



Instituto Politécnico Nacional

Escuela Superior de Cómputo



Sistemas Distribuidos

Prof. **Benjamín Cruz Torres**

Proyecto final
Aplicación web en la nube

Grupo: 7CV1

Equipo: Express

Integrantes:

1. Amado Becerra Frederick D.
2. Hernández Reyes Julio C.
3. Martínez Robledo Luis A.
4. Mejía Romo Pablo E.
5. Mejorada Nolasco Alan A.

Fecha: 23/06/2023

PROYECTO FINAL: APLICACIÓN WEB EN LA NUBE

OBJETIVO DE LA PRÁCTICA: implementar una aplicación web en la nube con el fin de integrar los conocimientos adquiridos a lo largo de la unidad de aprendizaje.

ESCENARIO

Se desea implementar un sistema experto que determine, mediante una serie de preguntas a una empresa, el modelo de servicio que le conviene y las características de éste. El sistema tendrá los siguientes requerimientos:

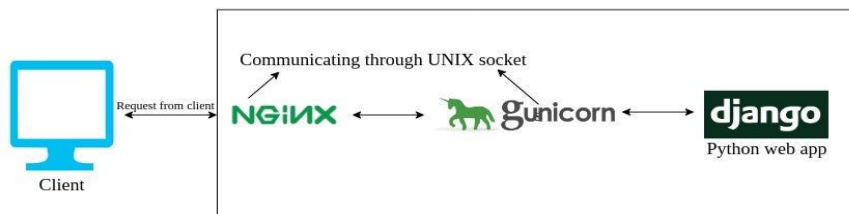
1. El sistema estará montado en la nube.
2. Las preguntas que presentará el sistema serán cerradas o, en su defecto, de respuestas numéricas.
3. El sistema guardará las respuestas en una base de datos en la nube.
4. Implementar una estrategia de copias de seguridad de los discos duros en la nube.
5. El sistema deberá contar con un equilibrador de carga público para balancear la carga de trabajo.
6. El sistema deberá contar con una directiva de firewall para evitar ataques de SQL Injection o PHP injection

RECURSOS NECESARIO PARA REALIZAR EL PROYECTO FINAL

1. Computadora con conexión a internet
2. Tener una suscripción a Microsoft Azure: Azure for Students

INTRODUCCIÓN

La implementación de una aplicación Django[1] en la nube es una forma eficiente y escalable de poner en marcha tu sitio web o aplicación. Azure, una plataforma en la nube de Microsoft, ofrece una serie de servicios y herramientas que facilitan el despliegue y la gestión de aplicaciones web, incluyendo Django.



Para configurar Django en Azure[2], se recomienda utilizar un servidor web como NGINX para manejar las solicitudes entrantes y un servidor de aplicaciones como Gunicorn para ejecutar la aplicación Django. NGINX actúa como un proxy inverso, enviando las solicitudes al servidor de aplicaciones y gestionando el enrutamiento y la

seguridad, mientras que Gunicorn se encarga de ejecutar la aplicación Django y gestionar las solicitudes HTTP.

A continuación, se presentan los pasos generales para implementar Django en Azure utilizando NGINX y Gunicorn[3]:

1. **Crear una instancia virtual en Azure:** Comienza creando una máquina virtual en Azure con el sistema operativo adecuado para tu aplicación Django. Puedes elegir entre varias opciones, como Ubuntu, CentOS o Windows Server.
2. **Instalar y configurar NGINX:** Una vez que la máquina virtual esté lista, instala NGINX y configúralo como un proxy inverso. Configura los archivos de configuración de NGINX para que las solicitudes entrantes se redirijan correctamente al servidor de aplicaciones Gunicorn.
3. **Instalar y configurar Gunicorn:** A continuación, instala Gunicorn en tu máquina virtual y configúralo para ejecutar la aplicación Django. Especifica la ruta del archivo WSGI de Django y configura el número de trabajadores Gunicorn según tus necesidades de rendimiento.
4. **Configurar el firewall y los puertos:** Asegúrate de permitir el tráfico entrante en el firewall de la máquina virtual y abrir los puertos necesarios para que NGINX y Gunicorn funcionen correctamente.

5. **Configurar la base de datos y otros servicios:** Si tu aplicación Django utiliza una base de datos u otros servicios, asegúrate de configurarlos correctamente en Azure. Puedes optar por utilizar servicios administrados de bases de datos como Azure Database for PostgreSQL o Azure SQL Database.

6. **Desplegar la aplicación Django:** Copia los archivos de tu aplicación Django en la máquina virtual y asegúrate de que estén configurados correctamente. Asegúrate de instalar todas las dependencias de tu proyecto utilizando un entorno virtual.

7. **Reiniciar los servicios y probar la aplicación:** Reinicia NGINX y Gunicorn para aplicar los cambios de configuración. Luego, prueba tu aplicación Django accediendo a ella a través de la dirección IP pública de tu máquina virtual en Azure.

ANÁLISIS

REQUERIMIENTOS FUNCIONALES:

Registro de usuarios: Permitir que los usuarios se registren en el sistema para acceder a las funcionalidades y servicios ofrecidos.

Gestión de encuestas: Permitir a los usuarios completar y enviar encuestas para determinar su modelo ideal de servicios en la nube.

Procesamiento de solicitudes backend: Las dos máquinas backend deben ser capaces de recibir solicitudes y procesarlas según la lógica de negocio definida.

Comunicación entre las máquinas: Establecer una comunicación segura y eficiente entre las máquinas backend y la máquina frontend para transferir datos y resultados.

