PROGRAMACIÓN II

1º AÑO

Clase Nº 9: Desarrollo y Distribución de una Aplicación Flask

Contenido:

En la clase de hoy trabajaremos los siguientes temas:

- Incorporar diseño visual responsivo mediante Bootstrap.
- Implementar autenticación segura con login y contraseñas cifradas.
- Permitir la carga de archivos e imágenes desde formularios web.
- Empaquetar y distribuir la app como ejecutable usando PyInstaller.

1. Presentación

Hasta este punto de la cursada, hemos desarrollado aplicaciones web funcionales que permiten la interacción del usuario a través de formularios y el almacenamiento persistente de datos mediante el uso de bases de datos relacionales con SQLAlchemy. Esto nos ha brindado una base sólida en la creación de sistemas dinámicos, capaces de responder a las acciones del usuario y conservar información entre sesiones.

En la clase de hoy, vamos a dar un paso fundamental hacia el desarrollo profesional de software: aprenderemos a empaquetar nuestras aplicaciones

para que puedan ejecutarse de forma independiente, sin necesidad de que el usuario instale Python ni configure entornos de desarrollo. Es decir, transformaremos nuestros proyectos en productos reales, listos para ser distribuidos y utilizados por cualquier persona en un entorno de producción.

Pero no solo se trata de distribuir una aplicación. También vamos a incorporar buenas prácticas de desarrollo que agregan valor y profesionalismo a nuestros proyectos. Integraremos un sistema de autenticación seguro con cifrado de contraseñas, mejoraremos la experiencia del usuario a través de un diseño visual moderno y adaptable con Bootstrap, y aprenderemos a gestionar la carga de archivos e imágenes desde la web, cuidando aspectos de seguridad y organización.

Con estos nuevos conocimientos, comenzaremos a consolidar una visión más completa del desarrollo de aplicaciones: desde su concepción y funcionalidad, hasta su presentación visual, seguridad, y finalmente, su distribución. Esto nos acerca al tipo de proyectos que enfrentan los profesionales del software en el mundo real, y nos prepara para afrontar desafíos con herramientas y criterios sólidos.

Actividad 1: Mira el siguiente video y reflexiona sobre la distribución de aplicaciones Flask con PyInstaller:

Cómo crear un ejecutable (.exe) con PyInstaller en Python y Flask – Curso Práctico

Objetivo:

Este video ofrece una guía clara y paso a paso sobre cómo convertir una aplicación Flask en un archivo ejecutable utilizando PyInstaller. Muestra desde la instalación de la herramienta, hasta los comandos necesarios para empaquetar una aplicación con plantillas HTML y archivos estáticos.

Instrucciones:

Observa detenidamente el proceso explicado en el video.

Tomá nota de los pasos principales: preparación del entorno, comandos utilizados y configuración necesaria.

Reflexiona y respondé las siguientes preguntas:

- ¿Por qué es importante poder generar un ejecutable para una app web?
- ¿Qué ventajas ofrece esta forma de distribución en entornos donde no se puede instalar Python?
- ¿Cuáles son los posibles desafíos al usar PyInstaller en aplicaciones Flask?

Desarrollo

Diseño visual con Bootstrap

Bootstrap es uno de los frameworks front-end más utilizados en el desarrollo web moderno. Proporciona una amplia colección de componentes predefinidos, estilos y utilidades basados en HTML, CSS y JavaScript que

permiten crear interfaces atractivas, funcionales y adaptables a distintos dispositivos (computadoras, tablets, celulares) sin necesidad de escribir código CSS desde cero.

Gracias a Bootstrap, es posible construir rápidamente páginas web con una apariencia profesional, incorporando elementos como formularios bien estructurados, botones estilizados, menús de navegación, tarjetas informativas, barras de progreso, alertas y más. Además, su sistema de rejilla (grid) facilita la organización del contenido en columnas y filas de manera responsive, es decir, que se ajusta automáticamente al tamaño de pantalla del usuario.

