

# **PinkBox**

Group 01

**Group Name:** PinkBox

**Group Members:**

Aarithi Rajendren

Lara Afont

Erriana Thomas

Isabelle Carminati

**Github URL:** <https://github.com/laraafont/Group-1-SWE-Project>

# Table of Contents

<b>Section 1: Project Description</b>	<b>2</b>
PinkBox: Revolutionizing the Movie Marketplace:	2
Value Proposition:	3
How PinkBox Addresses the Challenge of Fragmented Movie Access	3
Features and Functionalities of PinkBox	4
Architectural Pattern of PinkBox	5
System Context Model	6
Use Case Model	7
<b>Section 2: Code Management, Project Management, and Test Plan</b>	<b>8</b>
Code Management:	8
Project Management:	9
Test Plan:	9
<b>Section 3: Technical Details, Installation Instructions, Login and Access Credentials and API Keys</b>	<b>11</b>
Frontend Architecture	11
Backend Architecture	12
Database Design	12
Installation Instructions	12
Session Authentication	13
API Keys	13
Access Credentials	13
<b>Section 4: Risk Management Plan, Software Quality Attributes and explanations</b>	<b>14</b>
<b>References</b>	<b>15</b>

## Section 1: Project Description

---

### **PinkBox: Revolutionizing the Movie Marketplace:**

PinkBox is an innovative centralized movie marketplace designed to tackle common challenges faced by movie enthusiasts, such as fragmented access to films, scattered subscriptions, and the absence of a unified platform for managing physical or digital movie collections. By incorporating an array of user-focused features, a robust technical foundation, and an intuitive user experience, PinkBox provides a scalable and modern solution tailored to both casual viewers and dedicated collectors.

Built on the MERN stack seen in Figure 1, PinkBox leverages cutting-edge technologies, including React.js for the front end and Express.js with Node.js for the backend, to ensure a seamless, user-friendly, and responsive application. The platform is thoughtfully engineered to be modular and scalable, accommodating evolving user needs and technological advancements.

PinkBox redefines how movie enthusiasts buy, sell, and curate their favorite films. As a comprehensive one-stop shop, it allows users to access, purchase, and manage movies without the constraints of multiple streaming subscriptions. By connecting users directly to production companies, the platform facilitates straightforward transactions and ensures authenticated access to digital content. This innovative initiative is driven by a collaborative team of specialists in front-end and back-end development, united by a shared vision of delivering a world-class application.

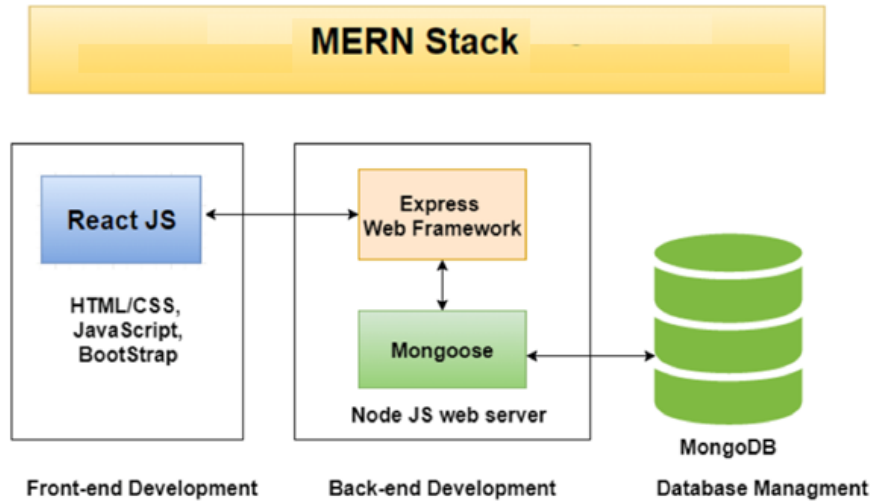


Figure 1: MERN Stack

### Value Proposition:

The strategic vision of PinkBox goes beyond simplifying the process of discovering and purchasing movies. It empowers users through advanced filtering options, personalized wishlists, and customized experiences. Whether users are planning their next movie night or expanding a carefully curated collection, PinkBox is designed to cater to diverse preferences while enhancing both discovery and management.

### How PinkBox Addresses the Challenge of Fragmented Movie Access

PinkBox effectively resolves the issue of scattered movie access by offering a single, reliable marketplace. This eliminates the need for juggling multiple streaming subscriptions, enabling users to purchase movies directly and own them digitally or physically. A robust backend powered by Express.js plays a pivotal role in managing user authentication, processing transactions, and providing APIs for accessing detailed movie information.

1. **User Authentication:** Secure user sessions are facilitated using JWT (JSON Web Tokens), which verify users with every request. The fetchuser middleware further enhances security by validating the auth-token sent in the HTTP header, ensuring access to specific features remains exclusive to authenticated users.
2. **Movie Management:** Dedicated API endpoints, such as /addmovie and /removemovie, empower administrators to manage the movie catalog effectively. The system prevents duplicate entries by enforcing unique movie titles, safeguarding database integrity.
3. **Shopping Cart:** The shopping cart functionality is supported by API endpoints like /addtocart and /removefromcart. These endpoints update cart data stored in the MongoDB database, ensuring changes are both persistent and accurate.

By centralizing movie access and leveraging robust backend functionalities, PinkBox simplifies the movie-buying process while enhancing user experience and addressing the challenges associated with fragmented access.

## Features and Functionalities of PinkBox

PinkBox provides a comprehensive suite of features designed to streamline and enhance the movie-buying experience for its users:

### 1. User Authentication

- Secure signup and login processes are implemented using JWT (JSON Web Tokens).
- The signup endpoint facilitates the creation of new user accounts, while the login endpoint verifies credentials and generates a JWT for authenticated sessions, ensuring secure user interactions.

### 2. Movie Listing and Management

- **Add Movie:** Administrators can add new movies using the /addmovie endpoint. Image uploads are handled by the multer middleware, which stores images in the /upload/images directory.
- **Remove Movie:** Administrators can remove movies via the /removemovie endpoint to ensure the database remains current and accurate.
- **List Movies:** Users can browse the entire movie catalog using the /allmovies endpoint, which retrieves comprehensive movie data from the MongoDB database.
- **Searching and Filtering:**
  - Users can search for movies by title through the /moviebytitle endpoint.
  - Alternatively, users can filter movies by genre using the /moviesbygenre endpoint, enabling efficient and personalized content discovery.

### 3. Shopping Cart

- **Add to Cart:** Users can add movies to their shopping cart via the /addtocart endpoint, which updates their cartData in the database.
- **Remove from Cart:** Movies can be removed from the cart through the /removefromcart endpoint, ensuring the cart reflects the user's current preferences.
- **View Cart:** The /getcart endpoint allows users to view their cart's contents, retrieving cartData from the database and displaying it seamlessly on the frontend.

### 4. Purchase Confirmation

- Upon completing a purchase, the /sendEmail endpoint sends an email confirmation containing order details and authenticated streaming URLs.
- The system utilizes Nodemailer to ensure prompt and accurate email delivery, enhancing user trust and providing a smooth post-purchase experience.

By combining these advanced features with a robust backend and an intuitive frontend, PinkBox delivers a seamless, efficient, and satisfying movie-buying journey for its users.

