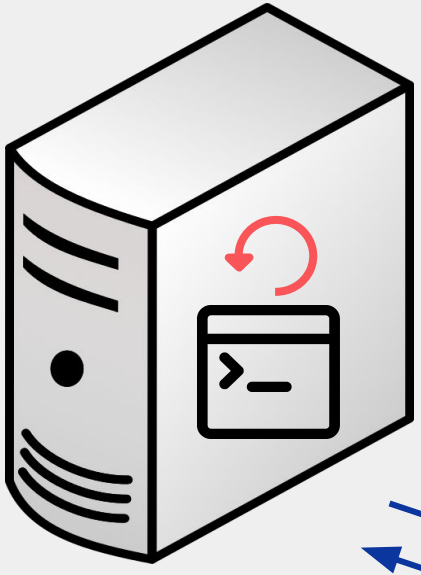
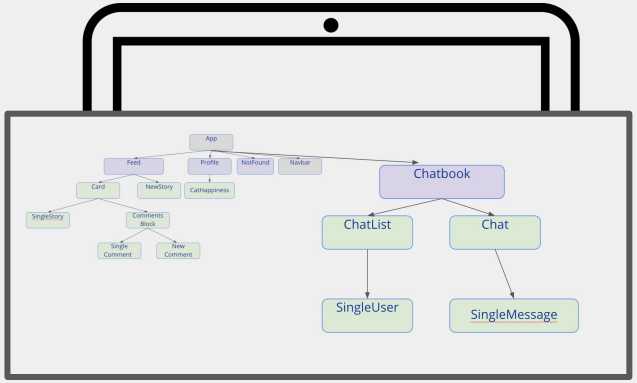
Recap



server

GET /api/chat, ALL_CHAT



client

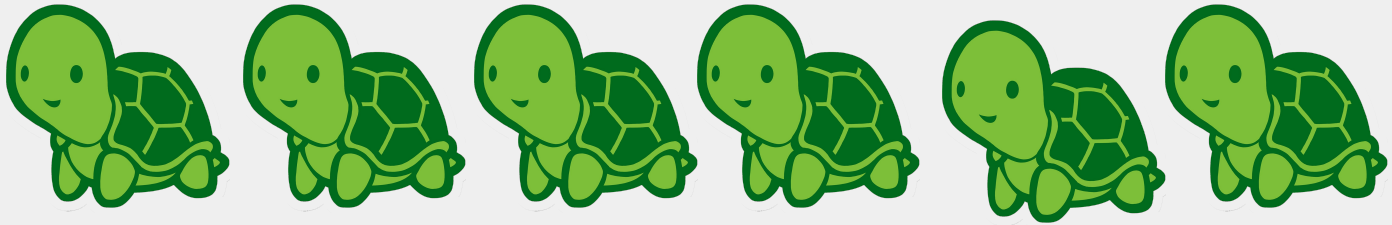
ALL_CHAT:
[...]
Alex: “i love aaron sippy cup

They want ALL_CHAT

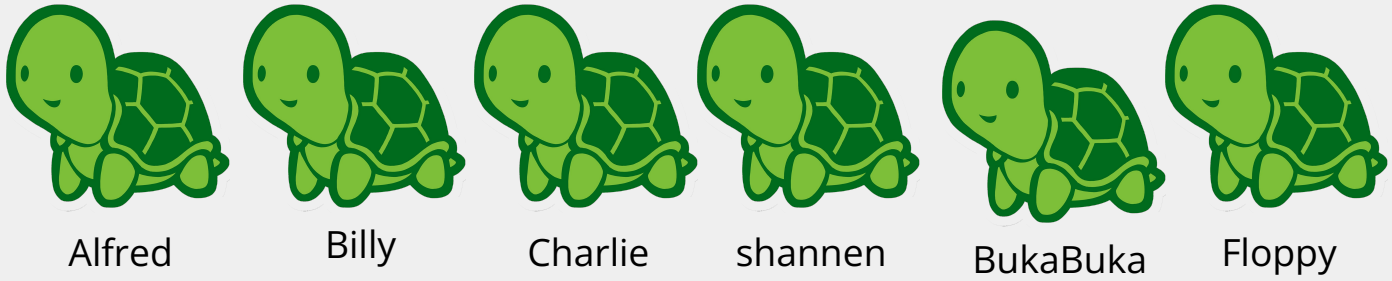
MongoDB

ALL_CHAT:
[...]
Alex: “i love aaron sippy cup

map



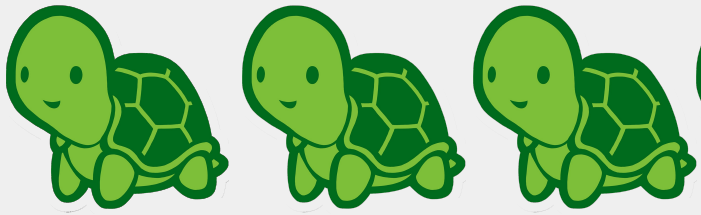
map



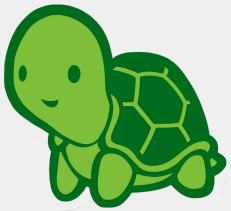

```
turtles.map(t =>
putHatOnMyTurtle(t))
```

map

```
putHatOnMyTurtle = (turtle) => {
  if (turtle.startsWith("B")) {
    return turtle.hatted()
  } else {
    return turtle
  }
}
```

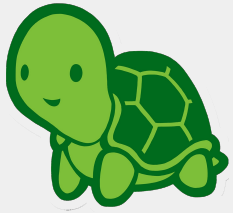


map

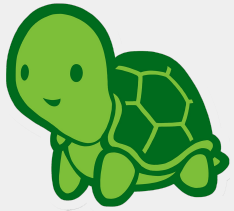


Alfred

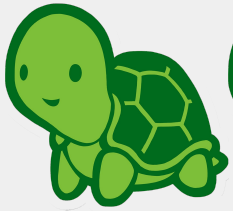
```
putHatOnMyTurtle = (turtle) => {
  if (turtle.startsWith("B")) {
    return turtle.hatted()
  } else {
    return turtle
  }
}
```



Billy

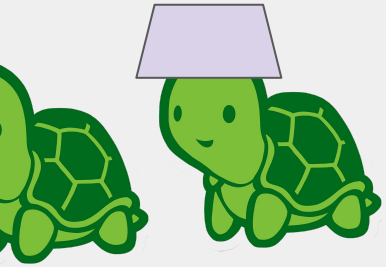


Charlie



shannen

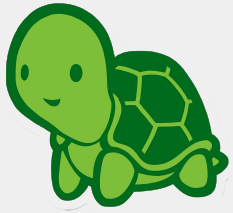
map



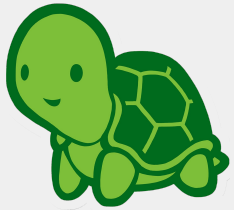
Alfred

Billy

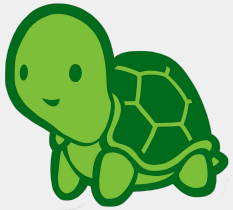
```
putHatOnMyTurtle = (turtle) => {
  if (turtle.startsWith("B")) {
    return turtle.hatted()
  } else {
    return turtle
  }
}
```



Charlie

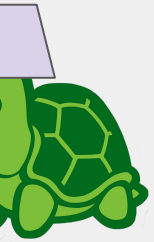


shannen

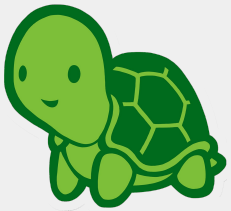


BukaBuka

map

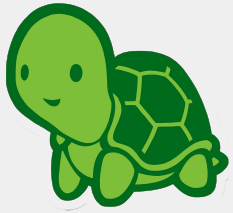


Billy

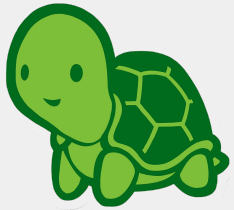


Charlie

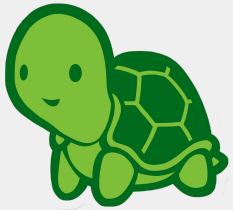
```
putHatOnMyTurtle = (turtle) => {
  if (turtle.startsWith("B")) {
    return turtle.hatted()
  } else {
    return turtle
  }
}
```



shannen



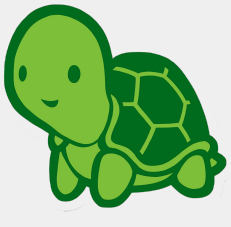
BukaBuka



Floppy

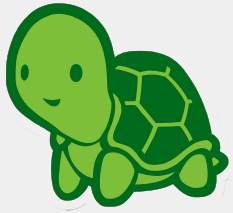
map

Charlie

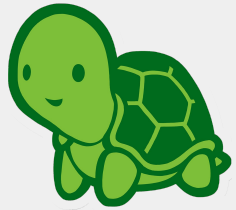


shannen

```
putHatOnMyTurtle = (turtle) => {
  if (turtle.startsWith("B")) {
    return turtle.hatted()
  } else {
    return turtle
  }
}
```

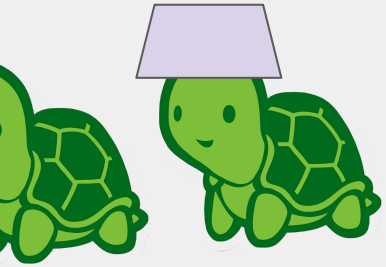


BukaBuka



Floppy

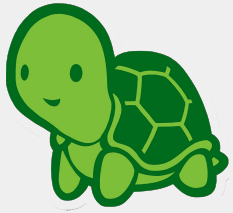
map



hannen

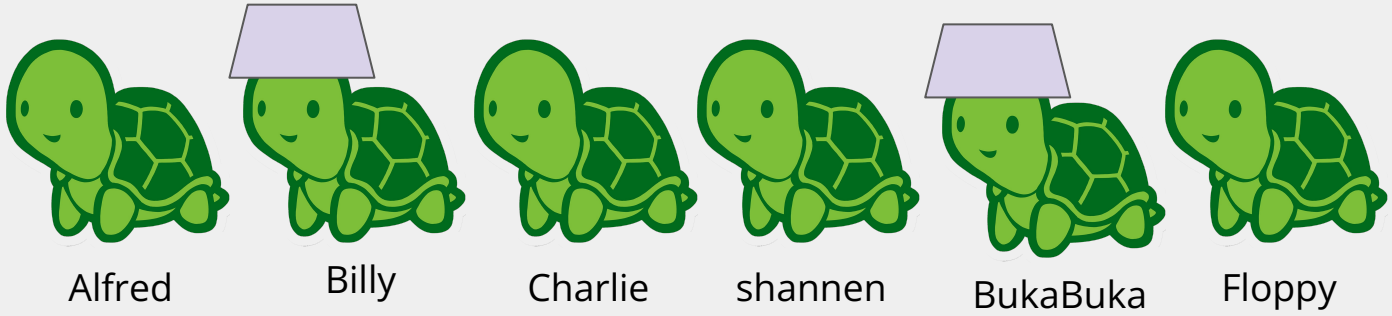
BukaBuka

```
putHatOnMyTurtle = (turtle) => {
  if (turtle.startsWith("B")) {
    return turtle.hatted()
  } else {
    return turtle
  }
}
```



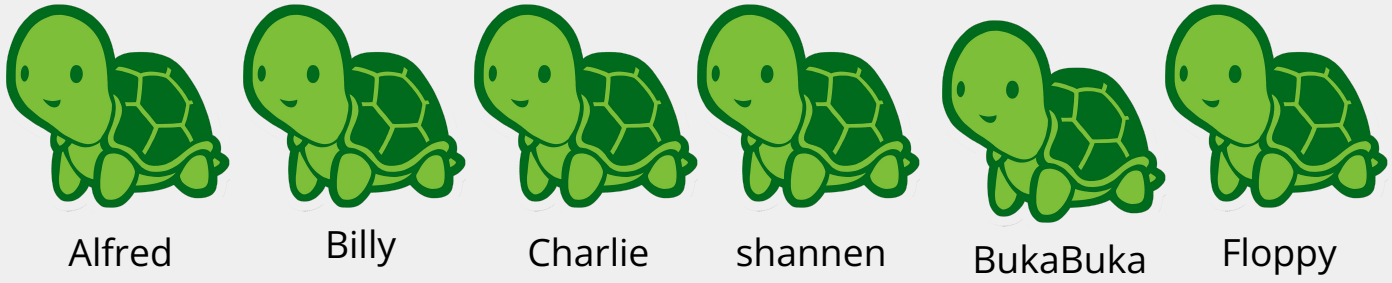
Floppy

map




```
turtles.map(t =>
putHatOnMyTurtle(t))
```

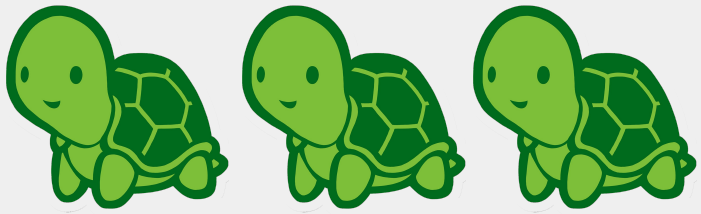
filter



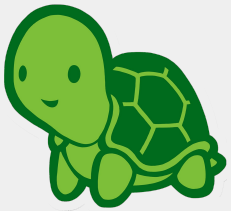

```
turtles
    .filter(t => t.name!== “Billy”)
```

filter

```
(t) => t.name !== “Billy”
```

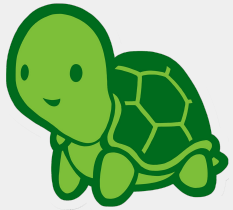


filter

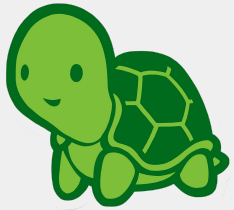


Alfred

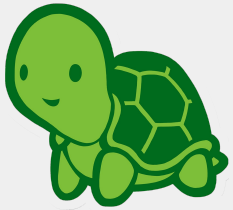
```
(t) => t.name !== “Billy”
```



Billy



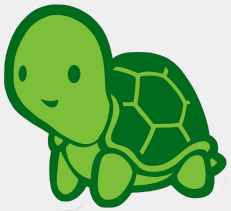
Charlie



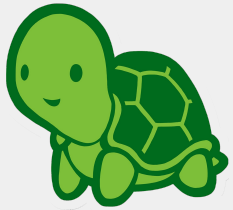
shannen

filter

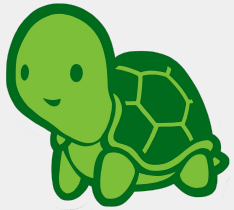

```
(t) => t.name !== "Billy"
```



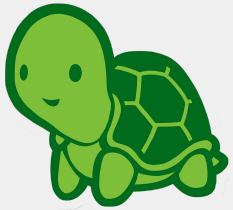
Alfred



Billy

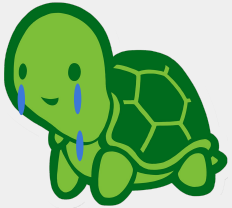


Charlie



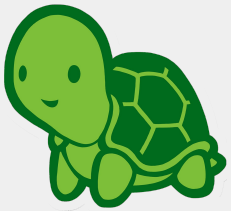
shannen

filter

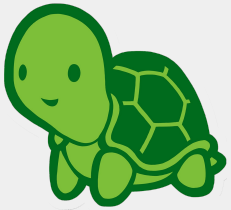


Billy

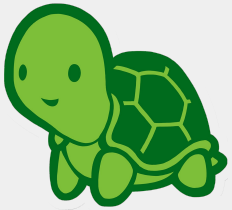
```
(t) => t.name !== “Billy”
```



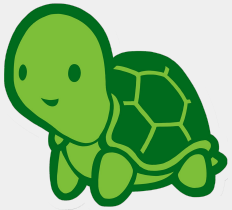
Alfred



Charlie

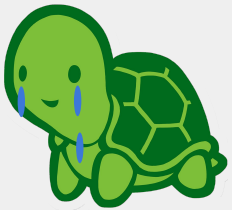


shannen



BukaBuka

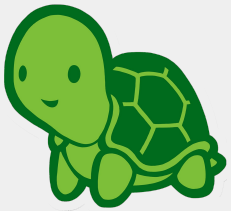
filter



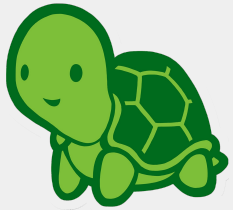
Billy

```
(t) => t.name !== "Billy"
```

