

# Roleta WiFi

Para sorteios - usando integração IoT-Java

# Roleta WiFi

- Pulso gerados por acionamento de chave reed próximo a roda de roleta/bicicleta, onde estão colados imãs
- Cada pulso é transmitido via WiFi (alternativas podem ser bluetooth, RF, etc. ou ligação direta via fio)
- Pulso é recebido via MQTT/mensageria (usando Java/Camel) e seleciona próximo item de uma lista
- Interface gráfica (JavaFX ou Web CSS) mostra números rolando na tela, até os pulsos terminarem no número sorteado.

# Por que?

- Sorteio é mais justo
  - Algoritmos e funções randômicas geram números pseudo-aleatórios (é possível prever o resultado)
  - Pulso gerados externamente eliminam esse problema
- É visualmente mais interessante ver a roleta girando e interagindo com o computador
- A visão da roda girando causa antecipação do resultado, torcida para que seu número seja sorteado; é mais emocionante

# Teste

- Usando um reed + 8 imãs presos na roda (na verdade são 7, pois um deles caiu :)



# MQTT Subscriber

- Java + CDI + Apache Camel

## RouletteRouteBean.java - TESTE

```
package iot.web.roulette;

import org.apache.camel.Exchange;
import org.apache.camel.Processor;
import org.apache.camel.builder.RouteBuilder;

public class RouletteRouteBean extends RouteBuilder {

    String[] people = {
        "Jack", "Jill", "John", "Jair", "Jota", "Jane", "Judy", "Judd",
        "Jamy", "Jimy", "Jana", "Jian", "Jaar"
    };

    int count = 0;
    int pos = 0;

    @Override
    public void configure() throws Exception {
        from("mqtt:test?subscribeTopicName=/esp8266/pulse&host=tcp://127.0.0.1:1883")
            .process(new Processor() {
                @Override
                public void process(Exchange ex) {
                    System.out.println(++count + ": " + people[pos] + "("+pos+ ")");
                    pos++;
                    if(pos >= people.length) {
                        pos = 0;
                    }
                }
            });
    }
}
```

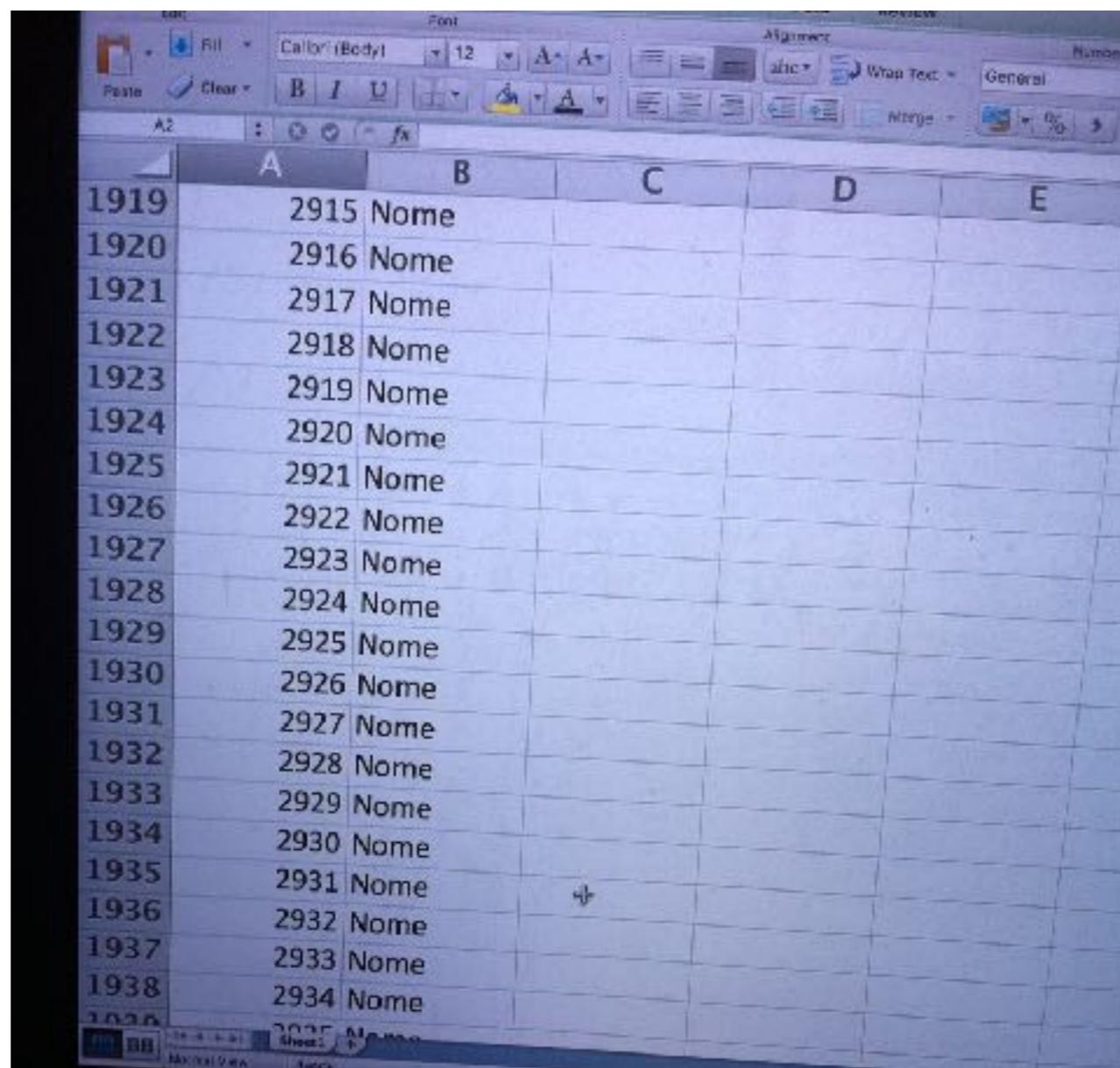
## Dados de teste

## pom.xml

```
<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>7.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-core</artifactId>
        <version>2.19.0</version>
        <type>jar</type>
    </dependency>
    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-mqtt</artifactId>
        <version>2.19.0</version>
        <type>jar</type>
    </dependency>
    <dependency>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-cdi</artifactId>
        <version>2.19.0</version>
        <type>jar</type>
    </dependency>
</dependencies>
```

# Fonte de dados

- Dados reais estão em arquivo XSLX (Excel)



A screenshot of an Excel spreadsheet titled "2025 Nome". The spreadsheet contains two columns of data: Year (A) and Name (B). The years from 1919 to 1938 are listed in column A, and the corresponding names "Nome" are listed in column B. The data starts at row 2 and continues down to row 38. The Excel ribbon and various toolbars are visible at the top of the screen.

A	B
1919	2915 Nome
1920	2916 Nome
1921	2917 Nome
1922	2918 Nome
1923	2919 Nome
1924	2920 Nome
1925	2921 Nome
1926	2922 Nome
1927	2923 Nome
1928	2924 Nome
1929	2925 Nome
1930	2926 Nome
1931	2927 Nome
1932	2928 Nome
1933	2929 Nome
1934	2930 Nome
1935	2931 Nome
1936	2932 Nome
1937	2933 Nome
1938	2934 Nome
1939	

# Leitura do Excel (usando Apache POI)

```
public class RouletteRouteBean extends RouteBuilder {  
  
    public static final int DATA_IDX = 0; // qual coluna excel estão os dados a serem usados no sorteio  
    public static final int COLS = 5; // numero de colunas a verificar  
    public static final String DADOS = "dados.xlsx"; // arquivo de dados  
  
    int count = 0;  
    int pos = 0;  
  
    List<String> tickets = new ArrayList<>();  
  
    public RouletteRouteBean() {  
        init();  
    }  
  
    //@PostConstruct  
    public void init() {  
        InputStream is = this.getClass().getResourceAsStream(DADOS);  
        //System.out.println("IS: " + is);  
        Workbook workbook = null;  
        try {  
            workbook = new XSSFWorkbook(is);  
            //System.out.println("WB: " + workbook);  
            Sheet sheet = workbook.getSheetAt(0);  
            DataFormatter formatter = new DataFormatter();  
  
            for (int rowIndex = 0; rowIndex <= sheet.getLastRowNum(); rowIndex++) {  
                Row row = sheet.getRow(rowIndex);  
                if (row != null) {  
                    String cellValue = null;  
                    for (int colIndex = 0; colIndex < COLS; colIndex++) {  
                        if (colIndex == DATA_IDX) {  
                            Cell cell = row.getCell(colIndex);  
                            if (cell != null) {  
                                cellValue = formatter.formatCellValue(cell);  
                                tickets.add(cellValue);  
                                break;  
                            }  
                        }  
                    }  
                }  
            }  
        } catch (IOException e) {  
        }  
    }  
}
```

