Guía Detallada para el Desarrollo de una Aplicación Web con Flask, HTML, CSS, JavaScript y MSSQL

Introducción

Este documento es una guía paso a paso que abarca el ciclo completo de desarrollo de una aplicación web funcional. El objetivo es proporcionar un marco de trabajo claro, desde la concepción inicial de la idea hasta su puesta en marcha. Las tecnologías centrales para este proyecto son Python con el framework Flask para el desarrollo del backend, HTML, CSS y JavaScript para la interfaz de usuario, y Microsoft SQL Server (MSSQL) como sistema gestor de base de datos. Se priorizará el uso de herramientas y tecnologías complementarias gratuitas. Esta guía se enfoca en las fases de análisis, diseño, una descripción conceptual de la implementación y el despliegue, omitiendo la generación de casos de prueba detallados.

Fase 1: Análisis y Planificación

Esta etapa es fundamental para definir el alcance, los objetivos y la viabilidad del proyecto. Una planificación adecuada minimiza riesgos y sienta las bases para un desarrollo exitoso.

1.1. Definición de Requisitos Funcionales

Los requisitos funcionales describen lo que el sistema debe hacer. Basándonos en la información proporcionada (incluyendo los documentos de referencia), los requisitos clave son:

- **Gestión de Usuarios y Autenticación Segura:**
  - **Inicio de Sesión:** Permitir a los usuarios acceder al sistema mediante credenciales (nombre de usuario/email y contraseña).
  - **Autenticación de Dos Factores (2FA):** Implementar un segundo factor de autenticación donde, tras el login exitoso, se envíe un código de verificación único al correo electrónico registrado del usuario. Este código será necesario para completar el acceso.
  - **Gestión de Roles:**
    - admin: Acceso completo a todas las funcionalidades del sistema, incluyendo la gestión de usuarios y la configuración general.
    - director: Permisos idénticos al rol admin.
    - consulta: Acceso restringido únicamente a la visualización de reportes.
  - **Cierre de Sesión (Logout):** Permitir a los usuarios finalizar su sesión de forma segura.

- **Gestión de Solicitudes de Estudiantes:**
  - **Formulario de Admisión:** Un formulario web para registrar la información de nuevos estudiantes. Los campos requeridos son:
    - Nombre completo.
    - Apellidos.
    - País de origen (seleccionable de una lista predefinida: Guatemala, El Salvador, Honduras, Panamá, República Dominicana, Paraguay, Colombia, Costa Rica, Nicaragua, México).
    - Ciudad (un campo desplegable que se actualizará dinámicamente según el país seleccionado, mostrando las ciudades principales).
    - Otras señas de la dirección (campo de texto libre).
    - Último grado académico alcanzado.
    - DNI (Documento Nacional de Identidad) o identificador único.
    - Fecha de nacimiento.
    - Correo electrónico.
    - Número de teléfono.
    - Año de la solicitud.
  - **Validación de Datos:** Asegurar que los datos ingresados en el formulario cumplan con los formatos y restricciones esperados (ej. campos obligatorios, formato de correo, formato de fecha).
- **Carga Masiva de Datos Históricos:**
  - **Importación desde Excel:** Funcionalidad que permita a los usuarios con roles admin o director cargar un archivo en formato Excel (.xlsx) conteniendo datos de estudiantes históricos.
  - **Procesamiento y Almacenamiento:** El sistema deberá leer el archivo Excel, validar los datos de cada fila y almacenarlos en la base de datos. Se debe evitar la duplicidad de registros, utilizando el DNI como identificador único para la comparación.
- **Generación y Visualización de Reportes:**
  - **Reporte General de Estudiantes:** Una vista que muestre un listado de todos los estudiantes registrados en el sistema, con su información relevante.
  - **Control de Acceso a Reportes:** El acceso a la funcionalidad de reportes estará disponible para todos los roles, pero las acciones dentro de los reportes (si las hubiera, como exportar o modificar) podrían estar restringidas según el rol.
- **Seguridad Integral (funcionalidades transversales):**
  - **Protección contra Vulnerabilidades Comunes:** Adherencia a las recomendaciones de OWASP Top 10 para prevenir ataques comunes (ej. Inyección SQL, XSS, CSRF).

