Elixir

Présentation du projet →

Valentin Kaelin

14.06.2023

Rappel du projet

Canvas de dessin collaboratif en temps réel.

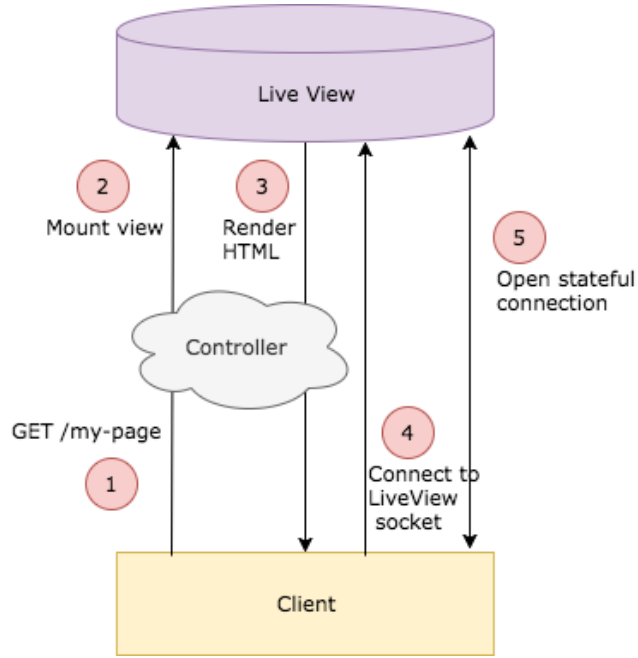
- Projet en Elixir avec le framework web Phoenix
- Canvas partagé
- Synchronisation à chaque point et pas à chaque fin de trait
- Tests de montée en charge

Démonstration

pmw.fly.dev

Architecture globale

- Projet Phoenix: MVC classique
  - Rendu côté serveur
  - Aucune base de données utilisée
  - Gestion du state dans la mémoire
- Canvas en temps réel -> Websocket
  - LiveView
  - Petit code JavaScript côté client



Source: elixirschool.com

Backend (1)

Premier rendu de la page

Controller pour les couleurs

```elixir
defmodule PmwWeb.PageController do
  use PmwWeb, :controller

  @colors [
    '#FFFFFF',
    '#E4E4E4',
    '#888888',
    # ...
  ]

  def home(conn, _params) do
    render(conn, :home, layout: false, colors: @colors)
  end
end
```

Templating avec HEEx

```
<div class="fixed bottom-0 left-0 ml-3 mb-3 z-50">
  <div id="colors" class="hidden">
    <%= for color <- @colors do %>
      <button
        data-color={color}
        class="color"
      >
        <div style={"background-color: #{color}"}></div>
      </button>
    <% end %>
  </div>
  <button id="choose-color">
    <svg color="#FFFFFF"><!-- ... --></svg>
  </button>
</div>

<div id="container"></div>
```

Backend (2)

Gestion des Websockets

```
defmodule PmwWeb.CanvasChannel do
  use PmwWeb, :channel

  def join("canvas:points", _payload, socket) do
    state = Canvas.getAll()
    {:ok, state, socket}
  end

  def handle_in("first_point", payload, socket) do
    broadcast_from(socket, "first_point", payload)
    Canvas.add_initial(payload["id"], payload["x"], payload["y"], payload["color"])
    {:reply, {:ok, %{}}, socket}
  end

  def handle_in("new_point", payload, socket) do
    broadcast_from(socket, "new_point", payload)
    Canvas.add(payload["id"], payload["x"], payload["y"])
    {:reply, {:ok, %{}}, socket}
  end
end
```

Backend (3)

Stockage du state dans un Agent

```elixir
defmodule Canvas do
  use Agent

  def start_link(_) do
    Agent.start_link(fn -> %{} end, name: __MODULE__)
  end

  def add_initial(line_id, x, y, color) do
    Agent.update(__MODULE__, fn state ->
      state
      |> Map.put(line_id, [x, y])
      |> Map.put("#{line_id}-color", color)
    end)
  end

  def add(line_id, x, y) do
    Agent.update(__MODULE__, fn state ->
      Map.put(state, line_id, [x, y | Map.get(state, line_i
    end)
```

- Format de l'id d'une ligne: date-uuid

Exemple: 1686126356197-5898e1a7-4576-4043-86a2-b38fa3debdc4

- Stockage des lignes:

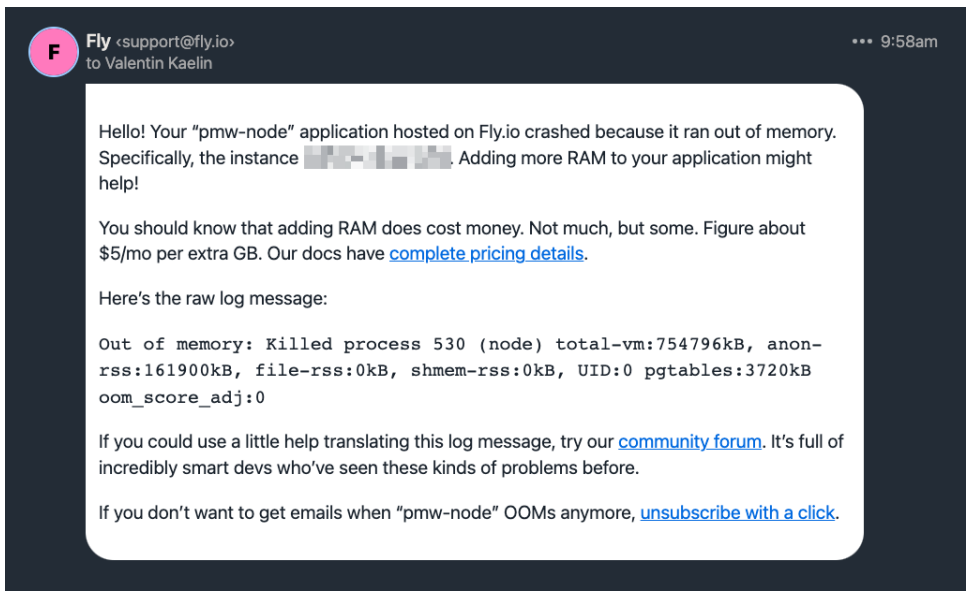
Tableau de points [x1, y1, x2, y2, ...]

Frontend

```
import { Socket } from "phoenix";

const socket = new Socket("/socket", {
  params: { token: window.userToken },
});
socket.connect();

const channel = socket.channel("canvas:points", {});
channel
  .join()
  .receive("ok", (state) => {
    // Affiche le state actuel au chargement de la page
  })

channel.on("first_point", (payload) => {
  // Crée la ligne
});

channel.on("new_point", (payload) => {
  // Ajoute le point à la ligne
});
```

Déploiement

- Déployé sur Fly.io (PaaS dans le genre d'Heroku)
- Free tier: 1 Shared CPU, 256MB RAM
- Test de déployer la version Node de l'application:

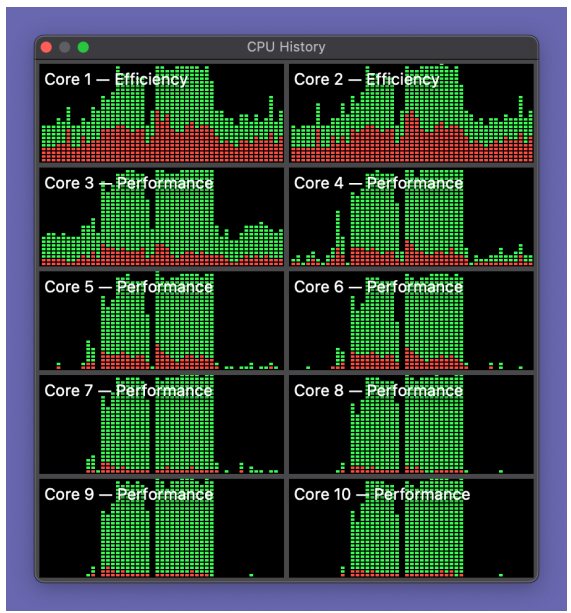


Tests de montée en charge

Utilisation d'Artillery

Types de tests possibles:

- Simples requêtes HTTP
- Websocket
- Tests "end-to-end" avec un navigateur headless



Problèmes rencontrés

- Tests Websockets avec Phoenix: compliqué
- Bottleneck des tests "end-to-end": le CPU de mon ordinateur

Test Artillery

```yaml
config:
  target: https://pmw.fly.dev
  phases:
    - duration: 60
      arrivalRate: 2
      maxVusers: 10
      name: 10 Users drawing
  engines:
    playwright: {}
  processor: "./draw.js"
scenarios:
  - engine: playwright
    flowFunction: "draw"
    flow: []
```

draw.yml

```javascript
async function pickColor(page) {
  await page.locator("#choose-color").click();
  const randomColor = random(16) + 1;
  await page.locator(`button.color:nth-child(${randomColo
}

async function line(page) {
  await page.mouse.move(randomX(), randomY());
  await page.mouse.down();

  await page.mouse.move(randomX(), randomY());
  await page.mouse.up();
}

async function draw(page) {
  await page.goto("https://pmw.fly.dev/");

  for (let i = 0; i < NB_LINES; i++) {
    await pickColor(page);
    await line(page);
  }
}
```

