BLOCK NONCE #2: Dependency Injection

What?

Dependency Injection (DI) is a design pattern where an object receives its dependencies from an external source rather than creating them itself. It is like giving your code a toolbox instead of making it build its own tools.

Why?

Decouples components – making it way easier to swap parts without breaking everything.
Improves testability - you can inject fake dependencies (mocks or stubs) when testing.

How?

```rust
struct WalletService {
  provider: BlockchainProvider, // Without DI
}
impl WalletService {
  fn new() -> Self {
    WalletService {
      provider: BlockchainProvider::connect() // Hard-Coded dependency
    }
  }
}
```

Testing WalletService is difficult - you would need a real blockchain in your test....

```rust
struct WalletServiceDI<'a> {
  provider: &'a dyn BlockchainProviderTrait, // With DI
}
impl<'a> WalletServiceDI<'a> {
  fn new(provider: &'a dyn BlockchainProviderTrait) -> Self {
    WalletServiceDI { provider } // The provider is passed in
  }
}
```

Now you can simply use a mock blockchain and only test the behavior of WalletService.

TIP💡 It can also be optional - give the user the ability to provide the dependencies, but allow them to pass None to get the default dependency.