Laboratory Session #05

Distributed Systems Programming

Daniele Bringhenti

MQTT Features

- MQTT (Message Queuing Telemetry Transport) is a standard client-server **publish-subscribe** messaging transport protocol, usually based on TCP.
- The main features of MQTT are:
  1) **simplicity** (low requirements of processing or battery power)
  2) **efficiency** (lightweight transport);
  3) **scalability** (millions of devices);
  4) **reliability** (support for unreliable networks).

MQTT Features

- MQTT (Message Queuing Telemetry Transport) is a standard client-server **publish-subscribe** messaging transport protocol, usually based on TCP.
- The main features of MQTT are:
  1) **simplicity** (low requirements of processing or battery power)
  2) **efficiency** (lightweight transport);
  3) **scalability** (millions of devices);
  4) **reliability** (support for unreliable networks).

MQTT is suitable for **Internet-of-Things M2M** communications.

Topics of the Laboratory Session

Laboratory Session #05 covers the following activities:

Integration of **MQTT** functionalities in the implementation of the React client

Integration of **MQTT** functionalities in the implementation of the ToDoManager service

MQTT

Topics of the Laboratory Session

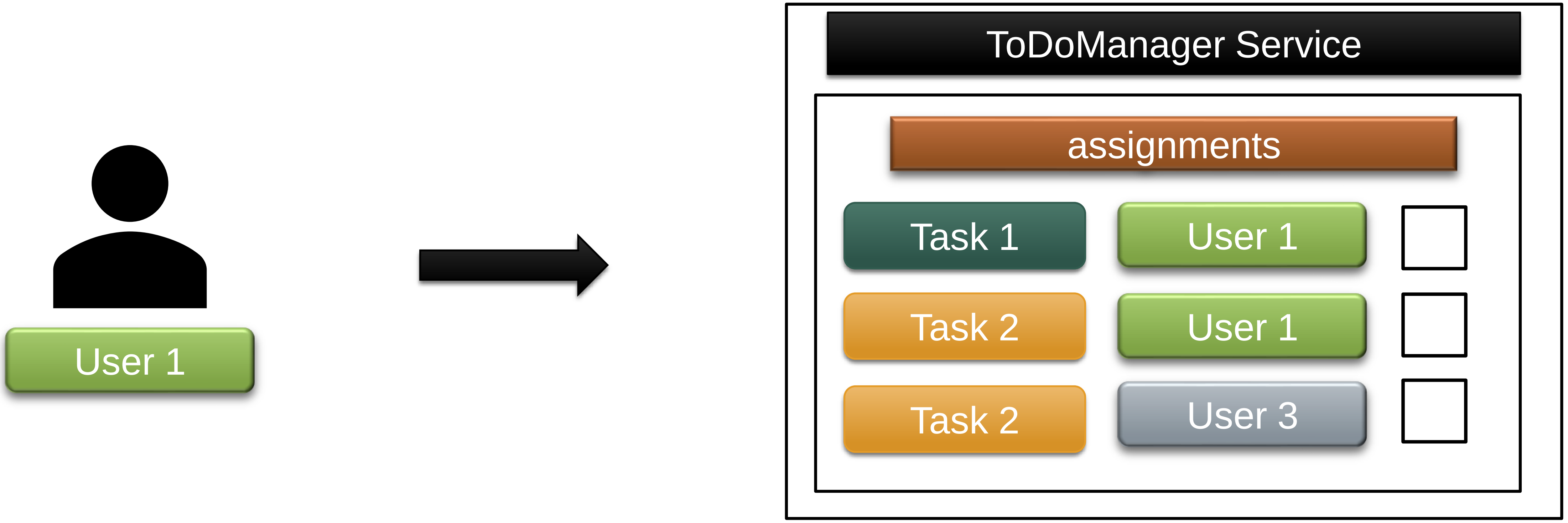
Laboratory Session #05 covers the following activities:

- Integration of **MQTT** functionalities in the implementation of the React client
- Integration of **MQTT** functionalities in the implementation of the ToDoManager service

MQTT

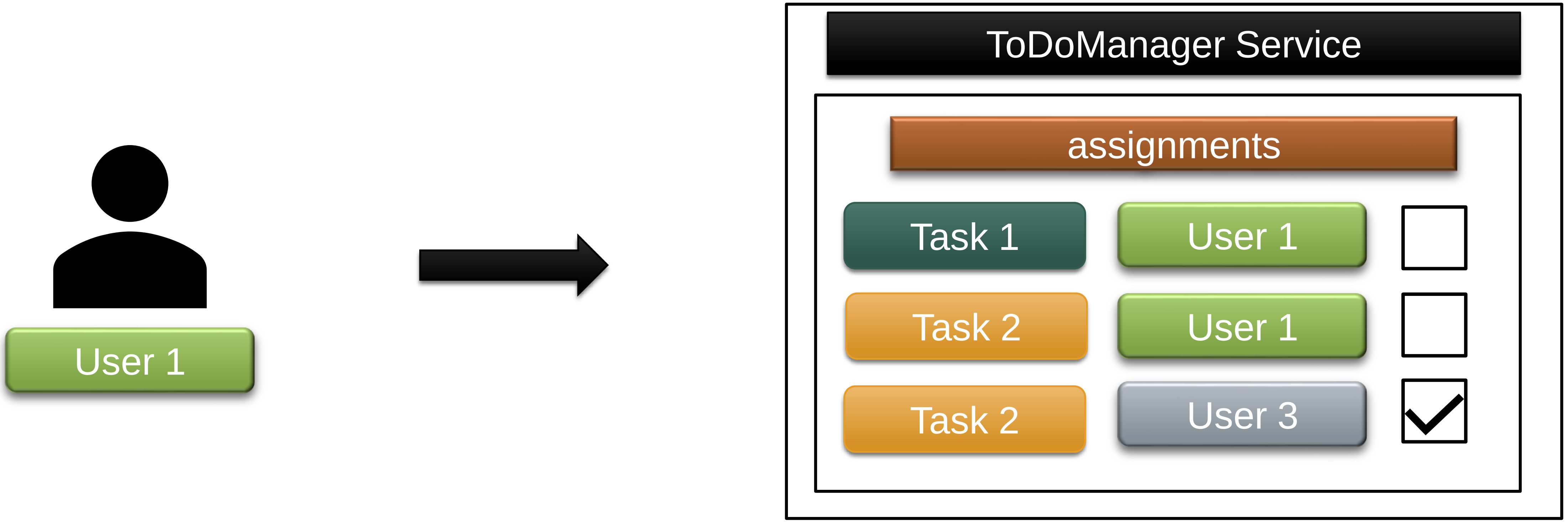
- Restriction of the **task selection** operation, with impact on **both** the ToDoManager and the React client

Task Selection in the ToDoManager service



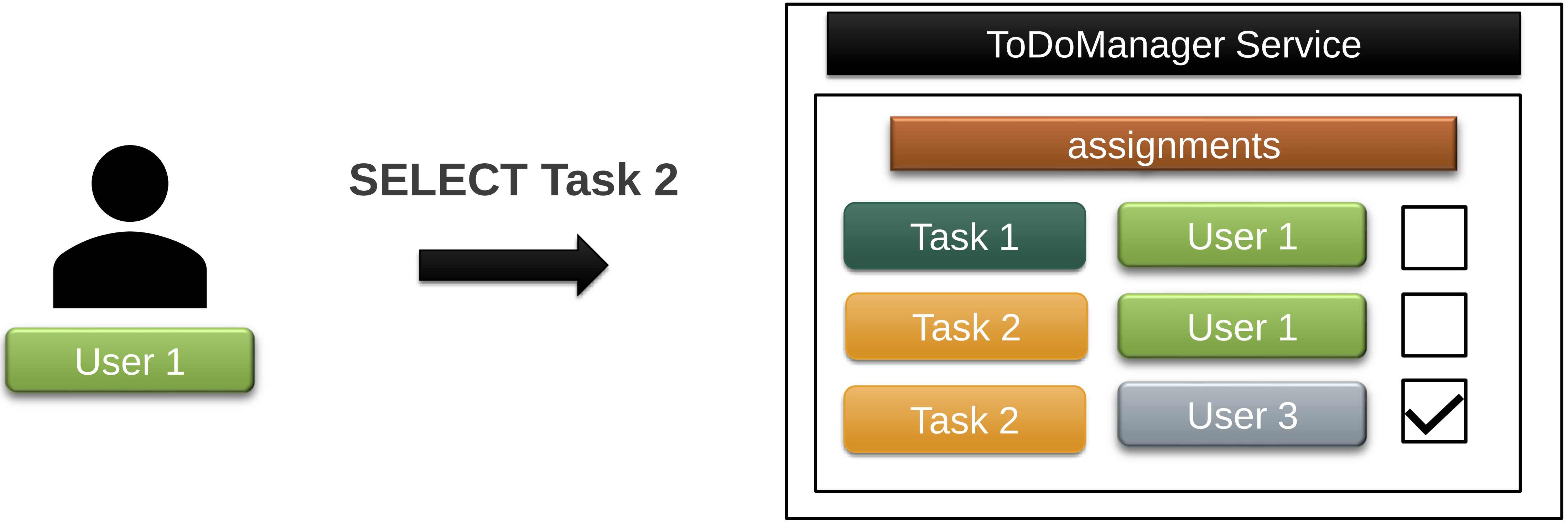
- A task can be the active task for **at most one** user at a time.
- In case a user tries to select a task which is active for another user, the operation **fails**.

