I used C# with the .NET Core 2.0 Framework to implement this solution.

I implemented a static class called ManifestParser (ManifestParser.cs) that reads the manifest files and parses out the data to hydrate a collection of Manifest Records. The static method ParseManifestFile is responsible for reading in each line of the file and parsing out the contents to hydrate a ManifestRecord object. The ParseManifestFile method returns a collection of Manifest records containing the stadium manifest data.

The ManifestRecord (ManifestRecords.cs) is a model object that represents the information within our domain such as the section Id, section name, rowed, row and name. I added a property called IsSuite to represent the scenario where some sections are not subdivided into rows such as suites.

My implementation of the readManifest() method on the Normalizer class invokes the method ParseManifest on the static ManifestParser class. I also added a private member field called \_ManifestRecords an in-memory cache to access the stadium manifest data. Please note that I could have also used a dictionary where the sectionId would have been my key and a list of ManifestRecord objects for that corresponding section Id would have been my value.

My implementation of the Normalizer method attempted to match the provided section with the section name from the manifest utilizing string comparisons. I implemented a method called LookUpSectionName that takes in the provided section parameter and attempts to parse it out to a corresponding section name in the stadium manifest based on the numeric digits in the provided section description. If there were no matches, I would invalidate the section input, however, if there were corresponding matches, I would then attempt to retrieve the section ID.

I proceeded to look for the section id based on a direct string comparison between the section description in the manifest with the return value from the LookUpSectionName method. I also added another comparison check utilizing a string contains to check for a specific partition in the manifest section description with the provided section description. If this comparison was utilized (i.e. bContainsComparision was set to true) an additional step was utilized utilizing a regular expression and incorporating the provided row description was incorporated in an attempt to find the Section Id.

Once I found the section Id, I proceeded to retrieve all the manifest records for that particular sectionId. I then filtered these results using a string comparison between the provided row information with the row name from the manifest in an attempt to find the row Id. If there was a matching row Id, I would validate the record and populate the Normalization result with the found section id and row id.