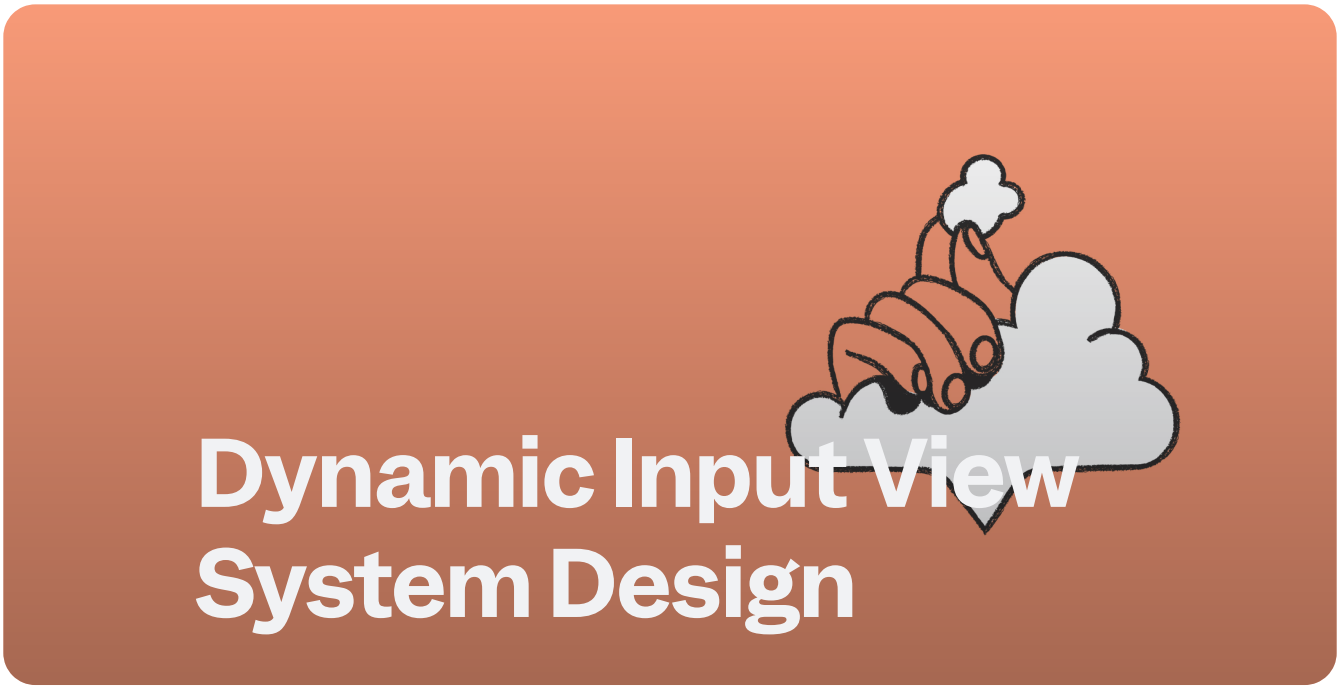


Overview

The Product team wants us to build a flexible, API-configurable system that will allow them to test different input UIs and different input options.

How will it work?

The demo app's UI is divided into two parts, a collection view that displays the items selected, and a table view that allows the user to select their preferred items. Based on the given examples, this design document will be focused on the UI that allows the user to select their preferred items.

Backend API

The API GET requests to need to contain fields that will describe how the iOS will display the UI

componentID - This will indicate which UI component to use. The demo app uses one specific component (ListView), this will allow the backend to specify which id they would like to utilize. (Ex: matrix of buttons, text input with search functionality.)

Returns a JSON collection, which would consist of configurable layout options. (ex: components translation, size, margin size, layout, etc...)

demographicID - This will indicate which demographic collection to use. The demo app uses the Pronoun collection type, this will allow the backend to specify which demographic collection to use. (Ex: sexual orientation, ethnicity, or religion)

Returns a JSON collection, which the iOS app would use to build an Encodable Model for the demographic model.

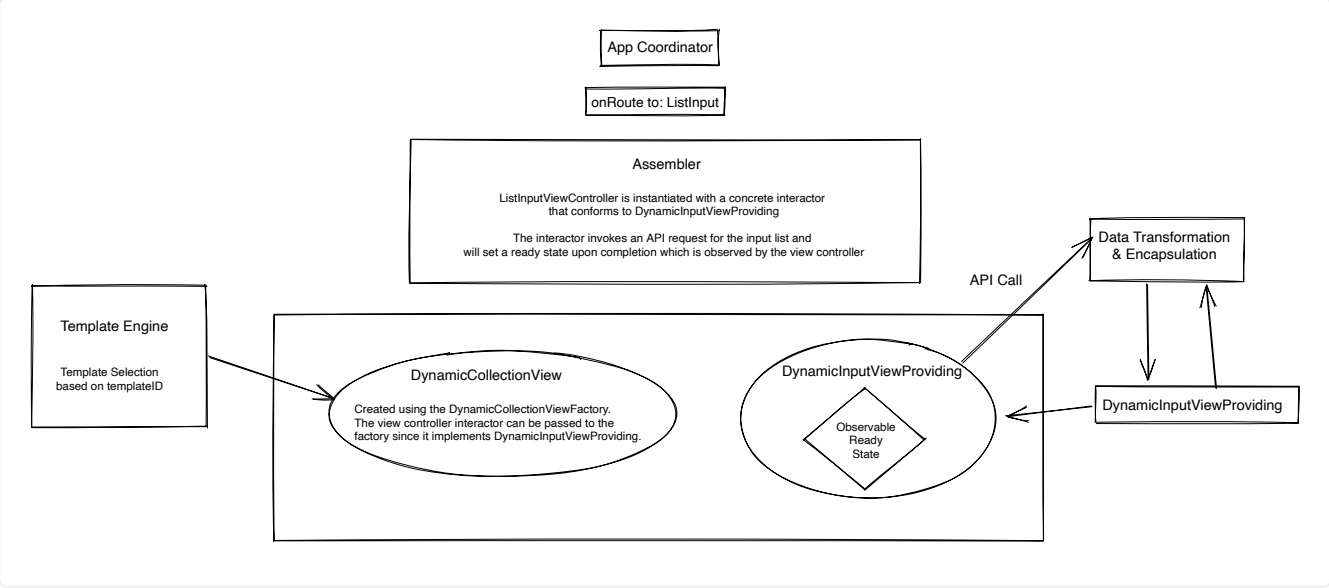
iOS Client

The iOS app will have a similar architecture to the demo app (MVVM) but will be data-driven based on the componentID and the demographicID that are received from the Backend API calls. We will develop a data-driven component that is based on the UICollectionView class instead of the UITableView in the demo, in which the layout and the cells will be configurable based on the data received from the backend API.

A new component (DynamicCollectionView) will be built around the UICollectionView. This component will display the list of items that the user can select from. The DynamicInputViewProviding protocol will be implemented by the new DynamicListInputView class interactor and passed to the new Component for assembly. This will replace the existing UITableView within the demo app.

There will be a factory class that will assemble and instantiate all of the components based on the componentID and the demographicID. If a template is not found, a default will be used to display the content.

```
DynamicCollectionViewFactory.makeDynamicCollectionView( with interactor
) -> DynamicCollectionView
```



Testing

Unit testing, integration testing, and UI testing are important for any app. The architecture of the app will enable a test-driven design approach.