Design Rationale:

From the given requirements, I separated out the actors:

1. TCP Clients
2. TCP Server that spawns a new thread for each client to keep accepting new clients while serving existing ones. – Class **PackagingIndexServer**
3. Worker thread that handles client’s request. – Class **IndexingServiceProvider**
4. A data store to perform CRUD operations. – Class **PackageIndexDataStore**

Designing client is not the scope of the project (created for test purposes).

To separate business/logic from the worker thread, designed a new interface **IInputHandler**. Similarly, to abstract data store and provide an API interface, designed **IPackageIndexer**.

**IInputHandler** exposes methods to

1. get parsed input from raw request and also to validate it.
2. Execute the command from the parsed input.

**IPackageIndexer** exposes methods to INDEX, QUERY and REMOVE packages from data store while abstracting the data structures used and internal working of each command.

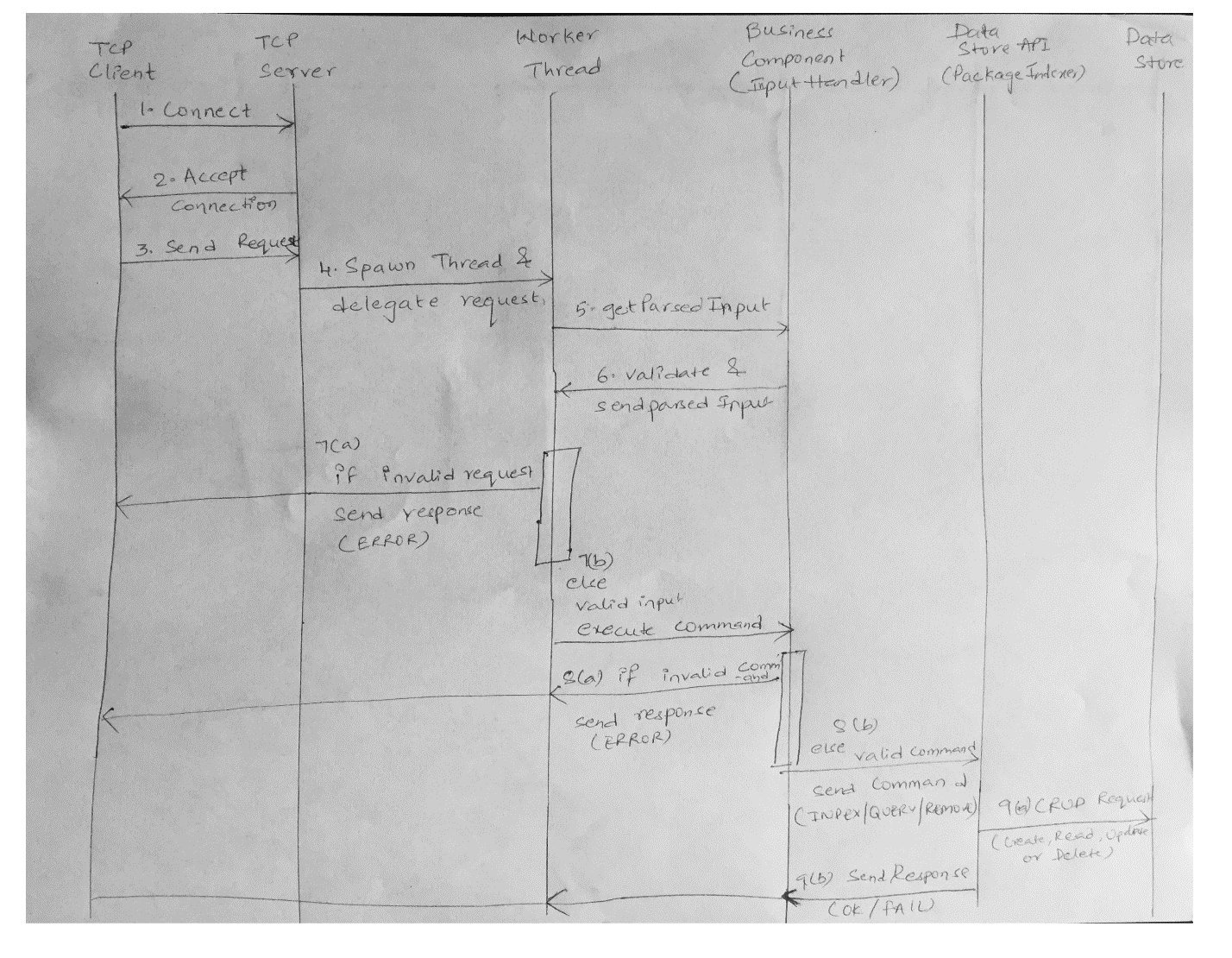
Selection of Data Structures:

There are 2 aspects of data

1. Store for package names: Data structure ideally should be having unique values which meant Set is a good interface.
2. Store for storing the dependencies: Map is ideal to store dependencies as a key value pair.

It also provides good response times for the operations being performed that is put, remove and get.

Using map will avoid duplicating of package names in Set and Map. As this data store was being accessed by multiple clients synchronization had to be provided for performing write operations and is handled in PackageIndexer class.



Future scope:

Data store can be serialized/stored in a database to back up and restore the current index if any updates to the app are needed.