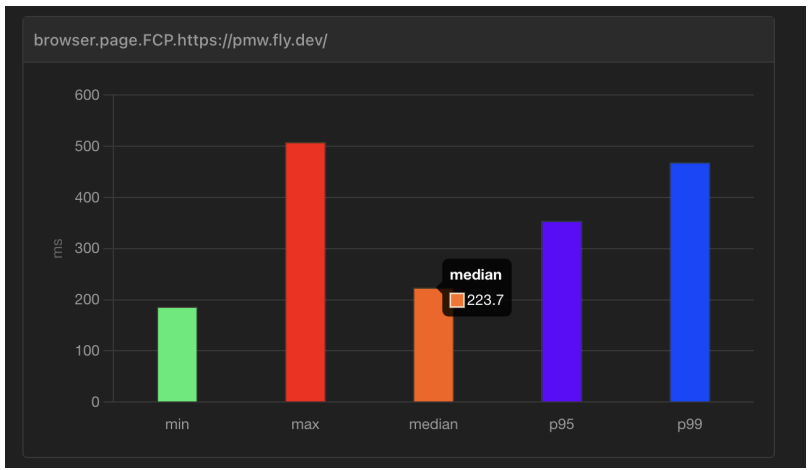
draw.js

Résultats (1)

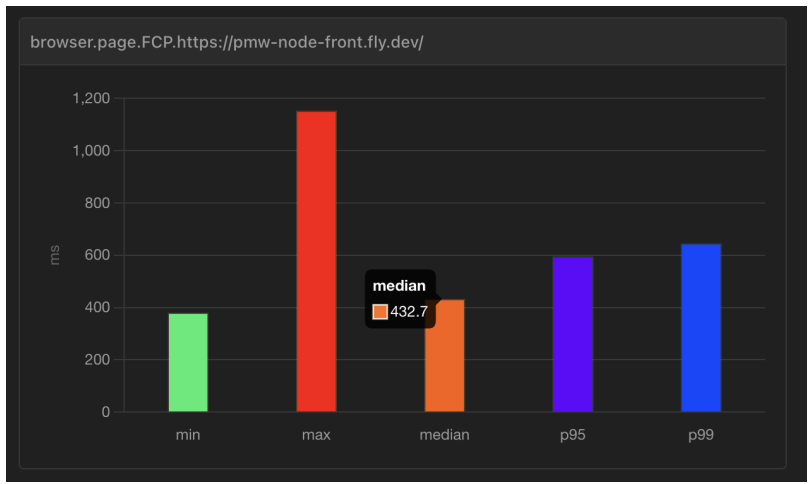
Résultats (2)

FCP: First Contentful Paint

Elixir



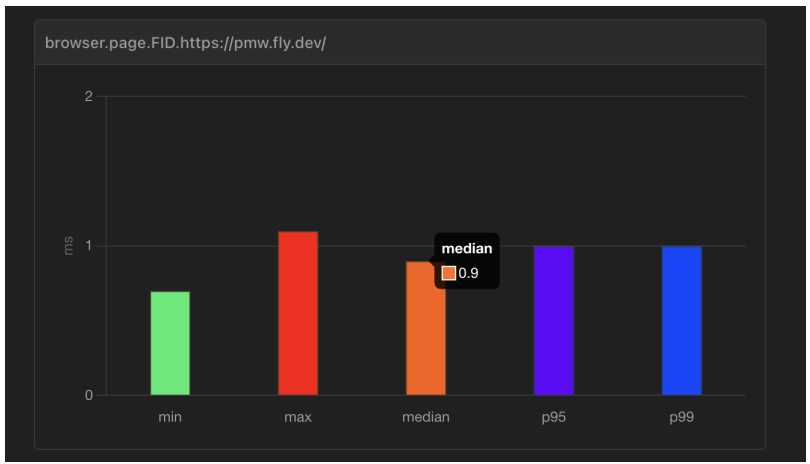
Node



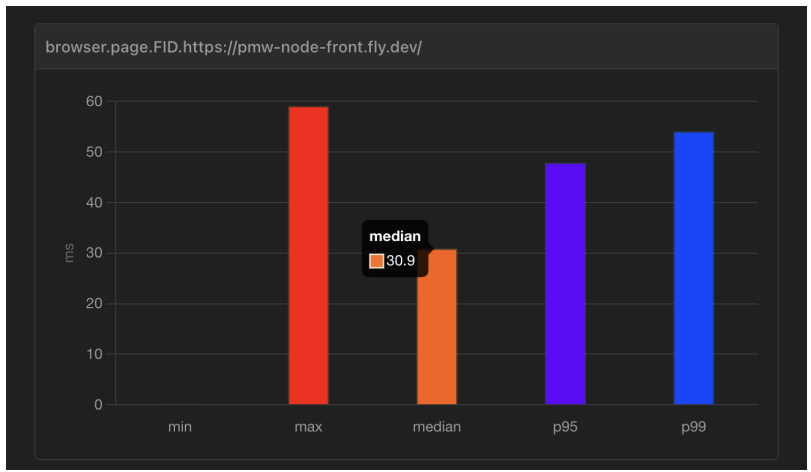
Résultats (3)

FID: First Input Delay

Elixir



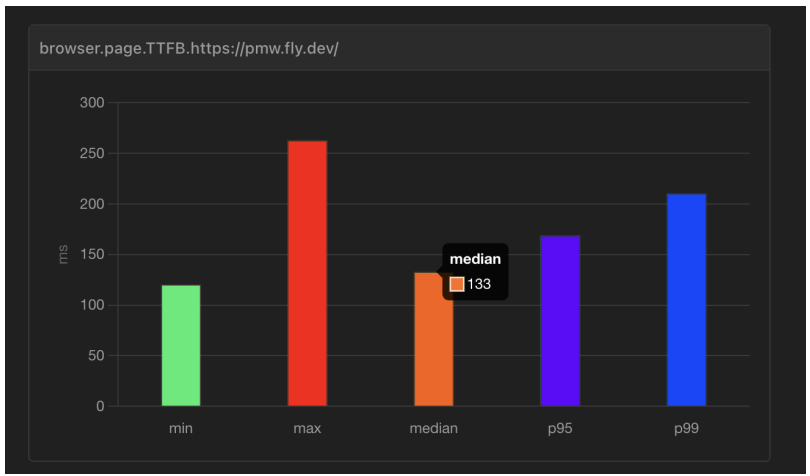
Node



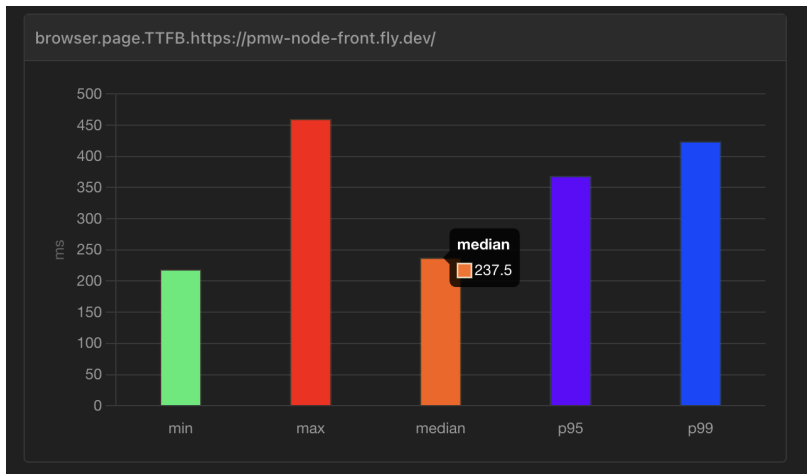
Résultats (4)

TTFB: Time To First Byte

Elixir



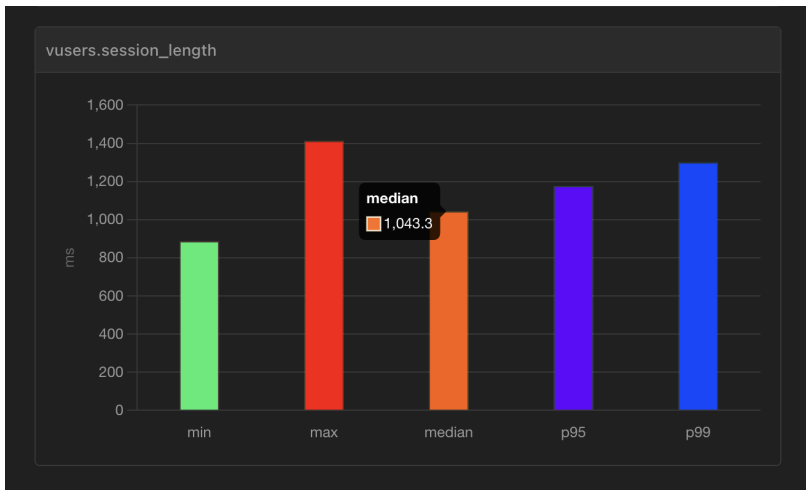
Node



Résultats (5)

Longeur de session

Elixir



Node