Task Selection in the ToDoManager service



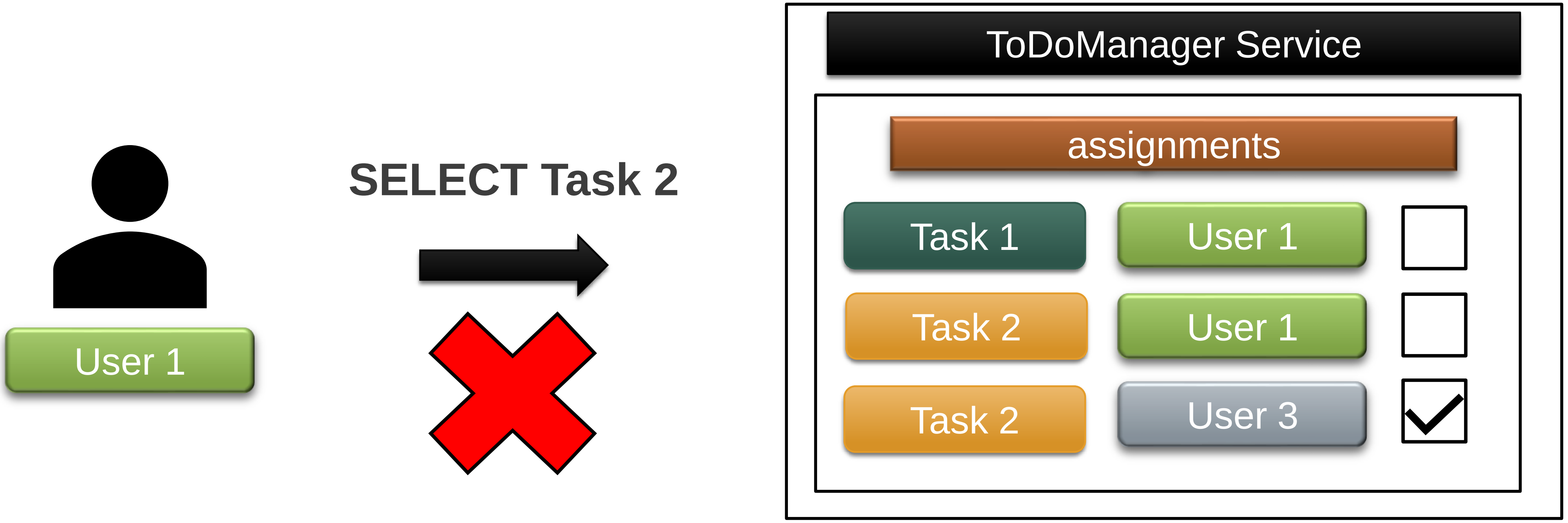
- A task can be the active task for **at most one** user at a time.
- In case a user tries to select a task which is active for another user, the operation **fails**.

Task Selection in the ToDoManager service



- A task can be the active task for **at most one** user at a time.
- In case a user tries to select a task which is active for another user, the operation **fails**.

Task Selection in the ToDoManager service



- A task can be the active task for **at most one** user at a time.
- In case a user tries to select a task which is active for another user, the operation **fails**.

Task Selection in the React client

- This new constraint demands for a **synchronization** among clients:
  - **eventual consistency** is acceptable;
  - at the end, all clients will **agree** about task selections.

- When a user tries to select a task, there are **two** options:
  1. the selection remains in a **pending state** until a confirmation or refusal of the selection comes from the server;
  2. the selection appears immediately as active to the user, but if it is later refused by the server, it is undone (**optimistic approach**).

  In both cases, the user must be informed about a failed selection with an **alert**.

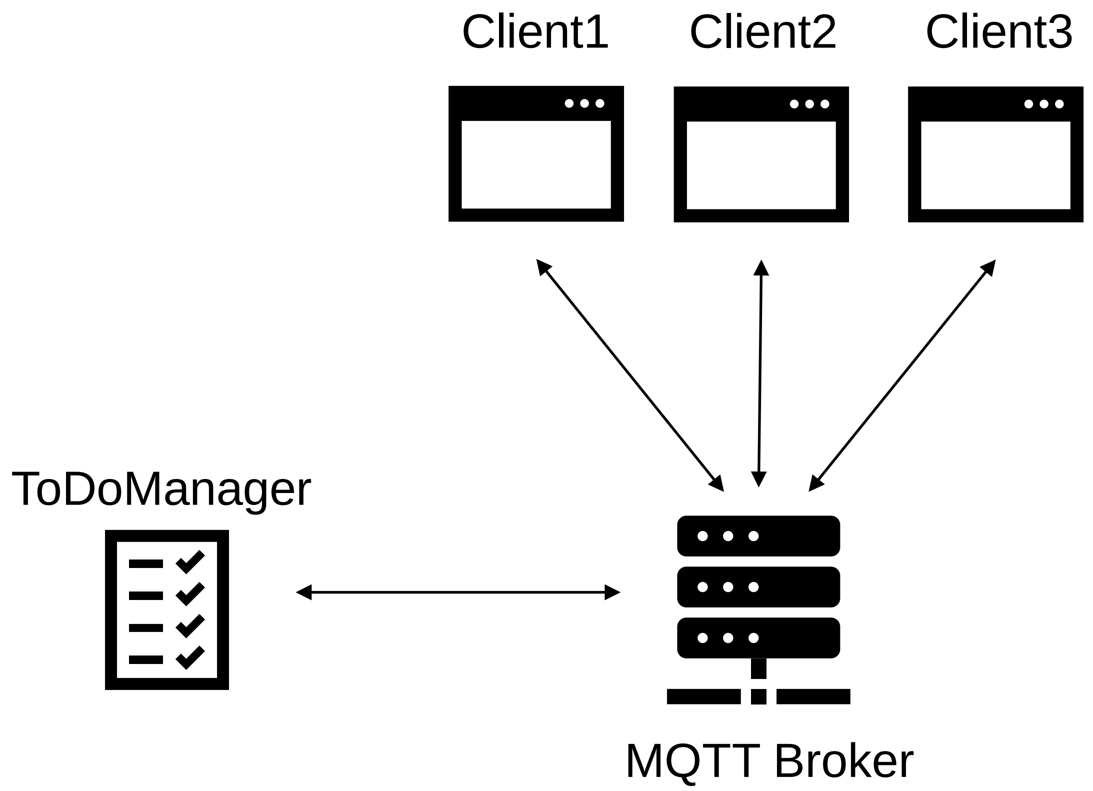
MQTT Communication

- Both the **ToDoManager** (TDM) service and the **React** client are extended with the functionality to communicate by using MQTT:
  - ToDoManager **publishes** MQTT messages;
  - the React client **subscribes** to topics and **receives** MQTT messages.

- The MQTT broker is **Eclipse Mosquitto**:
  - the recommended version is 1.6.12;
  - replace the **mosquitto.conf** file with the one you have been provided with;
  - launch Mosquitto with the following command:

```
mosquitto –v –c mosquitto.conf
```

MQTT communication (TDM - initial situation)



tasks

id
1
2
3
4

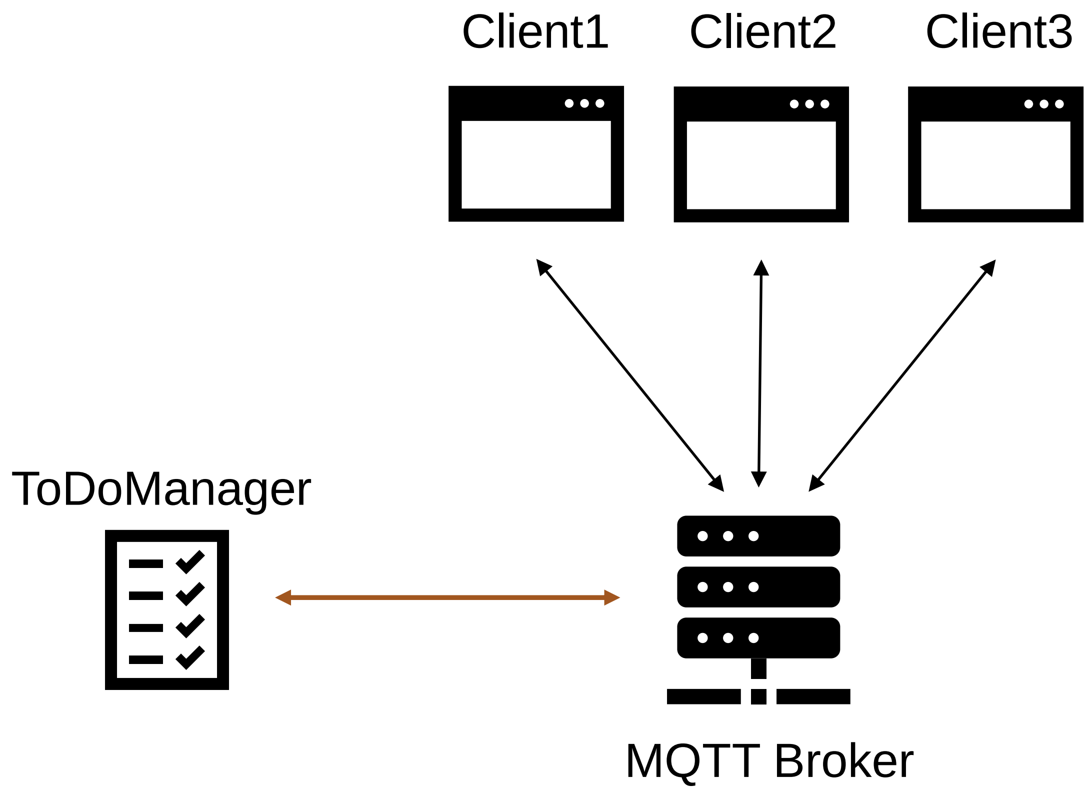
users

id	name
1	User
2	Frank
3	Karen
4	Rene

assignments

task	user	active
1	2	1
2	2	0
2	3	0
3	4	1

MQTT communication (TDM – connection establishment)



