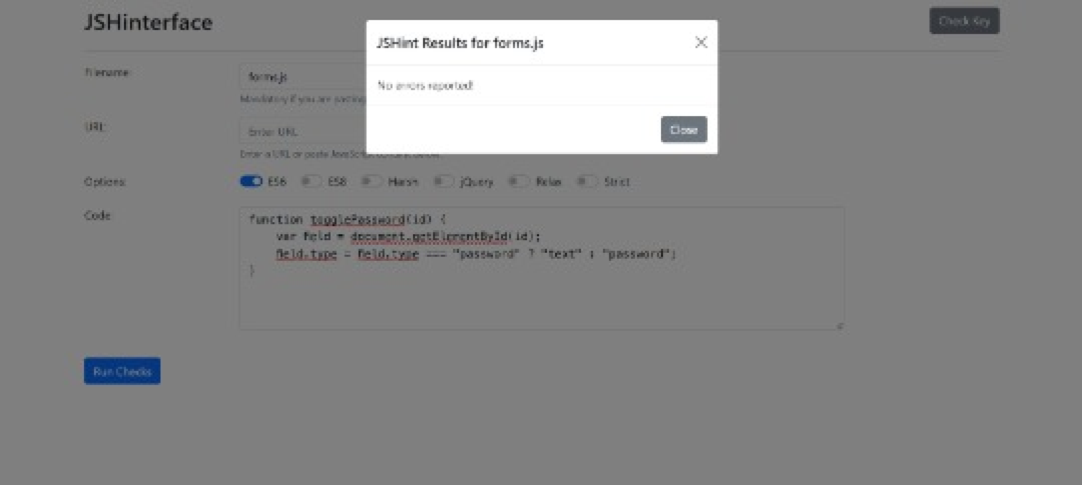


JSHinterface

Check Key

Filename: forms.js

Mandatory if you are pasting

URL: Enter URL

Enter a URL or paste JavaScript code below.

Options: [✓ ES6] ES8 Marsh jQuery Relax Strict

Code:

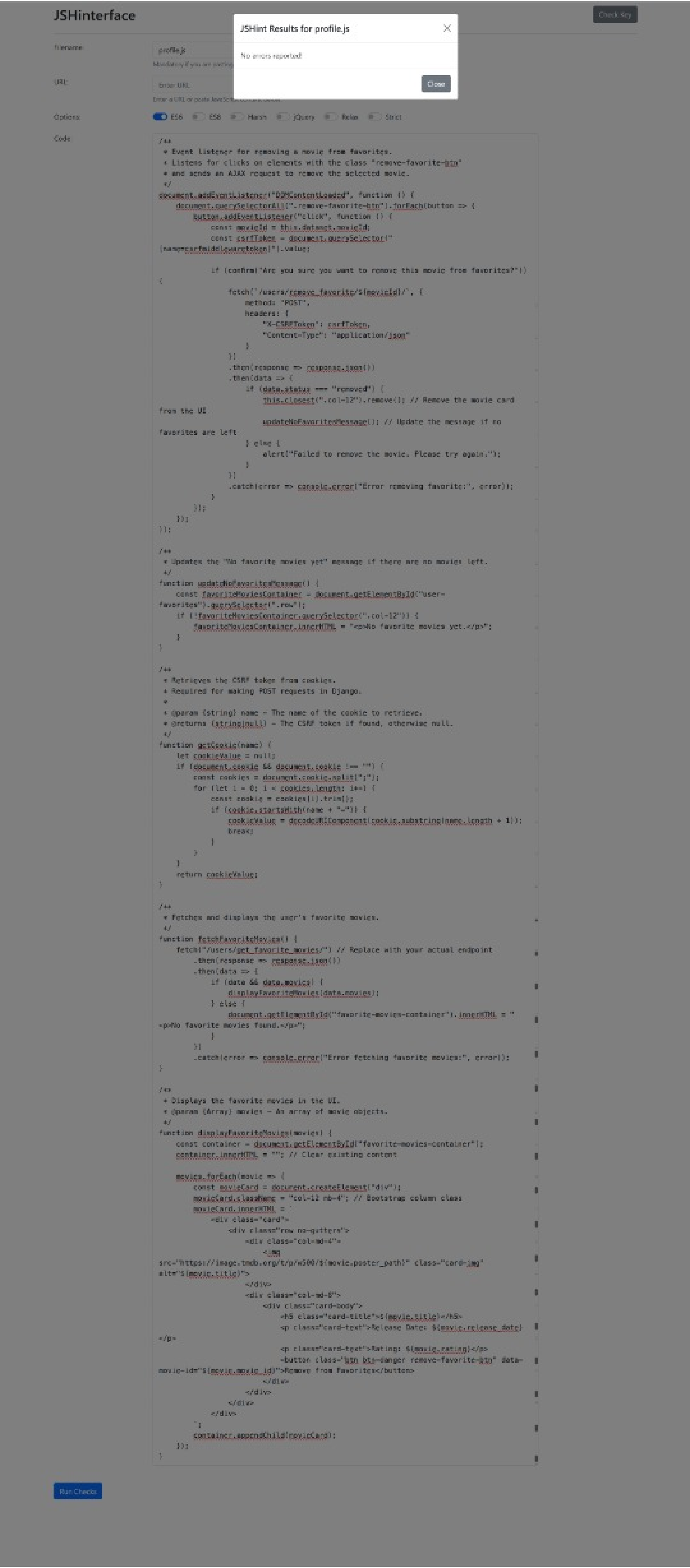
```
function togglePassword(id) {
    var field = document.getElementById(id);
    field.type = field.type === "password" ? "text" : "password";
}
```

