

01

# Clean Architecture

COURS TDD



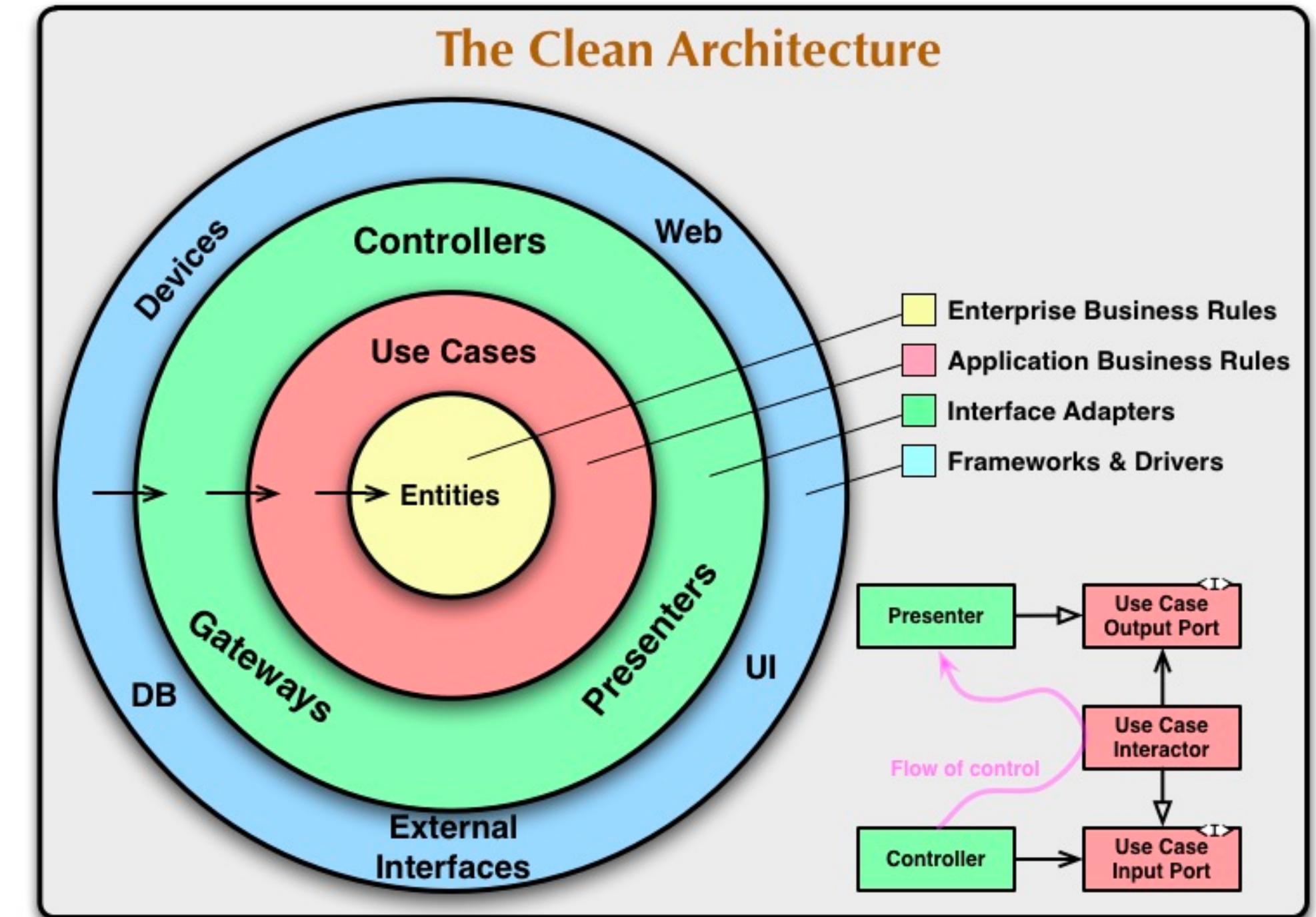
CentraleSupélec

Tomas Gonzalez  
Imane Largou  
Guillaume Malle  
Daniel Stulberg Huf

# Sommaire

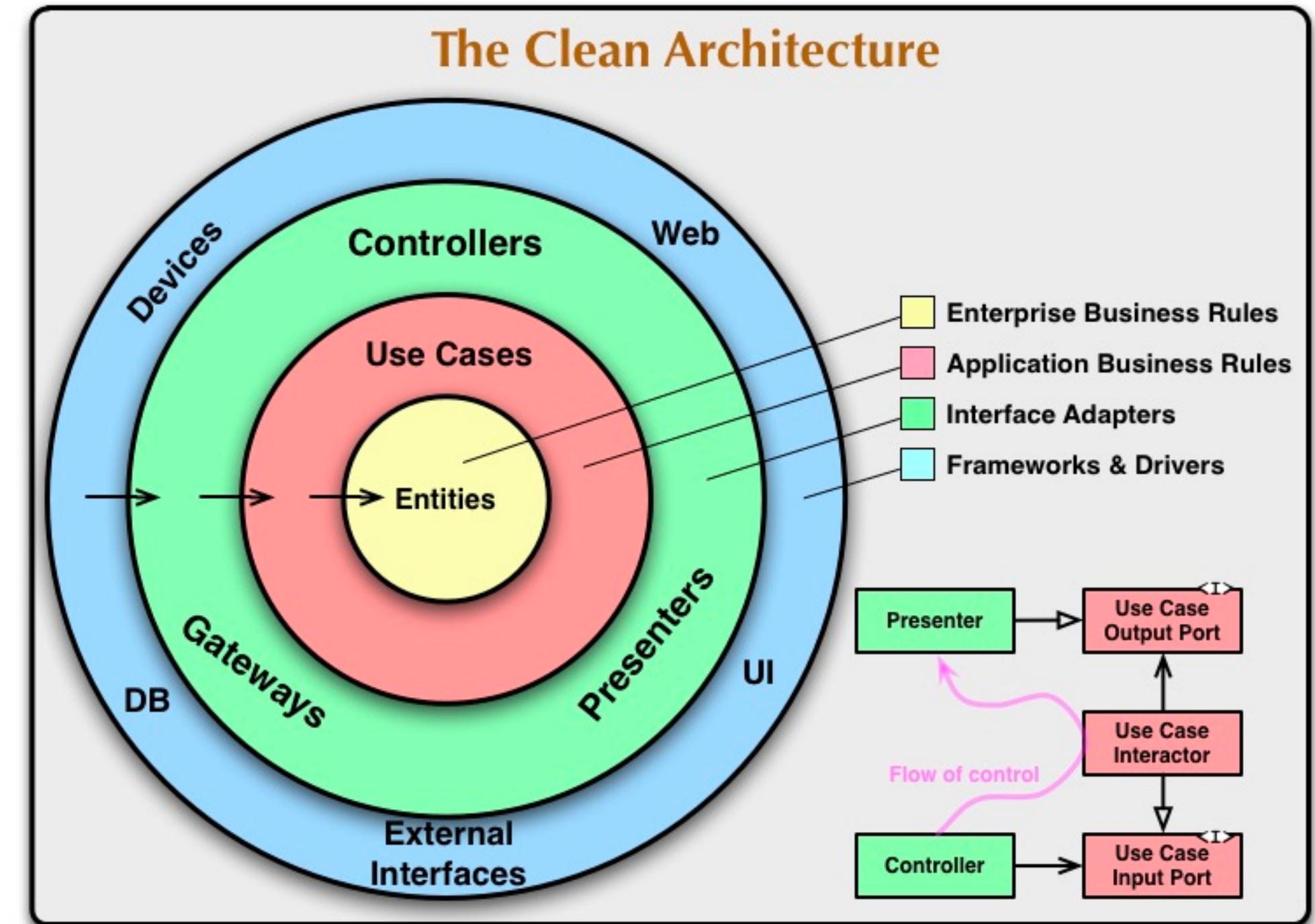
- 
- 01 PRÉSENTATION DE LA CLEAN ARCHITECTURE
  - 02 CLEAN ARCHI + TDD
  - 03 CAS APPLIQUÉ : COMMONCENT\$
  - 04 DEMO & RETOUR D'EXPÉRIENCE

- **Idée principale** : séparer le code en couches indépendantes entre elles.
  - Les dépendances du code source ne peuvent pointer que vers l'intérieur du cercle.
  - Séparation en couches abstraction
    - **Entities**: règles de gestion à l'échelle de l'entreprise
    - **Use Cases**: règles de gestion spécifiques à l'application
    - **Controllers**: les adaptateurs qui convertissent les données entre les use cases/entities et les composants externes
    - **Frameworks**: Database, framework web ...
- volatilité ↑ ↓



source : Robert C. Martin (Uncle Bob)

- La testabilité de chaque brique logicielle est traitée de manière indépendante.
- Plusieurs façons différentes d'implémentation (pas seulement le standard à quatre couches).
- Tout objet déclaré dans une couche externe ne doit pas être appelé dans une couche interne.
- La communication d'une couche intérieure vers une couche extérieure se fait à travers des interfaces.
- L'injection de dépendance permet de mocker les dépendances.

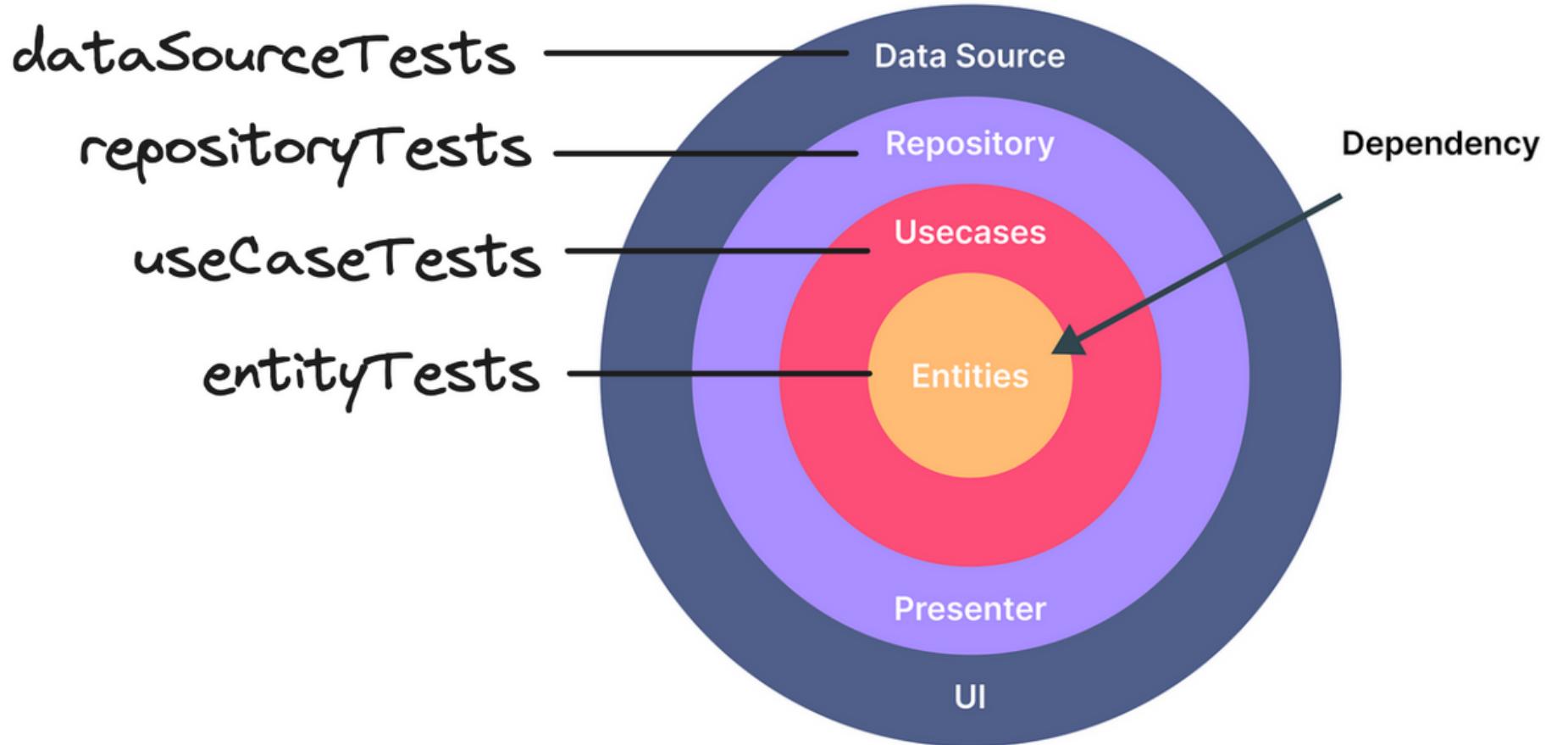


## Avantages

- Séparation des responsabilités facilite tester chaque couche indépendamment.
- L'injection de dépendance est facilement testable avec des mocks.
- On peut faire du refactoring sans affecter d'autres couches.

## Inconvénients

- Setup des test peut être compliqué à mettre en place.
- Un système simple peut devenir complexe en s'adaptant à la méthodologie



# CAS APPLIQUÉ :

## APPLICATION DE RÉPARTITION DE DÉPENSES



### GROUPES ET MEMBRES

- Un groupe peut être créé avec un nom et une liste de membres.
- Des membres peuvent être ajoutés ultérieurement à un groupe.
- Un membre a un nom.



### SOLDES

- Un groupe a un solde total, qui est la somme de toutes les dépenses enregistrées par ses membres.
- Un groupe a un solde différentiel, qui est la décomposition de ce que chaque membre doit à tous les autres membres du groupe.
- Un groupe a un solde différentiel simplifié. Ex : Si A doit à B, et B doit à C, le solde simplifié considère que A doit à C.



## CommonCent\$



### DÉPENSES

- Un membre peut enregistrer une dépense dans un groupe.
- Une dépense a un titre, un montant, une date, un payeur et une répartition.
- Les répartitions peuvent être enregistrées en valeur absolue ou en pourcentage, de manière égale ou inégale entre les membres du groupe.

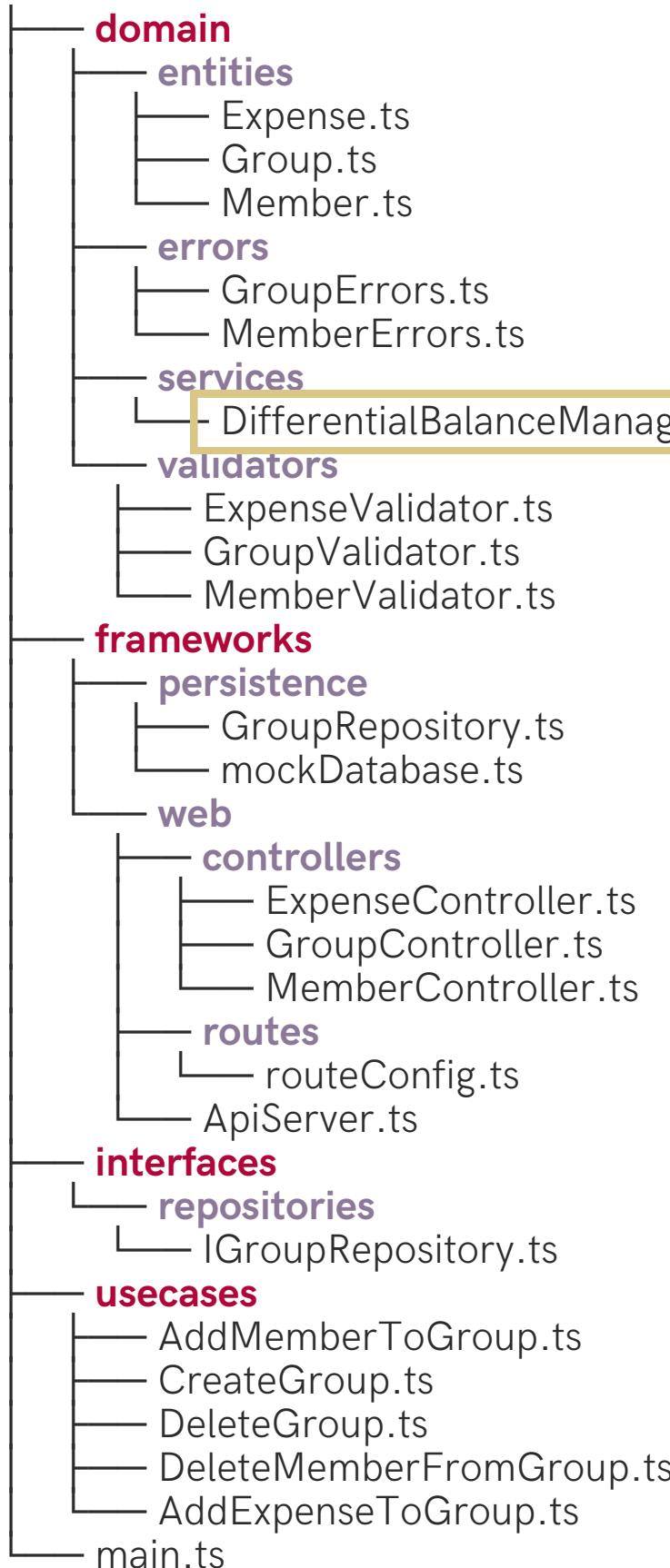


### REMBOURSEMENT

- Un membre peut rembourser partiellement ou totalement ce qu'il doit à un autre membre.
- Le solde différentiel entre le payeur et le receveur changera en fonction de ce qui a été payé.

# La Clean Archi dans notre appli

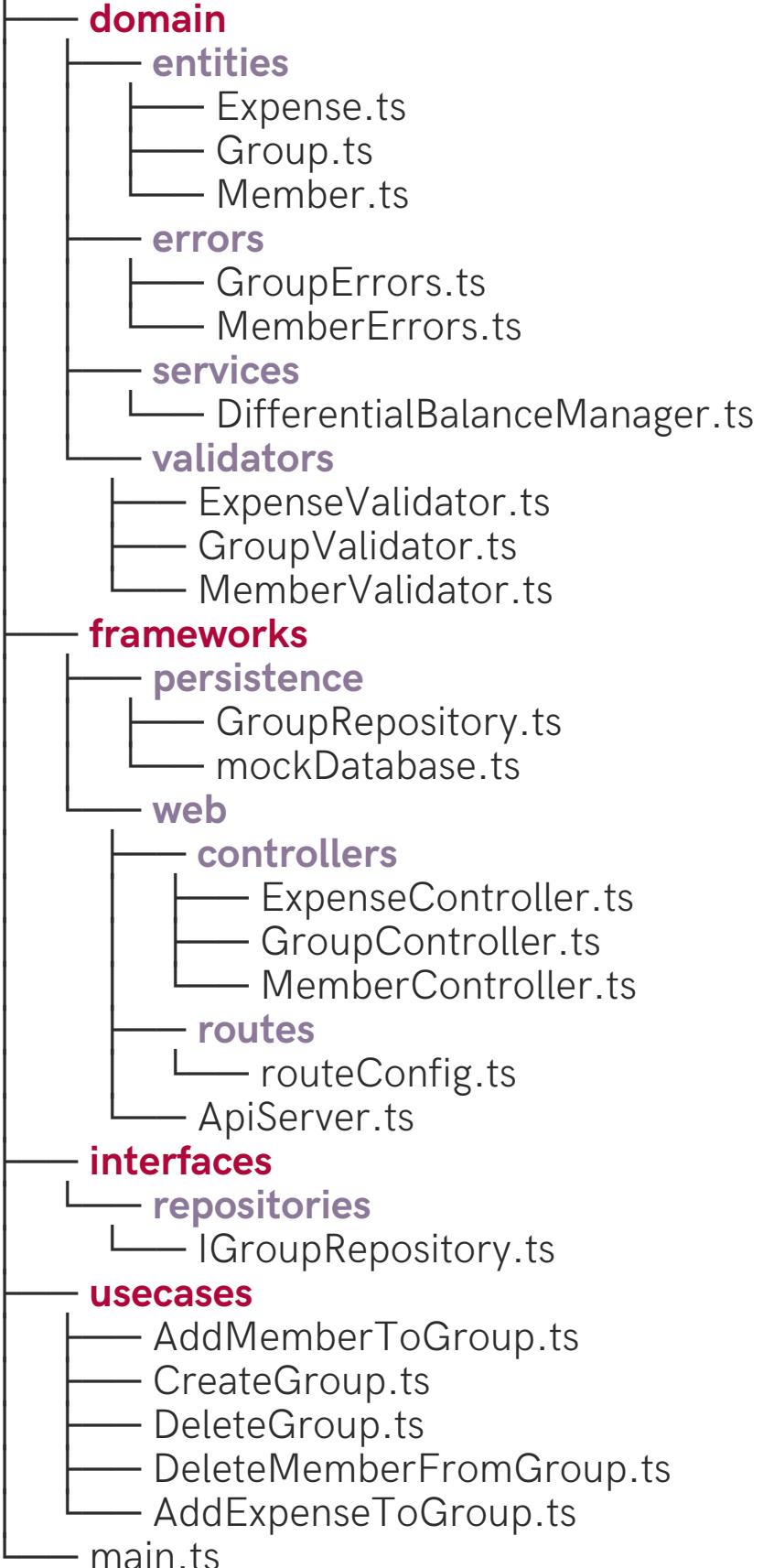
07



## src/domain/entities/Group.ts

```
● ● ●
1 import { Expense } from "./Expense";
2 import { Member } from "./Member";
3 import { MemberNotFoundError } from "../errors/GroupErrors";
4 import { DifferentialBalanceManager } from "../services/DifferentialBalanceManager";
5
6 export class Group {
7   id: string;
8   name: string;
9   members: Member[];
10  expenses: Expense[];
11  total_balance: number;
12  private balanceManager: DifferentialBalanceManager;
13
14  constructor(id: string, name: string, members: Member[] = []) {
15    this.id = id;
16    this.name = name;
17    this.members = members;
18    this.expenses = [];
19    this.total_balance = 0;
20    this.balanceManager = new DifferentialBalanceManager(members);
21  }
22
23  addMember(member: Member): void {
24    this.members.push(member); injection de dépendance
25    this.balanceManager.addMember(member);
26  }
}
```