Comparativa: Bootstrap vs CSS Puro

Características	Bootstrap	CSS puro
Diseño responsive automático	Sí	No (hay que hacerlo manualmente)
Componentes visuales predefinidos	Sí (botones, formularios, menús, etc.)	No (todo debe diseñarse desde cero)
Sistema de grillas (grid)	Sí, con clases de filas y columnas	No (requiere escribir media queries y flexbox)
Curva de aprendizaje	Baja	Media/Alta
Velocidad de desarrollo	Alta	Baja
Requiere escribir CSS manual	No	Sí
Facilidad de integración con Flask	Muy alta	Media

En proyectos con Flask, la integración de Bootstrap es muy sencilla. Basta con incluir el enlace al CDN (Content Delivery Network) en el archivo base.html, que actúa como plantilla principal de nuestra aplicación. Desde allí, se heredan los estilos a todas las vistas, permitiendo mantener una estética uniforme en todo el sitio sin repetir código.

```
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
```

Esta separación entre la lógica de la aplicación (en Python/Flask) y el diseño visual (con Bootstrap) no solo mejora la organización del proyecto, sino que también permite escalar la interfaz fácilmente a medida que la aplicación crece.

Ejemplo de plantilla base.html con Bootstrap:

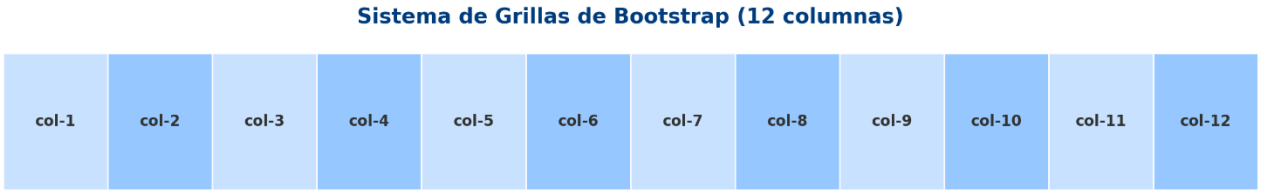
```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Mi App Flask</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Mi App</a>
        </div>
    </nav>
    <div class="container mt-4">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

¿Qué es el sistema de grillas de Bootstrap?

El sistema de grillas (grid system) de Bootstrap divide cada fila del diseño en 12 columnas iguales. Este sistema es clave para construir interfaces responsivas, es decir, que se adapten automáticamente a diferentes tamaños de pantalla (como computadoras, tablets o celulares).

Cada elemento de contenido puede ocupar una o más columnas, y se organizan dentro de "filas" (.row) que a su vez están dentro de un contenedor (.container o .container-fluid).

La grilla utiliza clases como col-4, col-6, col-md-8, etc., donde el número indica cuántas columnas ocupa el elemento (de un total de 12), y el prefijo (col-, col-sm-, col-md-, etc.) define el tamaño de pantalla en el que esa distribución se aplica.



📐 Ejemplos básicos en HTML

🧩 *Ejemplo 1: Tres columnas iguales*

```html
<div class="container">
  <div class="row">
    <div class="col-4 bg-primary text-white">Columna 1</div>
    <div class="col-4 bg-secondary text-white">Columna 2</div>
    <div class="col-4 bg-success text-white">Columna 3</div>
  </div>
</div>
```

Cada columna ocupa 4/12 del ancho total. El total suma 12, por lo tanto, se distribuyen perfectamente en una fila.

🧩 *Ejemplo 2: Dos columnas desiguales*

```
<div class="container">
  <div class="row">
    <div class="col-8 bg-info text-white">Columna más ancha (8/12)</div>
    <div class="col-4 bg-dark text-white">Columna más angosta (4/12)</div>
  </div>
</div>
```

Una columna ocupa dos tercios del ancho, la otra un tercio.

📱 *Ejemplo 3: Grilla responsiva*

```
<div class="container">
  <div class="row">
    <div class="col-12 col-md-6 bg-warning">Se adapta según el tamaño</div>
    <div class="col-12 col-md-6 bg-light">Se adapta también</div>
  </div>
</div>
```

En pantallas pequeñas (celulares), cada columna ocupa el 100% del ancho (col-12). En pantallas medianas o grandes, se dividen en dos columnas de 50% cada una (col-md-6).

✅ Beneficios clave del sistema de grillas

- Permite **diseños adaptativos** sin escribir media queries manuales.
- Organiza visualmente la información de forma profesional.
- Aumenta la productividad y legibilidad del código HTML.

- Es compatible con todos los navegadores modernos.

Actividad 2: Aplicando el Sistema de Grillas

Objetivo: Comprender y aplicar el sistema de grillas de Bootstrap para estructurar contenidos de manera responsiva.

Copiá el siguiente código HTML en tu editor (puede ser VS Code, CodePen o cualquier entorno web):

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejercicio Grid Bootstrap</title>
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <div class="container mt-4">
    <div class="row">
      <!-- Completa aquí con columnas que sumen 12 -->
    </div>
  </div>
</body>
</html>
```

Dentro del , agregá **3 columnas** que ocupen partes iguales del ancho disponible (en pantallas grandes) y que en pantallas pequeñas ocupen todo el ancho una debajo de la otra.

