Design Architecture

Menoreh library use the design pattern SOLID Principle.

We split the project into multiple packages in order to maintain explicit dependencies for each package with clear boundaries that enforce the single responsibility principle. Modularizing our project like this has many benefits including but not limited to:

- Easy to reuse packages across multiple projects
- CI/CD improvements in terms of efficiency (run checks on only the code that has changed)
- Easy to maintain the packages in isolation with their dedicated test suites, semantic versioning, and release cycle/cadence

Layering our code is incredibly important and helps us iterate quickly and with confidence. Each layer has a single responsibility and can be used and tested in isolation. This allows us to keep changes contained to a specific layer in order to minimize the impact on the entire application. In addition, layering our application allows us to easily reuse libraries across multiple projects (especially with respect to the data layer).

To use this library you have to understand

- Solid Principle
- Bloc State management
- Get It
- Equatable
- Dartz

Architecture

Architecture

Feature Layer

This layer contains all of the application-specific features and use cases. Each feature generally consists of some UI and business logic. Features should generally be independent of other features so that they can easily be added/removed without impacting the rest of the codebase. Within each feature, the state of the feature along with any business logic is managed by blocs. Blocs interact with zero or more repositories. Blocs react to events and emit states which trigger changes in the UI. Widgets within each

feature should generally only depend on the corresponding bloc and render UI based on the current state. The UI can notify the bloc of user input via events.[1]

```
├── lib
│   ├── core
│   │   ├── extensions
│   │   ├── languages
│   │   ├── themes
│   │   ├── utils
│   │   └── values
│   └── presentation
│       ├── routes
│       ├── widgets
│       ├── bloc
│       └── pages
│           ├── main
│           └── app
├── packages
│   ├── menoreh_domain
│   └── menoreh_data
├── assets
└── test
```

Domain layer

This layer combines one or more data providers and applies "business rules" to the data. Each component in this layer is called a repository and each repository generally manages a single domain. Packages in the repository layer should generally only interact with the data layer.[1]

```
├── lib
│   └── src
│       ├── entities
│       ├── repositories
│       └── usecases
└── test
```

Data layer

This layer is the lowest layer and is responsible for retrieving raw data from external sources such as a databases, APIs, and more. Packages in the data layer generally should not depend on any UI and can be reused and even published on pub.dev as a standalone package.[1]

```
├── lib
│   ├── core
│   │   ├── config
│   │   ├── exception
│   │   ├── failure
│   │   ├── usecase
│   │   └── values
│   ├── helper
```

```
│   ├── service
│   └── src
│       ├── datasource
│       ├── models
│       └── repositories
└── test
```

1. bloclibrary.dev 

Last update: January 5, 2023 17:43:51

Dibuat: January 5, 2023 17:43:51

Authors: