Introducción

TP1

TP2

TP3

En líneas generales el trabajo se encuentra muy bien organizado y con las responsabilidades bien divididas. Se mantiene correctamente la parametrización en F a lo largo del código y se especializa en el main con el uso de `IOApp`. Pensamos que esto trae dos ventajas, en primer lugar evita el uso de `unsafeRunSync()` para materializar los diversos resultados y a su vez permitiría cambiar `IO` por otra mónada como por ejemplo `Task` de la biblioteca `ZIO` sin modificar varias partes del código.

Luego en cuanto al transactor utilizado consideramos que está implementado acorde a lo esperado ya que se hace buen uso de los recursos con el pool de conexiones Hikari. Además se abre la conexión en un único lugar y no con cada transacción. Por otro lado pensamos que está bien abstraido ya que usa `Resource` de la librería `cats` lo que delega en la misma la responsabilidad de manejar los recursos.

La implementación del trait `Cache` la encontramos muy similar a la de nuestro trait `Repository`. Creemos que se logran encapsular eficientemente las transacciones y a nuestro criterio también ayuda a la testeabilidad del código ya que permite mockear la funcionalidad de acceso a la base de datos y sus querys.

En cuanto al trait `Scorer` consideramos que está bueno que hayan usado la mónada `Sync` ya que notaron que no se requería una mónada más compleja para dicha implementación. Podemos notar en la línea 33 un code smell:

```
Score(Try(resultRecord.get("prediction").toString.toDouble).getOrElse(0.0))
```

Esto es ya que se eligió devolver un valor específico y materializar el resultado en caso de error cuando se podría mapear el resultado de la mónada `Try`.

Adicionalmente en la línea 17 consideramos que no se maneja el error si el archivo no existe:

```
  val evaluator: ModelEvaluator[_] = new LoadingModelEvaluatorBuilder()
                                     .load(new File(modelPathname))
                                     .build
```

Luego en `ScoreService` no vemos un manejo explícito del estado de la caché ya que los métodos `getScoreFromCache` y `saveScoreInCache` están acoplados. A nosotros nos pasó lo mismo y pensamos que podríamos haber usado la mónada `State` o `Writer`.

En `Fpfiuba43Server` el archivo pmml no es configurable ya que está atado a tener el nombre `model.pmml` y al current workdir.

Finalmente, nos pareció muy bueno el uso de la librería `scalaMock` para testear el servicio sin conectarse a la base de datos ya que de esta manera se logra testear unitariamente el mismo.