## Architectural Pattern of PinkBox

PinkBox is built on a robust and modern architectural foundation seen in Figure 2, emphasizing scalability, maintainability, and efficient data handling. The system integrates multiple architectural patterns to ensure seamless functionality and optimal performance:

1. **Microservices Architecture:** PinkBox adopts a microservices model to divide the system into independent and scalable modules. Key functionalities, such as user authentication, movie management, and order processing, are isolated into distinct services. This modular approach facilitates scalability, maintainability, and efficient troubleshooting.
2. **Client-Server Pattern:** A clear separation of concerns is achieved through the client-server model. The frontend, developed with React.js, interacts seamlessly with backend services powered by Node.js and Express. This architecture ensures secure and efficient handling of user requests, such as retrieving movie information or performing shopping cart actions.
3. **Layered Pattern:** The architecture employs a layered approach, categorizing the system into distinct layers:
  - **User Interface Layer:** Focused on delivering a smooth and responsive user experience.
  - **Domain Logic Layer:** Processes business logic, such as user actions and movie transactions.
  - **Database Access Layer:** Facilitates secure and efficient interactions with the MongoDB Atlas database, hosted on Google Cloud for enhanced scalability and reliability.

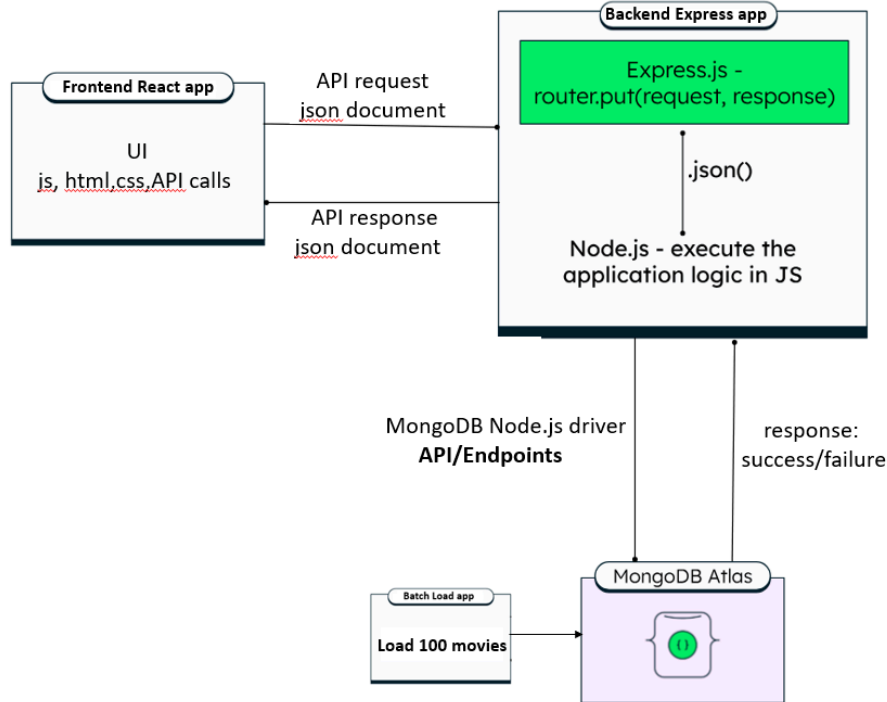


Figure 2: System Architecture

This strategic combination of architectural patterns ensures that PinkBox delivers a cohesive, secure, and user-centric platform while maintaining adaptability to future technological advancements.

## System Context Model

The context model seen in Figure 3 illustrates the interactions between users, the PinkBox application, production companies, and payment processing services. Each component's boundaries and responsibilities are clearly defined, allowing transparent data flows and smooth integration with external entities.

Frontend/Clients interact directly with the application through a modern web interface.

Backend Services process requests, authenticate users, manage the shopping cart, and communicate with third-party movie APIs and payment gateways.

Database securely stores user data, movie catalogs, orders, and transaction histories.

