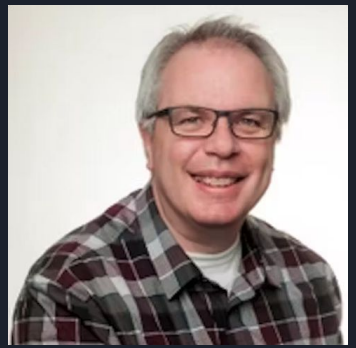
Generative AI para el Desarrollo de Software

Curso sobre el uso de la Inteligencia Artificial Generativa (IAG) en el desarrollo de software, impartido por Laurence Moroney.



Dependencias en el Desarrollo de Software

El desarrollo de software moderno se basa en dependencias que facilitan la creación de aplicaciones. Este curso aborda la gestión de dependencias y el papel de los Modelos de Lenguaje de Aprendizaje (LLM) en este proceso.

¿Qué es una Dependencia?

- Una dependencia es una biblioteca o módulo necesario para el funcionamiento de un proyecto.

  Se clasifican en:

  - Dependencias internas:
    - Módulos creados por el desarrollador.
  - Dependencias externas:
    - Bibliotecas de terceros.

Importancia de las Dependencias

- Las dependencias permiten a los desarrolladores aprovechar el código existente, lo que acelera el proceso de desarrollo

- Ejemplos de dependencias comunes
  - Desarrollo Web
    - Flask, Django
  - Análisis de Datos
    - NumPy, Pandas

Desafíos Comunes

- **Conflictos de Versiones:**
  - **Diferentes versiones de la misma biblioteca.**
- **Vulnerabilidades de Seguridad:**
  - **Riesgos por bibliotecas desactualizadas.**
- **Dependencias Transitivas:**
  - **Redes complejas de dependencias.**

Cómo los LLM Pueden Ayudar

- Los LLM son herramientas valiosas para:
  - Generación de Ideas:
    - Identificar bibliotecas adecuadas.
  - Información sobre Dependencias:
    - Detalles de paquetes desconocidos.
  - Detección de Conflictos:
    - Identificar problemas de compatibilidad.
  - Resolución de Conflictos:
    - Sugerencias para resolver problemas

Recomendaciones para el Uso de LLM

- **Probar con Proyectos Reales:**
  - **Interactuar sobre proyectos recientes**
- **Proporcionar Contexto:**
  - **Ofrecer detalles específicos para respuestas más precisas.**

Introducción a los Entornos Virtuales

- Un entorno virtual es un espacio aislado que gestiona las dependencias de un proyecto sin interferir con otros

- Ventajas:
  - Aislamiento:
    - Cada proyecto tiene su propio conjunto de dependencias.
  - Reproducibilidad:
    - Asegura que el proyecto se ejecute igual en diferentes entornos.
  - Facilidad de Administración:
    - Facilita la gestión y actualización de dependencias

Entorno Virtual en Python

- 1. Instalar Python.
  2. Navegar al directorio deseado.

  3. Crear el entorno virtual:
     - python3 -m venv myenv.

  4. Activar el entorno:
  - Linux o Mac:
    - source myenv/bin/activate
  - Windoes
    - ./myenv/Scripts/activate
  5. Instalar dependencias: `pip install requests`.
  6. Listar paquetes: `pip list`.
  7. Desactivar el entorno: `deactivate`.

Actividad Práctica

- Crear entornos virtuales `env1` y `env2`.
  - En `env1`, instalar:
    - requests sin versión específica.
  - En `env2`, instalar:
    - `requests` versión `2.31.0`.
- Activar y desactivar entornos usando `source` y `deactivate`

Actividad Práctica

- Crear entornos virtuales `env1` y `env2`.
  - En `env1`, instalar:
    - requests sin versión específica.
  - En `env2`, instalar:
    - `requests` versión `2.31.0`.
- Activar y desactivar entornos usando `source` y `deactivate`

Gestión de Dependencias en Proyectos de Software

Exploración de cómo los Modelos de Lenguaje (LLM) facilitan la gestión de dependencias en proyectos de software.

Herramientas y comandos útiles para manejar bibliotecas y sus dependencias,

Introducción a la Gestión de Dependencias

- Proyectos modernos tienen múltiples dependencias directas y transitivas.
- La gestión efectiva es crucial para los desarrolladores.
- Herramientas de Gestión de Paquetes:
  - Python:
    - Pip, Poetry, Pipen, Conda
  - JavaScript:
    - Npm, Yarn
  - C#
    - Nuget

Comandos Útiles

- Comandos de pip
  - Instalar un paquete:
    - pip install nombre_del_paquete
  - Instalar múltiples paquetes desde un archivo
    - pip install -r requirements.txt
  - Desinstalar un paquete
    - pip uninstall nombre_del_paquete
  - Actualizar un paquete a la última versión
    - pip install --upgrade nombre_del_paqute
  - Listar todos los paquetes instalados
    - pip list
  - Mostrar información sobre un paquete específico
    - pip show nombre_del_paquete
  - Buscar un paquete
    - pip search nombre_del_paquete

Comandos Útiles

- Comandos de pip
  - Generar un archivo `requirements.txt` con los paquetes instalado
    - pip freeze > requirements.txt
  - Instalar una versión específica de un paquete
    - pip install nombre_del_paquete==versión

Uso de pip-tools

Herramienta para mantener actualizadas las dependencias.

- **Comandos principales**
  - **pip compile**
    - **Genera `requirements.txt`**
  - **pip sync**
    - **Sincroniza el entorno.**

Proceso de Instalación

- Instalar `pip-tools`.
  - Crear `requirements.in`
    - con dependencias directa
  - Ejecutar `pip-compile`
    - para generar `requirements.txt`.

- Para replicar un entorno virtual:
  - Crear un segundo entorno (ej. `env2`).
    - Copiar requirements.txt del entorno original.
    - usar `pip-sync` para sincronizar dependencias.

¿Qué hace la biblioteca 'itsdangerous?

La biblioteca `itsdangerous` en Python se usa para firmar datos de forma segura y verificar su integridad sin necesidad de cifrado. Es útil cuando necesitas enviar datos confiables entre partes sin exponerlos a modificaciones malintencionadas.

Principales usos:

1. **Firmar y verificar datos**
   - Puedes firmar cualquier objeto serializable (como cadenas o JSON) y luego verificar su autenticidad.
2. **Tokens temporales**
   - Puedes generar tokens con vencimiento, útiles para restablecimiento de contraseñas, validación de correos, etc.
3. **Prevención de manipulación de datos**
   - Garantiza que los datos enviados no haya[illegible]o alterados.

Tabla de Comandos Útiles

Comando	Descripción
pip list	Muestra los paquetes instalados en el entorno
pip freeze	Lista las dependencias con sus versiones.
pip install	Instla un paquete.
pip-compile	Genera `requirements.txt` a partir de `requirements.in
pip-sync	Sincroniza el entorno con `requirements.txt`

Resolución de Conflictos de Dependencias con LLMs

Los conflictos de dependencias son un desafío común en el desarrollo de software.

Los LLMs pueden ayudar a identificar y resolver estos conflictos.

Conflictos de Dependencias

- Ocurren cuando:
  - Dos o más librerías requieren diferentes versiones de la misma dependencia.
  - Resultan en problemas de compatibilidad y errores en el proyecto.
- Pasos para Resolvelo:
  - Identificación del Conflicto:
    - Usa un LLM para identificar conflictos en `requirements.txt`.
  - Búsqueda de Compatibilidad:
    - Verifica versiones compatibles sugeridas por el LLM.
  - Actualización de Dependencias:
    - Actualiza según recomendaciones y itera.

Conflictos de Dependencias

- **Si no se resuelve el conflicto:**

  - **Investigación :**
    - **Busca soluciones en la web o Stack Overflow.**

  - **Librerías Alternativas:**
    - **Un LLM puede sugerir librerías con diferentes dependencias.**

Conflictos de Dependencias

- Si no se resuelve el conflicto:
  - Investigación :
    - Busca soluciones en la web o Stack Overflow.
  - Librerías Alternativas:
    - Un LLM puede sugerir librerías con diferentes dependencias.
- No hay una solución única para la gestión de dependencias.
- Los LLMs son herramientas valiosas para identificar problemas y sugerir soluciones.

Conflictos de Dependencias

- Si no se resuelve el conflicto:
  - Investigación :
    - Busca soluciones en la web o Stack Overflow.
  - Librerías Alternativas:
    - Un LLM puede sugerir librerías con diferentes dependencias.
- No hay una solución única para la gestión de dependencias.
- Los LLMs son herramientas valiosas para identificar problemas y sugerir soluciones.

Seguridad en Dependencias de Software

la seguridad en las dependencias de software, incluyendo riesgos, herramientas y el uso de LLM.

Riesgos de Seguridad en Dependencias

- Las dependencias pueden introducir vulnerabilidades.
  Principales riesgos:
  - 1. Paquetes Desactualizados:
    - Vulnerabilidades por falta de actualizaciones.
  - 2. Paquetes Transitorios:
    - Dificultad para rastrear actualizaciones.
  - 3. Dependencias No Mantenidas:
    - Riesgo de vulnerabilidades desconocidas.

Herramientas para Verificar Vulnerabilidades

- Herramientas útiles en Python:
  - pip-audit:
    - Audita dependencias y reporta vulnerabilidades.
- bandit:
  - Análisis de código estático.
- pipcheck:
  - Verifica paquetes desactualizados.
- CI/CD:
  - Integrar seguridad automatizada.

Ejemplo Práctico

- **Instalación de pip-audit:**
  - **Auditar dependencias y detectar advertencias.**

- **Uso de LLM:**
  - **Consultar sobre seguridad de paquetes.**

- **Actualización de Paquetes:**
  - **Solicitar soluciones al LLM para vulnerabilidades.**

Reflexiones sobre el Uso de LLM

- Los LLM son útiles, pero no deben ser la única fuente de información. Complementar con herramientas específicas es esencial.
- Ventajas de los LLM:
  - Generación de Ideas:
    - Excelentes para sugerir bibliotecas y paquetes.
  - Información sobre Dependencias:
    - Proporcionan detalles sobre las dependencias.
  - Identificación de Conflictos y Vulnerabilidades:
    - Ofrecen soluciones relacionadas con dependencias.
- Debilidades de los LLM:
  - Importante conocer la fecha de los datos de entrenamiento.
    - Menos útiles para bibliotecas no populares o nuevas.

Aplicación en Diferentes Lenguajes

- JavaScript NPM:
  - Administrador de paquetes que gestiona dependencias.
  - Archivo package.json:
    - Contiene información sobre la aplicación y dependencias.
- Uso de LLM con NPM:
  - Análisis de Código:
    - Ayuda a analizar dependencias.
  - Auditoría de Dependencias:
    - Audita el conjunto actual de dependencias
- npm audit:
  - Comprobar vulnerabilidades.
- npm outdated:
  - Verificar paquetes desactualizados.
- npm install <nombre_del_paquete> --save:
  - Actualizar un paquete específico.

Resumen

Los LLM son herramientas valiosas para resumir bibliotecas y depurar conflictos.

Útiles en equipos grandes y entornos con múltiples lenguajes.

Permiten a los desarrolladores crear proyectos más interesantes y eficientes.

Los LLM facilitan la gestión de dependencias en el desarrollo de software, independientemente del lenguaje utilizado.