Architecture Design

Our application, ‘Anti-Patrola’ requires hybrid client-server (layered) architecture with unique approach – instead of thin client, we are using fat client, which means that the architecture is also distributed.
Developing an application in Flutter (with Dart) requires this approach.
We chose to go this way, because we plan to release the application as a mobile app (in production, outside of this course), and by creating it in Flutter, we are making sure that it is cross-platform by design.
Also, the architecture is distributed, because the application is community-driven which means – all users report patrols and the state of the data depends on the users. The architecture is also data-centric.

We provided images with conceptual, executional and implementational views of the architecture, but we would like to explain the fat client approach.

By using Flutter (and Dart) we are developing for web and mobile at the same time. At the end, the web part is compiled to HTML, CSS and JavaScript, and the mobile part is compiled to native code.
The client’s architecture is also layered architecture, which contains 3 layers:
1. UI Layer – only the Flutter’s widgets (and OpenStreetMap library’s widgets). This helps us to design responsive and reactive UI, that is not tightly-coupled with the business logic and the data layer.
2. Business Logic Layer – separates the Flutter Widgets from the data. We are using BLOC Pattern (Business Logic Component pattern) as a middle-man between the UI and the Data layer. Also, here we have services that are needed (as Authentication Service, Geolocation Service, …).
We also use Singleton pattern for dependency injection.
3. Data/Network Layer – transforming JSON to DTOs (and vice-versa) and making REST Calls to the backend.
Additionally, the bottom components don’t know anything about the top components (which they are servicing).