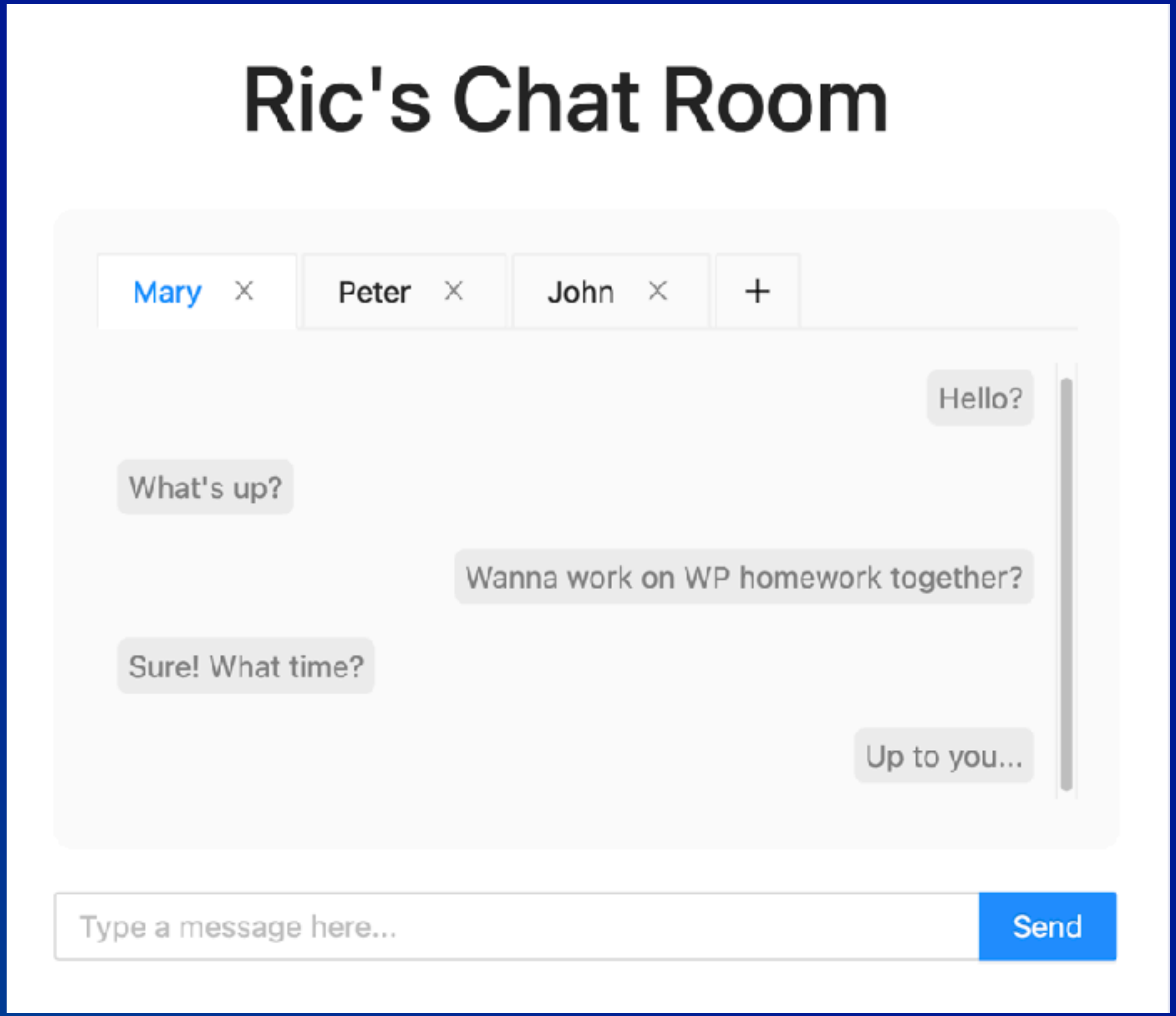
# 11. Migrating from WebSocket to GraphQL


Ric's Chat Room

Hello?

What's up?

Wanna work on WP homework together?

Sure! What time?

Up to you...

Ric Huang / NTUEE

(EE 3035) Web Programming

先預告一下，Hack#3 是有關於
GraphQL, 所以大家一定要花時間
把 Lecture Notes #10 & 11 看懂

## 先講一下 HW#8...

- 題目：把 HW#7 ChatRoom 的 WS 改成 GraphQL
    - 如果你的 HW#7 沒有寫完/寫好，我們開放你可以使用別人的 HW#7 來改寫，但請在 README 註明
    - 基本要求與 HW#7 相同，但 HW#8 要求支援 1x1 的對話，因此，UI/UX 以及 DB 都需要做一些修正
    - 我們會將 deadline 延長一個星期，讓大家有時間準備 Hack#3, 之後再來把 HW#9 完成

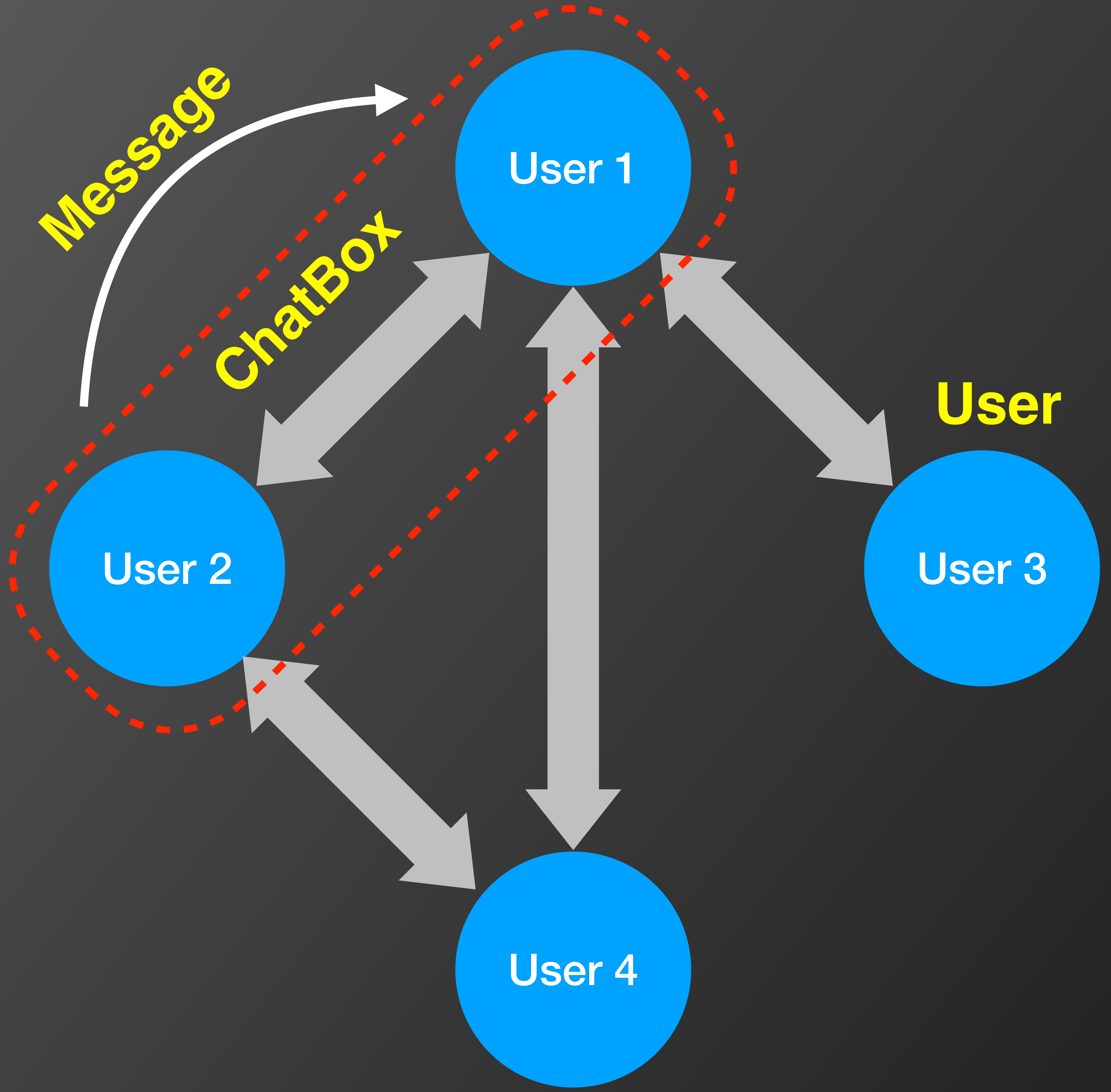
**底下的講義先跟大家提示如何把 ChatRoom 的前後端用 GraphQL 串起來**

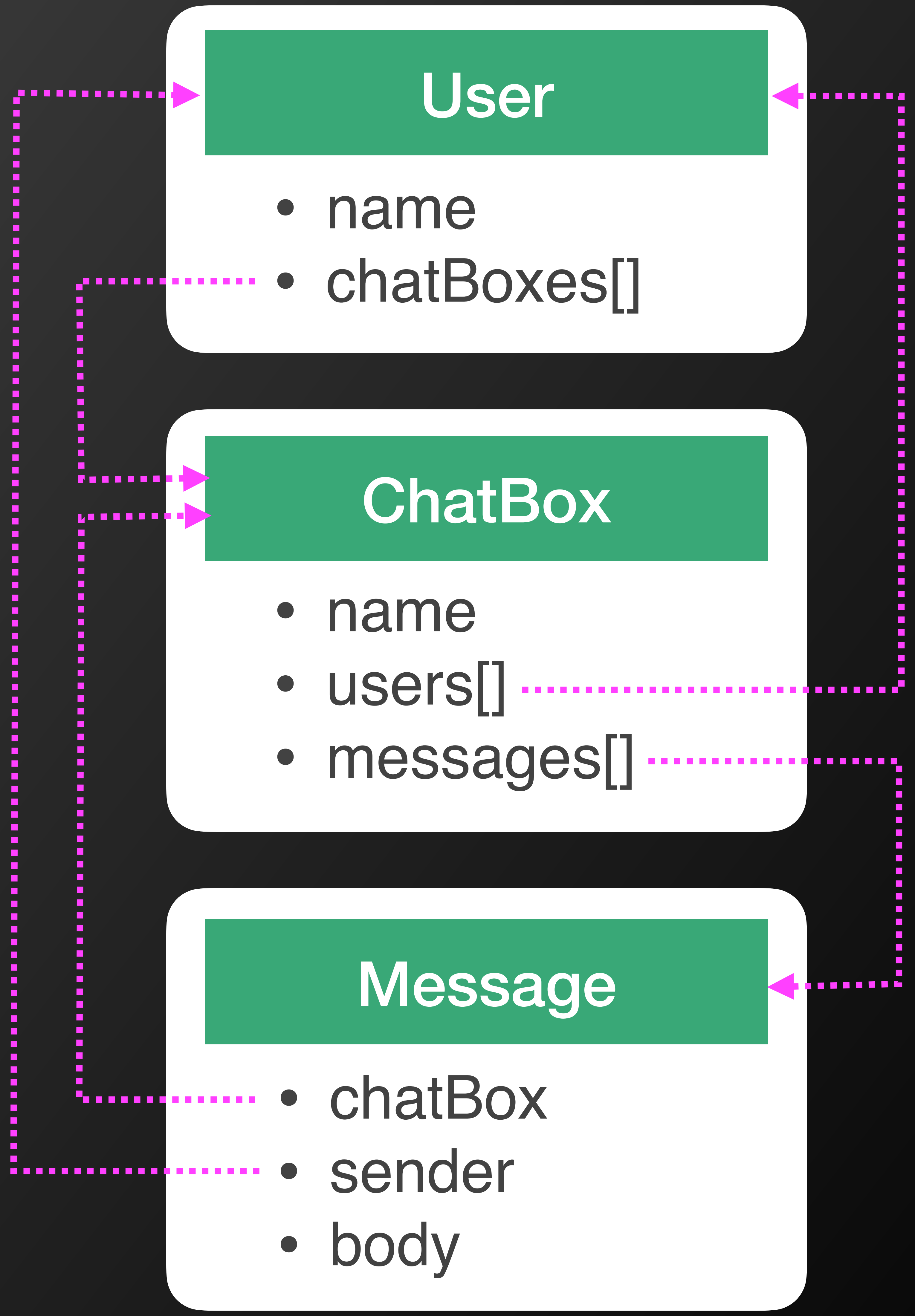
## Before we migrate HW#7 to GraphQL...

兩件事情先做：

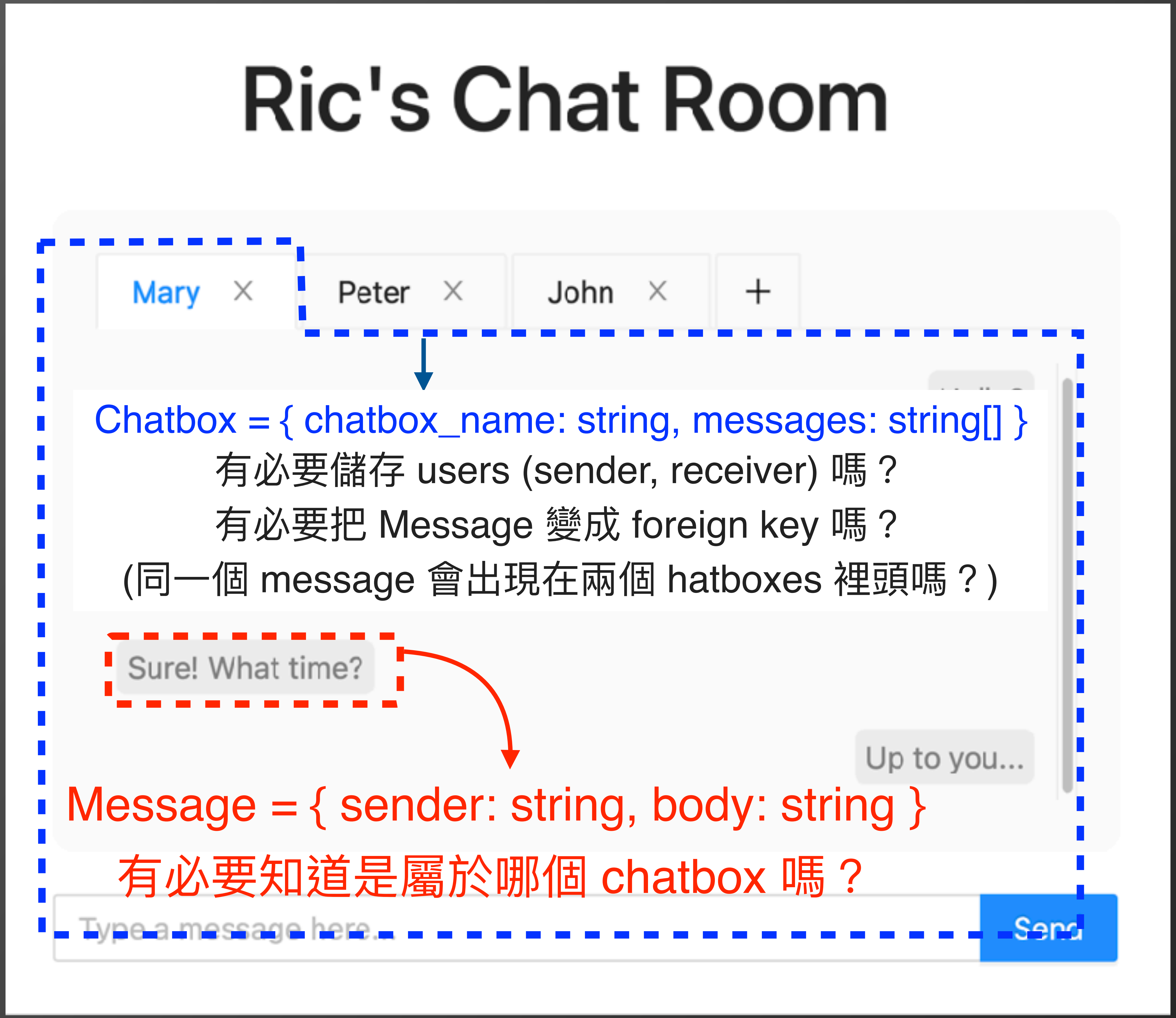
1. Optimize 一下 ChatRoom 的 DB Schema
2. 把 GraphQL 升級到新版

# Recall on ChatRoom DB





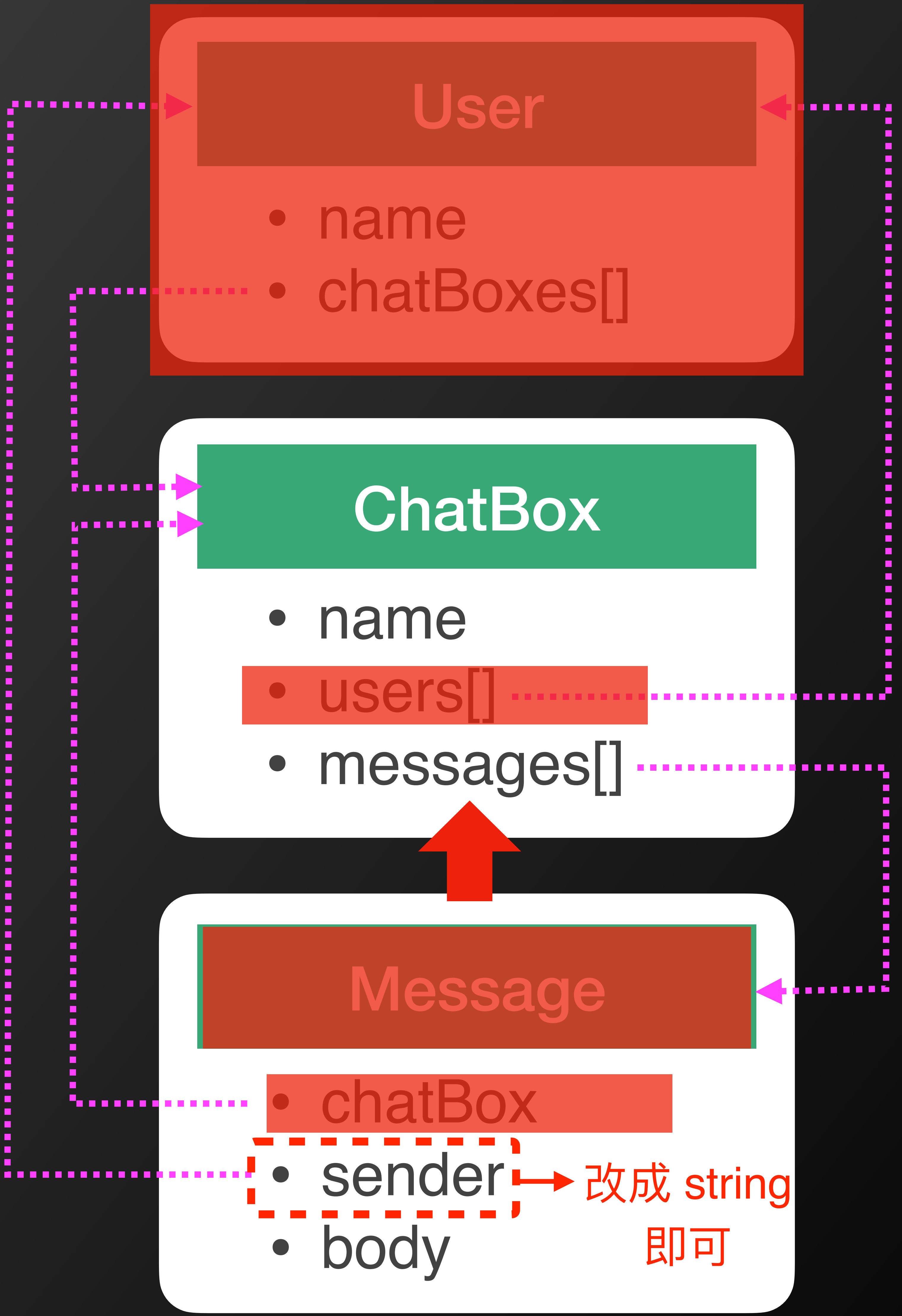
# Recall on ChatRoom DB



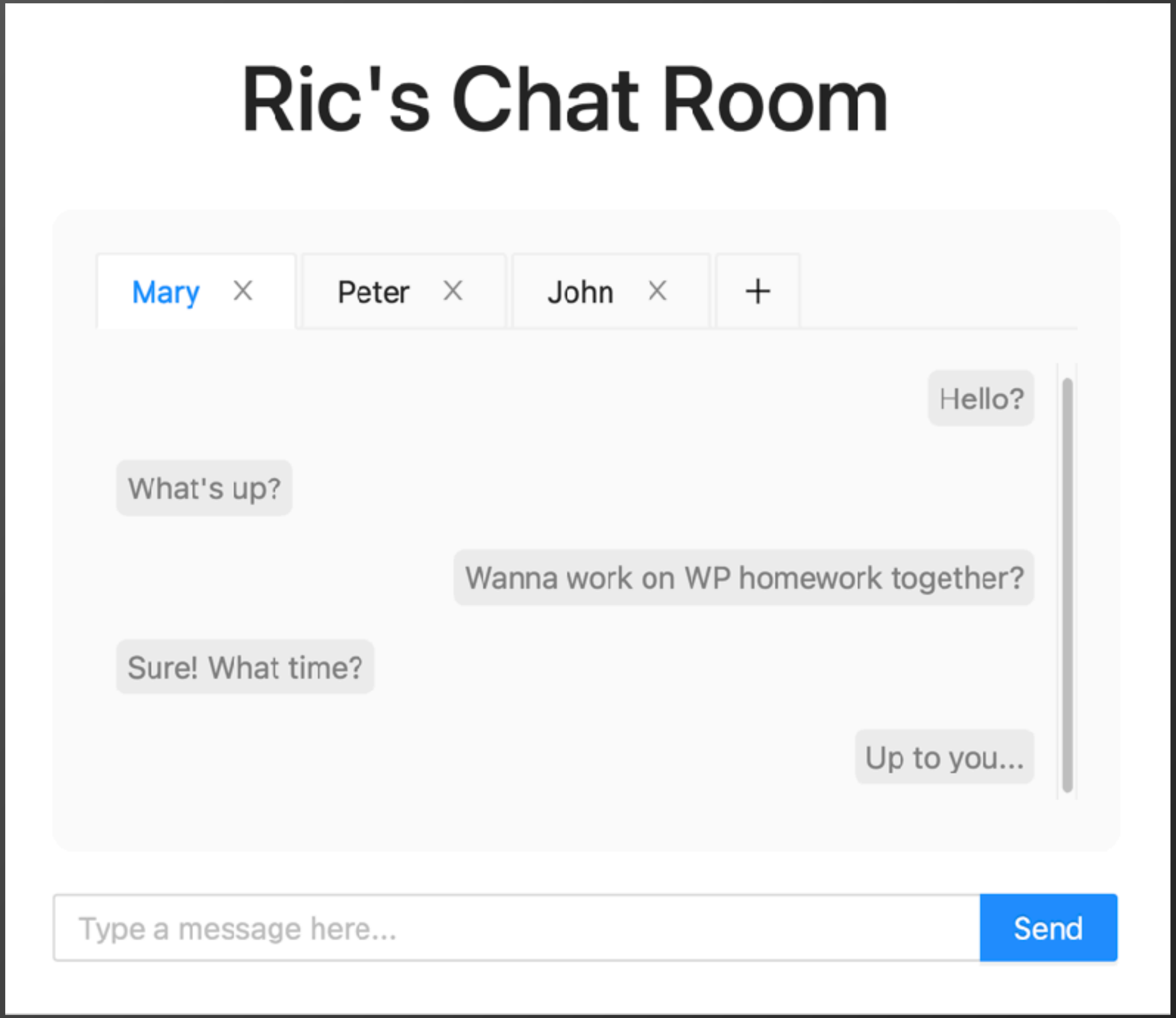
Ric's Chat Room

Chatbox = { chatbox_name: string, messages: string[] }
有必要儲存 users (sender, receiver) 嗎？
有必要把 Message 變成 foreign key 嗎？
(同一個 message 會出現在兩個 hatboxes 裡頭嗎？)

Message = { sender: string, body: string }
有必要知道是屬於哪個 chatbox 嗎？



## Simplified ChatRoom DB


Ric's Chat Room

Hello?

What's up?

Wanna work on WP homework together?

Sure! What time?

Up to you...

**ChatBox**

- name
- messages [{sender, body}]

## Let's see what happens if we update the GraphQL tutorial to the latest version

- In "backend/package.json", we have:

```
"dependencies": {
  "graphql-yoga": "^1.17.4"
}
```

- However, the latest graphql-yoga is ver. 3.1.1
- Let's reinstall the packages from scratch with the latest versions

```
> rm -rf package.json node_modules yarn.lock
> yarn init -y
> yarn add graphql-yoga
> yarn add -D @babel/cli @babel/core @babel/node @babel/
  plugin-proposal-class-properties @babel/plugin-
  proposal-object-rest-spread @babel/plugin-transform-
  arrow-functions @babel/preset-env nodemon
```

## Let's "yarn start" and see what happens...

- 記得要把 "scripts:" 加回來 "package.json"
- Error:

```
node:internal/modules/cjs/loader:998
  throw err;
  ^

Error: Cannot find module 'graphql'
Require stack:
- /Users/ric...
```

➡ 要 "yarn add graphql"

- Error:

```
Error: Cannot find module 'uuid/v4'
```

➡ 要 "yarn add -D uuidv4"
➡ 在 "src/resolver/Mutation.js", 要改成：

```
import {v4 as uuidv4} from 'uuid';
```

# 還是不 work...

- 請參考官方的說明/tutorial

1. 要使用 createPubSub, createSchema, createYoga

```
import { createPubSub, createSchema, createYoga } from 'graphql-yoga'

const pubsub = createPubSub();

const yoga = createYoga({
  schema: createSchema({
    typeDefs: ...,
    resolvers: {
      ...,
    },
  }),
  context: {
    db,
    pubsub,
  },
});
```

# 還是不 work...

- 請參考官方的說明/tutorial

2. 要使用 createServer

```
import { createServer } from 'node:http'
...
const server = createServer(yoga)
server.listen({ port: process.env.PORT | 5000 }, () => {
  console.log(`The server is up on port ${process.env.PORT | 5000}!`);
});
```

3. typeDefs: "schema.graphql" 不能直接用 string 指定

```
import * as fs from 'fs'
...
  schema: createSchema({
    typeDefs: fs.readFileSync(
      './src/schema.graphql',
      'utf-8'
    ),
```

## 還是不 work...

- 請參考官方的說明/tutorial

4. Default "graphql-yoga" 的 routing 改變了...

```
% http://localhost:5000/graphql
```

or

```
const yoga = createYoga(...),
  context: { ... },
  graphqlEndpoint: '/',
});
```

應該要可以 work 了！

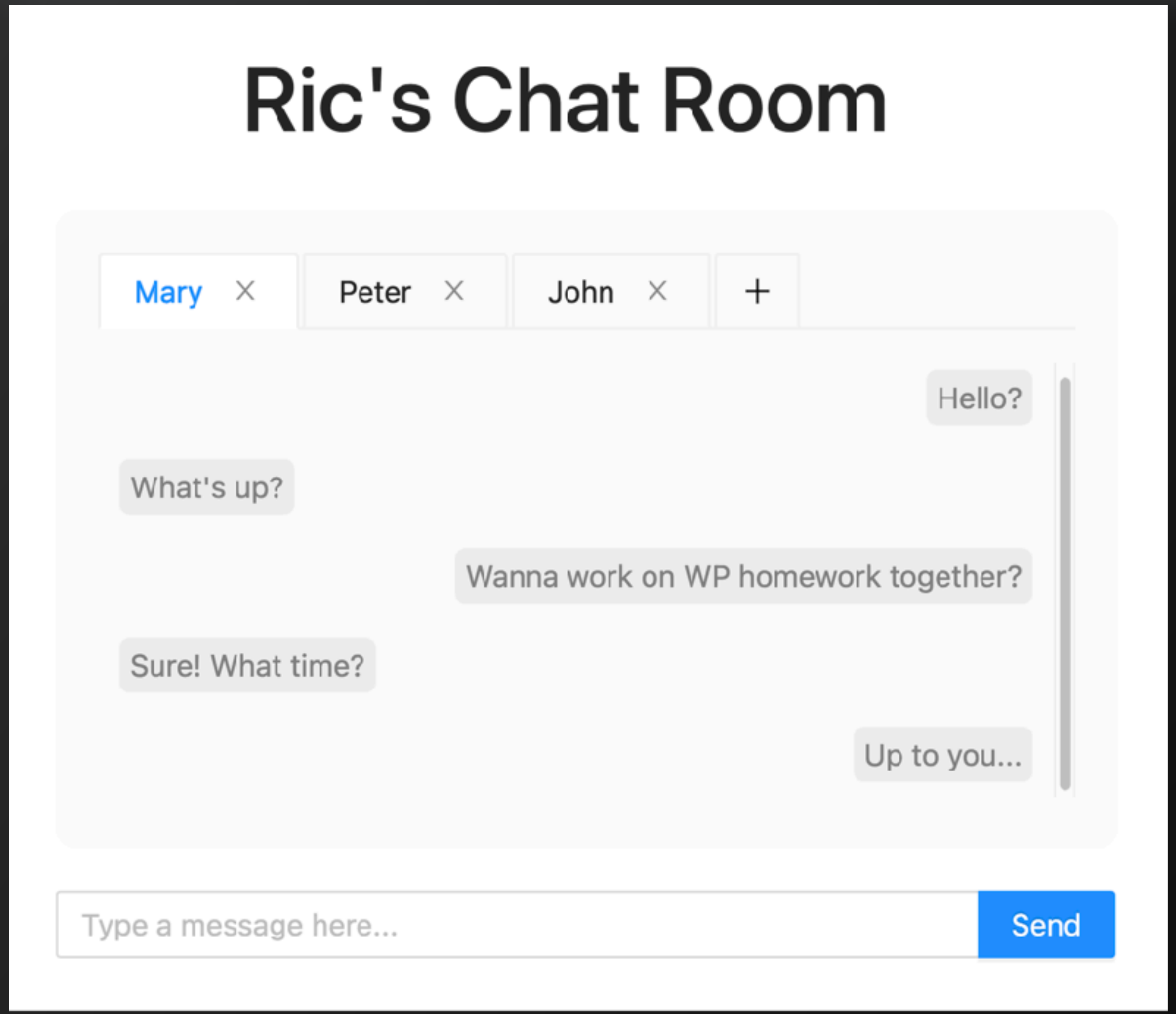
# 還記得 ChatRoom 的前後端嗎？

- 前端

```
public/
  index.html
src/
  index.js
  Containers/
    App.js
    SignIn.js
    ChatRoom.js
    hooks/
      useChat.js
  Components/
    ChatModal.js
    LogIn.js
    Message.js
    Title.js
```

- 後端

```
src/
  server.js
  mongo.js
  wsConnect.js
  models/
    chatbox.js
```


Ric's Chat Room

Mary ×   Peter ×   John ×   +

Hello?
What's up?
Wanna work on WP homework together?
Sure! What time?
Up to you...