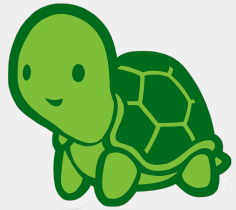
Alfred



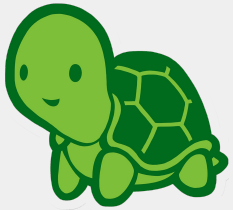
Charlie



shannen

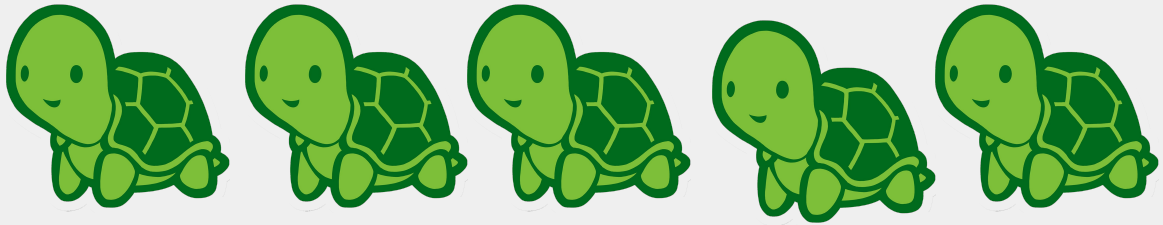


BukaBuka



Floppy

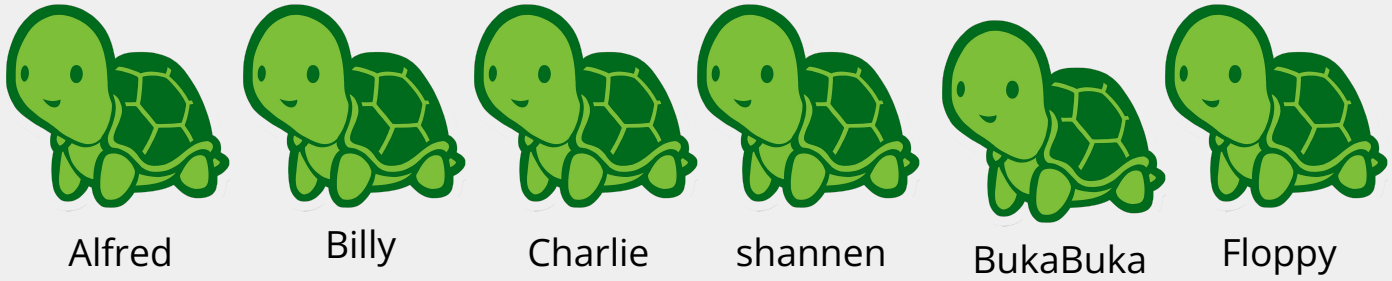
filter




```
turtles
    .filter(t => return t.name!== “Billy”)
```

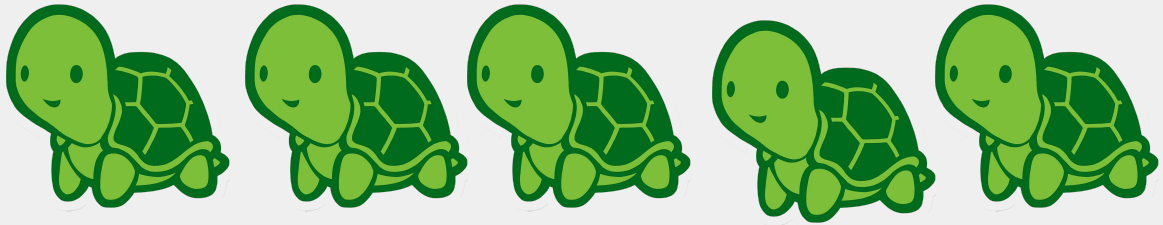
filter + map

filter + map



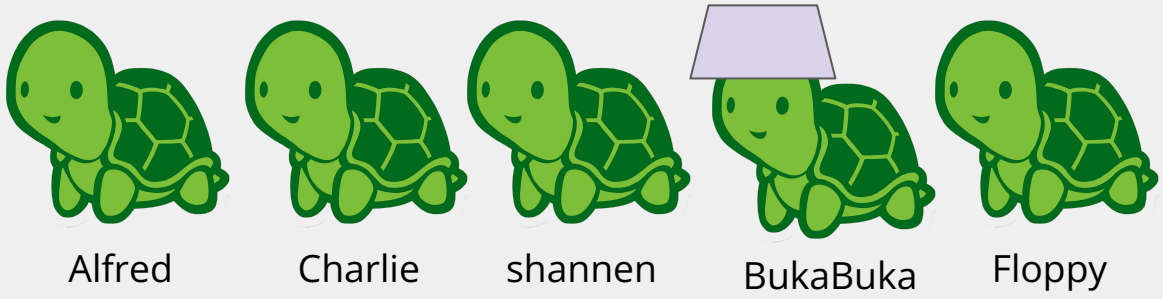

```
turtles
     .filter(t => t !== “Billy”)
     .map(t =>
putHatOnMyTurtle(t))
```

filter + map




```
turtles
     .filter(t => t !== “Billy”)
     .map(t =>
putHatOnMyTurtle(t))
```

filter + map

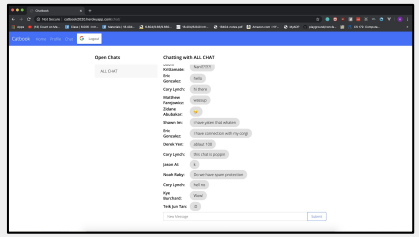



```
turtles
    .filter(t => t !== “Billy”)
    .map(t => putHatOnMyTurtle(t))
```

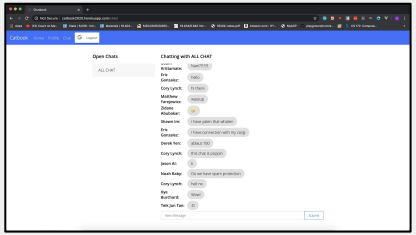
Chatbook! (part 2)

Main Slides: weblab.to/w9-slides
Side Slides: weblab.to/w9-context

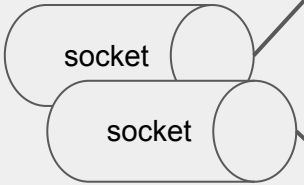
Recap

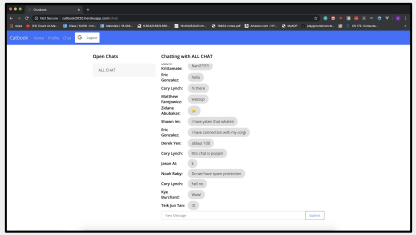


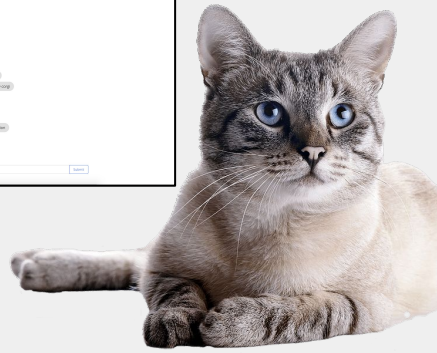
Whiskers



Fluffy

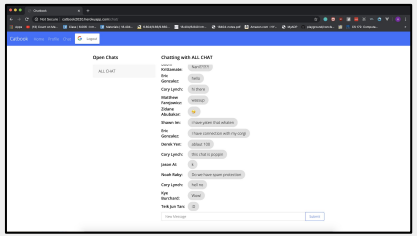






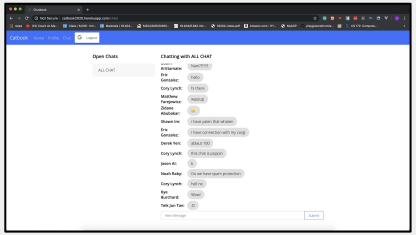
Jeff

Recap



socket

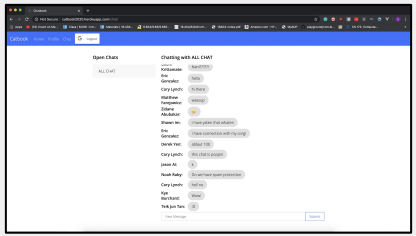
Whiskers

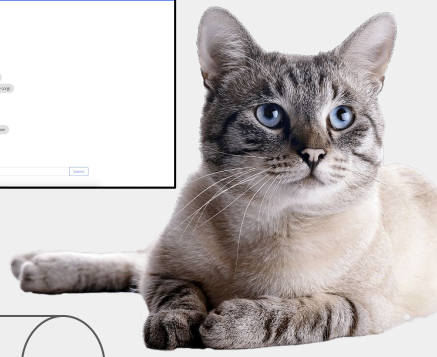


Fluffy

socket

socket

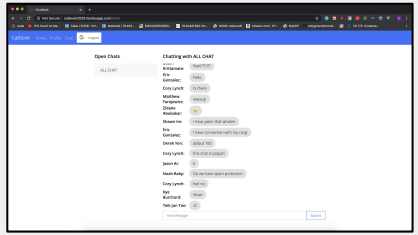




socket

Jeff

Recap

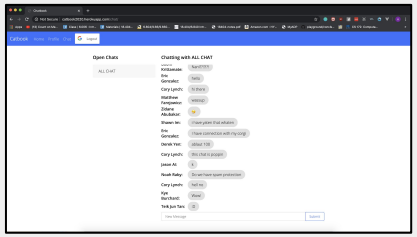


Fluffy

socket

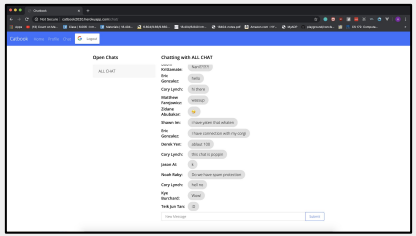
socket

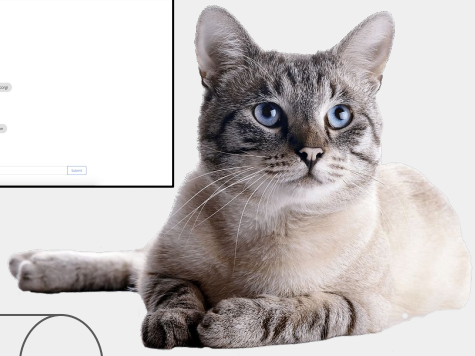
socketManager.getIo()



socket

Whiskers

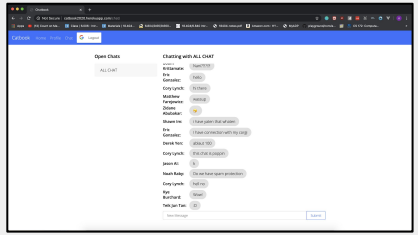




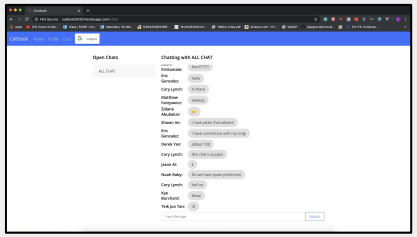
socket

Jeff

Recap



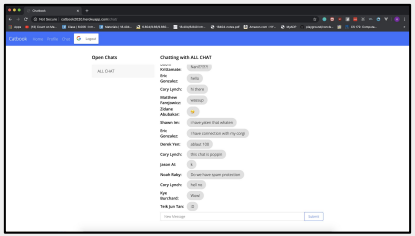
Fluffy



Whiskers

socket

client-socket.js

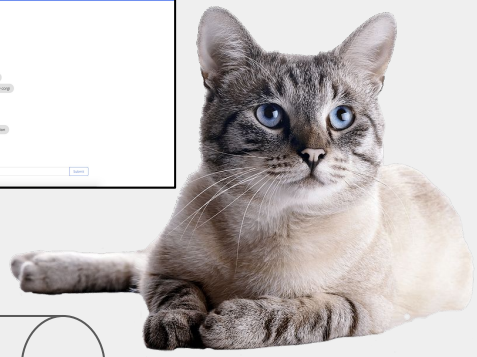


socket

socket

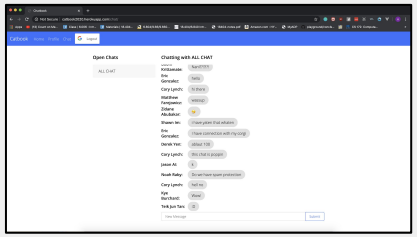
server-socket.js

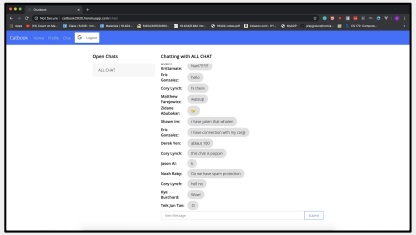
socket



Jeff

Recap



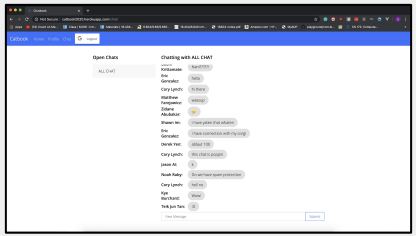


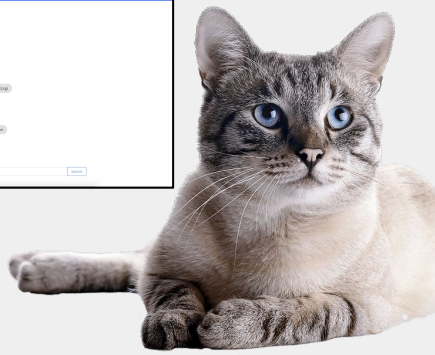
```
socket.on(“meow”, msg => sayBack(msg))
```

Whiskers

Fluffy

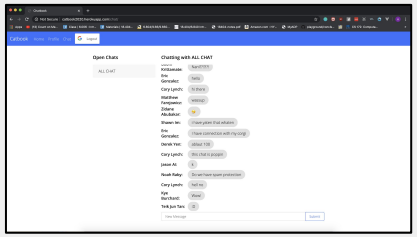
```
socket.on(“meow”, msg => sayBack(msg))
```



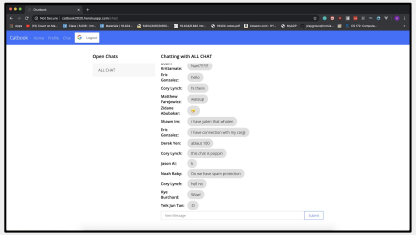


Jeff

Recap

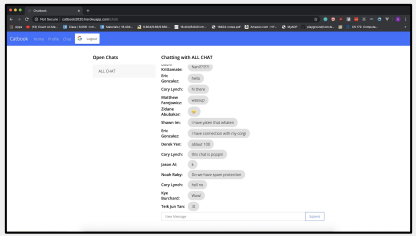


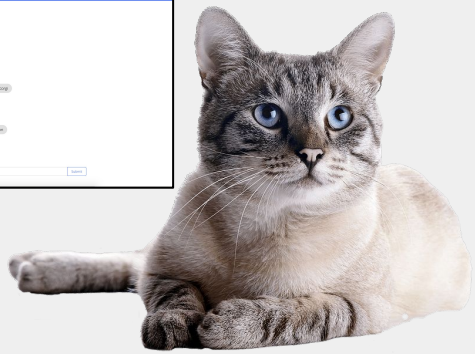
Whiskers



Fluffy

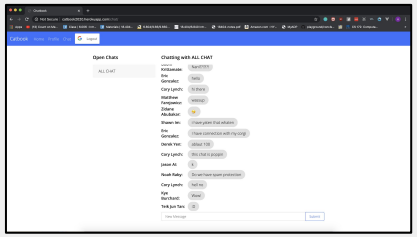
POST /api/meow
“i crave violence”



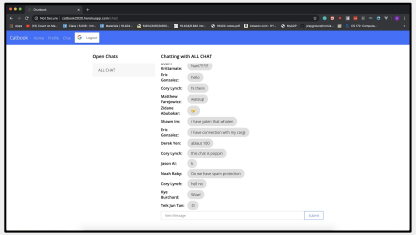


Jeff

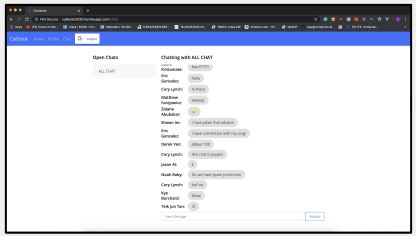
Recap



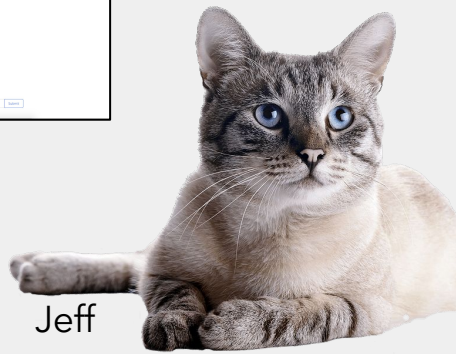
Whiskers



Fluffy

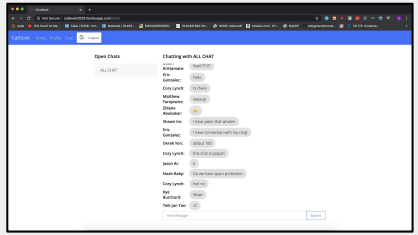


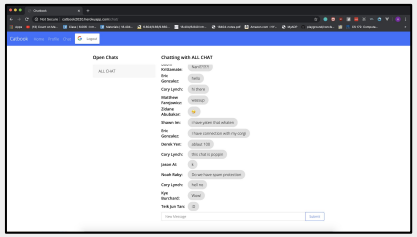
```
socketManager.getIo()
  .emit("meow", "i crave violence")
```



Jeff

Recap



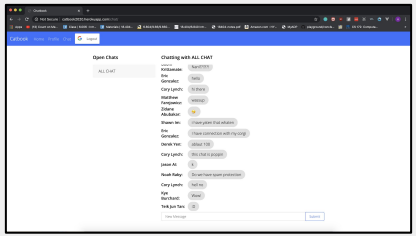


sayBack(“i crave violence”)

Whiskers

Fluffy

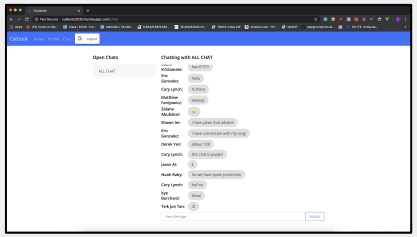
sayBack(“i crave violence”)



