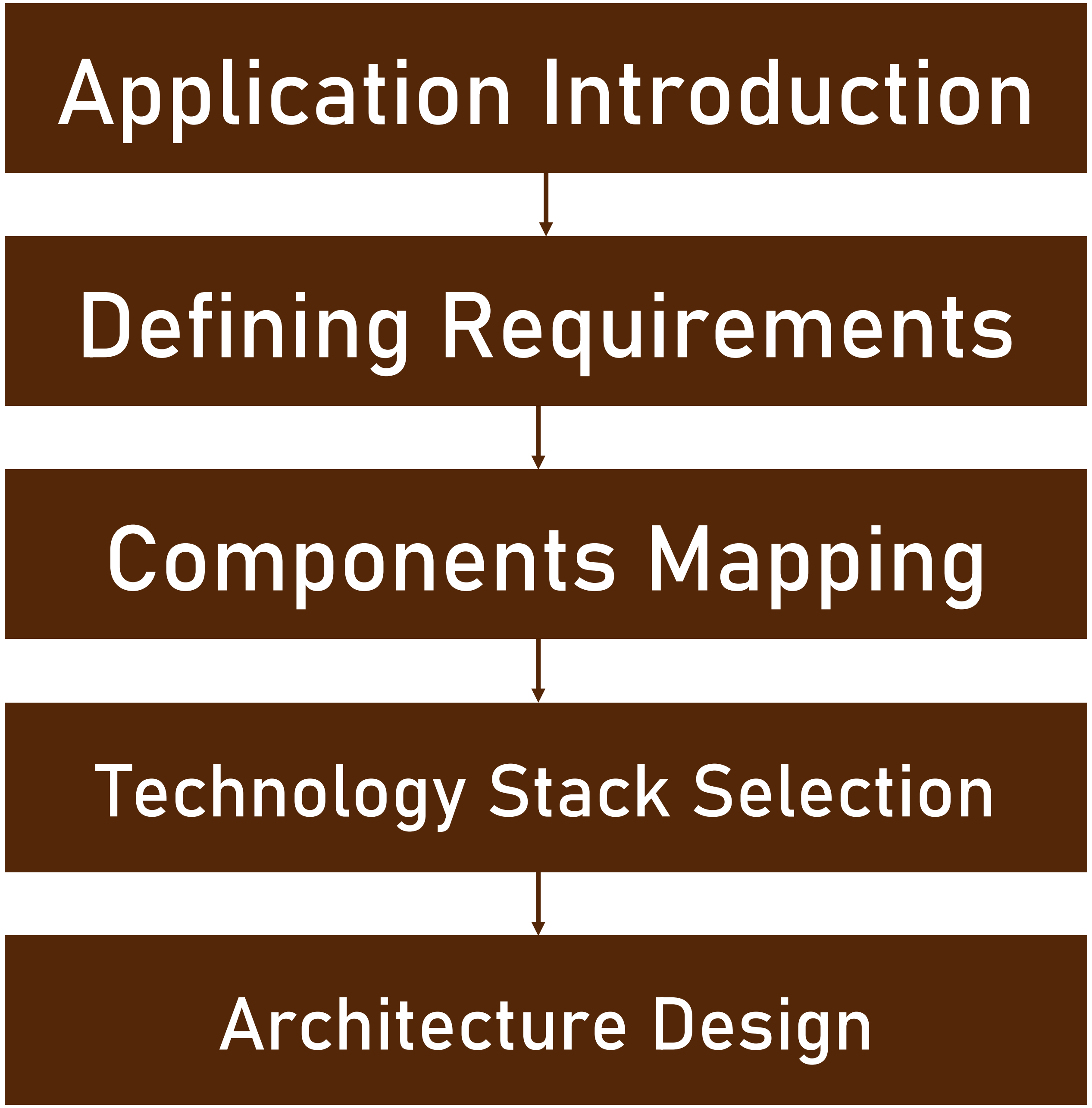
Case Study

Memi Lavi
www.memilavi.com

A Real World Application


Application Introduction
Defining Requirements
Components Mapping
Technology Stack Selection
Architecture Design

Architecture Document

Download

Read

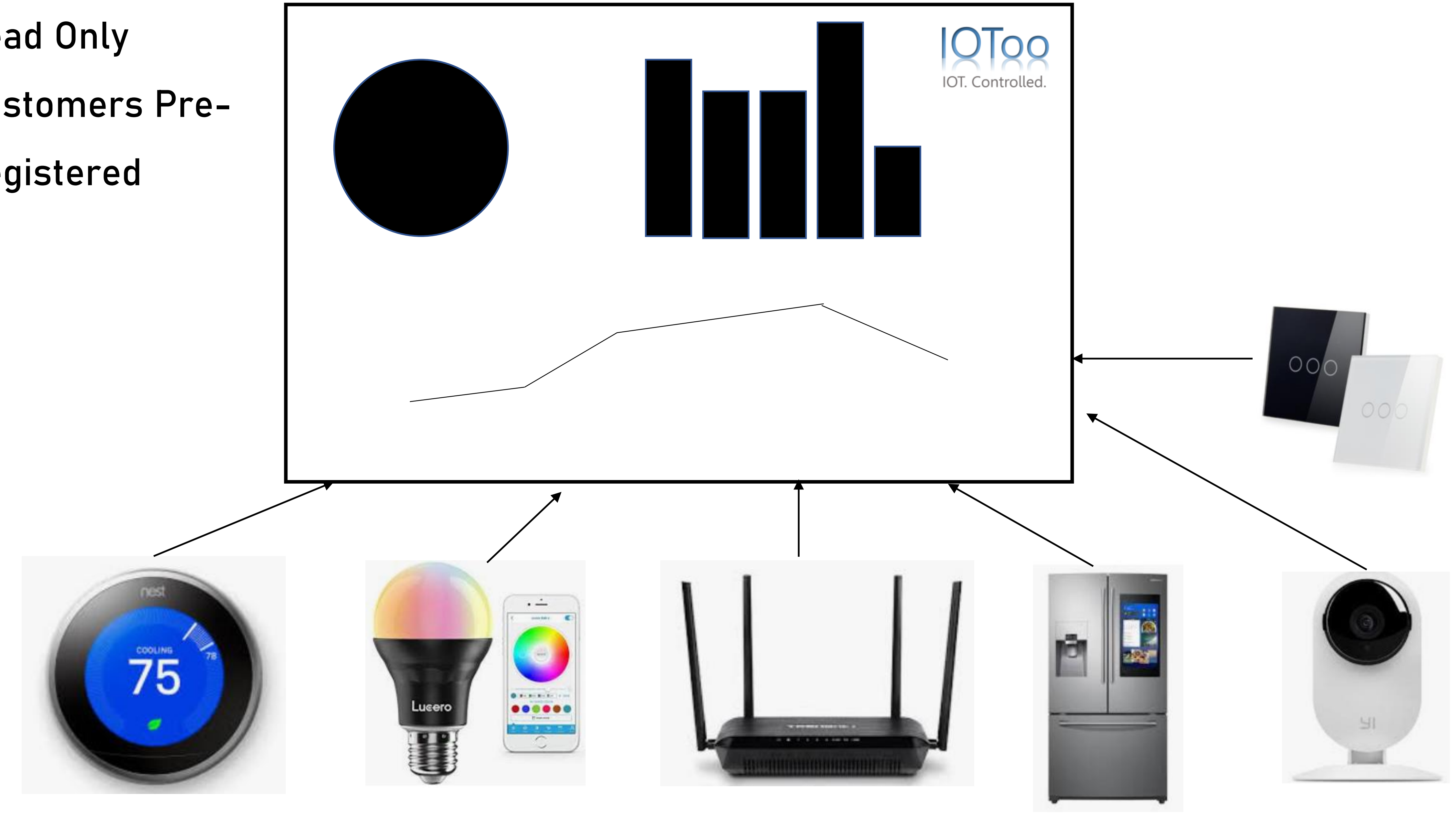
Use

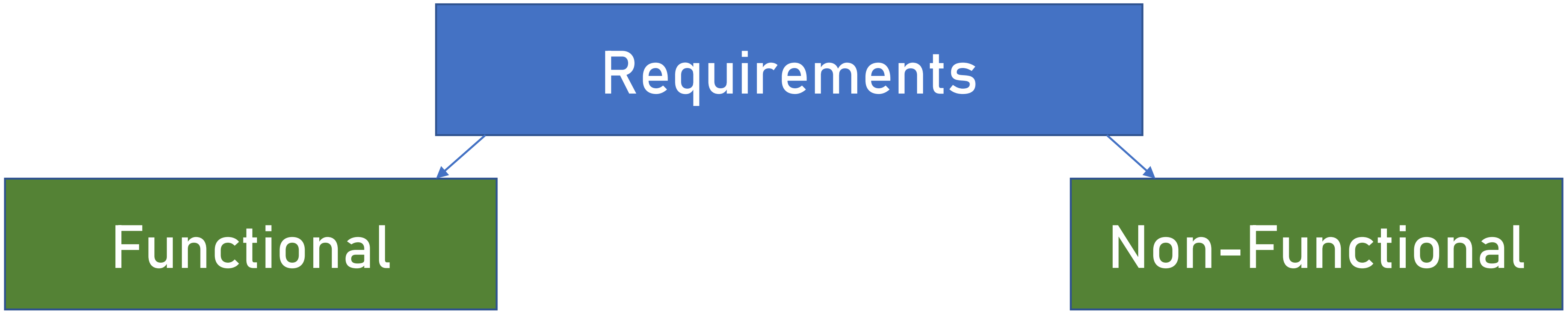
IOToo

IOT. Controlled.