Run Checks

JSHint Results for forms.js

No errors reported!

Close



JSHint Results for profile.js

No errors reported!

Filename: profile.js
Mandatory if you are pasting

URL: Enter URL
Enter a URL or paste JavaScript code below.

Options: [✓ ES6] ES8 Harsh jQuery Relax Strict

Code:

```
/**
 * Event Listener for removing a movie from favorites.
 * Listens for clicks on elements with the class "remove-favorite-btn"
 * and sends an AJAX request to remove the selected movie.
 */
document.addEventListener("DOMContentLoaded", function () {
    document.querySelectorAll(".remove-favorite-btn").forEach(button => {
        button.addEventListener("click", function () {
            const movieId = this.dataset.movieId;
            const csrfToken = document.querySelector("
[name=csrfmiddlewaretoken]").value;

            if (confirm("Are you sure you want to remove this movie from favorites?"))
{
                fetch(`/users/remove_favorite/${movieId}/`, {
                    method: "POST",
                    headers: {
                        "X-CSRFToken": csrfToken,
                        "Content-Type": "application/json"
                    }
                })
                .then(response => response.json())
                .then(data => {
                    if (data.status === "removed") {
                        this.closest(".col-12").remove(); // Remove the movie card
from the UI
                        updateNoFavoritesMessage(); // Update the message if no
favorites are left
                    } else {
                        alert("Failed to remove the movie. Please try again.");
                    }
                })
                .catch(error => console.error("Error removing favorite:", error));
            }
        });
    });
});

/**
 * Updates the "No favorite movies yet" message if there are no movies left.
 */
function updateNoFavoritesMessage() {
    const favoriteMoviesContainer = document.getElementById("user-
favorites").querySelector(".row");
    if (!favoriteMoviesContainer.querySelector(".col-12")) {
        favoriteMoviesContainer.innerHTML = "<p>No favorite movies yet.</p>";
    }
}

/**
 * Retrieves the CSRF token from cookies.
 * Required for making POST requests in Django.
 *
 * @param {string} name - The name of the cookie to retrieve.
 * @returns {string|null} - The CSRF token if found, otherwise null.
 */
function getCookie(name) {
    let cookieValue = null;
    if (document.cookie && document.cookie !== "") {
        const cookies = document.cookie.split(";");
        for (let i = 0; i < cookies.length; i++) {
            const cookie = cookies[i].trim();
            if (cookie.startsWith(name + "=")) {
                cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
                break;
            }
        }
    }
    return cookieValue;
}

/**
 * Fetches and displays the user's favorite movies.
 */
function fetchFavoriteMovies() {
    fetch("/users/get_favorite_movies/") // Replace with your actual endpoint
        .then(response => response.json())
        .then(data => {
            if (data && data.movies) {
                displayFavoriteMovies(data.movies);
            } else {
                document.getElementById("favorite-movies-container").innerHTML = "
<p>No favorite movies found.</p>";
            }
        })
        .catch(error => console.error("Error fetching favorite movies:", error));
}

/**
 * Displays the favorite movies in the UI.
 * @param {Array} movies - An array of movie objects.
 */
function displayFavoriteMovies(movies) {
    const container = document.getElementById("favorite-movies-container");
    container.innerHTML = ""; // Clear existing content

    movies.forEach(movie => {
        const movieCard = document.createElement("div");
        movieCard.className = "col-12 mb-4"; // Bootstrap column class
        movieCard.innerHTML = `
            <div class="card">
                <div class="row no-gutters">
                    <div class="col-md-4">
                        <img
src="https://image.tmdb.org/t/p/w500/${movie.poster_path}" class="card-img"
alt="${movie.title}">
                    </div>
                    <div class="col-md-8">
                        <div class="card-body">
                            <h5 class="card-title">${movie.title}</h5>
                            <p class="card-text">Release Date: ${movie.release_date}
</p>
                            <p class="card-text">Rating: ${movie.rating}</p>
                            <button class="btn btn-danger remove-favorite-btn" data-
movie-id="${movie.movie_id}">Remove from Favorites</button>
                        </div>
                    </div>
                </div>
            </div>
        `;
        container.appendChild(movieCard);
    });
}
```

