O'REILLY®

LIVE ONLINE TRAINING

Architecture Katas Semi-Finalists

Sarah Taraporewalla

Principal Technologist
Director
Enterprise Modernisation,
Platforms & Cloud
ThoughtWorks
@sarahtarap
https://sarahtaraporewalla.com/

Luca Mezzalira

VP of Architecture at DAZN
International Speaker
Published Author
@lucamezzalira
https://lucamezzalira.com

Nathanial T. Schutta

Developer Advocate, VMWare
Architect as a Service
@ntschutta
http://www.ntschutta.io/

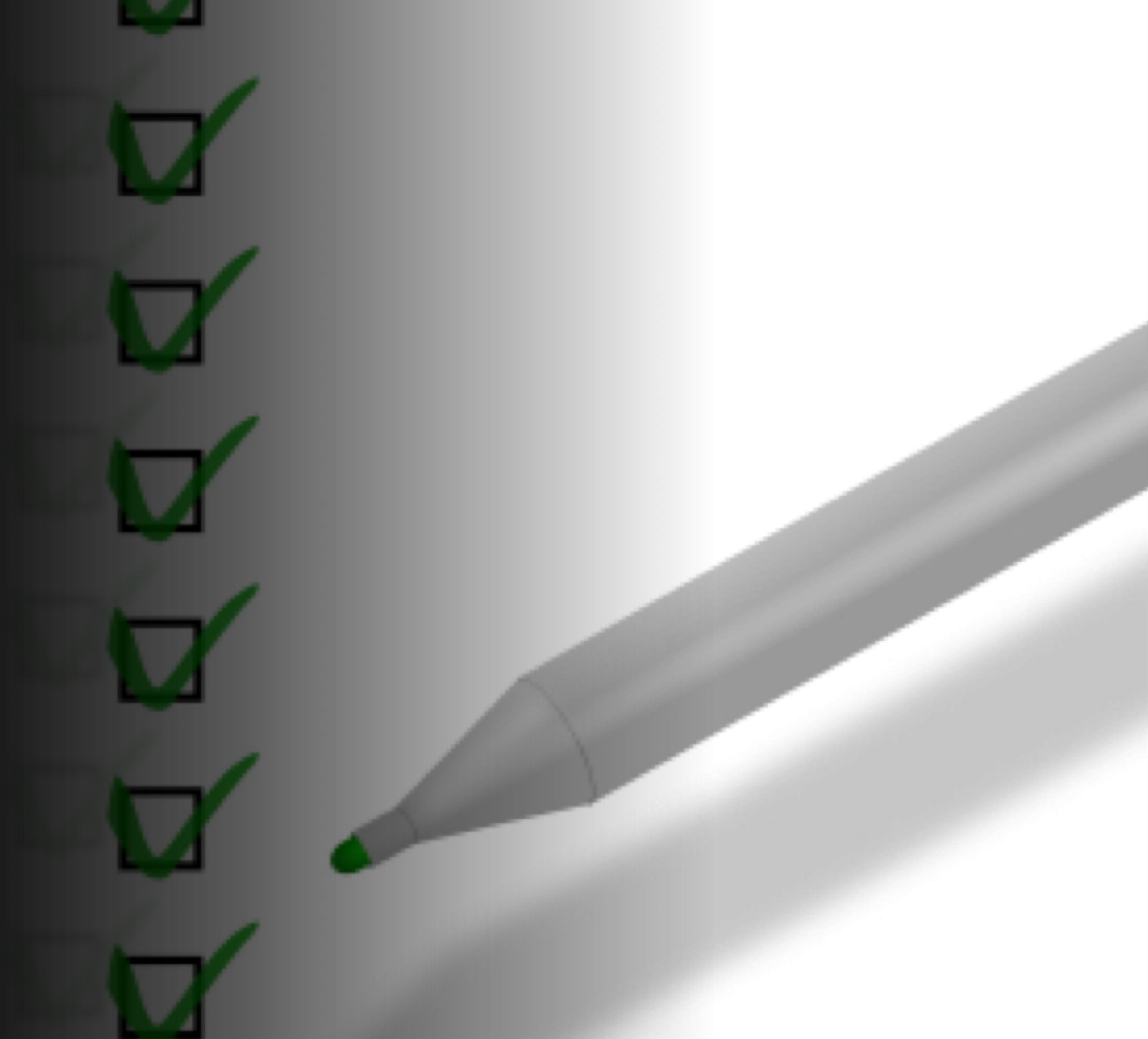
Mark Richards

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

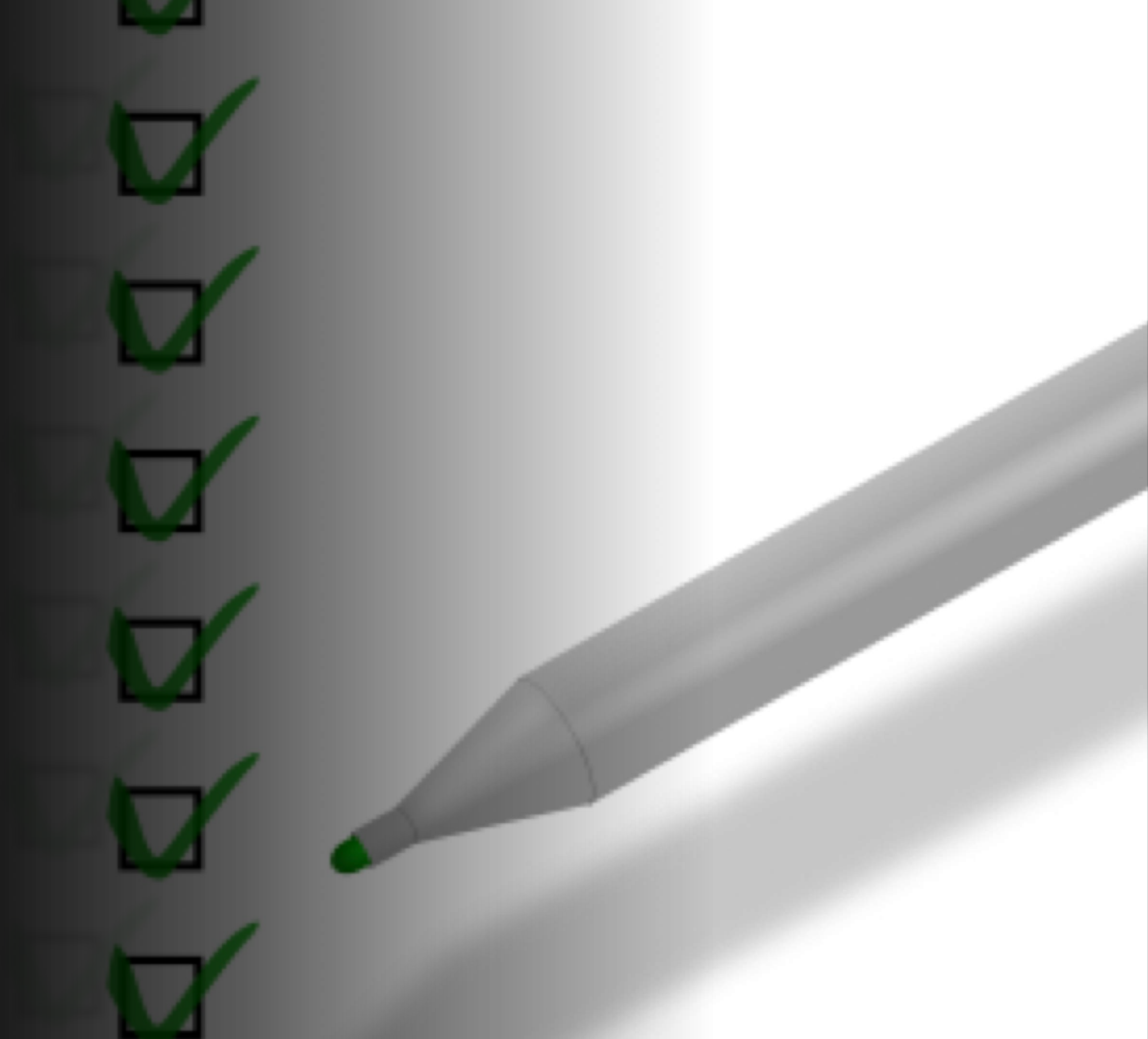
Overall Impressions

- ✓ Pragmatic approaches for a startup needing to integrate with 3rd party services
- ✓ Well thought out preparation using actors, use cases, journeys, and sequences
- ✓ Good demonstration of the understanding of the problem and the requirements
- ✓ Proposals with evolutionary architecture in mind
- ✓ Use of domain-driven design techniques and modeling
- ✓ Great narratives explaining the approach and high-level architecture
- ✓ Effective use of several architecture and design patterns
- ✓ Great use of architecture decision records to document and justify decisions

Judges Criteria

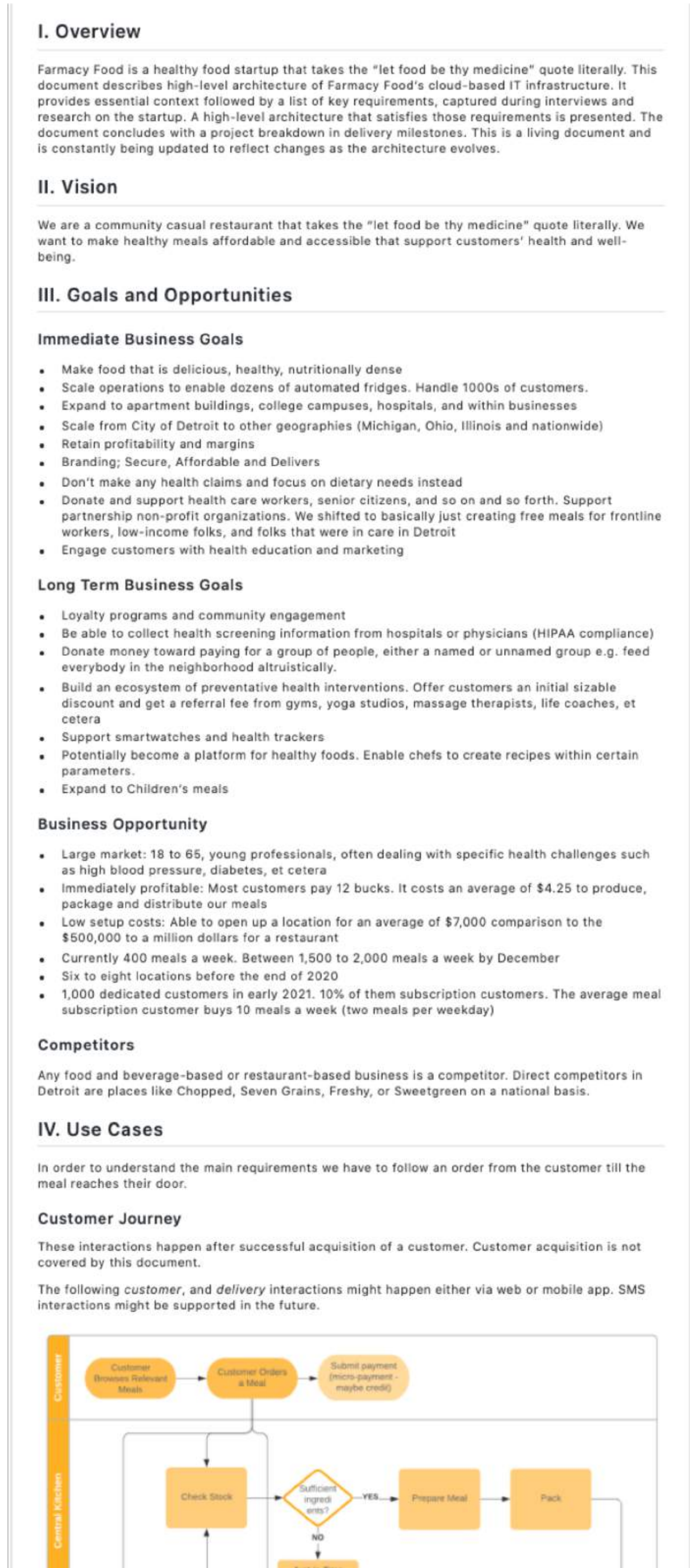


Clarity of narrative, organization, and supporting documentation



Judges Criteria

clarity - narrative, organization, supporting documentation



I. Overview

Farmacy Food is a healthy food startup that takes the "let food be thy medicine" quote literally. This document describes high-level architecture of Farmacy Food's cloud-based IT infrastructure. It provides essential context followed by a list of key requirements, captured during interviews and research on the startup. A high-level architecture that satisfies those requirements is presented. The document concludes with a project breakdown in delivery milestones. This is a living document and is constantly being updated to reflect changes as the architecture evolves.

II. Vision

We are a community casual restaurant that takes the "let food be thy medicine" quote literally. We want to make healthy meals affordable and accessible that support customers' health and well-being.

III. Goals and Opportunities

Immediate Business Goals

- Make food that is delicious, healthy, nutritionally dense
- Scale operations to enable dozens of automated fridges. Handle 1000s of customers.
- Expand to apartment buildings, college campuses, hospitals, and within businesses
- Scale from City of Detroit to other geographies (Michigan, Ohio, Illinois and nationwide)
- Retain profitability and margins
- Branding; Secure, Affordable and Delivers
- Don't make any health claims and focus on dietary needs instead
- Donate and support health care workers, senior citizens, and so on and so forth. Support partnership non-profit organizations. We shifted to basically just creating free meals for frontline workers, low-income folks, and folks that were in care in Detroit
- Engage customers with health education and marketing

Long Term Business Goals

- Loyalty programs and community engagement
- Be able to collect health screening information from hospitals or physicians (HIPAA compliance)
- Donate money toward paying for a group of people, either a named or unnamed group e.g. feed everybody in the neighborhood altruistically.
- Build an ecosystem of preventative health interventions. Offer customers an initial sizable discount and get a referral fee from gyms, yoga studios, massage therapists, life coaches, et cetera
- Support smartwatches and health trackers
- Potentially become a platform for healthy foods. Enable chefs to create recipes within certain parameters.
- Expand to Children's meals

Business Opportunity

- Large market: 18 to 65, young professionals, often dealing with specific health challenges such as high blood pressure, diabetes, et cetera
- Immediately profitable: Most customers pay 12 bucks. It costs an average of $4.25 to produce, package and distribute our meals
- Low setup costs: Able to open up a location for an average of $7,000 comparison to the $500,000 to a million dollars for a restaurant
- Currently 400 meals a week. Between 1,500 to 2,000 meals a week by December
- Six to eight locations before the end of 2020
- 1,000 dedicated customers in early 2021. 10% of them subscription customers. The average meal subscription customer buys 10 meals a week (two meals per weekday)

Competitors

Any food and beverage-based or restaurant-based business is a competitor. Direct competitors in Detroit are places like Chopped, Seven Grains, Freshy, or Sweetgreen on a national basis.

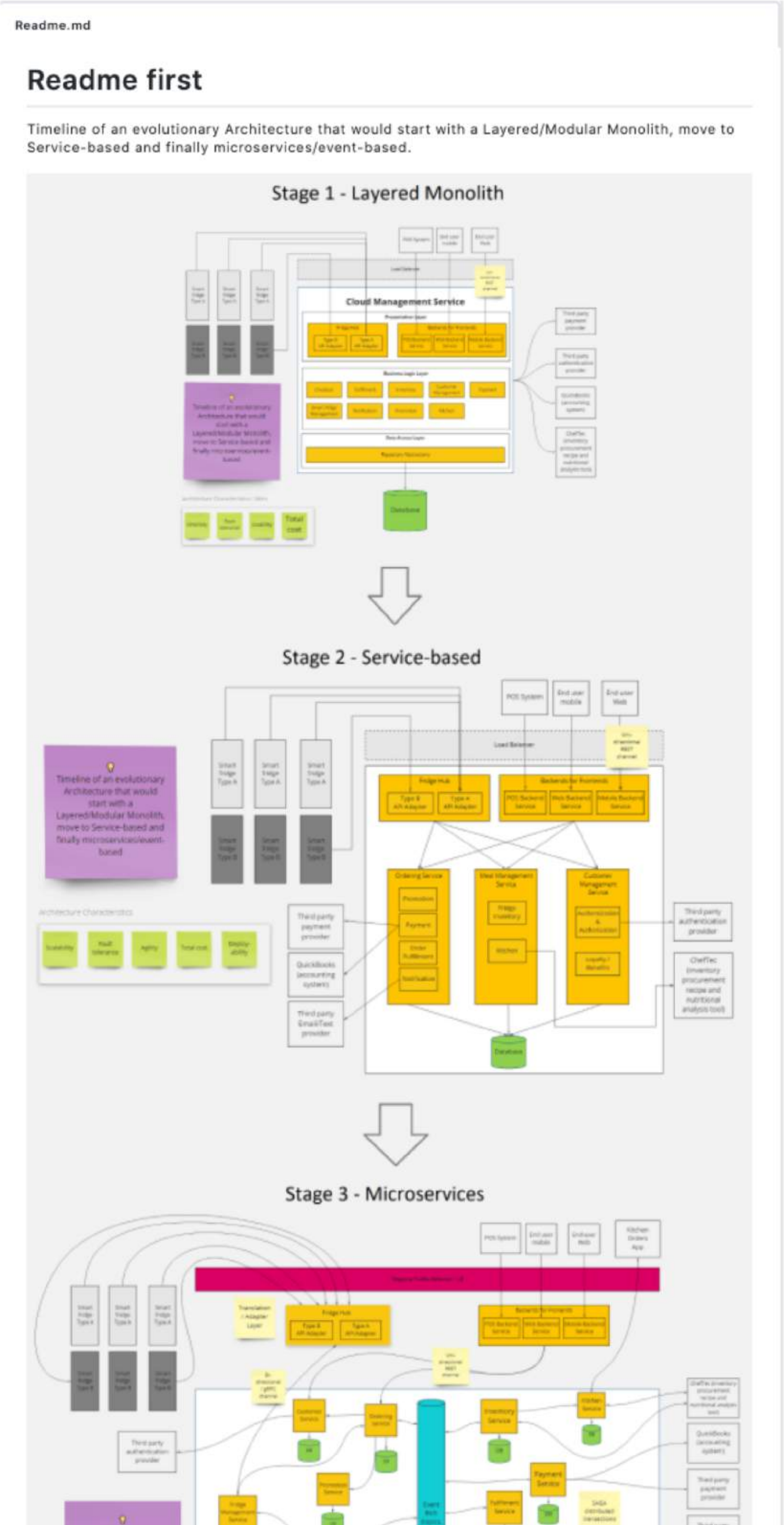
IV. Use Cases

In order to understand the main requirements we have to follow an order from the customer till the meal reaches their door.

Customer Journey

These interactions happen after successful acquisition of a customer. Customer acquisition is not covered by this document.