Luego, modifica el diseño para que:

- La **primera columna** ocupe el 6/12 del ancho en pantallas medianas (md) o más.
- Las **otras dos columnas** ocupen 3/12 cada una.
- Aplicá clases de color (bg-primary, bg-success, bg-warning) para visualizar los bloques.

Login con cifrado de contraseñas

En el desarrollo de aplicaciones profesionales, **nunca se deben almacenar ni verificar contraseñas en texto plano**, es decir, tal como el usuario las escribió. Hacerlo representa un riesgo crítico: si la base de datos se ve comprometida, todas las contraseñas quedarían expuestas directamente.

Para evitar esto, se utilizan **funciones de hash**, que transforman la contraseña original en una cadena irreconocible y no reversible. Es decir, no se puede volver a obtener la contraseña original desde el hash. De este modo, lo que se guarda en la base de datos no es la contraseña real, sino su versión cifrada.

Flask facilita esta tarea gracias al módulo werkzeug.security, que proporciona funciones como:

- generate_password_hash(): para generar un hash seguro a partir de una contraseña ingresada.
- check_password_hash(): para verificar si una contraseña proporcionada coincide con su versión cifrada.

```python
from werkzeug.security import generate_password_hash

# Contraseña original
contraseña = 'miclave123'

# Generar hash
hash_generado = generate_password_hash(contraseña)

print("Contraseña original:", contraseña)
print("Hash generado:", hash_generado)
```

¿Qué hace esto?

- Cifra la contraseña 'miclave123'.
- Muestra el resultado en consola.
- Este hash se guarda luego en la base de datos en lugar de la contraseña real.

Esta práctica de seguridad es **esencial**: protege los datos sensibles de los usuarios frente a posibles filtraciones, errores de programación o ataques externos. Implementar hash correctamente es un paso fundamental para que nuestras aplicaciones estén preparadas para el mundo real, donde la seguridad de la información es tan importante como la funcionalidad.

💻 Ejemplo: Login seguro con Flask y hash de contraseñas

```python
from flask import Flask, render_template, request, redirect, url_for, session
from werkzeug.security import generate_password_hash, check_password_hash

app = Flask(__name__)
app.secret_key = 'clave_secreta_para_la_sesion'

# Simulación de una base de datos con un solo usuario
# En una aplicación real, esto se consultaría desde una base de datos
usuarios = {
    'admin': generate_password_hash('1234')  # La contraseña '1234' se guarda como hash
}

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        usuario = request.form['usuario']
        password = request.form['password']

        # Verificamos si el usuario existe y la contraseña ingresada coincide con el hash almacenado
        if usuario in usuarios and check_password_hash(usuarios[usuario], password):
            session['usuario'] = usuario  # Inicia sesión
            return redirect(url_for('dashboard'))
        else:
            return "Credenciales incorrectas. Intenta nuevamente."

    return render_template('login.html')

@app.route('/dashboard')
def dashboard():
    if 'usuario' not in session:
        return redirect(url_for('login'))  # Si no está logueado, lo redirige
    return f"Bienvenido, {session['usuario']}"

@app.route('/logout')
def logout():
    session.pop('usuario', None)  # Cierra la sesión
    return redirect(url_for('login'))

if __name__ == '__main__':
    app.run(debug=True)
```

Puntos clave del código:

- **generate_password_hash('1234')**: convierte la contraseña en un hash cifrado seguro.
- **check_password_hash(hash, '1234')**: compara el hash almacenado con la contraseña ingresada.
- **session**: permite mantener la sesión iniciada una vez que el usuario se autentica correctamente.

- Se utiliza una **clave secreta** (app.secret_key) para proteger los datos de sesión.
- Este ejemplo puede escalarse fácilmente para funcionar con múltiples usuarios desde una base de datos real.

Actividad 3: Sistema básico de verificación de contraseñas

Objetivo: Implementar con tus propias líneas de código un sistema simple de registro y verificación de contraseñas usando funciones de hash.