JSHinterface
Check Key

JSHint Results for movie_details.js

No errors reported!

Close

Filename: movie_details.js

Mandatory if you are pasting

URL: Enter URL

Enter a URL or paste JavaScript code below.

Options: ES6 [✓] ESB Harsh jQuery Relax Strict

Code

```javascript
/**
 * Extracts the movie ID from the URL.
 * @returns {number|null} The extracted movie ID or null if not found.
 */
function extractMovieIdFromUrl() {
    const path = window.location.pathname;
    const match = path.match(/\/movies\/(\d+)\/?$/);
    return match ? parseInt(match[1], 10) : null;
}

/**
 * Fetches movie details and updates the DOM.
 * @param {number} movieId - The ID of the movie.
 */
function fetchMovieDetails(movieId) {
    fetch(`/movies/api/details/${movieId}/`)
        .then(response => response.json())
        .then(data => {
            document.getElementById("movie-poster").src =
`https://image.tmdb.org/t/p/w500/${data.poster_path}`;
            document.getElementById("movie-title").textContent = data.title;
            document.getElementById("release-date").textContent = data.release_date;
            document.getElementById("rating").textContent = data.vote_average;
            document.getElementById("overview").textContent = data.overview;
        });
}

/**
 * Updates the UI of the favorite button based on favorite status.
 */
function updateFavoriteButtonUI(button, isFavorite) {
    button.innerHTML = `<i class="fa-${isFavorite ? "solid" : "regular"} fa-heart">
</i> ${isFavorite ? "Remove" : "Add to Favs"}`;
    button.classList.toggle("btn-danger", isFavorite);
    button.classList.toggle("btn-outline-danger", !isFavorite);
}

/**
 * Toggles the favorite status of a movie and updates the UI.
 */
function toggleFavorite(button, event) {
    if (event) {
        event.stopPropagation();
    }
    button.disabled = true;

    console.log("Toggling favorite for movie:", button.dataset.movieId);

    const movieId = parseInt(button.dataset.movieId, 10);
    const formData = new URLSearchParams({
        "movie_id": movieId,
        "title": button.dataset.title || "",
        "poster_path": button.dataset.posterPath || "",
        "release_date": button.dataset.releasedate || "",
        "rating": button.dataset.rating || "0"
    });

    fetch("/movies/toggle_favorite/", {
        method: "POST",
        headers: {
            "X-CSRFToken": getCookie("csrftoken"),
            "Content-Type": "application/x-www-form-urlencoded"
        },
        body: formData.toString()
    })
    .then(response => response.json())
    .then(data => {
        console.log("Favorite toggle response:", data);
        const isNowFavorite = data.status === "added";
        console.log("UI should reflect favorite:", isNowFavorite);
        updateFavoriteButtonUI(button, isNowFavorite);
    })
    .catch(error => console.error("Error toggling favorite:", error))
    .finally(() => {
        button.disabled = false;
    });
    console.log("Sending poster_path:", button.dataset.posterPath);
}

/**
 * Debounces the favorite button click to prevent multiple requests.
 * @param {Function} func - The function to debounce.
 * @param {number} delay - The delay in milliseconds.
 * @returns {Function} - A debounced version of the function.
 */
function debounce(fn, delay) {
    let timeoutId;
    return function (...args) {
        clearTimeout(timeoutId);
        timeoutId = setTimeout(() => fn.apply(this, args), delay);
    };
}


/**
 * Initializes the favorite button.
 * @param {number} movieId - The ID of the movie.
 */
function initializeFavoriteButtons(movieId) {
    const favButton = document.querySelector(".favorite-btn");
    if (!favButton) return;

    // Fetch user's favorite movies and check if the current movie is a favorite
    fetch("/movies/get_favorite_movies/")
        .then(response => response.json())
        .then(data => {
            const favoriteMovieIds = data.favorite_movie_ids;
            // Initialize main movie button first
            initializeMainMovieFavoriteButton(favButton, movieId, favoriteMovieIds);
            // Then initialize similar movie buttons
            initializeSimilarMoviesFavoriteButtons(favoriteMovieIds);
        })
        .catch(error => console.error("Error fetching favorites:", error));
}

/**
 * Initializes main movie favorite button and its event listener.
 */
function initializeMainMovieFavoriteButton(favButton, movieId, favoriteMovieIds) {
    // REMOVE ANY PREVIOUS EVENT LISTENERS
    favButton.replaceWith(favButton.cloneNode(true));
    const newButton = document.querySelector(".favorite-btn");

    // Check if the main movie is in the user's favorites
    const isFavorite = favoriteMovieIds.includes(movieId);
    updateFavoriteButtonUI(newButton, isFavorite);

    // Create debounced version of toggleFavorite inside here
    const debouncedToggle = debounce((event) => {
        toggleFavorite(newButton, event);
    }, 300);

    newButton.addEventListener("click", debouncedToggle);
}

/**
 * Initializes favorite buttons for similar movies.
