Marc Brehmer

Developing a clean architecture-inspired React application with MVVM

Frontend

code.talks

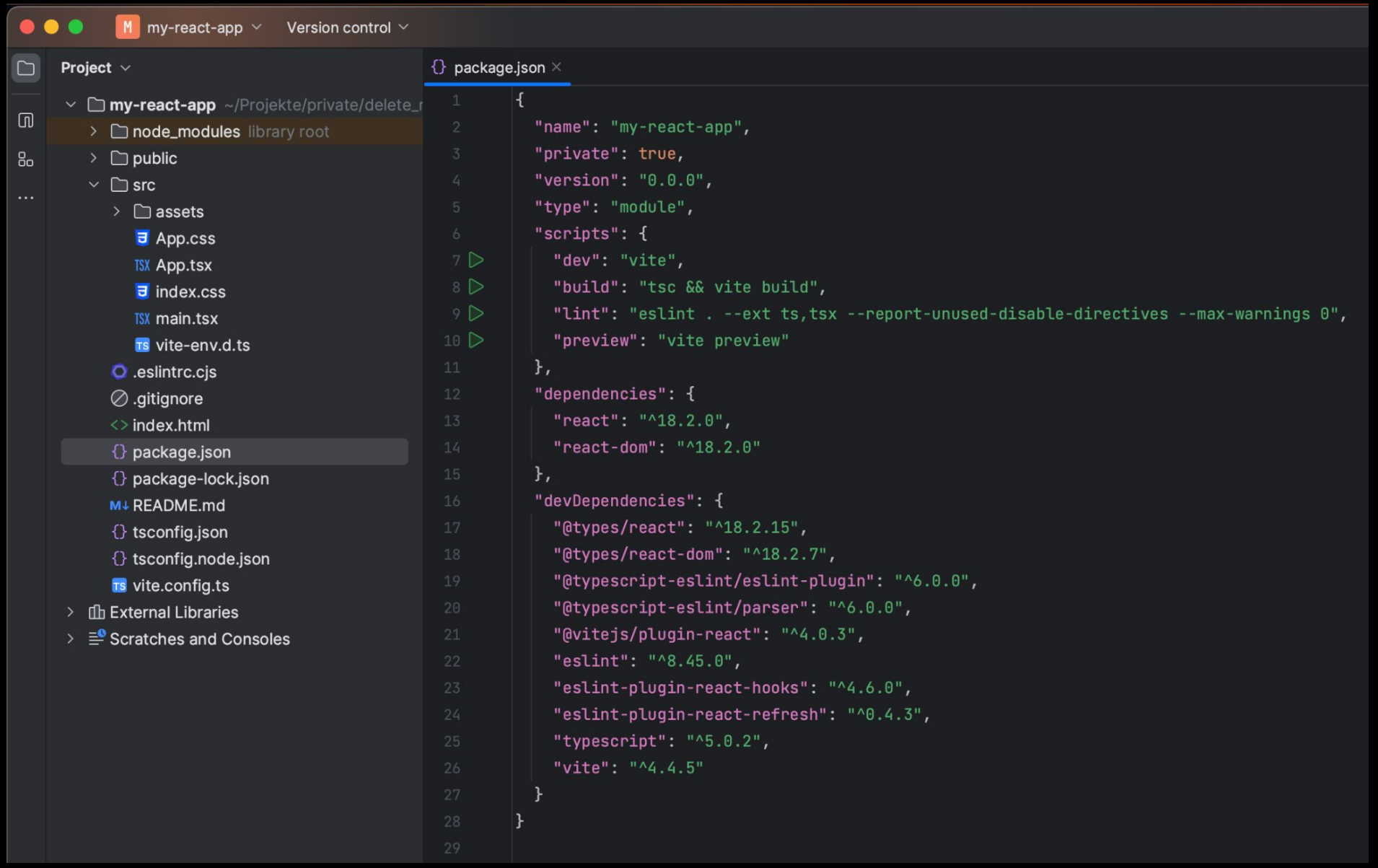
About me

- Software Engineer since 2013
- @Spaceteams
- Technology agnostics - No boundaries or biases
- Staying ahead of the game - It’s Spacetime

marc-brehmer marc_brehmer disco-joern

spaceteams

Setting up a React app


my-react-app
Version control
Project
my-react-app ~/Projekte/private/delete_r
node_modules library root
public
src
assets
App.css
App.tsx
index.css
main.tsx
vite-env.d.ts
.eslintrc.cjs
.gitignore
index.html
package.json
package-lock.json
README.md
tsconfig.json
tsconfig.node.json
vite.config.ts
External Libraries
Scratches and Consoles
package.json
{
"name": "my-react-app",
"private": true,
"version": "0.0.0",
"type": "module",
"scripts": {
"dev": "vite",
"build": "tsc && vite build",
"lint": "eslint . --ext ts,tsx --report-unused-disable-directives --max-warnings 0",
"preview": "vite preview"
},
"dependencies": {
"react": "^18.2.0",
"react-dom": "^18.2.0"
},
"devDependencies": {
"@types/react": "^18.2.15",
"@types/react-dom": "^18.2.7",
"@typescript-eslint/eslint-plugin": "^6.0.0",
"@typescript-eslint/parser": "^6.0.0",
"@vitejs/plugin-react": "^4.0.3",
"eslint": "^8.45.0",
"eslint-plugin-react-hooks": "^4.6.0",
"eslint-plugin-react-refresh": "^0.4.3",
"typescript": "^5.0.2",
"vite": "^4.4.5"
}
}

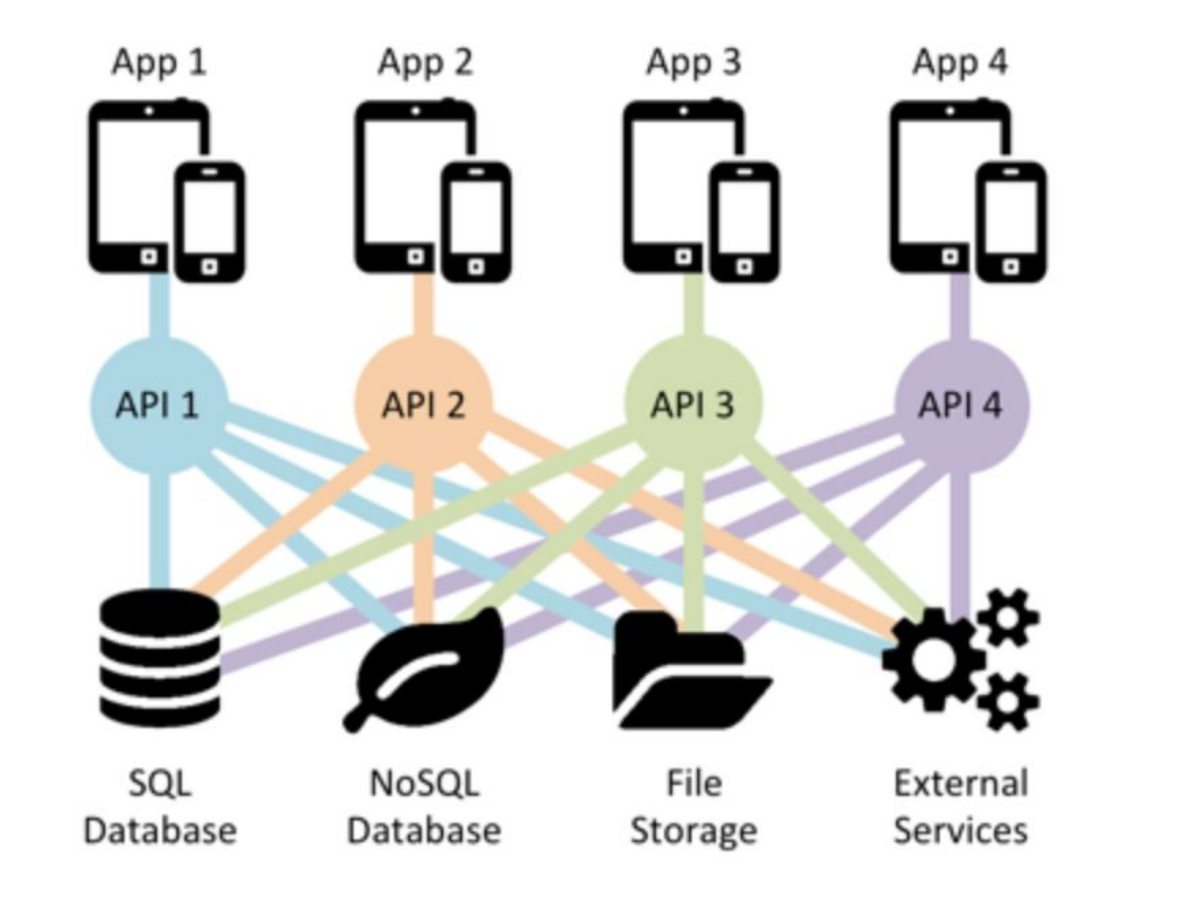
spaceteams

Issues incoming...

