# ARQUITECTURA LIMPIA

Iván Álvarez López UO264862

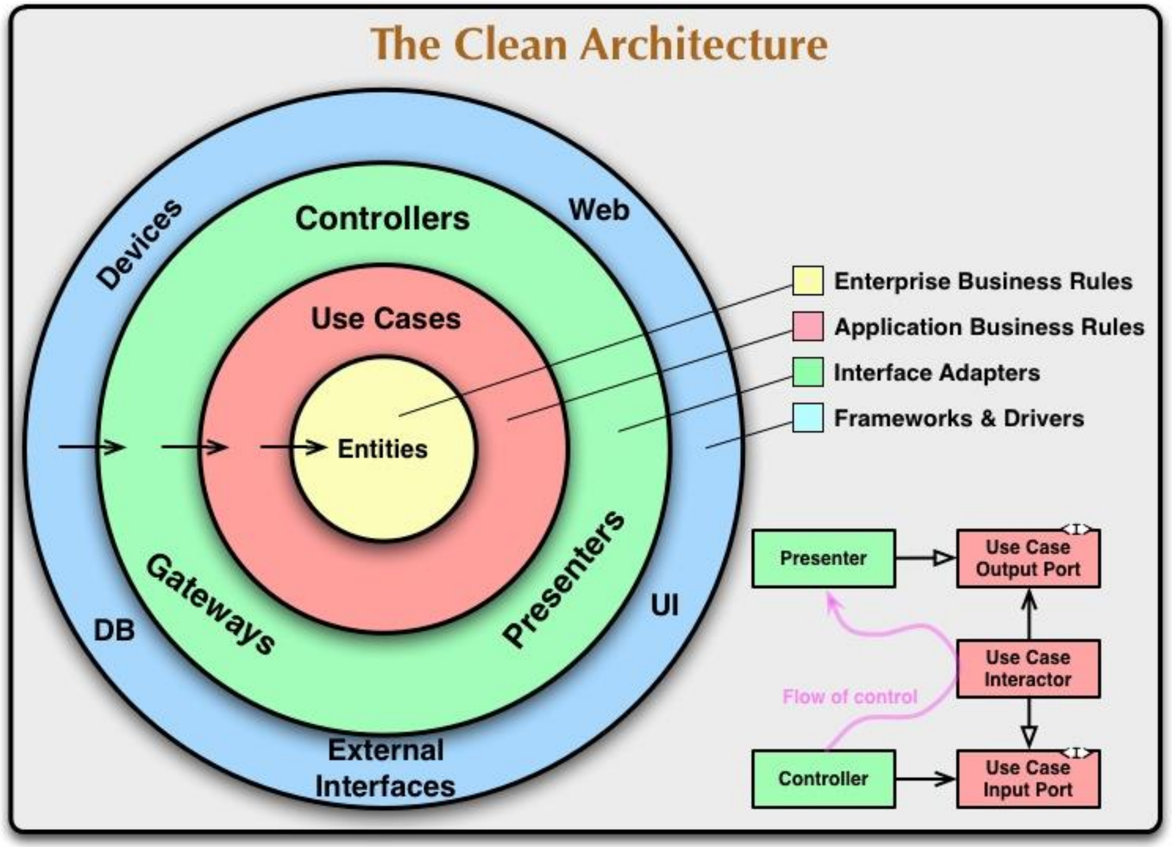
Daniel Pascual López UO269728

Fernando Giganto Rodríguez UO257125

## ÍNDICE

# INTRODUCCIÓN

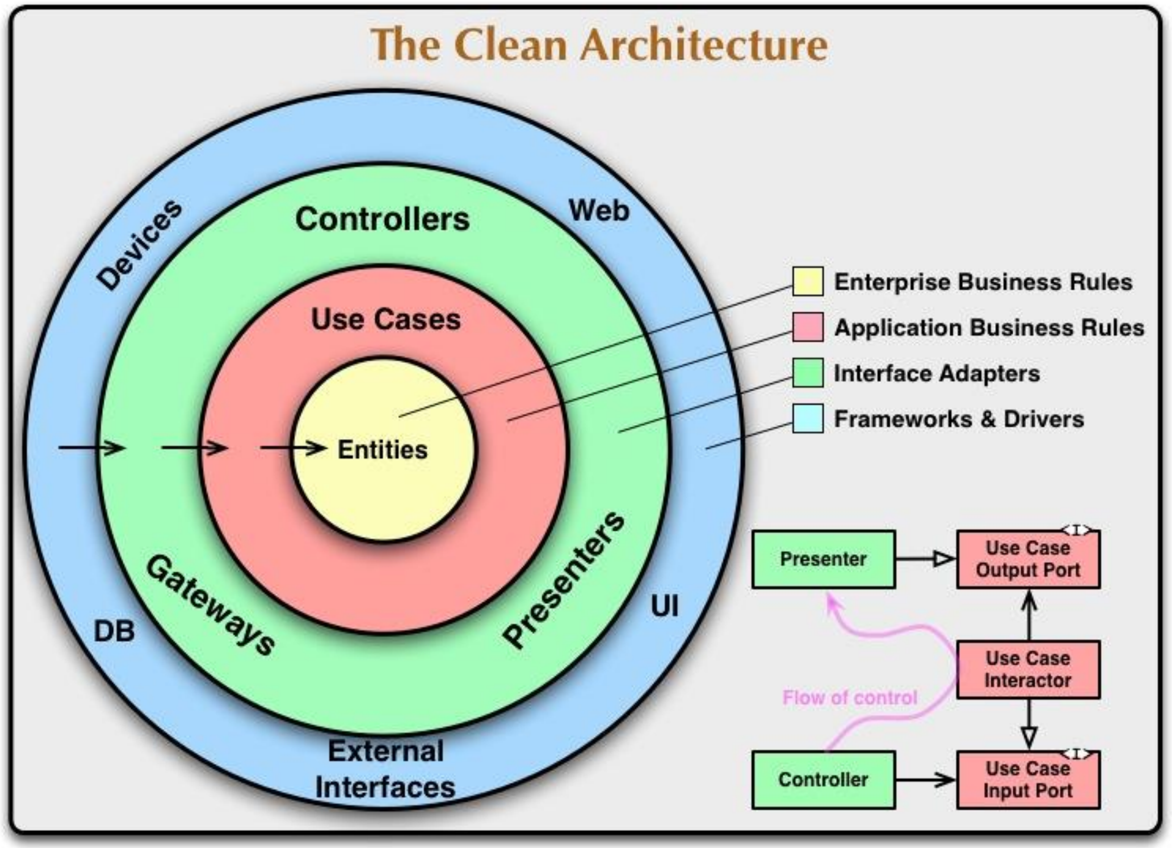
1. Independiente al Framework
2. Testable
3. Independiente UI
4. Independiente BD
5. Independiente agentes externos



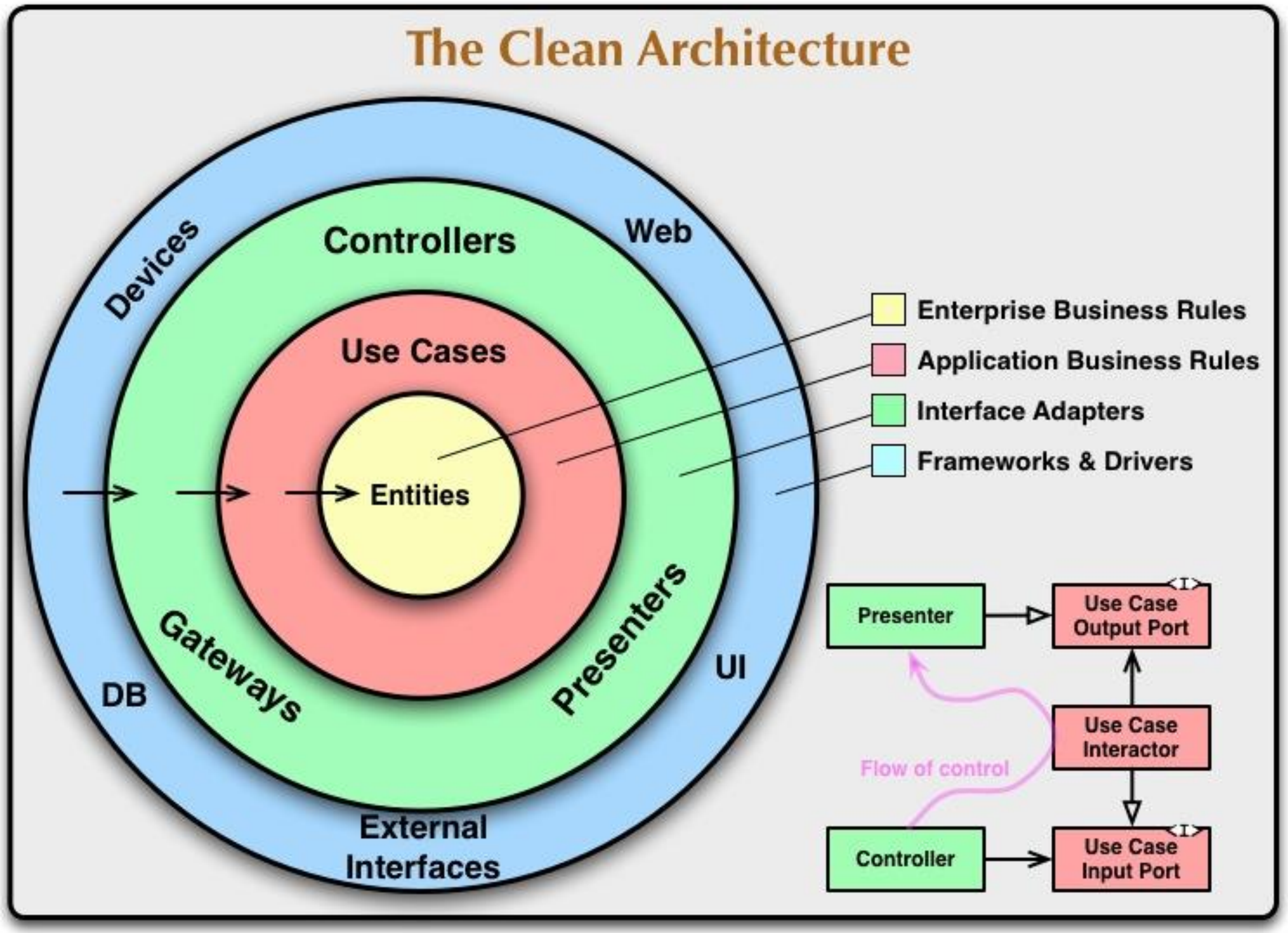
## REGLA DE DEPENDENCIA

Las dependencias del código fuente solo pueden apuntar hacia adentro.

Del mismo modo, los formatos de datos utilizados en un círculo exterior no deben ser utilizados por un círculo interior, especialmente si esos formatos son generados por un marco en un círculo exterior. No queremos que nada en un círculo externo impacte en los círculos internos.



# ESQUEMA ARQUITECTURA



Fuente: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

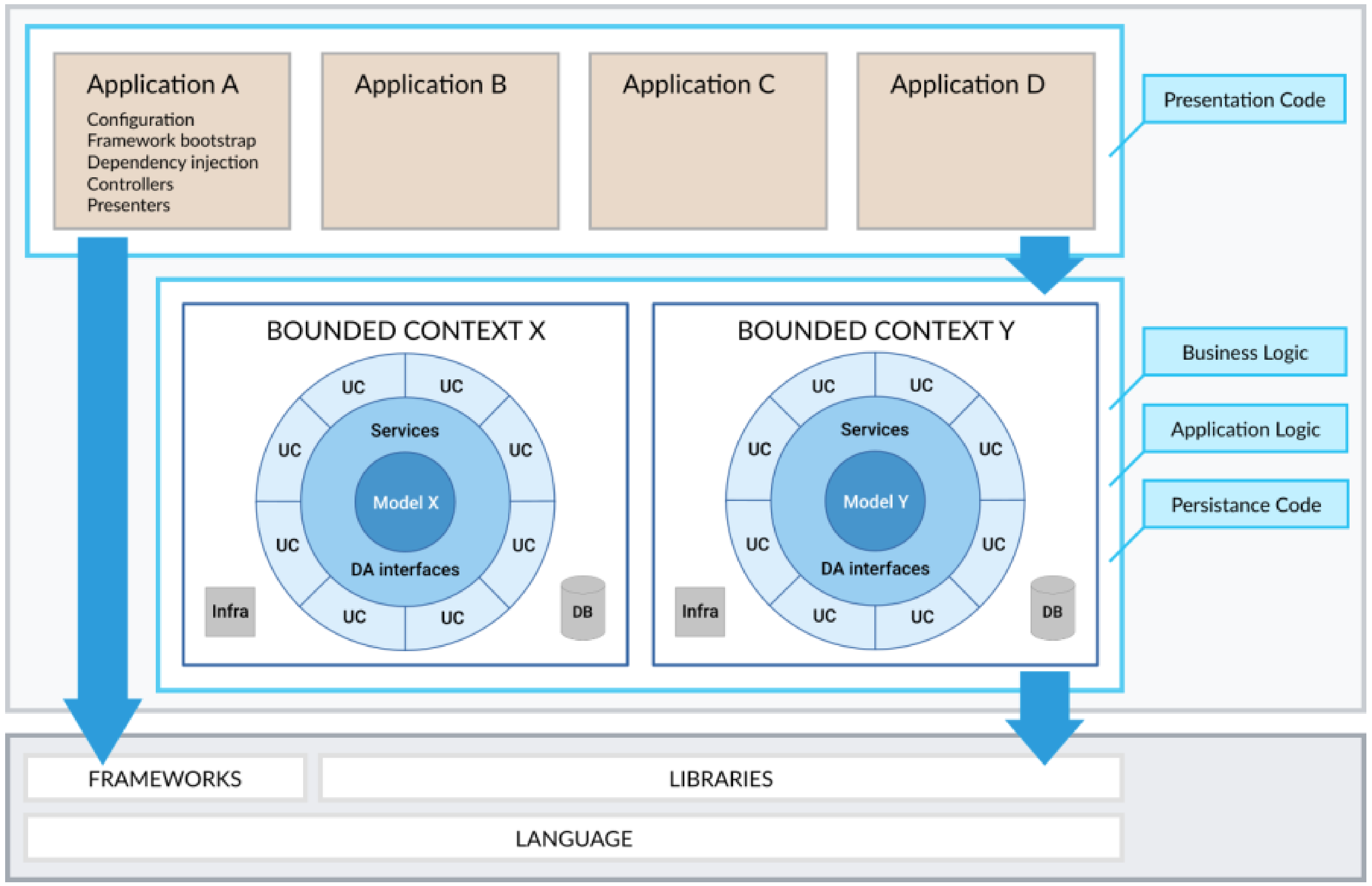
# ARQUITECTURA LIMPIA + BOUNDED CONTEXT

Bounded Context:

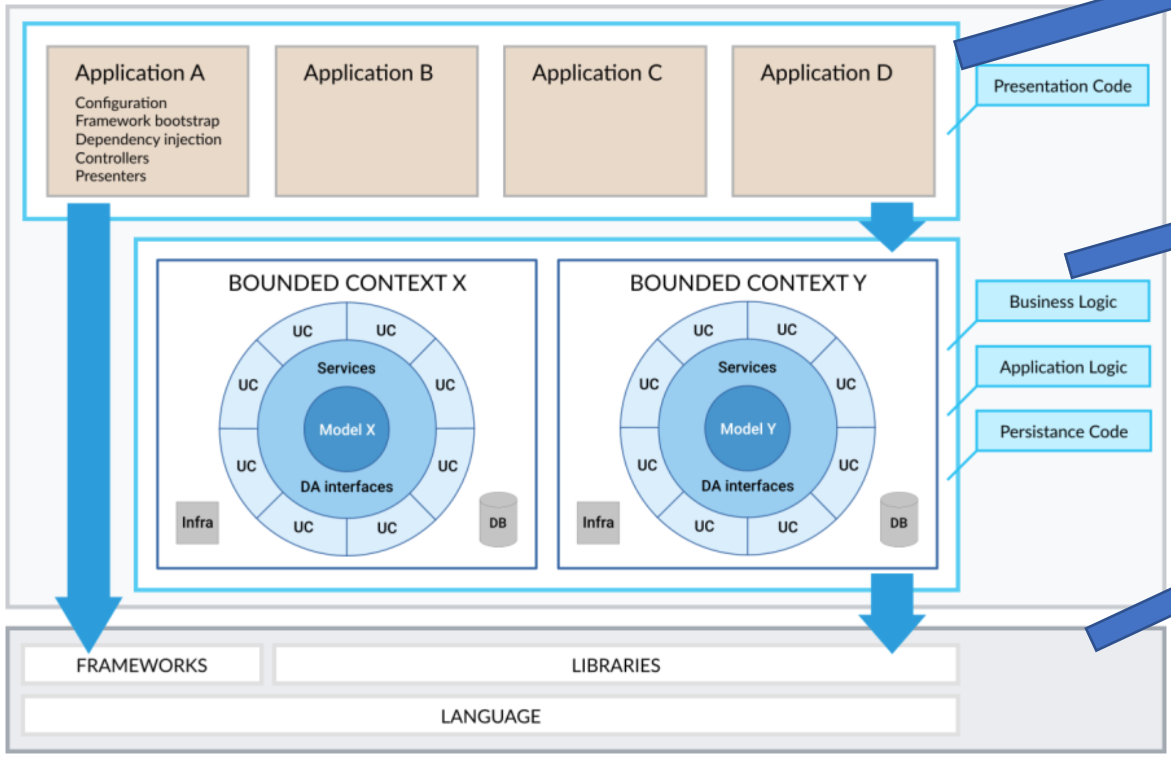
-Un dominio puede dividirse en subdominios

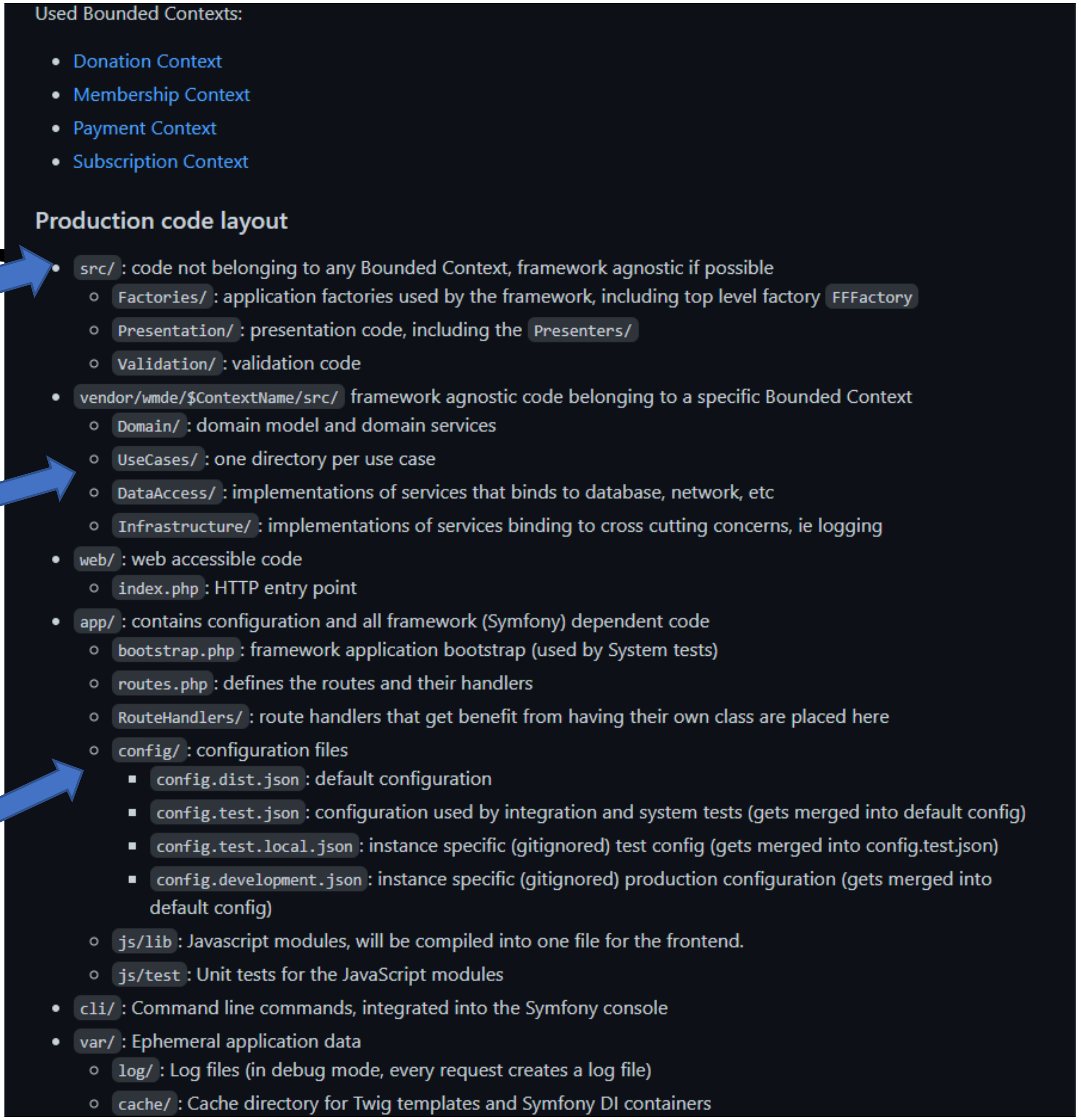
- Los subdominios relacionados
se agrupan en un BC

- Estos se pueden comunicar en las fronteras de los BC
mediante servicios o adapters.



# EJEMPLO:




Used Bounded Contexts:

- Donation Context
- Membership Context
- Payment Context
- Subscription Context

## Production code layout

- `src/`: code not belonging to any Bounded Context, framework agnostic if possible
  - `Factories/`: application factories used by the framework, including top level factory `FFFactory`
  - `Presentation/`: presentation code, including the `Presenters/`
  - `Validation/`: validation code
- `vendor/wmde/$ContextName/src/` framework agnostic code belonging to a specific Bounded Context
  - `Domain/`: domain model and domain services
  - `UseCases/`: one directory per use case
  - `DataAccess/`: implementations of services that binds to database, network, etc
  - `Infrastructure/`: implementations of services binding to cross cutting concerns, ie logging
- `web/`: web accessible code
  - `index.php`: HTTP entry point
- `app/`: contains configuration and all framework (Symfony) dependent code
  - `bootstrap.php`: framework application bootstrap (used by System tests)
  - `routes.php`: defines the routes and their handlers
  - `RouteHandlers/`: route handlers that get benefit from having their own class are placed here
  - `config/`: configuration files
    - `config.dist.json`: default configuration
    - `config.test.json`: configuration used by integration and system tests (gets merged into default config)
    - `config.test.local.json`: instance specific (gitignored) test config (gets merged into config.test.json)
    - `config.development.json`: instance specific (gitignored) production configuration (gets merged into default config)
  - `js/lib`: Javascript modules, will be compiled into one file for the frontend.
  - `js/test`: Unit tests for the JavaScript modules
- `cli/`: Command line commands, integrated into the Symfony console
- `var/`: Ephemeral application data
  - `log/`: Log files (in debug mode, every request creates a log file)
  - `cache/`: Cache directory for Twig templates and Symfony DI containers

# IMPLEMENTACIÓN DE CLEAN ARCHITECTURE EN *REACT*

- Frecuentemente, se tiende a utilizar *React* como arquitectura global de la aplicación.
- Esto presenta problemas, como en el caso de la inversión de dependencias.
    - Da lugar a diseños no reutilizables. (Poca reusabilidad)
    - Los componentes de la aplicación casi nunca están predispuestos a cambios. (Poca flexibilidad)
- Idealmente, *React* debería comprender únicamente el ámbito de la presentación al usuario.

Fuente: "React as an Implementation Detail - Chris Kiehl (Jun 05, 2019)" [Inglés]

# MAL EJEMPLO DE IMPLEMENTACIÓN EN *REACT*

- Componente fuertemente acoplado:

```
const Headline: React.StatelessComponent =
({stuff}) => (<Title>{stuff.map(x => x.thing.name).join('-")}</Title>)
```

- El acoplamiento repercute en niveles superiores de la aplicación:

```
const MyThingList: React.Stateless =
      ({things}) => ({_.orderBy(things, ['group', 'createdOn’])
            .map(stuff =><Headline stuff={stuff} />)})
```

Fuente: "React as an Implementation Detail - Chris Kiehl (Jun 05, 2019)" [Inglés]

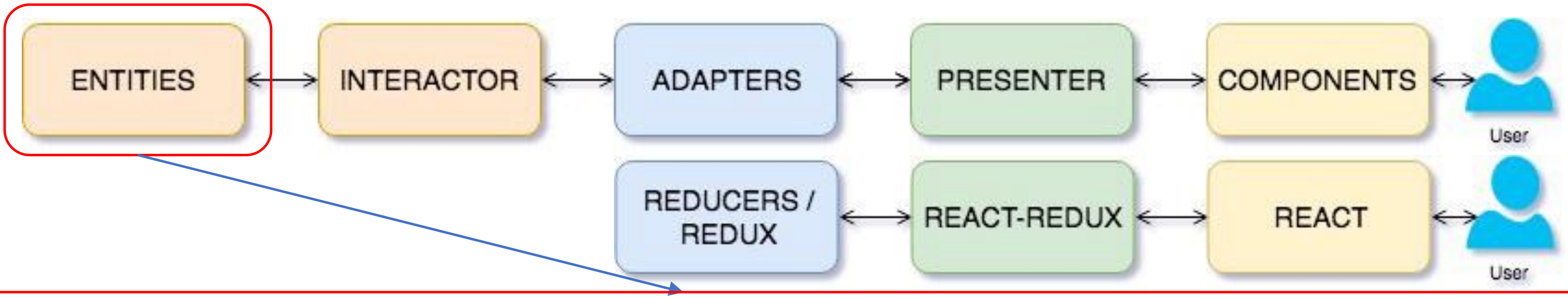
## MISMO EJEMPLO MEJOR PENSADO EN *REACT*

- La lógica de negocio es extraída del componente:

```
const Headline: React.StatelessComponent =
    ({title}) => (<Title>{title}</Title>)
```

Fuente: "React as an Implementation Detail - Chris Kiehl (Jun 05, 2019)" [Inglés]

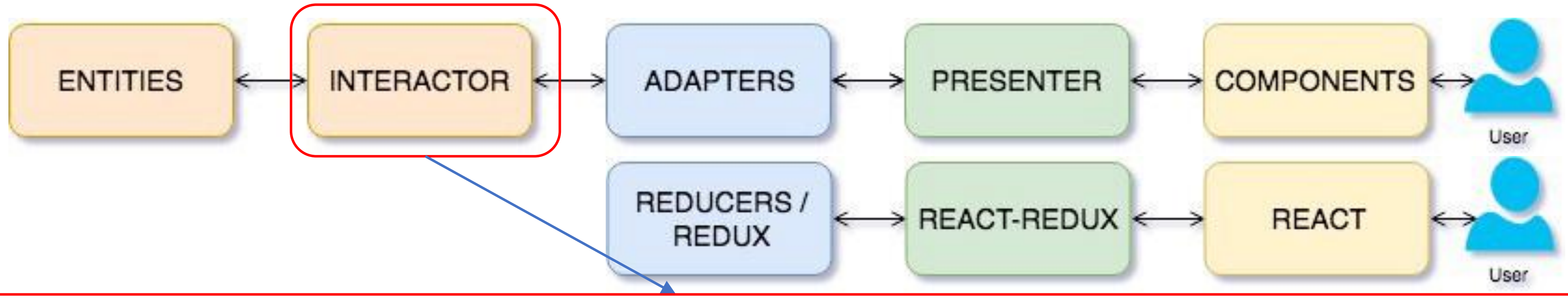
## EJEMPLO: CONTADOR (ENTIDAD)



```
export class Counter {
  count: number;
  constructor(startNumber: number) {
    this.count = startNumber;
  }
  increment(qty?: number) {this.count += qty ? qty :1;}
  decrement(qty?: number) {this.count -= qty ? qty :1;}
}
```

Fuente: "Arquitetura limpa para bases de código React - Eduardo Morôni (Jun 27, 2018)" [Portugués]

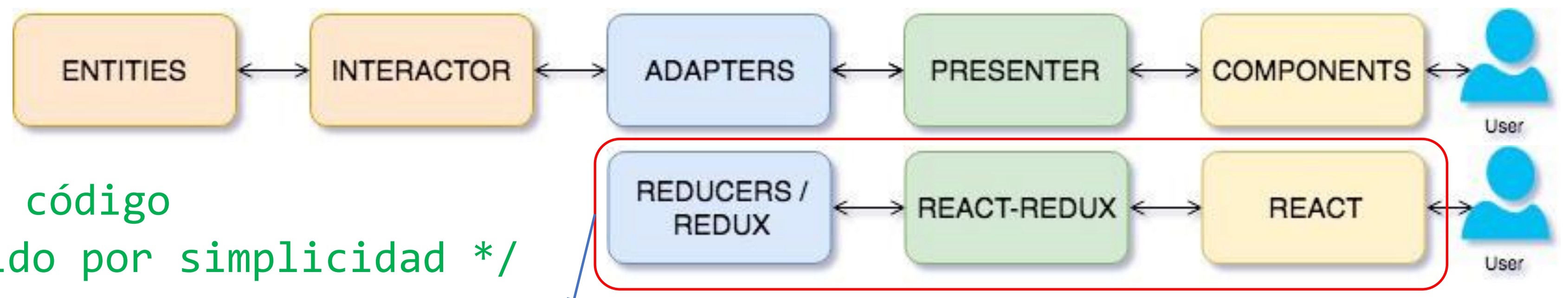
## EJEMPLO: CONTADOR (*INTERACTOR* / CASO DE USO)



```
import {Counter} from “../entities”;
export class CounterInteractor {
  higherBound: number = 10;
  count: Counter;
  constructor(startNumber: number, higherBound: number = 10) {
    this.count = new Counter(startNumber);   this.higherBound = higherBound;
  }
  increment(qty?: number): Counter {
    this.count.increment(qty);
    if (this.counter.count >= this.higherBound)
      this.counter = new Counter(this.higherBound);
    return this.counter;
  }
  decrement(qty?: number) {/* Omitido por simplicidad */}
}
```

Fuente: "Arquitetura limpa para bases de código React - Eduardo Morôni (Jun 27, 2018)" [Portugués]

# EJEMPLO: CONTADOR (*ADAPTERS, PRESENTERS, COMPONENTS*)



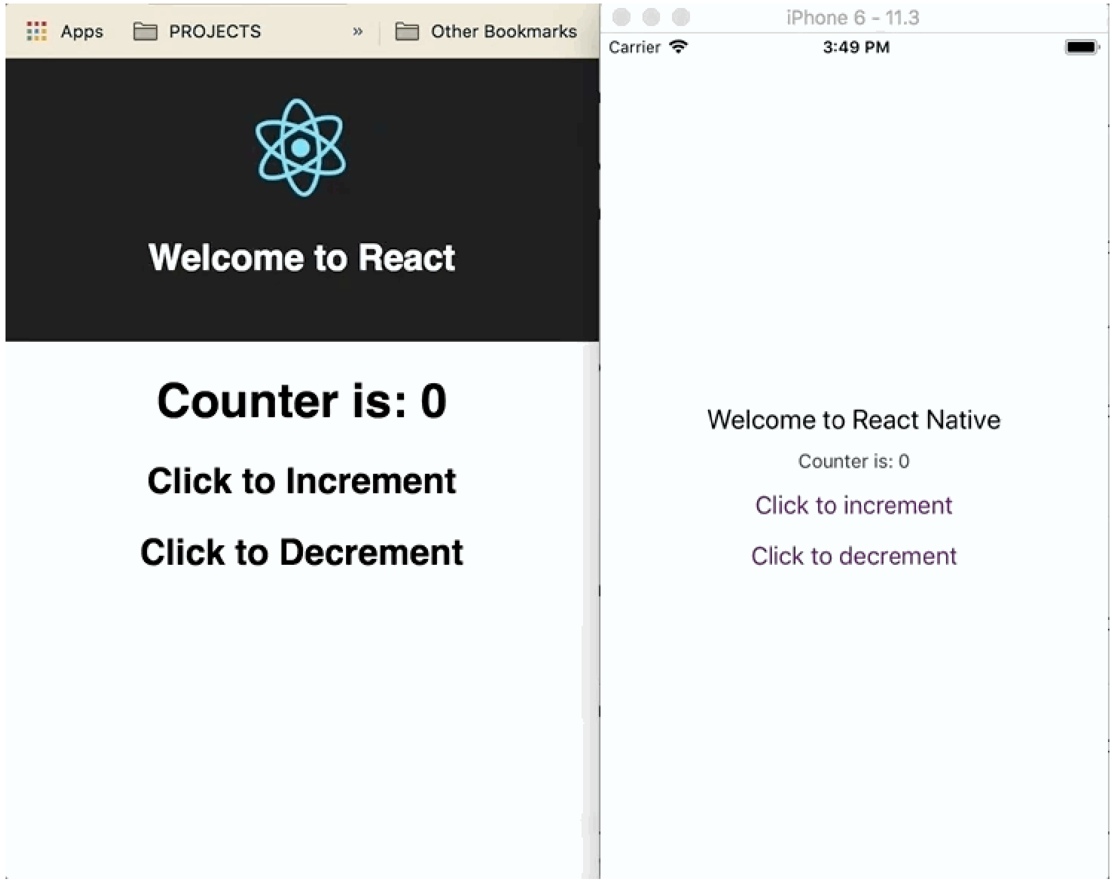
```
/* Resto del código
        omitido por simplicidad */
```

```
const incrementReducer =
  (counter: StateSliceType,
  action: ActionType,):
    StateSliceType => {
  const interactor =
    new CounterInteractor(counter);
  interactor.increment(action.qty);
  return new Counter(
    interactor.counter.count
  );
};
```

```
export const counterReducer =
  (state: StateSliceType =
    INITIAL_STATE,
  action: ActionType,):
    StateSliceType => {
  switch (action.type) {
    case INCREMENT:
      return incrementReducer(state, action);
    case DECREMENT:
      return decrementReducer(state, action);
    default:
      return state;
  }
};
```

Fuente: "Arquitetura limpa para bases de código React - Eduardo Morôni (Jun 27, 2018)" [Portugués]

# EJEMPLO: CONTADOR (ENTIDAD)



Welcome to React

Counter is: 0

Click to Increment

Click to Decrement

Welcome to React Native

Counter is: 0

Click to increment

Click to decrement

Fuente: "Arquitetura limpa para bases de código React - Eduardo Morôni (Jun 27, 2018)" [Portugués]

PREGUNTAS