 * @param {Array} favoriteMovieIds - Array of favorite movie IDs.
 */
function initializeSimilarMoviesFavoriteButtons(favoriteMovieIds) {
    const similarMovieButtons = document.querySelectorAll(".similar-movie-fav-btn");

    similarMovieButtons.forEach(button => {
        const movieId = parseInt(button.dataset.movieId, 10);

        // Check if this similar movie is in the user's favorites
        const isFavorite = favoriteMovieIds.includes(movieId);
        updateFavoriteButtonUI(button, isFavorite);

        // Create debounced version of toggleFavorite inside here for similar movies
        const debouncedToggle = debounce((event) => {
            toggleFavorite(button, event);
        }, 300);

        button.addEventListener("click", debouncedToggle);
    });
}



/**
 * Initializes backdrop image carousel.
 */
function initializeBackdropCarousel() {
    const backdrops = document.querySelectorAll('.backdrop-image');
    if (!backdrops.length) return;

    let index = 0;
    function showNextBackdrop() {
        backdrops.forEach(backdrop => backdrop.style.opacity = 0);
        backdrops[index].style.opacity = 1;
        index = (index + 1) % backdrops.length;
    }

    showNextBackdrop();
    setInterval(showNextBackdrop, 5000);
}

/**
 * Initializes logo image carousel.
 */
function initializeLogoCarousel() {
    const logos = document.querySelectorAll('.logo-image');
    if (!logos.length) return;

    let index = 0;
    function showNextLogo() {
        logos.forEach(logo => logo.style.opacity = 0);
        logos[index].style.opacity = 1;
        index = (index + 1) % logos.length;
    }

    showNextLogo();
    setInterval(showNextLogo, 3000);
}

/**
 * Handles review actions (edit & delete).
 */
function setupReviewActions() {
    document.querySelectorAll(".edit-review-btn").forEach(button => {
        button.addEventListener("click", function () {
            document.getElementById("edit-review-id").value = this.dataset.reviewId;
            document.getElementById("edit-review-text").value =
this.dataset.reviewText;
            document.getElementById("edit-review-rating").value =
this.dataset.reviewRating;

            new bootstrap.Modal(document.getElementById("editReviewModal")).show();
        });
    });

    document.getElementById("editReviewForm").addEventListener("submit", function (e)
{
        e.preventDefault();
        const reviewId = document.getElementById("edit-review-id").value;
        const reviewText = document.getElementById("edit-review-text").value;
        const reviewRating = document.getElementById("edit-review-rating").value;

        fetch(`/reviews/update/${reviewId}/`, {
            method: "POST",
            headers: { "X-CSRFToken": getCookie("csrftoken"), "Content-Type":
"application/json" },
            body: JSON.stringify({ review_text: reviewText, rating: reviewRating })
        })
        .then(response => response.json())
        .then(data => {
            if (data.success) {
                document.getElementById(`review-text-${reviewId}`).textContent =
reviewText;
                // Update the rating display
                const ratingElement = document.getElementById(`review-
rating-${reviewId}`);
                if (ratingElement) {
                    ratingElement.textContent = `${reviewRating}/5`;
                }

bootstrap.Modal.getInstance(document.getElementById("editReviewModal")).hide();
            } else {
                alert("Error updating review.");
            }
        });
    });

    document.querySelectorAll(".delete-review-btn").forEach(button => {
        button.addEventListener("click", function () {
            const reviewId = this.dataset.reviewId;
            if (!confirm("Are you sure you want to delete this review?")) return;

            fetch(`/reviews/delete/${reviewId}/`, {
                method: "POST",
                headers: { "X-CSRFToken": getCookie("csrftoken") }
            })
            .then(response => response.json())
            .then(data => {
                if (data.success)
document.getElementById(`review-${reviewId}`).remove();
                else alert("Error deleting review.");
            });
        });
    });
}

/**
 * Retrieves CSRF token from cookies.
 */
function getCookie(name) {
    return document.cookie.split("; ").reduce((acc, cookie) => {
        const [key, value] = cookie.split("=");
        return key === name ? decodeURIComponent(value) : acc;
    }, null);
}

/**
 * Executes initialization functions after the DOM content is fully loaded.
 */
document.addEventListener("DOMContentLoaded", function () {
    const movieId = extractMovieIdFromUrl();
    if (movieId) {
        fetchMovieDetails(movieId);

        // Initialize both the main movie's and similar movies' favorite buttons
        initializeFavoriteButtons(movieId);

        initializeBackdropCarousel();
        initializeLogoCarousel();
        setupReviewActions();
    } else {
        console.warn("Movie ID not found in URL. Check if the URL format is
correct.");
    }
});
```