Integración de modelos de Machine Learning: Incorporar el modelo de ML entrenado en el sistema para realizar predicciones y determinar el modelo ideal de servicios en la nube para cada usuario.

REQUERIMIENTOS NO FUNCIONALES:

Seguridad: Garantizar la seguridad de los datos y la comunicación mediante el uso de protocolos de cifrado, autenticación y control de acceso.

Escalabilidad: Diseñar el sistema de manera que sea capaz de manejar un aumento en la carga de usuarios y solicitudes, escalando horizontalmente según sea necesario.

Disponibilidad: Mantener el sistema en funcionamiento y disponible para los usuarios, minimizando el tiempo de inactividad y realizando copias de seguridad periódicas de los datos.

Rendimiento: Optimizar el rendimiento del sistema para proporcionar respuestas rápidas y eficientes a las solicitudes de los usuarios.

Tolerancia a fallos: Implementar mecanismos de detección y recuperación de fallos para minimizar el impacto en la disponibilidad y funcionalidad del sistema.

REGLAS DE NEGOCIO:

Regla de negocio: Validación de respuestas de encuestas

Descripción: Todas las preguntas de la encuesta deben ser respondidas antes de enviarla.

Acción: El sistema debe verificar que no haya preguntas sin responder y mostrar un mensaje de error si falta alguna respuesta.

Regla de negocio: Recomendación de modelos basada en encuestas

Descripción: El sistema debe recomendar el modelo ideal de servicios en la nube basándose en las respuestas proporcionadas en la encuesta.

Acción: El sistema debe procesar las respuestas de la encuesta utilizando un modelo de Machine Learning entrenado y proporcionar una recomendación personalizada al usuario.

Regla de negocio: Procesamiento de solicitudes backend equilibrado

Descripción: El sistema debe distribuir equitativamente las solicitudes backend entre las dos máquinas disponibles.

Acción: El sistema debe utilizar un algoritmo de equilibrio de carga para dirigir las solicitudes de manera adecuada y evitar una sobrecarga desigual en las máquinas.

Regla de negocio: Gestión de errores y notificaciones

Descripción: El sistema debe monitorear y gestionar errores en el procesamiento de solicitudes, enviando notificaciones a los administradores cuando ocurran errores críticos.

Acción: El sistema debe registrar los errores en un registro de errores y notificar al equipo de administradores por correo electrónico o mediante una herramienta de notificación.

DISEÑO

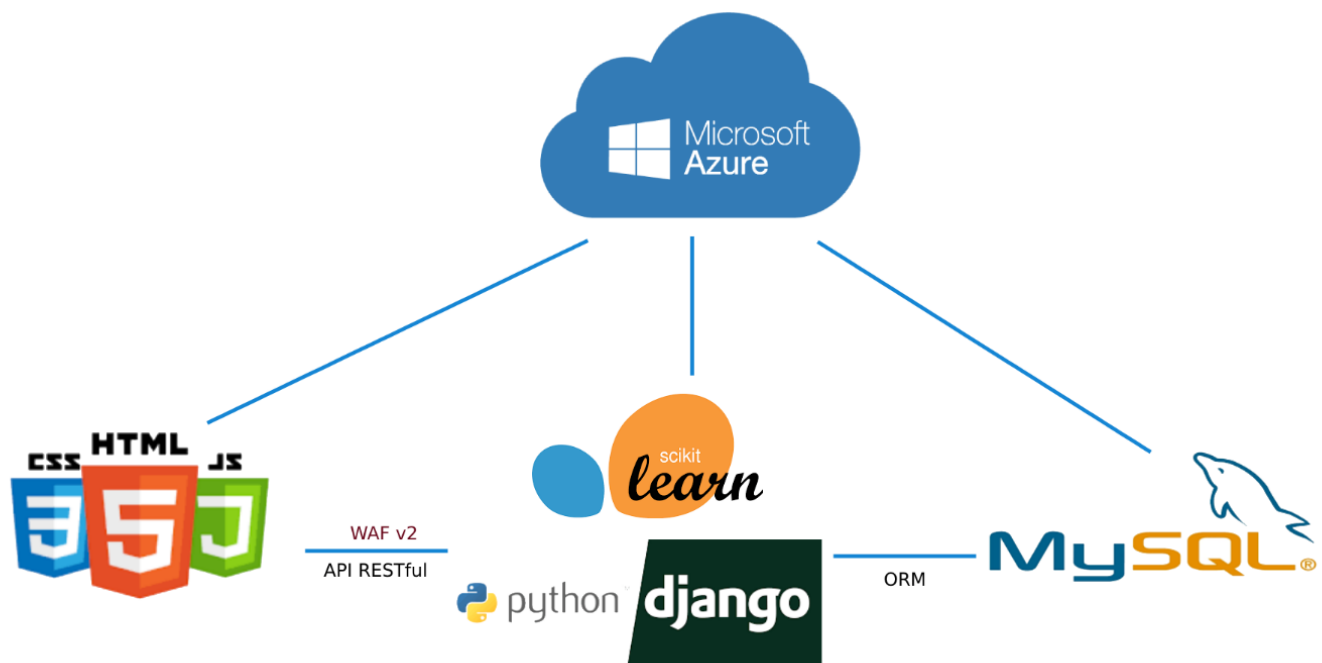


Ilustración 1. Arquitectura del sistema

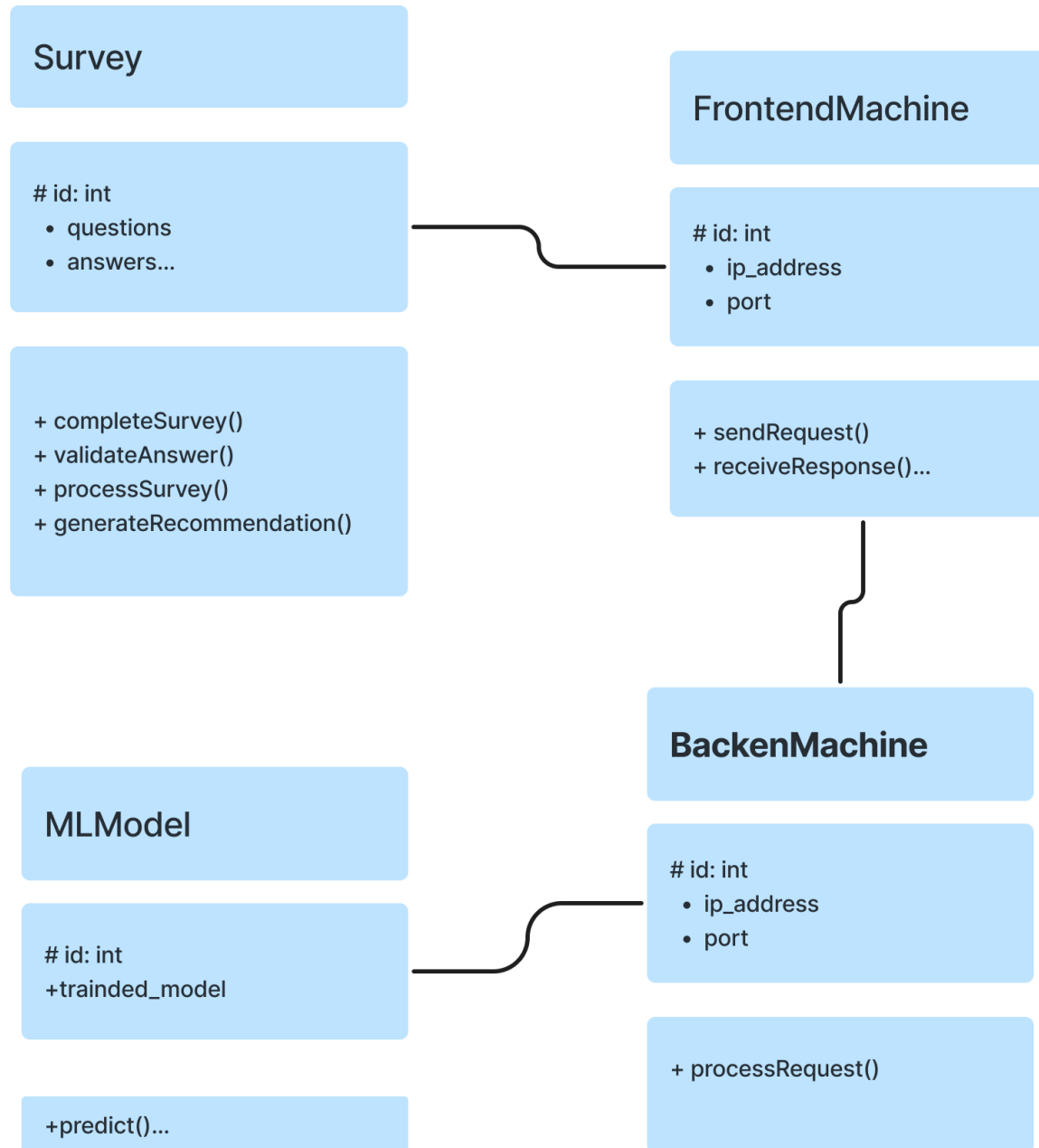


Ilustración 2. Diagrama de Clases

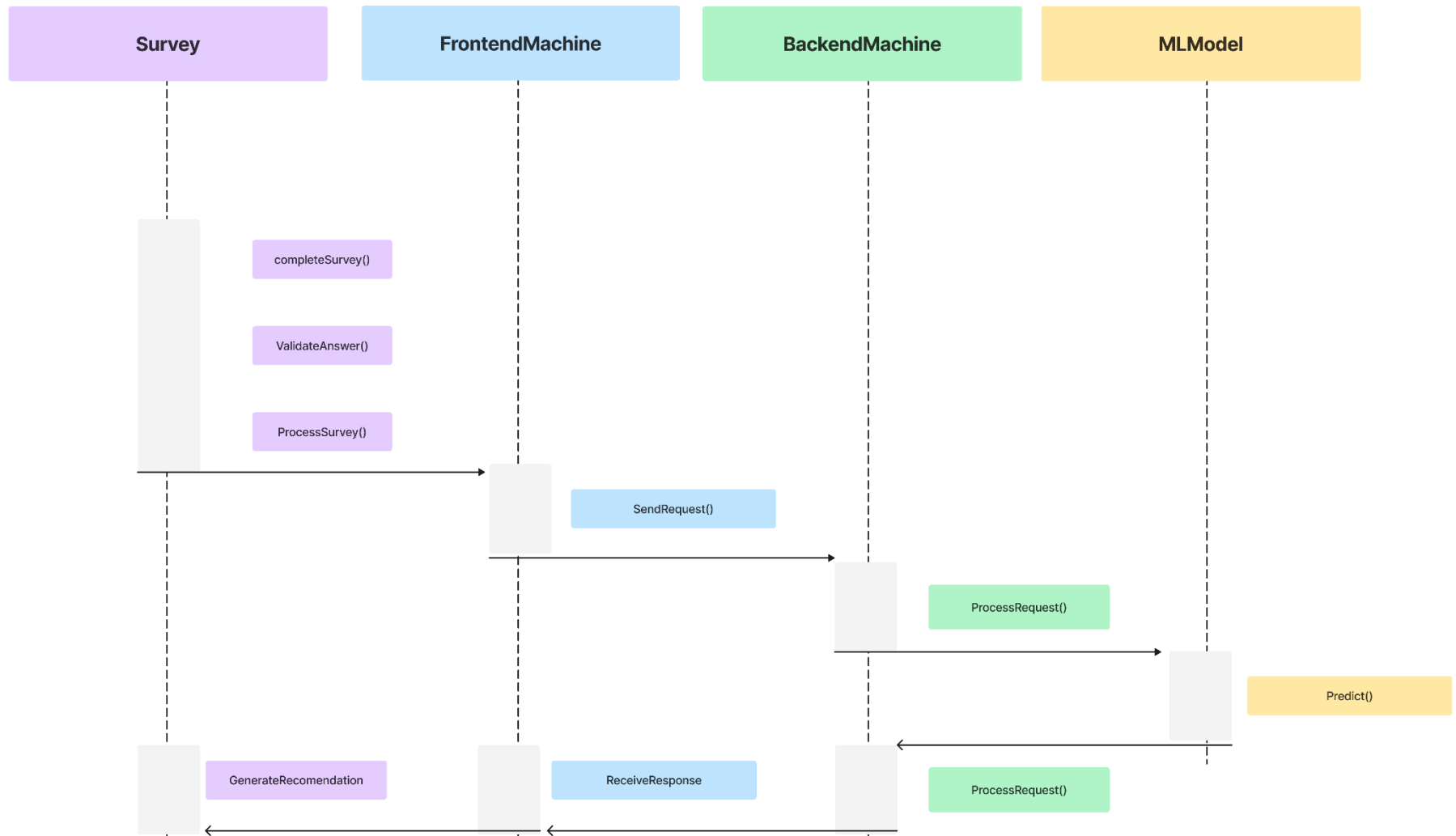


Ilustración 3. Diagrama de Flujo

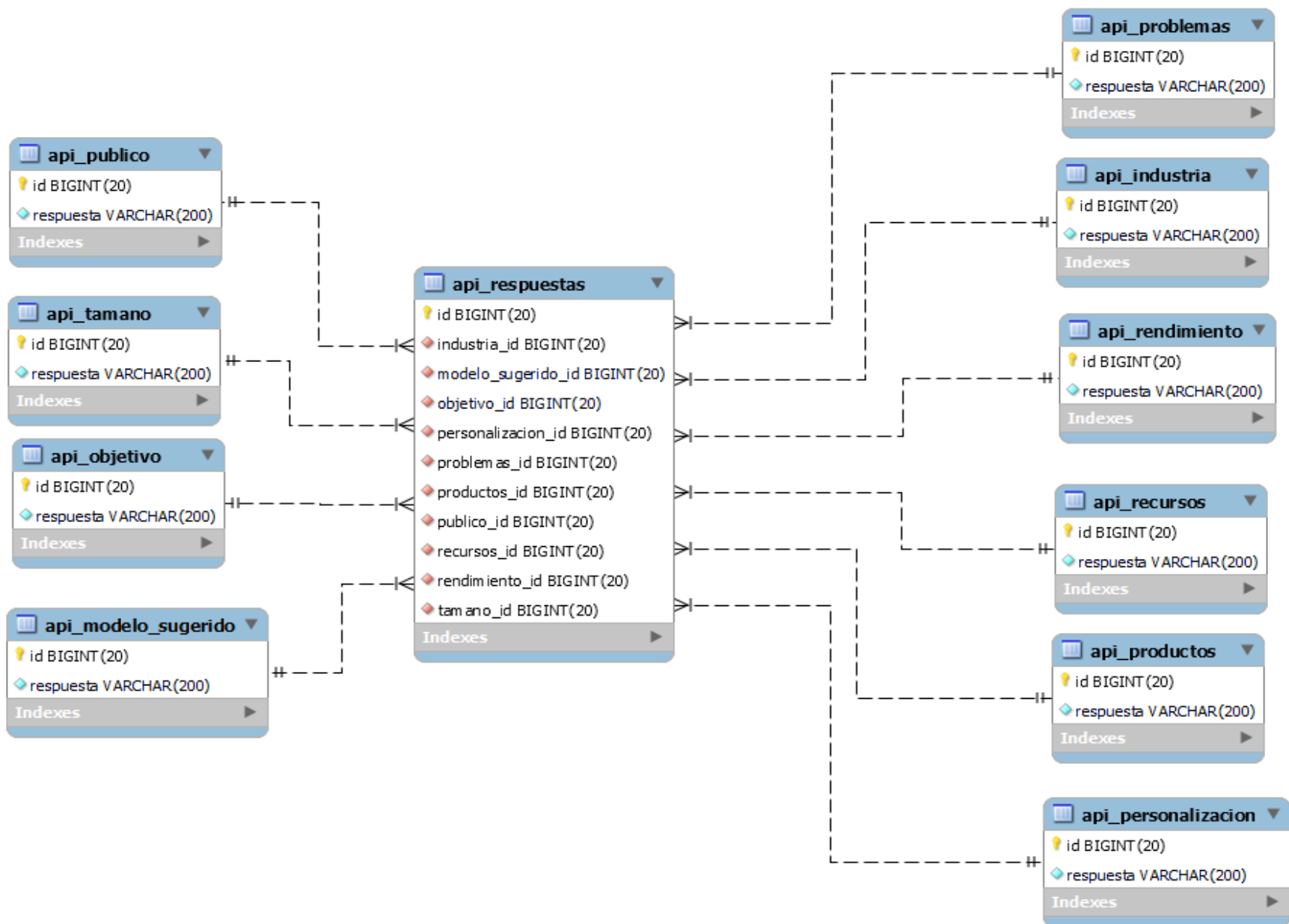


Ilustración 4. Diagrama de modelo de datos

En este proyecto, se enfrentó un escenario en el que se desea determinar el modelo de servicio ideal a partir de 10 preguntas. Para abordar este desafío, se utilizó Scikit-learn para desarrollar una red neuronal que pueda realizar este cálculo. La red neuronal se entrenará con un conjunto de datos que consta de 10 respuestas, junto con su respectivo modelo de servicios.

