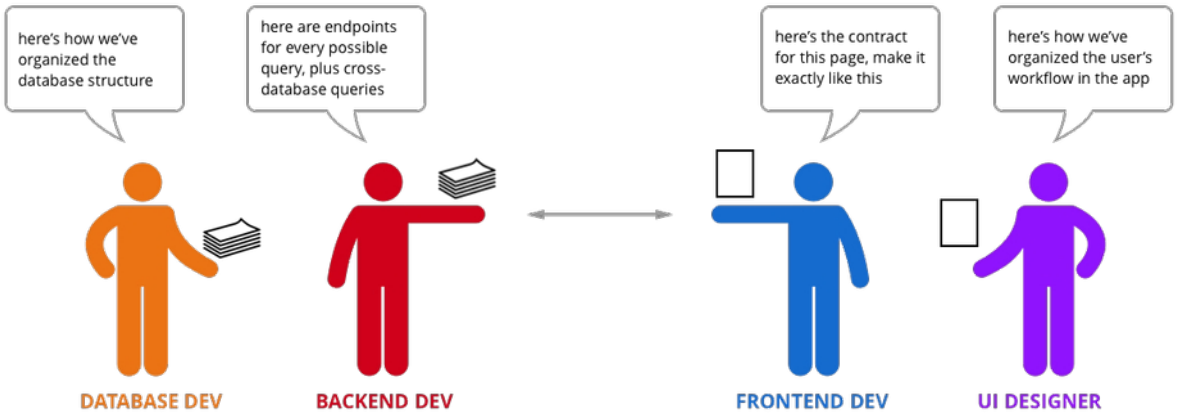
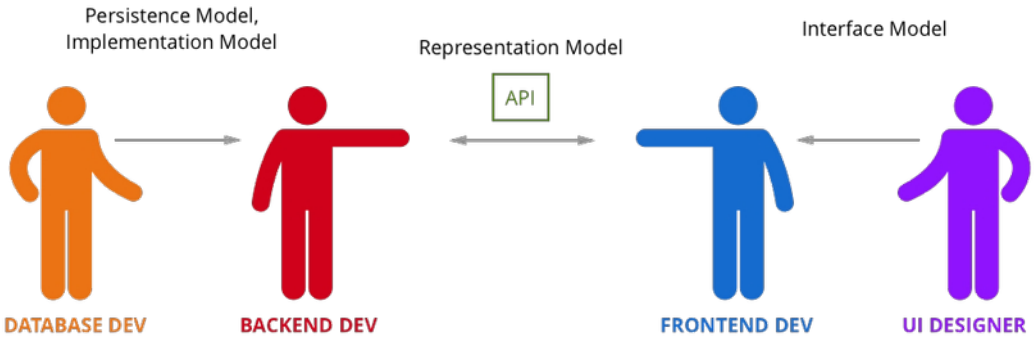
Applying design thinking to APIs

You may associate 'design' with visual interfaces, but an API is also a type of user interface. While the person using a visual interface may not be the same person using the API to build a visual interface, they're both users. If we leave the user out of the picture, the API's model is driven by backend devs and frontend devs. These two groups have different priorities, as well as constituents with priorities of their own.

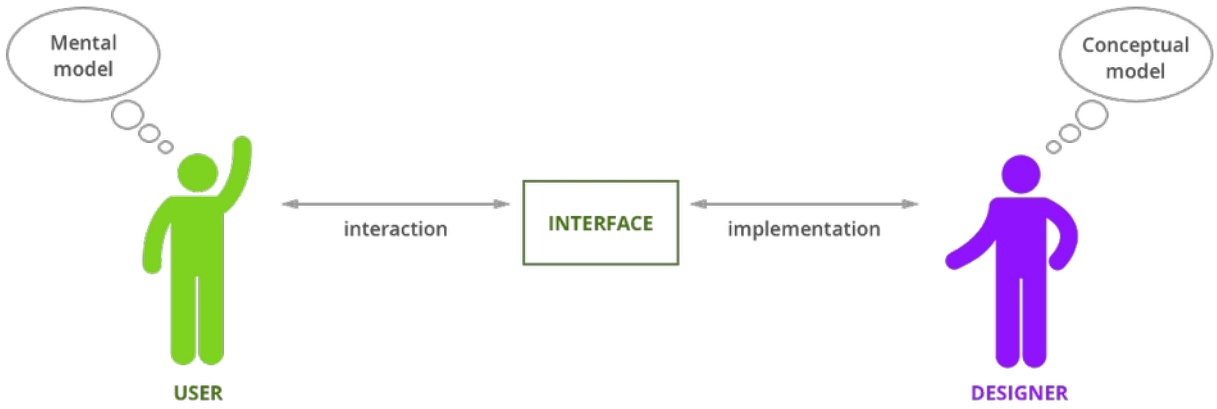


The approaches differ because each person has a different underlying model.



The backend's easiest default is to replicate the database structure: either a persistence model (simple queries). an implementation model (cross-database or logic-added queries), or both. The frontend's easiest path is following the visual design: an interface model. Simply replicating the database is too fine-grained for efficient frontend use, and aligning too closely to the UI reduces versatility and flexibility for alternate uses. That means the API -- as an intermediary -- requires its own model.

The foundation of the API's discrete model should be the user's understanding of the interaction.



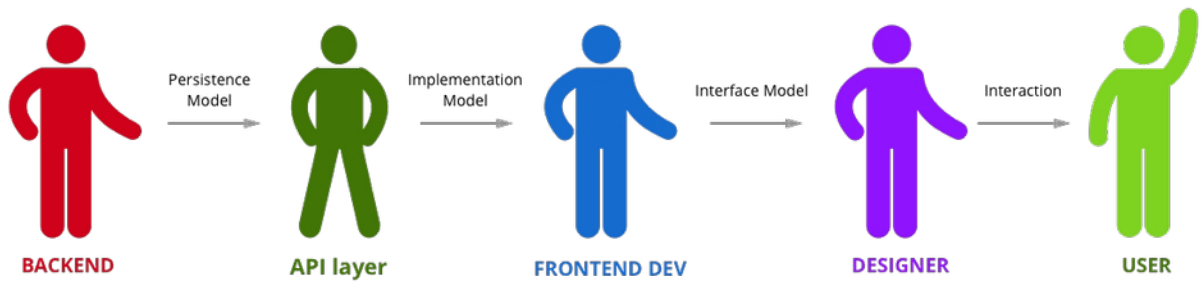
Users make assumptions and plan tasks based on their mental model. If an API's model doesn't take user assumptions into consideration, users will find the interface, product, or system hard to learn hard to use, and abandonment rates will spike sharply.

(include classic thermostat example?)

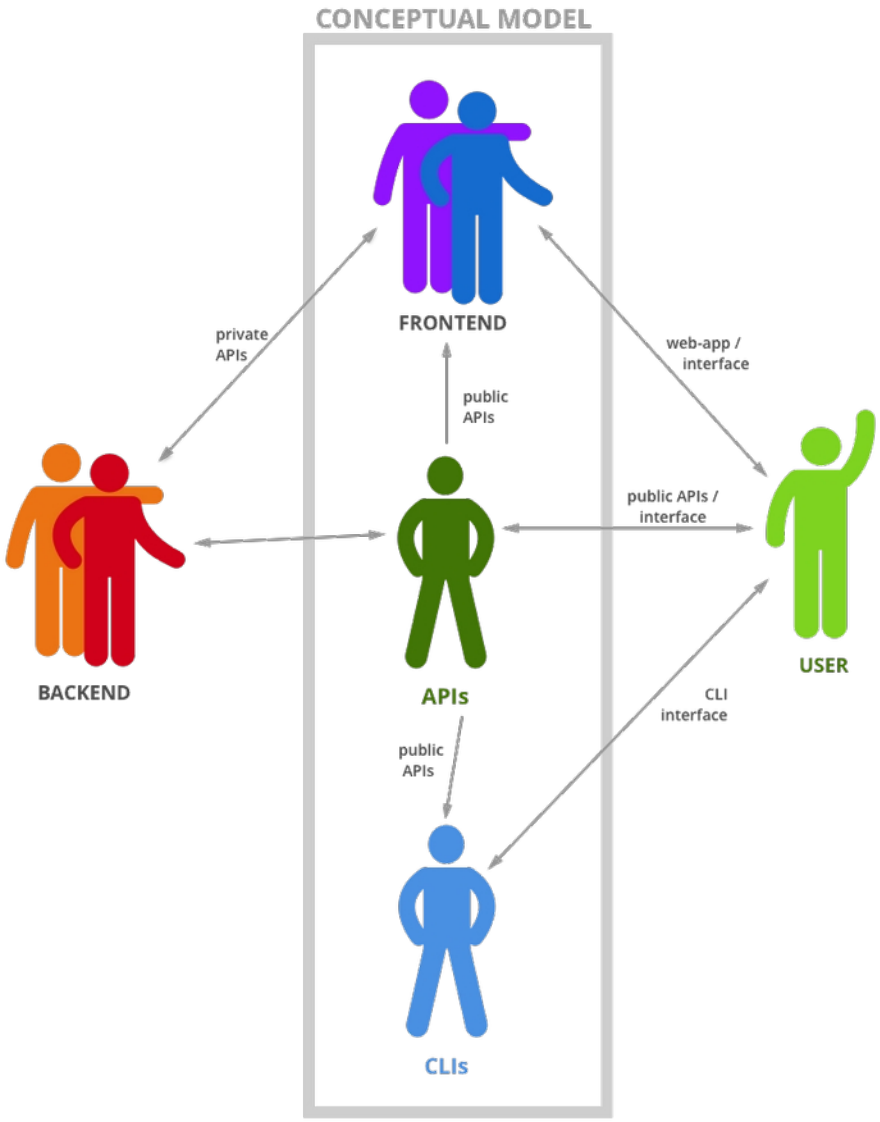
There will be times that the conceptual model will deviate from the user's mental model, such as versioning, or the introduction of new paradigms that buck convention. These will require additional in-depth user testing, given the risks in disconnecting the mental model and the conceptual model.

What does this have to do with APIs?

If you're thinking this is more of a designer's concern than a developer's, it's probably because you're used to a linear process. Data begins at the backend, is integrated in some fashion in an API layer, which is consumed by the frontend, which is in turn styled by the designers, and finally handed to the user.



At best, the designer is the only one taking the user into consideration, and the user will experience cognitive friction in any interface beyond the designer-influenced UI. But when all user-facing interactions are informed by a shared conceptual model, this risk is alleviated.



With the conceptual model as foundation, each type of user interaction will be cleaner, simpler, and easier to understand. It should be focused on the tasks, with just enough information to provide the required functionality.