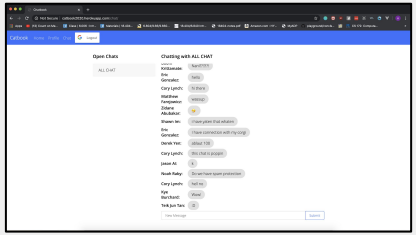
Jeff

But what about sending DMs? 🤔

The problem of identity

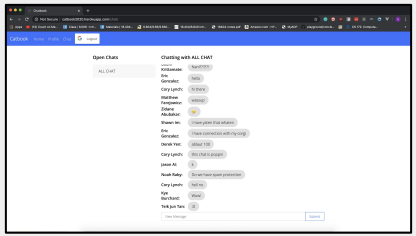


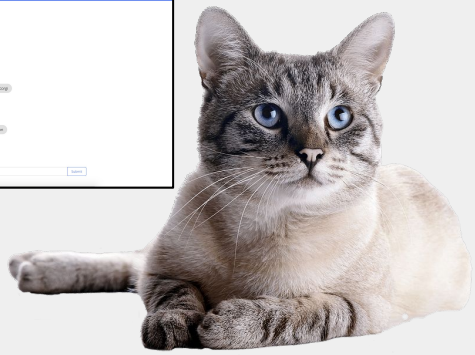
Whiskers



Fluffy

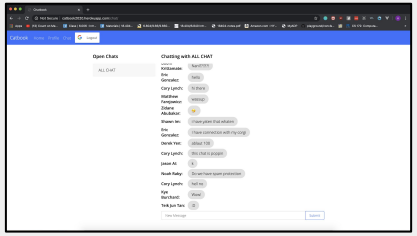
POST /api/meow “FOOD”
To Whiskers

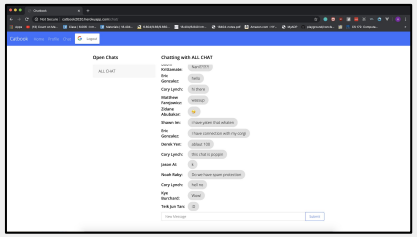




Jeff

The problem of identity

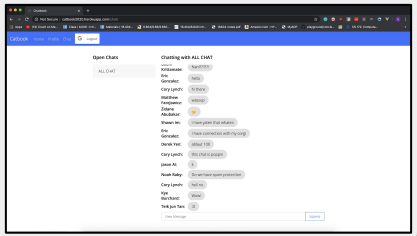


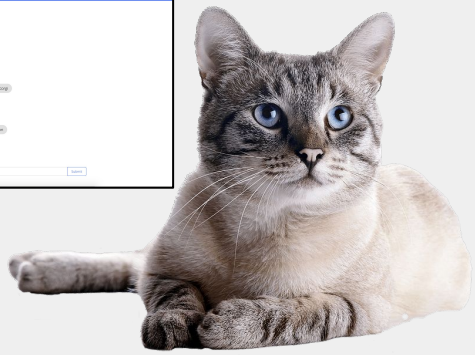


Whiskers

Fluffy

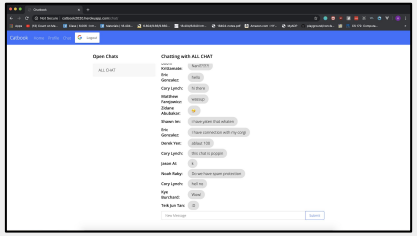
new meow “FOOD”
(ws)

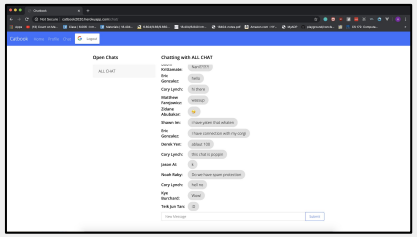




Jeff

The problem of identity



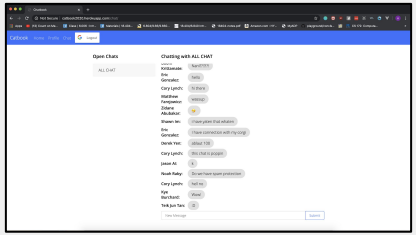


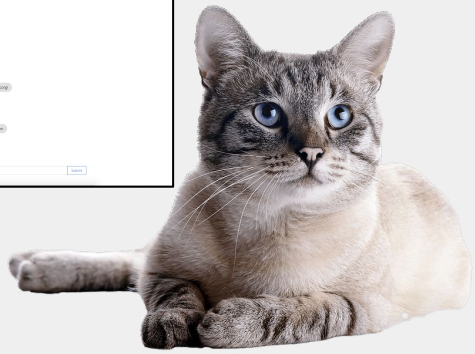
Whiskers

Fluffy

POST /api/meow “FOOD”
To Whiskers

new meow “FOOD”
(ws)





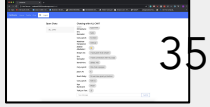
Jeff

io.emit alone is **public** to all sockets, so we need more!

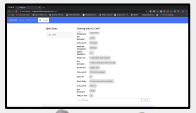
io.emit behind the scenes

Fluffy

POST /api/meow “FOOD”
To Whiskers



Whiskers



Jeff

```
socketManager.getIo().emit(“meow”, “FOOD”)
(ws)
```

io.emit behind the scenes

Fluffy

POST /api/meow “FOOD”
To Whiskers

Whiskers’ socket

Jeff’s socket

Whiskers

Jeff

io.emit behind the scenes

Fluffy

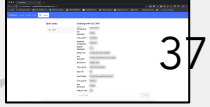
POST /api/meow “FOOD”
To Whiskers

socketManager.getIo()

IO Object

socket

socket



Whiskers

Jeff

io.emit behind the scenes

Fluffy

POST /api/meow “FOOD”
To Whiskers

IO Object

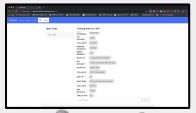
socket1

socket2

socket1.emit(“meow”, “FOOD”)

socket2.emit(“meow”, “FOOD”)

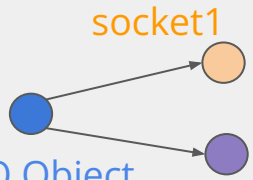
Whiskers

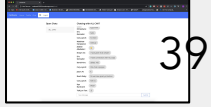


Jeff

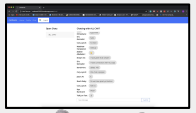
Problem: server doesn't know Whiskers' socket

Fluffy





Whiskers



Jeff

But Whiskers knows its socket!

Fluffy

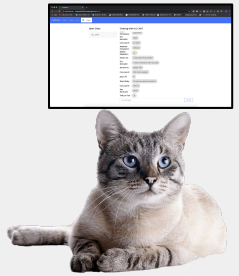
socket1

IO Object

socket2

(on initial connection)
Btw you are connected to
socket1

Whiskers



Jeff

But Whiskers knows its socket!

Fluffy

POST /api/initsocket hi it's whiskers, my socket is socket1

Whiskers

socket1

IO Object

socket2

(on initial connection) Btw you are connected to socket1

Jeff

Tell server which socket the user is

Fluffy

```
POST /api/initsocket hi it's whiskers,
my socket is socket1
```

Whiskers

socket1

IO Object

socket2

```
(on initial connection)
Btw you are connected to
socket1
```

Server can check with **req.user**!

Jeff

Tell server which socket the user is

Whiskers

Fluffy

```
socketManager.addUser(whiskers, socket1)
```

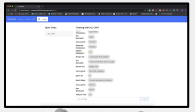
socket1

socket2

IO Object

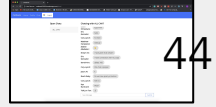
```
(on initial connection)
Btw you are connected to
socket1
```

Server can check with **req.user**!



Jeff

Tell server which socket the user is



Whiskers

Fluffy

```
socketManager.addUser(whiskers, socket1)
```

Whiskers' socket

```
(on initial connection)
Btw you are connected to
socket1
```

IO Object

Jeff's socket

Server can check with **req.user**!

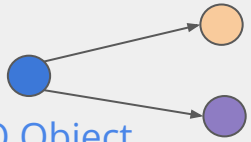
Jeff

getSocketFromUserID

Fluffy

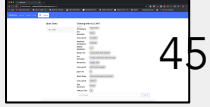
POST /api/meow “FOOD”
To Whiskers

Whiskers’ socket

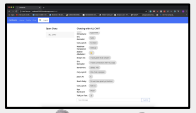


IO Object

Jeff’s socket



Whiskers



Jeff

We can now use **socketManager.getSocketFromUserID** to get a user’s socket!
(this function is given to you by staff)

getSocketFromUserID

Fluffy

POST /api/meow “FOOD”
To Whiskers

IO Object

Whiskers’ socket

Jeff’s socket

```
socketManager
.getSocketFromUserID(“whiskers_id”)
.emit(“meow”, “FOOD”)
```

Whiskers

Jeff

We can now use **socketManager.getSocketFromUserID** to get a user’s socket!
(this function is given to you by staff)