Arquivo de dados  
(resource no mesmo  
pacote)

Procura coluna  
que contém  
dados

Escolhe a  
coluna onde  
estão os dados

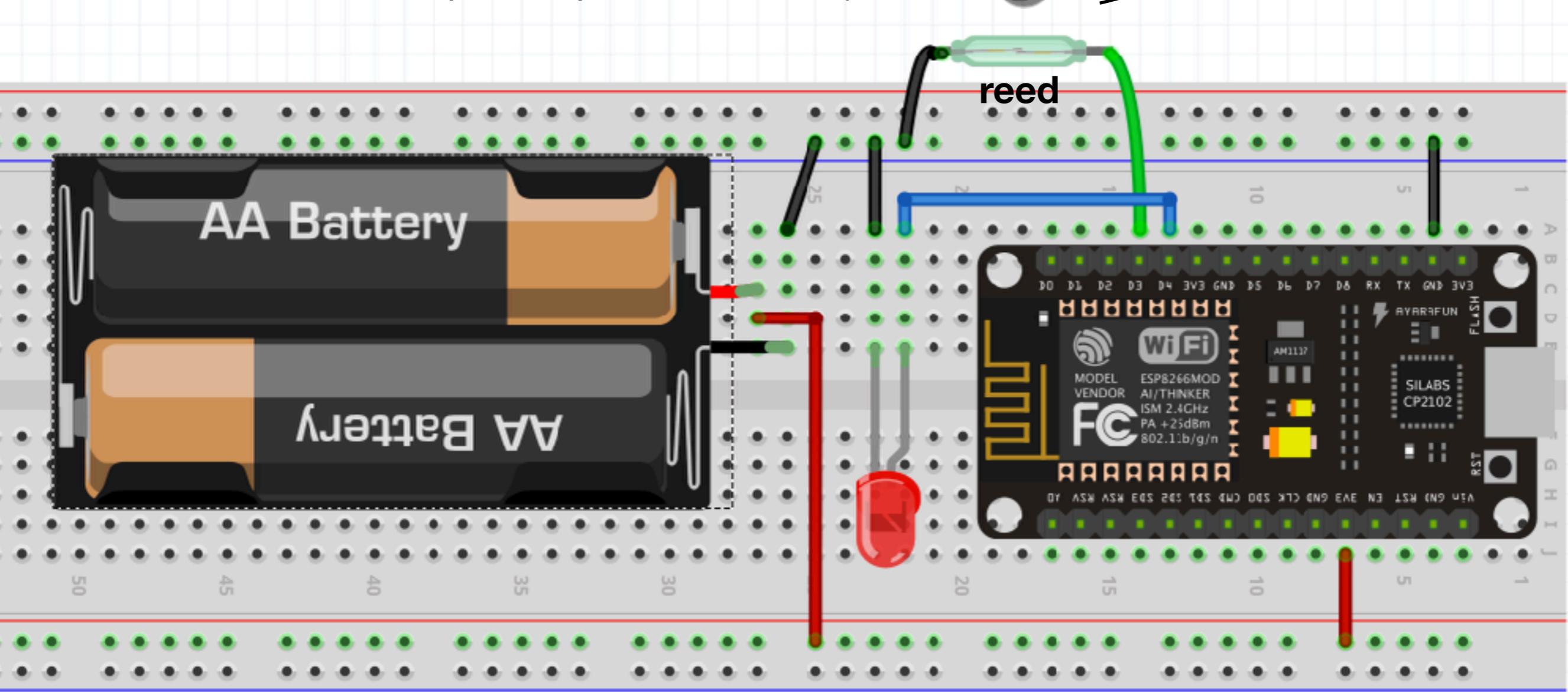
Extrai dado de uma  
célula e adiciona no  
ArrayList

# MQTT Server

- Mosquitto
- /usr/local/sbin/mosquitto -v (Mac OS)
- Publisher (ESP8266) e Subscriber (Java+Camel)  
conectam-se automaticamente

# Circuito

- Módulo **ESP8266** ESP-12F NodeMCU Lolin V3
- **LED**
- Interruptor magnético **reed** (fecha o circuito - conecta D3 a GND - quando próximo de imã)



# Roleta

Chave  
magnética  
(reed)

Imãs

ESP8266

```

#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Argo Navis";
const char* password = "xxxxxx";
const char* mqtt_server = "192.168.1.102";

WiFiClient espClient;
PubSubClient client(espClient);
char data[1] = {'1'};

const int REED = 0; // 3
const int LED = 2; // 4
int last = HIGH; // start high

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("WiFi connected - ESP IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP8266Client")) {
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