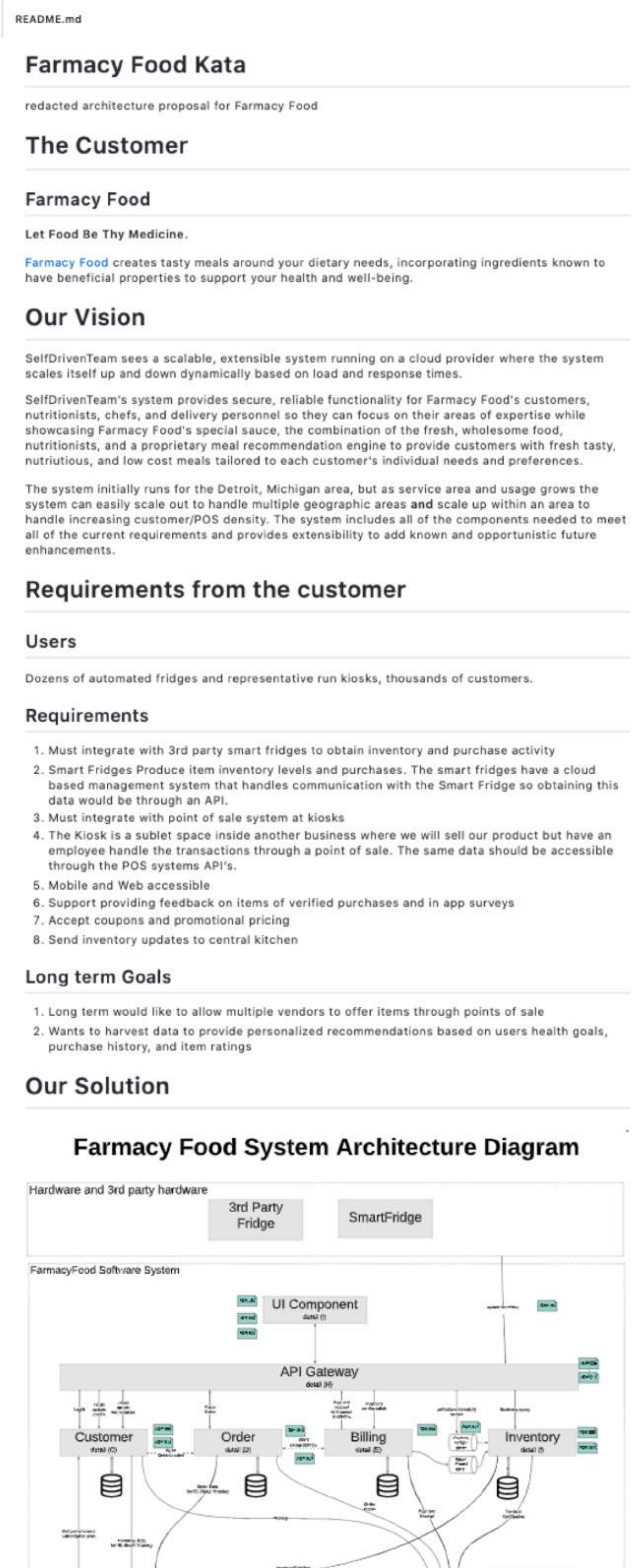
The following *customer*, and *delivery* interactions might happen either via web or mobile app. SMS interactions might be supported in the future.



Readme.md

Readme first

Timeline of an evolutionary Architecture that would start with a Layered/Modular Monolith, move to Service-based and finally microservices/event-based.



README.md

Farmacy Food Kata

redacted architecture proposal for Farmacy Food

The Customer

Farmacy Food

Let Food Be Thy Medicine.

Farmacy Food creates tasty meals around your dietary needs, incorporating ingredients known to have beneficial properties to support your health and well-being.

Our Vision

SelfDrivenTeam sees a scalable, extensible system running on a cloud provider where the system scales itself up and down dynamically based on load and response times.

SelfDrivenTeam's system provides secure, reliable functionality for Farmacy Food's customers, nutritionists, chefs, and delivery personnel so they can focus on their areas of expertise while showcasing Farmacy Food's special sauce, the combination of the fresh, wholesome food, nutritionists, and a proprietary meal recommendation engine to provide customers with fresh tasty, nutriutious, and low cost meals tailored to each customer's individual needs and preferences.

The system initially runs for the Detroit, Michigan area, but as service area and usage grows the system can easily scale out to handle multiple geographic areas **and** scale up within an area to handle increasing customer/POS density. The system includes all of the components needed to meet all of the current requirements and provides extensibility to add known and opportunistic future enhancements.

Requirements from the customer

Users

Dozens of automated fridges and representative run kiosks, thousands of customers.

Requirements

1. Must integrate with 3rd party smart fridges to obtain inventory and purchase activity
2. Smart Fridges Produce item inventory levels and purchases. The smart fridges have a cloud based management system that handles communication with the Smart Fridge so obtaining this data would be through an API.
3. Must integrate with point of sale system at kiosks
4. The Kiosk is a sublet space inside another business where we will sell our product but have an employee handle the transactions through a point of sale. The same data should be accessible through the POS systems API's.
5. Mobile and Web accessible
6. Support providing feedback on items of verified purchases and in app surveys
7. Accept coupons and promotional pricing
8. Send inventory updates to central kitchen

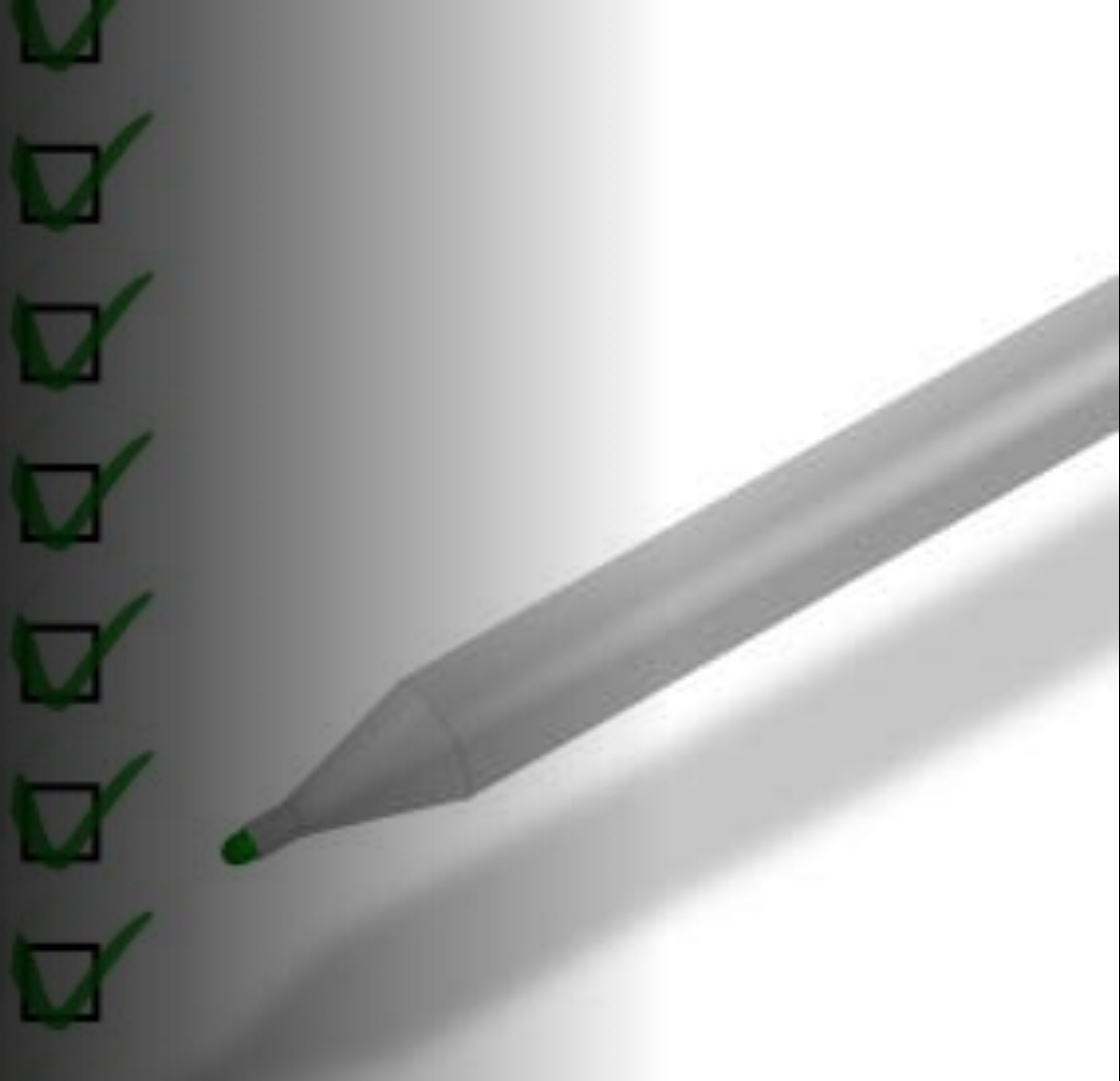
Long term Goals

1. Long term would like to allow multiple vendors to offer items through points of sale
2. Wants to harvest data to provide personalized recommendations based on users health goals, purchase history, and item ratings

Our Solution

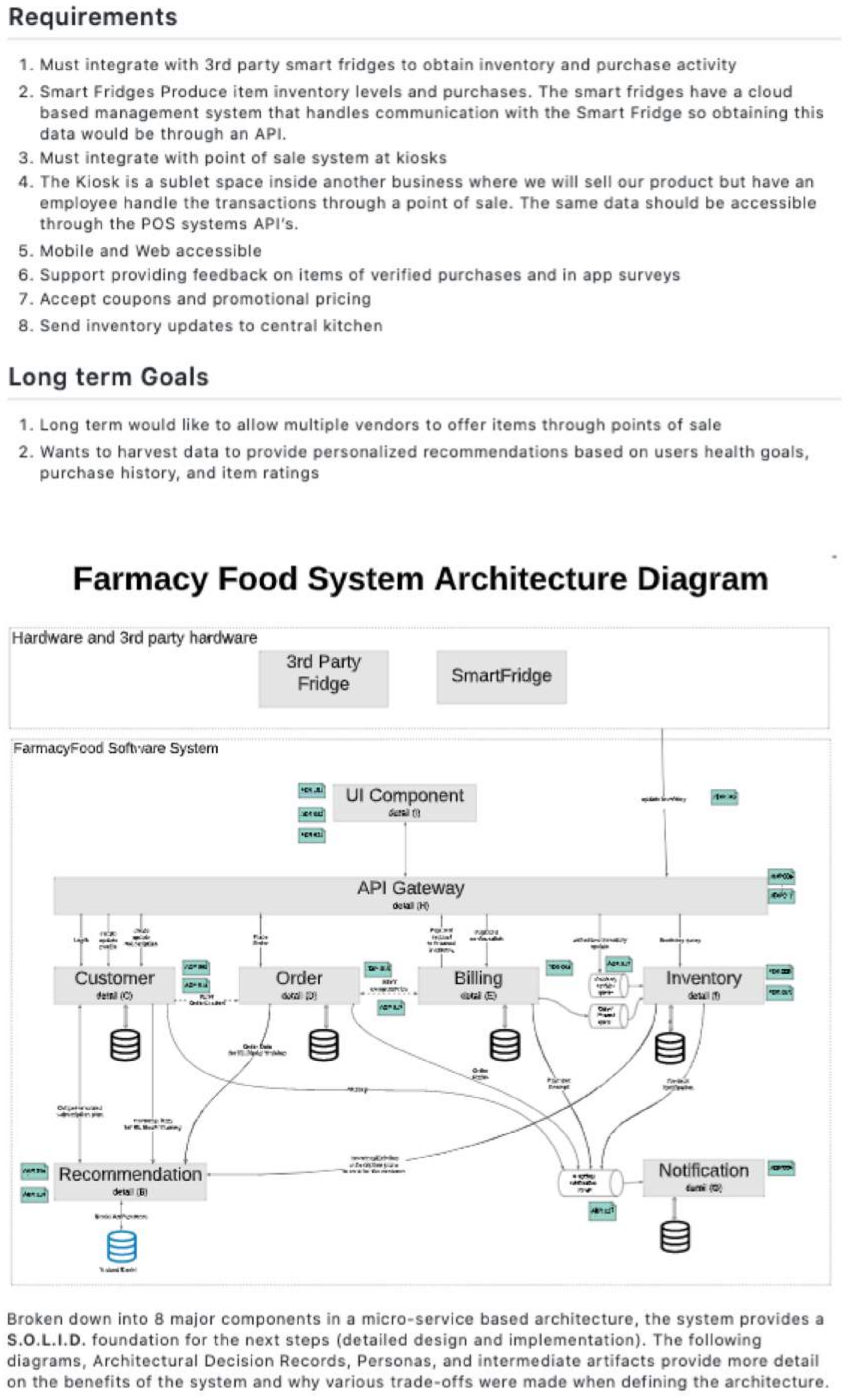
Farmacy Food System Architecture Diagram

Understanding of the requirements and completeness of solution



Judges Criteria

understanding of the requirements and completeness of solution



Requirements

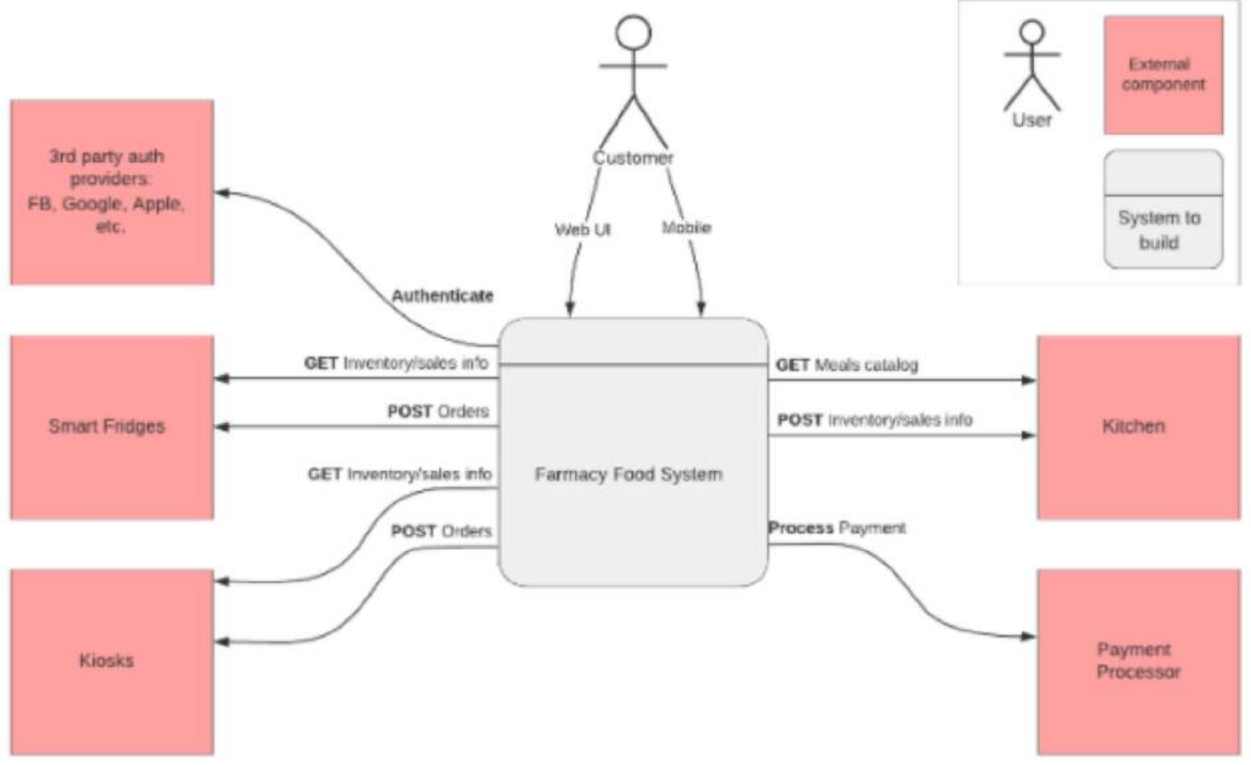
1. Must integrate with 3rd party smart fridges to obtain inventory and purchase activity
2. Smart Fridges Produce item inventory levels and purchases. The smart fridges have a cloud based management system that handles communication with the Smart Fridge so obtaining this data would be through an API.
3. Must integrate with point of sale system at kiosks
4. The Kiosk is a sublet space inside another business where we will sell our product but have an employee handle the transactions through a point of sale. The same data should be accessible through the POS systems API's.
5. Mobile and Web accessible
6. Support providing feedback on items of verified purchases and in app surveys
7. Accept coupons and promotional pricing
8. Send inventory updates to central kitchen

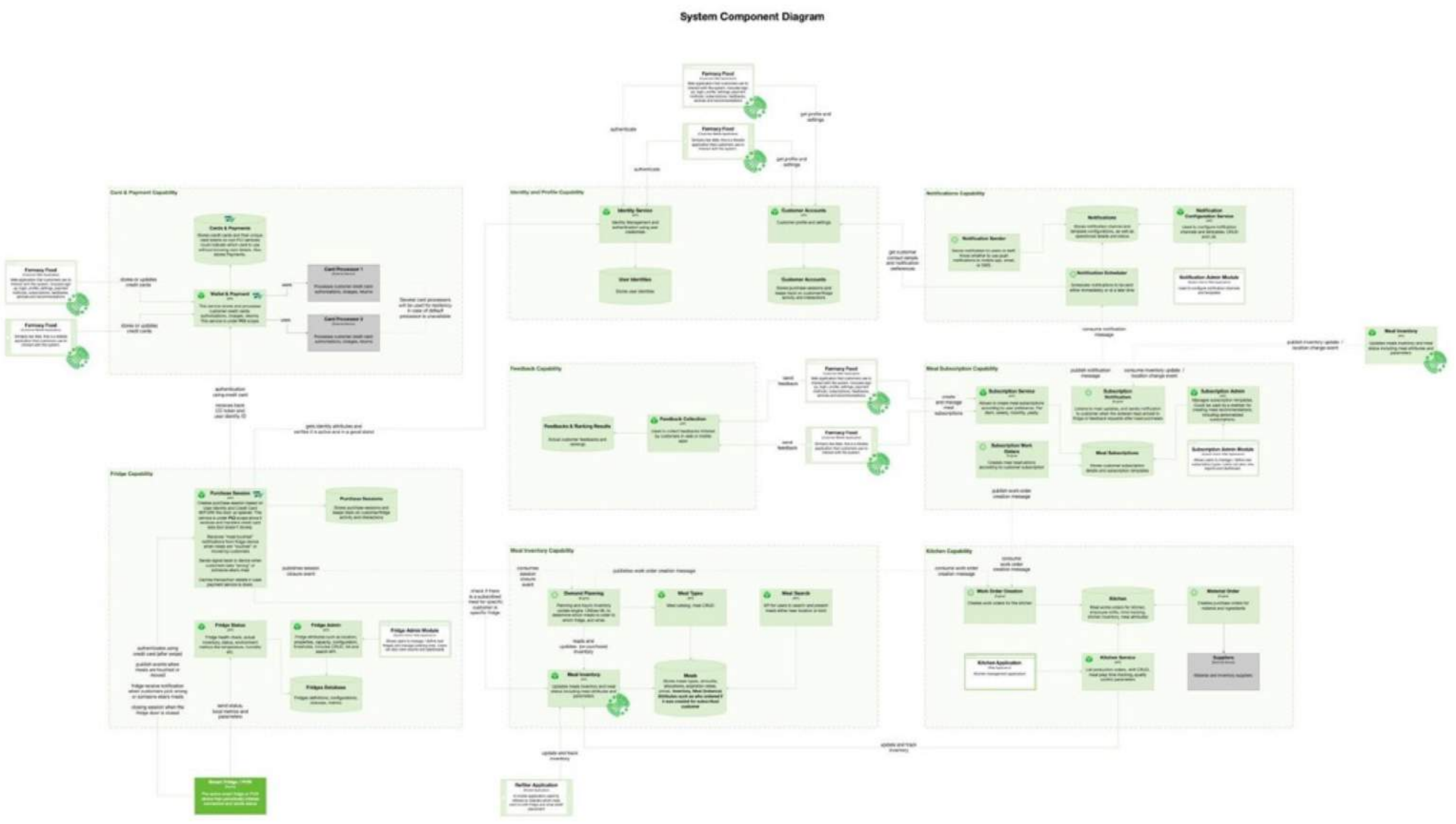
Long term Goals

1. Long term would like to allow multiple vendors to offer items through points of sale
2. Wants to harvest data to provide personalized recommendations based on users health goals, purchase history, and item ratings

