Laboratory Activity #04

Distributed Systems Programming

Daniele Bringhenti, Francesco Pizzato

WebSocket Features

- The **WebSocket** API is an advanced technology that allows to open a two-way **interactive** communication session between the user's browser and a server.
- The main features of WebSockets are:
  1) reliable low-latency general-purpose bidirectional channels;
  2) reuse of the Web infrastructure;
  3) framing and messaging mechanism with no message length limit;
  4) in-band additional closing mechanism.

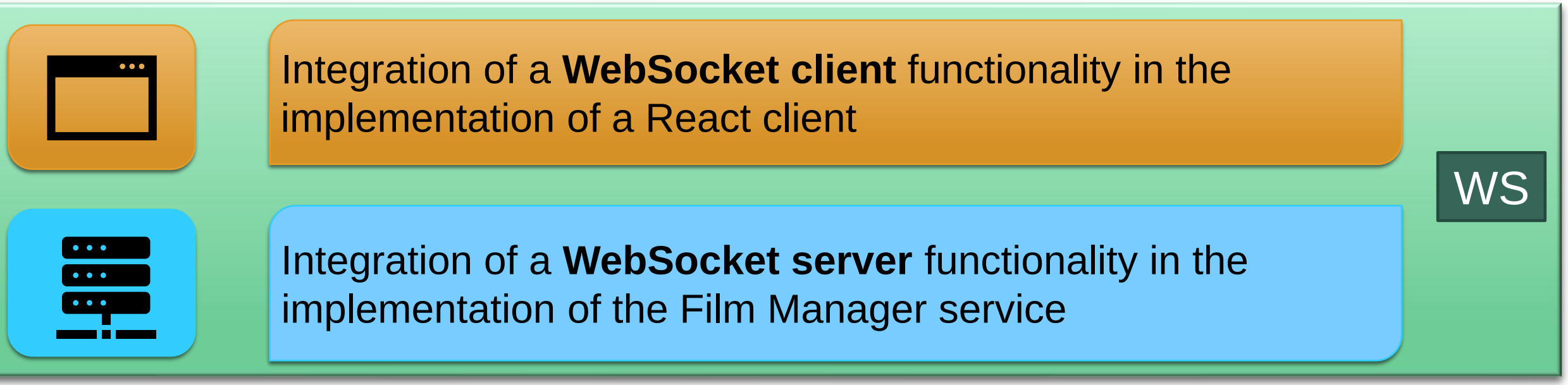
WebSocket Features

- The **WebSocket** API is an advanced technology that allows to open a two-way **interactive** communication session between the user's browser and a server.
- The main features of WebSockets are:
  1) reliable low-latency general-purpose bidirectional channels;
  2) reuse of the Web infrastructure;
  3) framing and messaging mechanism with no message length limit;
  4) in-band additional closing mechanism.

WebSockets are suitable for **continuous, highly interactive** communications.

Topics of the Laboratory Activity

Laboratory Activity #04 covers the following activities:



Topics of the Laboratory Activity

Laboratory Activity #04 covers the following activities:

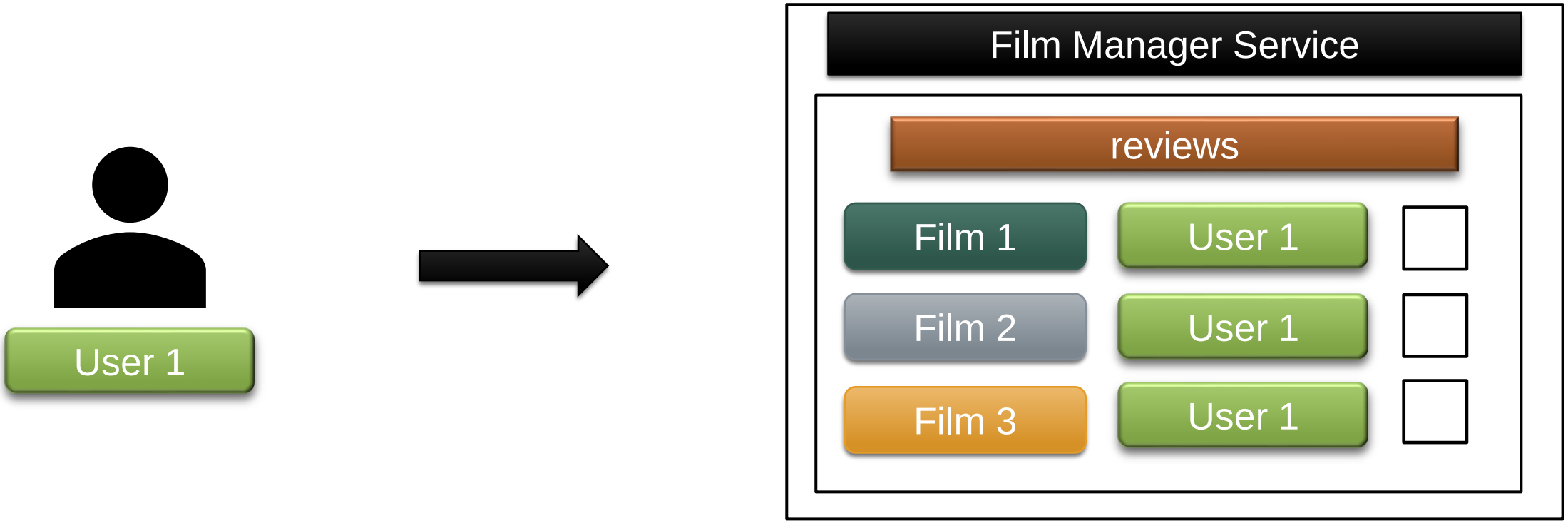
Integration of a **WebSocket client** functionality in the implementation of a React client

Integration of a **WebSocket server** functionality in the implementation of the Film Manager service

WS

Definition of a new **REST API** exposed by the Film Manager service for the management of **film selection**

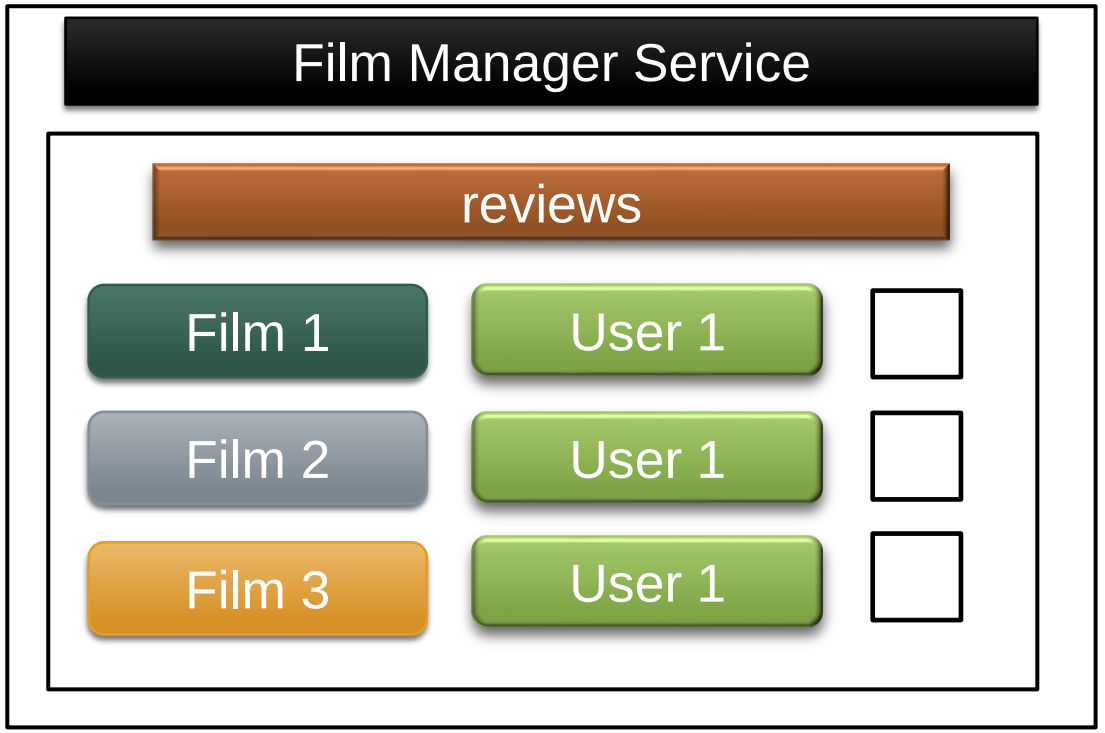
Film Selection in the Film Manager service



- A user can **select** a public film as their active film.
- The active film of a user must be a film for which a review request has been **previously issued** to that user.
- There must exist **at most one** active film for each user.