```

# MQTT Publisher

```

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(REED, INPUT_PULLUP);
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

void loop() {
  // Conecta ao servidor MQTT
  if (!client.connected()) {
    reconnect();
  }
  if(!client.loop()) {
    client.connect("ESP8266Client");
  }

  // Obtem estado do Reed
  int state = digitalRead(REED);

  // detecta e envia pulso
  if(state == LOW) {
    if(last == HIGH) {
      client.publish("/esp8266/pulse", data);
      Serial.println("pulse");
      digitalWrite(LED, HIGH);
    } else {
      Serial.println("-");
    }
    last = LOW;
  } else { // state = HIGH
    digitalWrite(LED, LOW);
    last = HIGH;
    Serial.println(">>>");
  }
}

```

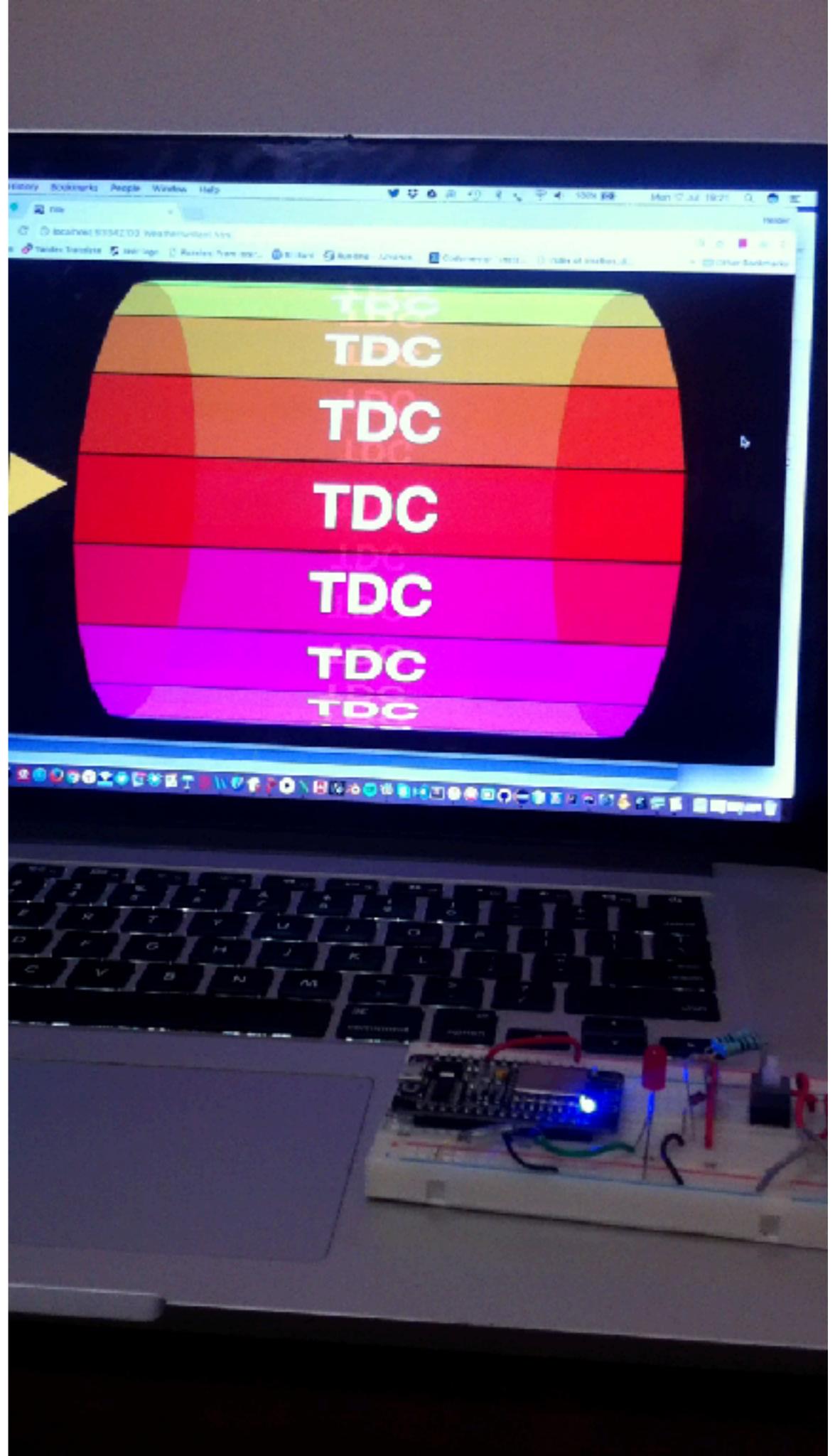
# Camel (rota de mensageria)