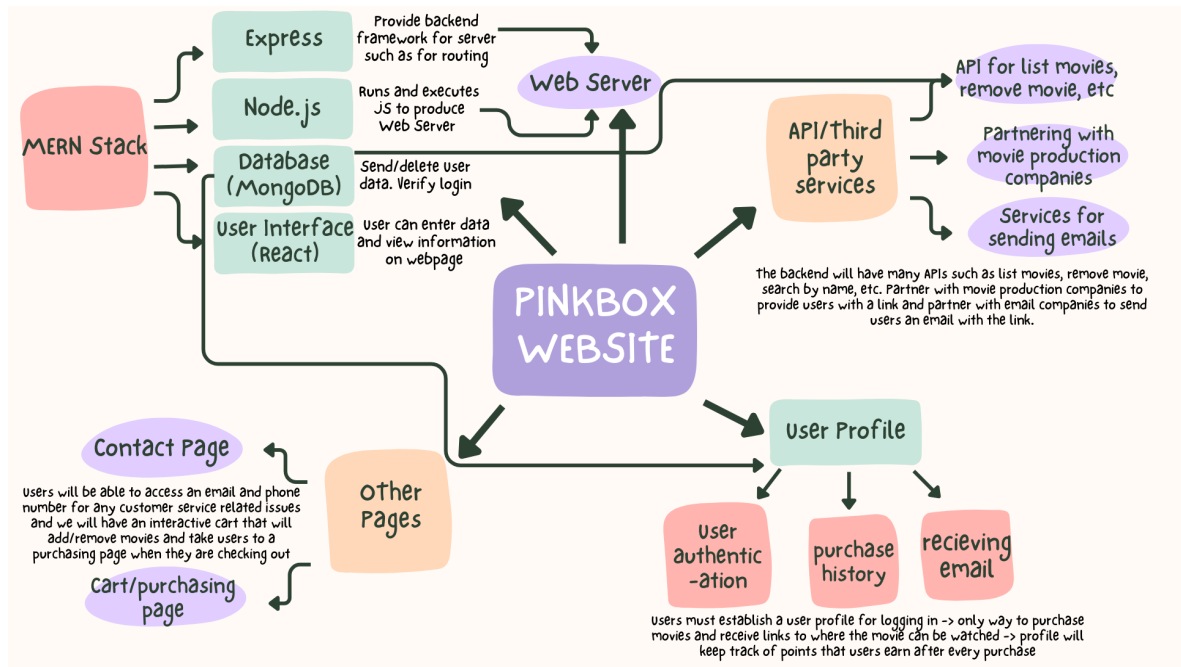


Figure 3: System Context Model

## Use Case Model

The PinkBox platform is designed with a range of core user scenarios, supported by robust system actions to ensure a seamless and secure experience:

### 1. User Authentication

- Facilitates secure sign-up and login processes through the use of JWT (JSON Web Tokens) seen in Figure 4.