Film Selection in the Film Manager service

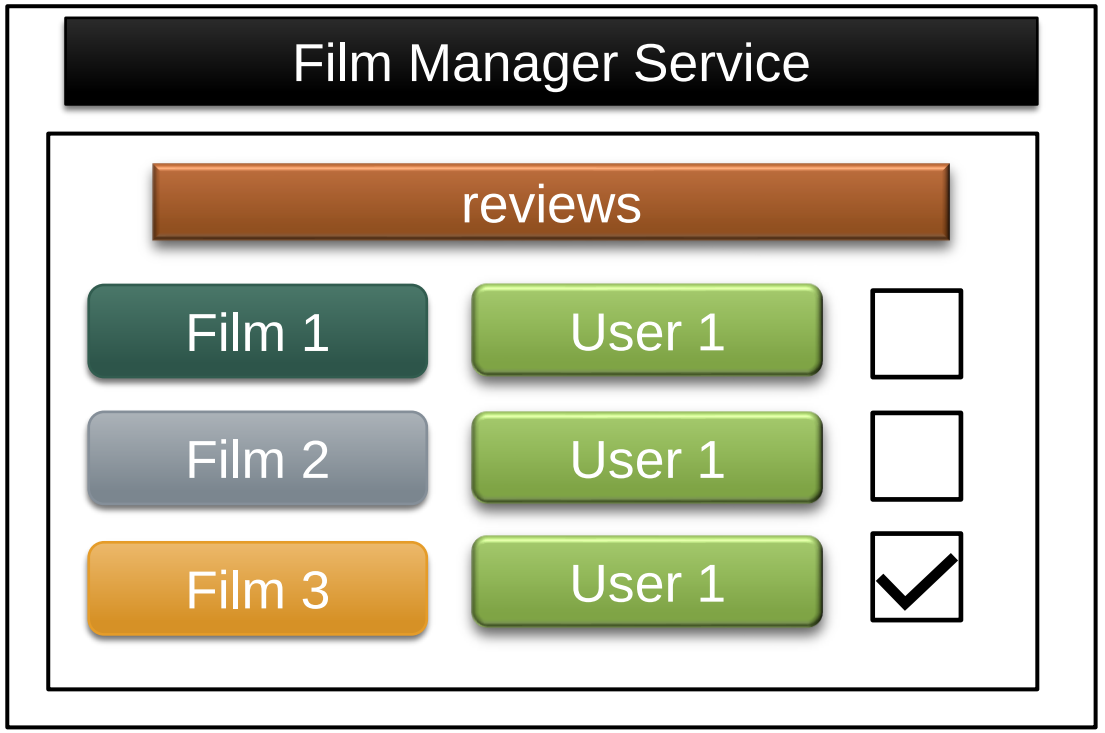
SELECT Film 3



- A user can **select** a public film as their active film.
- The active film of a user must be a film for which a review request has been **previously issued** to that user.
- There must exist **at most one** active film for each user.

Film Selection in the Film Manager service

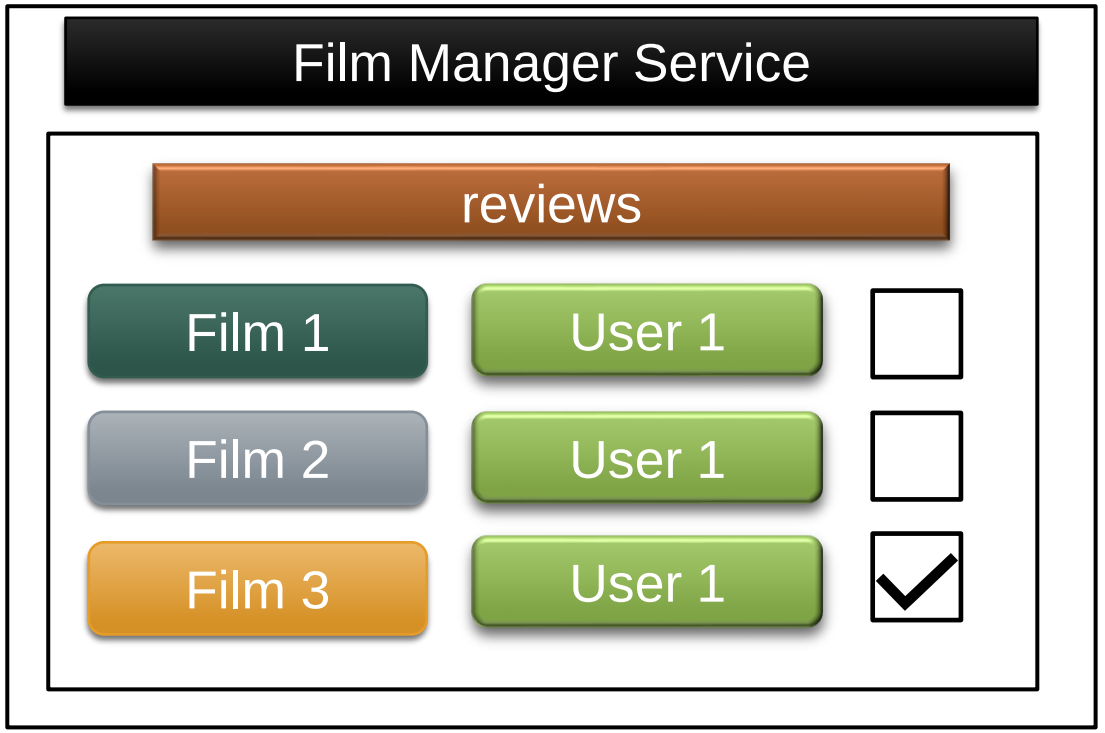
SELECT Film 3



- A user can **select** a public film as their active film.
- The active film of a user must be a film for which a review request has been **previously issued** to that user.
- There must exist **at most one** active film for each user.

