Knowist Live Academy

https://www.knowist.ac

Angular Reference App Architecture

With Angular, what runs in the browser become much larger and powerful that previously, which largely depended on server-side coding, and so we need to put some effort into architectural considerations. Our application reference architecture is based on the idea of software semantic models, with each semantic model type looking at the application from a distinct viewpoint.

For the System Model, we focus on layering, and in particular how the coordination layer acts to combine capabilites. We also look at the app shell (app.component.ts) which is the root component.

For the Feature Model, we explore how the application functionality users require can be delivered, and how additional features could be added in future without disturbing existing features.

For the Domain Model (yes, we are great fans of the book by Eric Evans), topics covered include:

- Ubiquitous Language
- Entities and Value Objects
- Aggregates (and the aggregate root)
- Bounded Contexts (and how these might map to distinct microservices)
- Software Layering
- Domain Services
- Core Domain

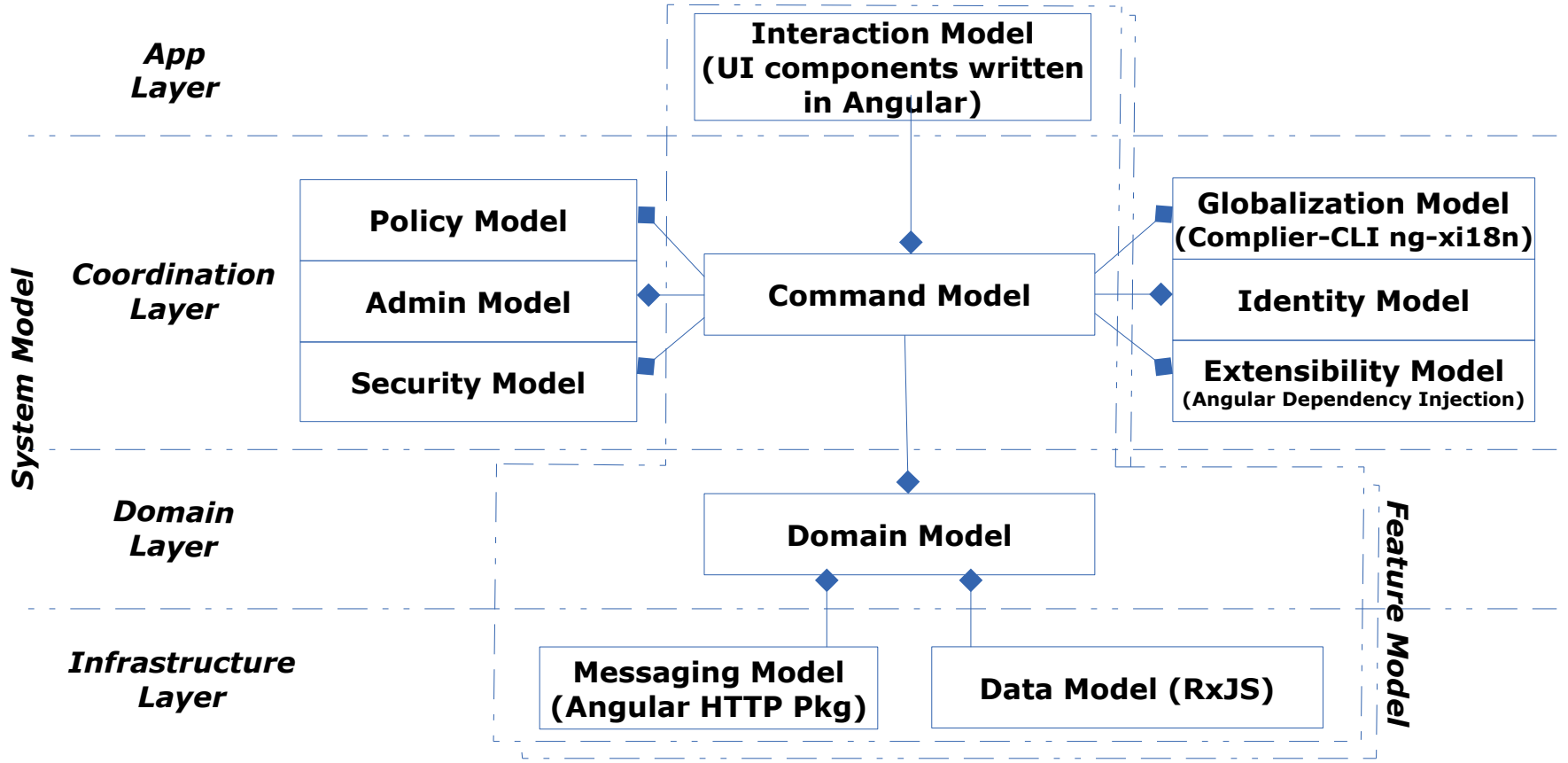
For the Extensibility Model, we show how Angular's powerful dependency injection system can help compose larger client-side applications.

For the Data Model, we show how RxJS can be used to stream incoming data into an observable and process its results.

For the Messaging Model, we use the Angular HTTP client package to call a REST API exposed by a remote server is a managed manner (We like the use of Open API/Swagger).

For the Interaction Model, we build a rich user experience, with non-UI logic imported from other models.

We have a number of additional helper models for application services shared among multiple features.



NOTES:

• The domain model is the base library and is imported by the other models (think of it as the filling of a sandwich, rather than a traditional layer in a hierarchy). • Two implementations of the Messaging Model are provided, one for unit tests (in-memory-web-api), and the other that works against a real backend • To allow client progammability, an SDK (not shown) could also be provided	• Assuming we intend to support multiple App Models, then the Command Model handles common code for commands and queries; it also manages the unit of work (transaction). • In pursuit of separation of concerns, we isolate policy, admin, identity, globalization and possibly other areas • Large domains should be subdivided into bounded contexts, and each should run against a separate backend microservice