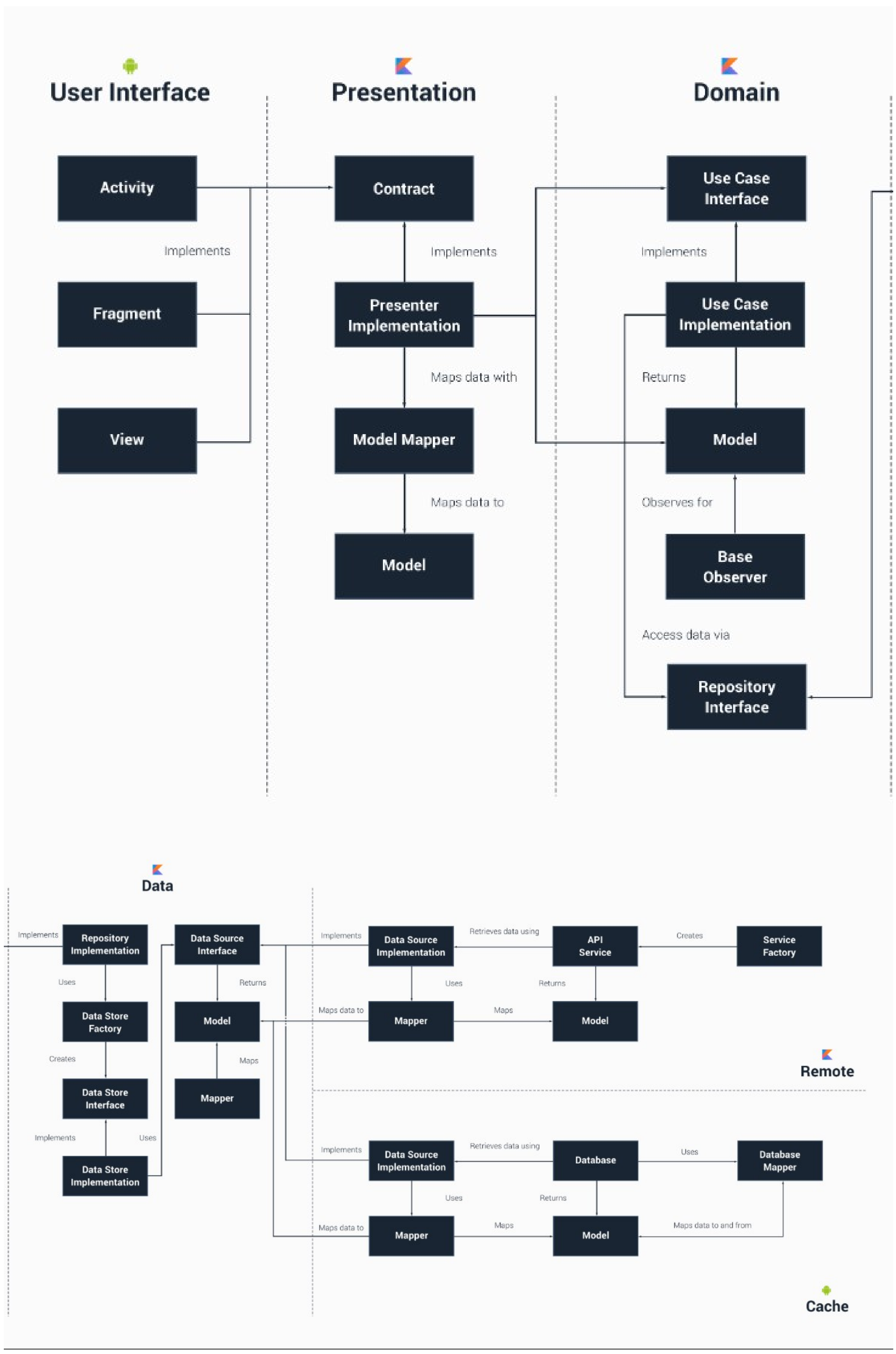
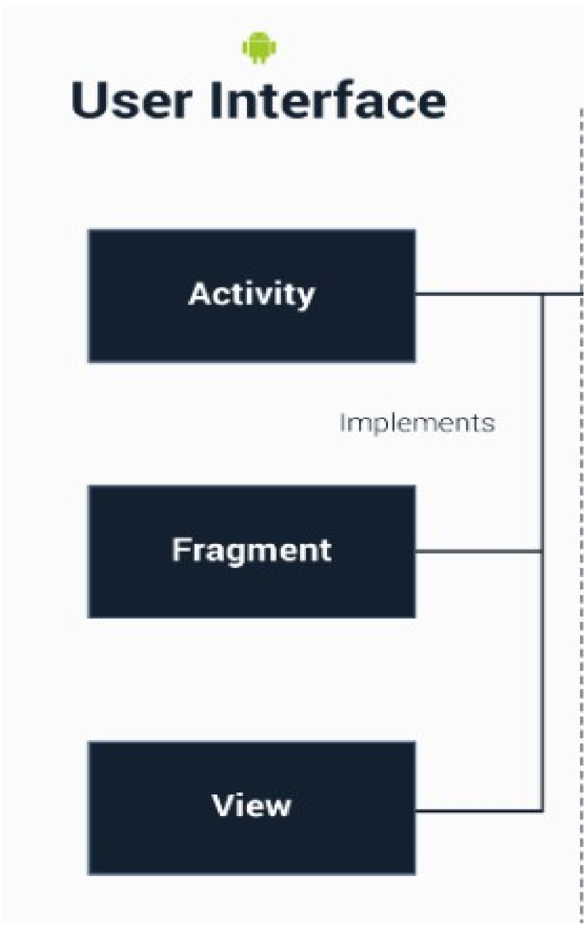
