

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO

Hệ phân tán

LAB 02 thực hành

Sinh viên thực hiện: **Nguyễn Thanh Hà**

Mã số sinh viên: **20210298**

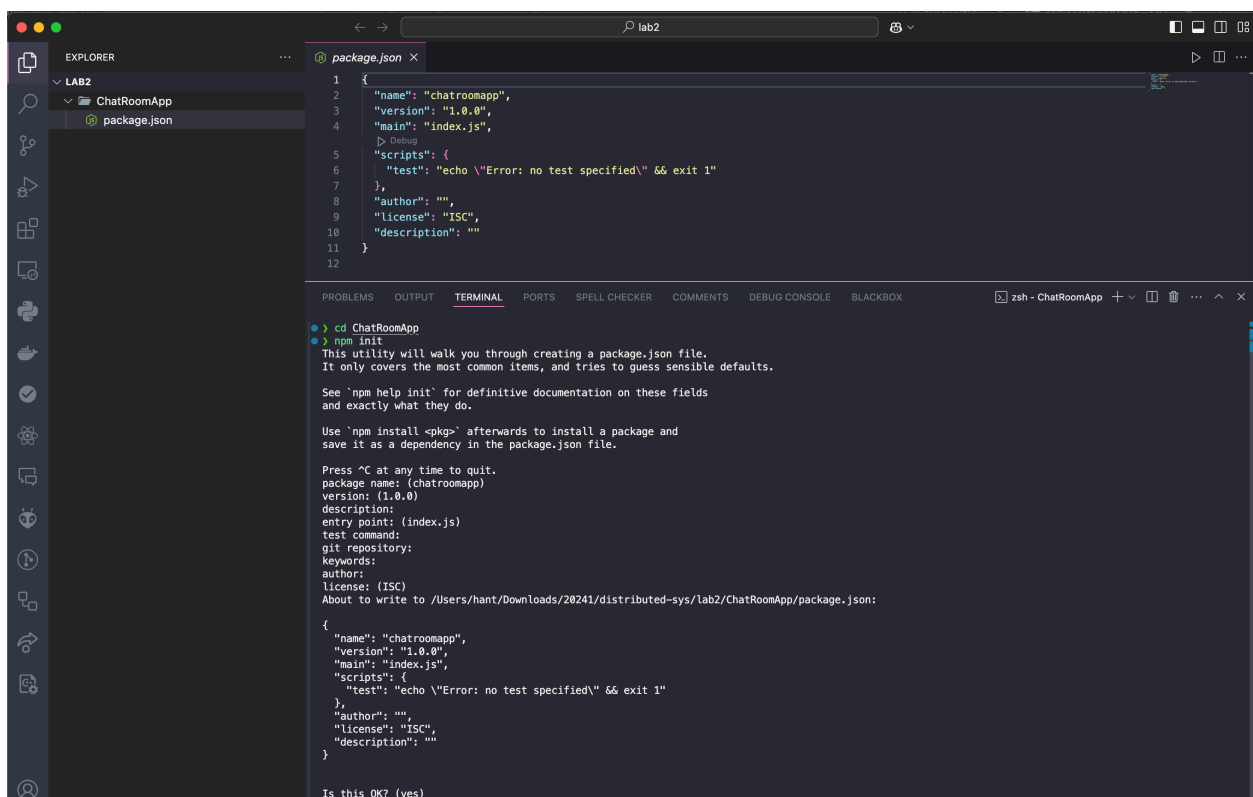
Giảng viên: **TS. Trần Hải Anh**

Hà Nội, 11 – 2024

1. Xây dựng một Chat room sử dụng socket.io

Câu hỏi 1: Tập nào vừa xuất hiện trong thư mục ChatRoomApp? Nó được sử dụng để làm gì?

- Tập package.json xuất hiện nó được sử dụng để định nghĩa chi tiết thông tin cho project
- Lưu thông tin “name”, “version”, “description”, “main”, “scripts”, “author”, “license”, “dependencies”.



The screenshot shows a VS Code editor with a project named 'ChatRoomApp' in a folder 'LAB2'. The 'package.json' file is open, showing the following content:

```
1 {
2   "name": "chatroomapp",
3   "version": "1.0.0",
4   "main": "index.js",
5   "scripts": {
6     "test": "echo \\\"Error: no test specified\\\" && exit 1"
7   },
8   "author": "",
9   "license": "ISC",
10  "description": ""
11 }
12
```