Film Selection in the Film Manager service

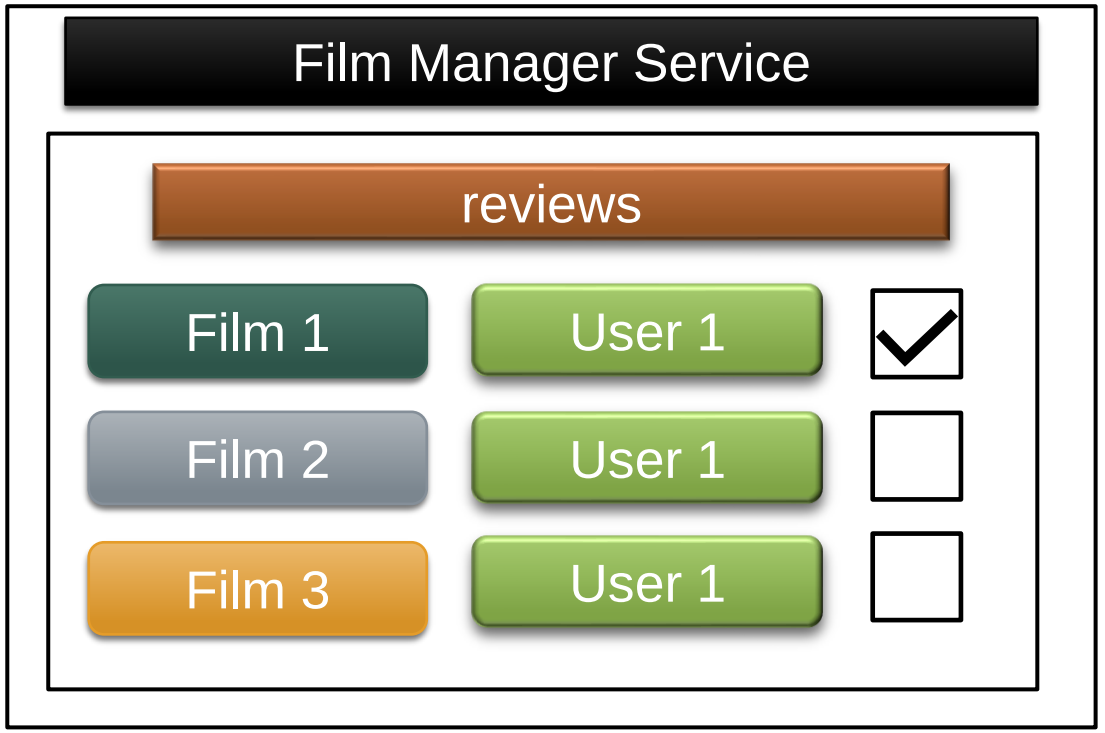
SELECT Film 1



- A user can **select** a public film as their active film.
- The active film of a user must be a film for which a review request has been **previously issued** to that user.
- There must exist **at most one** active film for each user.

Film Selection in the Film Manager service

SELECT Film 1



- A user can **select** a public film as their active film.
- The active film of a user must be a film for which a review request has been **previously issued** to that user.
- There must exist **at most one** active film for each user.

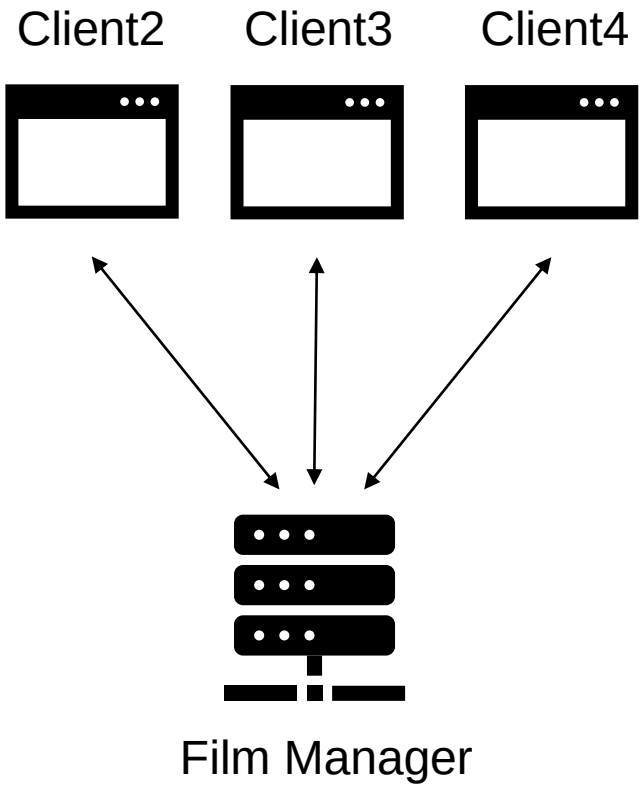
Status (logged-in users and active films)

- Both the Film Manager service and the React client are **extended** with the functionality to communicate by using WebSockets channels:
  - **Server**: Film Manager;
  - **Client**: an instance of the React client.

- These channels are used by the server to inform all the clients about:
  1) the current status of the **logged-in users**;
  2) the status of their **active films**.

How is the **WebSocket** communication organized?

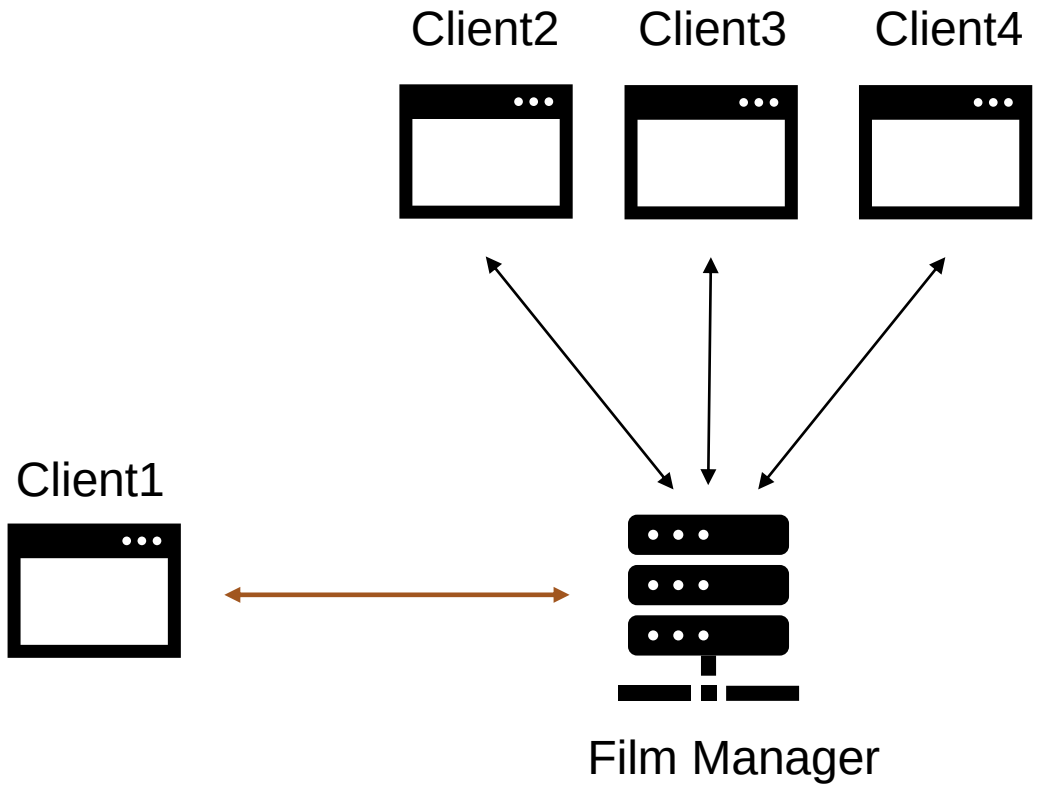
WebSocket communication (initial situation)