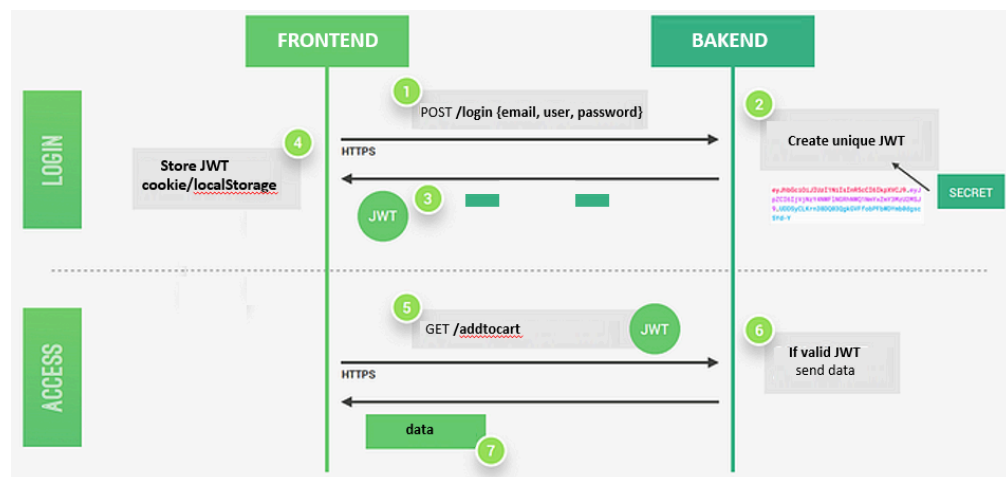




Figure 4: Json Web Token Handling

2. **Browsing and Searching**
  - Allows users to filter movies by platform, category, or genre for efficient discovery of desired content.
3. **Cart Management**
  - Enables users to add, remove, and update items in their shopping cart with precision and ease.
4. **Checkout and Purchase**
  - Supports smooth transaction completion, including email confirmations and the delivery of purchased movies or streaming links.
5. **Wishlist Management**
  - Provides users with the ability to save movies for future interest and revisit them at their convenience.
6. **Compliance and Security**
  - Enforces copyright terms, addresses movie format issues, and communicates potential consequences of violations, such as possible account bans.
  -

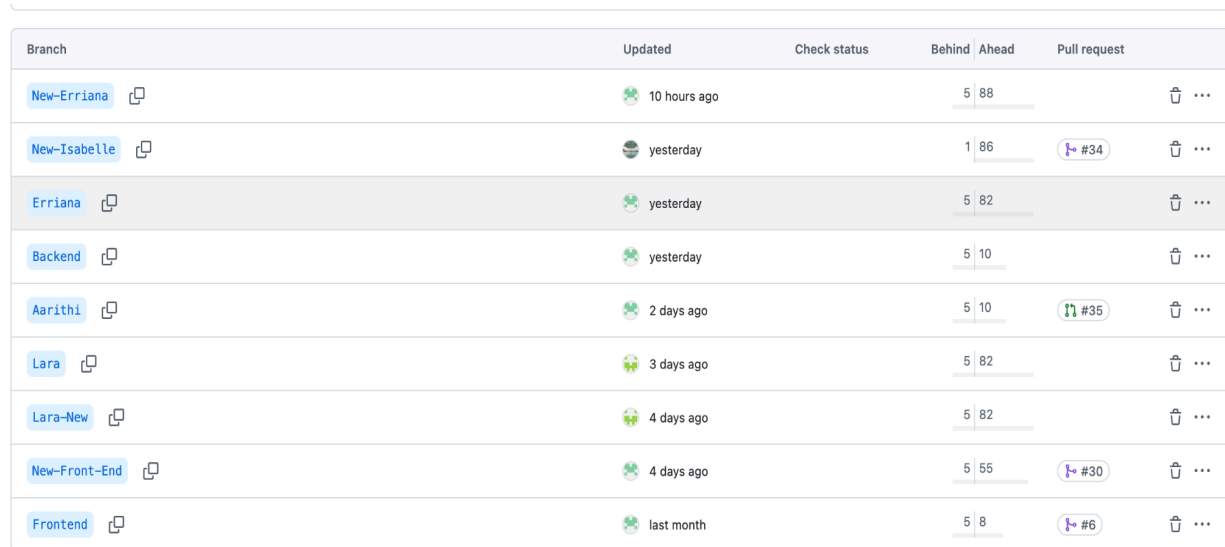
These use case models collectively uphold PinkBox's commitment to reliable business logic, efficient feature delivery, and comprehensive safeguards, ensuring a user-centric and secure ecosystem.

## Section 2: Code Management, Project Management, and Test Plan

---

### Code Management:

Working with several files involving complex code calls for effective management. This is especially since we're a team of four, with two frontend and two backend developers. To achieve this, we utilized github and the branches.



Branch	Updated	Check status	Behind   Ahead	Pull request
New-Erriana	10 hours ago		5   88	...
New-Isabelle	yesterday		1   86	#34  ...
Erriana	yesterday		5   82	...
Backend	yesterday		5   10	...
Aarithi	2 days ago		5   10	#35  ...
Lara	3 days ago		5   82	...
Lara-New	4 days ago		5   82	...
New-Front-End	4 days ago		5   55	#30  ...
Frontend	last month		5   8	#6  ...

Here we can see all the branches we used to manage the different programmers and implementations. We strategize by creating our respective branches and then also one for the font and back end. We plan to then merge the front-end branch, which includes the backend code, into main. Before then we've made several pull requests as seen in the image above. This was also used as a preventative measure for confusion or code conflicts. That way we have an easier time distinguishing the issue, where it's from, and who it's from.

