クリーンアーキテクチャーまとめ

概要

- こちらを参考にクリーンアーキテクチャについて学ぼうとした
- 意味がわからなかったので、自身でソースコードを書いてみた
- 今回はクリーンアーキテクチャとはどういうものなのかを最終的に自分の観点からまとめる

クリーンアーキテクチャについて(自分の認識では)

- 依存性の方向性を逆転させ、ビジネスロジックから詳細を分離するアーキテクチャパターン
- 各レイヤーは自身とその接している外のレイヤーに依存する
- 「依存する」という表現は機能を変更した際に影響を及ぼすことをいう
- 利点としては大規模プロジェクトにて役割を分割しているので保守がしやすくなる
- 逆に DB の構成が変わるなどのことが起こると全てのレイヤー修正する場合ありめんどくさくなる
- 小さなプロジェクトではクリーンアーキテクチャーを使っても利点は感じられない

各階層について

`domain`

- ここではルールを中心に記載する
- 基本ここは定義するのみ？
- 例えば、空文字であった場合エラーを返すという部分をここに記載しておく
- 上記例の仕様はほぼほぼ変わることはない
- ここはコア(心臓)の部分である
- 仕様変更とかに寄らない根本的な機能にしておく
- 仕様変更になっても変更しなくて済むことが多い

`applications`の`usecase`

- CRUD(Create,Read,Update,Delete)の機能を作っている
- ここでは`id`の生成であったりバリデーション判定を呼び出したりしている
- また、バリデーション機能にて仕様変更が多い処理場合はこちらで定義することが多い
- `domain`に定義するか、`usecase`に定義するかは仕様次第

`applications`の`Repository`

- ここでは CRUD の定義をしている
- 一つ外の`getways`の`interface`を定義している

`interface`の`getways`

- ここでは D`の操作等を書いている
- `Repository`を interface としている
- DB を変更しても`Repository`に依存したデータの設計であればすぐに取り替えられる

`interface`の`controller`

- mvc モデルの c の部分のコントローラー
- クライアントから受け取ったデータ等を受け取ったり返したりする部分を実装している

`interface`の`presenters`

- ここでは外部で扱いやすいデータの形に変換している
- 実際はもっと複雑なことをしている？
- ここはまだ定義がいまいち分かっていません

`infrastructure`

- ここでは DB の詳細設定やフレームワークを格納する
- 主に`controller`を呼び出してレスポンスを返す役割をしている

その他

- クリーンアーキテクチャにて実装する場合はきちんと意味なくても各レイヤーを設置しておく
- 仕様変更になった場合に初めて恩恵が受けられるらしい
- ぶっちゃけ実装しまくって覚えていくというくらい複雑なアーキテクチャ