Status

userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7

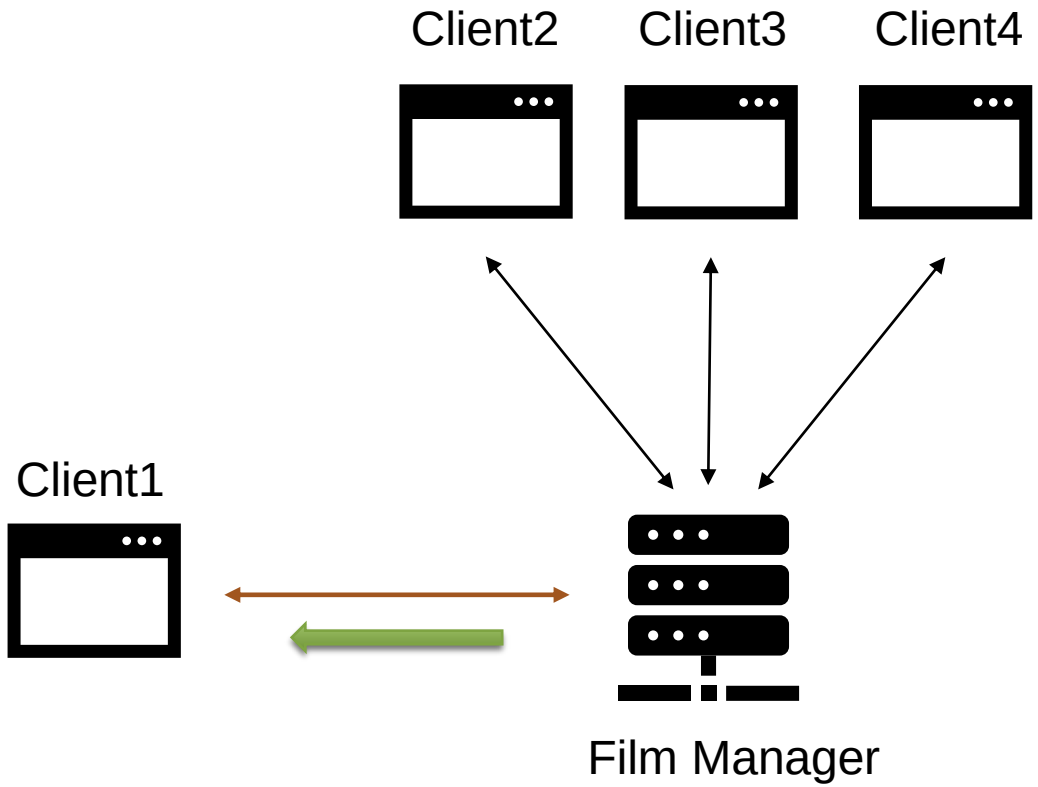
WebSocket communication (connection establishment)



Status

userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7

WebSocket communication (connection establishment)

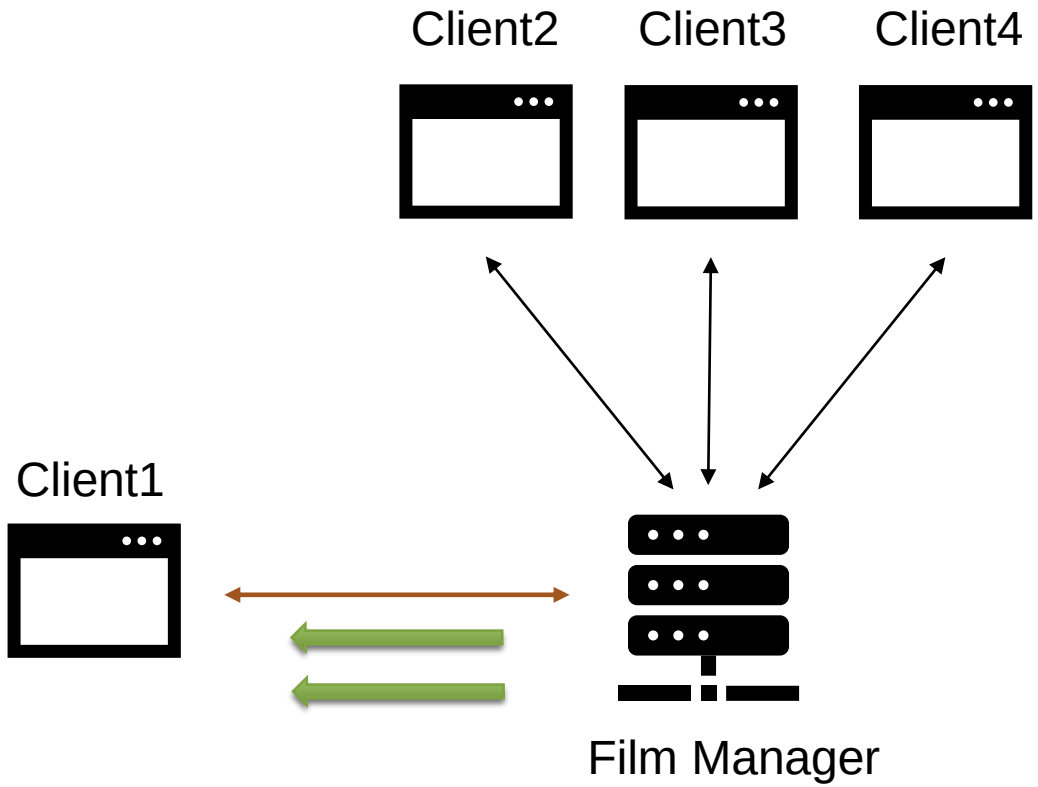


Status

userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7

```
{
    "typeMessage" :"login",
    "userId: "2",
    "userName": "Frank",
    "filmId": "5",
    "filmTitle": "Title5"
}
```

WebSocket communication (connection establishment)

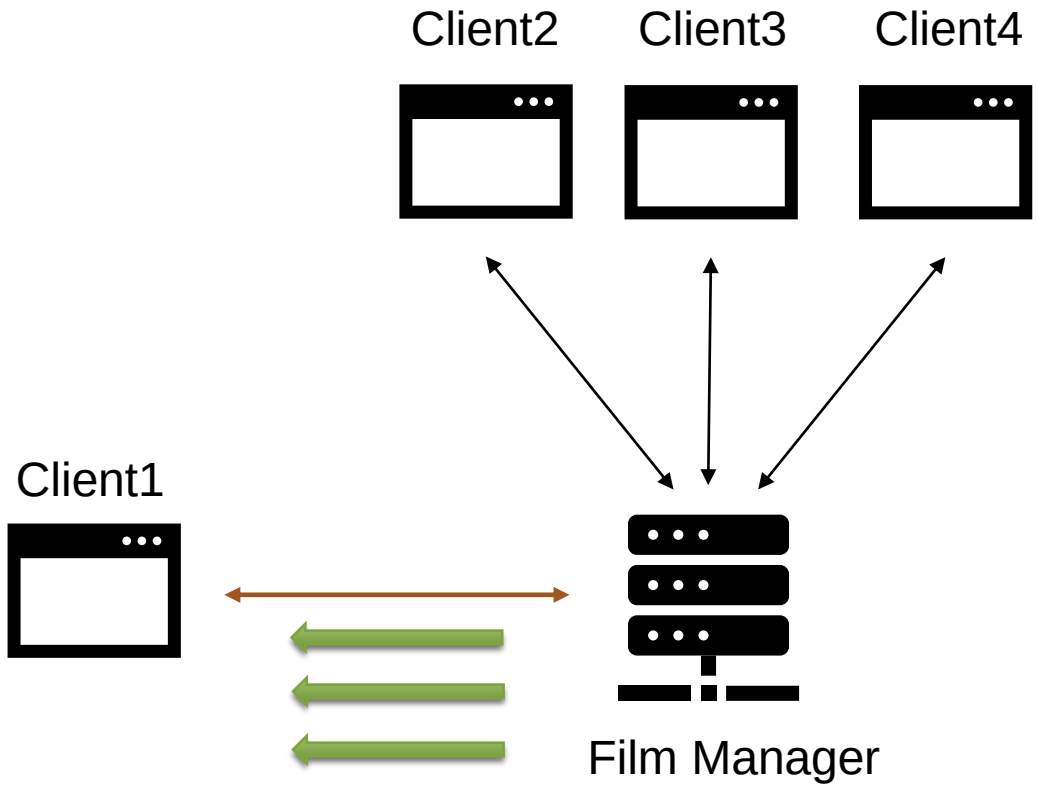


Status

userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7

```
{
    "typeMessage" :"login",
    "userId: "3",
    "userName": "Karen"
}
```