- **from:** Cliente MQTT (recebe pulso)
- Processador (grava próximo ticket da roleta no payload)
- **to:** Servidor WebSocket (manda ticket)

```
@Override  
public void configure() throws Exception {  
    //from("timer://foo?fixedRate=true&period=300")  
    from("mqtt:test?subscribeTopicName=/esp8266/pulse&host=tcp://127.0.0.1:1883")  
        .process(new Processor() {  
            @Override  
            public void process(Exchange ex) {  
                ++count;  
                String body = tickets.get(pos); ← Recebe pulso enviado para Mosquitto (porta 1883) pelo ESP8266  
                System.out.println(body);  
                ex.getOut().setBody(body); ← Lê item da posição 0 até tickets.size() e coloca no body da mensagem  
                pos++; ← Reinicia, quando lista acabar  
                if (pos >= tickets.size()) {  
                    pos = 0; ← Manda mensagem para WebSocket  
                }  
            }  
        }).to("websocket://localhost:9292/ws?sendToAll=true");  
}
```

# Interface gráfica

- HTML5 + CSS + WebSocket
- Página é cliente WebSocket e recebe dados de aplicativo Camel/Java
- Aplicativo Camel/Java passa dados lidos da planilha Excel



# Animação 3D (CSS 3)

- Baseado em <https://desandro.github.io/3dtransforms/docs/carousel.html>

Secure <https://desandro.github.io/3dtransforms/docs/carousel.html>

Yandex.Translate [QR logo](#) [Russian: From Inter...](#) [Brilliant](#) [Rus4mc - Advance...](#) [CodeMentor | Insta...](#) [Index of /malhas\\_di...](#)

## Intro to CSS 3D transforms

Introduction  
Perspective  
3D transform functions  
Card Flip  
Cube  
Rectangular prism  
Carousel  
Conclusion

Just like our previous 3D objects, to show any one panel, we need only to apply the reverse transform on the carousel. Here's the style to show the fifth panel:

```
transform: translateZ(-288px) rotateY(-160deg);
```

[See Example: Carousel 1](#)