tasks

id
1
2
3
4

users

id	name
1	User
2	Frank
3	Karen
4	Rene

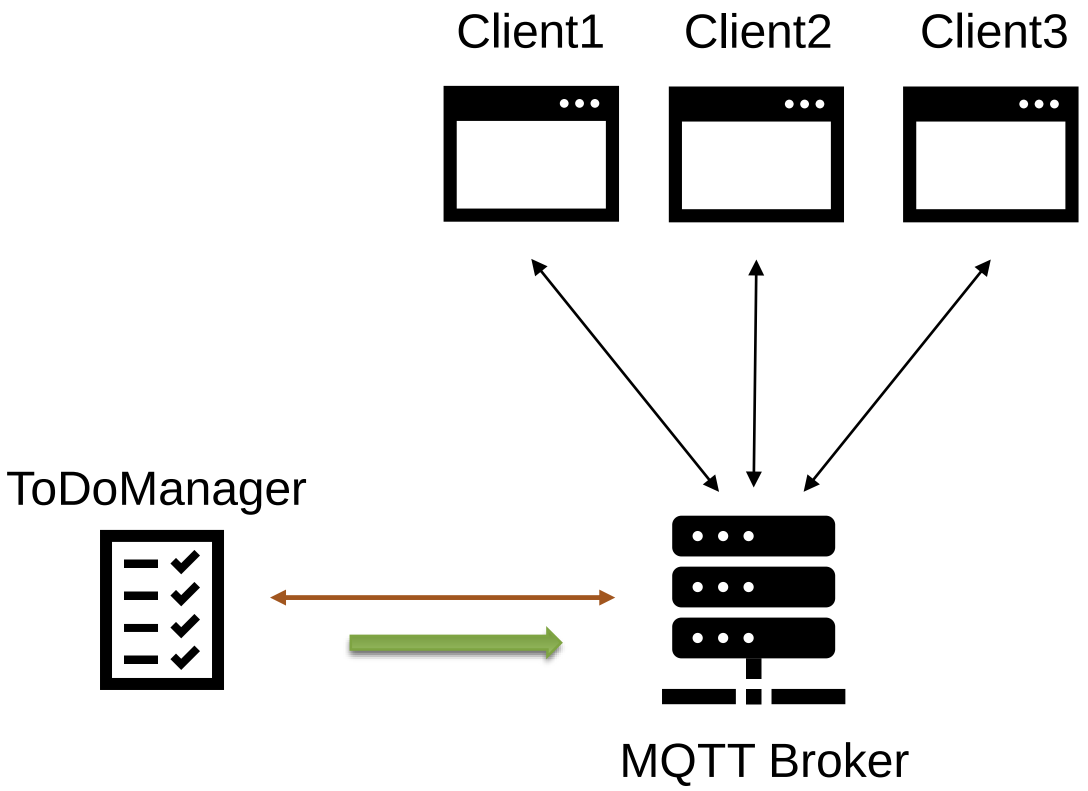
assignments

task	user	active
1	2	1
2	2	0
2	3	0
3	4	1

After the ToDoManager service successfully establishes a connection with the broker:

- it **publishes** a message for each existing task;
- each message must have the **retained** flag set to true.

MQTT communication (TDM – connection establishment)



tasks
id
1
2
3
4

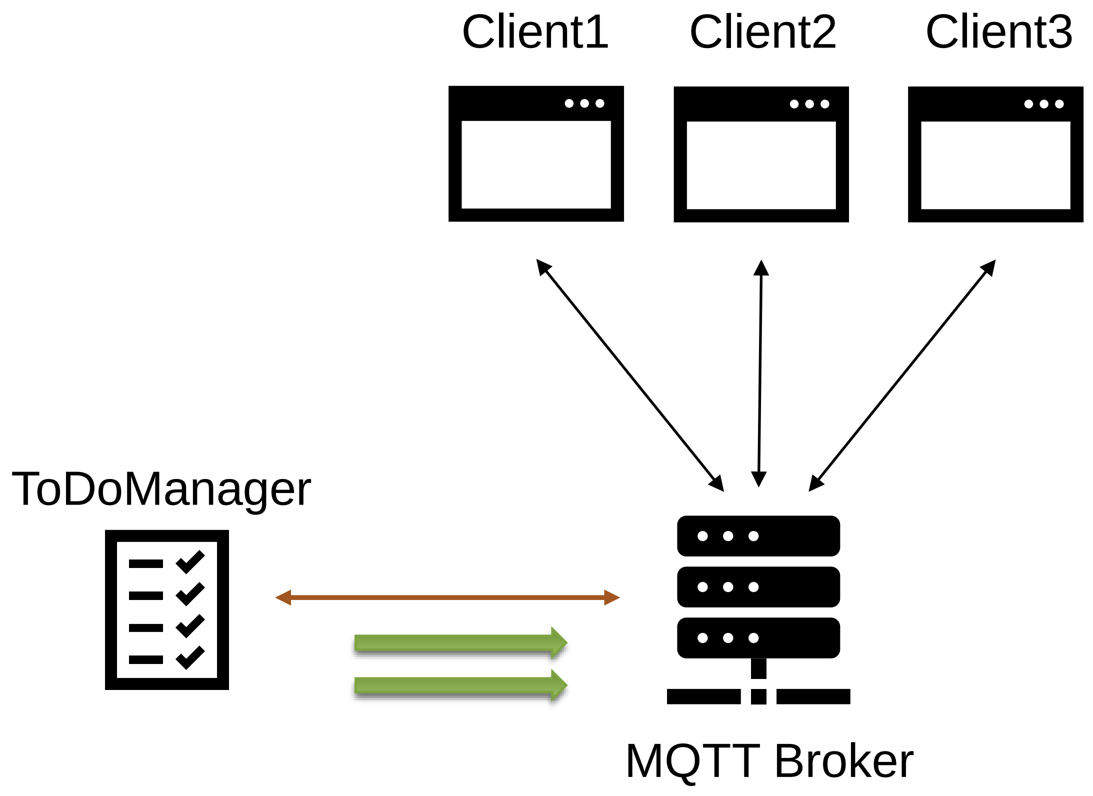
users	
id	name
1	User
2	Frank
3	Karen
4	Rene

assignments		
task	user	active
1	2	1
2	2	0
2	3	0
3	4	1

Topic: “1”

```
{
        “status” : “active”,
        “userId:  “2”,
        “userName”:  “Frank”
}
```

MQTT communication (TDM – connection establishment)



tasks

id
1
2
3
4

users

id	name
1	User
2	Frank
3	Karen
4	Rene

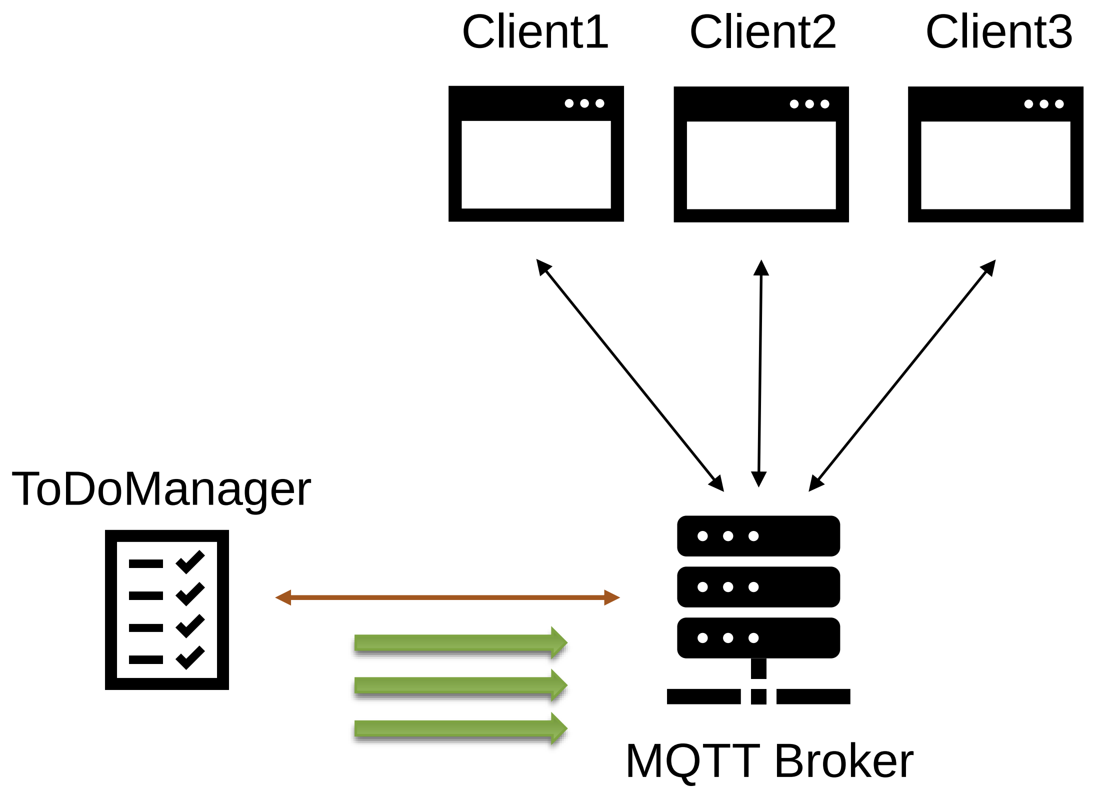
assignments

task	user	active
1	2	1
2	2	0
2	3	0
3	4	1

Topic: “2”

```
{
        “status” : “inactive”
}
```

MQTT communication (TDM – connection establishment)



