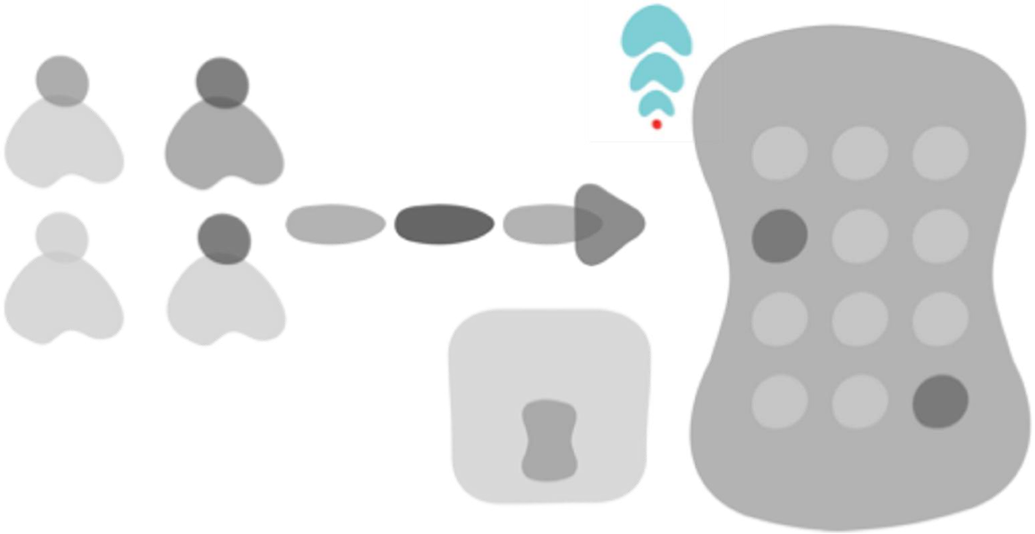
Architecture Katas

Winter 2024



Neal Ford
Thoughtworks
Director / Software Architect / Meme Wrangler
http://www.nealford.com

Mark Richards
Independent Consultant
Hands-on Software Architect, Published Author
Founder, DeveloperToArchitect.com
@markrichardssa

Semi-Finals

The Business Problem

MonitorMe

StayHealthy, Inc. is a large and highly successful medical software company located in San Francisco, California, US. They currently have 2 popular cloud-based SAAS products: *MonitorThem* and *MyMedicalData*.

MonitorMe

StayHealthy, Inc. is a large and highly successful medical software company located in San Francisco, California, US. They currently have 2 popular cloud-based SAAS products: *MonitorThem* and *MyMedicalData*.

MonitorThem a comprehensive data analytics platform that is used for hospital trend and performance analytics—alert response times, patient health problem analytics, patient recovery analysis, and so on.

MonitorMe

StayHealthy, Inc. is a large and highly successful medical software company located in San Francisco, California, US. They currently have 2 popular cloud-based SAAS products: *MonitorThem* and *MyMedicalData*.

MonitorThem a comprehensive data analytics platform that is used for hospital trend and performance analytics—alert response times, patient health problem analytics, patient recovery analysis, and so on.

MyMedicalData is a comprehensive cloud-based patient medical records system used by doctors, nurses, and other heath professionals to record and track a patients heath records with guaranteed partitioning between patient records.

MonitorMe

StayHealthy, Inc. is a large and highly successful medical software company located in San Francisco, California, US. They currently have 2 popular cloud-based SAAS products: *MonitorThem* and *MyMedicalData*.

MonitorThem a comprehensive data analytics platform that is used for hospital trend and performance analytics—alert response times, patient health problem analytics, patient recovery analysis, and so on.

MyMedicalData is a comprehensive cloud-based patient medical records system used by doctors, nurses, and other heath professionals to record and track a patients heath records with guaranteed partitioning between patient records.

StayHealthy, Inc. is now expanding into the medical monitoring market, and is in need of a new medical patient monitoring system for hospitals that monitors a patients vital signs using proprietary medical monitoring devices built by StayHealthy, Inc.

Architectural Katas Judges

Jacqui Read
Consultant Software Architect
linkedin.com/in/jacquelineread
@tekiegirl

Andrew Harmel-Law
Tech Principal, Thoughtworks
linkedin.com/in/andrewharmellaw
@al94781

Architectural Katas Judges

Diana Montalion
Founder, Mentrix Group
linkedin.com/in/dianamontalion/
@dianamontalion

Sergey Zinchenko
Technical Director, SD Interra
linkedin.com/in/sergey-zinchenko-2836a51

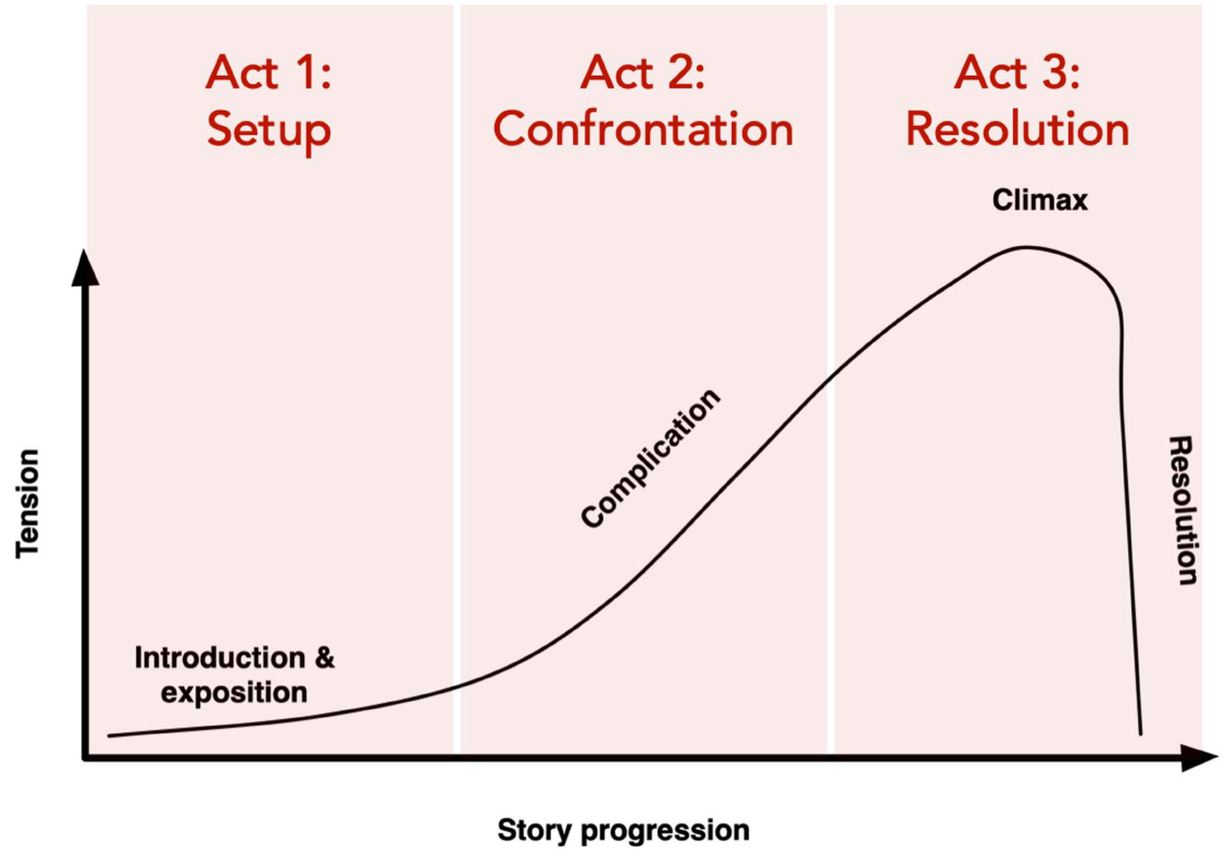
Pro Tips

PRO TIP

Tell a compelling story about your architectural solution

PRO TIP

Tell a compelling story about your architectural solution



PRO TIP

Make sure your solution reflects your architecture characteristics

Integration

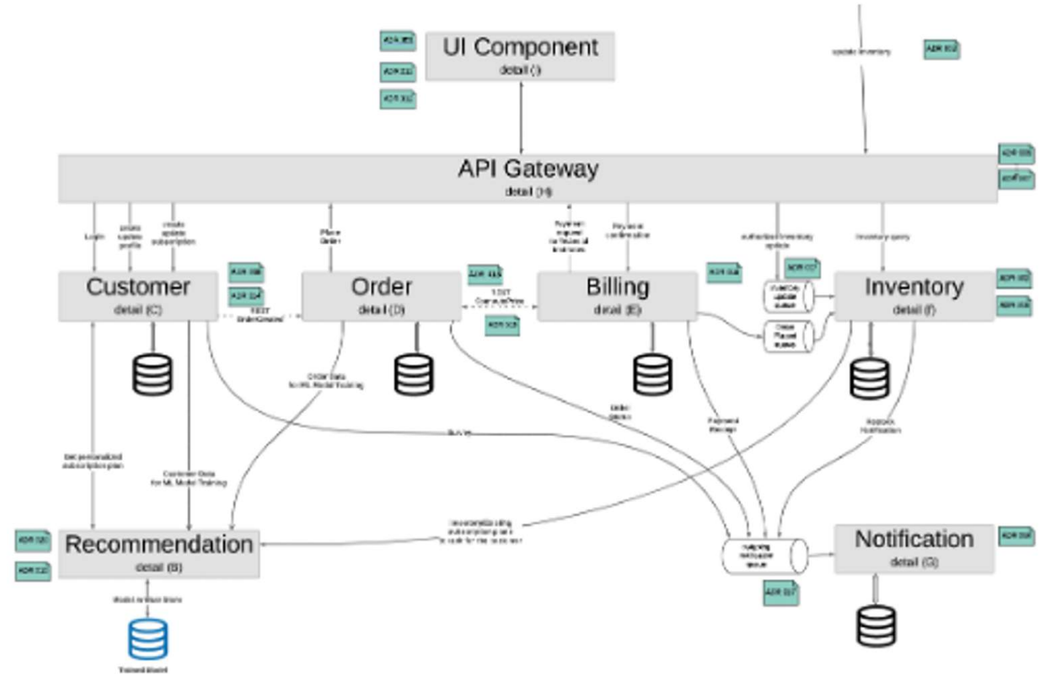
Feasibility

Agility

Availability

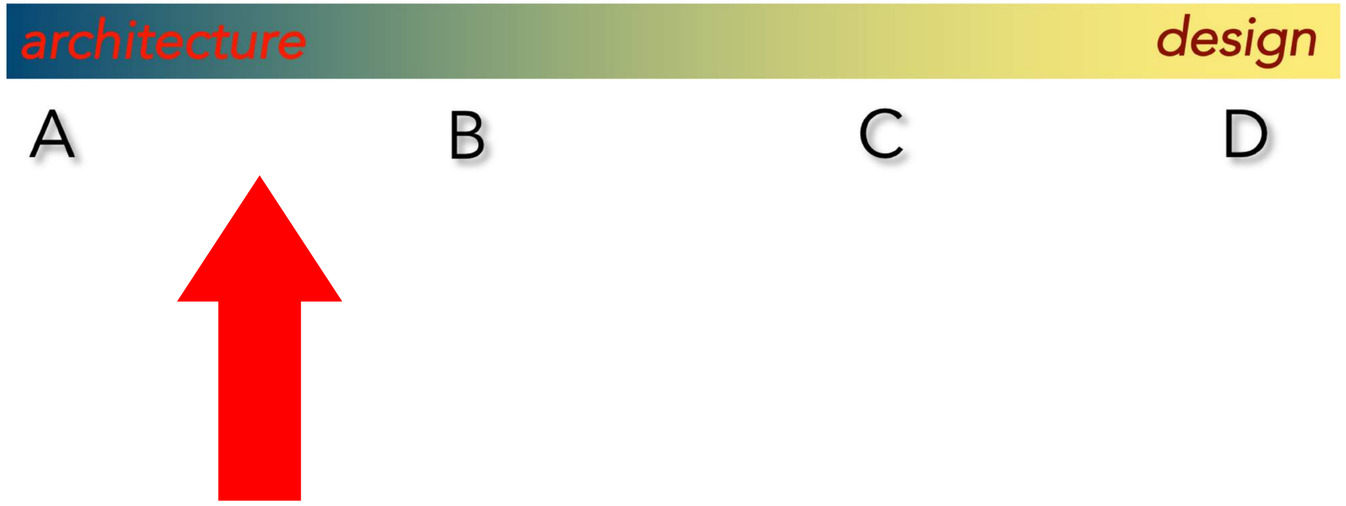
Security

Scalability



PRO TIP

Properly justify your architecture decisions



Date...../...../.....

Title

Context

...

Decision

...

Consequences

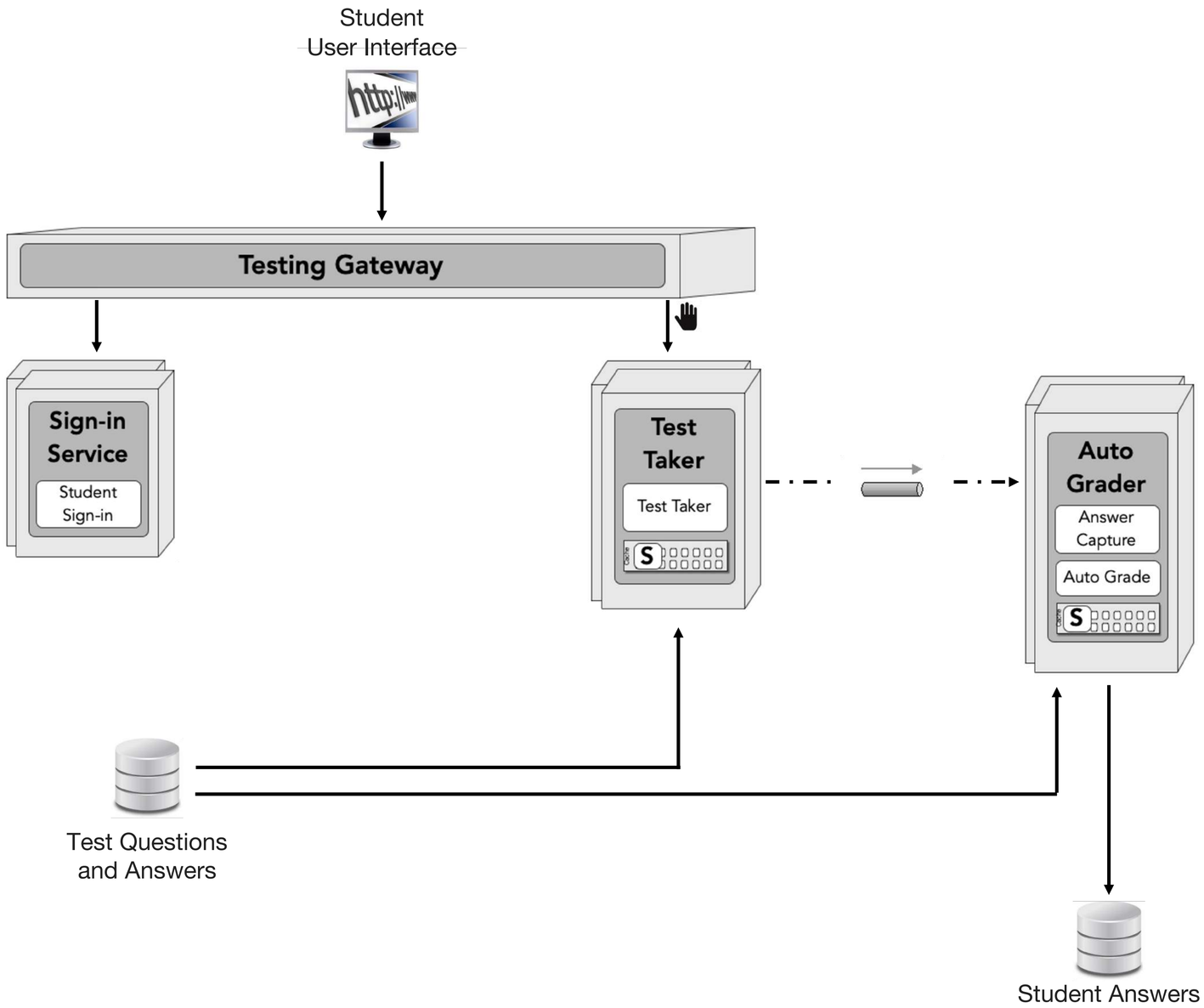
...

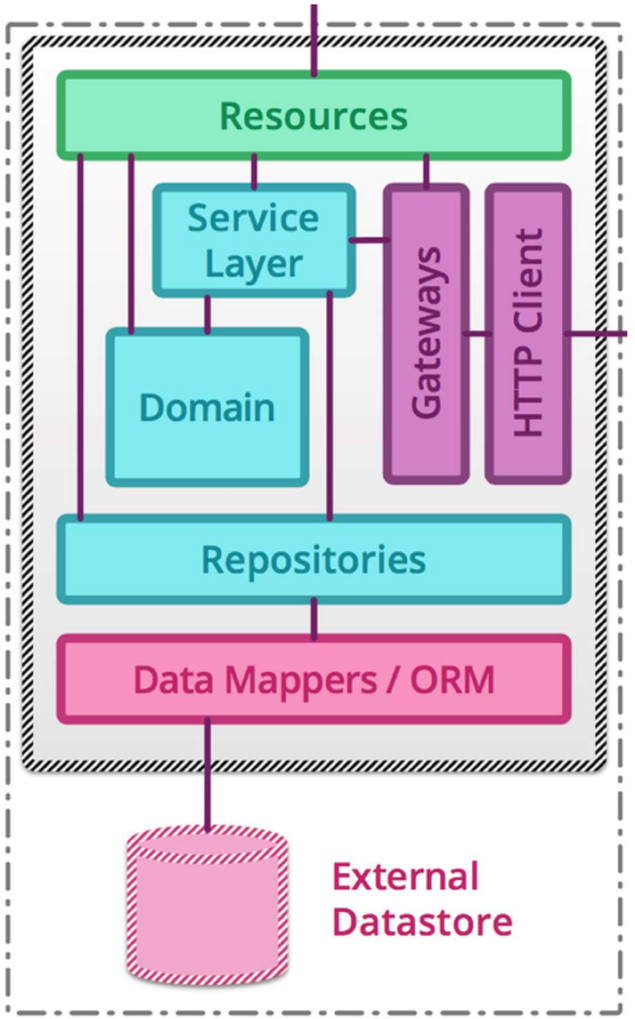
Tradeoffs

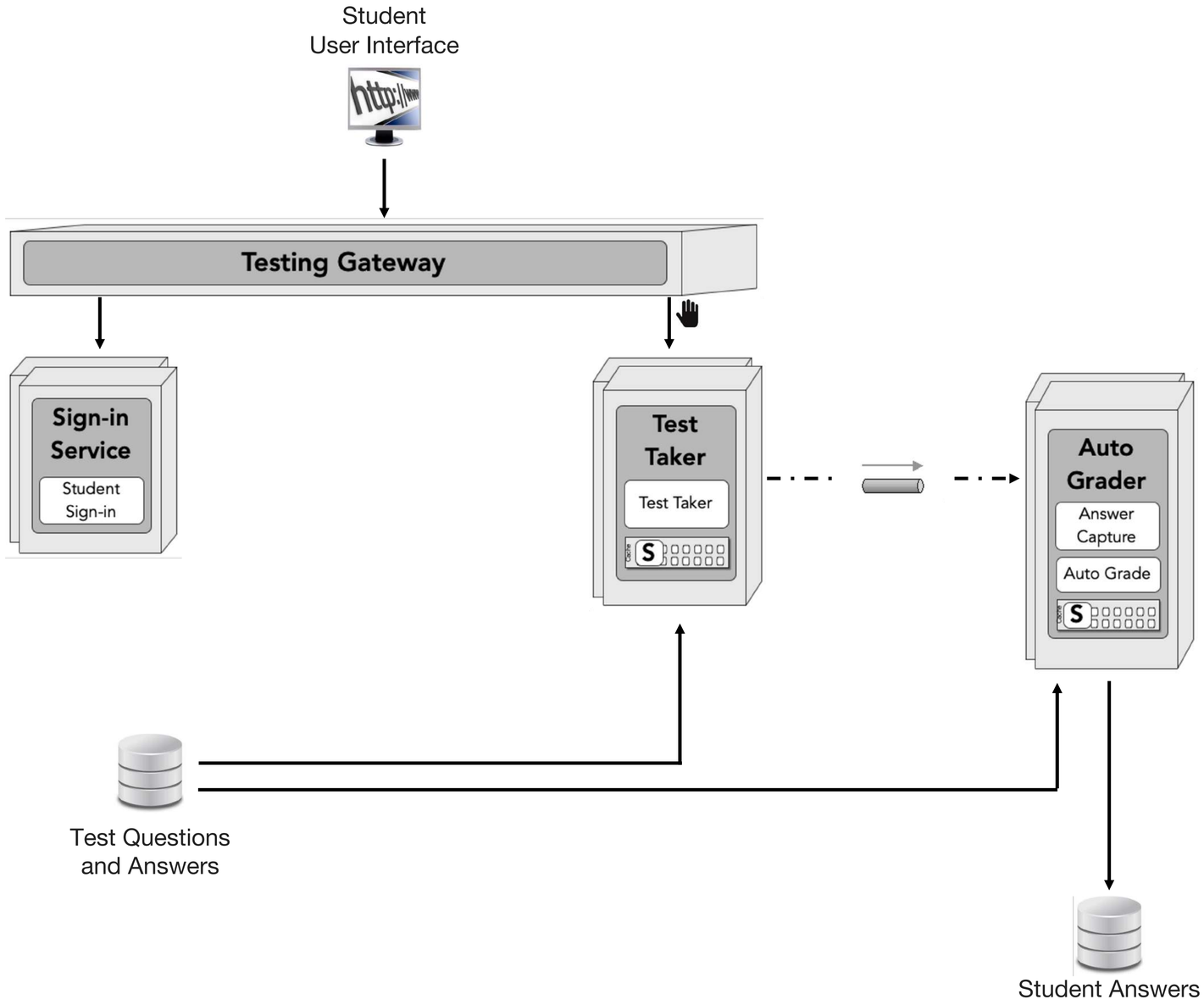
...

