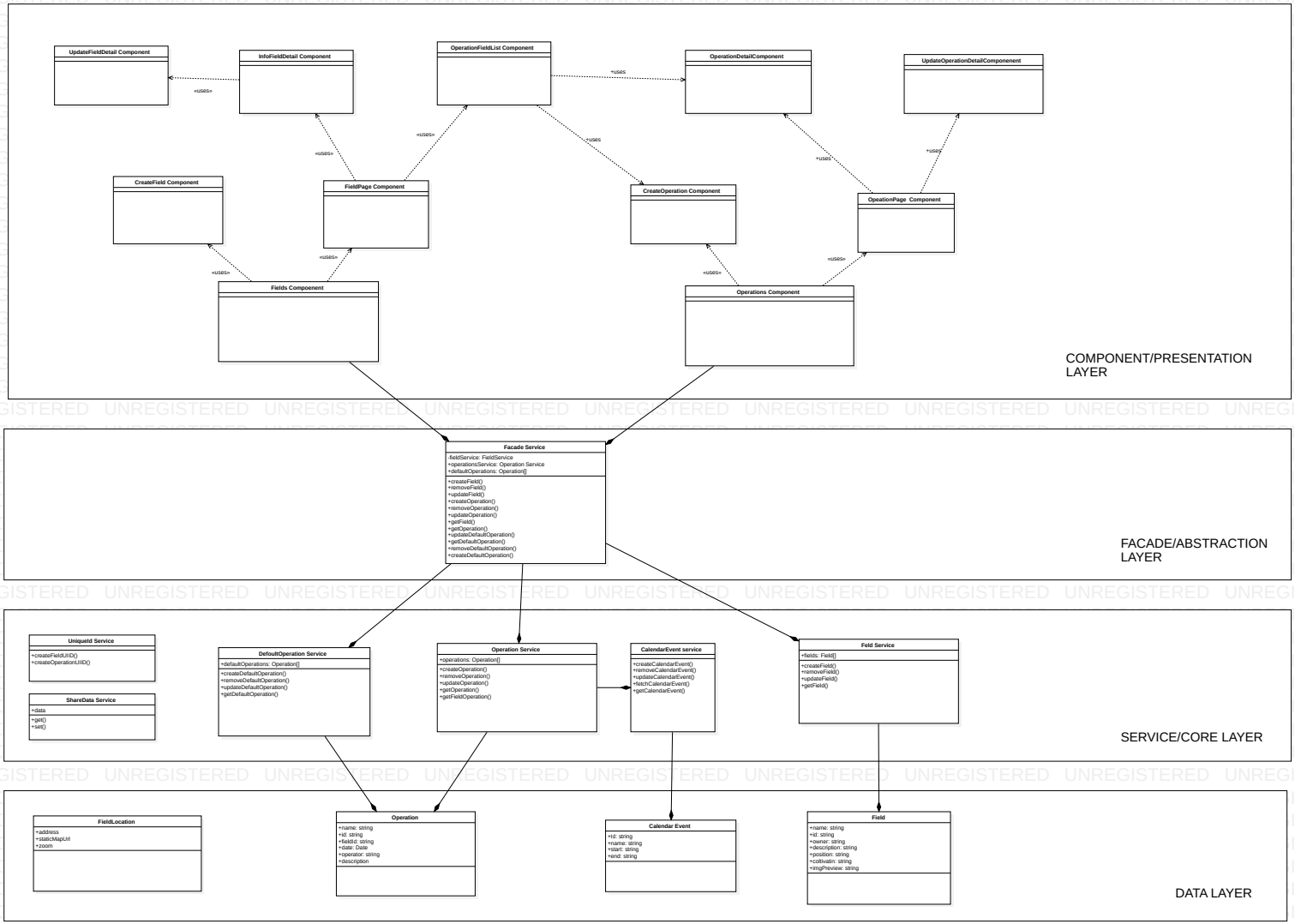
UNREGISTERED



This is the place where all our Angular components live. The only responsibilities of this layer are to present and to delegate. In other words, it presents the UI and delegates user's actions to the core layer, through the abstraction layer. It knows what to display and what to do, but it does not know how the user's interactions should be handled.

The abstraction layer decouples the presentation layer from the core layer and also has it's very own defined responsibilities. This layer exposes the streams of state and interface for the components in the presentation layer, playing the role of the facade. This kind of facade sandboxes what components can see and do in the system. When it comes to the state, the abstraction layer makes our components independent of the state management solution. Components are given Observables with data to display on the templates (usually with async pipe) and don't care how and where this data comes from. To manage our state we can pick any state management library that supports RxJS (like NgRx) or simple use BehaviorSubjects to model our state. In the example above we are using state object that internally uses BehaviorSubjects (state object is a part of our core layer). In the case of NgRx, we would dispatch actions for the store.

Here is where core application logic is implemented. All data manipulation and outside world communication happen here. If for state management, we were using a solution like NgRx, here is a place to put our state definition, actions and reducers.