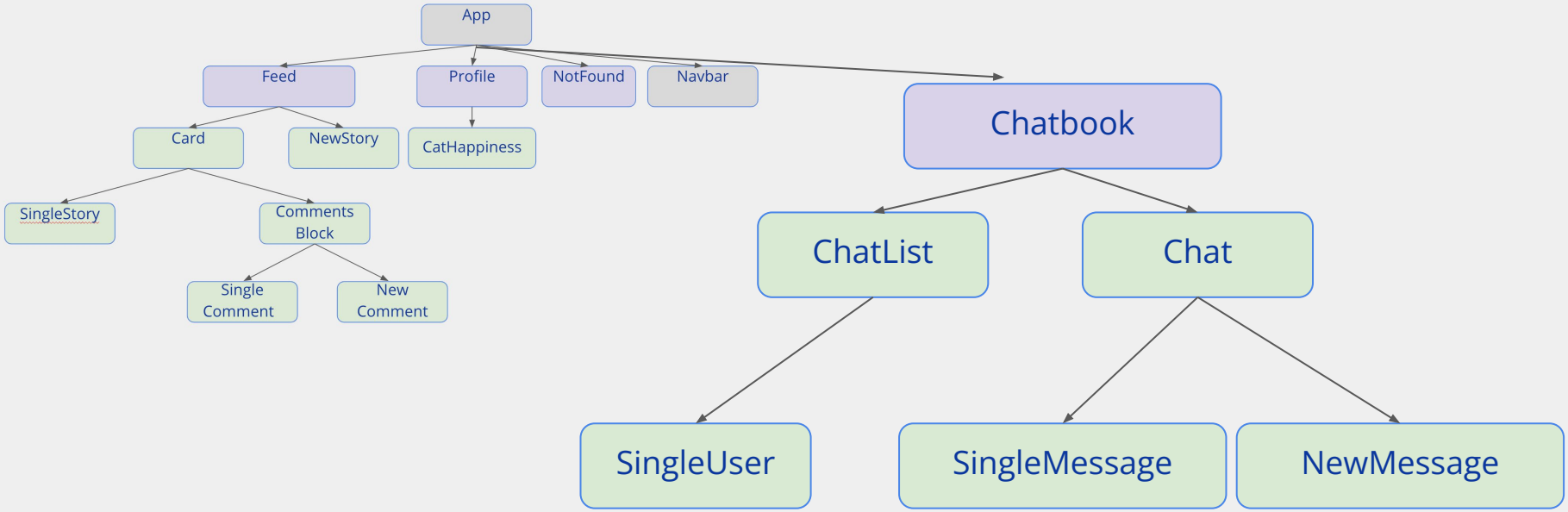
weblab.to/chatbook-docs
weblab.to/socket-guide

YAY! We now have everything we need for DMs

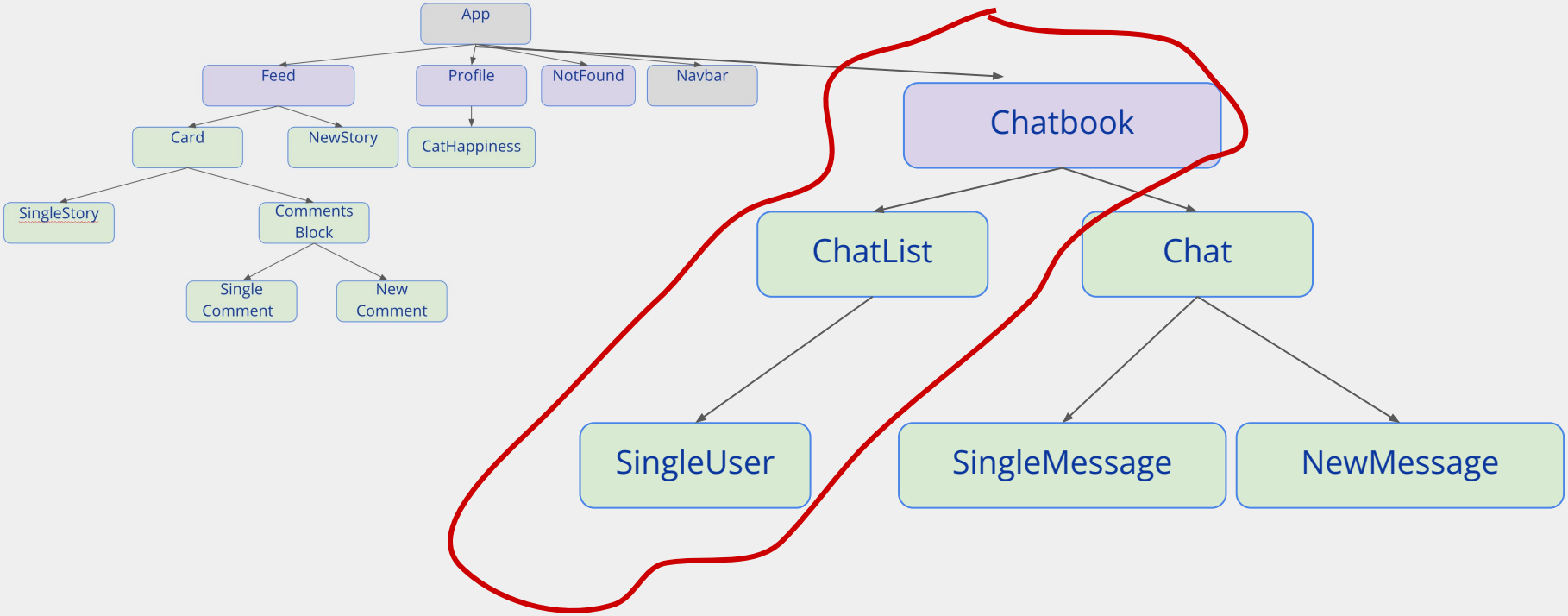
Let’s talk about the plan ^___^

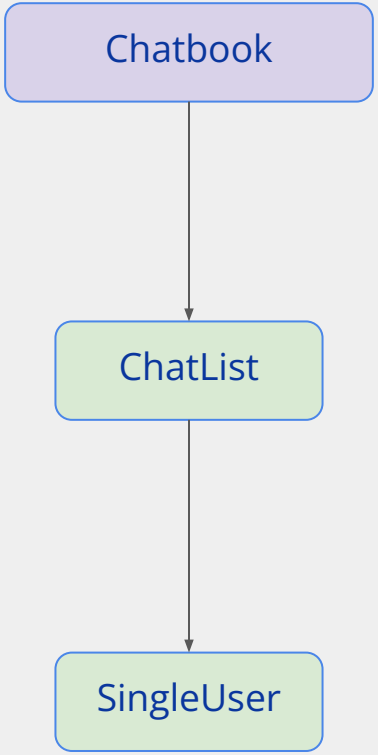
weblab.to/example

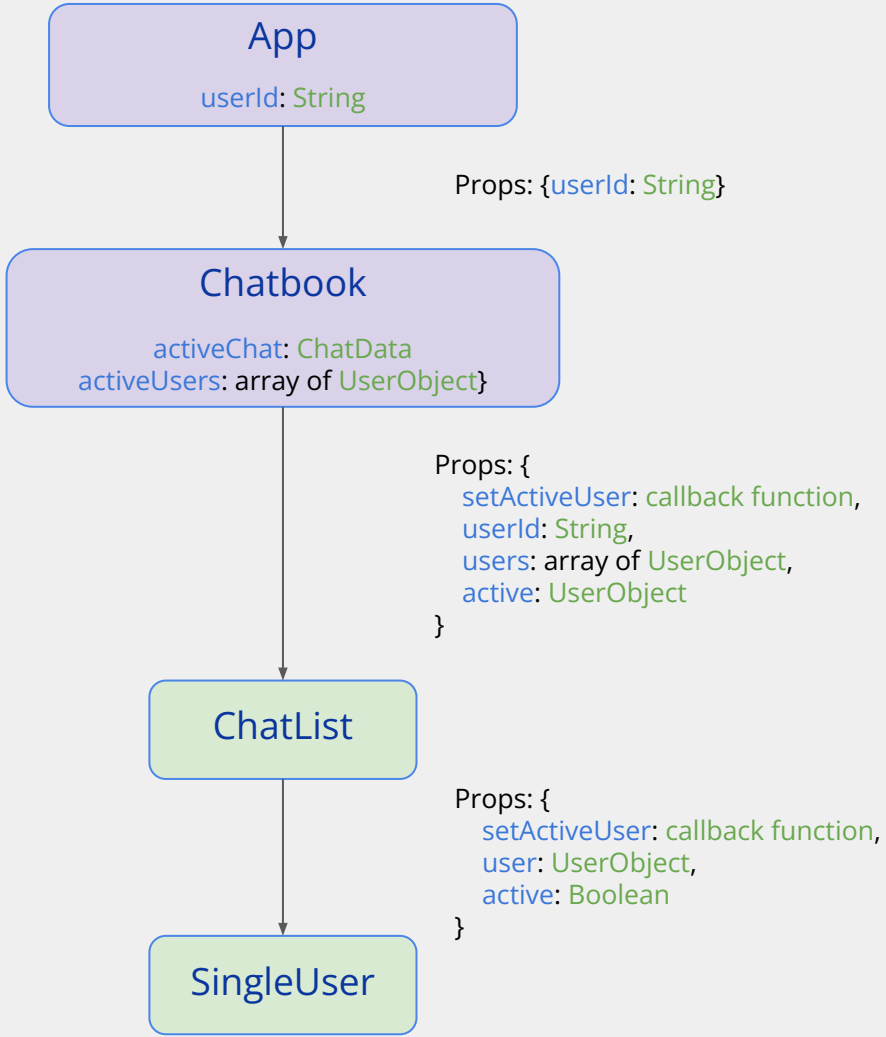
Chatbook Component Tree



Chatbook Component Tree




Chatbook
ChatList
SingleUser


App
userId: String
Props: {userId: String}
Chatbook
activeChat: ChatData
activeUsers: array of UserObject}
Props: {
setActiveUser: callback function,
userId: String,
users: array of UserObject,
active: UserObject
}
ChatList
Props: {
setActiveUser: callback function,
user: UserObject,
active: Boolean
}
SingleUser

App

userId: String

Props: {userId: String}

Chatbook

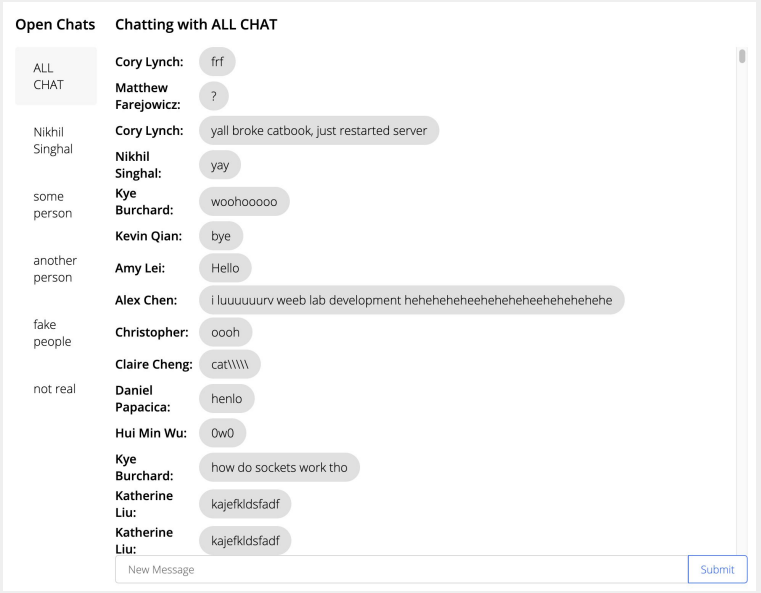
activeChat: ChatData
activeUsers: array of UserObject}

Props: {
setActiveUser: callback function,
userId: String,
users: array of UserObject,
active: UserObject
}

ChatList

Props: {
setActiveUser: callback function,
user: UserObject,
active: Boolean
}

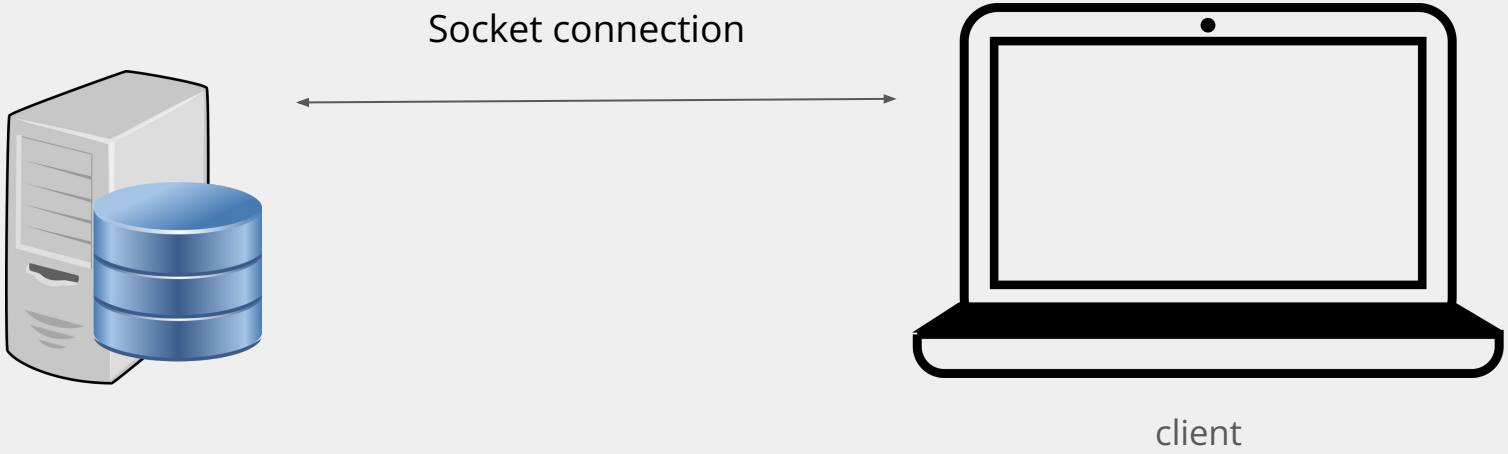
SingleUser

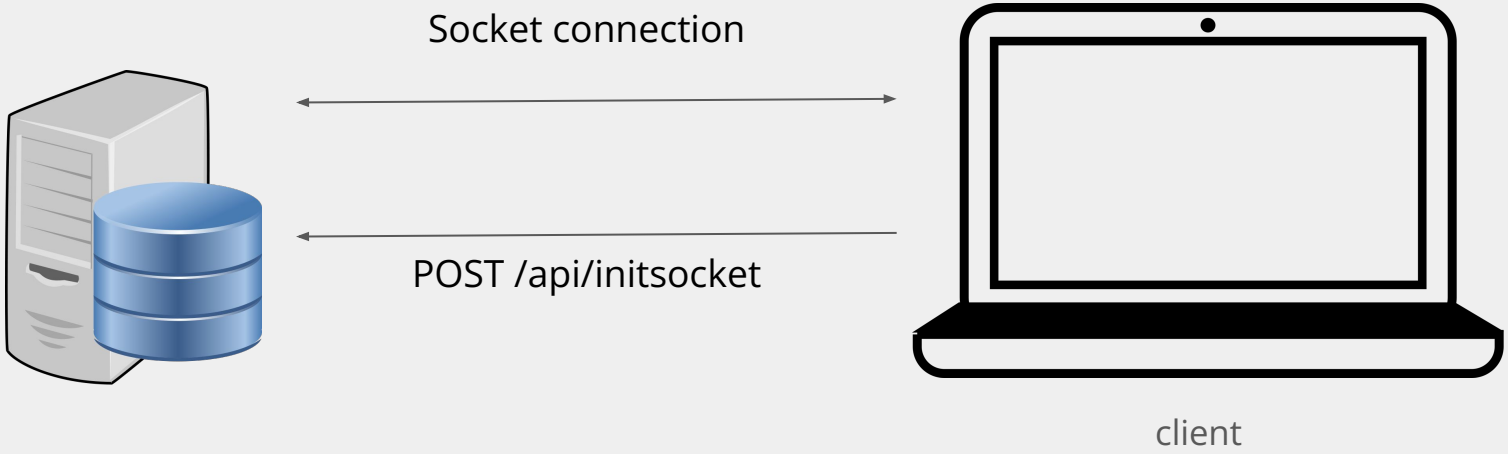


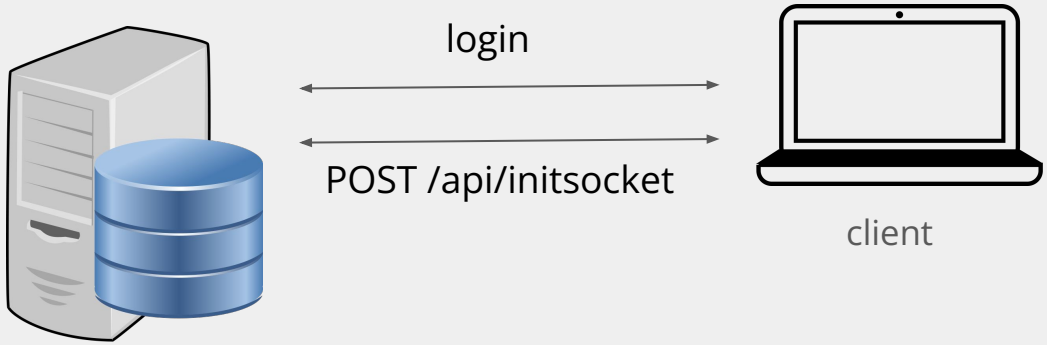
UserObject: {
_id: String,
name: String
}

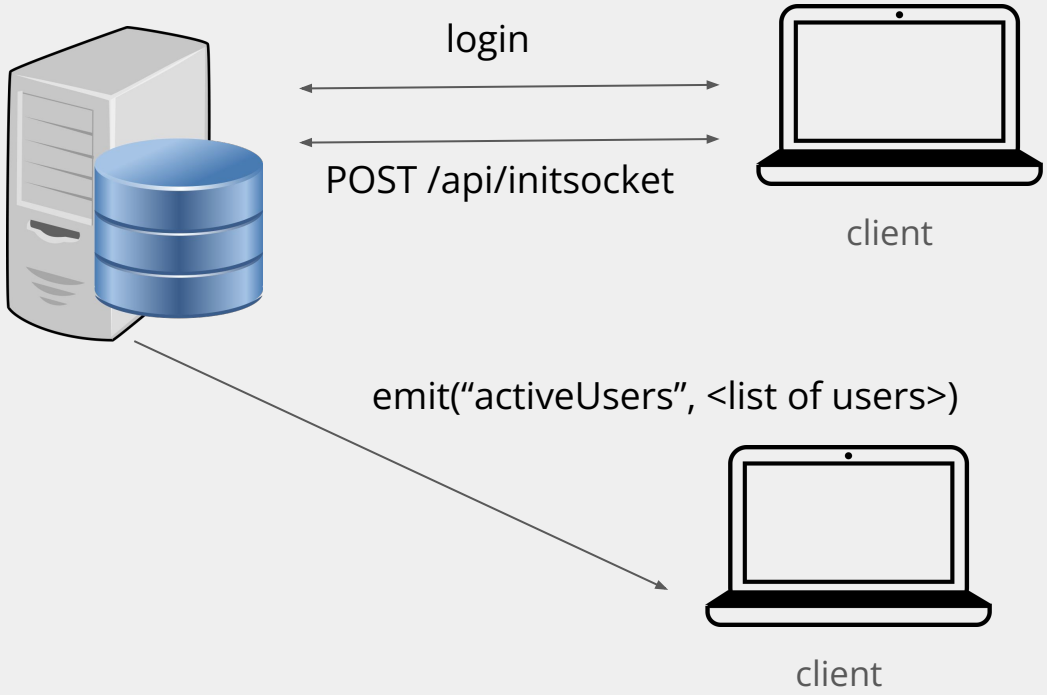
ChatData: {
messages: array of MessageObject,
recipient: UserObject
}