Code Duplication

Source: Mr. Hugo Weaving in “The Matrix Reloaded”, 2003. Photograph by Warner Bros./Photofest

Tight Coupling

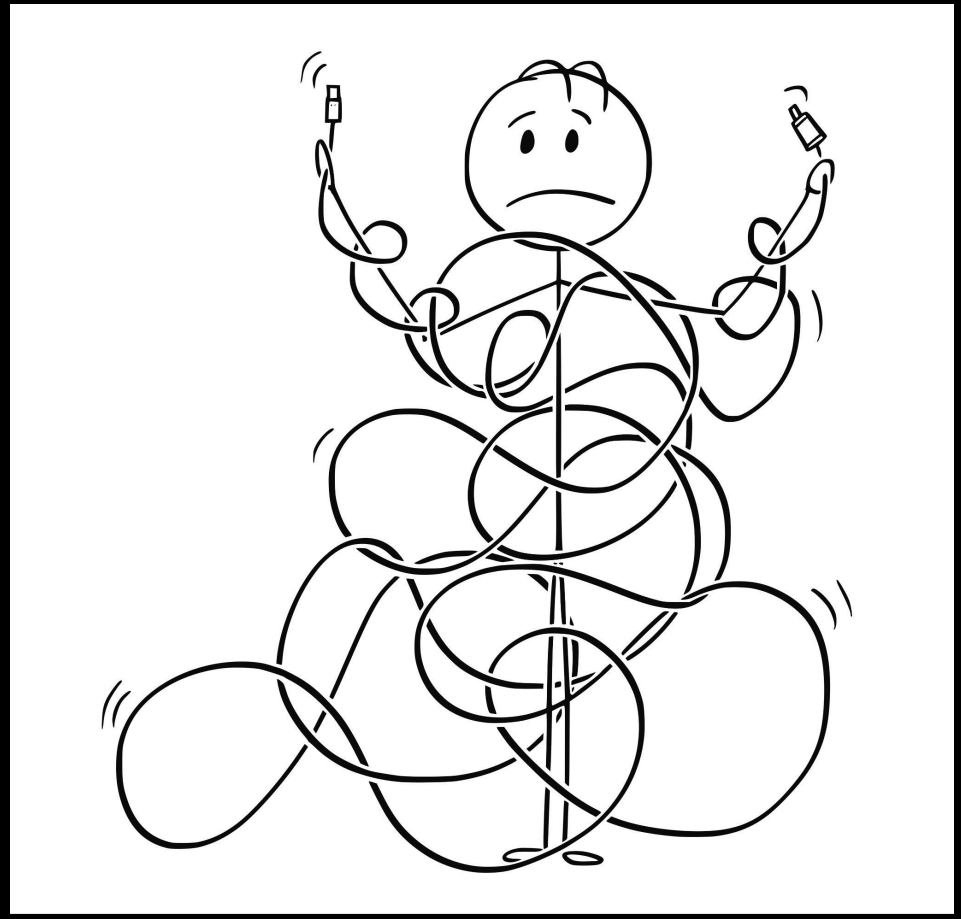


Source: [Jason Gilmore - What is Loose Coupling in REST APIs?]

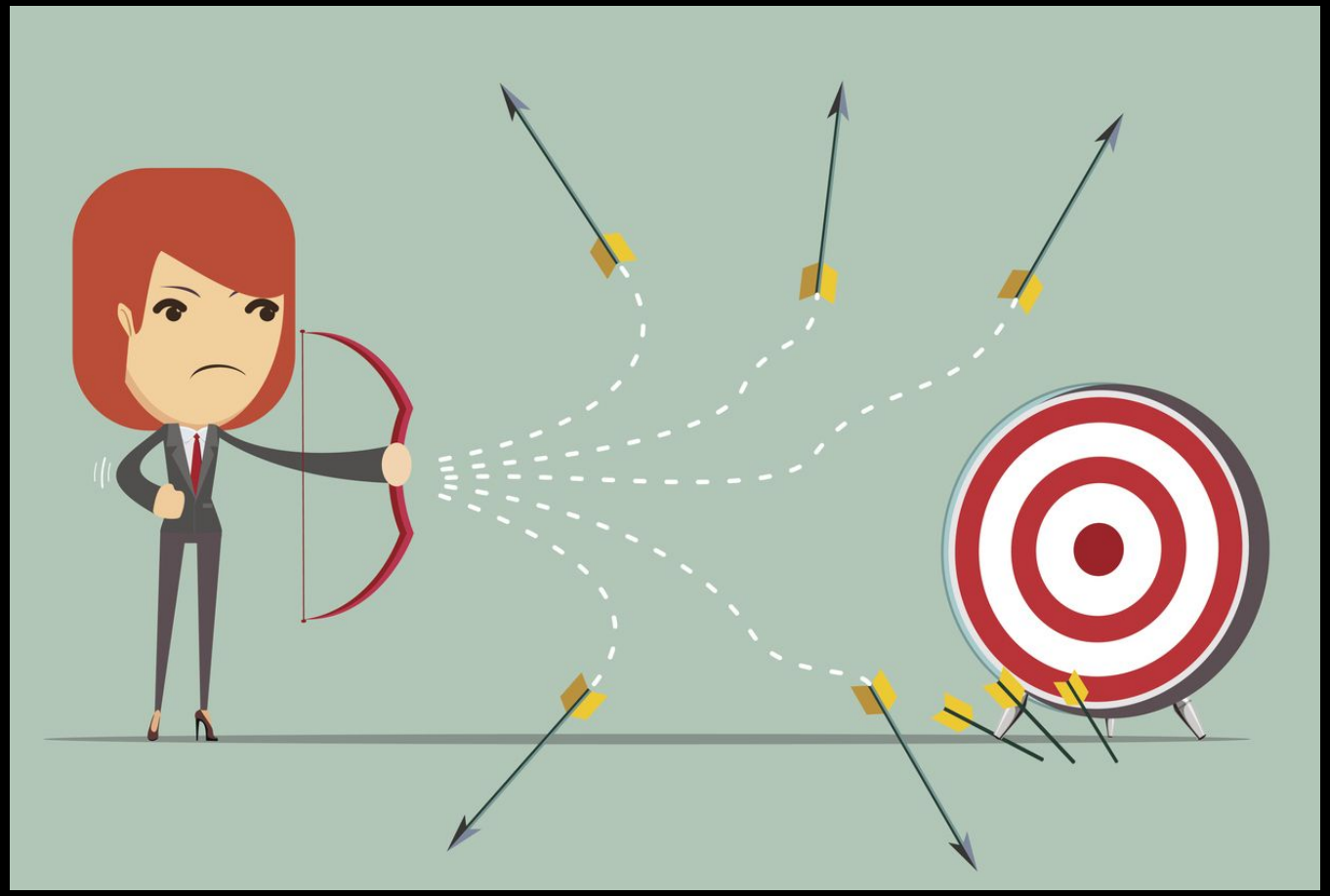
Scalability Challenges

Source: [Aran Davies - How to Scale a Software Product?]

Testing Complexity

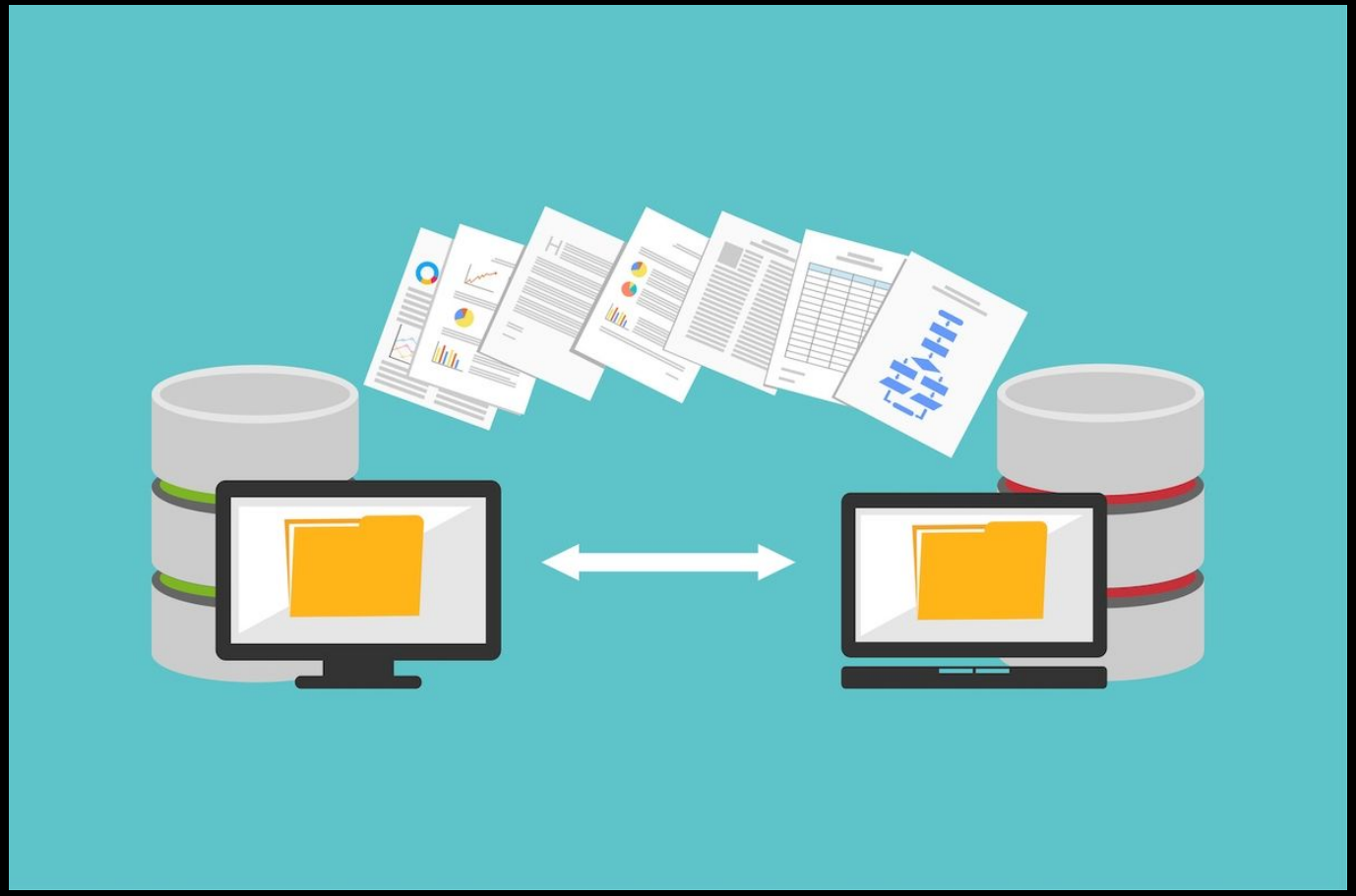


Inconsistencies



Source: [Jenna Gourlay - The 3 Secrets For Consistency... From the Consistently Inconsistent]

