+++ date = '2025-02-21T10:19:38-08:00' draft = false title = 'Practica 3' +++

Proyecto Todo en Haskell usando Stack

Este documento detalla el proceso de construcción de una aplicación de tipo "Todo list" utilizando el lenguaje Haskell y la herramienta de gestión de proyectos Stack. La aplicación permite gestionar tareas desde la línea de comandos de forma interactiva.

---

1. Instalación y Preparación

Stack

Stack es una herramienta que facilita la gestión de proyectos en Haskell, incluyendo dependencias y compilación. En mi caso, uso una Mac con procesador Apple M1, lo que complicó un poco la instalación. Finalmente, logré instalar Stack utilizando un script proporcionado por el equipo de Haskell.

Verifiqué que Stack estaba funcionando correctamente ejecutando su comando principal, que mostró las opciones disponibles y confirmó la instalación exitosa.

---

2. Creación del Proyecto

Una vez instalado Stack, utilicé el comando `stack new` para crear un nuevo proyecto llamado `todo`. Esto generó una estructura básica de carpetas que incluye los directorios para el código fuente, las pruebas y archivos de configuración.

Durante la creación, Stack solicitó algunos metadatos del proyecto como el nombre del autor, email, entre otros, que completé según mi información personal.

---

3. Configuración de Dependencias

En el archivo `package.yaml`, agregué dependencias necesarias para funcionalidades adicionales que planeaba utilizar, como por ejemplo:

- `dotenv`: Para manejo de variables de entorno si se llegaba a requerir.
- `open-browser`: Para abrir enlaces desde la terminal si fuera útil más adelante.

Estas dependencias se listan en la sección correspondiente del archivo, lo que permite que Stack las instale automáticamente.

---

4. Verificación del Proyecto Base

Antes de comenzar a modificar el código, verifiqué que el proyecto base compilara y ejecutara correctamente. Usé los comandos `stack test` y `stack run` para asegurarme de que el entorno estaba listo y que no había errores en la configuración inicial.

🧱 5. Diseño de la Aplicación

La aplicación es una utilidad de línea de comandos que interpreta comandos del usuario para manipular una lista de tareas. El diseño se centró en simplicidad y claridad.

Los comandos disponibles son:

- `+ <texto>`: Agrega una nueva tarea.
- `- <número>`: Elimina la tarea con ese número.
- `s <número>`: Muestra el contenido de la tarea indicada.
- `e <número>`: Permite editar el contenido de una tarea.
- `l`: Lista todas las tareas actuales.
- `r`: Invierte el orden de la lista de tareas.
- `c`: Borra todas las tareas.
- `q`: Sale de la aplicación.

🔄 6. Flujo de Ejecución

La aplicación inicia mostrando los comandos disponibles y luego entra en un bucle de espera de entrada del usuario.

Cada comando ingresado se interpreta y se ejecuta la acción correspondiente. Las operaciones sobre la lista de tareas son funcionales, es decir, se devuelven nuevas versiones de la lista en lugar de modificarla en el lugar.

🔬 7. Pruebas

Para asegurar el correcto funcionamiento de las funciones clave, implementé una prueba automatizada en el archivo de pruebas del proyecto.

Esta prueba se enfocó en verificar que la función que edita una tarea en una posición determinada realmente la reemplaza correctamente.

Las pruebas fueron ejecutadas con `stack test` y pasaron sin errores.

🚀 8. Compilación como Ejecutable

Compilé el proyecto como ejecutable usando `stack install` con una opción para colocar el binario en la carpeta actual.

Esto generó un archivo ejecutable que luego moví manualmente a `/usr/local/bin`, permitiéndome ejecutarlo desde cualquier parte del sistema simplemente escribiendo `todo`.

📚 9. Recursos de Apoyo

Durante el desarrollo consulté:

- 📓 **Blog que sirvió de guía**: Usé como base un artículo específico que enseña a crear esta app.
- 📘 **Libro "Learn You a Haskell for Great Good!"**: Me ayudó a comprender conceptos como listas, IO, recursión y tipos de datos.
- 🧪 **REPL interactivo (`stack repl`)**: Lo utilicé para probar funciones pequeñas antes de integrarlas al proyecto.

☑ 10. Conclusiones

Este proyecto fue una excelente manera de introducirme en Haskell de forma práctica. Me permitió:

- Comprender cómo funciona Stack y cómo estructurar un proyecto.
- Trabajar con listas y entradas/salidas en Haskell.
- Aplicar pruebas básicas para garantizar la calidad del código.
- Crear un flujo de trabajo funcional desde la terminal.

Esta aplicación puede seguir ampliándose con persistencia en archivos, interfaz gráfica u otras características, pero ya es una base sólida para entender programación funcional.