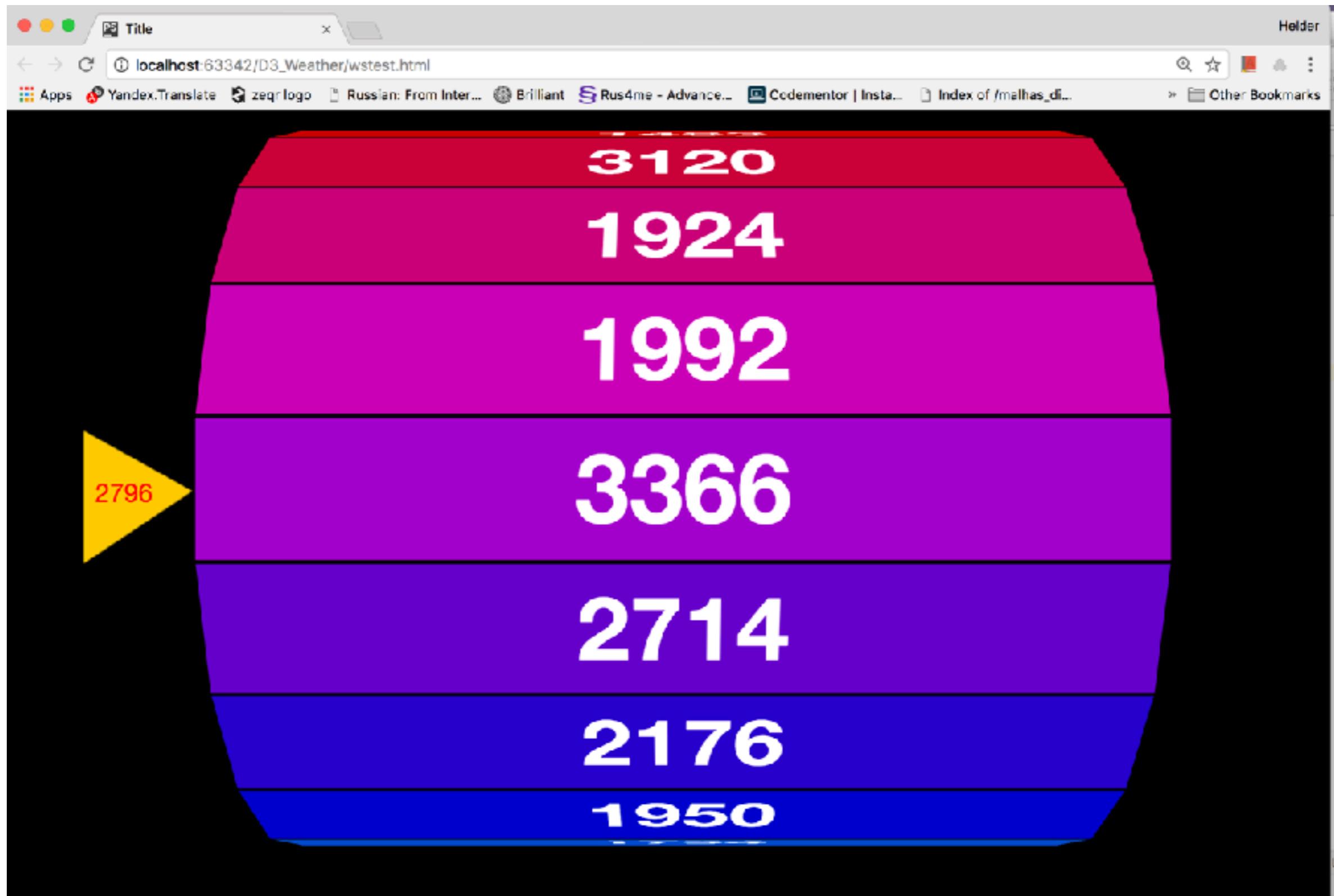


By now, you probably have two thoughts:

1. Re-writing transform styles for each panel looks to be tedious.
2. Why bother doing high school math – Aren't robots supposed to be doing all this work for us?

And you're absolutely right. The repetitive nature of 3D objects lend themselves to scripting. We can offload all the monotonous transform styles to our dynamic script, which, if done right, will be more flexible than the hard-coded version.

# Com dados embaralhados + contagem de pulsos



# Cliente WebSocket

- Recebe ticket e grava em componente HTML

```
var ws = new WebSocket('ws://localhost:9292/ws');
```

```
var count = 0;  
var pulses = 0;
```

Chamado quando  
mensagem é recebida

```
ws.onmessage = function (evt) {  
    pulses++;  
    var pos = count + 10;  
    if(pos > 20) count = -10;  
    console.log("Message received = " + evt.data);  
    $("#carousel figure:nth-child("+pos+")").text(evt.data);  
    $('#next').click();  
    ++count;  
    $(".arrow p").text(pulses)  
};
```

Atualiza figura oculta (na  
parte de trás do cilindro  
giratório de 20 figuras)

Avança para próximo  
(gira o cilindro)

Mostra quantos pulsos  
foram recebidos

# Código

- <https://github.com/helderdarocha/IoTProjects>

The screenshot shows the GitHub repository page for `helderdarocha/IoTProjects`. The repository has 4 commits, 1 branch, 0 releases, and 1 contributor. The latest commit was made 36 minutes ago. The repository contains files for `esp_roulette`, `iot-web-roulette`, and `README.md`. The `README.md` file was updated 36 minutes ago. The repository description is "Random iot project code".

Random iot project code

4 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

helderdarocha committed on GitHub Update README.md Latest commit e5129a1 36 minutes ago

esp\_roulette Updates 40 minutes ago

iot-web-roulette Updates 40 minutes ago

README.md Update README.md 36 minutes ago

README.md

## IoTProjects

Random iot project code

- `esp_roulette` - Code for ESP8266 MQTT client (sends pulses when D3 is connected to GND)
- `iot-web-roulette` - Code for MQTT client (Camel/Java) and WebSocket server (Camel/Java) and

# Sugestões de alteração

- Ligar MQTT via Arduino diretamente em USB com fio (evita usar WiFi) **+ performance**
- Interface JavaFX (evita usar WebSocket) **+performance**
- Incluir outros dados na figura (em vez e apenas o número): nome do participante, foto, animações, logotipos, etc. **-performance**
- Em vez de cilindro giratório, fazer animação com cubo do TDC girando em todas as direções, como um dado, e distribuindo os dados em quadrados **-performance**
- Pegar os dados em tempo real (ex: Twitter) **-performance**