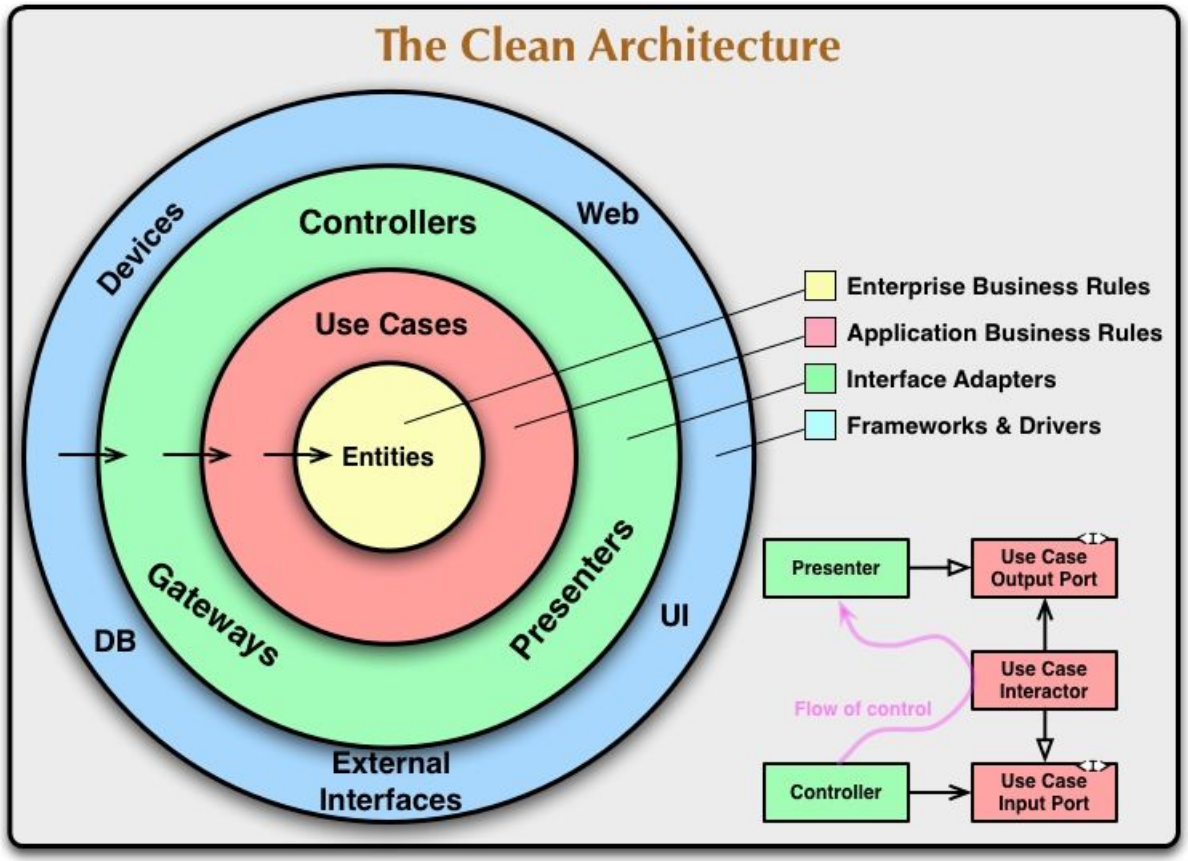
Clean Architecture for React

The non-boring way (the Melara way)

A bit of history

Clean Architecture is a software design philosophy introduced by **Robert C. Martin**, also known as "**Uncle Bob.**"

- **Guidelines and Principles**: Clean Architecture offers guidelines and principles for designing modular and maintainable software systems.
- **Separation of Concerns**: The primary goal is to achieve a clear separation of concerns, enhancing flexibility, scalability, and testability.
- **Flexibility**: Clean Architecture promotes adaptability to changes in technologies or frameworks by keeping core business logic independent.
- **Scalability**: The architecture is designed to scale efficiently as the software system evolves, ensuring it remains manageable.
- **Testability**: The separation of concerns facilitates easy testing of components, particularly the core business logic, contributing to overall system reliability.


The Clean Architecture
Devices
Web
Controllers
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port

The important bits

- **UI**: Data representation (see the text on screen) (Layer 4).
- **Presenter**: Adapts the data from the Use Case to the UI. (data.text to TextView.text)
- **Use case**: Individual piece of the application's functionality. (get the data object)
- **Entities**: Data structures (data object properties like text: string) (Layer 1).

My easy non boring way

What I hate?

- **Dependency injection**: Let's have short lived memory objects and long lived singletons without injection. (But let's keep the separation of concerns) with a SSOT Dependency Manager.

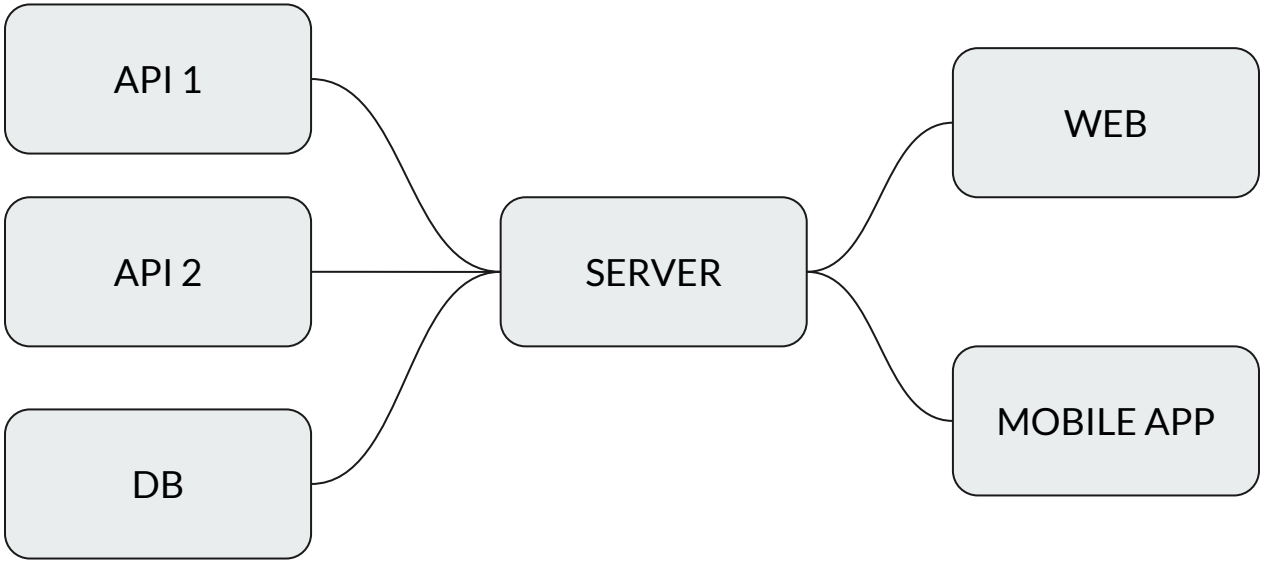
  const db = connect(string) => const function (dbInstance) => dependencyManager.getDbInstance

Part 1: Single source of truth

A Single Source of Truth (SSOT) server refers to a centralized data repository or database that serves as the authoritative and definitive source for a particular set of information within an organization

- **Consistency:** Absence of variations or discrepancies in data.
- **Accuracy:** Correctness and precision of data. Free from errors.
- **Reliability of information:** Trustworthiness and dependability of data.

Single place for all the requests



Part 2: Where Clean Architecture shines

Story: A pharmacy is expanding from Spain to Portugal. They have sensors for keeping storage temperature/pressure at a desired level. They want to monitor this data from an App.

What we currently have: Backend, a Climate API and a Website but everything is in Spanish.

Trouble: Spanish developers only speak Spanish/English and Portuguese developers only speak Portuguese/English.

Goal: Build a white label application that can handle both languages and both locations. App should display current storage temperature without changing the API language (yet). Common: English

Let's build a fake SSOT

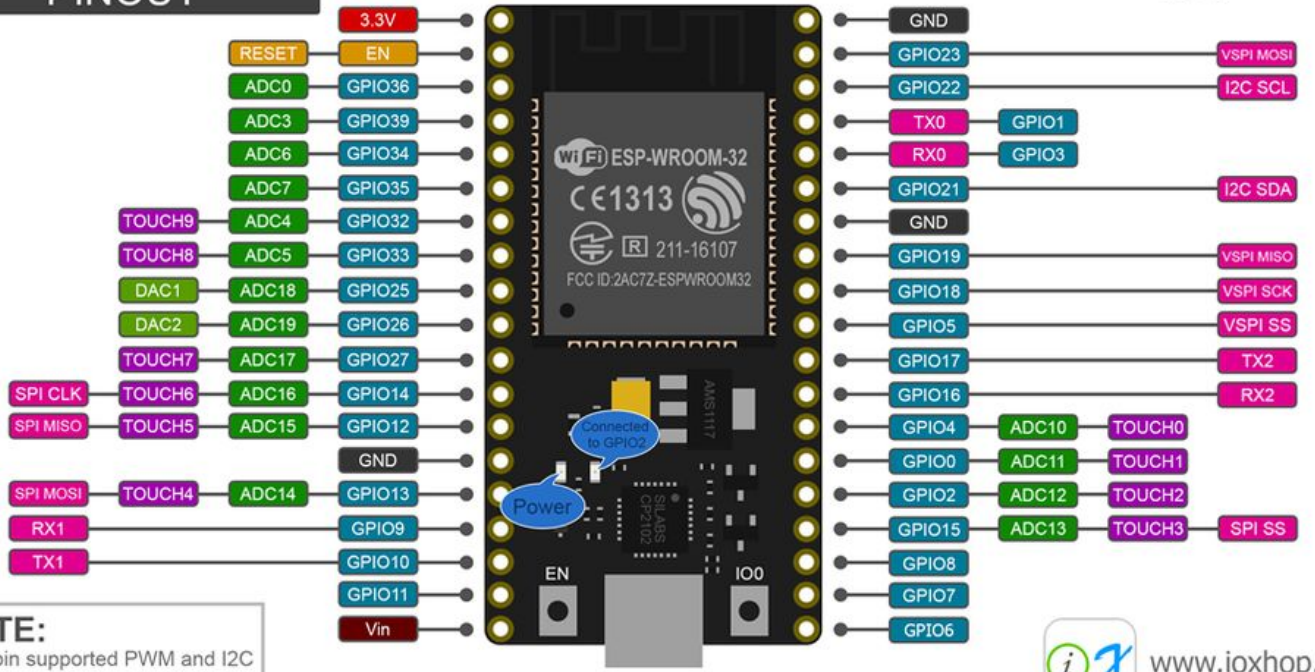
1. Backend web server
2. Temperature/Pressure API

Backend Web Server (Node32s)

1. Dual Core @ 80Mhz
2. Works with C/C++
3. Works with MicroPython
4. Has Wifi and Bluetooth
5. Can be used with Firmata and **Johnny-Five**

NodeMCU-32S

PINOUT



NOTE:

All pin supported PWM and I2C

Pin current 6mA (Max. 12mA)

www.ioxhop.com

Temperature/Pressure API: BMP085

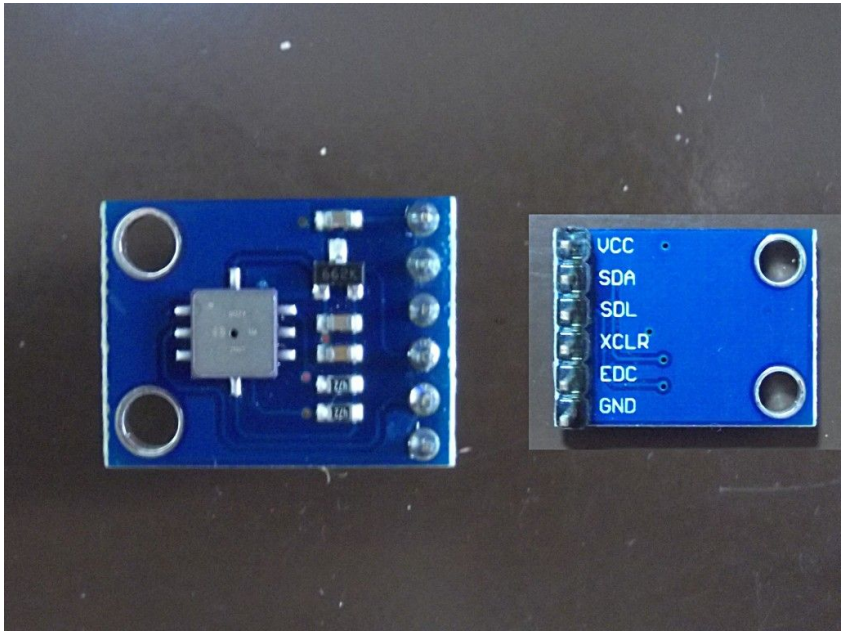
VCC - Positive - 3.3v

GND - Negative - Ground

It uses **I2C** (Inter-Integrated Circuit communication)

SDA: Data

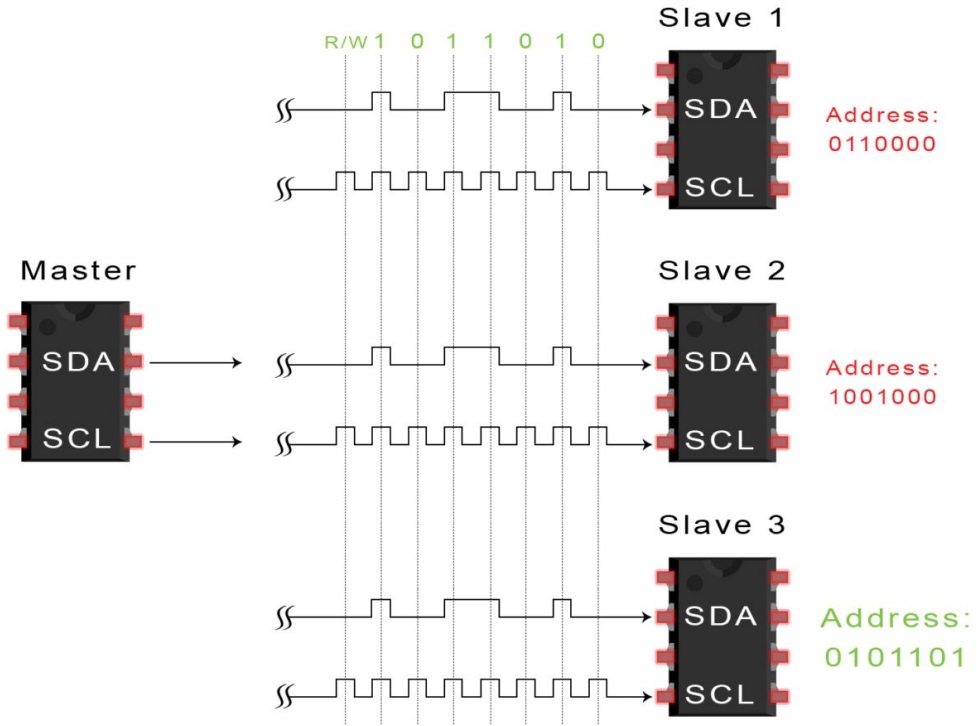
SCL/SDL: Clock

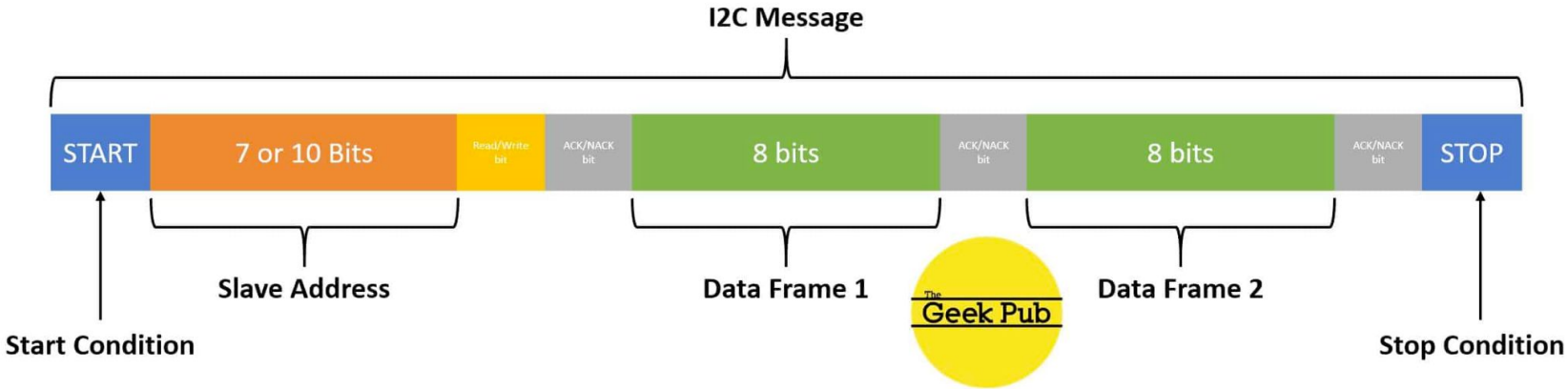


How I2C works

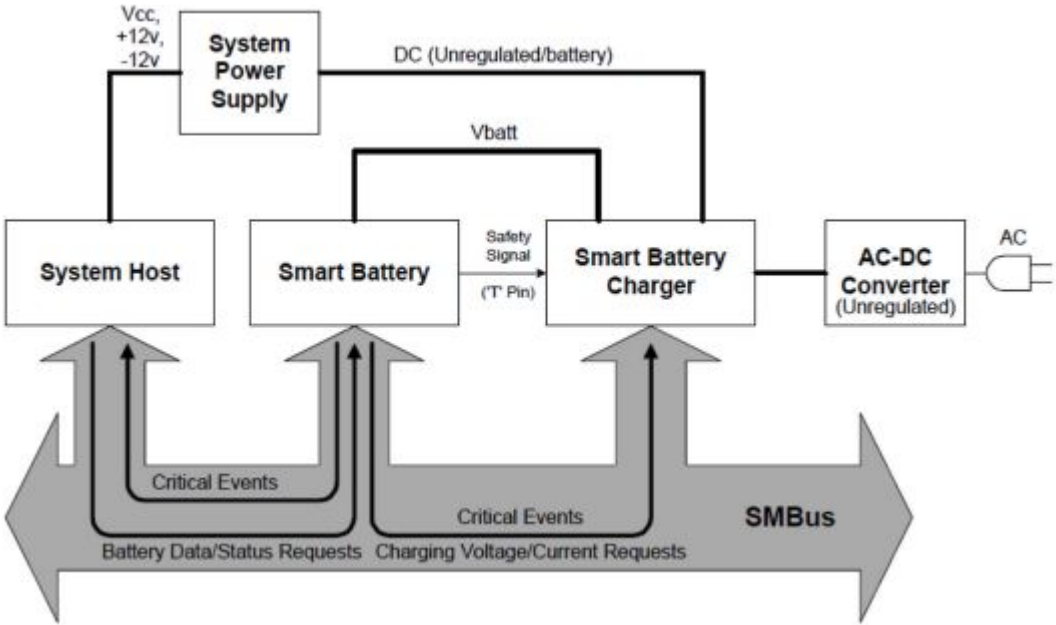
Master -> Primary (Main) device

Slave -> Secondary device




I2C Message
START
7 or 10 Bits
Read/Write bit
ACK/NACK bit
8 bits
ACK/NACK bit
8 bits
ACK/NACK bit
STOP
Slave Address
Data Frame 1
Data Frame 2
Start Condition
Stop Condition
The Geek Pub

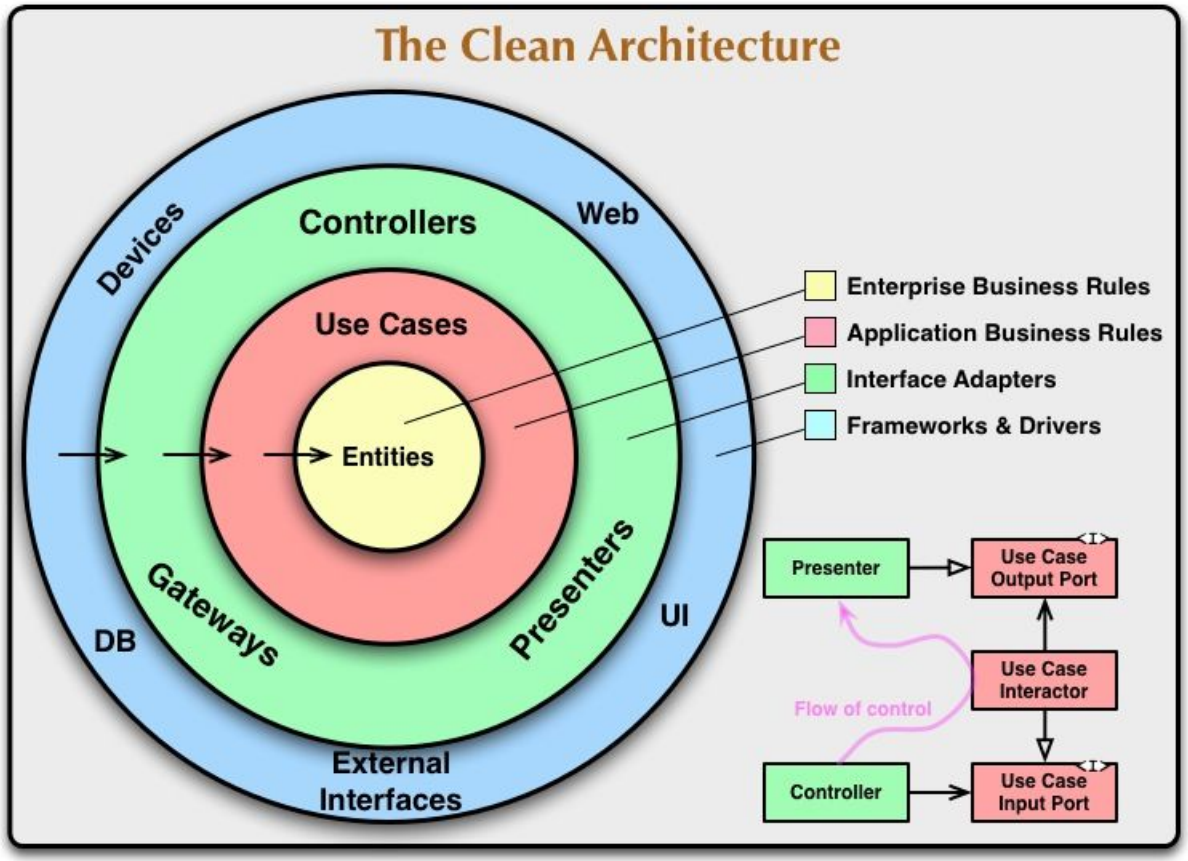
SMBus = I2C



Address	Value
0x00	01001010
0x01	10111010
0x02	01011111
0x03	00100100
0x04	01000100
0x05	10100000
0x06	01110100
0x07	01101111
0x08	10111011
...	...
0xFE	11011110
0xFF	10111011

Part 3: Entities

Where all the fun begins


The Clean Architecture
Devices
Web
Controllers
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port

Entities: Data Structure

- **Database**: Table structure for customers (id)
- **Response**: Returned value from an API after request to DB (customer_id)
- **Request**: Request a value from the API (customer_id)
- **App**: The in-app naming convention (customerId)

Entities: Types

Group all entities into single concerns like a directory called Customer

- **DB**: CustomerDataBase *{id: UUID}*
- **Response**: CustomerResponse *{customer_id: UUID}*
- **Request**: CustomerRequest *{customer_id: UUID}*
- **App**: Customer *{customerId: UUID}*

Entities: Converters

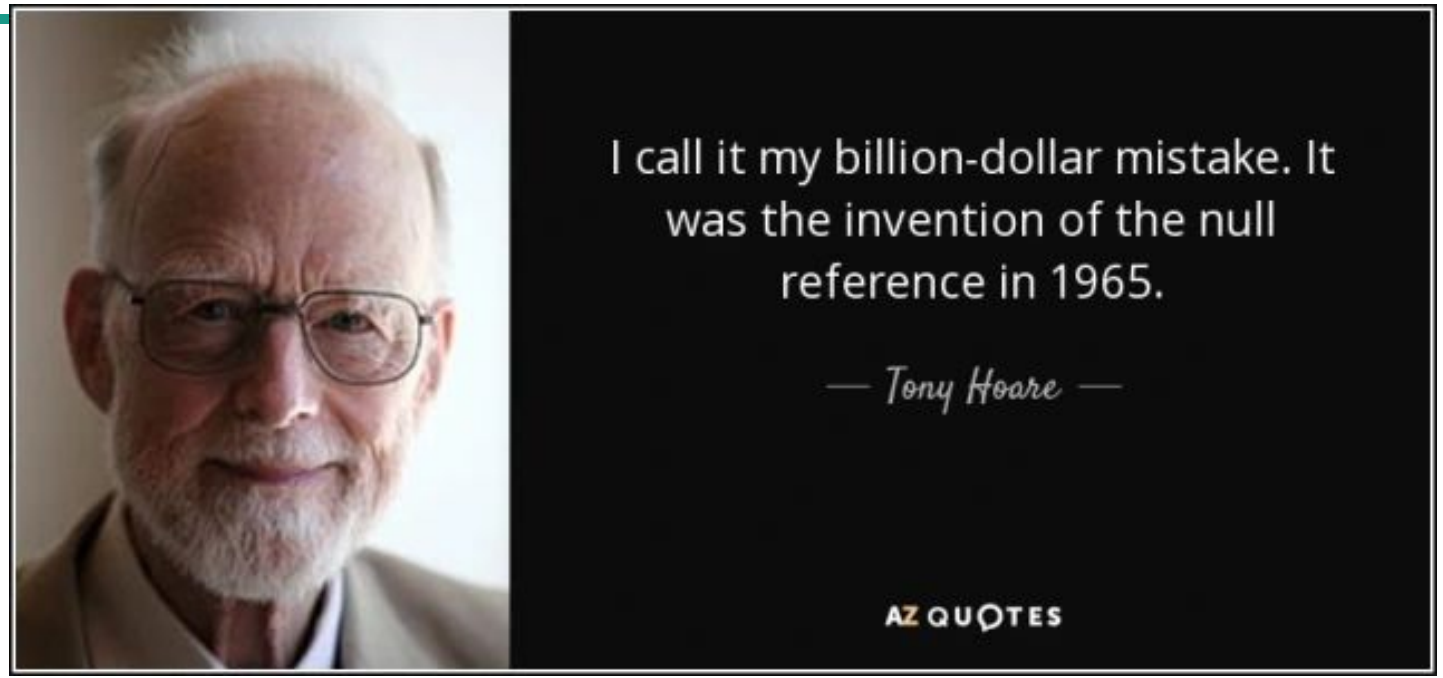
Aside from the Entities we can place a converter file that will include all or conversion needs.

In a nutshell: Map functions (without using map because it sucks). We will use new objects and manually map every property. You can use Map function as long it doesn't get too complicated to understand in the future.

- **customerDbtoApi** -> customer.id will become customer.customer_id
- **customerApitoDb** -> customer.customer_id will become customer.id
- **customerResponseToCustomer** -> customer.customer_id will become customer.customerId
- **customerToCustomerRequest** -> customer.customerId will become customer.customer_id

Entities: The great thing about converters

- You updated the DB property name? Update the converter to use the new name.
- You updated the API property name? Update the converter to use the new name.
- You updated the App property name? Why would you do that? The idea is to keep it as is.


I call it my billion-dollar mistake. It was the invention of the null reference in 1965.
— Tony Hoare —
AZ QUOTES

How to avoid null?

In a 2009 presentation, Tony Hoare stated that introducing null references was his "billion-dollar mistake." The mistake he referred to is the introduction of a value (null) that represents the absence of a reference to an object or the lack of a meaningful value.

The issue with null is that it can lead to null reference errors or NullPointerExceptions when developers forget to check whether a reference is null before dereferencing it. These errors can be challenging to debug and can potentially lead to serious issues in software, making it a costly mistake in terms of both development time and potential negative impacts on applications.

You can avoid it by always setting to 0, ''or wrapping into a DataWrapper: {data: data, status: ok | error}