    - **Sanitización de Entradas:** Todas las entradas de datos provenientes del usuario (formularios, archivos) deben ser validadas y sanitizadas antes de ser procesadas o almacenadas.
    - **Protección CSRF (Cross-Site Request Forgery):** Implementar tokens CSRF en todos los formularios que realicen modificaciones de estado en el servidor.

1.2. Definición de Requisitos No Funcionales

Estos requisitos definen cómo debe ser el sistema, sus atributos de calidad.

- **Tecnologías Específicas (Stack Definido):**
    - Backend: Python con el framework Flask.
    - Frontend: HTML5, CSS3, JavaScript (ECMAScript 6+).
    - Base de Datos: Microsoft SQL Server (se recomienda utilizar la edición Express para entornos de desarrollo y despliegues gratuitos/económicos).
    - Herramientas Complementarias: Todas las librerías o servicios adicionales deben ser de uso gratuito.
- **Seguridad:**
    - **Prácticas de Desarrollo Seguro:** Integrar la seguridad desde el diseño (Security by Design) y seguir las guías de OWASP.
    - **Almacenamiento Seguro de Credenciales:** Las contraseñas de los usuarios deben almacenarse hasheadas utilizando algoritmos robustos (ej. bcrypt).
    - **Comunicaciones Seguras:** Utilizar HTTPS en el entorno de producción para cifrar la comunicación entre el cliente y el servidor.
- **Usabilidad:**
    - **Interfaz Intuitiva:** El diseño de la interfaz de usuario debe ser claro, fácil de entender y navegar.
    - **Aspecto Visual Profesional:** Se utilizará un framework CSS como Bootstrap para asegurar un diseño responsivo y visualmente agradable.
    - **Retroalimentación al Usuario:** El sistema debe proporcionar mensajes claros al usuario sobre el resultado de sus acciones (ej. éxito o error al guardar un formulario).
- **Rendimiento:**
    - **Tiempos de Respuesta:** La aplicación debe responder a las interacciones del usuario en tiempos aceptables.
    - **Manejo de Carga de Archivos:** Establecer límites razonables para el tamaño de los archivos Excel que se pueden subir (ej. 2-5MB) para evitar sobrecarga del servidor.
- **Despliegue:**
    - **Portabilidad:** La aplicación debe ser desplegable en un servidor web estándar. Se explorarán opciones de plataformas de despliegue gratuitas o de

bajo costo.
- **Mantenibilidad:**
  - **Código Organizado:** El código fuente debe estar bien estructurado, ser modular y seguir convenciones de estilo para facilitar su comprensión y futuras modificaciones.
  - **Documentación Interna:** Comentarios adecuados en el código para explicar la lógica compleja.

1.3. Elección del Stack Tecnológico (Confirmado)

- **Lenguaje de Programación Backend:** Python (versión 3.8 o superior).
- **Framework Backend:** Flask.
- **Base de Datos:** Microsoft SQL Server (Express Edition recomendada para gratuidad).
- **ORM (Object-Relational Mapper) / Conector DB:** SQLAlchemy junto con pyodbc como driver para la conexión con MSSQL.
- **Frontend:** HTML5, CSS3, JavaScript (ES6+).
- **Framework CSS:** Bootstrap 5 (o la versión estable más reciente).
- **Librerías Python Adicionales Clave:**
  - Flask-SQLAlchemy: Para la integración de SQLAlchemy con Flask.
  - Flask-Login: Para la gestión de sesiones de usuario y autenticación.
  - Flask-WTF: Para la creación y validación de formularios, y protección CSRF.
  - Flask-Mail: Para el envío de correos electrónicos (necesario para la funcionalidad 2FA).
  - bcrypt: Para el hashing seguro de contraseñas.
  - bleach: Para la sanitización de HTML y prevención de XSS.
  - Flask-Talisman: Para configurar cabeceras HTTP de seguridad.
  - pandas y openpyxl: Para la lectura y procesamiento de archivos Excel.
  - python-dotenv: Para la gestión de variables de entorno y configuración.
- **Servidor WSGI (para producción):** Gunicorn (para entornos Linux/macOS) o Waitress (para entornos Windows).

