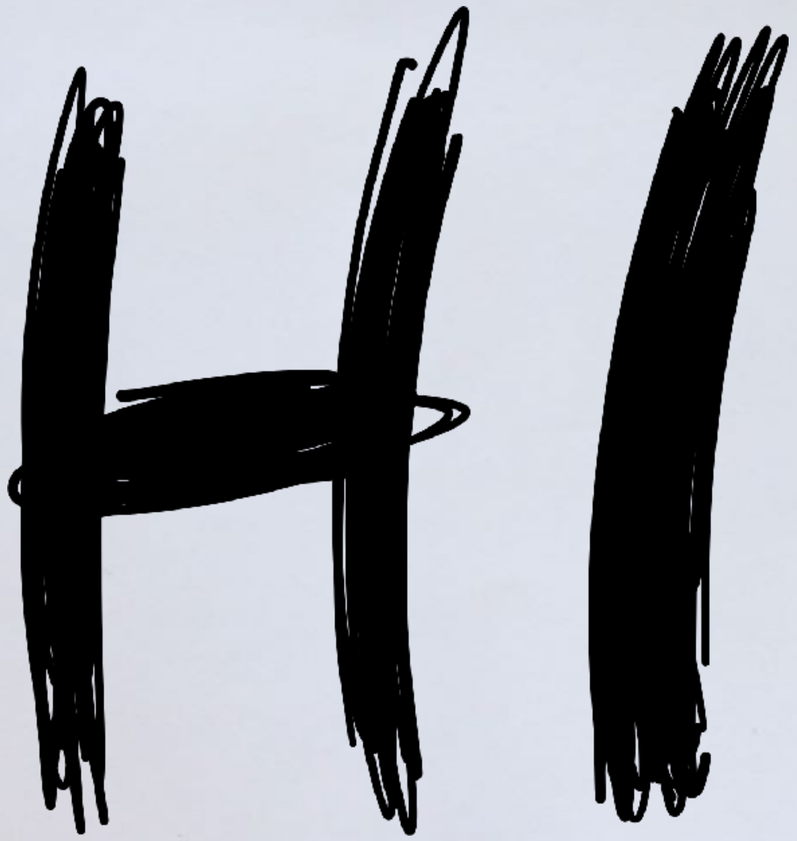


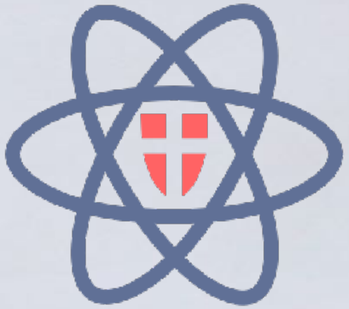
@ryyppy

Patrick Stapfer

Taming Legacy Codebases

Teaching about Type Systems & JS

GraphQL & React

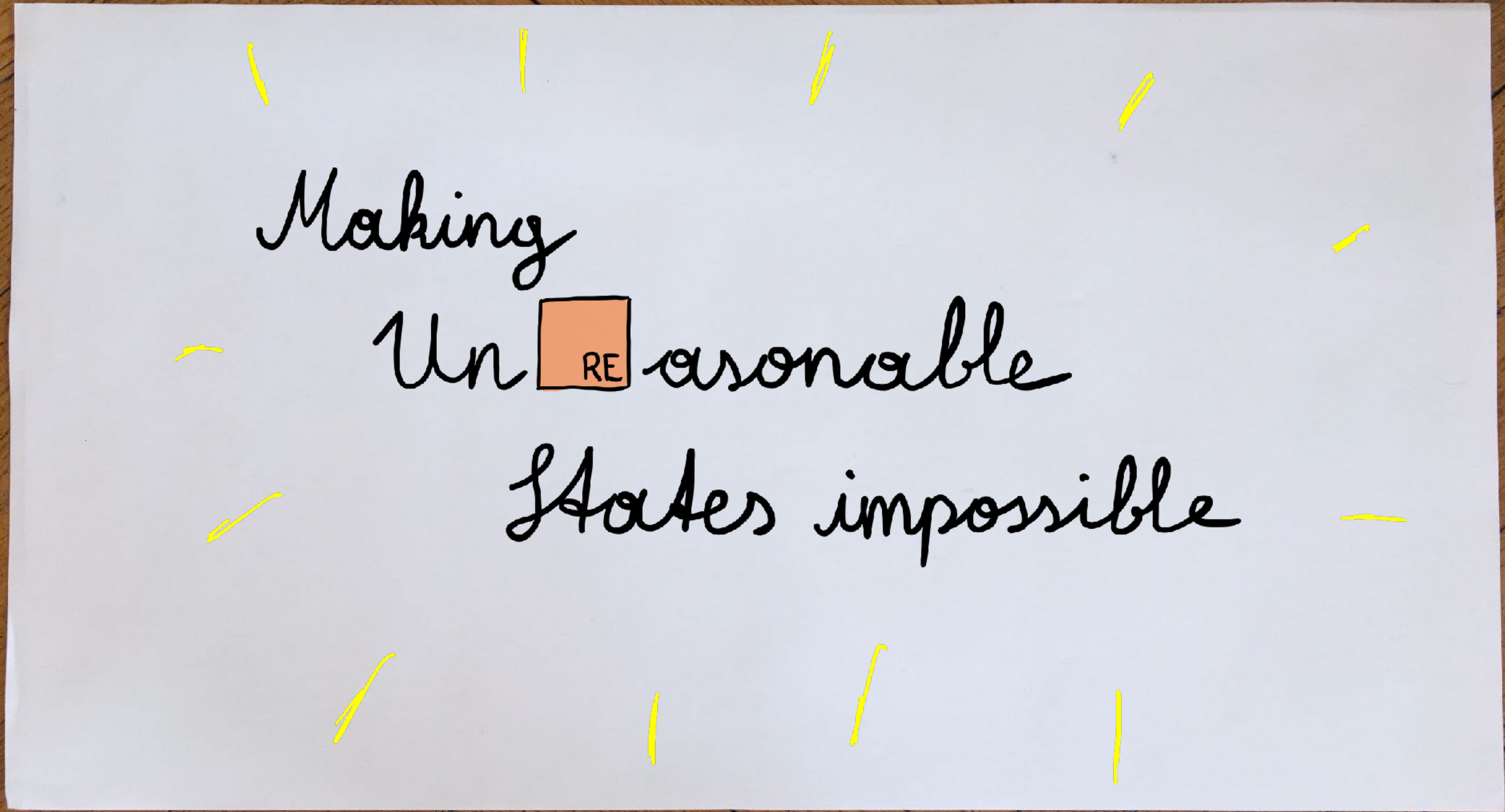


RE VIENNA

@ryyppy

ryyppy

@ryyppy ryyppy


Making
Un RE asonable
States impossible

Patrick Stapfer