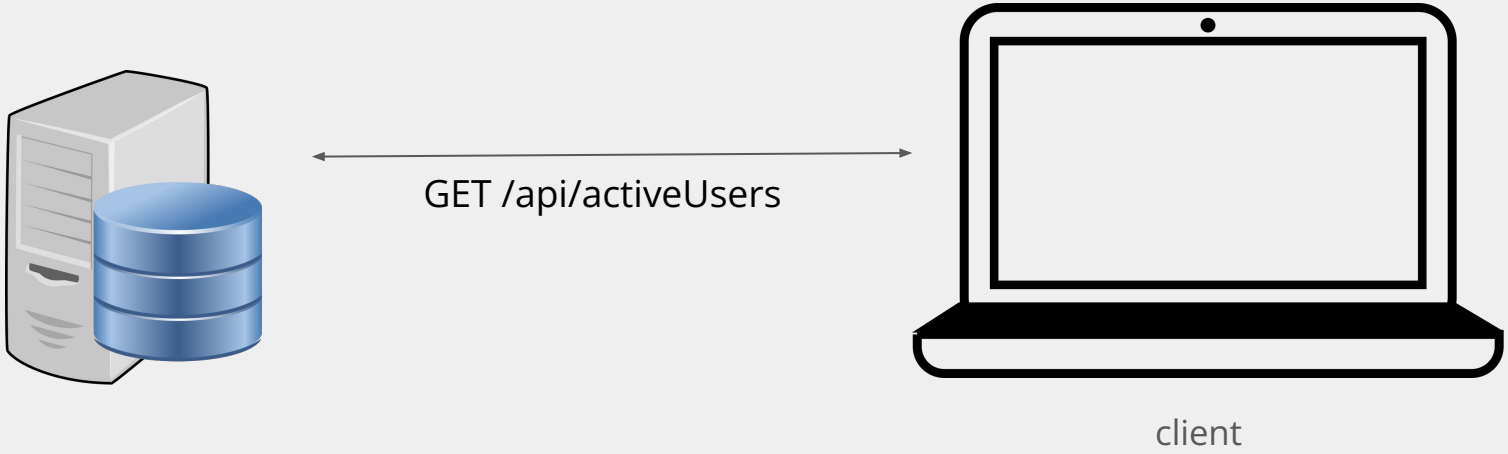
Backend API


Socket connection
client


Socket connection
POST /api/initsocket
client


login
POST /api/initsocket
client


login
POST /api/initsocket
client
emit(“activeUsers”, <list of users>)
client


GET /api/activeUsers
client

Now let's code it up!

Go to catbook-react!

git fetch
git reset --hard
git checkout w9-starter
npm install
Start the hotloader and server

[Live Coding]

Step 3: Add ChatList component to Chatbook

- Insert the ChatList component we just built into Chatbook
- Check **ChatList.js** for the 4 props that you should pass in
- Hint1: Components are initialized like so:

```
<ComponentName prop1name=prop1val prop2name=prop2val ../>
```

Step 3: Add ChatList component to Chatbook

- Insert the ChatList component we just built into Chatbook
- Check **ChatList.js** for the 4 props that you should pass in
- Hint1: Components are initialized like so:

```
<ComponentName prop1name=prop1val prop2name=prop2val ../>
```

- Hint2: users prop should be the `activeUsers` state

Step 3: Add ChatList component to Chatbook

- Insert the ChatList component we just built into Chatbook
- Check **ChatList.js** for the 4 props that you should pass in
- Hint1: Components are initialized like so:

```
<ComponentName prop1name=prop1val prop2name=prop2val ../>
```

- Hint2: users prop should be `activeUsers` held in state
- Hint3: look inside the state `activeChat` which should hold the current active user and pass that as the active prop

Step 3: Add ChatList component to Chatbook

- Insert the ChatList component we just built into Chatbook
- Check **ChatList.js** for the 4 props that you should pass in
- Hint1: Components are initialized like so:

```
<ComponentName prop1name=prop1val prop2name=prop2val ../>
```

- Hint2: users prop should be `activeUsers` held in state
- Hint3: look inside the state `activeChat` which should hold the current active user and pass that as the active prop
- Hint4: userId was passed into this component as a prop

File: Chatbook.js Line: 86

Types & APIs: weblab.to/chatbook-docs

Step 4: Implement `/api/activeUsers`

- Send back an object with the field **`activeUsers`** set to the currently active users
- Hint1: You can send back a response with **res.send(<object>)**

Step 4: Implement `/api/activeUsers`

- Send back an object with the field **`activeUsers`** set to the currently active users
- Hint1: You can send back a response with **res.send(<object>)**
- Hint2: Object syntax is {<field-name> : <field-value>}

Step 4: Implement `/api/activeUsers`

- Send back an object with the field **`activeUsers`** set to the currently active users
- Hint1: You can send back a response with **res.send(<object>)**
- Hint2: Object syntax is {<field-name> : <field-value>}
- Hint3: You can get all the connected users with **`socketManager.getAllConnectedUsers()`**

[Live Coding]

Step 6: Catch the `activeUsers` event

- Use **“socket.on”** to catch the `activeUsers` event.
- In the callback function, set the state of **activeUsers** to the new list of active users
- Hint1: look at `socket.on(“message”)` in the same file to see how to listen on a socket event!

Step 6: Catch the `activeUsers` event

- Use **"socket.on"** to catch the `activeUsers` event.
- In the callback function, set the state of **activeUsers** to the new list of active users
- Hint1: look at `socket.on("message")` in the same file to see how to listen on a socket event!
- Hint2: look at `get("/api/activeUsers")` to see what we should do to update the state to the new list of activeUsers

Step 7: Implement setActiveUser

- We implemented **setActiveUser** partially a while ago, you just need to change it to **set the state** instead of console logging! Do the following:
  1. First, set the **state activeChat** to the passed in **user** as recipient and **empty** message array as messages
  2. Then, figure out a clean way to load the message history (may need to add code elsewhere)
- Hint: You can load the message history by calling `loadMessageHistory(user)`

[Live Coding]

Step 10: Emit a message event to recipient

- We emitted a message to the currently signed in user, now we just need to emit to the recipient!
  1. Emit a “message” event to the socket connected to the recipient
- Hint: Look at how we sent the message back to the sender (req.user)

Step 10: Emit a message event to recipient

- We emitted a message to the currently signed in user, now we just need to emit to the recipient!
  1. Emit a “message” event to the socket connected to the recipient
- Hint1: you can get the recipient id using req.body.recipient

Step 10: Emit a message event to recipient

- We emitted a message to the currently signed in user, now we just need to emit to the recipient!
  1. Emit a “message” event to the socket connected to the recipient
- Hint1: you can get the recipient id using req.body.recipient
- Hint2: you want to emit the same contents as the message we just emitted to the current user

Step 10: Emit a message event to recipient

- We emitted a message to the currently signed in user, now we just need to emit to the recipient!
  1. Emit a “message” event to the socket connected to the recipient
- Hint1: you can get the recipient id using req.body.recipient
- Hint2: you want to emit the same contents as the message we just emitted to the current user
- Hint3: use socketManager.getSocketFromUserID!

Now you try! Catch the `forceDisconnect` event

- Use **“socket.on”** to catch the `forceDisconnect` event.
- In the callback function, set the state of **socketDisconnected** to **false**
- Hint: this is what we did to catch the message event and change state in Chatbook.js

```
socket.on("activeUsers", (data) => {
  this.setState({
    activeUsers: [ALL_CHAT].concat(data.activeUsers),
  });
});
```

SHOUT: socketManager.getIO()
whisper: socketManager.getSocketFromUserId()

client-socket & server-socket

Details: weblab.to/socket-guide

Sockets in Practice

```
// Client, inside useEffect() with cleanup!
socket.on("eventName", (data) => {
   // Do stuff with data
});

// Server, probably inside an API call or helper function
socketManager.getIO().emit("eventName", data); // SHOUT

// whisper
socketManager.getSocketFromUserID(userID).emit("eventName", data);
```