# La Clean Archi dans notre appli



## src/usecases/AddMemberToGroup.ts

08

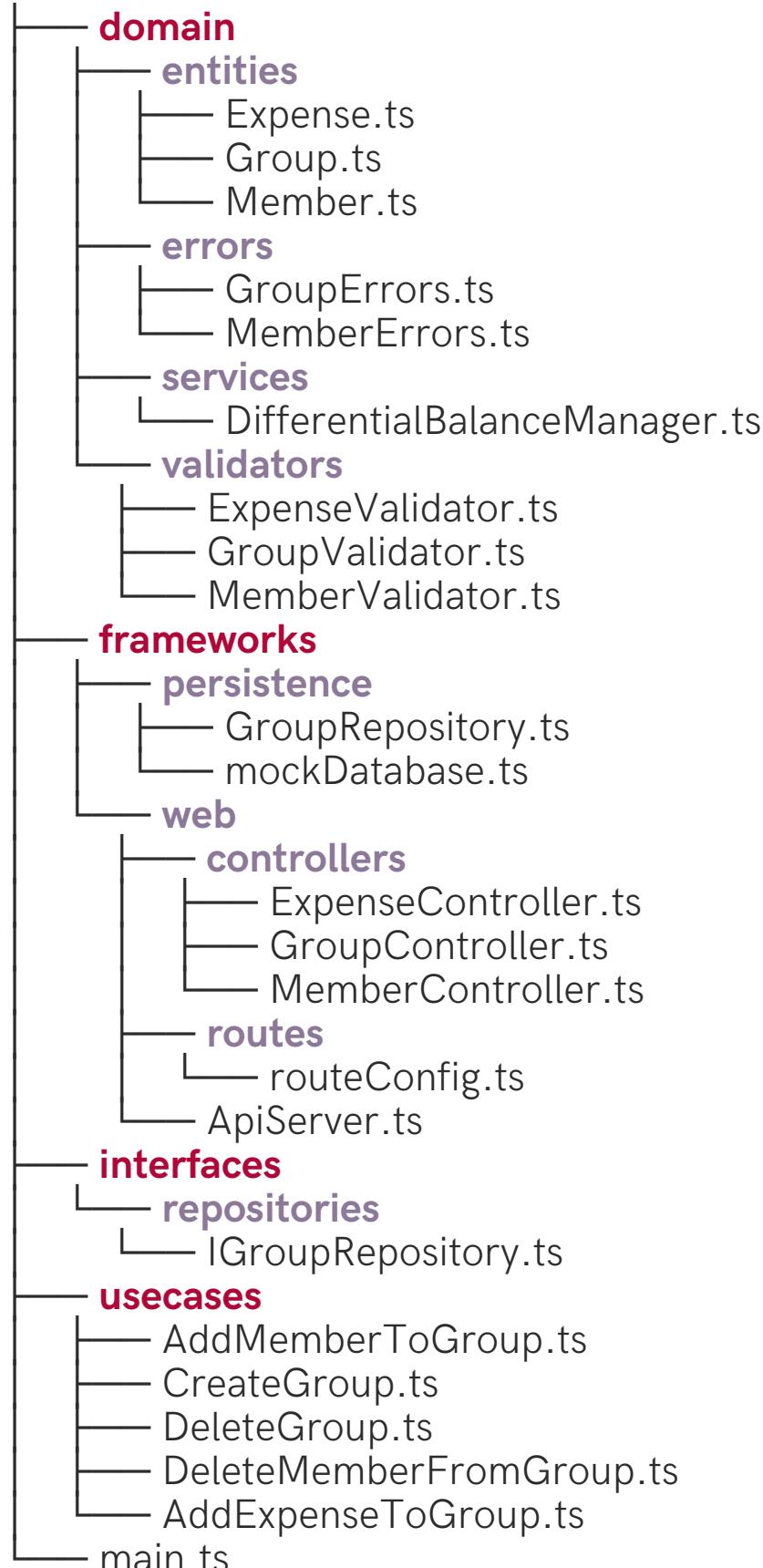


respect de la règle de dépendance

```
1 import { IGroupRepository } from "../interfaces/repositories/IGroupRepository";
2 import { Member } from "../domain/entities/Member";
3 import { GroupNotFoundError } from "../domain/errors/GroupErrors";
4 import { MemberValidator } from "../domain/validators/MemberValidator";
5
6 export class AddMemberToGroup {
7     private repository: IGroupRepository;
8
9     constructor(repository: IGroupRepository) {
10         this.repository = repository;
11     }
12
13     execute(groupId: string, memberName: string): any {
14         const group = this.repository.findGroup(groupId);
15         if (!group) {
16             throw new GroupNotFoundError();
17         }
18
19         MemberValidator.validateMemberName(memberName);
20         MemberValidator.validateMemberIsNotAlreadyInGroup(group.members, memberName);
21
22         const member = new Member(memberName);
23         group.addMember(member);
24
25         return group;
26     }
27 }
28
```

# La Clean Archi dans notre appli

09

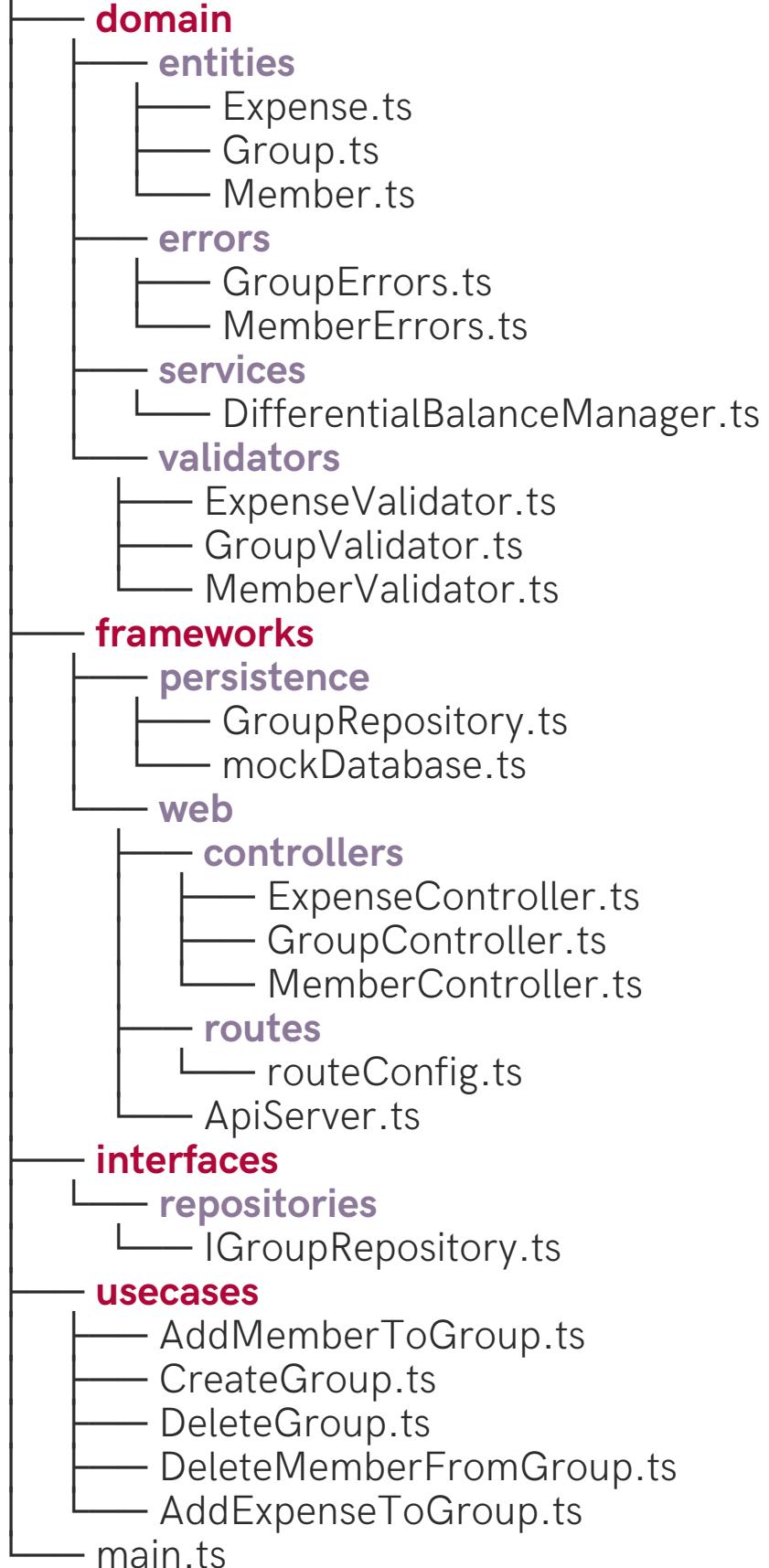


## src/interfaces/repositories/IGroupRepository.ts



```
1 import { Group } from "../../domain/entities/Group";
2
3 export interface IGroupRepository {
4     addGroup(group: Group): void;
5     deleteGroup(groupId: string): boolean;
6     findGroup(groupId: string): Group | undefined;
7 }
```

# La Clean Archi dans notre appli



## src/frameworks/web/ApiServer.ts

10

```
● ● ●

1 import express, { Application } from 'express';
2 import { GroupController } from './controllers/GroupController';
3 import { MemberController } from './controllers/MemberController';
4 import { ExpenseController } from './controllers/ExpenseController';
5 import { RoutePaths } from './routes/routeConfig';

6
7 export class ApiServer {
8     private app: Application;
9
10    constructor(groupController: GroupController, memberController: MemberController, expenseController: ExpenseController) {
11        this.app = express();
12        this.app.use(express.json());
13        this.app.post(RoutePaths.createGroup, groupController.createGroup);
14        this.app.delete(RoutePaths.deleteGroup, groupController.deleteGroup);
15        this.app.post(RoutePaths.addMember, memberController.addMemberToGroup);
16        this.app.delete(RoutePaths.deleteMember, memberController.deleteMemberFromGroup);
17        this.app.post(RoutePaths.addExpense, expenseController.addExpenseToGroup);
18    }
19
20    public getApp(): Application {
21        return this.app;
22    }
23
24    public static run(port: number, groupController: GroupController, memberController: MemberController, expenseController: ExpenseController) {
25        const server = new ApiServer(groupController, memberController, expenseController);
26        server.getApp().listen(port, () => {
27            console.log('Server is running on http://localhost:' + port);
28        });
29    }
30 }
```

# Le TDD dans notre appli

11

```
controllerTests
├── addMembertoGroupController.test.ts
├── createGroupController.test.ts
└── deleteGroupController.test.ts
└── deleteMemberFromGroupController.test.ts
└── AddExpenseToGroupController.test.ts

entityTests
└── Expense.test.ts
└── Group.test.ts
└── Member.test.ts

persistenceTests
└── GroupRepository.test.ts
└── mockDatabase.test.ts

useCaseTests
└── addMemberToGroup.test.ts
└── createGroup.test.ts
└── deleteGroup.test.ts
└── deleteMemberFromGroup.test.ts
└── AddExpenseToGroup.test.ts

webTests
└── apiServer.test.ts
```

la règle de dépendance  
reste toujours valide pour  
les tests !

## tests/useCaseTests/addExpenseToGroup.test.ts

```
● ● ●

1  describe("Member Adds Expense To Group Use Case", () => {
2
3    let groupRepository: GroupRepository;
4    let createGroup: CreateGroup;
5    let memberAddsExpenseToGroup: MemberAddsExpenseToGroup;
6    let addMemberToGroup: AddMemberToGroup;
7
8    beforeEach(() => {
9      resetMockDatabase();
10     groupRepository = new GroupRepository();
11     createGroup = new CreateGroup(groupRepository);
12     memberAddsExpenseToGroup = new MemberAddsExpenseToGroup(groupRepository);
13     addMemberToGroup = new AddMemberToGroup(groupRepository);
14   });
15
16  describe("Expense addition", () => {
17    it("should update differential balances correctly after adding an expense in percentages", () => {
18      const groupName = "Trip";
19      const members = [new Member("Alice"), new Member("Bob")];
20      const group = createGroup.execute(groupName, members);
21
22      const title = "Lunch";
23      const amount = 20;
24      const payerName = "Alice";
25      const date = new Date();
26      const isPercentual = true;
27      const split = {
28        "Alice": 80,
29        "Bob": 20
30      };
31
32      memberAddsExpenseToGroup.execute(group.id, title, amount, payerName, date, isPercentual, split);
33
34      expect(group.getDifferentialBalance("Alice", "Bob")).toBe(4);
35      expect(group.getDifferentialBalance("Bob", "Alice")).toBe(-4);
36    });
  
```

# Le TDD dans notre appli

12

```
controllerTests
├── addMembertoGroupController.test.ts
├── createGroupController.test.ts
└── deleteGroupController.test.ts
└── deleteMemberFromGroupController.test.ts
└── AddExpenseToGroupController.test.ts

entityTests
└── Expense.test.ts
└── Group.test.ts
└── Member.test.ts

persistenceTests
└── GroupRepository.test.ts
└── mockDatabase.test.ts

useCaseTests
└── addMemberToGroup.test.ts
└── createGroup.test.ts
└── deleteGroup.test.ts
└── deleteMemberFromGroup.test.ts
└── AddExpenseToGroup.test.ts

webTests
└── apiServer.test.ts
```

## tests/controllerTests/createGroupController.test.ts

```
● ● ●

1  it("createGroup should successfully create a group with no members and return the appropriate response", async () => {
2      // Arrange
3      mockReq.body = { name: "Adventure Club" };
4
5      const expectedResponse = {
6          id: "1",
7          name: "Adventure Club",
8          members: []
9      };
10
11     mockCreateGroup.execute.mockReturnValue(
12         new Group("1", "Adventure Club", [])
13     );
14
15     // Act
16     await groupController.createGroup(mockReq, mockRes);
17
18     // Assert
19     expect(mockCreateGroup.execute).toHaveBeenCalledWith("Adventure Club");
20     expect(mockRes.status).toHaveBeenCalledWith(201);
21     expect(mockRes.json).toHaveBeenCalledWith(expectedResponse);
22 });
```

# Demo & Retour d'expérience



# Les avantages de la Clean Archi

14

## Indépendance vis-a-vis du framework / UI / base de données

Les couches externes +techniques peuvent changer sans que cela affecte la logique métier.



## Testabilité

Les couches sont définis avec des interfaces claires, rendant les modules plus faciles à tester isolément ou à mocker pour les tests d'intégration.



## Développement parallèle

Différentes équipes peuvent travailler sur différentes couches simultanément, en faisant des changements robustes sans se gêner mutuellement.

# Les inconvénients de la Clean Archi

15

## 👎 Complexité initiale

Tâches complexes et qui prennent du temps, surtout dans les petites applications où une architecture plus simple pourrait suffire.

## 👎 Sur-ingénierie

Surcharge pour des projets qui ne nécessitent pas un tel niveau de modularité pour justifier la complexité supplémentaire.

## 👎 Gestion des dépendances

Bien que la Clean Archi vise à réduire les dépendances, si elles sont nombreuses ça peut devenir un défi en soi.



16

CLEAN ARCHITECTURE

**MERCI!**

QUESTIONS ?

SINON... ON VOUS SOUHAITE UNE VIE HEUREUSE! ☺