Difficulty in Migration

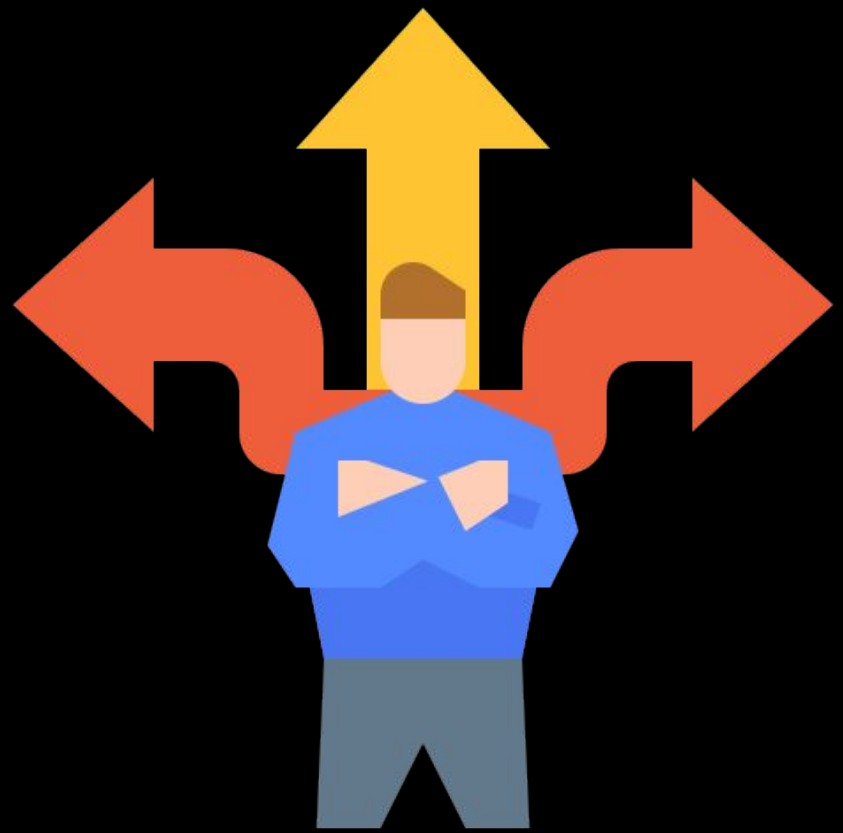


Source: Ali Azhar - What is data migration?

Reduced Maintainability

Source: Nina Perez - 3 types of Maintenance and How to go from Reactive to Predictive with IoT Technology

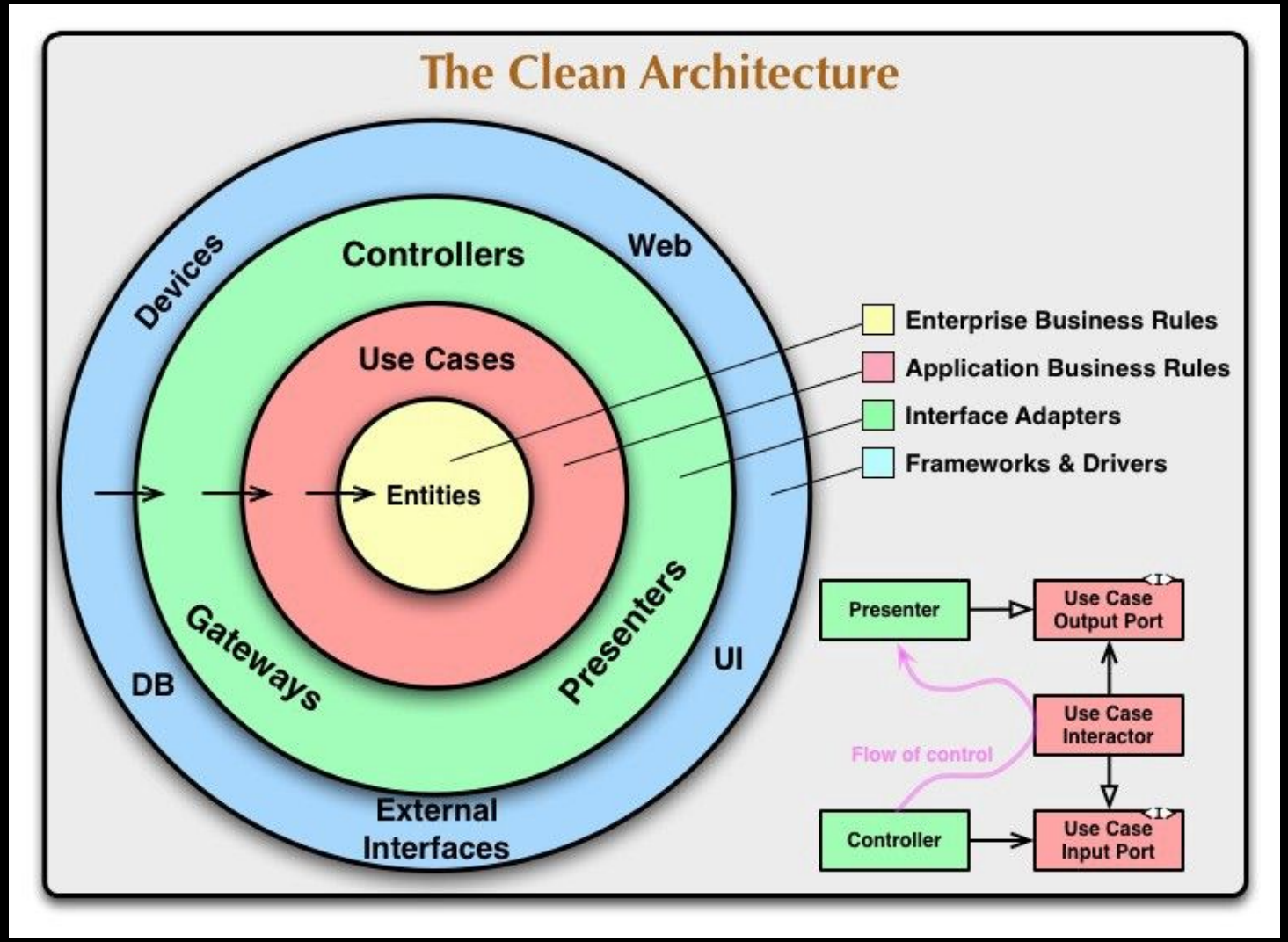
Lack of Flexibility



Source: Kavya Garg - Flexibility

spaceteams

Clean Architecture

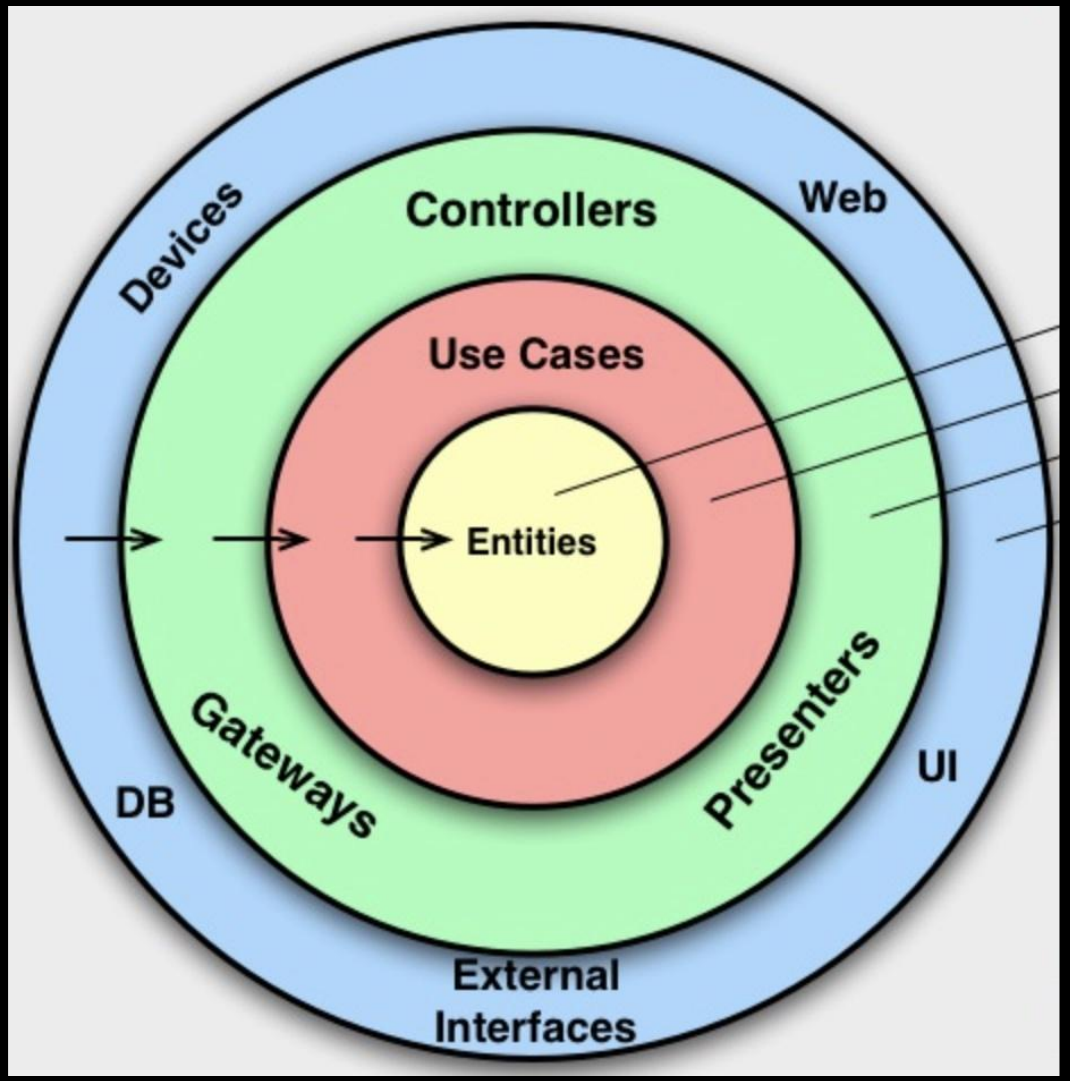


Source: [Robert C. Martin (Uncle Bob) - The Clean Code Blog]

