Universidad Nacional de General Sarmiento

INFORME DEL TRABAJO GRUPAL “POKÉDEX”

- **Materia:** Introducción a la programación
- **Comision:** 09
- **Docentes:** Miguel Rodríguez y Nancy Nores
- **Alumnos:**
  - Bertoneri Eliana
  - Castro Yanel
  - Muñoz Juliana
- **Fecha de entrega:** 01/07/2025

• **Índice**:

• Introducción:

En este trabajo hicimos una Pokedéx. Esta es una página web en donde se pueden visualizar las imágenes de los distintos pokémones y datos importantes relacionados a ellos, como sus tipos, altura, peso y nivel base. Gran parte del proyecto ya estaba resuelto, nosotras terminamos algunas funciones que estaban incompletas y renovamos su interfaz gráfica. Para realizar el trabajo se utilizó Visual Studio Code (editor de código), Python (back-end), Django (para poder hacer la página web), una API (nos proporcionó las imágenes e información sobre los pokémones) y Bootstrap (para renderizar las cards). Además, para la estructura y diseño de la página web se utilizaron seis archivos HTML (index, header, home, login, favourites, footer) y uno de CSS. Para que las distintas integrantes puedan observar los cambios y para controlar las versiones del trabajo se utilizó Git Hub.

• Desarrollo:

- Instalación de los programas:

Durante la instalación tuvimos algunos inconvenientes, tuvimos que instalar Python, Django y Bootstrap varias veces para que funcionen. Nos aparecía un mensaje que decía que no estaba instalado o que debía actualizarse. Para desinstalarlo/actualizarlo, recurrimos a la ayuda de la Inteligencia Artificial, y nos fue guiando para instalar correctamente las tecnologías.

- Funciones obligatorias:

Con ayuda y guía de los docentes, pudimos completar las siguientes funciones obligatorias:

(1) views.py

```python
# esta función obtiene 2 listados: uno de las imágenes de la API y otro de favoritos, ambos en formato Card,
# y los dibuja en el template 'home.html'.
def home(request):
    #Punto 1 ----------
    images = services.getAllImages()
    favourite_list = []

    return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })
```

La función ***home*** recibe como parámetro el request. Devuelve dos listados: el de las imágenes de la API y el de favoritos. En nuestro caso, al no haber desarrollado la función de favoritos, devuelve un listado vacío en la lista de favoritos.
Para obtener las imágenes llamamos la función ***getAllImages*** que se encuentra en ***services***.

(2) services.py

```python
# función que devuelve un listado de cards. Cada card representa una imagen de la API de Pokemon
def getAllImages():
    # debe ejecutar los siguientes pasos:
    # 1) traer un listado de imágenes crudas desde la API (ver transport.py)
    # 2) convertir cada img. en una card.
    # 3) añadirlas a un nuevo listado que, finalmente, se retornará con todas las card encontradas.

    #Punto 2 -------------
    lista_img=transport.getAllImages()
    lista_cards=[]
    for elemento in lista_img:
        card=translator.fromRequestIntoCard(elemento)
        types_aux = []
        for t in card.types:
            types_aux.append(get_type_icon_url_by_name(t))
        card.types_imgs=types_aux
        lista_cards.append(card)
    return lista_cards
```

La función ***getAllImages*** ubicada en ***services.py*** obtiene una lista de objetos JSON de la función ***getAllImages*** ubicada en ***transport.py***. Luego, con el bucle for recorremos la lista de imágenes y las transformamos en cards (para ello utilizamos la función ***fromRequestIntoCard*** de ***traslator.py***). Cada card es guardada en una lista de cards. Adentro de este for, tenemos otro, el cual se encarga de procesar los tipos de pokémon y, posteriormente, guardarlos.
Con estas dos funciones implementadas, podemos visualizar correctamente las distintas cartas de los pokémones en “Galería”.

(3) home.html
Luego, debíamos modificar el HTML de Home para que el borde de las cards cambie de color dependiendo del tipo de pokémon (si es “grass”, debe ser verde; si es “fire”, debe ser rojo; si es “water” debe ser azul; y los tipos restantes deben ser naranja). Para esto, utilizamos los condicionales de Django de la siguiente manera:

```
</div>

  <div class="row row-cols-1 row-cols-md-3 g-4">
  {% if images|length == 0 %}
  <h2 class="text-center">La búsqueda no arrojó resultados...</h2>
  {% else %} {% for img in images %}
  <div class="col">
      <!-- evaluar si la imagen pertenece al tipo fuego, agua o planta -->
      <div class="card border-dark border-4 {% if 'grass' in img.types %} bg-success {% elif 'fire' in img.types %} bg-danger {% elif 'water' in img.types %} bg-info {% else %} bg-warning {% endif %} mb-3 ms-5" style="max-width: 540px; ">
          <div class="row g-0">
              <div class="col-md-4">
                  <img src="{{ img.image }}" class="card-img-top" alt="imagen">
              </div>
```

Código de la imagen:
<div class="card border-dark border-4 {% if 'grass' in img.types %} bg-success {% elif 'fire' in img.types %} bg-danger {% elif 'water' in img.types %} bg-info {% else %} bg-warning {% endif %} mb-3 ms-5" style="max-width: 540px; ">

En lugar de cambiar el color de los bordes dependiendo del tipo de pokémon, cambiamos el color del fondo y modificamos los bordes para que todos sean negros. Para esto, leímos la documentación de Bootstrap.

<div class="card border-dark border-4

El borde es *dark*

Modificamos el grosor del borde de las cards

{% if 'grass' in img.types %} bg-success

CONDICIONAL

Si 'grass' está en img.types, el fondo será *sucess*

{% elif 'fire' in img.types %} bg-danger

Si 'fire' está en img.types, el fondo será *danger*

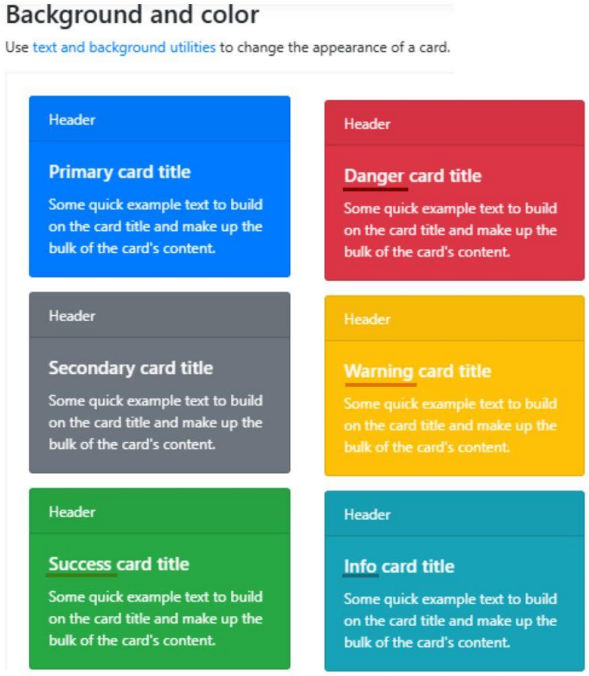
{% elif 'water' in img.types %} bg-info

Si 'water' está en img.types, el fondo será *info*

{% else %} bg-warning {% endif %}

Sino, el fondo será *warning*

mb-3 ms-5" style="max-width: 540px; ">



Documentacion de Bootstrap sobre cards

De esta manera, logramos que las cards se vean como se muestran a continuación:

- Funciones opcionales:

Buscador I (según el nombre del pokémon)

En esta función, que se ubica en ***views.py***, se deben filtrar correctamente los pokémones para que aparezcan solamente los que coinciden con la búsqueda, es decir, con lo ingresado por el usuario. En caso de que el usuario no ingrese ningún dato, debe mostrar todas las imágenes, al igual que cuando apretamos “Galería”.

```python
# función utilizada en el buscador.
def search(request):
    name = request.POST.get('query', '')

    # si el usuario ingresó algo en el buscador, se deben filtrar las imágenes por dicho ingreso.
    if (name != ''):
        images = services.filterByCharacter(name)
        favourite_list = []

        return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })
    else:
        return redirect('home')
```

La función ***search*** recibe como parámetro el request. Si lo que ingresó el usuario no está vacío, se llama a la función ***filterByCharacter*** de ***services.py***. Esta recibe como parámetro

lo que ingreso el usuario, y devuelve el listado de las cards que coincidan con lo ingresado. En caso contrario, si lo ingresado por el usuario esta vacío, redirige a “Home.”
A continuación, podemos observar la función ***filterByCharacter***, mencionada anteriormente:

```python
# función que filtra según el nombre del pokemon.
def filterByCharacter(name):
    filtered_cards = []
    name=name.lower()
    for card in getAllImages():
        if name in card.name:
            filtered_cards.append(card)
        # debe verificar si el name está contenido en el nombre de la card, antes de agregarlo al listado de filtered_cards.


    return filtered_cards
```

Como dijimos, recibe como parámetro lo ingresado por el usuario, crea una lista vacía de cartas filtradas y convierte todos los caracteres de la cadena ingresada a minúscula, para que no haya problemas cuando recorremos las cards. Luego, con un ciclo for se recorren las cards de la lista de ***getAllImages***, y, en caso de que lo ingresado se encuentre en el nombre de la card, esta se añade a la lista de cartas filtradas. Finalmente, esta función devuelve la lista de cartas filtradas.

Buscador II (según tipo fuego, agua o planta)

El segundo buscador es muy similar al primero, la diferencia es que este en lugar de filtrar por nombres, filtra por tipo de pokémon. Para esto, utilizamos la función ***filter_by_tipe*** que se ubica en ***services.py***. Esta recibe como parámetro el request. En una variable ***type*** obtiene el tipo de pokémon que el usuario solicito mostrar. Luego, si el usuario apretó algún tipo, llama a la función ***filterByType***, con parámetro ***type*** (la variable mencionada anteriormente). Esta función, que vamos a explicar más adelante de manera más detallada, se encarga de traer un listado de imágenes filtradas, las cuales tienen el tipo de pokémon que el usuario pidió. Luego retorna la lista de imágenes filtradas.
En caso contrario, si type == ' ', redirige a “Galería”.

```python
# función utilizada para filtrar por el tipo del Pokemon
def filter_by_type(request):
    type = request.POST.get('type', '')

    if type != '':
        images = services.filterByType(type) # debe traer un listado filtrado de imágenes, segun si es o contiene ese tipo.
        favourite_list = []

        return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })
    else:
        return redirect('home')
```

La función ***filterByType*** de ***services*** es la siguiente:

```python
# función que filtra las cards según su tipo.
def filterByType(type_filter):
    filtered_cards = []
    type_filter=type_filter.lower()
    for card in getAllImages():
        if type_filter in card.types:
            filtered_cards.append(card)
        # debe verificar si la casa de la card coincide con la recibida por parámetro. Si es así, se añade al listado de filtered_cards.
    return filtered_cards
```

Como lo mencionamos, esta función recibe como parámetro el tipo de pokémon que solicitó el usuario. Crea una lista vacía de cartas filtradas, y, al igual que el anterior buscador, convierte los caracteres del tipo de pokémon en minúscula, para evitar problemas cuando recorramos las cards. A continuación, con un ciclo for, recorre cada card en la lista de ***getAllImages,*** y, si el tipo de carta solicitada por el usuario está en los tipos de la card, la

agrega a la lista de cartas filtradas. Una vez terminado el ciclo, retorna la lista de cartas filtradas.

- Interfaz gráfica:

En cuanto a la interfaz gráfica, modificamos el fondo de la página, las fuentes, el color del header y agregamos animaciones a los títulos. Además, agregamos un GIF de Pikachu saludando en index.html.

Fondo

```css
body {
    background-image: url("./images/fondo-index.png");
    background-size: cover;
}
```

Para esto, modificamos el archivo CSS, antes había un color liso de fondo, lo modificamos para que el fondo sea un degradado de amarillo a rojo. Para guardar la imagen que pusimos como fondo, creamos una carpeta llamada ***images*** en ***static***. Luego, para establecer esta imagen como fondo, utilizamos la propiedad ***background-image***, y copiamos la ruta de la imagen. A continuación, utilizamos la propiedad ***background-size*** para modificar el tamaño del fondo, utilizamos ***cover*** para que la imagen cubra completamente el fondo.

Fuentes

```
{% extends 'header.html' %} {% load static %} {% block content %} {% if request.user.is_authenticated %}
<h2 class="text-center titulos animate__animated animate__bounce">Bienvenido {{ user.username | upper }}!</h2>
{% else %}
<h2 class="text-center titulos animate__animated animate__bounce">Bienvenido!</h2>
{% endif %}
<p class="text-center font">
    Mirá las imágenes desde <a href="{% url 'home' %}">este link.</a>
</p>

<!-- GIF Pikachu -->
<img src="{% static '/images/gif-pikachu.gif' %}" class="gif_pikachu" alt="Imagen de Pikachu saludando">

{% endblock %}
```

```css
.font {
    font-family: "Farsan", cursive;
    font-weight: 400;
    font-style: normal;
    font-size: 20px;
    color: black;
}

/* Solamente los titulos tienen la fuente de Pokemon */
.titulos {
    font-family: 'Fuente Pokemon', sans-serif;  /* En caso de que no funcione la fuente Pokemon, utilizará sans-serif */
    margin-top: 25px;
    margin-bottom: 20px;
    font-size: 45px;
}
```

Para modificar las fuentes creamos dos clases: ***font*** y ***titulos***. La clase ***font*** se utiliza en todos los textos excepto en los títulos, mientras que la clase ***titulos*** se utiliza en los títulos de cada HTML: “Bienvenido!”, “Buscador de Pokémon” e “Inicio de sesión”. En CSS

declaramos la fuente con *font-family*, y luego le agregamos algunos detalles como el tamaño, el color, el estilo, entre otros. La fuente que utilizamos en la clase ***font***, “Farsan”, la importamos de Google Fonts (https://fonts.google.com/). Para esto, copiamos el código que nos dio Google Fonts en el ***head*** de ***header.html*** de la siguiente manera:

```
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>POKEDEX - IP</title>
    {% load bootstrap5 %} {% bootstrap_css %} {% bootstrap_javascript %} {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static 'styles.css' %}">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <!-- Font "Farsan" - Google Fonts -->
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Farsan&display=swap" rel="stylesheet">
    <!-- Animación títulos de CSS (Página Animate.css) -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css"/>
</head>
```

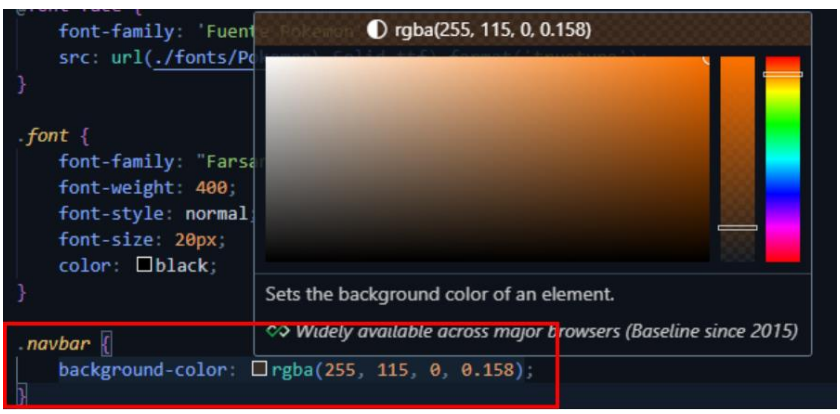
Luego, declaramos “Farsan” como la fuente, en la primera imagen se puede observar.

En la clase ***titulos***, importamos una fuente personalizada. Descargamos de internet la fuente de Pokémon para aplicársela a los títulos. Creamos una carpeta de ***fonts*** dentro de ***static***. Allí, guardamos los archivos de la fuente. Para declararla, utilizamos la regla ***@font-face***, en donde definimos el nombre de la fuente, la ruta y formato.

```
/* Fuentes Personalizadas */
@font-face {
    font-family: 'Fuente Pokemon';
    src: url(./fonts/Pokemon\ Solid.ttf) format('truetype');
}
```

Color header

Para cambiar el color del header, utilizamos la propiedad ***background-image*** en la clase ***navbar*** de la siguiente manera:



Cards

Además de los colores que cambian dependiendo del tipo de pokémon, modificamos el tamaño y estilo de la información de las cards. Para hacerlo, creamos la clase ***info-cards*** y en esta utilizamos las propiedades font-size y font-weight de la siguiente manera:

```css
.info-cards {
    font-size: 15px;
    font-weight: bold;
}
```

Animaciones

Nos pareció interesante agregar animaciones a los títulos, para esto, buscamos animaciones en la página Animate.css (https://animate.style/). Luego, para importarlas al proyecto, debíamos copiar el código que nos dio la página en el ***head*** de ***header.html*** de la siguiente manera:

```html
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>POKEDEX - IP</title>
    {% load bootstrap5 %} {% bootstrap_css %} {% bootstrap_javascript %} {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static 'styles.css' %}">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <!-- Font "Farsan" - Google Fonts -->
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Farsan&display=swap" rel="stylesheet">
    <!-- Animación titulos de CSS (Página Animate.css) -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css"/>
</head>
```

Para que estas animaciones funcionen en el texto que queríamos, debimos copiar dos clases que nos dio la página y pegarlas en el elemento, de la siguiente manera:

```
<h2 class="text-center titulos animate__animated animate__bounce">Bienvenido {{ user.username | upper }}!</h2>   index.html
{% else %}
<h2 class="text-center titulos animate__animated animate__bounce">Bienvenido!</h2>   index.html
<h1 class="text-center titulos animate__animated animate__bounceInLeft">Buscador de Pokemon</h1>   home.html
<h2 class="text-center titulos animate__animated animate__backInLeft">Inicio de sesión</h2>   login.html
```

NOMBRE DE LA ANIMACIÓN

GIF Pikachu

Además, agregamos un GIF de Pikachu saludando en ***index.html***. Primero, lo descargamos de internet, luego lo agregamos a la carpeta ***images*** en ***static***. Modificamos el HTML de la siguiente manera:

```html
<!-- GIF Pikachu -->
<img src="{% static '/images/gif-pikachu.gif' %}" class="gif-pikachu" alt="Imagen de Pikachu saludando">
```

CLASE git-pikachu

Luego, para modificar su tamaño y ajustarlo, creamos una clase ***gif-pikachu***, y en CSS escribimos lo siguiente:

```css
.gif-pikachu {
    width: 400px;
    height: auto;
    display: block;
    margin: 0 auto;
    padding-top: 10px ;
}
```

- Problemas encontrados y funciones que nos hubieran gustado implementar:

Al principio del trabajo tuvimos algunos problemas, especialmente para instalar y actualizar los distintos programas. Luego, tuvimos que desactivar el antivirus ya que no nos permitía conectarnos a la API y obtener la información. Estos problemas fueron resueltos con ayuda de la Inteligencia Artificial. Luego, con el fondo de la página también tuvimos un problema, habíamos usado una imagen de degradado, y cuando se repetía no se veía correctamente. Intentamos hacerlo con los degradados de CSS, pero tampoco pudimos lograr que se visualizara correctamente. Finalmente, para solucionar este problema, utilizamos una imagen más larga, entonces no tuvimos problema con que se corte y se repita.

Luego, también quedaron algunas funciones que nos hubiera gustado implementar, como la de favoritos. Intentamos hacer una parte de esta, y logramos hacer que aparezca la lista de favoritos vacía, pero por cuestiones de tiempo no pudimos terminarla. Hubo cosas que no entendimos, por eso no lo agregamos a Git Hub, pero nos hubiera gustado implementarla.

• Conclusión:

Durante el trabajo aprendimos a utilizar tecnologías que no conocíamos, como Django y Bootstrap. Además, aprendimos a utilizar Git Hub, a subir modificaciones, copiar repositorios y observar los cambios entre las versiones. También utilizamos HTML y CSS, aprendimos a modificar la estructura y diseño de las páginas web.

En conclusión, el trabajo nos pareció interesante, pudimos poner en práctica los temas aprendidos durante la materia (como el if y for), y observar casos de usos reales de estos. En cuanto a la dificultad, nos pareció difícil, pero con la guía de los docentes pudimos entender las distintas funciones y resolver los problemas que se nos fueron presentando. Nos gustó mucho hacer este proyecto, pudimos aprender más cosas relacionadas a lo que estudiamos en la materia, y creemos que esto nos va a servir para futuros proyectos.