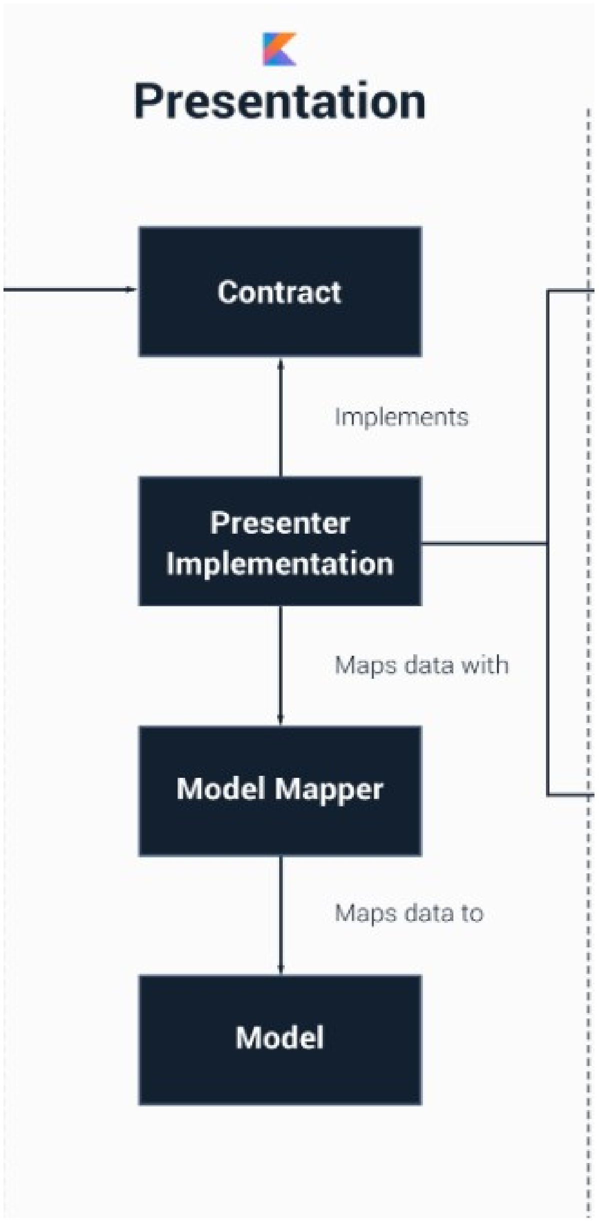