IOT = Internet of Things

- Read Only
- Customers Pre-Registered





What the system should do

What the system should deal with

1. Receive status updates from IOT Devices
2. Store the updates for future use
3. Query the updates

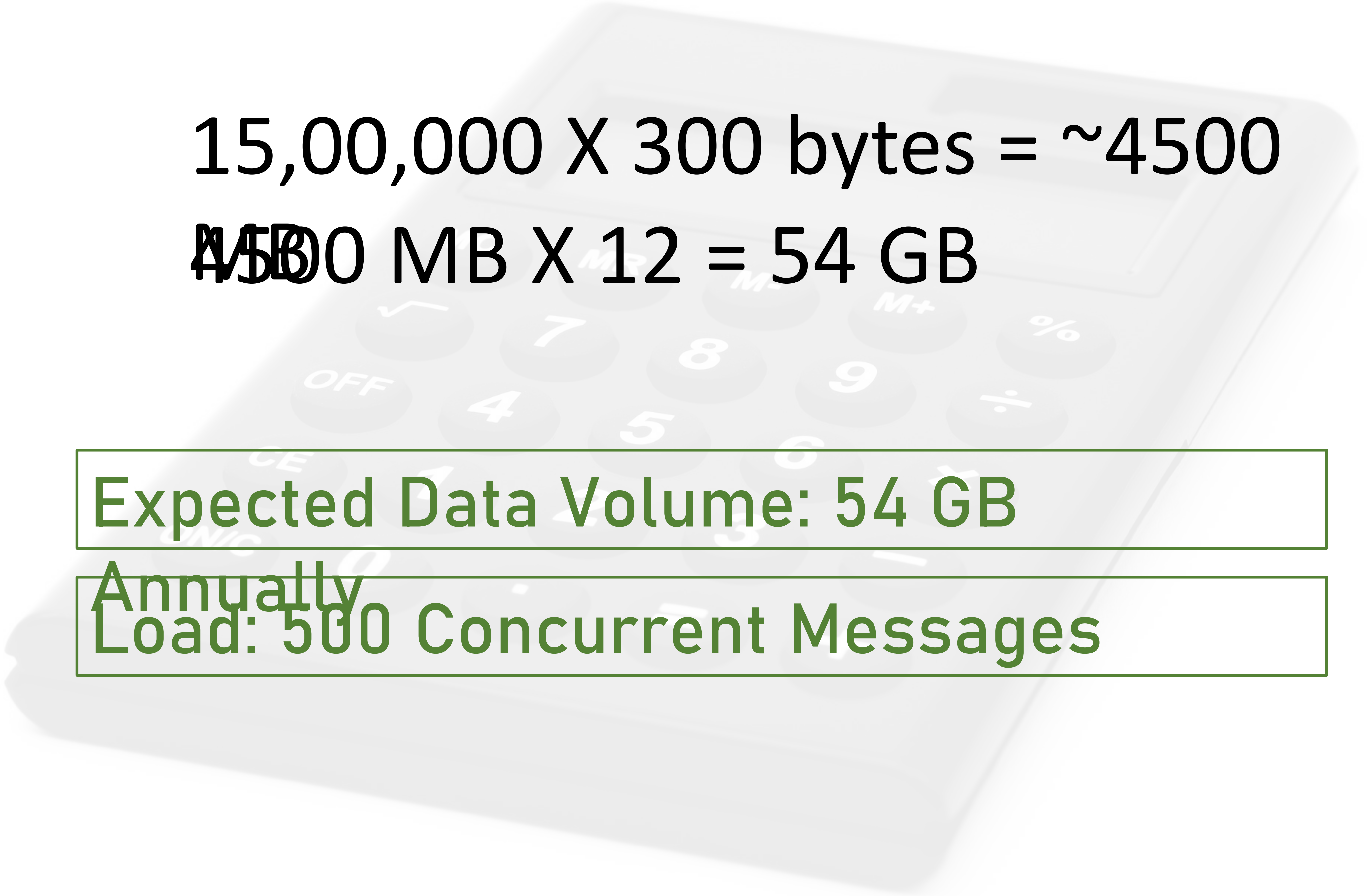
Write down the non-functional requirements relevant for this system

What We Know

1. Messages are received from IOT devices
2. Probably a <u>lot</u> of messages
3. Affects the load
4. Affects the data volume

What We Ask

1. *"How many concurrent messages should the system expect in peak time?"*	500
2. *"What is the total number of messages per month?"*	15,000,000
3. "*What is the average size of a message?"*	300 bytes



15,00,000 X 300 bytes = ~4500 MB

4500 MB X 12 = 54 GB

Expected Data Volume: 54 GB Annually

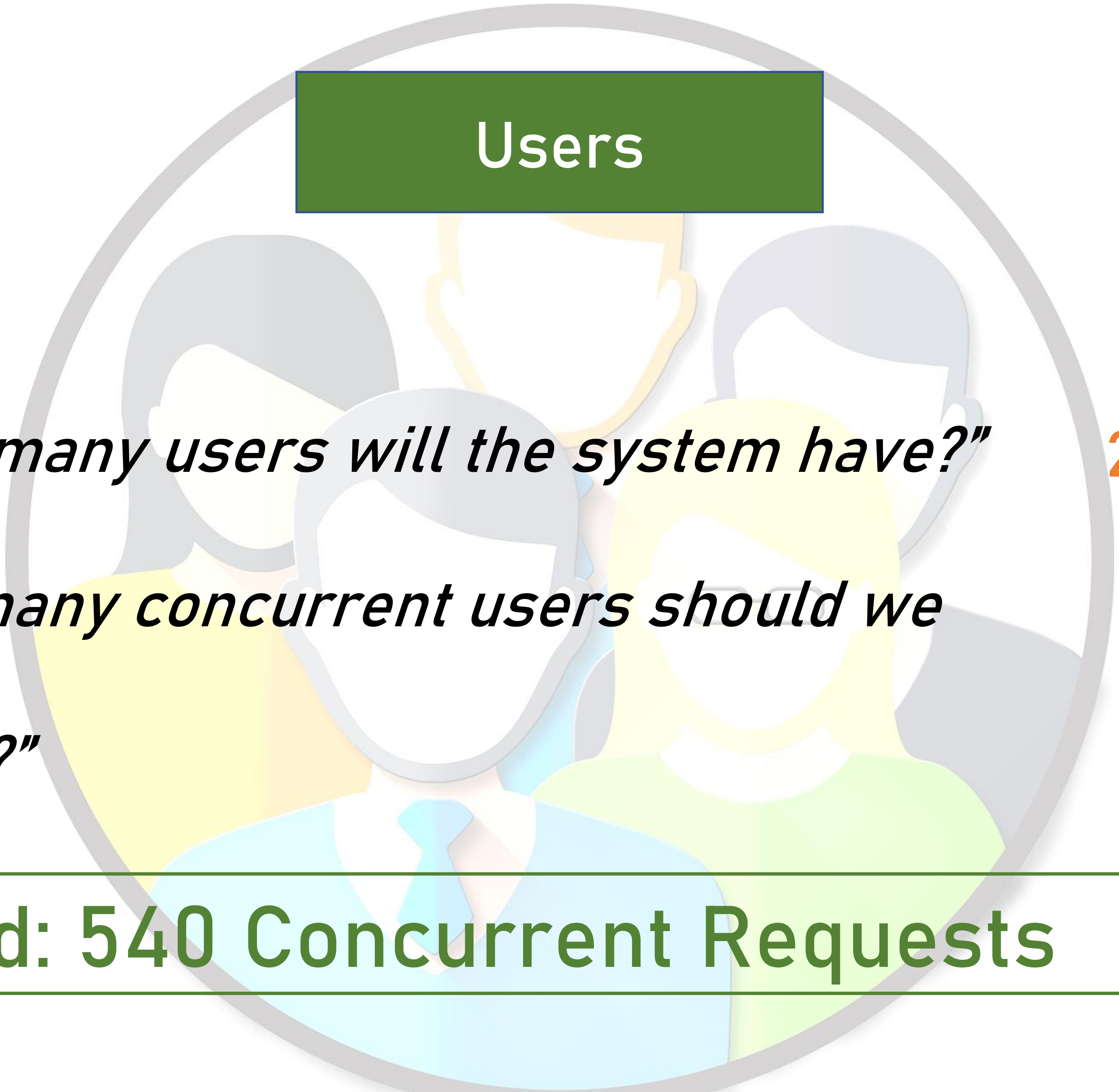
Load: 500 Concurrent Messages

Message Loss

No Message Loss!

99%

Users



4. *"How many users will the system have?"* 2,000,000

5. *"How many concurrent users should we expect?"* 40

Load: 540 Concurrent Requests

SLA

6. *"What is the maximum downtime allowed?"*

100 Uptime!

SLA Has Lots of Factors

Hardware

Virtualization

Network

Database Servers

SLA Software Level

Silver

Gold

Platinum

- Fully Stateless
- Easily Scaled Out
- Logging & Monitoring

Requirements

Functional

What the system should do

1. Receive status updates from IOT Devices
2. Store the updates for future use
3. Query the updates

Non-Functional

What the system should deal with

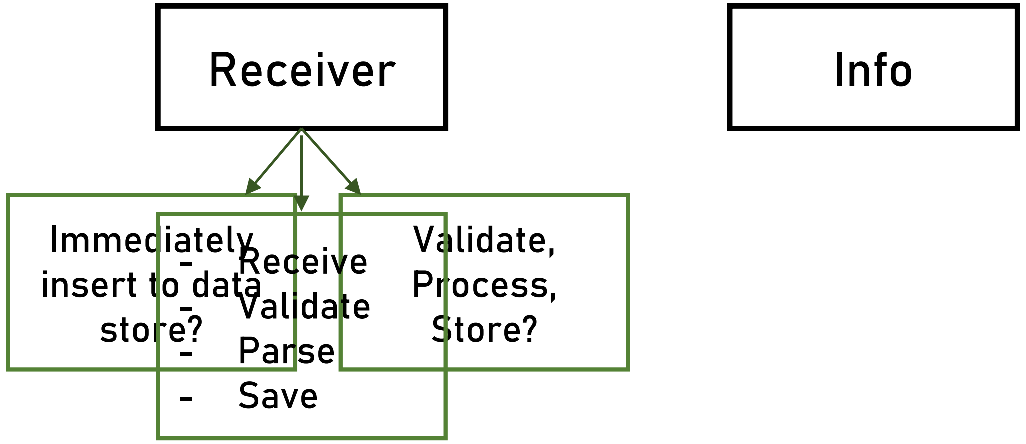
1. Data Volume: 54 GB Annually
2. 540 Concurrent Requests
3. 1% Message Loss
4. 2,000,000 Users
5. SLA: Platinum

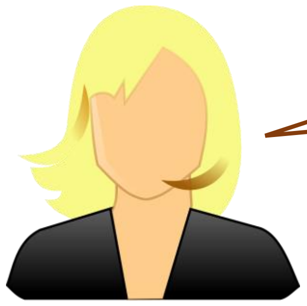
Components

Based on requirements:

1. Receive status updates from IOT Devices
2. Store the updates for future use
3. Query the updates

1. 540 Concurrent Requests





- 4 types of devices (& formats)
- 3 use JSON, 1 uses fixed-format
- Validation is a must

- Receive
- Validate
- Parse
- Save

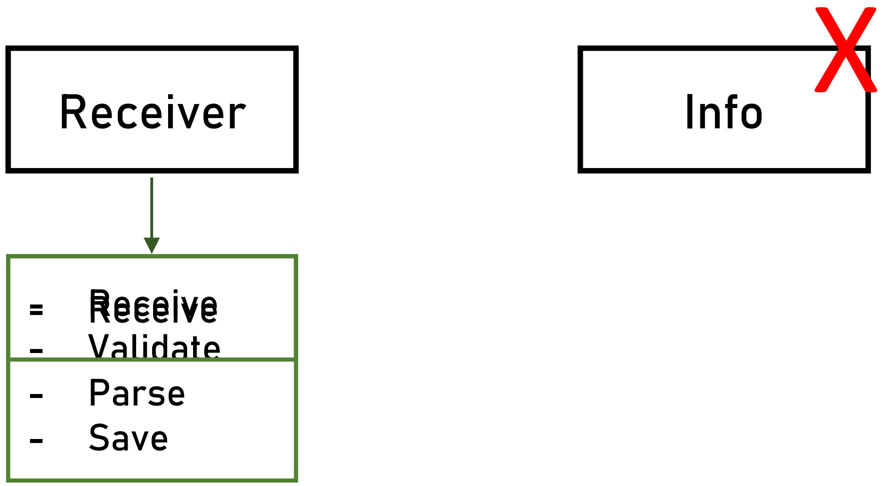
- Data is independent from source
- Fully Accessible
- Extremely important when data is received from multiple sources

Components

Based on requirements:

1. Receive status updates from IOT Devices
2. Store the updates for future use
3. Query the updates

1. 540 Concurrent Requests

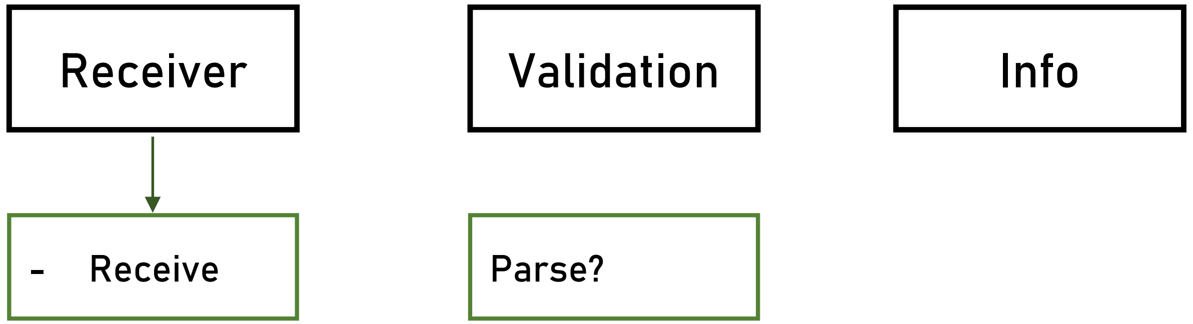


Components

Based on requirements:

1. Receive status updates from IOT Devices
2. Store the updates for future use
3. Query the updates

1. 540 Concurrent Requests

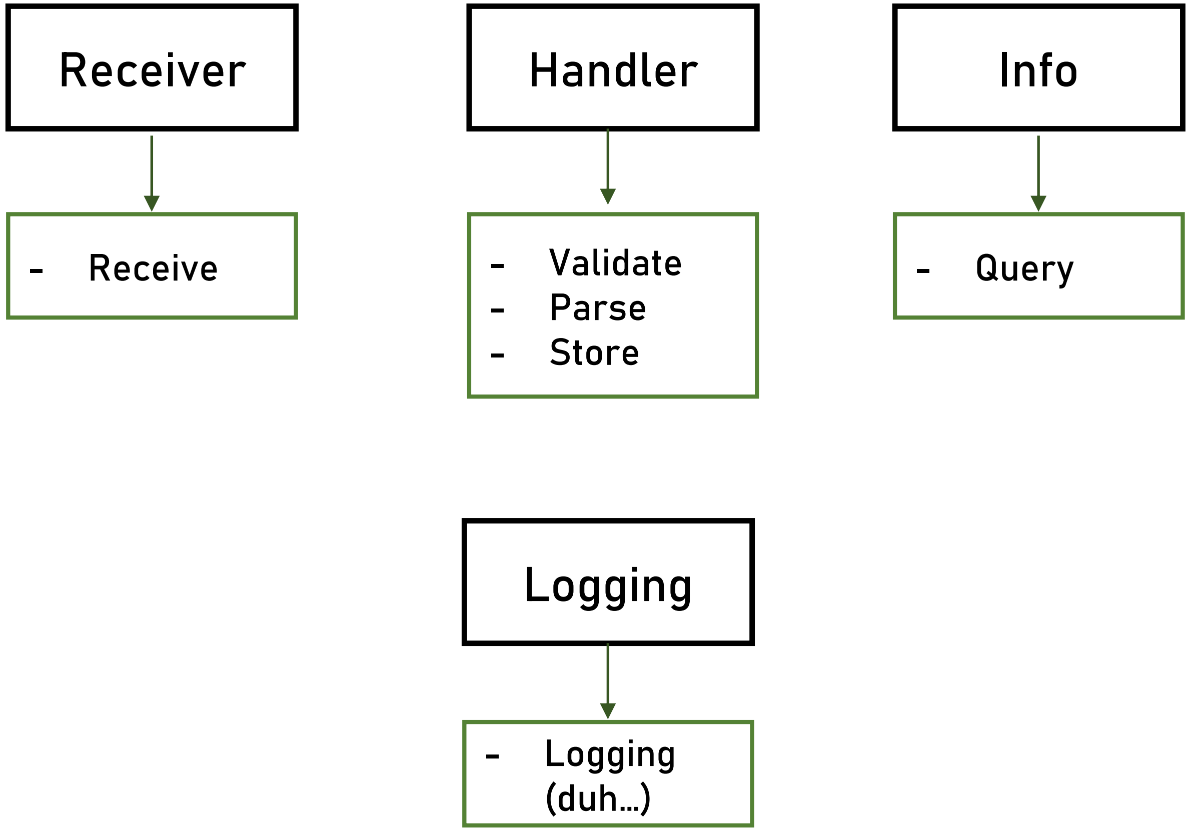


Components

Based on requirements:

1. Receive status updates from IOT Devices
2. Store the updates for future use
3. Query the updates

1. 540 Concurrent Requests

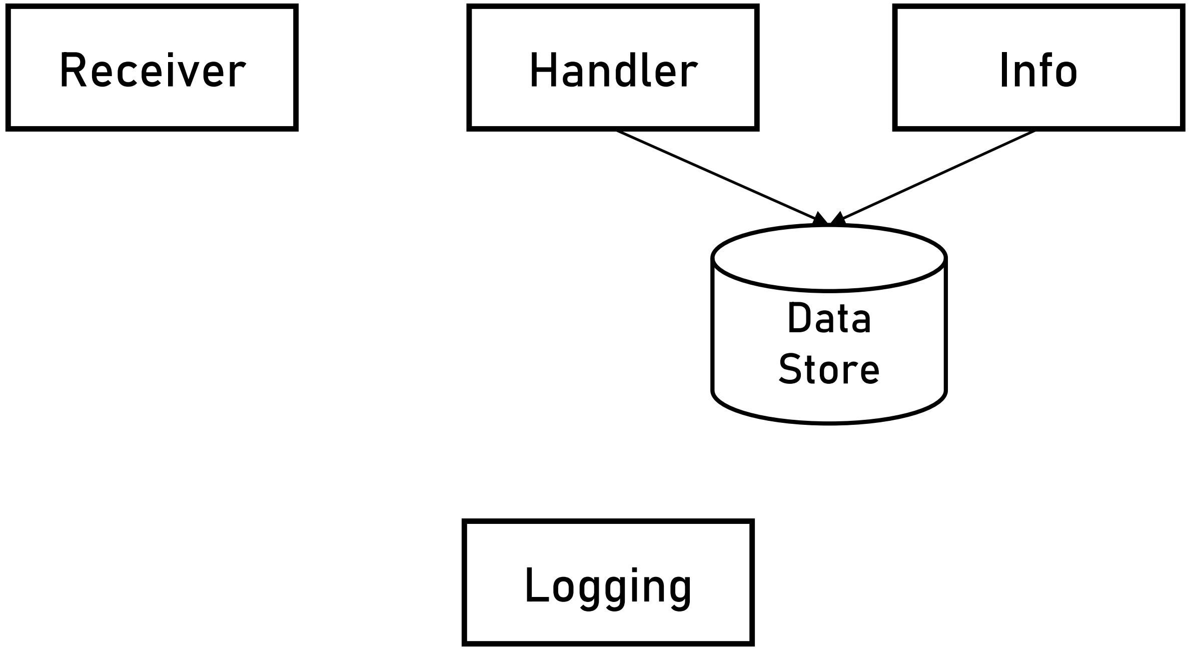


Components

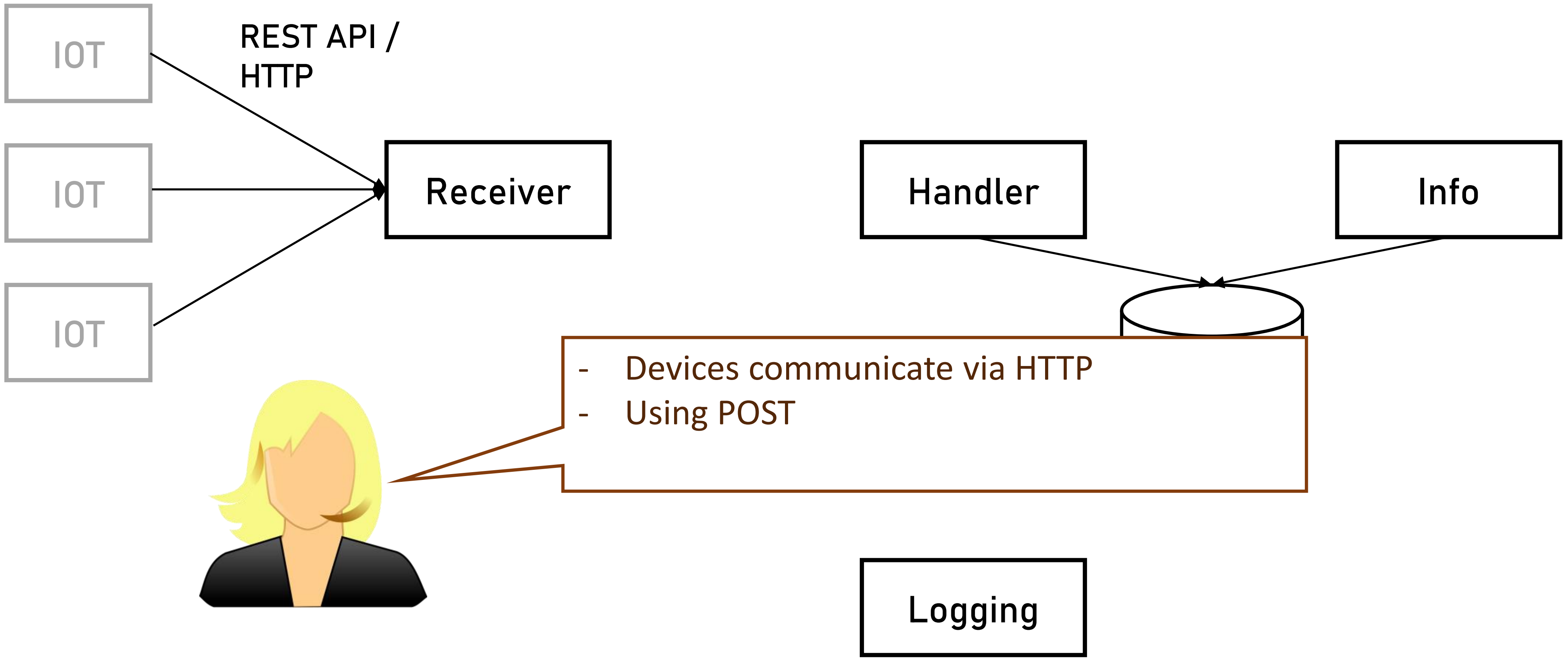
Based on requirements:

1. Receive status updates from IOT Devices
2. Store the updates for future use
3. Query the updates

1. 540 Concurrent Requests

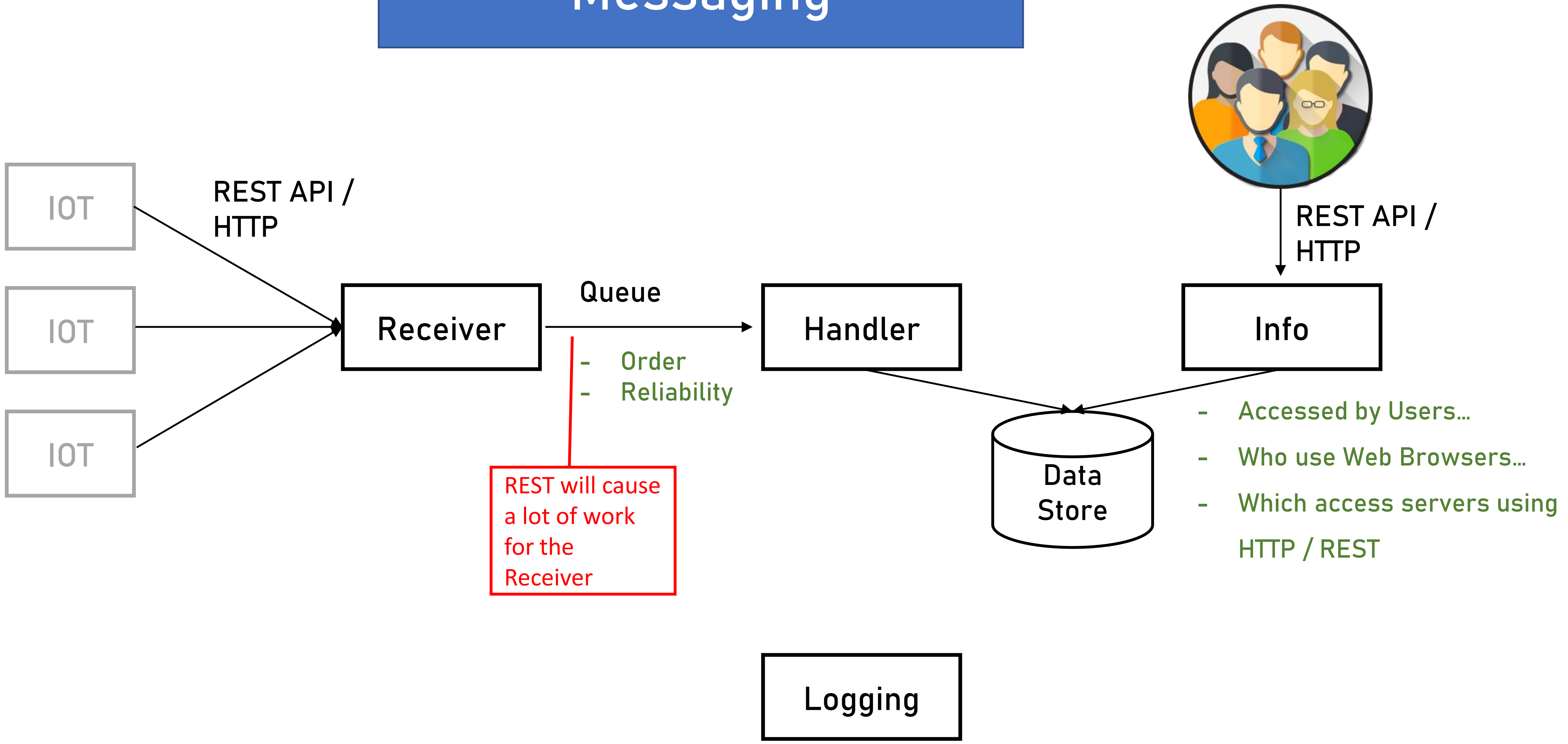


Messaging

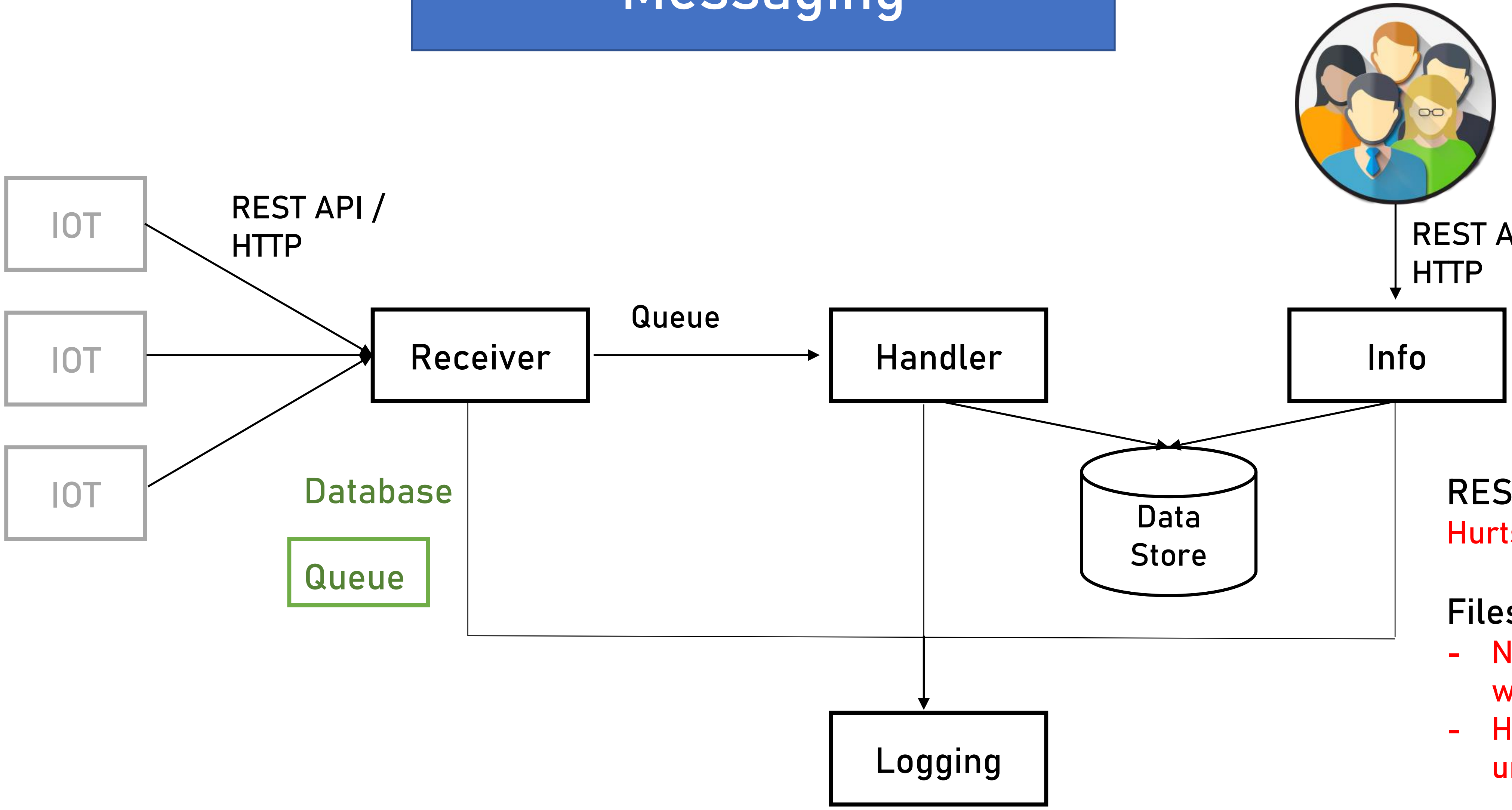


- Devices communicate via HTTP
- Using POST

Messaging


IOT
IOT
IOT
REST API / HTTP
Receiver
Queue
- Order
- Reliability
REST will cause a lot of work for the Receiver
Handler
Info
REST API / HTTP
Data Store
- Accessed by Users…
- Who use Web Browsers…
- Which access servers using HTTP / REST
Logging

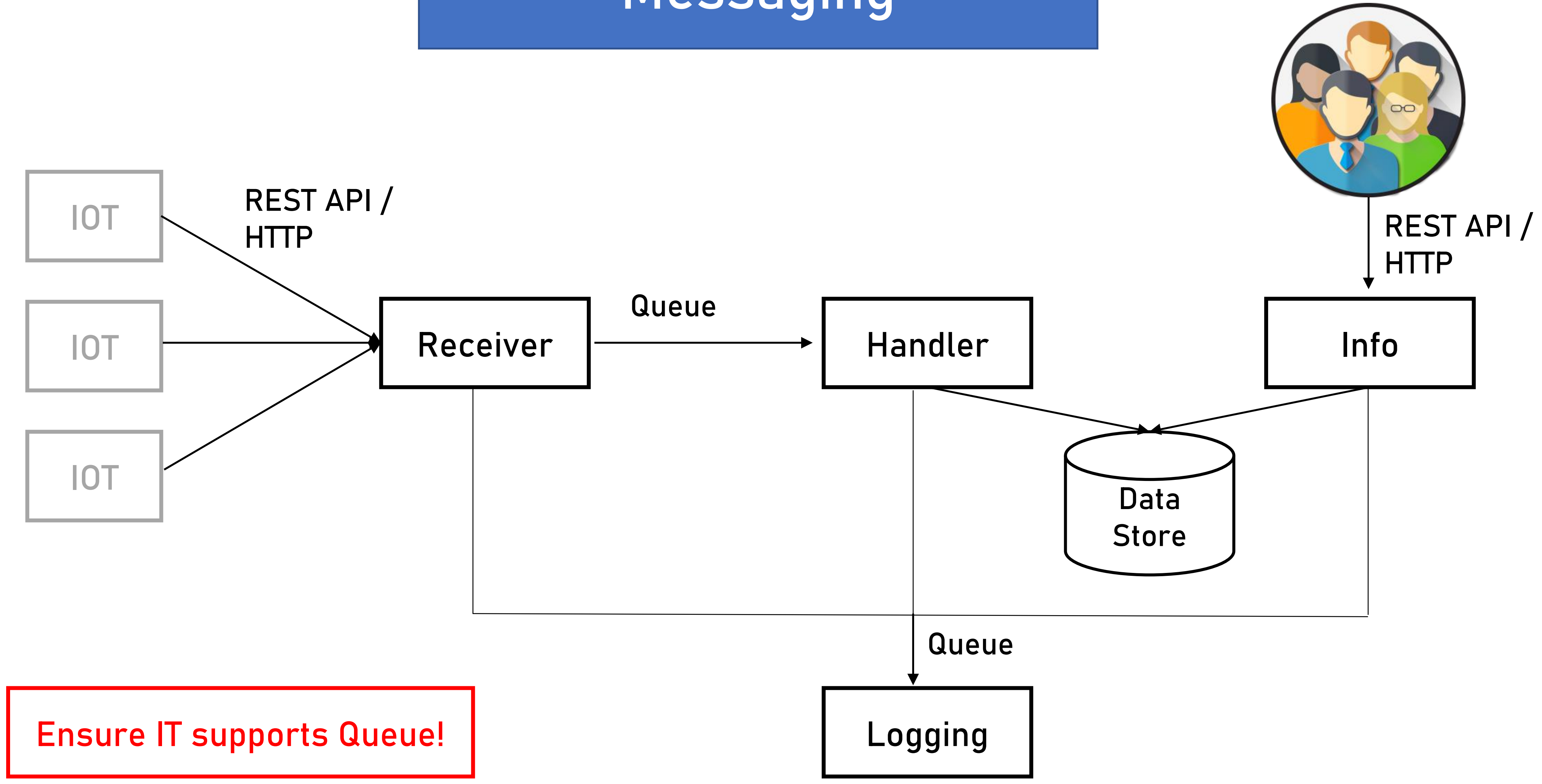
Messaging



REST?
Hurts Performance

Files?
- Not compatible with the cloud
- Highly uncontrolled

Messaging



Ensure IT supports Queue!