The terminal output shows the command 'npm init' being executed, which prompts the user to create a package.json file. The output includes the following information:

```
zsh - ChatRoomApp
> cd ChatRoomApp
> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

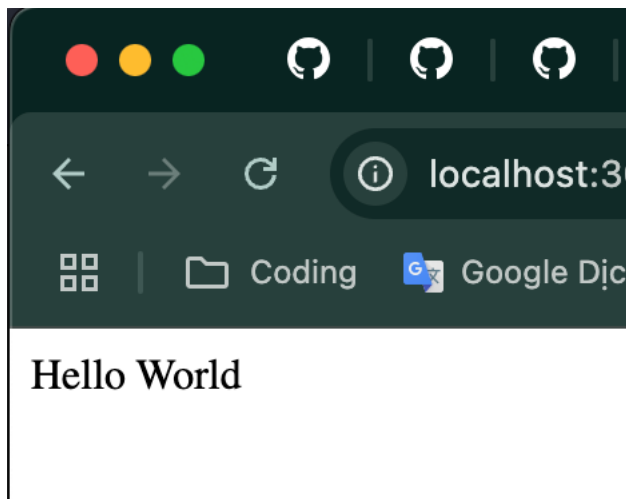
See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (chatroomapp)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/hant/Downloads/20241/distributed-sys/lab2/ChatRoomApp/package.json:
{
  "name": "chatroomapp",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "description": ""
}

Is this OK? (yes)
```

Câu hỏi 2: Mở trình duyệt và gõ vào đó địa chỉ <http://localhost:3000>, bạn sẽ nhận được thông điệp gì?



- Thông điệp “hello world”.

Câu hỏi 3: Bạn hãy thử reload (Ctrl-R) lại trình duyệt. Bạn có nhìn thấy gì mới xuất hiện trên cửa sổ không? Nếu không có gì xuất hiện hết thì là vì sao?

- Không hiện gì mới cả.

- Không hiện vì chỉ cài đặt socket.io cho server chứ chưa cài đặt socket.io cho client.

Câu hỏi 4: Refresh trang localhost:3000, bạn nhìn thấy thông điệp nào?

- Thấy thêm một dòng “*New user connected*” trên Console.

2. Phát triển hệ thống RPC sử dụng RabbitMQ

Câu hỏi 6: Đây là đoạn code mà Server gán correlationID vào câu trả lời?

Đoạn mã mà server gán correlationId vào câu trả lời là trong phần xử lý callback của server, nơi server trả lời lại client. Cụ thể, đoạn mã sau đây gán correlationId vào thuộc tính của câu trả lời (BasicProperties) và sau đó gửi lại câu trả lời cho client

```
DeliverCallback deliverCallback = (consumerTag, delivery) -> {  
    BasicProperties replyProps = new BasicProperties.Builder()  
        .correlationId(delivery.getProperties().getCorrelationId())  
        .build();  
  
    String response = "";
```

Câu hỏi 7: Dựa vào cả code của Client và Server để giải thích đây là đoạn code mà Client gửi yêu cầu lên cho Server thông qua hàng đợi rpc_queue và tạo ra một hàng đợi mới để chờ câu trả lời của Server.

- Client gửi yêu cầu lên cho Server qua rpc_queue thông qua phương thức channel.basicPublish()

```
channel.basicPublish("", requestQueueName, props, message.getBytes(StandardCharsets.UTF_8));
```

- Tạo hàng đợi mới để chờ response từ server và gửi thông điệp với correlationId và replyTo

```
String replyQueueName = channel.queueDeclare().getQueue();  
BasicProperties props = new BasicProperties.Builder()  
    .correlationId(correlationId)  
    .replyTo(replyQueueName)  
    .build();
```

- Chờ và nhận response từ Server

```
final BlockingQueue<String> response = new ArrayBlockingQueue<>( capacity: 1);
String cTag = channel.basicConsume(replyQueueName, b: true, (consumerTag, delivery) -> {
    if (delivery.getProperties().getCorrelationId().equals(corrId)) {
        response.offer(new String(delivery.getBody(), StandardCharsets.UTF_8));
    }
}, consumerTag -> {});
```

Câu hỏi 8: Bây giờ hãy thử thêm một chút delay vào chương trình Server bằng cách thêm vào đoạn code sau ở dưới dòng: `response += fib(n);`

```
try {
    Thread.sleep(2000);
} catch (InterruptedException _ignored) {
    Thread.currentThread().interrupt();
}
```

Chương trình Server sẽ ngủ 2s đối với mỗi request. Hãy dịch lại chương trình Server và chạy nó.

Mở cùng lúc nhiều cửa sổ command và chạy nhiều chương trình Client trên đó cùng lúc.

Cùng lúc đó mở một cửa sổ command khác và chạy dòng lệnh sau:

```
>rabbitmqctl.bat list_queues name messages_ready
```

`messages_unacknowledged` Bạn nhận được kết quả hiển thị gì? Giải thích!

Kết quả chạy được:

```
rpc_queue 0 2
```

Giải thích kết quả:

- **rpc_queue**: Đây là tên của hàng đợi (queue) mà các client đang gửi yêu cầu đến server.
- **messages_ready**: Số lượng tin nhắn (message) đang chờ trong hàng đợi để được server xử lý. Trong kết quả trên, giá trị này là 0, có thể xảy ra khi server đang bận xử lý một request và chưa kịp trả lời các client.
- **messages_unacknowledged**: Số lượng tin nhắn đã được server nhận và đang được xử lý nhưng chưa được xác nhận (acknowledged). Trong kết quả trên,

giá trị này là 2, có thể là do server đang xử lý 2 request từ 2 client khác nhau với delay 2 giây mỗi lần.