User Interface
Presentation
Domain
Activity
Fragment
View
Implements
Contract
Implements
Presenter Implementation
Maps data with
Model Mapper
Maps data to
Model
Use Case Interface
Implements
Use Case Implementation
Returns
Model
Observes for
Base Observer
Access data via
Repository Interface
Data
Implements
Repository Implementation
Uses
Data Store Factory
Creates
Data Store Interface
Implements
Uses
Data Store Implementation
Data Source Interface
Returns
Model
Maps
Mapper
Implements
Data Source Implementation
Retrieves data using
API Service
Creates
Service Factory
Uses
Returns
Maps data to
Mapper
Maps
Model
Remote
Implements
Data Source Implementation
Retrieves data using
Database
Uses
Database Mapper
Uses
Returns
Maps data to
Mapper
Maps
Model
Maps data to and from
Cache



User Interface

This layer makes use of the Android Framework and is used to create all of our UI components to display inside of the Browse Activity. The layer receives its data from the Presentation layer and when retrieved, the received models are mapped using the Bufferoo Mapper so that the model can be mapped to this layer's interpretation of the Bufferoo instance, which is the BufferooViewModel. The Activity makes use of the BrowseContract to enable communication to and from the presenter.



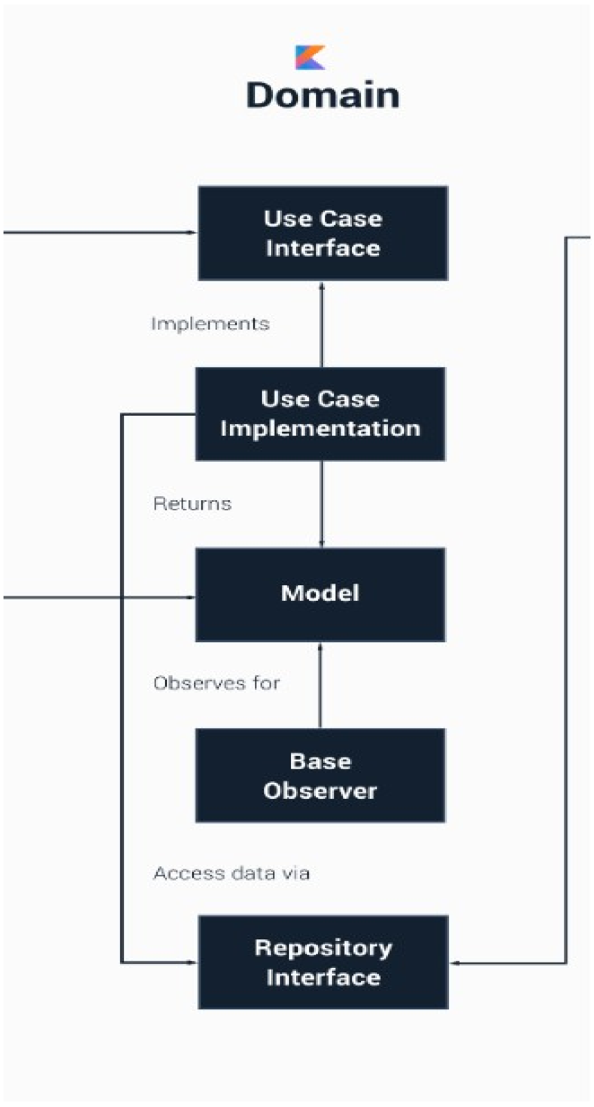
Presentation

This layer's responsibility is to handle the presentation of the User Interface, but at the same time knows nothing about the user interface itself. This layer has no dependence on the Android Framework, it is a pure Kotlin module. Each Presenter class that is created implements the Presenter interface defined within an instance of a contract - in this case the BrowseContract, which also contains an interface for the View interface.

When a Presenter is constructed, an instance of this View is passed in. This view is then used and the presenter is set for it using the implemented setPresenter() call.

The presenters use an instance of a SingleUseCase from the Domain layer to retrieve data. Note here that there is no direct name reference to the UseCase that we are using - we do inject an instance of the GetBufferoos UseCase, however.

The presenter receives data from the Domain layer in the form of a Bufferoo. These instances are mapped to instance of this layers model, which is a BufferooView using the BufferooMapper.

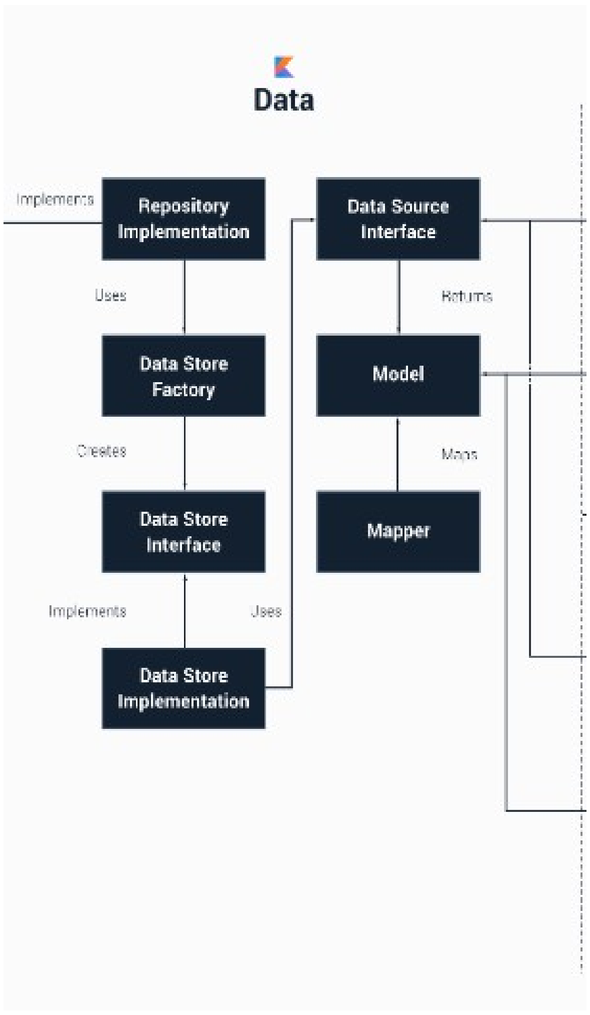


Domain

The domain layer responsibility is to simply contain the UseCase instance used to retrieve data from the Data layer and pass it onto the Presentation layer. In our case, we define a GetBufferoos - this use case handles the subscribing and observing of our request for data from the BufferooRepository interface. This UseCase extends the SingleUseCase base class - therefore we can reference it from outer layers and avoid a direct reference to a specific implementation.

The layer defines the Bufferoo class but no mapper. This is because the Domain layer is our central layer, it knows nothing of the layers outside of it so has no need to map data to any other type of model.

The Domain layer defines the BufferooRepository interface which provides a set of methods for an external layer to implement as the UseCase classes use the interface when requesting data.

