



24. リアルタイムお絵かきアプリ

1. はじめに

概要

`Node.js` と `Socket.IO` を用いて、複数ユーザーが同じキャンバスにリアルタイムで描画できるアプリ。描画は線として滑らかに表現され、色・太さの変更やキャンバスのクリアも全員に同期される。

システム構成

技術	役割	説明
Node.js	サーバー実行環境	JavaScript をブラウザ外で動かし、バックエンド処理を実現する
Express	Web フレームワーク	HTTP リクエスト処理やルーティング、静的ファイル配信を担当する
Socket.IO	リアルタイム通信	WebSocket をベースにした双方向通信を提供し、全クライアントと同期する
Tailwind CSS	UI スタイリング	ユーティリティクラスで効率的にデザインを適用し、レイアウトや見た目を整える
nodemon	開発支援ツール	ファイル変更を監視し、自動で Node.js サーバーを再起動して

技術	役割	説明
		くれる

ディレクトリ構成

```
□ socket-draw
  |- server.js
  |- package.json
  \- public
    |- index.html
    \- js
      \- app.js
```

機能仕様

機能	説明
線描画	マウスの動きに追従して滑らかな線を描画
色変更	<input type="color"> で描画色を変更可能
太さ変更	<input type="range"> で線の太さを変更可能
クリア	ボタン押下でキャンバスを全員分リセット
リアルタイム同期	Socket.IO を利用し全クライアント間で描画/消去を同期

2. プロジェクト作成

Node.js 初期化

プロジェクト `my-socket-app` を作成し、`Node.js` を初期化します。

ターミナル

```
npm init -y
```

パッケージインストール

Express & Socket.IO

`Express` と `Socket.IO` をインストールします。

ターミナル

```
npm install express socket.io
```

Nodemonitor

`nodemon` をインストールします。

ターミナル

```
npm install --save-dev nodemon
```

3. Expressサーバー

サーバープログラム

`server.js` を作成し、`Express` サーバを作成します。

server.js

```
const express = require("express");
const http = require("http");
const { Server } = require("socket.io");
const path = require("path");

const app = express();
const server = http.createServer(app);

app.use(express.static(path.join(__dirname, "public")));

server.listen(3000, () => {
  console.log("http://localhost:3000");
});
```

ドローキャンバス

public/index.html にドローキャンバスを作成します。

public/index.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <title>リアルタイムお絵かき</title>
  <script src="https://cdn.tailwindcss.com"></script>
  <script src="/socket.io/socket.io.js"></script>
</head>
<body class="bg-gray-100 flex flex-col items-center p-6">
  <h2 class="text-2xl font-bold mb-4">_REALTIME DRAWING CANVAS</h2>

  <div class="bg-white shadow-lg rounded-lg p-4">
    <canvas id="canvas" width="600" height="400" class="border border-gray-400 rounded"></canvas>
    <div class="mt-3 flex gap-3">
      <button id="clearBtn" class="px-4 py-2 bg-red-500 text-white rounded hover:bg-red-600">
        クリア
      </button>
      <input type="color" id="colorPicker" value="#000000" class="border p-1 rounded">
      <input type="range" id="sizePicker" min="1" max="20" value="3" class="w-32">
    </div>
  </div>

</body>
</html>
```

Expressサーバー起動

`package.json` の `scripts` にサーバ起動コマンド `"dev": "nodemon server.js"` を追加します。

`package.json`

```
{  
  ...  
  "scripts": {  
    "dev": "nodemon server.js",  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  ...  
}
```

サーバー起動

`npm` コマンドでサーバーを起動します。

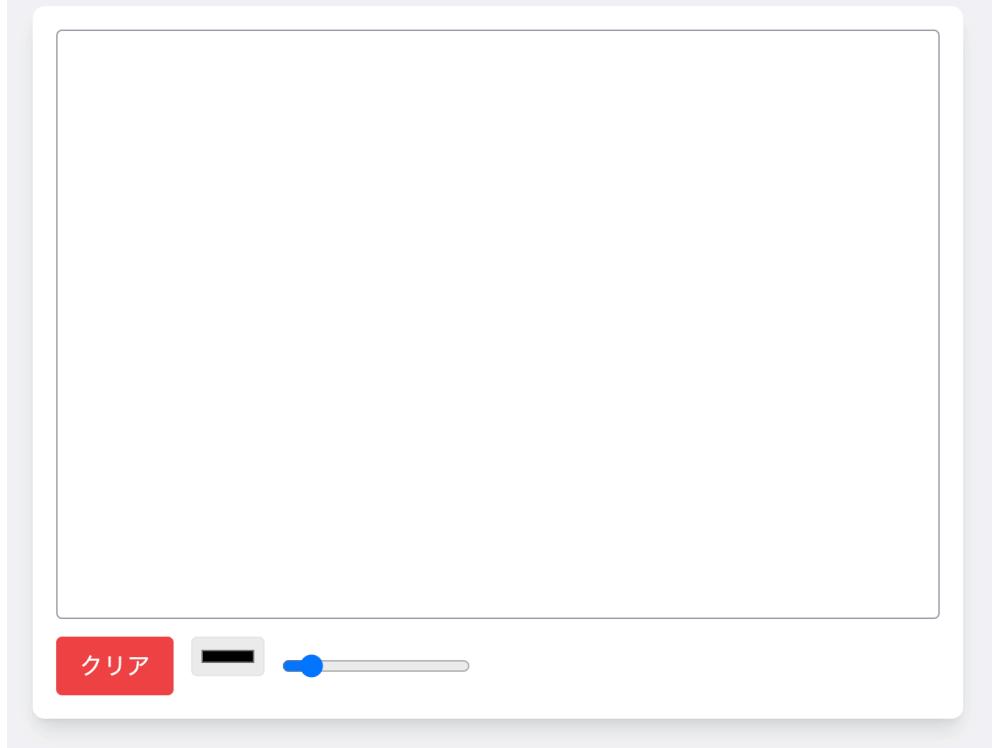
ターミナル

