



SOFE 2800U Web Programming

Movie Review Website

Group 12

Ibrahim Syed

Mahir Patel

Muska Ebadi

Objective

Our group consists of three members: Muska Ebadi, Mahir Patel, and Ibrahim Syed. For our Web Programming final project, together we created a movie review website called Flicks & Giggles, a website that had the look and feel of a modern movie streaming platform, where users can easily search for films, read what other people are saying, and share their own thoughts about the same movies in a super simple way.

The primary objective of our project is to develop a comprehensive web application that showcases the knowledge we have gained in this course. On the front end, we used HTML, CSS, and JavaScript to build a multi-page interface that matches up with what we needed: a bunch of pages with a consistent look, navigation through the site, movie cards, and review cards, plus lots of interactive features like forms and modals that you'd expect to see on this kind of website. To make things more interesting, we decided to store reviews in the browser using localStorage. However, we took it a step further and added a PHP and MySQL backend, allowing the reviews to be saved to a real database as well. When it comes to functionality, the site supports all the core CRUD operations: users can create and read their own reviews, and we also have a hidden admin mode where an admin can go in and tweak and delete existing reviews. Overall, we aimed to integrate design, client-side scripting, and server-side programming into a fully functional movie review system, one that exceeds the requirements of this assignment.

Outcomes

The primary goal we aim to accomplish through Flicks & Giggles is to establish a virtual platform that enables users to quickly locate films, read candid and direct reviews from fellow audience members, and express their personal views using a simple review form. Our site works with everyday viewers instead of relying solely on professional critics and ratings from large platforms. As a result, everyone is allowed to rate a movie from 1 to 5 stars and post comments, which will be visible immediately on the Browse Reviews page. Once implemented, this design puts more emphasis on active engagement with movies by writing reviews, rather than passively watching trailers.

From a tech standpoint, another key objective is to represent a functional implementation of a web application pipeline using the tools from this course. We want the site to clearly demonstrate how form data is processed, including working HTML forms, client-side validation and logic in JavaScript, storing data for the long term through both browser localStorage and a MySQL database, and how the server-side aspects work with PHP. A clear goal of ours is for every review that users enter to be created, stored, displayed, updated, and deleted reliably, thereby proving that we have implemented all parts of CRUD. We also aim to utilize SQL's built-in aggregate functions (such as averages and counts of ratings) to support features like trending movies and average ratings per film, demonstrating our ability to go beyond the simple "insert and select" examples that are the norm.

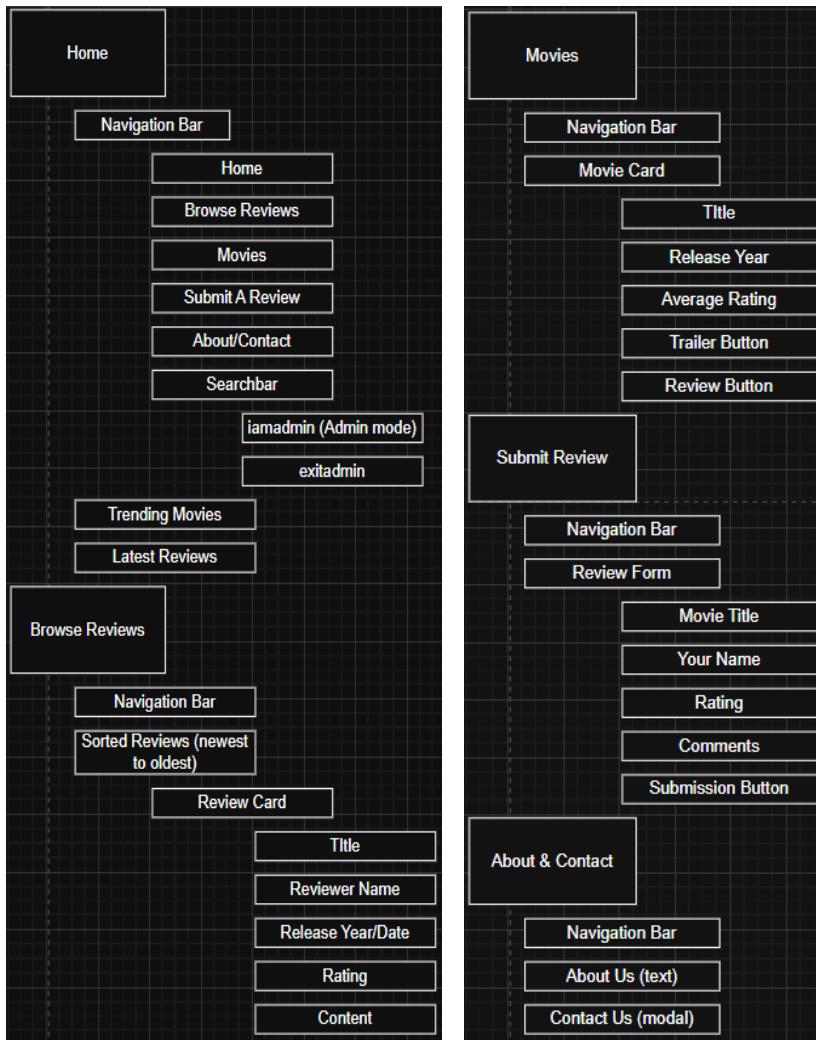
Finally, we want the site to be visually consistent and responsive, so it looks and works smoothly on laptops, tablets, and phones, giving the impression of a real movie platform rather than just a basic class project. To achieve this, we combine a clear and intuitive navigation bar