Entities

Enterprise Business Rules

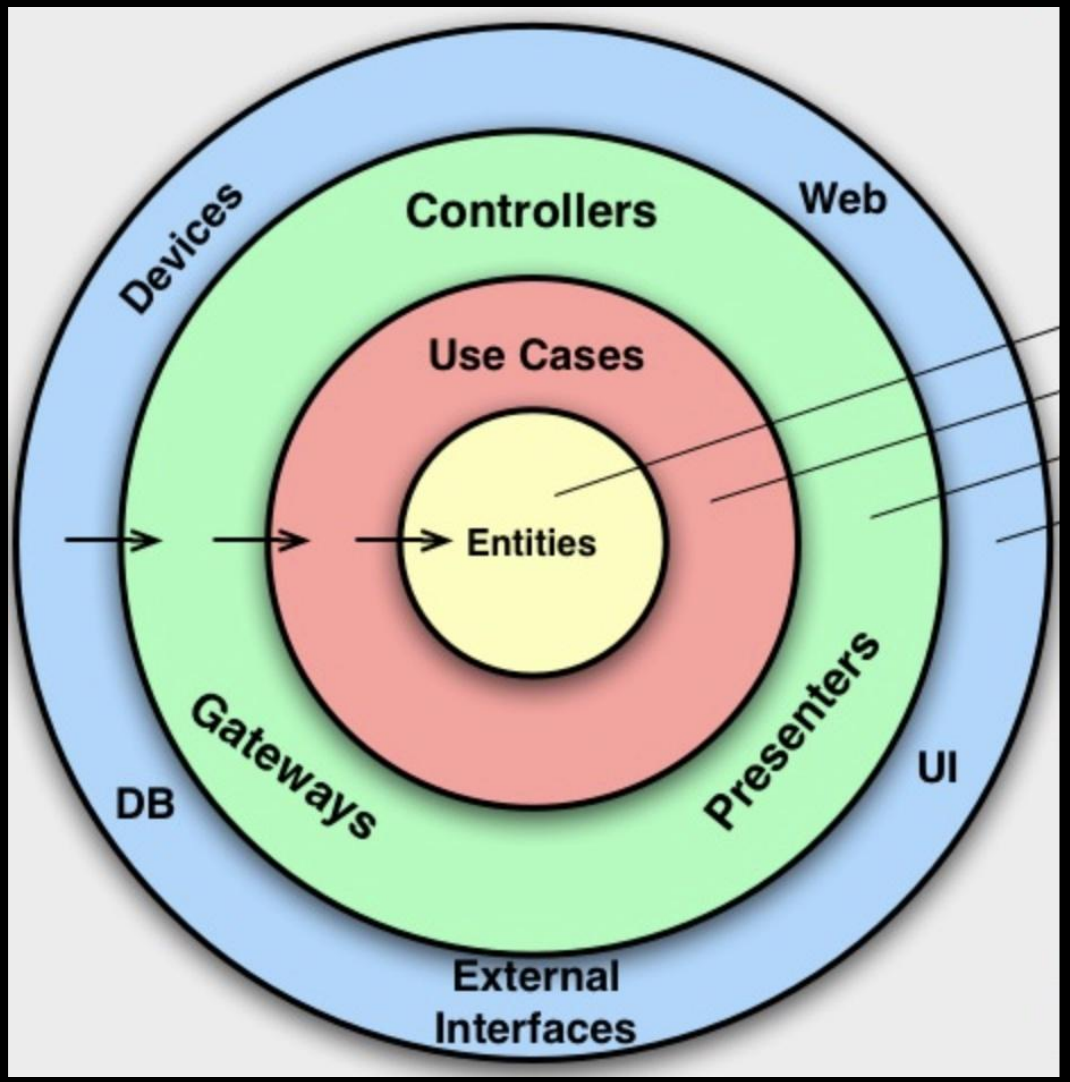
- Represent domain objects
- Contains business rules & business data
- Separated from every other concern in the application