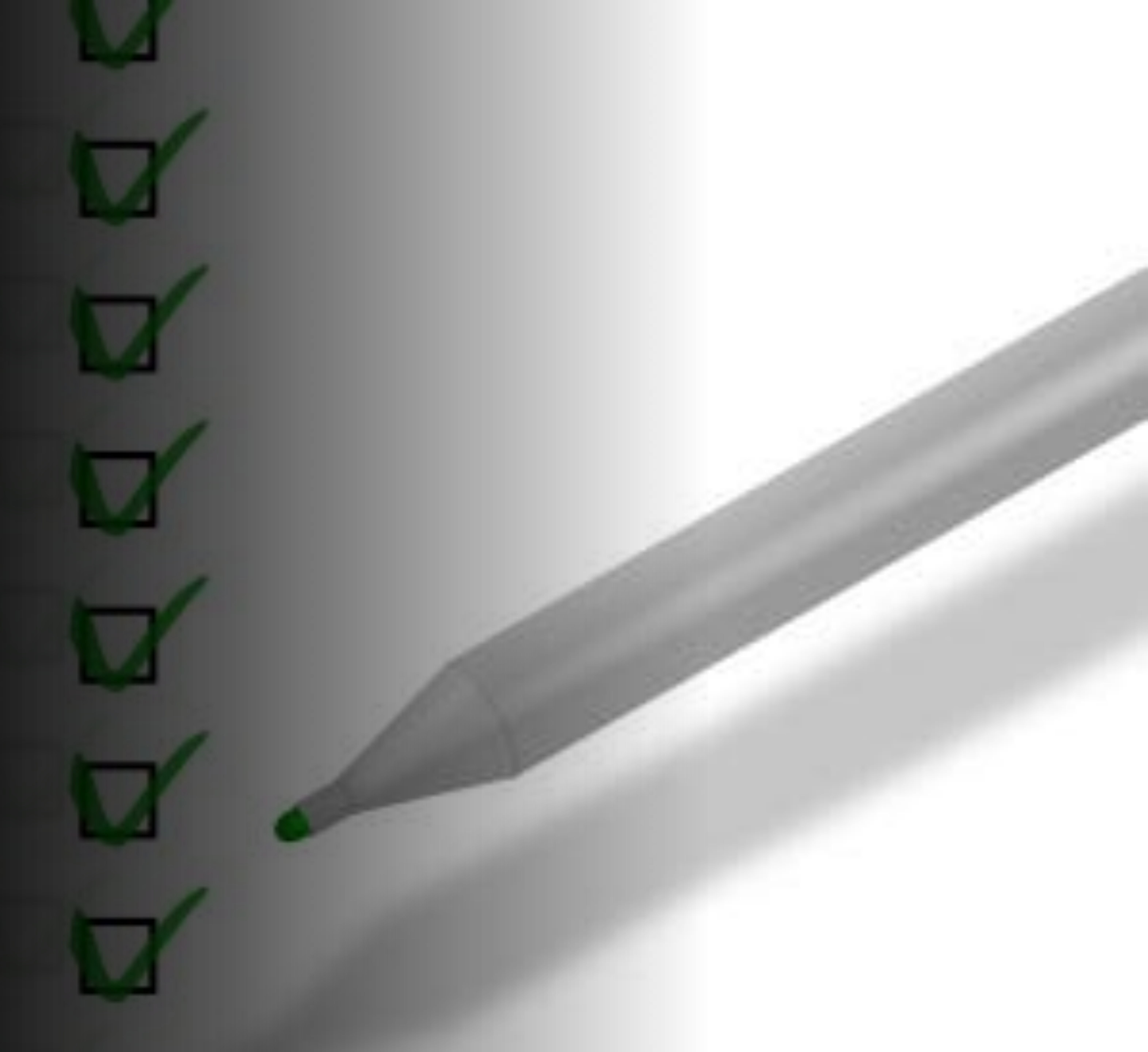
Farmacy Food System Architecture Diagram

Broken down into 8 major components in a micro-service based architecture, the system provides a S.O.L.I.D. foundation for the next steps (detailed design and implementation). The following diagrams, Architectural Decision Records, Personas, and intermediate artifacts provide more detail on the benefits of the system and why various trade-offs were made when defining the architecture.



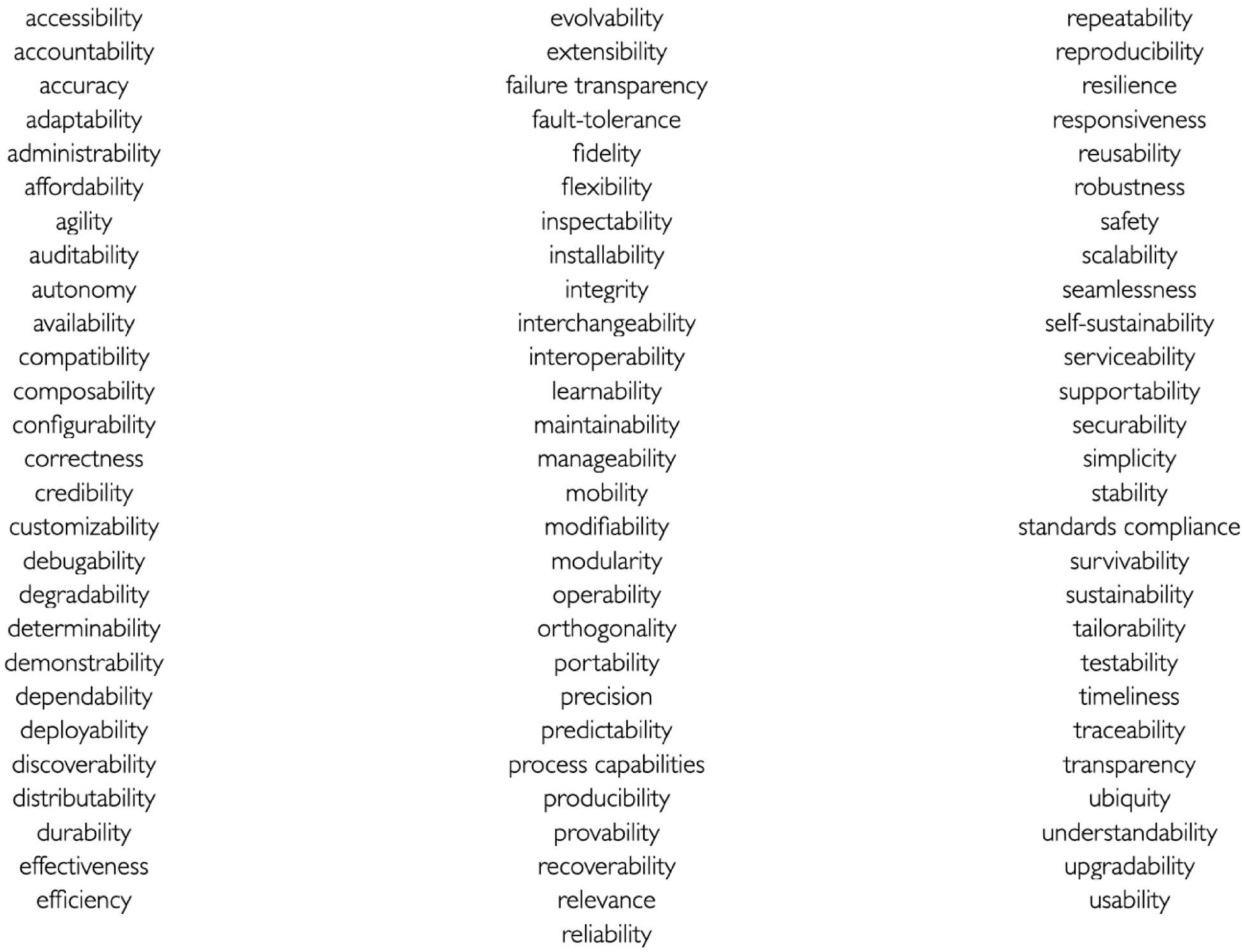


Identification of supporting architecture characteristics



Judges Criteria

identification of supporting architecture characteristics



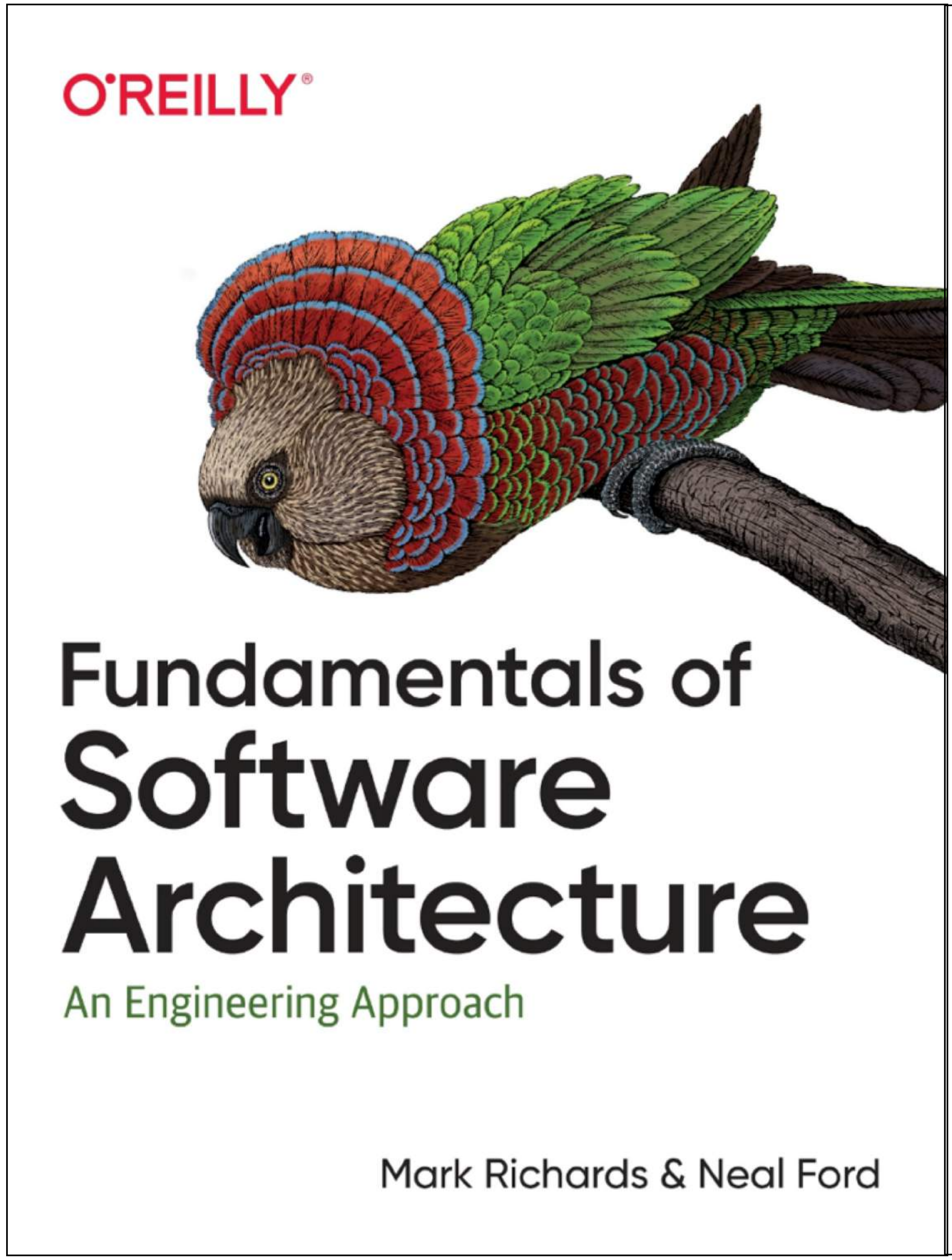
accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy
availability
compatibility
composability
configurability
correctness
credibility
customizability
debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

evolvability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity
interchangeability
interoperability
learnability
maintainability
manageability
mobility
modifiability
modularity
operability
orthogonality
portability
precision
predictability
process capabilities
producibility
provability
recoverability
relevance
reliability

repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness
self-sustainability
serviceability
supportability
securability
simplicity
stability
standards compliance
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

Judges Criteria

identification of supporting architecture characteristics



O'REILLY

Fundamentals of Software Architecture

An Engineering Approach

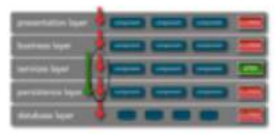
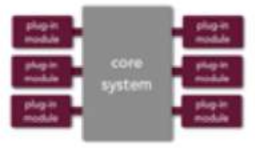
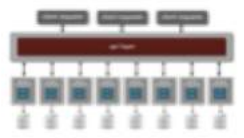
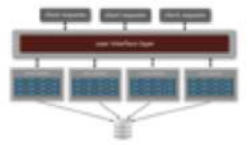
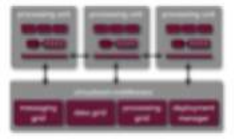
Mark Richards & Neal Ford

First Law of Software Architecture

"*Everything* in software architecture is a tradeoff"

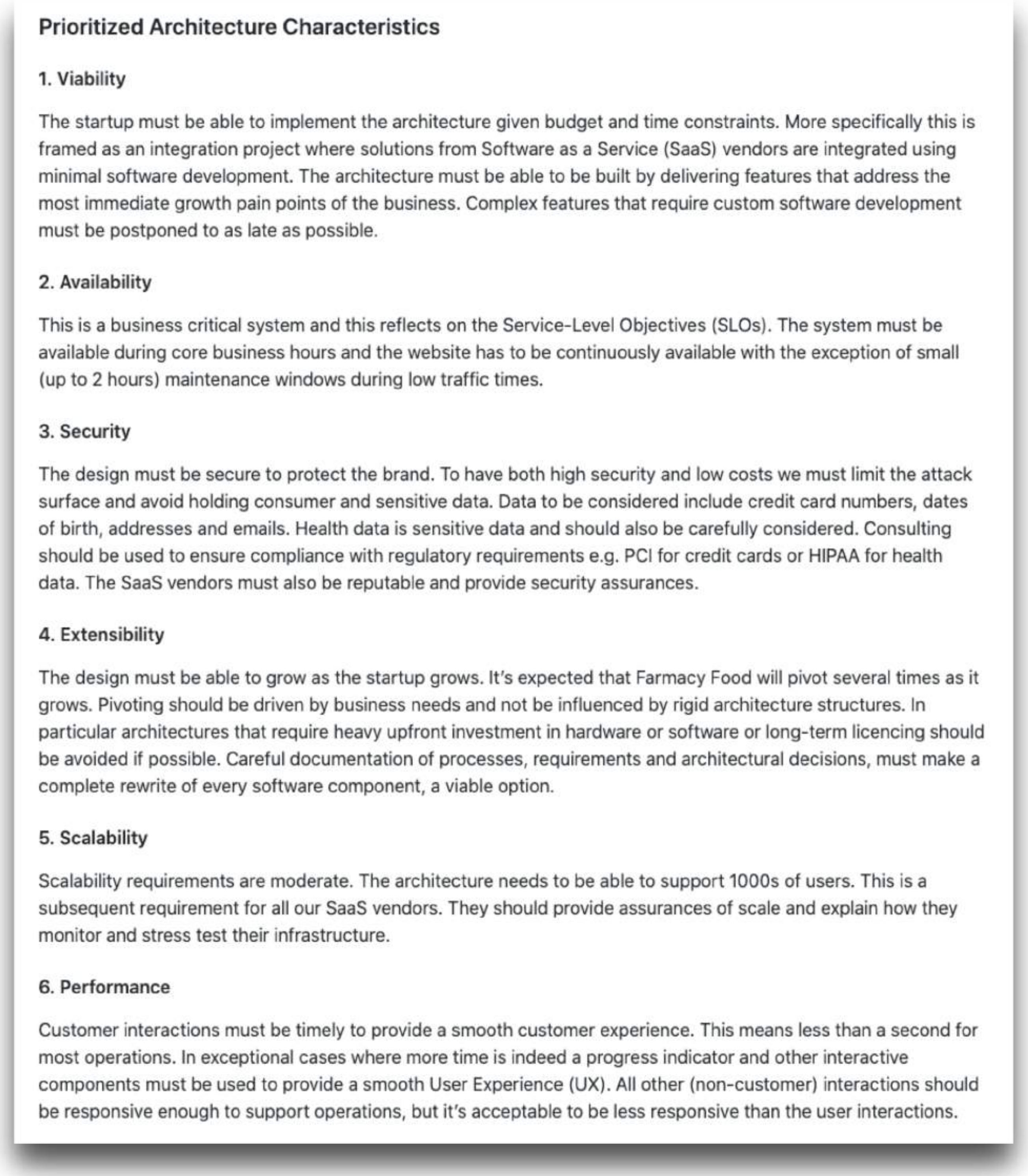
Judges Criteria

identification of supporting architecture characteristics

	 layered	 modular monolith	 microkernel	 microservices	 service-based	 service-oriented	 event-driven	 space-based
agility	★	★★	★★★	★★★★★	★★★★	★	★★★	★★
abstraction	★	★	★★★	★	★	★★★★★	★★★★	★
configurability	★	★	★★★★	★★★	★★	★	★★	★★
cost	★★★★★	★★★★★	★★★★★	★	★★★★	★	★★★	★★
deployability	★	★★	★★★	★★★★★	★★★★	★	★★★	★★★
domain part.	★	★★★★★	★★★★★	★★★★★	★★★★★	★	★	★★★★★
elasticity	★	★	★	★★★★★	★★	★★★	★★★★	★★★★★
evolvability	★	★	★★★	★★★★★	★★★	★	★★★★★	★★★
fault-tolerance	★	★	★	★★★★★	★★★★	★★★	★★★★★	★★★
integration	★	★	★★★	★★★	★★	★★★★★	★★★	★★
interoperability	★	★	★★★	★★★	★★	★★★★★	★★★	★★
performance	★★★	★★★	★★★	★★	★★★	★★	★★★★★	★★★★★
scalability	★	★	★	★★★★★	★★★	★★★	★★★★★	★★★★★
simplicity	★★★★★	★★★★★	★★★★	★	★★★	★	★	★
testability	★★	★★	★★★	★★★★★	★★★★	★	★★	★
workflow	★	★	★★	★	★	★★★★★	★★★★★	★

Judges Criteria

identification of supporting architecture characteristics



Prioritized Architecture Characteristics

1. Viability

The startup must be able to implement the architecture given budget and time constraints. More specifically this is framed as an integration project where solutions from Software as a Service (SaaS) vendors are integrated using minimal software development. The architecture must be able to be built by delivering features that address the most immediate growth pain points of the business. Complex features that require custom software development must be postponed to as late as possible.

2. Availability

This is a business critical system and this reflects on the Service-Level Objectives (SLOs). The system must be available during core business hours and the website has to be continuously available with the exception of small (up to 2 hours) maintenance windows during low traffic times.

3. Security

The design must be secure to protect the brand. To have both high security and low costs we must limit the attack surface and avoid holding consumer and sensitive data. Data to be considered include credit card numbers, dates of birth, addresses and emails. Health data is sensitive data and should also be carefully considered. Consulting should be used to ensure compliance with regulatory requirements e.g. PCI for credit cards or HIPAA for health data. The SaaS vendors must also be reputable and provide security assurances.

4. Extensibility

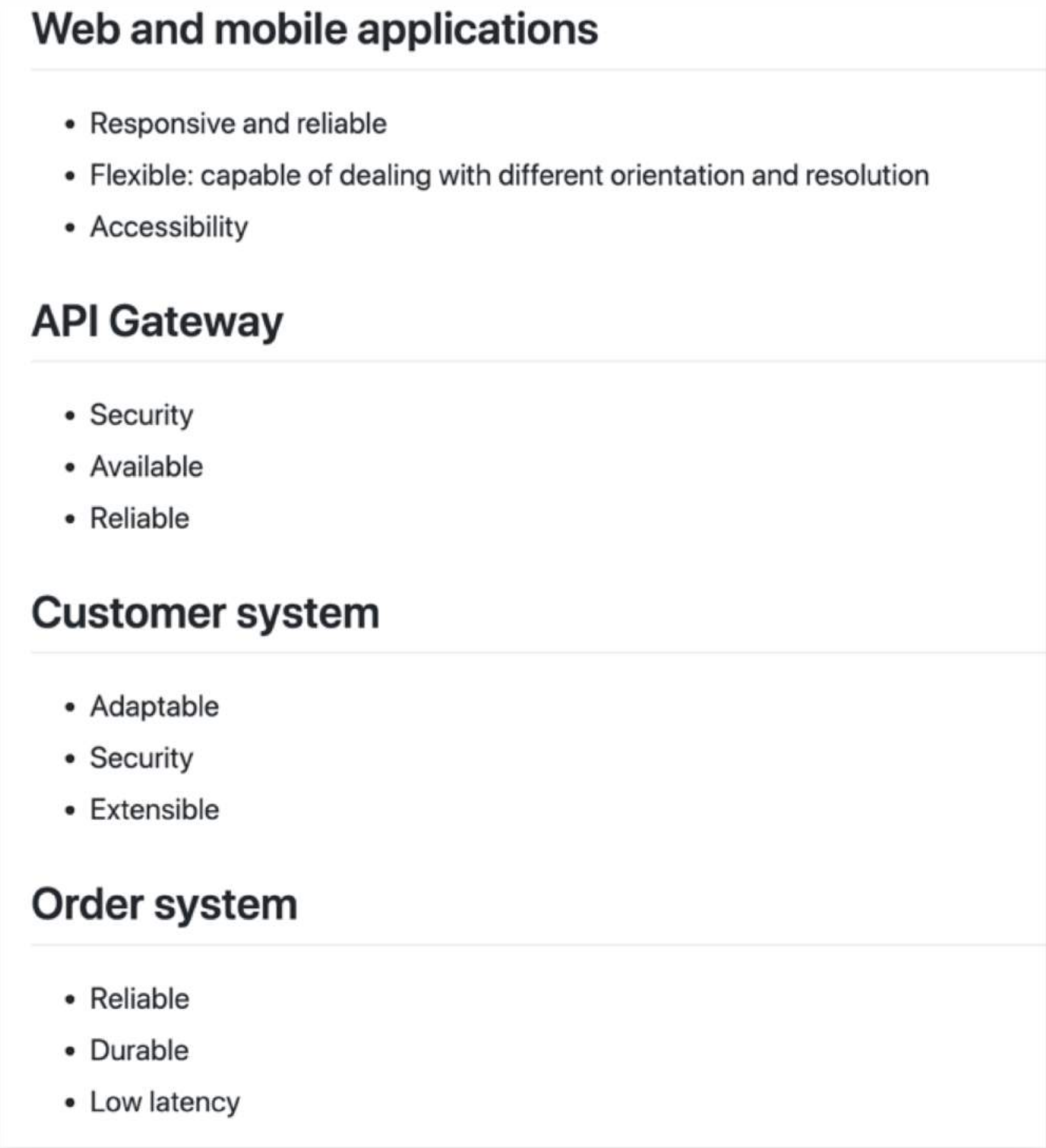
The design must be able to grow as the startup grows. It's expected that Farmacy Food will pivot several times as it grows. Pivoting should be driven by business needs and not be influenced by rigid architecture structures. In particular architectures that require heavy upfront investment in hardware or software or long-term licencing should be avoided if possible. Careful documentation of processes, requirements and architectural decisions, must make a complete rewrite of every software component, a viable option.

5. Scalability

Scalability requirements are moderate. The architecture needs to be able to support 1000s of users. This is a subsequent requirement for all our SaaS vendors. They should provide assurances of scale and explain how they monitor and stress test their infrastructure.

6. Performance

Customer interactions must be timely to provide a smooth customer experience. This means less than a second for most operations. In exceptional cases where more time is indeed a progress indicator and other interactive components must be used to provide a smooth User Experience (UX). All other (non-customer) interactions should be responsive enough to support operations, but it's acceptable to be less responsive than the user interactions.



Web and mobile applications

- Responsive and reliable
- Flexible: capable of dealing with different orientation and resolution
- Accessibility

API Gateway

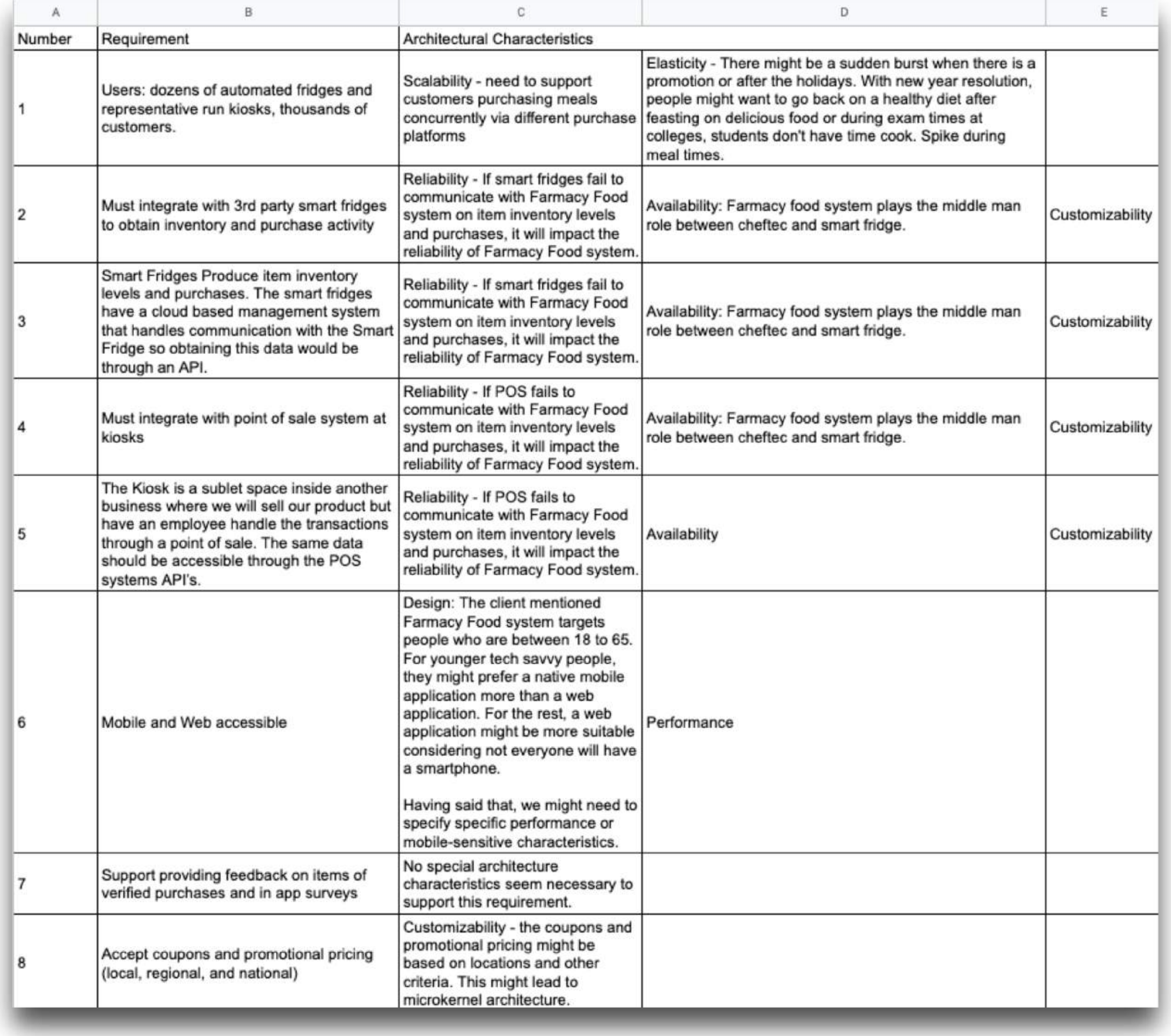
- Security
- Available
- Reliable

Customer system

- Adaptable
- Security
- Extensible

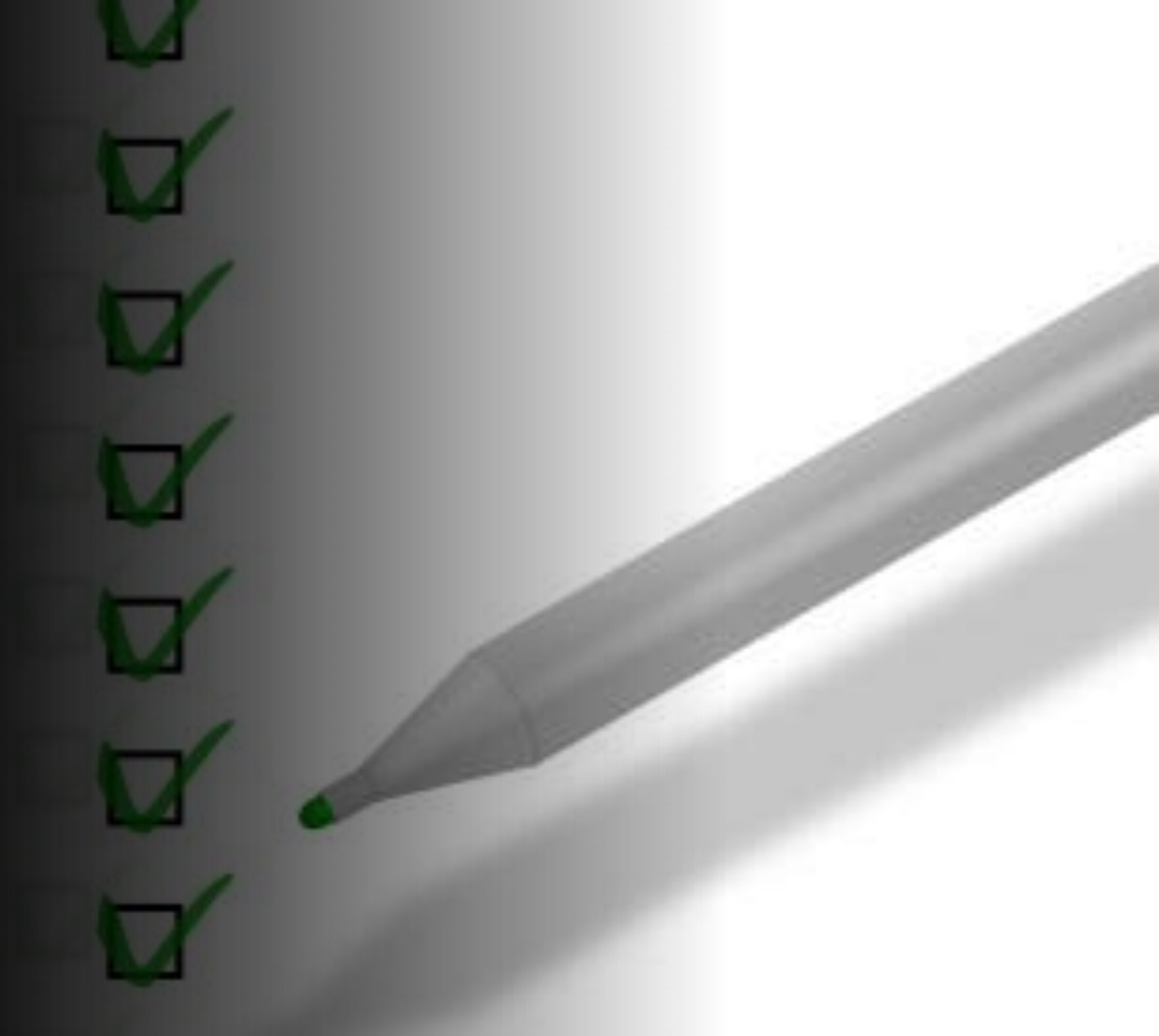
Order system

- Reliable
- Durable
- Low latency



A	B	C	D	E
Number	Requirement	Architectural Characteristics		
1	Users: dozens of automated fridges and representative run kiosks, thousands of customers.	Scalability - need to support customers purchasing meals concurrently via different purchase platforms	Elasticity - There might be a sudden burst when there is a promotion or after the holidays. With new year resolution, people might want to go back on a healthy diet after feasting on delicious food or during exam times at colleges, students don't have time cook. Spike during meal times.	
2	Must integrate with 3rd party smart fridges to obtain inventory and purchase activity	Reliability - If smart fridges fail to communicate with Farmacy Food system on item inventory levels and purchases, it will impact the reliability of Farmacy Food system.	Availability: Farmacy food system plays the middle man role between cheftec and smart fridge.	Customizability
3	Smart Fridges Produce item inventory levels and purchases. The smart fridges have a cloud based management system that handles communication with the Smart Fridge so obtaining this data would be through an API.	Reliability - If smart fridges fail to communicate with Farmacy Food system on item inventory levels and purchases, it will impact the reliability of Farmacy Food system.	Availability: Farmacy food system plays the middle man role between cheftec and smart fridge.	Customizability
4	Must integrate with point of sale system at kiosks	Reliability - If POS fails to communicate with Farmacy Food system on item inventory levels and purchases, it will impact the reliability of Farmacy Food system.	Availability: Farmacy food system plays the middle man role between cheftec and smart fridge.	Customizability
5	The Kiosk is a sublet space inside another business where we will sell our product but have an employee handle the transactions through a point of sale. The same data should be accessible through the POS systems API's.	Reliability - If POS fails to communicate with Farmacy Food system on item inventory levels and purchases, it will impact the reliability of Farmacy Food system.	Availability	Customizability
6	Mobile and Web accessible	Design: The client mentioned Farmacy Food system targets people who are between 18 to 65. For younger tech savvy people, they might prefer a native mobile application more than a web application. For the rest, a web application might be more suitable considering not everyone will have a smartphone. Having said that, we might need to specify specific performance or mobile-sensitive characteristics.	Performance	
7	Support providing feedback on items of verified purchases and in app surveys	No special architecture characteristics seem necessary to support this requirement.		
8	Accept coupons and promotional pricing (local, regional, and national)	Customizability - the coupons and promotional pricing might be based on locations and other criteria. This might lead to microkernel architecture.		