Para desarrollar una red neuronal utilizando Scikit-learn, correspondiente a lo antes mencionado, se pueden seguir los siguientes pasos:

1. Importar las bibliotecas necesarias:

```
import pandas as pd
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
```

2. Definir una clase **data_set_attraction** que represente el conjunto de datos de atracción. Esta clase tendrá atributos para almacenar los conjuntos de entrenamiento y prueba:

```
class data_set_attraction:
    def __init__(self, x_train, y_train, x_test, y_test):
        self.x_train = x_train
        self.y_train = y_train
        self.x_test = x_test
        self.y_test = y_test
```

3. Crear una función **generate_train_test** que cargue los datos de un archivo y genere el conjunto de entrenamiento y prueba utilizando la función **train_test_split**:

```
def generate_train_test(file_name):
    df = pd.read_excel(file_name)

    X = df.drop(['Modelo_sugerido'], axis=1).values
    Y = df['Modelo_sugerido'].values

    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0, shuffle=True)

    return data_set_attraction(x_train, y_train, x_test, y_test)
```

4. La función **train_model** se encarga de entrenar un modelo de red neuronal utilizando el conjunto de datos **corpus**. Primero, se asignan las características y etiquetas de entrenamiento y prueba a variables separadas. Luego, se crea un objeto **MLPClassifier** con los parámetros deseados y se entrena el modelo utilizando los datos de entrenamiento. A continuación, se realizan predicciones en los datos de prueba. Para su posterior uso, el modelo entrenado se guarda en un archivo llamado 'modelo.pkl' utilizando la biblioteca **pickle**. Esto permitirá cargar el modelo en otro contexto, como una API de Django, y realizar predicciones en nuevos datos.

```
def train_model(corpus):
    x_train = corpus.x_train
    x_test = corpus.x_test
    y_train = corpus.y_train
    y_test = corpus.y_test

    model = MLPClassifier(activation='relu', alpha=0.0001, hidden_layer_sizes=100, solver='adam');
    model.fit(x_train, y_train);
    y_pred = model.predict(x_test)

    with open('modelo.pkl', 'rb') as f:
        pickle.dump(model, f)
```


Para hacer uso de este modelo de machine learning dentro de la tarea objetivo, se utilizó el Framework Django, ya que además de ser bastante flexible, se le pueden agregar las bibliotecas necesarias para poder ejecutar la red neuronal.

Paso 1: Configuración básica

- Crear un proyecto y configurar la base de datos en el archivo **settings.py**.
- Agregar la aplicación API a la configuración del proyecto en el archivo **settings.py**.
- Asegúrese de tener Django REST Framework instalado.

Paso 2: Define las URLs

- En el archivo **urls.py** del proyecto, se agregan el siguiente código para configurar las URL de la API:

```
from django.urls import path
from .views import *

app_name = 'api'

urlpatterns = [
    path('industrias', Industria_API.as_view()),
    path('objetivos', Objetivo_API.as_view()),
    path('productos', Productos_API.as_view()),
    path('tamano', Tamano_API.as_view()),
    path('publico', Publico_API.as_view()),
    path('problemas', Problemas_API.as_view()),
    path('rendimiento', Rendimiento_API.as_view()),
    path('personalizacion', Personalizacion_API.as_view()),
    path('recursos', Recursos_API.as_view()),
    path('respuestas', Respuestas_API.as_view()),
]
```

Cada URL corresponde a los diferentes métodos HTTP, todas las url proporcionadas cuentan solamente con el método Get los cuales solo obtienen las respuestas de las preguntas a excepción de la url de **respuestas**, ya que como su nombre lo indica, es la url que proporciona la respuesta, predicha por la red neuronal, a partir de las respuestas proporcionadas de un sitio web, en este caso.

Además de hacer las migraciones de las bases de datos, y agregar el método GET a cada url, la clase **Respuestas_API** es una parte de un programa que recibe respuestas a un formulario y proporciona un modelo de servicio basado en esas respuestas.

Cuando se envían las respuestas a través de una solicitud al programa, este verifica si las respuestas son válidas y correctas. Si son válidas, realiza lo siguiente:

1. El programa extrae los datos de las respuestas y los guarda temporalmente.
2. Utiliza un modelo de machine learning previamente entrenado para hacer una predicción basada en las respuestas.
3. Con la predicción realizada, obtiene un modelo de servicio sugerido.
4. El programa guarda las respuestas en una base de datos junto con el modelo de servicio sugerido.
5. Finalmente, el programa te devuelve una respuesta que incluye el modelo de servicio sugerido.

Si las respuestas no son válidas o hay algún problema, el programa informará sobre los errores encontrados.

En resumen, la clase **Respuestas_API** se encarga de recibir respuestas, procesarlas utilizando un modelo de machine learning y ds una recomendación de modelo de servicio basada en esas respuestas.

Con esto, la clase **Respuestas_API** maneja las solicitudes POST, valida los datos enviados, realiza la predicción utilizando un modelo de machine learning y guarda los resultados en la base de datos, devolviendo una respuesta HTTP con los datos del modelo.

Para visualizar las respuestas obtenidas desde Django, se desarrolló un sitio web utilizando HTML, CSS y JavaScript. El objetivo principal del sitio web era presentar un formulario simple para recopilar información y enviarla a través de una API. Para lograr esto, se hizo uso de la tecnología Ajax con jQuery para consumir la API de Django y realizar las solicitudes de manera asíncrona.

En este proyecto en particular, se optó por no utilizar ningún framework como Angular, React o Vue.js debido a que el objetivo era implementar un formulario sencillo y no se requería la complejidad adicional que ofrecen estos frameworks. HTML, CSS y JavaScript proporcionaron todas las herramientas necesarias para crear la interfaz de usuario, aplicar estilos y agregar interactividad al formulario.

La biblioteca jQuery, específicamente su módulo Ajax, fue utilizada para realizar las solicitudes HTTP de forma sencilla y eficiente. Ajax permite enviar y recibir datos en segundo plano sin tener que recargar la página completa, lo cual es especialmente útil para interactuar con una API y actualizar dinámicamente el contenido de la página sin interrupciones.

El fragmento de código en JavaScript que la biblioteca jQuery, sirve para realizar solicitudes a una API en Django y manipular el contenido de un formulario y una respuesta.

- En primer lugar, se utiliza `$(document).ready(function(){})` para asegurarse de que el código se ejecute una vez que el documento HTML esté completamente cargado.
- A continuación, se realizan varias solicitudes Ajax utilizando `$.ajax({})`. Cada solicitud Ajax se realiza a una URL específica de la API en Django, utilizando el método HTTP GET para obtener datos en formato JSON.
- Después de cada solicitud exitosa, se realiza una función de éxito (**success**) que recibe la respuesta de la API. Dentro de esta función, se realiza una iteración (`response.forEach()`) en la respuesta recibida, donde cada elemento se procesa.
- Dentro del bucle `forEach`, se selecciona un elemento del formulario utilizando un identificador específico (`$('#industria')`, `$('#objetivo')`, etc.). Luego, se agrega una opción al elemento seleccionado utilizando `option.append($('').val(element.id.toString()).text(element.respuesta))`. Esto crea un nuevo elemento `<option>` dentro del elemento seleccionado, con el valor y el texto obtenidos de la respuesta de la API.
- Después de completar todas las solicitudes Ajax y de haber agregado las opciones correspondientes a los elementos del formulario, se define una función para manejar el envío del formulario (`$('#formPreguntas').submit(function(event){})`). Dentro de esta función, se evita el comportamiento predeterminado de enviar el formulario (`event.preventDefault()`).
- A continuación, se recopilan los datos del formulario en un objeto llamado `datosFormulario`, donde cada valor se extrae de los elementos del formulario utilizando `parseInt($('#industria').val())`, `parseInt($('#objetivo').val())`, etc.
- Luego, se convierte el objeto `datosFormulario` en una cadena JSON utilizando `JSON.stringify(datosFormulario)`. Esto es necesario para enviar los datos al servidor a través de una solicitud Ajax.

Finalmente, se realiza una solicitud Ajax adicional utilizando el método HTTP POST para enviar los datos del formulario al servidor. En caso de éxito, se muestra una notificación de éxito utilizando la biblioteca SweetAlert con el resultado obtenido de la respuesta de la API. En caso de error, se muestra una notificación de error.

En resumen, este programa en JavaScript utiliza la biblioteca jQuery y Ajax para realizar solicitudes a una API en Django. Recupera datos de la API y los utiliza para llenar un formulario en un sitio web. Además, permite enviar los datos del formulario al servidor y muestra una notificación con la respuesta obtenida.

Para montar el proyecto en Azure se siguieron los pasos de la práctica 10, más unos otros ajustes dentro de las máquinas virtuales que sirvieron las peticiones en el BackEnd.

- **Instalación de los paquetes necesarios**

Antes de comenzar es necesario tener todos los componentes necesarios para la configuración de la máquina virtual

- `sudo apt-get install nginx`
- `sudo apt-get install python3-venv`
- `sudo apt-get install python3-dev default-libmysqlclient-dev build-essential`

- **Preparación del entorno virtual**

Instalar la herramienta **venv** para crear entornos virtuales:

```
pip install virtualenv
```

Crear un entorno virtual llamado "Django-venv"

```
Python -m venv Django-venv
```

Activar el entorno virtual

```
Source Django-venv/bin/activate
```

- **Instalación de paquetes necesarios**

Se deben instalar los paquetes necesarios dentro del entorno virtual creado:

```
pip install scikit-learn django djangorestframework djangorestframework-simplejwt django-model-utils mysqlclient gunicorn django-cors-headers
```

- **Migración de la base de datos**

Esta migración se hace a una base de datos creada en Azure

```
mysql -u nombre_usuario -p proyecto < sql_db.sql
```

Hay que asegurarse de reemplazar "nombre_usuario" con el nombre de usuario de MySQL; en este caso 'proyecto' no se cambia ya que es el nombre de la base de datos.

- **Configuración del servidor de desarrollo**

Creación de un archivo de configuración para Gunicorn

```
mkdir conf
sudo nano conf/gunicorn_config.py
```

Dentro del achivo de configuración, se agregaron las siguientes líneas:

```
command = '/ruta/a/entorno/vitual/bin/gunicorn'
pythonpath = '/ruta/del/proyecto/proyecto_sd'
bind = '0.0.0.0:8000'
workers = 3
```

Por último, se ejecuta el puente entre Gunicorn y el servidor de desarrollo de Django

```
gunicorn -c conf/gunicorn_config.py proyecto_sd.wsgi
```

Para ejecutarlo en segundo plano se recomienda hacer **Ctrl+z**.

- **Configuración del servidor local**

Iniciar NGINX

```
sudo systemctl start nginx
```

Configuración del archivo que hace la conexión entre NGINX y Gunicorn

```
server {  
    listen 80;  
    server_name direccion_gateway;  
    location / {  
        proxy_pass http://0.0.0.0:8000;  
    }  
}
```

Una vez hecho esto, se hace enlaza con la carpeta de sitios activos, para poder acceder a los Endpoints de Django

```
sudo ln -s /etc/nginx/sites-available/proyecto_sd
```

Hecho esto, se reinicia el servidor y ya se puede acceder a la url configurada en el **server_name** para hacer uso de los recursos que proporciona cada una de las URL's configuradas en Django.

```
sudo systemctl restart nginx
```

PRUEBAS

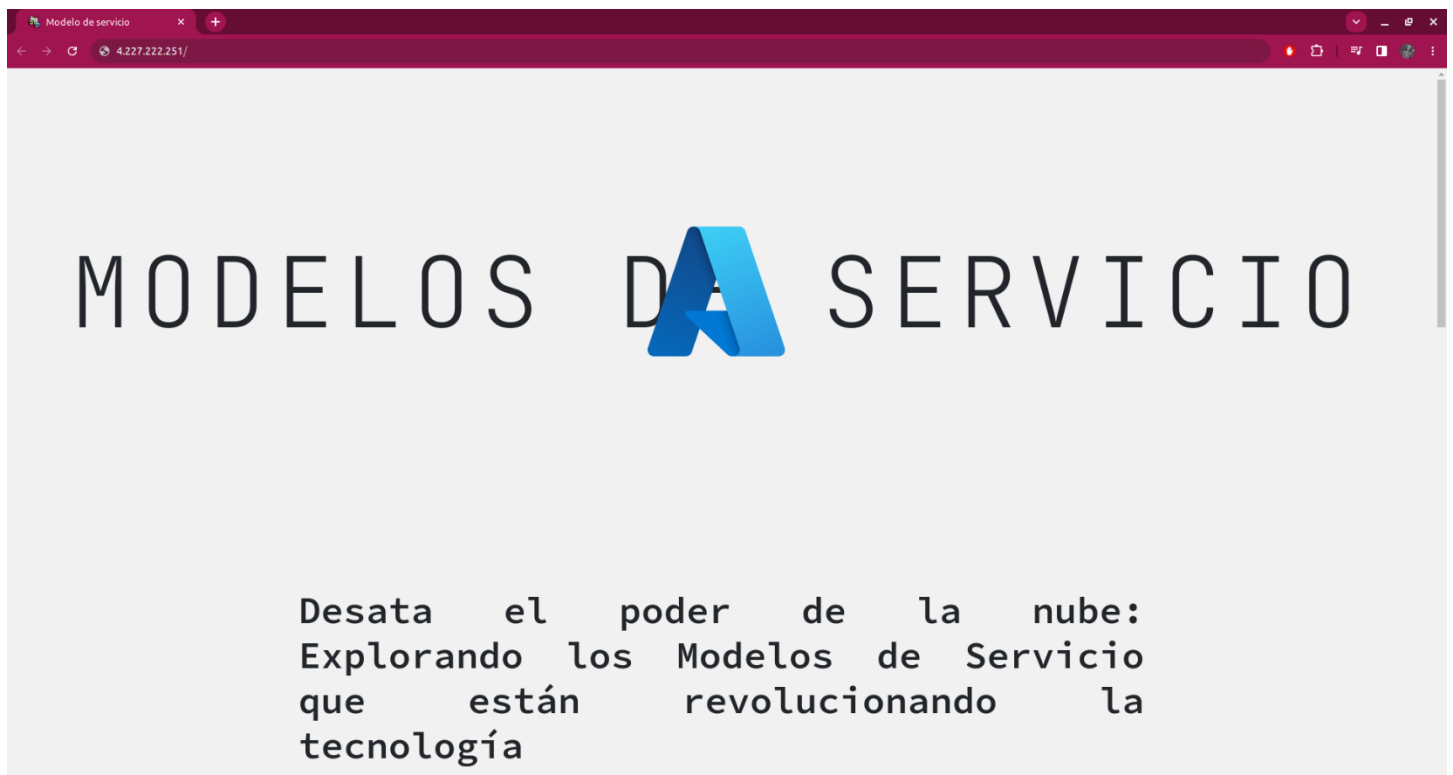


Ilustración 5. Ingreso a la página principal

Modelo de servicio x +

← → ↻ 4.227.222.251/

nube perfecta para tu negocio: ¡Completa nuestro formulario y encuentra tu mejor opción en minutos!

¿En qué industria opera tu empresa?

Tecnología

¿Cuál es su objetivo principal?

Innovación

¿Qué tipo de productos o servicios ofrece tu empresa actualmente?

Desarrollo

¿Cuál es el tamaño de tu empresa?

Mediana

¿Cuál es tu público objetivo o mercado objetivo?

Profesionales

¿Qué problemas o desafíos específicos esperas abordar con el modelo de servicio?

Eficiencia

¿Cuáles son tus expectativas: rendimiento, eficiencia o escalabilidad del modelo de servicio?

Rendimiento

¿Qué es lo más importante para tu modelo de servicio?

Personalización

¿Cuál sería un factor que limitaría tu modelo de servicio?

Presupuesto

Calcular

Ilustración 6. Formulario con las respuestas posibles proporcionadas por la API en Django

Modelo de servicio x +

← → ↻ 4.227.222.251/

nube perfecta para tu negocio: ¡Completa nuestro formulario y encuentra tu mejor opción en minutos!

¿En qué industria opera tu empresa?

Tecnología

¿Cuál es su objetivo principal?

Innovación

¿Qué tipo de productos o servicios ofrece tu empresa actualmente?

Desarrollo

¿Cuál es el tamaño de tu empresa?

Mediana

¿Cuál es tu público objetivo o mercado objetivo?

Profesionales

¿Qué problemas o desafíos específicos esperas abordar con el modelo de servicio?

Eficiencia

¿Cuáles son tus expectativas: rendimiento, eficiencia o escalabilidad del modelo de servicio?

Rendimiento


¿Qué es lo más importante para tu modelo de servicio?

Personalización

¿Cuál sería un factor que limitaría tu modelo de servicio?

Presupuesto

Calcular



Registro almacenado

Modelo sugerido: Servicio gestionado de marketing

OK

Ilustración 7. Modelo sugerido por el modelo entrenado y servido por Django

CUESTIONARIO

Responde las siguientes preguntas:

¿Cuáles crees que sean las ventajas de tener tu proyecto montado en la nube en lugar de usar un servidor tradicional?

Escalabilidad: La nube permite escalar verticalmente o horizontalmente según las necesidades del proyecto. Puedes aumentar o disminuir la capacidad de almacenamiento, poder de procesamiento y recursos de red de manera flexible y rápida.

Alta disponibilidad: Los servicios en la nube generalmente ofrecen una alta disponibilidad, lo que significa que tu aplicación estará accesible para los usuarios en cualquier momento y desde cualquier lugar. Los proveedores de la nube suelen tener redundancia en sus servidores y centros de datos para minimizar los tiempos de inactividad.

Flexibilidad geográfica: Puedes desplegar tu proyecto en la nube en diferentes regiones geográficas para acercar tu aplicación a los usuarios finales y mejorar la latencia.

Seguridad: Los proveedores de la nube implementan medidas de seguridad avanzadas para proteger tus datos y aplicaciones. Tienen equipos especializados en la seguridad de la infraestructura y ofrecen opciones de cifrado y autenticación.

Gestión simplificada: La nube facilita la gestión de recursos informáticos, ya que no es necesario preocuparse por la adquisición, instalación y mantenimiento de hardware. Además, los servicios en la nube suelen ofrecer herramientas de monitoreo, escalado automático y administración centralizada.

En cuanto al proyecto en sí, ¿Cómo te sentiste al realizarlo? ¿fueron complicadas las secciones?

Fue un poco complicado al principio porque no sabíamos los alcances que podía tener o si se necesitaba algo tan complicado para poder lograr un funcionamiento adecuado que cumpliera con todos los requisitos, por lo que se tuvo que realizar una gran investigación de los mejores elementos que pudimos integrar en el proyecto. Al final en base a que también se presento en la EXPOESCOM se decidió implementar algo un poco mas complejo de lo que se requería, al integrar machine learning se logro este factor de innovación y de algo mas completo al sistema.

¿Te parecieron demasiadas secciones o consideras que fueron las adecuadas?

Fueron las secciones adecuadas para un proyecto final

¿Sobre el tiempo dedicado a la realización, ¿consideras que fue el adecuado o fue excesivo??

Fue un tiempo razonable, pero creo que estuvo justo para lo que se quería lograr al final, hubiera estado bien un poco mas de tiempo.

CONCLUSIONES

En conclusión, el proyecto tiene como objetivo principal implementar y desplegar una aplicación Django en Azure utilizando NGINX y Gunicorn. Se proporcionan pasos detallados para configurar el entorno y asegurar un funcionamiento eficiente y escalable de la aplicación.

El proyecto incluye una serie de requerimientos funcionales y no funcionales que deben tenerse en cuenta durante la implementación. Estos requerimientos van desde el registro de usuarios, la gestión de encuestas y el procesamiento de solicitudes backend, hasta la integración de modelos de Machine Learning para generar recomendaciones personalizadas.

Se han establecido reglas de negocio para garantizar la validez de las respuestas de las encuestas, así como para equilibrar la carga de trabajo en las máquinas backend y gestionar errores de manera adecuada.

Al montar el proyecto en la nube en lugar de utilizar un servidor tradicional, se obtienen ventajas como la escalabilidad, alta disponibilidad, flexibilidad geográfica, seguridad y una gestión simplificada de los recursos informáticos.

En general, el proyecto ofrece una guía completa para implementar una aplicación Django en Azure, cubriendo aspectos clave como la configuración del servidor web, el despliegue de la aplicación, la comunicación entre componentes y la integración de modelos de Machine Learning. Siguiendo los pasos y teniendo en cuenta los requerimientos y reglas de negocio, es posible desarrollar una aplicación robusta y eficiente en la nube. Escribe las conclusiones sobre el proyecto final}

BIBLIOGRAFÍA

1. Django Software Foundation. (s.f.). Django documentation. Recuperado de <https://docs.djangoproject.com/>
2. Microsoft. (s.f.). Azure documentation. Recuperado de <https://docs.microsoft.com/azure/>
3. NGINX. (s.f.). NGINX documentation. Recuperado de <https://docs.nginx.com/>