Use Cases

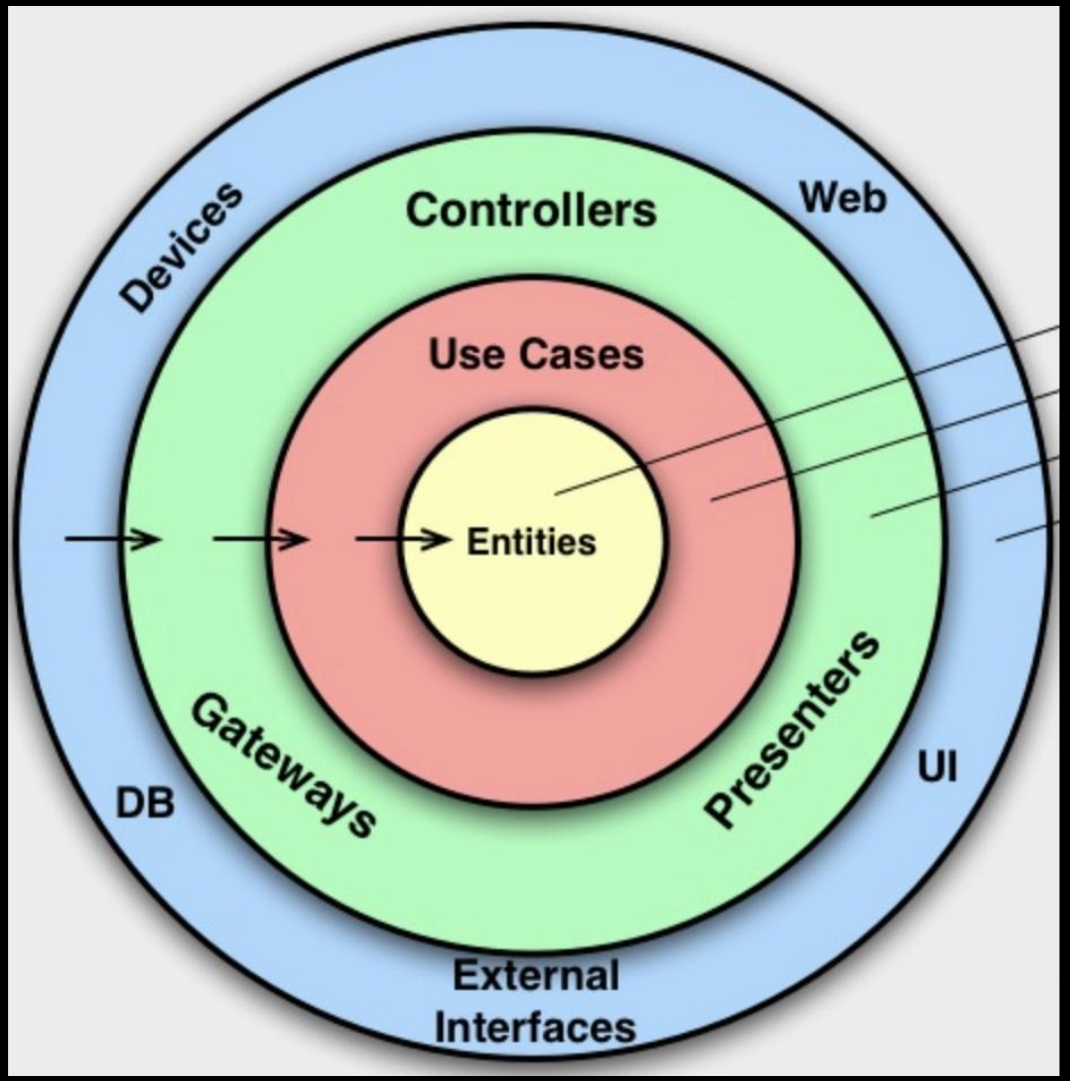
Application Business Rules

- Represent business actions
- Pure business logic



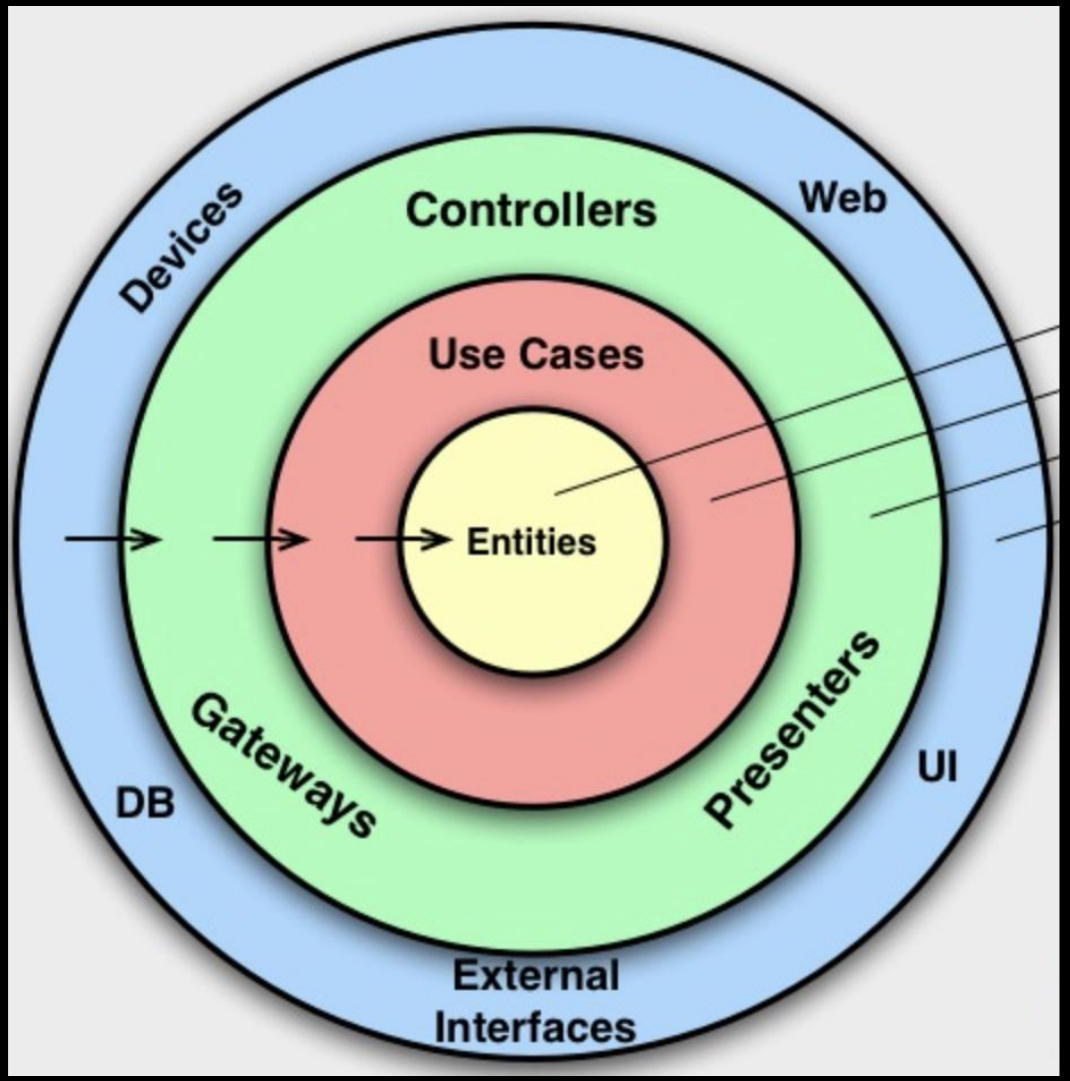
Interface Adapters

- Converts data from data layer to use case / entity layer & vice versa
- Implement the interfaces defined by the use case
- Trigger use cases
- Presenters, views & controllers all belong here



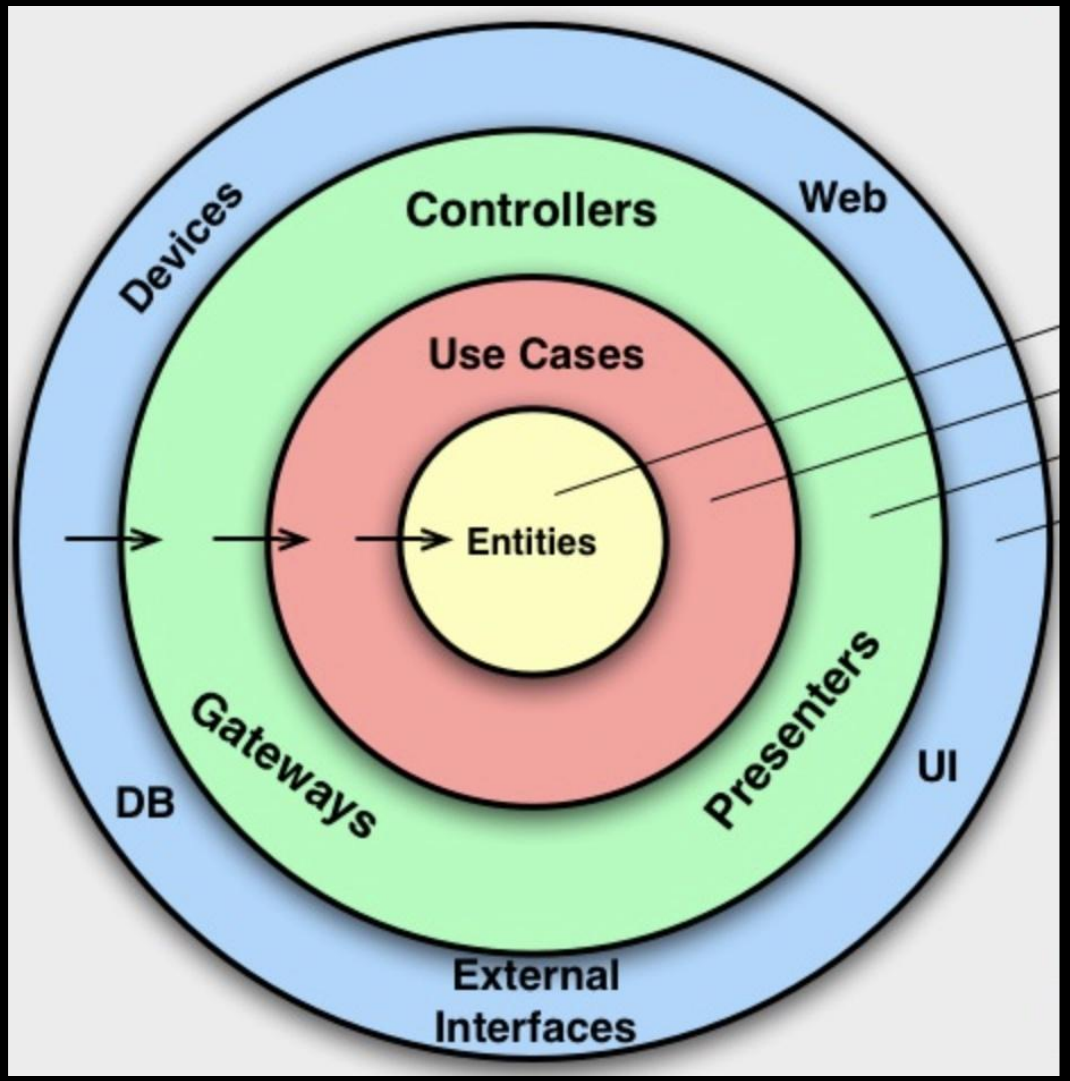
Frameworks & Drivers

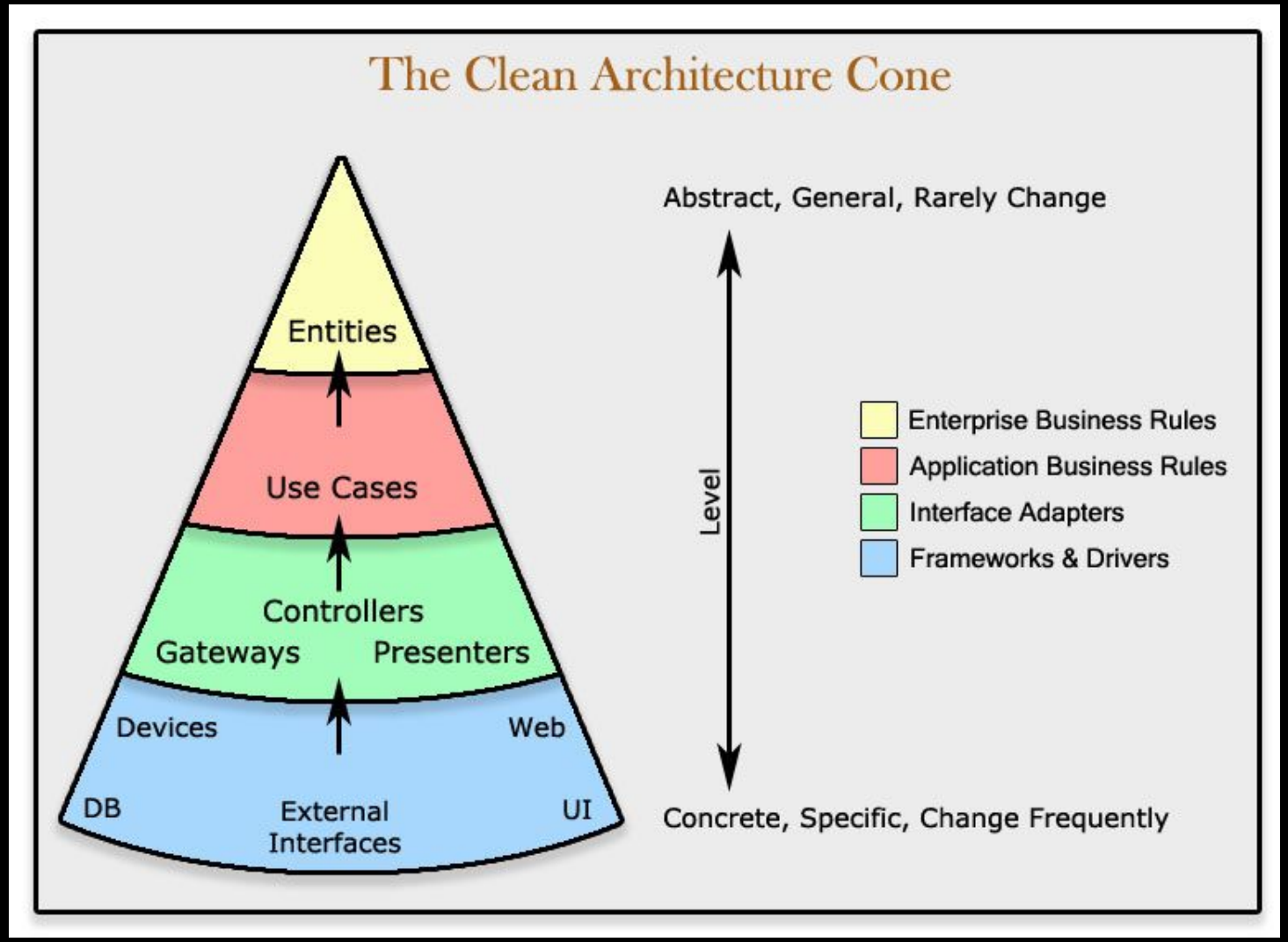
- Glue code that hooks up the various layers
- Composed of frameworks & tools
  - e.g., DB, Web Framework, etc.
- Isolates all details, like infrastructure, etc.
  - Can do little harm to the application
  - Makes it easy to replace



Crossing Boundaries

- Flow of control from the outside to the inside
- Dependency Inversion Principle:
  - Higher level components know nothing of the lower level components
  - Example
    - Use case needs to call a presenter
    - Use case calls an interface
    - Implementation of interface is provided by the interface adapter layer





Source: [Michael Outlaw - Clean Architecture – Make Your Architecture Scream]

How does this solve our issues?

CA's answer to issues

- Tight Coupling
  - CA promotes separation of concerns
  - Clear boundaries between different layers
  - Decoupling of components & modules by using interfaces & dependency injection
- Scalability Challenges
  - CA emphasizes modularity & organization
  - Isolating new features within specific layers or modules
- Testing Complexity
  - Components are independent & testable in isolation

CA's answer to issues

- Code Duplication
  - Creation of reusable components & services
- Inconsistencies
  - CA emphasizes consistent patterns & design principles throughout the app
- Difficulty in Migration
  - Modularity allows to encapsulate data access & other external dependencies
  - Switching components: Focus on updating specific layers or interfaces,
    - Minimizes the impact on the rest of the app

CA’s answer to issues

- Reduced Maintainability
  - Separation of concerns & modular design make maintenance easier
  - Updates & fixes within specific layers without affecting the entire app
- Lack of Flexibility
  - CA encourages loose coupling & abstraction
  - This makes replacing components or technologies easier when needed

Let's apply it with...

spaceteams

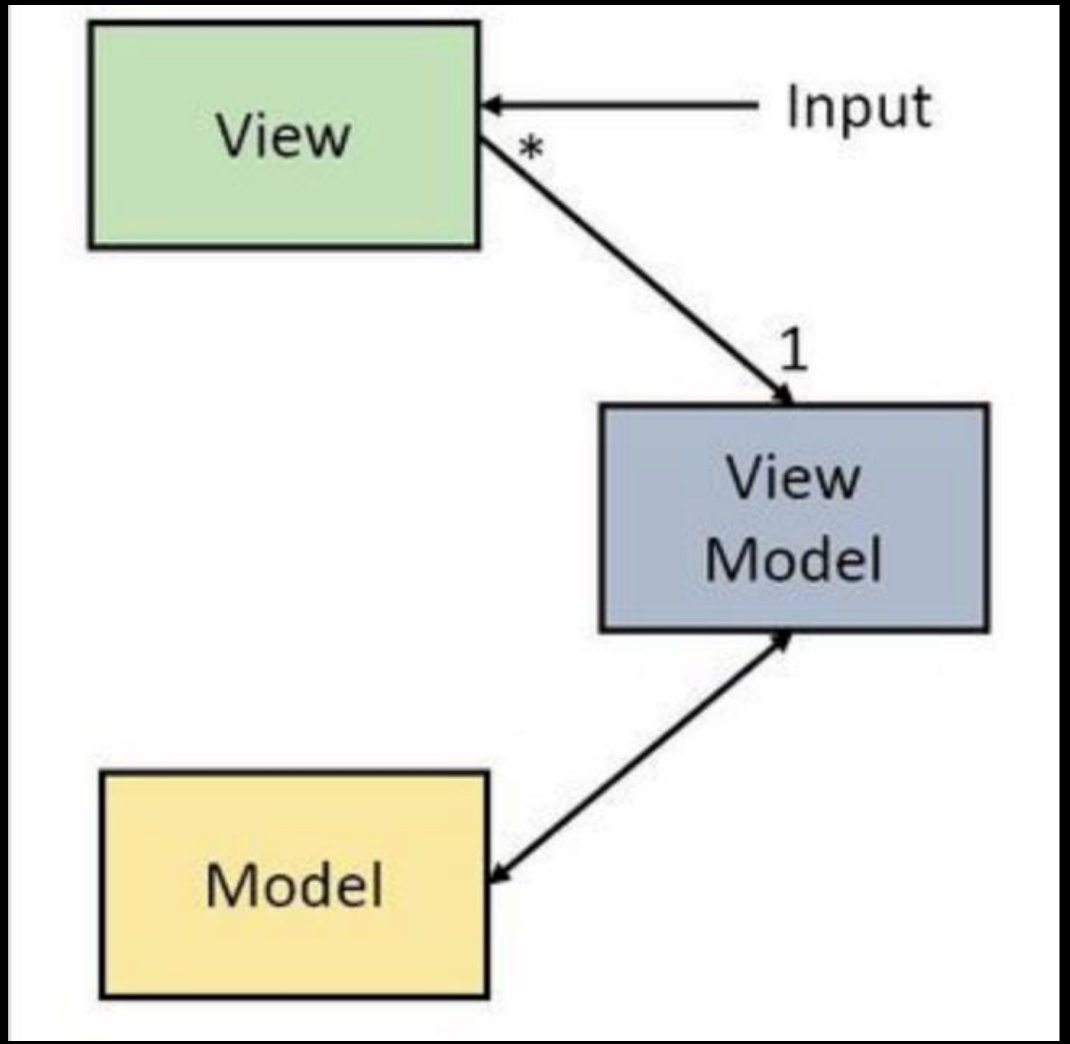
MVVM

MV* - Patterns

- Goal of the MV* - Patterns:
  - Separation of Concerns
  - Maintainability
  - Reusability
  - Testability

MVVM

- Model
  - Represents the app's data & business logic
- View
  - Displays data from the ViewModel in UI
  - Sends user input back to the ViewModel
- ViewModel (VM)
  - Exposes data from the Model to the View
  - Handles user interactions & processes them into actions on the Model



Source: T. Drilling & S. Augsten - Durchblick im JavaScript-Ökosystem

Why do we use MVVM?

- Separation of Concerns
  - Enforces separation between UI (View), presentation logic (VM), & underlying data (Model)
- Layers of Abstraction
  - VM acts as layer that abstracts the View from the Model,
    - VM keeps UI concerns separate from business logic
- Testability & Isolation
  - VM can be tested in isolation without needing the actual UI components

Why do we use MVVM?

- Business Logic
  - VM centralizes presentation logic & business rules
  - Use cases remain distinct components
  - VM coordinates use case execution as needed
  - This maintains a separation of concerns & modularity

- Data Flow
  - VM prepares & updates data for the View
    - Aligns with principles of CA by maintaining controlled flow of data from VM to View

Why do we use MVVM?

- Dependency Injection
  - VM can receive services & repositories via dependency injection
    - Enables VM to interact with external data sources & services without being tightly coupled to them
    - Helps to achieve the Dependency Inversion Principle within the context of CA

spaceteams

A clean architecture-inspired app

Example: TODO app

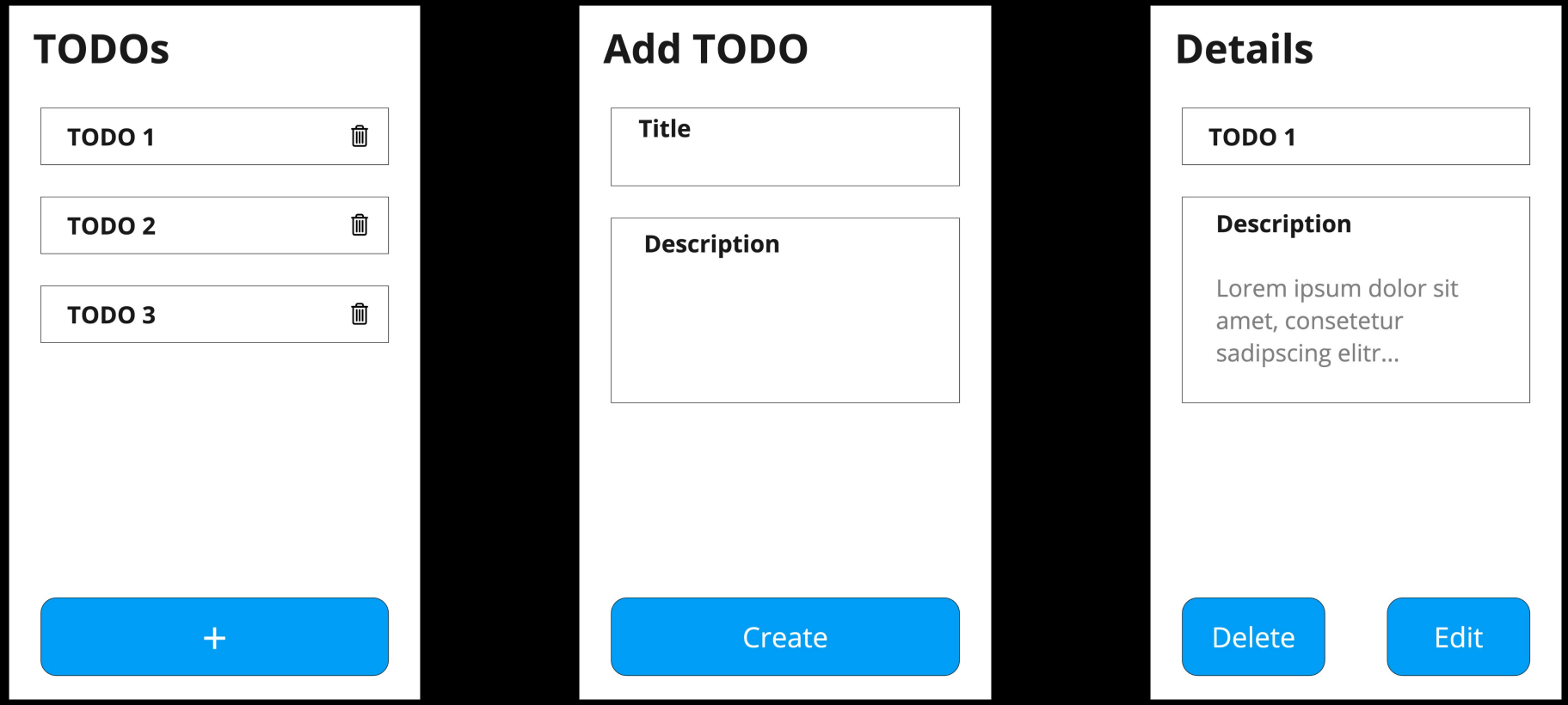
spaceteams

Example: TODO app

Features

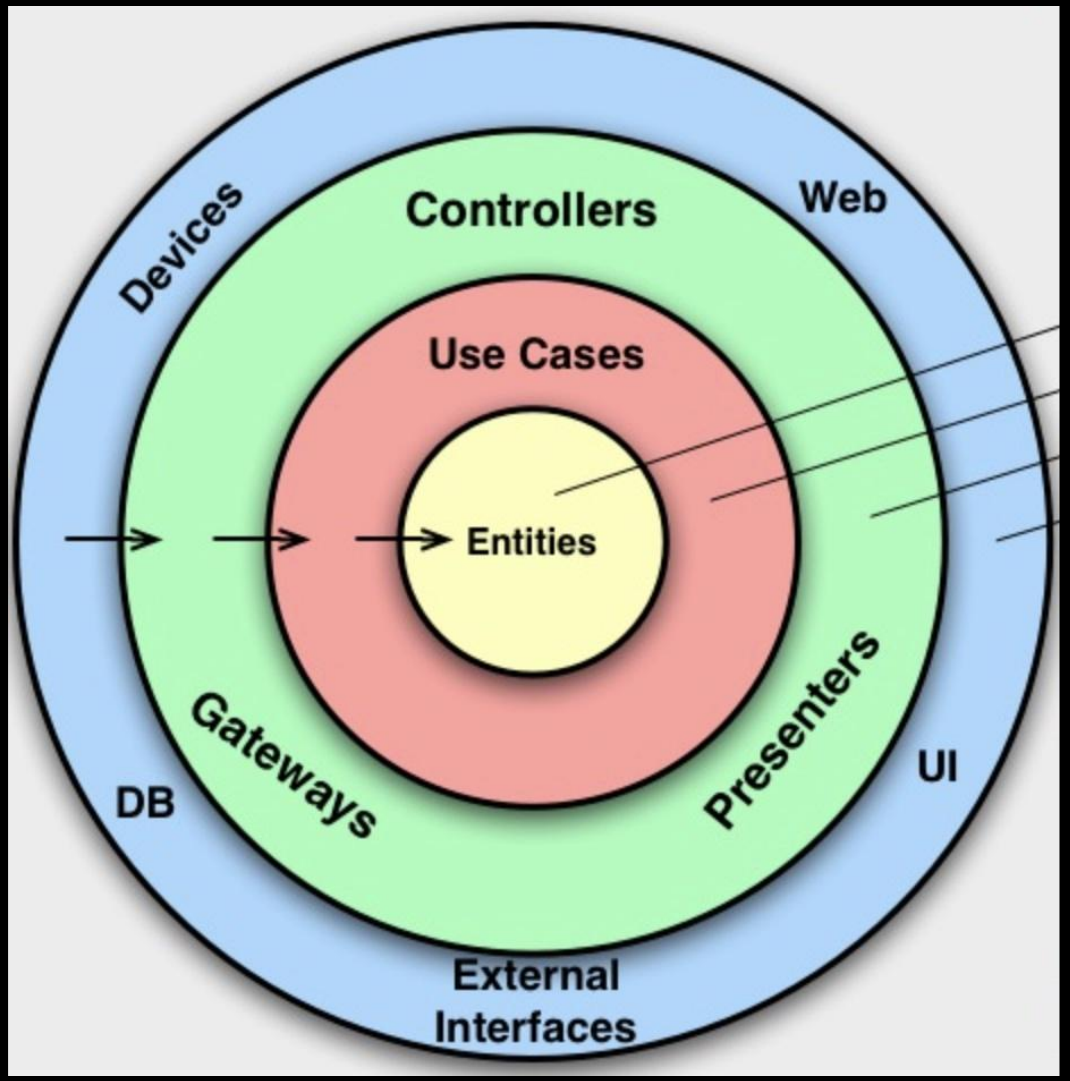
- List Todos
- Create a Todo
- Delete an existing Todo
- Update an existing Todo

UI


TODOs
TODO 1
TODO 2
TODO 3
+
Add TODO
Title
Description
Create
Details
TODO 1
Description
Lorem ipsum dolor sit amet, consetetur sadipscing elitr...
Delete
Edit

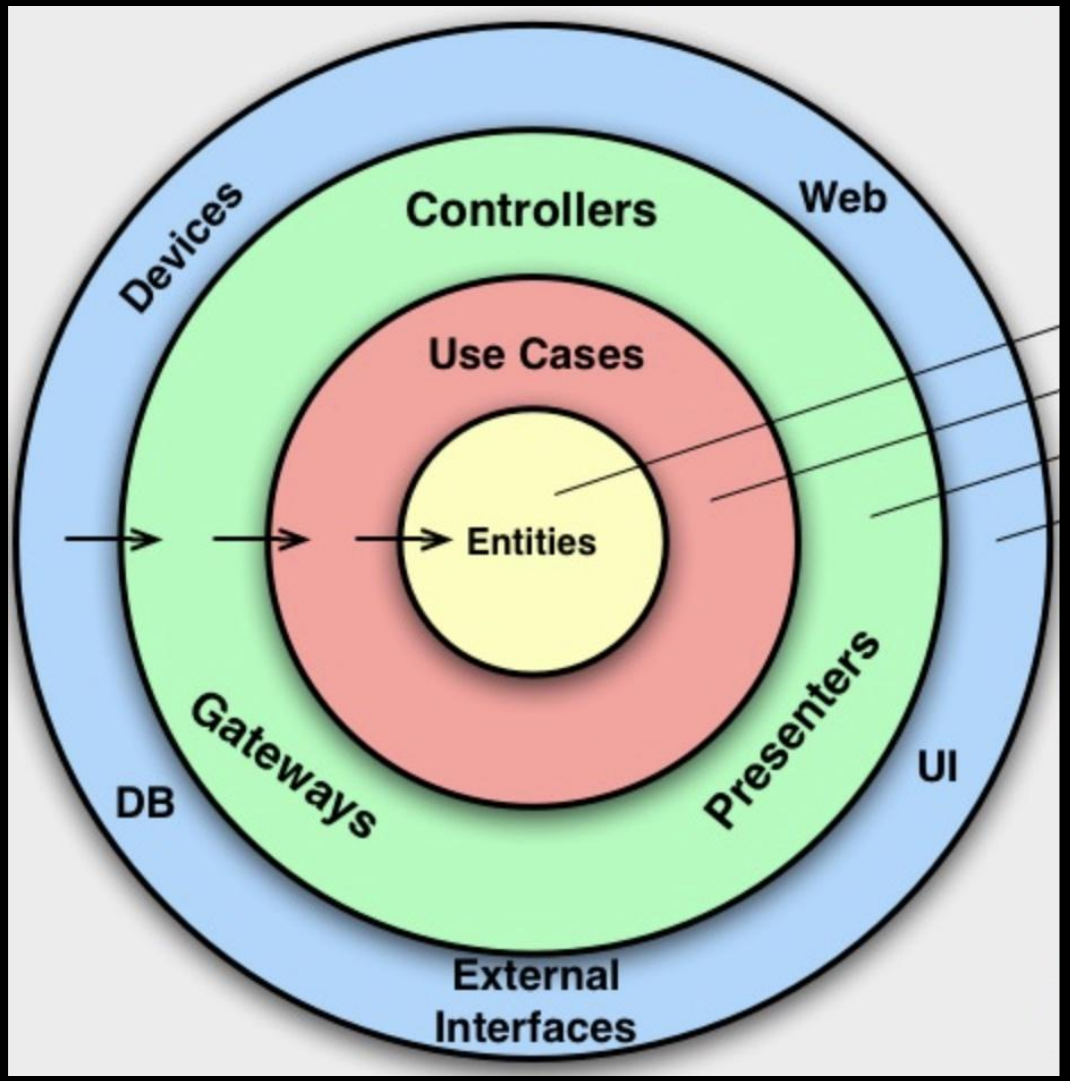
What layer?