Diagrams – types, level of detail, completeness



Judges Criteria

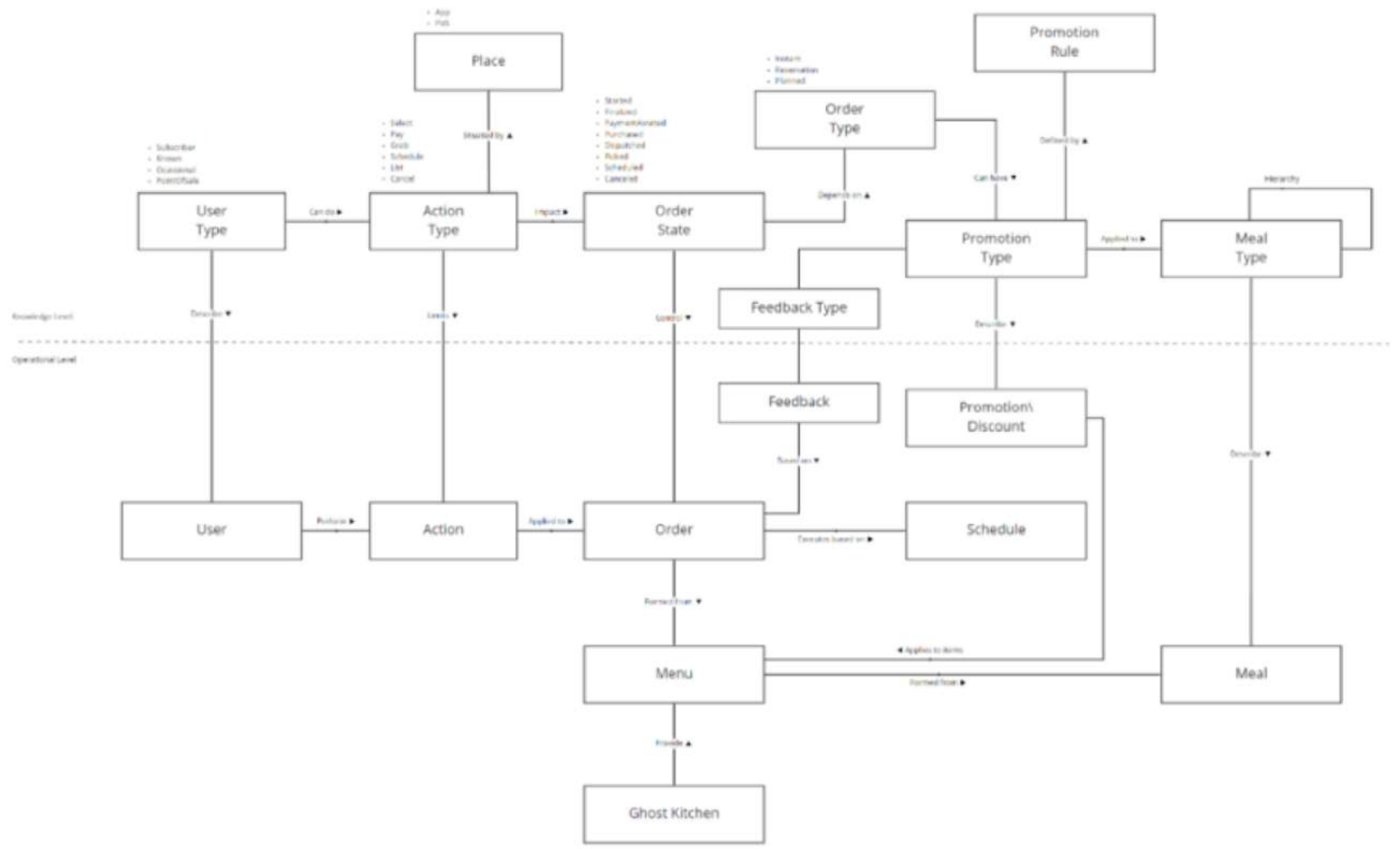
diagrams - types, level of detail, completeness