JSHinterface
Check Key
Filename
movies.js
Mandatory if you are pasting
URL
Enter URL
Enter a URL or paste JavaScript content below
Options
ES6
ES8
Mocha
jQuery
Relax
Strict
Code
Run Checks
Close

JSHint Results for movies.js

No errors reported!

Options: [✓ ES6] ES8 Mocha jQuery Relax Strict

```
/**
 * Adds a "Back to Top" button that appears when the user scrolls down.
 */
function initializeBackToTopButton() {
    const backToTopButton = document.getElementById("back-to-top");
    if (!backToTopButton) return;

    window.addEventListener("scroll", function () {
        backToTopButton.style.display = window.scrollY > 300 ? "flex" : "none";
    });

    backToTopButton.addEventListener("click", function () {
        window.scrollTo({ top: 0, behavior: "smooth" });
    });
}

/**
 * Initializes favorite buttons, ensuring favorite status persists and event listeners
are bound.
 */
function initializeFavoriteButtons() {
    document.querySelectorAll(".favorite-btn").forEach(button => {
        const isFavorite = button.dataset.isFavorite === "true";
        updateFavoriteButtonUI(button, isFavorite);

        button.addEventListener("click", function (event) {
            toggleFavorite(this, event);
        });
    });
}

/**
 * Debounces the favorite button click to prevent multiple rapid requests.
 * @param {Function} func - The function to debounce.
 * @param {number} delay - The delay in milliseconds.
 * @returns {Function} - A debounced version of the function.
 */
function debounceFavoriteClick(func, delay) {
    let timeoutId;
    return function (...args) {
        if (timeoutId) {
            clearTimeout(timeoutId);
        }

        const context = this;

        timeoutId = setTimeout(() => {
            func.apply(context, args);
        }, delay);
    };
}

/**
 * Toggles the favorite status of a movie and updates the UI.
 */
function toggleFavorite(button, event) {
    if (event && typeof event.stopPropagation === "function") {
        event.stopPropagation();
    }

    button.disabled = true;

    const movieId = parseInt(button.dataset.movieId, 10);
    const formData = new URLSearchParams({
        "movie_id": movieId,
        "title": button.dataset.title || "",
        "poster_path": button.dataset.posterpath || "",
        "release_date": button.dataset.releasedate || "",
        "rating": button.dataset.rating || "0"
    });

    fetch("/movies/toggle_favorite/", {
        method: "POST",
        headers: {
            "X-CSRFToken": getCookie("csrftoken"),
            "Content-Type": "application/x-www-form-urlencoded"
        },
        body: formData.toString()
    })
        .then(response => response.json())
        .then(data => {
            const isNowFavorite = data.status === "added";
            updateFavoriteButtonUI(button, isNowFavorite);
            button.dataset.isFavorite = isNowFavorite.toString();
        })
        .catch(error => console.error("Error toggling favorite:", error))
        .finally(() => {
            button.disabled = false;
        });
}

/**
 * Updates the UI of the favorite button based on favorite status.
 */
function updateFavoriteButtonUI(button, isFavorite) {
    button.innerHTML = `<i class="fa-${isFavorite ? "solid" : "regular"} fa-heart">
</i> ${isFavorite ? "Remove" : "Add to Favs"}`;
    button.classList.toggle("btn-danger", isFavorite);
    button.classList.toggle("btn-outline-danger", !isFavorite);
}

/**
 * Fetches and displays movie details from the server.
 */
function initializeMovieDetails() {
    const movieIdMatch = window.location.pathname.match(/\/movies\/(\d+)\//);
    if (!movieIdMatch) return;

    fetch(`/movies/${movieIdMatch[1]}/`)
        .then(response => response.json())
        .then(data => {
            document.getElementById("movie-title").textContent = data.title;
            document.getElementById("movie-overview").textContent = data.overview;
            document.getElementById("movie-rating").textContent = data.vote_average;
            document.getElementById("movie-poster").src =
`https://image.tmdb.org/t/p/w500/${data.poster_path}`;
        })
        .catch(error => console.error("Error fetching movie details:", error));
}

/**
 * Handles review form submission via AJAX.
 */
function initializeReviewForm() {
    const reviewForm = document.getElementById("review-form");
    if (!reviewForm) return;

    reviewForm.addEventListener("submit", function (event) {
        event.preventDefault();
        const formData = new FormData(reviewForm);
        const movieId = window.location.pathname.split("/")[2];

        fetch(`/movies/${movieId}/submit_review/`, {
            method: "POST",
            headers: { "X-CSRFToken": getCookie("csrftoken") },
            body: formData
        })
            .then(response => response.json())
            .then(data => {
                if (data.status === "success") location.reload();
                else alert("Error submitting review. Please try again.");
            })
            .catch(error => console.error("Error submitting review:", error));
    });
}

let currentPage = 1;
let totalPages = Infinity;
let isFetching = false;
const fetchedPages = new Set();

/**
 * Debounces a function to limit its execution until a certain time has passed
 * after the last call. Prevents excessive function calls from rapid events.
 *
 * @param {Function} func - The function to debounce.
 * @param {number} delay - The delay in milliseconds.
 * @returns {Function} - A debounced version of the function.
 */
function debounce(func, delay) {
    let timer;
    return function (...args) {
        clearTimeout(timer);
        timer = setTimeout(() => func(...args), delay);
    };
}

/**
 * Fetches and appends the next page of movies to the DOM.
 * Ensures that movies are not duplicated and updates the current page and total
pages.
 */
function loadMoreMovies() {
    if (isFetching || currentPage >= totalPages || fetchedPages.has(currentPage + 1))
{
        return;
    }

    isFetching = true;
    let nextPage = currentPage + 1;
    fetchedPages.add(nextPage);

    const sortBySelect = document.getElementById("sortSelect");
    let sortByValue = "popularity.desc";
    if (sortBySelect) {
        sortByValue = sortBySelect.value;
    }

    fetch(`/movies/?sort_by=${sortByValue}&page=${nextPage}`, {
        headers: { "X-Requested-With": "XMLHttpRequest" }
    })
        .then(response => response.json())
        .then(data => {
            if (data.movies.length > 0) {
                appendMoviesToDOM(data.movies);
                currentPage = nextPage;
                totalPages = data.total_pages;
            } else {
                console.warn(`No movies found for page ${nextPage}`);
            }
        })
        .catch(error => {
            console.error("Error fetching more movies:", error);
        })
        .finally(() => {
            isFetching = false;
        });
}

/**
 * Appends movie cards to the DOM, creating HTML elements based on the movie data.
 *
 * @param {Array} movies - An array of movie objects to append to the DOM.
 */
function appendMoviesToDOM(movies) {
    const movieContainer = document.querySelector(".row.mt-4");
    if (!movieContainer) return;

    const movieRow = document.createElement("div");
    movieRow.className = "row g-4";

    movies.forEach(movie => {