### Project Management:

To manage the overall project, we relied on team meetings and discord. We'd meet once a week to complete assignments by splitting the work up evenly. For the codebase, we each set our goals depending on our current sprint. We'd work with our partner (other front/backend developer) to strategize ways to work together and complete a task. Consistent collaborations between two developers always occurred to which then as a whole team we share our progress in our weekly meetings. We'd also reach out to one another for any help and would take over if ever needed. We also used Trello for Sprint 0 and Sprint 1.

## Test Plan:

We've run a series of tests that would check for any edge cases or bugs within our system. We also utilized one of our assignments, Software Testing, to ensure that we can test our code in a more standardized way.

PROJECT NAME		PinkBox							PinkBox		
PREPARED BY		Isabelle Carminati, Aarithi Rajendren, Lara Afont, Erriana Thomas									
DATE OF CREATION		3/7/2025									
DATE OF REVIEW											
ITEM NUMBER	TEST CASE ID	TEST CASE DESCRIPTION	PRECONDITION	TEST STEPS	TEST DATA	EXPECTED RESULT	POSTCONDITION	ACTUAL RESULT	STATUS	COMMENTS	
1	111	Shopping Cart Modification	User needs to have	1. Users need to	Username:	Successful	Users cannot modify	User cannot click on	Fail	We need to	
2	112	First Time User	If a user is a first	1. User tries to login	Username:	Unsuccessful login	User cannot access	User is prompted	Fail	We need to	
3	113	Receiving Email Link	Users need to have a	1. Select movies to	A full shopping cart	Email with link to	Successful checkout	No checkout	Fail	We need to	
4	114	Product Filtering	The homepage	1. Log in to the	Searching up	The movies	"Drama" movies are	No homepage just	Fail	Work on	
5	115	Updating Pages	Different pages need	1) Have at least 2	Ways to navigate	The page the user	The appropriate page	A way to navigate	Fail	Test case	
6	116	Display of Product Information	Movie data and	1) Have a movie's	Movie image and a	The movie	A visual with the	A way to view a	Fail	Test case	
7	117	Invalid login attempt	Have a user	1) Have an account	A valid account in	An error message will	User won't be logged	User can't log in and	Fail	Test case	
8	118	Duplicate Username	Have a user create	1) Have an existing	A valid account in	An error will appear	The user won't be	The user will be	Fail	Test case	
9	119	Search Movies	Allow "search" to narrow down movie selection	1. User will have a specific movie in mind 2. User will be able to click on the search feature 3. User will start typing the movie name and any similar movie will populate	User types in a movie to search	Movie titles that are similar to the movie typed in will populate	User will view all movies similar to the movie that was searched	Users will be able to conveniently search the movie they want and be able to check out	Fail	Test passes if the movie the user searched is in our database	

We haven't completed all of the above test cases yet, but that's because we're currently resolving issues from previous tests. One that we're currently struggling with, would be the checkout page. This is important since that is the essence of what an e-commerce website is about. Without the ability to check out, how can you really buy anything?

With that being said, we planned on resolving this by working together as a team and finding any more potential errors. In our last meeting, we made a list of things that we need to refine or implement but also bugs that need to be fixed. Besides from the checkout issue, we're also working on ensuring the cart is properly updated and that users can have a way to log in and out as they're currently going to stay logged in until they reset the local host.

Our plan is to have all of this and the listed items in our chats to be completed by our presentation time on Friday. So far excellent progress has been made and we're on track to have everything done by then!

## Section 3: Technical Details, Installation Instructions, Login and Access Credentials and API Keys

---

PinkBox is a full-stack developed website designed to offer a seamless experience for browsing and purchasing movies. Developed with the MERN technology stack, PinkBox delivers dynamic user interactions and efficient server functionality.

