Announcements

- Office Hours tonight 6-9pm, Hackathon 😋🕐 7pm-1am tomorrow (Friday) night!
  - In 32-082
- If you felt really lost on previous material, 😖come to 56-114 and we'll have a couple staff members to walk through the content more slowly.
- 🪧Milestone 2 (Minimum Viable Product) due next Wednesday (Jan 24) 6:00pm!

W10: Gamebook

Kenneth Choi & Michael Kuoch

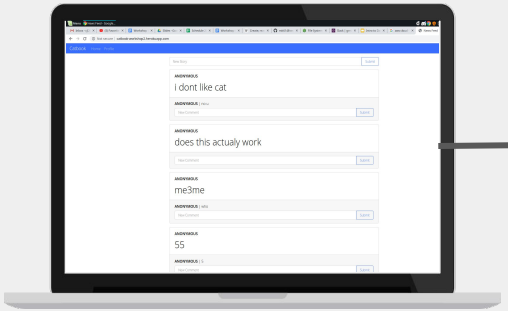

AMONG US
LEAGUE of LEGENDS
Agar.io
POKÉMON GO
New Horizons
Welcome to Animal Crossing
2024
VALORANT
GENSHIN IMPACT
FORTNITE

What makes real time games a bit different?

Complicated game logic and state

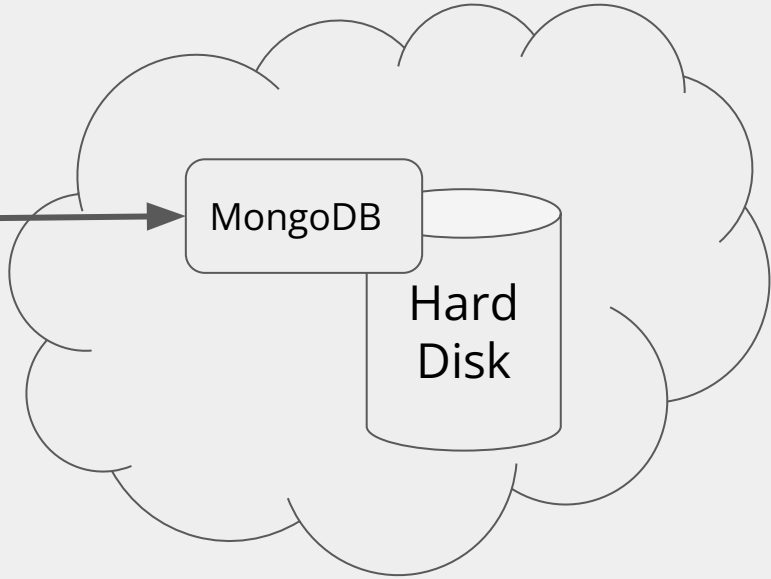
Performance super important

Authenticates with Google



User (Client)

Backend
(Server)



MongoDB on Atlas

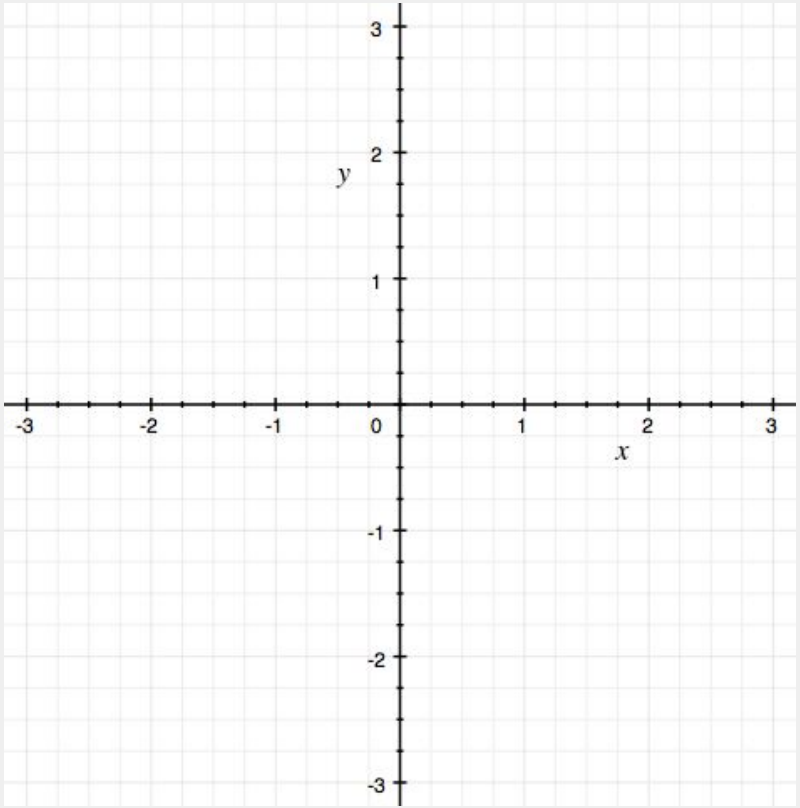
HTML

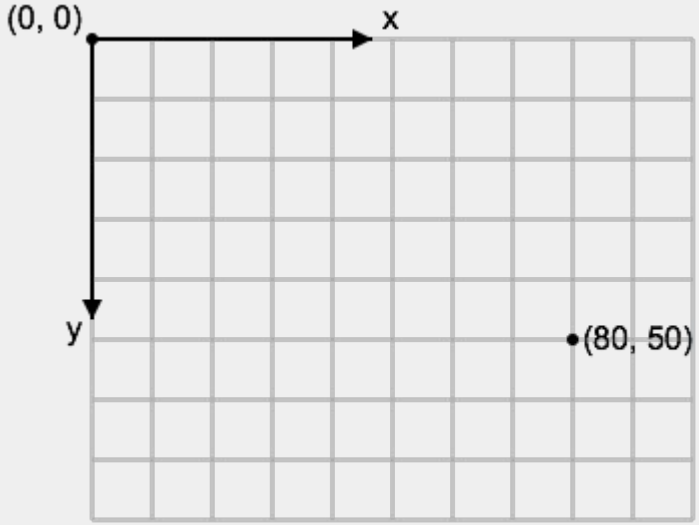
HTML5 Canvas to render

Game logic and game state
live here

What is HTML Canvas?

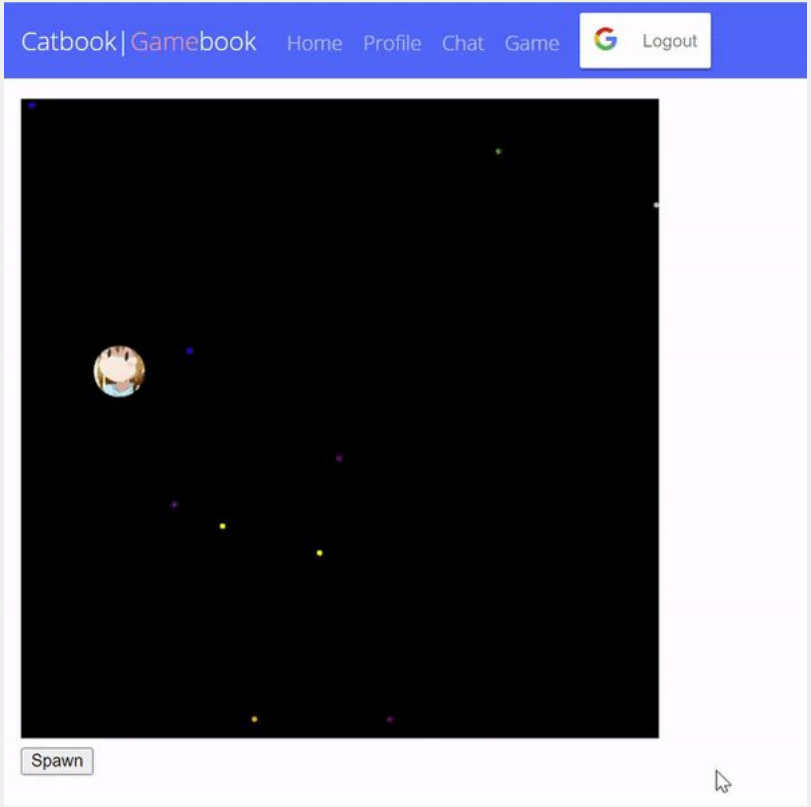
Normal Coordinates vs Canvas





What we’re building today (weblab.is/example)

What we're building today (weblab.is/example)

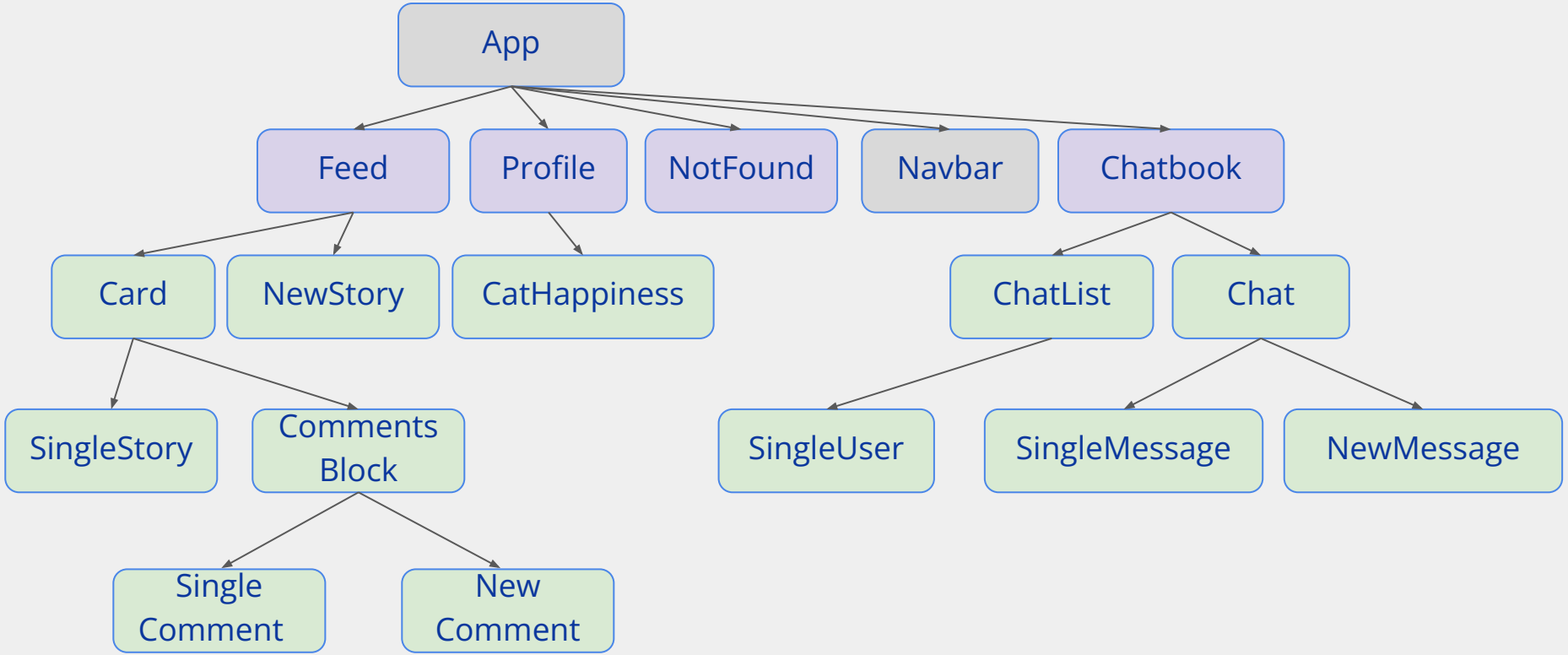


Any questions so far?

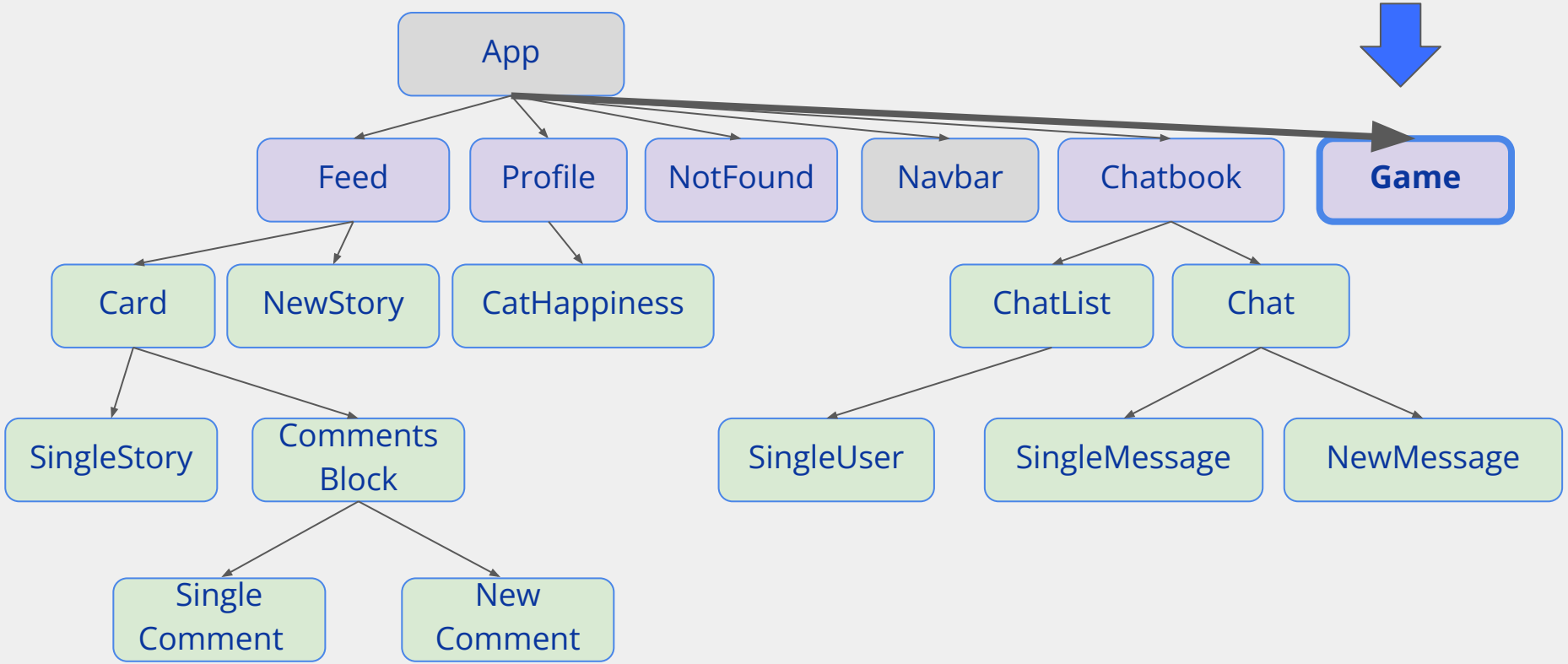
How to design a web game

Gamebook!

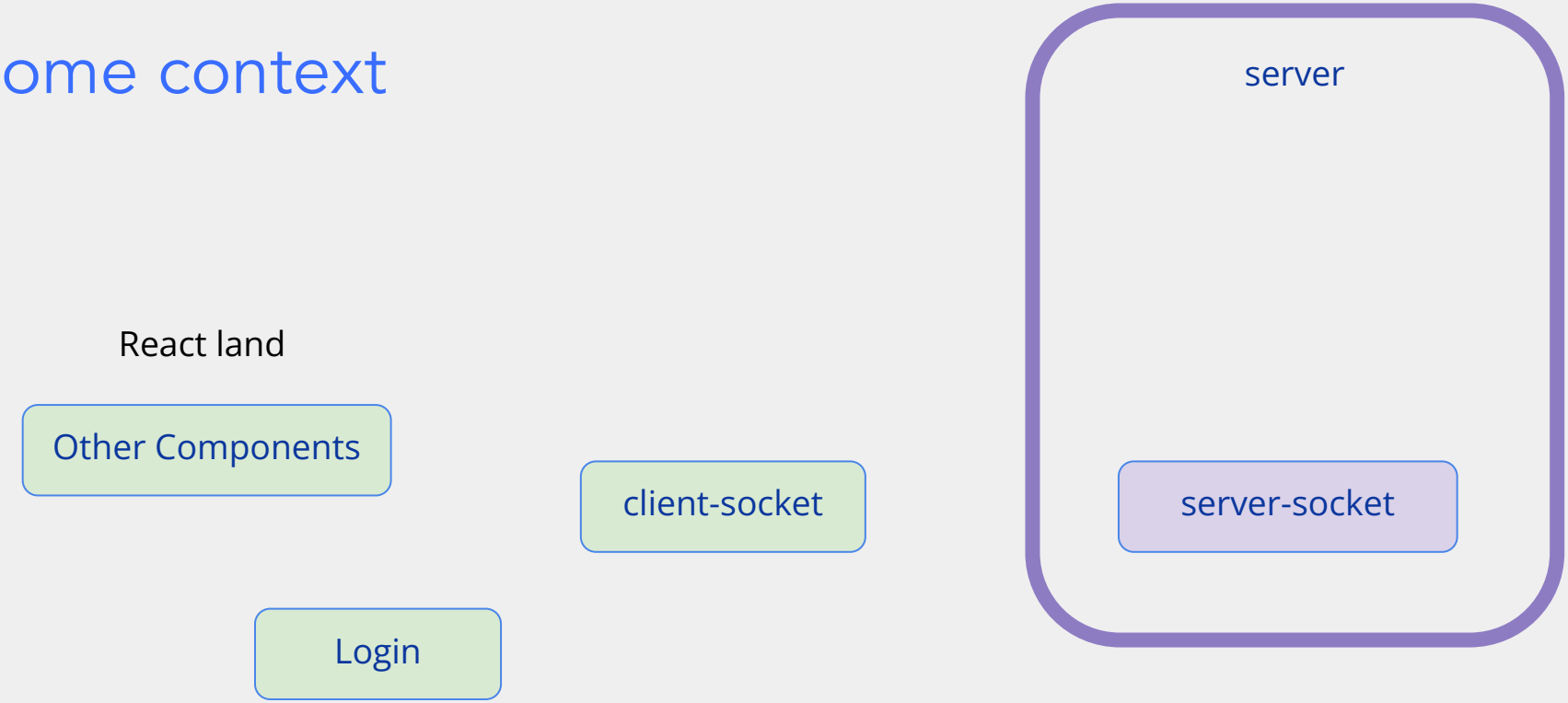
Catbook: The story so far:



