the number of times a movie has been played.
  Input: movieId, incrementBy
  Success Output: success: true, message: "Play count updated"
  Fail Output: success: false, message: "Failed to update play count"

- **Function Name**: addFeedback
  POST: '/api/movies/feedback/add', auth.addFeedback
  Parameters: req, res
  Result: Adds feedback for a movie.
  Input: movieId, feedbackText
  Success Output: success: true, message: "Feedback added"
  Fail Output: success: false, message: "Failed to add feedback"

- **Function Name**: getAllMovies
  GET: '/api/movies/getAllMovies', auth.getAllMovies
  Parameters: req, res
  Result: Retrieves all movies stored in the database.
  Input:
  Success Output: success: true, movies: allMovies
  Fail Output: success: false, message: "Failed to retrieve movies"

- **Function Name**: getAllUsers
  GET: '/api/users/getAllUsers', auth.getAllUsers
  Parameters: req, res
  Result: Retrieves all users registered in the system.
  Input:
  Success Output: success: true, users: allUsers
  Fail Output: success: false, message: "Failed to retrieve users"

- **Function Name**: updateMovieNote
  POST: '/api/movies/update-note', auth.updateMovieNote
  Parameters: req, res
  Result: Updates notes or remarks attached to a movie.
  Input: movieId, notes
  Success Output: success: true, message: "Note updated"
  Fail Output: success: false, message: "Failed to update note"

3 Notes for Future Developers

3.1 Operational Notes

- Always run `npm install` after pulling the project to ensure all dependencies are present.
- MongoDB must be running locally on port `27017`, or the `MONGO_URI` environment variable must be properly set to point to a remote instance.
- This project uses Express.js as a backend server and relies on the following main API route groups:
  - `/api/account/*` – for user authentication, session management, and account-level operations.
  - `/api/movies/*` – for movie data, search, and analytics operations.
  - `/api/editor/*` – for content editors to add, update, or delete movies.
  - `/api/marketing/*` – for marketing team analytics like play count and feedback.
- Ensure that static files (HTML/CSS/JS) are served correctly from the root directory. Place all assets and client-facing resources within their appropriate subdirectories like `/Viewer`, `/ContentEditor`, etc.
- The server entry point is `server.js`. All routing begins from this file and delegates functionality to modular controllers.

3.2 Licensing and Terms

- This software is released under the GNU General Public License v3.0. A full copy of the license is included in the root directory of the repository as `LICENSE`.
- All users must agree to the Terms and Conditions, which are available at `/Assets/terms.html` and linked from the website's landing page.
- Contributions should follow the repository's license terms, including requirements for open-source redistribution and modification disclosures.

3.3 Credits

- This software and document were developed and written by Spencer Baysinger and Ivy Swenson.
- This project was developed as a project for UAF CS-F372 Software Construction course.