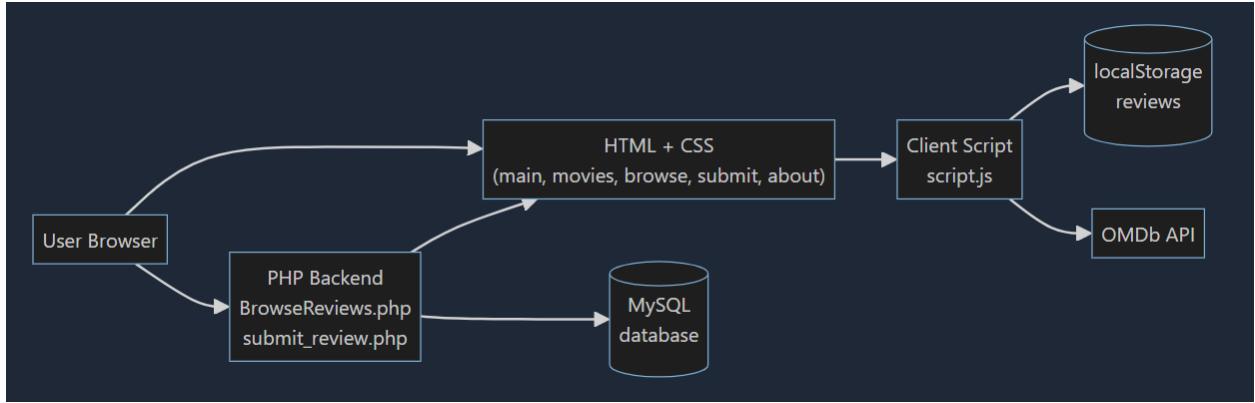
with some repeated movie and review card layouts and a contact form that pops up in a modal window, so the site feels like something people would actually want to use. If we achieve these goals, our website will not only meet the minimum requirements but will also serve as a strong showcase that demonstrates our skills in designing, building, and running a complete web application from scratch, including the database.

Site Structure & Design

The website is structured across several interconnected pages as shown with the flowchart below, all linked through a consistent top navigation bar. The Home page serves as the landing page, highlighting trending movies alongside the most recent reviews. The Browse Reviews page lists all reviews from newest to oldest, providing details such as movie title, release year, review date, and reviewer name. The Movies page presents films in a responsive grid of movie cards, each containing a poster, release year, average rating, and buttons to view the trailer or submit a review. The Submit Review page provides a form for entering a name, movie title, rating, and comments. The localStorage of the browser retains submitted reviews

permanently, while in the PHP backend they are also recorded in a MySQL reviews table, hence reviews are still accessible in the same browser and can be transferred through the database across different sessions and devices. The About & Contact page adheres to the same visual style as the rest of the site, outlining the website's purpose and offering a "Contact Us" button that opens a modal displaying phone, email, and social media contact information. Across all pages, the design emphasizes consistency and readability through a background image, light rounded content cards with subtle shadows, clear typography, and layouts that remain accessible on various screen sizes, creating a clean and modern movie-themed experience.