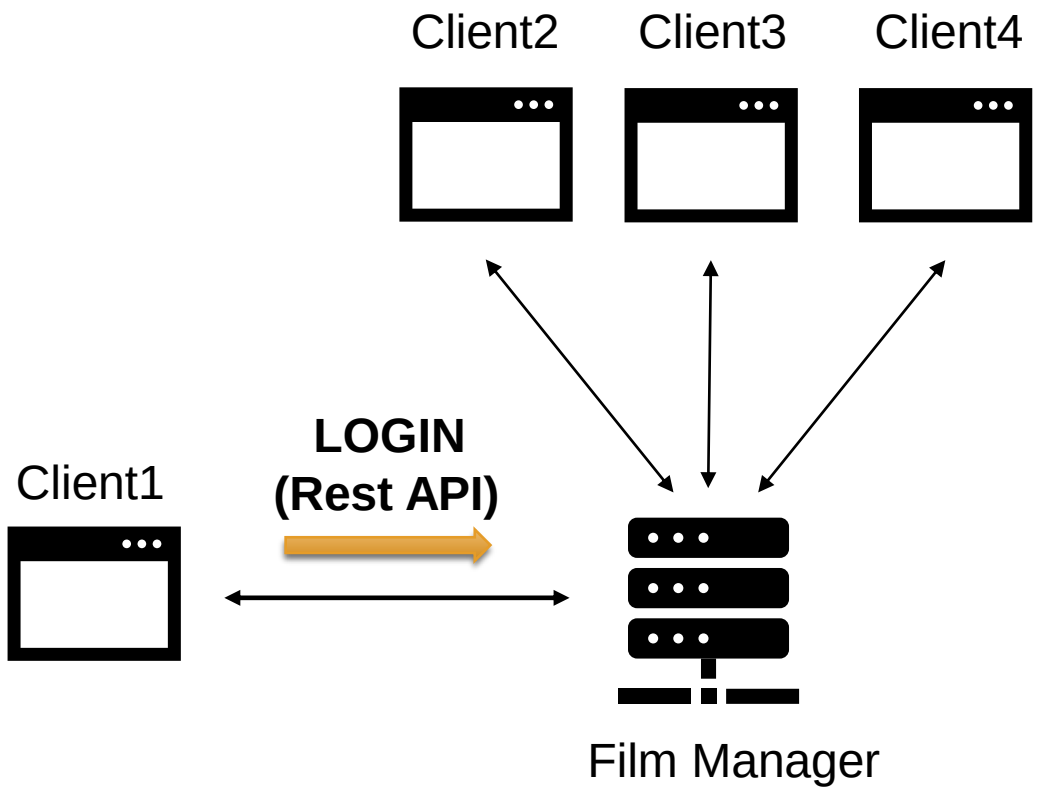
WebSocket communication (connection establishment)



Status			
userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7

```
{
    "typeMessage" :"login",
    "userId: "4",
    "userName": "Rene",
    "filmId": "7",
    "filmTitle": "Title7"
}
```

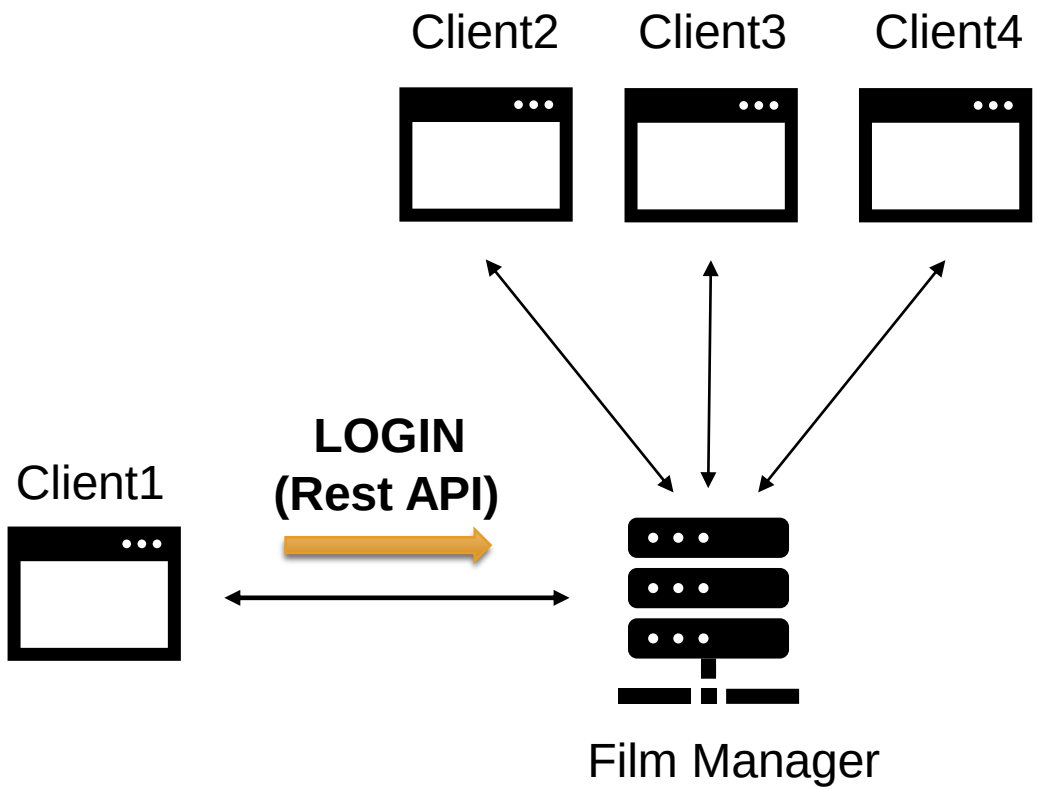
WebSocket communication (login)



Status

userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7

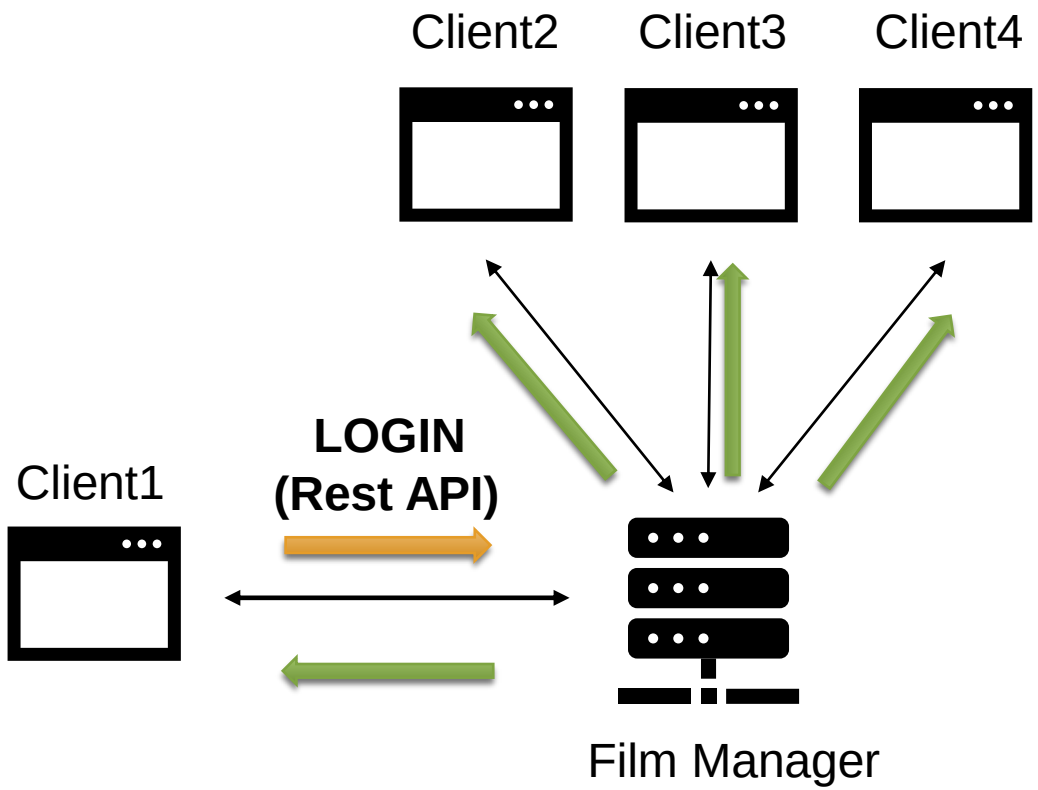
WebSocket communication (login)



