TP5 - Herramientas de construcción de software

1- Objetivos de Aprendizaje

- Utilizar herramientas de construcción de software y manejo de paquetes y dependencias

2- Unidad temática que incluye este trabajo práctico

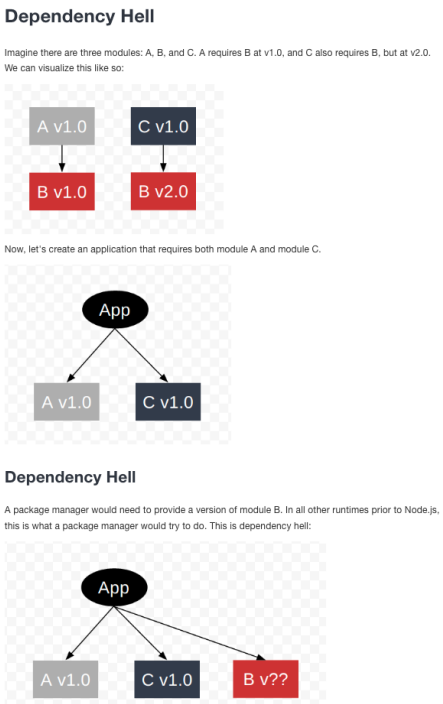
Este trabajo práctico corresponde a la unidad Nº: 3 (Libro Continuous Delivery: Cap 6 y 13)

3- Consignas a desarrollar en el trabajo práctico:

- Las aplicaciones utilizadas son del tipo "Hello World", dado que el foco del trabajo práctico es como construirlas y no el funcionamiento de la aplicación en sí.
- En los puntos en los que se pida alguna descripción, realizarlo de la manera más clara posible.

4- Desarrollo:

Dependency Hell / DLLs Hell:



Un software administrador de paquetes es una herramienta esencial en el desarrollo de software que permite a los programadores gestionar eficientemente las bibliotecas, módulos y componentes de código reutilizable que se utilizan en un proyecto. En esencia, actúa como un intermediario que simplifica la incorporación, actualización y eliminación de estas piezas de software en una aplicación.

Su funcionalidad principal abarca desde la descarga automatizada de las bibliotecas necesarias hasta la resolución de conflictos y la garantía de coherencia en las versiones. Los administradores de paquetes

también facilitan la especificación de las dependencias exactas o los rangos de versiones que un proyecto requiere, asegurando así que el software funcione de manera predecible y estable.

Estos sistemas, como npm para el ecosistema JavaScript o NuGet para el ecosistema .NET, reducen la carga de trabajo manual al automatizar tareas tediosas y propensas a errores. Al hacerlo, fomentan la reutilización de código, la colaboración entre desarrolladores y la creación eficiente de aplicaciones sólidas y escalables.

En resumen, un software administrador de paquetes es una herramienta esencial para la gestión eficiente de bibliotecas y componentes de código en proyectos de desarrollo de software. Proporciona una estructura organizada para incorporar, gestionar y mantener estas dependencias, lo que contribuye a la eficiencia, la coherencia y la calidad en el proceso de desarrollo.

Utilizar administradores de paquetes como npm (Node Package Manager) y NuGet en lugar de no hacerlo ofrece varias ventajas significativas en el desarrollo de software:

Gestión de dependencias eficiente: En el desarrollo de software, es común utilizar bibliotecas y módulos externos para agilizar el proceso de programación. Sin embargo, manejar manualmente las dependencias puede ser complicado y propenso a errores. Los administradores de paquetes resuelven este problema al automatizar la gestión de dependencias. Te permiten definir las bibliotecas que tu proyecto necesita y aseguran que se instalen correctamente, eliminando la necesidad de descargar y configurar cada componente individualmente.

Coherencia y estabilidad: Los administradores de paquetes garantizan que todas las versiones de las bibliotecas utilizadas en tu proyecto sean coherentes y compatibles entre sí. Esto previene problemas de incompatibilidad y conflictos que podrían surgir al mezclar diferentes versiones de bibliotecas manualmente. Además, la capacidad de especificar versiones exactas o rangos de versiones en los administradores de paquetes asegura que tu proyecto funcione de manera predecible y estable en todo momento.

Reutilización de código y colaboración: Los administradores de paquetes fomentan la reutilización de código al permitir a los desarrolladores compartir sus bibliotecas y componentes con otros. Esto significa que puedes aprovechar el trabajo de otros y no tienes que reinventar la rueda en cada proyecto. La comunidad de desarrolladores contribuye con una amplia gama de paquetes, lo que acelera el desarrollo y mejora la calidad del software.