@ryyppy

Help @jordwalke @_chenglou ! I need a zen-like tweet regarding impossible state via a strong type system to put in my upcoming presi 🤡

jordanIsNotAFunction
@jordwalke

Following

Replying to @ryyppy @_chenglou

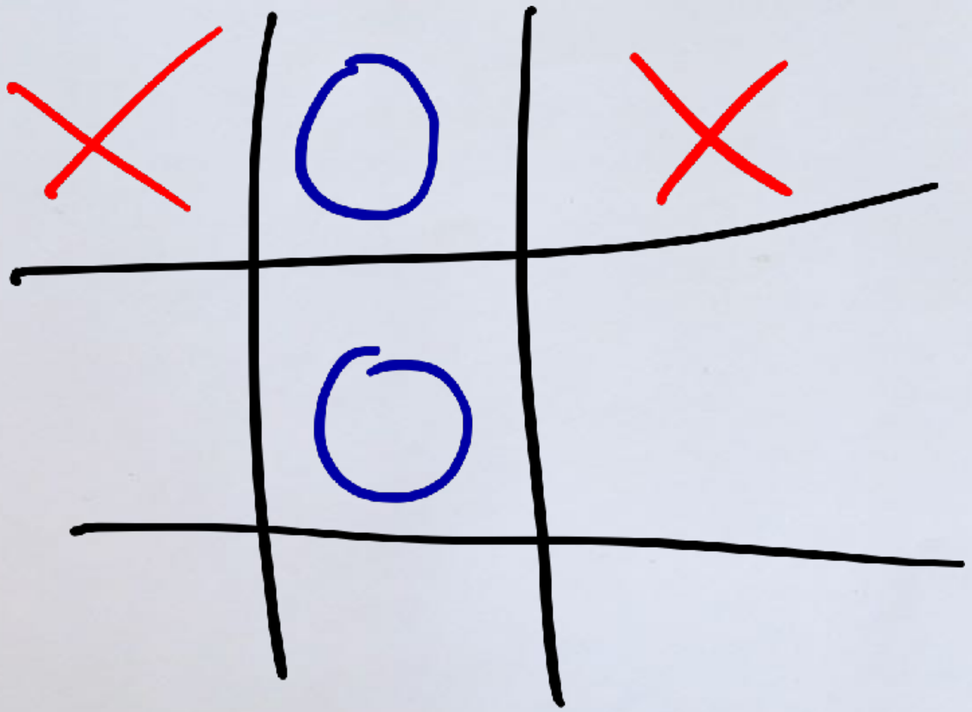
Oh man, on the spot?! That's tough. My best memeing comes spontaneously.

10:21 PM - 5 Sep 2017

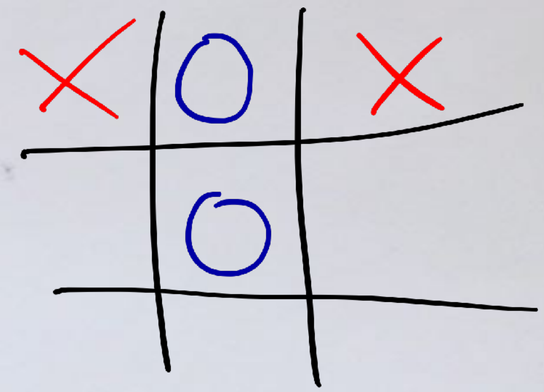
Well Researched Topic

- Effective ML Revisited - Yaron Minsky
- Types & Properties = Software: Designing with Types - Mark Seemann
  - Video
- Designing with Types: Making illegal states unrepresentable - Scott Wlaschin
- Making impossible states impossible - Richard Feldman
- Back to the Basics - Using Flow - A.Sharif
- Making Impossible States Impossible in ReasonML - A. Sharif

Learning By Doing (Reason & OCaml)



First Attempt (JS)



```
<Tictactoe>


const state = {
  board: [null, ..., null],
  progress: "turn",
  player: "cross",
};
```

Add some types w/ Flow…

```
type Token = "cross" | "circle" | "empty";

type Board = Array<Token>;

type Player = "cross" | "circle";

type Progress = "turn" | "win" | "draw";

type State = {
  board: Board,
  progress: Progress,
  player: Player | null,
};
```

Confusing Design...

```ts
type Token = "cross" | "circle" | "empty";
type Board = Array<Token>;
type Player = "cross" | "circle";
type Progress = "turn" | "win" | "draw";
type State = {
  board: Board,
  progress: Progress,
  player: Player | null,
};

const state: State = {
  board: [...],
  progress: "win",
  player: null,
};
```

???

Refactor Progress & Player Relation

```
type Progress =
  { type: "turn", player: Player }
  | { type: "win", player: Player }
  | { type: "draw" }


type State = {
  board: Board,
  progress: Progress,
  player: Player | null,
};
```

Types in Action (Flow)

```
const progress: Progress = {
  type: "turn",
  player: "cross"
};

switch (progress.type) {
 case "turn":
   const {player} = progress;
   return `Player ${player}'s turn`;
 case "win":
   const {player} = progress;
   return `Player ${player} won`;
};
```

"Pattern Matching" in Flow

switch is no expression

progress = typeof Object

```
switch (progress.type) {
  case "turn":
    const {player} = progress;
    return `Player ${player}'s turn`;
  case "winner":
    const {player} = progress;
    return `Player ${player} won`;
};
```

flow infer is brittle

verbose syntax

typooooooos

No checks for exhaustiveness

Let's design the state in Reason!

```
type token = Cross | Circle | Empty

type board = ...

type player = Cross | Circle

type progress =
  | Turn(player)
  | Win(player)
  | Draw

type state = {
  board: board,
  progress: progress
};
```

Types in Action (Reason)

```
let progress = Turn(Cross);


switch (progress) {
  | Turn(p) => "Player" ++ p_to_str(p) ++ "'s turn"
  | Win(p) => "Player" ++ p_to_str(p) ++ "won"
  | Draw => "It's a draw!"
};
```

```
switch (progress.type) {
 case "turn":
   const {player} = progress;
   return `Player ${player}'s turn`;
 case "win":
   const {player} = progress;
   return `Player ${player} won`;
};
```

