

# Laboratory Session #03

Distributed Systems Programming

**Daniele Bringhenti**



- 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.

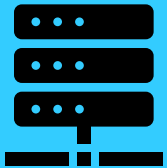
- The **WebSocket** API is an advanced technology that allows to open a two-way **interactive** communication session between the user's browser and a server.
- The main features of WebSockets are:
  - 1) reliable low-latency general-purpose bidirectional channels;
  - 2) reuse of the Web infrastructure;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

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

Laboratory Session #03 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 ToDoManager service

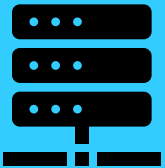
WS

Laboratory Session #03 covers the following activities:



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

WS

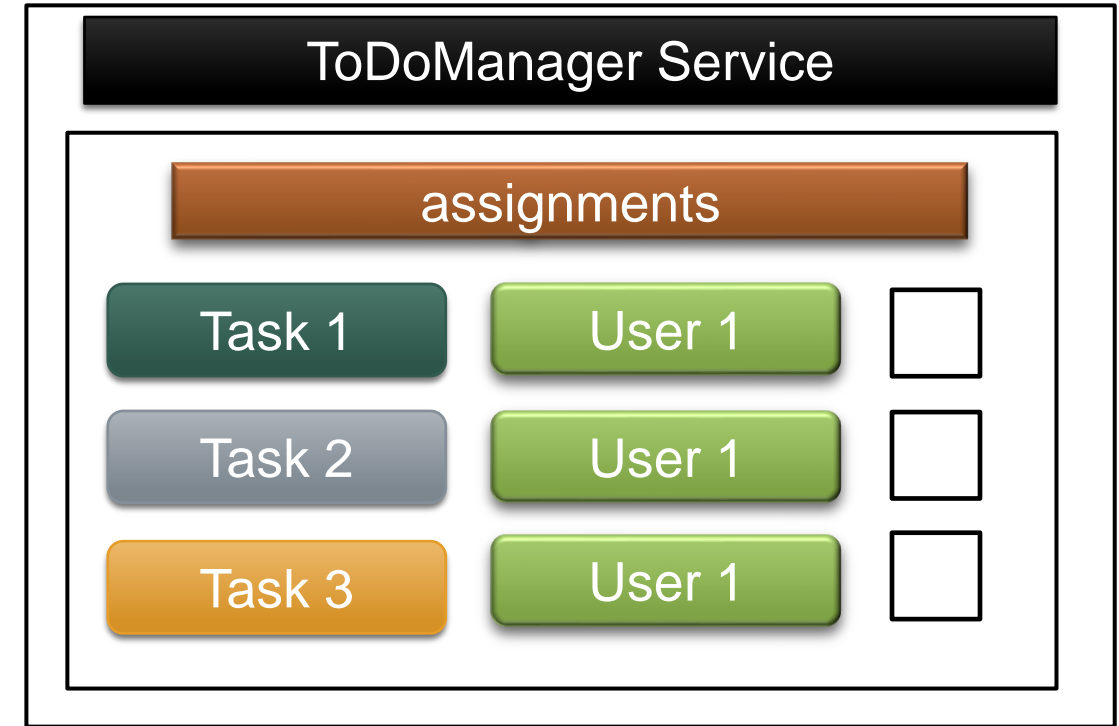


Integration of a **WebSocket server** functionality in the implementation of the ToDoManager service

{ REST }

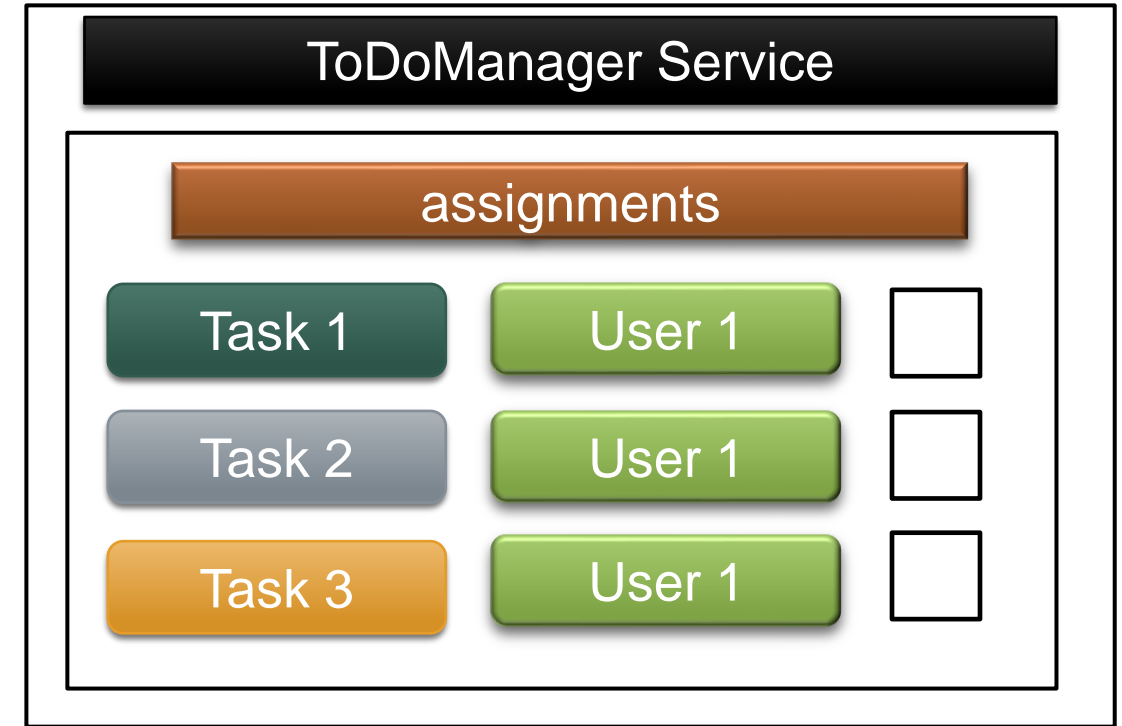
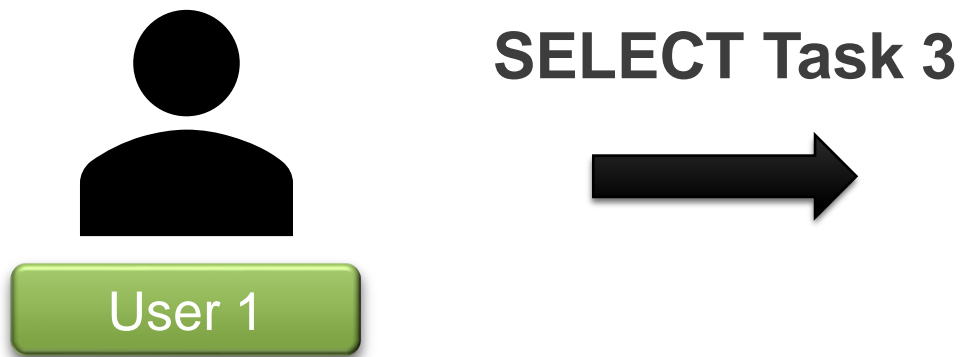
Definition of new **REST APIs** exposed by the ToDoManager service for the management of **task selection**

# Task Selection in the ToDoManager service



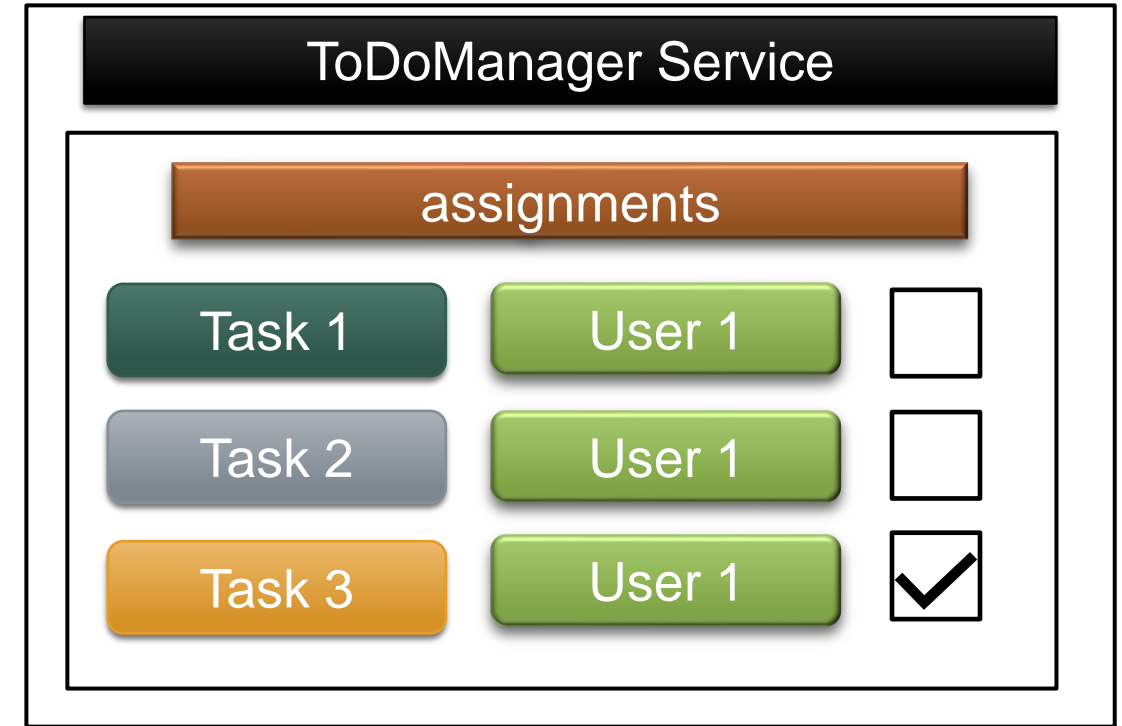
- A user can **select** a task as their active task.
- The active task of a user must be a task **previously assigned** to that user.
- There must exist **at most one** active task for each user.

# Task Selection in the ToDoManager service



- A user can **select** a task as their active task.
- The active task of a user must be a task **previously assigned** to that user.
- There must exist **at most one** active task for each user.

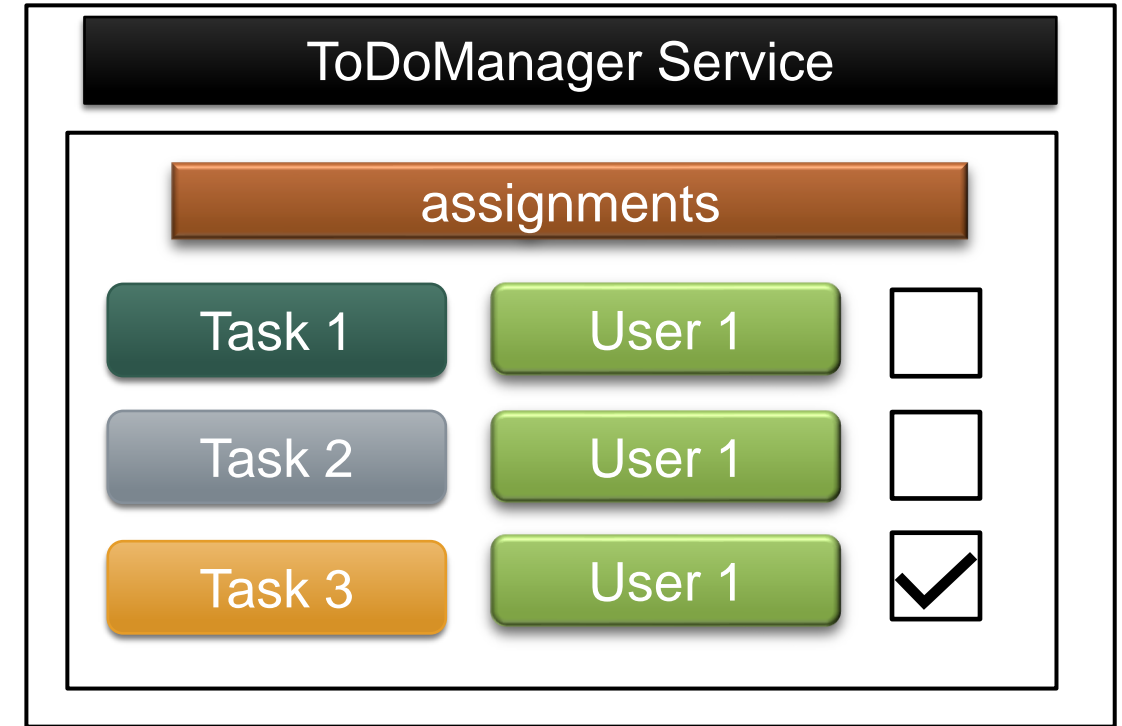
# Task Selection in the ToDoManager service



- A user can **select** a task as their active task.
- The active task of a user must be a task **previously assigned** to that user.
- There must exist **at most one** active task for each user.

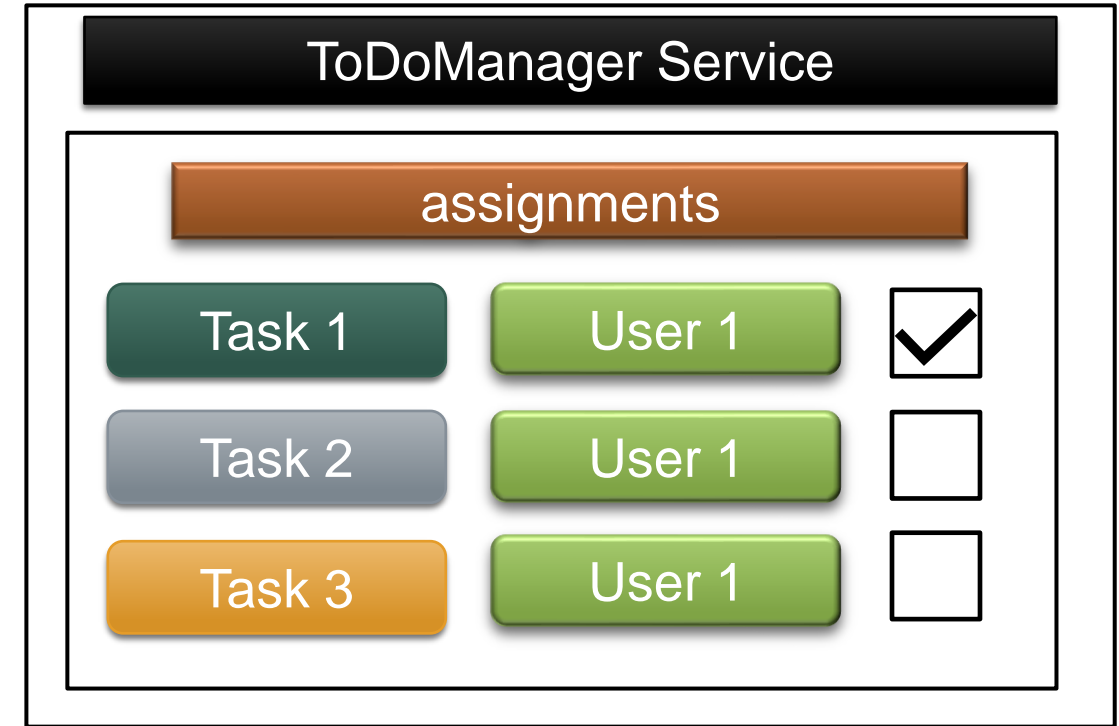
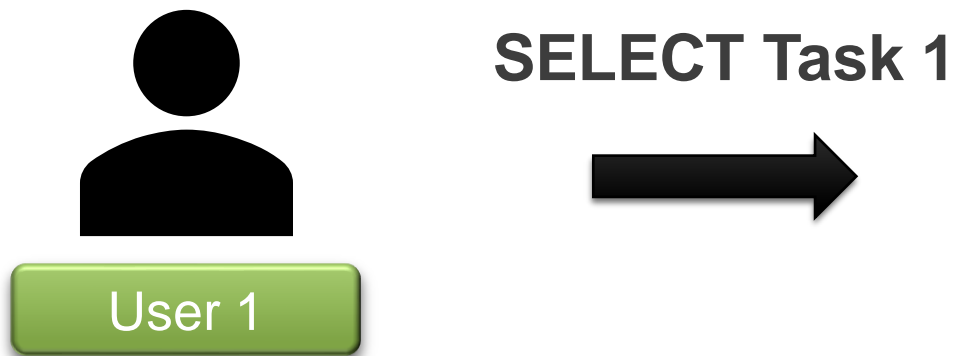


# Task Selection in the ToDoManager service



- A user can **select** a task as their active task.
- The active task of a user must be a task **previously assigned** to that user.
- There must exist **at most one** active task for each user.

# Task Selection in the ToDoManager service

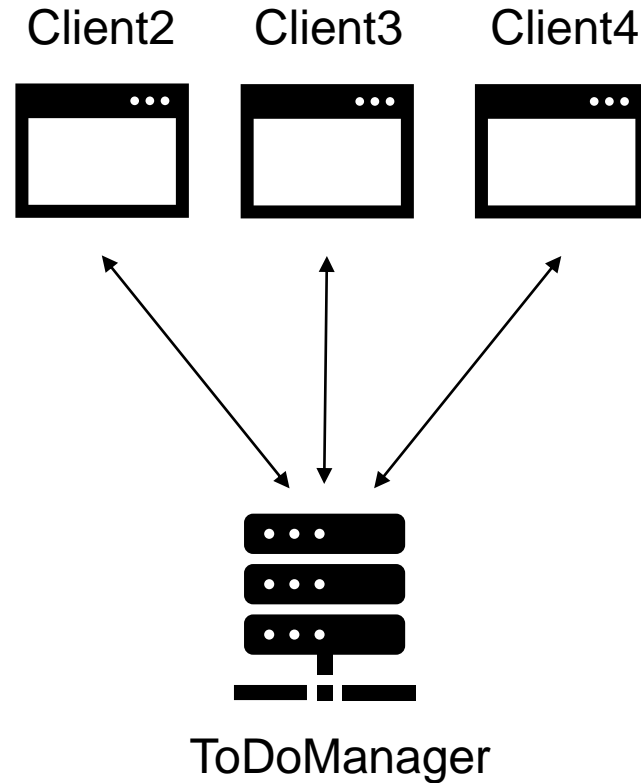


- A user can **select** a task as their active task.
- The active task of a user must be a task **previously assigned** to that user.
- There must exist **at most one** active task for each user.

- Both the ToDoManager service and the React client are **extended** with the functionality to communicate by using WebSockets channels:
  - **Server:** ToDoManager;
  - **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 tasks**.

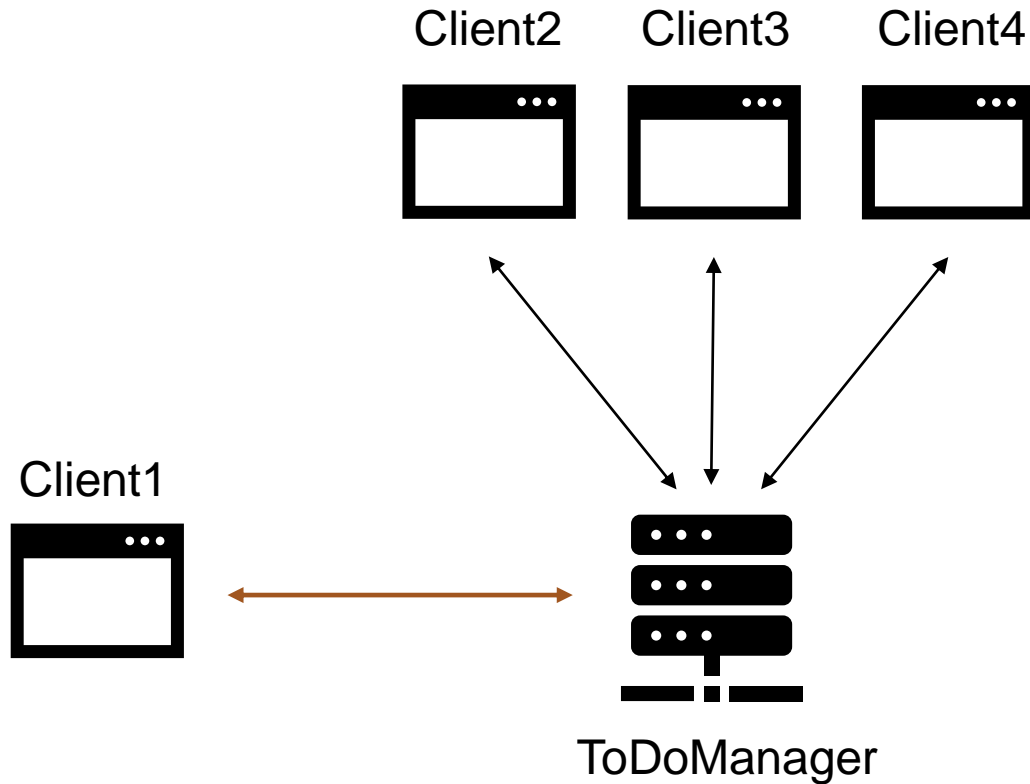
How is the **WebSocket** communication organized?

# WebSocket communication (initial situation)



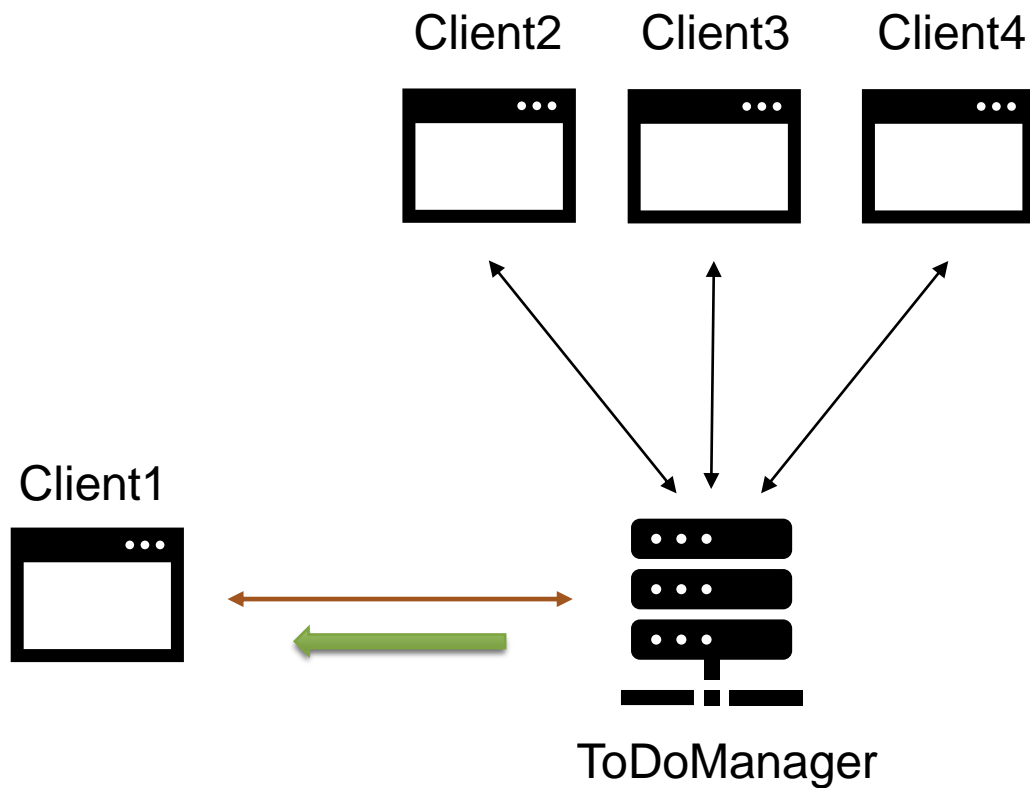
Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film

# WebSocket communication (connection establishment)



Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film

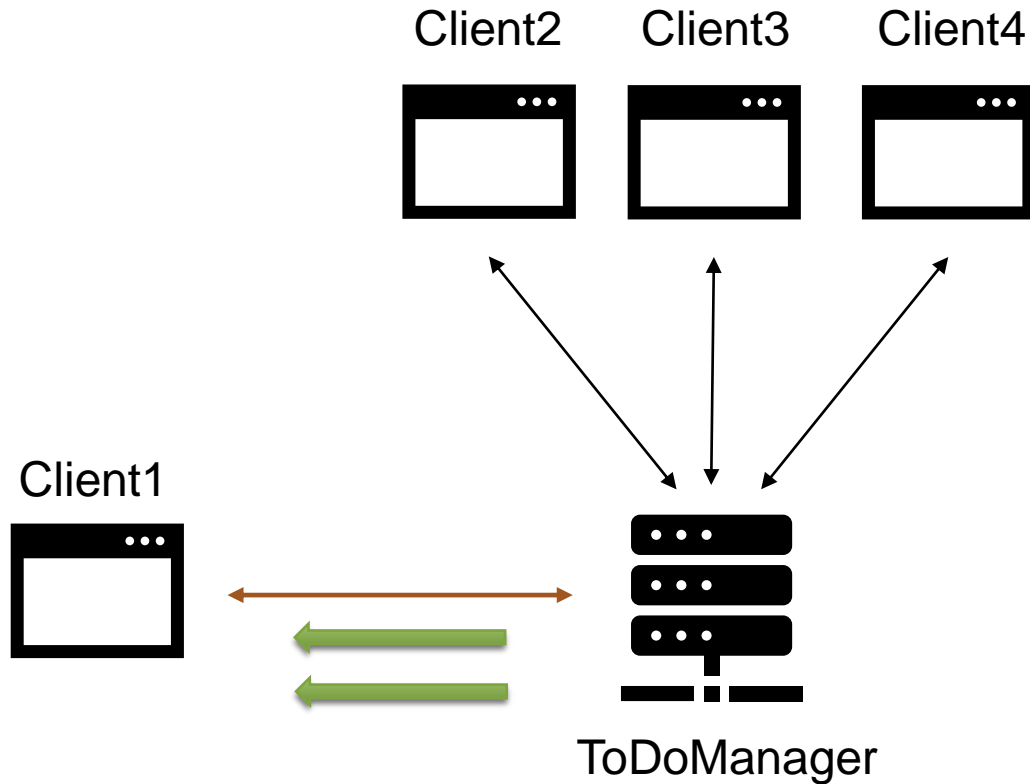
# WebSocket communication (connection establishment)



Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film

```
{  
  "typeMessage": "login",  
  "userId": "2",  
  "userName": "Frank",  
  "taskId": "5",  
  "taskName": "Study JSON"  
}
```

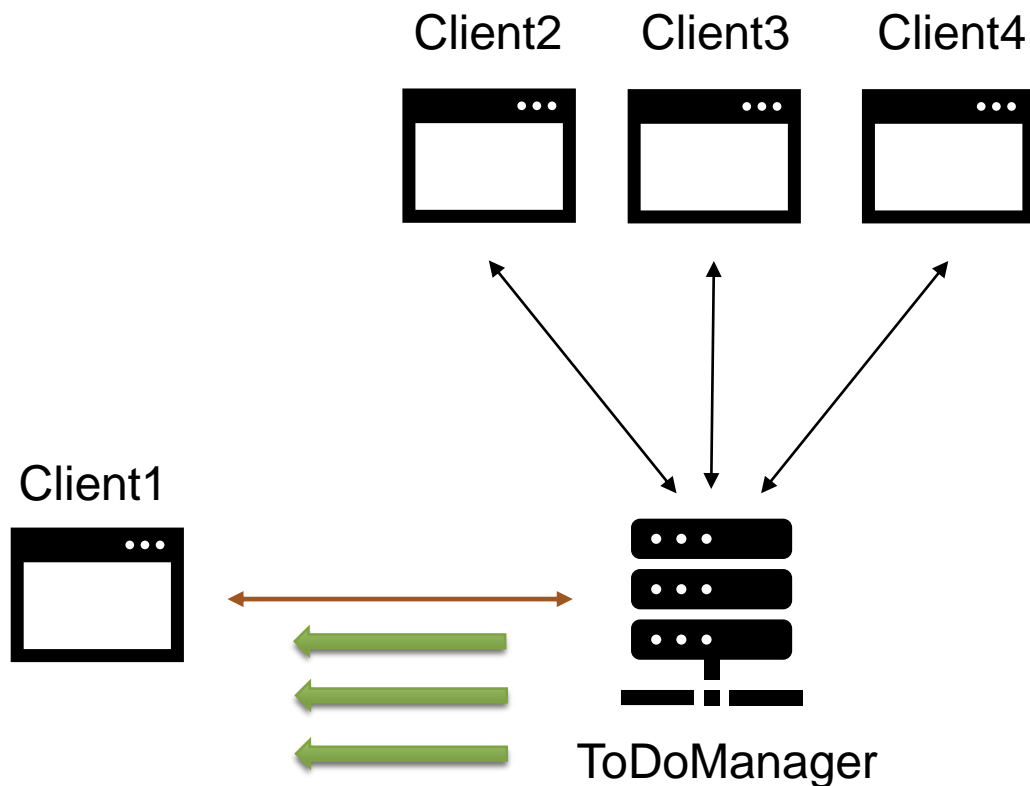
# WebSocket communication (connection establishment)



Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film

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

# WebSocket communication (connection establishment)

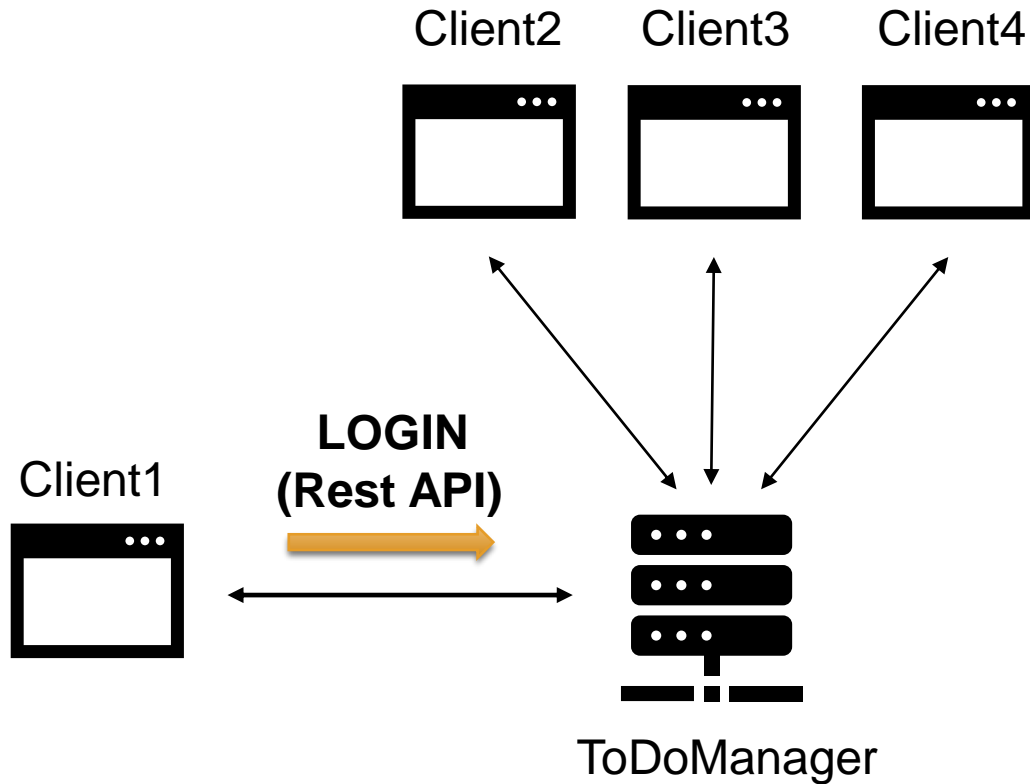


Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film

```
{  
  "typeMessage": "login",  
  "userId": "4",  
  "userName": "Rene",  
  "taskId": "7",  
  "taskName": "Watch film"  
}
```

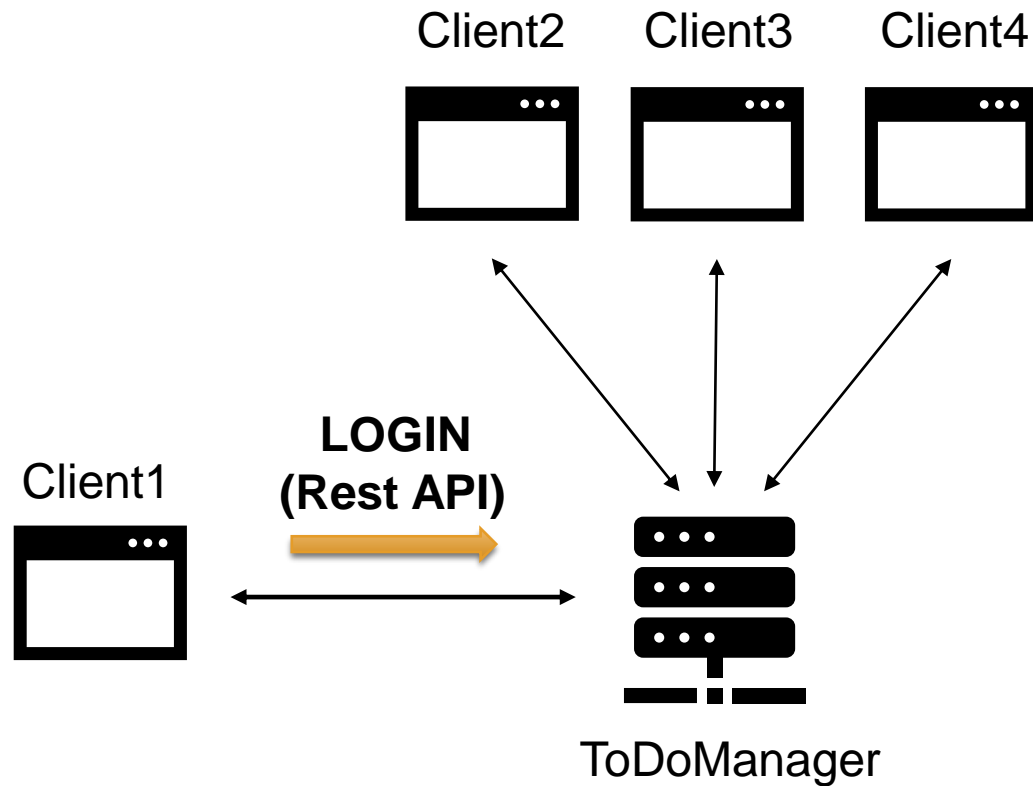


# WebSocket communication (login)



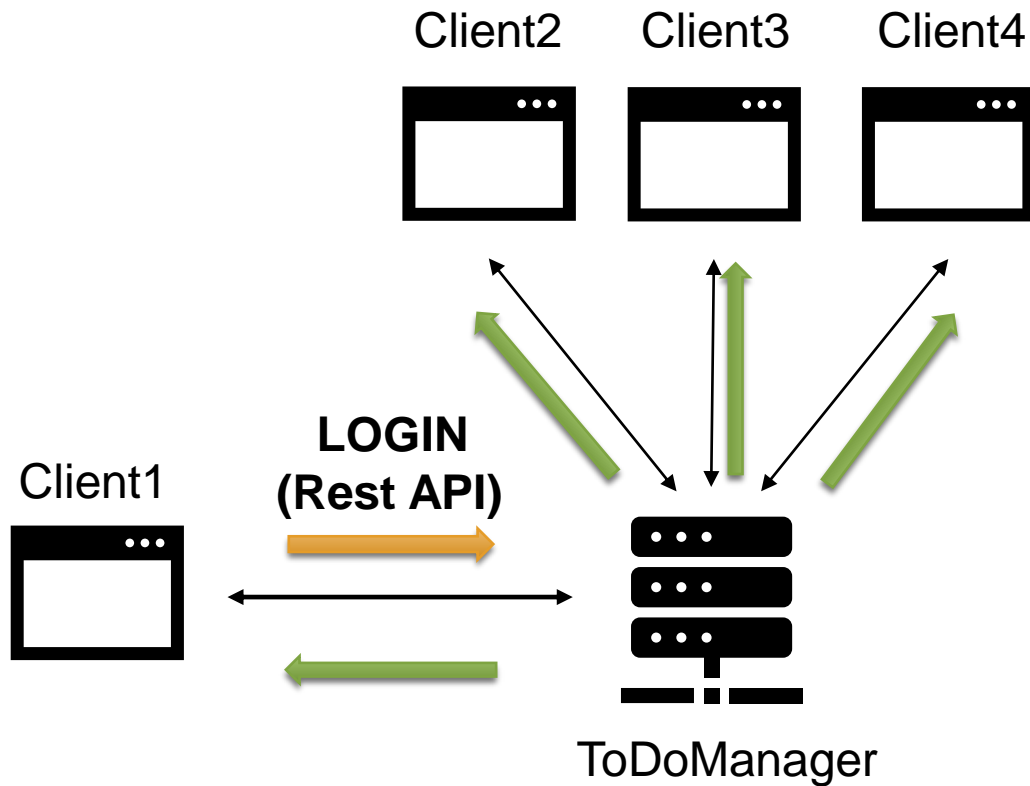
Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film

# WebSocket communication (login)



Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film
5	Beatrice	-	-

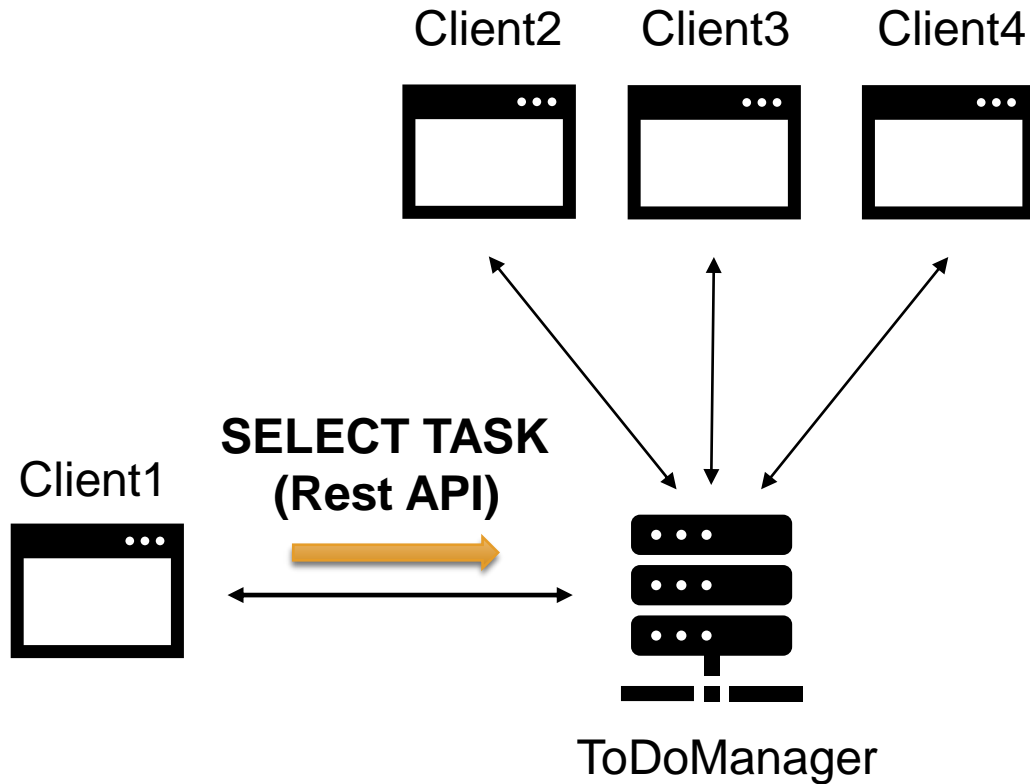
# WebSocket communication (login)



Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film
5	Beatrice	-	-

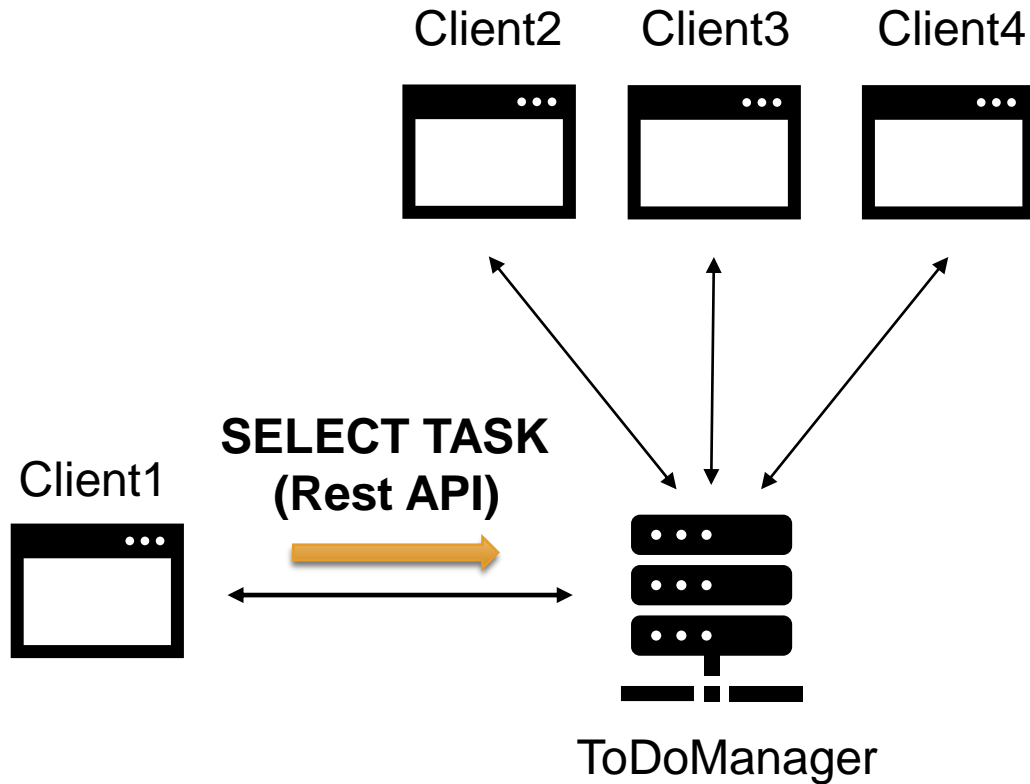
```
{  
  "typeMessage": "login",  
  "userId": "5",  
  "userName": "Beatrice"  
}
```

# WebSocket communication (task selection)



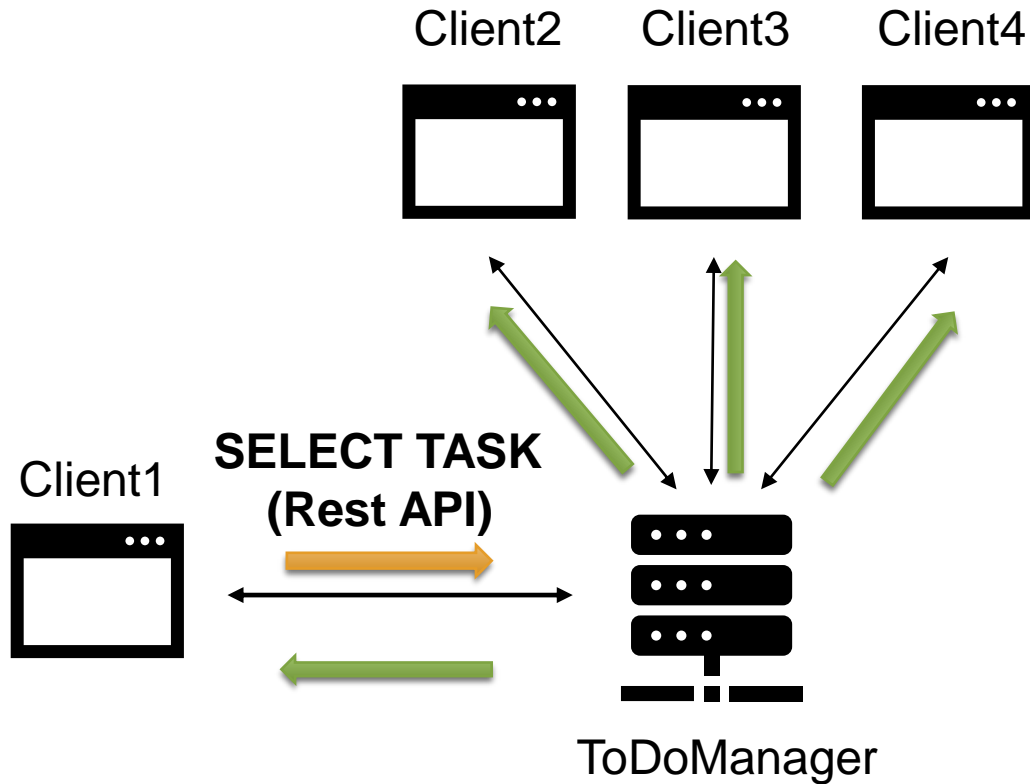
Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film
5	Beatrice	-	-

# WebSocket communication (task selection)



Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film
5	Beatrice	9	Read book

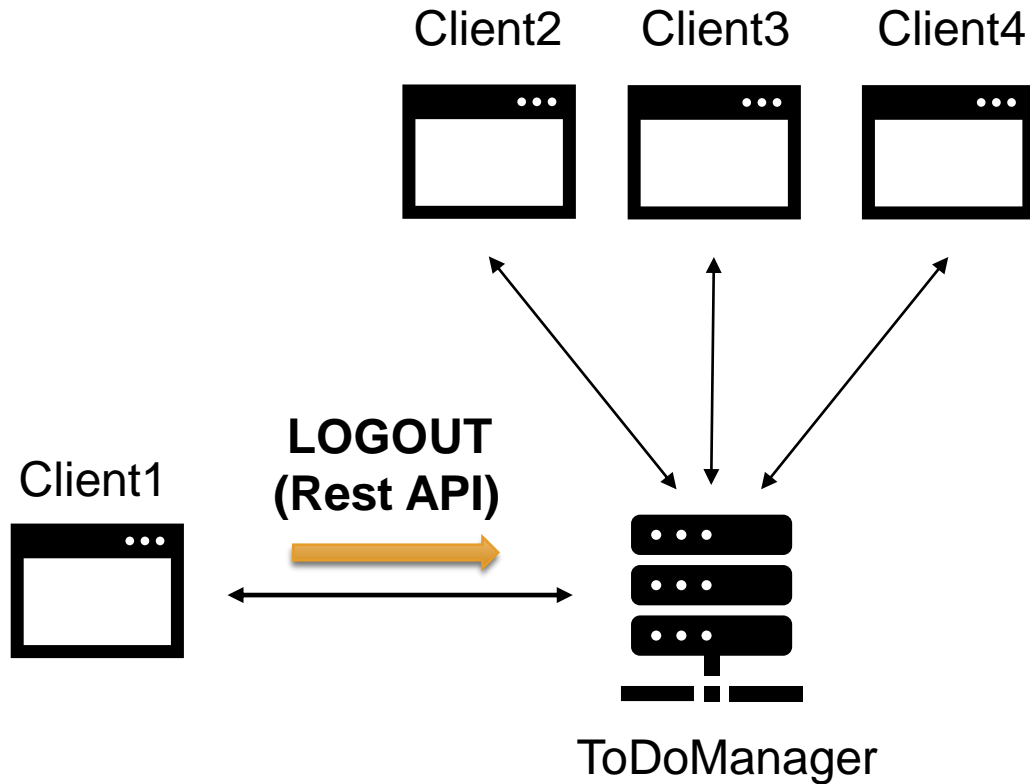
# WebSocket communication (task selection)



Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film
5	Beatrice	9	Read book

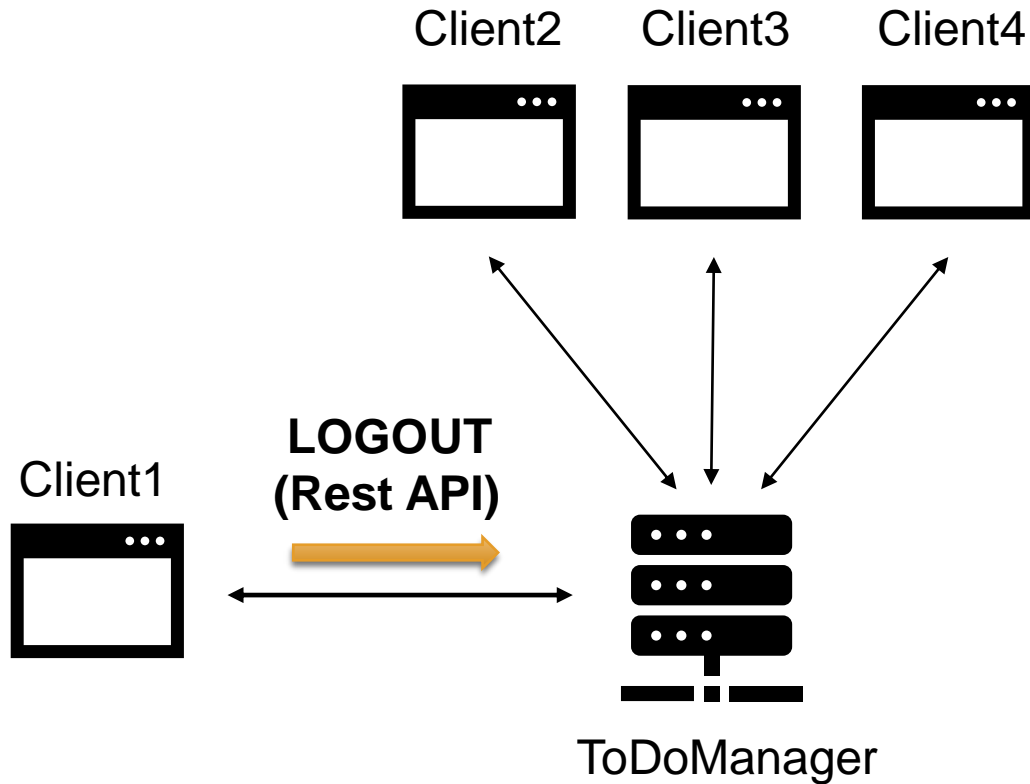
```
{  
  "typeMessage": "update",  
  "userId": "5",  
  "userName": "Beatrice",  
  "taskId": "9",  
  "taskName": "Read book"  
}
```

# WebSocket communication (logout)



Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film
5	Beatrice	9	Read book

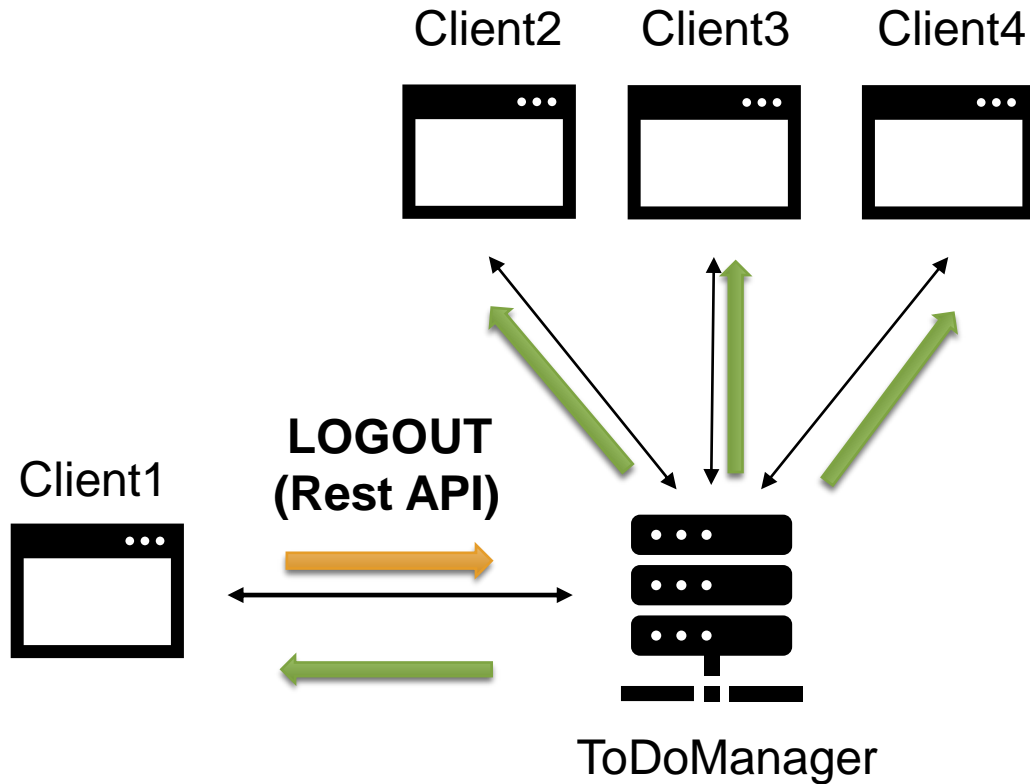
# WebSocket communication (logout)



Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film



# WebSocket communication (logout)



Status			
userId	userName	taskId	taskName
2	Frank	5	Study JSON
3	Karen	-	-
4	Rene	7	Watch film

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

# How does the React client *react*?



- The **Online** page displays:
    - the list of users that are **currently logged-in** the ToDoManager service;
    - for each listed user, the user name and the user id;
    - for each listed user, the task name and task id of the active task (if any).
  - **All the pages** (My Tasks, Public Tasks, Online and Assignment) 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*?



**ToDo Manager**   [My Tasks](#)   [Public Tasks](#)   [Online](#)   [Assignment](#)   [Logout](#)

Online Users

User: User

User: Beatrice

User: Frank

Online Users

**User Name: User**

UsedID: 1

Task Selected: 9 Study for the exam

**User Name: Beatrice**

UsedID: 5

Task Selected: 7 Walk the dog outside

**User Name: Frank**

UsedID: 2

Task not selected

# 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.js** file, **useEffect** 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;
  - call ***setDirty(true)***; to refresh the displayed content.

# 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.js** file, **useEffect** 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;
  - call ***setDirty(true)***; to refresh the displayed content.



Let's see how the React client should *react*!





# Thanks for your attention!

**Daniele Bringhenti**  
daniele.bringhenti@polito.it