```
npm run dev
```

起動確認

ブラウザで `http://localhost:3000` にアクセスし、Express が動作するか確認します。

リアルタイムお絵かきキャンバス



4. Canvasドロー

JS

public/js/app.js に Canvas ドロー処理を作成します。

public/js/app.js

```
const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");

let drawing = false;
let lastX = 0, lastY = 0;
let color = document.getElementById("colorPicker").value;
let size = document.getElementById("sizePicker").value;

// 共通: 線を描く処理
function drawLine(x1, y1, x2, y2, color, size) {
    ctx.strokeStyle = color;
    ctx.lineWidth = size;
    ctx.lineCap = "round";
    ctx.beginPath();
    ctx.moveTo(x1, y1);
```

```

    ctx.lineTo(x2, y2);
    ctx.stroke();
}

// 共通: クリア処理
function clearCanvas() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
}

// -----
// イベントリスナー
// -----
// 描画開始
canvas.addEventListener("mousedown", (e) => {
    drawing = true;
    [lastX, lastY] = [e.offsetX, e.offsetY];
});

// 描画終了
canvas.addEventListener("mouseup", () => drawing = false);
canvas.addEventListener("mouseout", () => drawing = false);

// マウス移動時
canvas.addEventListener("mousemove", (e) => {
    if (!drawing) return;
    const x = e.offsetX, y = e.offsetY;

    drawLine(lastX, lastY, x, y, color, size);

    [lastX, lastY] = [x, y];
});

// 色変更
document.getElementById("colorPicker").addEventListener("input", (e) => {
    color = e.target.value;
});

// 太さ変更
document.getElementById("sizePicker").addEventListener("input", (e) => {
    size = e.target.value;
});

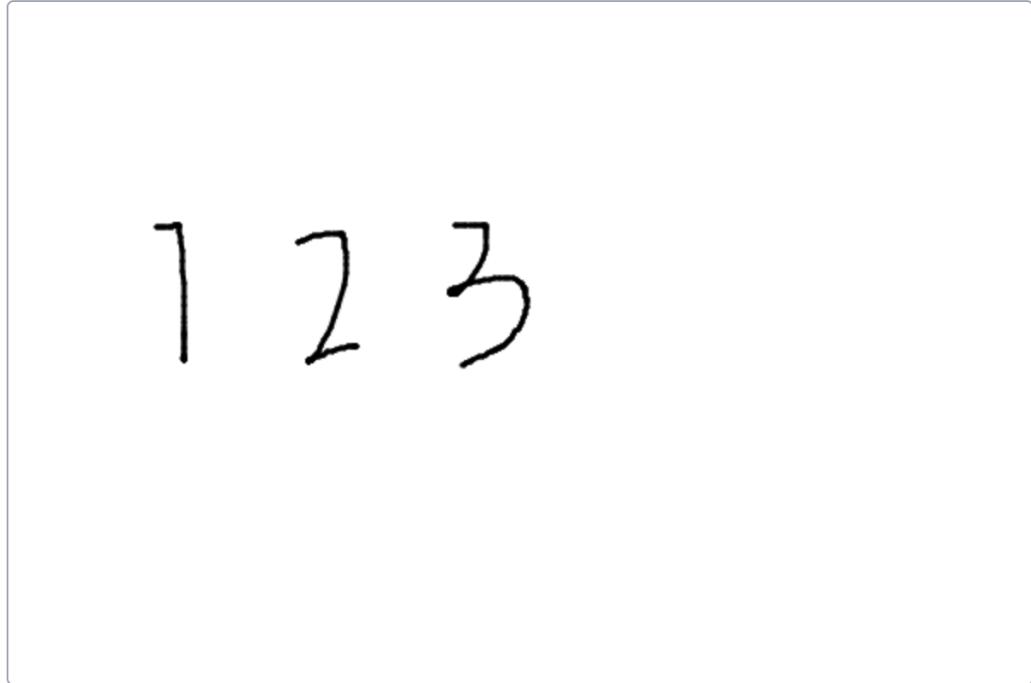
// クリアボタン
document.getElementById("clearBtn").addEventListener("click", () => {
    clearCanvas();
});

```

動作確認

ドロ一処理できるか確認します。

リアルタイムお絵かきキャンバス



1 2 3

クリア



5. Socket.IO 通信

Socket.IO サーバー

server.js に Socket.IO の送受信処理を追加します。

server.js