The system architecture of Flicks & Giggles, as shown in the figure above, begins with the user's browser requesting the HTML and CSS pages (Home, Movies, Browse Reviews, Submit Review and About/Contact) from the web server. Once the page is fully loaded, the client-side script `script.js` runs in the browser and handles the majority of the interactive features by calling the OMDb API through `fetch()`, and storing review data in the browser's `localStorage` to track currently trending movies and the latest reviews. By utilizing the hidden administrative tools implemented in the script, the admin can edit or delete the reviews already in existence. The PHP/MySQL backend is also shown in the diagram for server-side storage for reviews. Each time a user submits a review, the review form data goes from the user's browser to PHP scripts such as `submit_review.php`, which communicate with the MySQL database and insert the review into the reviews table. The `BrowseReviews.php` page is responsible for retrieving the stored reviews in the database and presenting the reviews back to the browser in HTML formatting. This structure clearly separates concerns: the structure and the look are handled by HTML and CSS, the logic on the client-side and the external API communications are handled by JavaScript, `localStorage` provides client-side data retention, and the PHP + MySQL layer ensures that the reviews are securely kept on the server and remain accessible across sessions and devices.

reviews		
	id	PK
int	username	
varchar	movie_title	
text	review_text	
int	rating	
datetime	created_at	

The reviews table on the left has six columns that store everything you need to know about a single movie review. The `id` column is an integer primary key that auto-increments, so each review is automatically given a unique number by MySQL. The `username` column stores the name of the person writing the review as a `VARCHAR`, and it can be empty if the user wants to review anonymously. The `movie_title` column holds the title of the movie as a `VARCHAR`, and it's a required field, so every review is tied to a specific film. The `review_text` column is a `TEXT` field that holds the full written content of the user's review and can take longer comments. The `rating` column keeps the numeric rating the user

gave the movie, and in our PHP code we check to make sure it's always between 1 and 5. Finally the `created_at` column is a `DATETIME` field that records exactly when the review was added and it sets itself to the current time automatically every time a new review is added, saving you from having to remember to do that yourself.

Technical Specifications

For Flicks & Giggles, we utilized a standard yet comprehensive web stack. The front end of the site is built with HTML for the basic page structure, some CSS for achieving the layout and visual styling, and plain JavaScript for making the whole thing interactive. We used HTML to sort out the main sections (the nav bar at the top, the movie grid, the review form, the review list, and that modal contact box that pops up when you need it). CSS ensures that the navigation and movie/review cards are well-aligned on both large and small screens. We also applied the CSS function clamp() to ensure font sizes are responsive across various screen sizes. We attach event listeners on the JavaScript side, which allow buttons and forms to react to user actions, access and alter the DOM, apply fetch() to make inquiries to the OMDb API and retrieve movie data in JSON format, and store user reviews in localStorage so they remain available across sessions. On the server end, we implement PHP (using PDO) along with a MySQL database as our server technology to store the reviews in a durable database, thus providing not only browser storage but also a complete client-server setup.

To run this project on a different machine, it is sufficient to set up a simple local web server. Our site was created with the standard AMP stack, such as XAMPP or WAMP, using a typical Apache/MySQL/PHP 8 configuration. To execute it on a different computer, a person would need to install software such as XAMPP, enable the Apache and MySQL services, and transfer the project folder to the web root directory. After that, they would access phpMyAdmin or the MySQL command line, set up a new database named reviewdatabase, and apply our SQL script to generate the reviews table. In the config.inc.php file, we define the database connection settings as PHP constants DBHOST, DBNAME, DBUSER, DBPASS, and the combined DBCONNSTRING. These values will only have to be modified for a different server to be equivalent to the local MySQL username, password, and host. Once completed, one can access the site at the link <http://localhost/FlicksAndGiggles/main.html>, and the PHP pages, such as BrowseReviews.php and insertReview.php, will automatically connect to the database using the newly updated settings.

