Franco Bustos

Sistema Solar
Ejercicio

Tecnología

Para la realización del ejercicio utilicé las siguientes tecnologías:

- ASPNET Core Web Api 2.1
- Base de datos MySql
- Hosting en Google Cloud App Engine
- ORM -> Entity Framework Core
- Swagger UI

Modelo de Datos

He utilizado una base de datos relacional para almacenar las estructuras creí convenientes para soportar las solicitudes del ejercicio y dejar una solución óptima y escalable. He elegido MySql, que entre las facilidades de conexión y uso gratuito, tiene compatibilidad con SQL Cloud en App Engine. Si bien puede parecer que una base de datos relacional es demasiado para almacenar tan pocos registros, he pensado el sistema en algo que pueda escalar y registrar más datos, con posiblemente más planetas. Para la Api, al momento de acceder al un pronóstico por dia en particular utilizo una caché previo al acceso a BD. Elegí esta arquitectura ante un kvs en redis porque considero que esto me da más flexibilidad para escalar y guardar datos históricos, aunque utilizar redis para el caso también hubiese solucionado el problema.

La aplicación una vez que arranca, ejecuta un job que inserta los planetas (los 3 planetas del ejercicio) en la tabla Planeta. Estos planetas se toman por default del appsettings.json, ya que me permitió configurar algunas propiedades de los planetas antes de correr el job. Por ejemplo la posición inicial, el sentido de giro, la velocidad angular.

Job

```
public Job Run(IEnumerable<Planeta> planetas, int anios = 10, string fechaInicio = null)
{
    try
    {
        if (anios <= 0)
        {
            return null;
        }

        // Se inicializan los planetas y se borran los pronosticos existentes
        planetas = Initialize(planetas).ToList();
        // Calculo de la fecha de inicio
        var fecha = string.IsNullOrEmpty(fechaInicio) ? DateTime.Today : Convert.ToDateTime(fechaInicio);
        // Se crea el job
        var job = this.CreateJob(anios, fecha);
        // Se corre el pronosticador
        _pronosticoService.PronosticarClima(planetas, anios, fecha, job.JobId);

        return job;
    }
    catch (Exception ex)
    {
        _logger.LogError("Ha ocurrido un error mientras se ejecutaba el job: ", ex.Message);
        return null;
    }
}
```

Lo primero que hace el job es eliminar los pronósticos anteriores que pudieron haber e insertar los planetas si es que no existen en la BD, Luego se calcula la fecha de inicio si no es seteada en configuración, por defecto utiliza la fecha de hoy.

Se crea el Job en la BD con la fecha de inicio y la cantidad de años a pronosticar.

Luego se ejecuta la función PronosticarClima. Esta función itera sobre el total de días a pronosticar y en base a la posición actualizada del planeta (se va actualizando en cada iteración) se calcula el clima para dicho día en base a las condiciones del ejercicio. Para determinar si los planetas están alineados utilizo la tangente entre los puntos y a su vez con el sol (posición |0.0|), esto me determina si el clima es Sequía u Óptimo. Si no es ninguno de estos, los planetas forman un triángulo que si contiene al sol el clima es Lluvia. Para saber si contiene al sol voy calculando el área de los triángulos internos entre el sol y dos puntos. La sumatoria de los triángulos internos debe ser igual al area total. En caso de que el área no sea igual, es porque el triángulo formado por las posiciones de los planetas

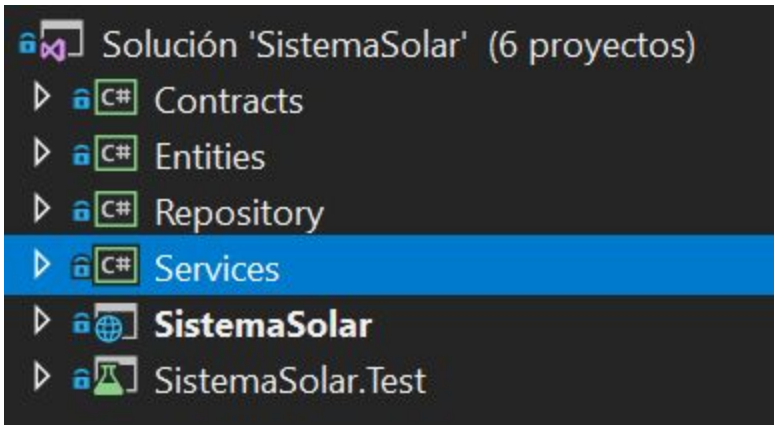
no contienen al sol, y como el ejercicio no habla sobre esta posibilidad, guardo el clima “Indeterminado”.

En caso de que el clima sea de lluvia, para saber el pico del nivel de lluvia durante los pronósticos, voy almacenando el perímetro de los triángulos y al final del job, los de mayor perímetro tendrán el clima “Lluvia intensa”.

Todo esto se guarda en la base de datos en _repository.Pronostico.Save();

Arquitectura

La arquitectura del proyecto está hecha en capas, desacoplando lo que son los endpoints en los Controllers de la WebApi, los servicios y el repositorio.



El proyecto SistemaSolar es la Api donde se definen los endpoints y arranca la ejecución. Además aquí es donde se registran los contratos y repositorios, logging, contexto de DB, etc, para la inyección de dependencias. También se encuentra el appsettings.json con la configuración del proyecto y el app.yaml con los datos necesarios para la publicación en AppEngine.

Contracts almacena las interfaces de todo el sistema para definir la declaración de los servicios y repositorios. Services es el proyecto donde se ejecuta la lógica del job, pronóstico y los planetas, es usada por los controllers, y a su vez se conecta con el repositorio.

Dentro de Entities se encuentra el modelo de clases y las entidades auxiliares para todo el sistema.

También he decidido utilizar esta arquitectura porque me es flexible para escalar y desacoplar los distintos procesos. Además permite que la aplicación sea sencilla de entender y modificar en caso de ser necesario.

He agregado un proyecto de Test que está en progreso.

Visto el modelo de datos y la arquitectura, se puede observar que la aplicación está preparada para escalar e incluso trabajar en paralelo, aunque el ejercicio se pudiera resolver dentro de un solo proyecto y con acceso a base de datos mediante queries hardcodeadas o simplemente almacenando los datos en memoria. Además el sistema soporta correr el job con más cantidad de años de las que dice el ejercicio y también más planetas (no para los casos de lluvia si son más de 3 planetas).

Endpoints

La aplicación utiliza Swagger para mostrar y documentar los endpoints, con lo cual se puede ver parámetros y responses desde la url inicial y a su vez ir probando los endpoints.

Cada uno describe que hace y que devuelve, y responden a los requerimientos del ejercicio. Además he agregado uno para correr el Job [POST] de manera manual, en caso de que se quisiere.

Link a la aplicación

https://sistema-solar-dot-sistema-solar-1992.appspot.com

Link al repositorio de Github

He utilizado el branch develop para desarrollo y master para la versión “prod”.

https://github.com/fbustos/sistema-solar