tasks

id
1
2
3
4

users

id	name
1	User
2	Frank
3	Karen
4	Rene

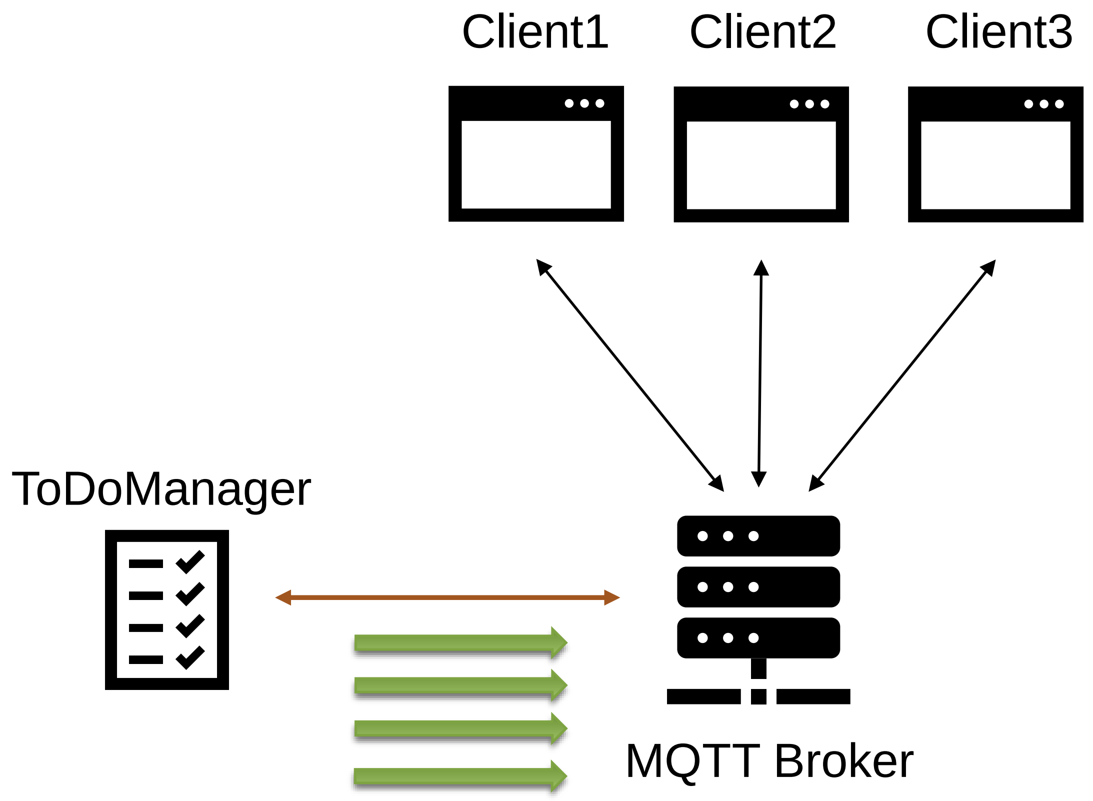
assignments

task	user	active
1	2	1
2	2	0
2	3	0
3	4	1

Topic: “3”

```
{
        “status” : “active”,
        “userId:  “4”,
        “userName”:  “Rene”
}
```

MQTT communication (TDM – connection establishment)



tasks
id
1
2
3
4

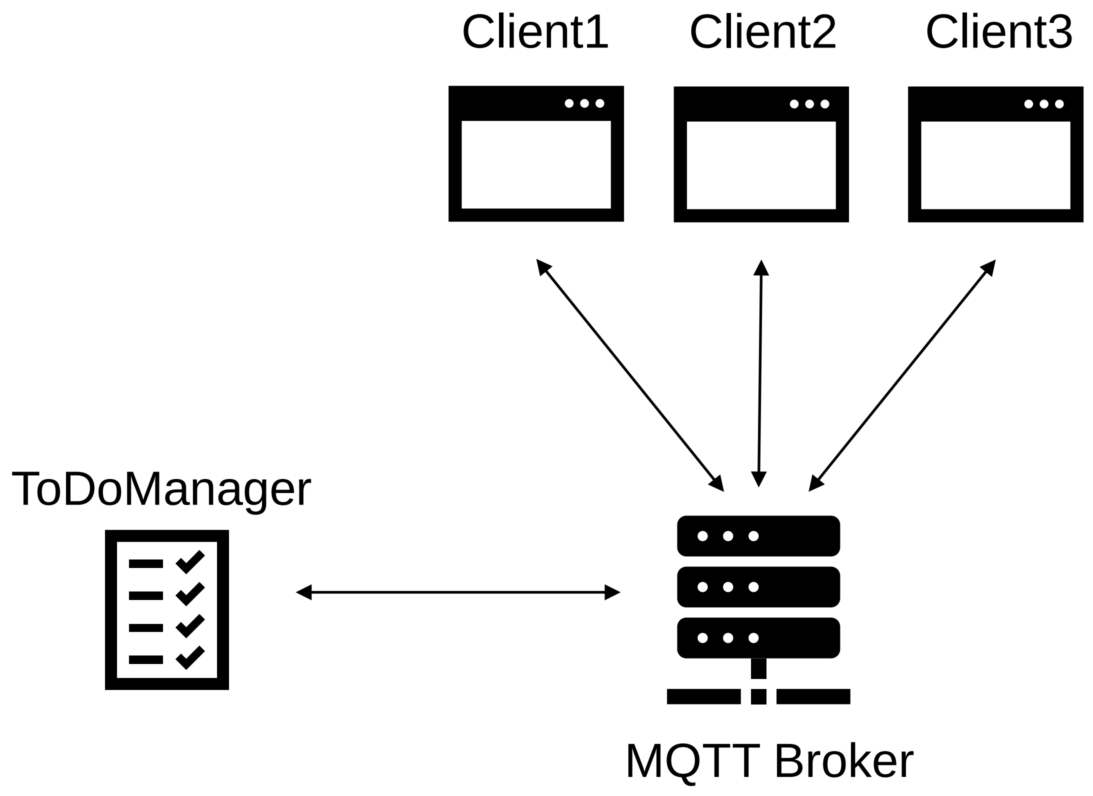
users	
id	**name**
1	User
2	Frank
3	Karen
4	Rene

assignments		
task	**user**	**active**
1	2	1
2	2	0
2	3	0
3	4	1

Topic: “4”

```
{
        “status” : “inactive”
}
```

MQTT communication (TDM – task selection)



tasks

id
1
2
3
4

users

id	name
1	User
2	Frank
3	Karen
4	Rene

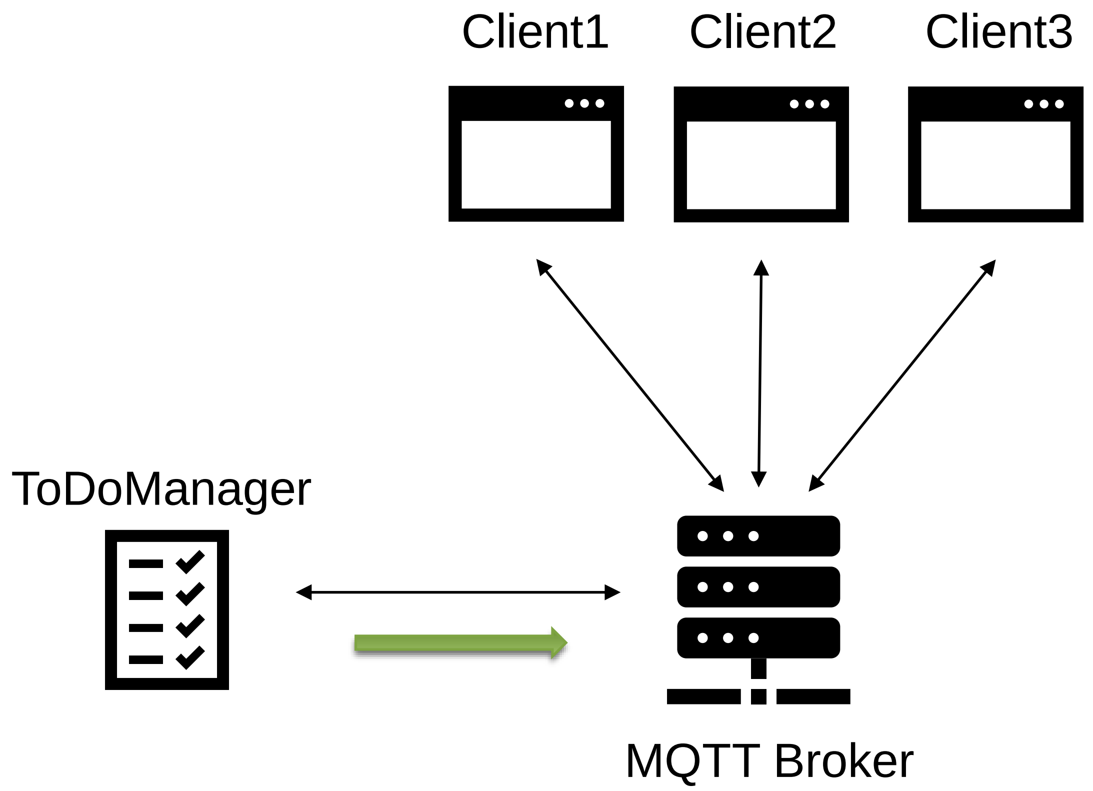
assignments

task	user	active
1	2	**0**
2	2	**1**
2	3	0
3	4	1

When a task becomes **active** for a different user, the ToDoManager publishes a retained message, conveying:

- the **active** status of that task;
- the **id** and **name** of the user who selected it.

MQTT communication (TDM – task selection)



tasks

id
1
2
3
4

users

id	name
1	User
2	Frank
3	Karen
4	Rene

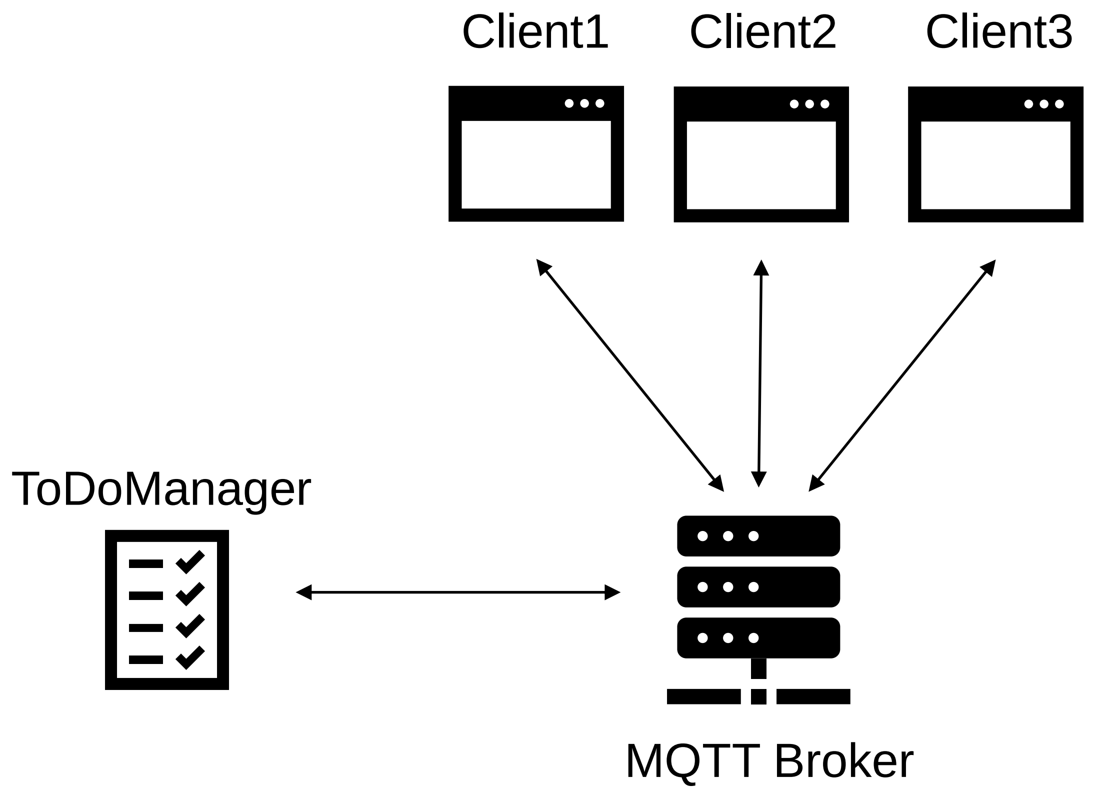
assignments

task	user	active
1	2	**0**
2	2	**1**
2	3	0
3	4	1

Topic: “2”

```
{
        “status” : “active”,
        “userId:  “2”,
        “userName”:  “Frank”
}
```

MQTT communication (TDM – task “de”selection)



tasks

id
1
2
3
4

users

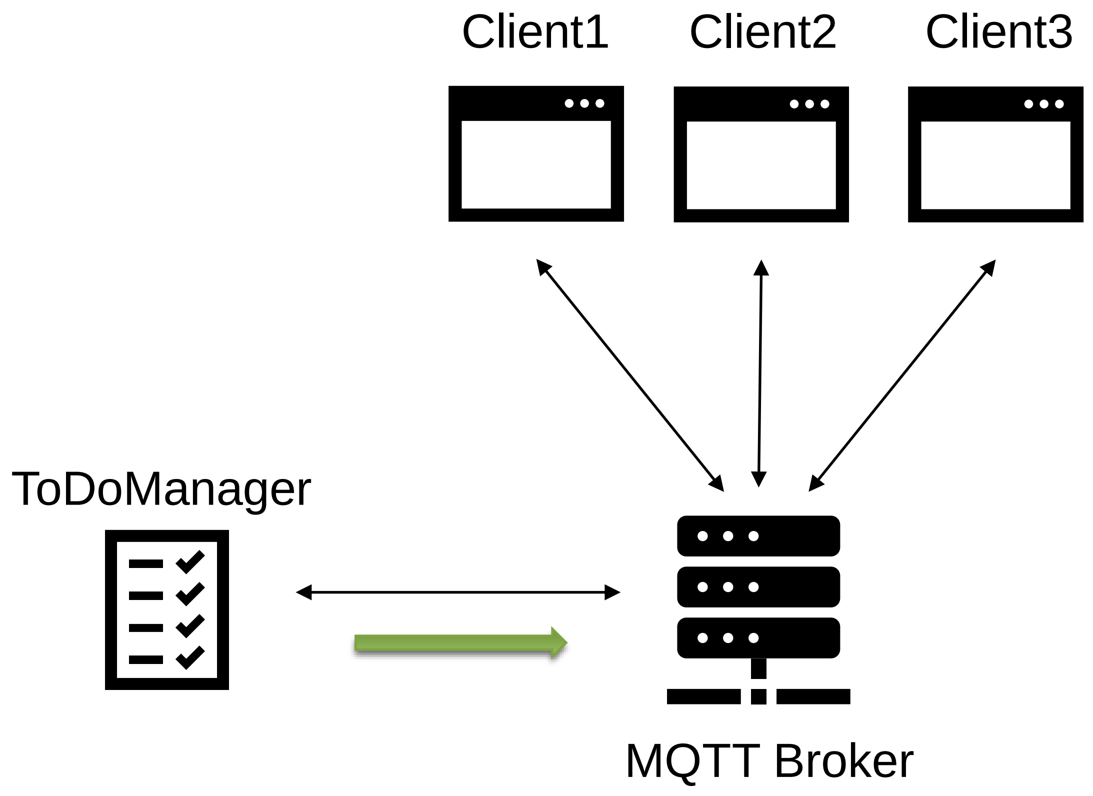
id	name
1	User
2	Frank
3	Karen
4	Rene

assignments

task	user	active
1	2	**0**
2	2	**1**
2	3	0
3	4	1

When a task is **not active** anymore for any user, the ToDoManager service publishes a retained message, conveying the **inactive** status of that task.

MQTT communication (TDM – task "de"selection)



tasks

id
1
2
3
4

users

id	name
1	User
2	Frank
3	Karen
4	Rene

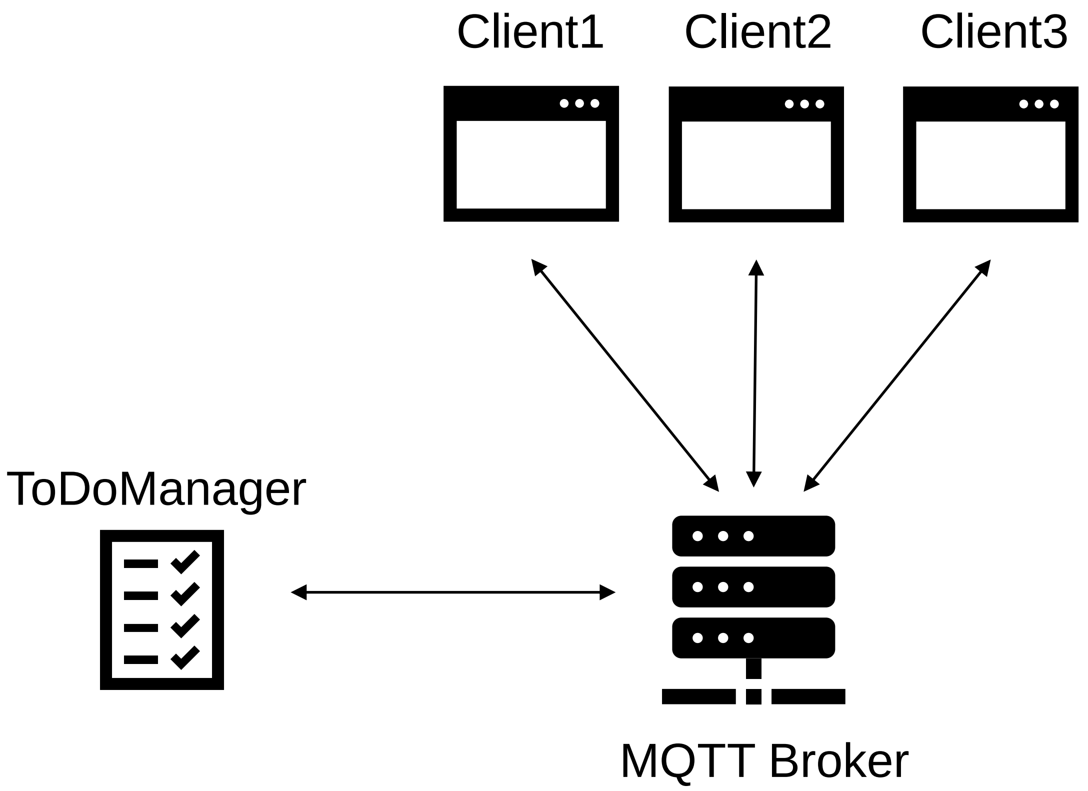
assignments

task	user	active
1	2	**0**
2	2	**1**
2	3	0
3	4	1

Topic: "1"

```
{
    "status" : "inactive"
}
```

MQTT communication (TDM – task creation)