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

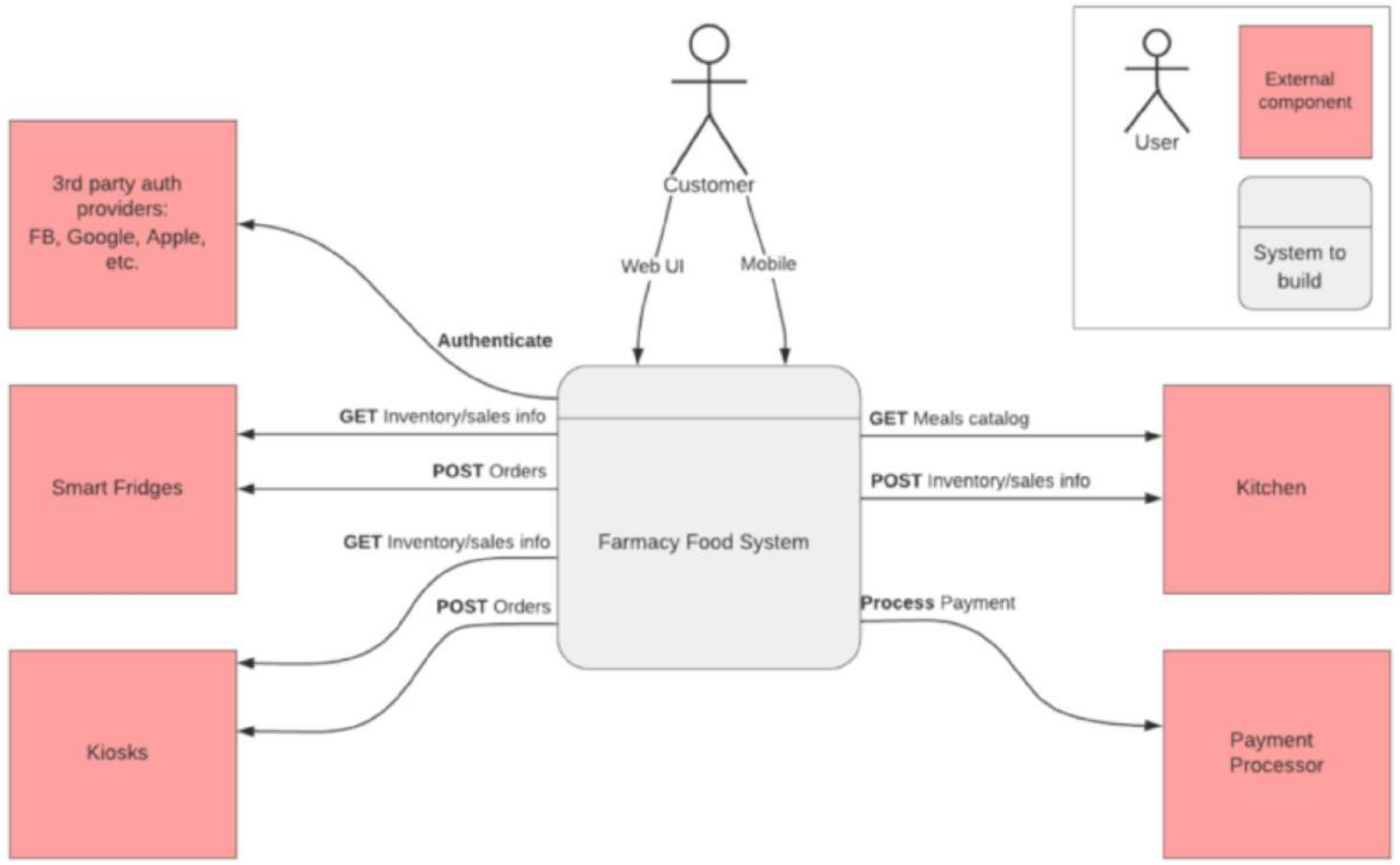
- Neal Ford

Judges Criteria

diagrams - types, level of detail, completeness



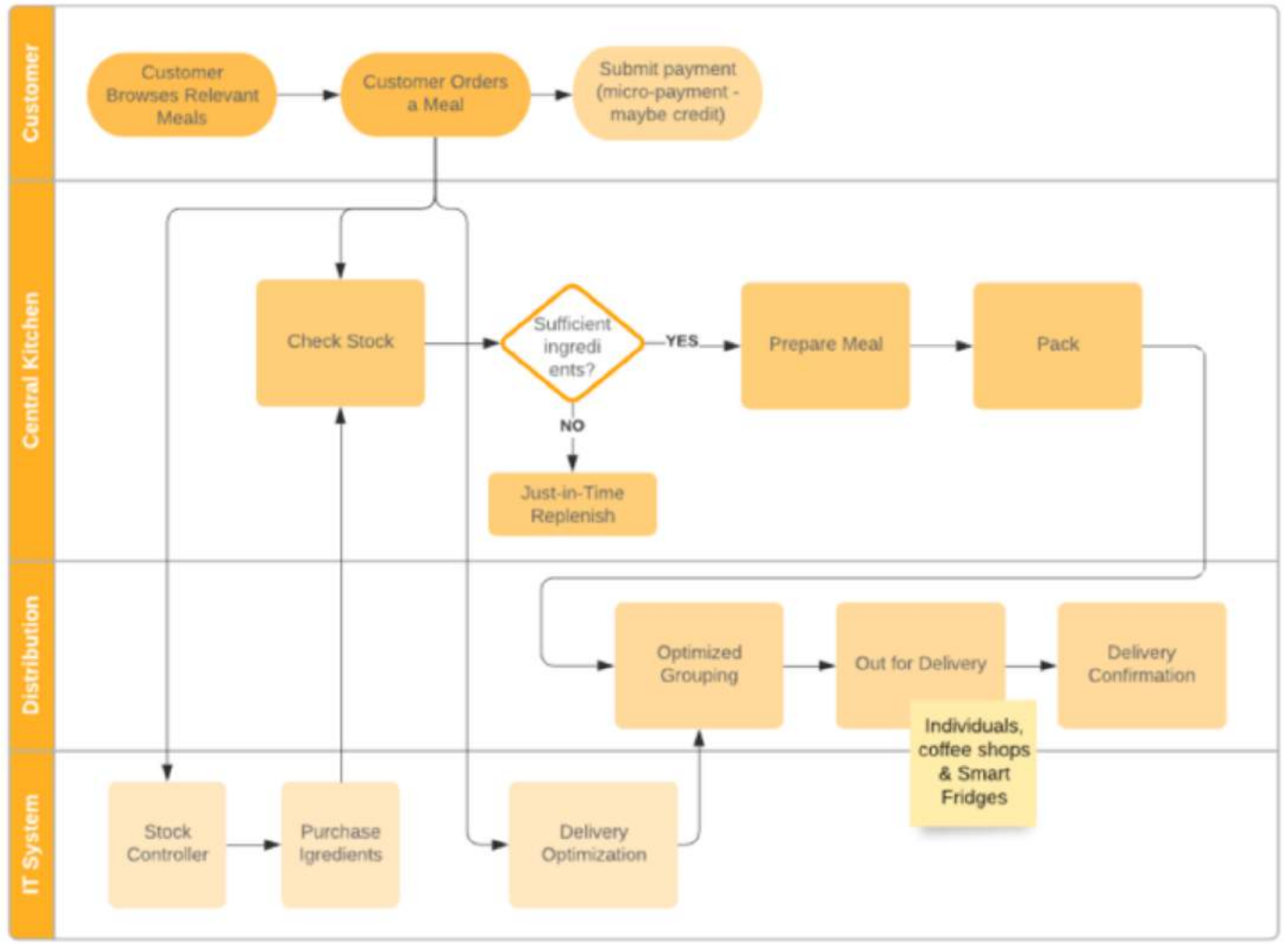
component diagrams



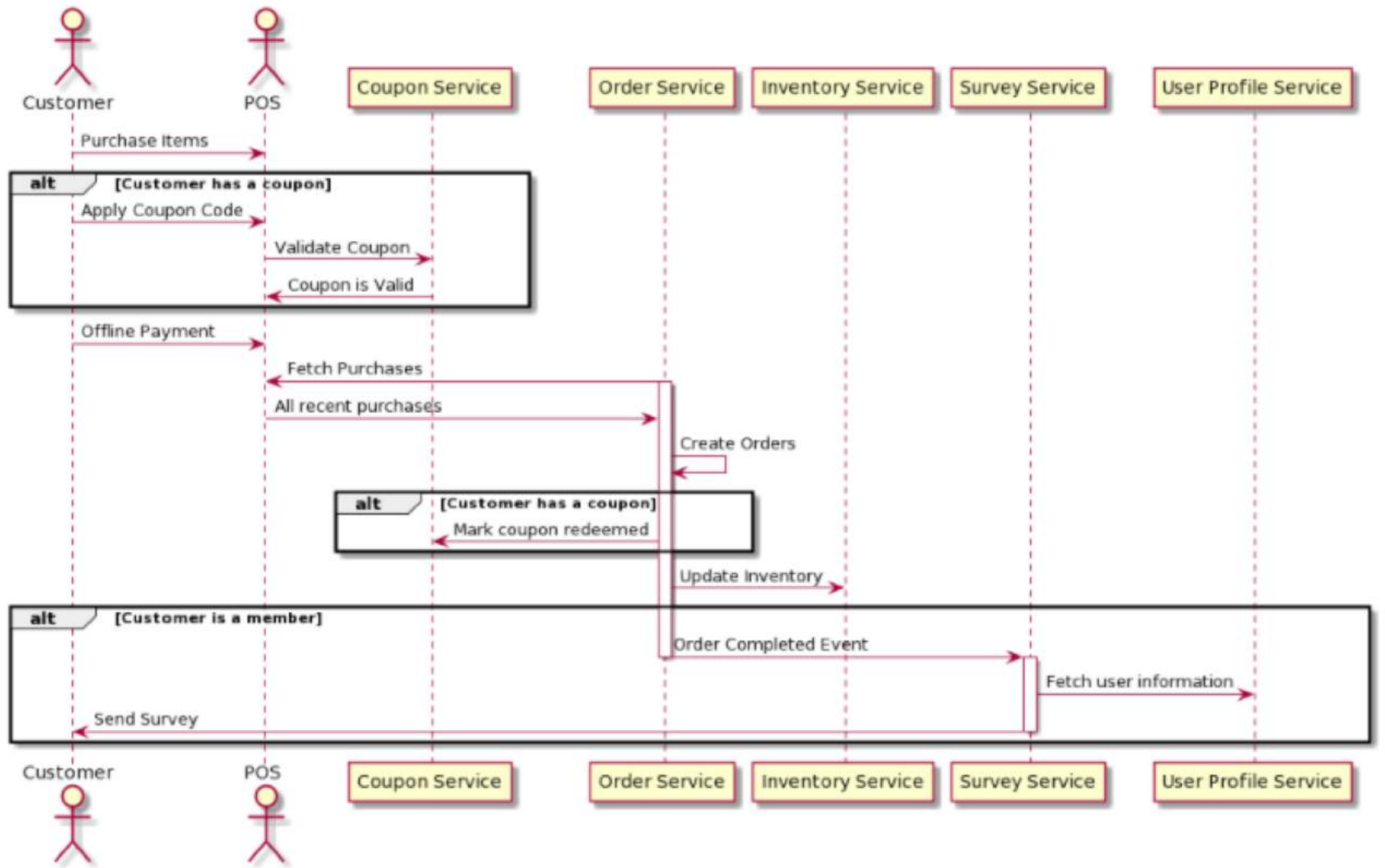
context diagrams

Judges Criteria

diagrams - types, level of detail, completeness



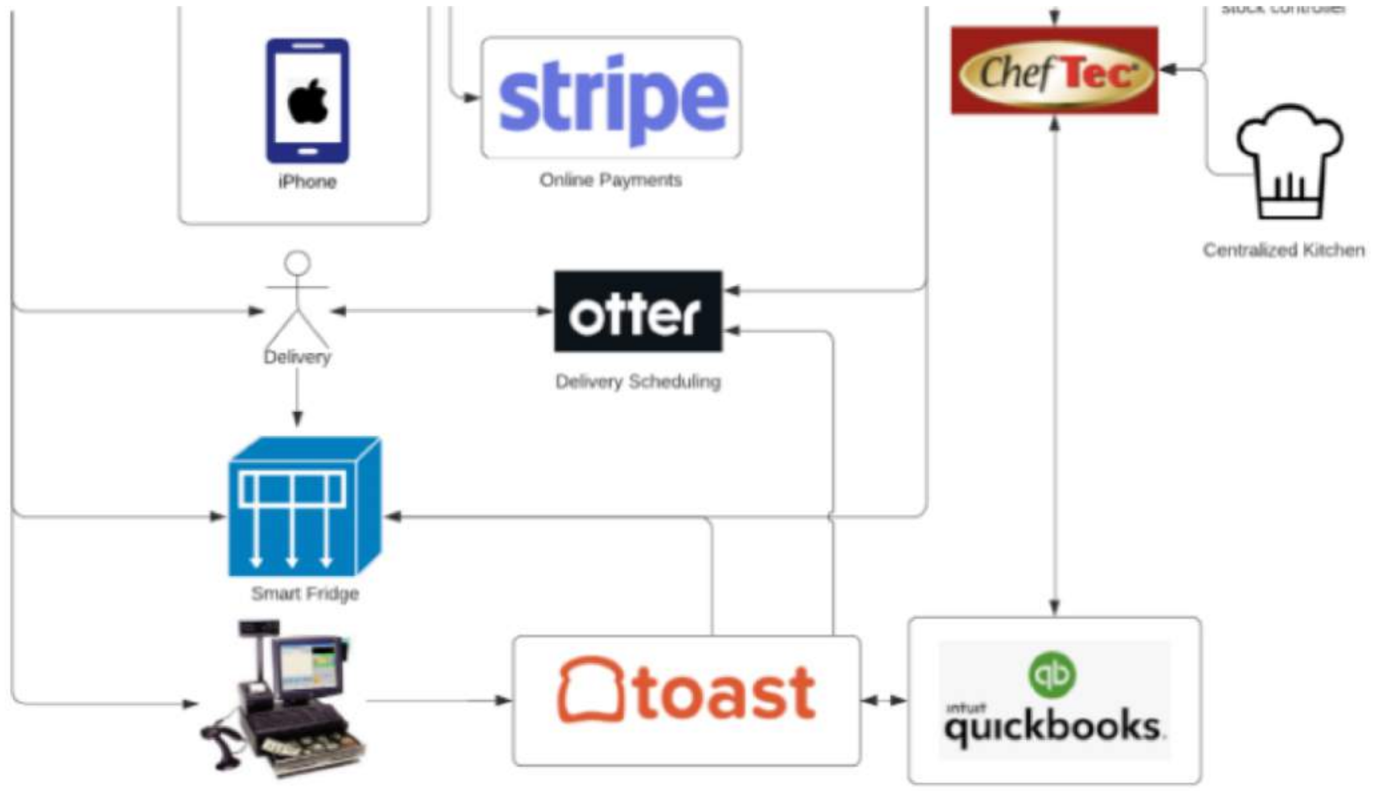
user journey diagrams



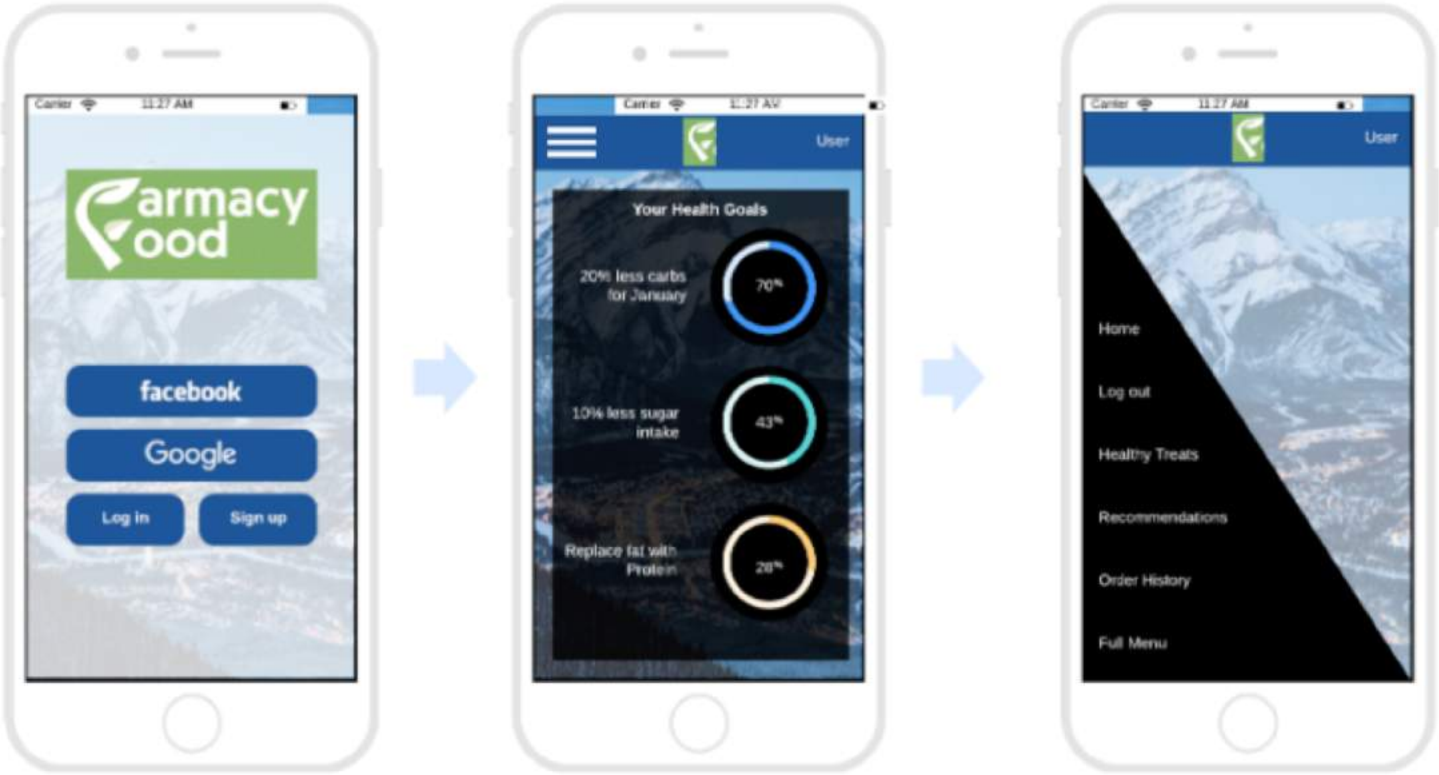
sequence diagrams

Judges Criteria

diagrams - types, level of detail, completeness

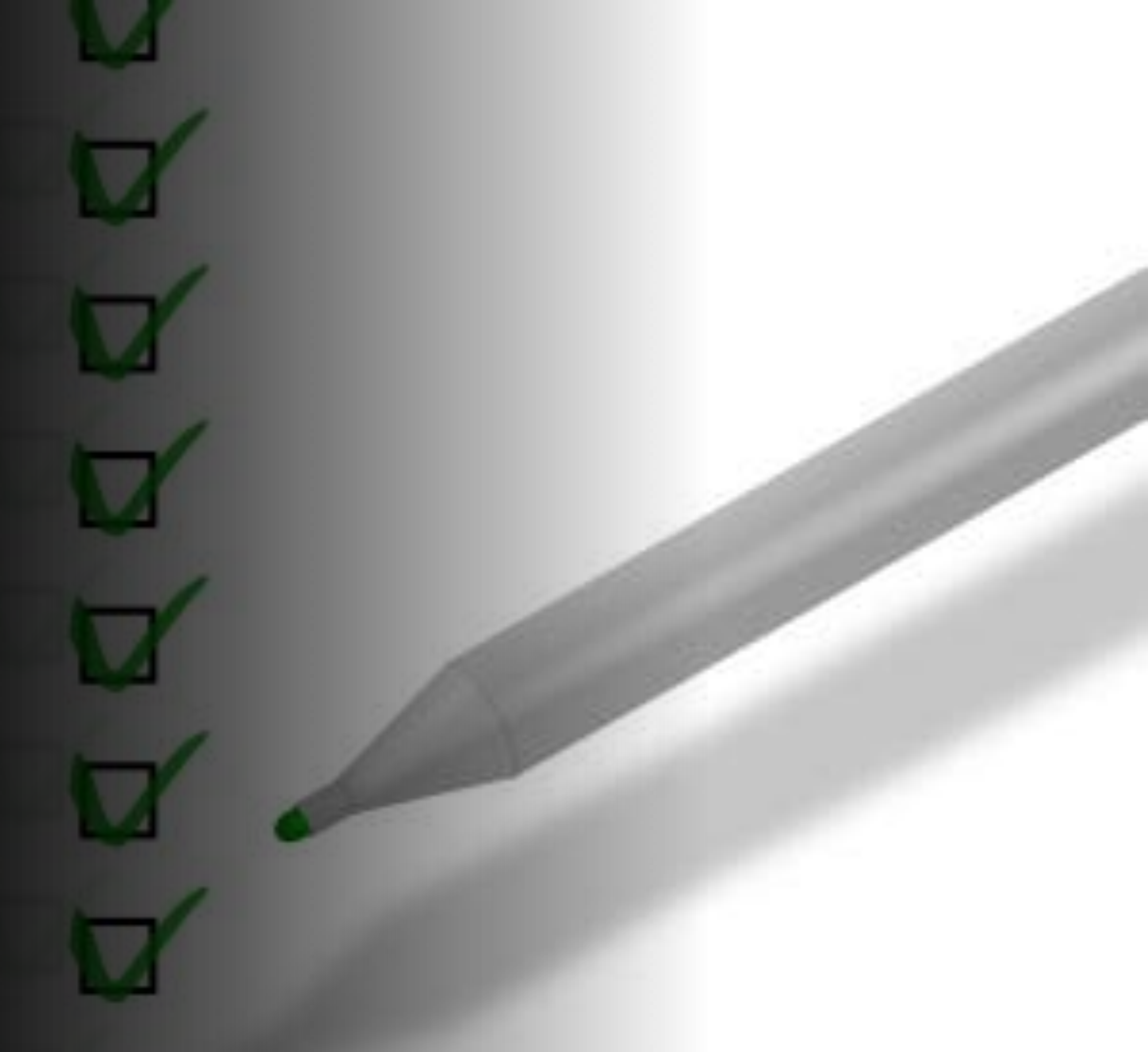


system-level diagrams



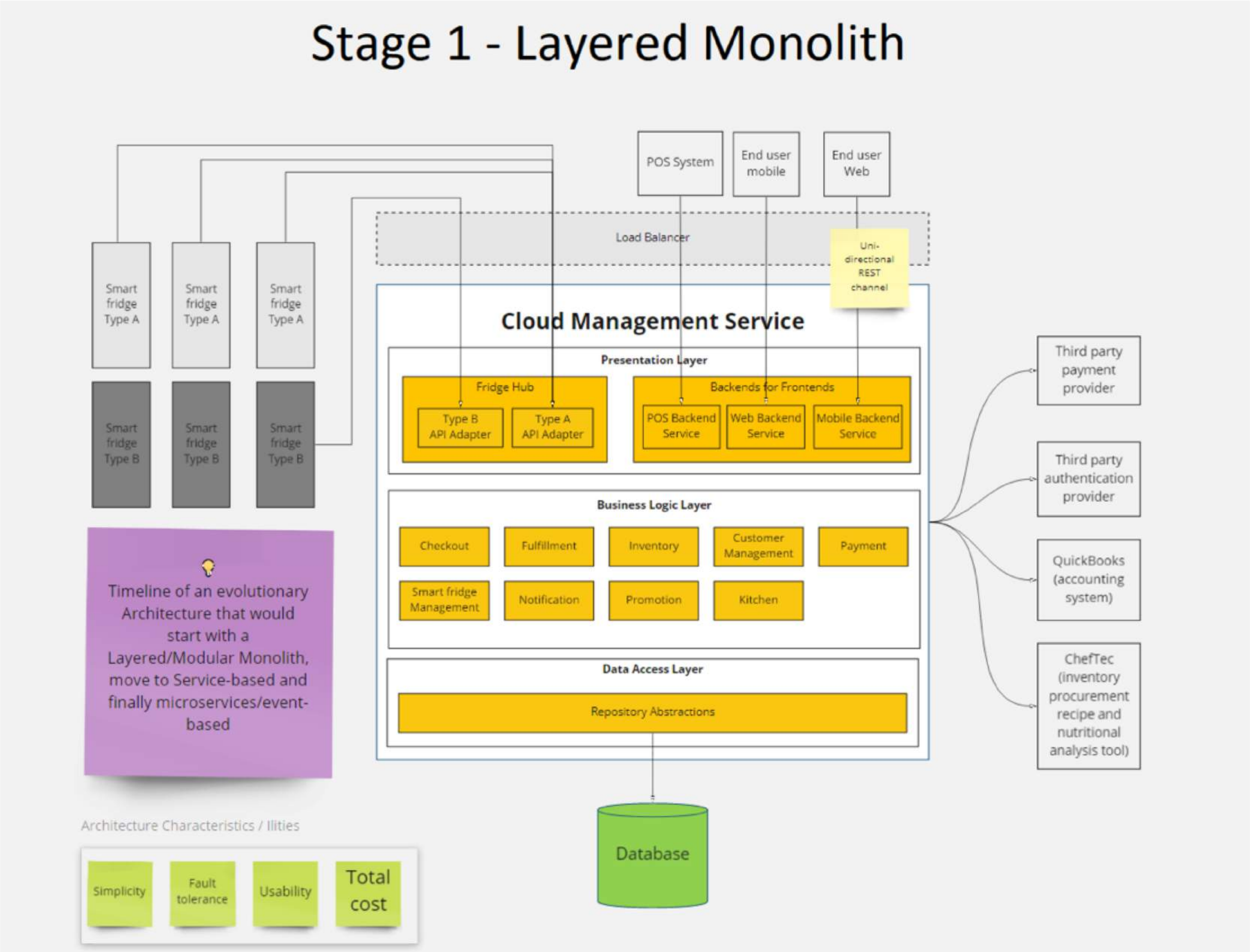
user interface mockups

Overall systems architecture



Judges Criteria

overall systems architecture

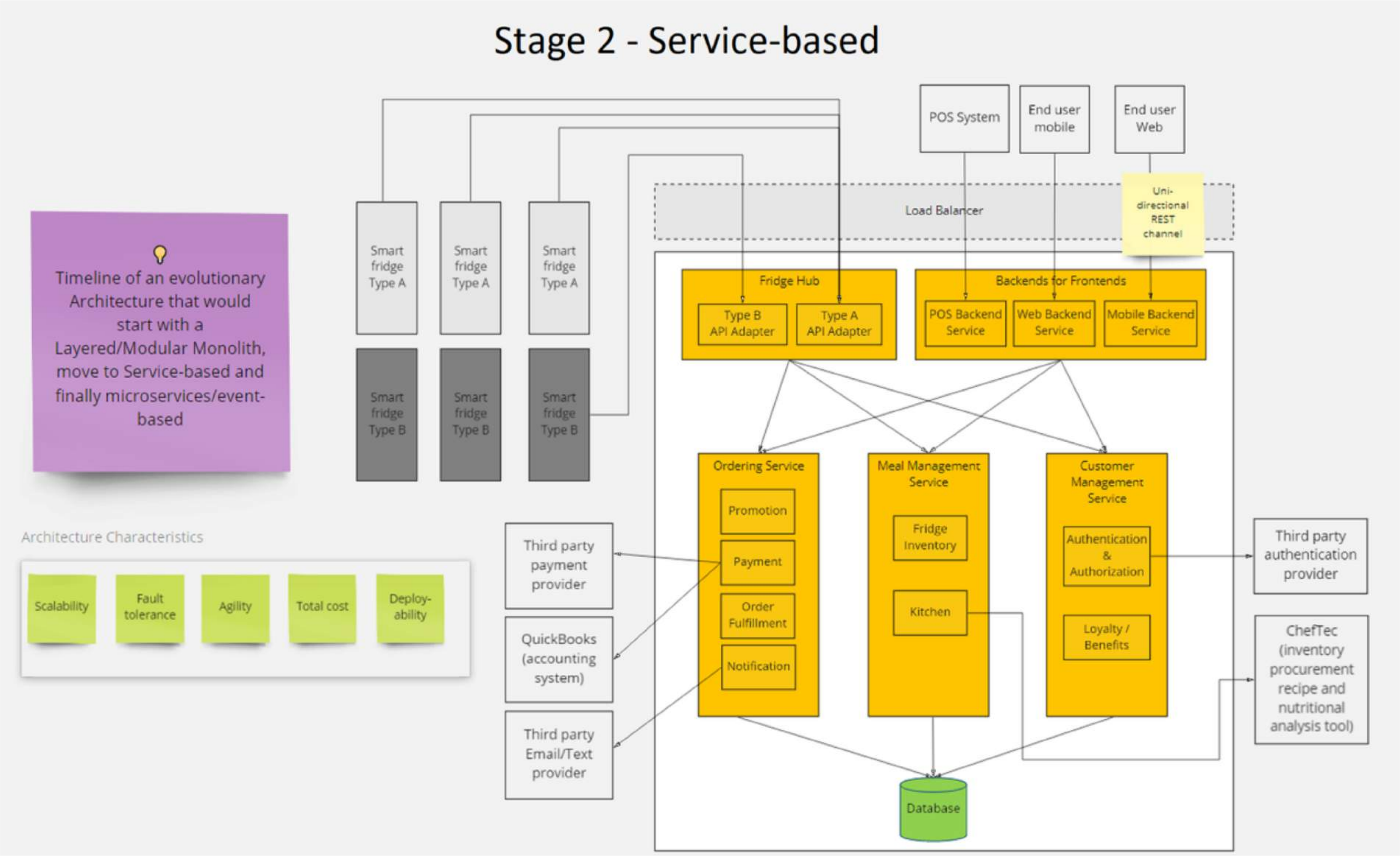


Stage 1 - Layered Monolith

Timeline of an evolutionary Architecture that would start with a Layered/Modular Monolith, move to Service-based and finally microservices/event-based

Judges Criteria

overall systems architecture

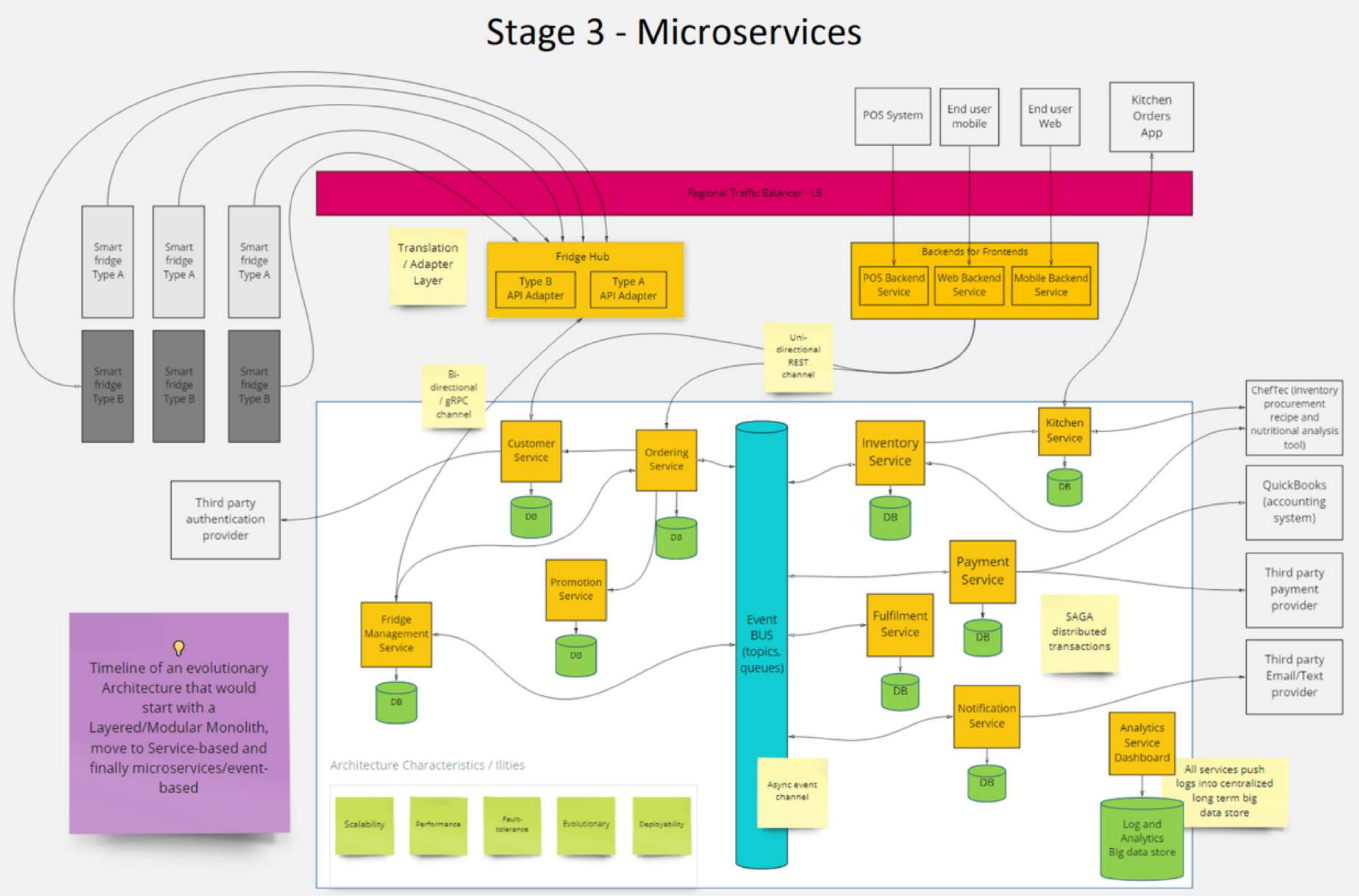


Stage 2 - Service-based

Timeline of an evolutionary Architecture that would start with a Layered/Modular Monolith, move to Service-based and finally microservices/event-based

