Trabajo de Fin De Máster

RecoFood

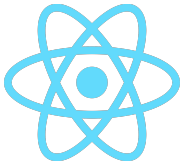
Silvia Donaire Serrano
Elena Racero González
Manuel Fajardo Jiménez

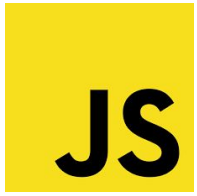
Máster de Inteligencia Artificial y Big Data
C.P.I.F.P - Alan Turing

¿Qué es RecoFood?

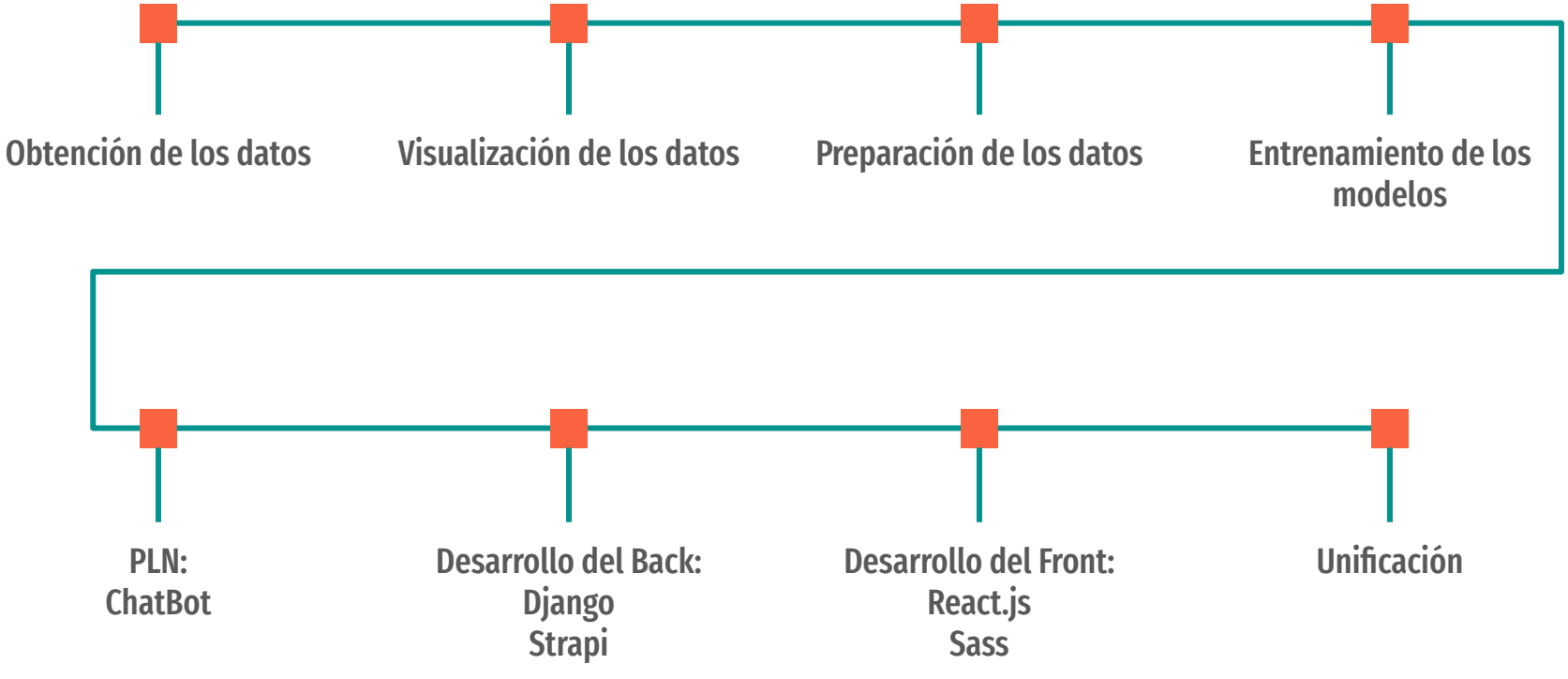
Es una genialidad creativa, una app desarrollada por nosotros para reconocer platos culinarios mediante imagen, así como obtención de recetas y aporte calórico para el usuario.

TECNOLOGÍAS





Desarrollo del proyecto



Obtención de datos

Dataset food101
TensorFlow web

Dataset de 101.000 imágenes de 101 comidas diferentes.

Dataset IMC
Kaggle

Dataset con datos de 741 individuos que contiene información de la edad, el peso, la altura, el IMC y la categoría.

JSON Macros
Realizado por nosotros

Json que contiene todos los macronutrientes de cada comida del dataset.

API Gemini
Web Google Cloud

API para la obtención de respuestas para nuestro ChatBot.

JSON

Carbohidratos

Por cada 100 gr

Calorias

Por cada 100 gr

Proteinas

Por cada 100 gr

Grasas

Por cada 100 gr

MACRONUTRIENTES

Scrapping de Imagenes

https://www.recetasgratis.net/receta-de-pizza-de-pepperoni-74152.html

'https://www.recetasgratis.net/receta-de-tacos-rancheros-76180.html

'https://www.recetasgratis.net/receta-de-churros-mexicanos-75726.html'

BeautifulSoup

Urllib

Exclusión de Imágenes determinadas

Modelo de clasificación de imágenes

Visualización de los datos

Imagen aleatoria de cada comida del dataset.

Preparación de los datos

Generalización de los datos.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    rotation_range=40,
    brightness_range=[0.5, 1.5],
    width_shift_range=0.2,
    height_shift_range=0.2,
    validation_split=0.25)

test_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.25)
```

Preparación de los datos

Creación de los conjuntos de entrenamiento y validación con la generalización.

```
train_generator = train_datagen.flow_from_directory(
    base_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset = "training")