```
const io = new Server(server);

io.on("connection", (socket) => {
  console.log("ユーザー接続:", socket.id);

  socket.on("draw", (data) => {
    socket.broadcast.emit("draw", data);
  });

  socket.on("clear", () => {
    io.emit("clear"); // 全員に反映
  });
});
```

```
socket.on("disconnect", () => {
  console.log("ユーザー切断:", socket.id);
});
});
```

Socket.IO 受信イベント

ソケット作成

app.js に Socket.IO を作成します。

public/js/app.js

```
const socket = io();
```

ドローイベント送信

マウス移動時に、ドロー情報をサーバ送信します。

public/js/app.js

```
canvas.addEventListener("mousemove", (e) => {
  if (!drawing) return;
  const x = e.offsetX, y = e.offsetY;

  drawLine(lastX, lastY, x, y, color, size);

  // Socket.IOサーバにイベント送信
  socket.emit("draw", { x, y, lastX, lastY, color, size });

  [lastX, lastY] = [x, y];
});
```

public/js/app.js

クリアボタンをクリック時に、Socket.IO サーバに送信します。

```
document.getElementById("clearBtn").addEventListener("click", () => {
  clearCanvas();
  // Socket.IOサーバにイベント送信
  socket.emit("clear");
});
```

イベント受信

Socket.IO からの受信イベントを追加します。

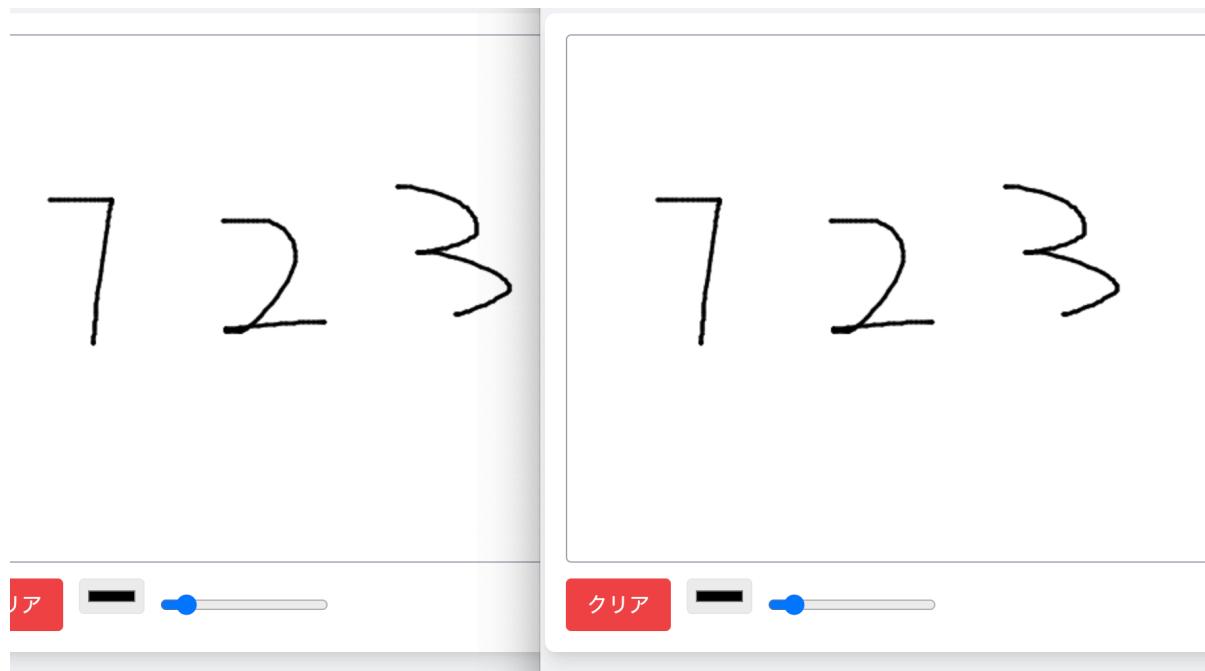
public/js/app.js

```
// ドローイベント
socket.on("draw", (data) => {
  drawLine(data.lastX, data.lastY, data.x, data.y, data.color, data.size);
});

// クリアイベント
socket.on("clear", () => {
  clearCanvas();
});
```

動作確認

複数のブラウザでドロー処理が同期するか確認します。



<< 簡易チャットアプリ

MongoDBの利用 >>

当サイトの教材をはじめとするコンテンツ（テキスト、画像等）の無断転載・無断使用を固く禁じます。これらのコンテンツについて権利者の許可なく複製、転用等する事は法律で禁止されています。尚、当ウェブサイトの内容をWeb、雑誌、書籍等へ転載、掲載する場合は「ロジコヤ」までご連絡ください。

© 2021-2025 logicoya.com