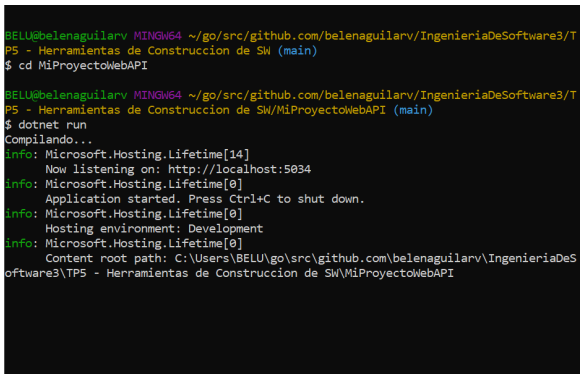
Simplificación del proceso de desarrollo: Al utilizar administradores de paquetes, te liberas de la carga de tener que rastrear manualmente las actualizaciones de las bibliotecas, buscar nuevas versiones y realizar descargas manuales. Esto simplifica significativamente el proceso de desarrollo y te permite concentrarte en la lógica de tu aplicación en lugar de gestionar las dependencias.

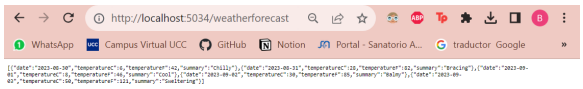
Mantenimiento y escalabilidad: Los proyectos de software tienden a crecer con el tiempo y pueden volverse complejos. Los administradores de paquetes facilitan el mantenimiento y la escalabilidad al proporcionar una forma ordenada de incorporar y gestionar nuevos componentes. Esto ayuda a mantener el código más limpio y facilita la adición de nuevas características y mejoras.

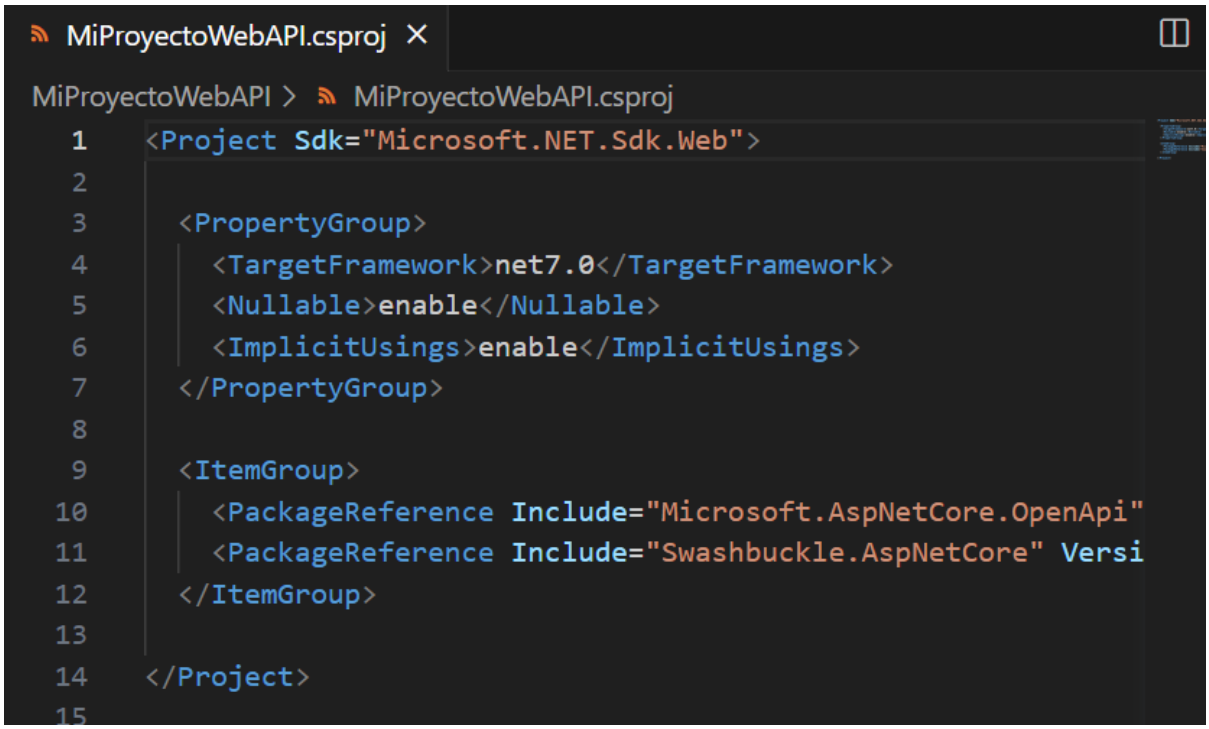
En resumen, utilizar administradores de paquetes como npm y NuGet es esencial en el desarrollo moderno debido a su capacidad para gestionar dependencias, garantizar la coherencia, fomentar la colaboración y simplificar el proceso de desarrollo. Estas herramientas ahorran tiempo, reducen errores y contribuyen a la creación de software más robusto y eficiente.

1- Ejemplo con C# y .NET Core

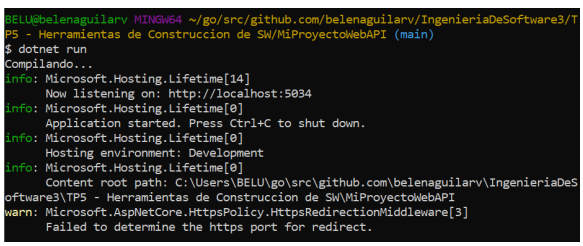
- Instalar el SDK de .NET Core: Asegúrate de tener el SDK de .NET Core instalado en tu sistema. Puedes descargarlo desde el sitio web oficial de .NET: https://dotnet.microsoft.com/download
- Crear un Proyecto de Web API:
- Abre una terminal y navega hasta la ubicación donde deseas crear tu proyecto. Luego, ejecuta el siguiente comando para crear un nuevo proyecto de Web API:

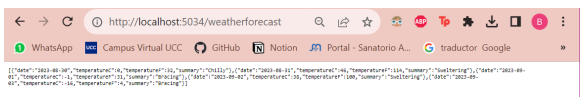


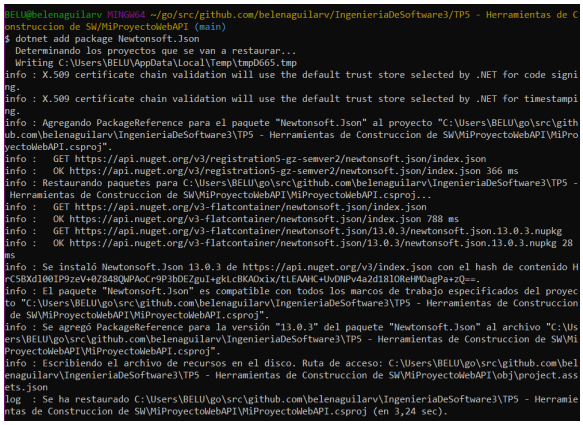


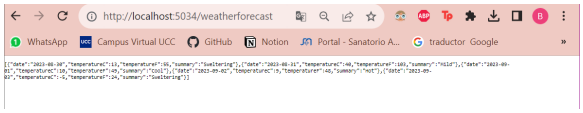


despues de borrar los directorios: **bin** y **obj**









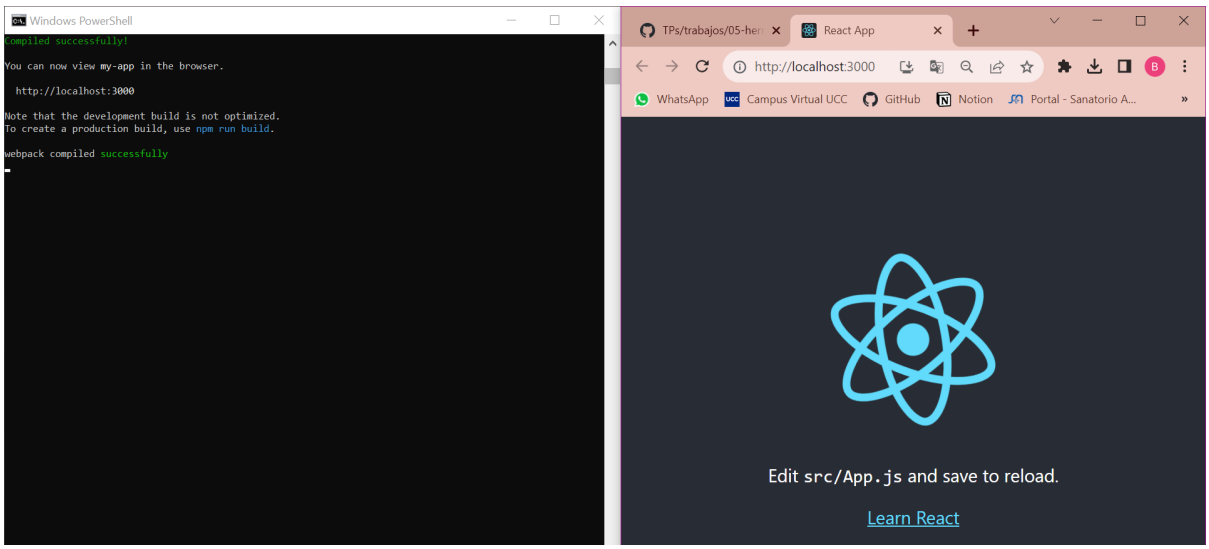
Es una buena práctica eliminar los directorios "bin" y "obj" antes de subir el código a un repositorio, especialmente en sistemas de control de versiones como Git. Estos directorios contienen archivos generados localmente durante la compilación y construcción, y no son necesarios para la colaboración en el código fuente.

