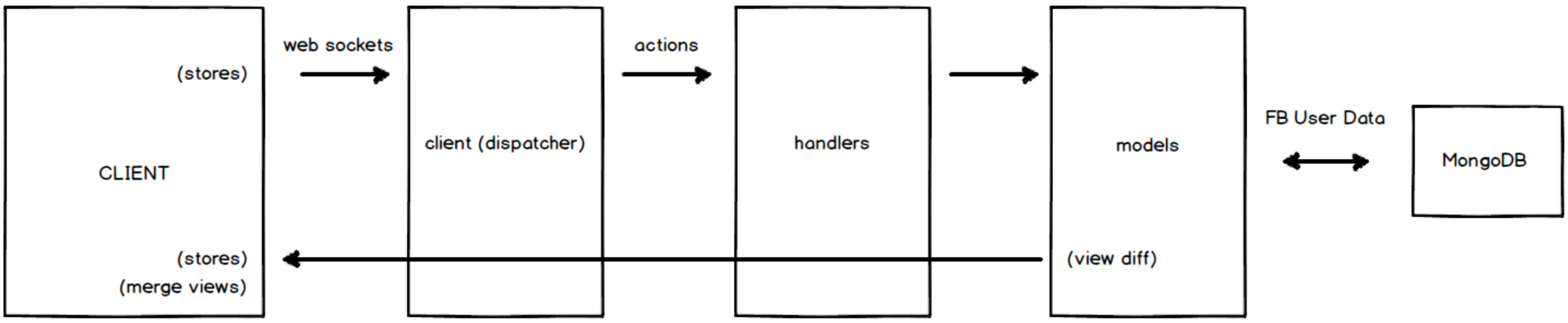
Backend Architecture

user interaction => callbacks => containers => stores =>
socket.io => client(dispatcher) => handlers => models => viewDiff => socket.io => stores => RxJs => containers => components



Backend Architecture: Dispatcher / Handlers / Models

EXAMPLE: PLAYING A CARD AND UPDATING VIEWS FOR EVERYONE PT 1

- In our card component - if someone played a card, then that constitues as an user interaction.
- That interaction is then going to be sent from the card component through a callback, and this callback will end up in a container. The containers are the components that are going to be responsible for interacting with our stores.
- Our container will then dispatch an action to the dispatcher - so it'll take that intent and wrap it up in an action.
- The dispatcher is then going to send that action to a store, into a store action handler.
- In some cases, the store isn't going to need to interact with the server - such as opening a popup login window. For other cases, the store will dispatch something to the server - such as playing a card.
- The store itself will say "socket.io! I have this action that I want you to handle!". It'll shoot that action right over into socket.io (web sockets), so now the action is officially out of the client.
- It will get sent to a class called "Client", which represents every individual connection to the server. It's just a class that handles things.
- Our Client on the server side is a dispatcher itself. So it receives the message from socket.io, and then dispatch the action to a handler.
- The handler is like a controller. It interacts with the model. It takes an action and invokes methods on the models.
- Depending on what state the Client is in, different handlers might be attached or might not be attached to a particular action. For example, if someone joins the lobby - the lobby handlers will be instantiated and will handle interactions with the lobby. But then when that person joins a game, a new instantiation of a game handler will be created, which then attaches itself to the game action, while the lobby handlers will be destroyed.
- Once the action is dispatched to a handler on the server, it then interacts with the model. So it says to the model, "Hey model, this guy wants to play a card, can he play a card?"

Backend Architecture: Dispatcher / Handlers / Models

EXAMPLE: PLAYING A CARD AND UPDATING VIEWS FOR EVERYONE PT 2: Dealing with individual views

- Our lobby area, game area, and individual players will all have their own "views". So when an interaction occurs,

the model will perform the interaction on itself, and then generate a new copy of its own view, and then compare its new view to its old view. It'll then create a "diff" between the new and the old views. Once the model is done performing its operations, it then creates a "viewDiff". The viewDiff might say something like "the messages array has 1 new item in it" or "the round has gone from null to having all this information" or "this player was the card czar and now he's not".

- So basically, the viewDiff just contains the properties of the view of whatever we're looking at, such as a game or a player, or the lobby.

- The viewDiff sends those changes straight back into socket.io, and to our client side. Once those changes have been received on the client side, all the clients that were listening to that particular view will receive the updated information regarding that view.