### Frontend Architecture

The frontend of PinkBox is constructed using React with the Vite bundler to ensure fast refreshes and quick module loading during the developing process. Component-based architecture creates efficient modular design and code reuse, while routing is managed by React to provide a single-page application experience. Custom CSS modules are used to create responsive and aesthetically pleasing UIX. The website allows for genre-based filtering, modal popups for movie details, and interactive cart functionality.

```
1  @import url('https://fonts.googleapis.com/css2?family=Bebas+Neue&display=swap');
2
3  /* Reset margins and padding for the entire page */
4  * {
5      margin: 0;
6      padding: 0;
7      box-sizing: border-box;
8  }
9
10 html, body {
11     height: 100%;
12     overflow: hidden; /* Keeps the page from scrolling */
13     font-family: 'Bebas Neue', sans-serif;
14 }
15
16 /* Login page styling */
17 .login-page {
18     height: 100vh; /* Changed from min-height */
19     display: flex;
20     justify-content: center; /* Center horizontally */
21     align-items: center; /* Center vertically */
22     background-repeat: no-repeat;
23     background-image: url('/MoviePosterBG2.png');
24     background-size: cover;
25     background-position: center;
26     padding: 20px;
27 }
```

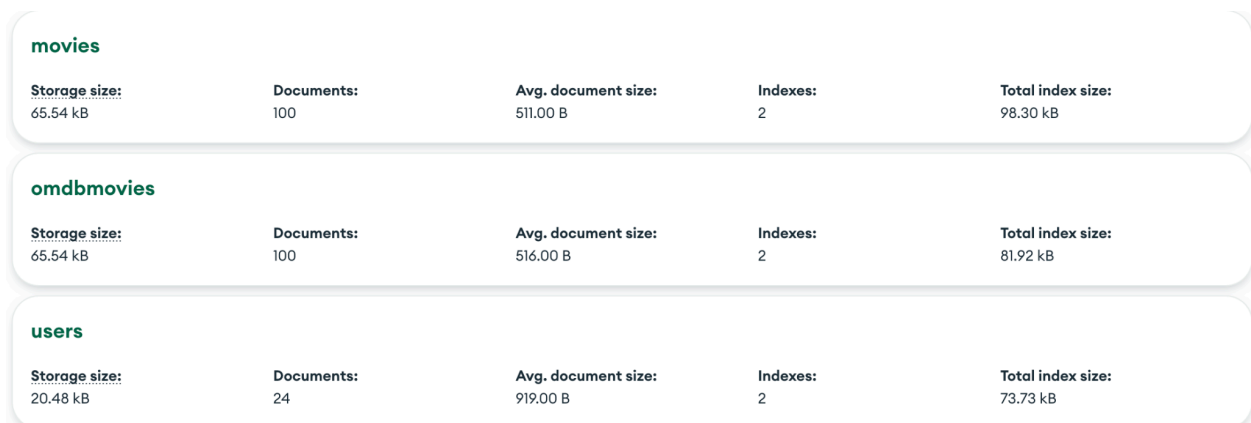
Figure 5: Example CSS file from login page

## Backend Architecture

On the backend, Node.js and Express.js serve as the foundation for routing and logic handling. Routes are modular and efficient in handling tasks such as user registration, login, cart management, and movie uploads. JWT are used for secure session handling, where users receive a token upon login, and this JWT is passed through all of the subsequent APIs in our code. Middleware is implemented to authenticate tokens and protect sensitive endpoints such as “/addtocart” and “/getcart.”

## Database Design

Data is handled and managed in MongoDB Atlas, a cloud-based NoSQL database. We have two main schemas: “Users” and “Movies.” The Users schema contains fields such as username, email, password, and their cart object. The Movies collection stores movie data including ID, title, description, genre, cost, image URL, and streaming URL. These schemas are defined using Mongoose, which provides a structure and schema validation for the MongoDB collections.