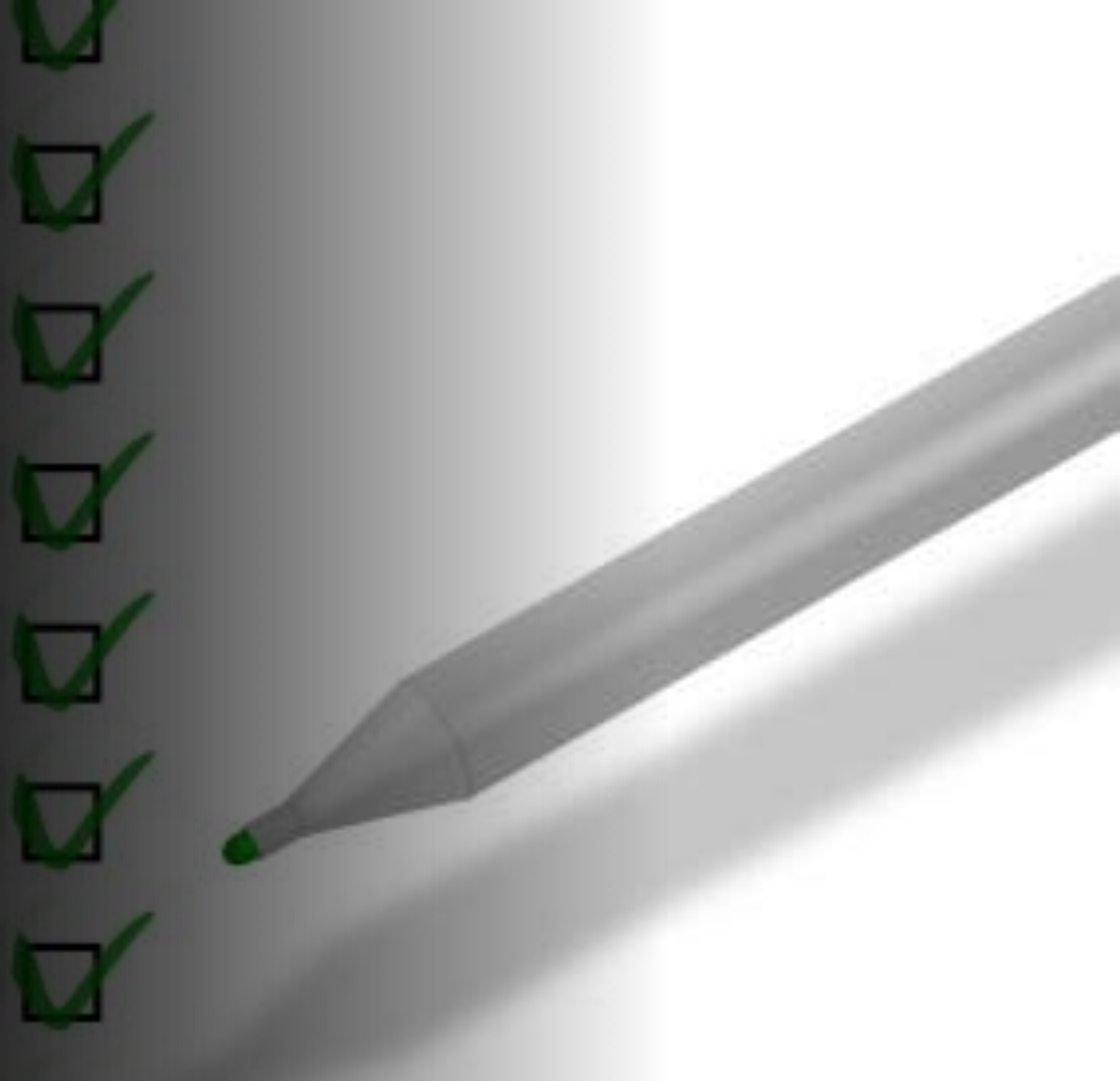
Judges Criteria

overall systems architecture



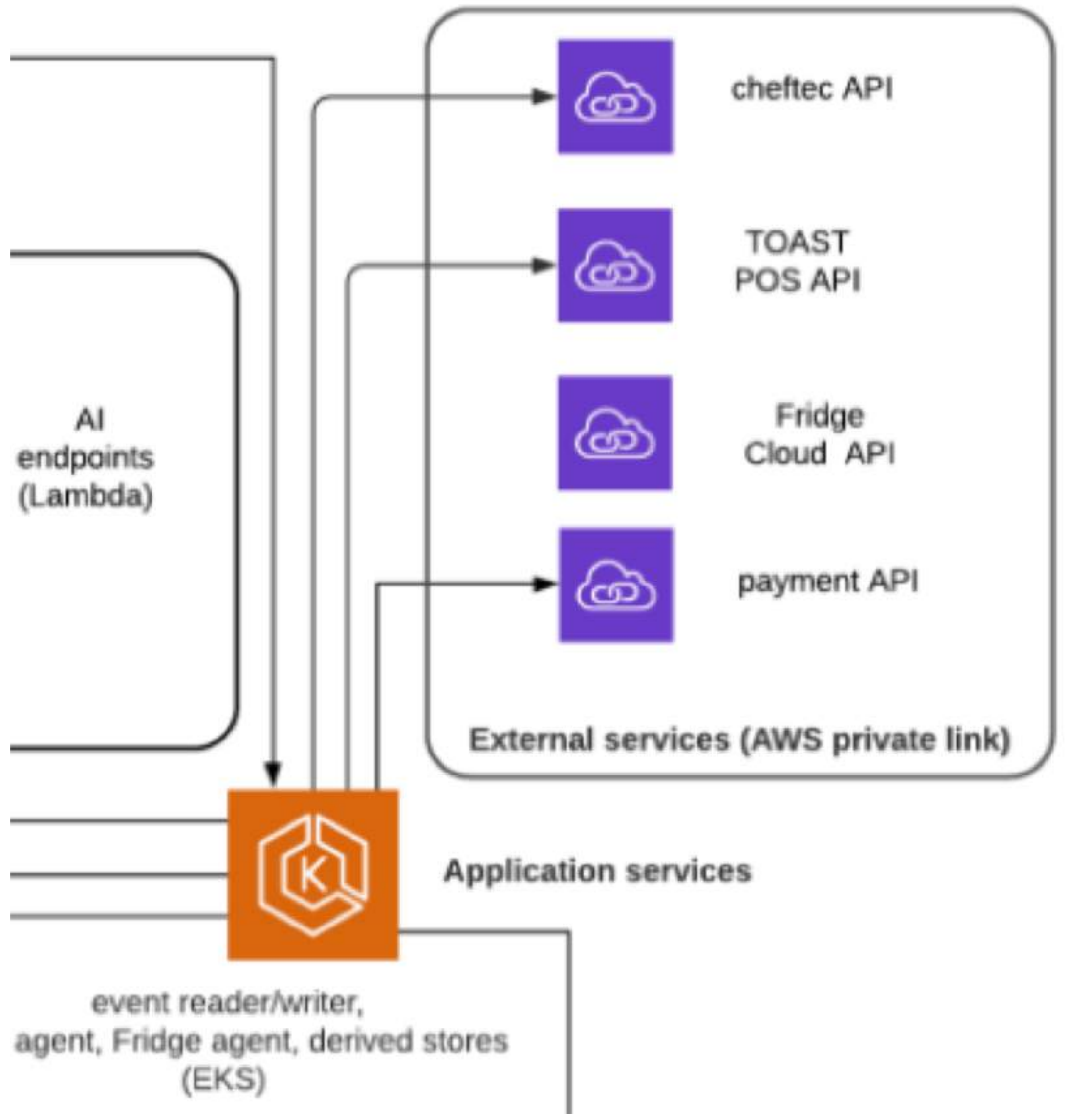
Timeline of an evolutionary Architecture that would start with a Layered/Modular Monolith, move to Service-based and finally microservices/event-based

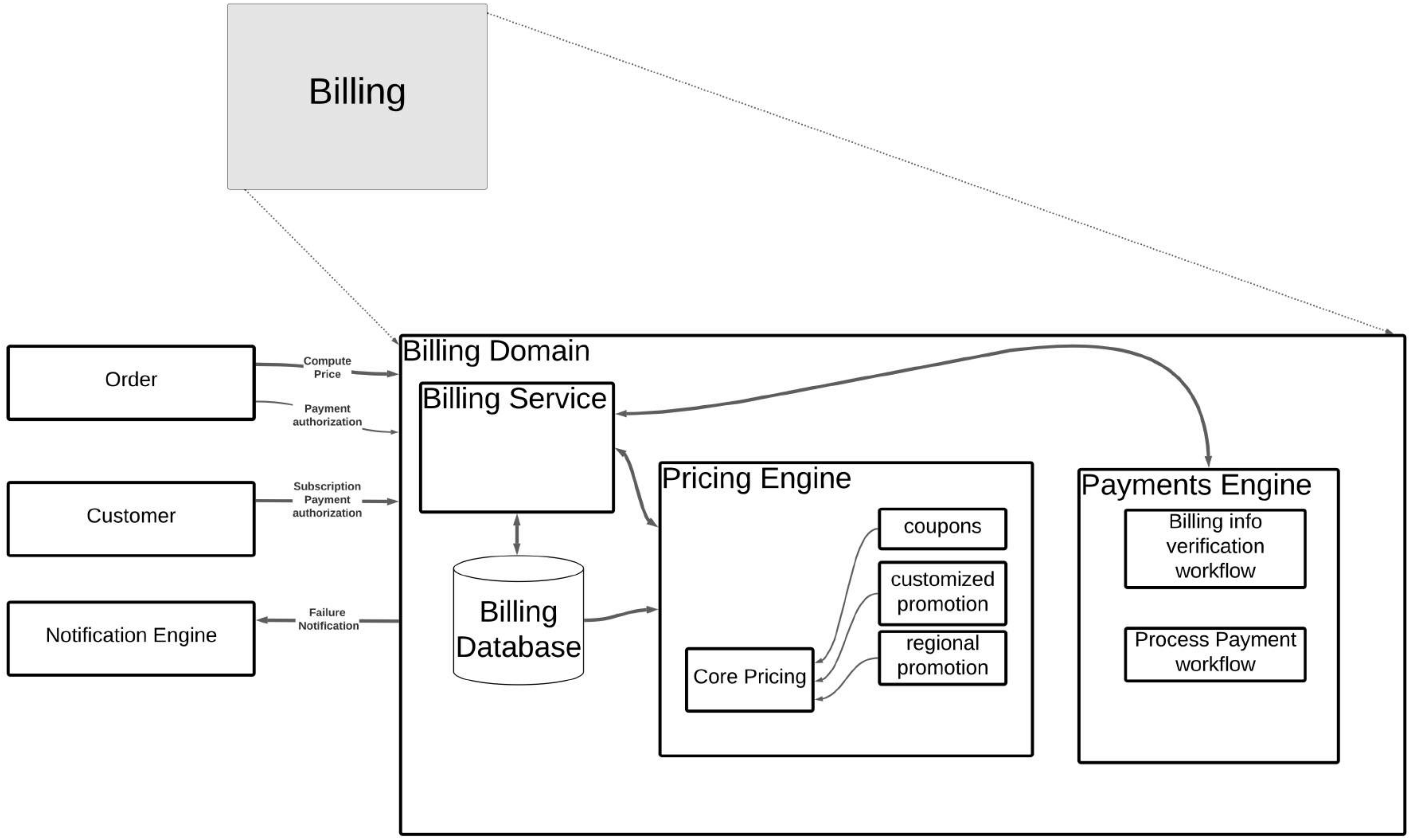
Integration architecture for required 3rd-party systems



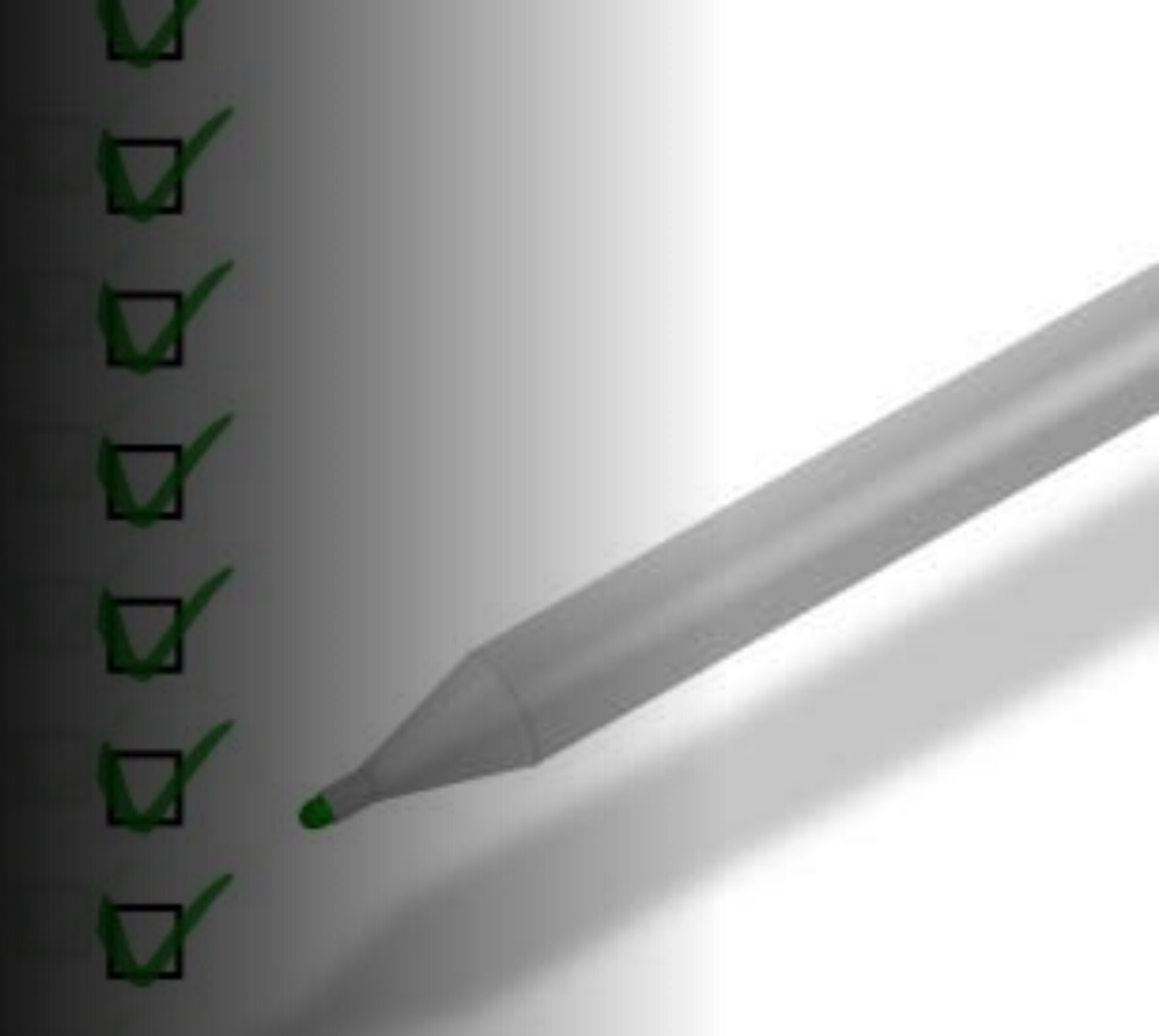
Judges Criteria

integration architecture for required third-party systems


cheftec API
TOAST POS API
Fridge Cloud API
payment API
External services (AWS private link)
AI endpoints (Lambda)
Application services
event reader/writer, agent, Fridge agent, derived stores (EKS)



Architecture decision records – documentation and justification



Judges Criteria

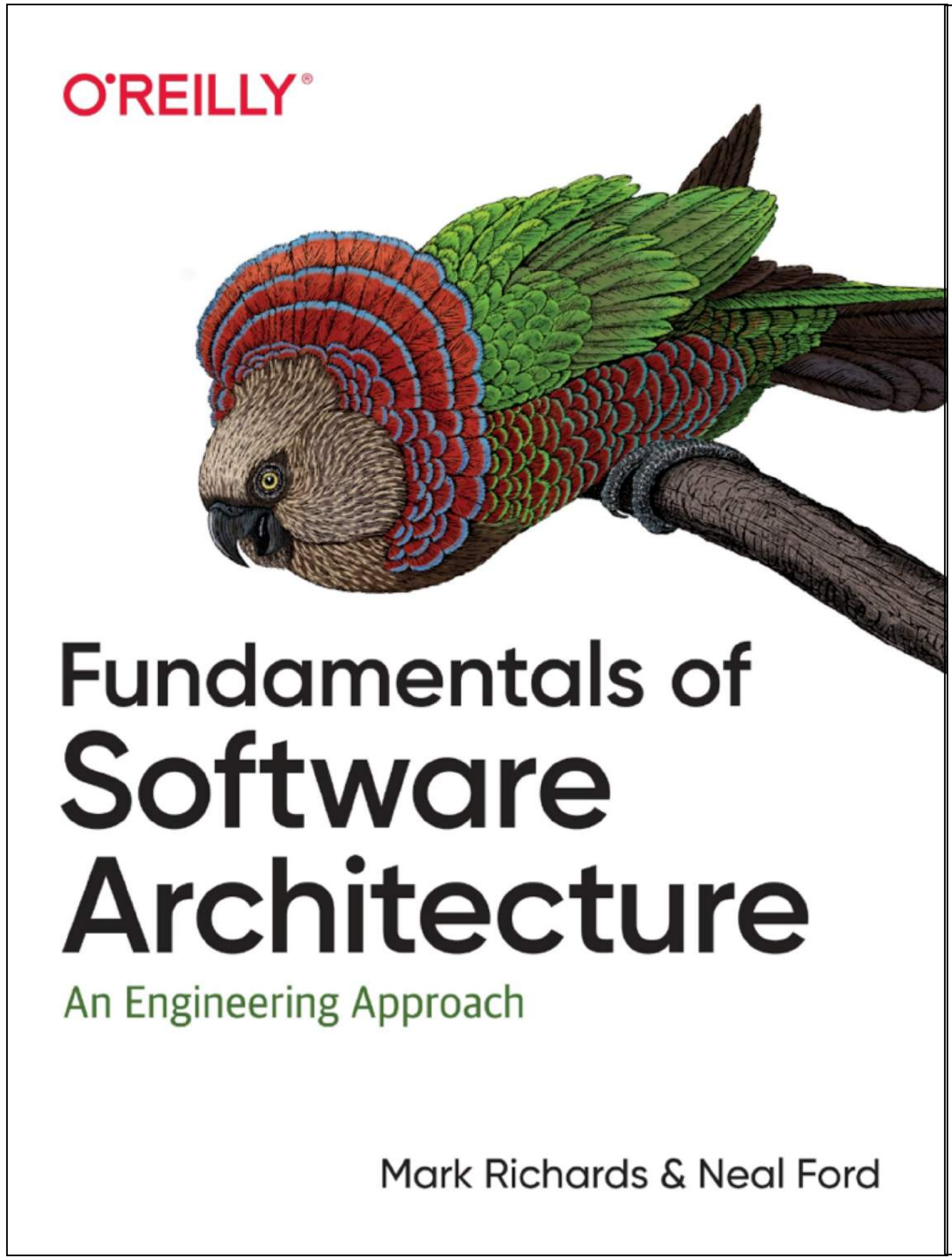
architecture decision records - documentation and justification

"We will keep a collection of records for *architecturally significant decisions*: those that affect the structure, non-functional characteristics, dependencies, interfaces, or construction techniques."