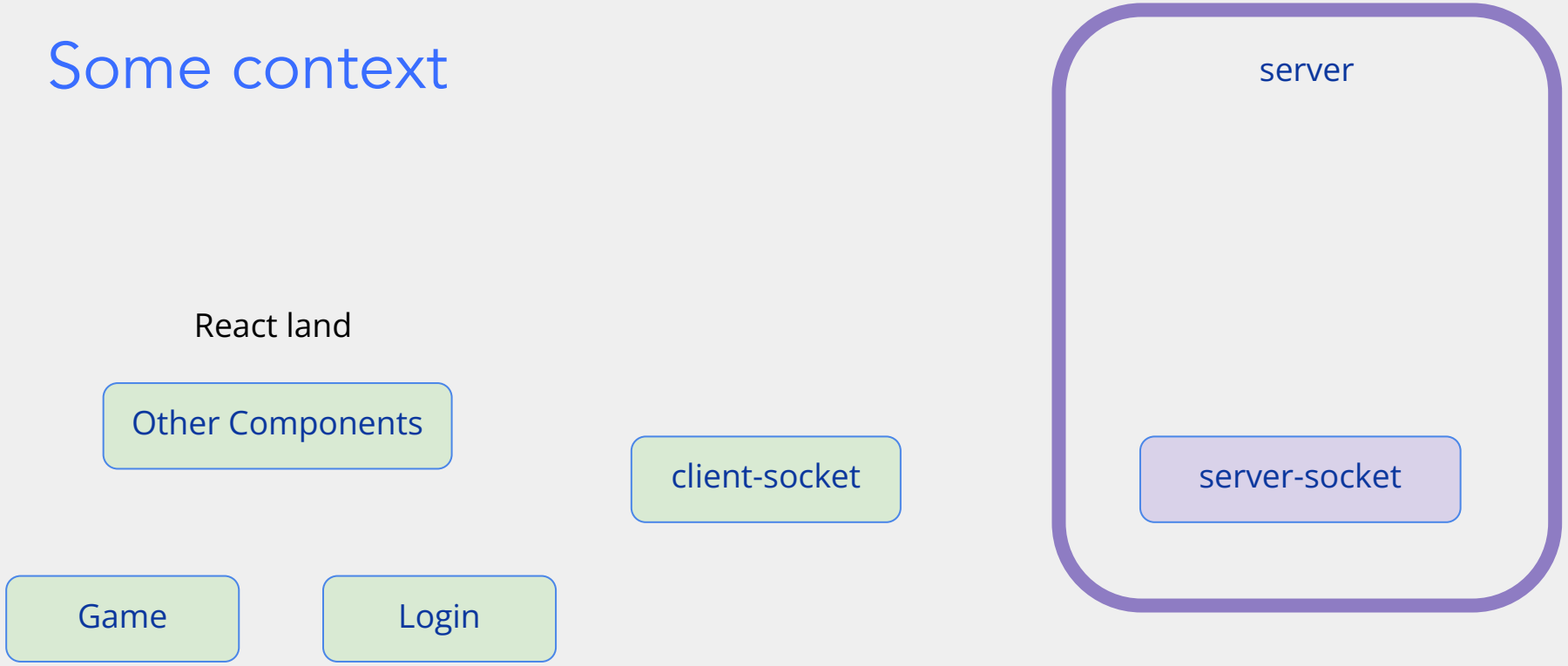
Gamebook!


App
Feed
Profile
NotFound
Navbar
Chatbook
Game
Card
NewStory
CatHappiness
ChatList
Chat
SingleStory
Comments Block
SingleUser
SingleMessage
NewMessage
Single Comment
New Comment

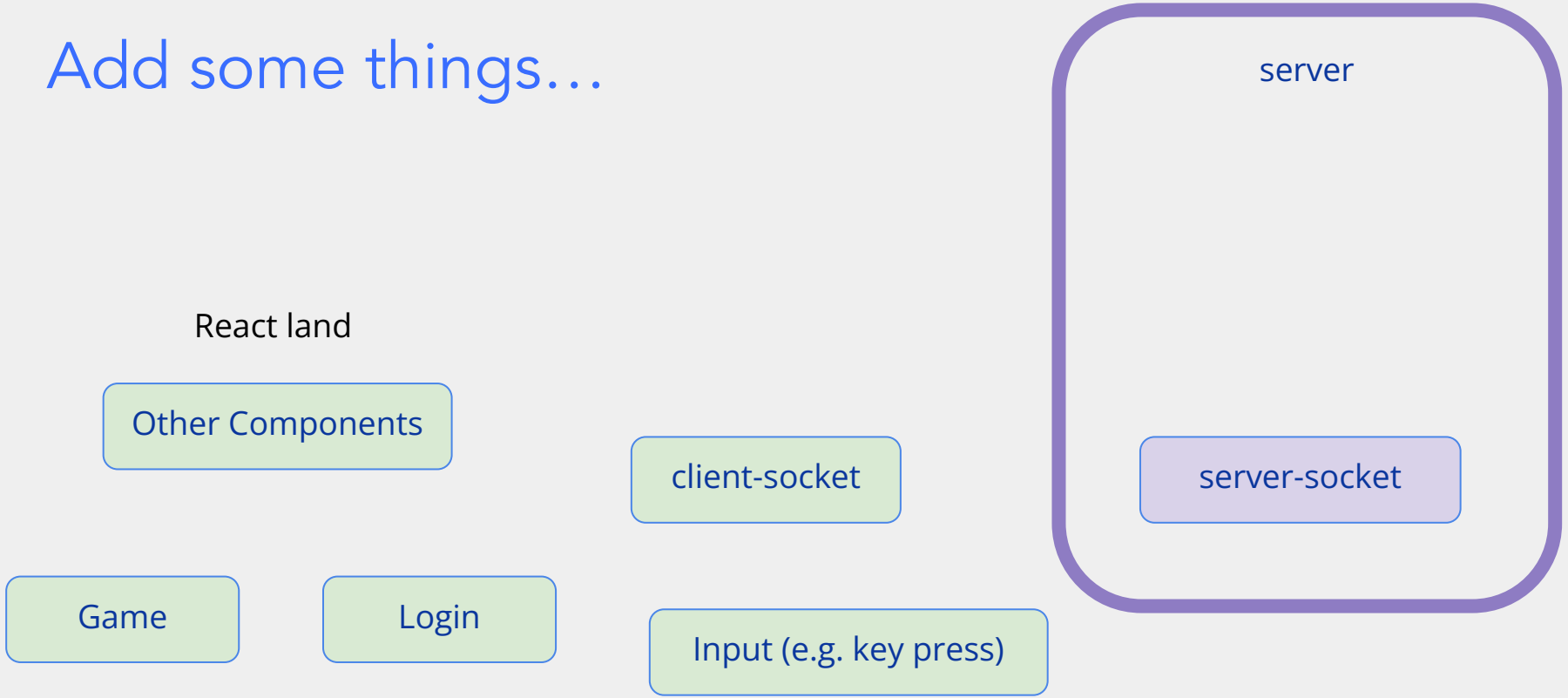
Some context



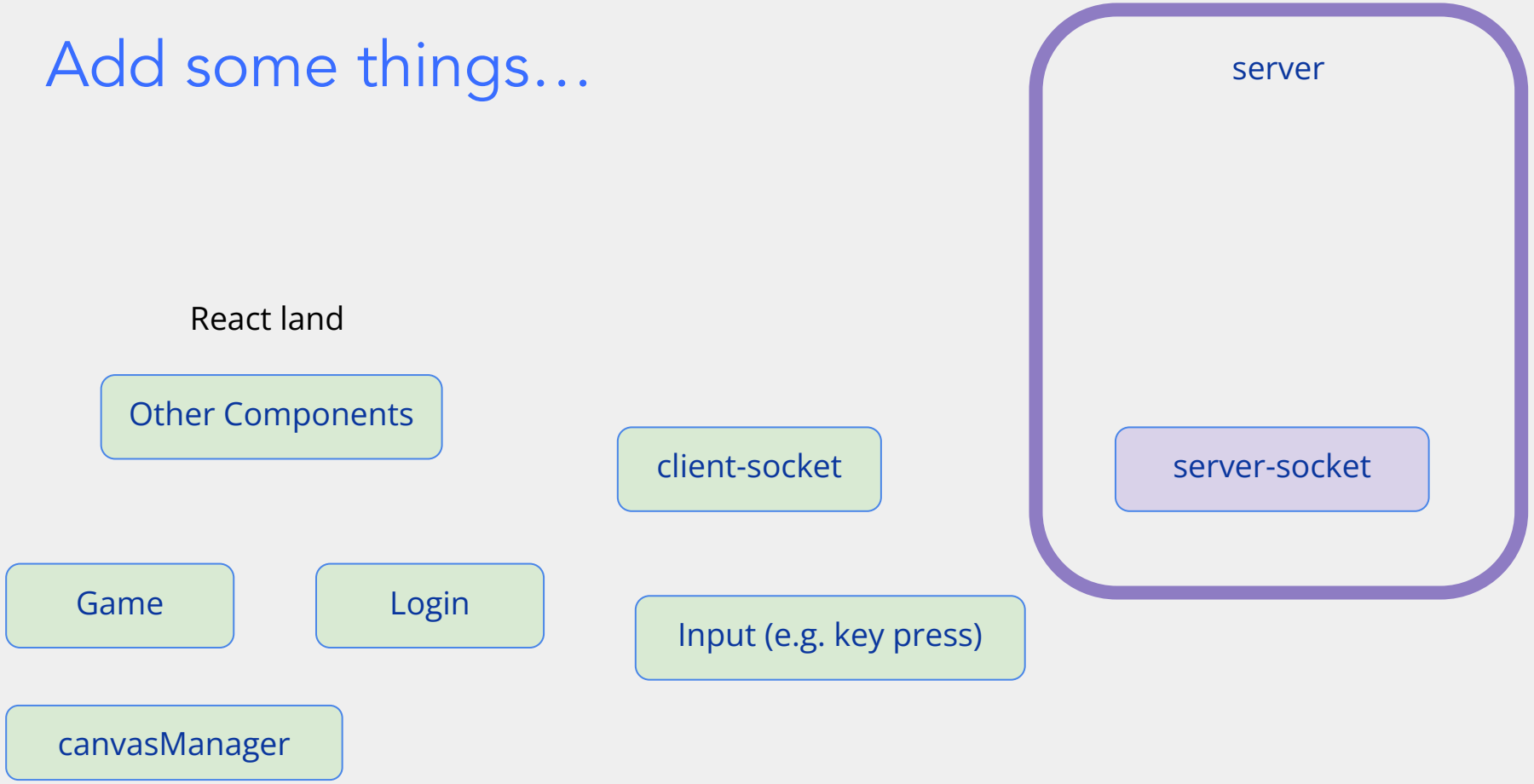
Some context



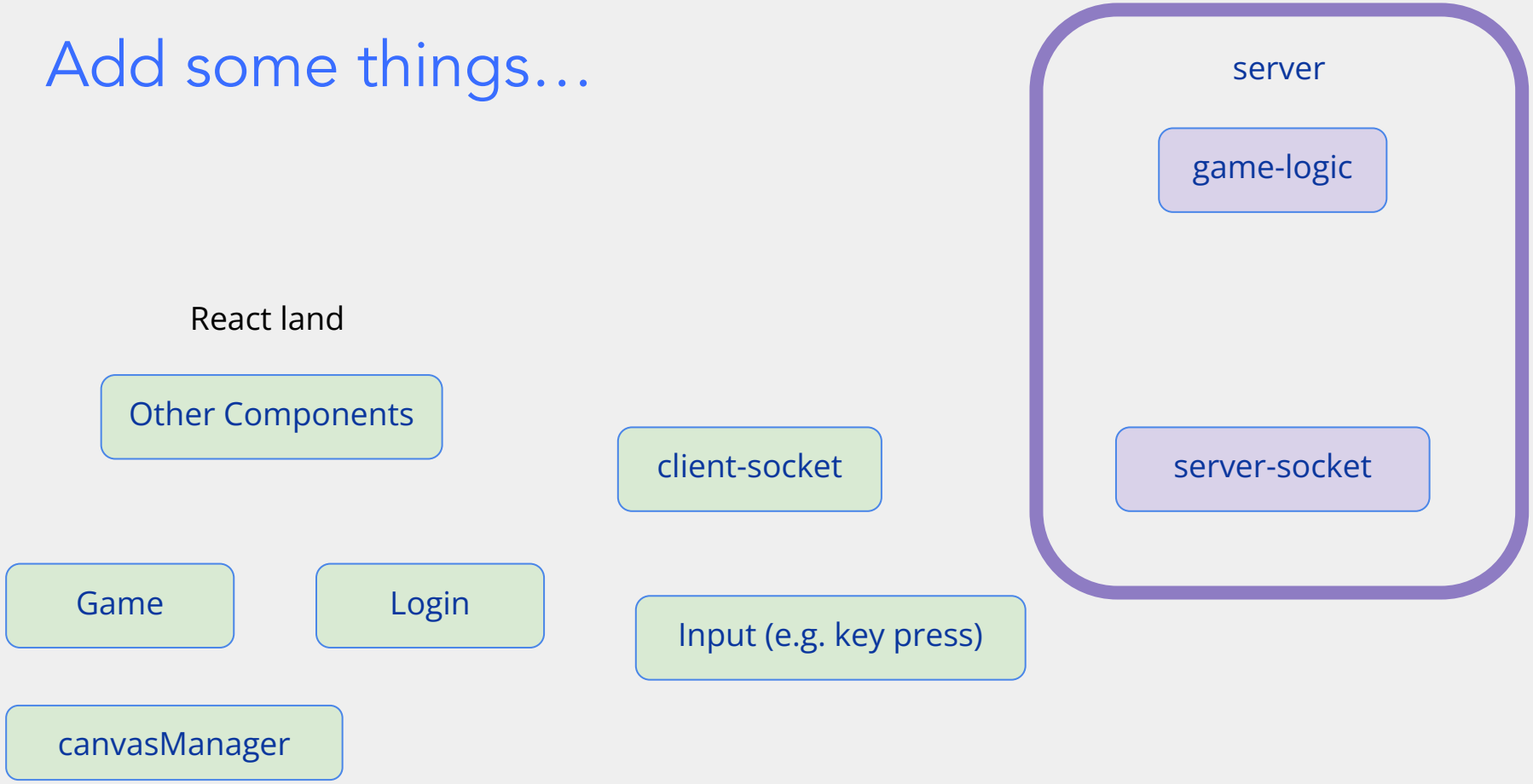
Add some things…



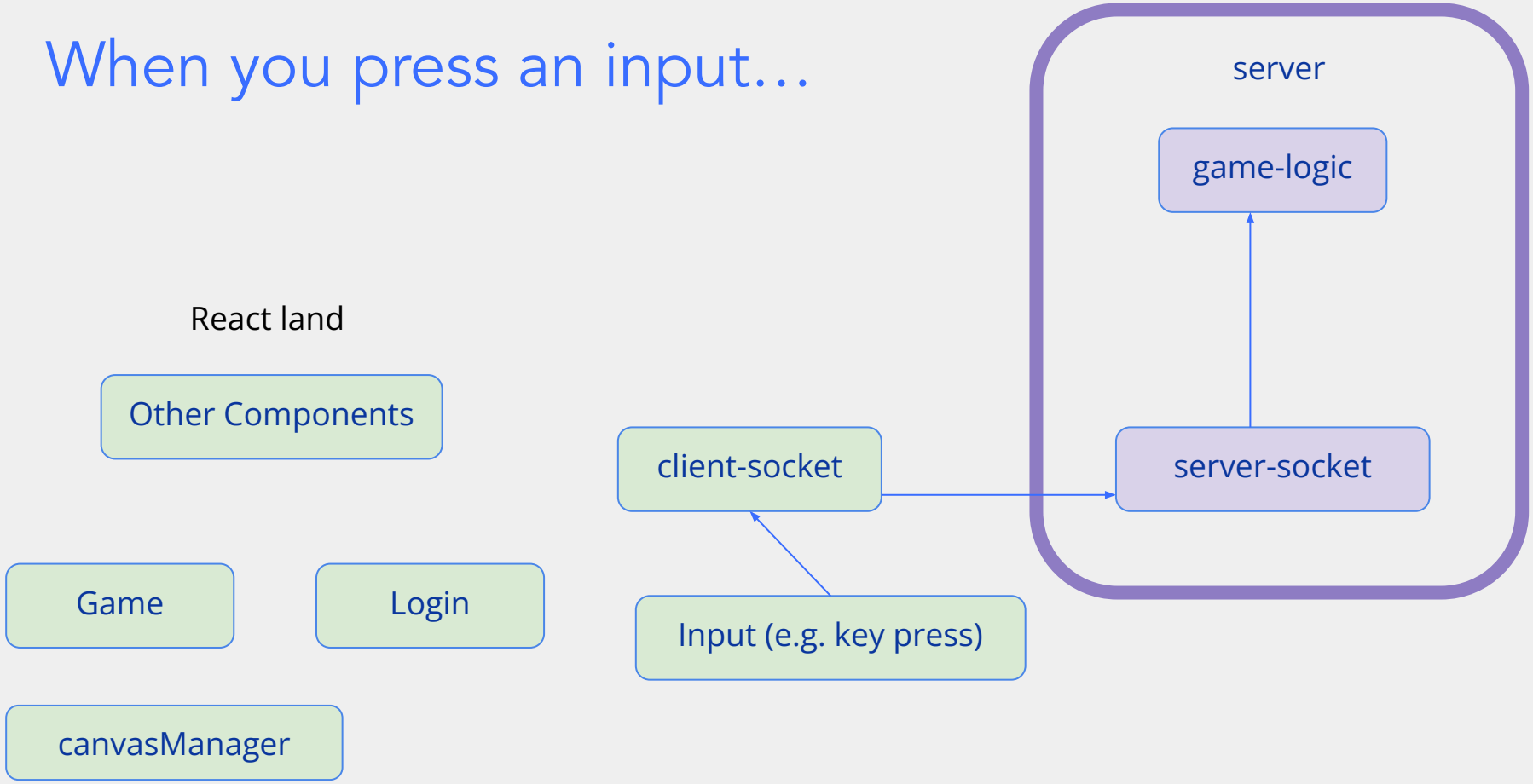
Add some things…



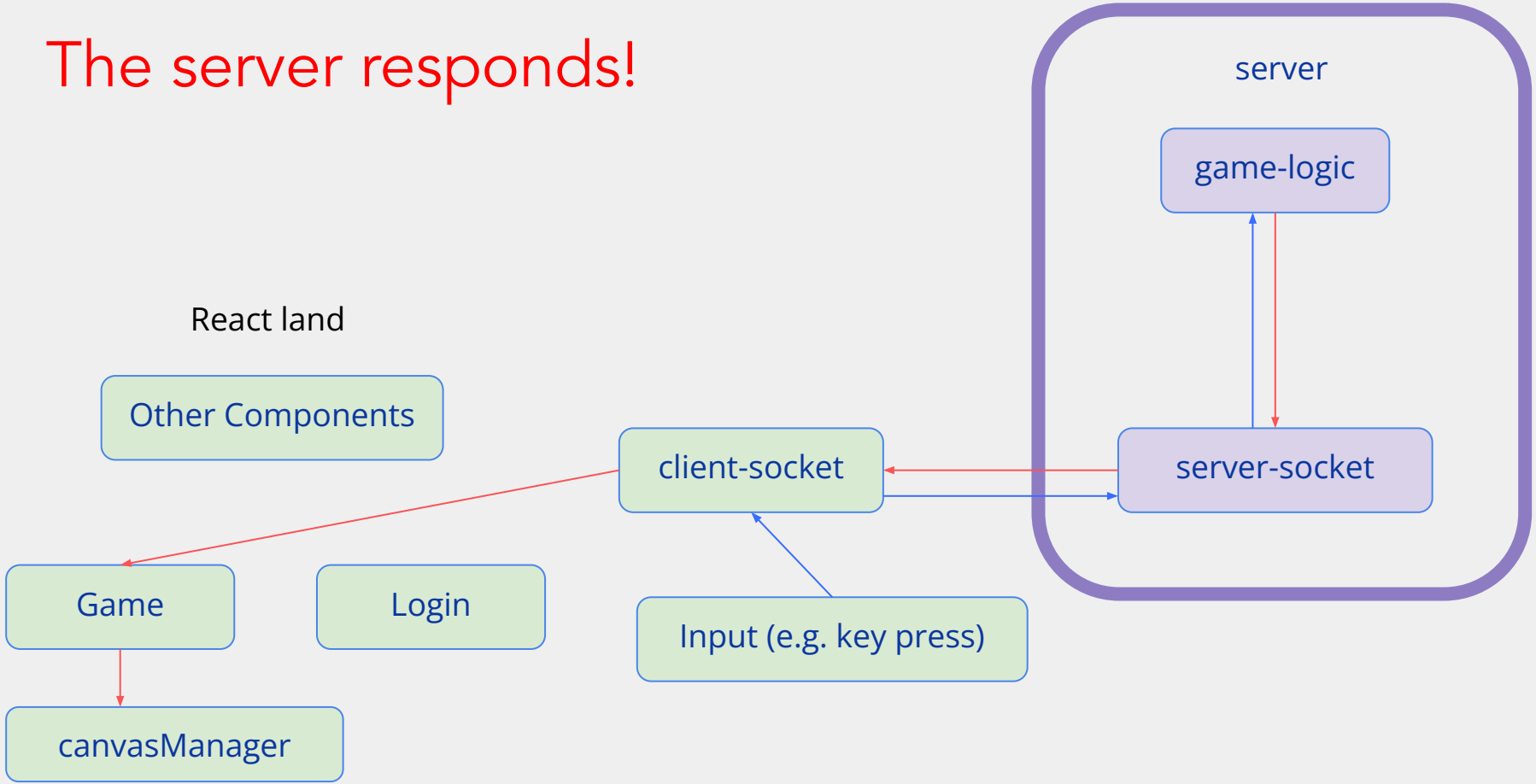
Add some things…



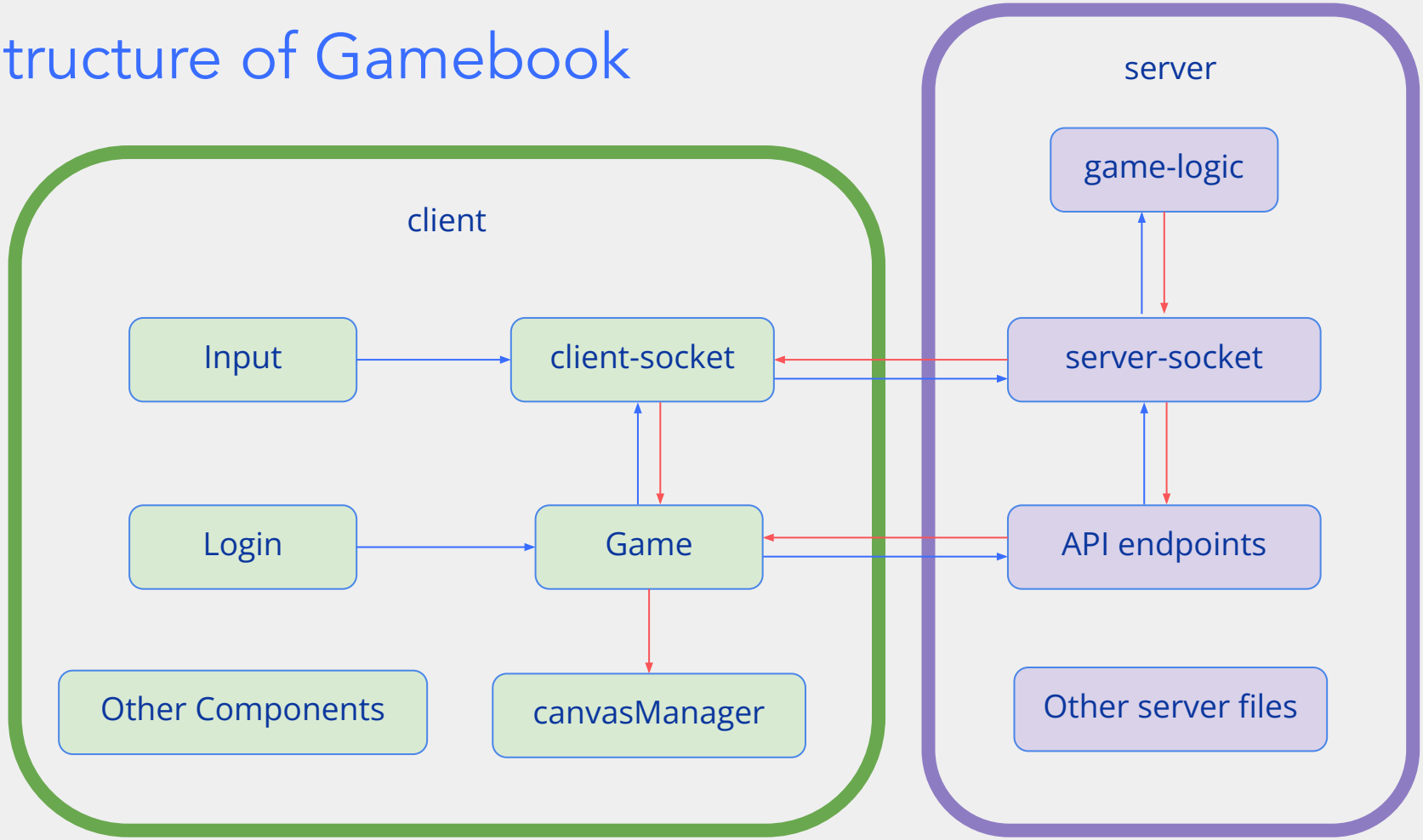
When you press an input…



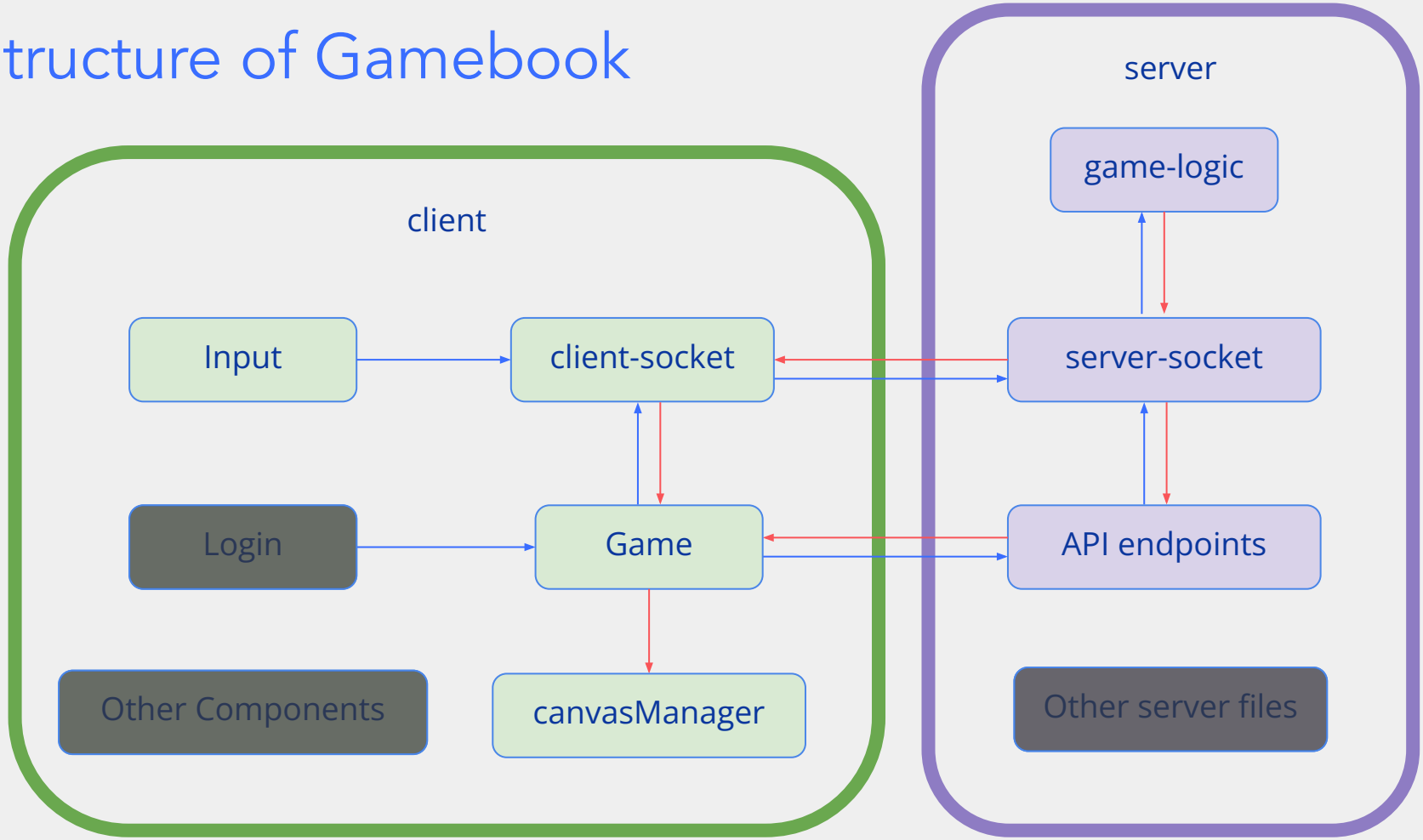
The server responds!



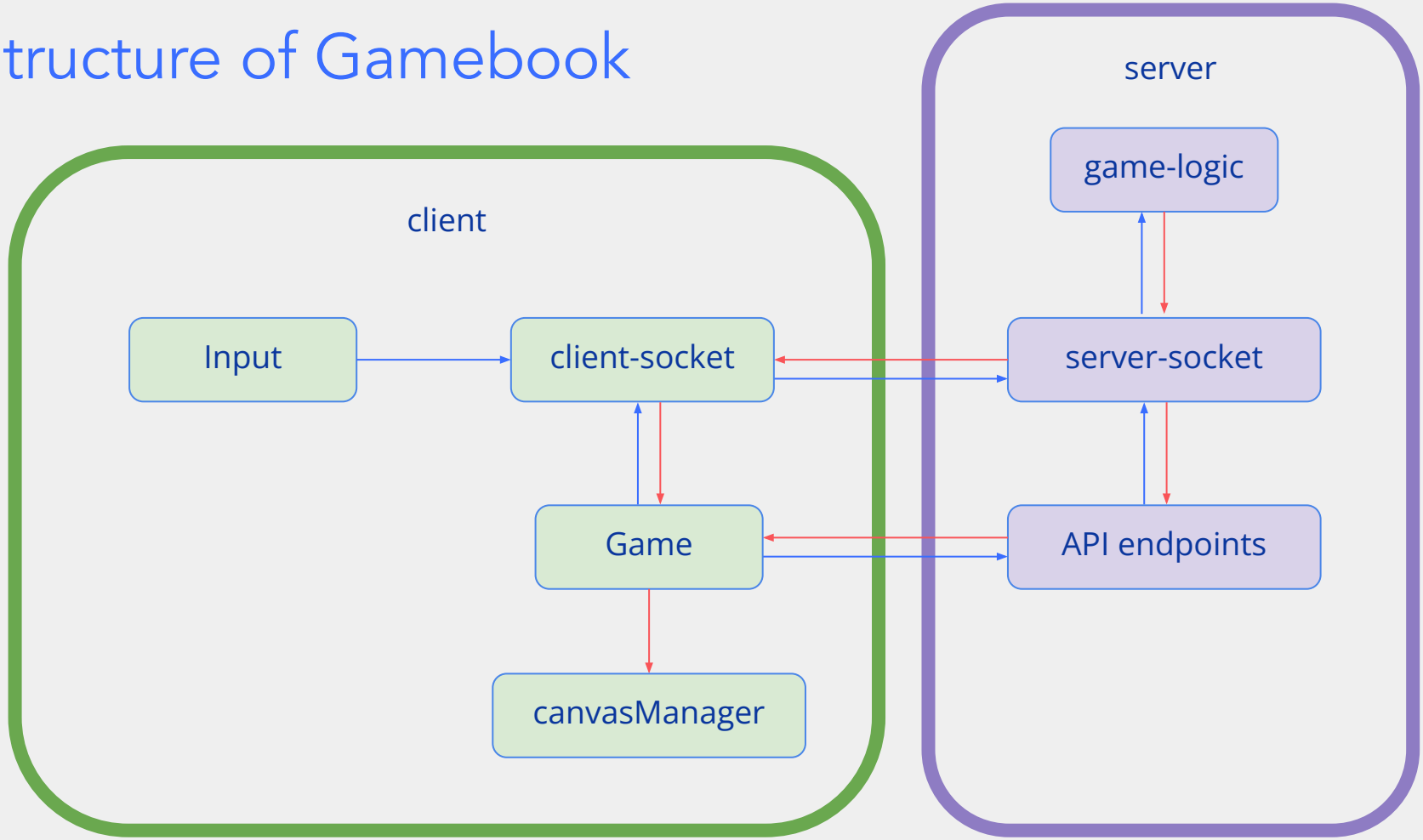
Structure of Gamebook



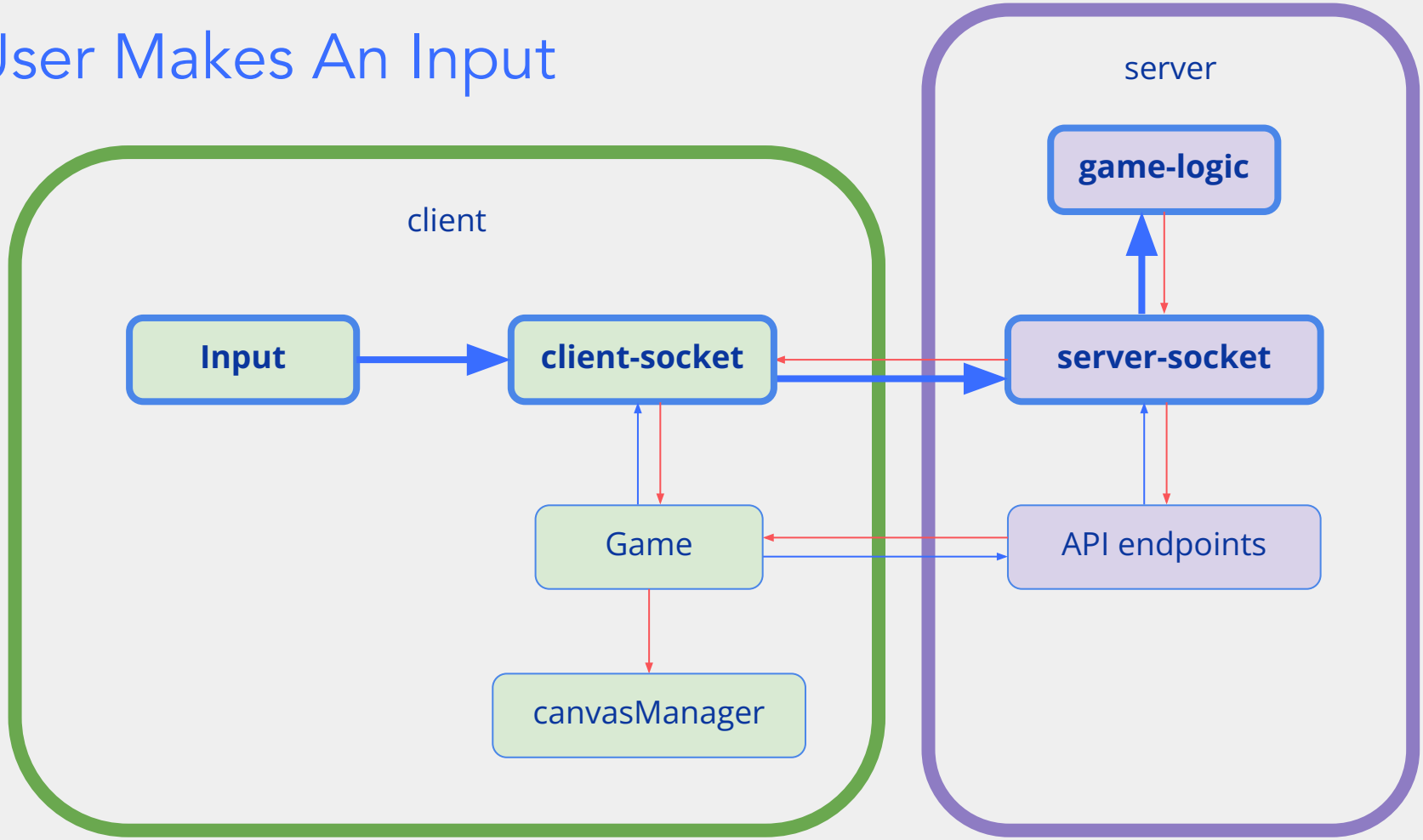
Structure of Gamebook



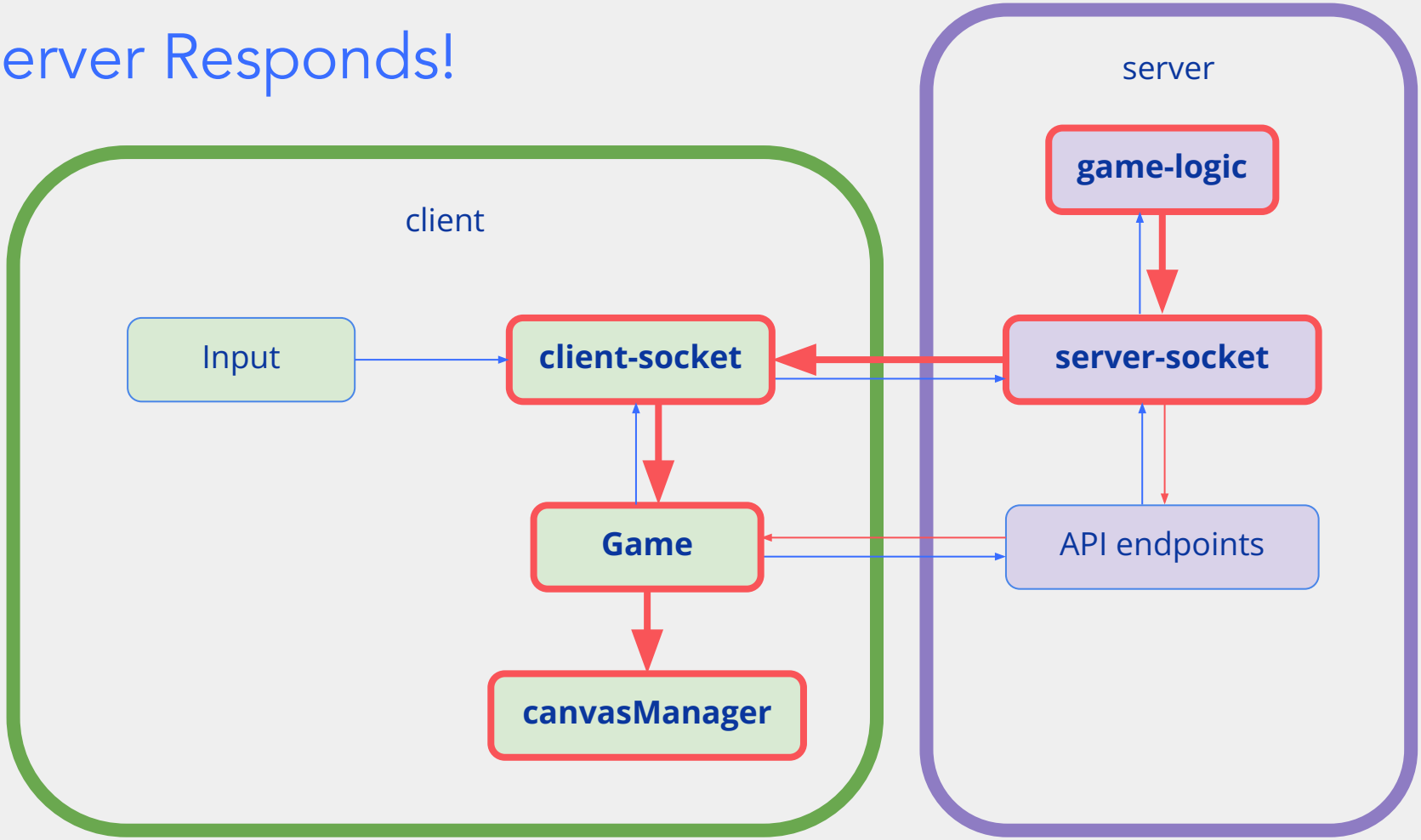
Structure of Gamebook



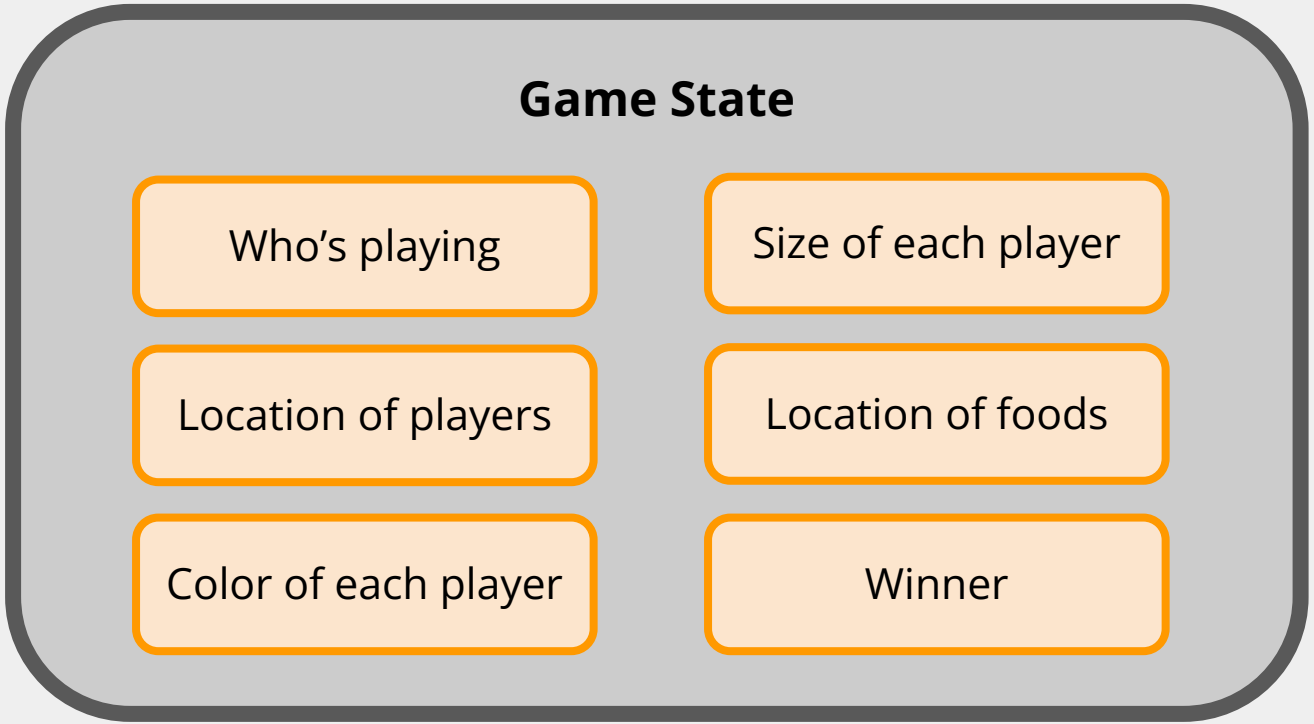
User Makes An Input



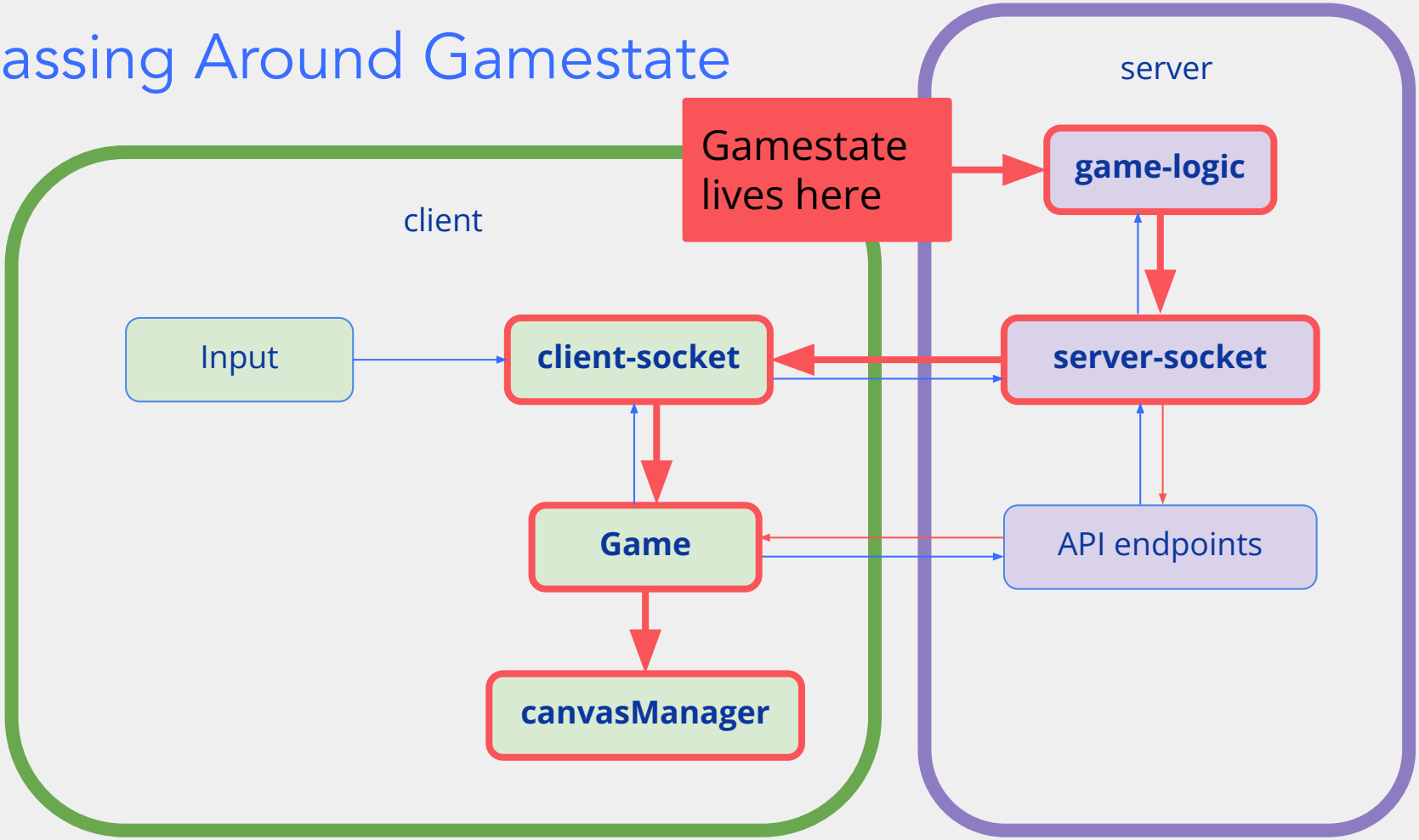
Server Responds!



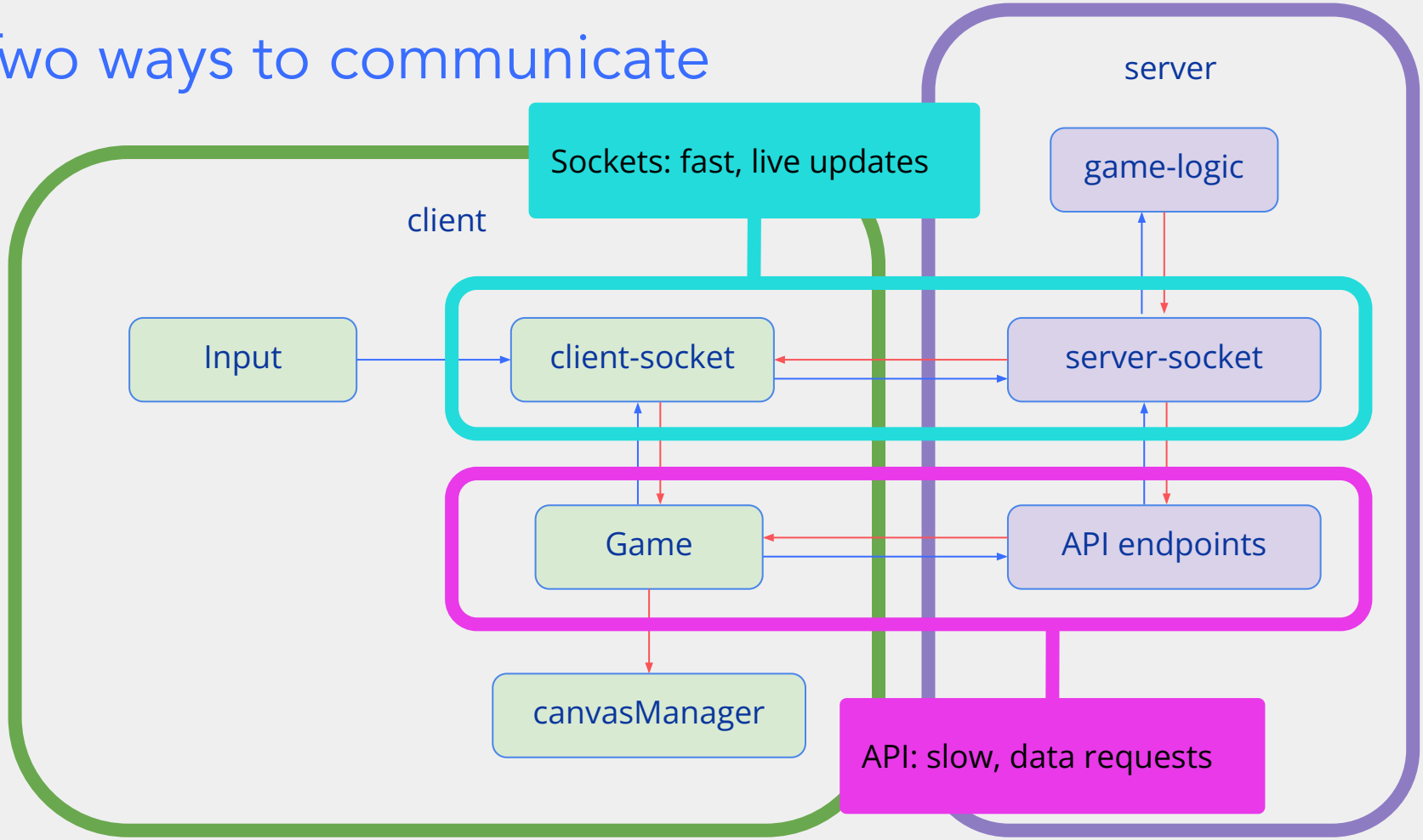
What is Game State?