tasks

id
1
2
3
4
5

users

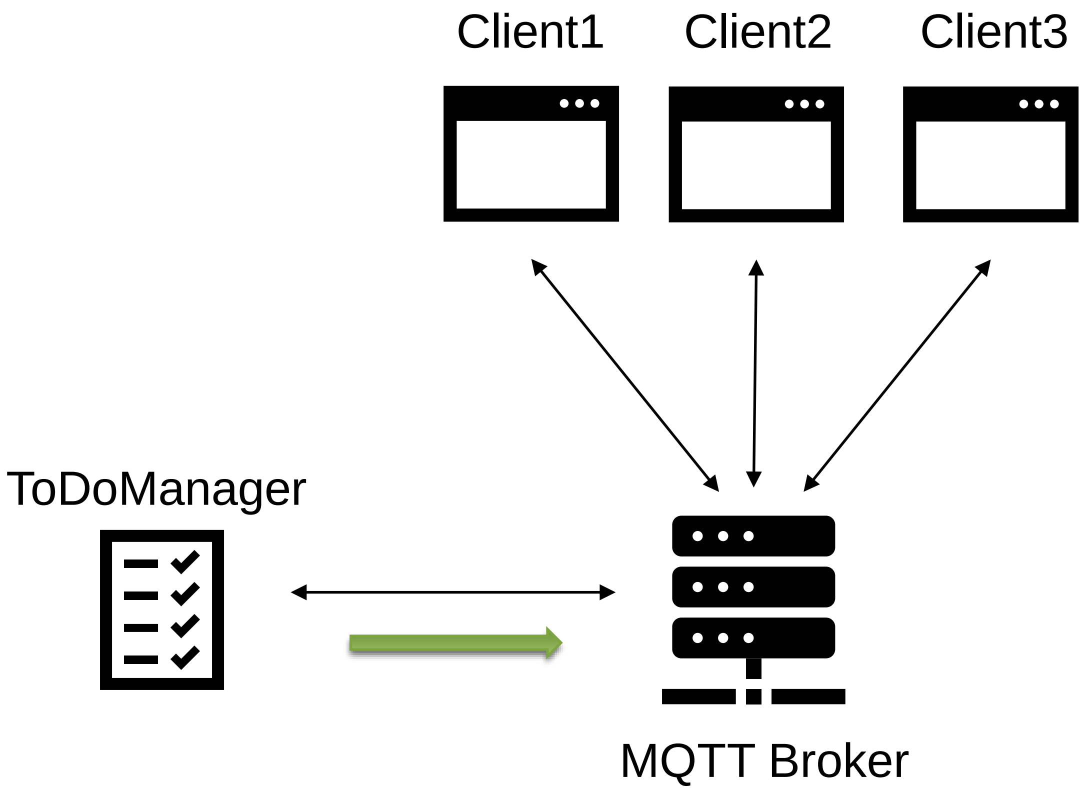
id	name
1	User
2	Frank
3	Karen
4	Rene

assignments

task	user	active
1	2	0
2	2	1
2	3	0
3	4	1

When a task is **created**, the ToDoManager service publishes a retained message, conveying the **inactive** status of that task.

MQTT communication (TDM – task creation)



tasks

id
1
2
3
4
5

users

id	name
1	User
2	Frank
3	Karen
4	Rene

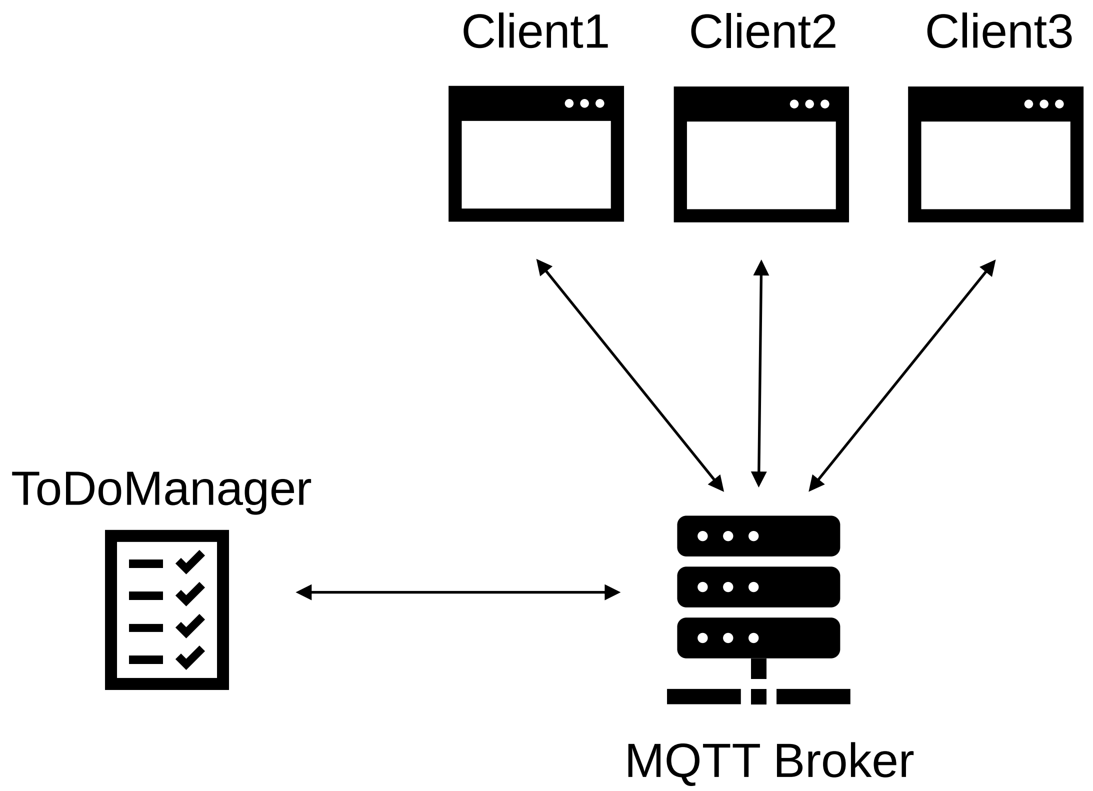
assignments

task	user	active
1	2	0
2	2	1
2	3	0
3	4	1

Topic: “5”

```
{
      “status” : “inactive”
}
```

MQTT communication (TDM – task deletion)



tasks

id
1
2
3
--
5

users

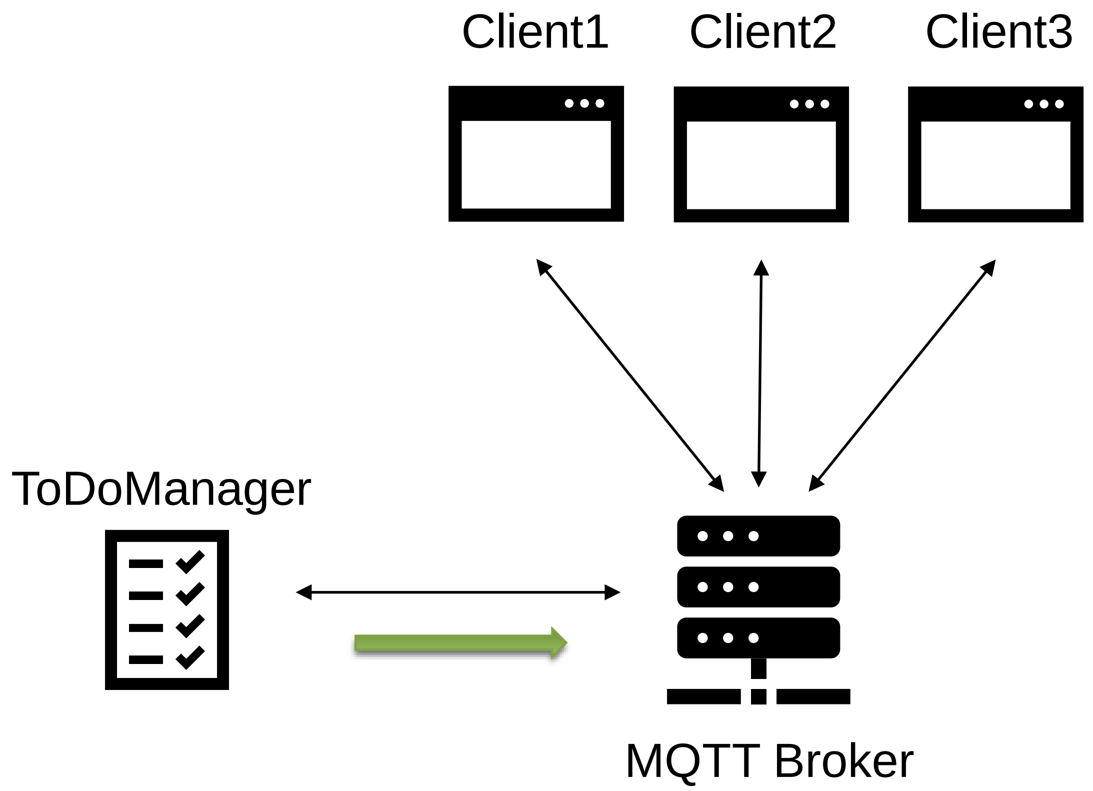
id	name
1	User
2	Frank
3	Karen
4	Rene

assignments

task	user	active
1	2	0
2	2	1
2	3	0
3	4	1

When a task is **deleted**, the ToDoManager service publishes a retained message informing the subscribed clients about this event (i.e., with the task status set to **deleted**).