What about the Board? Naive Edition.

```
type board = list(token)

let ok: board = [ Empty, Empty,... , Empty] (9)
let weird: board = [ Empty, Cross, Circle ]
let doh: board = [ Empty ]
```

+ More Generic
+ Kinda Clever

- **Cognitive Load**
- **Potentially unsafe code**
- **"More tests" to write**

Make unrepresentable Board Impossible!

```
type row = (token, token, token)
type board = (row, row, row)

let r1: row = (Empty, Empty, Empty) /* okay */
let r2: row = (Empty, Cross) /* fail */
let r3: row = (Empty, Cross, Cross, Cross) /* fail */
```

+ Type safe / no surprises

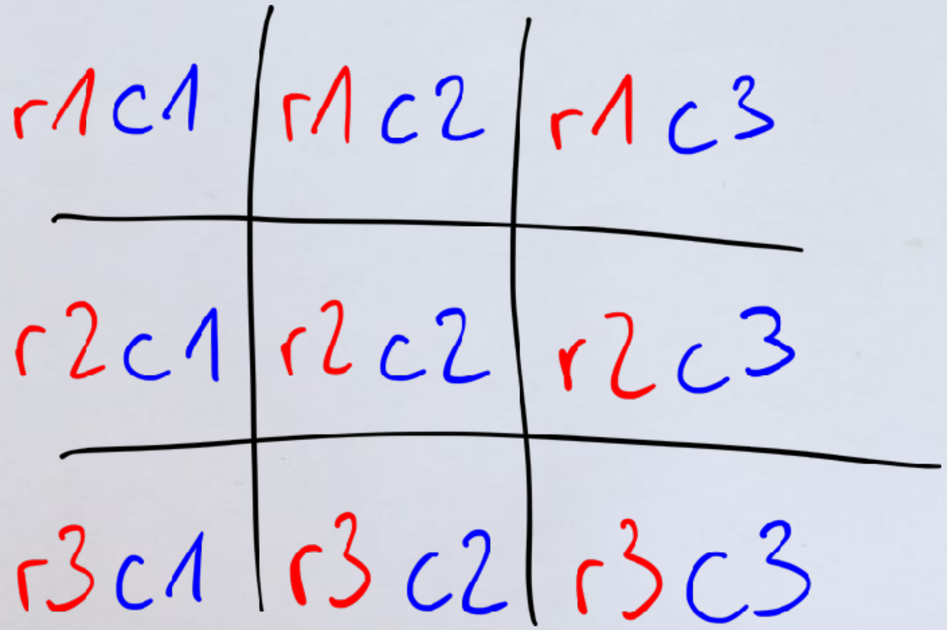
+ "Less tests" to write

- More verbose (but dead simple) code

Helper types

```
type colId = C1 | C2 | C3

type rowId = R1 | R2 | R3
```



Get a token from board (Column, Row)

```
getToken :: board -> rowId -> colId -> token
```

```
let getToken = (board, rid, cid) => {
  let (r1, r2, r3) = board;
  let fromRow = ((t1, t2, t3), cid) =>
    switch cid {
    | C1 => t1
    | C2 => t2
    | C3 => t3
    };
  switch rid {
  | R1 => fromRow(r1, cid)
  | R2 => fromRow(r2, cid)
  | R3 => fromRow(r3, cid)
  };
};
```

Add a token to the board

```
let updateBoard = (board, rid, cid, value) => {
  let (r1, r2, r3) = board;
  switch rid {
  | R1 =>
    let r = updateColumn(r1, cid, value);
    (r, r2, r3);
  | R2 =>
    let r = updateColumn(r2, cid, value);
    (r1, r, r3);
  | R3 =>
    let r = updateColumn(r3, cid, value);
    (r1, r2, r);
  };
};
```

```
let updateColumn = (row, cid, value) => {
  let (t1, t2, t3) = row;
  switch cid {
  | C1 => isEmptyToken(t1) ? (value, t2, t3) : row
  | C2 => isEmptyToken(t2) ? (t1, value, t3) : row
  | C3 => isEmptyToken(t3) ? (t1, t2, value) : row
  };
};
```

```
let isEmptyToken = (square: token) =>
  switch square {
  | Empty => true
  | _ => false
  };
```

Conclusion

More Rigid Design

More KISS than DRY

Forces edge-cases to be handled

Alternatives:

Combine both worlds

`boardToList <-> listToBoard`

Cheng Lou
@_chenglou

Following

Replying to @ryyppy @jordwalke

Remember high school? Different solutions to a problem was fine, but differing units in the end means you probably misunderstood the problem