validation_generator = test_datagen.flow_from_directory(
    base_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset = "validation")
```

Creación del modelo

Arquitectura preentrenada InceptionV3.

```
inception = InceptionV3(
    weights='imagenet',
    include_top=False,
    input_shape=(img_height, img_width, 3))

x_inception = inception.output
x_inception = GlobalAveragePooling2D()(x_inception)

predictions_inception = Dense(
    len(class_names),
    kernel_regularizer=l2(0.005),
    activation='softmax')(x_inception)

model_inception = Model(inputs=inception.input, outputs=predictions_inception)
```

Compilación del modelo

Optimizador SGD y función de pérdida “categorical_crossentropy”.

```
model_inception.compile(
    optimizer=SGD(learning_rate=0.0001, momentum=0.9),
    loss='categorical_crossentropy',
    metrics=['accuracy'])
```

Creación de Callbacks

EarlyStopping para detener el modelo si no hay mejoras en la métrica durante el entrenamiento y ModelCheckpoint para guardar el modelo en cada iteración.

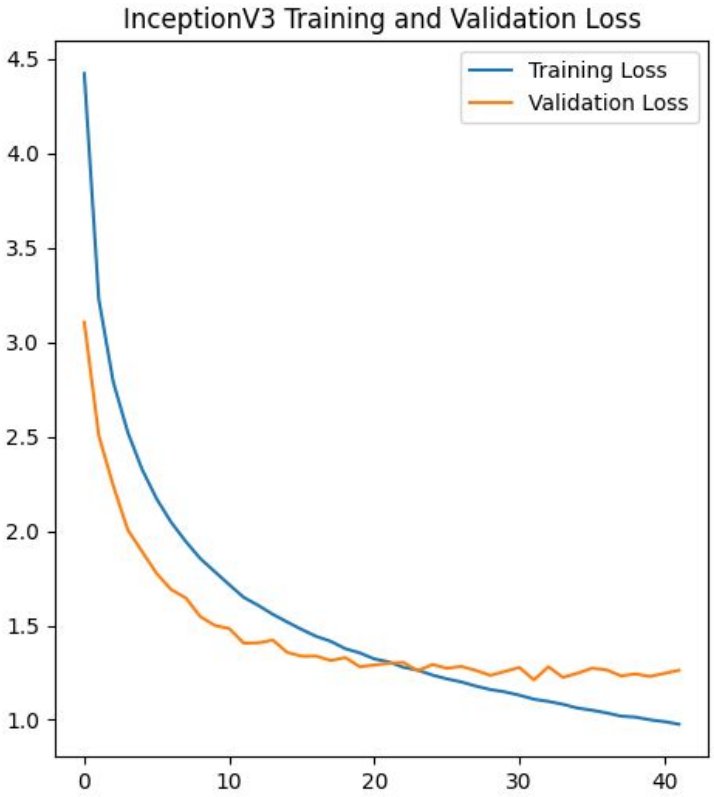
```
early_stopper_inception = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True)
checkpointer_inception = ModelCheckpoint(
    filepath='Modelos entrenados/best_model_trained.h5',
    verbose=1,
    save_best_only=True)
```

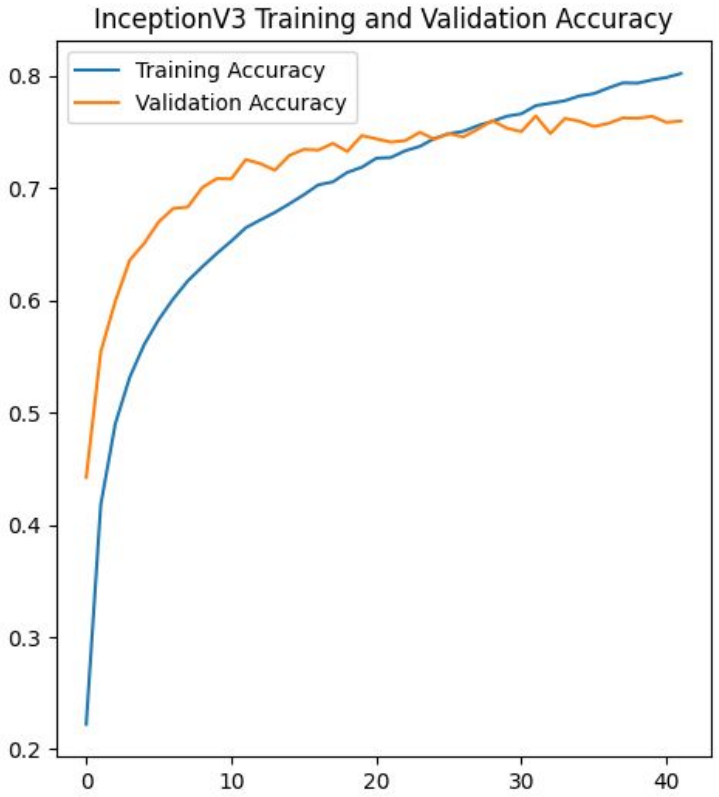
Entrenamiento del modelo

Utilización del conjunto de datos de entrenamiento y validación para entrenar el modelo.