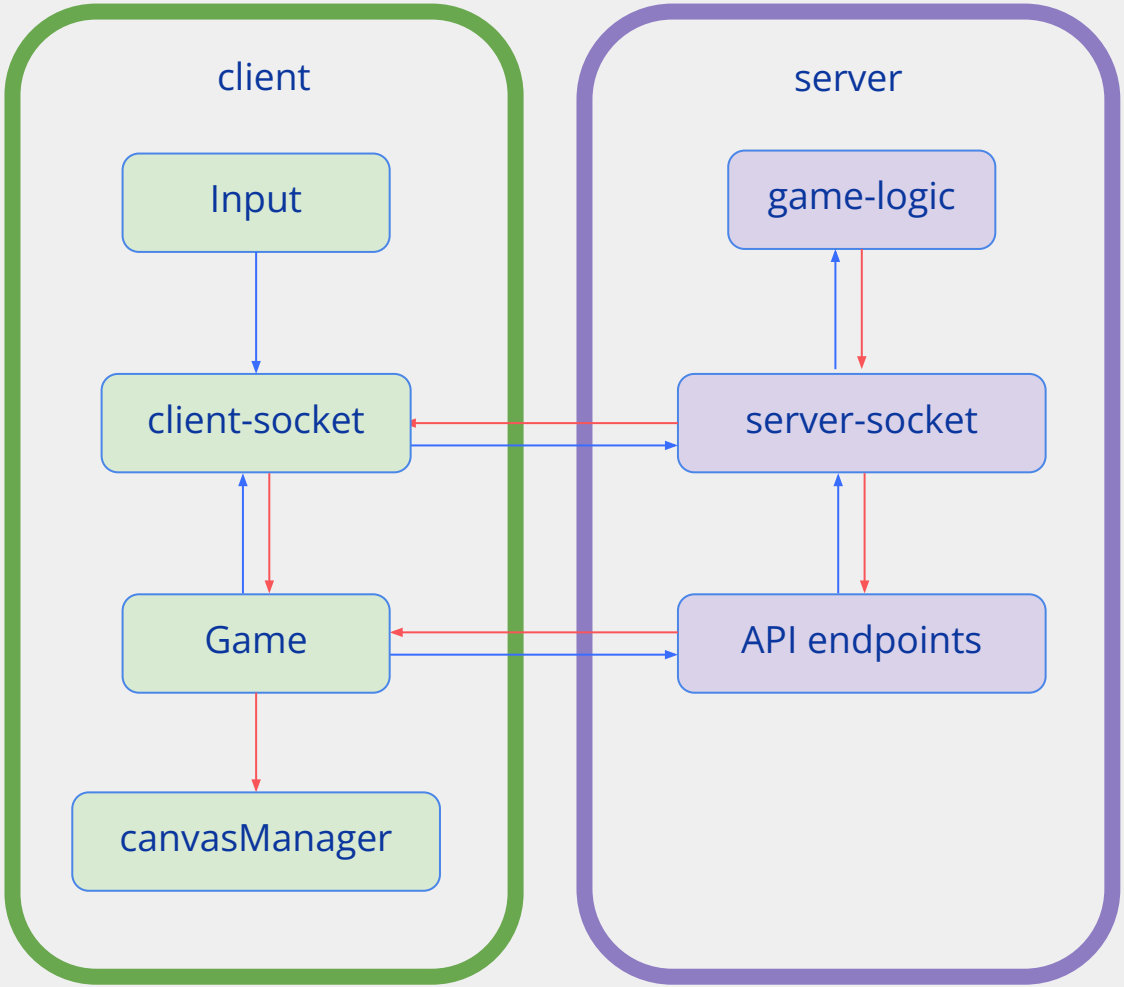


Passing Around Gamestate



Two ways to communicate




client
server
Input
game-logic
client-socket
server-socket
Game
API endpoints
canvasManager

Before we start!

Useful JS function: forEach

forEach

array.forEach((element) => {output})

- Calls function on each element in array
- Just like map, but mutates the array

forEach

myNumbers = [3, 1, 4, 1, 5, 9];

```
myNumbers.forEach((number) => {
    console.log(2*number);
});
```

Result:

6

2

8

2

10

18

forEach

myNumbers = [3, 1, 4, 1, 5, 9];

```
myNumbers.forEach((number) => {

    myNumbers.pop(number);

});
```

Removes the last element
of the array (and returns it)

Result:

myNumber = [3, 1, 4]

No undefined / out of bounds errors :)

Let’s make a game!

Let's make a game!!

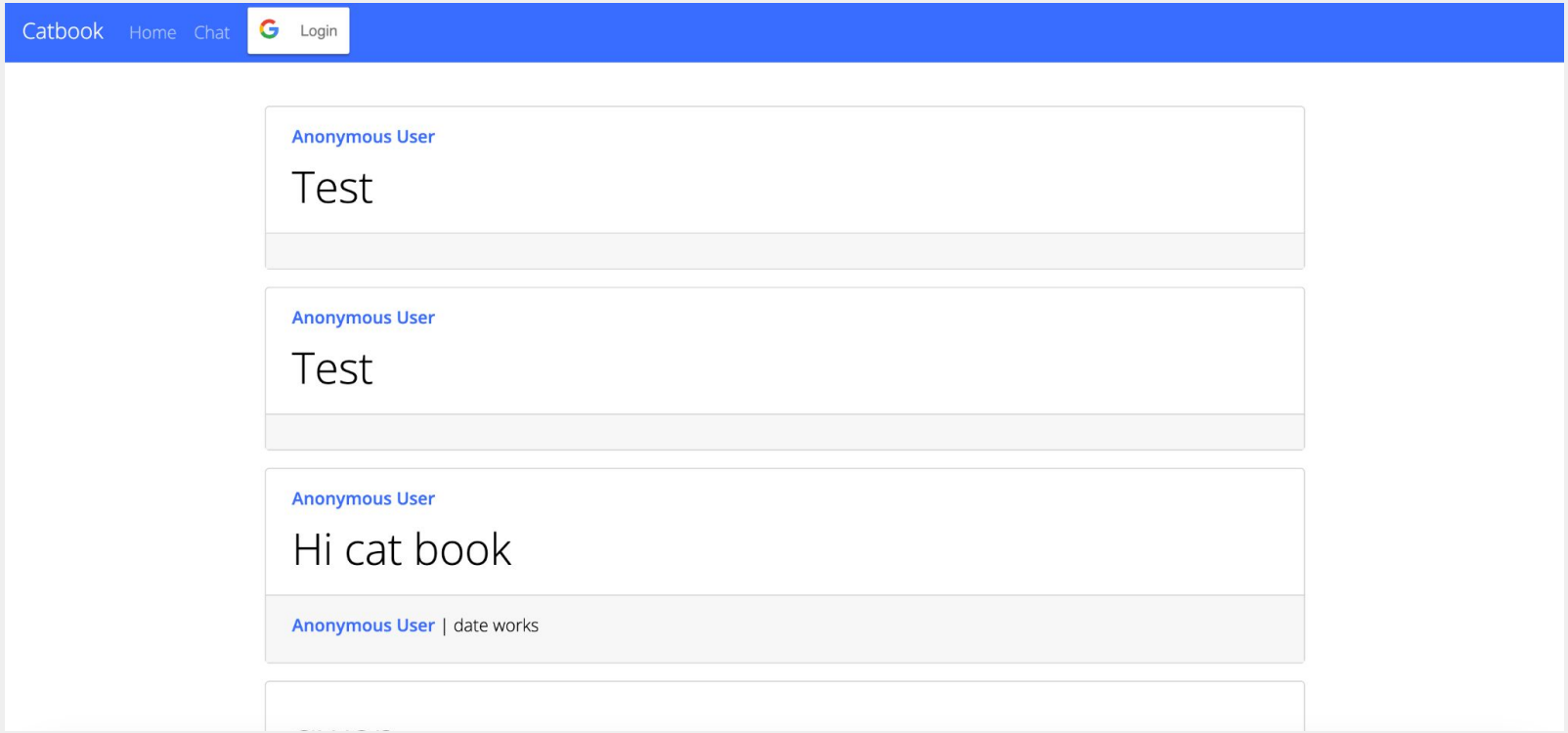
```
git fetch
git reset --hard
git checkout w10-starter
```

Step 0: Make a front-end page

It’s literally just blank

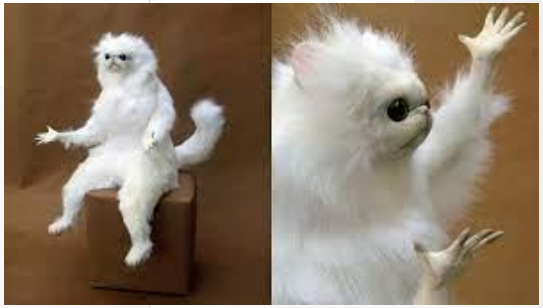
git fetch; git reset --hard; git checkout w10-starter

Previously, we made Catbook



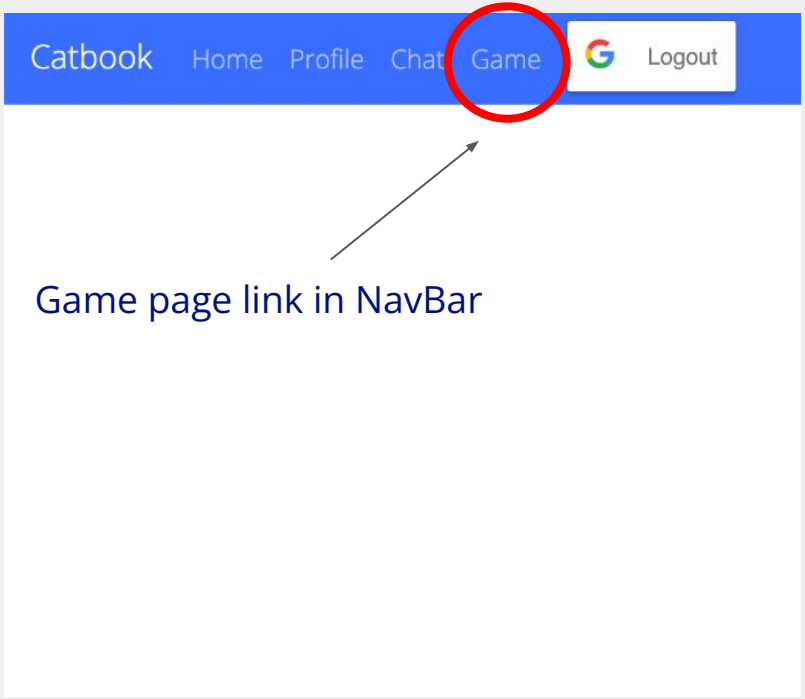
git fetch; git reset --hard; git checkout w10-starter

e Catbook



git fetch; git reset --hard; git checkout w10-starter

Step 0: Making A Blank Game Page



Step 0.1 Route to Game page in App.js (line 8 and 61)

Step 0.2 Add Game page links in NavBar.js (line 29)

Your Turn

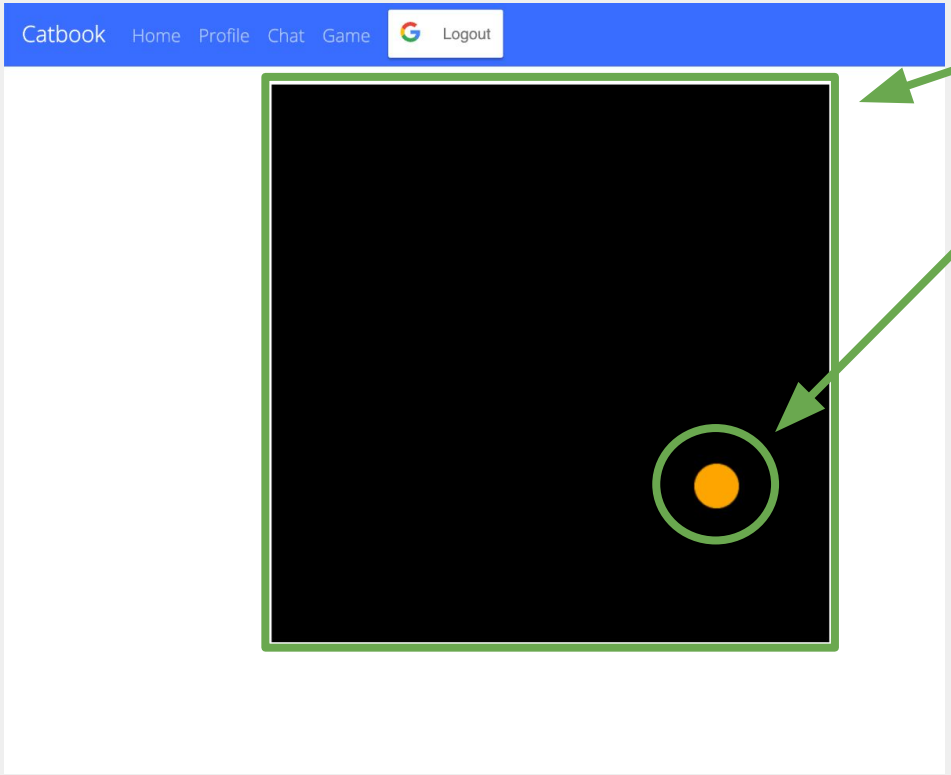
```
git reset --hard
git checkout w10-step1
```

Step 1: Basic game logic and display

Game server + Canvas

git fetch; git reset --hard; git checkout w10-step1

Step 1: Canvas & Basic Game Logic



Client: **Canvas** element on Game page

Client: Player (circle) drawn on canvas

Server:
- **gameState** that is constantly updated
    - winner
    - players

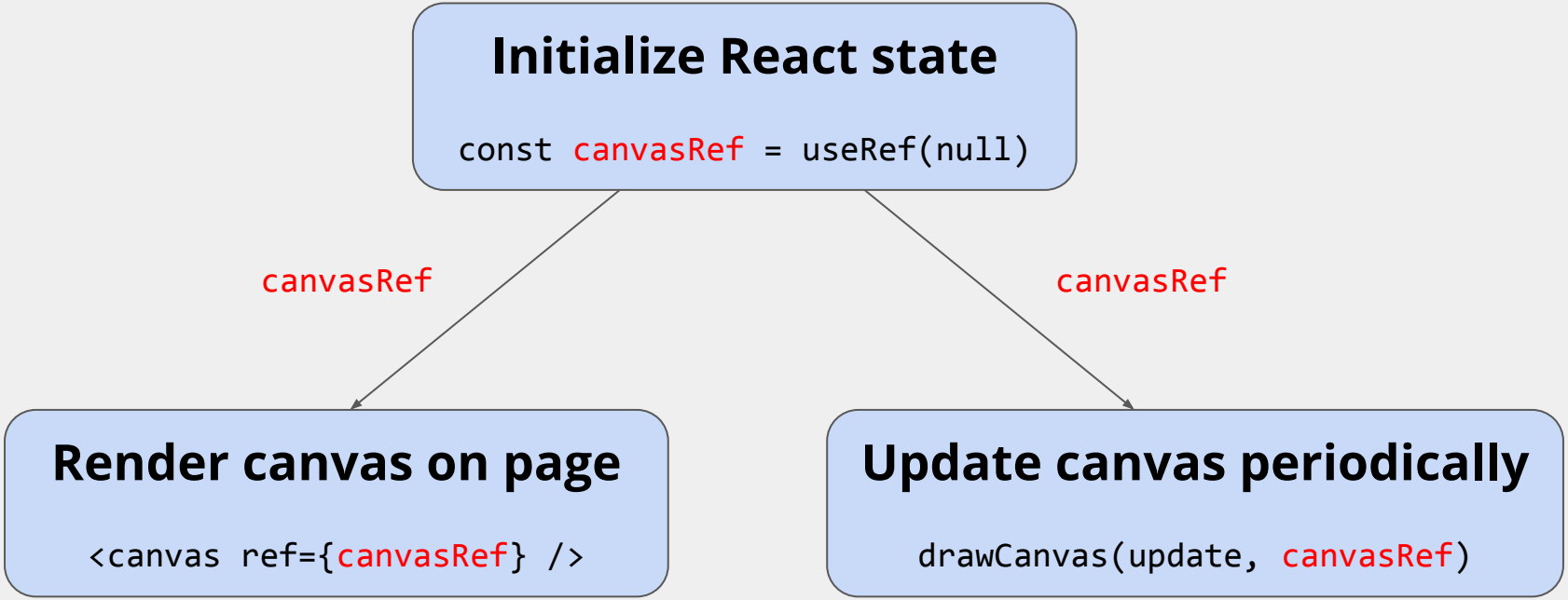
- **server socket** to send updates to client

- **spawn player** when a client socket connects

For now, we'll spawn in a player when they connect to Catbook, even if they're not on the Game page.

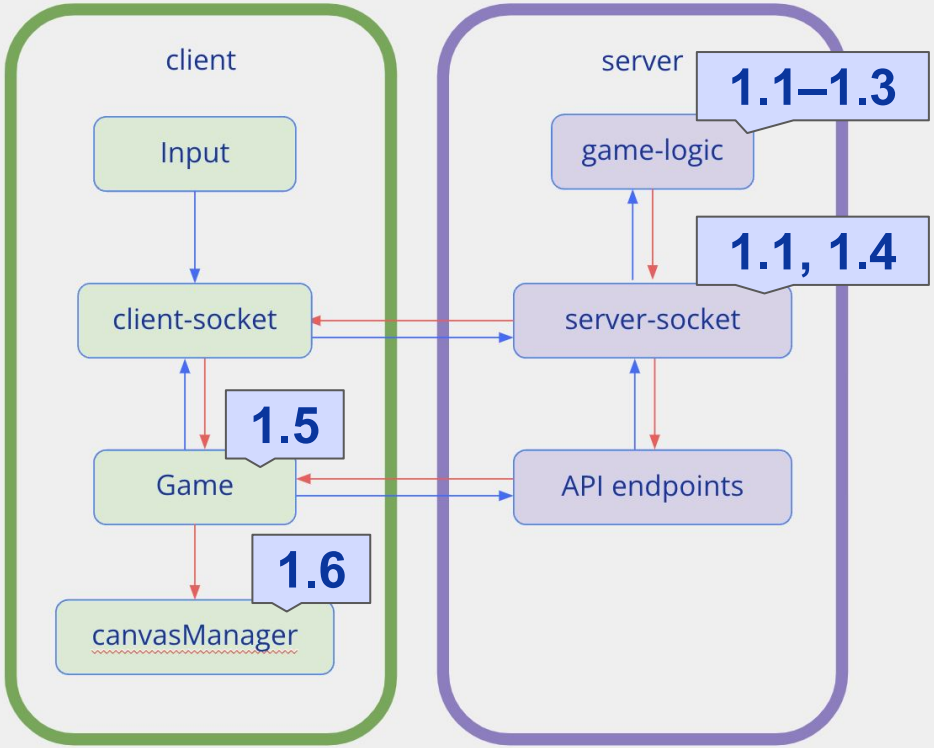
git fetch; git reset --hard; git checkout w10-step1

Render & Update Canvas at the same time: useRef

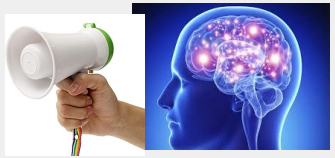


git fetch; git reset --hard; git checkout w10-step1

Step 1 - Let's break it down



Server



startRunningGame()
(1.1)

updateGameState()
(1.1) *(game loop)*

spawnPlayer()
(1.2, 1.4)

removePlayer()
(1.3)

sendGameState()
(1.1)

Joe

processUpdate()
(1.5)

drawCanvas()
(1.6)

git fetch; git reset --hard; git checkout w10-step1

Step 1 - Where is everything?