> 💡 Para evitar cargar código innecesario al repositorio tambien se pueden agregar reglas al archivo `.gitignore`

2. Ejemplo con node.js

```
BELU@belenaguilarv MINGW64 ~/go/src/github.com/belenaguilarv/IngenieriaDeSoftware3/TP5 - Herramientas de Construccion d
e SW/MiProyectoWebAPI (main)
$ node -v
v18.16.0
```

```
BELU@belenaguilarv MINGW64 ~/go/src/github.com/belenaguilarv/IngenieriaDeSoftware3/TP5 - Herramientas de Construccion d
e SW/MiProyectoWebAPI (main)
$ npx create-react-app my-app

Creating a new React app in C:\Users\BELU\go\src\github.com\belenaguilarv\IngenieriaDeSoftware3\TP5 - Herramientas de C
onstruccion de SW\MiProyectoWebAPI\my-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...


added 1441 packages in 1m

241 packages are looking for funding
  run `npm fund` for details

Installing template dependencies using npm...

added 69 packages, and changed 1 package in 17s

245 packages are looking for funding
  run `npm fund` for details
Removing template package using npm...


removed 1 package, and audited 1510 packages in 3s

245 packages are looking for funding
  run `npm fund` for details

6 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

Success! Created my-app at C:\Users\BELU\go\src\github.com\belenaguilarv\IngenieriaDeSoftware3\TP5 - Herramientas de Co
nstruccion de SW\MiProyectoWebAPI\my-app
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!
```



3. Build tools para otros lenguajes

Hacer una lista de herramientas de build (una o varias) para distintos lenguajes, por ejemplo (Rust -> cargo). Elegir al menos 10 lenguajes de la lista de top 20 o top 50 de tiobe: https://www.tiobe.com/tiobe-index/

1. **Java:**
   - Apache Maven
   - Gradle
2. **Python:**
   - pip (para la gestión de paquetes)
   - setuptools
   - virtualenv
3. **JavaScript/Node.js:**
   - npm (Node Package Manager)
   - yarn
   - Webpack
   - Gulp
4. **C#:**
   - MSBuild (utilizado por Visual Studio)
   - Cake
   - FAKE (F# Make)
5. **Ruby:**
   - Rake
   - Bundler (para la gestión de gemas)
6. **C/C++:**
   - Make
   - CMake
   - Ninja
   - Meson
7. **Go:**
   - go build
   - go install
   - go get (para la gestión de paquetes)
8. **Rust:**
   - Cargo
9. **PHP:**

- Composer (para la gestión de paquetes)
- Phing
- Ant

10. **Swift:**
    - Swift Package Manager
    - CocoaPods (para proyectos iOS/macOS)
11. **Kotlin:**
    - Gradle (también es utilizado por proyectos de Android)
12. **Perl:**
    - MakeMaker (utilizado para módulos CPAN)
13. **Haskell:**
    - Cabal
    - Stack
14. **Elixir:**
    - Mix (utilizado para proyectos Elixir)
15. **Scala:**
    - sbt (Scala Build Tool)
16. **Lua:**
    - LuaRocks (para la gestión de módulos)
17. **Objective-C:**
    - xcodebuild (utilizado por Xcode)
18. **Groovy:**
    - Gradle (también es utilizado por proyectos de Gradle)
19. **Perl:**
    - MakeMaker (utilizado para módulos CPAN)
20. **TypeScript:**
    - TypeScript Compiler (tsc)
    - Webpack (también es utilizado para proyectos JavaScript/Node.js)