MQTT communication (TDM – task deletion)



tasks

id
1
2
3
--
5

users

id	name
1	User
2	Frank
3	Karen
4	Rene

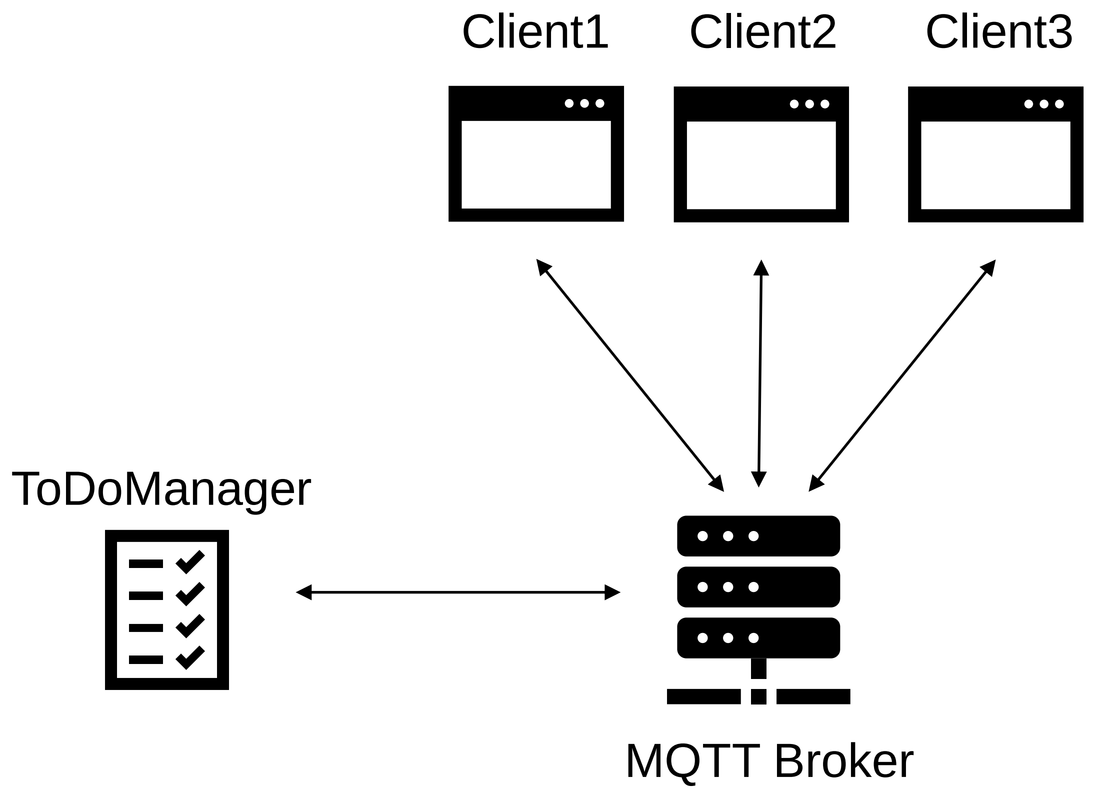
assignments

task	user	active
1	2	0
2	2	1
2	3	0
3	4	1

Topic: “4”

```
{
        “status” : “deleted”
}
```

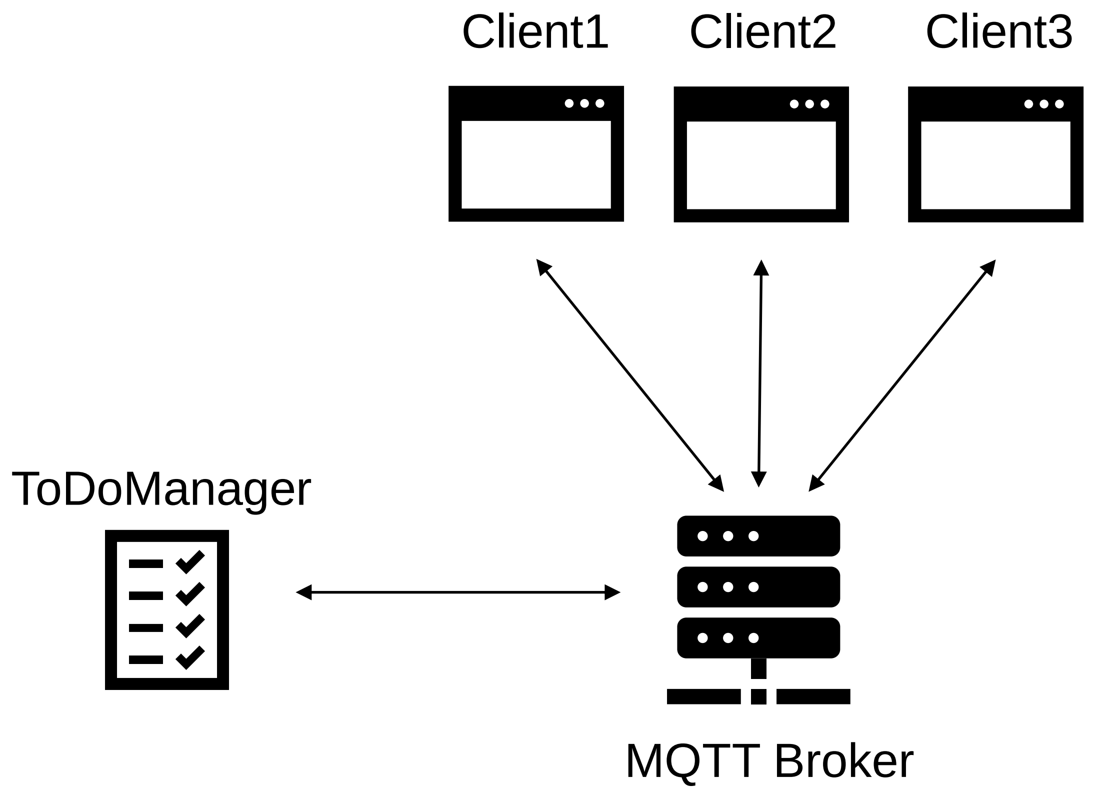
MQTT communication (Client - initial situation)

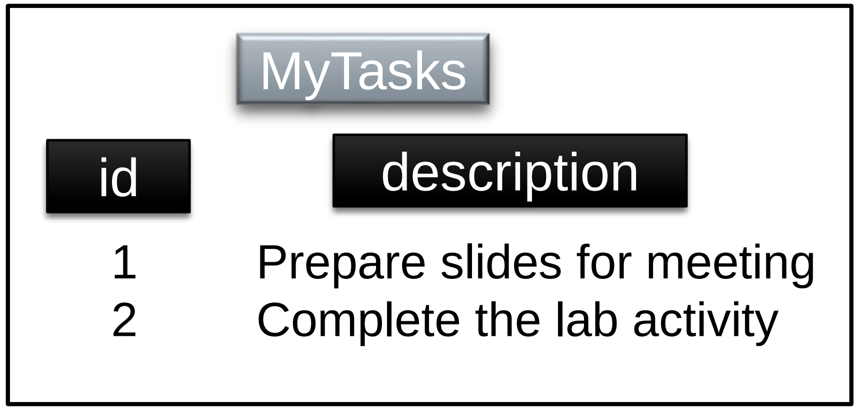


MyTasks

id	description
1	Prepare slides for meeting
2	Complete the lab activity

MQTT communication (Client - login)