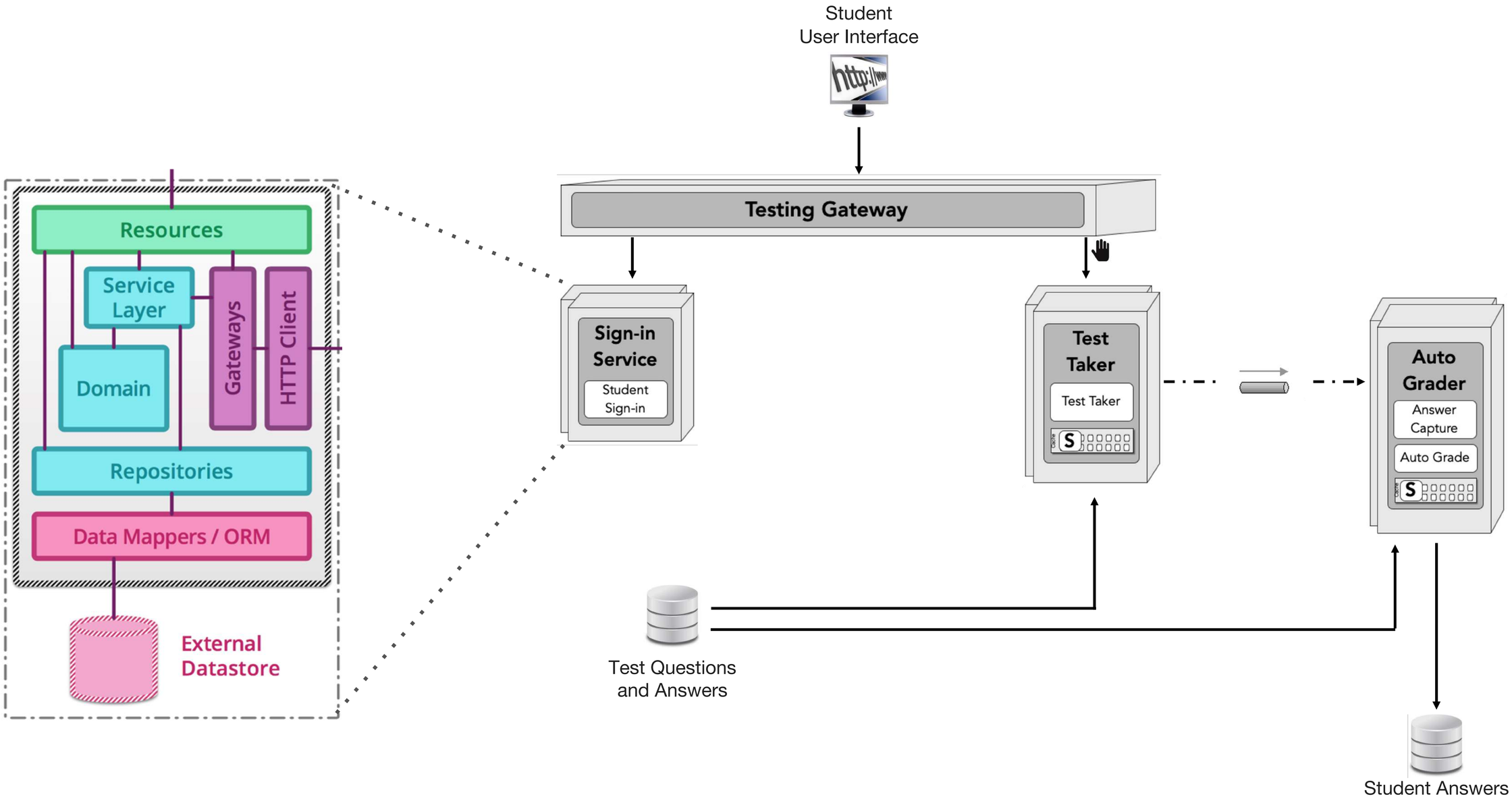
PRO TIP

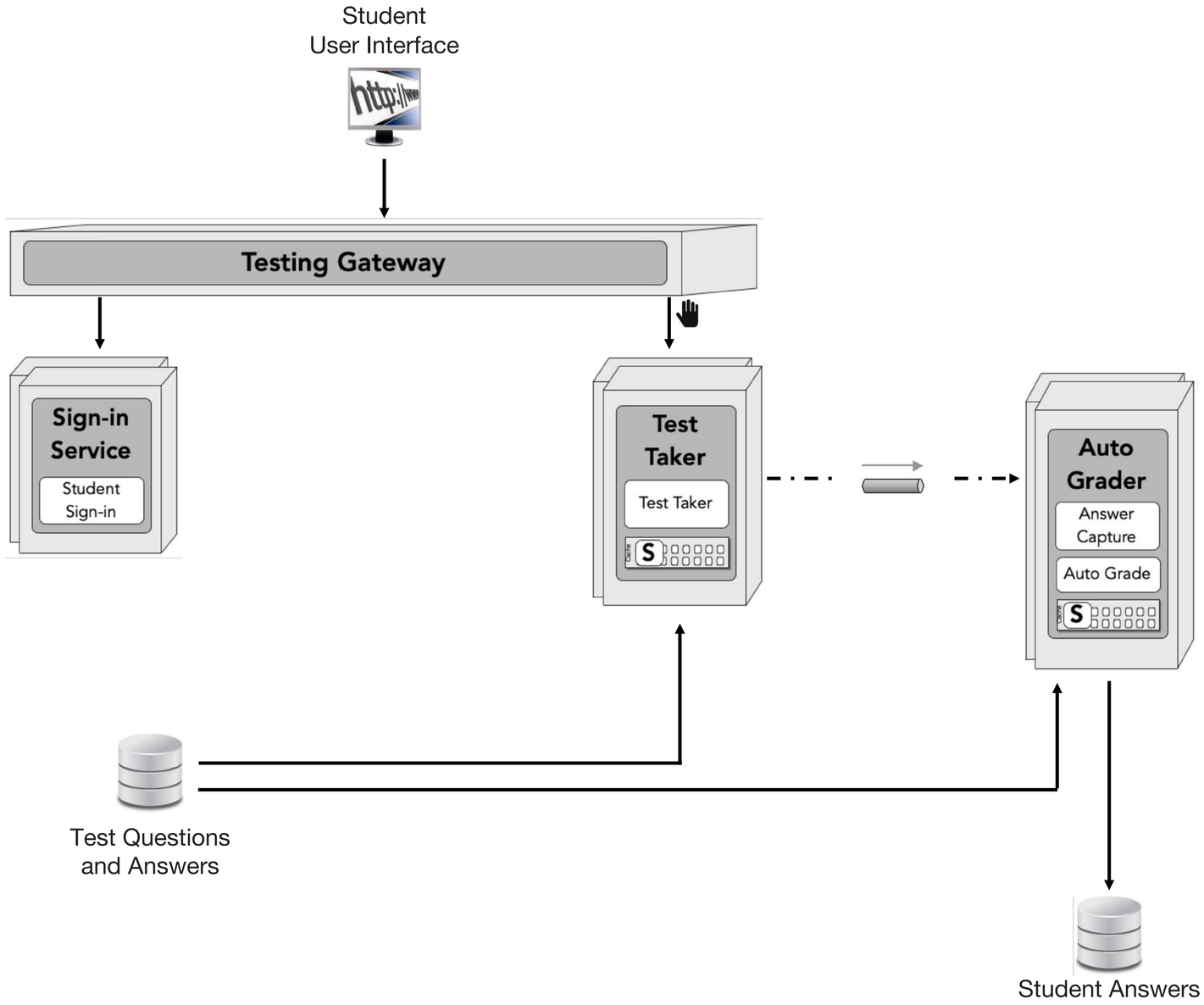
representational consistency



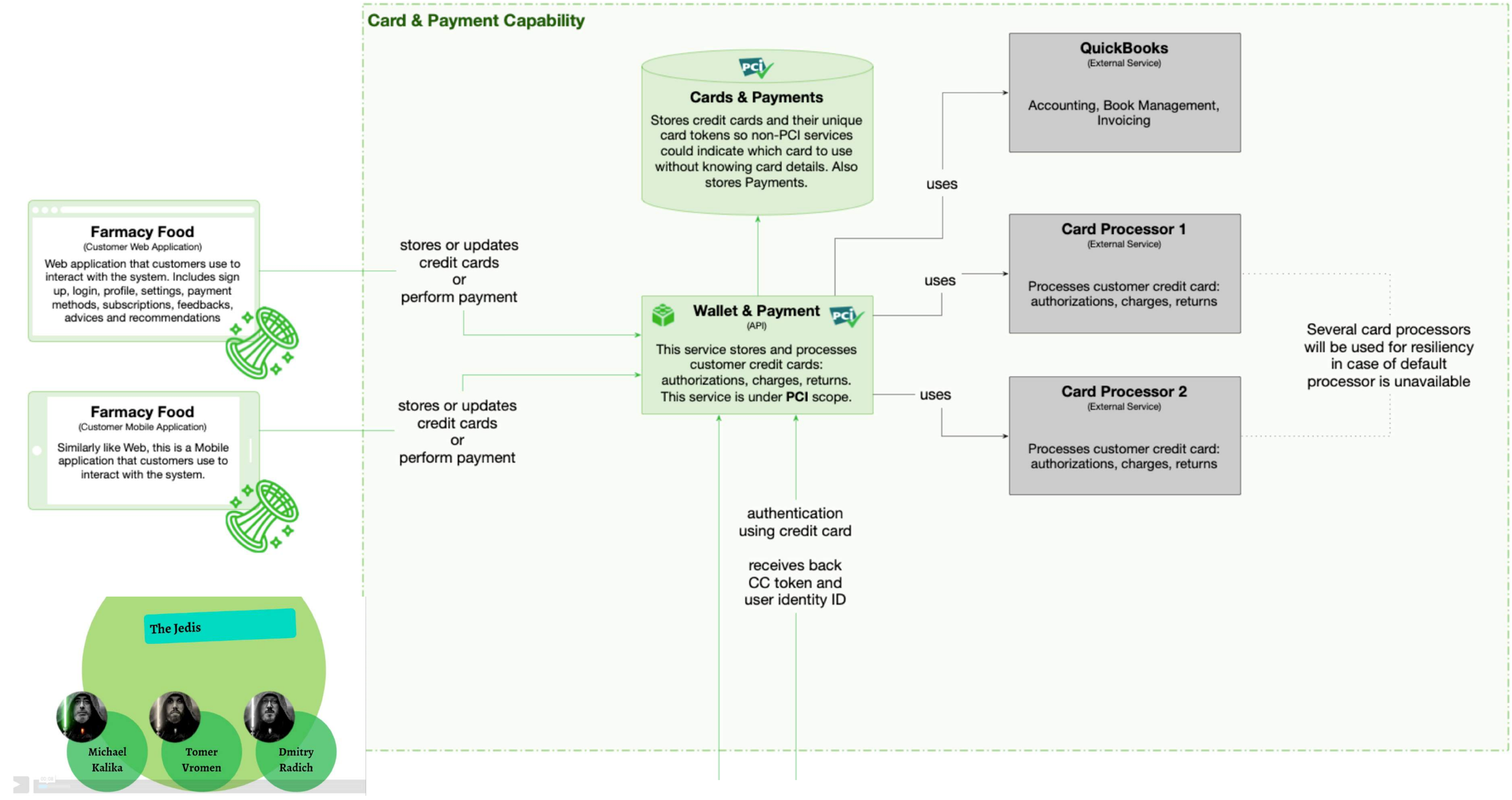

Resources
Service Layer
Gateways
HTTP Client
Domain
Repositories
Data Mappers / ORM
External Datastore

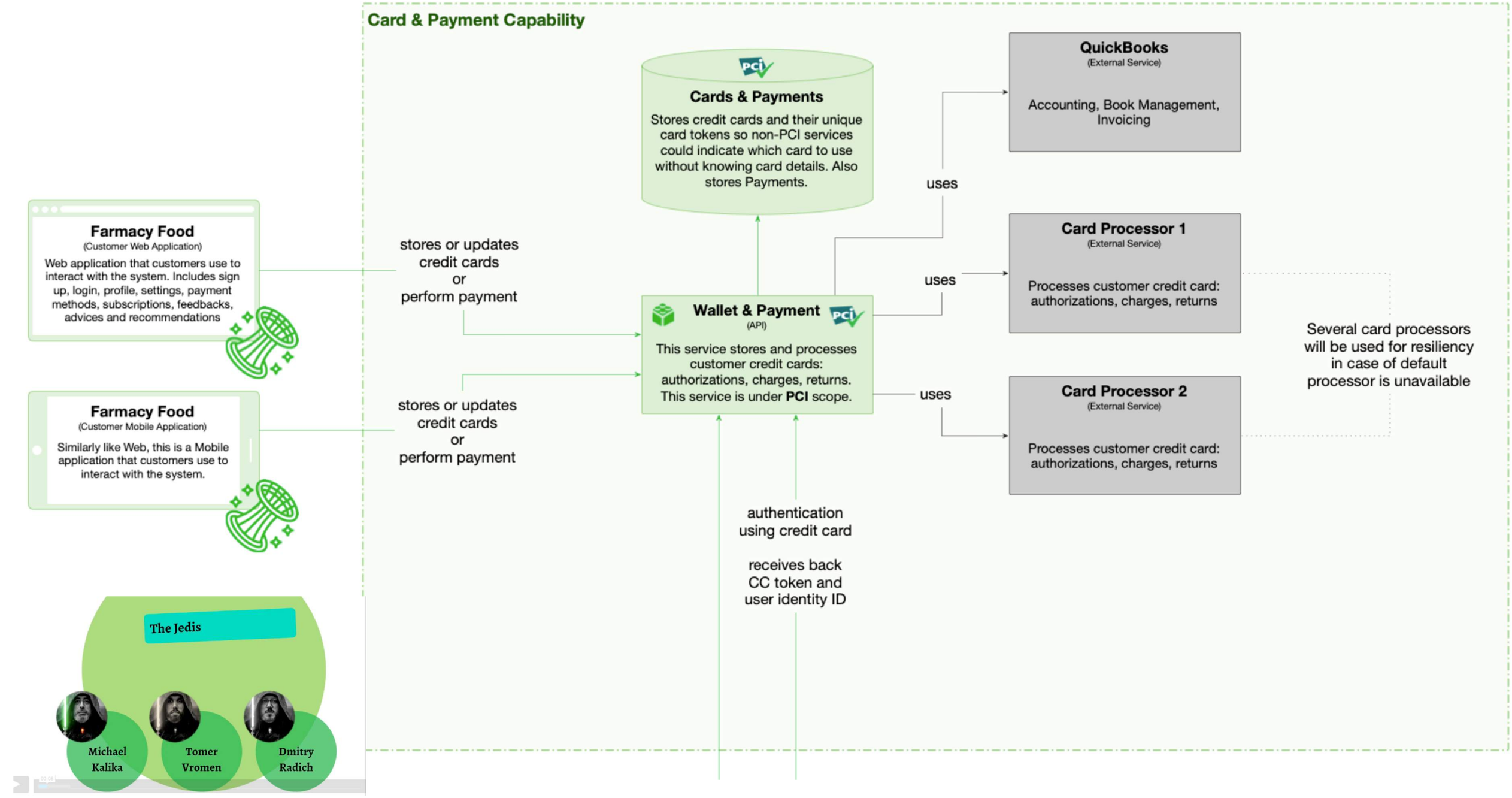

Student
User Interface
Testing Gateway
Sign-in
Service
Student
Sign-in
Test
Taker
Test Taker
Auto
Grader
Answer
Capture
Auto Grade
Test Questions
and Answers
Student Answers