Logging Service

- Very Important

- Other services use it

Logging Service

Steps:

- Decide on Application Type

- Decide on Technology Stack

- Design the Architecture

Application Type

What it does:

- Read log records from queue

- Validate the records

- Store in data store

Application Type

What it does:

- Read log records from queue
- Handle the records
- Save in data store

Application Type

• Web App & Web API	❌
• Mobile App	❌
• Console	❓
• Service	❓
• Desktop App	❌

Application Type

• Web App & Web API	✗
• Mobile App	✗
• Console	✓
• Service	✓
• Desktop App	✗

Technology Stack

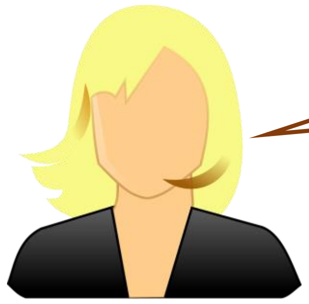
For:

- Component's Code
- Data Store

Technology Stack

Code Should:

- Access Queue's API
- Store Data



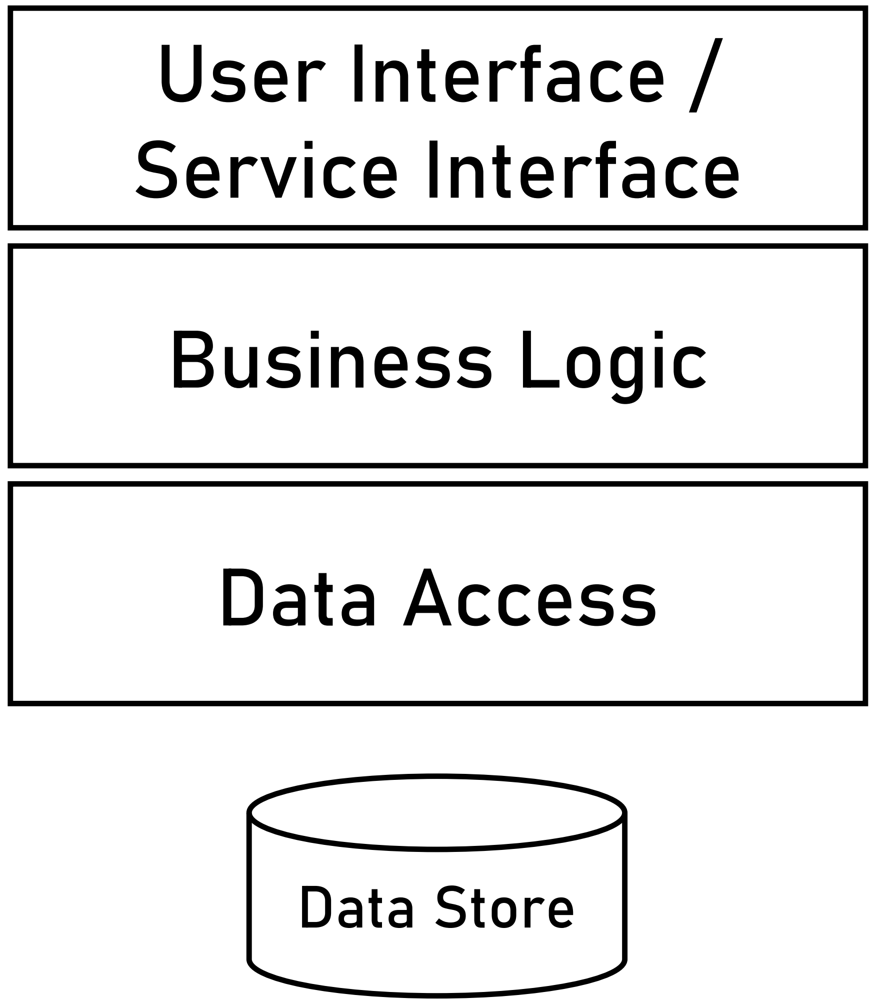
We're familiar with Microsoft stack, so we are expert in .NET and SQL Server

Technology Stack

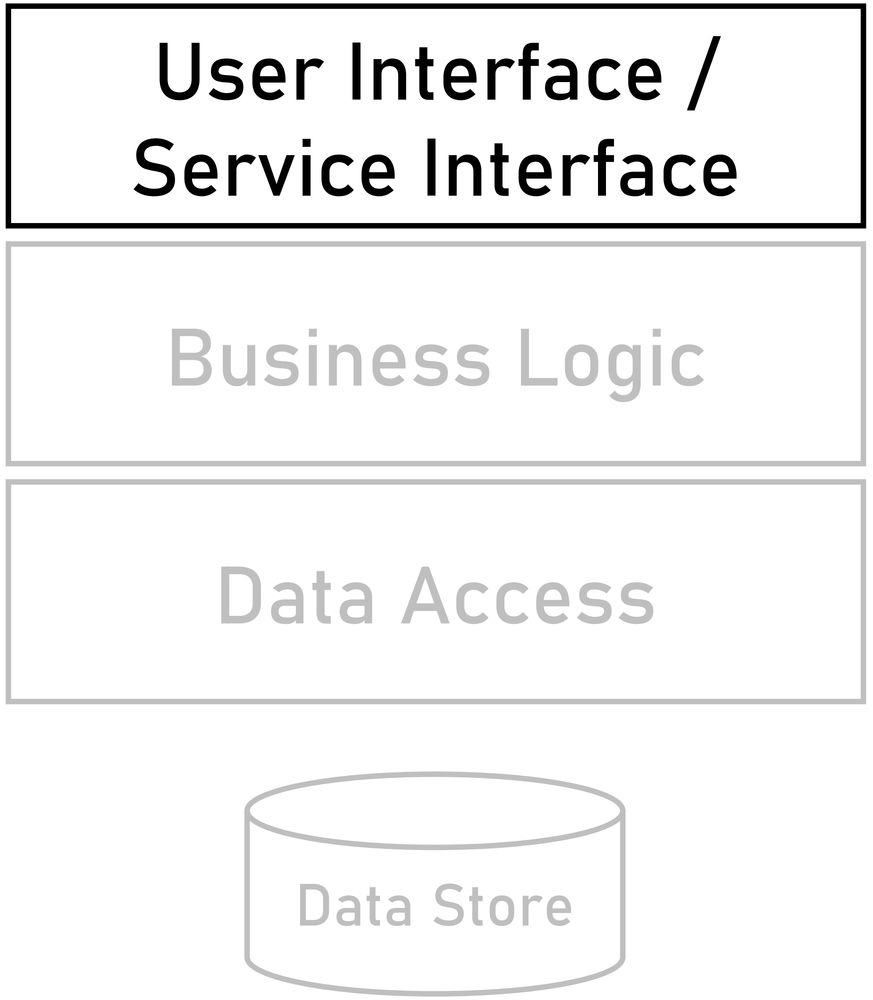
.NET Core

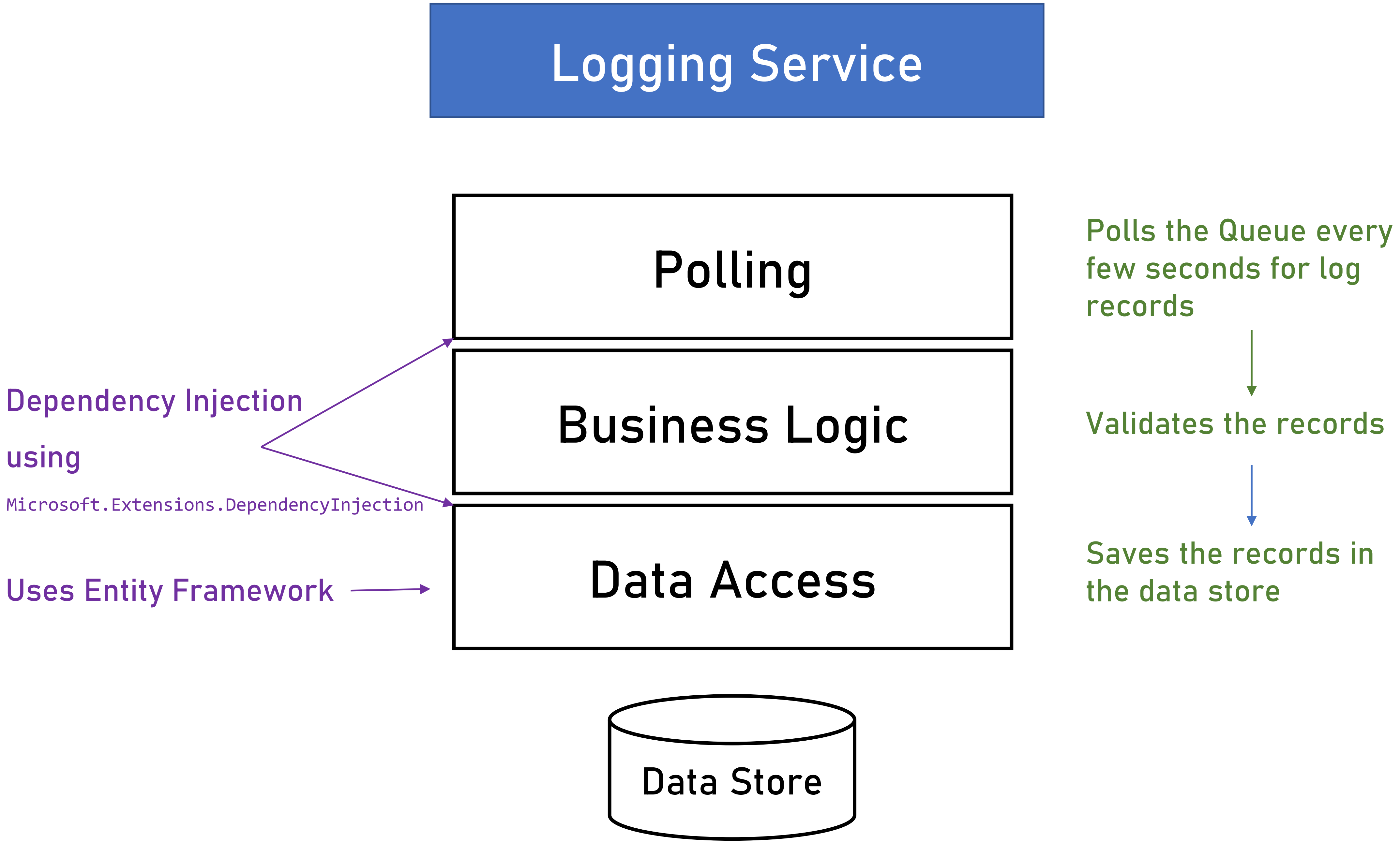
Microsoft® SQL Server®™

Architecture



Architecture




Logging Service
Polling
Business Logic
Data Access
Data Store
Dependency Injection
using
Microsoft.Extensions.DependencyInjection
Uses Entity Framework
Polls the Queue every few seconds for log records
Validates the records
Saves the records in the data store

Receiver Service

What it does:

- Receives messages from devices

- Sends messages to queue

Application Type

• Web App & Web API	✓
• Mobile App	✗
• Console	✗
• Service	✗
• Desktop App	✗

Technology Stack

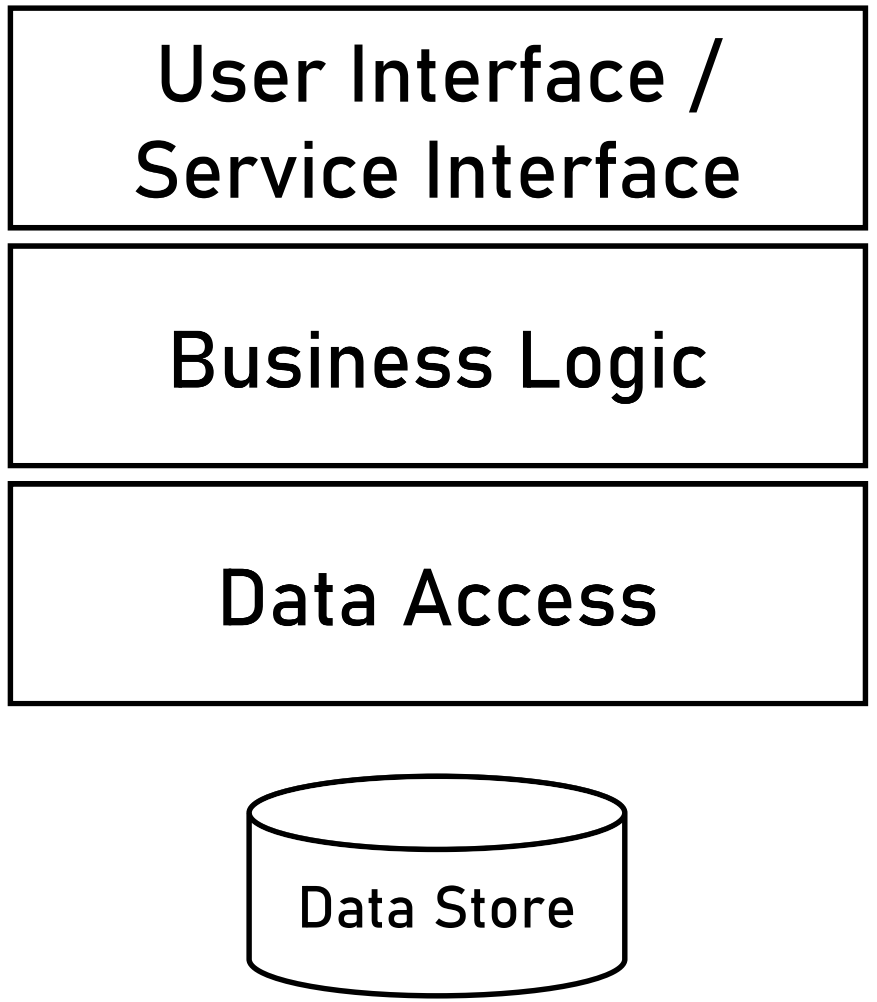
.NET Core has a great support for Web API apps

So…

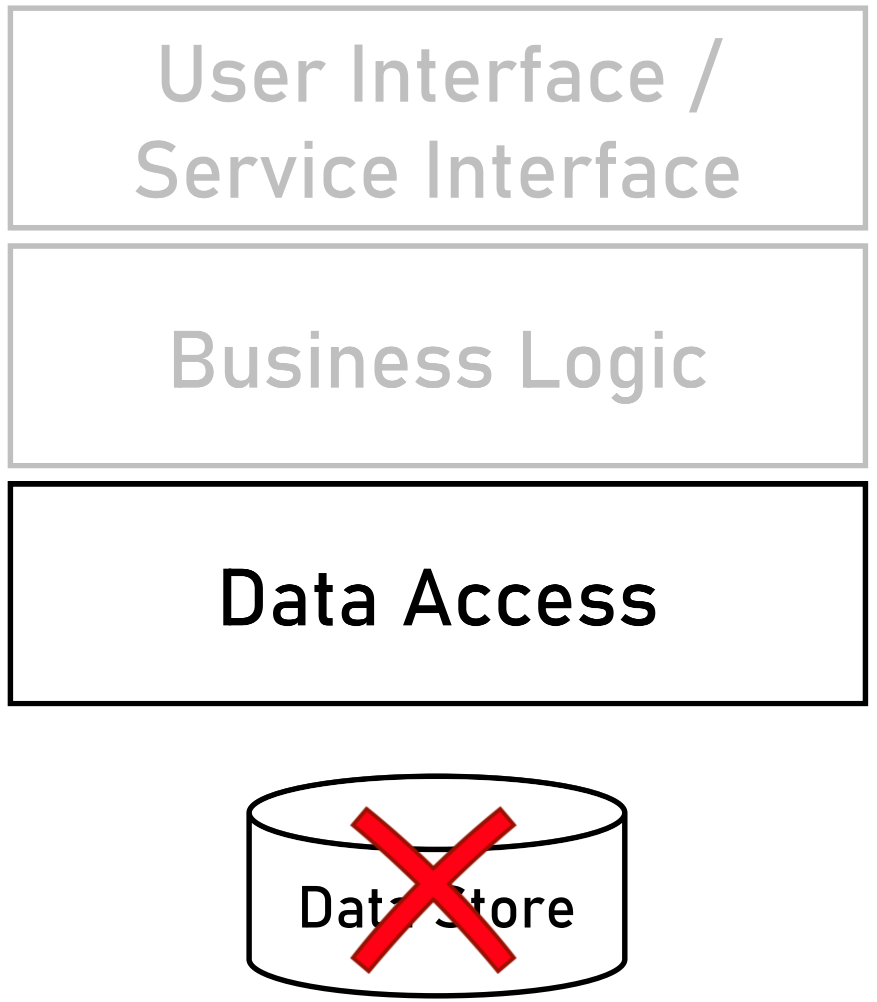
Technology Stack

.NET Core

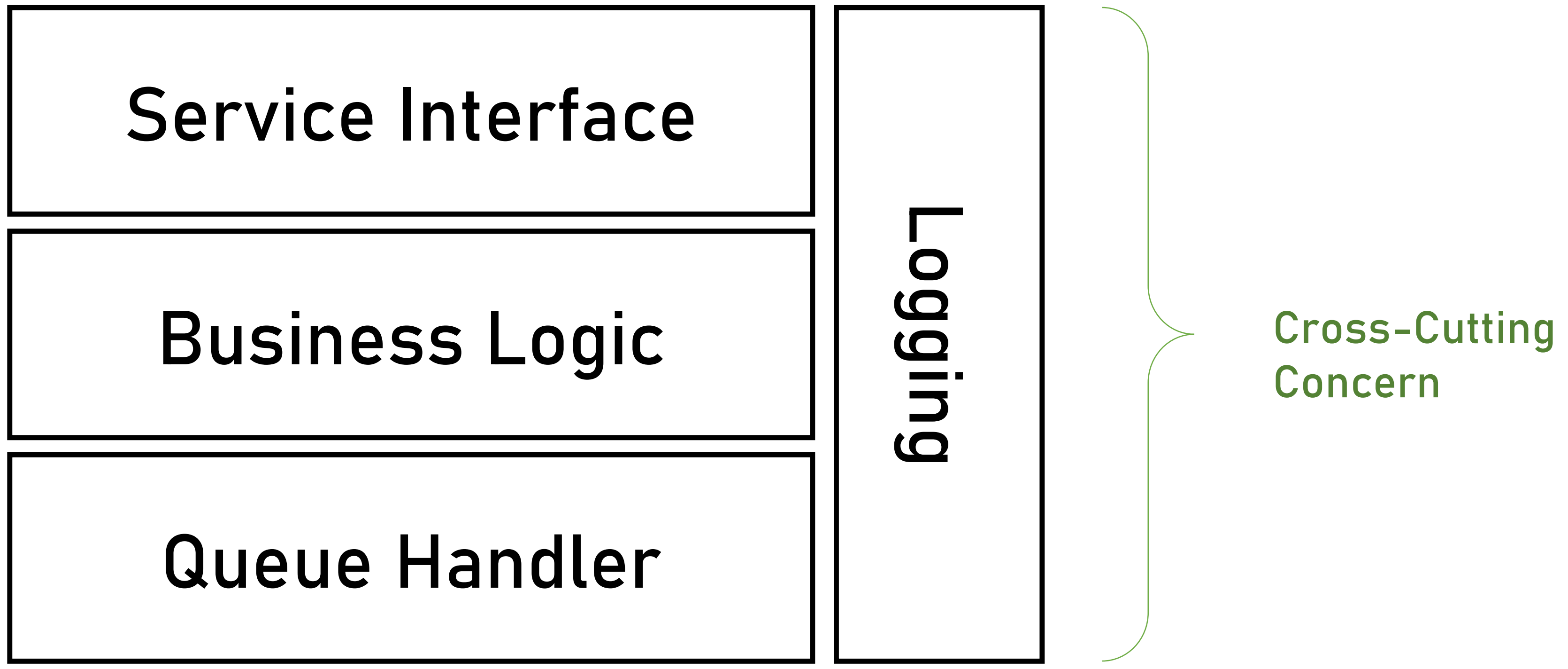
Architecture



Architecture

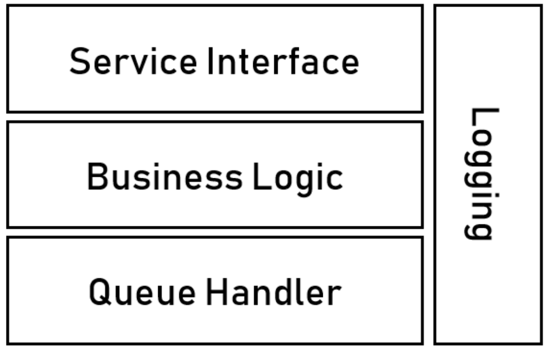


Architecture



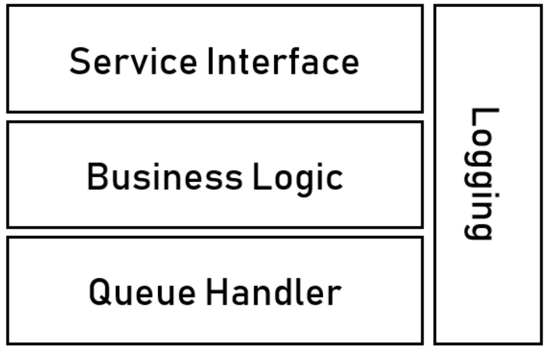
Non-Functional Requirements

Requirement	Compliant?
Load: 500 concurrent messages from devices	
Messages loss: 1%	



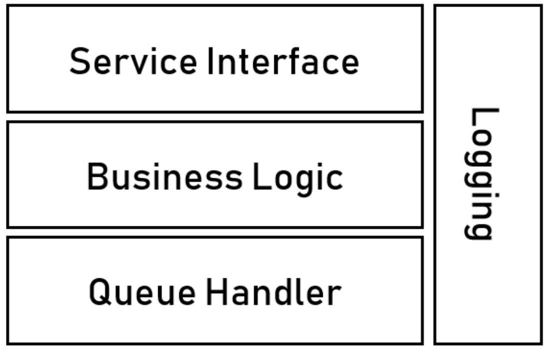
Non-Functional Requirements

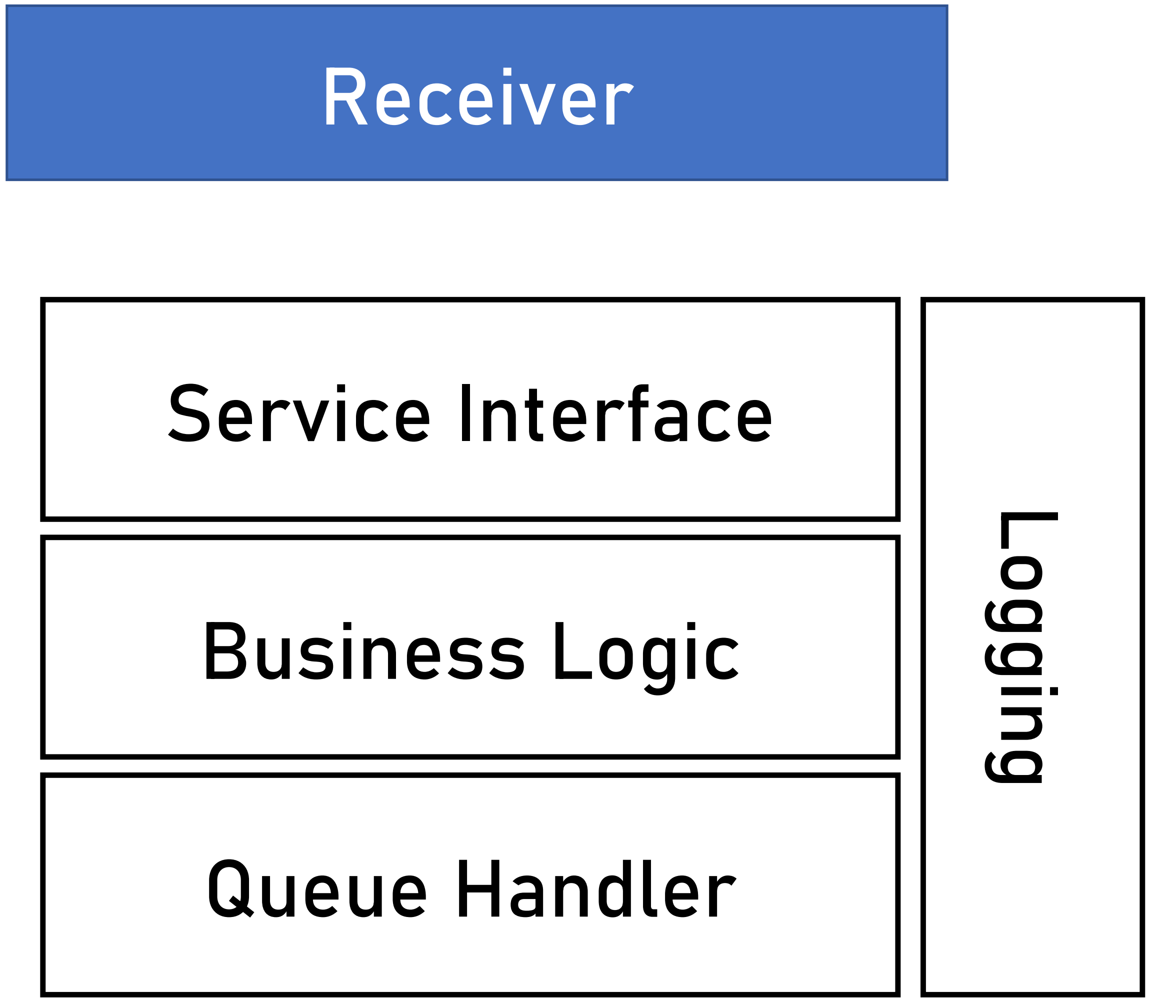
Requirement	Compliant?
Load: 500 concurrent messages from devices	Yes. - Architecture is stateless - Easily scaled-out - Service is very simple
Messages loss: 1%	



Non-Functional Requirements

Requirement	Compliant?
Load: 500 concurrent messages from devices	Yes. - Architecture is stateless - Easily scaled-out - Service is very simple
Messages loss: 1%	Yes. - REST API is quite reliable - Very low chance of for errors in such a simple service




Receiver
Service Interface
Business Logic
Queue Handler
Logging

Handler Service

What it does:

- Validates messages

- Parses messages

- Stores messages in data store

Messages wait in Queue

Application Type

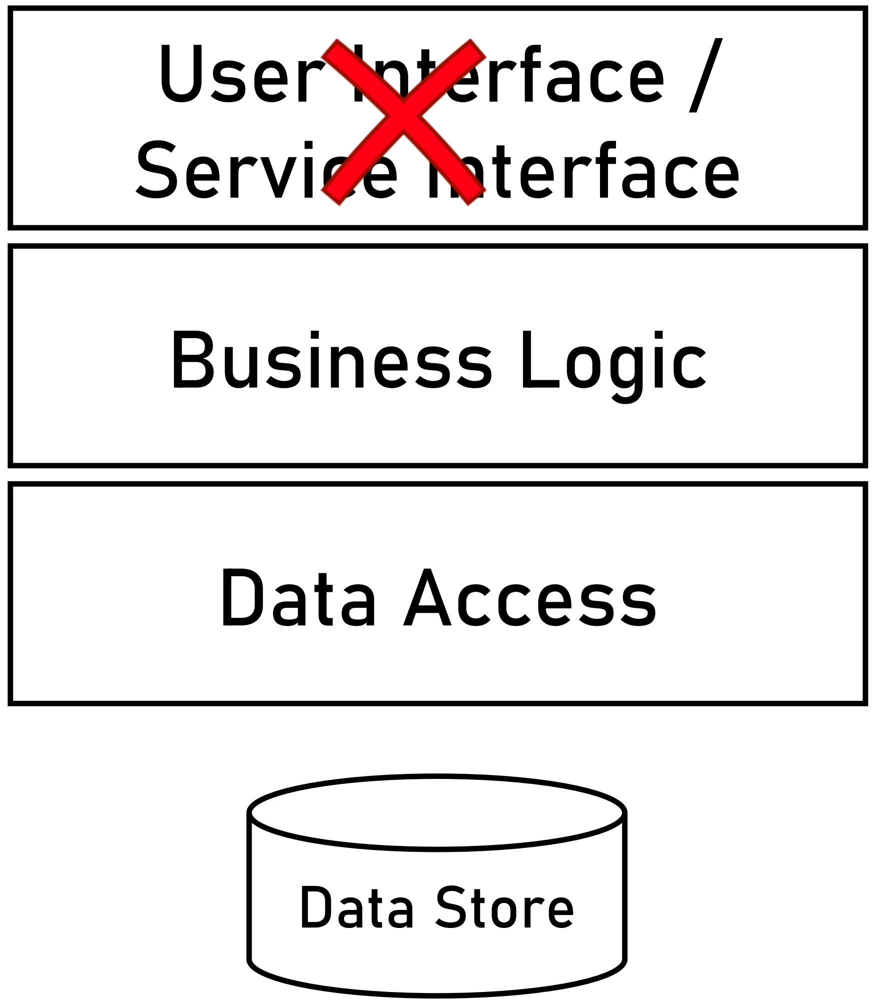
• Web App & Web API	✗
• Mobile App	✗
• Console	✗
• Service	✓
• Desktop App	✗