<b>movies</b>				
Storage size: 65.54 kB	Documents: 100	Avg. document size: 511.00 B	Indexes: 2	Total index size: 98.30 kB
<b>omdbmovies</b>				
Storage size: 65.54 kB	Documents: 100	Avg. document size: 516.00 B	Indexes: 2	Total index size: 81.92 kB
<b>users</b>				
Storage size: 20.48 kB	Documents: 24	Avg. document size: 919.00 B	Indexes: 2	Total index size: 73.73 kB

Figure 6: Image of schemas in MongoDB

## Installation Instructions

To install PinkBox locally, users should begin by cloning the repository using Git or GitHub Desktop. They must then run “npm install” in both the frontend and backend directories to install dependencies. The frontend can be started by running “npm run dev” through the Vite environment, while the back is started by running “node pinkbox.js” once you’re in the backend files. Make sure to configure a .env file in the backend folder that contains the environment variables necessary for database connection and token authentication.

## **Session Authentication**

Authentication is handled via secure JSON Web Tokens. When a user signs up or logs in, the backend associates a certain JWT for that user which is stored in localStorage. This token is passed through as the “auth-token” in each of the header sections of our APIs. If a token is missing or invalid, a 401 Unauthorized response is returned. Only authenticated users can modify their carts, as expected in most websites.

## **API Keys**

API keys and environment variables are stored securely in the .env files. The website does not reveal sensitive login information to the client, and future improvements include the addition of password hashing and integration with third-party payment processors. Movie images are handled through multer and stored in the backend, with file paths saved in the database for usage.

## **Access Credentials**

Security practices include token-based authentication, environment-based configuration for login information, and structured error handling. All user sessions are validated before they can access their cart or modify it. Planned security upgrades include HTTPS enforcement in production and role-based access control for admin functions such as adding or removing movies.

## Section 4: Risk Management Plan, Software Quality Attributes and explanations

---

During the development of PinkBox, our team recognized that perfection wasn't necessary for success. What mattered most was ensuring that the core features functioned well and effectively met users' needs. That's why we prioritized solving the most critical problems over addressing every minor issue. Outlining user stories helped us determine which features to implement and also allowed us to anticipate potential risks—issues that could arise and would benefit from having mitigation strategies in place beforehand.

The initial risks we identified included incompatible movie formats, corrupted movie files, and copyright infringement. Additional risks, such as account security and site navigation, emerged during development and were addressed as they arose. Since users would be receiving links to media files to watch their purchased movies on various devices, addressing the risk of incompatible formats was essential. PinkBox would inform users about supported formats and provide download links to media players in case playback issues occurred. To mitigate the issue of corrupted movie files, we proposed that movie companies test their files before distribution and offer re-download options or refunds if a file is unusable. Copyright infringement is another major concern when sharing media. To address this, users would be required to accept terms and conditions and be clearly informed about copyright policies before accessing any content.

In addition to risk management, we emphasized software quality through strategies that kept our goals focused and attainable. We utilized Key Process Indicators (KPIs), such as the Evolutionary Strategy and the Utilitarian Strategy, to guide our development approach. The Evolutionary Strategy, described in “The Challenge of Good Enough Software” by James Bach, involves alternating between observation and improvement. This allowed us to avoid solving every problem upfront and instead make adjustments as needed, reducing the risk of rework. Frequent iterations also helped us adapt to changing requirements more effectively. The Utilitarian Strategy helped us concentrate on implementing the system's core functionality with high quality, rather than adding unnecessary complexity that might compromise usability. This approach ensured our product remained simple, stable, and user-focused.

Additionally, Software Configuration Management (SCM) sub-practices such as Incremental Refactoring and the Copy-Merge work model played a critical role in maintaining code stability. These practices allowed us to make more thorough changes and conduct consistent testing, ensuring that commits remained smooth and free of conflicts. By breaking down larger development tasks into smaller, manageable steps, SCM helped enhance the overall effectiveness of our XP approach and supported the reliable delivery of new features.

## References

Asklund, Ulf, and Lars Bendix. *Software Configuration Management Practices for Extreme Programming Teams*. 2004.

Bach, James. *The Challenge of "Good Enough" Software*. 1995. Updated 2003.