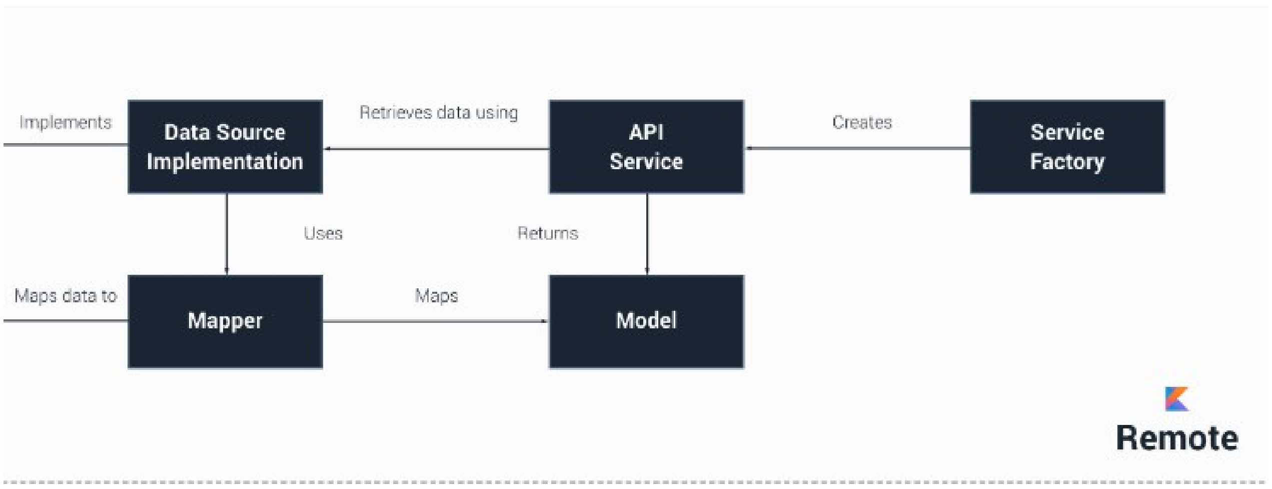


Data

The Data layer is our access point to external data layers and is used to fetch data from multiple sources (the cache and network in our case). It contains an implementation of the BufferooRepository, which is the BufferooDataRepository. To begin with, this class uses the BufferooDataStoreFactory to decide which data store class will be used when fetching data - this will be either the BufferooRemoteDataStore or the BufferooCacheDataStore - both of these classes implement the BufferooDataStore repository so that our DataStore classes are enforced.

Each of these DataStore classes also references a corresponding BufferooCache and BufferooRemote interface, which is used when requesting data from an external data source module.

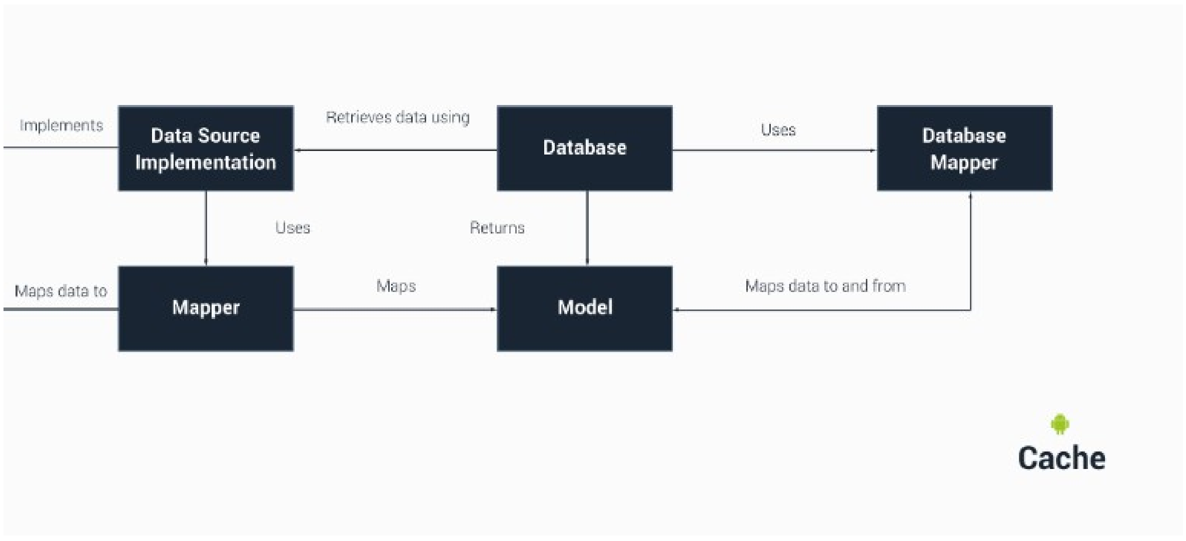
This layers data model is the BufferooEntity. Here the BufferooMapper is used to map data to and from a Bufferoo instance from the domain layer and BufferooEntity instance from this layer as required.



Remote

The Remote layer handles all communications with remote sources, in our case it makes a simple API call using a Retrofit interface. The BufferooRemoteImpl class implements the BufferooRemote interface from the Data layer and uses the BufferooService to retrieve data from the API.

The API returns us instances of a BufferooModel and these are mapped to BufferooEntity instance from the Data layer using the BufferooEntityMapper class.



Cache

The Cache layer handles all communication with the local database which is used to cache data.

The data model for this layer is the CachedBufferoo and this is mapped to and from a BufferooEntity instance from the Data layer using the BufferooEntityMapper class.