**دستور و روش و کدهای پیاده سازی معماری تمیز**

**به دلایل جوی کار و وقت کافی نتوانستم پیاده سازیش کنم**

**یاسین شهریاری کارشناسی ناپیوسته ترم6**

پیاده‌سازی معماری پاک (**Clean Architecture**) در **Rust** نیازمند تقسیم برنامه به لایه‌های مستقل است تا هر لایه جدا از دیگری باشد. این معماری شامل لایه‌های زیر است:

1. **Entities**: قوانین تجاری و منطق برنامه.
2. **Use Cases**: عملیات و جریان‌های کاری برنامه.
3. **Interface Adapters**: تبدیل داده بین لایه‌ها.
4. **Infrastructure**: لایه‌ای که ارتباط با سیستم‌های خارجی را مدیریت می‌کند (پایگاه داده، سرویس‌های وب).

در اینجا مثالی از یک برنامه ساده مدیریت **کارهای روزانه (Todo App)** ارائه می‌شود.

**ساختار پروژه**

css

Copy code

src/

│

├── entities/

│ └── todo.rs

├── use\_cases/

│ └── manage\_todo.rs

├── interfaces/

│ └── todo\_repository.rs

├── infrastructure/

│ └── in\_memory\_repository.rs

├── main.rs

**گام 1: تعریف موجودیت‌ها (Entities)**

**entities/todo.rs**

rust

Copy code

#[derive(Clone, Debug)]

pub struct Todo {

pub id: usize,

pub task: String,

pub completed: bool,

}

**گام 2: تعریف موارد استفاده (Use Cases)**

**use\_cases/manage\_todo.rs**

rust

Copy code

use crate::entities::todo::Todo;

pub struct TodoManager {

pub todos: Vec<Todo>,

}

impl TodoManager {

pub fn new() -> Self {

Self { todos: vec![] }

}

pub fn add\_todo(&mut self, task: String) {

let id = self.todos.len() + 1;

let todo = Todo { id, task, completed: false };

self.todos.push(todo);

}

pub fn list\_todos(&self) -> &Vec<Todo> {

&self.todos

}

pub fn complete\_todo(&mut self, id: usize) {

if let Some(todo) = self.todos.iter\_mut().find(|t| t.id == id) {

todo.completed = true;

}

}

}

**گام 3: رابط‌ها (Interfaces)**

**interfaces/todo\_repository.rs**

rust

Copy code

use crate::entities::todo::Todo;

pub trait TodoRepository {

fn save(&self, todo: &Todo);

fn get\_all(&self) -> Vec<Todo>;

}

**گام 4: زیرساخت‌ها (Infrastructure)**

**infrastructure/in\_memory\_repository.rs**

rust

Copy code

use crate::entities::todo::Todo;

use crate::interfaces::todo\_repository::TodoRepository;

pub struct InMemoryTodoRepository {

pub todos: Vec<Todo>,

}

impl InMemoryTodoRepository {

pub fn new() -> Self {

Self { todos: vec![] }

}

}

impl TodoRepository for InMemoryTodoRepository {

fn save(&self, todo: &Todo) {

println!("Saving todo: {:?}", todo);

}

fn get\_all(&self) -> Vec<Todo> {

self.todos.clone()

}

}

**گام 5: نقطه ورود اصلی**

**main.rs**

rust

Copy code

mod entities;

mod use\_cases;

mod interfaces;

mod infrastructure;

use use\_cases::manage\_todo::TodoManager;

fn main() {

let mut manager = TodoManager::new();

manager.add\_todo("Learn Rust".to\_string());

manager.add\_todo("Build a project".to\_string());

println!("Todos:");

for todo in manager.list\_todos() {

println!("{:?}", todo);

}

manager.complete\_todo(1);

println!("After completing first task:");

for todo in manager.list\_todos() {

println!("{:?}", todo);

}

}

**توضیحات**

1. **Entities**: شامل ساختار Todo.
2. **Use Cases**: کلاس TodoManager عملیات افزودن، لیست‌کردن و تکمیل کارها را مدیریت می‌کند.
3. **Interfaces**: تعریف رابطی برای ذخیره داده‌ها.
4. **Infrastructure**: پیاده‌سازی یک مخزن داده در حافظه.

این معماری انعطاف‌پذیری و قابلیت نگهداری بالا را تضمین می‌کند