Technology Stack

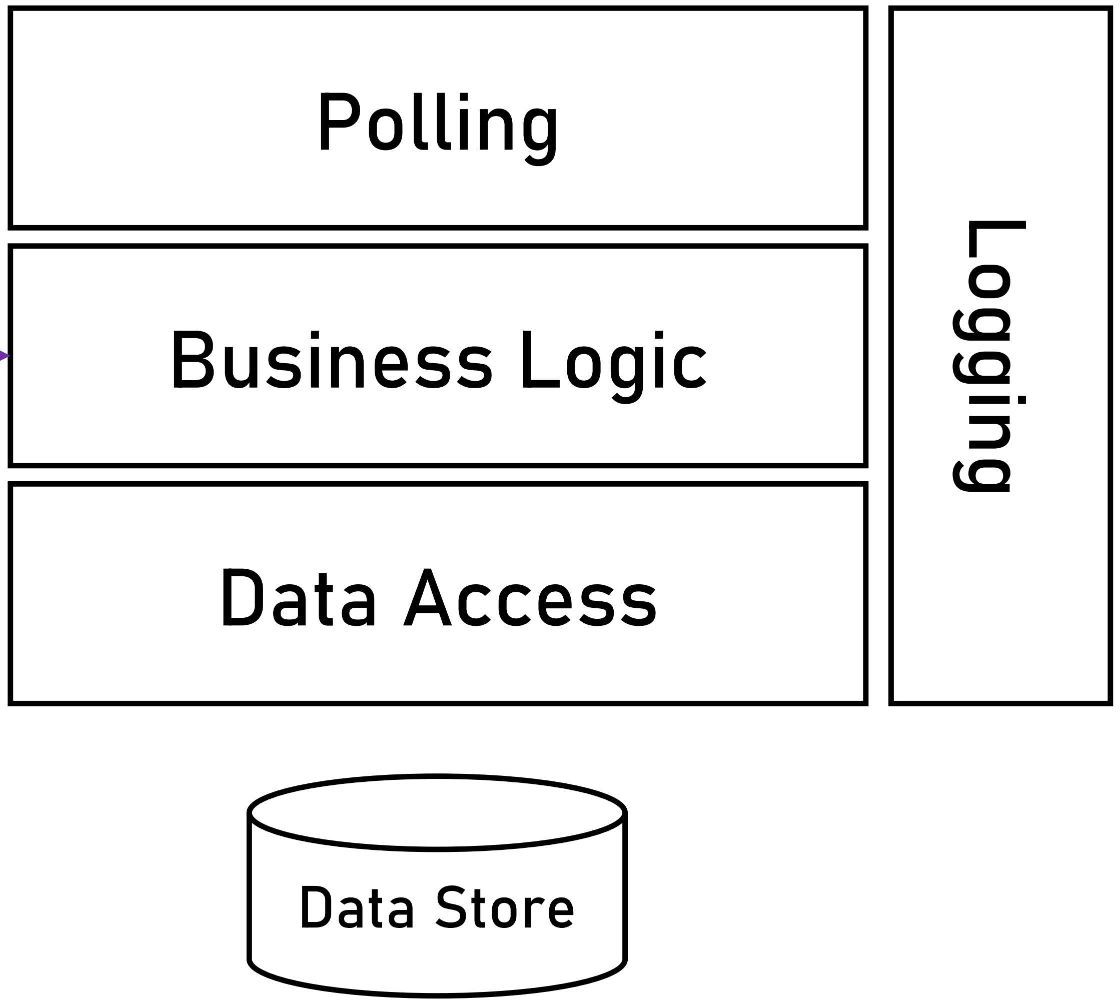
.NET Core

Microsoft® SQL Server®

Architecture

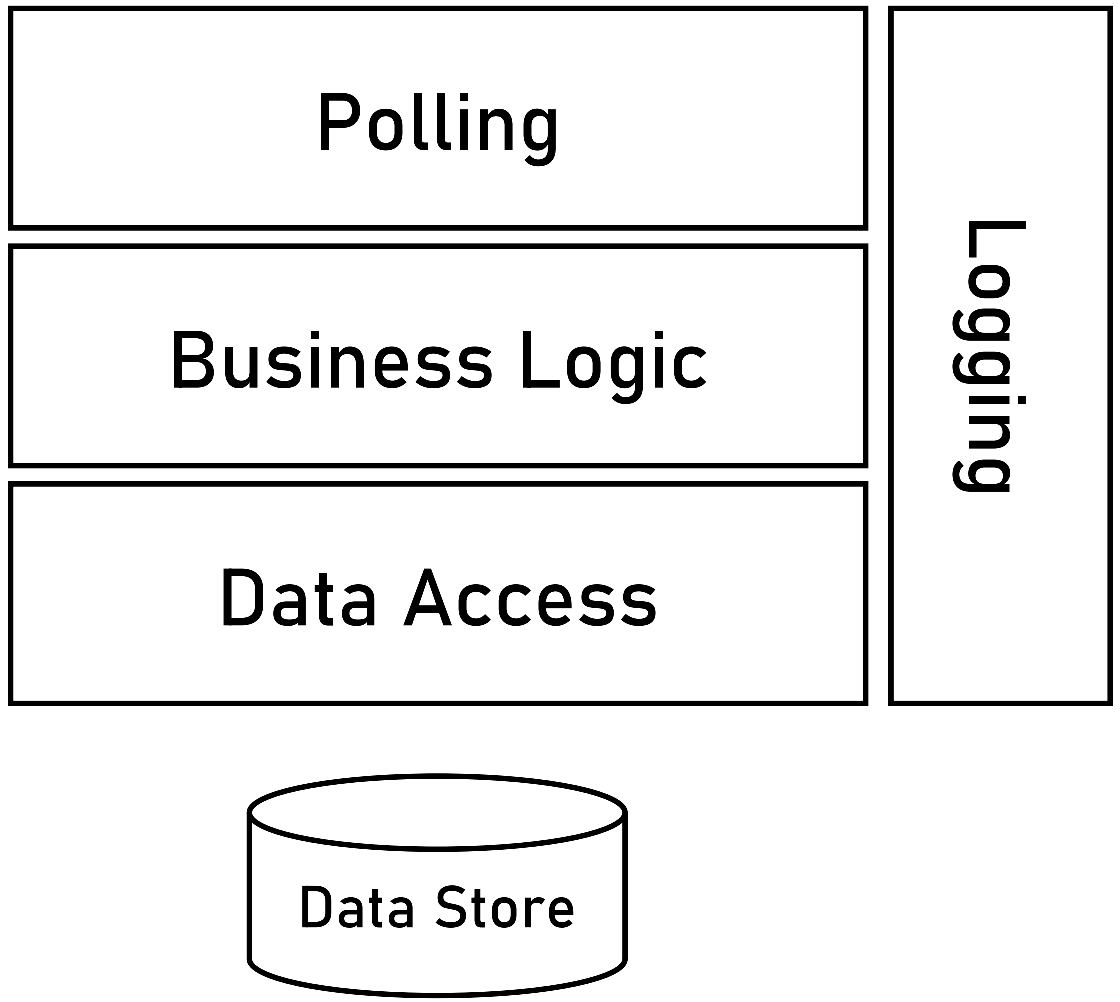


Architecture



Plug-In mechanism is a good idea here

Architecture



Info Service

What it does:

- Allows end users to query the data

What it doesn't:

- Displays the data

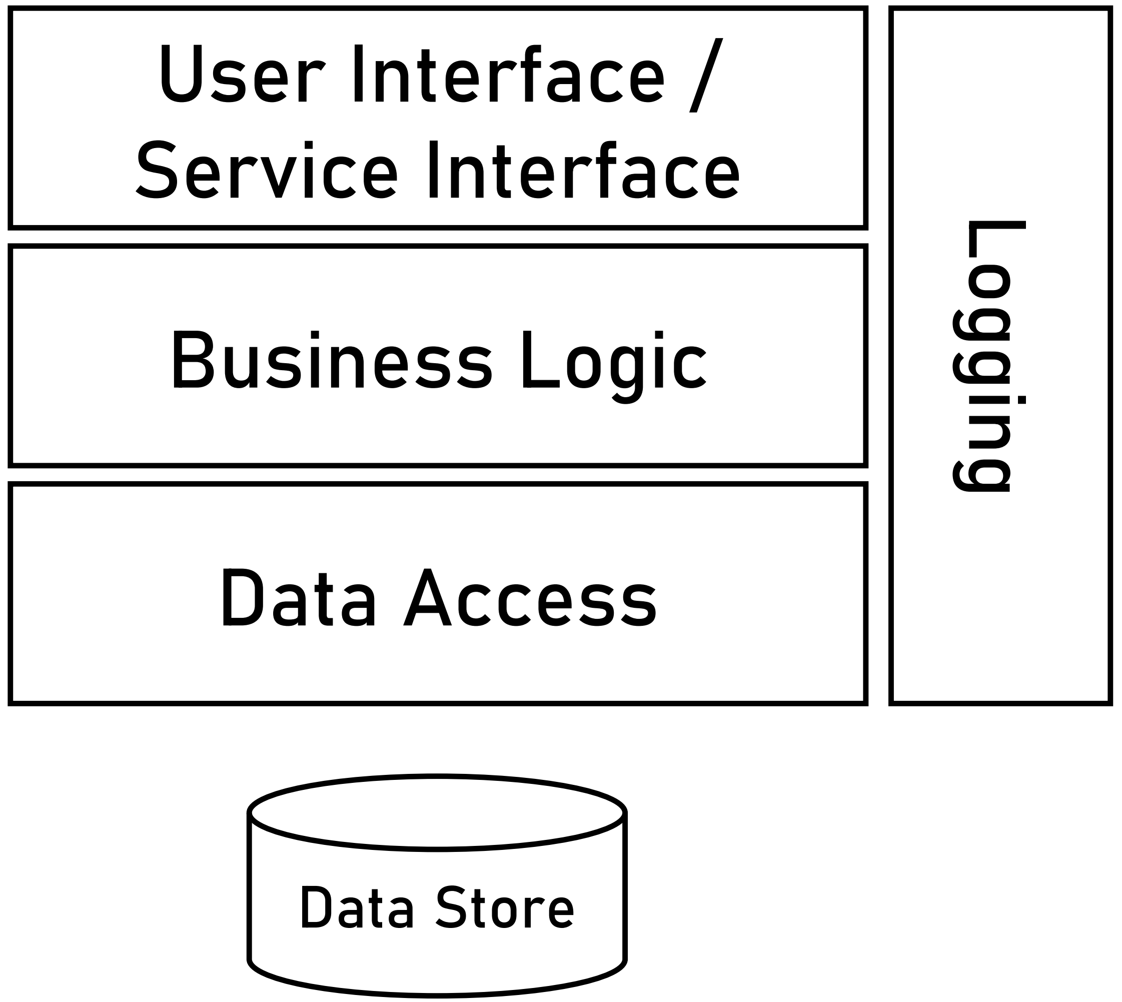
Application Type

- Web App & Web API ✓
- Mobile App ✗
- Console ✗
- Service ✗
- Desktop App ✗

Technology Stack

.NET Core

Architecture



API

- Current status of devices
- Past events

API

- Current status of devices for specific device and entire house
- Past events devices for specific device and entire house

API

Required Functionality:

Get all the updates for a specific house’s devices for a given time range

API

Required Functionality:

Get all the updates for a specific house’s devices for a given time range
Get the updates for a specific device for a given time range

API

Required Functionality:

Get all the updates for a specific house’s devices for a given time range
Get the updates for a specific device for a given time range
Get the current status of all the devices in a specific house

API

Required Functionality:

Get all the updates for a specific house's devices for a given time range
Get the updates for a specific device for a given time range
Get the current status of all the devices in a specific house
Get the current status of a specific device

Two Factors for API Design:

- Path
- Return code & contents

REST Refresher

Retrieve device #17:

```
GET /api/devices/17
```

```
200 OK
{
        “deviceId” : “17”,
        “type”     : “thermostat”,
        “houseId”  : “5331”
}
```

API

Required Functionality:

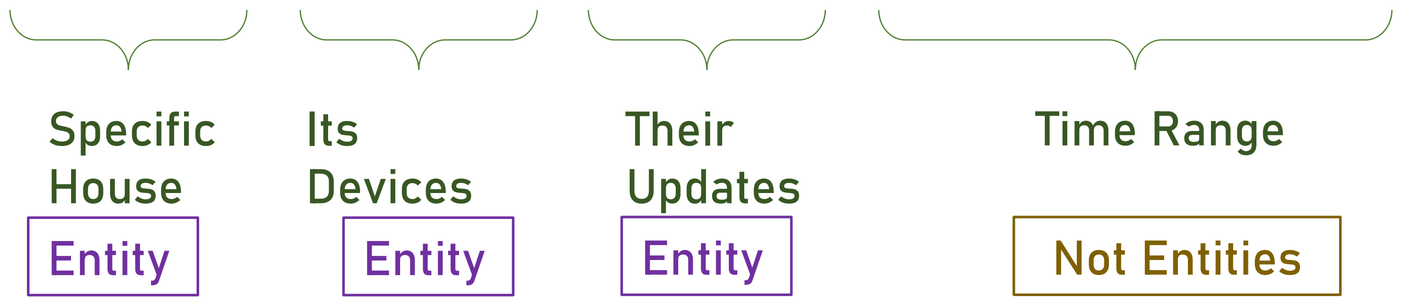
Get all the updates for a specific house’s devices for a given time range

API

Required Functionality:

Get all the updates for a specific house’s devices for a given time range

GET /api/house/*houseId*/devices/updates?from=*from*&to=*to*



Why not:

GET /api/house/*houseId*/devices/*from*/*to*

API

Returns:

```
GET /api/house/houseId/devices/updates?from=from&to=to
```

```
200 OK
```

```
404 Not Found
```

API

Functionality	Path	Return Codes
Get all the updates for a specific house's devices for a given time range	`GET /api/house/`*`houseId`*`/devices/updates?from=`*`from`*`&to=`*`to`*	200 OK 404 Not Found
Get the updates for a specific device for a given time range	`GET /api/device/`*`deviceId`*`/updates?from=`*`from`*`&to=`*`to`*	200 OK 404 Not Found
Get the current status of all the devices in a specific house	`GET /api/house/`*`houseId`*`/devices/status/current`	200 OK 404 Not Found
Get the current status of a specific device	`GET /api/device/`*`deviceId`*`/status/current`	200 OK 404 Not Found

Architecture Document

- Background ✓
- Requirements ✓
- Overall Architecture ✓
- Services' Drill Down ✓
- Executive Summary ✓

Get the Document!

Look in the Resources!

Contains everything we discussed

Use it as a template!

Do not remove the copyright notice

Read it thoroughly