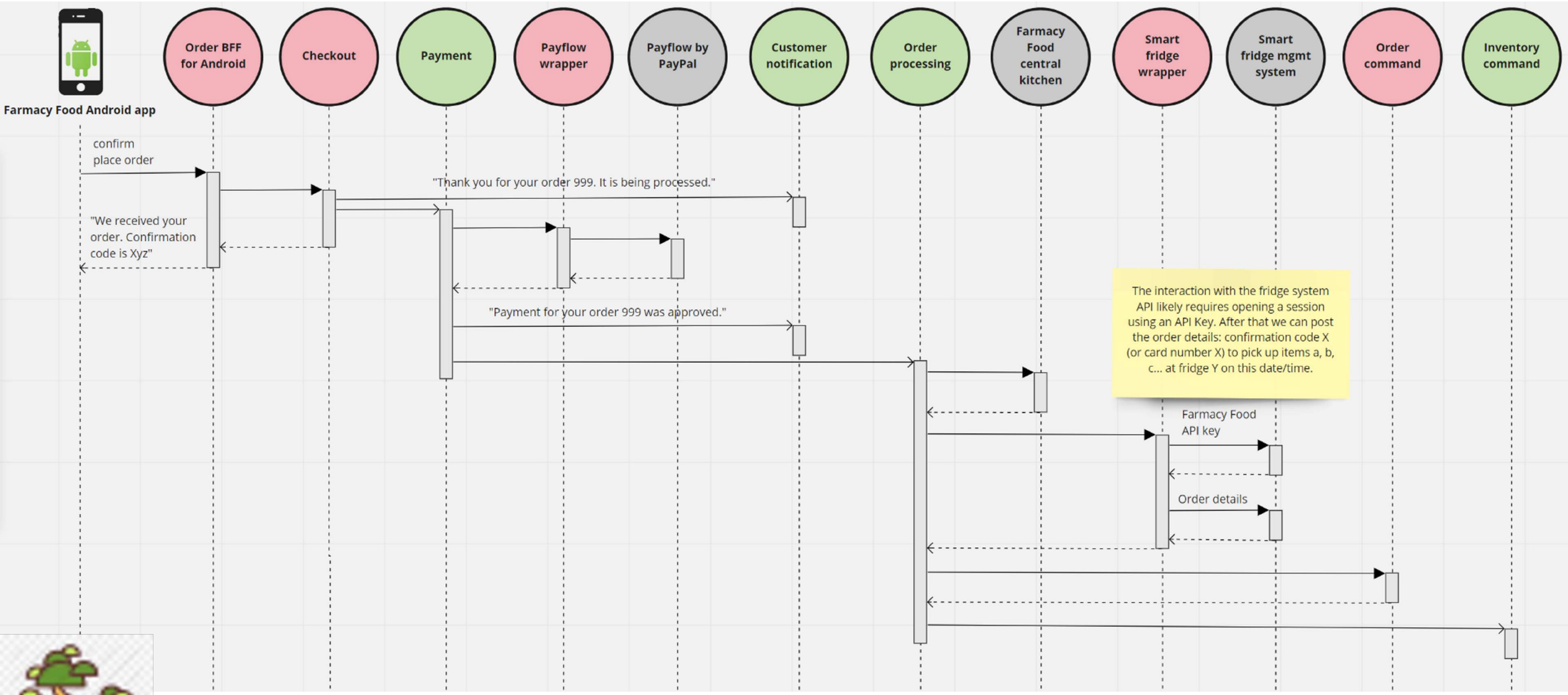

Student
User Interface
Testing Gateway
Resources
Service Layer
Domain
Gateways
HTTP Client
Repositories
Data Mappers / ORM
External Datastore
Sign-in Service
Student Sign-in
Test Taker
Test Taker
Auto Grader
Answer Capture
Auto Grade
Test Questions and Answers
Student Answers

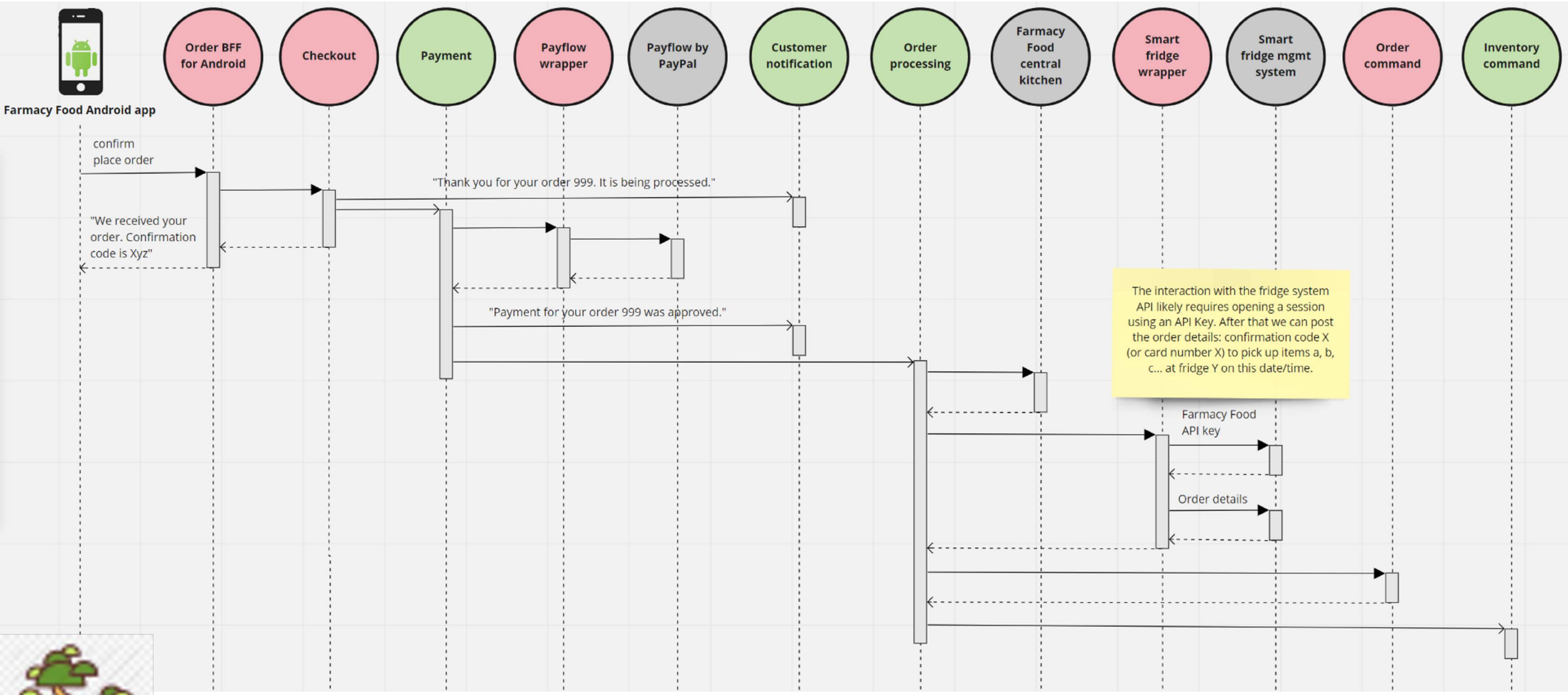

Student
User Interface
Testing Gateway
Sign-in
Service
Student
Sign-in
Test
Taker
Test Taker
Auto
Grader
Answer
Capture
Auto Grade
Test Questions
and Answers
Student Answers

Architecture Diagram Types

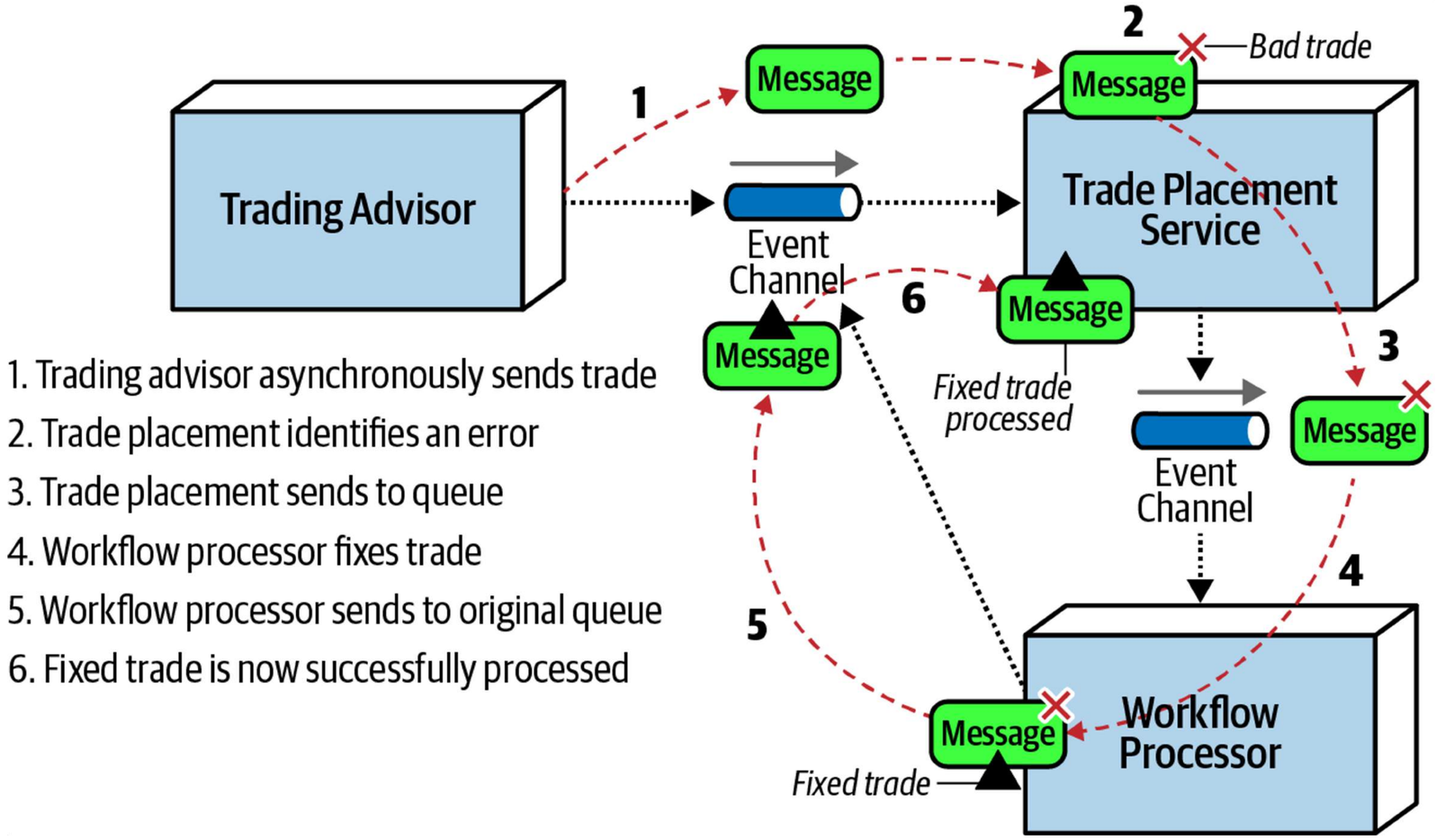

Card & Payment Capability
Cards & Payments
Stores credit cards and their unique card tokens so non-PCI services could indicate which card to use without knowing card details. Also stores Payments.
QuickBooks
(External Service)
Accounting, Book Management, Invoicing
uses
Card Processor 1
(External Service)
Processes customer credit card: authorizations, charges, returns
uses
Several card processors will be used for resiliency in case of default processor is unavailable
Farmacy Food
(Customer Web Application)
Web application that customers use to interact with the system. Includes sign up, login, profile, settings, payment methods, subscriptions, feedbacks, advices and recommendations
stores or updates credit cards or perform payment
Wallet & Payment
(API)
This service stores and processes customer credit cards: authorizations, charges, returns. This service is under PCI scope.
uses
Card Processor 2
(External Service)
Processes customer credit card: authorizations, charges, returns
Farmacy Food
(Customer Mobile Application)
Similarly like Web, this is a Mobile application that customers use to interact with the system.
stores or updates credit cards or perform payment
authentication using credit card
receives back CC token and user identity ID
The Jedis
Michael Kalika
Tomer Vromen
Dmitry Radich


Card & Payment Capability
Cards & Payments
Stores credit cards and their unique card tokens so non-PCI services could indicate which card to use without knowing card details. Also stores Payments.
QuickBooks
(External Service)
Accounting, Book Management, Invoicing
uses
Card Processor 1
(External Service)
Processes customer credit card: authorizations, charges, returns
uses
Several card processors will be used for resiliency in case of default processor is unavailable
Farmacy Food
(Customer Web Application)
Web application that customers use to interact with the system. Includes sign up, login, profile, settings, payment methods, subscriptions, feedbacks, advices and recommendations
stores or updates credit cards or perform payment
Wallet & Payment
(API)
This service stores and processes customer credit cards: authorizations, charges, returns. This service is under PCI scope.
uses
Card Processor 2
(External Service)
Processes customer credit card: authorizations, charges, returns
Farmacy Food
(Customer Mobile Application)
Similarly like Web, this is a Mobile application that customers use to interact with the system.
stores or updates credit cards or perform payment
authentication using credit card
receives back CC token and user identity ID
The Jedis
Michael Kalika
Tomer Vromen
Dmitry Radich

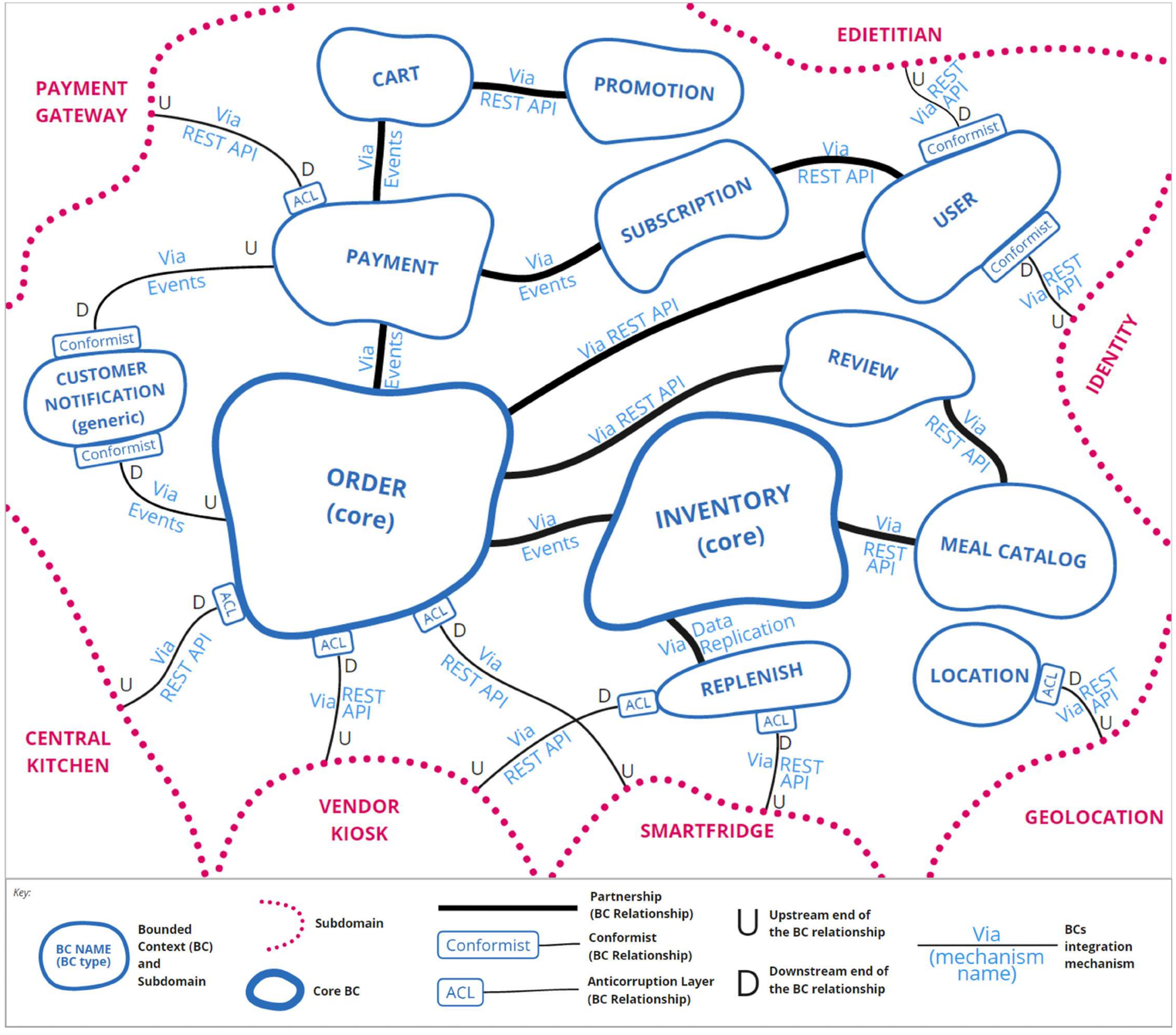

Farmacy Food Android app
Order BFF for Android
Checkout
Payment
Payflow wrapper
Payflow by PayPal
Customer notification
Order processing
Farmacy Food central kitchen
Smart fridge wrapper
Smart fridge mgmt system
Order command
Inventory command
confirm
place order
"Thank you for your order 999. It is being processed."
"We received your order. Confirmation code is Xyz"
"Payment for your order 999 was approved."
The interaction with the fridge system API likely requires opening a session using an API Key. After that we can post the order details: confirmation code X (or card number X) to pick up items a, b, c... at fridge Y on this date/time.
Farmacy Food API key
Order details


Farmacy Food Android app
Order BFF for Android
Checkout
Payment
Payflow wrapper
Payflow by PayPal
Customer notification
Order processing
Farmacy Food central kitchen
Smart fridge wrapper
Smart fridge mgmt system
Order command
Inventory command
confirm
place order
"Thank you for your order 999. It is being processed."
"We received your order. Confirmation code is Xyz"
"Payment for your order 999 was approved."
The interaction with the fridge system API likely requires opening a session using an API Key. After that we can post the order details: confirmation code X (or card number X) to pick up items a, b, c... at fridge Y on this date/time.
Farmacy Food API key
Order details

structure + serial numbering



1. Trading advisor asynchronously sends trade
2. Trade placement identifies an error
3. Trade placement sends to queue
4. Workflow processor fixes trade
5. Workflow processor sends to original queue
6. Fixed trade is now successfully processed


EDIETITIAN
PAYMENT GATEWAY
CART
PROMOTION
SUBSCRIPTION
USER
PAYMENT
CUSTOMER NOTIFICATION (generic)
ORDER (core)
REVIEW
INVENTORY (core)
MEAL CATALOG
REPLENISH
LOCATION
IDENTITY
CENTRAL KITCHEN
VENDOR KIOSK
SMARTFRIDGE
GEOLOCATION
Via REST API
Via Events
Via Data Replication
Conformist
ACL
U
D
Key:
BC NAME (BC type)
Bounded Context (BC) and Subdomain
Subdomain
Core BC
Partnership (BC Relationship)
Conformist (BC Relationship)
Anticorruption Layer (BC Relationship)
Upstream end of the BC relationship
Downstream end of the BC relationship
Via (mechanism name)
BCs integration mechanism

Judges Criteria

Clarity of narrative, organization, and supporting documentation

Judges Criteria

Clarity of narrative, organization, supporting documentation

Clarity (noun):

- the quality of being **coherent** and **intelligible.**
- the quality of **transparency** or **purity**.

Judges Criteria

Clarity of narrative, organization, supporting documentation

Architecture is Trust Building

Can I trust your thinking process?

Judges Criteria

Clarity - narrative, organization, supporting documentation

There is no right answer - just your recommendation

- You are **making a case for your idea**.
- **Show Me** (Be Missouri). How did you arrived at the conclusion? Help me be in the room. What cognitive steps did you take?
- What **reasons support your conclusions?**
- **What else did you consider**? Why did you not choose it? (Discernment!)
- **What don't you know?** You are guessing. Embrace the uncertainty. How will you learn?

Judges Criteria

Clarity - narrative, organization, supporting documentation

<whispers>

I’d love at least one artifact that isn’t a template.

</whispers>

Judges Criteria

Clarity - narrative, organization, supporting documentation

Your artifacts are a knowledge system

- **Architecture is a landscape. H**elp me not get lost in it. Guide me towards understanding.
- Your landing page is **an entry point**. Ideally, I can find the relevant information from there. (Think executive summary.)
- **Create relationships** between your artifacts. Build conceptual integrity into your architecture.
  - Fred Brooks says that “*conceptual integrity is the most important consideration for software system design.*”

Judges Criteria

Clarity - narrative, organization, supporting documentation

Create Relationships

- **Use layers:** Higher level information can link “down” to more details.
- **Use links:** interrelated information can link across artifacts.
- *CHEAT TIP*: Use a markdown tool designed for this.

Judges Criteria

Clarity - narrative, organization, supporting documentation

Why?

- Keep the WHY connected to the WHAT and the HOW.
- The Second Law of Software Architecture: WHY is more important than HOW. – *Fundamentals of Software Architecture by Neal Ford and Mark Richards*
  - *Why do you need performance? Will people die if it's slow?*
- Interrelate the **business** context with the technology context (use linking)

Judges Criteria

Clarity - narrative, organization, supporting documentation

Define

- Link to the glossary for anything that might be interpreted in multiple ways by multiple people.
- Or anything that justifies your decisions.

Judges Criteria

Clarity - narrative, organization, supporting documentation

MonitorMe requirements

Upon receiving the requirements from the stakeholders, our team proceeded to analyze and distill the information provided. This involved a thorough examination and refinement process to extract the essential elements and key details. By conducting this requirements distillation, we aimed to ensure a clear understanding of the project scope and objectives, enabling us to proceed with the development process effectively and efficiently. We came up with the business overview and distilled requirements. Frankly speaking, all these requirements can be covered by the four main data flows:

- Collecting data from sensors and visualizing them on the nurse's station.
- Alerting the medical staff on both mobile devices and the nurse's station when anomalies are detected based on defined and easily extendable rules.
- Browsing the patient's vital signs (maximum 24 hours).
- Sending the snapshot to *MyMedicalData*.

Exploring components

Judges Criteria

Clarity - narrative, organization, supporting documentation

Vitals Monitoring System	Software running on the physical devices connected to patient. This system reads the patient vitals through the sensors on the device and sync the data to a centralized system for further monitoring.
Concurrency	MonitorMe will be receiving a lot of data for every vital sign from the patients. It is crucial that incoming data gets processed simultaneously so alerts can be sent in the quickest time possible when required to enable fast response times Requirements: • analyze each patient's vital signs • Issue alert a medical professional • Trend or threshold (+ device failure) • More vital sign monitoring devices

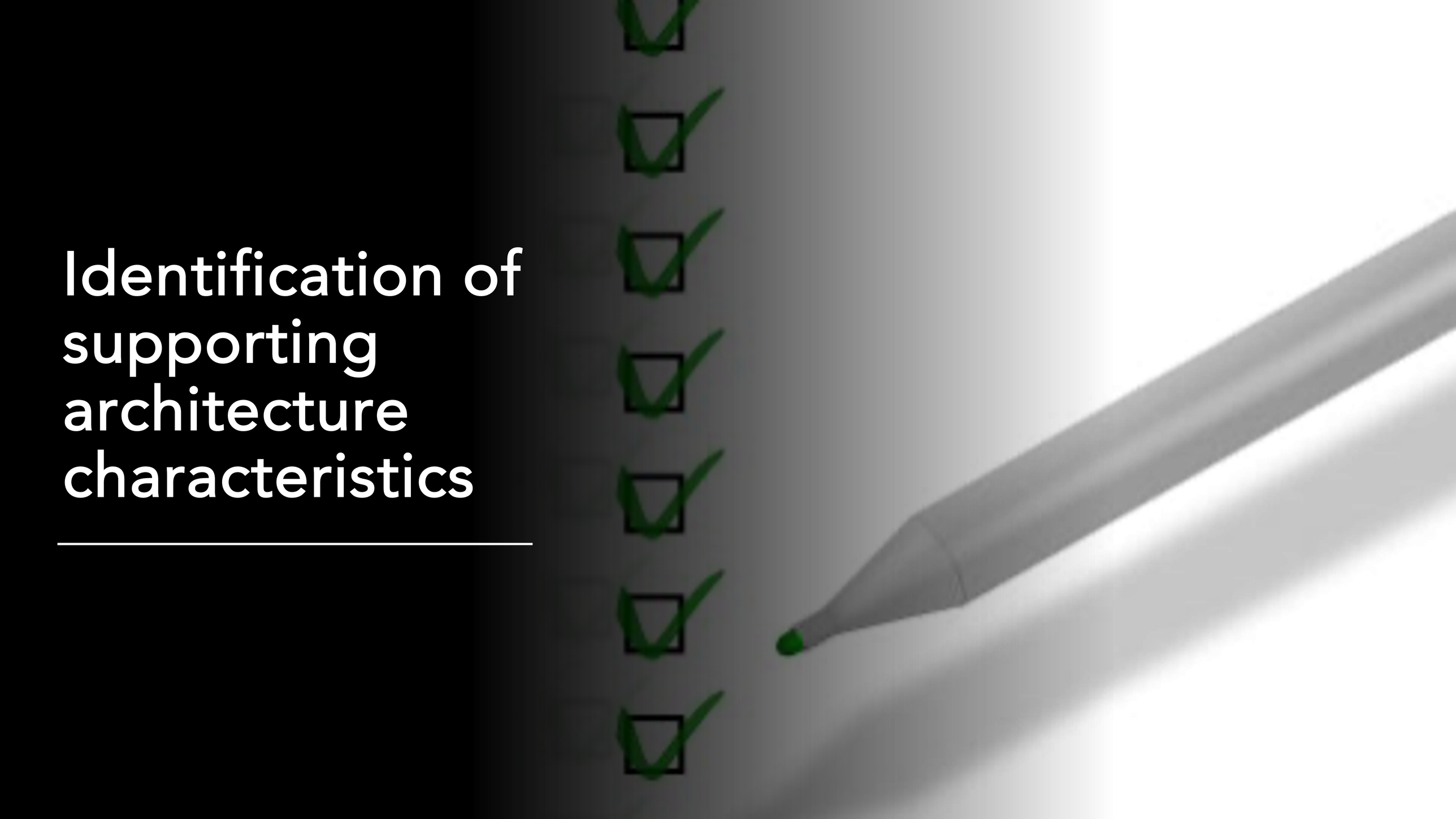
Judges Criteria

Clarity - narrative, organization, supporting documentation

VPN to the external systems	To decide in further steps whether establishing VPN for access to the MedicalData is worth considering for enhanced security and privacy. On this stage, it doesn't play significant role and this decision can be taken later.

Integration with MonitorThem	The integration with MonitorThem is not specified by stakeholders. The next step should clarify the point of integration, but it shouldn't change the architecture at all

Identification of supporting architecture characteristics



Judges Criteria

Supporting architecture characteristics

System/Project: MonitorMe
Architect/Team: BluzBrothers
Domain: Hospital
Date: 2024/02/17

Candidate Architecture Characteristics

performance	data integrity	deployability
responsiveness	data consistency	testability
availability	adaptability	abstraction
fault tolerance	extensibility	workflow
scalability	interoperability	configurability
elasticity	concurrency	recoverability

others: _____

a / b denotes characteristics that are related; some systems only need one of these, other systems may need both

Top 3	Driving Characteristics	Implicit Characteristics
☒	Performance	feasibility (cost/time)
☐	Security	security
☐	Availability	maintainability
☒	Evolvability	observability
☒	Elasticity	
☐	Fault-tolerance	Others Considered
☐	Configurability	

Instructions

- Identify no more than 7 driving characteristics.
- Pick the top 3 characteristics (in any order).
- Implicit characteristics can become driving characteristics if they are *critical* concerns.
- Add additional characteristics identified that weren't deemed as important as the list of 7 to the *Others Considered* list.

Architecture Characteristics Worksheet

System/Project: MonitorMe
Architect/Team: Team Low-Code
Domain: _____
Date: 14/02/2024

Candidate Architecture Characteristics

performance	data integrity	deployability
responsiveness	data consistency	testability
availability	adaptability	abstraction
fault tolerance	extensibility	workflow
scalability	interoperability	configurability
elasticity	concurrency	recoverability

others: _____

a / b denotes characteristics that are related; some systems only need one of these, other systems may need both

Top 3	Driving Characteristics	Implicit Characteristics
☐	performance	feasibility (cost/time)
☐	responsiveness	
■	concurrency	maintainability
■	availability	observability
☐	fault tolerance	
☐	Security	Others Considered
■	data integrity	recoverability
		abstraction
		configurability
		data consistency

Instructions

- Identify no more than 7 driving characteristics.
- Pick the top 3 characteristics (in any order).
- Implicit characteristics can become driving characteristics if they are *critical* concerns.
- Add additional characteristics identified that weren't deemed as important as the list of 7 to the *Others Considered* list.

Last updated August 30, 2023 — Created by Mark Richards, DeveloperToArchitect.com

Judges Criteria

Supporting architecture characteristics

Selected Characteristics

Top 3	Characteristics	Description
2	Performance	Medical professionals can't always be physically at a patients bed and will get informed in case of any anomalies detected for a patient. As this is a time critical thing and data needs to be visible as quickly as possible performance is one of the important characteristics for the system.
	Scalability	The system load is easy to forecast due to the defined maximum devices and patients per nurse station. However the system needs to be capable of handling additional load, when extending the range of available viral sign devices. It's also required to replicate this setup multiple in different hospitals.
1	Availability	The systems needs to have a high level vor availability due to the fact it monitors patients. An outage of critical components would put lives of the patients at risk.
	Flexibility	The overall system should be easy to be extended or adapted. Eventually different technology preferences are set when expanding oversees. This flexibility shall be provided however it's not the most important characteristic.
	Fault tolerance	Fault tolerance is a critical factor for the system as it needs to function also on outages. This covered by implementing redundancy within critical components and pays into the Availability characteristics
	Workflow	The application needs to have the ability to make decisions for anomaly detection and notification which we consider with this characteristic.
3	Agility	One of the most important business requirements is the ability to quickly adapt to new learnings or to extend the system by new vital sign devices which get developed within the industry. The system needs to be able to handle these frequent learnings StayHealthy Inc. will encounter.
	Interoperability	Event that the system is limited to some external systems, medical vital devices need to be replaced easily

Non-functional requirements

1. Availability The system should be designed to ensure high availability, meaning it should be accessible and usable by medical professionals at all times. This can be achieved through redundant systems, failover mechanisms, and regular maintenance and monitoring.
2. Reliability The system must be highly reliable, with minimal downtime or errors. This can be achieved through rigorous testing, quality assurance processes, and regular updates and maintenance. For example, while one sensor stops working, it should not impact other data collection. We should do multi samples for single report, so to make sure each individual measure error does not impact the accuracy, e.g. the Oxygen level is required to report every 5 seconds, but inside of each sensor device, we will implement the measurement of >10 times per seconds, this could even to be configurable by the healthcare professionals.
3. Performance and elasticity The system should be designed to handle a large volume of data traffic from users while maintaining fast response times and low latency. This can be achieved through: efficient message/data queue, architecture design, optimized database design, and appropriate hardware specifications.
4. Scalability The system should be able to scale up or down based on demand, allowing it to accommodate an increasing number of patients and data sources over time. This can be achived by structuring between 'nurse station' and 'MonitorMe', to make the flexibility for 'hospitals' or care centers to deploy with light weight or heavy weight.
5. Security Given the sensitive nature of the data, the system must ensure high levels of security to protect patient data from unauthorized access or breaches.

Judges Criteria

Supporting architecture characteristics

Driving characteristics	Justification
Concurrency	MonitorMe will be receiving a lot of data for every vital sign from the patients. It is crucial that incoming data gets processed simultaneously so alerts can be sent in the quickest time possible when required to enable fast response times Requirements: • analyze each patient's vital signs • Issue alert a medical professional • Trend or threshold (+ device failure) • More vital sign monitoring devices
Availability	Patients need to be monitored 24/7 and if problems arise MonitorMe must be able to act. Therefore, the system needs to be always available and working Requirements: • Store all vital sign reading for past 24 hr • View history, filter on time + vital sign • Has to be available 24/7
DataIntegrity	Data loss can be disastrous, especially if that data would trigger any alerts if it were to be analyzed. Data Integrity must be maintained to ensure correct data analysis Requirements: • Store all vital sign reading for past 24 hr • View history, filter on time + vital sign

Selected Architecture Characteristics

Top 3	Characteristic	Source
Y	Interoperability/Integration	Integration of patient's vital sign devices with MonitorMe (analyzing streaming data), and with MyMedicalData.
	Data Integrity	Vital sign data analyzed and recorded through MonitorMe must be as accurate as possible as the human lives are at stake.
	Scalability/Elasticity	StayHealthy, Inc. is looking towards adding more vital sign monitoring devices for MonitorMe in the future.
Y	[Real-Time] Performance	Vital signs data is send to consolidated monitoring screen with an average response time of 1 second or less.
Y	High Availability	System needs to be available all the times as the medical professional need to monitor patients vitals and take decision based on them.
	Deployability	We are proposing a micro service architecture with a service for each device ,being able to deploy the whole system seemlessly is important.

Implicit Architecture Characteristics

The following are a bedrock of architecture characteristics. They may not affect the *structure* but will feed into the overall architecture.

- Usability
- Security, authentication and authorization
- Maintainability
- Simplicity or observability?

Note that Configurability was considered for patients.

Judges Criteria

Supporting architecture characteristics

Think about

Judges Criteria

Supporting architecture characteristics

Think about

mapping characteristics

to requirements

Judges Criteria

Supporting architecture characteristics

Think about

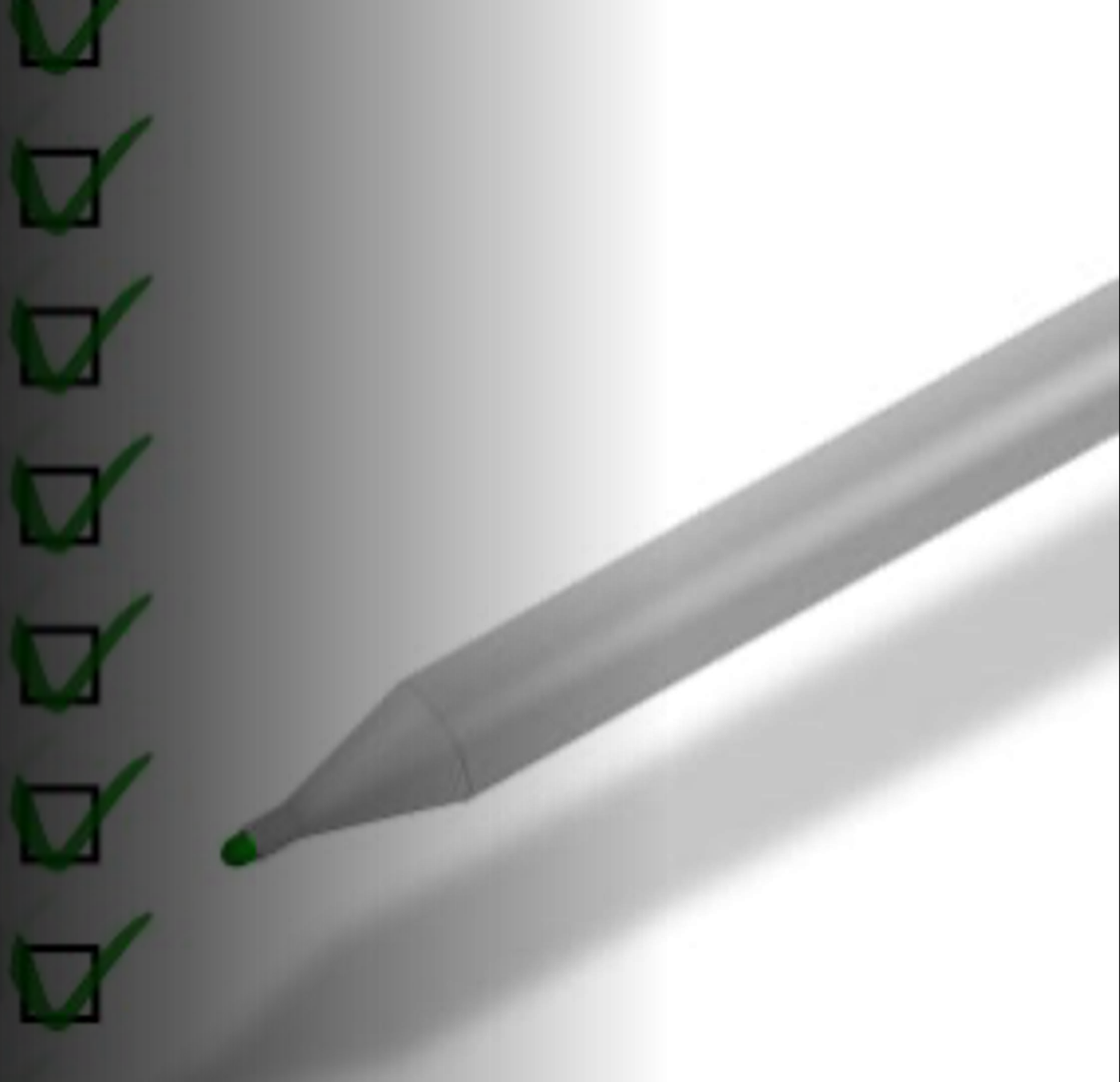
mapping characteristics

to requirements

&

referencing characteristics

Understanding of the requirements and completeness of solution



Original requirements are the keys.

NEEDS

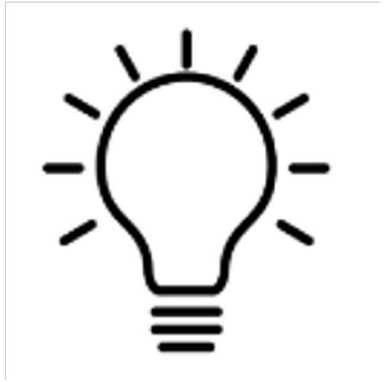
REQUIREMENTS

- Documented original requirements as is.
- Extracted real business needs.
- Read between the lines, write it down, repeat.

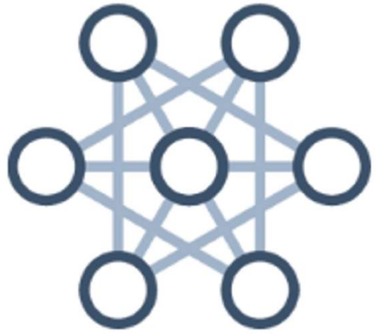
Capturing the fundamentals

- Narrative and architecture that captures and targets fundamental themes of the solution.
- A clear mapping and understanding of the original and deduced scope, goals and expectations to the solution.
- Clear capture of the key technical challenges and how your architecture addresses them.

Demonstrating a clear distillation of the problem



Identifies foundational assumptions & implications



Connects requirements to architecture characteristics

Outlines and targets fundamental functional and technical challenges

A detailed view of the moving parts

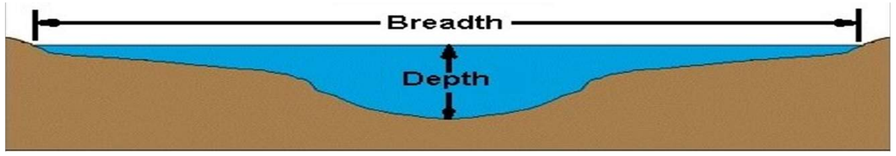
A clear illustration of how architecture solved the customer problem (addressed Business Need)

- Logical view of the architecture
- Sequence diagrams
- Flow of the end-to-end experience

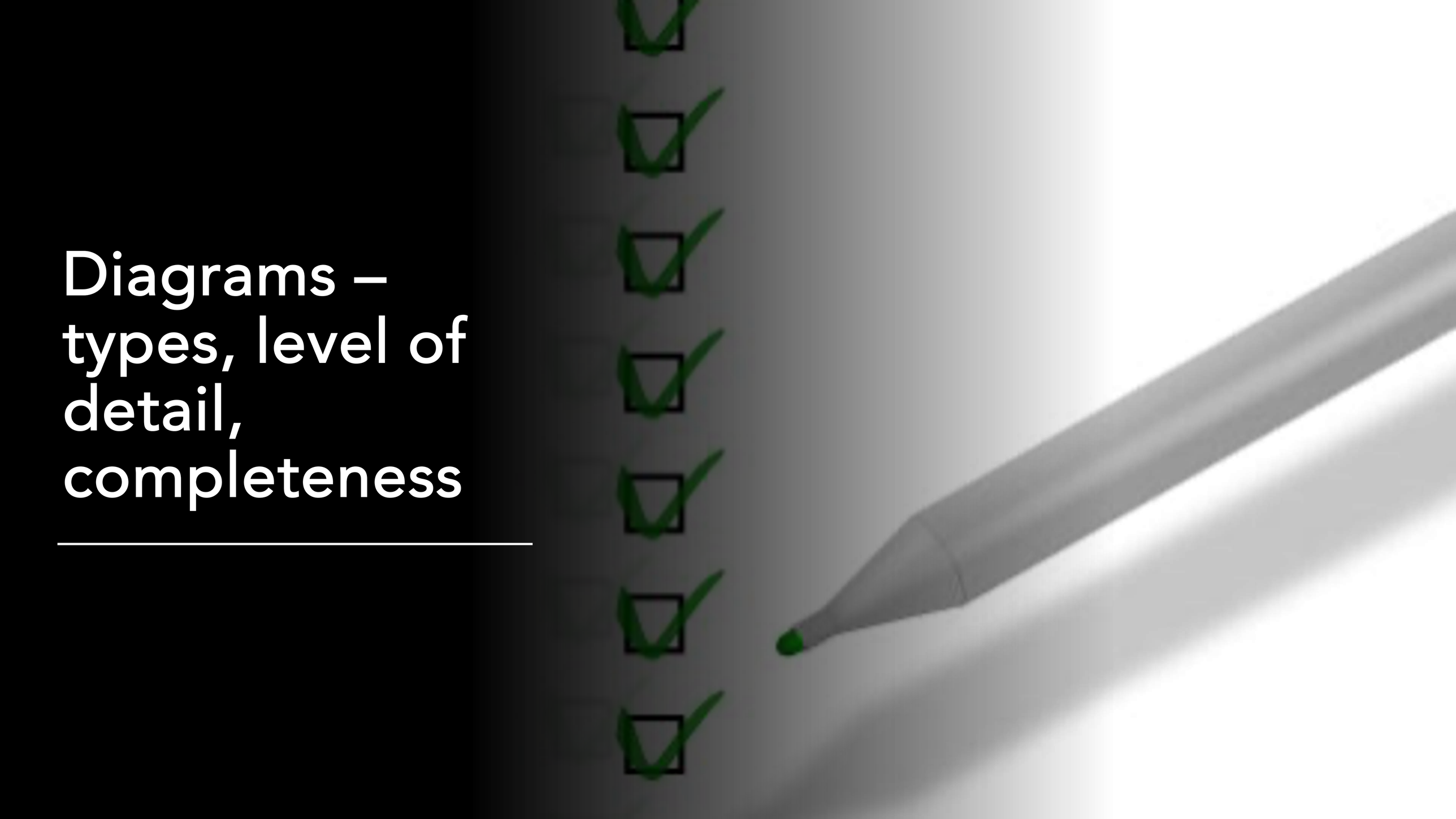
Highlights of the ***Why*** along with the ***What***

- Factors influenced your architecture choices
- Key drivers that shaped your strategy and approach
- Tension points that are addressed by your solution

Favor breadth over depth



Diagrams – types, level of detail, completeness



Judges Criteria

diagrams - types, level of detail, completeness

"The goal of a diagram is to convey a clear and shared understanding of the architecture"

- Neal Ford

Judges Criteria

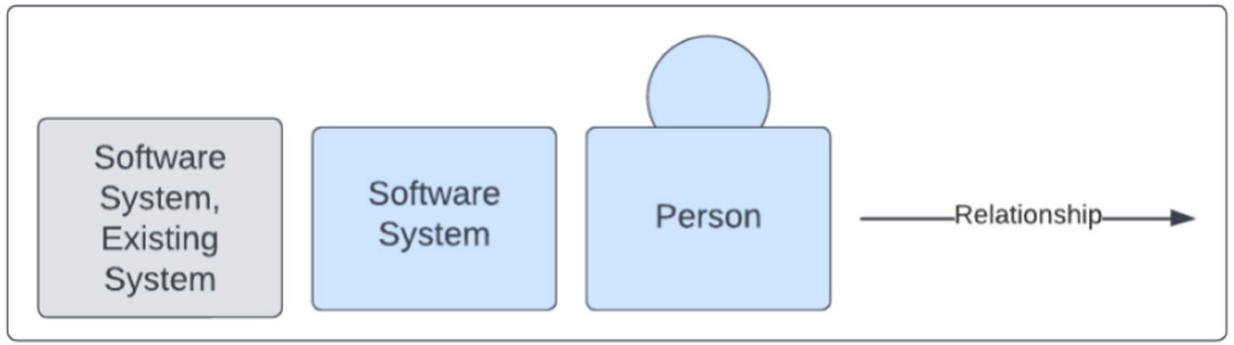
diagrams - types, level of detail, completeness

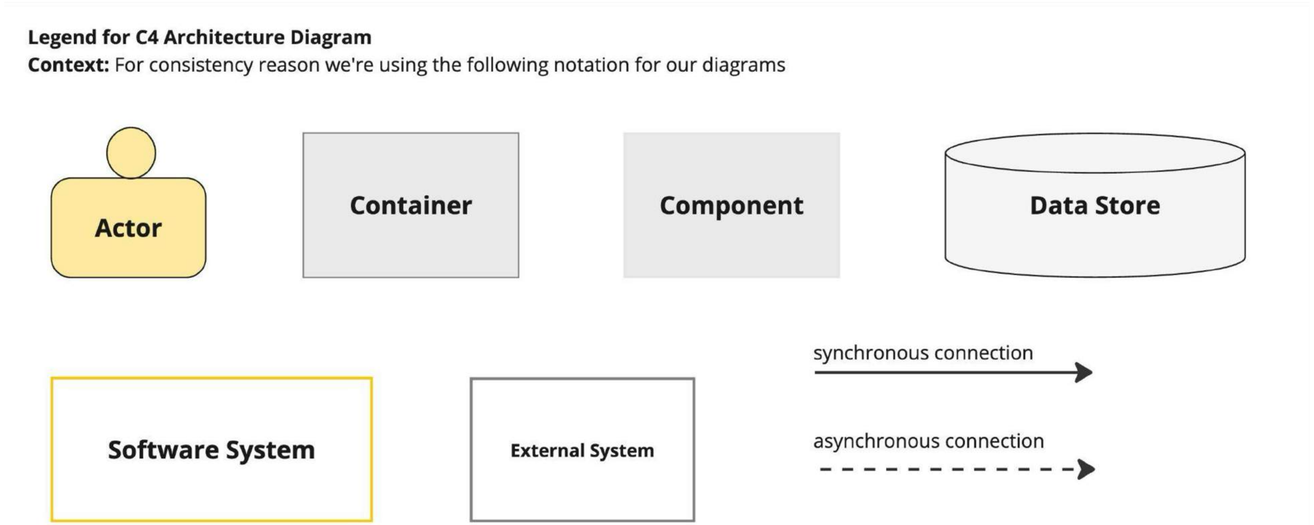
"Successful communication is the art and science of sharing or exchanging ideas and information,... resulting in **shared understanding**."

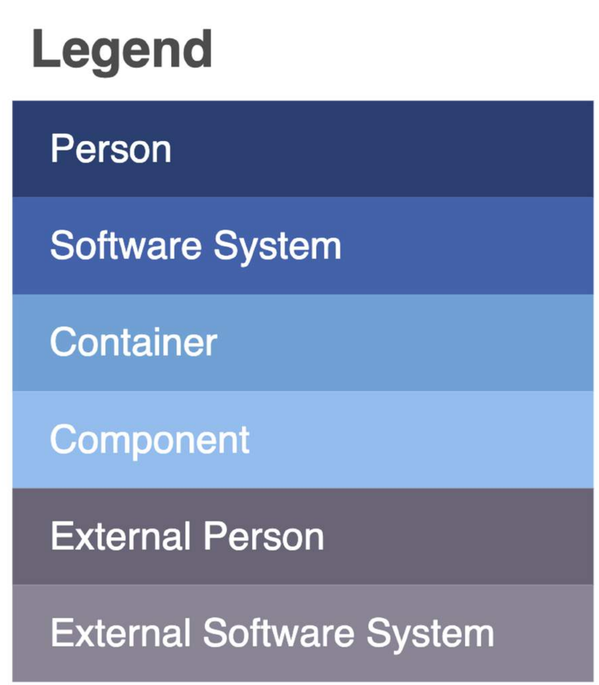
- Jacqui Read

Judges Criteria

diagrams - types, level of detail, completeness

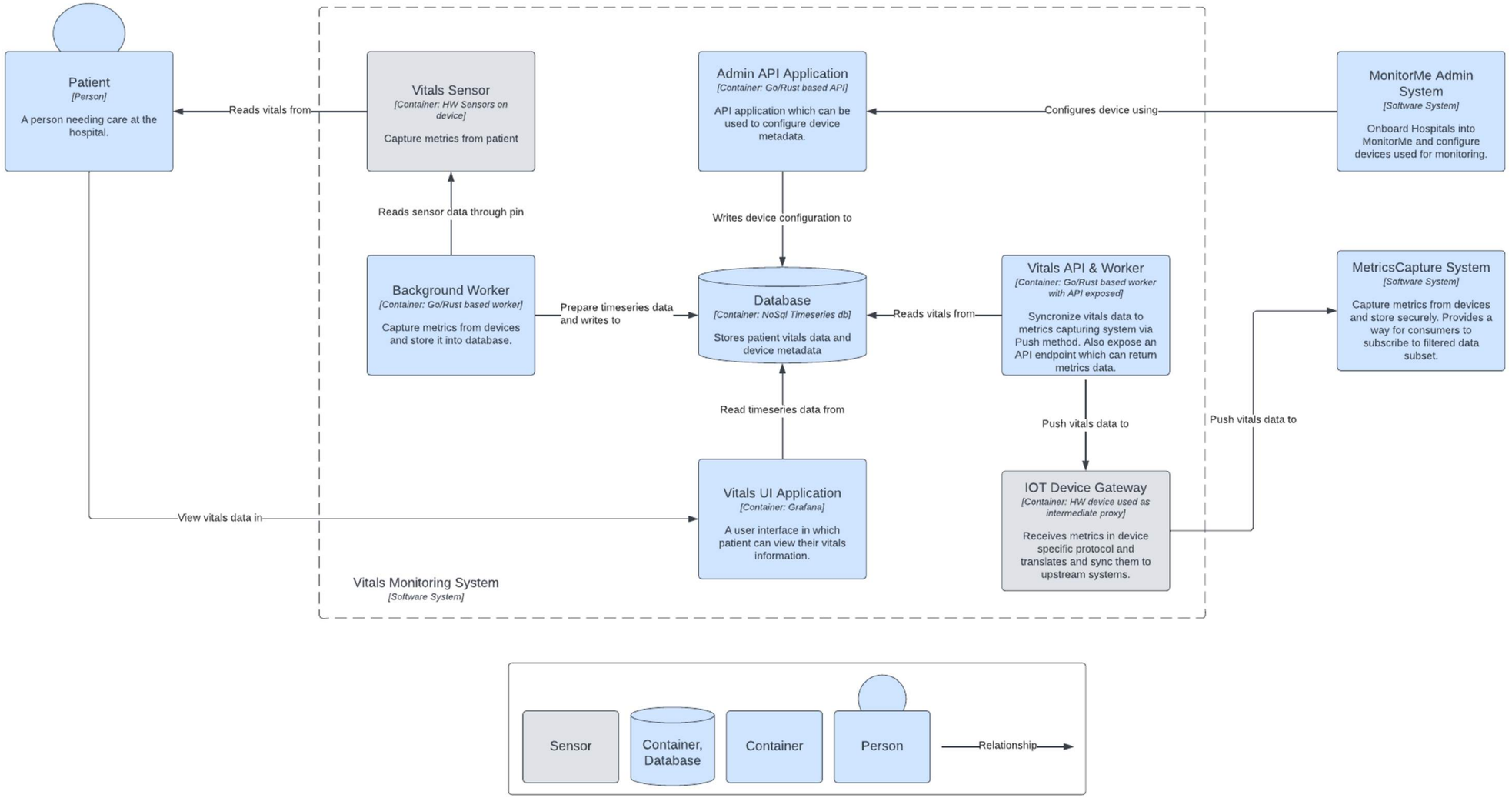






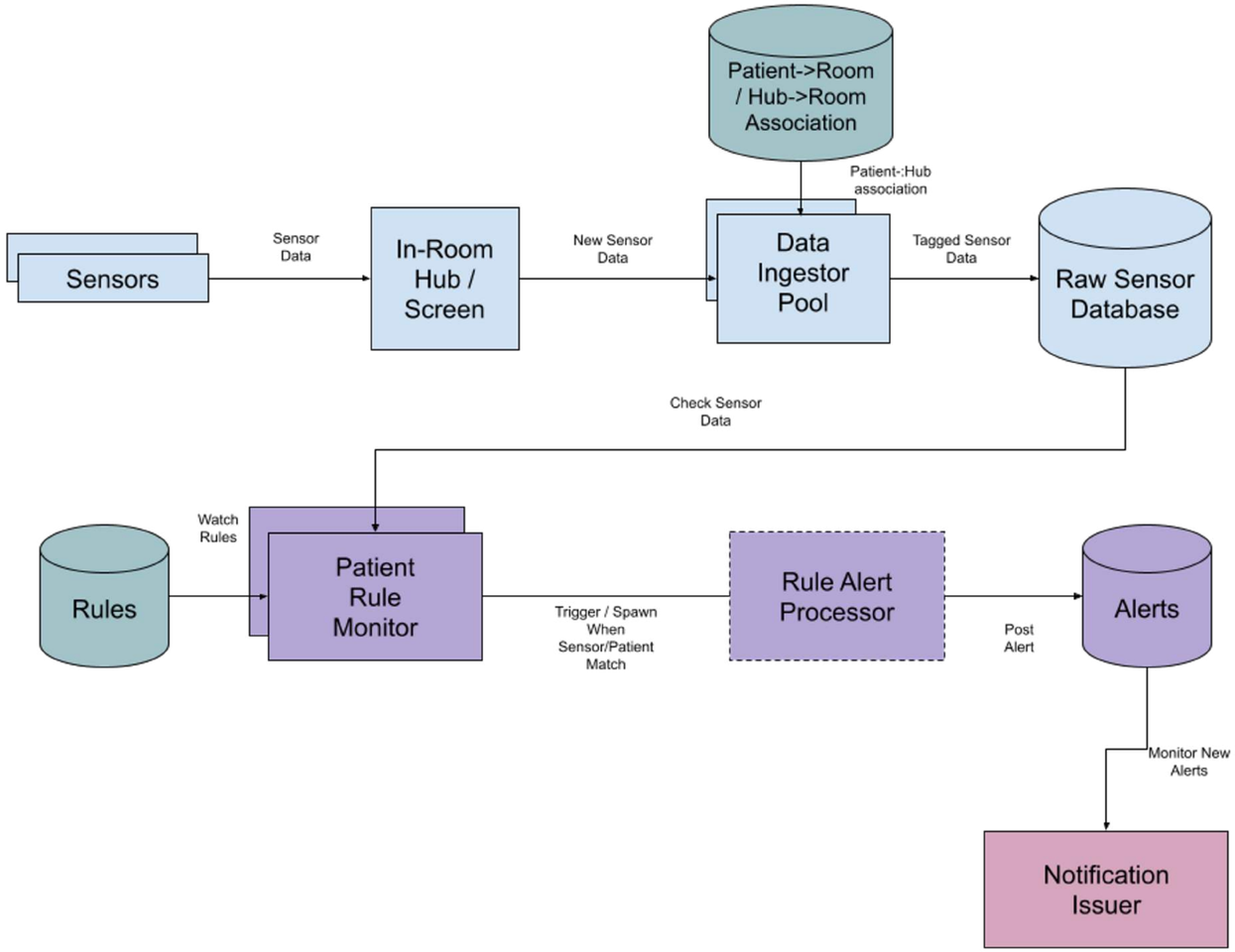
Judges Criteria

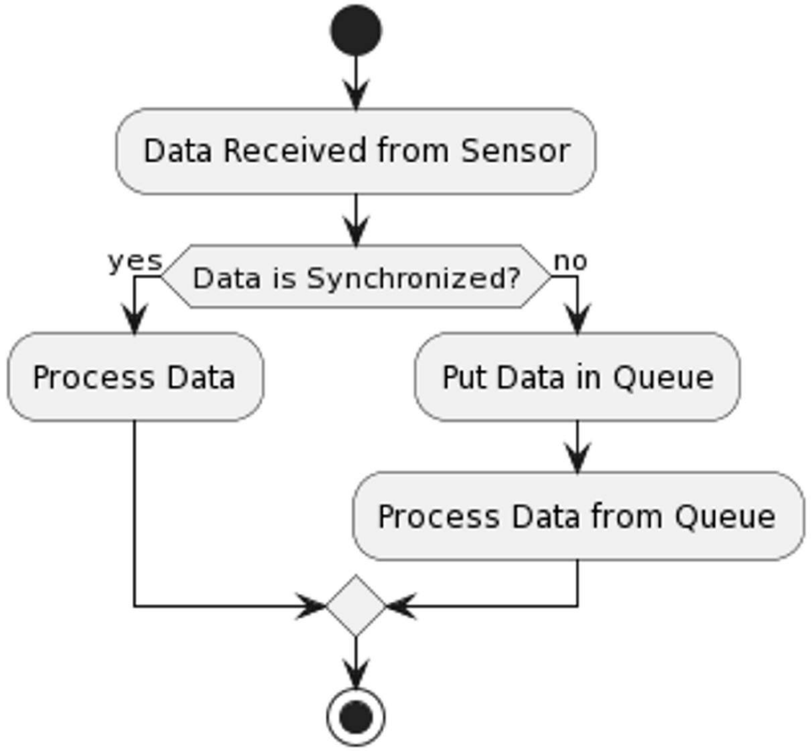
diagrams - types, level of detail, completeness



Judges Criteria

diagrams - types, level of detail, completeness





Judges Criteria

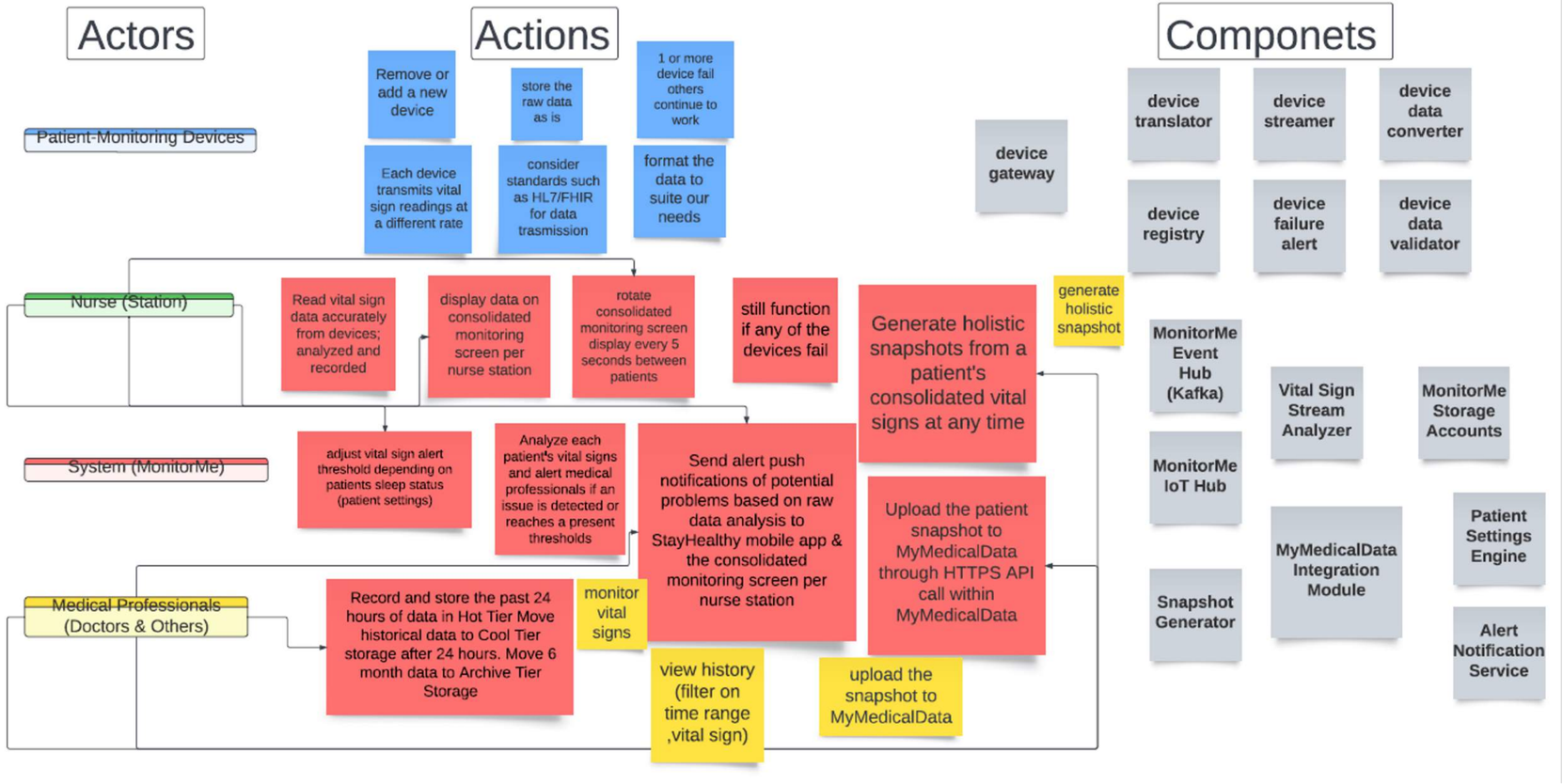
diagrams - types, level of detail, completeness

Medical professional journey - recieving alerts

Mary starts her shift at the hospital and logs into the Mobile MonitorMe app.

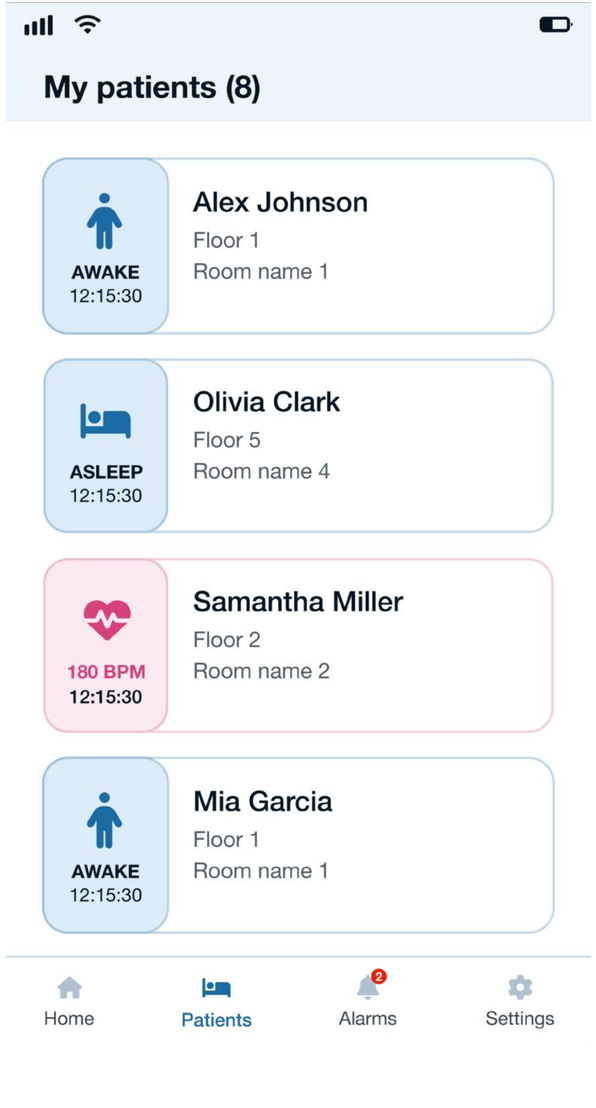
When a patients vital signs are out of the norm, MonitorMe sends an alert on the Consolidated Monitoring screen and the MonitorMe mobile app.

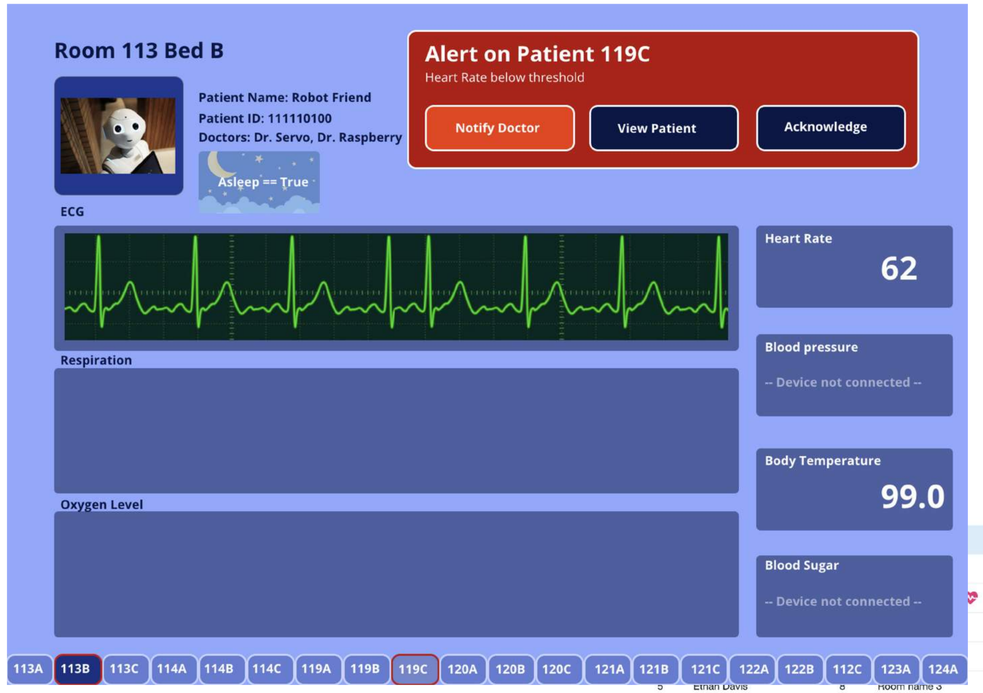
Marry will find the patient in his room and attend to him.



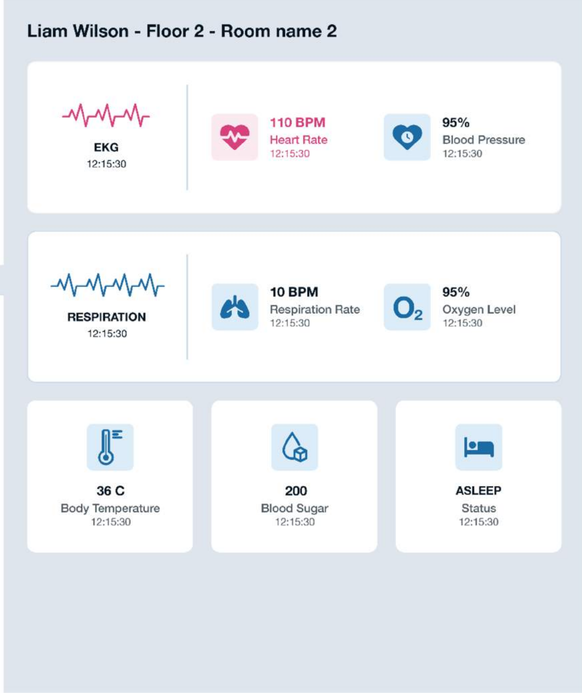
Judges Criteria

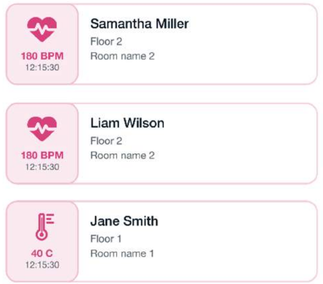
diagrams - types, level of detail, completeness





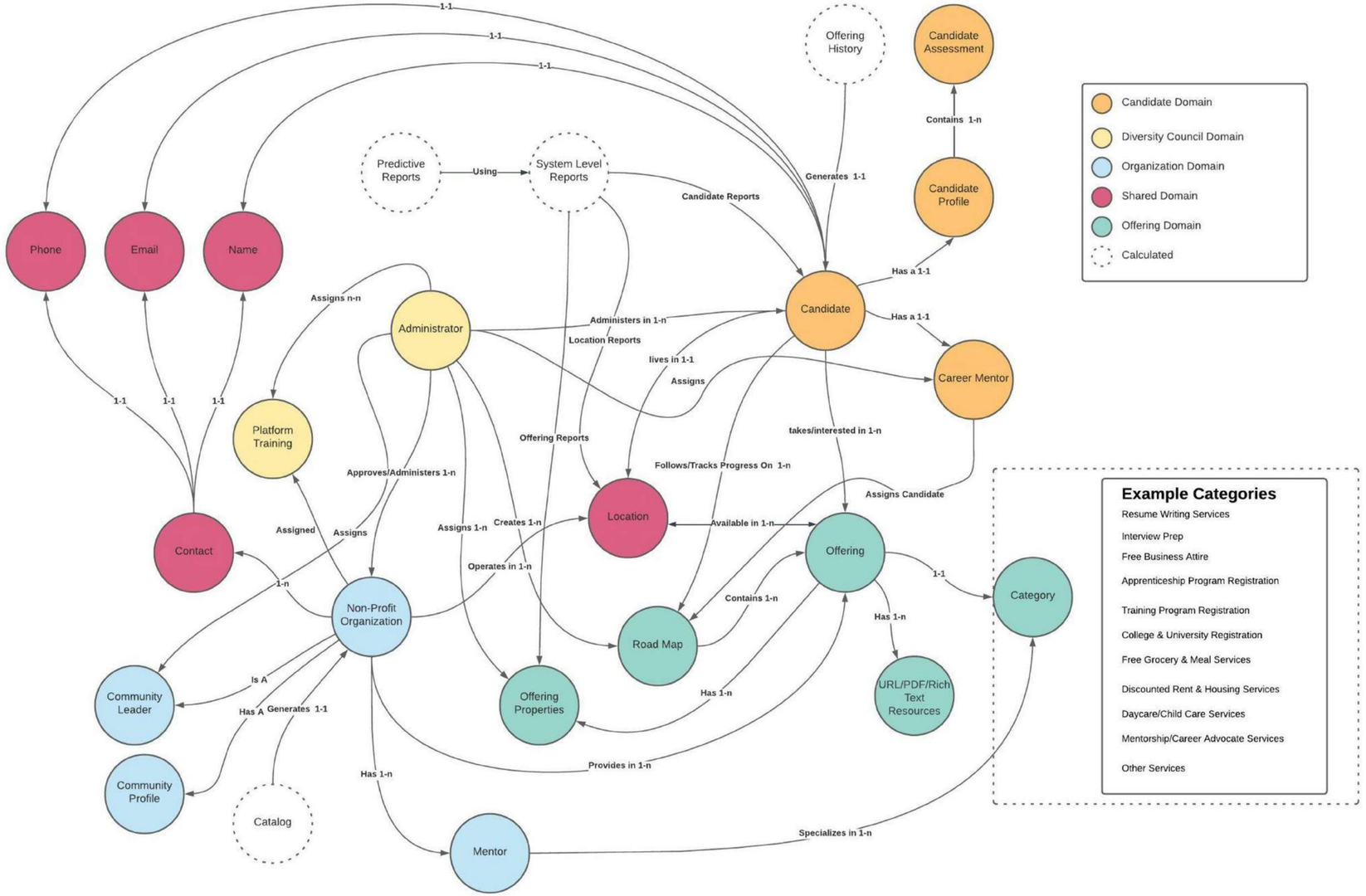
6	Ava Rodriguez	2	Room name 1
7	Liam Wilson	2	Room name 2
8	Mia Garcia	3	Room name 1
9	Noah Martinez	4	Room name 3
10	Emma Anderson	5	Room name 2
11	John Doe	3	Room name 1
12	Jane Smith	1	Room name 1
13	Michael Johnson	2	Room name 2
14	Emily Brown	1	Room name 3
15	David Wilson	6	Room name 2
16	Sarah Davis	4	Room name 2
17	Robert Taylor	9	Room name 1
18	Jennifer Anderson	7	Room name 3
19	Christopher Martinez	5	Room name 1
20	Amanda Thompson	1	Room name 3





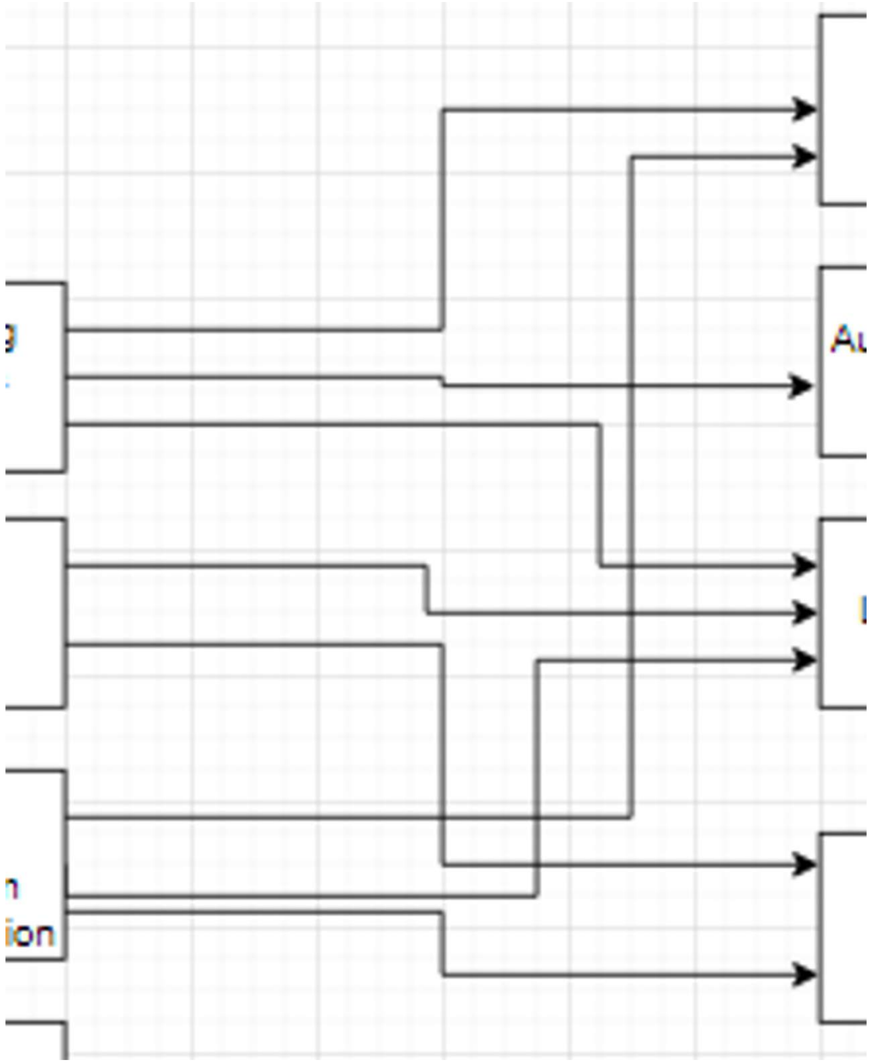
Judges Criteria

diagrams - types, level of detail, completeness



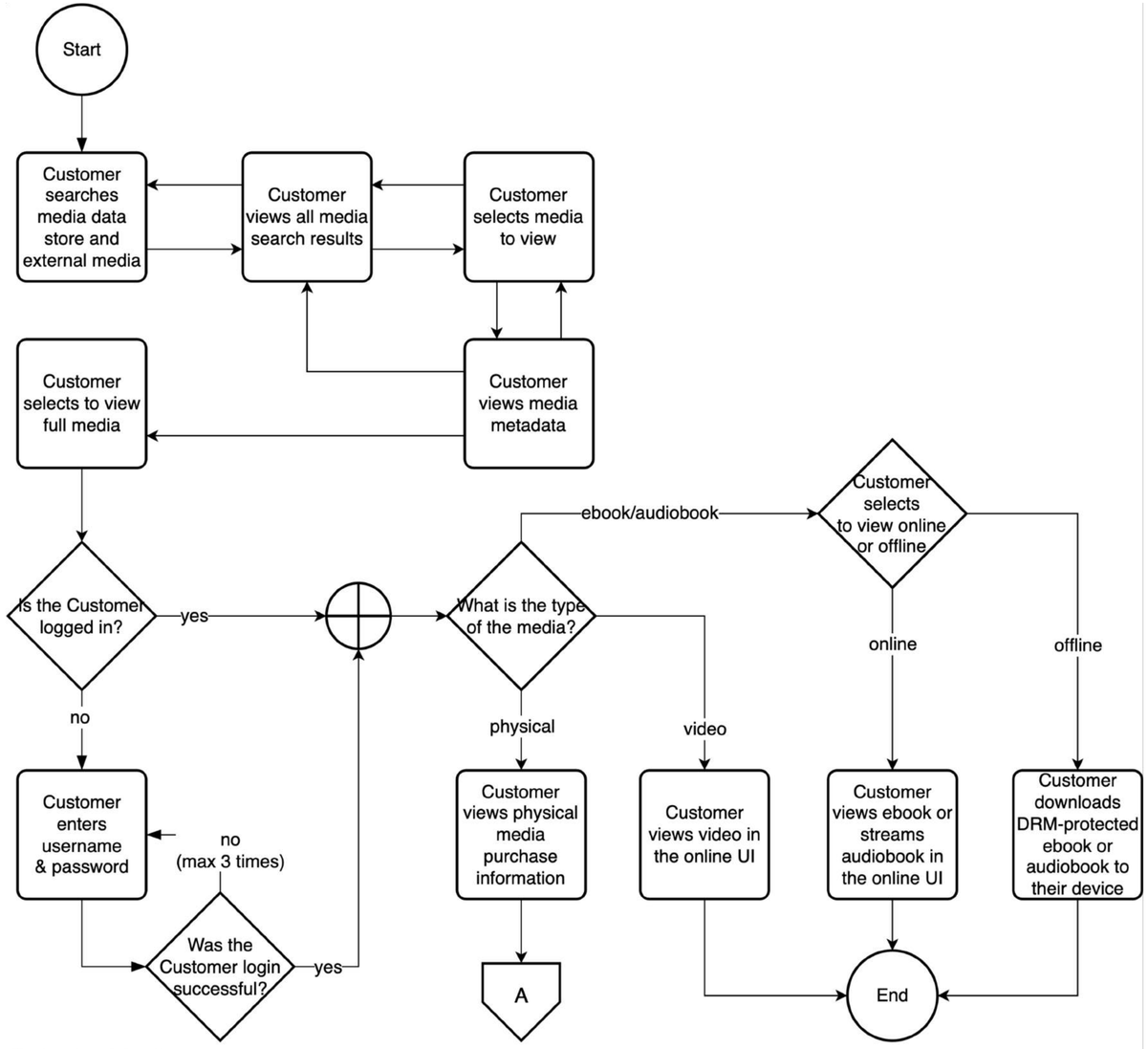
Judges Criteria

diagrams - types, level of detail, completeness



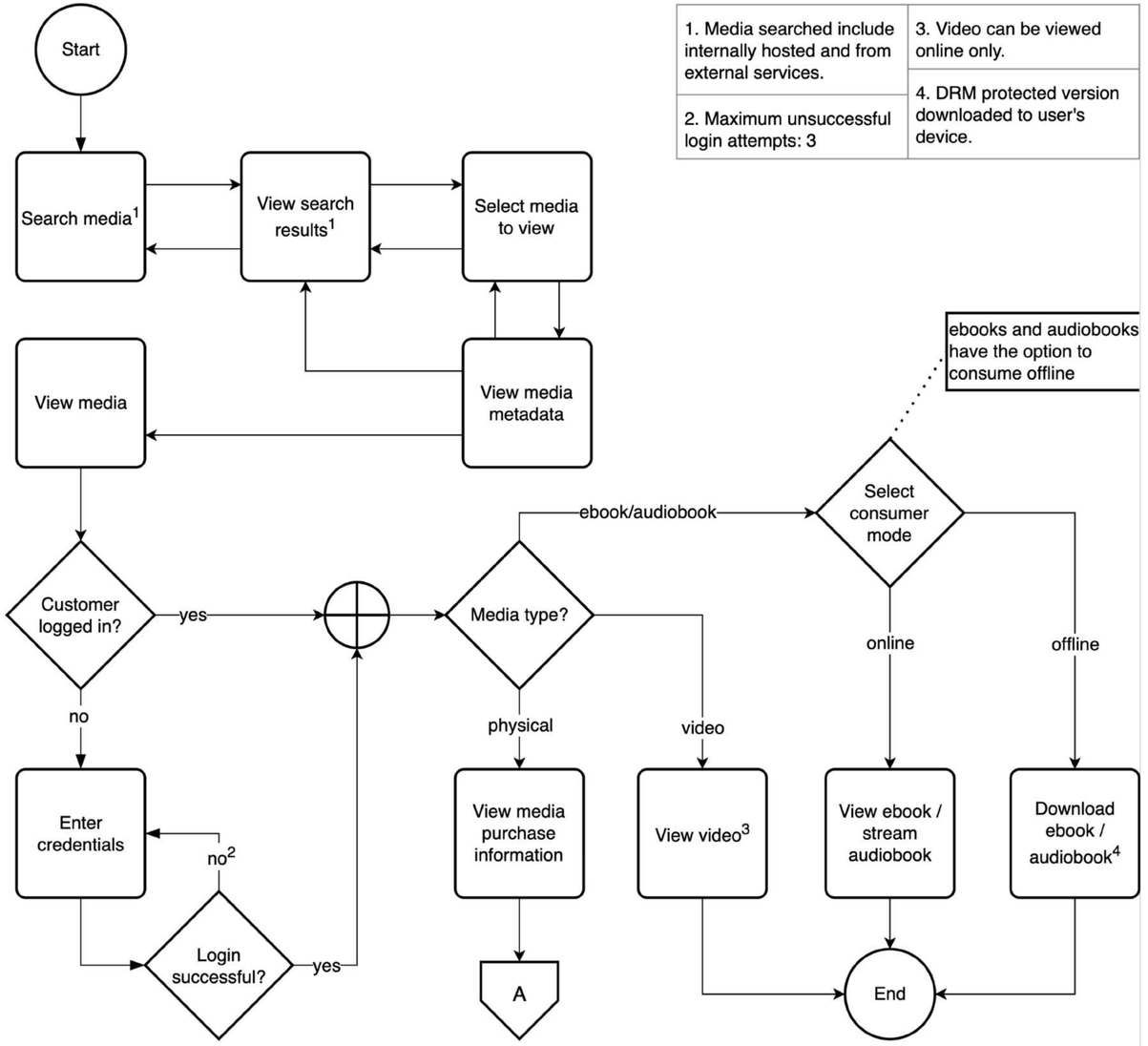
Judges Criteria

diagrams - types, level of detail, completeness



Judges Criteria

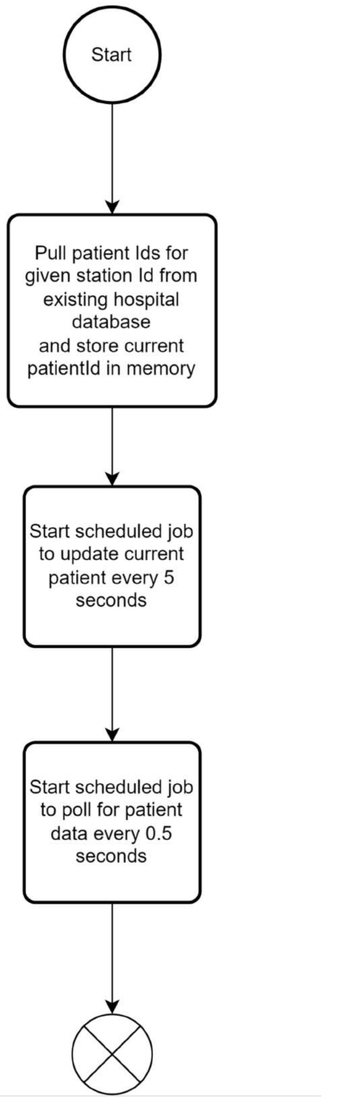
diagrams - types, level of detail, completeness



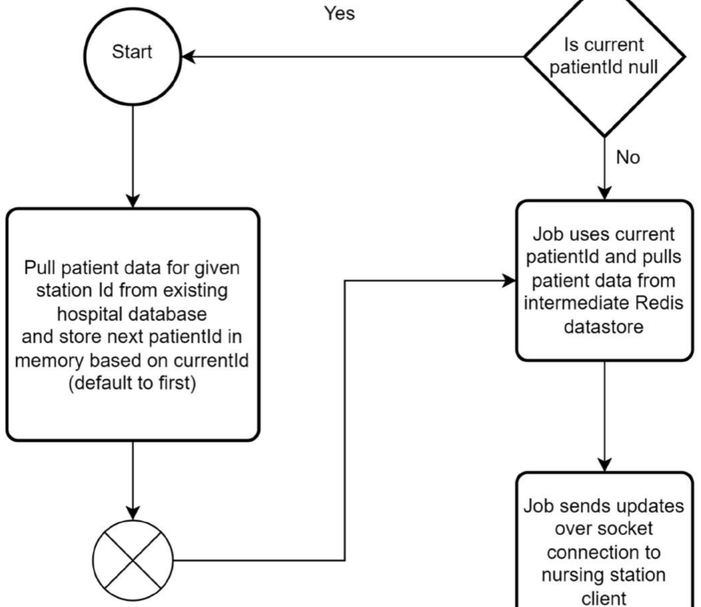
Judges Criteria

diagrams - types, level of detail, **completeness**

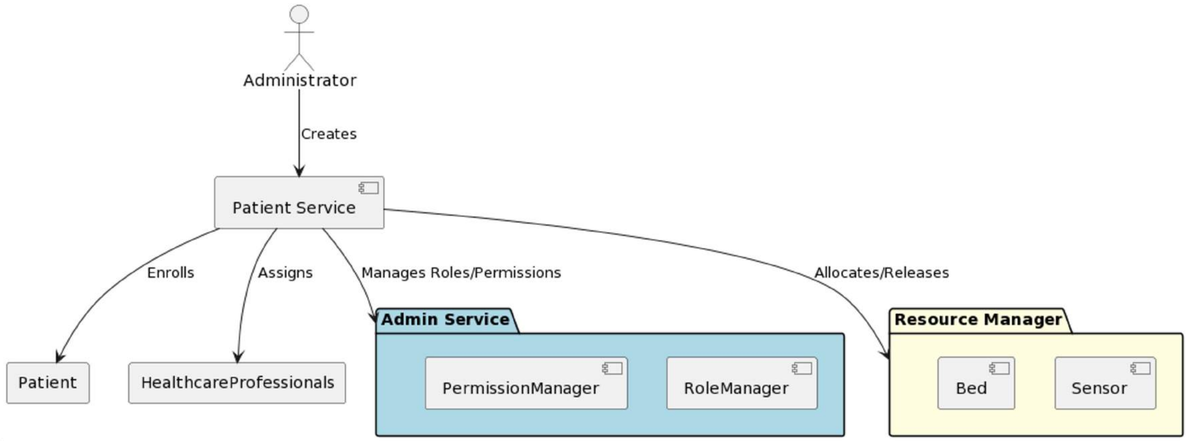
Streamer Logic Flowchart - Initialization

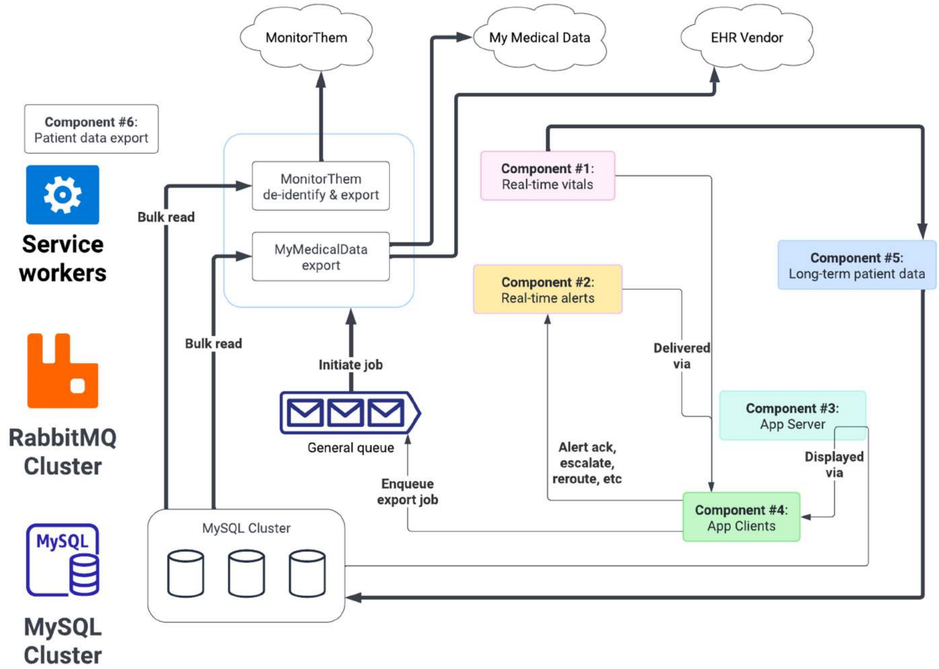


Streamer Logic Flowchart - Update patient



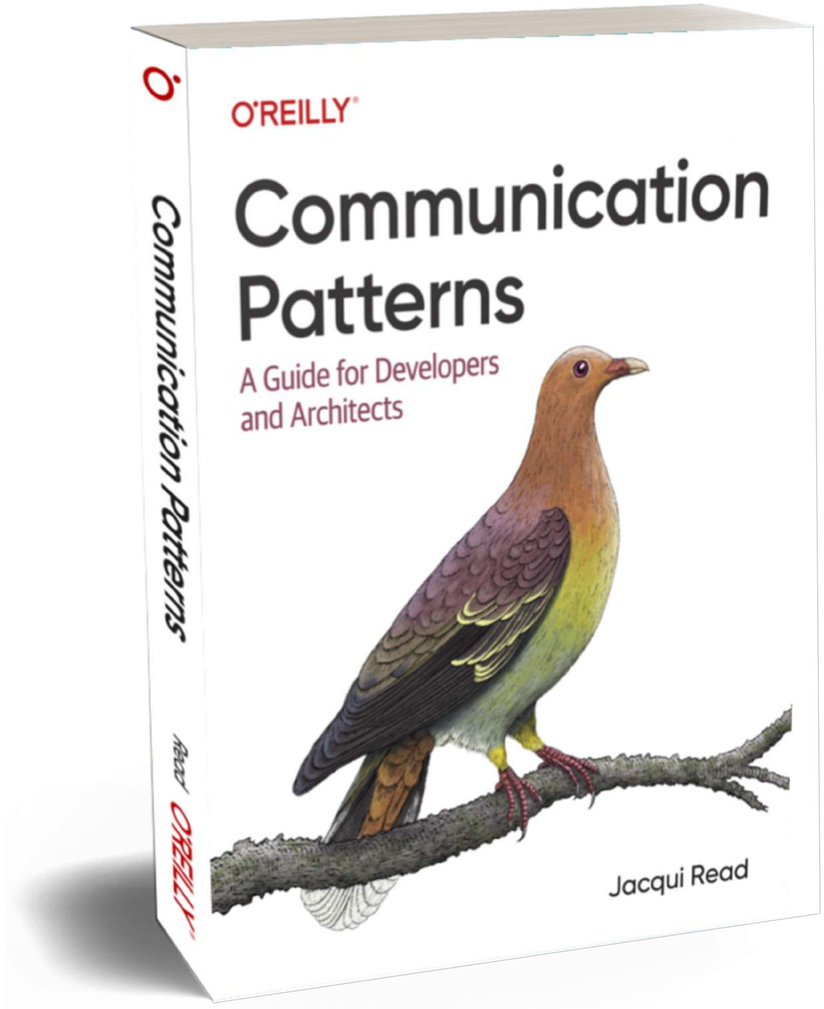
Streamer Logic Flowchart - Poll



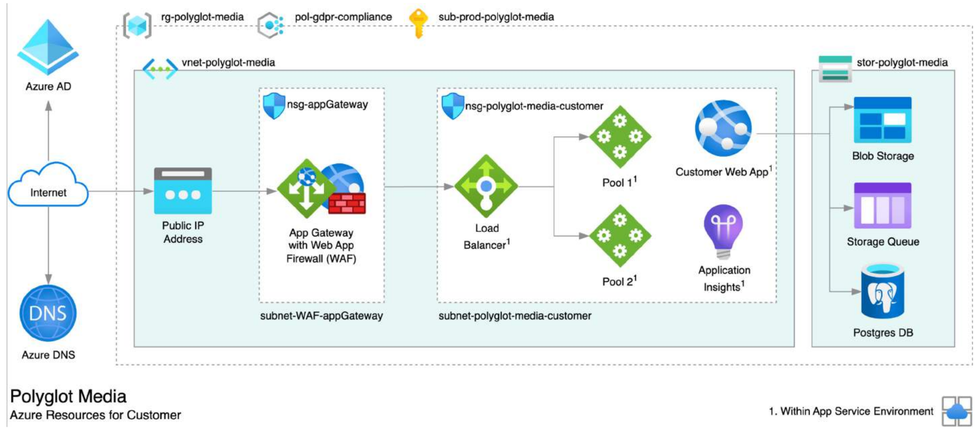


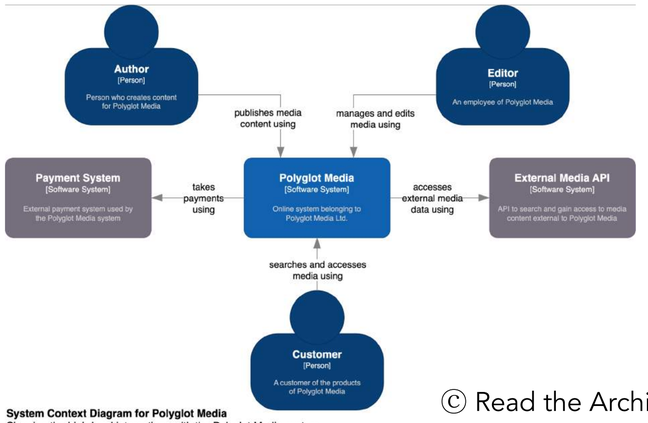
Judges Criteria

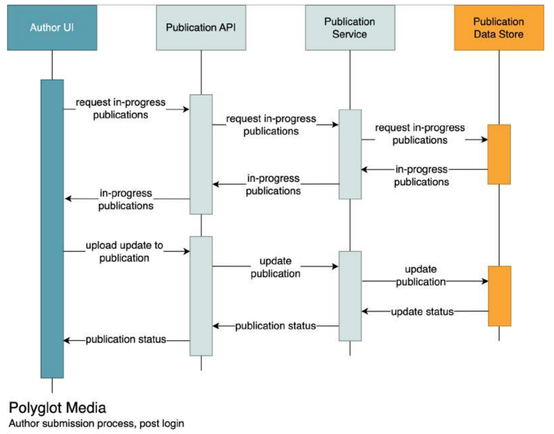
diagrams - types, level of detail, completeness



Part 1: Visual Communication



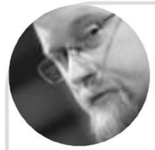




Architecture decision records – documentation and justification

Judges Criteria

architecture decision records - documentation and justification



DOCUMENTING ARCHITECTURE DECISIONS

Michael Nygard - November 15, 2011

AGILITY ARCHITECTURE

CONTEXT

Architecture for agile projects has to be described and defined differently. Not all decisions will be made at once, nor will all of them be done when the project begins.

Agile methods are not opposed to documentation, only to valueless documentation. Documents that assist the team itself can have value, but only if they are kept up to date. Large documents are never kept up to date. Small, modular documents have at least a chance at being updated.

Nobody ever reads large documents, either. Most developers have been on at least one project where the specification document was larger (in bytes) than the total source code size. Those documents are too large to open, read, or update. Bite sized pieces are easier for for all stakeholders to consume.

One of the hardest things to track during the life of a project is the motivation behind certain decisions. A new person coming on to a project may be perplexed, baffled, delighted, or infuriated by some past decision. Without understanding the rationale or consequences, this person has only two choices:

https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions

Judges Criteria

architecture decision records - documentation and justification

ADRs document

the act of deciding

not just

the decision and its consequences

Judges Criteria

architecture decision records - documentation and justification

ADRs document

the act of deciding

not just

the decision and its consequences

Judges Criteria

architecture decision records - documentation and justification

ADRs document

the act of deciding

not just

the decision and its consequences

ADR-044 Use an Event-Driven Distributed Architecture

Status

Decided, 2023-10-04
Supersedes [ADR-031 Use serverless functions]

Context

The Polyglot Media system is currently a distributed system consisting mostly of serverless functions. Moving from a monolith to a distributed serverless architecture was done to tackle problems with the responsiveness of the live system and a long lead time between functionality and bug fixes being coded and deployed to production.

Serverless architecture has not solved the problems with responsiveness or maintainability to the degree required, with functions being tightly coupled with many other functions.

A solution to the problems of responsiveness and time-to-market must be found.

Evaluation Criteria

See [ADR-002 Select Architecture Characteristics]

- *Responsiveness*: Customers have been complaining about the responsiveness of the system, and this must be addressed. This is deemed the most important criterion.
- *Maintainability*: The serverless functions improved the time-to-market of bug fixes and new functionality but not to the extent needed.
- *Deployability*: Alongside maintainability, deployability is important to improve time-to-market for bug fixes and new functionality.
- *Scalability*: Most complaints from customers about responsiveness have been around peak use times, so the system must handle peak numbers of users without impacting users.

Options

1. Microservices

Criteria	Score	Rationale
Responsiveness	★★★☆☆ 3/5	[Not inherently performant] but optimizations can be put in place at bottlenecks, e.g. scaling
Maintainability	★★★☆☆ 3/5	Dependencies can be an issue, many data stores to maintain
Deployability	★★★★★ 5/5	Deploy only what has changed
Scalability	★★★★★ 5/5	Scale on per-service basis
	Total: 16/20	**Other Trade-offs** - Must split data into [one data store per service] - Costs are usually high for building - Complex and is hard to create workflows

2. Service-based

Criteria	Score	Rationale
Responsiveness	★★★☆☆ 3/5	[Not inherently performant] but optimizations can be put in place at bottlenecks, e.g. scaling
Maintainability	★★★★☆ 4/5	Fewer data stores to maintain than microservices
Deployability	★★★★☆ 4/5	Deploy only what has changed, make sure changes to shared data store(s) don't affect other services
Scalability	★★★☆☆ 3/5	Scale individual services, harder to scale shared data store(s) unless using [caching]
	Total: 14/20	**Other Trade-offs** - Reasonably complex & hard to create workflows - Not as evolvable as microservices

3. Event-driven

Criteria	Score	Rationale
Responsiveness	★★★★★ 5/5	Generally [fire-and-forget], event processing can be scaled/optimized
Maintainability	★★★★☆ 4/5	Must manage event processing as well as services and data stores, events decouple services to be managed by individual teams
Deployability	★★★☆☆ 3/5	Manage event processing deployment and changes as well as services
Scalability	★★★★★ 5/5	Services and message processing can both be scaled
	Total: 17/20	**Other Trade-offs** - [Integration testing can be harder]

Decision

We will use event-driven architecture, based on the total score against the decision criteria and the fact that options 1 and 2 did not score well on the most important criterion—responsiveness.

Implications

Positive

- We gain overall higher responsiveness, maintainability, and scalability than the current serverless functions.
- Processing of events is scalable by spinning up additional instances of services to process queued events.
- Processing within services is scalable by spinning up additional instances of services to handle the need for processing within the services.

Negative

- Teams or individuals may need to learn new skills or technologies (such as event queues).
- Event management (such as queues) adds extra complexity to development and deployment.
- Integration testing and DevOps will need to be overhauled.

Consultation

- *Vlad*: Neither microservices nor service-based are [inherently performant], so likely will not boost responsiveness as much as we want.
- *Nikki*: I have experience in event-driven architecture, using queues. We monitored the length of queues and spun up extra instances of services to process long queues when needed.
- *Libby*: Queues and channels can be used to prioritize the processing of events, for example with the [ambulance pattern].
- *Mark*: Responsiveness needs to be the top priority due to customer complaints.
- *Libby*: Pure microservices would require that services do not share data stores. Splitting our data like this would require a lot of effort.

ADR-044 Use an Event-Driven Distributed Architecture

Status

Decided, 2023-10-04
Supersedes [ADR-031 Use serverless functions]

Context

The Polyglot Media system is currently a distributed system consisting mostly of serverless functions. Moving from a monolith to a distributed serverless architecture was done to tackle problems with the responsiveness of the live system and a long lead time between functionality and bug fixes being coded and deployed to production.

Serverless architecture has not solved the problems with responsiveness or maintainability to the degree required, with functions being tightly coupled with many other functions.

A solution to the problems of responsiveness and time-to-market must be found.

Evaluation Criteria

See [ADR-002 Select Architecture Characteristics]

- *Responsiveness*: Customers have been complaining about the responsiveness of the system, and this must be addressed. This is deemed the most important criterion.
- *Maintainability*: The serverless functions improved the time-to-market of bug fixes and new functionality but not to the extent needed.
- *Deployability*: Alongside maintainability, deployability is important to improve time-to-market for bug fixes and new functionality.
- *Scalability*: Most complaints from customers about responsiveness have been around peak use times, so the system must handle peak numbers of users without impacting users.

Options

1. Microservices

Criteria	Score	Rationale
Responsiveness	★★★☆☆ 3/5	[Not inherently performant] but optimizations can be put in place at bottlenecks, e.g. scaling
Maintainability	★★★☆☆ 3/5	Dependencies can be an issue, many data stores to maintain
Deployability	★★★★★ 5/5	Deploy only what has changed
Scalability	★★★★★ 5/5	Scale on per-service basis
	Total: 16/20	**Other Trade-offs** - Must split data into [one data store per service] - Costs are usually high for building - Complex and is hard to create workflows

---

2. Service-based

Criteria	Score	Rationale
Responsiveness	★★★☆☆ 3/5	[Not inherently performant] but optimizations can be put in place at bottlenecks, e.g. scaling
Maintainability	★★★★☆ 4/5	Fewer data stores to maintain than microservices
Deployability	★★★★☆ 4/5	Deploy only what has changed, make sure changes to shared data store(s) don't affect other services

Options

1. Microservices

Criteria	Score	Rationale
Responsiveness	★★★☆☆ 3/5	[Not inherently performant] can be put in place at bottlenecks, e.g. scaling
Maintainability	★★★☆☆ 3/5	Dependencies can be an issue, many data stores to maintain
Deployability	★★★★★ 5/5	Deploy only what has changed
Scalability	★★★★★ 5/5	Scale on per-service basis
	Total: 16/20	**Other Trade-offs** - Must split data into [one data store per service] - Costs are usually high for building - Complex and is hard to create workflows

2. Service-based

Criteria	Score	Rationale
Responsiveness	★★★☆☆ 3/5	[Not inherently performant] can be put in place at bottlenecks, e.g. scaling
Maintainability	★★★★☆ 4/5	Fewer data stores to maintain than microservices
Deployability	★★★★☆ 4/5	Deploy only what has changed, make sure changes to shared data store(s) don't affect other services

Decision

We will use event-driven architecture, based on the total score against the decision criteria and the fact that options 1 and 2 did not score well on the most important criterion—responsiveness.

Implications

Positive

- We gain overall higher responsiveness, maintainability, and scalability than the current serverless functions.
- Processing of events is scalable by spinning up additional instances of services to process queued events.
- Processing within services is scalable by spinning up additional instances of services to handle the need for processing within the services.

Negative

- Teams or individuals may need to learn new skills or technologies (such as event queues).
- Event management (such as queues) adds extra complexity to development and deployment.
- Integration testing and DevOps will need to be overhauled.

Consultation

- *Vlad*: Neither microservices nor service-based are [inherently performant], so likely will not boost responsiveness as much as we want.
- *Nikki*: I have experience in event-driven architecture, using queues. We monitored the length of queues and spun up extra instances of services to process long queues when needed.
- *Libby*: Queues and channels can be used to prioritize the processing of events, for example with the [ambulance pattern].
- *Mark*: Responsiveness needs to be the top priority due to customer complaints.
- *Libby*: Pure microservices would require that services do not share data stores. Splitting our data like this would require a lot of effort.

ADR-044 Change to Event-Driven Architecture

Status

Decided, 2023-10-04
Supersedes [ADR-031 Use serverless functions]

Decision

We will use Event-Driven architecture, based on its high scalability and responsiveness. Microservices and service-based don't support the most important criterion—responsiveness.

Context

The Polyglot Media system is currently a distributed system consisting mostly of serverless functions. Moving from a monolith to a distributed serverless architecture was done to tackle problems with the responsiveness of the live system and a long lead time between functionality and bug fixes being coded and deployed to production.

Serverless architecture has not solved the problems with responsiveness or maintainability to the degree required, with functions being tightly coupled with many other functions.

A solution to the problems of responsiveness and time-to-market must be found.

Implications

Positive

- We gain overall higher responsiveness, maintainability, and scalability than the current serverless functions.
- Processing of events is scalable by spinning up additional instances of services to process queued events.
- Processing within services is scalable by spinning up additional instances of services to handle the need for processing within the services.

Negative

- Teams or individuals may need to learn new skills or technologies (such as event queues).
- Event management (such as queues) adds extra complexity to development and deployment.
- Integration testing and DevOps will need to be overhauled.

ADR-044 Change to Event-Driven Architecture

Status

Decided, 2023-10-04
Supersedes [ADR-031 Use serverless functions]

Decision

We will use Event-Driven architecture, based on its high scalability and responsiveness. Microservices and service-based don't support the most important criterion—responsiveness.

Context

The Polyglot Media system is currently a
serverless functions. Moving from a monol
architecture was done to tackle problems with the responsiveness of the live system and a long lead time between functionality and bug fixes being coded and deployed to production.

Serverless architecture has not solved the problems with responsiveness or maintainability to the degree required, with functions being tightly coupled with many other functions.

A solution to the problems of responsiveness and time-to-market must be found.

Implications

Positive

- We gain overall higher responsiveness, maintainability, and scalability than the current serverless functions.
- Processing of events is scalable by spinning up additional instances of services to process queued events.
- Processing within services is scalable by spinning up additional instances of services to handle the need for processing within the services.

Negative

- Teams or individuals may need to learn new skills or technologies (such as event queues).
- Event management (such as queues) adds extra complexity to development and deployment.
- Integration testing and DevOps will need to be overhauled.

Judges Criteria

architecture decision records - documentation and justification

- AWS SNS for sending mobile push not...
- Choosing Grafana Embedded in SPA f...
- Deploy Admin System in cloud.md
- Gitops approach for deployment.md
- Polling Mechanism for Monitoring Das...
- Rancher for monitoring K3s clusters o...
- Serving SPA from container.md
- Use K3s as orchestrator platform with...
- Websocket for Alerts.md

Judges Criteria

architecture decision records - documentation and justification

Monolith/Microservices

Context:

The current design of the system consists of six microservices that allow for scaling, maintenance, and independent development:

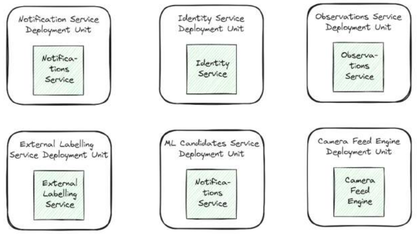
- Notifications Service
- Observations Service
- Identity Service
- External Labeling Service
- ML Candidates Service
- Camera Feed Engine

Since we have a small number (hundreds) of users, a microservices architecture may not be fully justified.

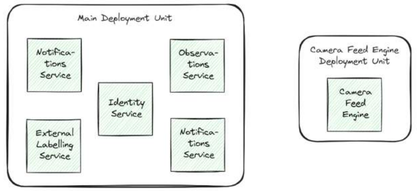
Decision:

We have decided to integrate five of our six microservices (excluding the Camera Feed Engine) into a cohesive monolithic setup while keeping the ability to communicate through messaging. If one of the services needs to scale individually, it can be easily done since its component in the monolith can be easily extracted into its own microservice.

The Camera Feed Engine will remain a separate and critical component of our architecture. It will run independently and will be optimized for dynamic scaling based on factors such as the number of cameras available and message delivery rates.



Microservices setup



Monolithic setup

The Camera Feed Engine represents a critical service within our architecture. It is designed to operate independently and is optimized for dynamic scaling based on various factors, such as the number of cameras available and the rates of messages delivered from cameras. This design ensures that this service can efficiently handle varying workloads while remaining separate from the monolithic setup.

Rationale:

This decision is made based on requirements, which state hundreds of users. By integrating the majority of our microservices into a monolithic setup, we can reduce the complexity associated with microservices communication and maintain specialized scalability for the Camera Feed Engine.

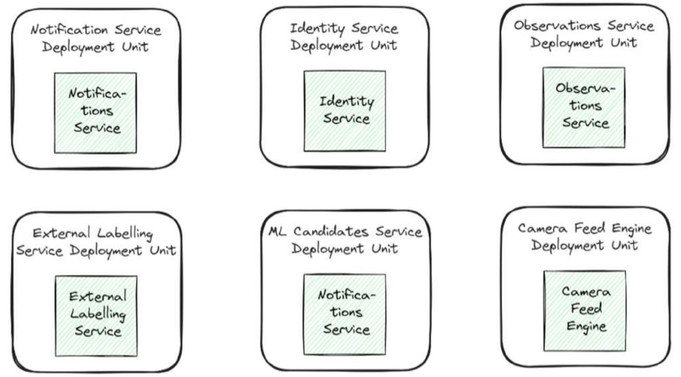
Status:

Approved

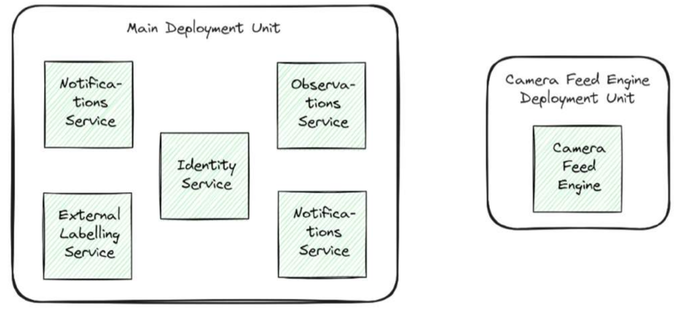
Decision:

We have decided to integrate five of our six microservices (excluding the Camera Feed Engine) into a cohesive monolithic setup while keeping the ability to communicate through messaging. If one of the services needs to scale individually, it can be easily done since its component in the monolith can be easily extracted into its own microservice.

The Camera Feed Engine will remain a separate and critical component of our architecture. It will run independently and will be optimized for dynamic scaling based on factors such as the number of cameras available and message delivery rates.



Microservices setup



Monolithic setup

The Camera Feed Engine represents a critical service within our architecture. It is designed to operate independently and is optimized for dynamic scaling based on various factors, such as the number of cameras available and the rates of messages delivered from cameras. This design ensures that this service can efficiently handle varying workloads while remaining separate from the monolithic setup.

Judges Criteria

architecture decision records - documentation and justification

Security as a Service ADR

Status

Accepted

Context

The system requires integration with various components type and less privilege access is required.

Evaluation Criteria

- Compatibility with different types of software components.
- Scalability to accommodate future additions of monitoring devices.
- Ensure data encryption and integrity at transit.

Options

- Implement direct authentication with each component.
- Utilize a CAS solution for standardized security integration.
- Use a service based security implementing standard authorization flows and access control protocols.

Decision

Use a service based security implementing standard authorization flows and access control protocols to streamline the integration process and ensure secururity and auditability with every software component (External device, mobile application, Api, ...).

Implications

Positive

- Simplifies integration process with various components.
- Provides a standardized interface for security.
- Enhances scalability and flexibility for future component additions.
- Reduces development effort for individual security adapters.

Negative

- Initial domain configuration and definition of centralized the scopes and roles.
- Additional training may be needed for maintenance personnel.

Options

- Implement direct authentication with each component.
- Utilize a CAS solution for standardized security integration.
- Use a service based security implementing standard authorization flows and access control protocols.

Judges Criteria

architecture decision records - documentation and justification

What was missing?

Judges Criteria

architecture decision records - documentation and justification

What was missing?

options considered,

Judges Criteria

architecture decision records - documentation and justification

What was missing?

options considered,

diagrams,

Judges Criteria

architecture decision records - documentation and justification

What was missing?

options considered,

diagrams,

links out and in

Judges Criteria

architecture decision records - documentation and justification

What was missing?

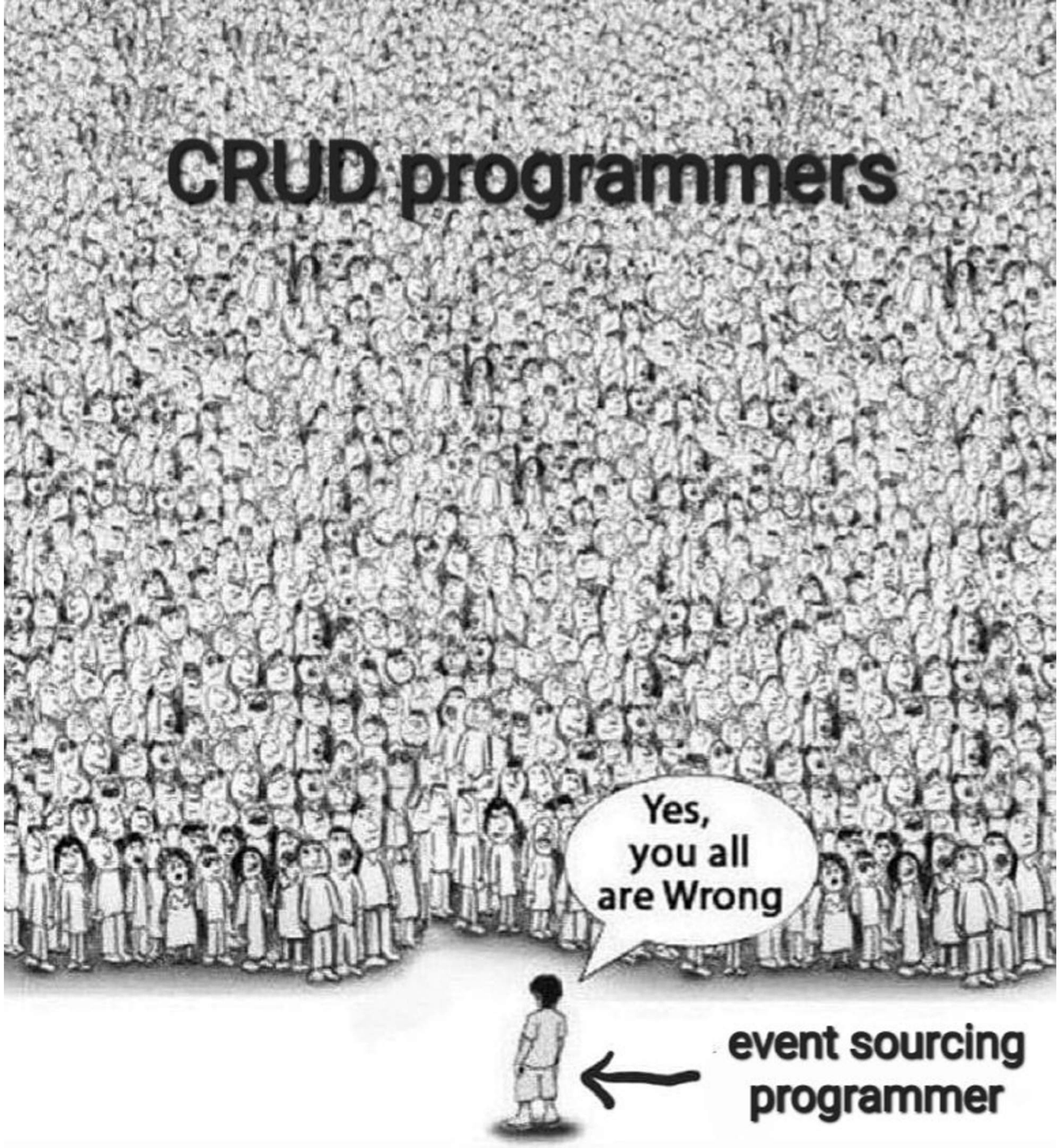
options considered,

diagrams,

links out and in

and ADRs on the broader, non-tech-choice decisions

Overall systems architecture


CRUD programmers
Yes,
you all
are Wrong
event sourcing
programmer

Judges Criteria

overall systems architecture

Data in Motion

Honorable Mentions…

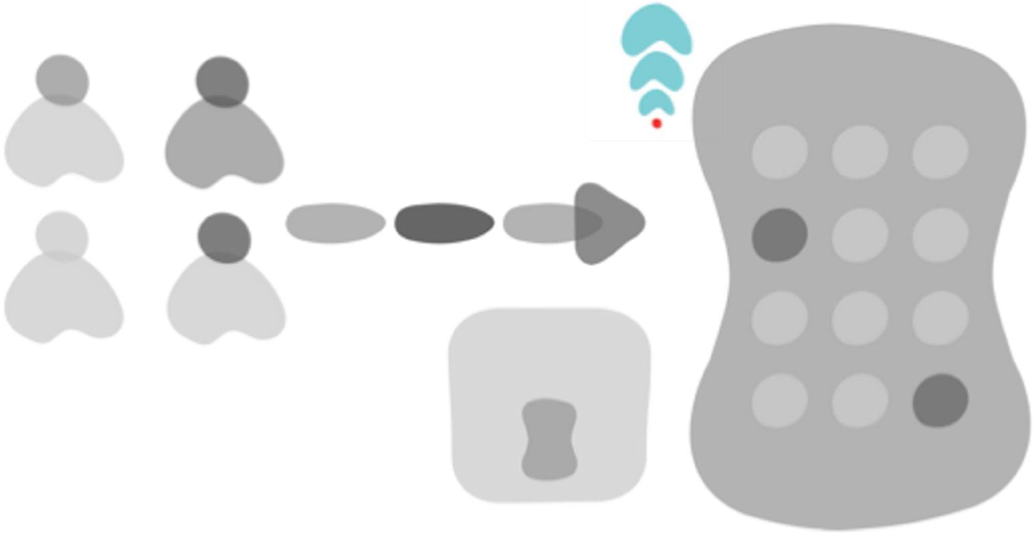
Semi-finalists

The Semi-Finalists

Congratulations Semi-Finalists!

Architecture Katas

Winter 2024



Neal Ford
Thoughtworks
Director / Software Architect / Meme Wrangler
http://www.nealford.com

Mark Richards
Independent Consultant
Hands-on Software Architect, Published Author
Founder, DeveloperToArchitect.com
@markrichardssa

Semi-Finals