- Michael Nygard

Judges Criteria

architecture decision records - documentation and justification



O'REILLY

Fundamentals of Software Architecture

An Engineering Approach

Mark Richards & Neal Ford

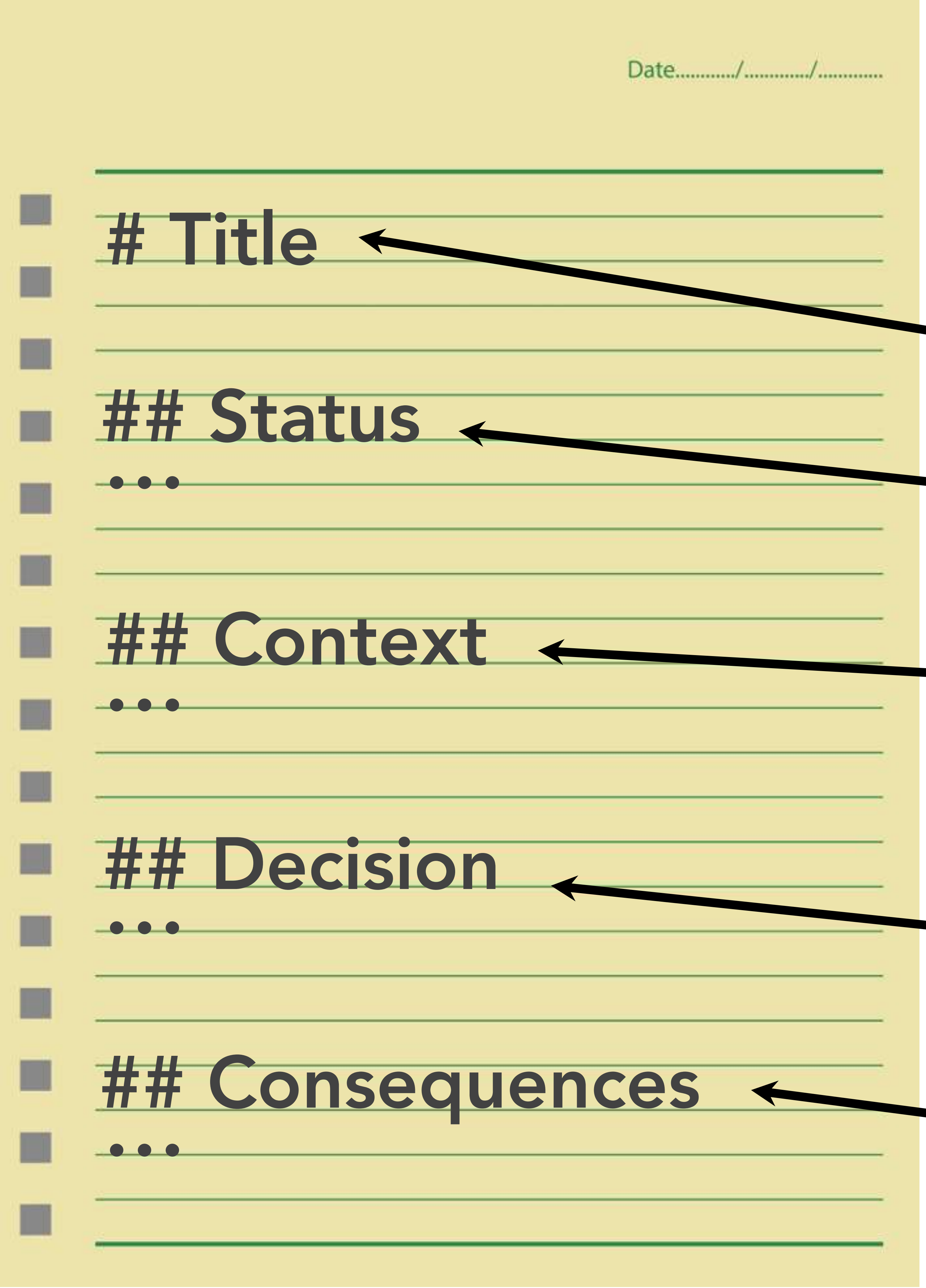
Second Law of Software Architecture

"*Why* is more important than *how*"

?

Judges Criteria

architecture decision records - documentation and justification



```
Date........../........../..........

# Title

## Status
•••

## Context
•••

## Decision
•••

## Consequences
•••
```

short text file; 1-2 pages long, one file per decision
markdown, textile, asciidoc, plaintext, etc.

short noun phrase

proposed, accepted, superseded

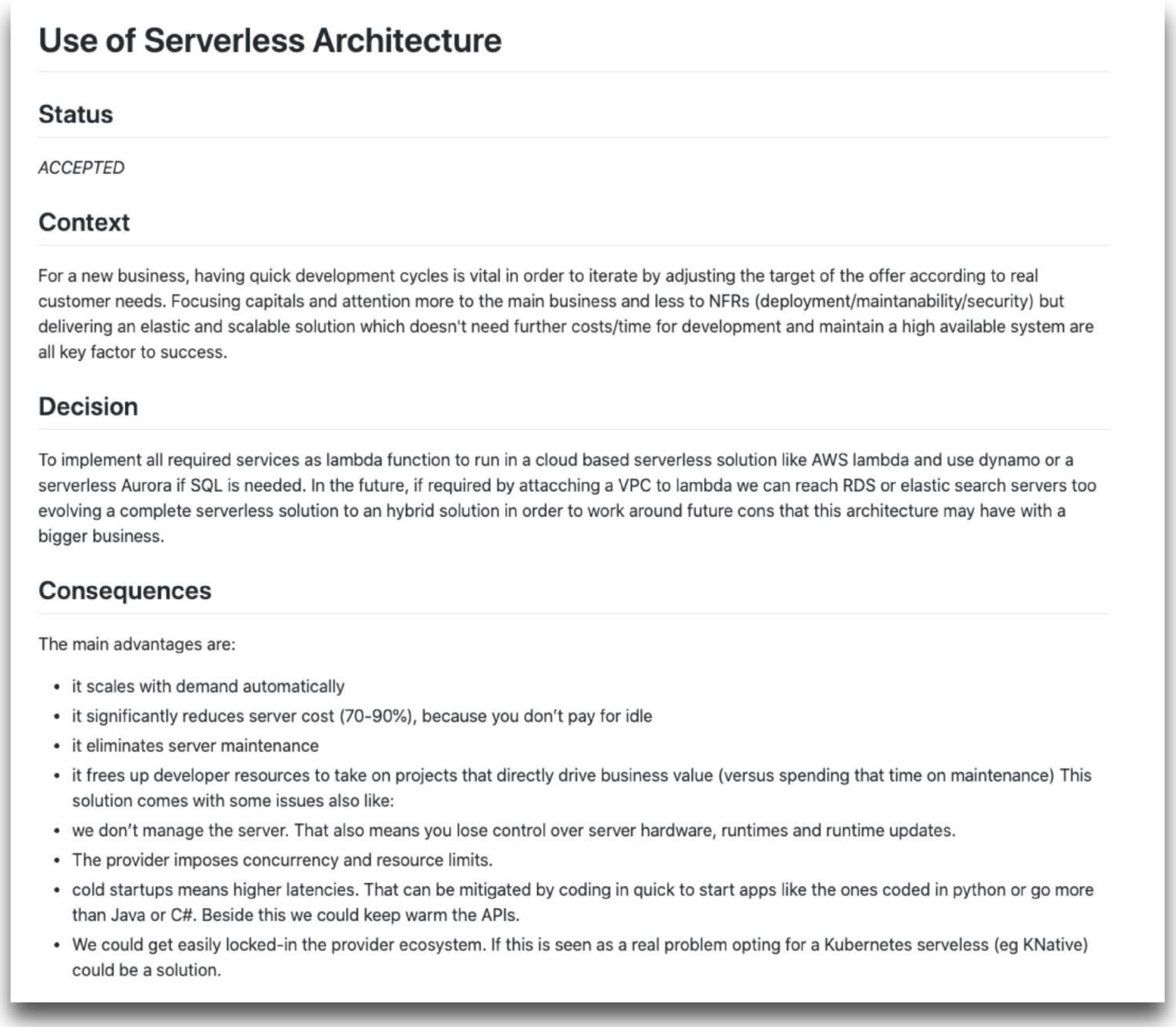
description of the problem and alternative solutions available (documentation)

justification (the "why")

tradeoffs and impact of decision

Judges Criteria

architecture decision records - documentation and justification



Use of Serverless Architecture

Status

ACCEPTED

Context

For a new business, having quick development cycles is vital in order to iterate by adjusting the target of the offer according to real customer needs. Focusing capitals and attention more to the main business and less to NFRs (deployment/maintanability/security) but delivering an elastic and scalable solution which doesn't need further costs/time for development and maintain a high available system are all key factor to success.

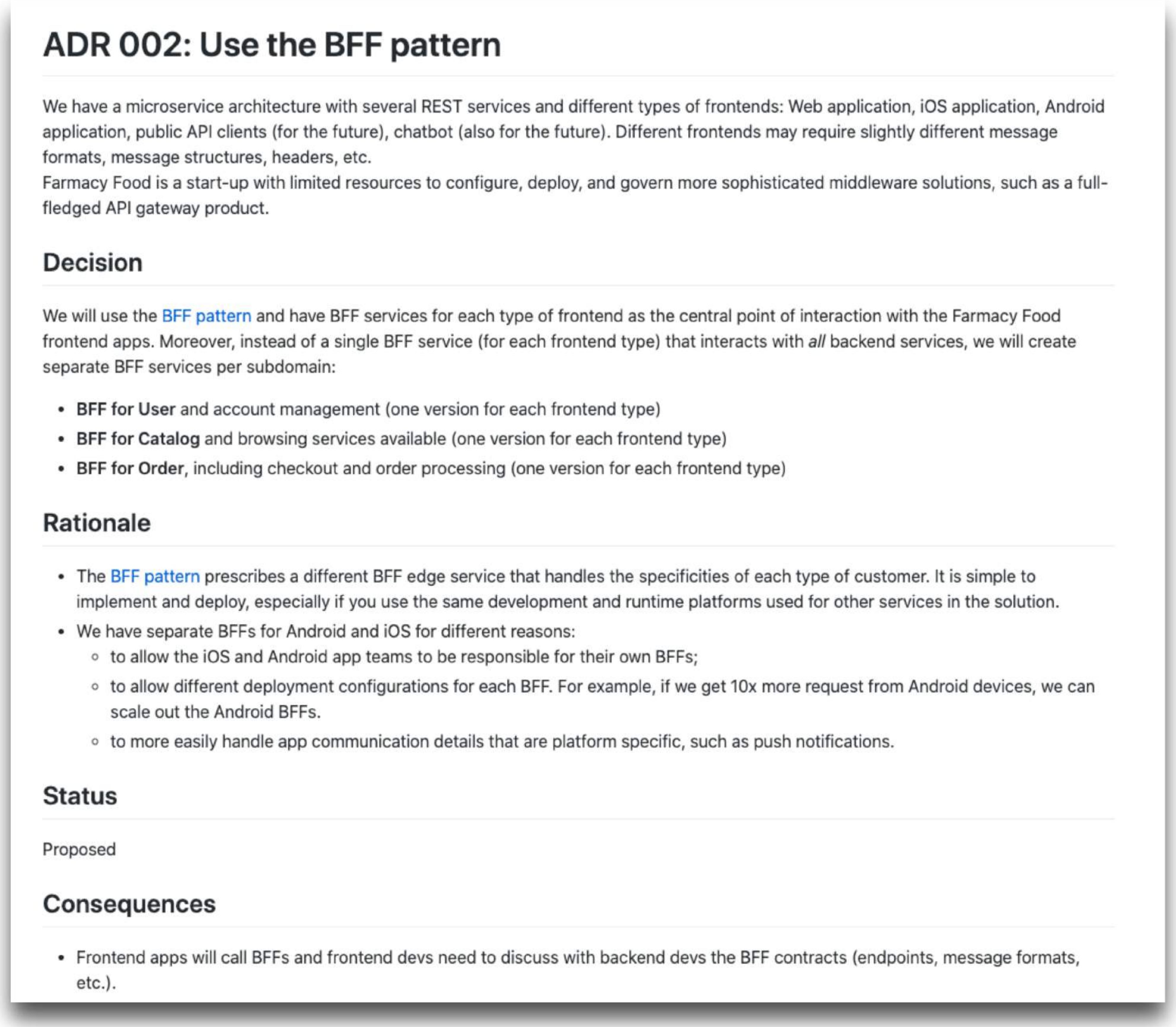
Decision

To implement all required services as lambda function to run in a cloud based serverless solution like AWS lambda and use dynamo or a serverless Aurora if SQL is needed. In the future, if required by attaching a VPC to lambda we can reach RDS or elastic search servers too evolving a complete serverless solution to an hybrid solution in order to work around future cons that this architecture may have with a bigger business.

Consequences

The main advantages are:

- it scales with demand automatically
- it significantly reduces server cost (70-90%), because you don't pay for idle
- it eliminates server maintenance
- it frees up developer resources to take on projects that directly drive business value (versus spending that time on maintenance) This solution comes with some issues also like:
- we don't manage the server. That also means you lose control over server hardware, runtimes and runtime updates.
- The provider imposes concurrency and resource limits.
- cold startups means higher latencies. That can be mitigated by coding in quick to start apps like the ones coded in python or go more than Java or C#. Beside this we could keep warm the APIs.
- We could get easily locked-in the provider ecosystem. If this is seen as a real problem opting for a Kubernetes serveless (eg KNative) could be a solution.



ADR 002: Use the BFF pattern

We have a microservice architecture with several REST services and different types of frontends: Web application, iOS application, Android application, public API clients (for the future), chatbot (also for the future). Different frontends may require slightly different message formats, message structures, headers, etc.
Farmacy Food is a start-up with limited resources to configure, deploy, and govern more sophisticated middleware solutions, such as a full-fledged API gateway product.

Decision

We will use the BFF pattern and have BFF services for each type of frontend as the central point of interaction with the Farmacy Food frontend apps. Moreover, instead of a single BFF service (for each frontend type) that interacts with *all* backend services, we will create separate BFF services per subdomain:

- **BFF for User** and account management (one version for each frontend type)
- **BFF for Catalog** and browsing services available (one version for each frontend type)
- **BFF for Order**, including checkout and order processing (one version for each frontend type)

Rationale

- The BFF pattern prescribes a different BFF edge service that handles the specificities of each type of customer. It is simple to implement and deploy, especially if you use the same development and runtime platforms used for other services in the solution.
- We have separate BFFs for Android and iOS for different reasons:
  - to allow the iOS and Android app teams to be responsible for their own BFFs;
  - to allow different deployment configurations for each BFF. For example, if we get 10x more request from Android devices, we can scale out the Android BFFs.
  - to more easily handle app communication details that are platform specific, such as push notifications.

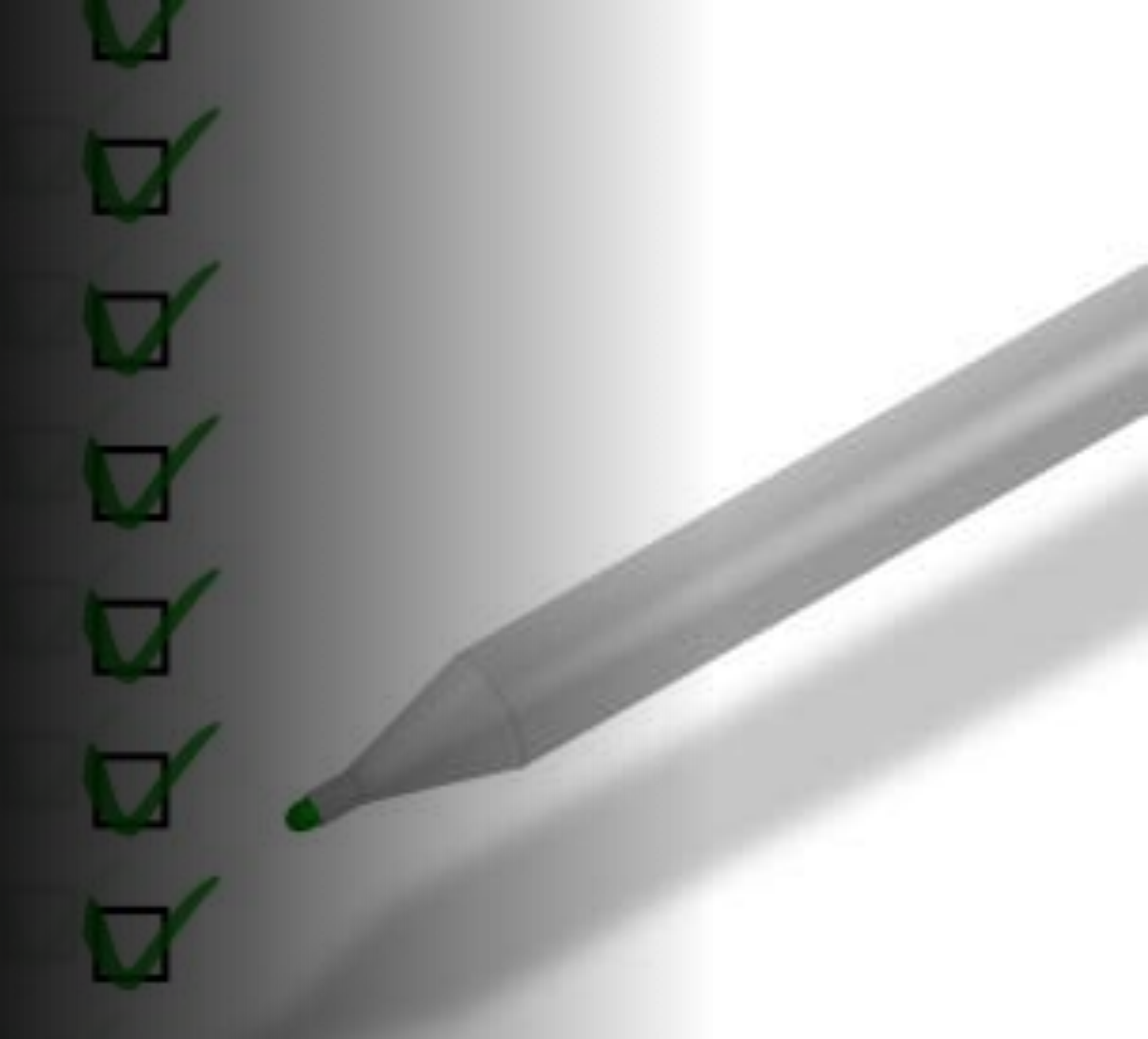
Status

Proposed

Consequences

- Frontend apps will call BFFs and frontend devs need to discuss with backend devs the BFF contracts (endpoints, message formats, etc.).

The semi-finalists…



The Semi-Finalists

Super Kings

Miyagi's Little Forests

The Jedis

Hananoyama

ArchColider

Hey Dragon

Jiakaturi

Pacman

SelfDrivenTeam

daVinci

O'REILLY®

LIVE ONLINE TRAINING

Architecture Katas Semi-Finalists

Sarah Taraporewalla

Principal Technologist Director
Enterprise Modernisation, Platforms & Cloud
ThoughtWorks
@sarahtarap
https://sarahtaraporewalla.com/

Luca Mezzalira

VP of Architecture at DAZN
International Speaker
Published Author
@lucamezzalira
https://lucamezzalira.com

Nathanial T. Schutta

Developer Advocate, VMWare
Architect as a Service
@ntschutta
http://www.ntschutta.io/

Mark Richards

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