Arquitectura de Software CMI (Config - Core - State Management - User Interface)
V 0.0.1

Te imaginas que sería del desarrollo de aplicaciones si no existieran las arquitecturas de Software, patrones y demás que hoy en día son importantes?, no hay que imaginarlo solo debemos ver al pasado en como era antes de la existencia de estos, el software era desarrollado de una forma libre pero esta llevaba a futuros problemas cuando se requería implementar una lógica más compleja, era difícil poder realizar cambios y una serie de problemas que no quisieras ni ver hoy en dia, el nacimiento de las arquitecturas de Software lo cambio todo, CMI es una arquitectura que toma como base la estructura **CMC**, haciendo que cada parte sea independiente de la otra, facilitando que se pueda reemplazar sin afectar del todo al resto, aun se puede llegar a un nivel de afectación nulo, se considero importante tener un bajo nivel de acoplamiento.

Filisofia:

- **Estructura:** Cuando se tiene las cosas bien estructuradas sabremos siempre donde estan todos los fractmentos que lo componen.
- **Clasificación:** Los fractmentos nunca deben estar dispersos, siempre tiene sus fractmentos clasificados y ordenados por su relación o como dicen "Un sitio para cada cosa y cada cosa en su sitio", Si no es asi debe refactorizar hasta cumplir con lo antes mencionado.
- **Stories:** No convierta su trabajo en una historia, aun que si es un escritor eso puede que este bien, es fudamental mantener nuestro trabajo los mas limpio posible y evitar

llenar de historis innecesarias, ¿Comentar nuestro espacio de trabajo es malo?, Podriamos dejar como un depende, siempre existen excepciones.

- **Unificación:** El equipo debe hablar en un solo idioma, o tendremos equipos donde no se entienden nada, si sigue juntos es muy seguro que terminaran separándose, para evitar este posible final se debe establecer una norma de trabajo para unificar a todo el equipo y a sus futuros miembros.

Principios:

- **Rollback:** Permite que todo cambio sea reversible a su estado anterior.
- **Adaptable:** permite adaptarse a los cambios continuos al utilizar la estructura CMC.
- **Simplicidad:** todo debe debe ser simple y claro, aplicando el principio de responsabilidad única.
- **Protegido:** Debe estar protegido para evitar modificaciones pero debe poder extenderse aplicando el principio **open/closed**.
- **Reemplazable:** La arquitectura CMI permite reemplazar cual parte de la estructura CMC, aplicando el principio de sustitución de liskov.
- **Fácil:** La UI debe ser simple de usar para los usuarios, nunca se debe sobrecarga una **Screen/Pages o una vista** con funciones innecesarias para el usuario, aun que siempre es mejor separar en más de una **Screen/Pages o vista** , no siempre es necesario que todo este en un mismo lugar.
- **Aislamiento:** Es importante tener todo aislado de

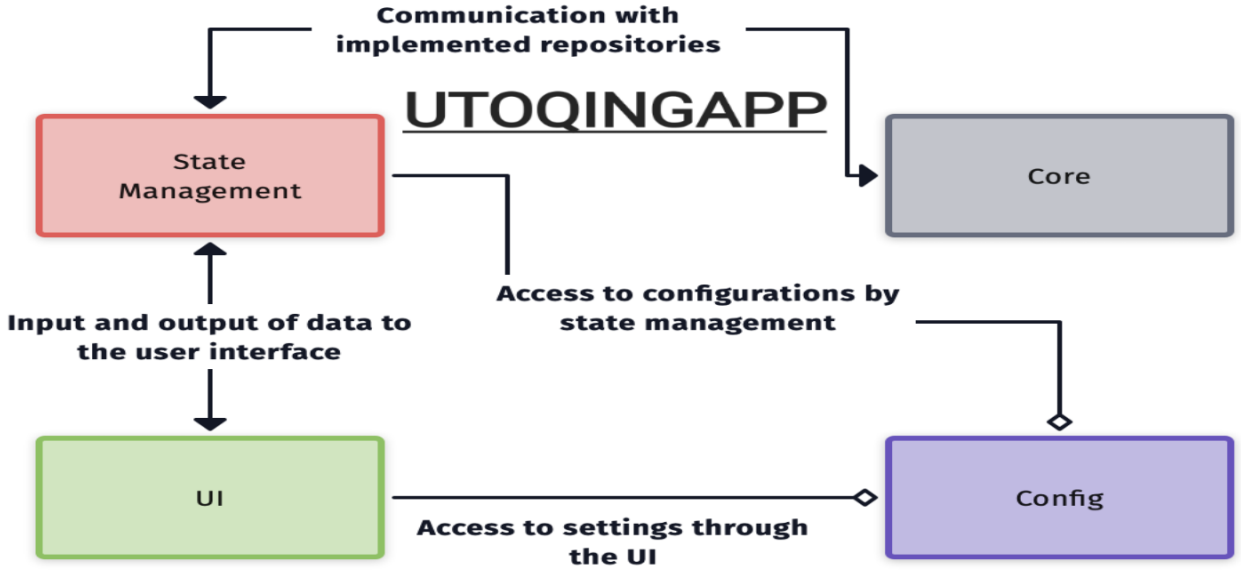
dependencias externas, se debe poder agregar o quitar dependencias en cualquier momento sin afectar a todo, pudiendo sólo cambiar una parte pequeña.