- Entities:
  - Todo
- Use Cases:
  - Get Todos
  - Get Todo
  - Create Todo
  - Update Todo
  - Delete Todo

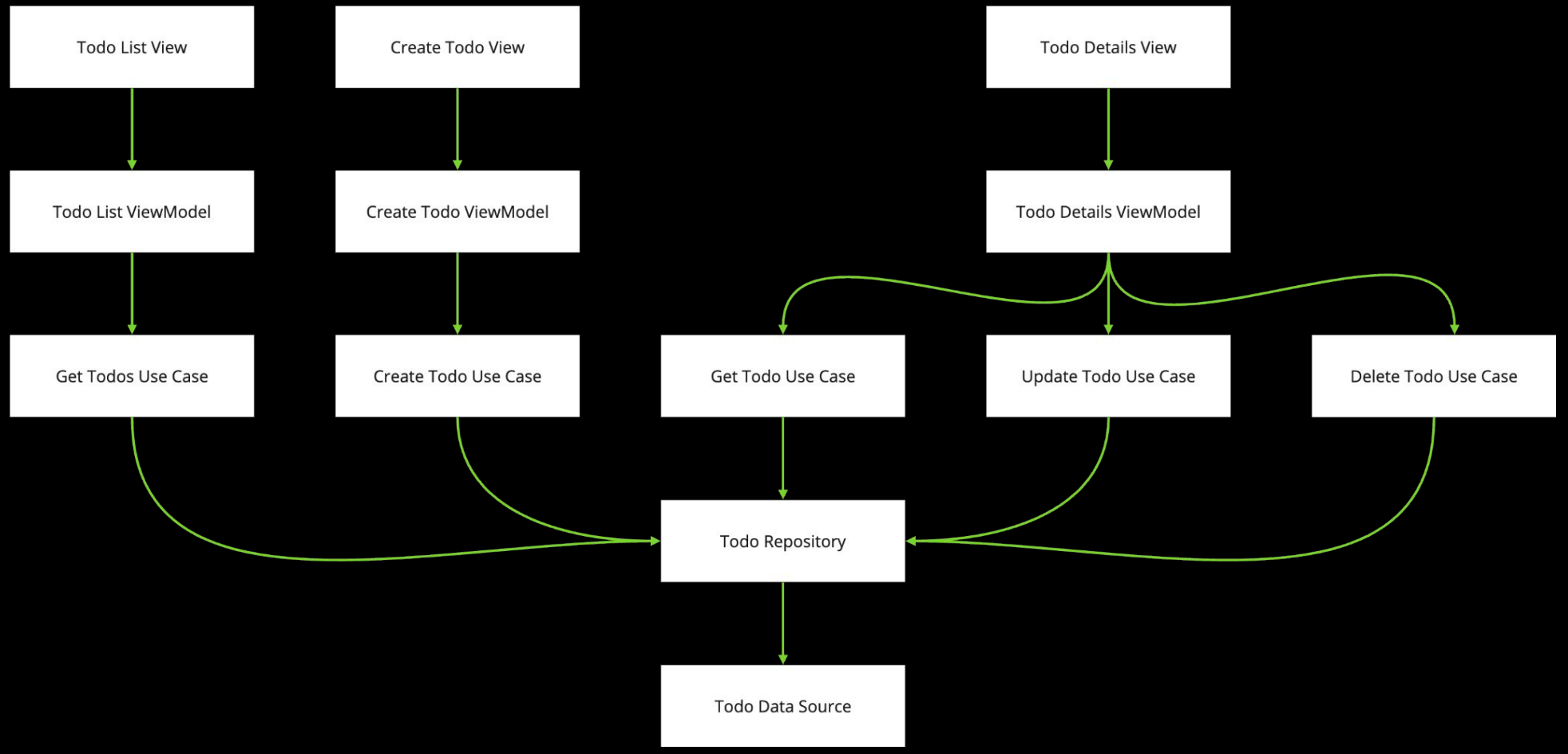


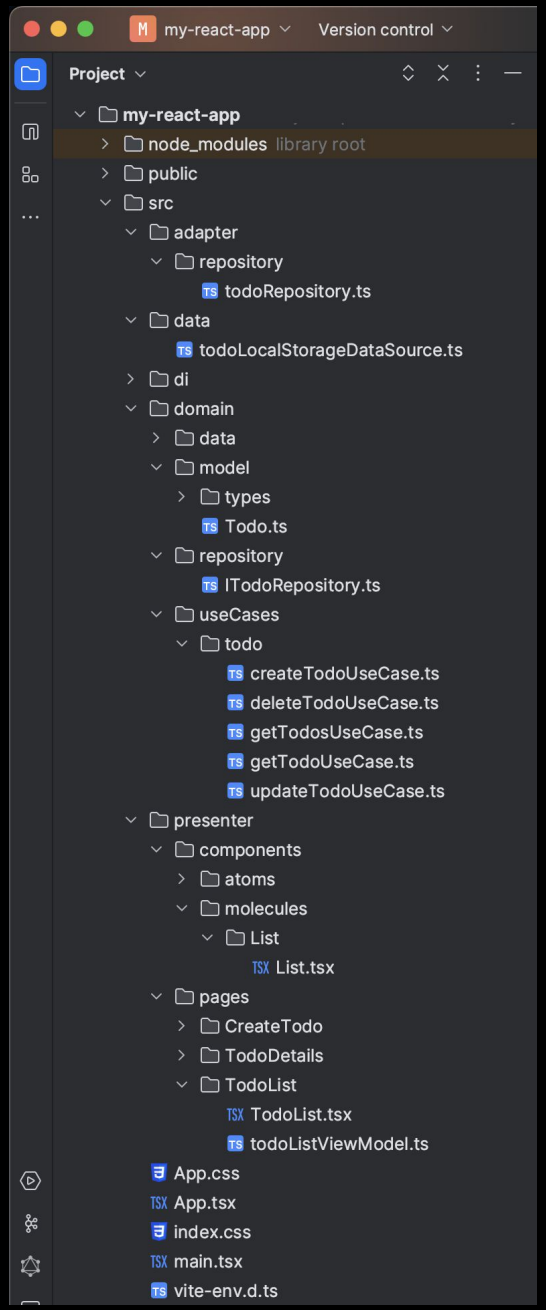
What layer?

- Interface Adapters:
  - Presenters
    - Todo List View & VM
    - Create Todo View & VM
    - Todo Details View & VM
  - Todo Repository
- Frameworks & Drivers:
  - Todo DB / Store



Application Flow




my-react-app
Version control
Project
my-react-app
node_modules library root
public
src
adapter
repository
todoRepository.ts
data
todoLocalStorageDataSource.ts
di
domain
data
model
types
Todo.ts
repository
ITodoRepository.ts
useCases
todo
createTodoUseCase.ts
deleteTodoUseCase.ts
getTodosUseCase.ts
getTodoUseCase.ts
updateTodoUseCase.ts
presenter
components
atoms
molecules
List
List.tsx
pages
CreateTodo
TodoDetails
TodoList
TodoList.tsx
todoListViewModel.ts
App.css
App.tsx
index.css
main.tsx
vite-env.d.ts

View

```
export const TodoList: FC = () => {
    const navigate : NavigateFunction  = useNavigate()

    const {todos, deleteItem} = DI.resolve( name: "todoListViewModel")

    return <div style={{display: "flex", flexDirection: "column", rowGap: "1rem"}}>
        <Header title="TODOs"/>
        <List
            items={todos}
            onItemClick={todo : Todo  => navigate( to: `/todo/detail/${todo.id}`)}
            onItemDelete={todo : Todo  => deleteItem(todo.id)}
        />
        <Button label={"+"} onClick={() => navigate( to: "/todo/create")}/>
    </div>
}
```

Dependency Injection

```
import {asFunction, asValue, createContainer} from "awilix";

const container : AwilixContainer<any>  = createContainer();

container.register( nameAndRegistrationPair: {
    todoRepository: asFunction(todoRepository),
    todoDataSource: asValue(TodoLocalStorageDataSource),
    todoListViewModel: asFunction(todoListViewModel),
    createTodoViewModel: asFunction(createTodoViewModel),
    todoDetailsViewModel: asFunction(todoDetailsViewModel),
    getTodosUseCase: asFunction(getTodosUseCase),
    deleteTodoUseCase: asFunction(deleteTodoUseCase),
    createTodoUseCase: asFunction(createTodoUseCase),
    getTodoUseCase: asFunction(getTodoUseCase),
    updateTodoUseCase: asFunction(updateTodoUseCase),
})

5+ usages
export const DI : AwilixContainer<any>  = container
```

ViewModel

```
export const todoListViewModel = ({getTodosUseCase, deleteTodoUseCase}: Dependencies) : {deleteTodo: (id: Id) => Promise<void>, todos: ...}  => {
    const [todos : Todo[] , setTodos : React.Dispatch<React.SetStateA... ] = useState<Todo[]>( initialState: []);

    1 usage
    const getTodos = async () : Promise<void>  => {
        const result : Todo[]  = await getTodosUseCase.execute()
        setTodos(result)
    }

    no usages
    const deleteTodo = async (id: Id) : Promise<void>  => {
        await deleteTodoUseCase.execute(id)
        setTodos(todos.filter(todo : Todo  => todo.id !== id))
    }

    1 usage
    const sortById = (prevTodo: Todo, todo: Todo) : number  => prevTodo.id < todo.id ? -1 : prevTodo.id > todo.id ? 1 : 0

    useEffect( effect: () : void  => {
        void getTodos()
    }, deps: [])

    return {todos: todos.sort(sortById), deleteTodo}
}
```

Use Cases

```
export const getTodosUseCase = ({todoRepository}: Dependencies): UseCase<Todo[]> => ({
    execute: () => todoRepository.get()
})
```

```
export const deleteTodoUseCase = ({todoRepository}: Dependencies): UseCaseWithParams<void, Id> => ({
    execute: (id: Id) => todoRepository.delete(id)
})
```

Repository

```
export const todoRepository = ({todoDataSource}: Dependencies): ITodoRepository => {
    1 usage
    const get = () => todoDataSource.getAll()

    1 usage
    const getById = (id: Id) => todoDataSource.getOne(id)

    1 usage
    const create = (title: string, description: string) =>
        todoDataSource.createOne( todo: {title, description})

    1 usage
    const update = (id: Id, title: string, description: string) =>
        todoDataSource.updateOne( todo: {id, title, description})

    1 usage
    const deleteTodo = (id: Id) => todoDataSource.deleteOne(id)

    return {get, getById, create, update, delete: deleteTodo}
}
```

Data Source

```
const COLLECTION_NAME: string = "todos"

5+ usages
export const getAll = (): Promise<Todo[]> => {
    try {
        const result : string | null  = localStorage.getItem(COLLECTION_NAME)
        return result !== null ? JSON.parse(result) : []
    } catch (error) {
        return Promise.reject(error)
    }
}

1 usage
export const deleteOne = async (id: Id): Promise<void> => {
    const todos : Todo[]  = (await getAll()).filter(({id: todoId : string }) : boolean  => todoId !== id)

    localStorage.setItem(COLLECTION_NAME, JSON.stringify(todos))
}
```

Thanks!

Any questions?

https://spaceteams.de