Escribí un script en Python que cumpla con los siguientes pasos:

- Pedile al usuario que ingrese una contraseña para registrarse.
- Generá un hash a partir de esa contraseña utilizando generate_password_hash().
- Mostrá el hash generado por pantalla (como si se estuviera guardando en una base de datos).
- Luego pedile al usuario que ingrese nuevamente una contraseña, simulando un login.
- Compará esa contraseña con el hash guardado usando check_password_hash().
- Mostrará un mensaje que indique si la verificación fue exitosa o no.

Carga de Archivos e Imágenes en Flask

Permitir la carga de archivos desde la web es una funcionalidad muy útil y común en aplicaciones modernas. Ya sea para subir imágenes de perfil, documentos, formularios escaneados u otro tipo de contenido, ofrecer esta

opción mejora la interacción y utilidad del sistema.

En Flask, este proceso se gestiona fácilmente a través de formularios HTML y la API request.files, que permite acceder a los archivos enviados por el usuario. El archivo se puede recibir, validar y guardar en una ubicación específica dentro del servidor.

Para garantizar un desarrollo seguro y ordenado, es importante tener en cuenta los siguientes aspectos:

- ✅ **Validación del tipo de archivo**: Nunca se debe confiar en lo que el usuario sube. Es necesario verificar la extensión del archivo (por ejemplo .jpg, .png, .pdf) para evitar que se suban archivos maliciosos.
- 📁 **Destino seguro**: Se recomienda almacenar los archivos en una carpeta interna como static/uploads/, fuera de las rutas críticas del servidor.
- 🧼 **Nombre del archivo**: Puede ser útil renombrar los archivos para evitar colisiones o sobreescrituras si dos usuarios suben archivos con el mismo nombre.

```html
<!-- subir.html -->
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Subir Archivo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body class="container mt-5">
    <h2>Subir imagen</h2>
    <form action="/subir" method="POST" enctype="multipart/form-data">
        <div class="mb-3">
            <input type="file" name="archivo" accept=".jpg, .jpeg, .png" class="form-control" required>
        </div>
        <button type="submit" class="btn btn-primary">Subir</button>
    </form>
</body>
</html>
```

Flask proporciona una forma sencilla de gestionar esto con pocas líneas de código, y combinándolo con HTML y Bootstrap se pueden construir formularios elegantes y funcionales.

```python
import os
from flask import Flask, request, render_template, redirect, url_for, flash
from werkzeug.utils import secure_filename

app = Flask(__name__)
app.secret_key = 'clave_segura'
app.config['UPLOAD_FOLDER'] = 'static/uploads'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}

# Verifica si la extensión del archivo es válida
def extension_valida(nombre):
    return '.' in nombre and nombre.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/subir', methods=['GET', 'POST'])
def subir_archivo():
    if request.method == 'POST':
        if 'archivo' not in request.files:
            flash('No se seleccionó ningún archivo')
            return redirect(request.url)

        archivo = request.files['archivo']
        if archivo.filename == '':
            flash('Nombre de archivo vacío')
            return redirect(request.url)

        if archivo and extension_valida(archivo.filename):
            nombre_seguro = secure_filename(archivo.filename)
            ruta = os.path.join(app.config['UPLOAD_FOLDER'], nombre_seguro)
            archivo.save(ruta)
            flash('Archivo subido exitosamente')
            return redirect(url_for('subir_archivo'))

        flash('Extensión no permitida')
        return redirect(request.url)

    return render_template('subir.html')

if __name__ == '__main__':
    app.run(debug=True)
```

✅ **Buenas prácticas aplicadas:**

- secure_filename() evita caracteres peligrosos en el nombre del archivo.
- Se valida la extensión para aceptar solo imágenes permitidas.
- El archivo se guarda en static/uploads/ para luego poder mostrarse en la web.

- Se usa flash() para mostrar mensajes de estado amigables al usuario.

Actividad 4: App en Flask para subir imágenes con ayuda de ChatGPT

Instrucciones:

1. Abrí ChatGPT y escribí el siguiente prompt:

   ✍️ *"Creá un ejemplo básico en Flask que permita subir una imagen y mostrarla luego en la misma página."*

2. Copiá el código que te genera y pegalo en un archivo llamado app.py.
3. Creá un archivo HTML (si el código no lo incluye) siguiendo el modelo de formulario que viste en clase o el que te sugiera ChatGPT.
4. Corré tu aplicación en el navegador y probá subir una imagen desde tu computadora.

PyInstaller: Empaquetado y distribución de una app Flask como ejecutable

Cuando desarrollamos una aplicación en Python, el entorno de ejecución requiere que el usuario tenga instalado Python y todas las dependencias necesarias. Sin embargo, gracias a **PyInstaller**, es posible **convertir tu aplicación Flask en un archivo ejecutable (.exe)** que funciona de forma autónoma en otros equipos, sin necesidad de instalar nada adicional.

Esto es especialmente útil cuando queremos distribuir nuestra app a usuarios finales no técnicos, o instalarla en computadoras donde no hay un entorno de

desarrollo.

⚙️ ¿Cómo funciona PyInstaller?

PyInstaller se encarga de:

- Leer el archivo principal de tu proyecto (por ejemplo, app.py) y detectar todas sus dependencias (librerías, módulos, recursos externos).
- Empaquetar todo en un **único ejecutable** o en una carpeta lista para ejecutar.
- Incluir automáticamente archivos esenciales como imágenes, hojas de estilo, plantillas HTML, etc., si se lo indicamos explícitamente.
- Generar versiones ejecutables compatibles con el sistema operativo (Windows, Linux, macOS).

🧩 Consideraciones especiales para Flask

Las aplicaciones Flask suelen depender de carpetas específicas, como:

- templates/ → donde se almacenan las plantillas HTML (renderizadas con Jinja2).
- static/ → donde se alojan archivos estáticos como CSS, imágenes o scripts JS.

💡 **Es fundamental incluir estas carpetas** al momento de generar el ejecutable, ya que Flask las necesita para funcionar correctamente.

Ejecutable básico:

```
pyinstaller --onefile app.py
```

Ejecutable incluyendo plantillas y archivos estáticos (en Windows):

```
pyinstaller --onefile --add-data "templates;templates" --add-data "static;static" app.py
```

Ejecutable sin consola (ideal para apps con interfaz web o visual):

```
pyinstaller --onefile --noconsole --add-data "templates;templates" --add-data "static;static" app.py
```

PyInstaller creará una carpeta dist/ que contendrá tu ejecutable. Podés copiarlo a cualquier otra computadora con Windows y ejecutarlo directamente, sin necesidad de instalar Python ni configurar entornos virtuales.

Foro de Instancia Oral – Distribución de una app Flask

Consigna: Grabá un video de aproximadamente 1 minuto en el que respondas a estas preguntas:

- 📌 ¿Cuál creés que es la principal **ventaja** de distribuir tu app Flask como ejecutable con PyInstaller?
- ⚠️ ¿Qué **dificultades técnicas** podrías enfrentar al hacerlo?
- 🛠️ ¿Qué **archivos o carpetas** deberías tener cuidado de incluir?
- 📦 ¿Te gustaría que tu app funcione como un programa "instalable"? ¿Por qué?

📋 Participación mínima esperada

- Una **publicación inicial** subiendo tu video con tu reflexión.
- Al menos **un comentario** en el video de un/a compañero/a, comparando o ampliando ideas.

👉 El objetivo no es que muestres código, sino que expliques con tus palabras como si lo contaras a alguien que no sabe de programación.

Criterios de Evaluación

1. **Claridad y Coherencia en las Respuestas:**

- Las respuestas deben ser claras, bien estructuradas y fáciles de seguir. Se evaluará la capacidad del estudiante para expresar sus ideas de manera coherente y lógica.

2. **Profundidad del Análisis:**
   - Se valorará la profundidad del análisis crítico en relación con el artículo. Los estudiantes deben demostrar que han comprendido los puntos clave y que pueden relacionarlos con sus propias experiencias y conocimientos.
3. **Argumentación y Justificación:**
   - Los estudiantes deben respaldar sus opiniones con argumentos sólidos y ejemplos concretos. Se evaluará la capacidad para justificar sus afirmaciones y reflexiones de manera efectiva.
4. **Participación y Colaboración en el Foro:**

Se tendrá en cuenta la participación activa en el foro, incluyendo la capacidad de responder a las intervenciones de otros compañeros de manera constructiva y respetuosa.