```
history = model_inception.fit(train_generator,
                    steps_per_epoch = len(train_generator),
                    validation_data = validation_generator,
                    validation_steps = len(validation_generator),
                    epochs = epoch,
                    verbose = 1,
                    callbacks = [early_stopper_inception, checkpointer_inception])
```

Métricas del rendimiento





Funcionamiento del modelo de clasificación de imágenes

Proceso de análisis

Imagen

¡Hamburguesa!

Transformación

Datos válidos para el modelo

Filtrado

Selección del alimento correspondiente

Carga del modelo

Comienza el análisis

Respuesta

101 resultados entre 0 y 1

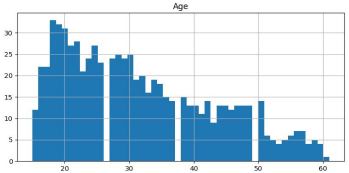
Modelo predictivo del IMC

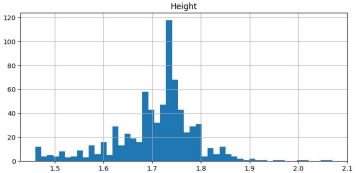
Preparación de los datos

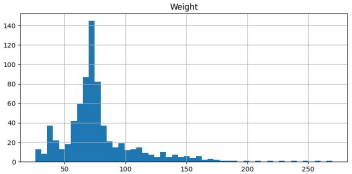
```
1  import numpy as np
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import seaborn as sns
```

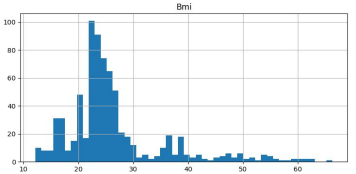
```
1  file_path = '/content/drive/MyDrive/Dataset/bmi.csv'
2  bmidata = pd.read_csv(file_path)
```

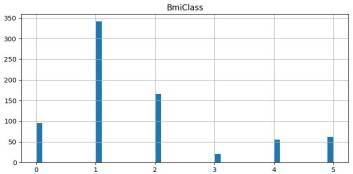
Gráficos y Correlaciones

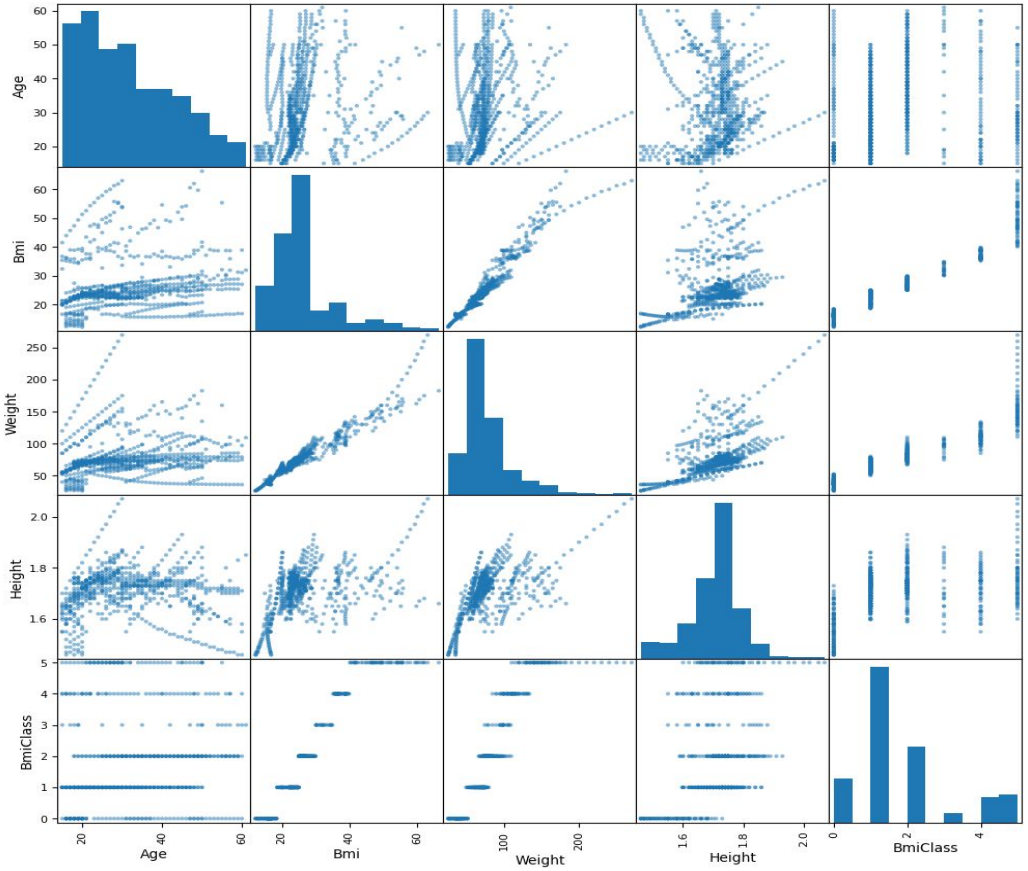












Entrenamiento y Random Forest

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

```python
from sklearn.ensemble import RandomForestRegressor

bmidata_model_v3 = RandomForestRegressor()

# Entrenamiento del modelo
bmidata_model_v3.fit(X_train, y_train)

# Predicción
y_pred = bmidata_model_v3.predict(X_test)

y_test_list = y_test.tolist()

print("Prec. real    Prec. estimado    Error absoluto    Porcentaje de acierto")
```

Predicción Obesidad (BMI)

Underweight	Random Forest
Normal Weight	Random Forest
Overweight	Random Forest
Obese Class(1, 2, 3)	Random Forest

Funcionamiento del modelo predictivo del IMC

Proceso de análisis

Datos

Transformación

Datos válidos para el modelo

Carga del modelo

Comienza el análisis

Respuesta

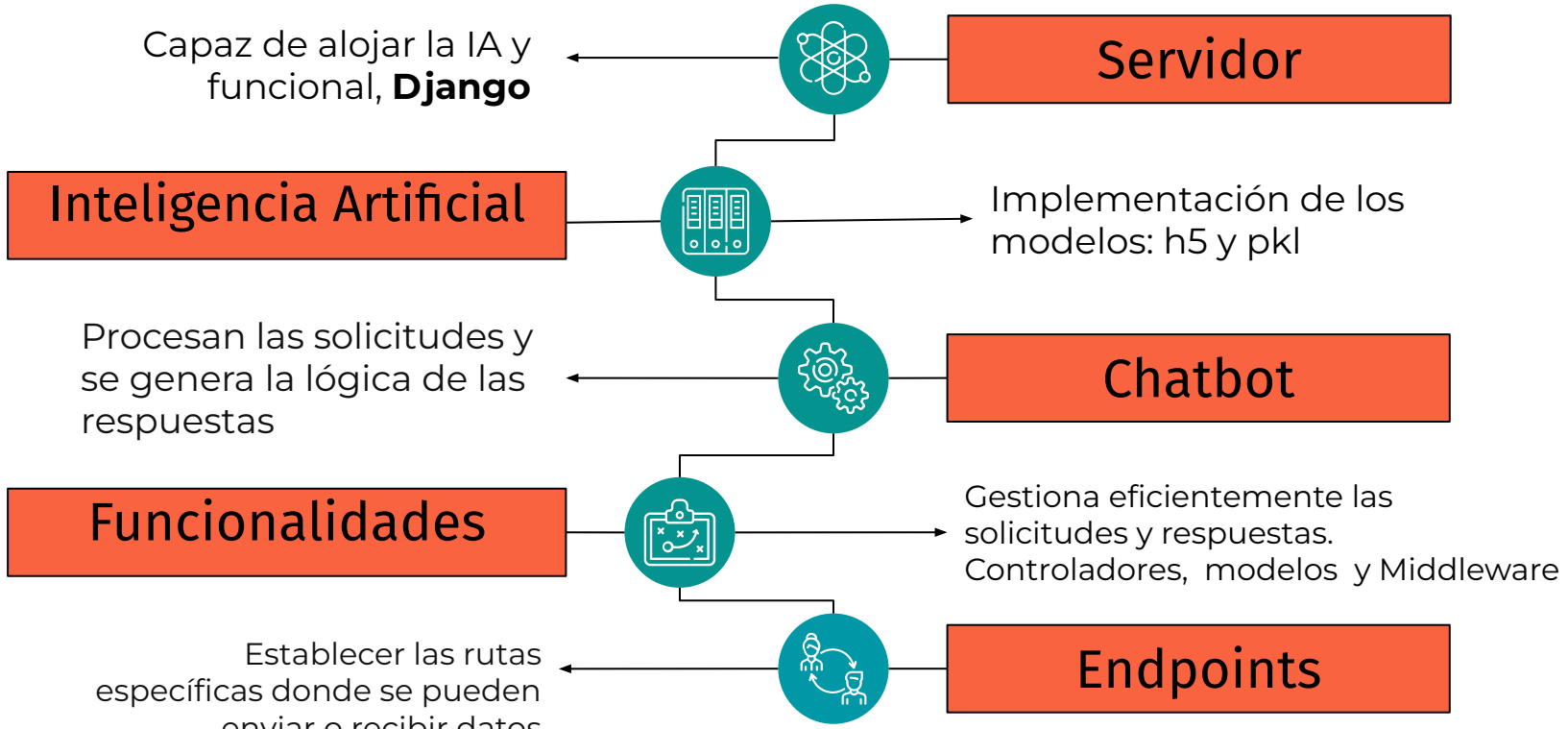
5 resultados entre 0 y 1

Filtrado

Selección de la respuesta correspondiente

Resultado IMC

Backend


Capaz de alojar la IA y funcional, **Django**
Servidor
Inteligencia Artificial
Implementación de los modelos: h5 y pkl
Procesan las solicitudes y se genera la lógica de las respuestas
Chatbot
Funcionalidades
Gestiona eficientemente las solicitudes y respuestas. Controladores, modelos y Middleware
Establecer las rutas específicas donde se pueden enviar o recibir datos
Endpoints

Chatbot

React

1. Preparación del mensaje del usuario y envío de la solicitud al servidor

```
const trimmedInput = inputValue.trim();
const response = await fetch("http://localhost:8000/chatbot/", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({ prompt: trimmedInput }),
});

if (!response.ok) throw new Error("Error en la solicitud");
```

6. Obtención de la respuesta del servidor

```
const data = await response.json();
console.log("Respuesta del servidor:", data);
```

7. Obtención y preparación del audio de la respuesta del servidor

```
const audioResponse = await fetch(
  `http://localhost:8000/media/${data.audio}`
);
const audioBlob = await audioResponse.blob();
const audioUrl = URL.createObjectURL(audioBlob);
setAudioRefs((prevRefs) => ({
  ...prevRefs,
  [data.message]: { url: audioUrl, isPlaying: false },
}));
```

Django

2. Obtiene los datos de la solicitud POST