Step 1.1: Initialize game state; start game on server; send game updates to client game-logic.js (line 24, 43, 57) + server-socket.js (line 14)

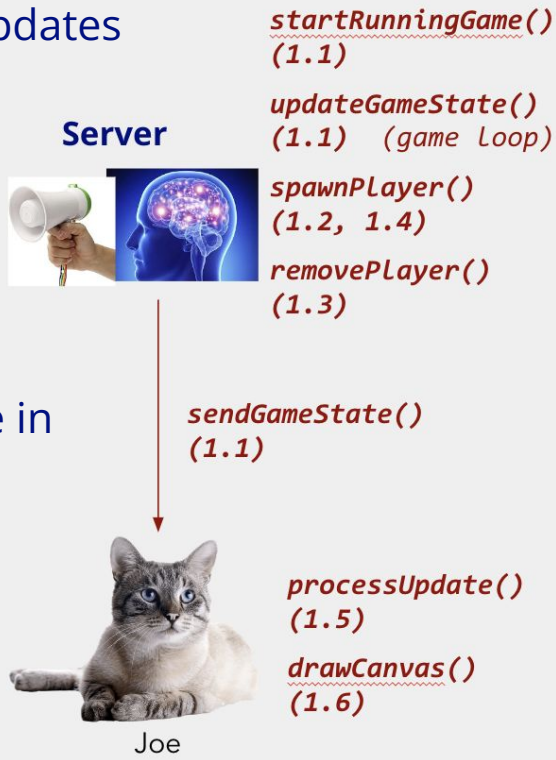
Step 1.2: Define `spawnPlayer()` in game-logic.js (line 34)

Step 1.3: Define `removePlayer()` in game-logic.js (line 51)

Step 1.4: Call `spawnPlayer()` when a user connects to the website in server-socket.js (line 34)

Step 1.5: Process updates from the server in the client in Game.js (line 12, 21)

Step 1.6: Draw players on the canvas in canvasManager.js (line 26, 43)



Your Turn

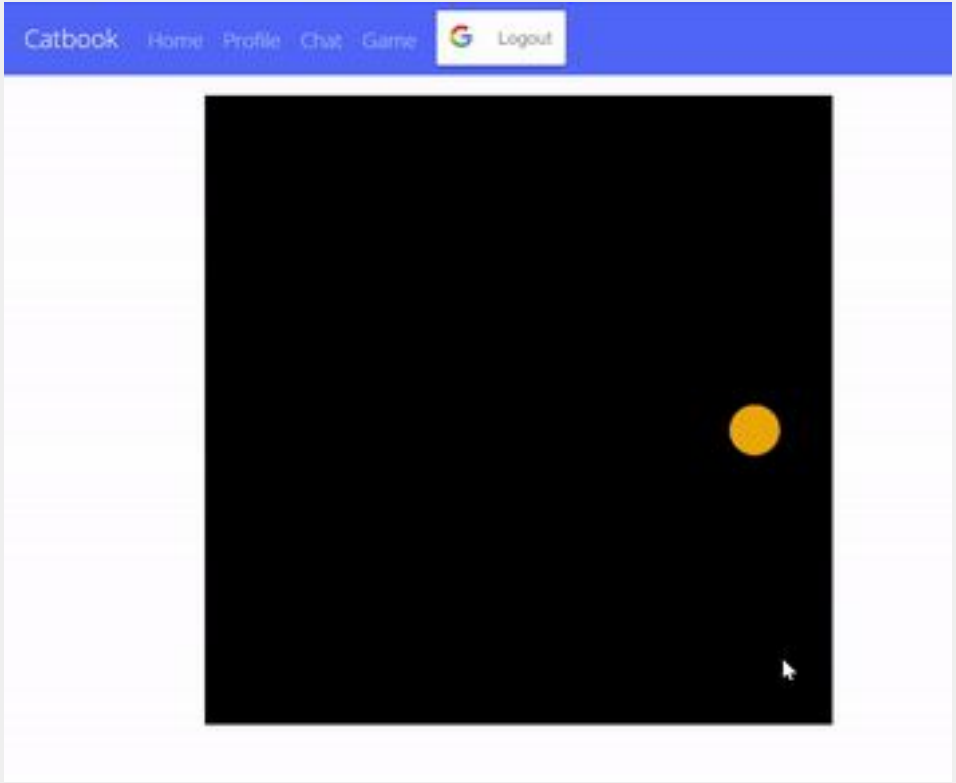
```
git reset --hard
git checkout w10-step2
```

Step 2: Let's move!

Reading and processing inputs

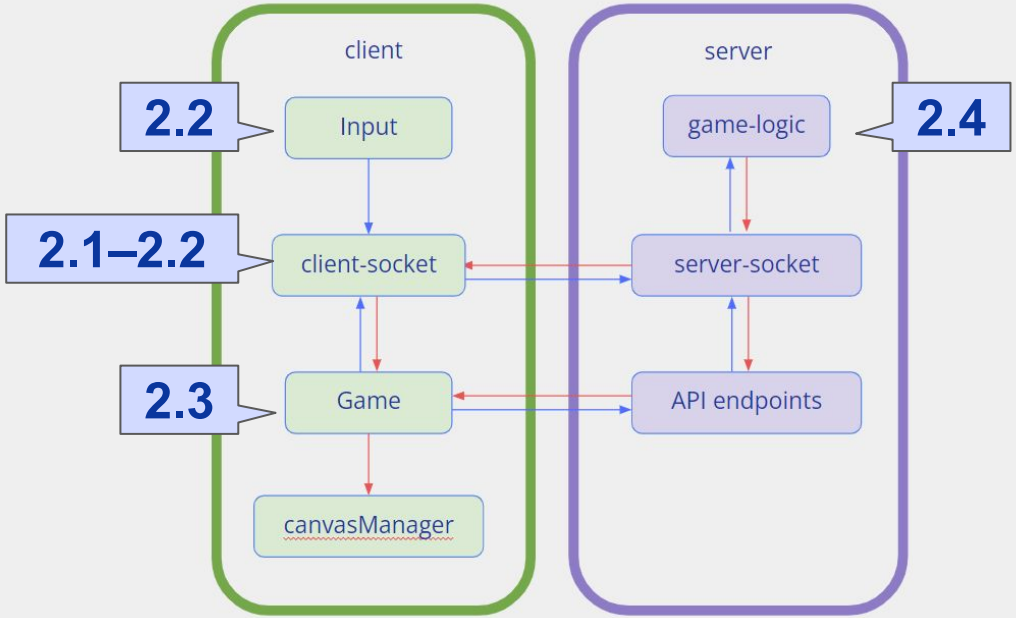
git fetch; git reset --hard; git checkout w10-step2

Step 2 - Moving!

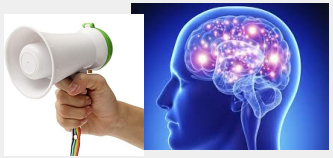


git fetch; git reset --hard; git checkout w10-step2

Step 2 - Let's break it down



Server



```
startRunningGame()
updateGameState()
spawnPlayer()
removePlayer()
```

movePlayer()
(2.4)

move()
(2.1)
(2.2)

```
sendGameState()
```

```
processUpdate()
drawCanvas()
```

addEventListener()
removeEventListener()
(2.3)

Joe

git fetch; git reset --hard; git checkout w10-step2

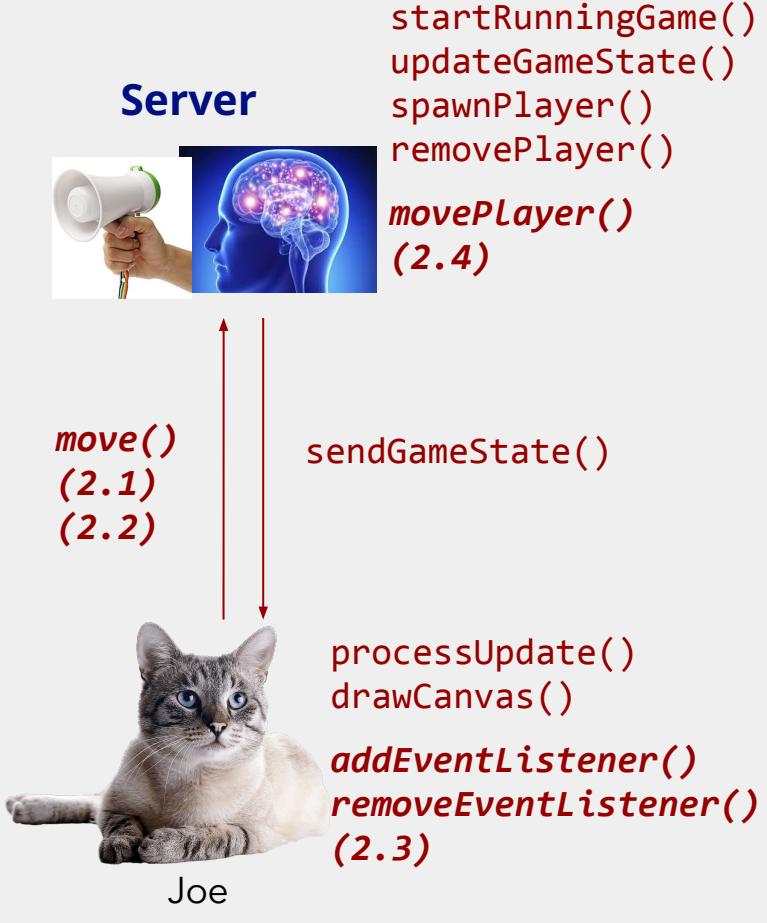
Step 2 - Where is everything?

Step 2.1: Send the move data to the server in client-socket.js (line 10) and input.js

Step 2.2: Handle keyboard inputs in input.js (line 4)

Step 2.3: Listen for keyboard inputs; and stop listening when the user leaves the page in Game.js (line 15)

Step 2.4: Define a function to move a player on the canvas given input directions in game-logic.js (line 47)



Your Turn

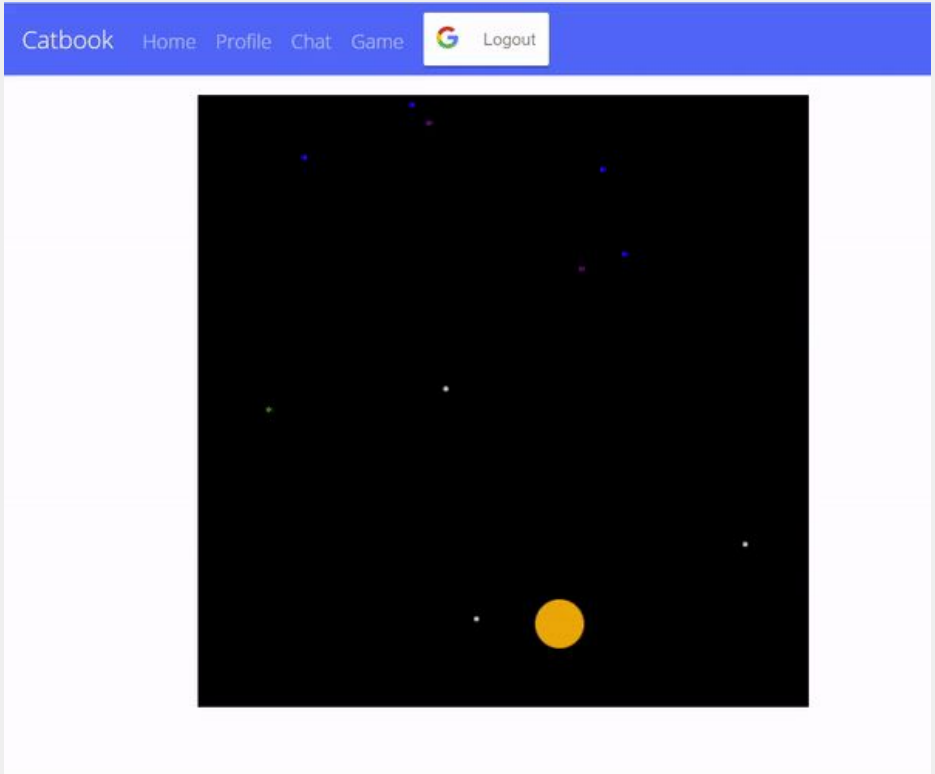
```
git reset --hard
git checkout w10-step3
```

Step 3: Eating food

I’m always a bit hungry

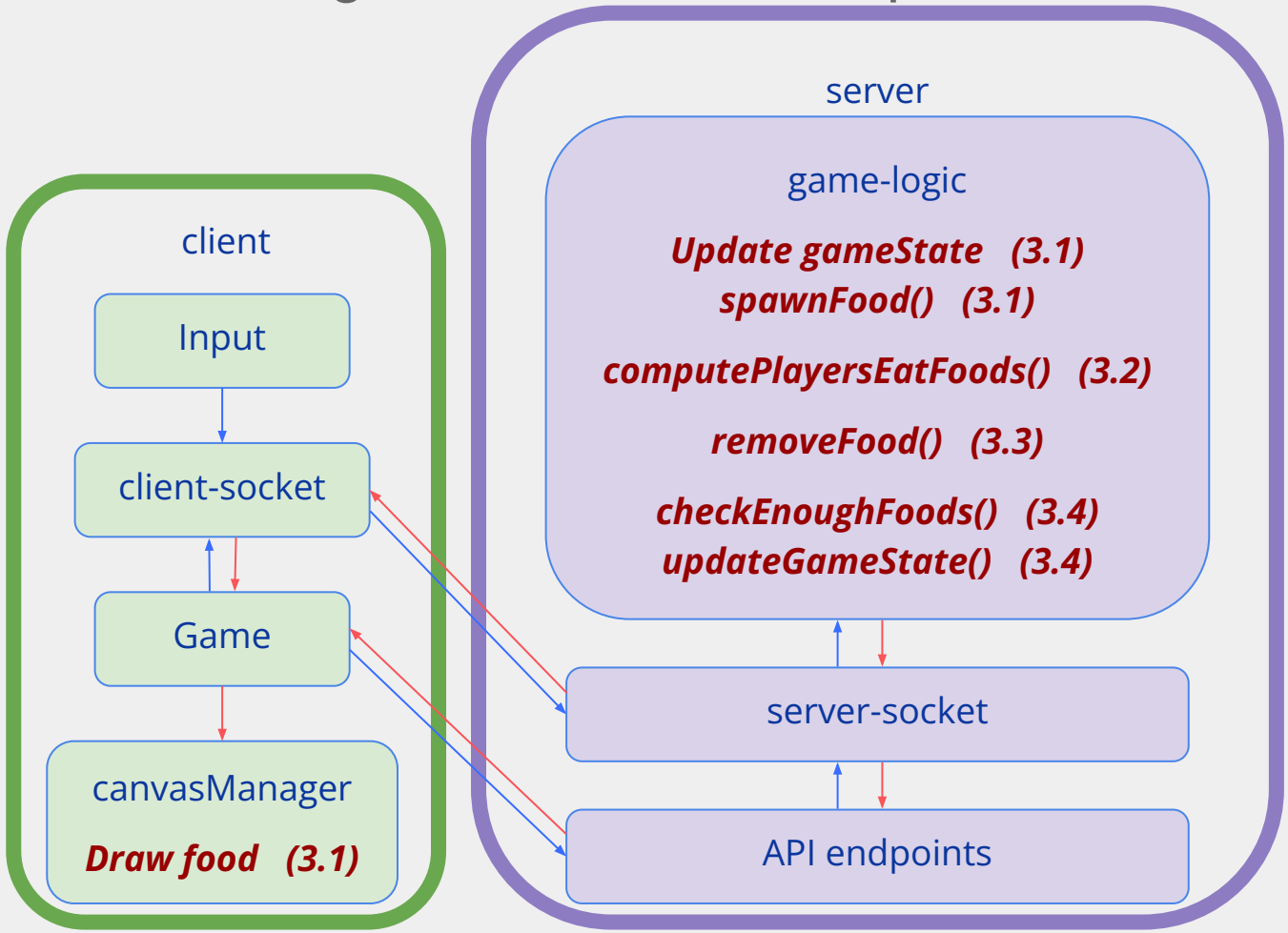
git fetch; git reset --hard; git checkout w10-step3

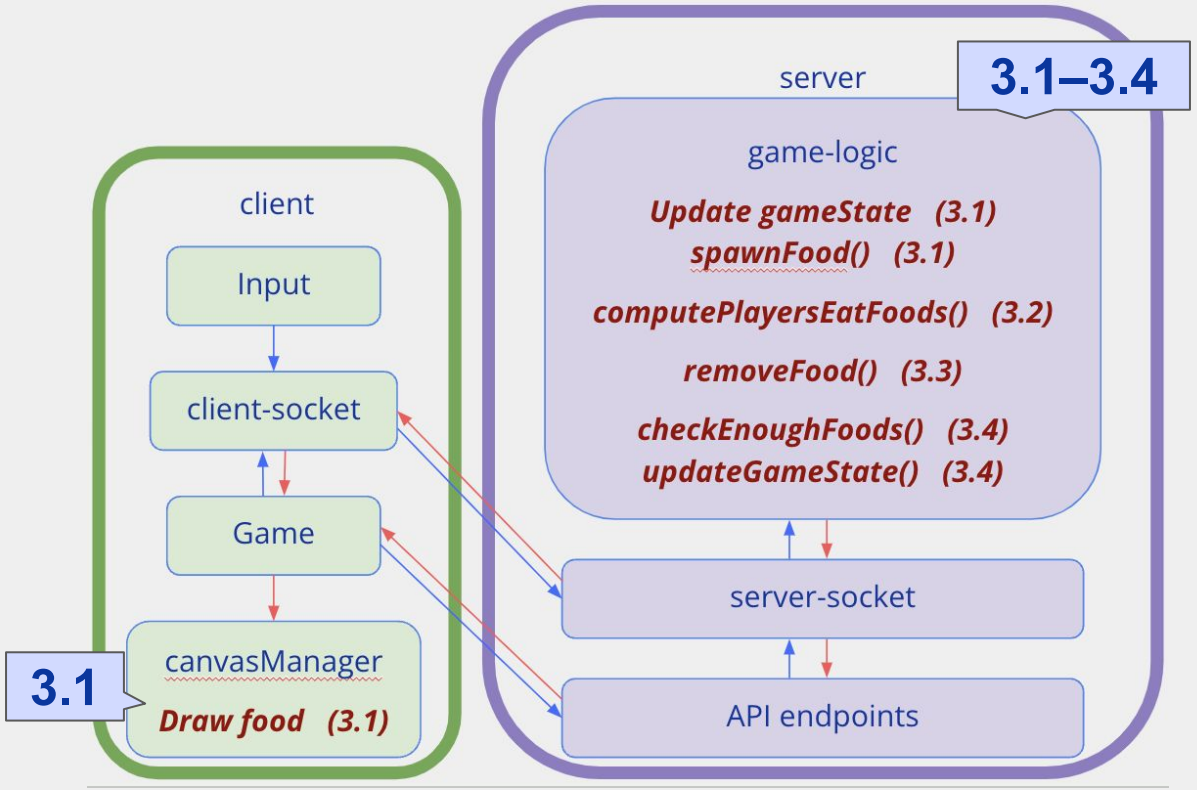
Step 3 - Eating (food)



git fetch; git reset --hard; git checkout w10-step3

Step 3 - Let's break it down




server
3.1–3.4
game-logic
Update gameState (3.1)
spawnFood() (3.1)
computePlayersEatFoods() (3.2)
removeFood() (3.3)
checkEnoughFoods() (3.4)
updateGameState() (3.4)
client
Input
client-socket
Game
canvasManager
Draw food (3.1)
3.1
server-socket
API endpoints

git fetch; git reset --hard; git checkout w10-step3

Step 3 - Let's break it down

Step 3.1: Add food to Gamebook

Step 3.2: Be able to check if any players are able to eat food

Step 3.3: Be able to remove food from the game

Step 3.4: Update game state to spawn food & check if players can eat food

git fetch; git reset --hard; git checkout w10-step3

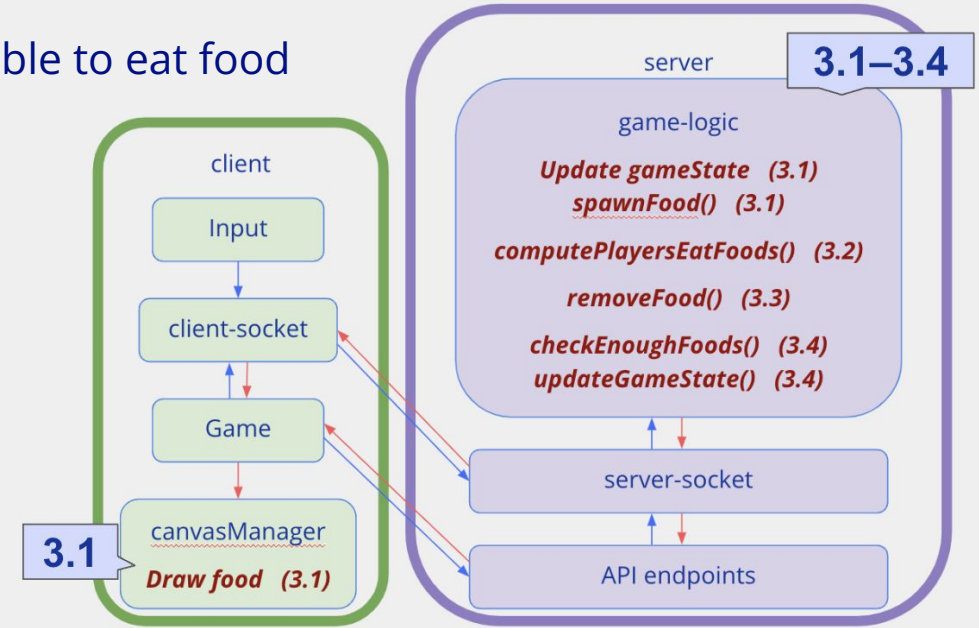
Step 3 - Let's break it down

Step 3.1: Add food

Step 3.2: Be able to check if any players are able to eat food

Step 3.3: Be able to remove food

Step 3.4: Update game state



git fetch; git reset --hard; git checkout w10-step3

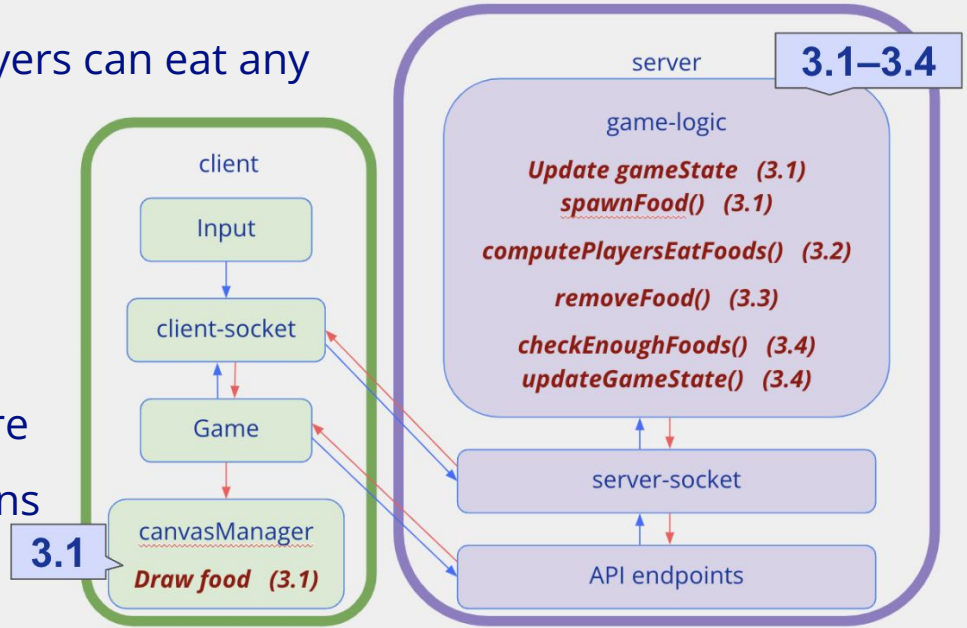
Step 3 - Let's break it down

Step 3.1: Add food to the game state; define a function to spawn foods; draw foods on the canvas in canvasManager.js (line 51) and game-logic.js (lines 59, 75)

Step 3.2: Define a function to check if any players can eat any foods in game-logic.js (line 45)

Step 3.3: Define and call a function to remove food in game-logic.js (lines 127, 38)

Step 3.4: Define a function to check if there are enough foods in the game & add food functions to game loop in game-logic.js (line 102, 112)



Your Turn

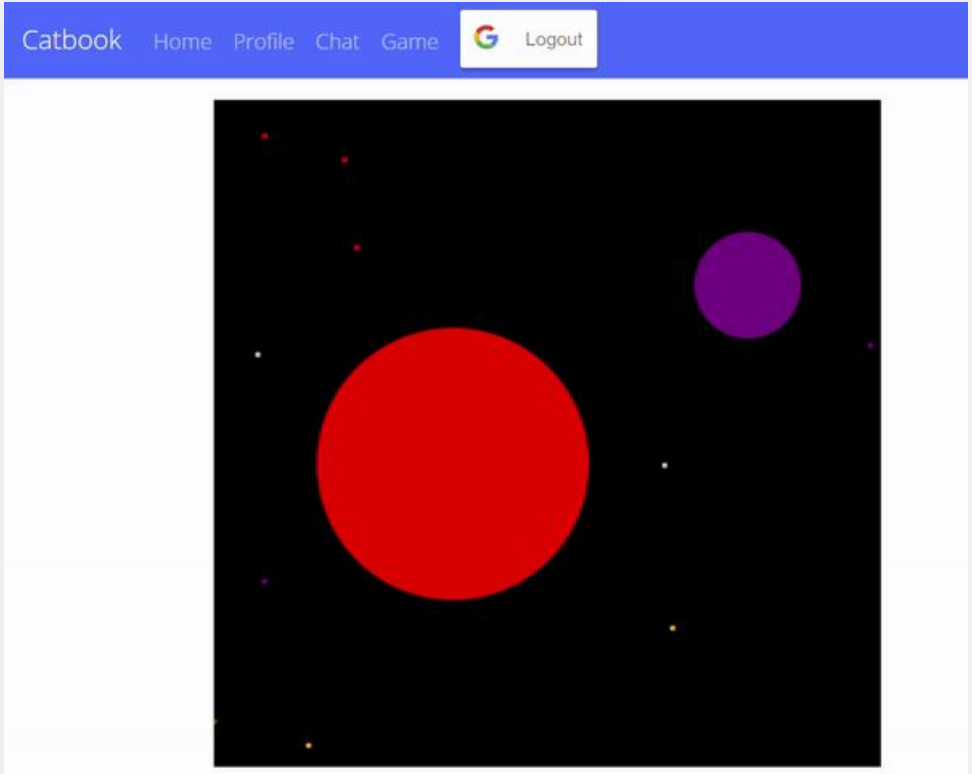
```
git reset --hard
git checkout w10-step4
```

Step 4: Eating other players

Eat or be eaten 😈

git fetch; git reset --hard; git checkout w10-step4

Step 4 - Eating (Blend It ;))



git fetch; git reset --hard; git checkout w10-step4

Step 4 - Let's break it down

Step 4.1:

Checks if a given player 1 can eat a given player 2. If yes, stores player 2 in `playersEaten`.

Step 4.2:

Tests if players can eat each other; stores which players are eaten.

Step 4.3:

Removes players who have been eaten.

Step 4.4:

Server will now check and remove any players who are eaten, every tick.

git fetch; git reset --hard; git checkout w10-step4

Step 4 - Let's break it down

Step 4.1: Fill out the `playerAttemptEatPlayer()` helper function in game-logic.js (line 28).

Checks if a given player 1 can eat a given player 2. If yes, stores player 2 in `playersEaten`.

Step 4.2: Fill out `computePlayersEatPlayers()` in game-logic.js (line 48).

Calls `playerAttemptEatPlayer()` on all pairs of players; stores which players are eaten.

Step 4.3: Call `removePlayer()` on each eaten player in game-logic.js (line 54).

Removes players who have been eaten.

Step 4.4: Update the game loop, once again, in game-logic.js (line 143).

Server will now check and remove any players who are eaten, every tick.

Your Turn

```
git reset --hard
git checkout w10-step5
```

Step 5: Respawning and Winning

Making the game replayable

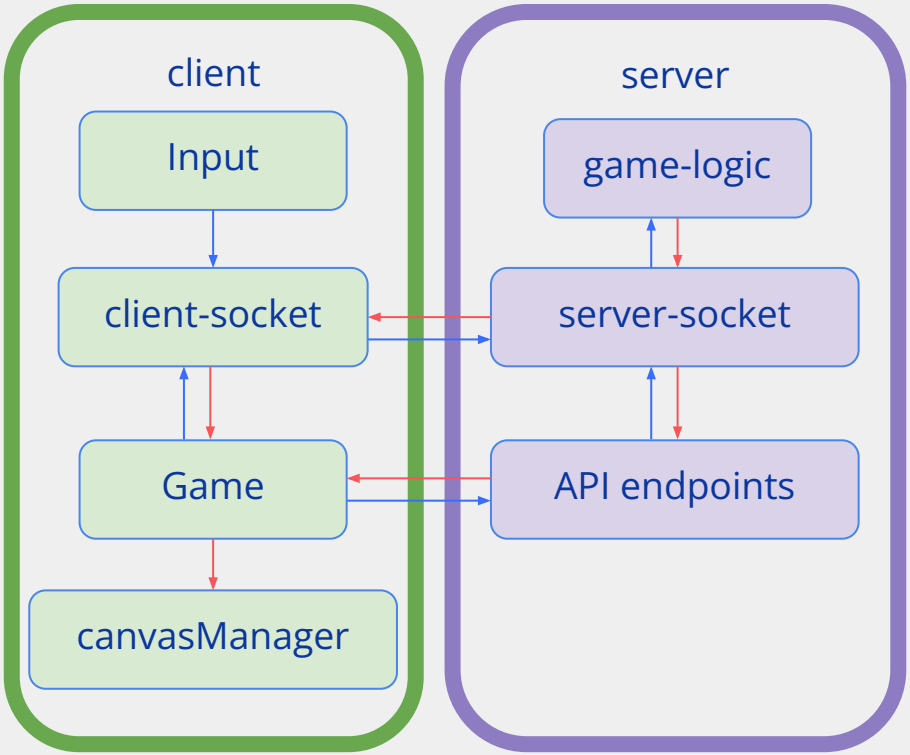
git fetch; git reset --hard; git checkout w10-step5

Step 5 - Respawning and Winning

2 ways to communicate client to server: sockets and apis

We used sockets for game updates because they need to be fast.

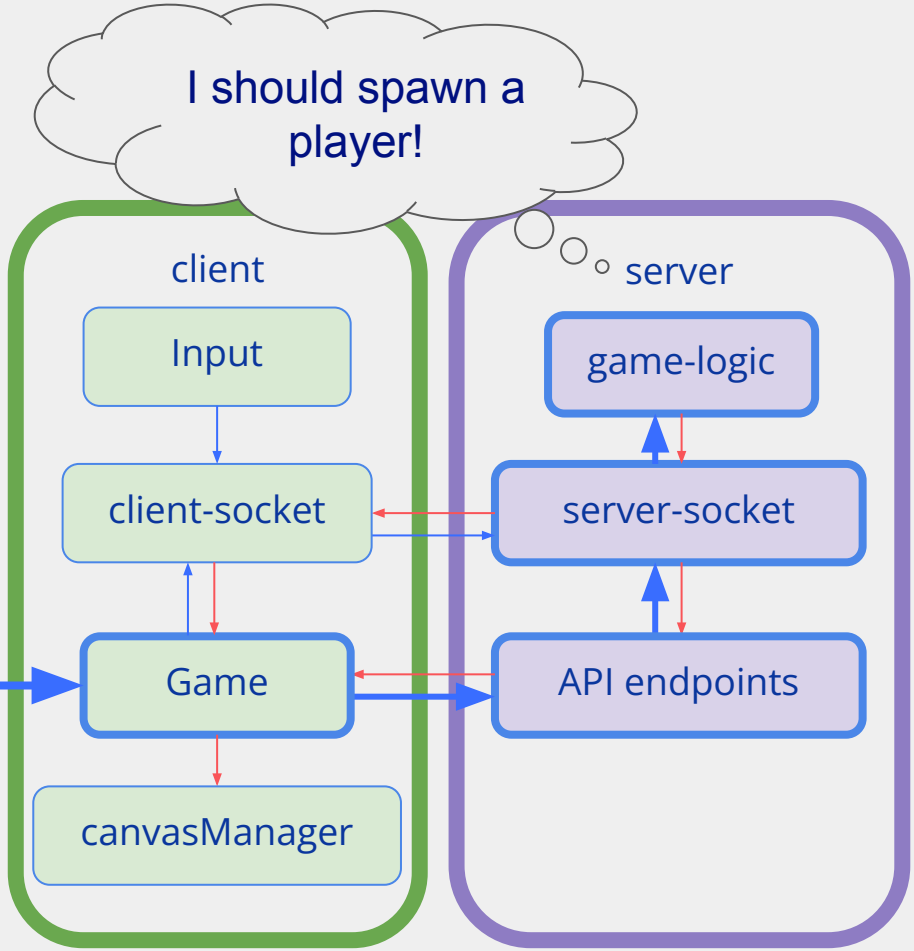
We'll use an API for spawning because we don't need speed (also to show you how you can use both :)).



git fetch; git reset --hard; git checkout w10-step5

Step 5 - Spawn Button

Clicking the spawn button here



git fetch; git reset --hard; git checkout w10-step5

Step 5 - What do we need to do?

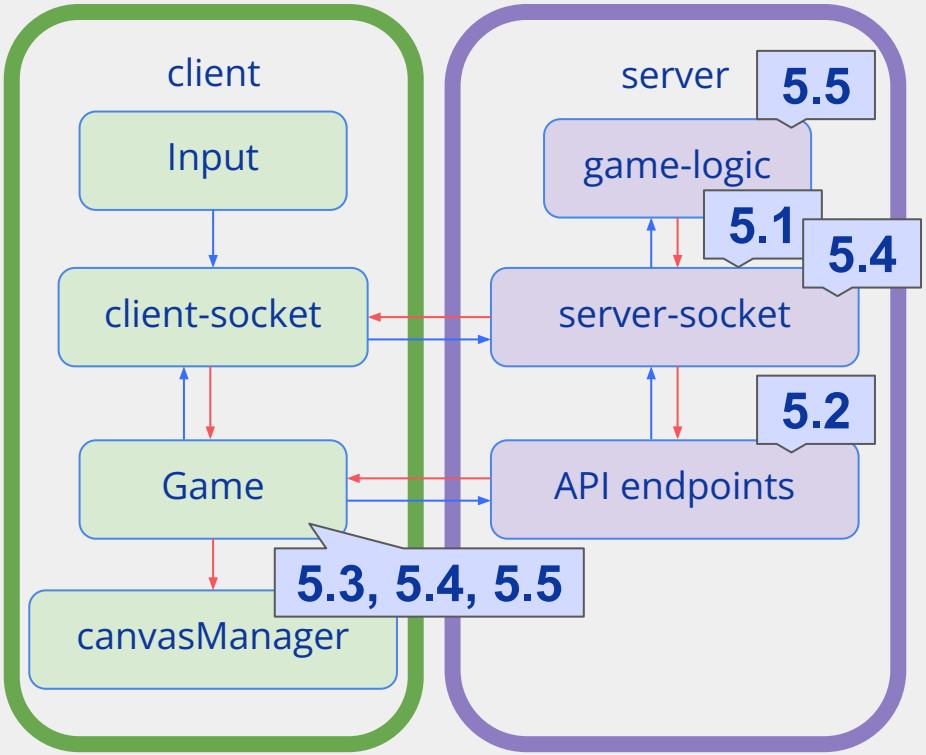
Step 5.1 Right now, players are being spawned when the socket connects. Let's remove that and make functions instead. server-socket.js (line 28-35 and line 41)

Step 5.2 Add API endpoints to use those functions. api.js (line 128)

Step 5.3 Add a spawn button on the Game page and make it call the API. Game.js (line 55)

Step 5.4 Make sure to cleanup! Game.js (line 24) and server-socket.js (line 57)

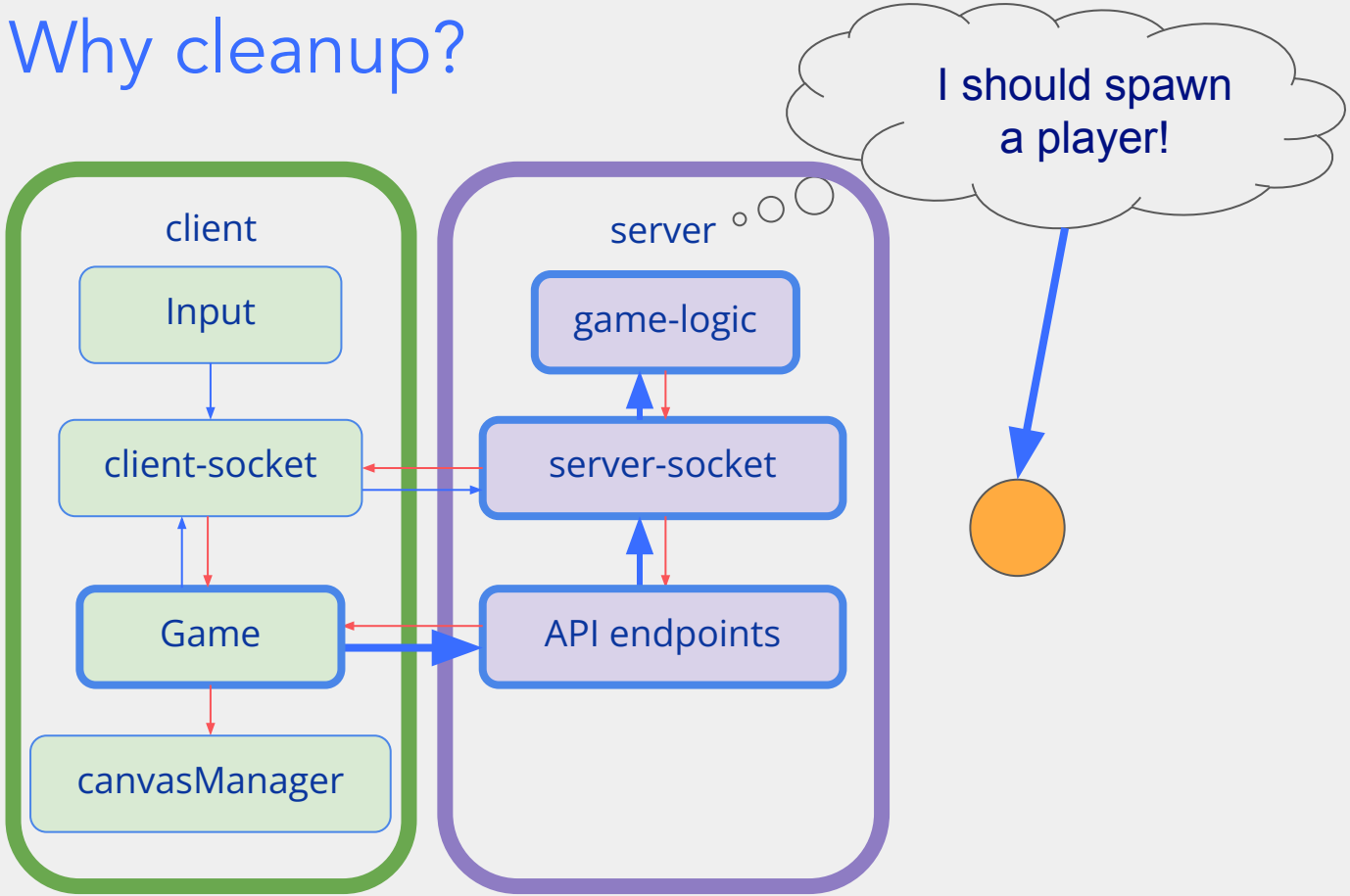
Step 5.5 Add winning! (check win on server and display winner on client!) game-logic.js (line 167) and Game.js (line 13, line 36, line 76)



git fetch; git reset --hard; git checkout w10-step5

Step 5.4 - Why cleanup?

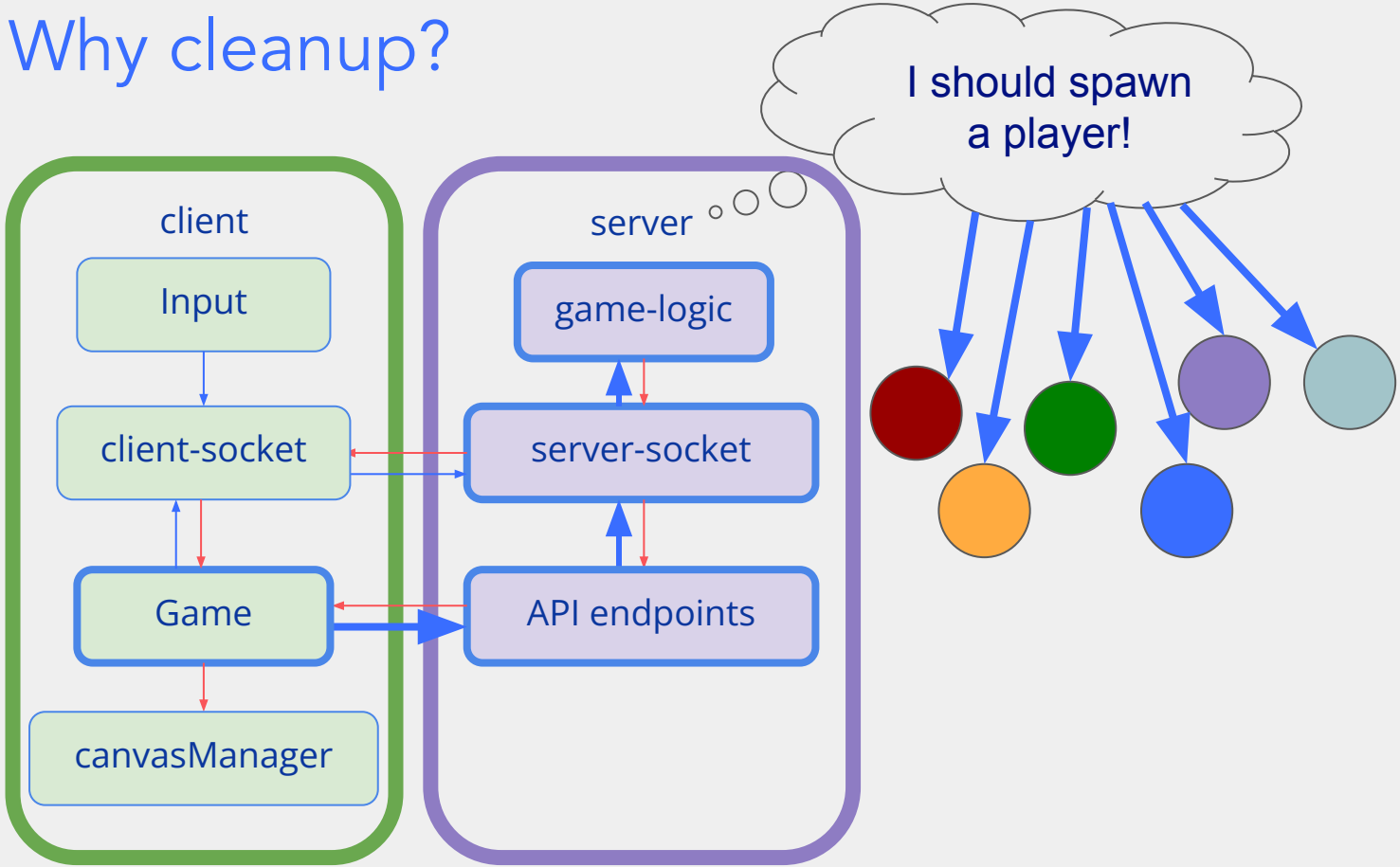
Client joins the game



git fetch; git reset --hard; git checkout w10-step5

Step 5.4 - Why cleanup?

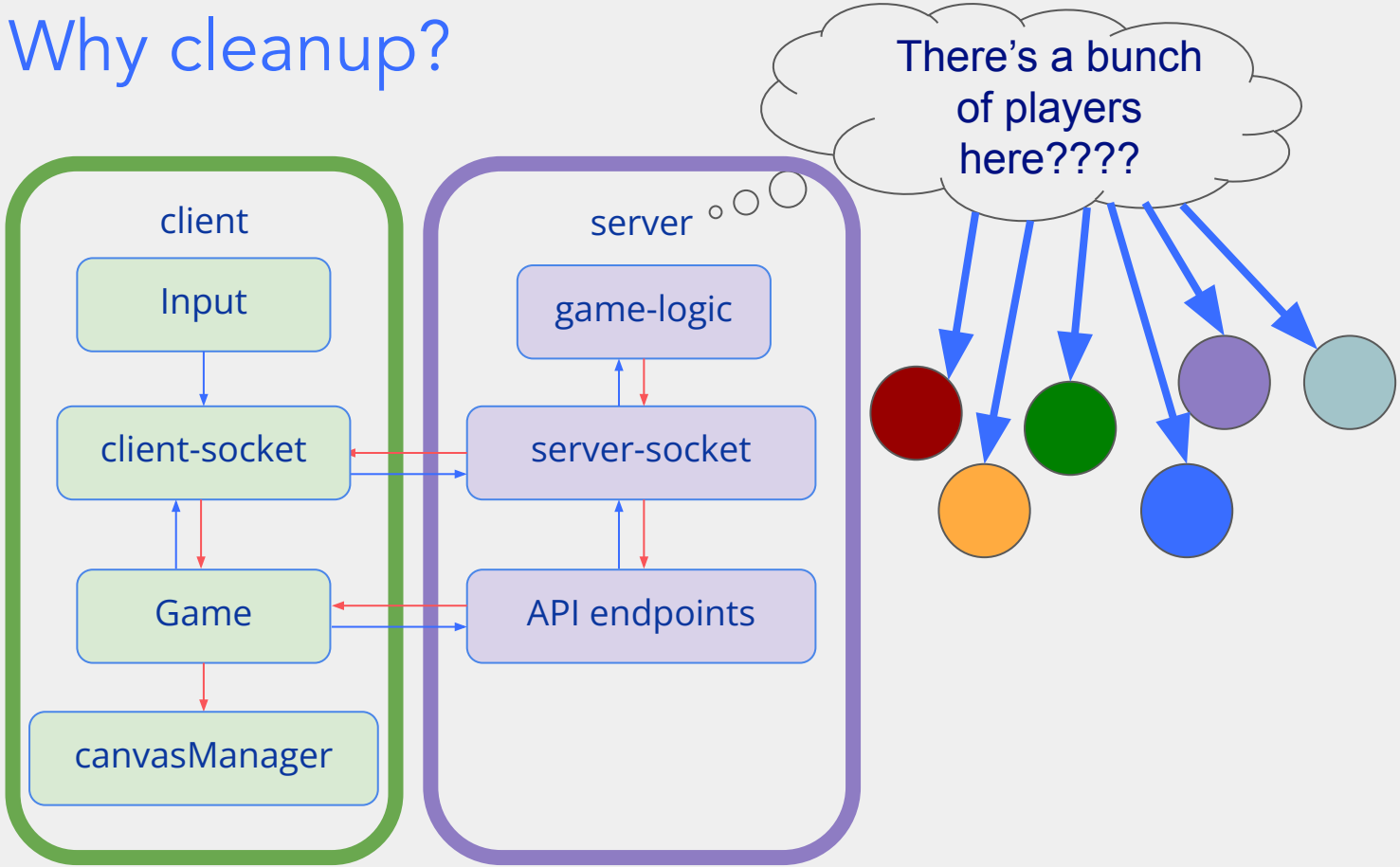
More clients join the game



git fetch; git reset --hard; git checkout w10-step5

Step 5.4 - Why cleanup?

Clients leave the game



git fetch; git reset --hard; git checkout w10-step5

Step 5 - Why to cleanup?

Joe

Server

git fetch; git reset --hard; git checkout w10-step5

Step 5 - How to cleanup?

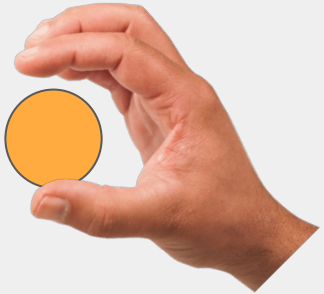
Joe

Server

git fetch; git reset --hard; git checkout w10-step5

Step 5 - How to cleanup?

Joe



Server

git fetch; git reset --hard; git checkout w10-step5

Step 5 - How to cleanup?

Joe

git fetch; git reset --hard; git checkout w10-step5

Step 5 - How to cleanup?

Joe

Server

git fetch; git reset --hard; git checkout w10-step5

Step 5 - How to cleanup?

Remove player when socket disconnects or when player navigates to a different page

Server

Gamebook Complete!

Wow that was tough!

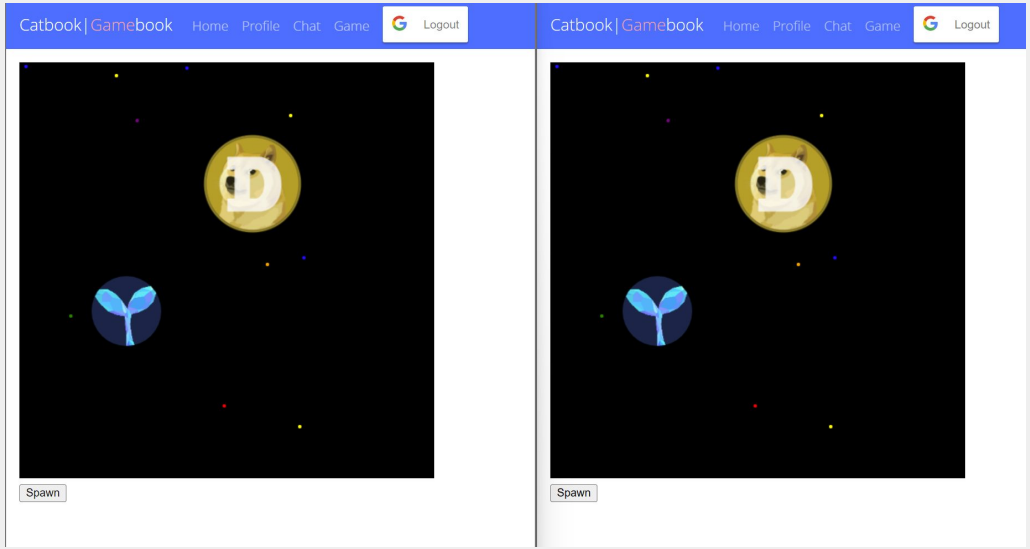
See the final product

```
git reset --hard
git checkout w10-complete
```

git fetch; git reset --hard; git checkout w10-complete

Some Extra Features

- Sprites
- Map bounds
- Server auto reset
- Extra styling (CSS)



weblab.is/example

Takeaways

- **Sockets** enable fast, live communication between the server and the client, while **API endpoints** are for slow, data communication
- **HTML Canvas** is a good way to render animations on the front end
- **Event listeners** on the client allow the website to take in user input
- All game logic should be done on the server
- The **game state**, stored on the server, is where the ground truth of the game should be stored
- We use a **server socket** to broadcast live updates to the clients, and use a **client socket** manager to receive updates from the server
- Upon a **component unmount** or a client **socket disconnect**, we must clean up the user from the game

Congrats!!

You made Gamebook!

Do not eat in the lecture hall :)

Be Back at 1:00pm :)