Definición de la estructura CMC:

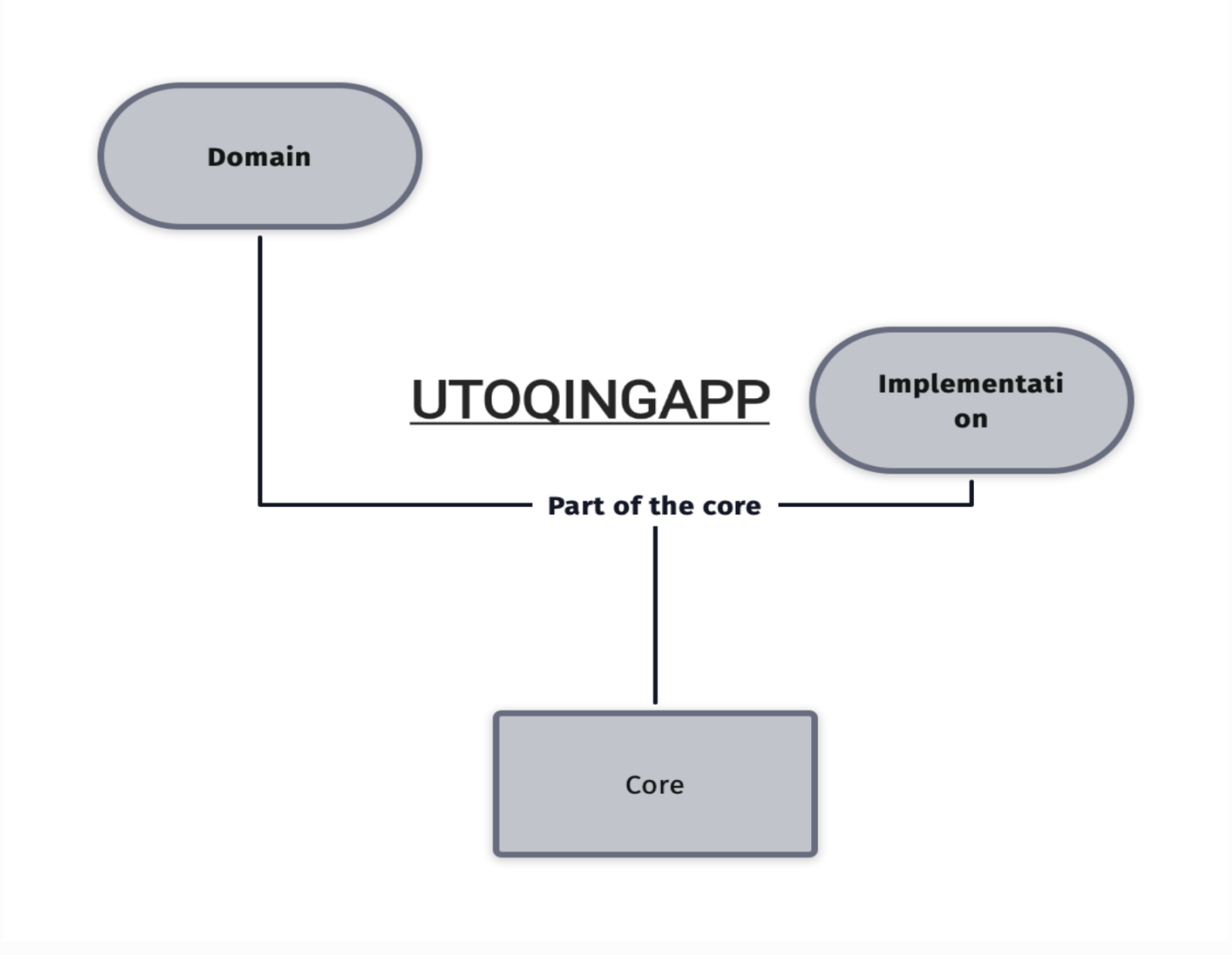
- **Capa:** Conjunto de **modulos** que cumple una función en común, no abuse de tener demasiadas **capas**.
- **Modulo:** Conjunto de **contenedores** que cumplen la función de un **modulo**.
- **Contenedor:** Son fractmentos que en conjunto forman un **módulo** o **contenedor**.

Importante: Evite tener demasiados niveles innecesarios, eso convierte el lugar de trabajo en un basurero.

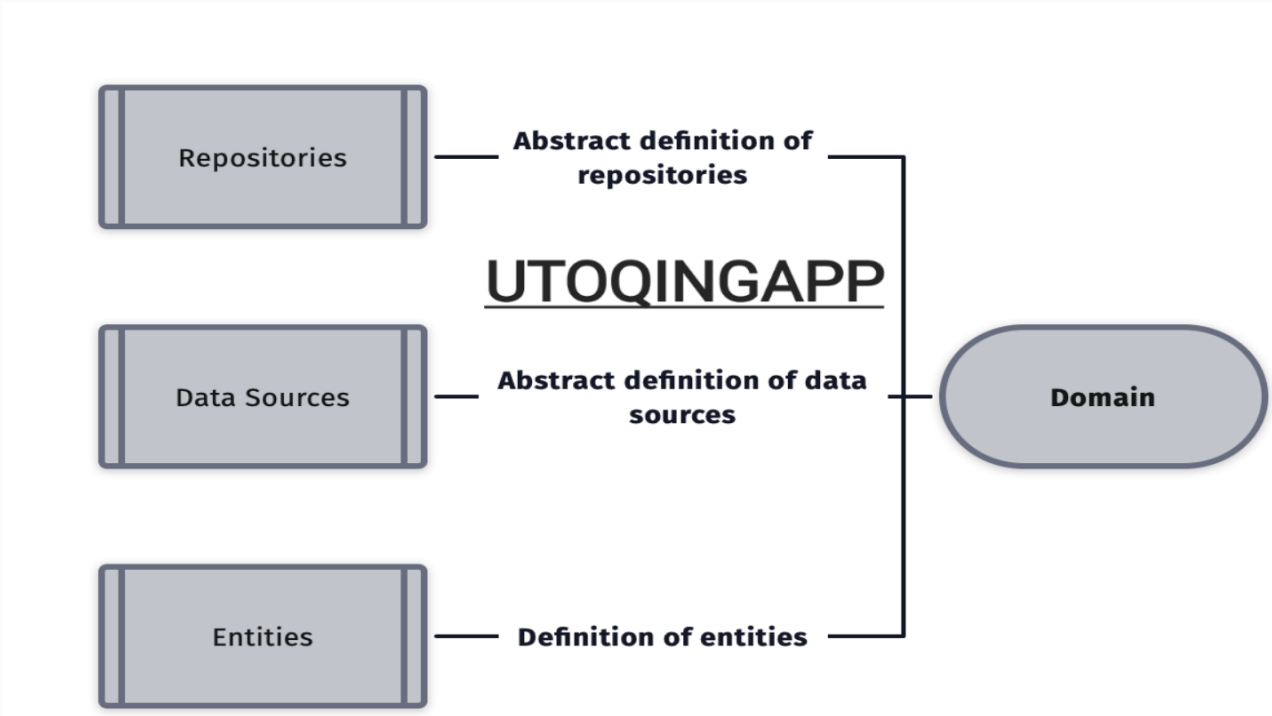
Definición de la arquitectura CMI:



- **Config:** Es la **capa** de configuración en donde se establecen módulos como Router,theme, constans, inputs, enviroment, notifications, languages, Helpers, Colors y de más configuraciones, las mencionadas son solo un ejemplo, puede ser usada por toda las capas menos por el **módulo domain** de la **capa Core**.
- **Core:** Es la **capa** que agrupa los **modulos domain y implementation**, son dos módulos importantes para el acceso a las **fuentes de datos**.



- **Domain:** Es un **módulo abstracto** que define las reglas del espacio de trabajo más no las implementa a excepción de las **entidades**.

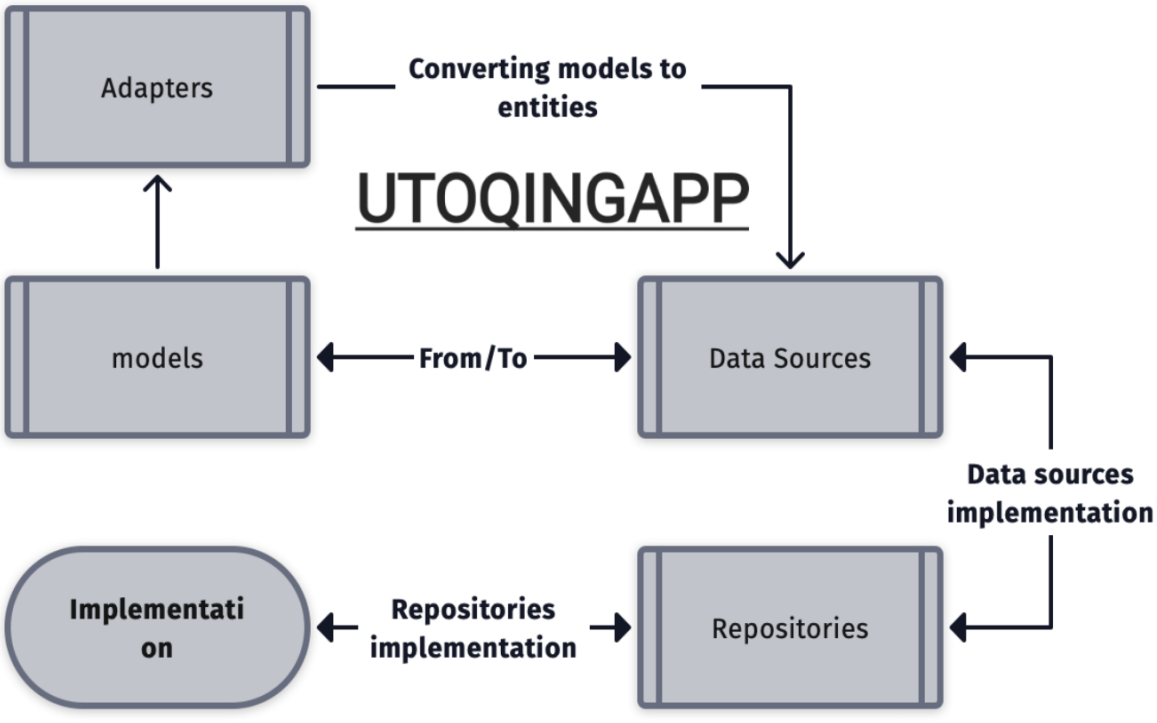


- **Data Sources:** El contenedor de **Data Sources** son una definición **abstracta** de las fuentes de datos, al definir de esta forma indicamos como queremos los datos.
- **Entities:** El contenedor de las **entidades** contiene las **clases sin metodos** que almacenan los datos transformados y son consumidas por el **módulo Implementation en su contenedor de Data Sources**, también pueden ser referenciadas en las capas de **Manager State y UI**.
- **Repositories:** Definición **abstracta** del contenedor de los repositorios que son consumidos por el **módulo Implementation** en su contenedor de los Data Sources, al definir de esta forma indicamos que **Data Sources** queremos usar.

Inportante: el módulo **Domain** debe estar compuesto de

puramente **código propio del lenguaje** es decir no debe depender de nada externo al lenguaje mismo, a excepción del contenedor **Entities** que tiene algunas excepciones.

- **Implementation:**



- **Data Sources:** Contenedor que Implementa el contenedor de los **Data Sources abstractos** del **módulo domain**, aqui podemos implementar todo el código necesario para conectar con nuestras **fuentes de datos**.
- **Models:** Contenedor que define una clse vonmetodos que almacenan los datos que vienen de los Data Sources implementados y su razón de existir es convertir el formato de origen en objetos que pueden ser consumidos por la implementación de los Data Sources haciendo uso

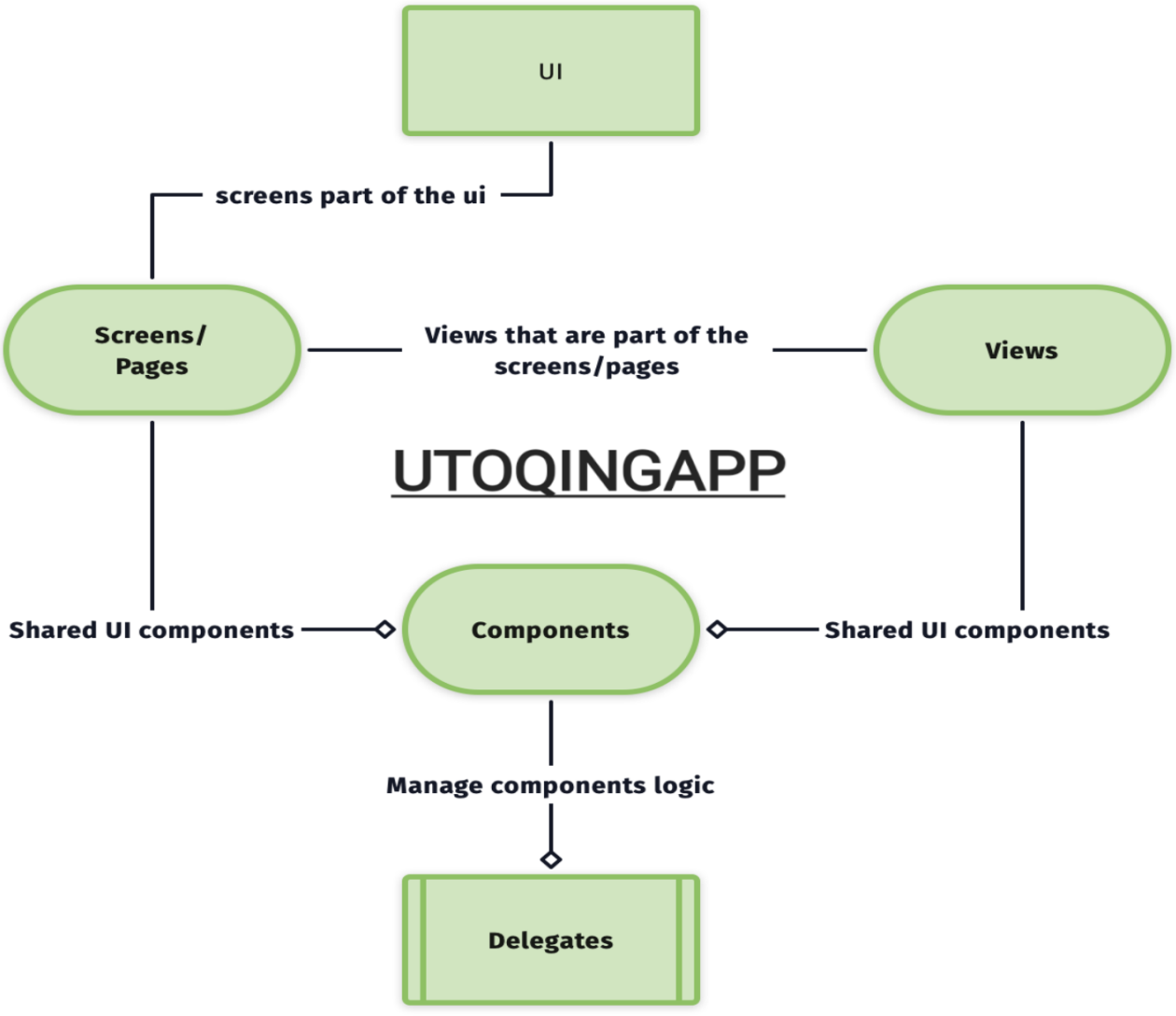
de los **adapters** o directamente si se necesita convertir a su formato de origen.

- **Adapters:** el contenedor de los Adapters son clases que contienen métodos estáticos que transforman los **modelos** completamente o parte de ellas a **entidades**, un **Adapter** debe entregar una **entidad** totalmente funcional y sin errores, solamente se pueden usar dentro del **modulo implementation en su contenedor de Data Sources**.
- **Repositories:** Contenedor que Implementa el contenedor de los **Data Sources implementados**, estos son los únicos que pueden tener una comunicación con el **State Management**.

- **State Management:** Capa donde se establce el patrón de diseño que se usará para administrar el estado de la capa de **UI**, también cumple la función de comunicar la capa **Core** con la capa **UI**.

UTOQINGAPP

- **User Interface (UI):** Es la **capa** con la que los usuarios interactua por decirlo de otra forma es la capa **visible** para el usuario.



- **Screens/Pages:** Es un **módulo** que contiene todas las pantallas de la **UI** que no tengan más de una **vista**, y se separan en contenedores por relación.
- **Views:** Es un **módulo** de **vista** que forman parte de una **Screens/Pages**, igualmente están agrupados en contenedores por relación.
- **Components:** Es un **módulo** que agrupa en contenedores por relación, los componentes y contiene un contenedor global que comparte entre cualquier **Screens/Pages** o **Views** sin mantener una relación.
- **Delegates:** Módulo que se encarga de manejar la **lógica** de

los **componentes** y puede contener piezas de UI.

Relación y Flujo de Control: