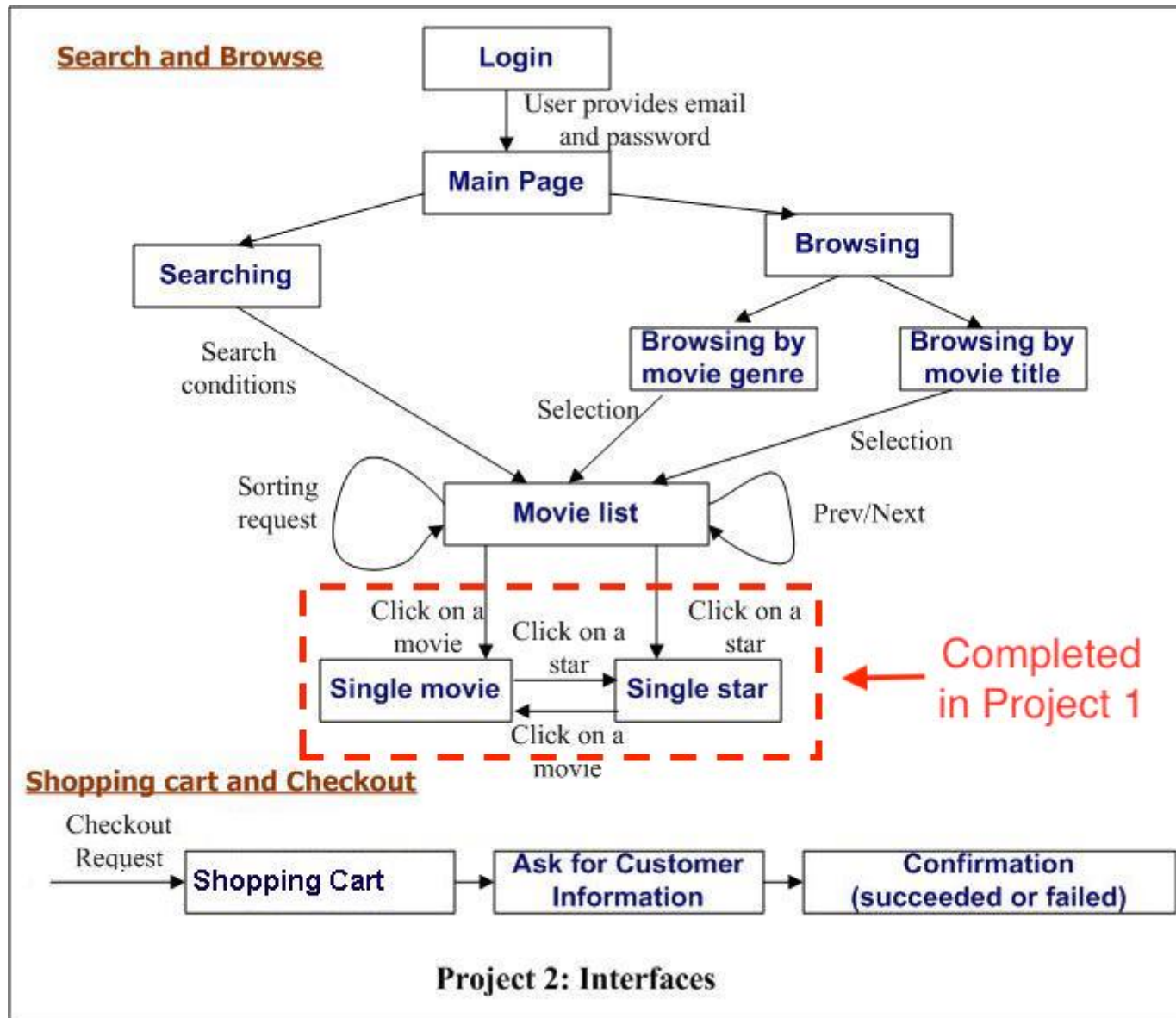


Project 2: Developing Fabflix Website

Due Apr 30 by 11:45pm **Points** 100 **Available** after Apr 12 at 12am

▼ Overview

In this project, you will build functionalities such as browsing movies by category, searching movies by condition, adding movies to the shopping cart, etc. The image below shows the workflow you should follow to create your website. See tasks for details.



You could:

- Use SPA (Single-page-application) instead of multiple-pages-design. As long as the content and flow of each page are maintained.
- You are encouraged to use your imagination to enhance the functionalities. You can use any appropriate client-side technologies to enhance your website's user interface and user experience (e.g., Javascript, CSS, HTML, etc.).

You must NOT

- **Use JSP to generate HTML pages in Project 2. We only allow the architecture in which Serve Generated HTML and the micro-services architecture ("backend-frontend separation").**
 - Implement the shopping cart with a database instead of an HTTP Session.
 - Implement any functionality (sorting, pagination, etc.) using framework or scaffolding tools like DataTables.
 - Implement any functionality in the web app by hard coding. e.g., Hard code all genre names in HTML instead of retrieving them from databases.
 - Store data directly in the frontend instead of the database.
-

▼ Task 1: Implement the Login Page

- The landing page of Fabflix could either be
 - Login Page directly. Any other URLs should be redirected to Login Page if you have not logged in yet.
 - **Main Page** with a "Login" link/button to the Login Page. Any other URLs should be redirected to Login Page if you have not logged in yet.
 - Customer cannot search or browse until login.
 - To log into the system, customers must enter the correct email and password.
 - You must use **HTTP POST** instead of **HTTP GET** so that the username and password will not be displayed on the address bar.
 - Customers' passwords should be hidden (as "****") when entered in the input box.
 - When authenticating user input information, columns `email` and `password` in the `customers` table can be used for reference.
 - Show different error messages when the user enters the wrong information.
 - After logging in, customers should be redirected to **Main Page**, which will be implemented in the next Task.
 - Optionally, add a "Logout" button to all pages after login.
-

▼ Task 2: Implement the Main Page (with Search and Browse)

Customers will be directed to Main Page after logging in. Customers can search and browse movies on the Main Page.

- If you separate search and browse pages, you must have the links to them on the Main Page.

- Customers need to be logged in to perform a search or browse. Optionally, Main Page could be your landing page and have a "Login" link/button to Login Page.
- **Searching:**
 - Customers can search for movies by single or multiple conditions:
 - title
 - year (should not support substring search)
 - director
 - star's name
 - When customers search for movies by multiple conditions, you should use **AND** logic to combine conditions.
 - Customers can click on the search button to jump to Movie List Page showing all the movies matching the search condition.
 - **Substring matching:**
 - Should support "substring matching" so that customers do not have to provide an exact value for an attribute. For example, if search by title keyword: "term", movies such as "Terminal" and "Terminator" should be returned.
 - Only String (VARCHAR) fields are required to support substring matching. Thus, year should not support it.
 - You can use the "LIKE" and "ILIKE" SQL operators for pattern matching in any way you wish. For example:

```
SELECT "column_name"
FROM "table_name"
WHERE "column_name" LIKE pattern
```
- Note:** **pattern** often consists of wildcards.
- Here are some examples:
 - 'A_Z': All strings that start with 'A', then follow by a single character, and end with 'Z'. E.g. 'ABZ' and 'A2Z'. However, 'AKKZ' does not satisfy the condition.
 - 'ABC%': All strings that start with 'ABC'. E.g. 'ABCD' and 'ABCABC'.
 - '%XYZ': All strings that end with 'XYZ'. E.g. 'WXYZ' and 'ZZXYZ'.
 - '%AN%': All strings that contain the pattern 'AN' anywhere. E.g. 'LOS ANGELES' and 'SAN FRANCISCO'.
 - Customer input should be keywords. Should **NOT** expect customer input to have any wildcard characters.
 - **Note:** Include how and where you use the LIKE/ILIKE predicate in your README.md.
- **Browsing:** Customers can browse movies by either genre or titles.
 - When browsing by movie genres:
 - Customers should have a list of all hyperlinked genres sorted alphabetically.

- - Customers can click on a hyperlinked genre to jump to Movie List Page showing all the movies matching the genre.
 - When browsing by movie title:
 - Customers should have a list of sorted single alphanumeric (0,1,2,3..A,B,C...X,Y,Z) characters. Additionally, include a character "*" in this list.
 - Customers can click on a hyperlinked character to jump to Movie List Page and show all the movies whose titles start with that character (for "*", it should match any movie whose title starts with non-alphanumeric characters).
 - Letters are case insensitive.
-

▼ Task 3: Extend Project 1

Extend Movie List Page

- Customers can find search or browse results on this page.
 - The results should have at least the following information about the movies:
 - title (hyperlinked to the corresponding Single Movie Page)
 - year
 - director
 - first three genres
 - sorted by alphabetical order.
 - each hyperlinked, as equivalent to browsing by this genre.
 - first three stars
 - sorted by the number of movies played (found in moviedb), decreasing order. Use alphabetical order to break ties.
 - each hyperlinked to the corresponding Single Star Page.
 - rating
- **Sorting:** Customers can sort the list by either first "title" then "rating", or first "rating" then "title", both in either ascending or descending order. (First, "title" then "rating" means that use title to sort and use rating to break ties.)
- **Previous/Next:**
 - As many movies could need to be shown on the Movie List Page, each page should only display a certain number (N) of movies.
 - And the page should have "Prev"/"Next" buttons, which allow customers to view all the movies. Your website should allow customers to change the number of movies ("N") shown per page.

- Customers can click on a drop-down menu to choose "N" from a list of predefined values: 10, 25, 50, 100.
- Pagination results should be fetched from the database. You can NOT cache more than 100 records in the frontend or backend servlet.

Extend Single Pages

- **Single Movie Page** should add:
 - all genres (same as MovieListPage, but not only the first 3)
 - sorted by alphabetical order.
 - each hyperlinked, as equivalent to browsing by this genre.
 - all stars (same as MovieListPage, but not only the first 3)
 - sorted by the number of movies played (found in moviedb), decreasing order. Use alphabetical order to break ties.
 - each hyperlinked to the corresponding Single Star Page.
 - rating (if not done in Project 1).
- **Single Star Page** should add:
 - all movies
 - sorted by year, decreasing order. Use alphabetical order to break ties.
 - each hyperlinked to the corresponding Single Movie Page.

Jump Functionality

- Customers can jump among Movie List Page, Single Movie Page, and Single Star Page using buttons or links instead of browser history.
 - Customers can jump between Single Movie Page and Single Star Page following hyperlinks. (same as Project 1)
 - Customers can jump back to Movie List Page from any single page. (same as Project 1)
 - **The status of the Movie List Page should be maintained if returned from single pages.** For example, if a customer search for "Jacky Chan" then goes to the third page of the search result and clicks the movie "Police Story" to a Single Movie Page. The customer then clicks the star "Brigitte Lin" to a Single Star Page. The customer should now be able to click a button to go back to Movie List Page on the third page for the search result. The returned Movie List Page should have the same search/browse condition, same sorting, and same pagination setup as before. Note: you should use session instead of browser history.

▼ Task 4: Implement the Shopping Cart

Customers can purchase movies (multiple copies allowed). You can generate a random price in moviedb for each movie. The to-be-purchased movies can be stored in the shopping cart.

- **Shopping Cart Page:**

- Should display:
 - movie title;
 - quantity of each movie;
 - price of each movie (you can decide, possibly randomize some value in moviedb);
 - total price.
- Should allow customers to modify the quantity
 - increase quantity;
 - decrease quantity;
 - delete (should remove this movie from the shopping cart).
- Should have a "Proceed to payment" button on the Payment Page.

- **Payment Page:**

- Should display the total price of the shopping cart.
- Should ask customers for
 - first name of the credit card holder;
 - last name of the credit card holder;
 - credit card number;
 - expiration date.
- Should have a "Place Order" button to proceed to the Place Order action.

- **Place Order action:**

- This action should be using HTTP POST instead of HTTP GET.
- The transaction succeeds only if customers can provide correct payment information which matches a record in the credit cards table (not those in the customers table).
 - If the transaction succeeds, it should be recorded in the system (in the "sales" table), and a confirmation page should be displayed. You might need to alter the "sales" table to support multiple movie copies.
 - If failed, it should return to the Payment Page with an error message for customers to re-enter payment information.

- **Add to Shopping Cart Button:** For all pages that display movie information (Movie List Page and Single Movie Page), each movie should have an "Add to Shopping Cart" button. Customers can click the button to increase the quantity of this movie in the shopping

cart. In addition, a message should display to indicate success/failure.

- **Checkout Button:** All pages except the Login Page should have a "Checkout" button. Customers should be directed to the Shopping Cart Page when clicking the "Checkout" button.

▼ Examples and Resources

Form example: HTTP Form for Login  <https://github.com/UCI-Chenli-teaching/cs122b-project2-form-example>

This example shows how HTML <form> sends user input to the backend. The HTML <form> tag is used for creating a form for user input. A form can contain text fields, checkboxes, radio buttons, and more. Forms are used to pass user data to a specified URL.

Session example: HTTP Session for Login  <https://github.com/UCI-Chenli-teaching/cs122b-project2-session-example>

This example shows how to use HTTP Session to maintain user-related information on the server, such as a shopping cart/list. HTTP requests are stateless. However, the server must differentiate between logged-in users and not logged-in users. To keep a user's status during HTTP request exchanges with the server, we need to use HTTP Session.

Login and Cart example  <https://github.com/UCI-Chenli-teaching/cs122b-project2-login-cart-example>

This example uses the Micro Services architecture design to implement the login, session, and form features.

Check MySQL Slow Query Log

The following steps show how to enable the logging of slow queries in MySQL and check the logs of slow queries.

In the first terminal, do the following:

```
shell> mysql -u root -p
mysql> SET GLOBAL slow_query_log = 'ON';
mysql> SHOW GLOBAL VARIABLES LIKE 'slow\_%'; SHOW GLOBAL VARIABLES LIKE 'long\_%';
mysql> set GLOBAL long_query_time=1.1, SESSION long_query_time=1.1;
mysql> select @@long_query_time;
```

Open a new terminal, and run the following command:

```
shell> sudo tail -f [The file of "slow_query_log_file"]
```


Go to the first terminal, and run the following:

```
mysql> select sleep(1.0);
mysql> select sleep(1.2);
```

The second terminal should show the second query, as its time is longer than the threshold.

▼ Rubric

General requirements: Same as project 1.

Pages	Items	Points
Login Page	Redirect any other pages to the Login Page if the user is not logged in	3
	Show an error message for an incorrect username or password	2
	Use HTTP POST	2
	Login with a correct username and password	4
Main Page	Access to searching and browsing pages/functionalities	2
Searching	Search by title	4
	Search by year	3
	Search by director	4
	Search by star's name	4
	Substring Matching is implemented correctly for searching title/director/star.	3
Movie List Page (Extended from P1)	Display all required information; the first three genres are sorted correctly and hyperlinked; the first three stars are sorted correctly and hyperlinked.	6
	Implemented Pagination. Frontend cache and backend cache both can not exceed 100 records	2
	Use "Prev/Next" buttons.	4
	Change the number of listings "N" per page	2
	Can sort the results on either first "title" then "rating", or first "rating" then "title", both in either ascending or descending order.	6
Single Pages	A Single Movie Page has all genres, stars sorted correctly and hyperlinked.	2

(Extended from P1)	A Single Star Page has all movies sorted correctly and hyperlinked.	2
Jump Functionality	Jump back to the Movie List Page from Single Pages without using the browser history (back button) or changing the url manually in browser address bar	2
	Search/browse, sorting, pagination conditions are maintained when the user jumps back to the Movie List Page	3
Browsing	Browse by movie title's first alphanumeric letter: click on each letter jumps to the movie list page that only consists of movies starting with this letter	4
	Browse by genre: click on each genre jumps to the Movie List Page that only consists of the movies of this genre	4
Shopping Cart Page	Display information about the current shopping cart (including movie title, price, quantity, and total price)	4
	Modify the quantity of each item	4
	Delete an item	3
Payment Page	Ask for basic customer transactional information	5
	Show an error message for an incorrect input	3
Place Order Action	Insert a sale record into "moviedb.sales" table	4
	Show a confirmation page that contains order details including sale ID, movies purchased and the quantities, and total price	4
Additional Performance Functionality	Have a Checkout button for every page after login; if the user clicks it, it will jump to the shopping cart page	2
	Have an "Add to Cart button" for each movie in the Movie List Page and each Single Movie Page; Indicate success/failure for a user click.	3
Extra Credit	Show efforts to beautify the page and table using CSS and JavaScript	0~5
Total credit		100 + 5

▼ Demonstration

In summary, we require 2 things for the demo:

1. A screen recording demo video showing your web application working on AWS.
2. Leave an AWS instance running for a week.

Here are the details:

Screen recording demo video

- **Please make sure your video is readable. Enlarge your font (terminal, browser) and make sure your video is not compressed.**
- The entire demo should be on AWS. **NO** local demo allowed.
- You need a terminal connects to AWS, and a web browser (with Network panel open) open to tomcat manager page on AWS (<http://<AWS public IP>:8080/manager/html>).
- Use screen recording tools (e.g, QuickTime Player for macOS and Open Broadcaster Software for Windows, or other equivalents)
- Please keep the length of the video within 20 minutes. However, do **NOT** edit the video in any way. Otherwise it could be treated as cheating.
- Upload the video to Youtube or other public media hosting website, then include your video URL (publicly accessible) in README.md, submit on GitHub.
- Preparation before the demo:
 - ssh onto AWS instance.
 - on AWS instance, prepare the MySQL database `moviedb`, load data properly.
 - remove your git repo from AWS instance, demo should start with a fresh cloned repo.
 - make sure MySQL and Tomcat are running on AWS instance.
- Demo on AWS instance:
 - **Commit check:**
 1. Clone your git repo to AWS instance: ``git clone <repo url>``. This step is to make sure you have a clean repo.
 2. ``cd`` into your repo and run ``git log``, show your latest commit.
 3. Failure to show commit check will result in an instant 0 of the project.
 - **Show database and data on AWS MySQL:**
 1. show your data counts:

```
mysql -u mytestuser -p -e "use moviedb;select count(*) from stars;select count(*) from movies;"
```
 2. show that you have no sales records made at the time before demo:

```
mysql -u mytestuser -p -e "select * from sales where saleDate = {demo date};"
```

- **Deploy your web application on AWS instance:**

1. inside your repo, where the pom.xml file locates, build the war file:

```
mvn package
```

2. show tomcat web apps, it should NOT have the war file yet:

```
ls -lah /var/lib/tomcat/webapps/
```

3. copy your newly built war file:

```
cp ./target/*.war /var/lib/tomcat/webapps/
```


4. show tomcat web apps, it should now have the new war file:

```
ls -lah /var/lib/tomcat/webapps/
```

- **Show your website in your web browser:**

- make sure you have Network panel opened in your web dev tools, filter with XHR, aside from your web app.
- refresh the tomcat manager page. You should see a new project (just deployed): project 2.
- **Login**
 - click the project link, which goes to your website's landing page (could be the Main Page or Login Page).
 - manually type in browser address bar, with some other urls, for example, **"/shopping-cart"**, or **"/api/checkout"**, pages should be all redirected to Login Page because your are not login yet.
 - enter email/account **"a@email.com"** with a wrong password. click login, an error message should display.
 - enter correct password **"a2"** for the same email/account. click login, it should succeed and redirect to Main Page.
- **Searching and add to cart:**
 - From Main Page, navigate to searching feature.
 - Search by **title "term" and star "tom"**, add a movie (refer as A) to shopping cart.
 - Search by **director "vid m" and year "2007"**, add a movie (refer as B) to shopping cart.
 - (if you do not support substring matching), show searches with exact match instead: the provided queries are substrings.
 - Search by title "The Terminal" and star "Tom Hanks" and year 2004, add this movie (same as A) to shopping cart.
 - Now you should have two copies of "The Terminal" (movie A) in the shopping cart.
- **Browsing and add to cart**
 - From Main Page, navigate to browsing feature
 - Browse by title **"Z"**.
 - Browse by title **"*"**.
 - Browse by genre **"Drama"**,

- click on another genre, go to a new Movie List Page;
- select one movie from the **new list**, go to single Movie Page;
- from Single Movie Page, add the movie (refer as C) to shopping cart.
- **Pagination, sorting, jumping functionality:**
 - Search with title "love", display with order **rating desc, title asc** as tie-breaker, **100** items per page.
 - on page 1, try to go to prev page (should not work, or no button).
 - go to last page, try to go to next page (should not work, or no button).
 - change the Movie List Page to display with order **rating asc, title desc** as tie-breaker, **25** items per page.
 - with order **rating asc, title desc** as tie-breaker, **25** items per page, go to the 3rd page, click the first movie, then first star of that movie, jump back to Movie List Page. it should go back to "love" search result at 3rd page, with order **rating asc, title desc** as tie-breaker, **25** items per page, as before.
- **Shopping Cart:**
 - Navigate to Shopping Cart Page, by clicking any checkout button on any pages after login.
 - Should display 3 movies (4 copies) in total: A, B from search, C from browsing. The movie "The Terminal" (A) should have 2 copies. So quantities now: A-2, B-1, C-1.
 - The following steps, make sure you refresh your shopping cart page after each operation:
 - Change Movie A quantity to 1, refresh shopping cart page, quantities should be A-1, B-1, C-1.
 - Change Movie B quantity to 3, refresh shopping cart page, number should be A-1, B-3, C-1.
 - Delete Movie C, refresh shopping cart page, number should be A-1, B-3.
 - Proceed to payment.
- **Payment Page:**
 - Enter wrong credit card information. Appropriate error messages should display.
 - Enter correct credit card information, place the order.
 - A sale conformation page should show.
 - On terminal, mysql, show new sales records:

```
mysql -u mytestuser -p -e "select * from sales where saleDate = {demo date};"
```
- **Done.**
- Here is an [example](https://youtu.be/1plzMpVZzM0)  <https://youtu.be/1plzMpVZzM0> demo video.

▼ Submission

Same as project 1, but please update README with the project 2 content.

Also, you'll need to include the **substring matching design** in README.