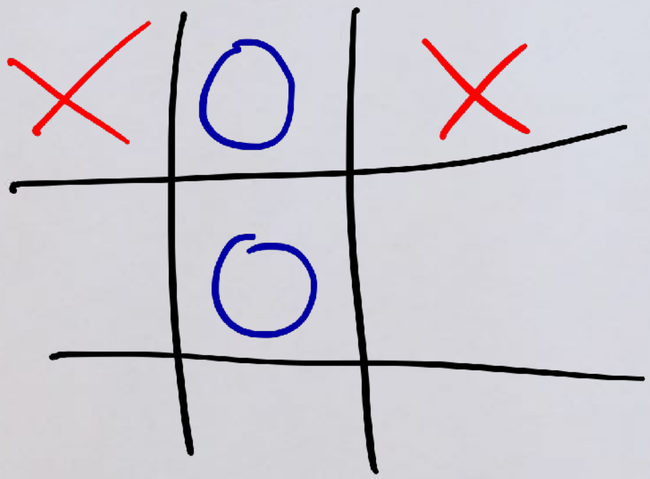
10:35 PM - 5 Sep 2017

Do we have more time for one more thing?

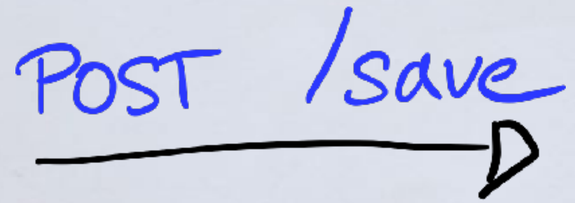
I wanted to show another example with async querying... but I had no server :-(

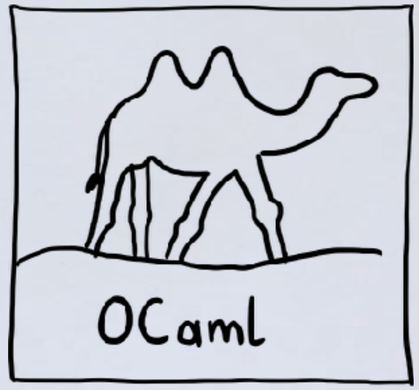
INTRODUCING

TicTacToe as a Service (TTTaaS)



Reason
Frontend





SERVER

Loader Component™

```
<TTTLoader/>
```

```
type loaderState = {
  data: option(tictactoeState),
  loading: bool
}
```

Loader Component™

```
<TTTLoader/>
```

```
type remoteData =
  | NotAsked
  | Pending
  | Success(Game.tictactoeState)
  | Error(string)
```

```
type loaderState = remoteData
```

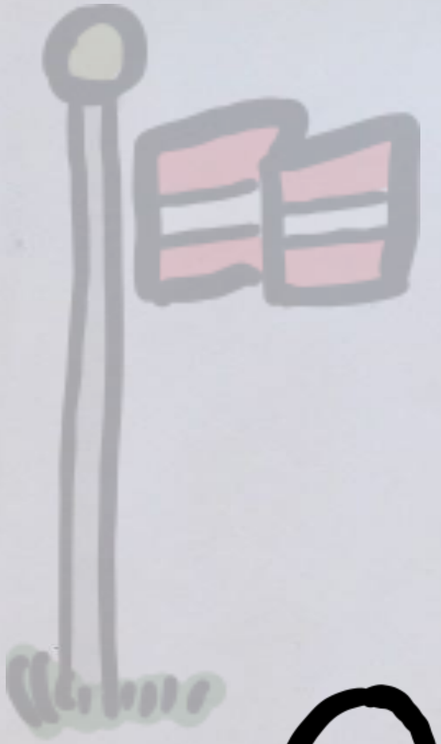
```
render: ({state}) =>
    switch state {
    | NotAsked => <div> ("Starting Application" |> se) </div>
    | Loading => <div> ("Loading Game..." |> se) </div>
    | Error(errorMsg) => <div> (errorMsg |> se) </div>
    | Success({board, progress}) => <Tictactoe board progress />
    }
```

Source Code for Game + Server:

http://github.com/ryyppy/reason-tictactoe

Join our Community

https://discord.gg/reasonml



THANKS!

QUESTIONS?

PATRICK STAPFER

@ryyppy ryyppy