```
elif request.method == "POST":
    data = json.loads(request.body)
    user_prompt = data.get("prompt", "")
```

3. Envía el mensaje del usuario a Gemini-Pro y obtiene la respuesta

```
if user_prompt:
    gemini_response = chat_session.send_message(user_prompt)
```

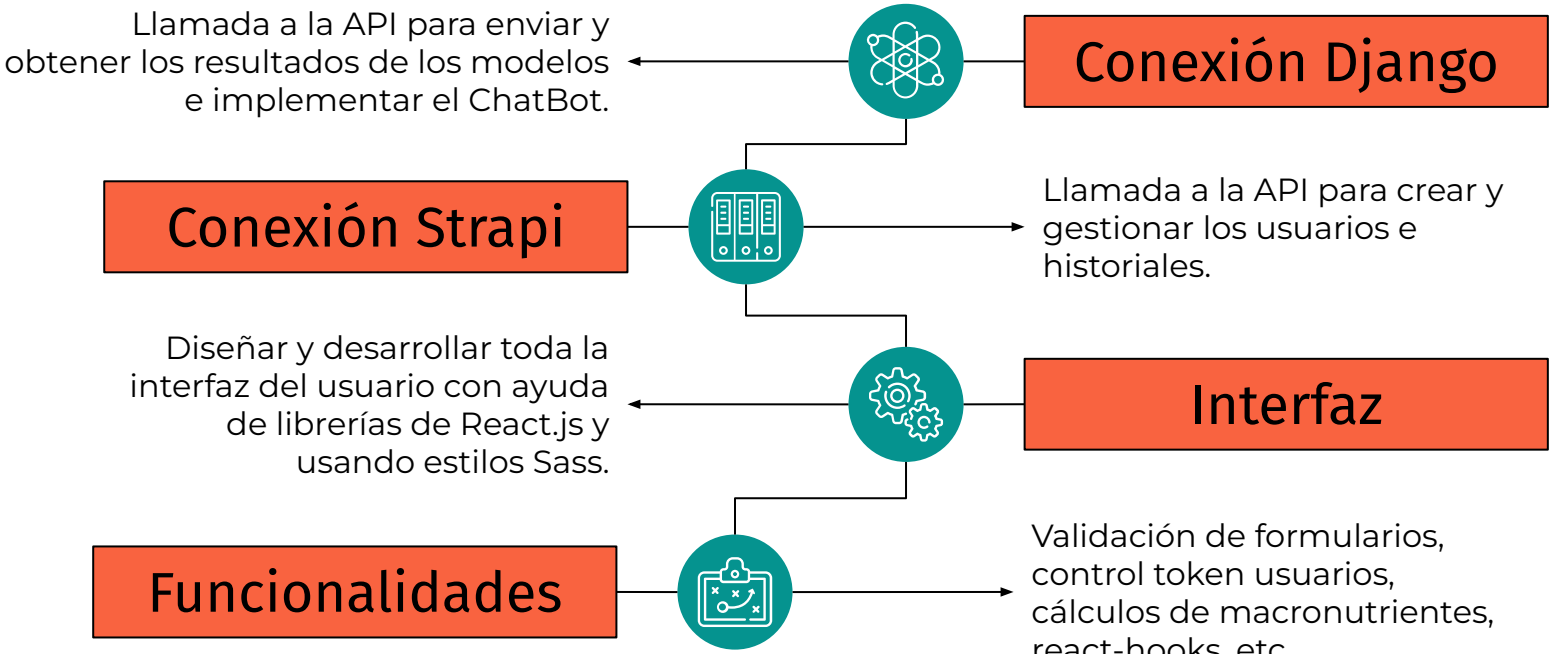
4. Convierte la respuesta del chatbot en audio

```
clean_text = re.sub(r'[^\w\s]', '', gemini_response.text)
tts = gTTS(text=clean_text, lang="es")
```

5. Devuelve la respuesta al cliente

```
    response_data = {"message": gemini_response.text, "audio": filename}
return JsonResponse(response_data)
```

Frontend en React.js



Realizado por

Silvia Donaire Serrano
Elena Racero González
Manuel Fajardo Jiménez

Máster de Inteligencia Artificial y Big Data
C.P.I.F.P - Alan Turing

2023/2024