ACTIVIDAD INTEGRADORA DE CIERRE

En esta actividad vas a aplicar todos los conocimientos trabajados en la clase:

Desarrollar una aplicación web profesional con Flask que integre diseño visual, autenticación segura, carga de archivos y empaquetado como ejecutable, tal como lo harías en un proyecto real para entregar a un cliente.

🛠️ **Consigna**: Desarrolla una aplicación Flask que cumpla con los siguientes

requisitos funcionales:

1. *Interfaz visual responsiva*

Utiliza Bootstrap para diseñar una interfaz limpia y adaptable.

Incluí una navegación básica (navbar) y un contenedor principal con márgenes adecuados.

2. *Sistema de login seguro*

Implementa un formulario de login.

Las contraseñas deben estar cifradas con generate_password_hash() y verificadas con check_password_hash().

El sistema debe permitir el acceso solo a usuarios autorizados.

3. *Distribución como ejecutable*

Utiliza PyInstaller para empaquetar tu aplicación como un archivo .exe.

El ejecutable debe incluir correctamente las carpetas templates/ y static/.

Cierre:

Hoy aprendimos a construir y empaquetar una aplicación Flask completa, pensada no solo para funcionar localmente, sino para ser compartida, distribuida y utilizada por otros usuarios de forma autónoma.

A lo largo de la clase trabajamos varios ejes fundamentales:

Por un lado, incorporamos el uso de Bootstrap, un framework de diseño que nos permitió transformar nuestras interfaces en experiencias visuales modernas, limpias y adaptables a distintos dispositivos. Aprendimos que una buena presentación no solo embellece, sino que facilita la usabilidad y mejora la percepción general de nuestra app.

Por otro lado, profundizamos en aspectos de seguridad, desarrollando un sistema de login con contraseñas cifradas utilizando werkzeug.security. Esta práctica profesional nos enseñó que proteger los datos del usuario es una responsabilidad técnica y ética, y que implementar medidas preventivas desde el principio es clave para cualquier proyecto serio.

También incorporamos la carga de archivos en nuestra aplicación, aprendiendo cómo recibir imágenes a través de formularios, validarlas y almacenarlas correctamente en el servidor. Esto abre las puertas a funcionalidades muy comunes en el mundo real, como perfiles de usuario, galerías, documentación digitalizada, etc.

Finalmente, llegamos al gran cierre: aprendimos a utilizar PyInstaller, una herramienta que nos permite convertir nuestro proyecto Flask en un archivo ejecutable (.exe), listo para instalar y usar en cualquier computadora, incluso si no tiene Python instalado. Comprendimos la importancia de empaquetar correctamente nuestras aplicaciones incluyendo las carpetas templates/ y static/, y exploramos distintas configuraciones de compilación para distribuir nuestros desarrollos como verdaderos productos de software.

Esta clase no solo reunió conocimientos técnicos, sino que también representó una mirada más completa y profesional del proceso de desarrollo: desde la interfaz hasta la seguridad, desde la interacción del usuario hasta la entrega final del producto.

En la próxima clase, haremos un repaso de todos los contenidos de la cursada, preparándonos para un parcial que nos permita poner a prueba lo aprendido con una mirada global.

Bibliografía obligatoria

- Ramírez Jiménez, Ó. (2021). *Python a fondo: Domina el lenguaje, la programación orientada a objetos y los aspectos avanzados*. Marcombo.

- Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

Recursos adicionales recomendados

- *Flask Documentation. (n.d.).* Flask User's Guide. *Flask – Pallets Project.* https://flask.palletsprojects.com/en/latest/
- *Bootstrap. (n.d.).* Introduction – Bootstrap v5.3. *GetBootstrap.* https://getbootstrap.com/docs/5.3/getting-started/introduction/
- *PyInstaller Development Team. (n.d.).* PyInstaller Documentation. https://pyinstaller.org/en/stable/
- *WerkZeug Developers. (n.d.).* Security utilities – werkzeug.security. *Pallets Project.* https://werkzeug.palletsprojects.com/en/2.3.x/utils/#module-werkzeug.security
- *EllibrodePython.com. (n.d.).* Tutorial: Cómo usar Flask y Bootstrap juntos. https://ellibrodepython.com/flask-bootstrap
- *YouTube – Código Facilito. (2022).* Cómo convertir tu app Python en un ejecutable con PyInstaller *[Video].* https://www.youtube.com/watch?v=EGpLaiXEj9I