Our server-side configuration is straightforward. All PHP files run on the local Apache HTTP server and talk to the database through a PDO connection created with the DSN string in DBCONNSTRING ("mysql:host=".DBHOST.";dbname=".DBNAME.");charset=utf8mb4"). We use the default MySQL port and utf8 (UTF-8MB4), so movie titles and comments with special characters are stored correctly. The insert script (insertReview.php) utilizes a prepared statement with named parameters, ensuring that values from the review form are bound safely. This approach is more secure and robust than building SQL strings by hand.

The database is compact and focused on the review data. We have one table named reviews with six columns: id (an INT primary key that auto increments and uniquely identifies each review), username (VARCHAR(100) to store the reviewer's name), movie_title (VARCHAR(255) to hold the full movie title and year), review_text (TEXT for free form review comments), rating (an unsigned integer restricted in the PHP code to values between 1 and 5), and created_at (DATETIME with a default of CURRENT_TIMESTAMP to store when the review was submitted). This maps directly to the fields in our HTML form and is flexible enough

to support SQL aggregate queries, such as AVG(rating) and COUNT(*) grouped by movie_title, to calculate average ratings and the number of reviews per film. Together, these technical choices, HTML/CSS/JavaScript with localStorage on the client side and a PDO-based PHP/MySQL backend on the server side, give us a complete and reproducible implementation of the Flicks & Giggles movie review site.

Limitation

Although Flicks & Giggles meets the requirements of the course project, it has several technical limitations. First, there is very basic input validation. We check that a rating is between 1 and 5 and that required fields like the movie title are not empty, but we do not do deeper validation or content filtering on names or comments. There is no profanity filter, no length limits on review text and no anti-spam measures like rate limiting or CAPTCHAs. So users could submit malformed, offensive or repeated content and the system would accept it without any additional checks.

The site is also not actually deployed on a public server. It runs locally on a machine using XAMPP or WAMP, so real users can't reach it through a public URL. From a realism point of view, we haven't addressed issues that come with deployment like configuring a production web server, managing domains and SSL certificates or monitoring the site in a live environment.

Additionally, there is no strong security beyond basic prepared statements in PHP. There is no real authentication: anyone can submit a review under any name and the "admin mode" is just activated by typing a secret code in the search bar and storing a flag in localStorage. We do not have password-protected accounts, password hashing, secure sessions, HTTPS encryption or protection against common web attacks like CSRF and XSS beyond the minimal escaping we do when outputting database values. In a real production system this would not be acceptable.

Another major limitation is related to data storage and scalability. Our implementation uses both browser localStorage and a MySQL database at the same time: JavaScript features like trending movies, latest reviews on the home page and the admin edit/delete tools work against localStorage, while the PHP scripts insertReview.php and BrowseReviews.php read from and write to the MySQL reviews table. So reviews saved in one place are not automatically synchronized with the other, which can lead to duplicate data and inconsistent views of the system. This hybrid approach also doesn't scale well: localStorage is tied to a single browser on a single device and has strict size limits and we do not have pagination, indexing or caching strategies for large number of database rows.

The website also relies on the external OMDb API in order to fetch movie posters and details. If the API is slow, offline, or if the key stops working, movie cards will display errors or placeholders instead of actual data. We also do not currently display friendly error messages in the event of API or database failure.

Finally, while the layout is responsive and visually consistent, we haven't fully addressed accessibility and usability concerns. We do not check colour contrast, keyboard-only navigation

or screen-reader support and not every interactive element has ARIA attributes. For a course project this is acceptable, but in a future version we would add robust validation and content filtering, deploy the site to a public server, add real authentication and encryption, unify all review operations around the database instead of localStorage and improve accessibility and scalability so the system is more like a production app.

Conclusion

In conclusion, this project successfully demonstrates our understanding of frontend and backend development by using HTML, CSS, JavaScript, PHP, and MySQL to make the website work properly. We designed and implemented a fully functional movie review platform that integrates external APIs, dynamic client-side features, and database-driven storage. While there are some limitations in terms of security and data integration, the project fulfills the course requirements and provides a solid foundation for further improvements in future versions.