1.4. Planificación del Proyecto (Hitos Generales)

1. **Configuración del Entorno de Desarrollo:** Preparación de las herramientas y software necesarios.
2. **Diseño y Creación de la Base de Datos:** Definición del esquema en MSSQL y creación de las tablas.
3. **Desarrollo del Módulo de Autenticación y Usuarios:** Implementación de login, logout, 2FA y gestión de roles.
4. **Desarrollo del Módulo de Gestión de Estudiantes:** Creación del formulario de

admisión y lógica de guardado.
5. **Desarrollo de la Funcionalidad de Carga Masiva:** Implementación de la subida y procesamiento de archivos Excel.
6. **Desarrollo del Módulo de Reportes:** Creación de la vista para mostrar los estudiantes.
7. **Integración de Componentes y Refinamiento:** Asegurar que todos los módulos funcionen conjuntamente.
8. **Preparación para el Despliegue:** Configuración de la aplicación para un entorno de producción.
9. **Despliegue:** Publicación de la aplicación en un servidor web.

Fase 2: Diseño

En esta fase se traduce el análisis en un plan técnico detallado para la construcción de la aplicación.

2.1. Diseño de la Base de Datos (Microsoft SQL Server)

Se definirá la estructura de la base de datos, incluyendo tablas, campos, tipos de datos, relaciones y restricciones.

- **Modelo Entidad-Relación (Conceptual):**
  - **Entidad Usuarios:** Almacenará la información de los usuarios del sistema.
    - Atributos: id (PK, autoincremental), username (email, único, no nulo), password_hash (no nulo), role (no nulo, ej: 'admin', 'director', 'consulta'), two_factor_code (para el código 2FA temporal), two_factor_expiry (timestamp para la expiración del código 2FA).
  - **Entidad Estudiantes:** Almacenará la información de los estudiantes admitidos o históricos.
    - Atributos: id (PK, autoincremental), nombre (no nulo), apellidos (no nulo), pais (no nulo), ciudad (no nulo), senas_direccion, grado_academico (no nulo), dni (único, no nulo), fecha_nacimiento (no nulo), correo (no nulo), telefono, anio_solicitud (no nulo).
- **Relaciones:**
  - No hay relaciones directas explícitas entre Usuarios y Estudiantes en este modelo básico, ya que la auditoría de quién creó qué estudiante se manejaría a nivel de aplicación o mediante logs, a menos que se decida añadir una FK creado_por_usuario_id en la tabla Estudiantes.
- **Consideraciones Adicionales:**
  - Se definirán índices en campos frecuentemente usados en búsquedas (ej. dni en Estudiantes, username en Users) para optimizar el rendimiento.

    - Se utilizarán tipos de datos apropiados en MSSQL para cada campo (ej. NVARCHAR para cadenas, DATE para fechas, INT para números enteros).

2.2. Diseño de la Arquitectura de la Aplicación (Flask)

Se definirá la estructura del proyecto Flask, promoviendo la modularidad y la organización del código.

- **Estructura de Directorios Sugerida:**
    - nombre_proyecto/
        - app/ (Directorio principal de la aplicación Flask)
            - __init__.py (Factory de la aplicación, inicialización de extensiones)
            - auth/ (Módulo para autenticación y gestión de usuarios)
                - routes.py (Rutas para login, logout, 2FA)
                - forms.py (Formularios WTForms para autenticación)
            - main/ (Módulo para las funcionalidades principales)
                - routes.py (Rutas para el formulario de estudiantes, carga de Excel, reportes)
                - forms.py (Formularios WTForms para estudiantes)
                - utils.py (Funciones de utilidad, ej. procesamiento de Excel)
            - models.py (Definiciones de los modelos de SQLAlchemy: User, Estudiante)
            - config.py (Clases de configuración para diferentes entornos)
            - decorators.py (Decoradores personalizados, ej. para verificar roles)
            - static/ (Archivos estáticos)
                - css/
                - js/
                - img/
            - templates/ (Plantillas HTML de Jinja2)
                - auth/ (Plantillas para login, 2FA)
                - main/ (Plantillas para formulario, reporte)
                - base.html (Plantilla base común)
                - Páginas de error (ej. 403.html, 404.html, 500.html)
        - .env (Archivo para variables de entorno, no versionado en Git)
        - .flaskenv (Variables de entorno específicas para el comando flask)
        - requirements.txt (Listado de dependencias de Python)
        - run.py (Script para ejecutar la aplicación en desarrollo, opcional)
        - .gitignore (Archivo para especificar qué archivos y directorios ignorar en Git)
- **Componentes Clave:**
    - **Blueprints de Flask:** Se utilizarán para organizar las rutas en módulos (auth,

main).
    - **Factory Pattern (create_app):** Para la creación e inicialización de la instancia de la aplicación Flask, lo que facilita la configuración y las pruebas.
    - **Extensiones de Flask:** Flask-SQLAlchemy, Flask-Login, Flask-WTF, Flask-Mail, Flask-Talisman, Flask-Migrate (opcional, pero recomendado para gestionar cambios en el esquema de la base de datos).

2.3. Diseño de la Interfaz de Usuario (UI) y Experiencia de Usuario (UX)

Se definirán los bocetos o wireframes de las principales pantallas y el flujo de navegación.

- **Páginas Principales:**
    - **Página de Inicio de Sesión (login.html):** Formulario para ingresar email y contraseña. Enlace a "Olvidé mi contraseña" (si se implementa).
    - **Página de Verificación 2FA (two_factor.html):** Formulario para ingresar el código de 6 dígitos recibido por correo.
    - **Panel Principal / Formulario de Admisión (index.html o form_estudiante.html):**
        - Visible para roles admin y director.
        - Contendrá el formulario de admisión de estudiantes.
        - Sección para la carga del archivo Excel.
        - Navegación clara a otras secciones (ej. Reporte).
    - **Página de Reporte de Estudiantes (reporte.html):**
        - Accesible para todos los roles autenticados.
        - Mostrará una tabla con la lista de estudiantes.
        - Podría incluir filtros básicos (ej. por año, por país).
    - **Plantilla Base (base.html):** Incluirá la estructura común (navegación principal, pie de página, inclusión de Bootstrap CSS y JS).
    - **Páginas de Error:** Diseños simples pero informativos para errores 403 (Acceso Denegado), 404 (No Encontrado), 500 (Error Interno del Servidor).
- **Flujo de Navegación:**
    - Usuario no autenticado -> Página de Login.
    - Login exitoso -> Página de Verificación 2FA.
    - Verificación 2FA exitosa -> Panel Principal (según rol).
        - Admin/Director: Acceso a Formulario de Estudiantes, Carga Excel, Reporte, Logout.
        - Consulta: Acceso a Reporte, Logout.
- **Consideraciones de Diseño:**
    - **Responsividad:** La interfaz debe adaptarse a diferentes tamaños de pantalla (escritorio, tableta, móvil) utilizando Bootstrap.

    - **Accesibilidad:** Seguir pautas básicas de accesibilidad web (WCAG) en la medida de lo posible (ej. contraste de colores, etiquetas para campos de formulario).
    - **Consistencia Visual:** Mantener un estilo coherente en toda la aplicación.

Fase 3: Implementación (Descripción Conceptual)

Esta sección describe cómo se llevarían a cabo las tareas de codificación y construcción de la aplicación, sin proporcionar el código fuente detallado.

3.1. Configuración del Entorno de Desarrollo

- **Instalación de Software:**
    - Instalar Python (versión 3.8+).
    - Instalar Microsoft SQL Server Express Edition y SQL Server Management Studio (SSMS).
    - Configurar un editor de código o IDE (ej. VS Code, PyCharm).
    - Instalar Git para el control de versiones.
- **Creación del Proyecto:**
    - Crear el directorio raíz del proyecto.
    - Inicializar un entorno virtual de Python (venv).
    - Activar el entorno virtual.
    - Instalar las dependencias listadas en requirements.txt usando pip.
    - Configurar los archivos .env y .flaskenv con las variables de entorno necesarias (claves secretas, cadenas de conexión a la base de datos, configuración de email).

3.2. Desarrollo del Backend (Flask y MSSQL)

- **Conexión y Modelos de Base de Datos:**
    - Configurar Flask-SQLAlchemy para conectarse a la instancia de MSSQL utilizando pyodbc.
    - Definir los modelos (User, Estudiante) en app/models.py como clases que heredan de db.Model de SQLAlchemy, mapeando los campos de las tablas diseñadas.
    - Crear las tablas en la base de datos MSSQL (ya sea manualmente con el script SQL generado en la fase de diseño, o utilizando db.create_all() de Flask-SQLAlchemy para el desarrollo inicial, o con Flask-Migrate para un manejo más robusto de las migraciones).
- **Módulo de Autenticación y Usuarios (app/auth):**
    - **Formularios:** Crear LoginForm y TwoFactorForm usando Flask-WTF en app/auth/forms.py.

    - **Rutas:**
        - Implementar la ruta /login (GET, POST):
            - GET: Mostrar el formulario de login.
            - POST: Validar credenciales contra la base de datos (comparando el hash de la contraseña con bcrypt). Si son válidas, generar un código 2FA, almacenarlo temporalmente en el modelo User junto con su tiempo de expiración, y enviarlo al correo del usuario usando Flask-Mail. Redirigir al usuario a la página de verificación 2FA.
        - Implementar la ruta /two_factor (GET, POST):
            - GET: Mostrar el formulario para ingresar el código 2FA.
            - POST: Validar el código ingresado contra el almacenado y verificar que no haya expirado. Si es válido, iniciar la sesión del usuario con login_user de Flask-Login y redirigir al panel principal. Limpiar el código 2FA de la base de datos.
        - Implementar la ruta /logout: Terminar la sesión del usuario con logout_user y redirigir a la página de login.
    - **Gestión de Sesiones:** Configurar Flask-Login (User loader, etc.).
    - **Protección de Rutas:** Usar el decorador @login_required de Flask-Login.
    - **Control de Roles:** Crear un decorador personalizado (ej. @role_required('admin', 'director')) en app/decorators.py para restringir el acceso a ciertas rutas o funcionalidades basándose en el campo role del usuario actual.
- **Módulo Principal (app/main):**
    - **Formulario de Estudiantes:**
        - Crear StudentForm usando Flask-WTF en app/main/forms.py, incluyendo validadores para cada campo.
        - Implementar la ruta para mostrar el formulario (GET) y procesar los datos enviados (POST).
        - En el POST, validar los datos del formulario. Si son válidos, sanitizar las entradas y crear una nueva instancia del modelo Estudiante, guardándola en la base de datos. Mostrar mensajes flash de éxito o error.
    - **Carga de Archivo Excel:**
        - En la misma ruta del formulario o una dedicada, manejar la subida de archivos (request.files).
        - Validar el tipo de archivo (solo .xlsx) y el tamaño.
        - Usar pandas para leer el archivo Excel.
        - Iterar sobre las filas, validar y sanitizar los datos de cada estudiante.
        - Verificar si el estudiante ya existe en la base de datos (por DNI) antes de insertarlo para evitar duplicados.

        - Guardar los nuevos estudiantes en la base de datos. Mostrar mensajes flash.
    - **Reporte de Estudiantes:**
        - Implementar una ruta que consulte todos los estudiantes de la base de datos usando SQLAlchemy.
        - Pasar la lista de estudiantes a una plantilla HTML para su visualización.
- **Seguridad Adicional:**
    - **Sanitización:** Utilizar bleach para sanitizar cualquier dato que provenga del usuario y que vaya a ser mostrado en plantillas HTML, para prevenir XSS.
    - **Protección CSRF:** Flask-WTF integra protección CSRF automáticamente en los formularios. Asegurarse de incluir el token CSRF en las plantillas.
    - **Cabeceras de Seguridad:** Configurar Flask-Talisman en app/__init__.py para establecer cabeceras HTTP de seguridad (ej. CSP, HSTS).
    - **Manejo de Errores:** Crear manejadores de error personalizados para códigos HTTP comunes (403, 404, 500) que muestren páginas de error amigables.

3.3. Desarrollo del Frontend (HTML, CSS, JavaScript)

- **Plantillas HTML (Jinja2):**
    - Desarrollar las plantillas HTML (.html) en el directorio app/templates/.
    - Utilizar la herencia de plantillas de Jinja2 ({% extends 'base.html' %}, {% block content %}) para mantener una estructura consistente.
    - Integrar los formularios de Flask-WTF en las plantillas.
    - Mostrar mensajes flash (get_flashed_messages()).
    - Renderizar dinámicamente los datos pasados desde las rutas de Flask (ej. la lista de estudiantes en el reporte).
- **Estilos con Bootstrap y CSS Personalizado:**
    - Incluir Bootstrap CSS (desde un CDN o localmente) en base.html.
    - Utilizar las clases de Bootstrap para dar estilo a los elementos HTML y crear layouts responsivos.
    - Crear un archivo CSS personalizado (ej. app/static/css/style.css) para estilos específicos que no cubra Bootstrap o para sobrescribir estilos por defecto.
- **Interactividad con JavaScript:**
    - Implementar la lógica para los selectores dinámicos de País/Ciudad en el formulario de admisión. Esto implicará:
        - Un script JS que escuche el evento change del selector de país.
        - Al cambiar el país, realizar una petición (podría ser a un endpoint de Flask que devuelva las ciudades en JSON, o tener los datos de ciudades pre-cargados en el JS si no son demasiados) para obtener las ciudades correspondientes.

        - Actualizar dinámicamente las opciones del selector de ciudad.
    - Añadir validaciones del lado del cliente como complemento a las validaciones del servidor, para mejorar la experiencia de usuario.
    - Cualquier otra interactividad necesaria (ej. modales, tooltips) utilizando JavaScript y/o componentes de Bootstrap.

3.4. Integración Backend y Frontend

- Asegurar que los datos fluyan correctamente entre las rutas de Flask (backend) y las plantillas Jinja2 (frontend).
- Verificar que los formularios envíen los datos correctamente y que las respuestas del servidor se reflejen adecuadamente en la interfaz de usuario.
- Probar la interactividad JavaScript y su comunicación con el backend si es necesario (ej. para los selectores dinámicos).

Fase 4: Despliegue

Esta fase consiste en poner la aplicación en un servidor para que sea accesible a los usuarios finales.

4.1. Preparación para Producción

- **Configuraciones Específicas de Producción:**
    - Asegurarse de que DEBUG esté establecido a False en la configuración de Flask.
    - Utilizar una SECRET_KEY fuerte y única, gestionada a través de variables de entorno.
    - Configurar el logging para un entorno de producción (ej. registrar errores en archivos o en un servicio de logging).
- **Elección y Configuración del Servidor WSGI:**
    - Para entornos Linux/macOS, Gunicorn es una opción popular y robusta.
    - Para entornos Windows, Waitress es una alternativa adecuada.
    - El servidor WSGI se encargará de manejar las peticiones HTTP y pasarlas a la aplicación Flask.
- **Servidor Web como Proxy Inverso (Recomendado):**
    - Configurar un servidor web como Nginx o Apache delante del servidor WSGI.
    - Beneficios: Manejo de conexiones estáticas, balanceo de carga (si se escala), terminación SSL/TLS (HTTPS), compresión, caching.
- **Base de Datos en Producción:**
    - Configurar la instancia de Microsoft SQL Server para el entorno de producción. Esto podría ser una instancia en un servidor dedicado, una máquina virtual, o un servicio de base de datos en la nube (ej. Azure SQL

Database, AWS RDS for SQL Server).
    - Asegurar que la cadena de conexión en la aplicación apunte a la base de datos de producción y que las credenciales sean seguras y gestionadas como variables de entorno.

4.2. Opciones de Despliegue (Plataformas Gratuitas/Económicas)

Se explorarán plataformas que permitan desplegar aplicaciones Flask y que ofrezcan compatibilidad con MSSQL o permitan conexiones a bases de datos externas.

- **PythonAnywhere:** Ofrece un plan gratuito que es bueno para aplicaciones pequeñas y prototipos. Permite desplegar aplicaciones WSGI como Flask. La conexión a una instancia externa de MSSQL sería necesaria.
- **Heroku:** Aunque su plan gratuito ha cambiado, sigue siendo una opción PaaS popular. Se necesitaría un add-on para MSSQL o conectar a una instancia externa.
- **Render:** Ofrece un plan gratuito para servicios web y bases de datos PostgreSQL. Para MSSQL, se requeriría una instancia externa.
- **Servidores VPS (Virtual Private Server):** Opciones como DigitalOcean, Linode, Vultr ofrecen VPS económicos donde se puede instalar y configurar todo el stack manualmente (SO, MSSQL Server, Python, Nginx, Gunicorn). Esto da más control pero requiere más administración.
- **Azure App Service / Azure SQL Database:** Si se busca una integración más nativa con el ecosistema Microsoft, Azure es una opción, aunque los planes gratuitos pueden ser limitados para bases de datos MSSQL de producción.

4.3. Proceso General de Despliegue (Conceptual)

1. **Transferir Código al Servidor:** Usar Git para clonar el repositorio en el servidor de producción.
2. **Configurar Entorno del Servidor:**
    - Instalar Python, pip y las dependencias del sistema operativo necesarias (ej. ODBC drivers para MSSQL).
    - Crear y activar un entorno virtual.
    - Instalar las dependencias de Python desde requirements.txt.
    - Configurar las variables de entorno para producción (SECRET_KEY, cadena de conexión a la BD de producción, configuración de email, etc.).
3. **Configurar Base de Datos:** Asegurar que la base de datos de producción esté creada, las tablas migradas (usando Flask-Migrate si se implementó, o scripts SQL) y accesible desde el servidor de la aplicación.
4. **Configurar Servidor WSGI:** Configurar Gunicorn o Waitress para servir la aplicación Flask (ej. número de workers, host, port).

5. **Configurar Proxy Inverso (Nginx/Apache):**
    - Configurar el servidor web para que actúe como proxy inverso, redirigiendo las peticiones al servidor WSGI.
    - Configurar el servicio de archivos estáticos.
    - Configurar SSL/TLS para habilitar HTTPS (ej. usando Let's Encrypt para certificados gratuitos).
6. **Iniciar la Aplicación:** Iniciar el servidor WSGI y el servidor web.
7. **Configurar DNS:** Apuntar el nombre de dominio de la aplicación (si se tiene) a la dirección IP del servidor.

4.4. Consideraciones de Seguridad Post-Despliegue

- **HTTPS:** Asegurar que todo el tráfico se sirva sobre HTTPS.
- **Actualizaciones Regulares:** Mantener actualizado el sistema operativo, el servidor web, Python, Flask y todas las dependencias para parchear vulnerabilidades conocidas.
- **Logs y Monitorización:** Configurar la recolección de logs de la aplicación y del servidor. Implementar monitorización básica para detectar errores o actividad inusual.
- **Copias de Seguridad:** Establecer una estrategia de copias de seguridad regulares para la base de datos.

Conclusión

Esta guía ha delineado las fases y consideraciones clave para el desarrollo de una aplicación web completa utilizando Flask, HTML, CSS, JavaScript y Microsoft SQL Server. Siguiendo este plan conceptual, se puede abordar el proyecto de manera estructurada, desde la definición de requisitos y el diseño, pasando por una descripción de la implementación de las funcionalidades y medidas de seguridad, hasta el despliegue final en un entorno de producción. El enfoque en tecnologías gratuitas y las mejores prácticas de desarrollo busca asegurar un producto final robusto, seguro y mantenible.