Status

userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7
5	Beatrice	-	-

WebSocket communication (login)

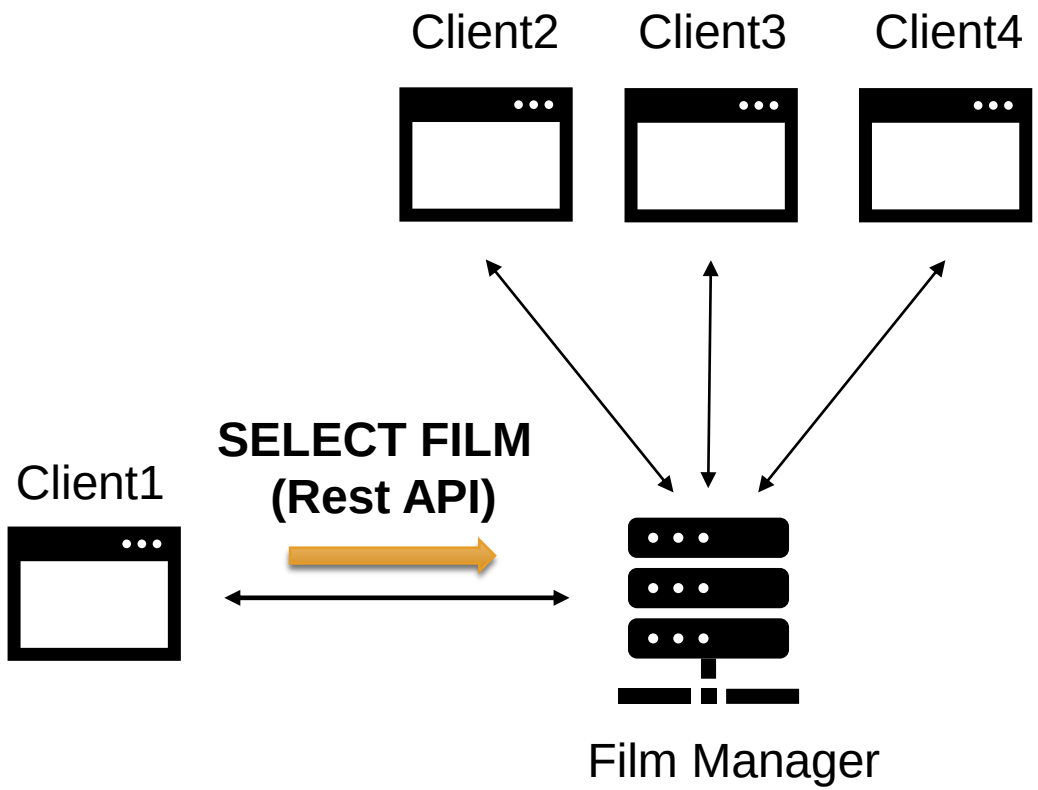


Status

userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7
5	Beatrice	-	-

```
{
    "typeMessage" :"login",
    "userId: "5",
    "userName": "Beatrice"
}
```

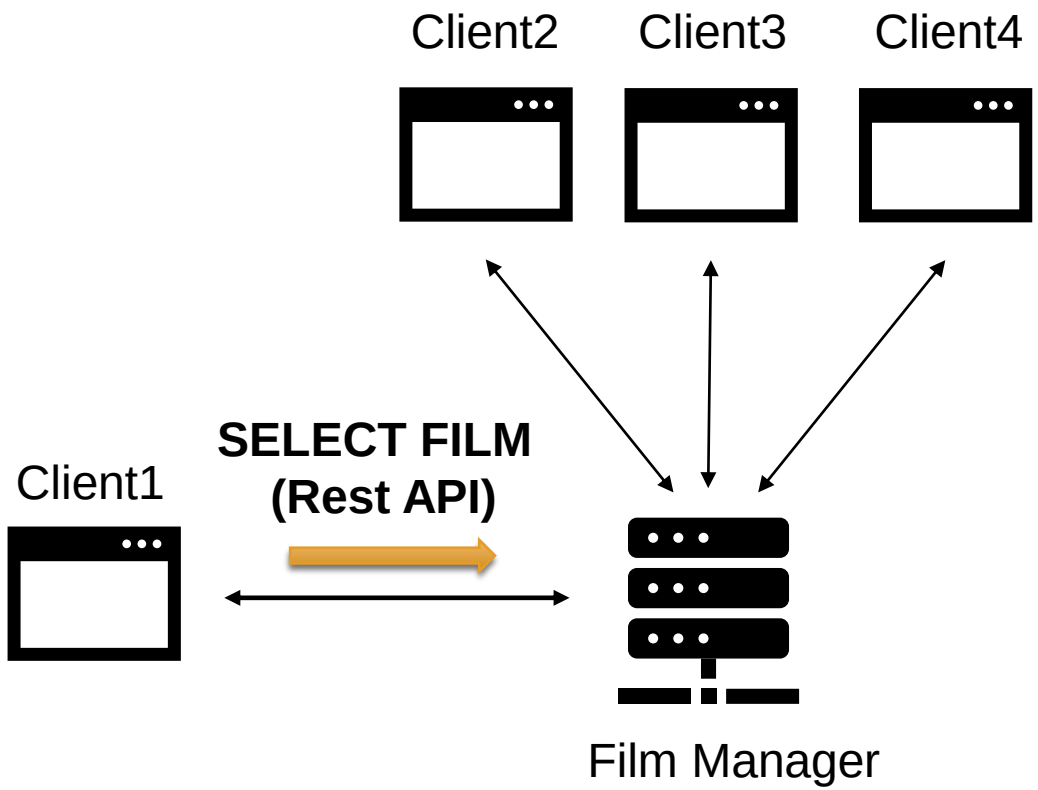
WebSocket communication (film selection)



Status

userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7
5	Beatrice	-	-

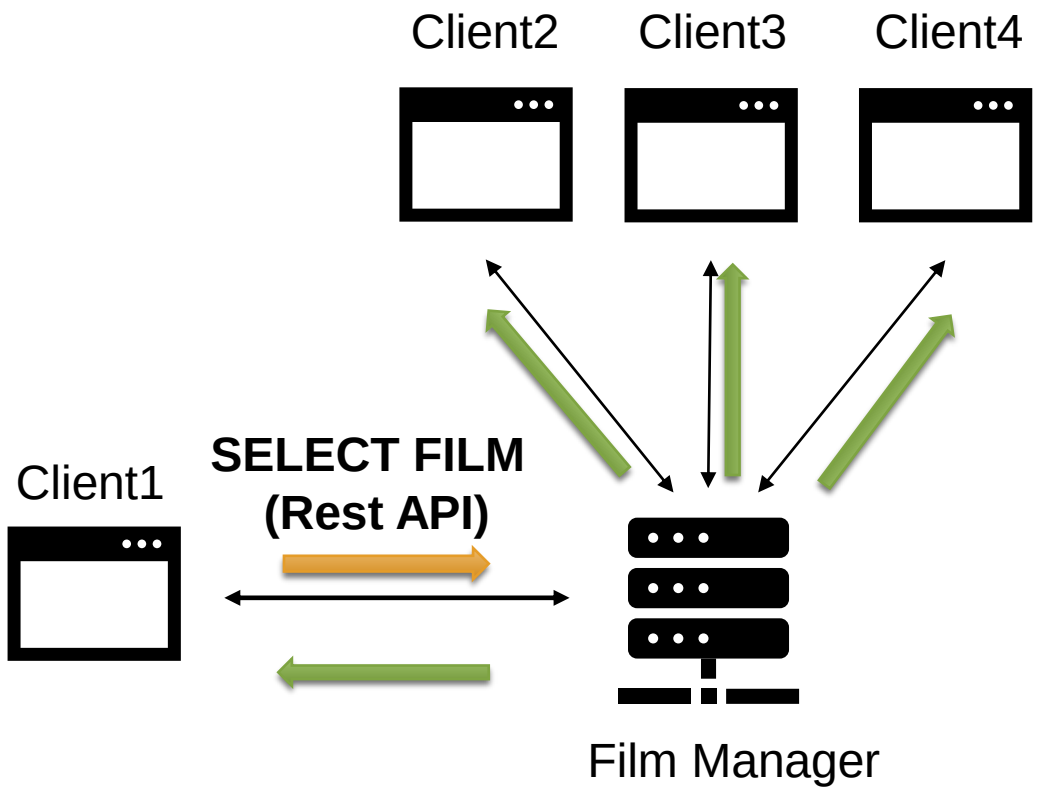
WebSocket communication (film selection)