要用 GraphQL 整合前後端

首先可以參考 "Modern GraphQL Tutorial"

先把他跟上頁的後端整合起來

把 GraphQL queries 寫好

先用 GraphQL-Yoga 測試

## Current ChatRoom vs. GraphQL Tutorial Backends

- GraphQL Tutorial Backend

```
src/
  index.js
  db.js
  resolvers/
    Query.js
    Mutation.js
    Subscription.js
    User.js
    Comment.js
  schema.graphql
```

- ChatRoom Backend

```
src/
  server.js
  mongo.js
  wsConnect.js
  models/
    chatbox.js
```

Things to DO —

1. Merge ChatRoom's "server.js" to GraphQL's "index.js"
2. 根據 "models/chatbox.js" MongoDB Schema, 定義 "schema.graphql"
3. 根據 "schema.graphql" 定義 resolvers
4. Test on GraphQL-Yoga!!
5. Revise schema.graphql for Mutation/ Subscription

## 1. Merge ChatRoom's "server.js" to GraphQL's "index.js"

- 準備工作：In GraphQL tutorial ---

a. 把原先的 "index.js" rename 成 "server.js"

b. 把 HW#7 的 "server.js" copy 過來再 rename 成 "index.js"

c. 最後再把 HW#7 的 "mongo.js" 以及 "models/chatbox.js" copy 過來
   => index.js: backend 的入口
   => server.js: 建立 GraphQL server
   => mongo.js: 建立 MongoDB 連線
   => models/chatbox.js: 定義 Mongo Schema

d. Copy ".env{.defaults}".

e. Yarn add mongoose. Yarn add -D dotenv-defaults.

## 1.a. 把原先的 "index.js" rename 成 "server.js"

- 把 "import db" 改成 "import ChatBoxModel"

```
import ChatBoxModel from './models/chatbox'
```

- 留下基本的 Query Chatbox resolvers
- 先把 pubsub comment out
- 直接 "export default server"
(而不是 "server.listen(...))

## 1.b. 把 HW#7 的 "server.js" copy 過來再 rename 成 "index.js"

- 把關於 http, express, mongoose, WebSocket… 的 code 都 comment out
- 需要的只有：

```
// import GraphQL server
import server from './server'

// import MongoDB connection
import mongo from './mongo'
mongo.connect();

const port = process.env.PORT || 4000;
server.listen({port}, () => {
  console.log(`Listening on http://localhost:${port}`);
});
```

## 1. Merge ChatRoom's "server.js" to GraphQL's "index.js"

- 準備工作：In GraphQL tutorial ---

a. 把原先的 "index.js" rename 成 "server.js"

b. 把 HW#7 的 "server.js" copy 過來再 rename 成 "index.js"

c. 最後再把 HW#7 的 "mongo.js" 以及 "models/chatbox.js" copy 過來
   => index.js: backend 的入口
   => server.js: 建立 GraphQL server
   => mongo.js: 建立 MongoDB 連線
   => models/chatbox.js: 定義 Mongo Schema

d. Copy ".env{.defaults}".

e. Yarn add mongoose. Yarn add -D dotenv-defaults.

## 2. 根據 "models/chatbox.js" MongoDB Schema, 定義 "schema.graphql"

- 定義一個簡單的 chatbox query

- models/chatbox.js

```
const ChatBoxSchema = new Schema({
  name: {
    type: String,
    required:
    [true, 'Name field is required.']
  },
  messages: [{
    sender: { type: String },
    body  : { type: String }, }],
});

const ChatBoxModel =
mongoose.model('ChatBox',
ChatBoxSchema);
```

- schema.graphql

```
type Query {
  chatbox(name: String!): ChatBox!
}

type ChatBox {
  name: String!
  messages : [Message!]
}

type Message {
  sender: String!
  body: String!
}
```

## 3. 根據 "schema.graphql" 定義 resolvers

- Query resolver

```
const Query = {
  chatbox: async (parent, { name }, { ChatBoxModel }) => {
    let box = await ChatBoxModel.findOne({ name });
    if (!box)
      box = await new ChatBoxModel({ name }).save();
    return box;
  },
};
export default Query;
```

- ChatBox resolver

```
const ChatBox = {
  messages: (parent) => (parent.messages),
};
export default ChatBox;
```

## 4. Test on GraphQL-Yoga!!

## 5. Revise schema.graphql for Mutation/Subscription

你需要哪些 queries?
什麼型態？

## 有哪些動作需要前端跟後端拿/修改資料？

- 一開始登入、從 SignIn 切換成 ChatRoom 畫面？

  **不用，這只是純前端的動作**

- 使用者按 ‘+’ 新增一個 ChatBox

  **需要 createChatBox, 拿回所有歷史對話記錄**

- 使用者新增一則對話

  **需要 createMessage, 拿回新增的對話字串，並通知所有相關的 chatboxes**

- 使用者按 ‘x’ 刪除一個 ChatBox

  **不用，這只是純前端的動作**

- 使用者按 ‘+’ 新增一個 ChatBox

  **需要 createChatBox, 拿回所有歷史對話記錄**

- 使用者新增一則對話

  **需要 createMessage, 拿回新增的對話字串，並通知所有相關的 chatboxes**

哪些是 query?

哪些是 mutation?

哪些是 subscription?