        const isFavorite = movie.is_favorite;
        const movieCard = document.createElement("div");
        movieCard.className = "col-12 col-md-6 col-lg-4 col-xxl-3 card-deck d-flex";

        movieCard.innerHTML = `
            <div class="card mb-4 shadow-sm" style="height: 100%;">
                <img src="${movie.poster_path ?
`https://image.tmdb.org/t/p/w500/${movie.poster_path}` : '/static/images/img-not-
found.png'}"
                    class="card-img-top" alt="${movie.title}" style="object-fit: cover;
height: 300px;">
                <div class="card-body d-flex flex-column justify-content-between">
                    <h5 class="card-title">${movie.title}</h5>
                    <p class="card-text">${movie.overview.split(" ").slice(0,
20).join(" ")}...</p>
                    <div class="btn-wrapper text-center d-flex justify-content-
between">
                        <a href="/movies/${movie.id}/" class="btn btn-dark card-
link">Show More</a>
                        <button class="btn ${isFavorite ? 'btn-danger' : 'btn-outline-
danger'} card-link favorite-btn btn-sm"
                                data-movie-id="${movie.id}" data-
title="${movie.title}"
                                data-posterpath="${movie.poster_path || ''}"
                                data-releasedate="${movie.release_date || ''}"
                                data-rating="${movie.vote_average || '0'}"
                                data-is-favorite="${isFavorite}">
                            <i class="fa ${isFavorite ? 'fa-solid' : 'fa-regular'} fa-
heart"></i>
                            ${isFavorite ? "Remove" : "Add to Favs"}
                        </button>
                    </div>
                </div>
            </div>
        `;

        movieRow.appendChild(movieCard);
    });

    movieContainer.appendChild(movieRow);
    initializeFavoriteButtons();
}

/**
 * Sets up infinite scrolling by adding a debounced scroll event listener.
 * Loads the first page of movies immediately.
 */
function setupInfiniteScroll() {
    window.addEventListener("scroll", debounce(() => {
        const scrollThreshold = document.body.offsetHeight - 200;
        const scrollPosition = window.innerHeight + window.scrollY;

        if (scrollPosition >= scrollThreshold && currentPage < totalPages &&
!isFetching) {
            loadMoreMovies();
        }
    }, 250));
}

/**
 * Retrieves the value of a cookie by name.
 */
function getCookie(name) {
    return document.cookie.split("; ").reduce((cookieValue, cookie) => {
        if (cookie.startsWith(name + "=")) return decodeURIComponent(cookie.split("=")
[1]);
        return cookieValue;
    }, null);
}

/**
 * Initializes various functionalities after the DOM content is fully loaded.
 */
document.addEventListener("DOMContentLoaded", function () {
    initializeBackToTopButton();
    initializeMovieDetails();
    initializeReviewForm();
    setupInfiniteScroll();
    initializeFavoriteButtons();

    fetch("/movies/get_favorite_movies/")
        .then(response => response.json())
        .then(data => {
            document.querySelectorAll(".favorite-btn").forEach(button => {
                const movieId = parseInt(button.dataset.movieId);
                if (isNaN(movieId)) {
                    console.error("Invalid movie ID:", button);
                    return;
                }
                const isFavorite = data.favorite_movie_ids.includes(movieId);
                updateFavoriteButtonUI(button, isFavorite);
                button.dataset.isFavorite = isFavorite.toString();
            });
        })
        .catch(error => console.error("Error fetching favorites:", error));
});
```