Status

userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7
5	Beatrice	9	Title9

WebSocket communication (film selection)

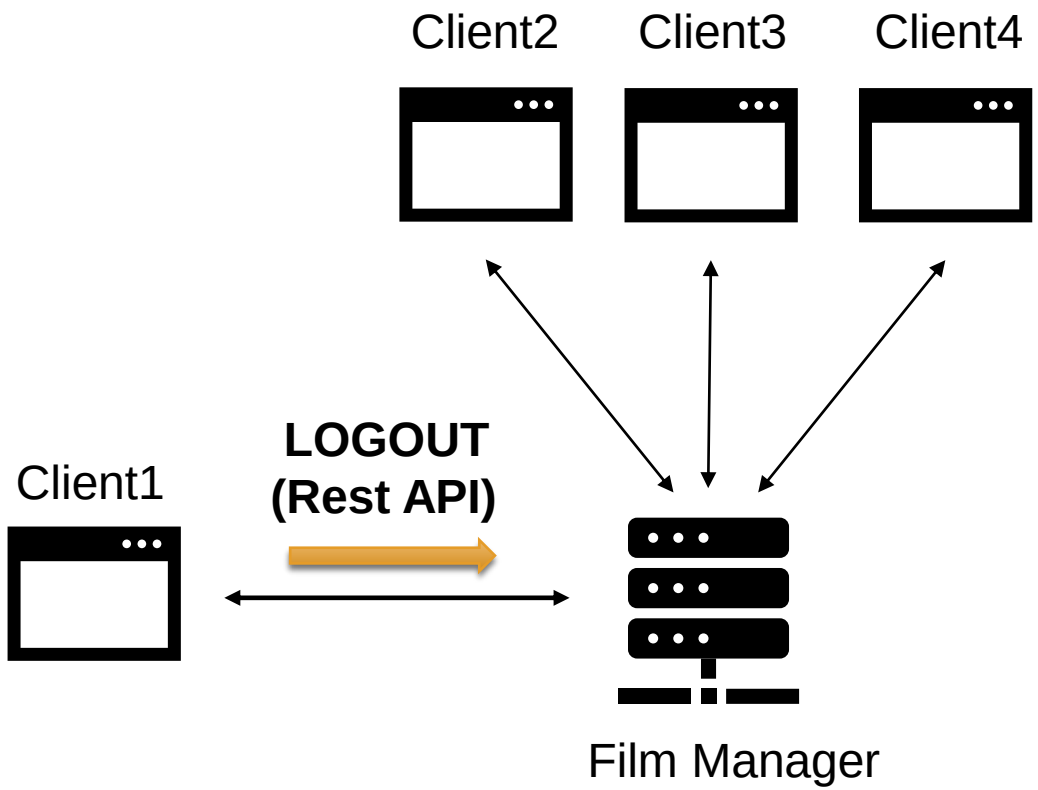


Status

userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7
5	Beatrice	9	Title9

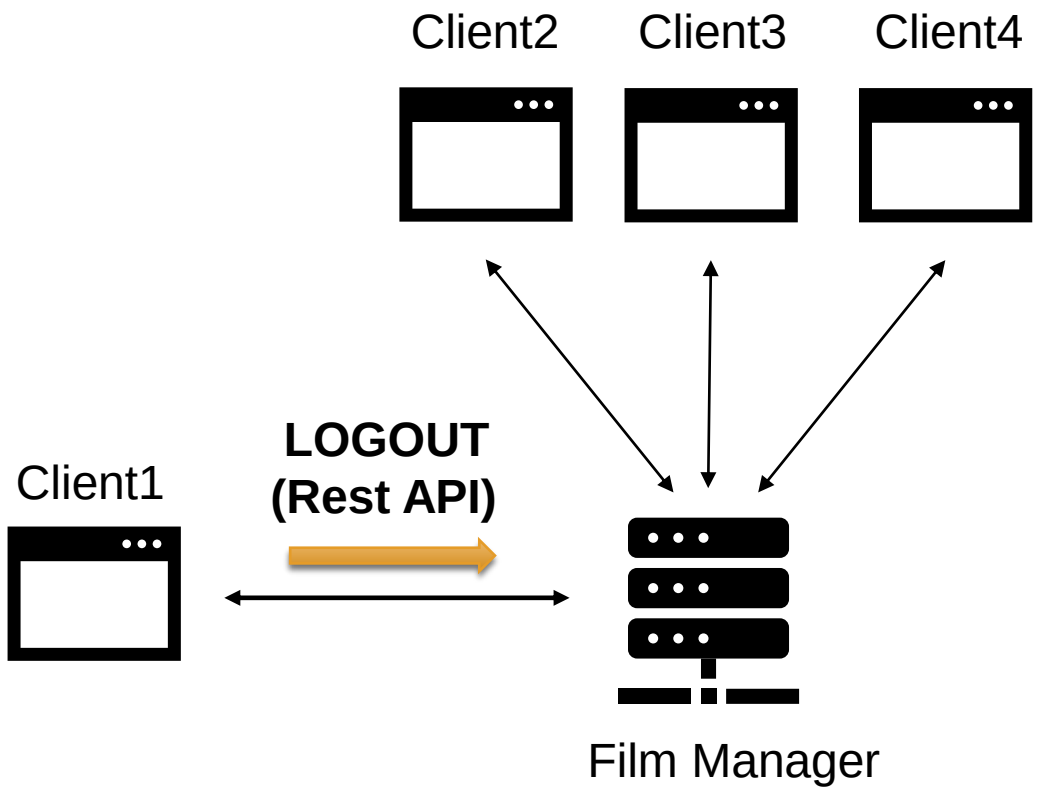
```
{
    "typeMessage" :"update",
    "userId: "5",
    "userName": "Beatrice",
    "filmId": "9",
    "filmTitle": "Title9"
}
```

WebSocket communication (logout)



Status			
userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7
5	Beatrice	9	Title9

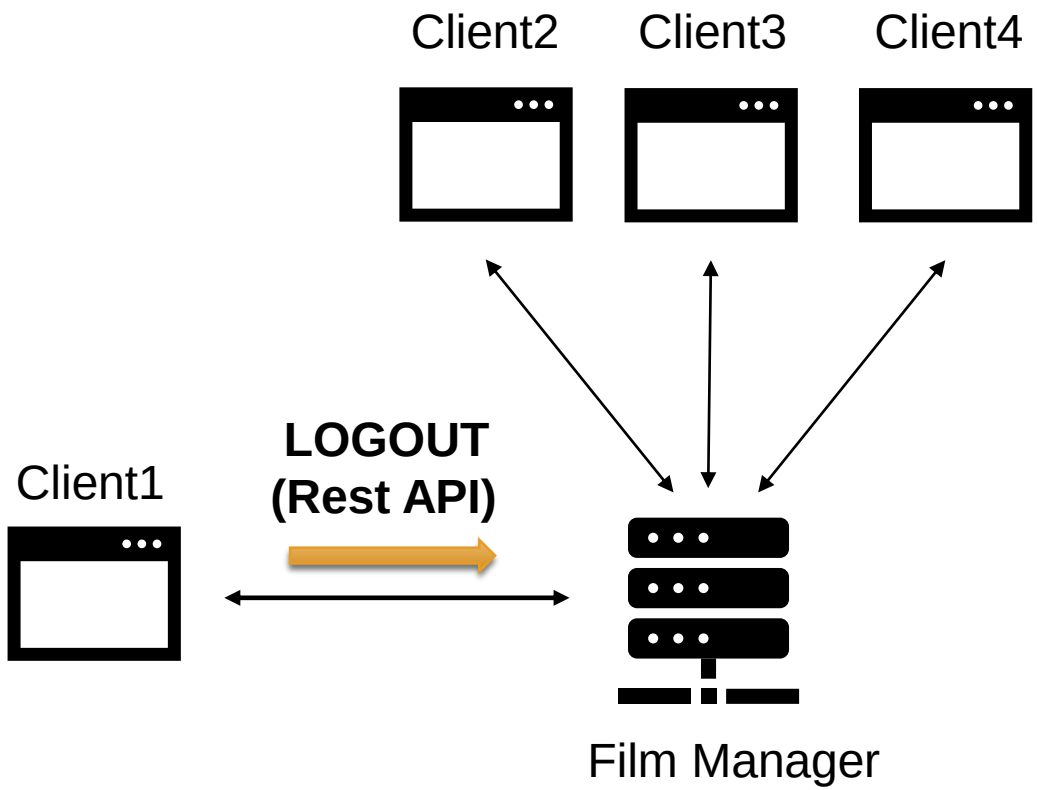
WebSocket communication (logout)