## 5. Revise schema.graphql for Mutation/Subscription

- 需要 createChatBox, 並拿回所有歷史對話記錄

```
// in "schema.graphql"
type Mutation {
  createChatBox(name1: String, name2: String): ChatBox!
}
```

```
// in "resolvers/mutation.js"
const Mutation = {
  createChatBox: (parent, { name1, name2 } ) => {
    return checkOutChatBox(name1, name2);
  },
};
```

## 5. Revise schema.graphql for Mutation/Subscription

- 需要 createMessage, 拿回新增的對話字串，並通知所有相關的 chatboxes

```
// in "schema.graphql"
type Mutation {
  createMessage
  (name: String!, to: String!, body: String!): Message!
}

type Subscription {
  message(from: String!, to: String!): Message!
}
```

## 5. Revise schema.graphql for Mutation/Subscription

- 需要 createMessage, 拿回新增的對話字串，並通知所有相關的 chatboxes

```
// in "resolvers/Mutation.js"
const Mutation = {
  createMessage: async (parent, { name, to, body }, { pubsub } )
  => {
    const chatBox = await checkOutChatBox(name, to);
    const newMsg = { sender: name, body };
    chatBox.messages.push(newMsg);
    await chatBox.save();

    const chatBoxName = makeName(name, to);
    pubsub.publish(`chatBox ${chatBoxName}`, {
      message: newMsg,
    });
    return newMsg;
  },
};
```

# 5. Revise schema.graphql for Mutation/Subscription

- 需要 createMessage, 拿回新增的對話字串，並通知所有相關的 chatboxes

```
// in "resolvers/Subscription.js"
const Subscription = {
  message: {
    subscribe: (parent, { from, to }, { pubsub }) => {
      const chatBoxName = makeName(from, to);
      return pubsub.subscribe(`chatBox ${chatBoxName}`);
    },
  },
};
```

## Again, Test on GraphQL-Yoga!!

## Test on GraphQL-Yoga!!

(每寫一個 query 就測一個，可以打開 MongoDB 後台看看資料有沒有寫正確)

Note: You may need to manually clear DB before testing

## 接下來把前端接起來...

- 首先，比較兩者的 package.json, 安裝下列套件：

  yarn add @apollo/client @apollo/react-hooks apollo-link apollo-link-ws apollo-utilities graphql react-apollo subscriptions-transport-ws

## 【WebSocket vs. GraphQL】

前端有什麼不同？

WebSocket 在後端用一個 dictionary { name, Set<ws> }
來記錄一個 chatbox name 所對應的所有 ws clients

GraphQL 則是利用其內建的 subscription 機制，
讓後端有訊息更新的時候，可以自動通知前端

## 先把 "src/index.js" 連上 Apollo service

```
import {
  ApolloClient, InMemoryCache, ApolloProvider
} from '@apollo/client';
import { ChatProvider } from "./containers/hooks/useChat"
import App from "./containers/App";
import reportWebVitals from "./reportWebVitals";

const client = new ApolloClient({
  uri: 'http://localhost:4000/graphql',
  cache: new InMemoryCache(),
});

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <ApolloProvider client={client}>
      <ChatProvider><App /></ChatProvider>
    </ApolloProvider>
  </React.StrictMode>
);
```

# 先寫好 gpl query strings

- graphql/mutations.js

```
import { gql } from '@apollo/client';

export const CREATE_CHATBOX_MUTATION = gql`
  mutation createChatBox($name1: String!, $name2: String!) {
    createChatBox(name1: $name1, name2: $name2) {
      name
      messages {
        sender
        body
      }
    }
  }
`;
```

- 同樣寫好 graphql/queries.js 以及 graphql/subscriptions.js, 然後一起 import 到 graphql/index.js

## 把 "useChat" 跟後端存取資料機制改成 GraphQL

- 把 Web Socket 機制都取代掉

```
import { useQuery, useMutation } from "@apollo/client";
import { CHATBOX_QUERY, CREATE_CHATBOX_MUTATION,
         MESSAGE_SUBSCRIPTION } from "../../graphql";
...
const ChatProvider = (props) => {
  // define states
  const { data, loading, subscribeToMore }
  = useQuery(CHATBOX_QUERY, {
    variables: {
      name1: me,
      name2: friend,
    },
  });

  const [startChat] = useMutation(CREATE_CHATBOX_MUTATION);
  ...
```

## 把 "useChat" 跟後端存取資料機制改成 GraphQL

```
useEffect(() => {
    try {
      subscribeToMore({
        document: MESSAGE_SUBSCRIPTION,
        variables: { from: me, to: friend },
        updateQuery: (prev, { subscriptionData }) => {
          if (!subscriptionData.data) return prev;
          const newMessage = subscriptionData.data.message.message;
          return {
            chatBox: {
              messages: [...prev.chatBox.messages, newMessage],
            },
          };
        },
      });
    } catch (e) {}
 }, [subscribeToMore]);
  ...
};
```

## Recall: 就這兩個動作需要前後端溝通

- 使用者按 ‘+’ 新增一個 ChatBox

**需要 createChatBox, 拿回所有歷史對話記錄**

```
const [startChat] =
       useMutation(CREATE_CHATBOX_MUTATION);
```

```
<ChatModal
  visible={modalVisible}
  onCreate={async ({ name }) => {
            await startChat({
              variables: { … },
            });
  ...
```

## Recall: 就這兩個動作需要前後端溝通

- 使用者新增一則對話

**需要 createMessage, 拿回新增的對話字串**

```
const [sendMessage] =
       useMutation(CREATE_MESSAGE_MUTATION);
```

```
<Input.Search
  onSearch={(msg) => {
    ...
    sendMessage({ variables: { … } });
    ...
  }}
</Input.Search>
```

希望大家都有順利練習到！

下下星期 Hack#3 100 分！

然後期末專題都有成功做出來！

感謝聆聽 ！

Ric Huang / NTUEE

(EE 3035) Web Programming