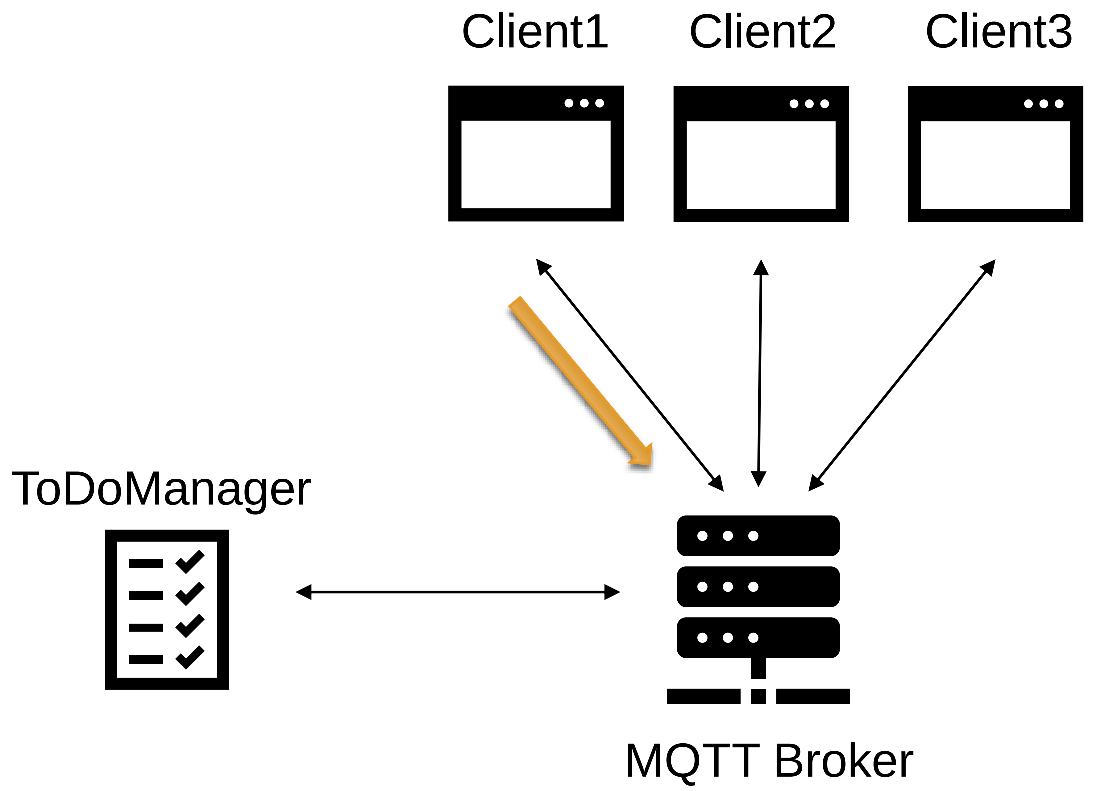


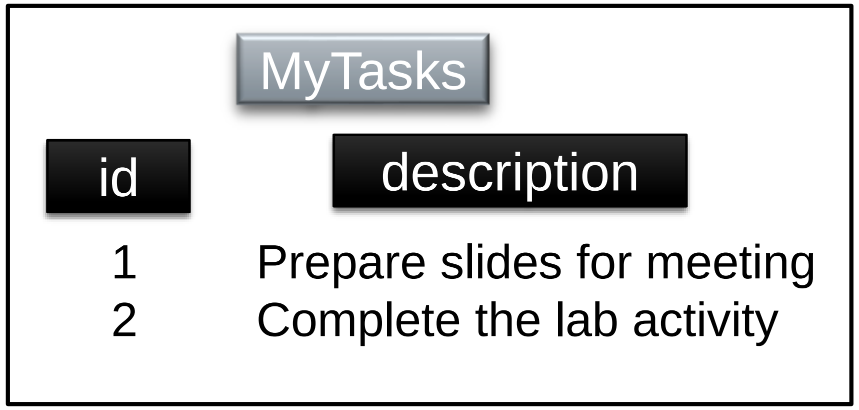
MyTasks

id	description
1	Prepare slides for meeting
2	Complete the lab activity

When a user **logs in** the ToDoManager service, the React client must **subscribe** to topics corresponding to the **id** of each task which is assigned to the user.

MQTT communication (Client - login)



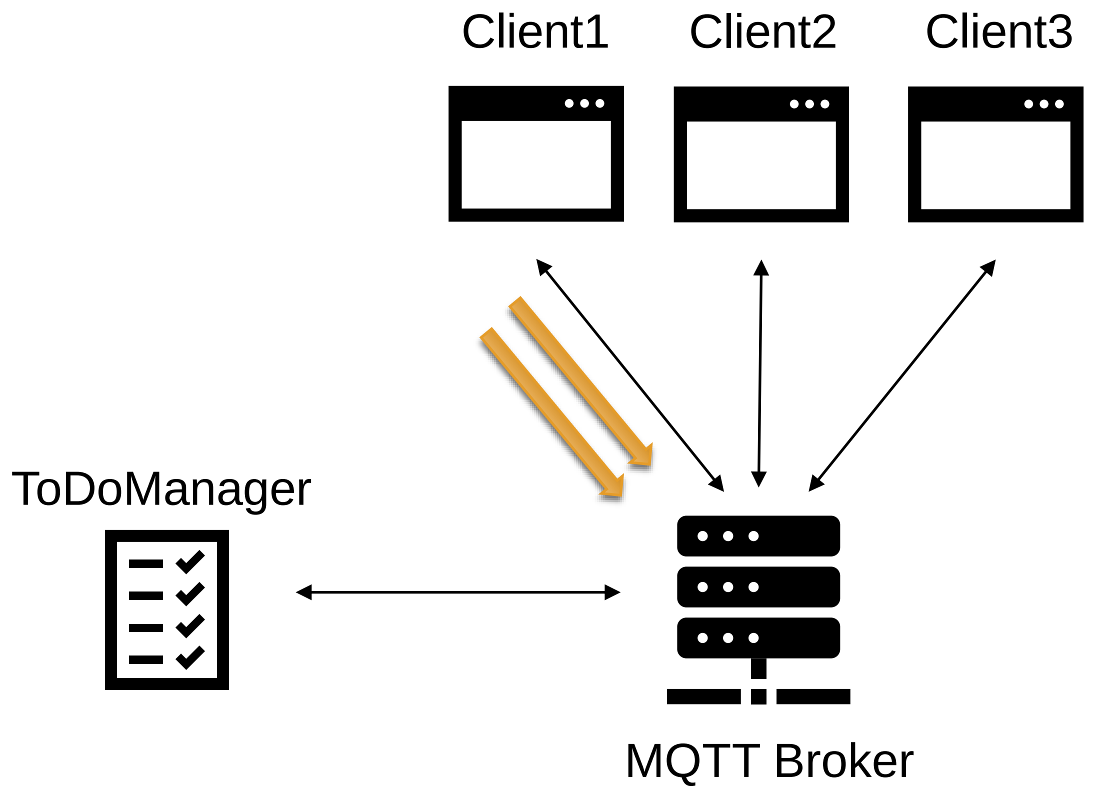


MyTasks

id	description
1	Prepare slides for meeting
2	Complete the lab activity

Subscribe to topic: “1”

MQTT communication (Client - login)



MyTasks

id	description
1	Prepare slides for meeting
2	Complete the lab activity

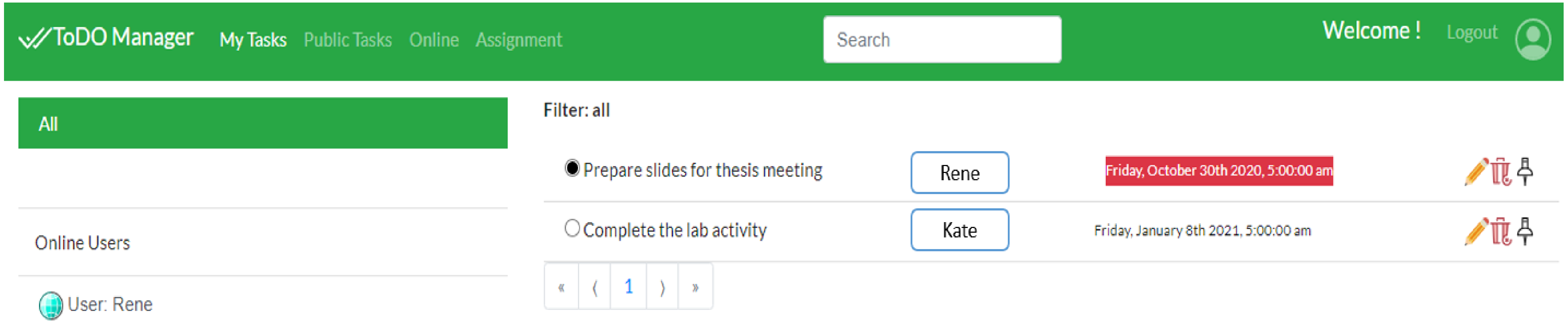
Subscribe to topic: “2”

How does the React client *react*?

- The React client may **receive** MQTT messages for the topic to which there exists a subscription:
  - after the subscription to each topic, retained messages are received;
  - after each status change for task selection, a new message is received.

- The React client **reacts** in the following ways:
  1. whenever the React client receives an MQTT message related to a task, it updates the status of the task in the **My Tasks** page of the GUI;
  2. whenever the selection of a task performed by the logged-in user **fails**, an **alert** message should be shown on the screen.

How does the React client *react*?



ToDO Manager My Tasks Public Tasks Online Assignment

Search

Welcome ! Logout

All

Online Users

User: Rene

Filter: all

● Prepare slides for thesis meeting	Rene	Friday, October 30th 2020, 5:00:00 am	
○ Complete the lab activity	Kate	Friday, January 8th 2021, 5:00:00 am	

« ‹ 1 › »

Final Tips

- For the communication with a web browser, MQTT messages must be encapsulated into WebSocket frames (**MQTT Over Websockets**):
  - ➢ the URL to be specified for the MQTT connection is **ws**://127.0.0.1:8080.

- For the reaction of the *React* client, you only need to use this line of code in **App.js**:

this.**displayTaskSelection**(*topic*, *parsedMessage*);

where:

- *topic* is a string representing the id of the task;
- *parsedMessage* is the JSON object retrieved after parsing the MQTT message.

Final Tips

- For the communication with a web browser, MQTT messages must be encapsulated into WebSocket frames (**MQTT Over Websockets**):
  - the URL to be specified for the MQTT connection is **ws**://127.0.0.1:8080.

- For the reaction of the *React* client, you only need to use this line of code in **App.js**:

this.**displayTaskSelection**(*topic*, *parsedMessage*);

where:

- *topic* is a string representing the id of the task;
- *parsedMessage* is the JSON object retrieved after parsing the MQTT message.

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

Thanks for your attention!

Daniele Bringhenti

daniele.bringhenti@polito.it