Status			
userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7
5	Beatrice	9	Title9

Remember that there is NOT an **explicit** logout operation in the *Film Manager* service, the server needs to keep the session time locally and eventually perform the needed operation (i.e., notify the other clients)

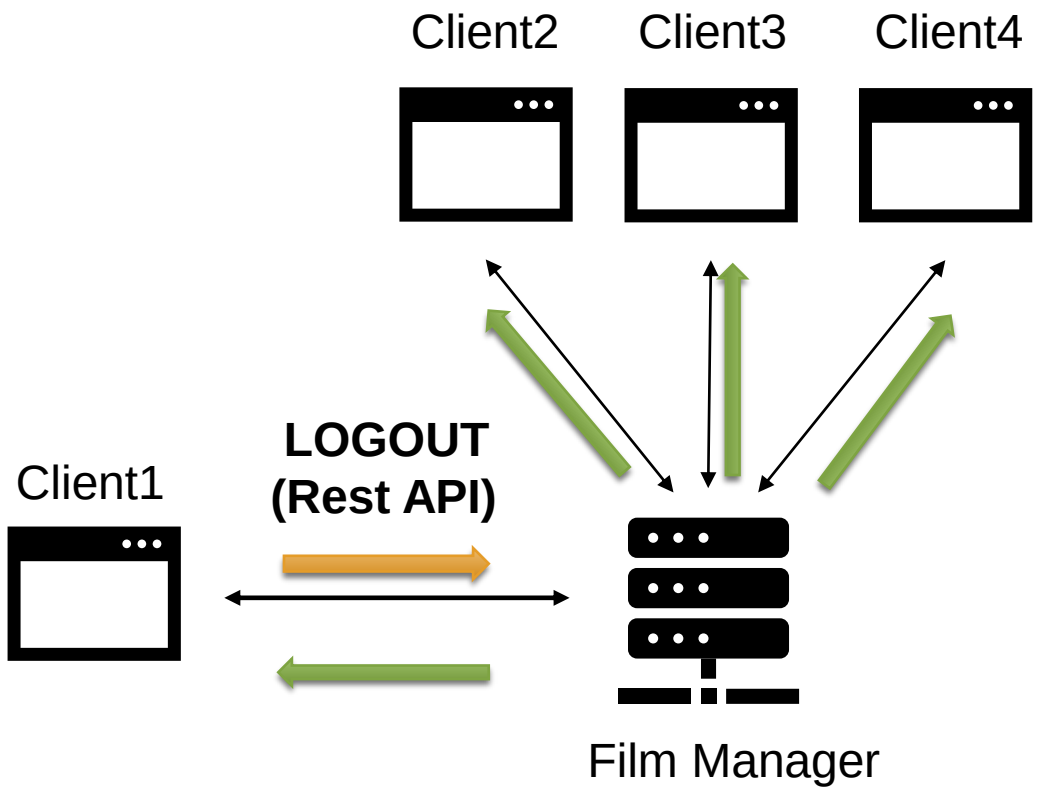
WebSocket communication (logout)



Status

userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7

WebSocket communication (logout)



Status			
userId	userName	filmId	filmTitle
2	Frank	5	Title5
3	Karen	-	-
4	Rene	7	Title7

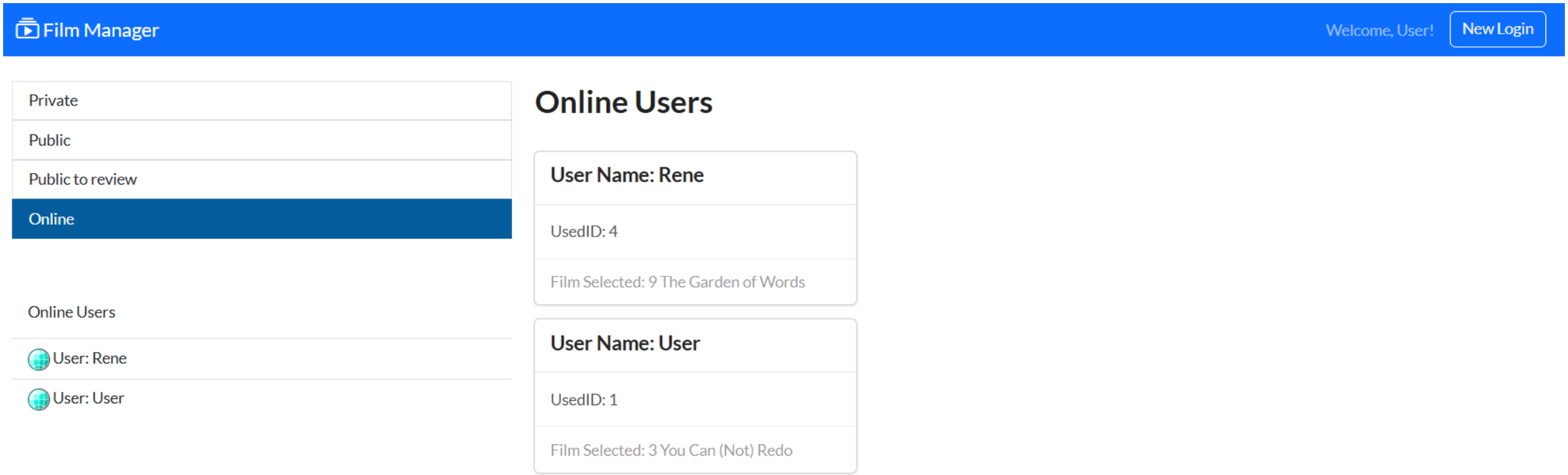
```
{
      "typeMessage" :"logout",
      "userId: "5"
}
```

How does the React client *react*?

- The **Online** page displays:
  - the list of users that are **currently logged-in** the Film Manager service;
  - for each listed user, the user name and the user id;
  - for each listed user, the film title and film id of the active film (if any).
- **All the pages** of the GUI display:
  - a list of the logged-in users in the left column;
  - for each entry of this list, the user name.

➢ Both the content of the Online page and the left column are **updated** as soon as the client **receives** a message in the WebSocket communication.

How does the React client *react*?



How should you make the React client *react*?

- In this activity, you must mainly focus on the WebSocket communication:
  - generation and sending of the messages server side;
  - receiving the messages client-side (in **App.jsx** file, **Main** function).

- For the reaction of the *React* client, you only need to:
  - update the ***onlineList*** array, with the latest message related to each logged-in user.

Thanks for your attention!

D. Bringhenti, F. Pizzato

daniele.bringhenti@polito.it

francesco.pizzato@polito.it

netgroup