Software architecture for developers

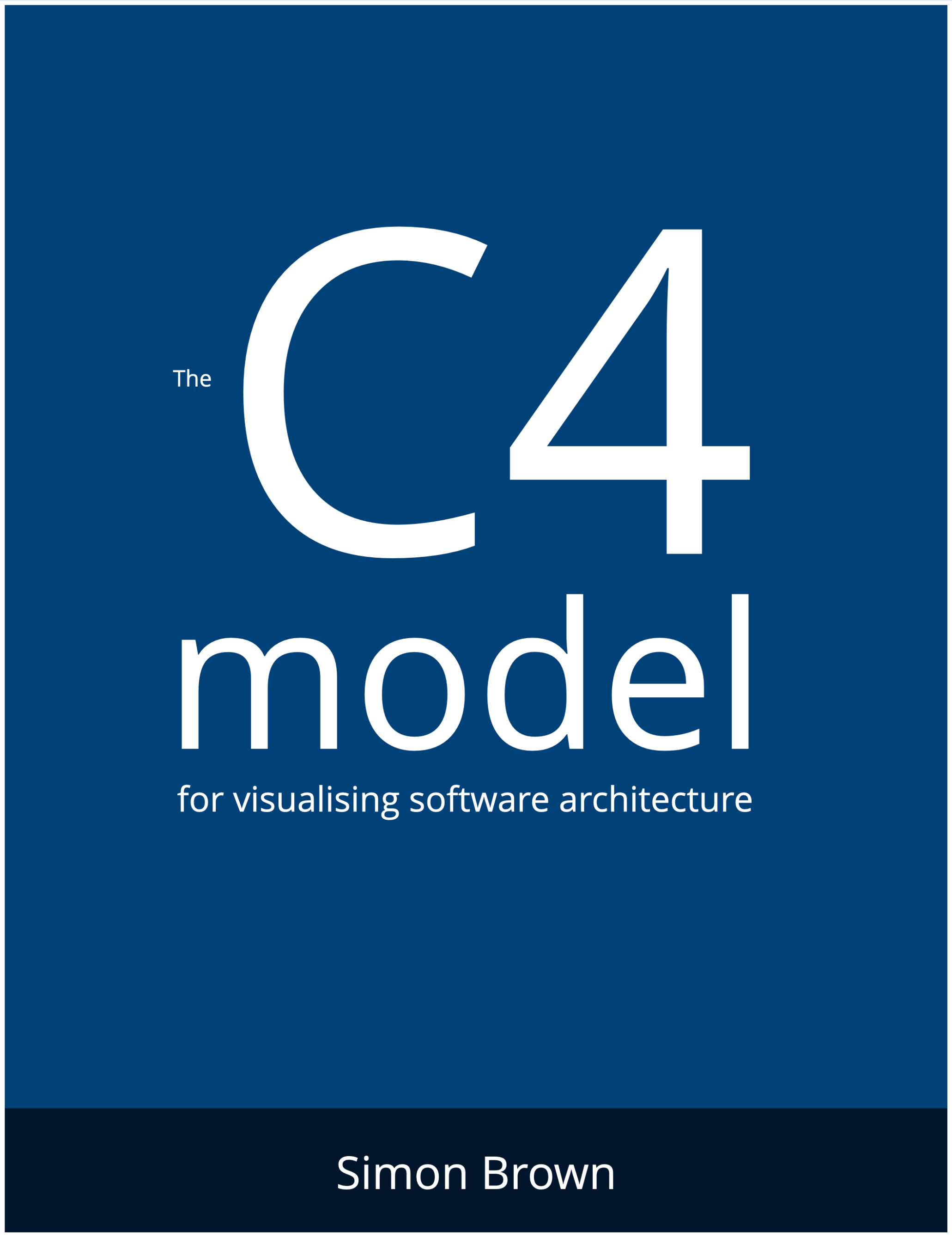
Simon Brown

Simon Brown

Independent consultant specialising in software architecture,
plus the creator of the C4 model and Structurizr

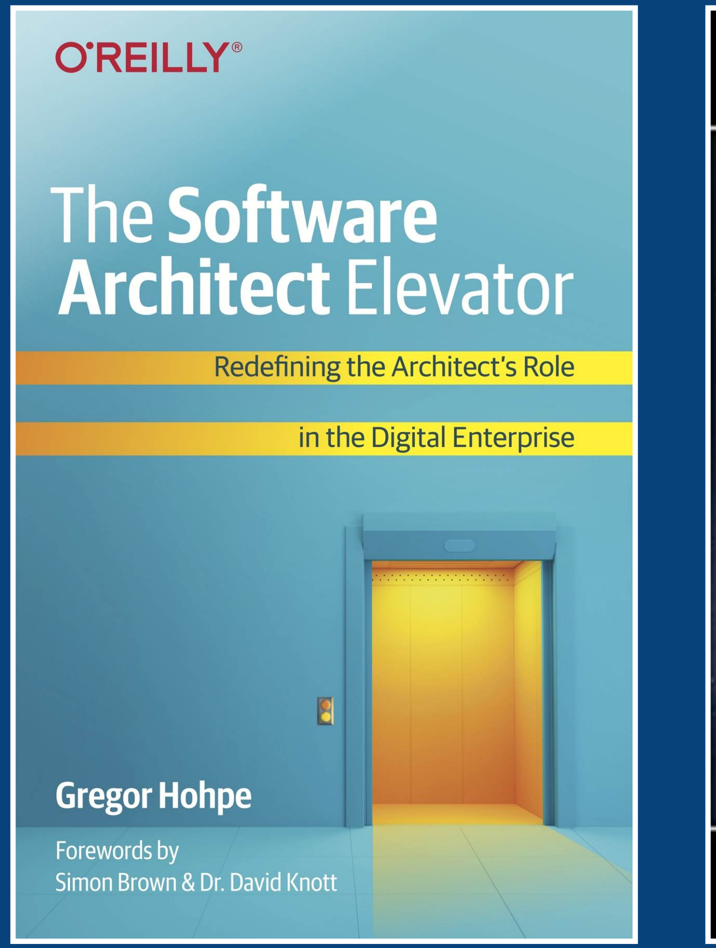
Software architecture for developers

Simon Brown


The C4 model

for visualising software architecture

Simon Brown

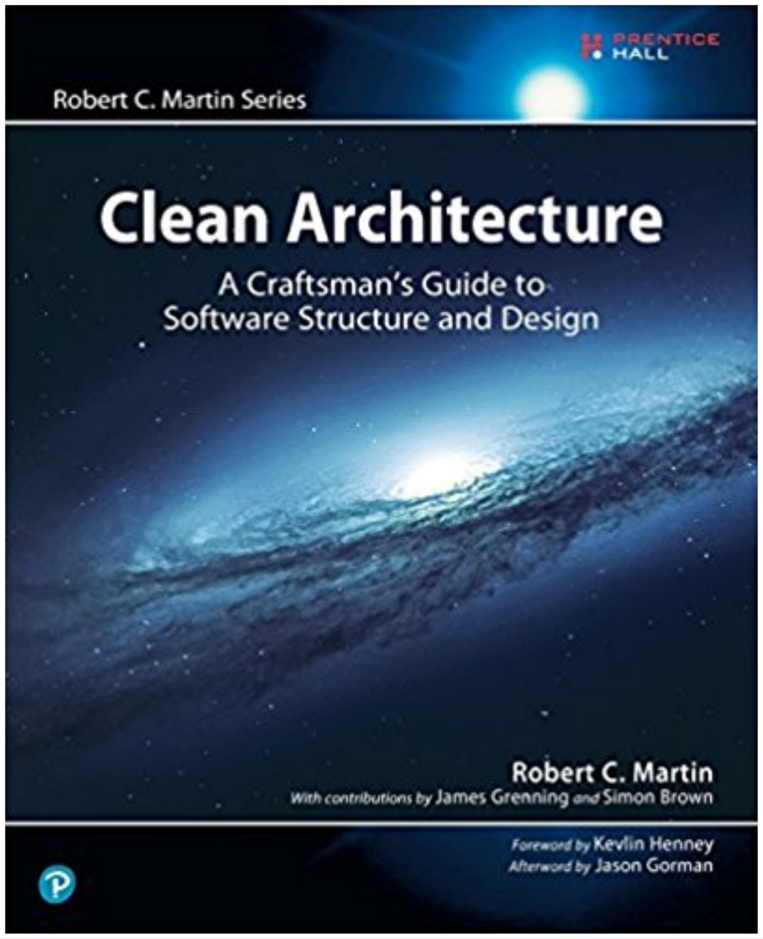

O'REILLY®

The Software Architect Elevator

Redefining the Architect's Role in the Digital Enterprise

Gregor Hohpe

Forewords by Simon Brown & Dr. David Knott


Robert C. Martin Series

PRENTICE HALL

Clean Architecture

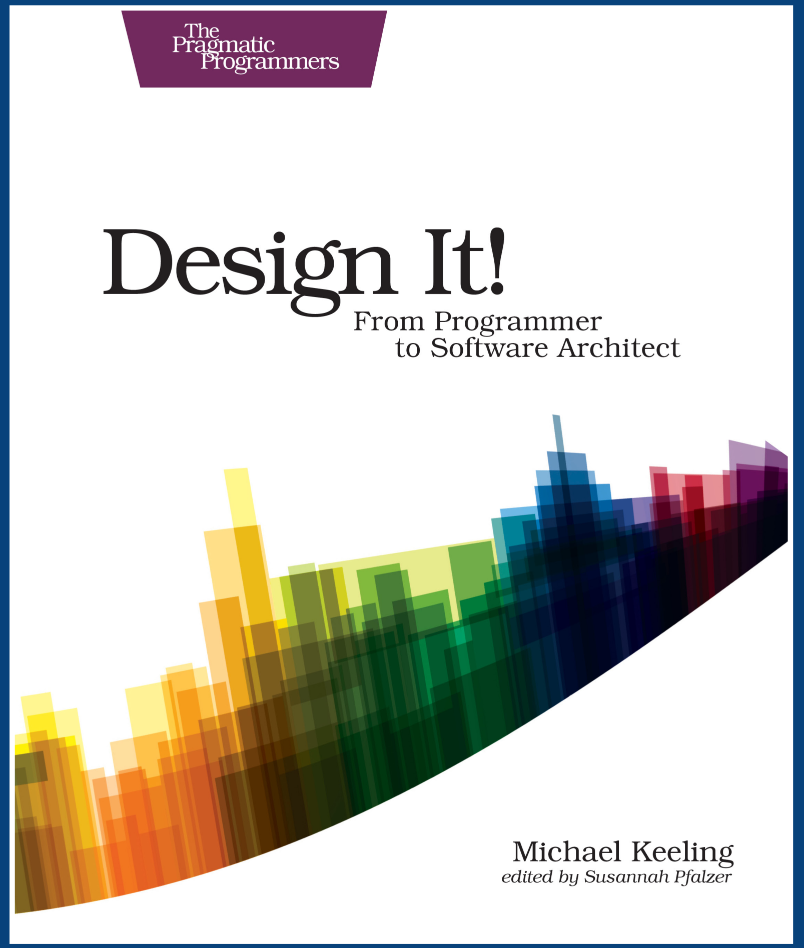
A Craftsman's Guide to Software Structure and Design

Robert C. Martin

With contributions by James Grenning and Simon Brown

Foreword by Kevlin Henney

Afterword by Jason Gorman


The Pragmatic Programmers

Design It!

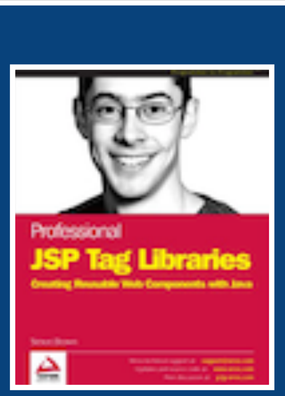
From Programmer to Software Architect

Michael Keeling

edited by Susannah Pfalzer

PROFESSIONAL JSP 2nd Edition

professional Java Servlets 2.3


Professional JSP Tag Libraries

Creating Reusable Web Components with Java

Simon Brown

Simon Brown, Sam Dalton, Dan Jepp, Dave Johnson, Sing Li, and Matt Raible

Pro JSP

THIRD EDITION

Pro JSP 2

FOURTH EDITION

Simon Brown, Sam Dalton, Daniel Jepp, Dave Johnson, Sing Li, and Matt Raible

Apress®

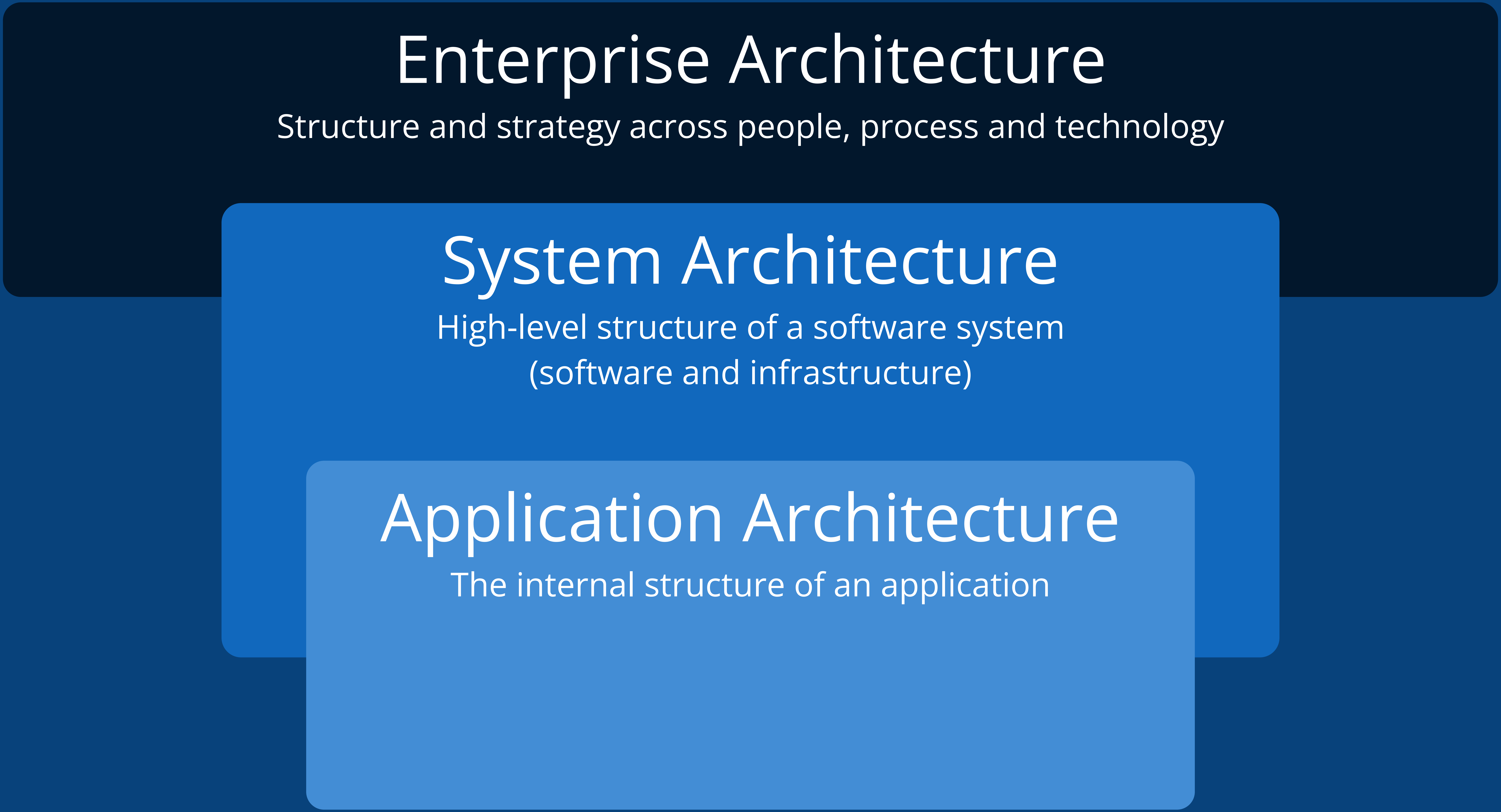
What is software architecture?

Structure

The definition of software in terms
of its building blocks and their interactions

Vision

The process of architecting;
making decisions based upon business goals,
requirements and constraints,
plus being able to communicate this to a team



Enterprise Architecture

Structure and strategy across people, process and technology

System Architecture

High-level structure of a software system
(software and infrastructure)

Application Architecture

The internal structure of an application

> As a noun, design is the named structure or behaviour of a system … a design thus represents one point in a potential decision space.

Grady Booch

> All architecture is design, but
> **not all design is architecture**.

Grady Booch

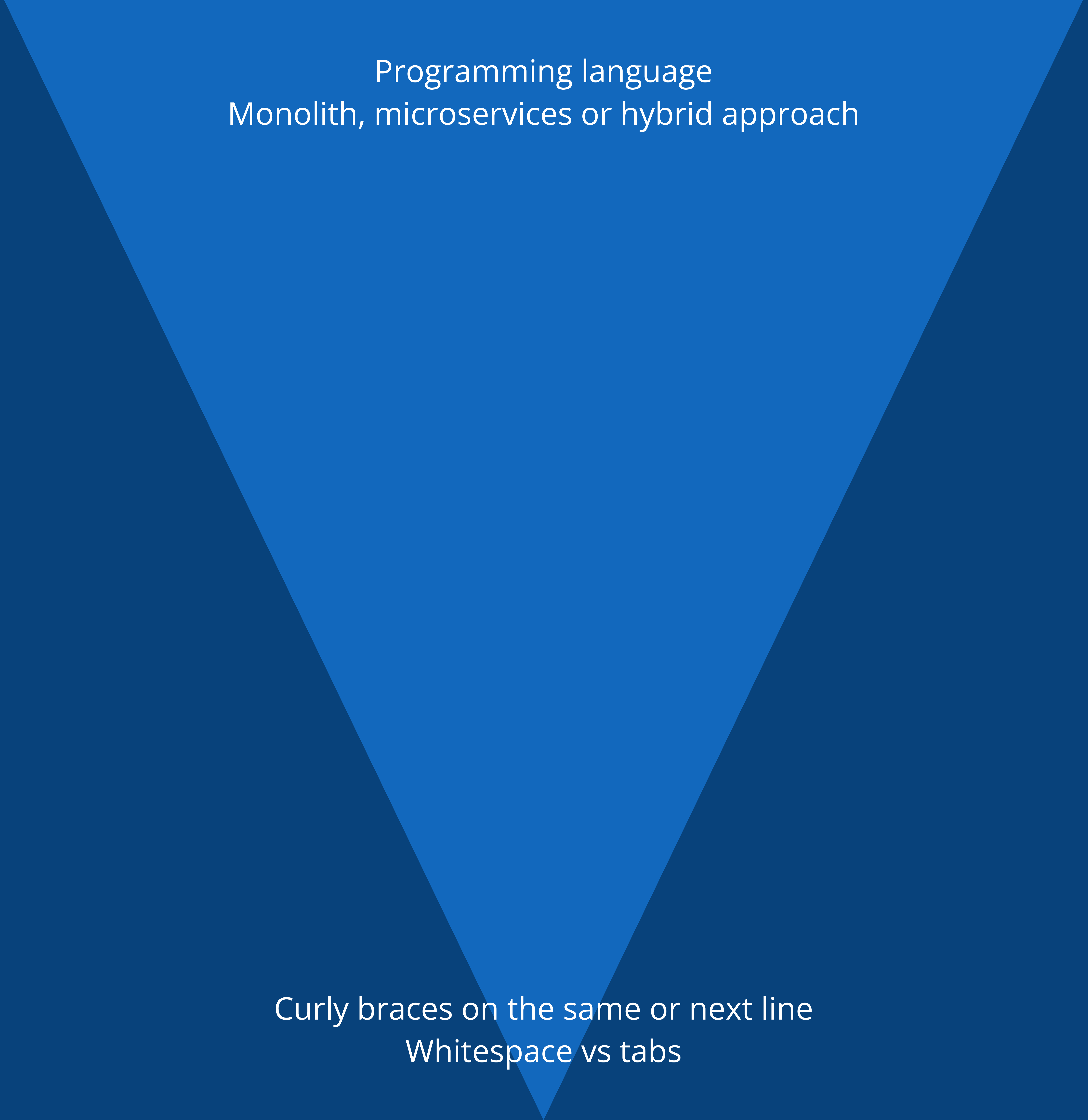
> Architecture represents the **significant decisions**, where significance is measured by **cost of change**.
>
> Grady Booch

As architects, we define
the **significant decisions**

Architecture

Design

Implementation



Programming language
Monolith, microservices or hybrid approach

Curly braces on the same or next line
Whitespace vs tabs

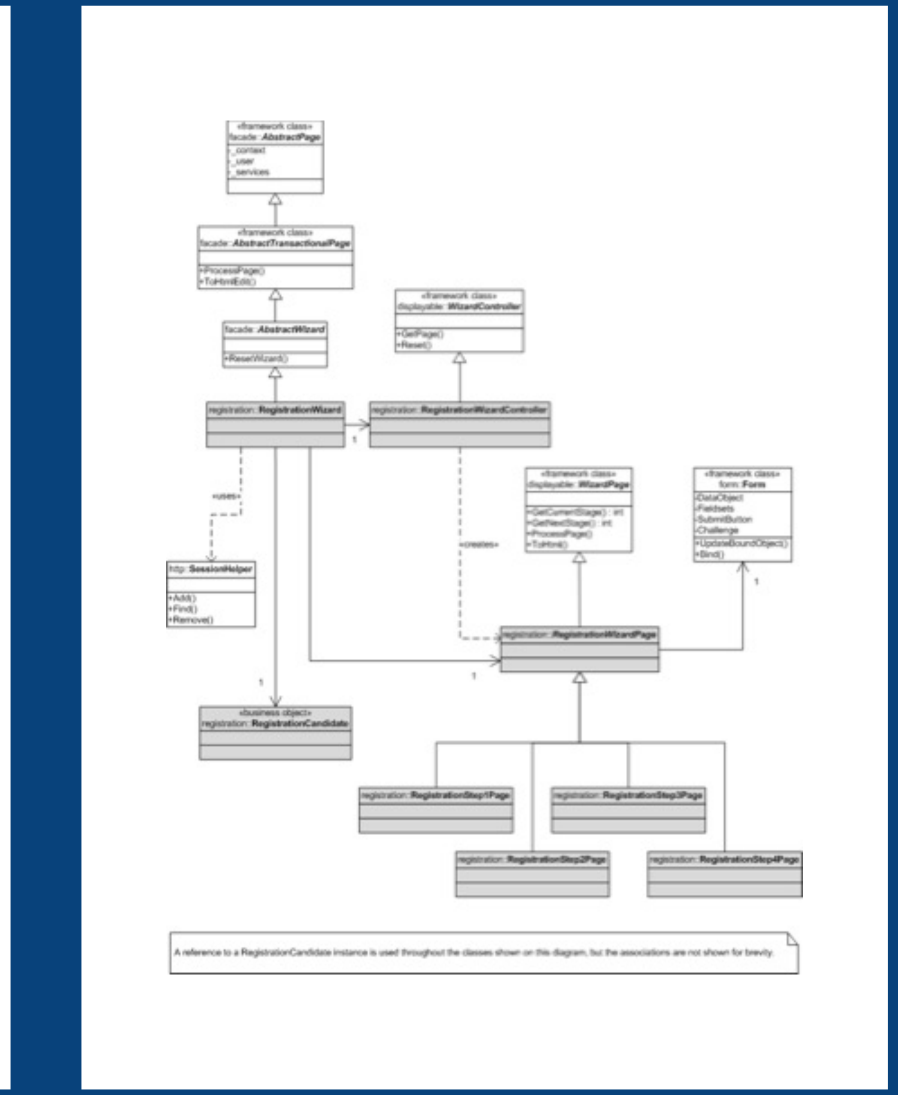
What happens if a software development team **doesn’t think about architecture**?

Chaos

Big ball of mud, spaghetti code, inconsistent approaches to solving the same problems, quality attributes are ignored, deployment problems, maintenance issues, etc

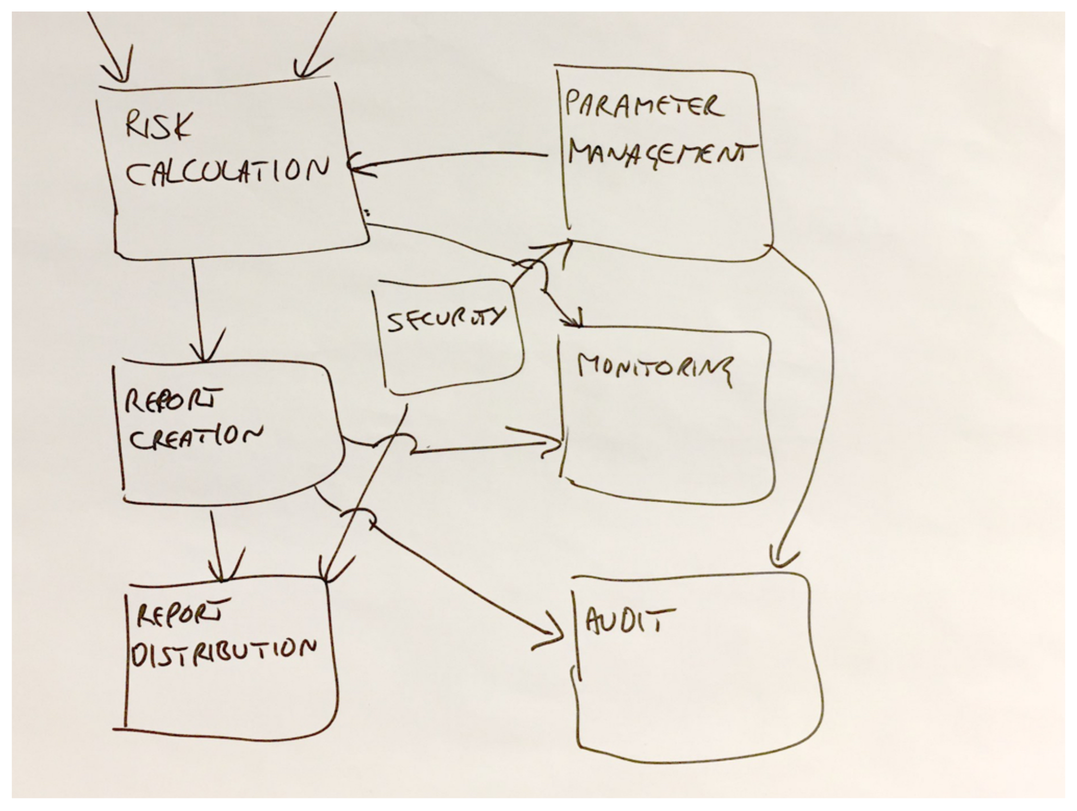
Big design up front

Software Architecture Document



A reference to a RegistrationCandidate instance is used throughout the classes shown on this diagram, but the associations are not shown for brevity.

vs



No design up front

> Big design up front is dumb. Doing no design up front is even dumber.

Dave Thomas

Software architecture helps us avoid chaos

Architectural drivers

Requirements

drive architecture

(use cases, user stories, features, etc)

Requirement

"a thing that is needed or wanted"

(this includes experiments and hypotheses too)

Don’t start designing software
if you have no inputs

Quality attributes

(also known as non-functional requirements,
cross-cutting concerns, service-level agreements, etc)

What **quality attributes** might be relevant for the ”Financial Risk System”?

- Performance
- Scalability
- Availability
- Security
- Disaster Recovery
- Accessibility
- Monitoring
- Management
- Audit
- Flexibility
- Extensibility
- Maintainability
- Interoperability
- Legal
- Regulatory
- Compliance
- i18n
- L10n

Create a **checklist** of quality attributes you regularly encounter

Understand how to **capture**, **refine** and **challenge** quality attributes

Software lives in the real world,
and the real world has

constraints

Typical constraints include
time and budget, technology,
people and skills, politics, etc

Constraints can **sometimes** be prioritised

Principles

are selected by the team

Development principles include coding conventions, naming guidelines, testing approaches, review practices, etc

Architecture and design principles typically relate to modularity or crosscutting concerns

(architectural layering, separation of concerns, stateless vs stateful, rich vs anaemic domain, security, error handling, logging, etc)

Ensure you have a good understanding of the requirements, quality attributes, constraints and principles to create

sufficient foundations

What about agile, and agility?

Agile is about moving fast, embracing change, releasing often, getting feedback, …

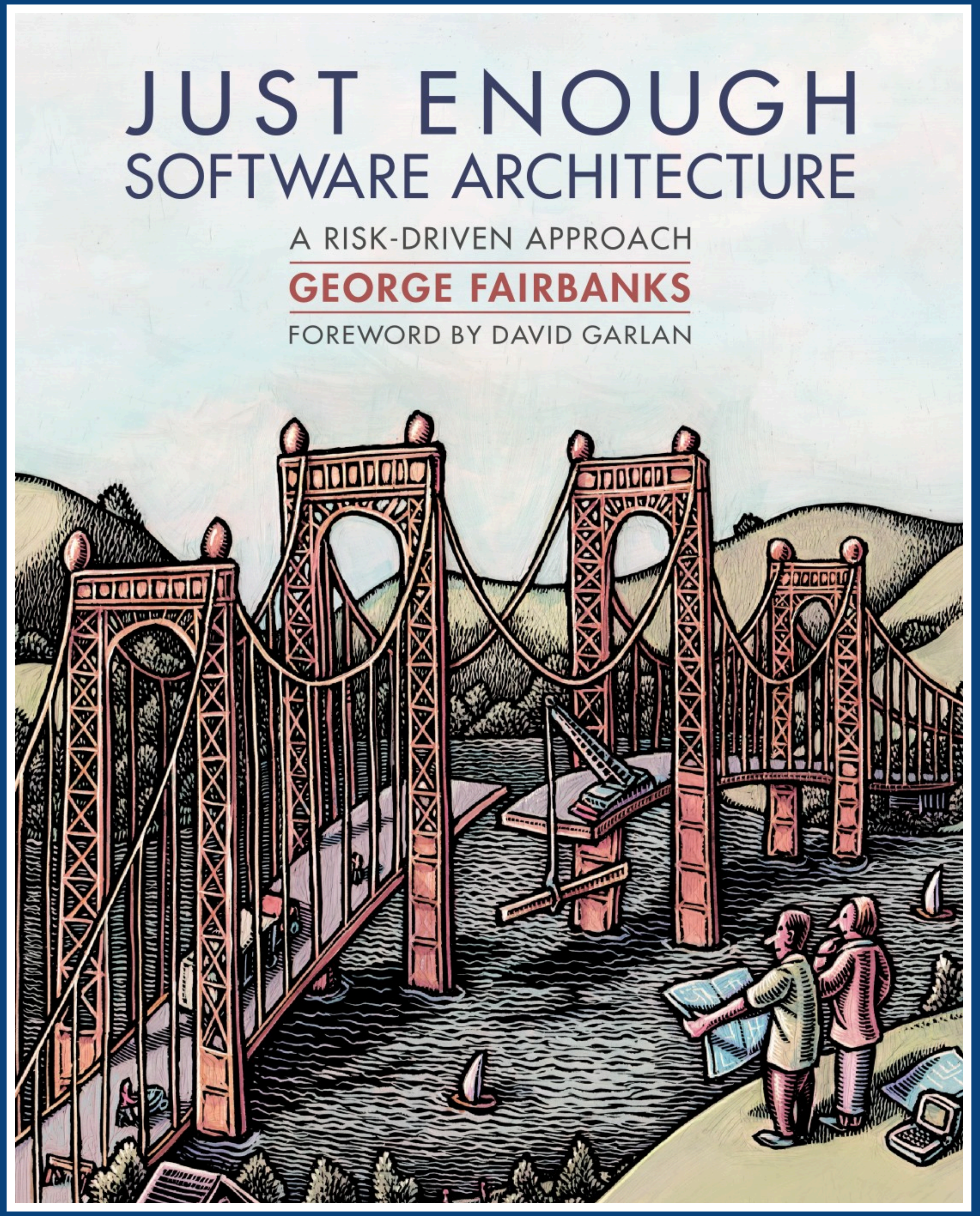
Agile is about a mindset of **continuous improvement**

Inspect and **adapt**

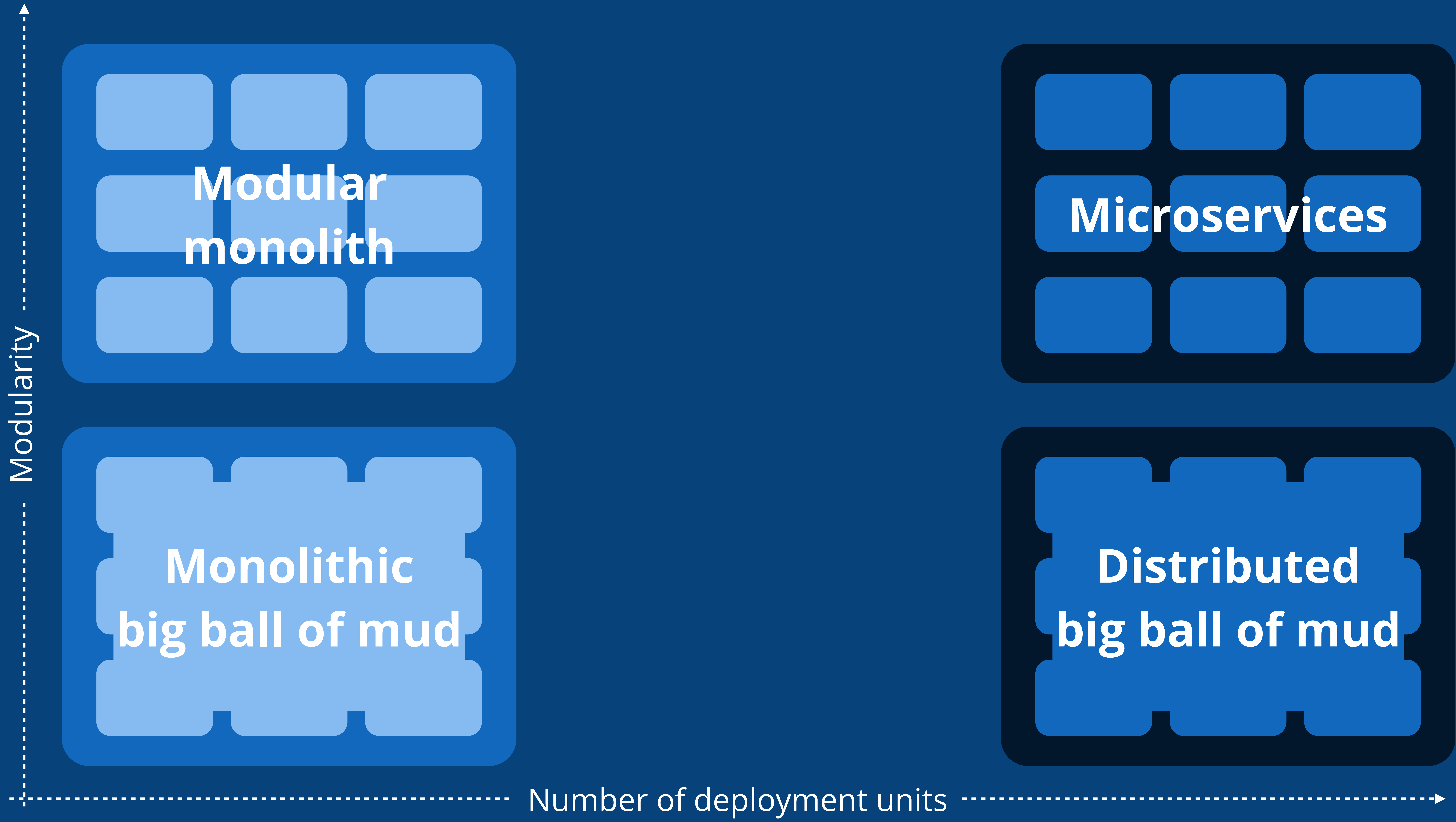
> Continuous attention to technical excellence and good design enhances agility.

Principle 9 of the Manifesto for Agile Software Development

A good architecture enables agility



A good architecture rarely happens through **architecture-indifferent design**

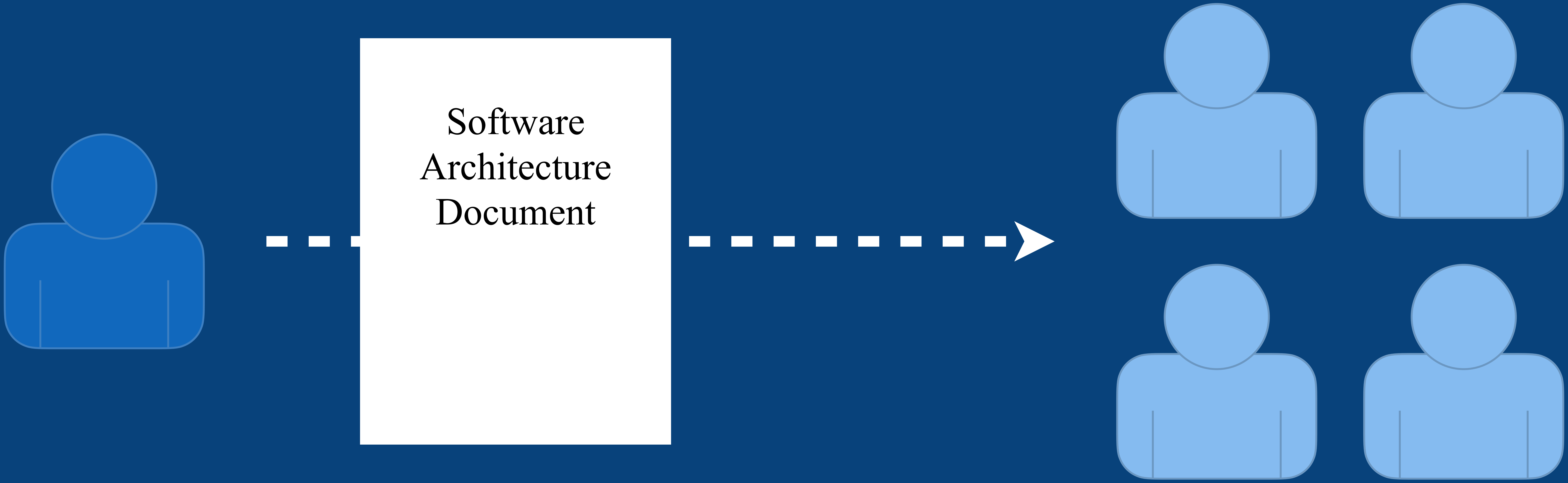

Modular monolith
Microservices
Monolithic big ball of mud
Distributed big ball of mud
Modularity
Number of deployment units

Agility is a

quality attribute

The software architecture role

Software development is not a relay sport



AaaS

Architecture as a Service

The software architecture role
is about the “**big picture**”
and, sometimes, this means
stepping away from the code

The software architecture role

(technical leadership, and responsible for the technical success of the project/product)

Architectural drivers

Understanding the goals; capturing, refining, and challenging the requirements and constraints.

Designing software

Creating the technical strategy, vision, alignment, and roadmap.

Technical risks

Identifying, mitigating and owning the technical risks to ensure that the architecture “works”.

Technical leadership

Continuous technical leadership and ownership of the architecture throughout the software delivery.

Quality assurance

Introduction and adherence to standards, guidelines, principles, etc plus management of technical debt.

“Software development teams don’t need architects”

Software development teams do need technical leadership

Every team needs technical leadership

Continuous technical leadership

(somebody needs to continuously steer the ship)

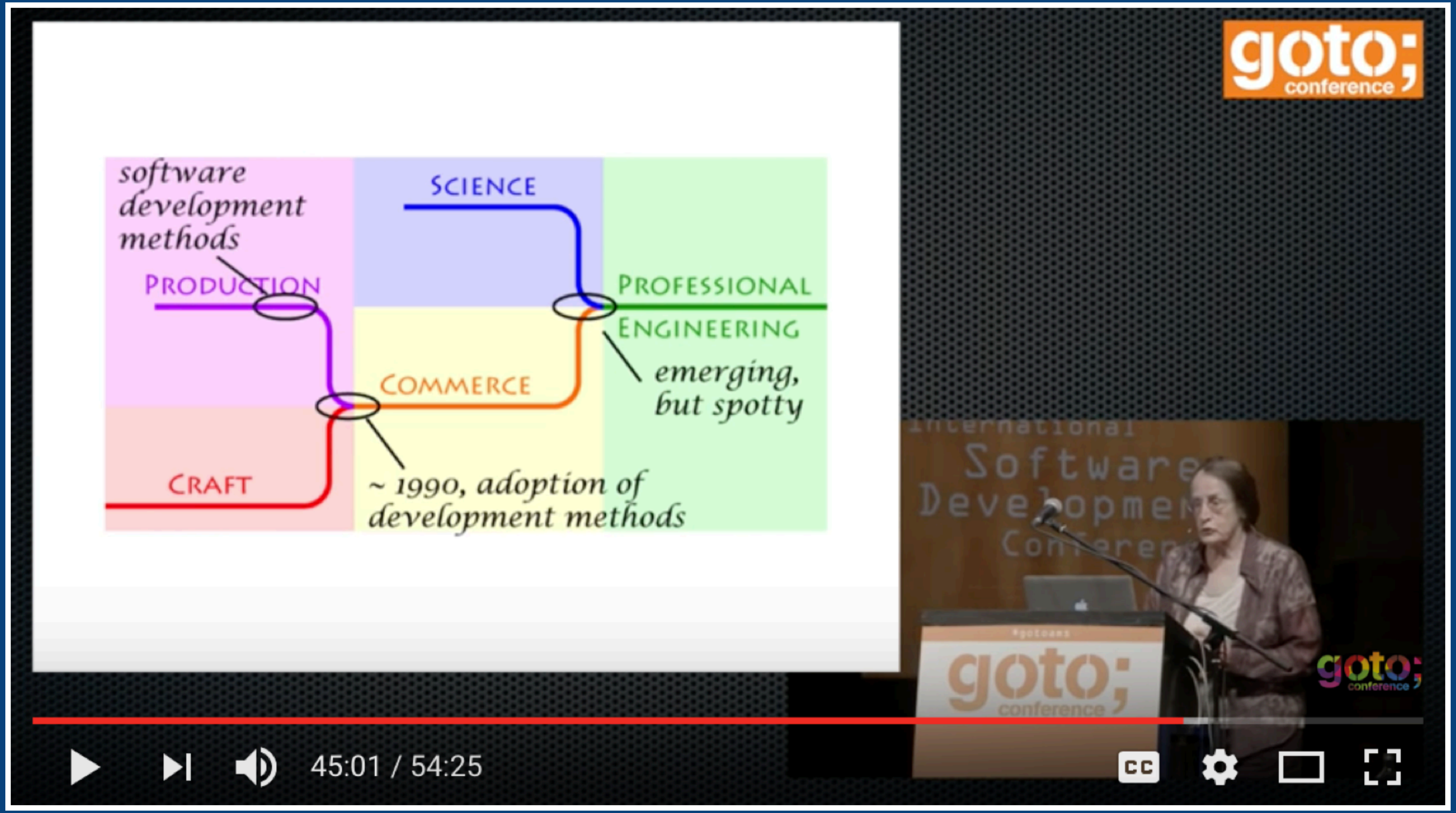
Should software architects write **code**?

Production code, prototypes, frameworks, foundations, code reviews, experimenting, etc

Don’t code all of the time!

There is often a tension between being “senior” and writing code…

Software architects
should be
master builders



software development methods

PRODUCTION

SCIENCE

PROFESSIONAL ENGINEERING

emerging, but spotty

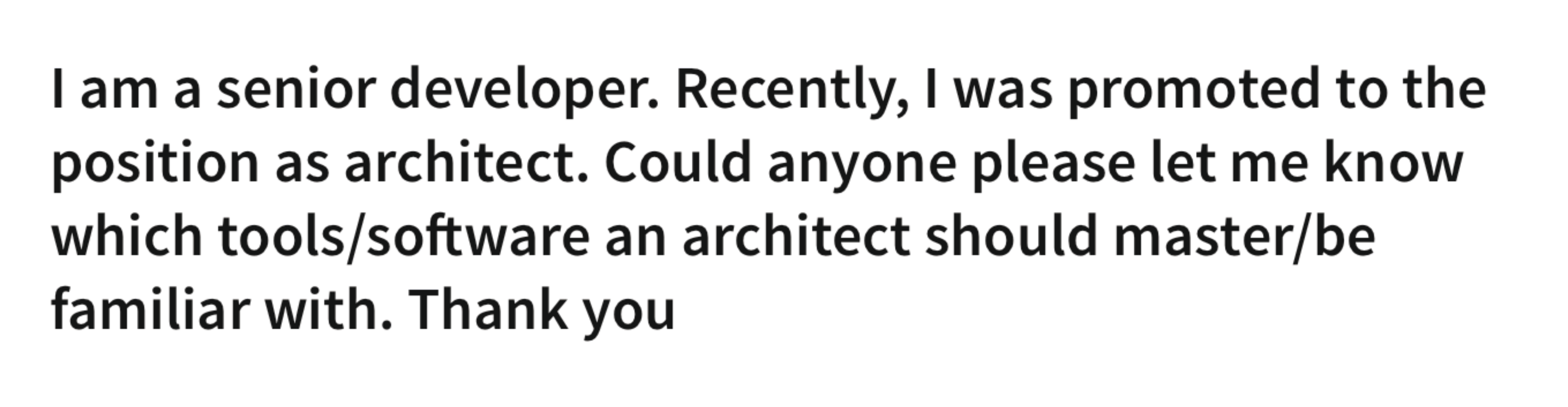
COMMERCE

CRAFT

~ 1990, adoption of development methods

Progress Toward an Engineering Discipline of Software

Mary Shaw



I am a senior developer. Recently, I was promoted to the position as architect. Could anyone please let me know which tools/software an architect should master/be familiar with. Thank you

Experience is important …
software architecture is not a rank!

Software architecture is not a “post-technical” career option!

T

Technology skills

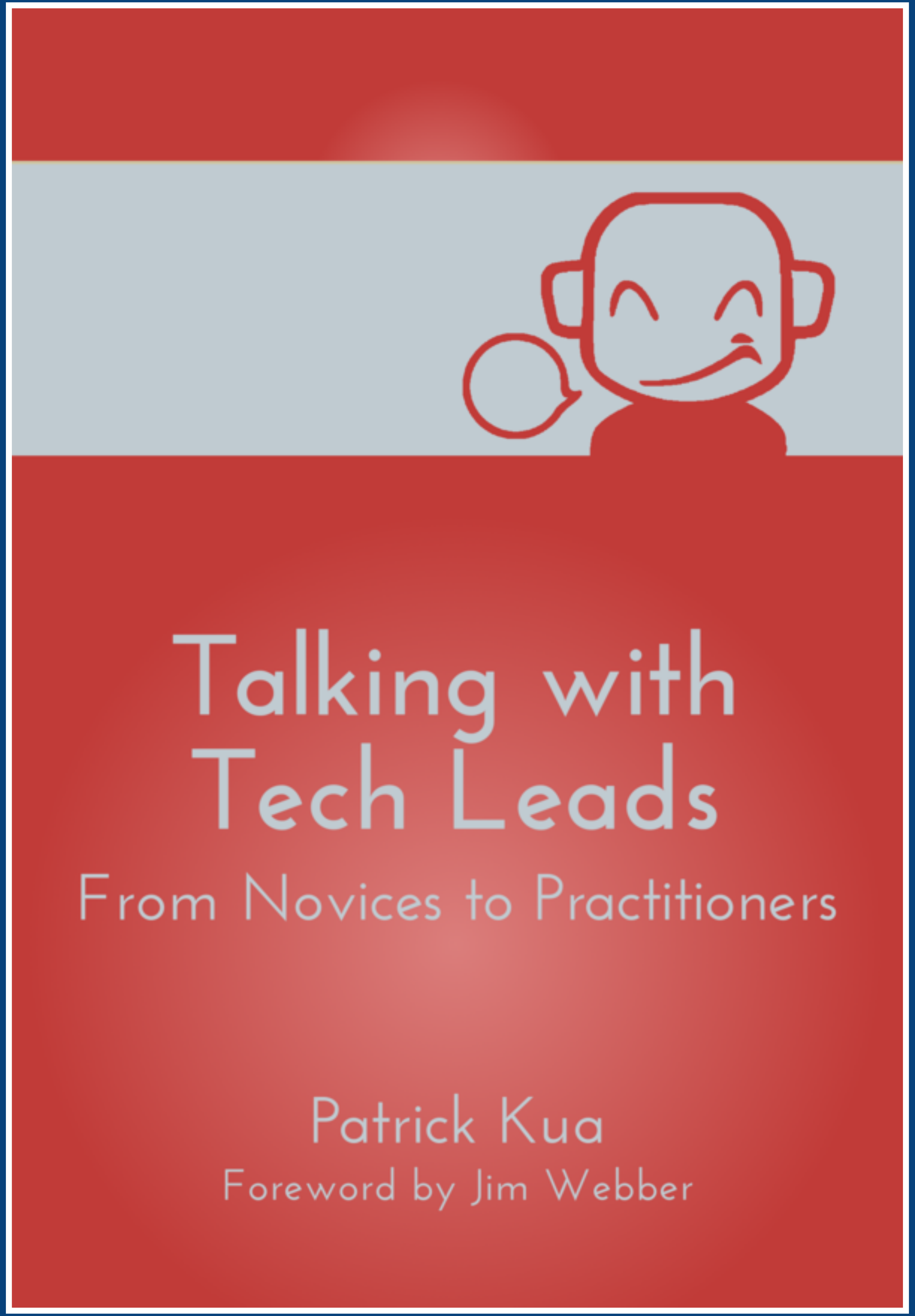
Good software architects
are typically
good software developers

The people designing software must understand technology …

all decisions involve trade-offs

Soft skills

(leadership, communication, presentation, influencing, negotiation, collaboration, coaching and mentoring, motivation, facilitation, political, etc)

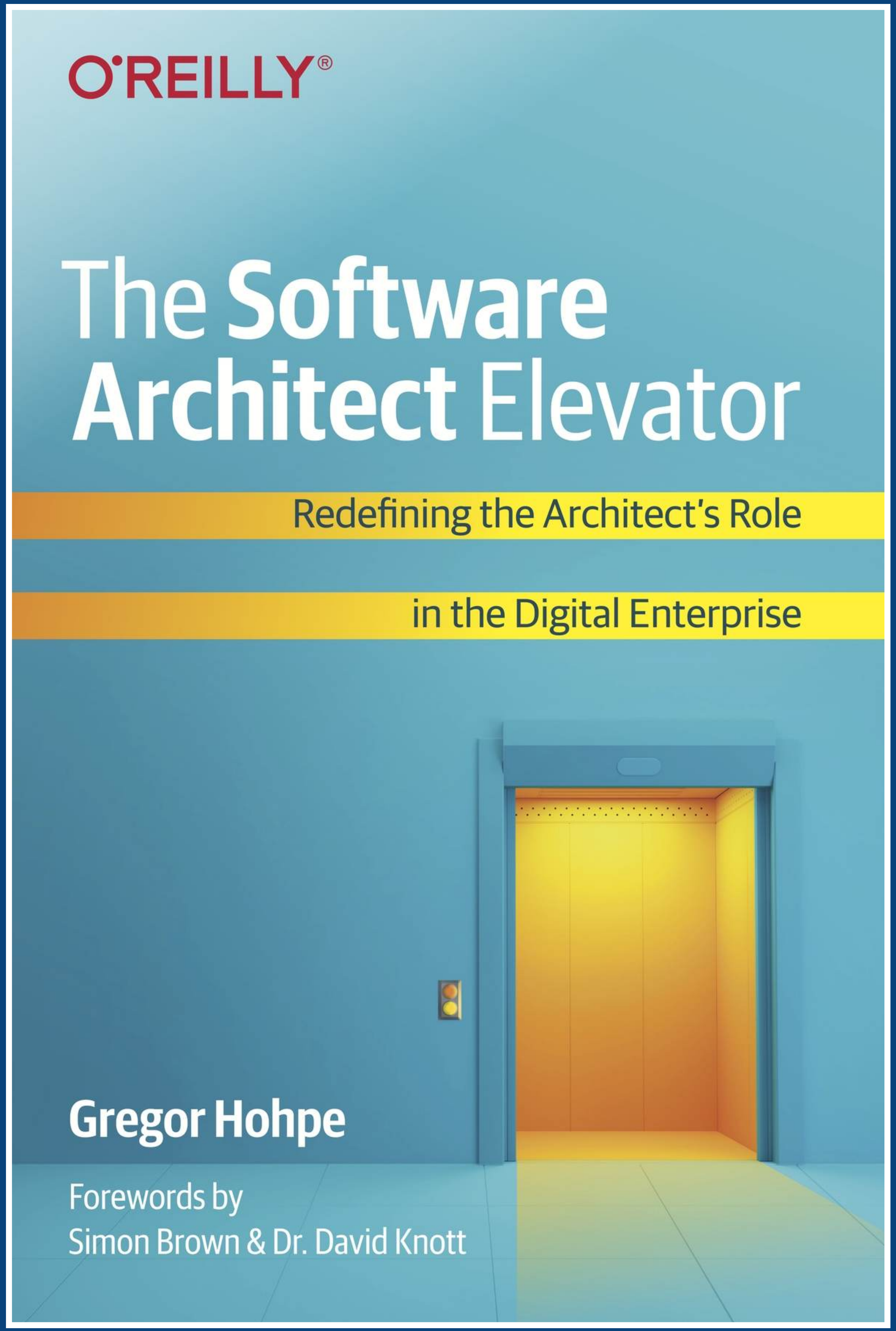


Talking with Tech Leads

From Novices to Practitioners

Patrick Kua

Foreword by Jim Webber



O'REILLY®

The Software Architect Elevator

Redefining the Architect's Role in the Digital Enterprise

Gregor Hohpe

Forewords by Simon Brown & Dr. David Knott

Domain knowledge

(or the ability to learn quickly)

The software architecture role is **multi-faceted**

(technology, soft skills, domain knowledge)

Software architects,
solution architects,
tech leads,
principal engineers?

Technical priorities
vs
product priorities?

The product owner(s) and software architect(s) are peers

(“Architecture Owner” is another term you can use)

“Everybody should be an architect”

“everybody is responsible for architecture”

!=

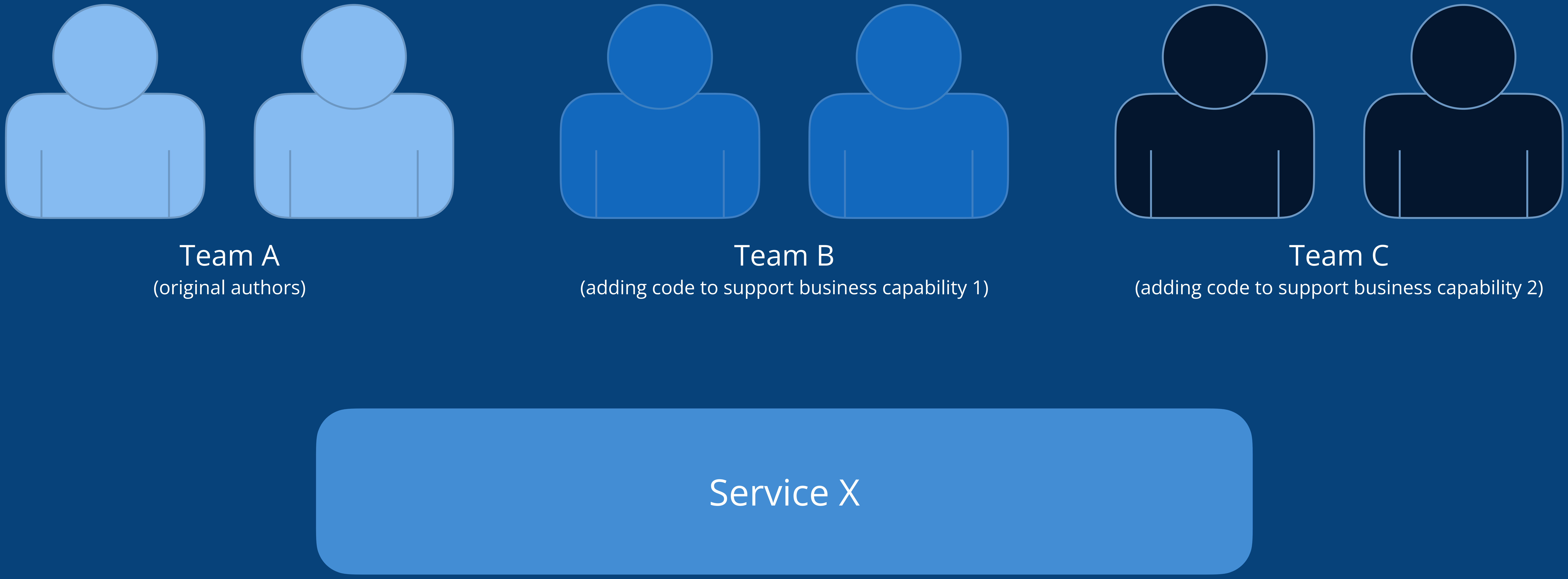
everybody being responsible for architecture

Everybody* should
own the architecture

“teams should be
agile, autonomous,
and self-organising”

just hire good people and trust them to do the right thing

Does everybody have the skills
and motivation to collaborate
on the software architecture role?



Team A
(original authors)

Team B
(adding code to support business capability 1)

Team C
(adding code to support business capability 2)

Service X

Product vs stream leadership

Hierarchies of architects, central architecture groups, technical design authorities, etc?

Decision making

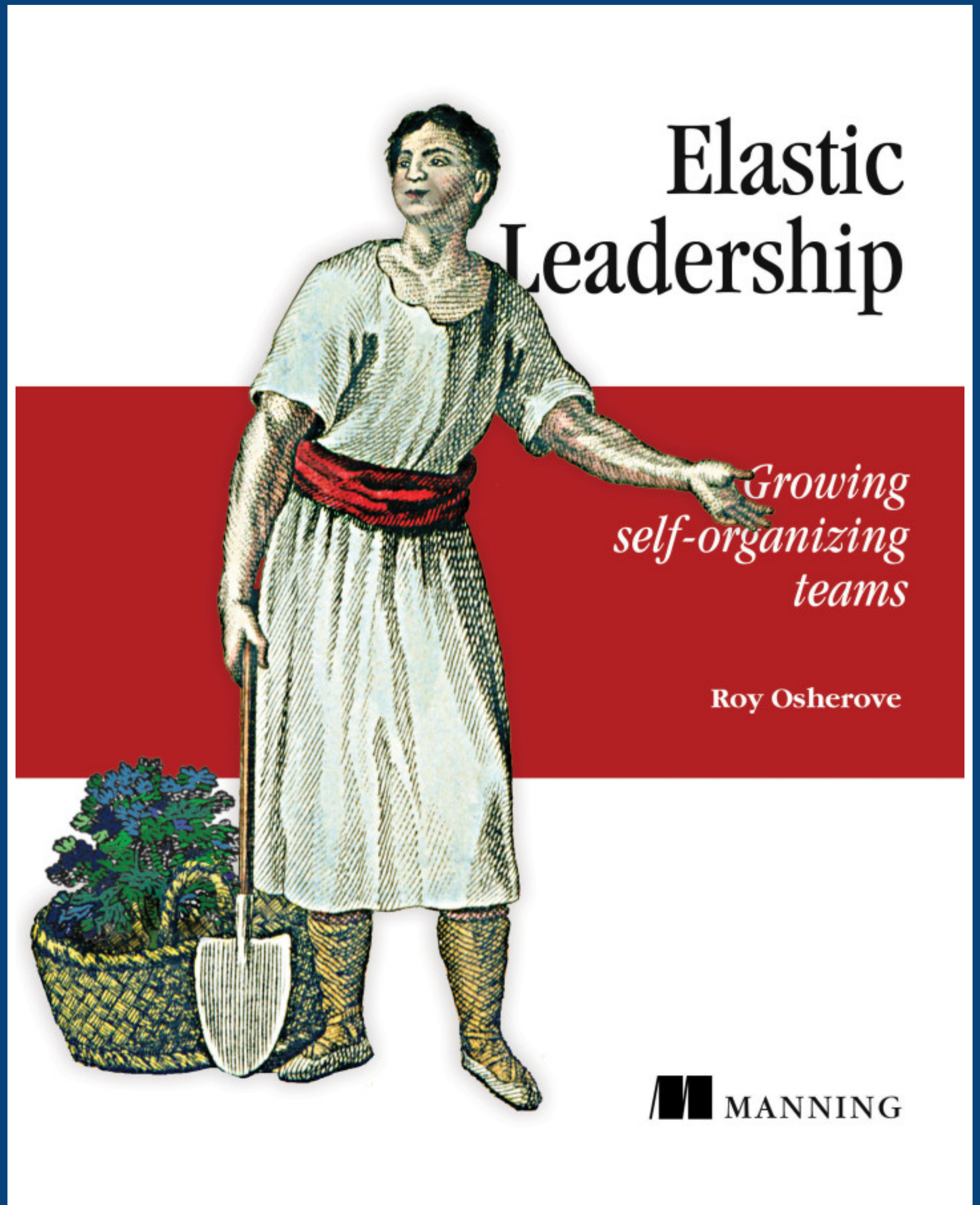
Centralised vs decentralised

Tactical vs strategic

Introducing control?

Avoiding chaos?

How much control do you need?



Different types of teams need different leadership styles

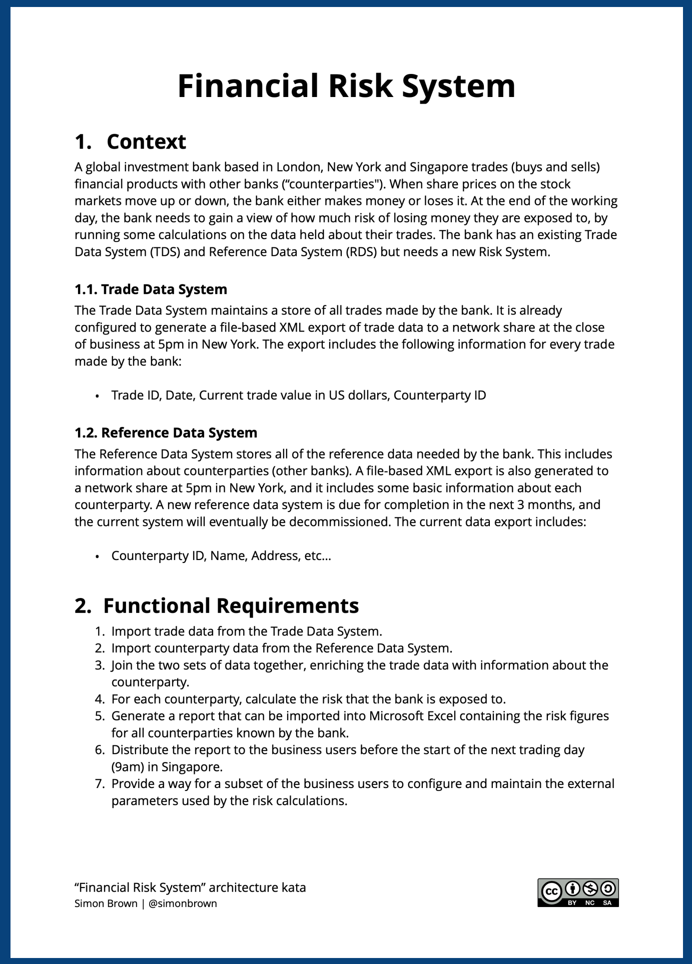
Pair architecting

Collaborative technical leadership

is not easy

Collaborate or fail

Draw one or more software architecture diagrams to describe a solution for the "Financial Risk System"



Financial Risk System

1. Context

A global investment bank based in London, New York and Singapore trades (buys and sells) financial products with other banks ("counterparties"). When share prices on the stock markets move up or down, the bank either makes money or loses it. At the end of the working day, the bank needs to gain a view of how much risk of losing money they are exposed to, by running some calculations on the data held about their trades. The bank has an existing Trade Data System (TDS) and Reference Data System (RDS) but needs a new Risk System.

1.1. Trade Data System

The Trade Data System maintains a store of all trades made by the bank. It is already configured to generate a file-based XML export of trade data to a network share at the close of business at 5pm in New York. The export includes the following information for every trade made by the bank:

- Trade ID, Date, Current trade value in US dollars, Counterparty ID

1.2. Reference Data System

The Reference Data System stores all of the reference data needed by the bank. This includes information about counterparties (other banks). A file-based XML export is also generated to a network share at 5pm in New York, and it includes some basic information about each counterparty. A new reference data system is due for completion in the next 3 months, and the current system will eventually be decommissioned. The current data export includes:

- Counterparty ID, Name, Address, etc…

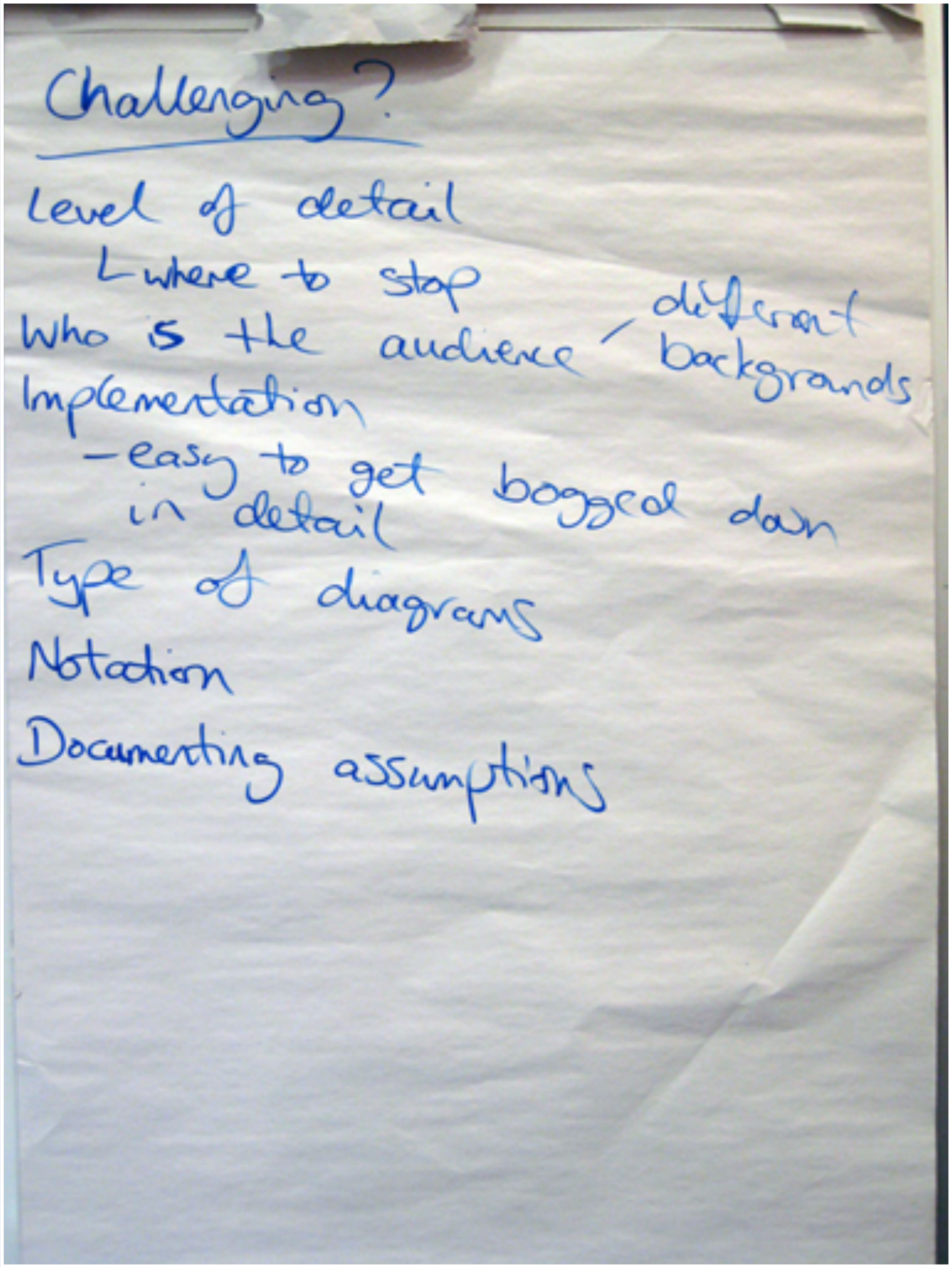
2. Functional Requirements

1. Import trade data from the Trade Data System.
2. Import counterparty data from the Reference Data System.
3. Join the two sets of data together, enriching the trade data with information about the counterparty.
4. For each counterparty, calculate the risk that the bank is exposed to.
5. Generate a report that can be imported into Microsoft Excel containing the risk figures for all counterparties known by the bank.
6. Distribute the report to the business users before the start of the next trading day (9am) in Singapore.
7. Provide a way for a subset of the business users to configure and maintain the external parameters used by the risk calculations.

"Financial Risk System" architecture kata
Simon Brown | @simonbrown

simonbrown.je

Did you find anything about this exercise challenging?



Challenging?

Level of detail

L where to stop

Who is the audience / different backgrounds

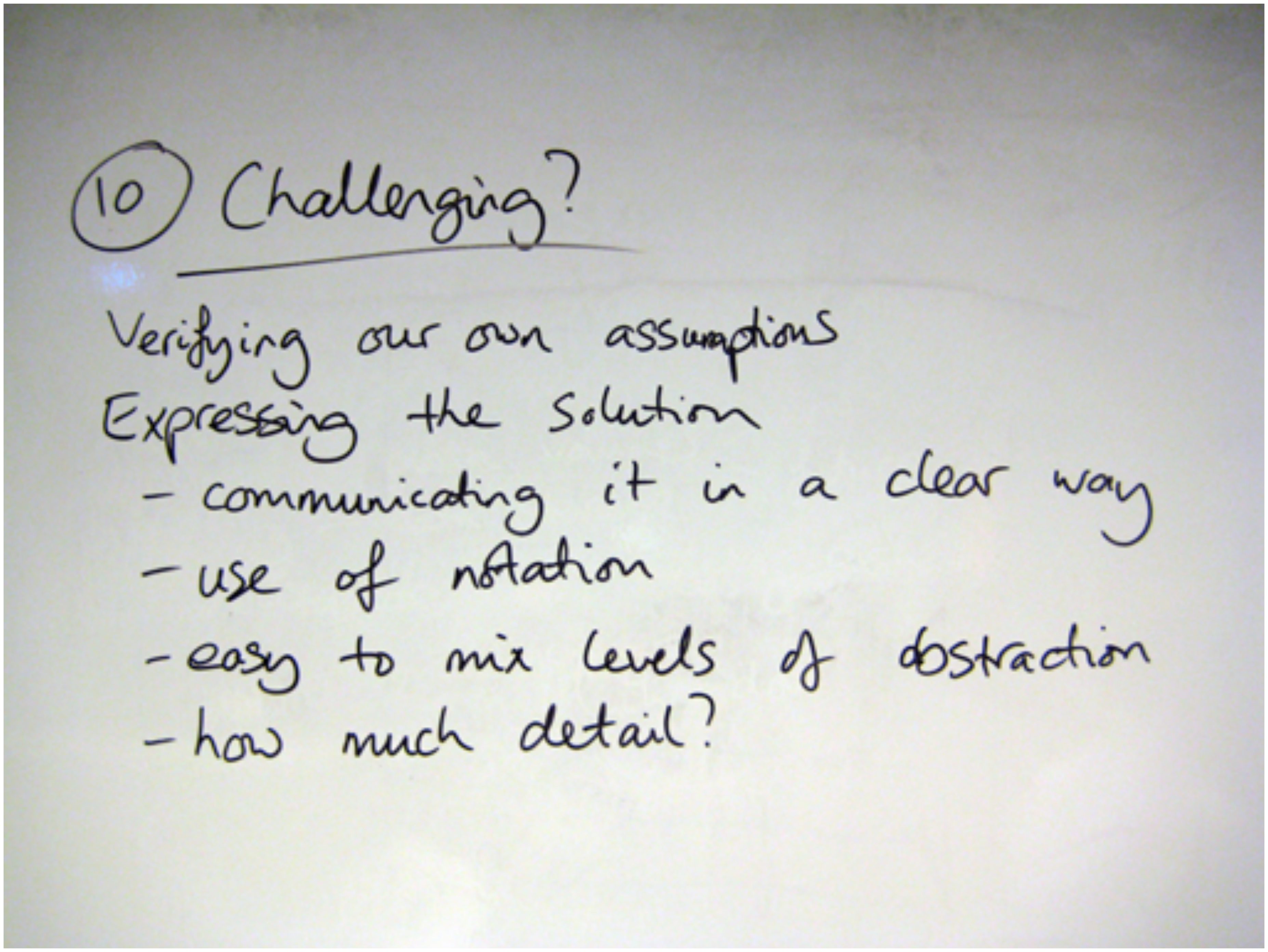
Implementation

- easy to get bogged down in detail

Type of diagrams

Notation

Documenting assumptions

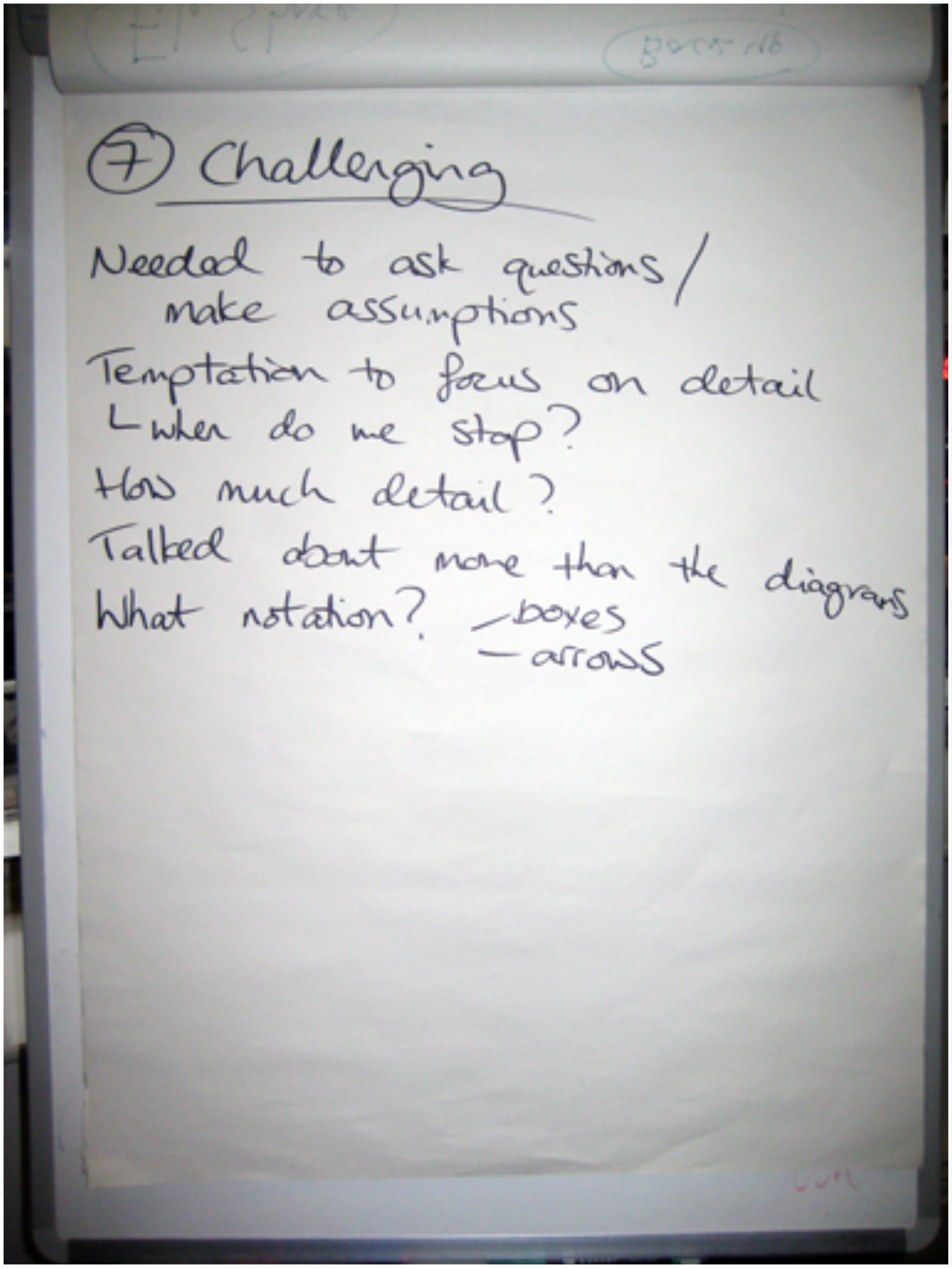


10 Challenging?

Verifying our own assumptions

Expressing the solution

- communicating it in a clear way
- use of notation
- easy to mix levels of abstraction
- how much detail?



7 Challenging

Needed to ask questions / make assumptions

Temptation to focus on detail

L when do we stop?

How much detail?

Talked about more than the diagrams

What notation? - boxes

- arrows

Take a quick look at the diagrams:

1. Does the solution satisfy the architectural drivers?
2. If you were the bank, would you buy this solution?

Swap your diagrams with another group

Review the diagrams

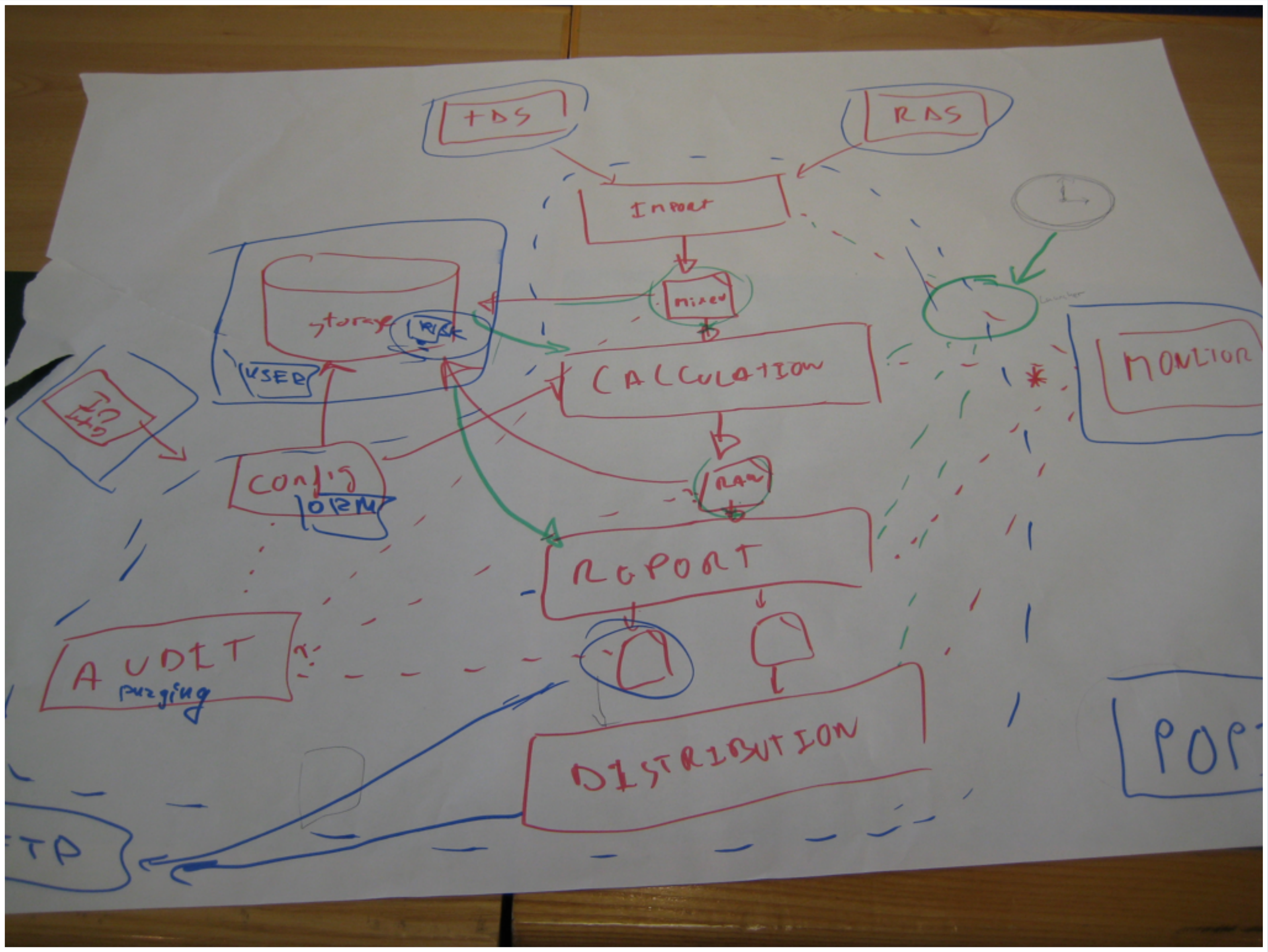
Focus on the diagrams rather than the design
… notation, colour coding, symbols, etc

3 things you like 🟩🟩🟩

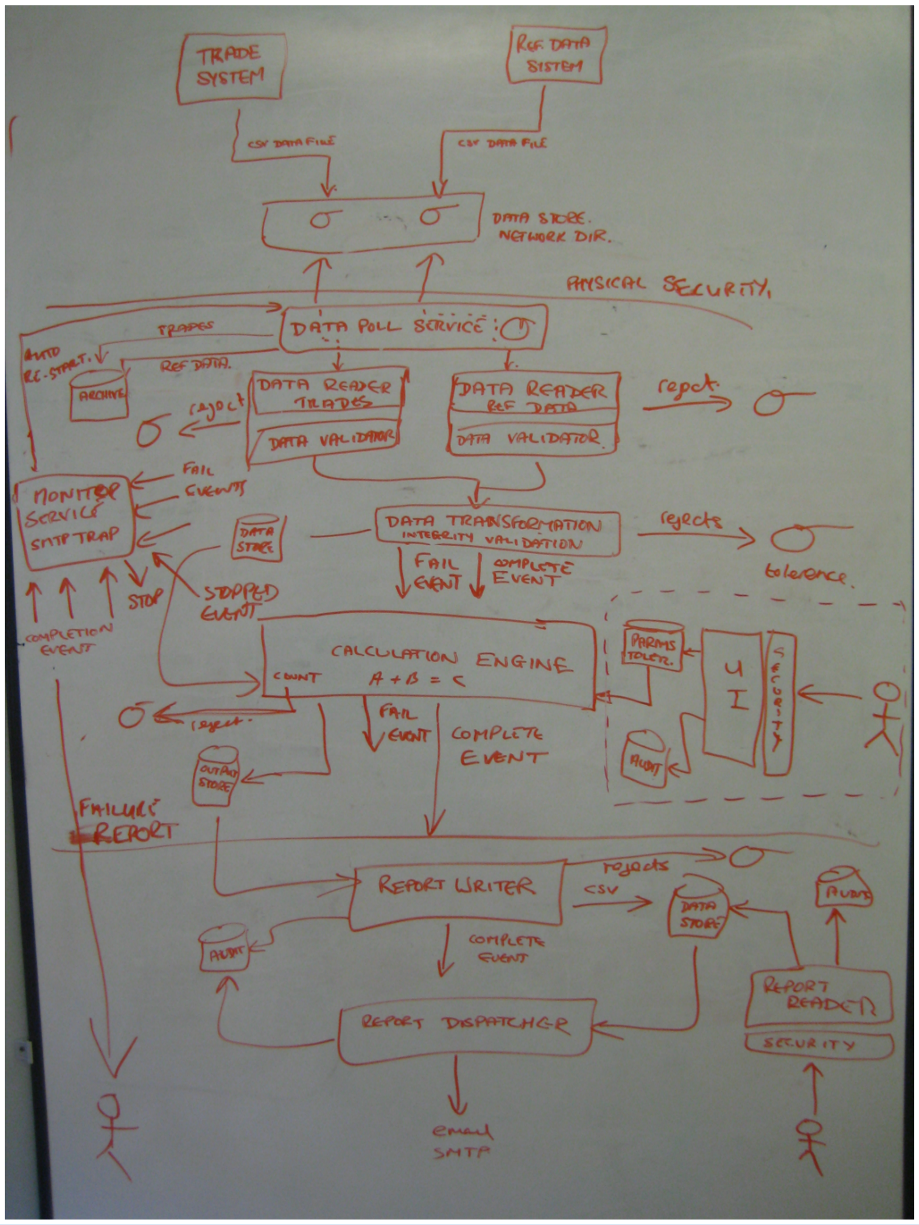
3 things that could be improved 🟥🟥🟥

A score between 1-10 🟨

1 1 1


TDS
RAS
Import
mixed
storage
RISK
USER
CALCULATION
MONITOR
config
ORM
RAW
REPORT
AUDIT
purging
DISTRIBUTION
FTP
POP

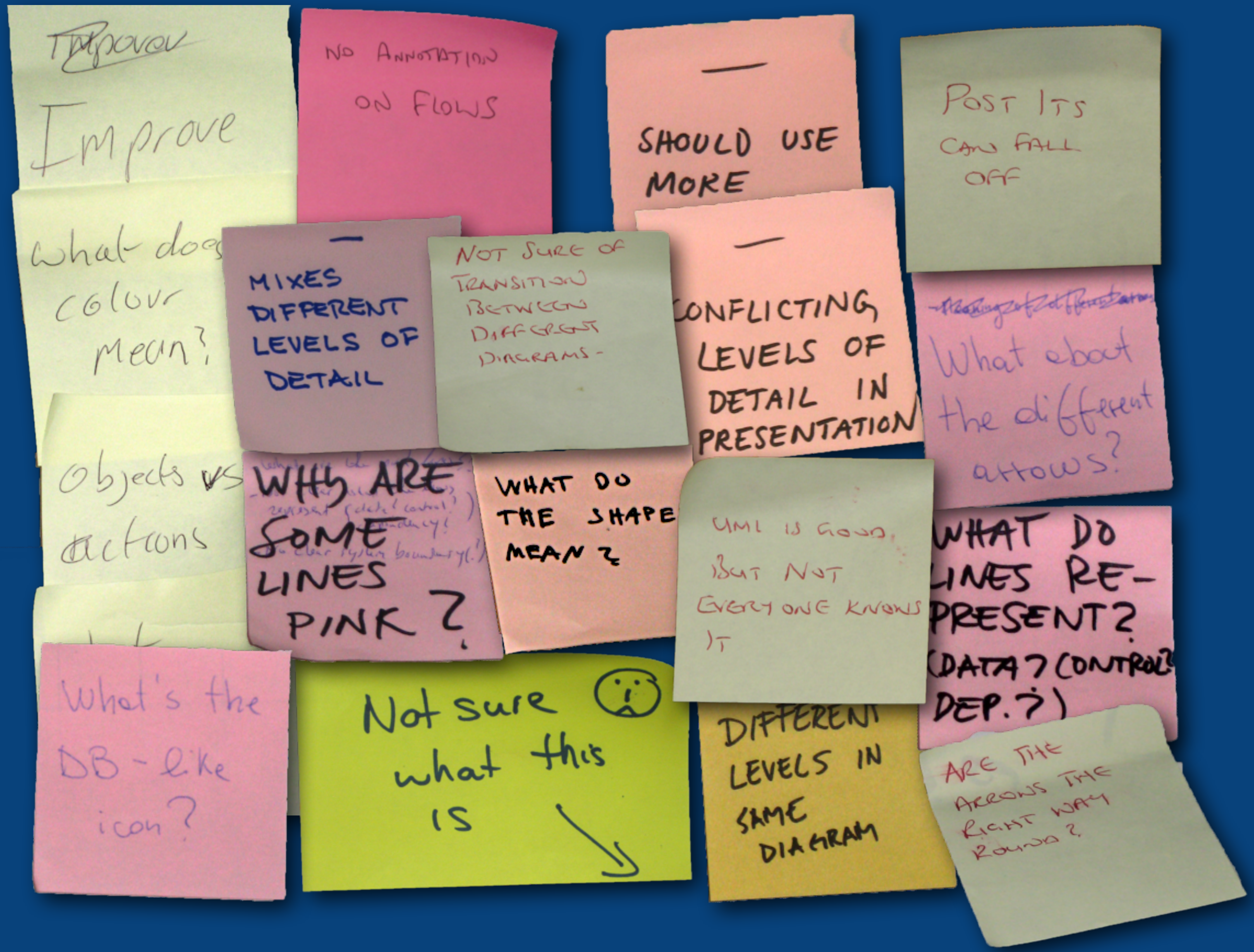
2 2 2

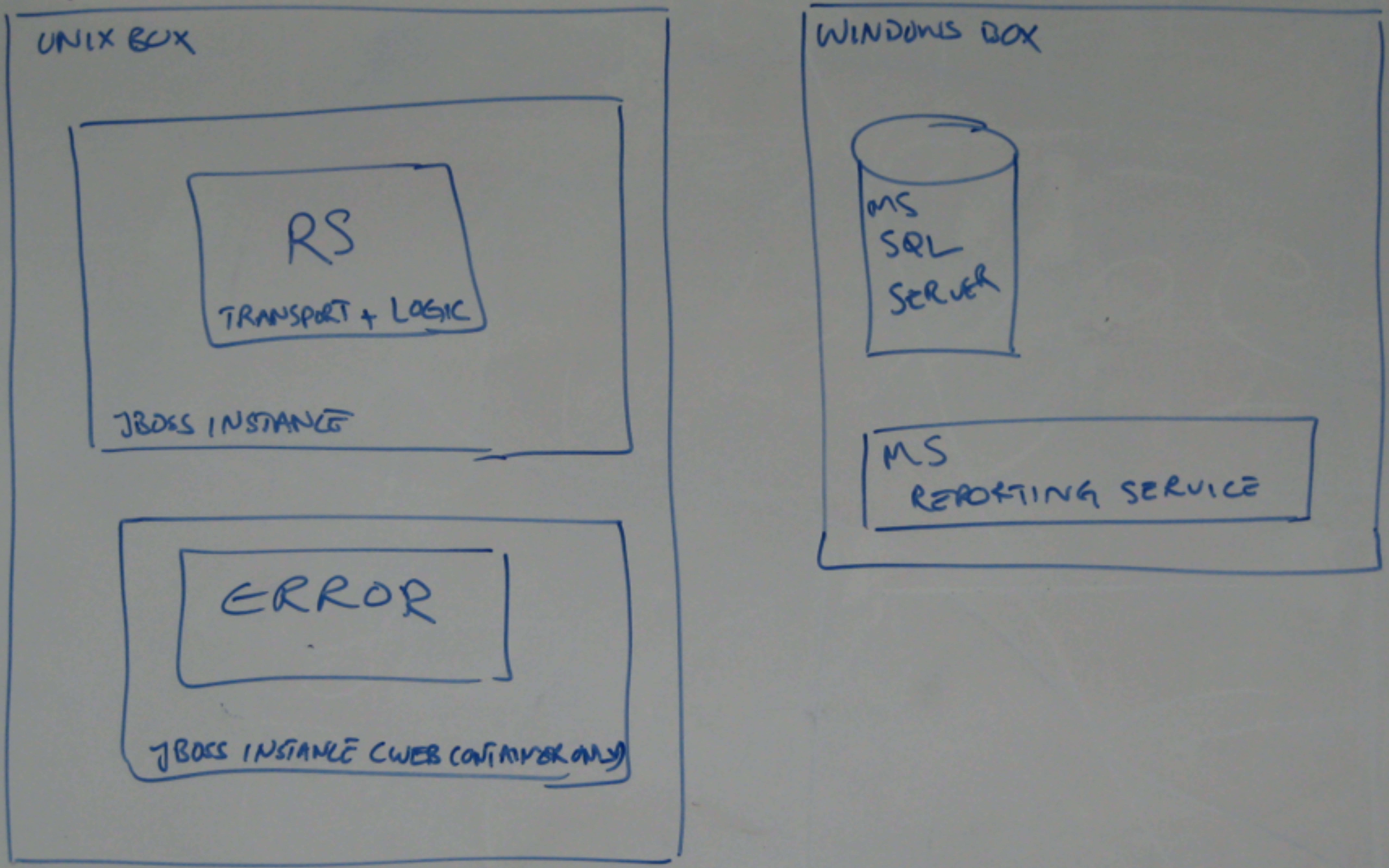

TRADE SYSTEM
REF DATA SYSTEM
CSV DATA FILE
CSV DATA FILE
DATA STORE. NETWORK DIR.
PHYSICAL SECURITY.
DATA POLL SERVICE
TRADES
REF DATA.
AUTO RE-START.
ARCHIVE
reject
DATA READER TRADES
DATA VALIDATOR
DATA READER REF DATA
DATA VALIDATOR
reject.
FAIL EVENTS
MONITOR SERVICE SMTP TRAP
DATA STORE
DATA TRANSFORMATION INTEGRITY VALIDATION
rejects
tolerance.
FAIL EVENT
COMPLETE EVENT
STOP
STOPPED EVENT
COMPLETION EVENT
CALCULATION ENGINE
COUNT
A + B = C
PARAMS TOLER.
UI
SECURITY
AUDIT
reject.
FAIL EVENT
COMPLETE EVENT
OUTPUT STORE
FAILURE REPORT
REPORT WRITER
rejects
CSV
DATA STORE
AUDIT
AUDIT
COMPLETE EVENT
REPORT READER
SECURITY
REPORT DISPATCHER
email SMTP

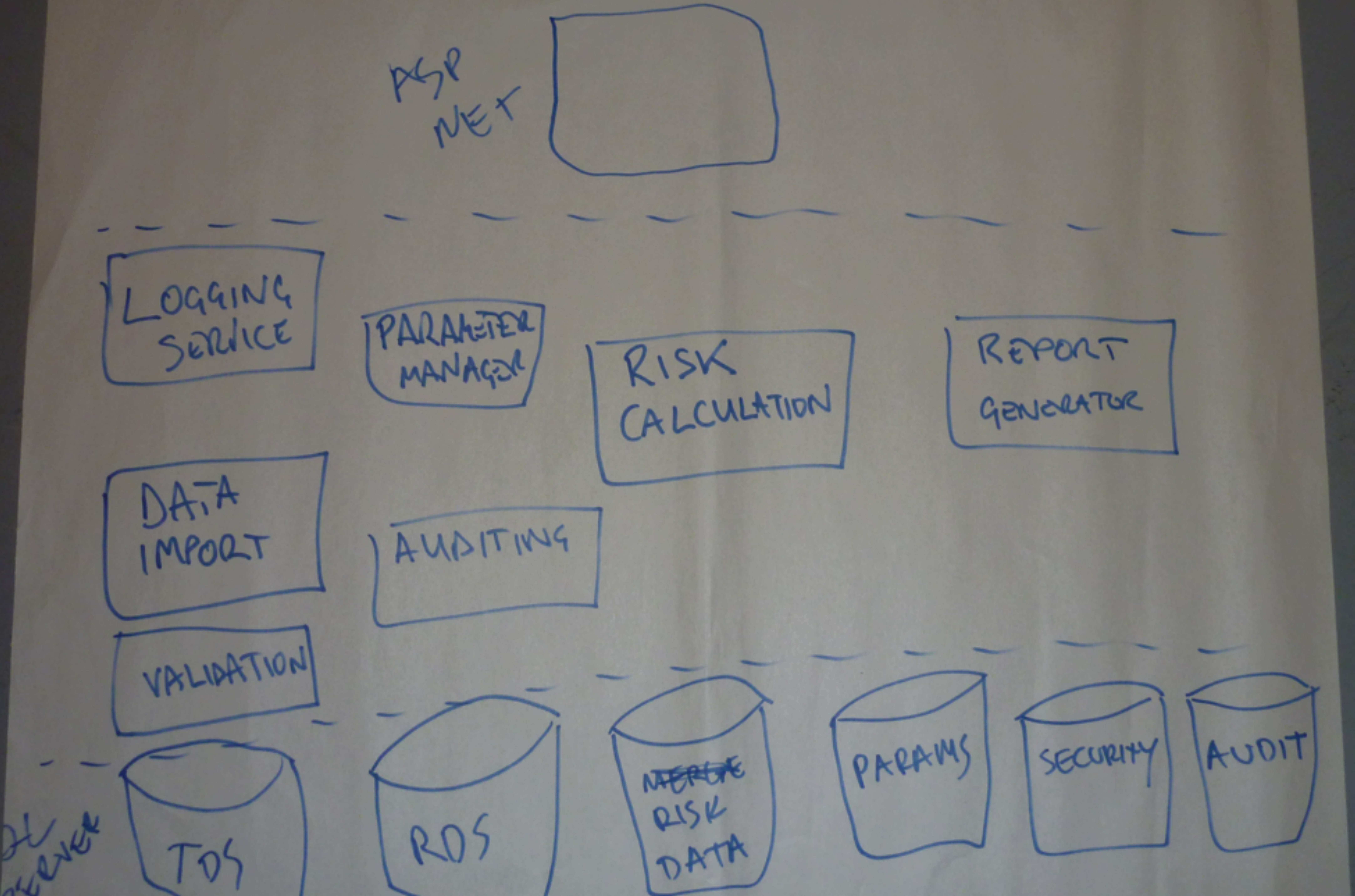
Information is likely
still stuck in your heads

This doesn’t make sense,
but we’ll explain it.

- What is this shape/symbol?
- What is this line/arrow?
- What do the colours mean?
- What level of abstraction is shown?
- Which diagram do we read first?


Improve
what does colour mean?
objects vs actions
No annotation on flows
— Mixes different levels of detail
Not sure of transition between different diagrams -
Why are some lines pink?
What do the shape mean?
— Should use more
— Conflicting levels of detail in presentation
UML is good, but not everyone knows it
Post Its can fall off
What about the different arrows?
What do lines represent? (data? control? dep.?)
What's the DB-like icon?
Not sure what this is
Different levels in same diagram
Are the arrows the right way round?


UNIX BOX
RS
TRANSPORT + LOGIC
JBOSS INSTANCE
ERROR
JBOSS INSTANCE (WEB CONTAINER ONLY)
WINDOWS BOX
MS SQL SERVER
MS REPORTING SERVICE


ASP
NET
LOGGING SERVICE
PARAMETER MANAGER
RISK CALCULATION
REPORT GENERATOR
DATA IMPORT
AUDITING
VALIDATION
SQL SERVER
TDS
RDS
~~MERGE~~
RISK DATA
PARAMS
SECURITY
AUDIT

FUNCTIONAL VIEW

File Retriever

Scheduler

Auditing

Reference Archiver

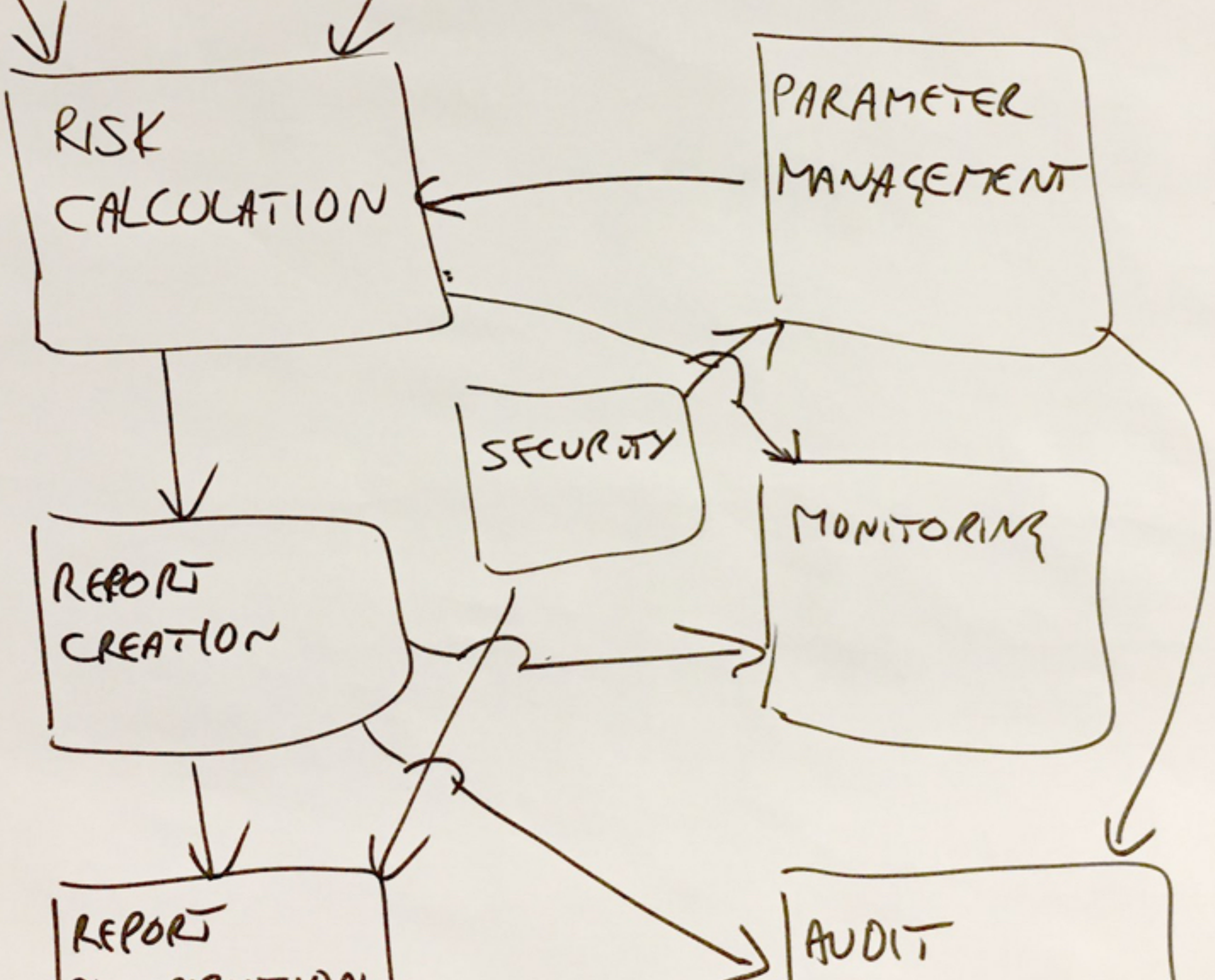
Risk Assesment Processor

Risk Parameter Configuration

Report

Import
mixed
storage
RISK
USER
Git UI
config
ORM
CALCULATION
RAW
REPORT
AUDIT
purging
Launcher
M
TION

TDS
RDS
trade report
ref report
Rg
TRANSPORT
S
ARCHIVE
S
action
data
error
error
AUDIT
S
MS Sql Svr
BUS. LOGIC
action
error
ERROR
MONITOR
REPORT GEN
S
action
error
Scheduler
config

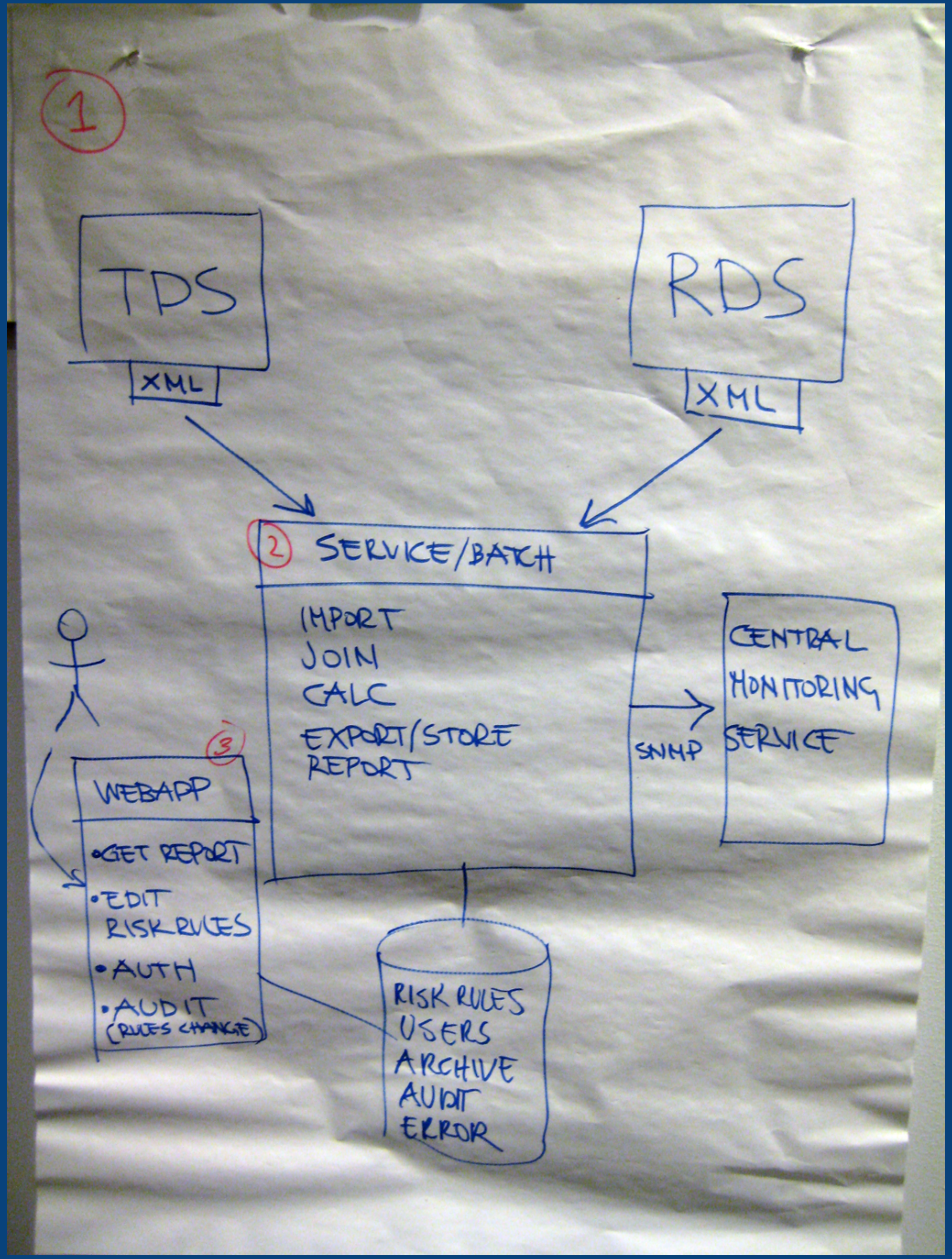

RISK CALCULATION
PARAMETER MANAGEMENT
SECURITY
MONITORING
REPORT CREATION
REPORT
AUDIT

2
SERVICE/BATCH
IMPORT
JOIN
CALC
EXPORT/STORE
REPORT
SNMP
CENTRAL
MONITORING
SERVICE
3
WEBAPP
•GET REPORT
•EDIT RISK RULES
•AUTH
•AUDIT
RISK RULES
USERS

The producer-consumer conflict of software architecture diagrams

I don't want to put technology choices on the diagrams...

Software design should be technology independent...



I wish these diagrams included technology choices...

Producer

Consumer

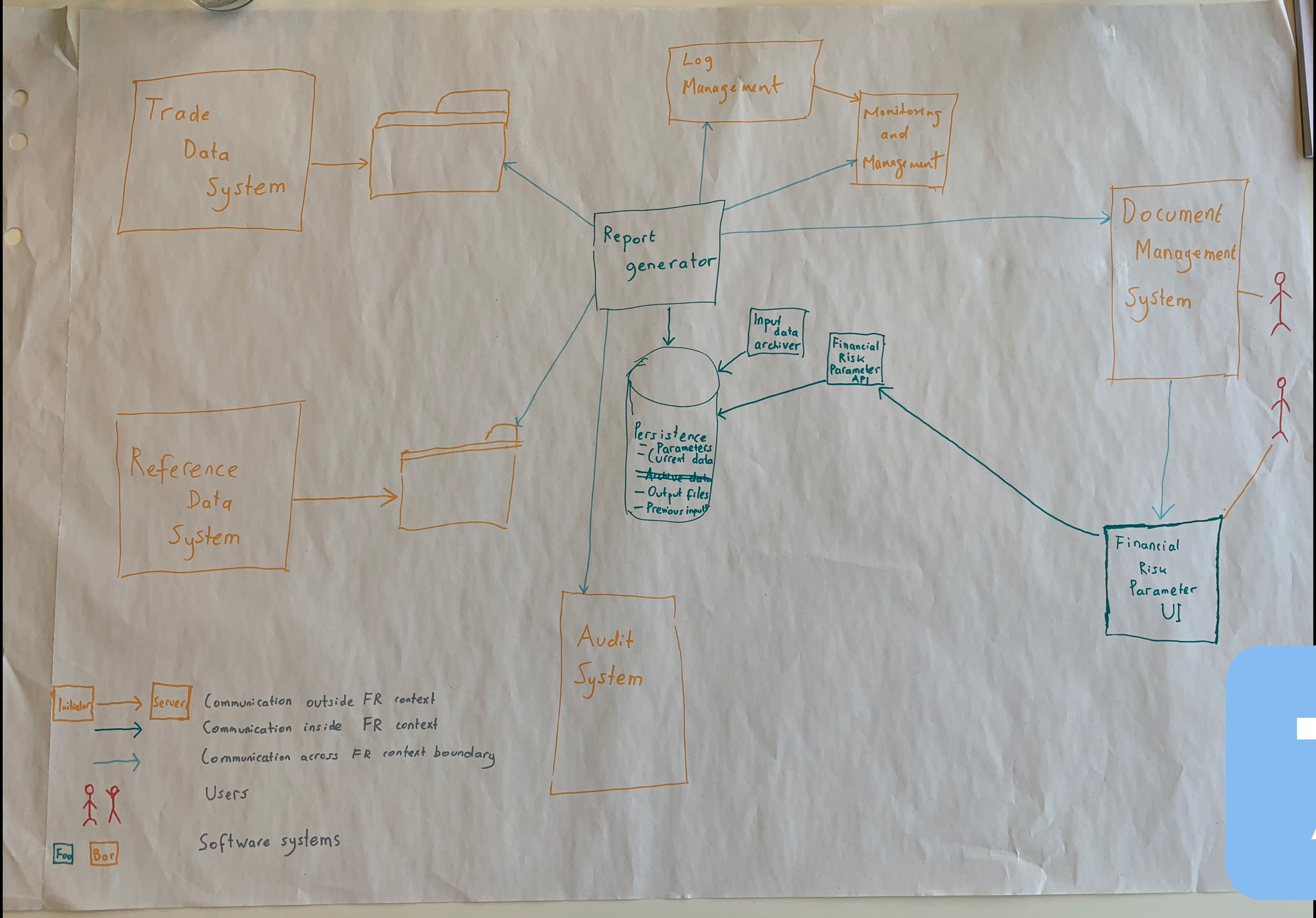
WIN SERVER
APPLICATION
CONSOLE - C#
SCHEDULER
CONSOLE - C#
DATA LOADER
C#
CALCULATION MANAGER
C#
NOTIFICATION MANAGER
C#
EXTERNAL SYSTEMS
SNMP SERVICE
ROS
ITDS
XML
check if exists
DATA MERGE
C#
XPath
XML READER
C#
RISK CALCULATION
C#
parameters
report to DB table
NOTIFICATION SERVICE
C#
SMTP
LOGGER
nLog
SNMP TRAP GENERATOR
DATA ACCESS
SQL SERVER

DATA POLL SERVICE
TRADES
AUTO RE-START.
REF DATA.
ARCHIVE
DATA READER TRADES
DATA VALIDATOR
reject
DATA READER REF DATA
DATA VALIDATOR.
reject.
FAIL EVENTS
MONITOR SERVICE SMTP TRAP
DATA STORE
DATA TRANSFORMATION
INTEGRITY VALIDATION
rejects
tolerence.
FAIL EVENT
COMPLETE EVENT
STOP
STOPPED EVENT
COMPLETION EVENT
CALCULATION ENGINE
COUNT
A + B = C
PARMS TOLER.
UI
SECURITY
AUDIT
reject.
FAIL EVENT
COMPLETE EVENT
OUTPUT STORE
FAILURE

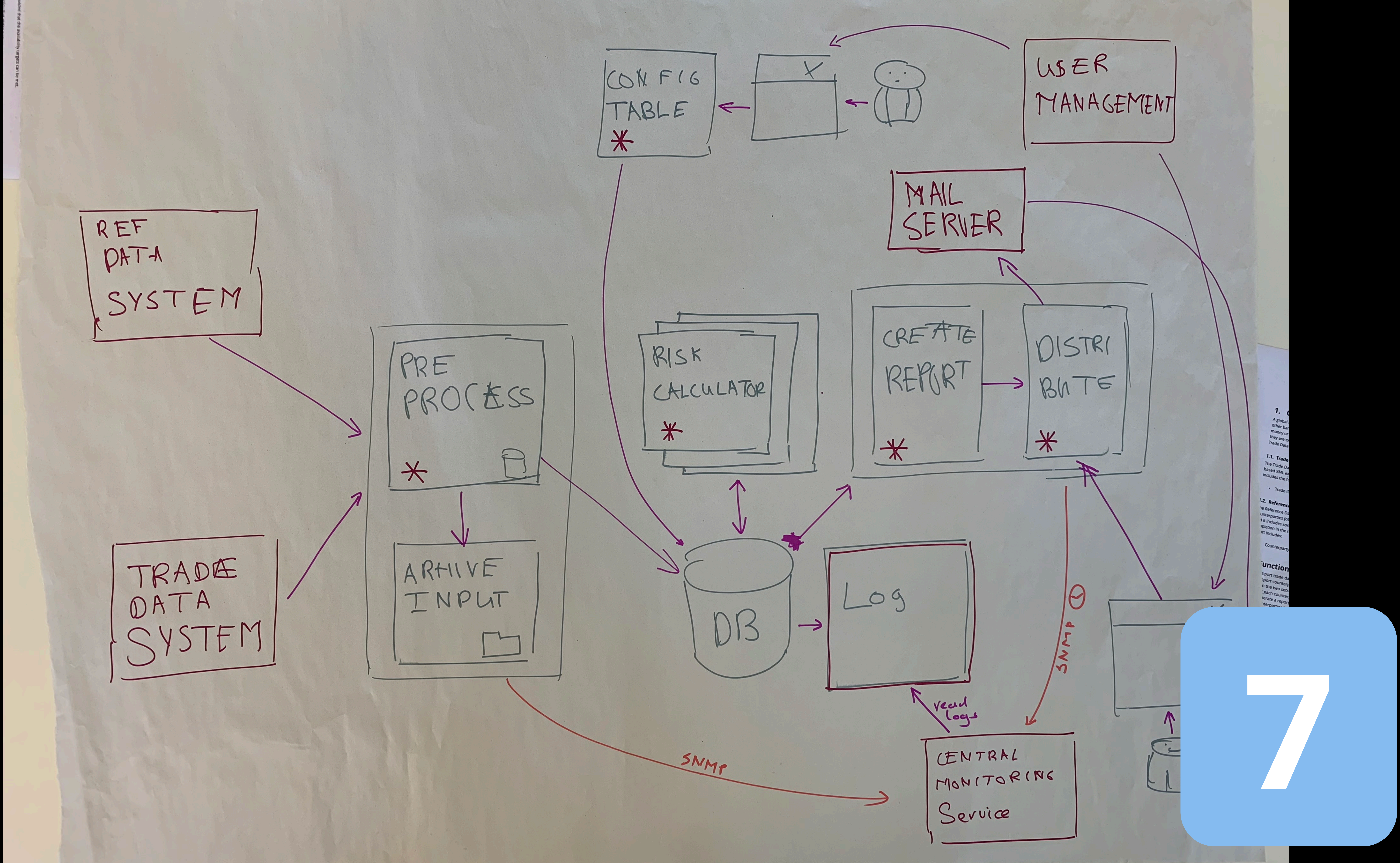
ENGINE
APP SERVER
WHICH DB TECH?
RING
WEB SERVER
REPORTING SERVICE

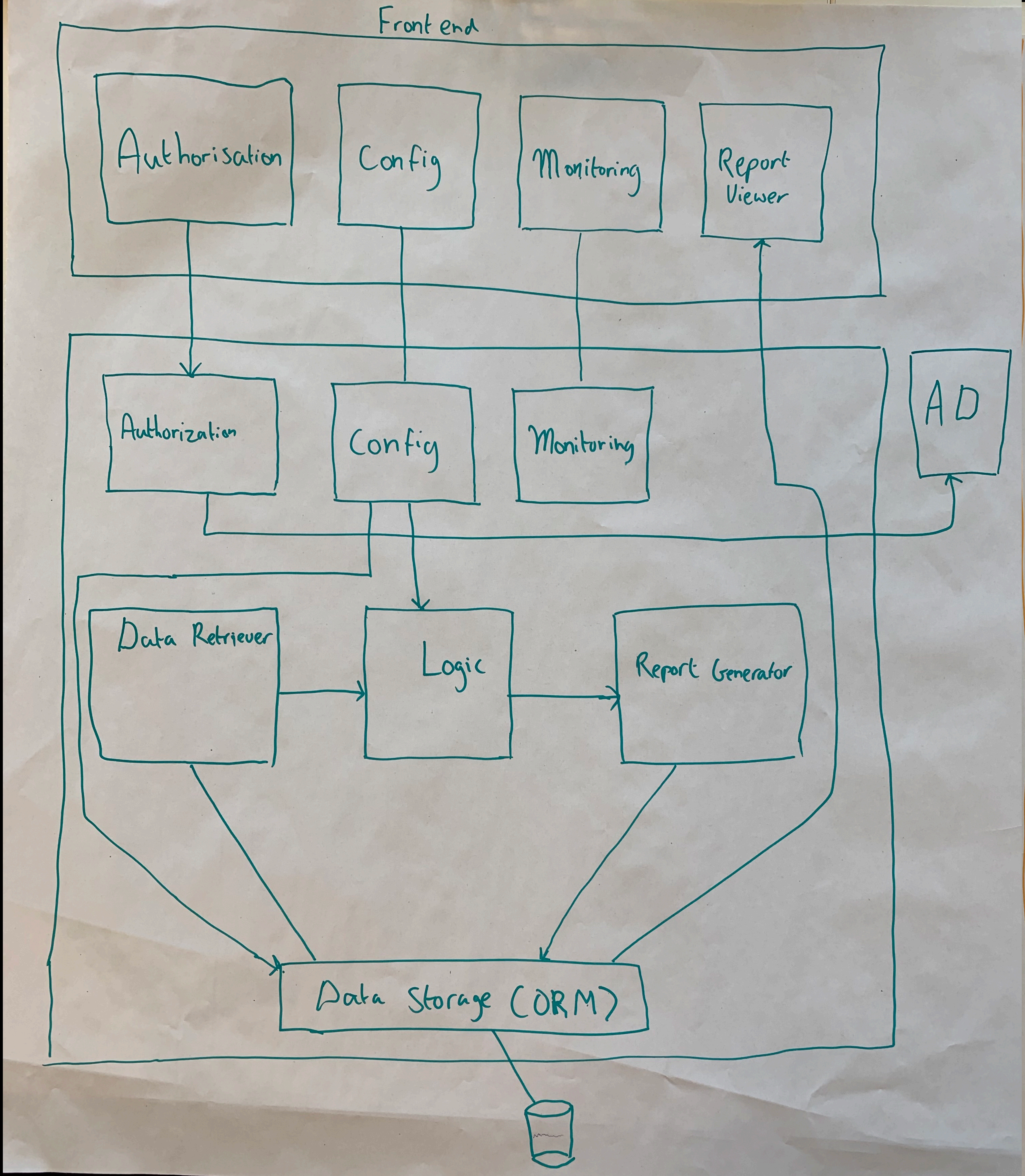
Data Normalizer
Risk ~~Calculator~~ Aggregator
CP Risk Calc
Report Monitor
(other machine)
Report Generator
Auditing Service
Report Publisher
~~Report Repository~~

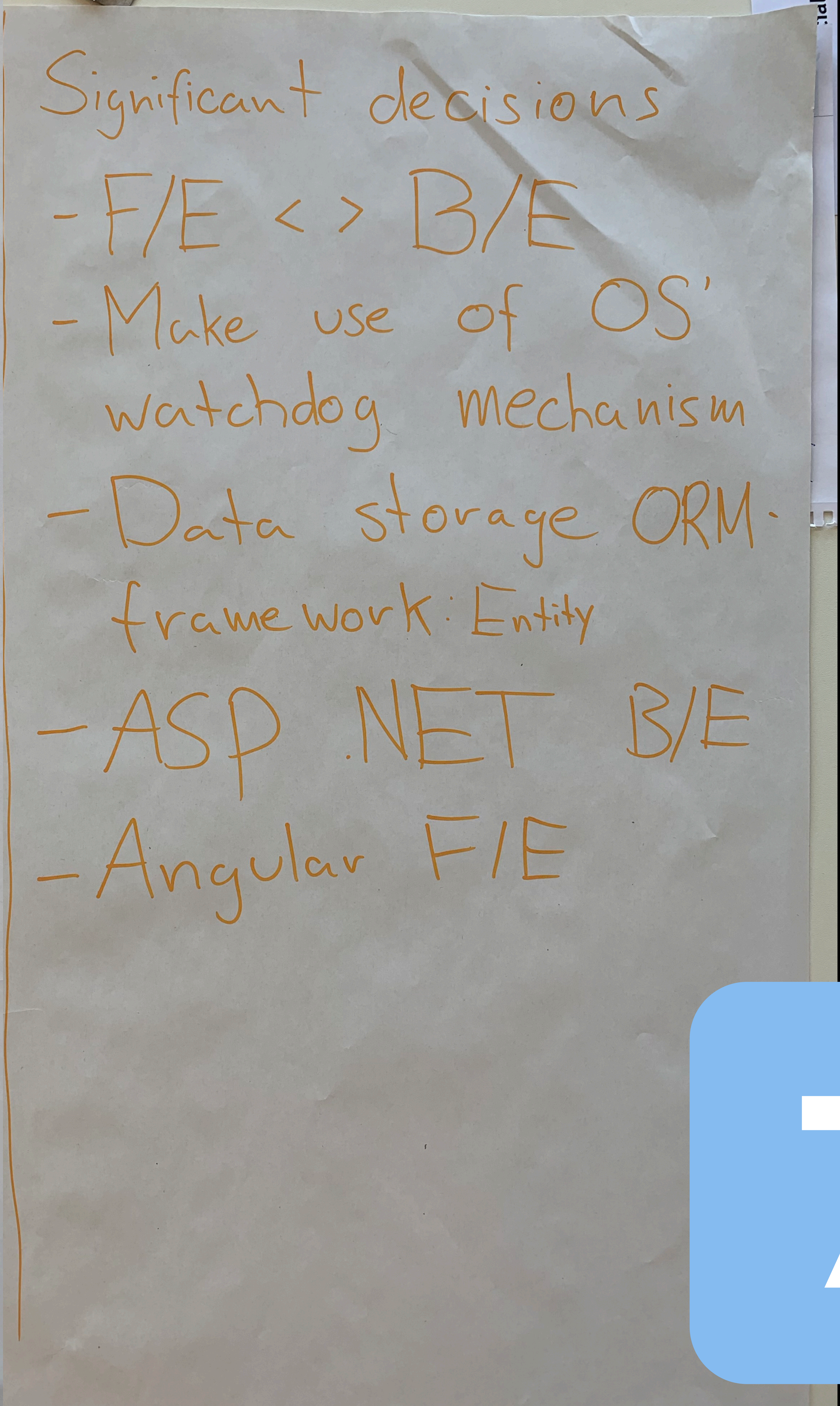
Params
Calcs
ret_curr
Error
App Log
Risk outputs
EH?
Batch
App Error Log
Date
- Risk Cel
- calcs
- Par
Params_Distrib
Detail
B_id:


Trade Data System
Reference Data System
Log Management
Monitoring and Management
Report generator
Document Management System
Input data archiver
Financial Risk Parameter API
Persistence
- Parameters
- Current data
- ~~Archive data~~
- Output files
- Previous input
Financial Risk Parameter UI
Audit System
Initiator
Server
Communication outside FR context
Communication inside FR context
Communication across FR context boundary
Users
Foo
Bar
Software systems

7

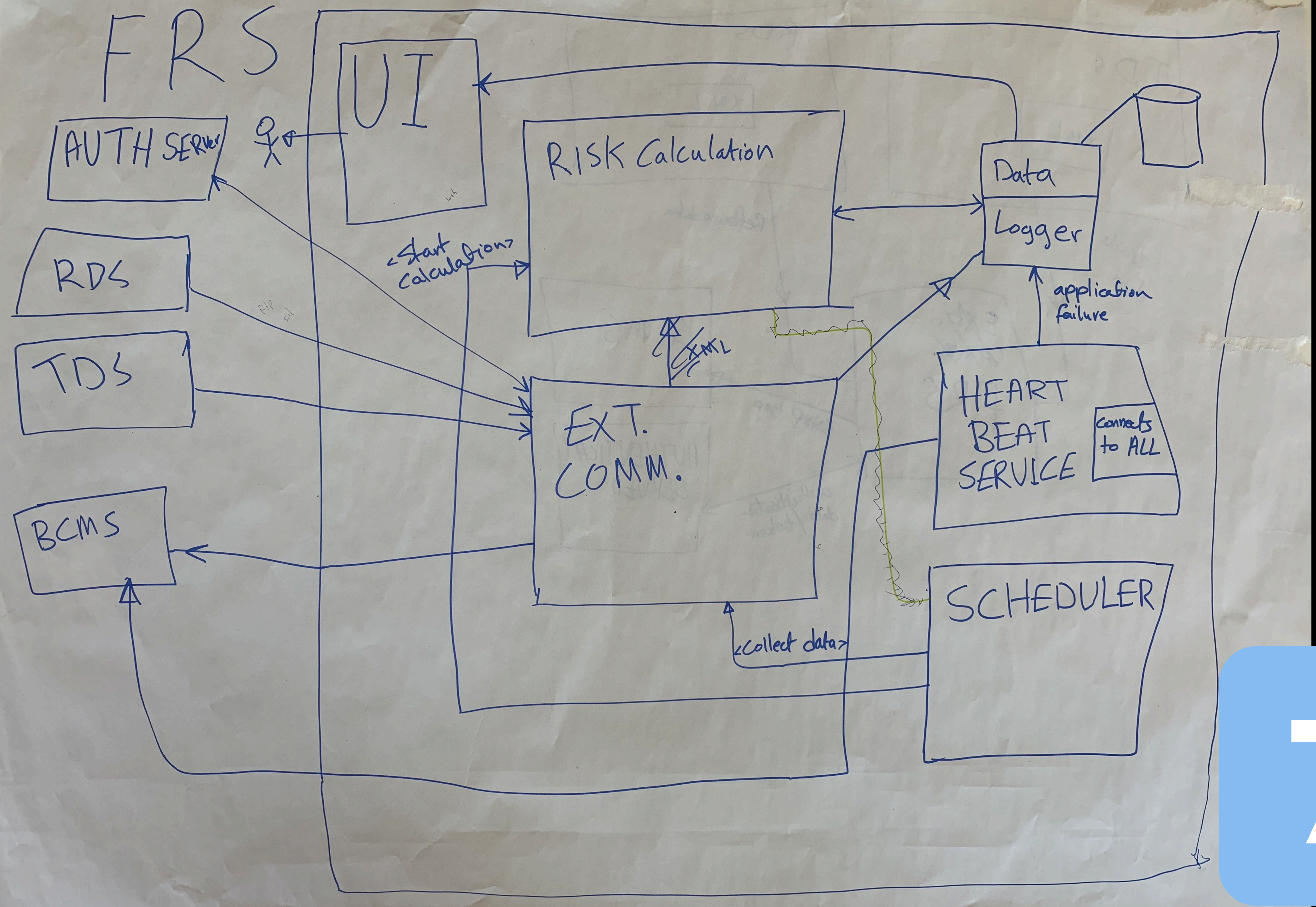

CONFIG TABLE
USER MANAGEMENT
MAIL SERVER
REF DATA SYSTEM
PRE PROCESS
RISK CALCULATOR
CREATE REPORT
DISTRIBUTE
TRADE DATA SYSTEM
ARHIVE INPUT
DB
Log
read logs
SNMP
SNMP
CENTRAL MONITORING Service
7



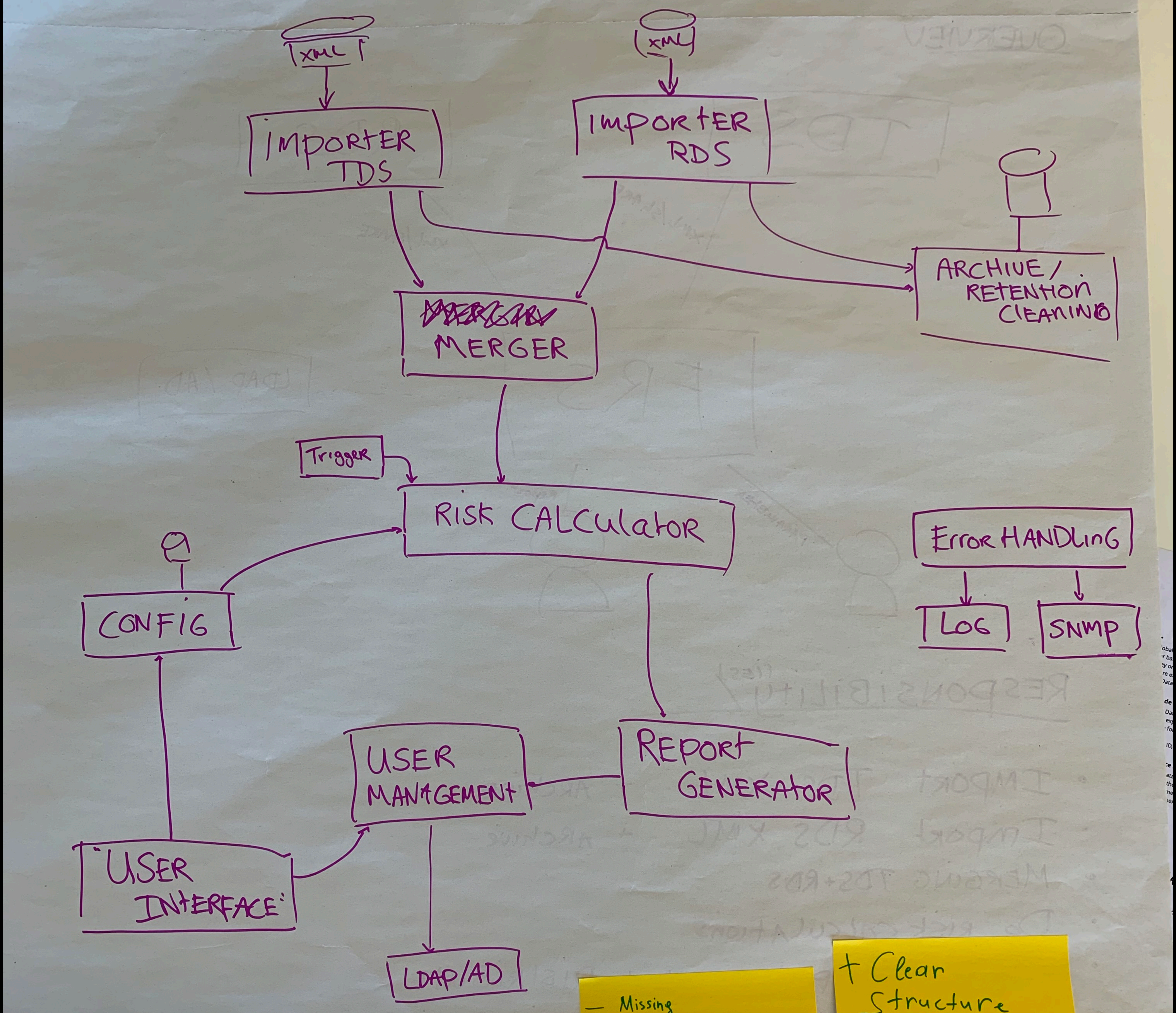


Significant decisions

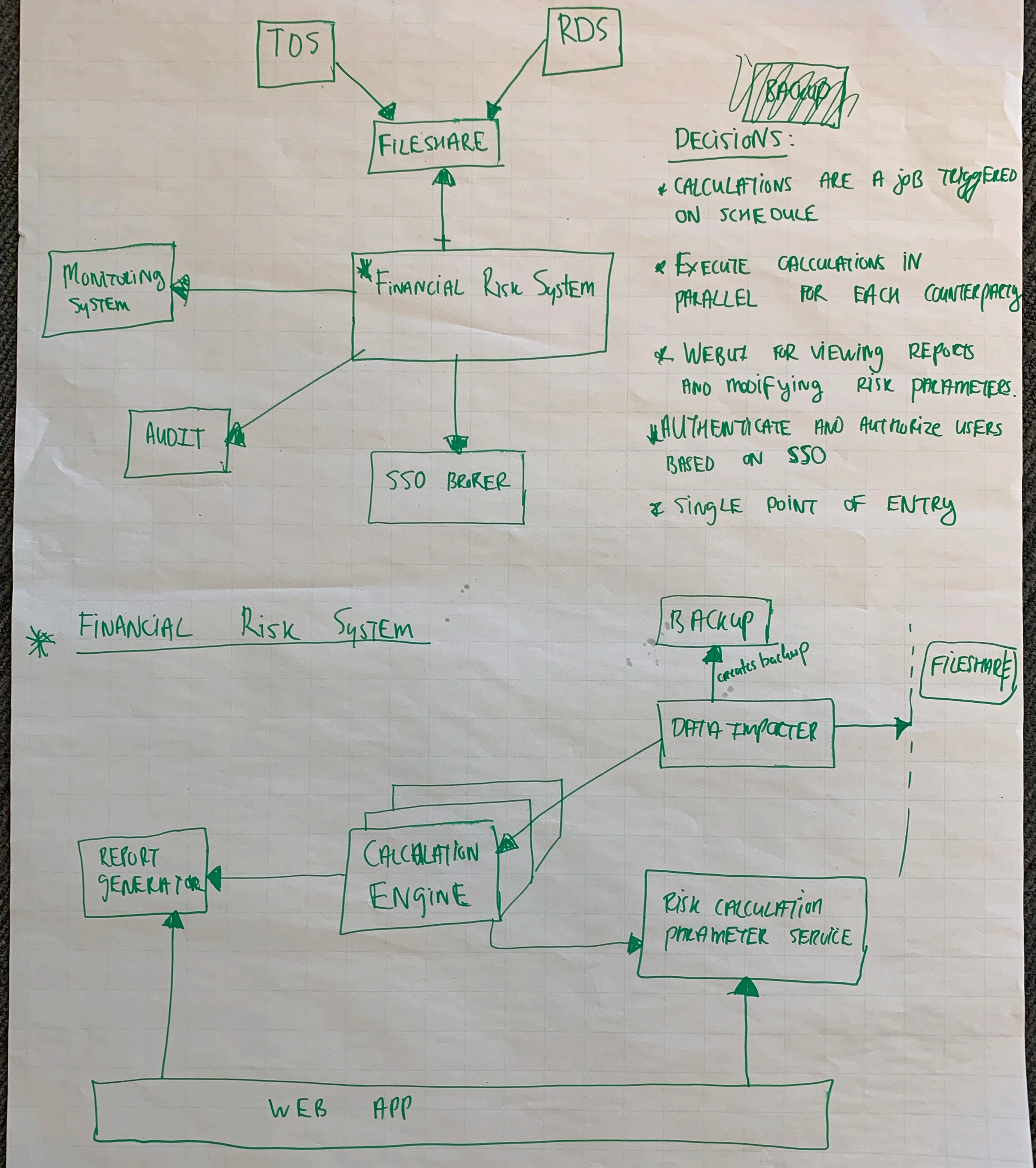
- F/E <> B/E
- Make use of OS' watchdog mechanism
- Data storage ORM. framework: Entity
- ASP .NET B/E
- Angular F/E


FRS
UI
AUTH SERVER
RISK Calculation
Data
Logger
RDS
TDS
<start calculation>
XML
EXT. COMM.
HEART BEAT SERVICE
connects to ALL
application failure
BCMS
SCHEDULER
<collect data>

7

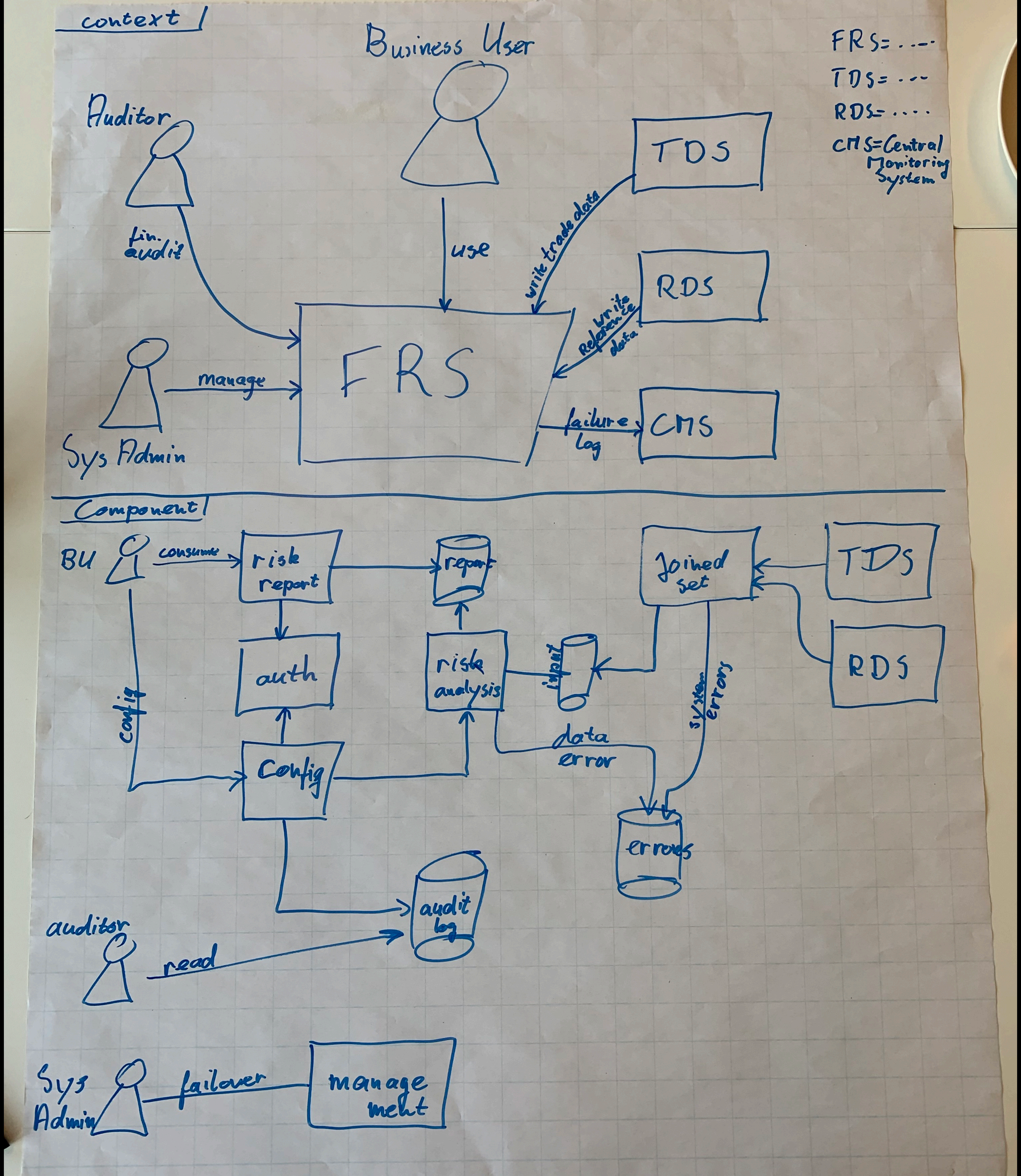

XML
XML
Importer TDS
Importer RDS
Archive / Retention Cleaning
~~MERGERY~~
MERGER
Trigger
Risk Calculator
Error Handling
Log
SNMP
Config
User Management
Report Generator
"User Interface"
LDAP/AD
– Missing
+ Clear Structure

7



DECISIONS:

- CALCULATIONS ARE A JOB TRIGGERED ON SCHEDULE
- EXECUTE CALCULATIONS IN PARALLEL FOR EACH COUNTERPARTY
- WEBUI FOR VIEWING REPORTS AND MODIFYING RISK PARAMETERS.
- AUTHENTICATE AND AUTHORIZE USERS BASED ON SSO
- SINGLE POINT OF ENTRY


context
Business User
Auditor
fin. audit
use
TDS
write trade data
RDS
write Reference data
FRS
manage
Sys Admin
failure log
CMS
FRS= ....
TDS= ...
RDS= ....
CMS=Central Monitoring System
Component
BU
consume
risk report
report
joined set
TDS
auth
risk analysis
input
RDS
config
system errors
data error
Config
errors
audit log
auditor
read
Sys Admin
failover
manage ment

6

Miro
Software Architecture Diagramming

Edrawsoft
Software Architecture Diagram | Ed...

Nulab
What is an architecture diagram, ...

Visual Paradigm Online
Software Architecture Diagram | Visual ...

Medium
Top 9 Architecture diagram software for ...

YouTube
Create Software Architecture Diagrams ...

Lucidchart
Draw 5 Types of Architectural D...

Edrawsoft
Application Architecture Diagram: A ...

ResearchGate
Instrumentation Software Architect...

SlideModel
Four Layers Modern Web Application ...

Lucidchart
Draw 5 Types of Architectural Diagrams ...

Red Hat
5 great diagramming tools for ...

IcePanel - Medium
Top 8 diagramming tools for software ...

LaTeX Stack Exchange
creating software architecture diagram ...

ResearchGate
Software architecture di...

Stack Overflow
tools for architectural diagram ...

predic8
What is Software Architecture

LinkedIn
Software architecture diagramming and ...

If you’re going to use “boxes & lines”, at least do so in a **structured way**, using a **self-describing notation**

Moving fast in the same direction

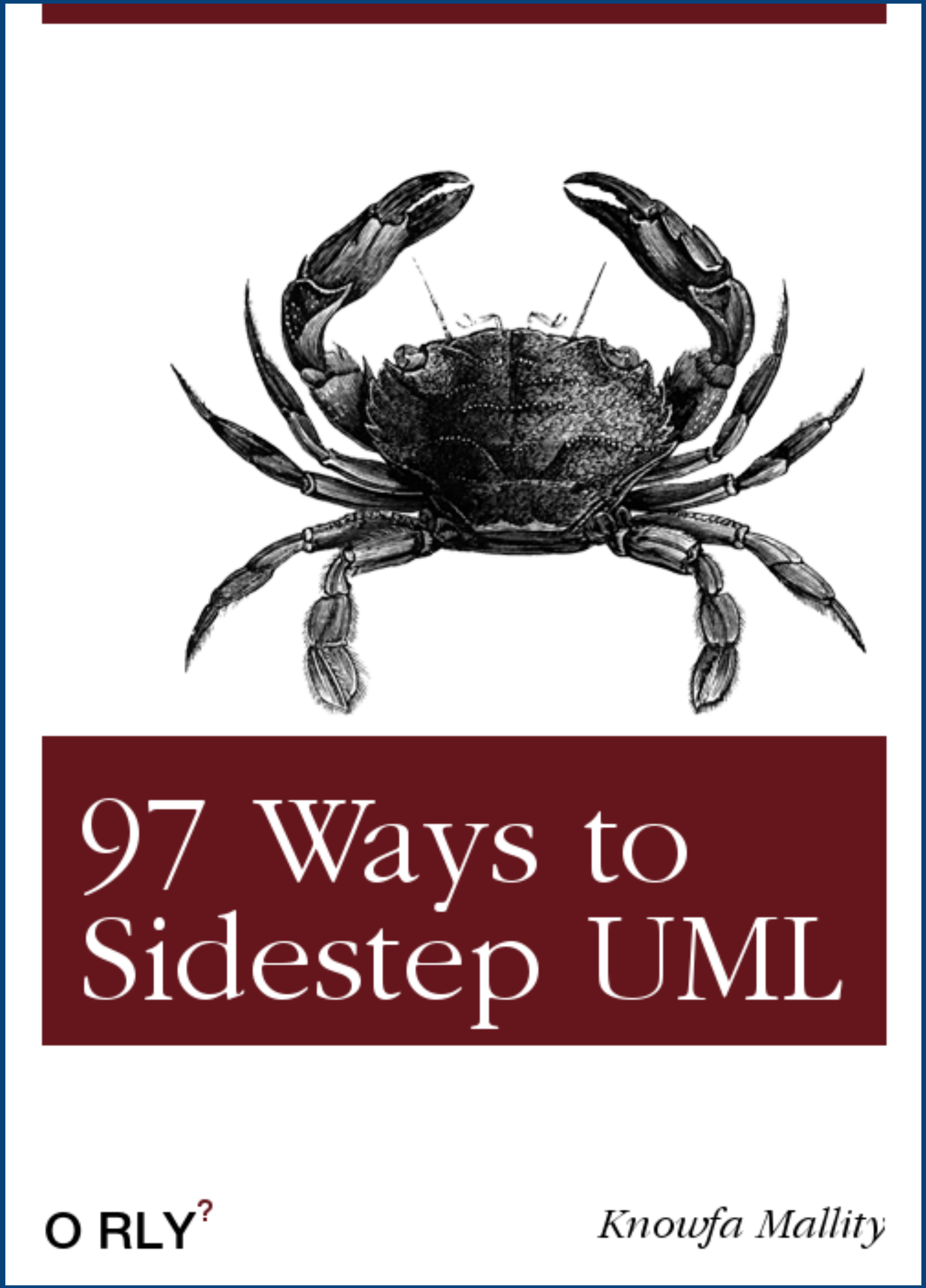
as a team requires

good communication

Do **you** use UML?

In my experience,

few people use UML



#2 “Not everybody else on the team knows it.”

#3 “I’m the only person on the team who knows it.”

#36 “You’ll be seen as old.”

#37 “You’ll be seen as old-fashioned.”

#66 “The tooling sucks.”

#80 “It’s too detailed.”

#81 “It’s a very elaborate waste of time.”

#92 “It’s not expected in agile.”

#97 “The value is in the conversation.”

If you’re using UML, ArchiMate,
SysML, BPML, DFDs, etc
and it’s working … keep doing so!

Who are the **stakeholders** that you need to communicate software architecture to; what **information** do they need?

There are many **different audiences** for diagrams and documentation, all with **different interests**

(software architects, software developers, operations and support staff, testers, Product Owners, project managers, Scrum Masters, users, management, business sponsors, potential customers, potential investors, ...)

The primary use for
diagrams and documentation is
communication and **learning**

Would you code it that way?

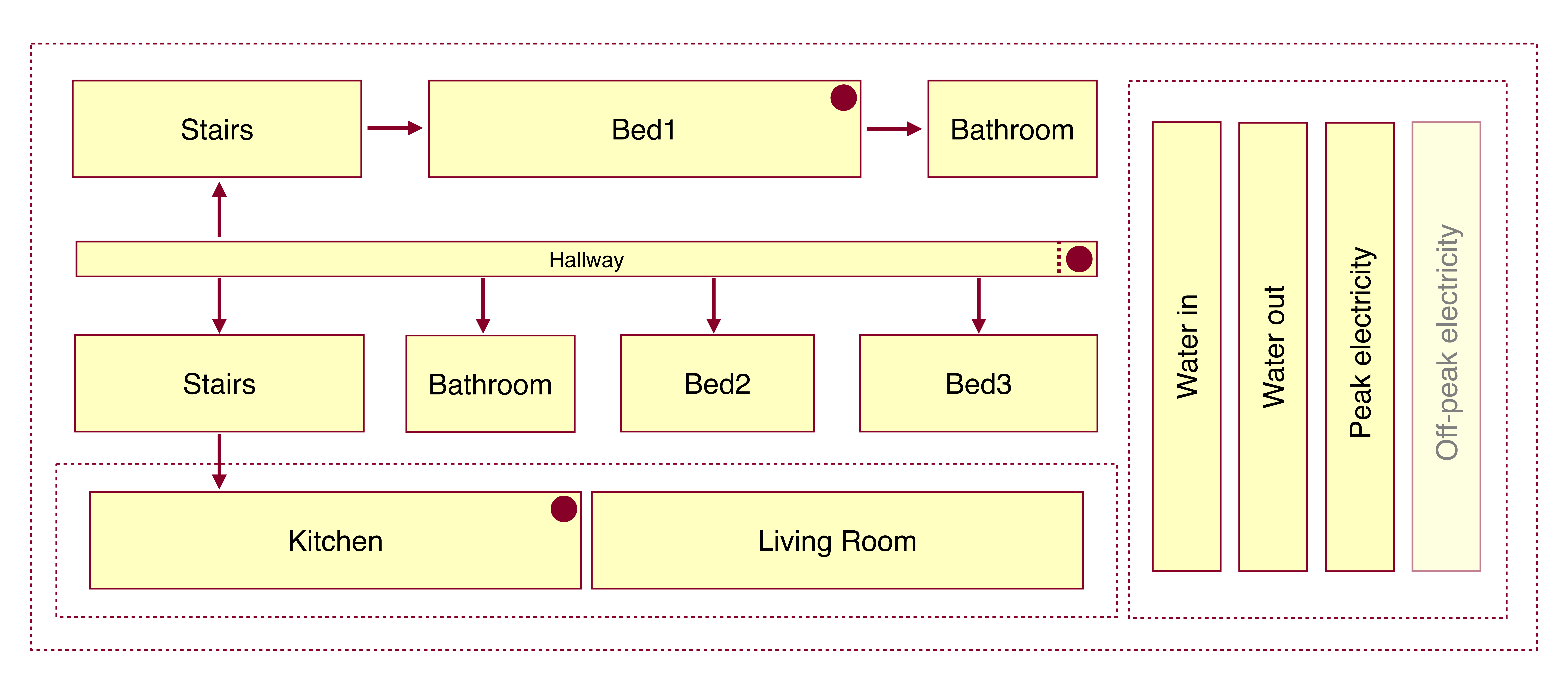
(ensure that your diagrams reflect your implementation intent)

Is that how it really works?

(ensure that your diagrams reflect
your actual codebase)

When drawing software architecture diagrams, think like a software developer

If software developers created building architecture diagrams…



> To describe a software architecture, we use a model composed of multiple views or perspectives.

Architectural Blueprints - The “4+1” View Model of Software Architecture
Philippe Kruchten

The description of an architecture—the decisions made—can be organized around these four views, and then illustrated by a few selected *use cases*, or *scenarios* which become a fifth view. The architecture is in fact partially evolved from these scenarios as we will see later.

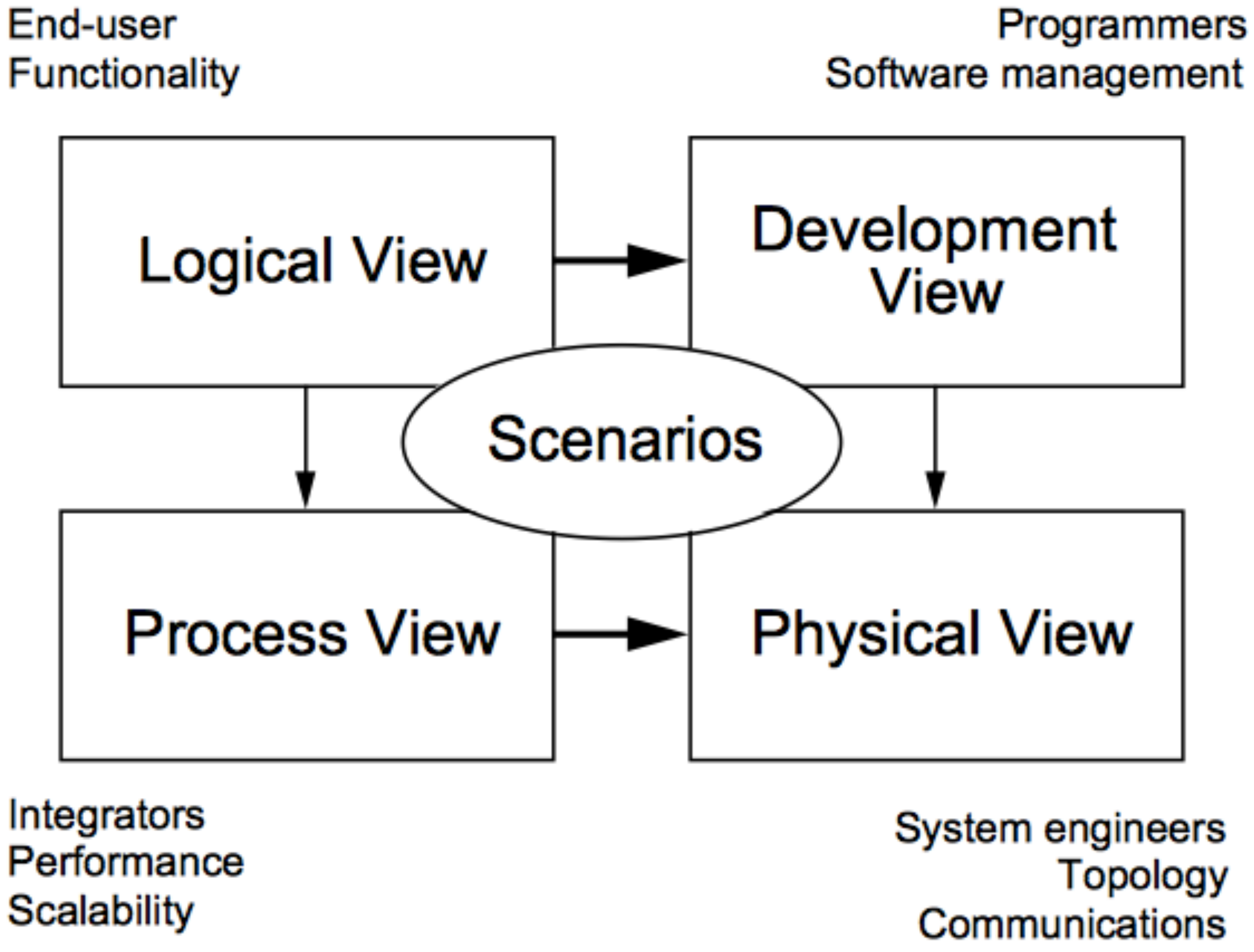
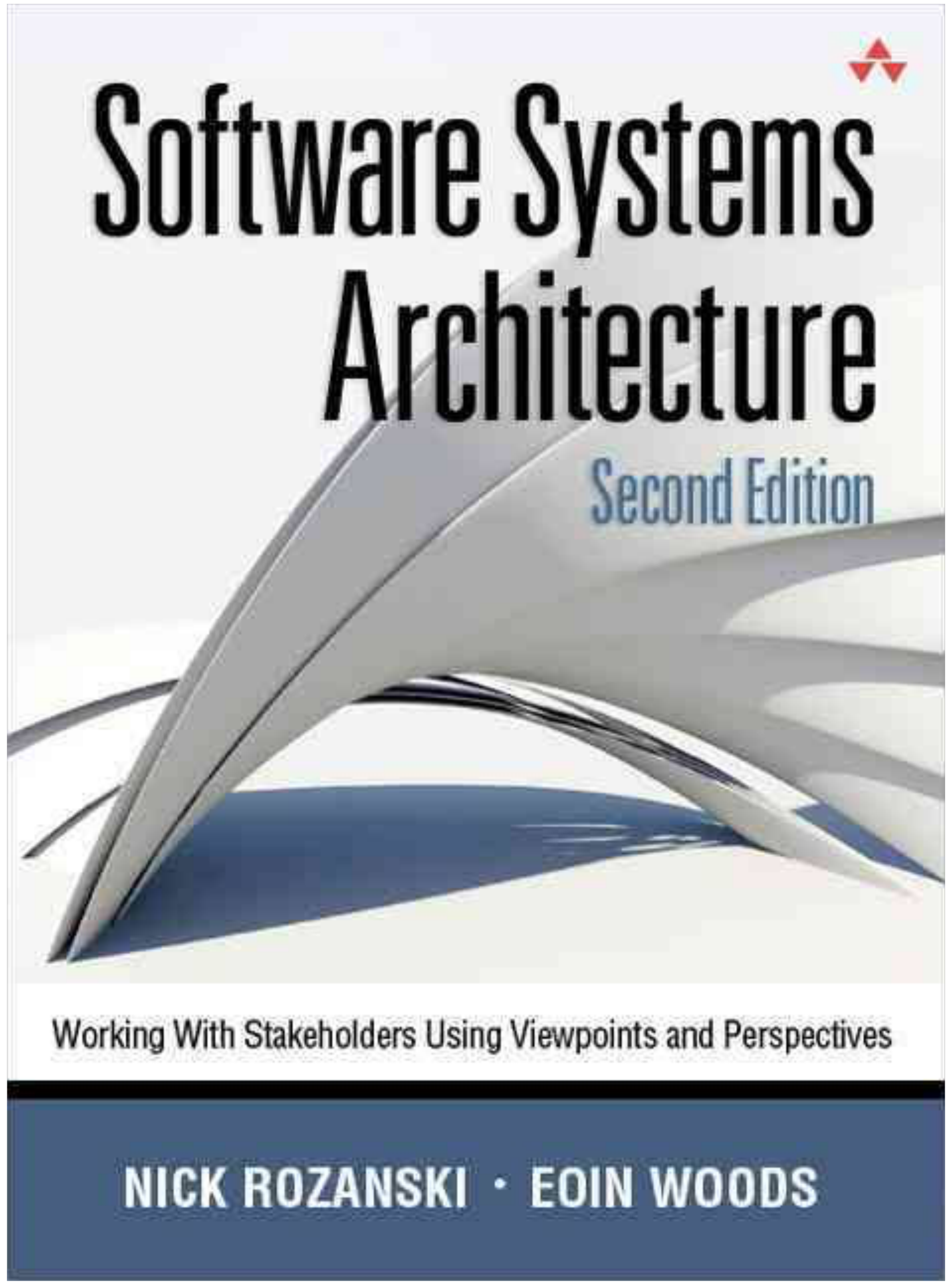


Figure 1 — The “4+1” view model

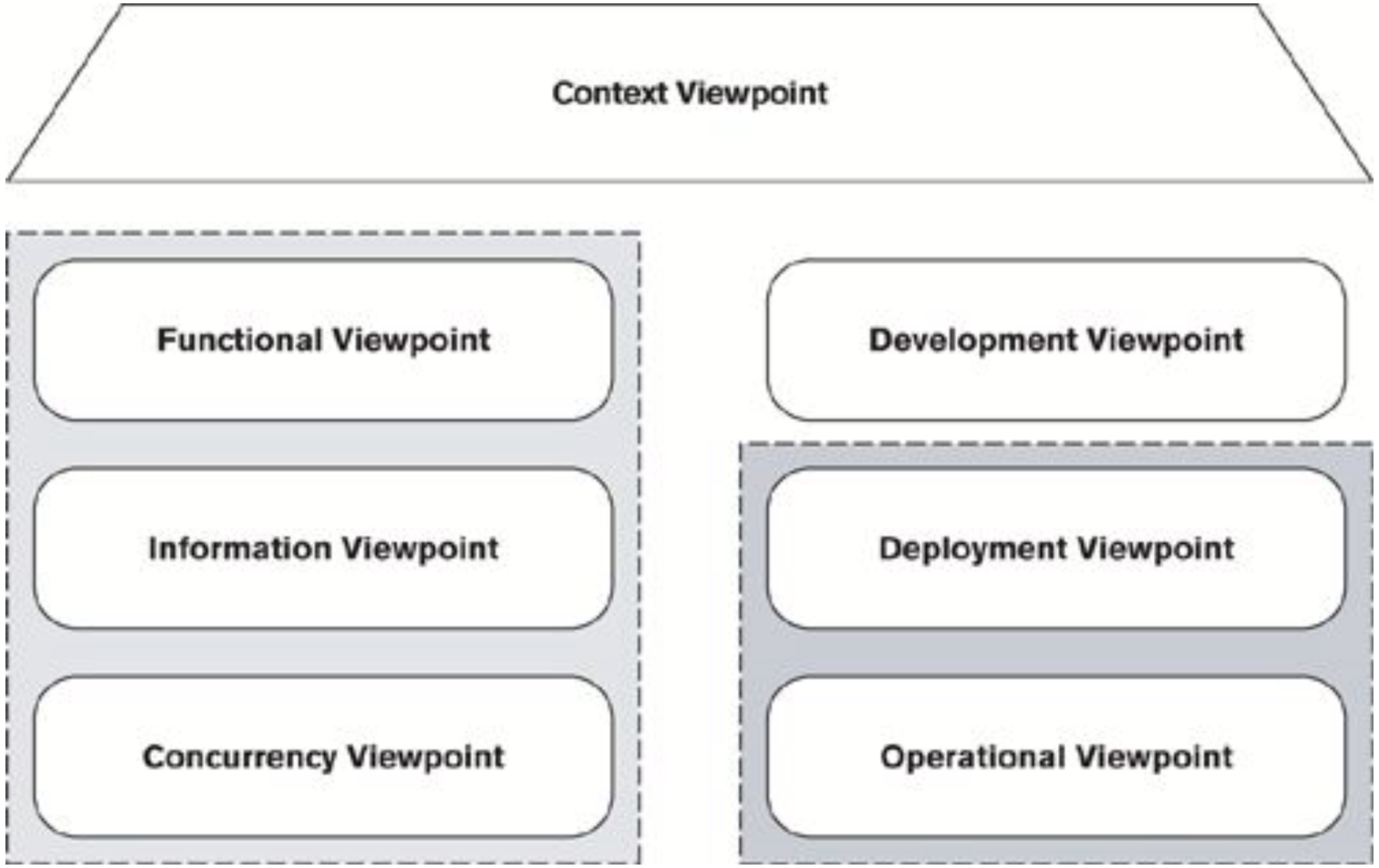


Software Systems Architecture

Second Edition

Working With Stakeholders Using Viewpoints and Perspectives

NICK ROZANSKI · EOIN WOODS



Context Viewpoint

Functional Viewpoint

Information Viewpoint

Concurrency Viewpoint

Development Viewpoint

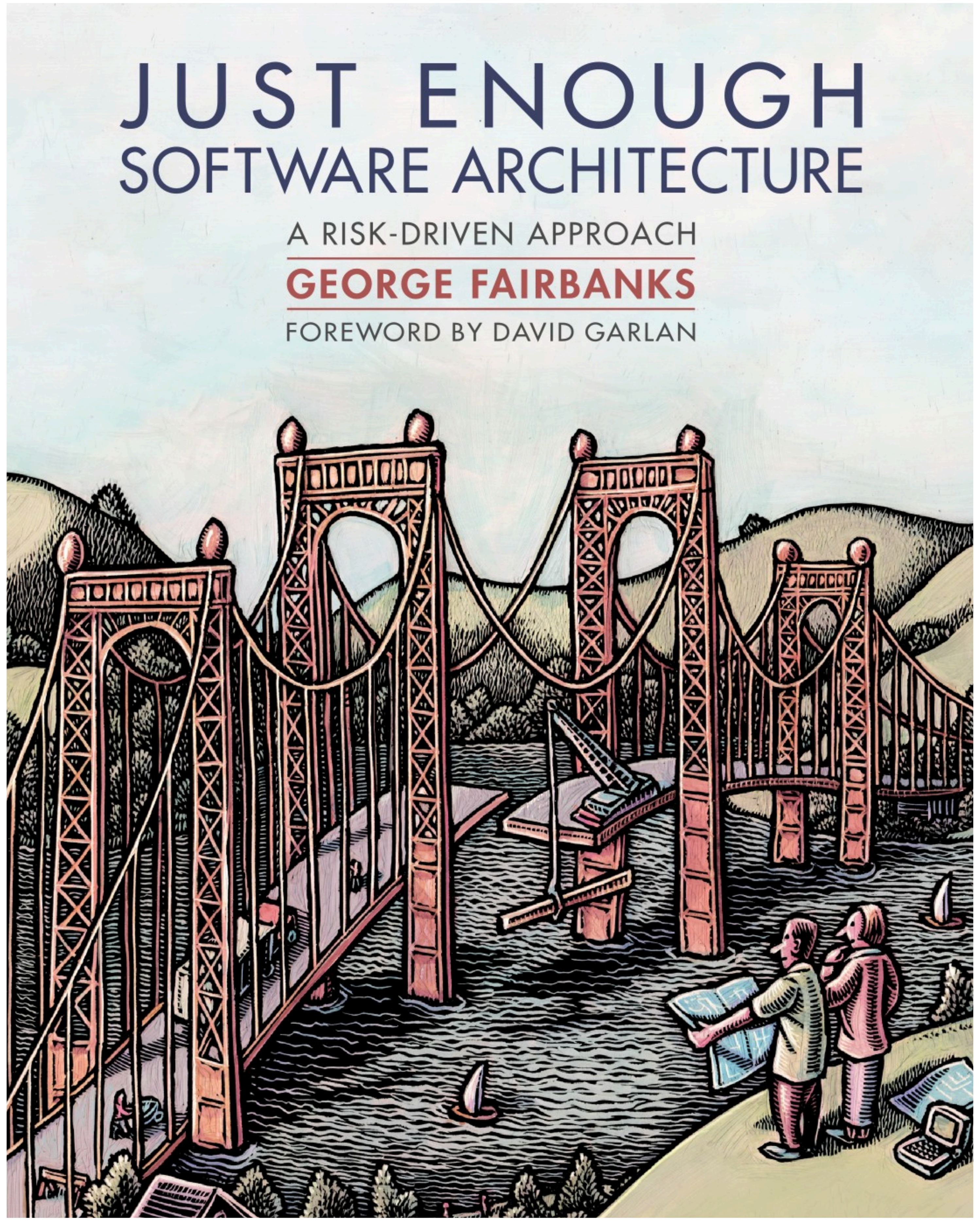
Deployment Viewpoint

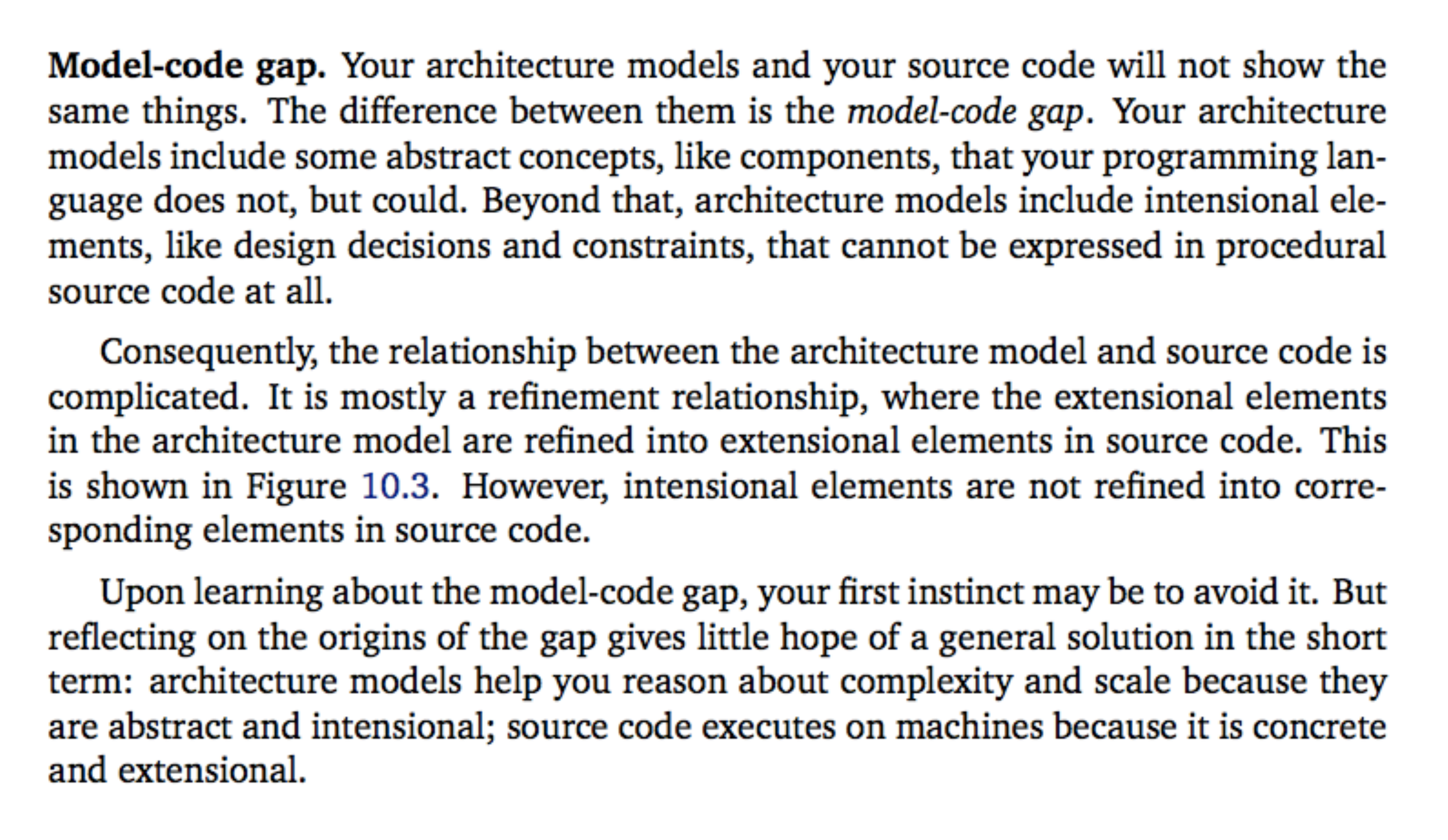
Operational Viewpoint

“Viewpoints and Perspectives”

Why is there a separation between the **logical** and **development** views?

> Our architecture diagrams don’t match the code.



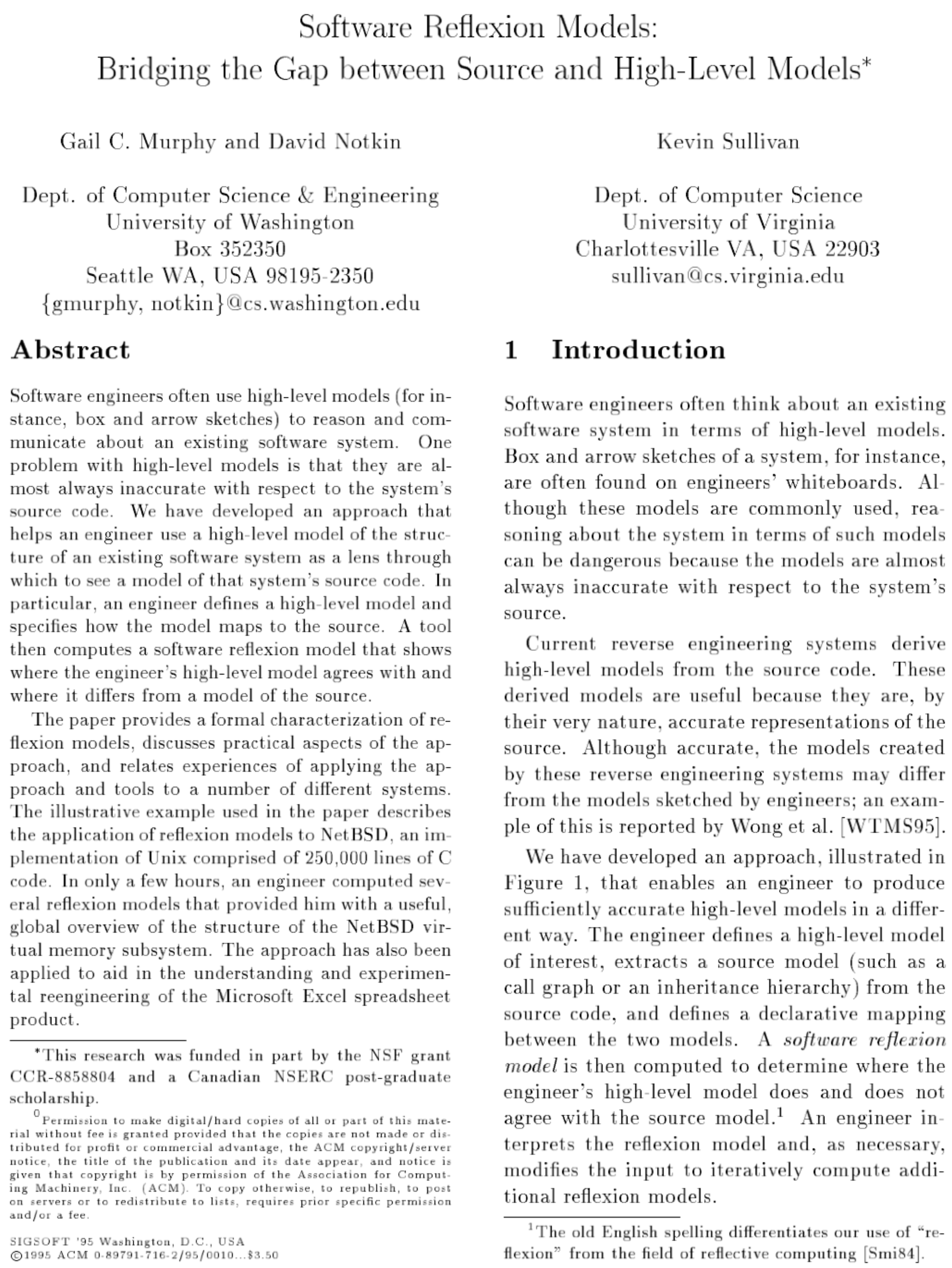


Model-code gap. Your architecture models and your source code will not show the same things. The difference between them is the *model-code gap*. Your architecture models include some abstract concepts, like components, that your programming language does not, but could. Beyond that, architecture models include intensional elements, like design decisions and constraints, that cannot be expressed in procedural source code at all.

Consequently, the relationship between the architecture model and source code is complicated. It is mostly a refinement relationship, where the extensional elements in the architecture model are refined into extensional elements in source code. This is shown in Figure 10.3. However, intensional elements are not refined into corresponding elements in source code.

Upon learning about the model-code gap, your first instinct may be to avoid it. But reflecting on the origins of the gap gives little hope of a general solution in the short term: architecture models help you reason about complexity and scale because they are abstract and intensional; source code executes on machines because it is concrete and extensional.

“model-code gap”



Software Reflexion Models:
Bridging the Gap between Source and High-Level Models*

Gail C. Murphy and David Notkin

Dept. of Computer Science & Engineering
University of Washington
Box 352350
Seattle WA, USA 98195-2350
{gmurphy, notkin}@cs.washington.edu

Kevin Sullivan

Dept. of Computer Science
University of Virginia
Charlottesville VA, USA 22903
sullivan@cs.virginia.edu

Abstract

Software engineers often use high-level models (for instance, box and arrow sketches) to reason and communicate about an existing software system. One problem with high-level models is that they are almost always inaccurate with respect to the system's source code. We have developed an approach that helps an engineer use a high-level model of the structure of an existing software system as a lens through which to see a model of that system's source code. In particular, an engineer defines a high-level model and specifies how the model maps to the source. A tool then computes a software reflexion model that shows where the engineer's high-level model agrees with and where it differs from a model of the source.

The paper provides a formal characterization of reflexion models, discusses practical aspects of the approach, and relates experiences of applying the approach and tools to a number of different systems. The illustrative example used in the paper describes the application of reflexion models to NetBSD, an implementation of Unix comprised of 250,000 lines of C code. In only a few hours, an engineer computed several reflexion models that provided him with a useful, global overview of the structure of the NetBSD virtual memory subsystem. The approach has also been applied to aid in the understanding and experimental reengineering of the Microsoft Excel spreadsheet product.

1 Introduction

Software engineers often think about an existing software system in terms of high-level models. Box and arrow sketches of a system, for instance, are often found on engineers' whiteboards. Although these models are commonly used, reasoning about the system in terms of such models can be dangerous because the models are almost always inaccurate with respect to the system's source.

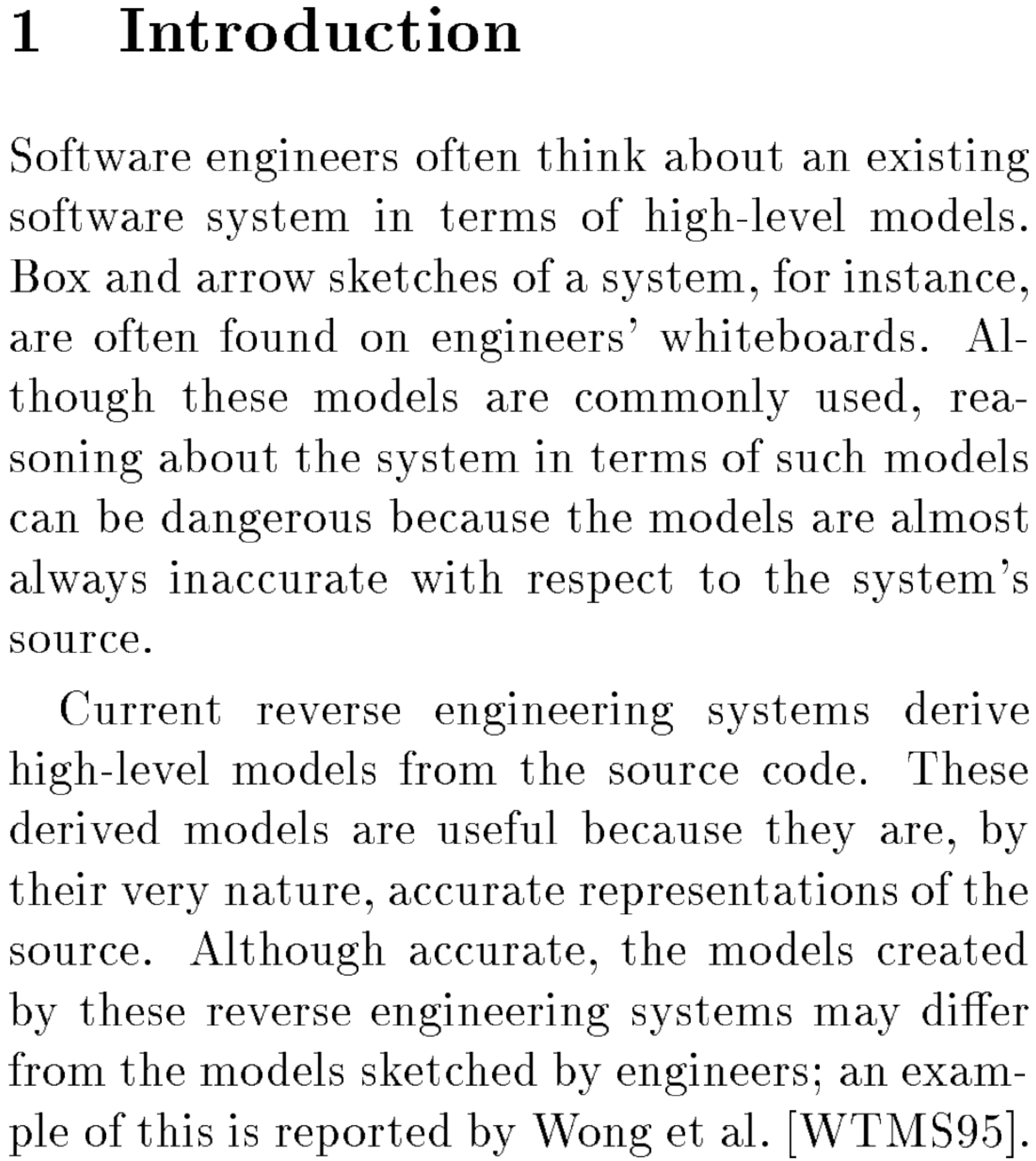
Current reverse engineering systems derive high-level models from the source code. These derived models are useful because they are, by their very nature, accurate representations of the source. Although accurate, the models created by these reverse engineering systems may differ from the models sketched by engineers; an example of this is reported by Wong et al. [WTMS95].

We have developed an approach, illustrated in Figure 1, that enables an engineer to produce sufficiently accurate high-level models in a different way. The engineer defines a high-level model of interest, extracts a source model (such as a call graph or an inheritance hierarchy) from the source code, and defines a declarative mapping between the two models. A *software reflexion model* is then computed to determine where the engineer's high-level model does and does not agree with the source model.[1] An engineer interprets the reflexion model and, as necessary, modifies the input to iteratively compute additional reflexion models.

*This research was funded in part by the NSF grant CCR-8858804 and a Canadian NSERC post-graduate scholarship.

SIGSOFT '95 Washington, D.C., USA
©1995 ACM 0-89791-716-2/95/0010...$3.50

[1] The old English spelling differentiates our use of "reflexion" from the field of reflective computing [Smi84].



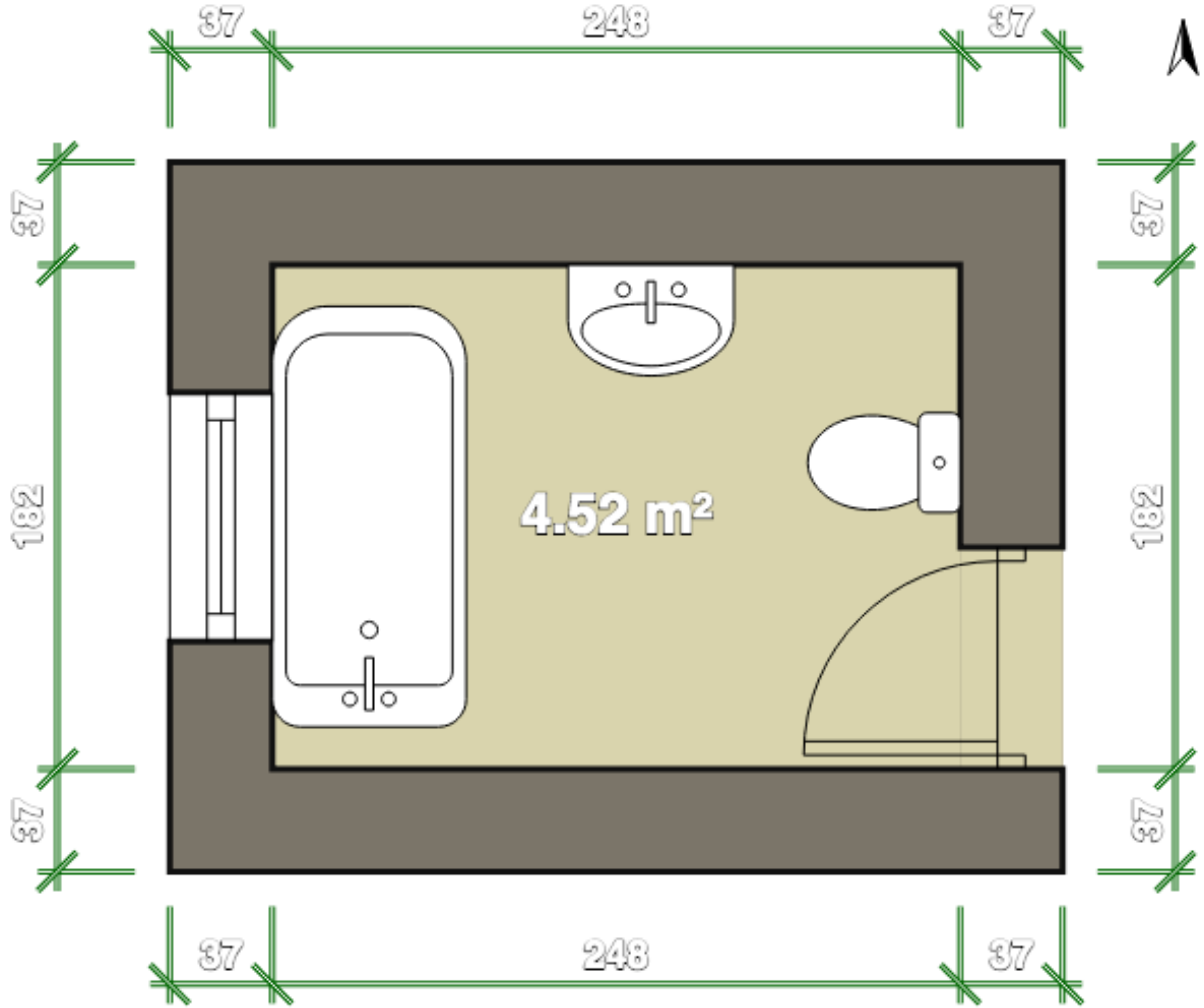
1 Introduction

Software engineers often think about an existing software system in terms of high-level models. Box and arrow sketches of a system, for instance, are often found on engineers' whiteboards. Although these models are commonly used, reasoning about the system in terms of such models can be dangerous because the models are almost always inaccurate with respect to the system's source.

Current reverse engineering systems derive high-level models from the source code. These derived models are useful because they are, by their very nature, accurate representations of the source. Although accurate, the models created by these reverse engineering systems may differ from the models sketched by engineers; an example of this is reported by Wong et al. [WTMS95].

We lack a **common vocabulary**
to describe software architecture

N Circular Rd
don Way
t Park
A4088
A5
A407
Finchley Rd
Holloway
A5200
A503
Green Lanes
STOKE NEWINGTON
A10
A107
A104
Orient Way
A112
A12
Wanstead Flats
A176
N Circular Rd
SEVEN K
Ilford
Ilford Ln
CHALK FARM
CAMDEN TOWN
York Way
A5203
A1
A104
A1207
Forest Ln
Romford Rd
STRATFORD
Green St
Katherine Rd
High St N
A124
Barking
A123
ZSL London Zoo
Albany St
KENSAL GREEN
A404
A5
A501
A501
A1208
High St
Boleyn Ground
A124
High St S
Alfreds W
A1209
A11
A12
A1205
CLERKENWELL
The British Museum
A5
A501
MARYLEBONE
PADDINGTON
A11
Newham Way
A1020
WHITE CITY
SOHO
CITY OF LONDON
WHITECHAPEL
A13
POPLAR
A112
Bayswater Rd
A1203
e Vale
SHEPHERD'S BUSH
Kensington Palace
London
River Thames
Southwark
CANARY WHARF
London City Airport
KENSINGTON
Royal Albert Hall
Buckingham Palace
A112
A219
WESTMINSTER
A2
Lower Rd
Isle of Dogs
River Thames
King St
A315
A3220
LAMBETH
A1206
PIMLICO
VAUXHALL
A200
WOOLWICH
River Thames
A3
A2
A102
A206
Lonsdale Rd
Castelnau
Fulham Palace Rd
A3218
CHELSEA
A3212
FULHAM
A3036
A206
A3220
A3205
A308
Brixton Rd
CAMBERWELL
PECKHAM
A207
A2
A207
A3
A203
A2217
A207
Rd
S Circular Rd
A3205
BATTERSEA
CLAPHAM
BRIXTON
A2
A20
A2213
A3
A208
A2
Priory Ln
A219
A3
A24
Brixton Hill
A214
HERNE HILL
EAST DULWICH
Dulwich Village
A204
LEE
Bexley Rd
Putney Heath
A3
S Circular Rd


37
248
37
37
182
37
37
182
37
4.52 m²
37
248
37

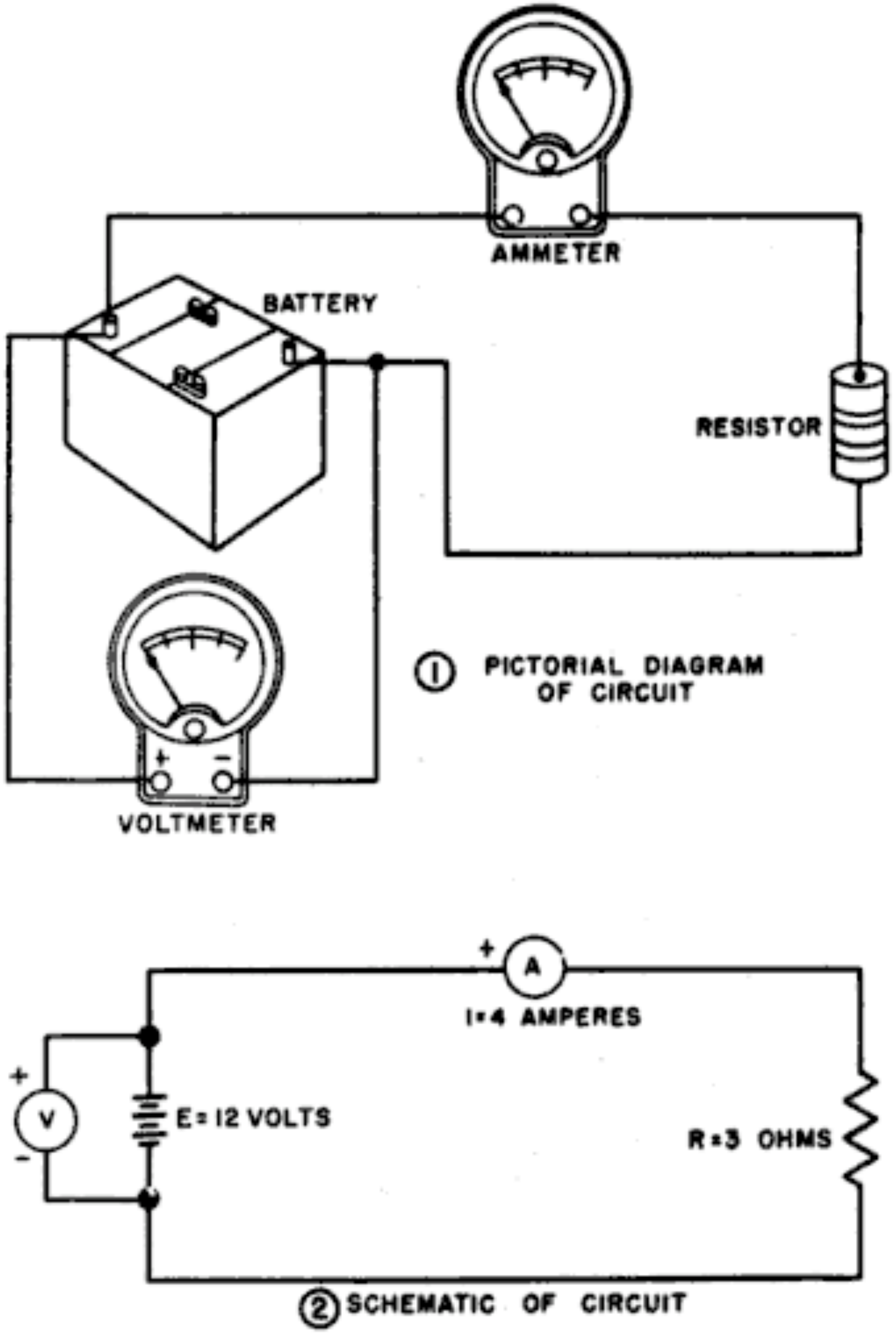
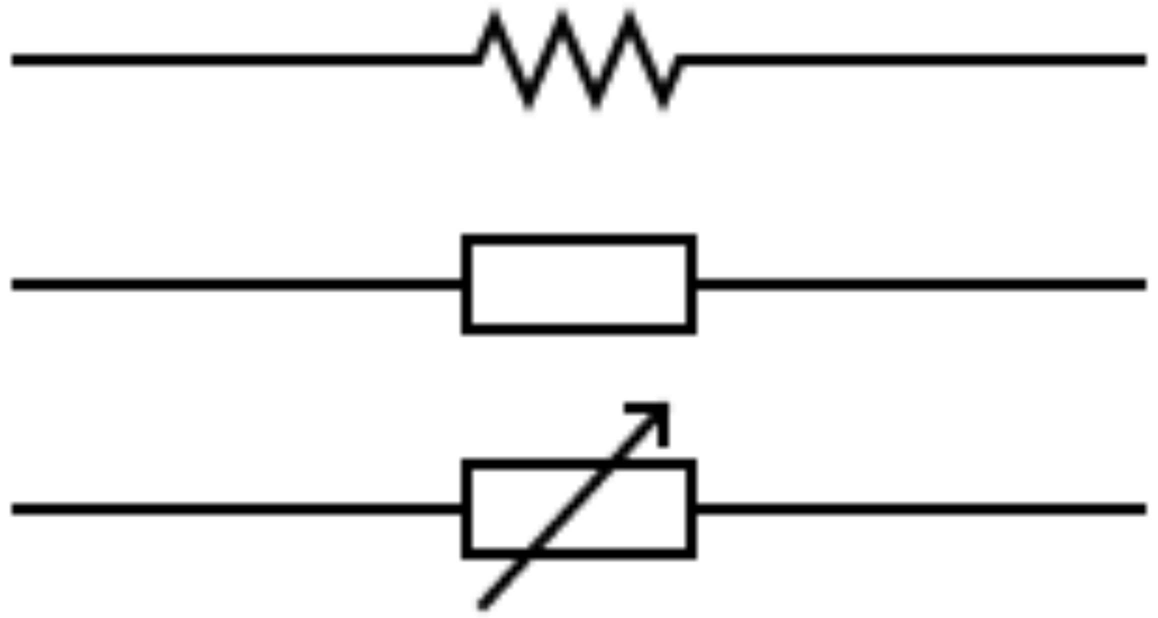
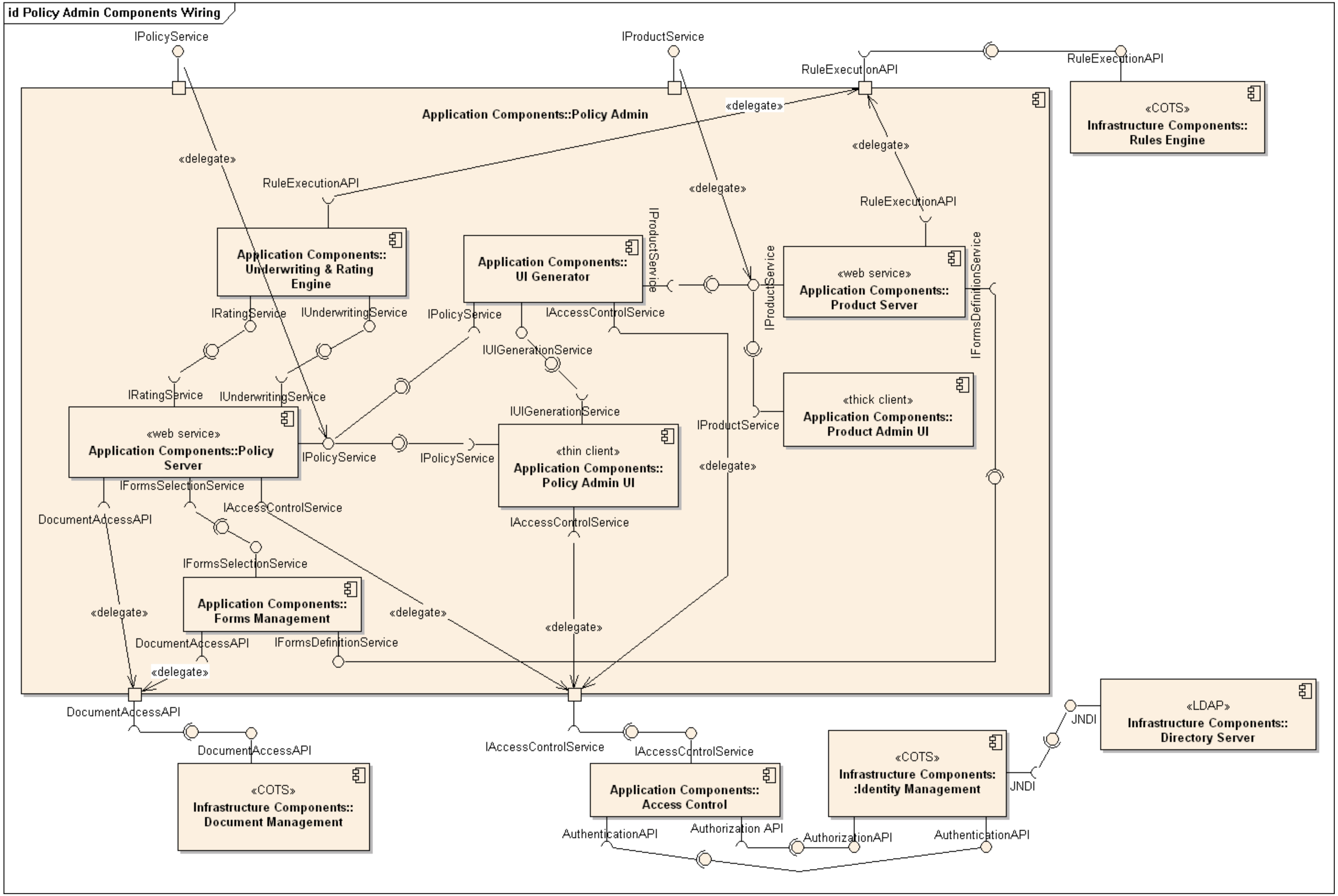


Figure 48. Diagram of a basic circuit.



https://en.wikipedia.org/wiki/Circuit_diagram

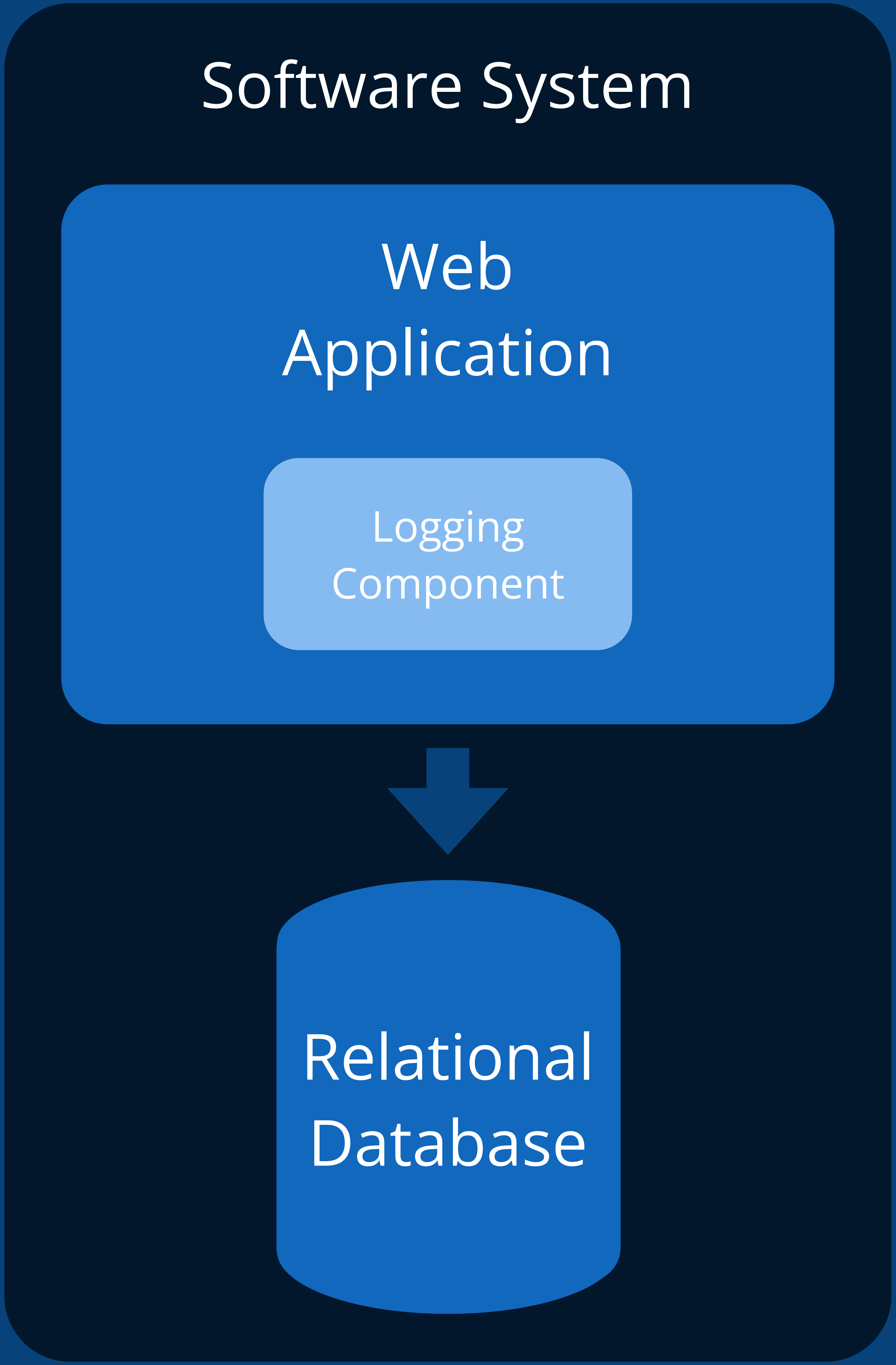


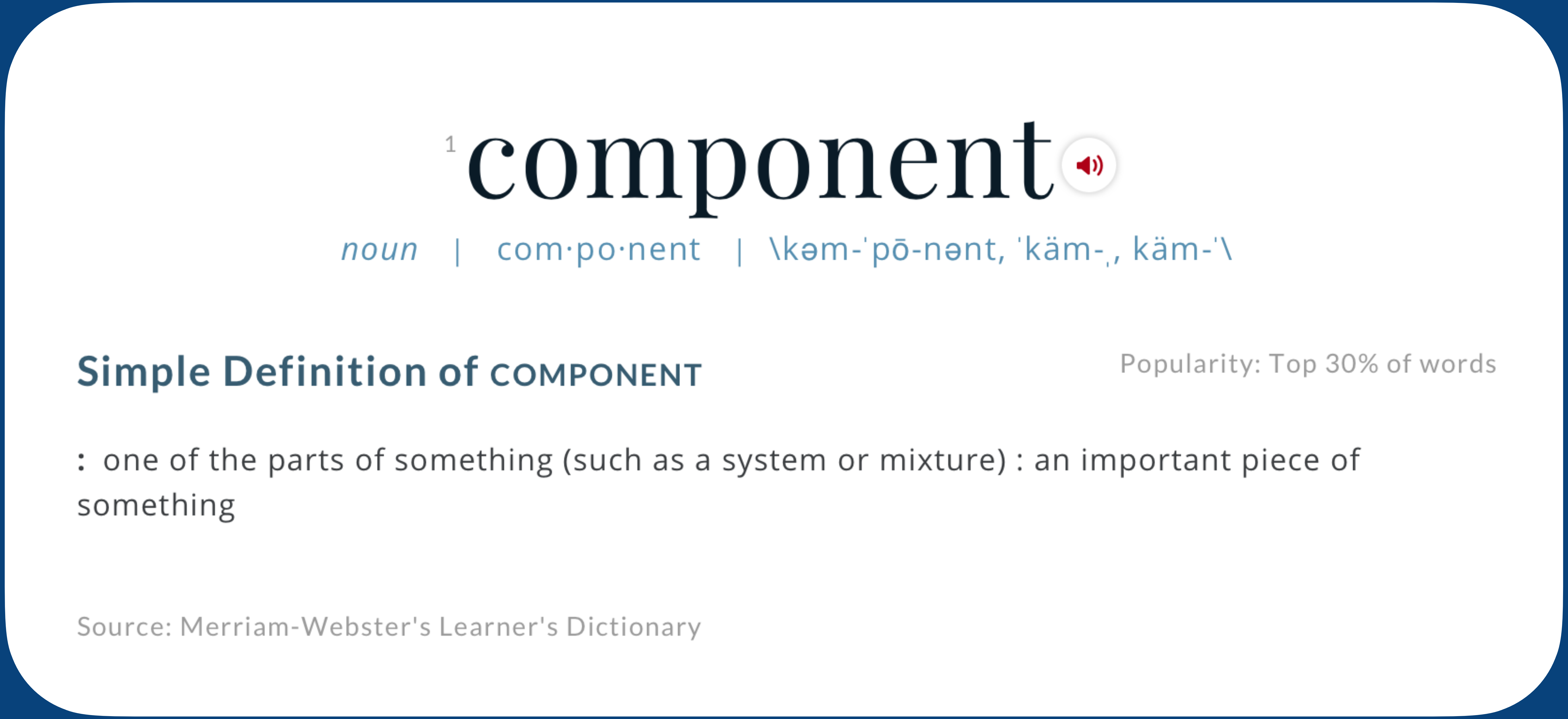
https://en.wikipedia.org/wiki/Component_diagram

Component

a modular unit with well-defined Interfaces that is replaceable within its environment

https://www.omg.org/spec/UML/2.5.1/PDF





[1] component

noun | com·po·nent | \kəm-ˈpō-nənt, ˈkäm-ˌ, käm-ˈ\

Simple Definition of COMPONENT

Popularity: Top 30% of words

: one of the parts of something (such as a system or mixture) : an important piece of something

Source: Merriam-Webster's Learner's Dictionary

Ubiquitous language

A **common set of abstractions**
is more important
than a common notation

Abstractions

Software System

Container
lient-side web app, server-side web app, console application,
obile app, database schema, file system, object store, etc)

Container
(e.g. client-side web app, server-side web app, console application,
mobile app, database schema, file system, object store, etc)

Container
(e.g. client-side web app, server-side web app, console applic
mobile app, database schema, file system, object store, e

Component

Component

Component

Code

Code

Code

A **software system** is made up of one or more **containers** (applications and data stores), each of which contains one or more **components**, which in turn are implemented by one or more **code** elements (classes, interfaces, objects, functions, etc).

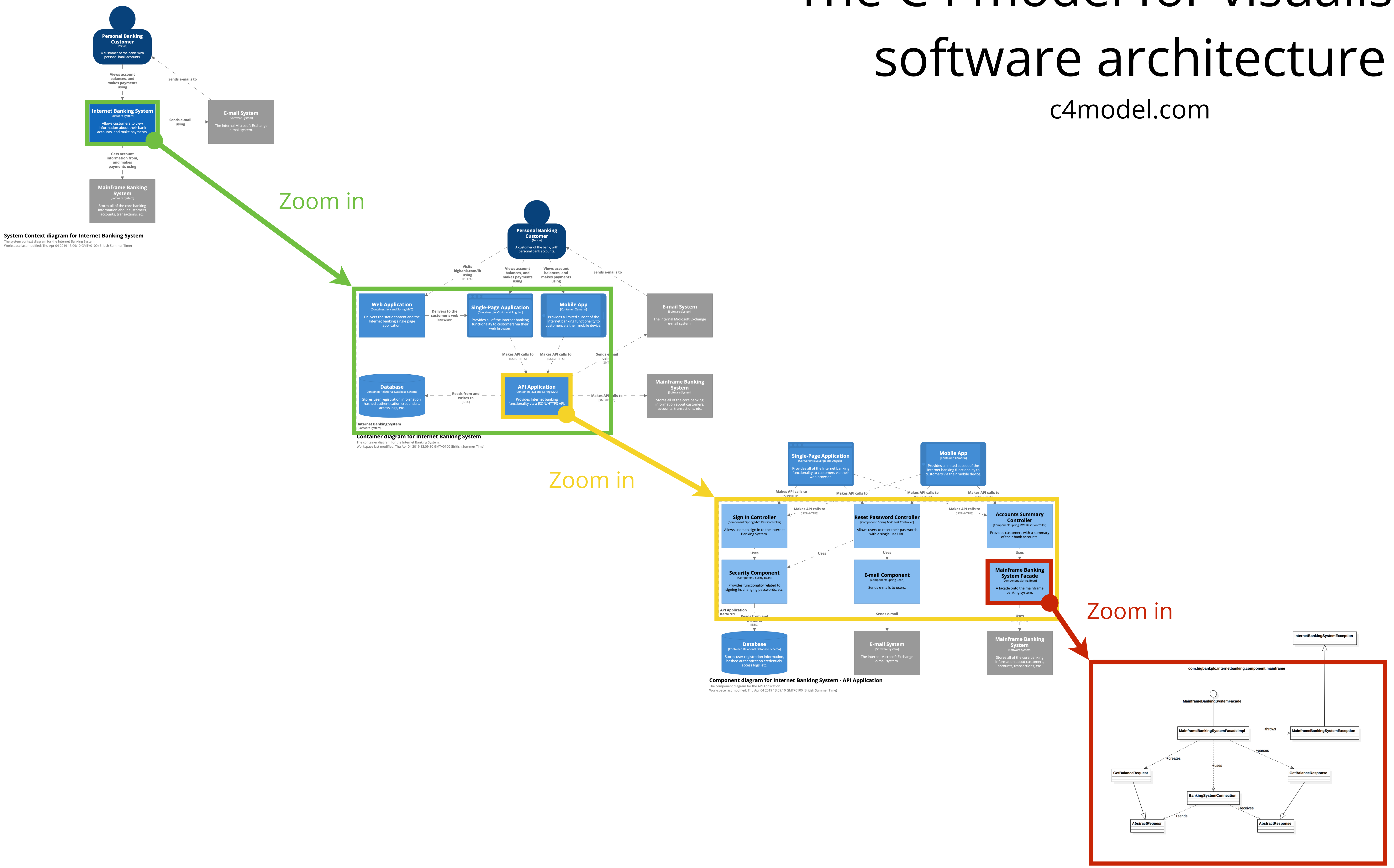
Static structure diagrams

C4

c4model.com

The C4 model for visualising software architecture

c4model.com

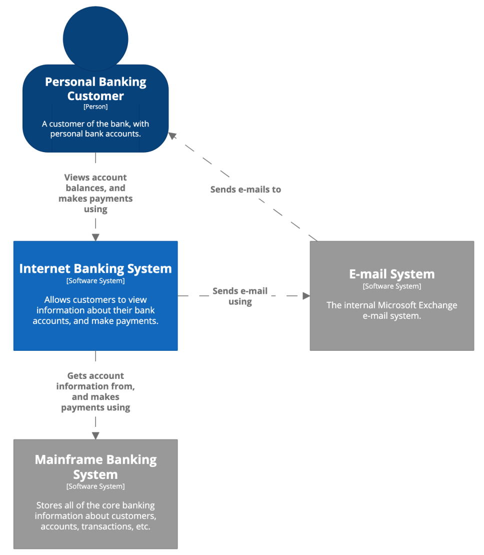

Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Views account balances, and makes payments using
Sends e-mails to
Internet Banking System
[Software System]
Allows customers to view information about their bank accounts, and make payments.
Sends e-mail using
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Gets account information from, and makes payments using
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
System Context diagram for Internet Banking System
The system context diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)
Zoom in
Visits bigbank.com/ib using
[HTTPS]
Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.
Delivers to the customer's web browser
Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.
Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.
Makes API calls to
[JSON/HTTPS]
Database
[Container: Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Reads from and writes to
[JDBC]
API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Internet Banking System
[Software System]
Container diagram for Internet Banking System
The container diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)
Zoom in
Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.
Reset Password Controller
[Component: Spring MVC Rest Controller]
Allows users to reset their passwords with a single use URL.
Accounts Summary Controller
[Component: Spring MVC Rest Controller]
Provides customers with a summary of their bank accounts.
Uses
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
E-mail Component
[Component: Spring Bean]
Sends e-mails to users.
Mainframe Banking System Facade
[Component: Spring Bean]
A facade onto the mainframe banking system.
API Application
[Container]
Sends e-mail
Component diagram for Internet Banking System - API Application
The component diagram for the API Application.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)
Zoom in
InternetBankingSystemException
com.bigbankplc.internetbanking.component.mainframe
MainframeBankingSystemFacade
MainframeBankingSystemFacadeImpl
+throws
MainframeBankingSystemException
+creates
+uses
+parses
GetBalanceRequest
GetBalanceResponse
BankingSystemConnection
+receives
+sends
AbstractRequest
AbstractResponse

Level 1
Context

Level 2
Containers

Level 3
Components

Level 4
Code



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Views account balances, and makes payments using

Sends e-mails to

Internet Banking System
[Software System]
Allows customers to view information about their bank accounts, and make payments.

Sends e-mail using

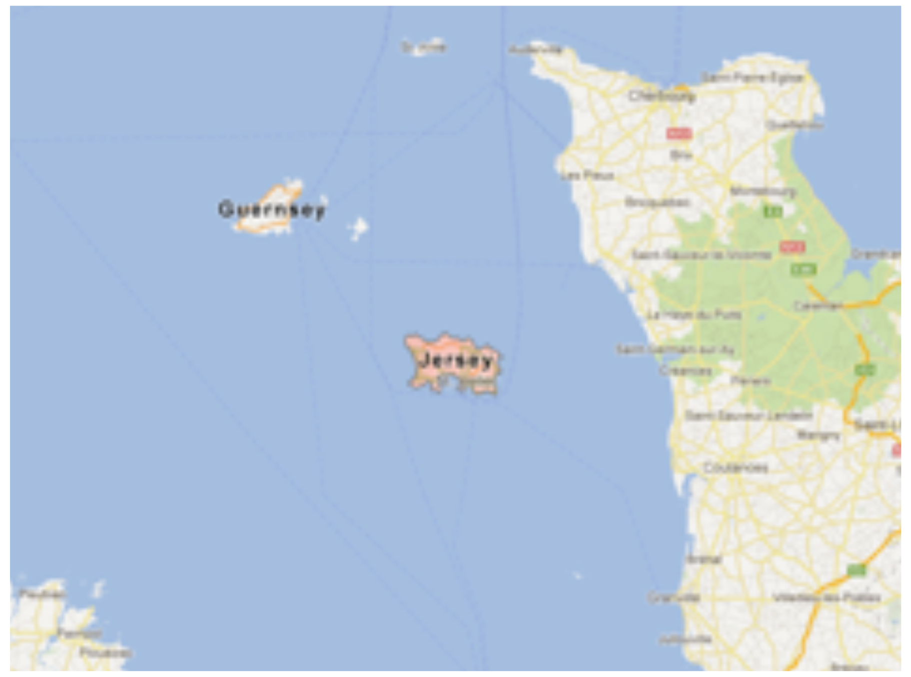
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

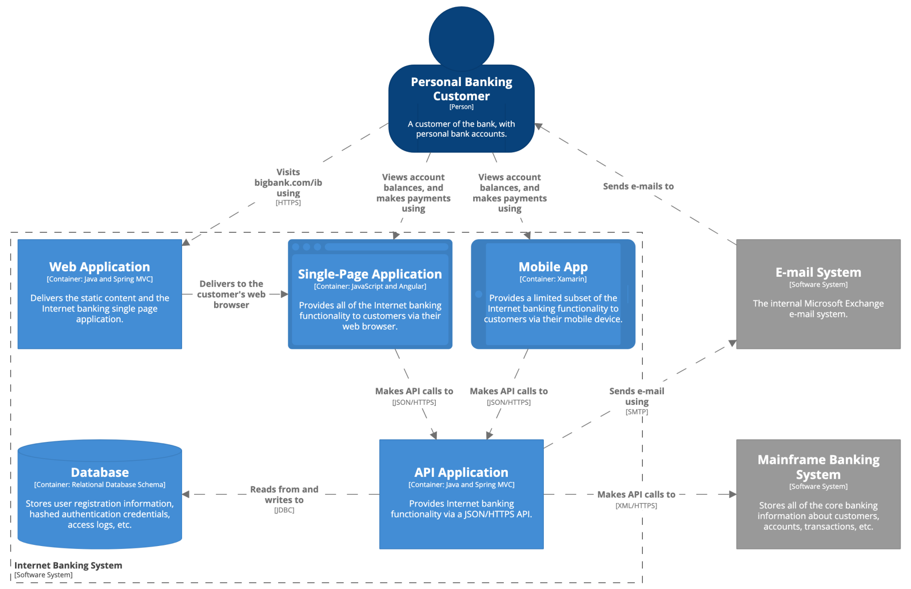
Gets account information from, and makes payments using

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)





Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Visits bigbank.com/ib using
[HTTPS]

Views account balances, and makes payments using

Views account balances, and makes payments using

Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Delivers to the customer's web browser

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sends e-mail using
[SMTP]

Database
[Container: Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

Reads from and writes to
[JDBC]

API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

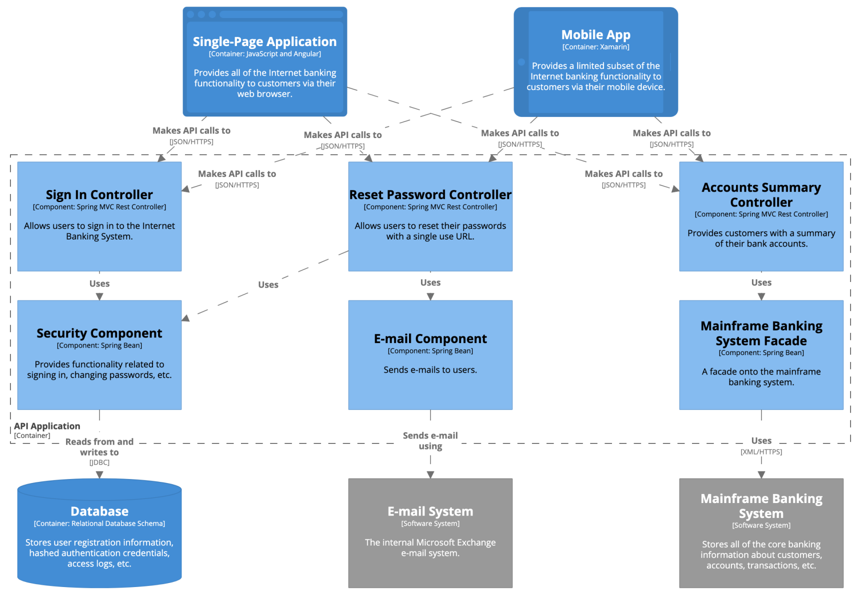
Makes API calls to
[XML/HTTPS]

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

Container diagram for Internet Banking System

The container diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)



Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.

Reset Password Controller
[Component: Spring MVC Rest Controller]
Allows users to reset their passwords with a single use URL.

Accounts Summary Controller
[Component: Spring MVC Rest Controller]
Provides customers with a summary of their bank accounts.

Uses

Uses

Uses

Uses

Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

E-mail Component
[Component: Spring Bean]
Sends e-mails to users.

Mainframe Banking System Facade
[Component: Spring Bean]
A facade onto the mainframe banking system.

API Application
[Container]

Reads from and writes to
[JDBC]

Sends e-mail using

Uses
[XML/HTTPS]

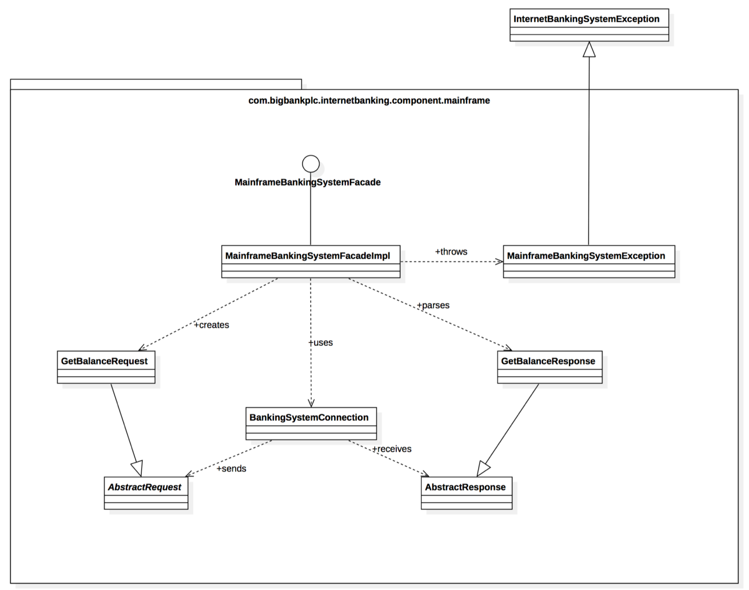
Database
[Container: Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Component diagram for Internet Banking System - API Application

The component diagram for the API Application.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)


InternetBankingSystemException
com.bigbankplc.internetbanking.component.mainframe
MainframeBankingSystemFacade
MainframeBankingSystemFacadeImpl
+throws
MainframeBankingSystemException
+creates
+uses
+parses
GetBalanceRequest
GetBalanceResponse
BankingSystemConnection
+receives
+sends
AbstractRequest
AbstractResponse

Diagrams are maps

that help software developers navigate a large and/or complex codebase

1. System Context

The system plus users and system dependencies.

2. Containers

The overall shape of the architecture and technology choices.

3. Components

Logical components and their interactions within a container.

4. Code (e.g. classes)

Component implementation details.

Overview first

Zoom & filter

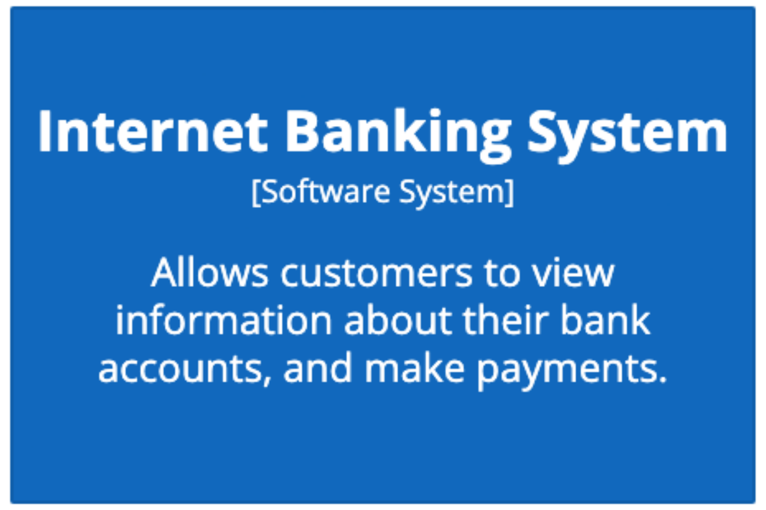
Details on demand

Example

(Internet Banking System)

Level 1

System Context diagram



Internet Banking System

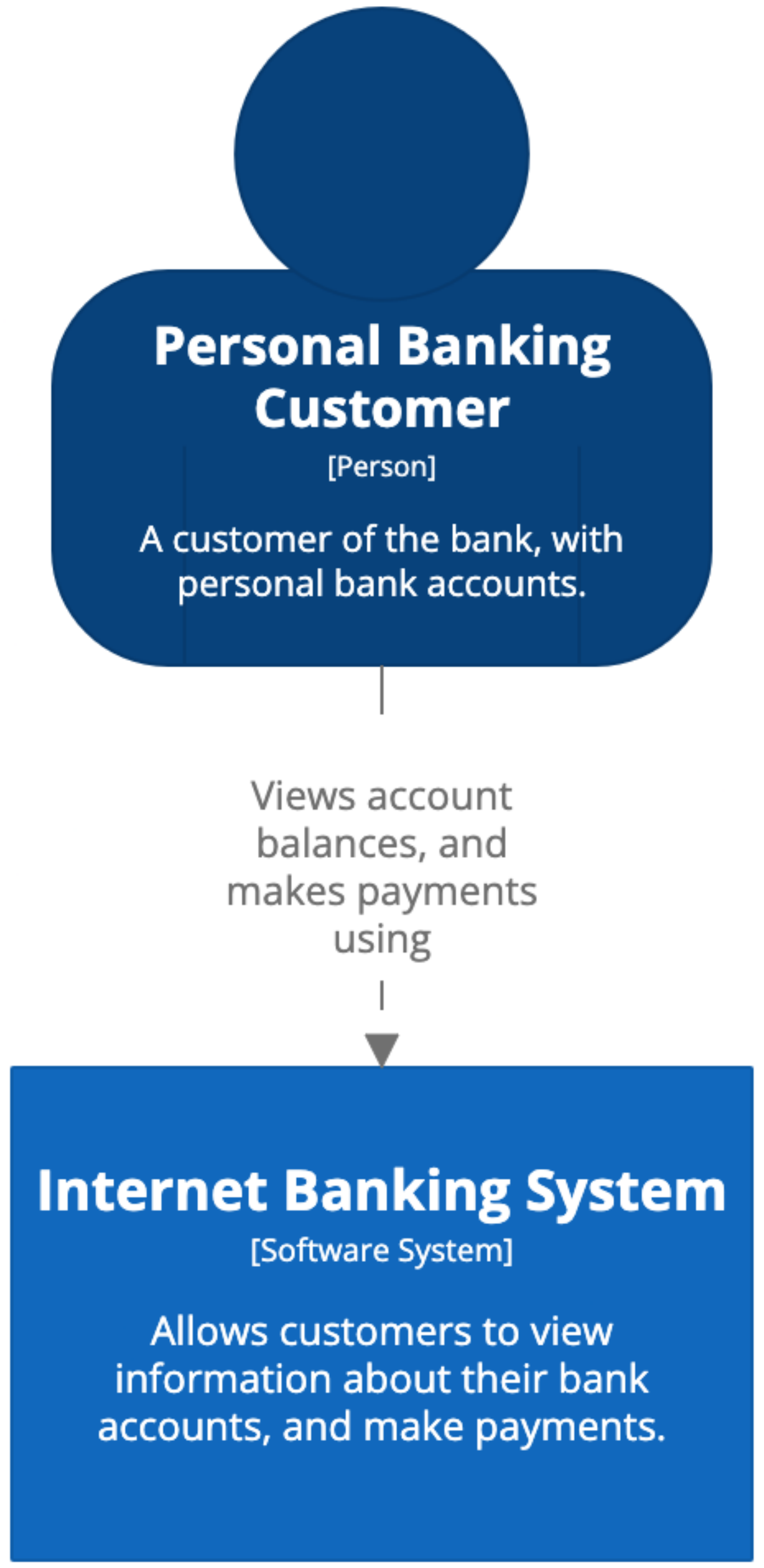
[Software System]

Allows customers to view information about their bank accounts, and make payments.

[System Context] Internet Banking System

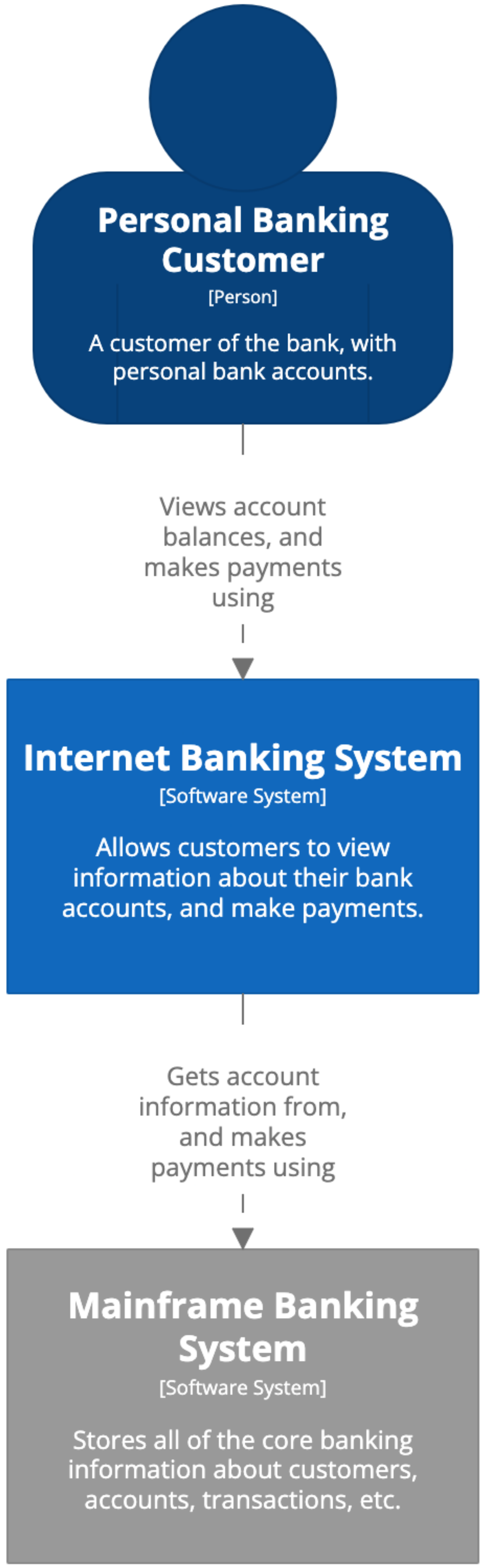
The system context diagram for the Internet Banking System.

Monday, 27 February 2023 at 15:25 Greenwich Mean Time


Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Views account balances, and makes payments using
Internet Banking System
[Software System]
Allows customers to view information about their bank accounts, and make payments.

[System Context] Internet Banking System

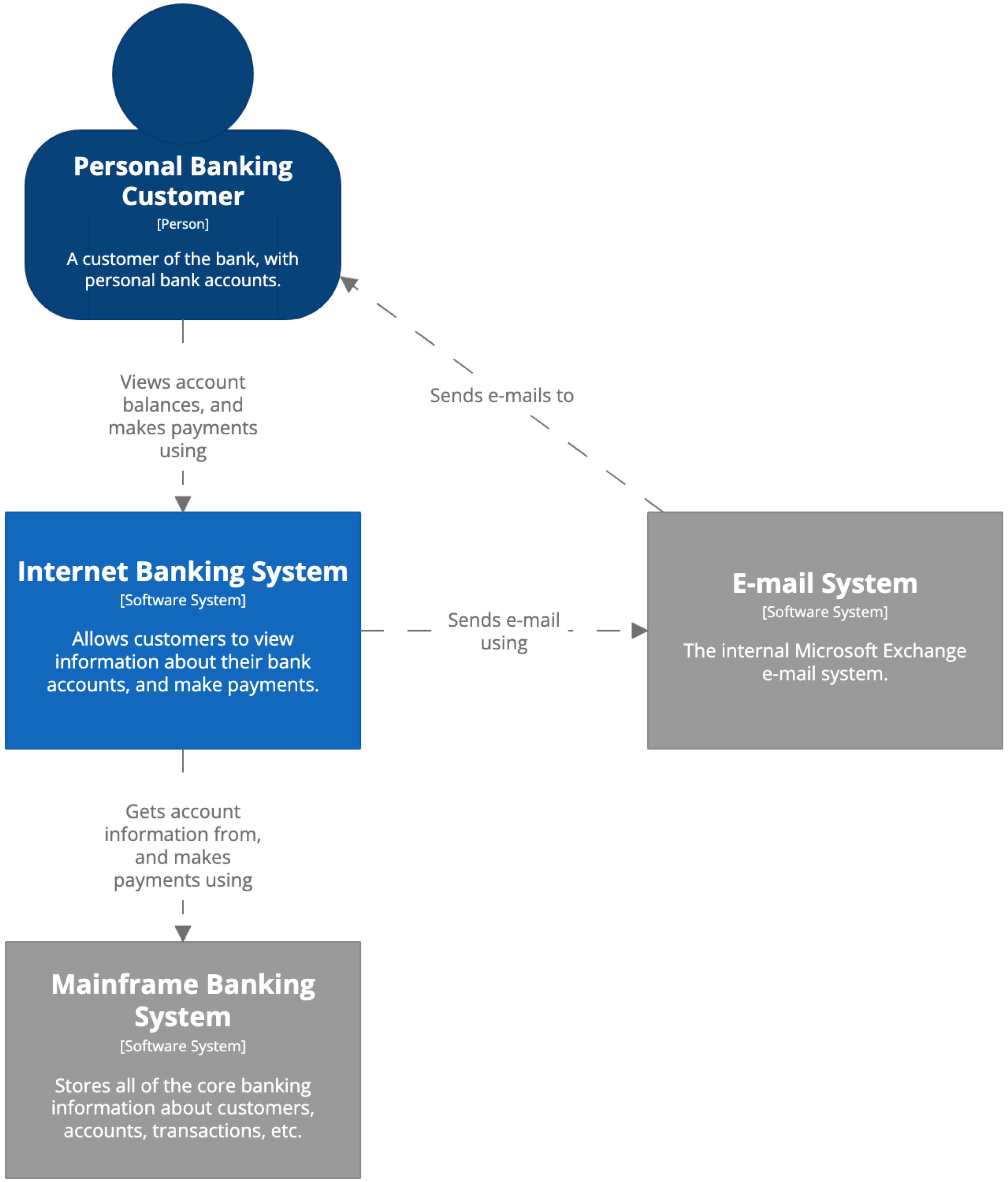
The system context diagram for the Internet Banking System.
Monday, 27 February 2023 at 15:25 Greenwich Mean Time


Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Views account balances, and makes payments using
Internet Banking System
[Software System]
Allows customers to view information about their bank accounts, and make payments.
Gets account information from, and makes payments using
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

[System Context] Internet Banking System

The system context diagram for the Internet Banking System.

Monday, 27 February 2023 at 15:25 Greenwich Mean Time


Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Views account balances, and makes payments using
Sends e-mails to
Internet Banking System
[Software System]
Allows customers to view information about their bank accounts, and make payments.
Sends e-mail using
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Gets account information from, and makes payments using
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

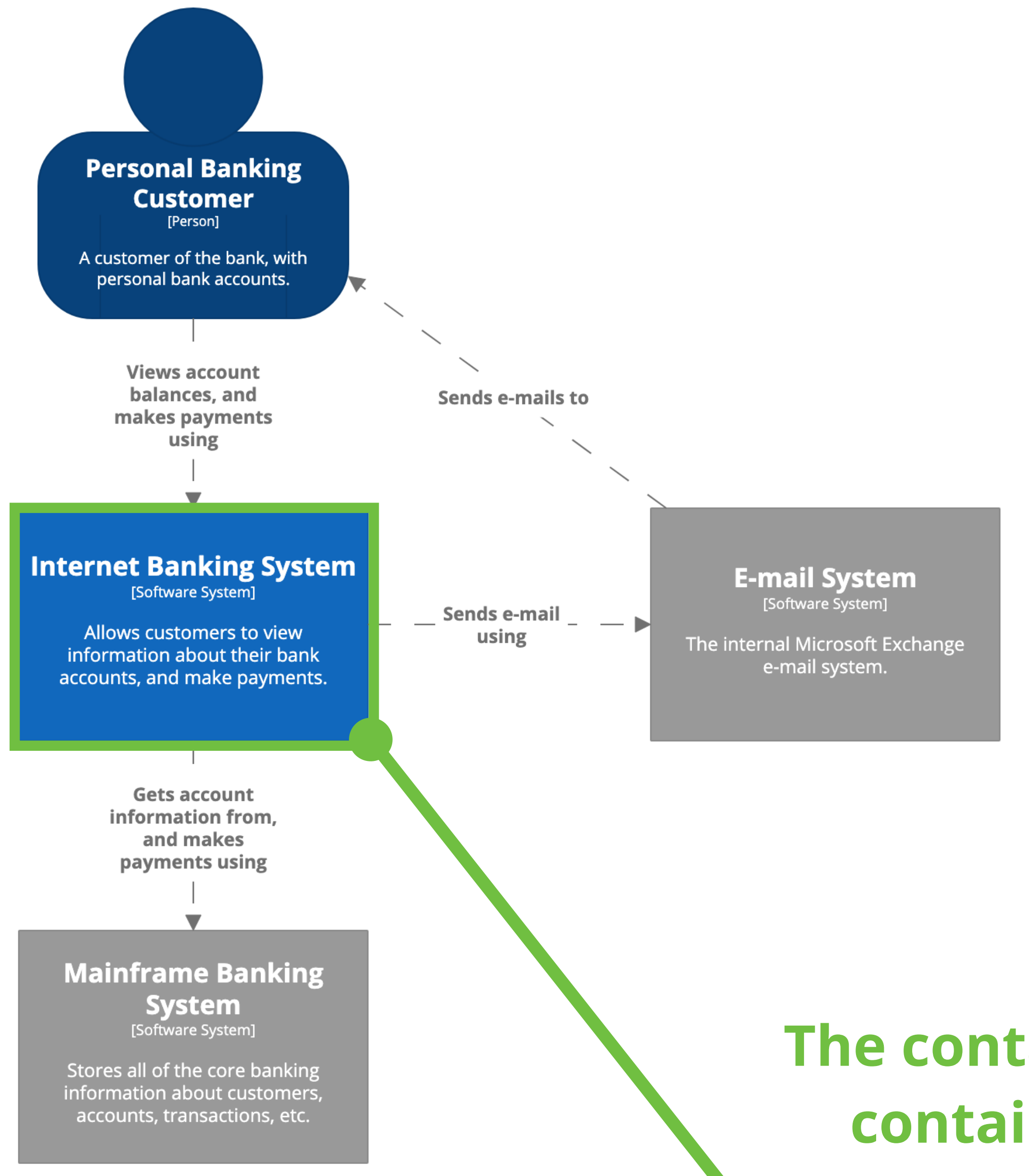
[System Context] Internet Banking System

The system context diagram for the Internet Banking System.

Monday, 27 February 2023 at 15:25 Greenwich Mean Time

Level 2

Container diagram



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Views account balances, and makes payments using

Sends e-mails to

Internet Banking System
[Software System]
Allows customers to view information about their bank accounts, and make payments.

Sends e-mail using

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

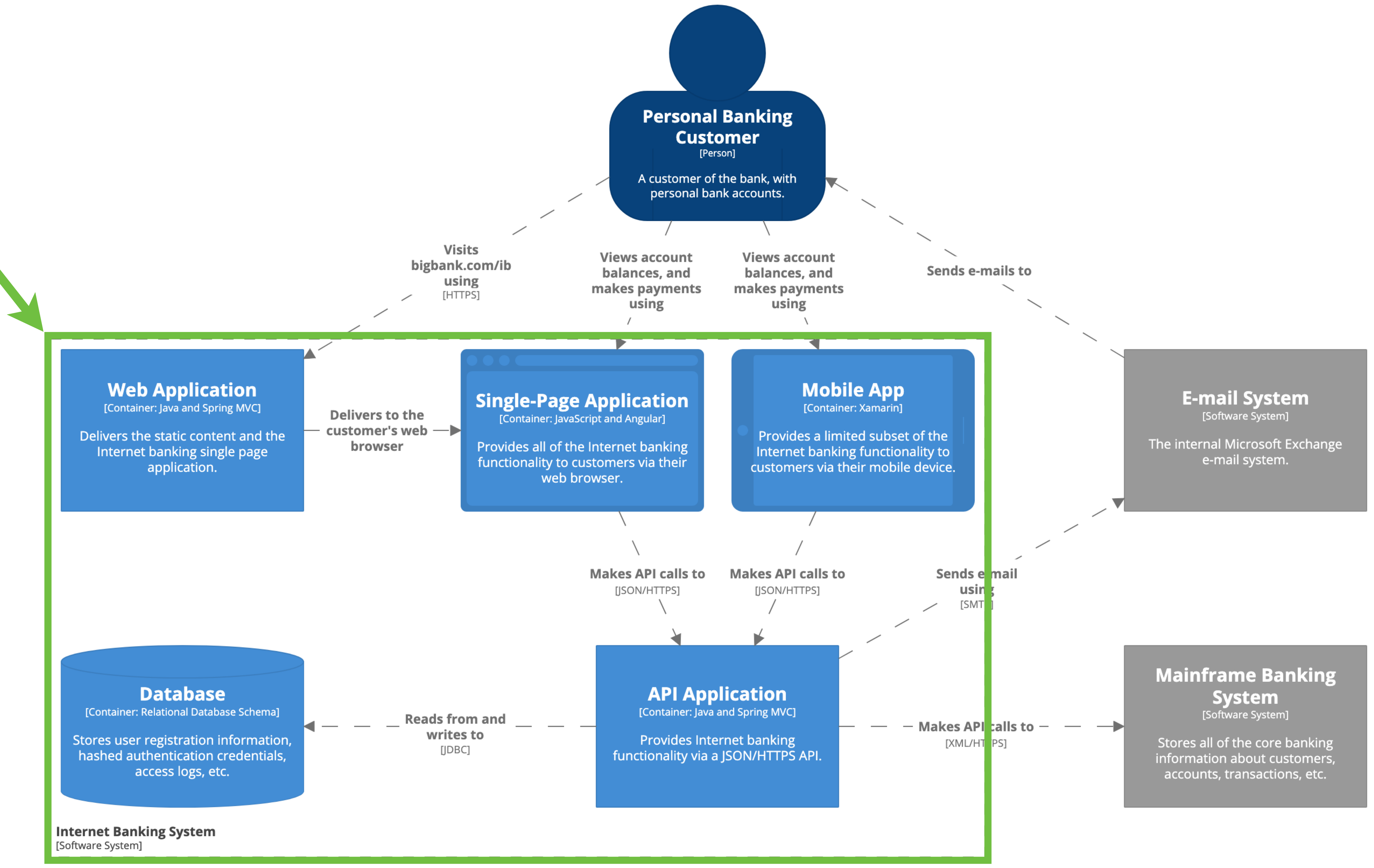
Gets account information from, and makes payments using

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)

The container diagram shows the containers that reside inside the software system boundary



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Visits bigbank.com/ib using
[HTTPS]

Views account balances, and makes payments using

Views account balances, and makes payments using

Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Delivers to the customer's web browser

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sends e-mail using
[SMTP]

Database
[Container: Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

Reads from and writes to
[JDBC]

API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

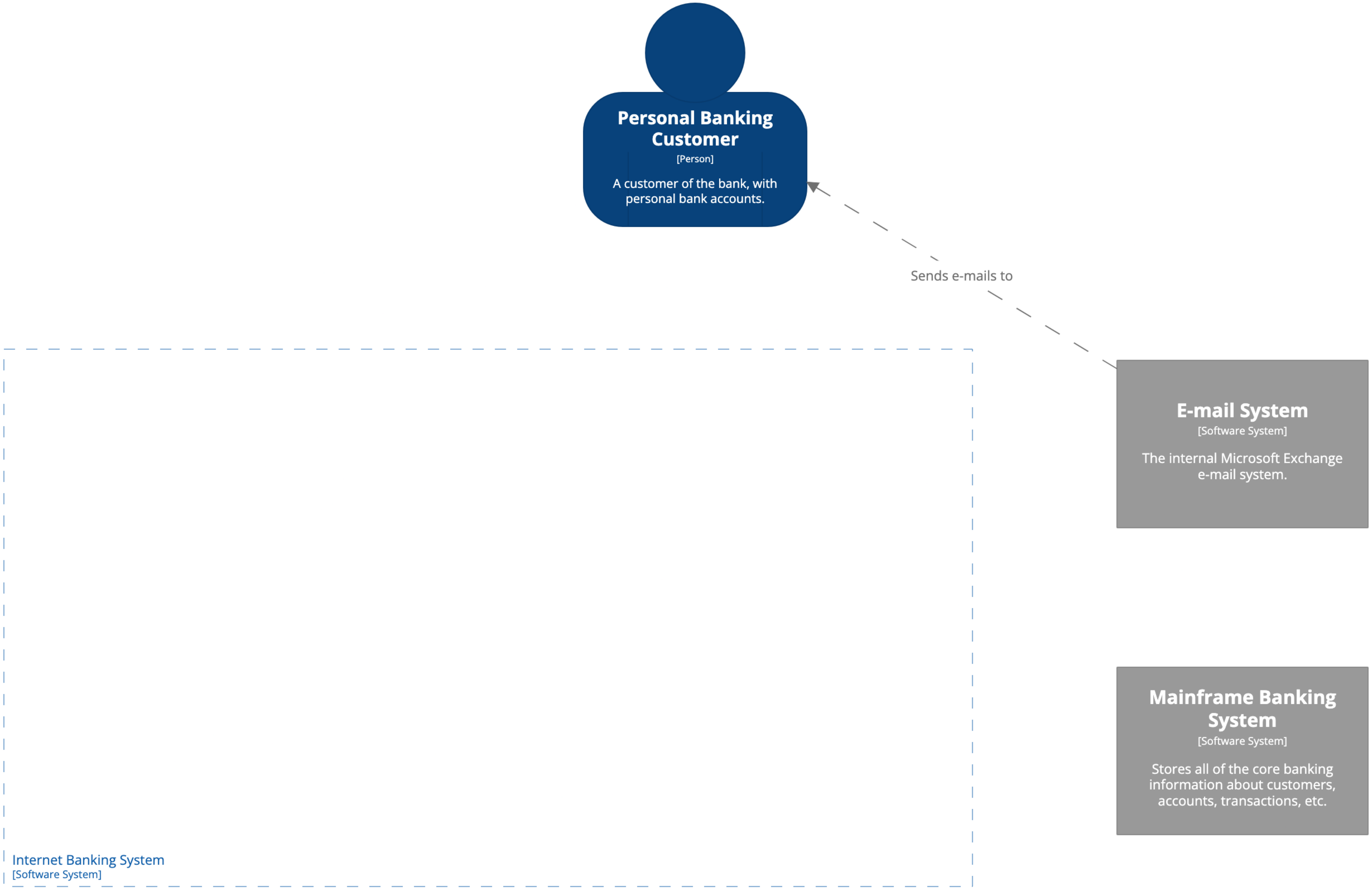
Makes API calls to
[XML/HTTPS]

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

Container diagram for Internet Banking System

The container diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)



Personal Banking Customer

[Person]

A customer of the bank, with personal bank accounts.

Sends e-mails to

E-mail System

[Software System]

The internal Microsoft Exchange e-mail system.

Mainframe Banking System

[Software System]

Stores all of the core banking information about customers, accounts, transactions, etc.

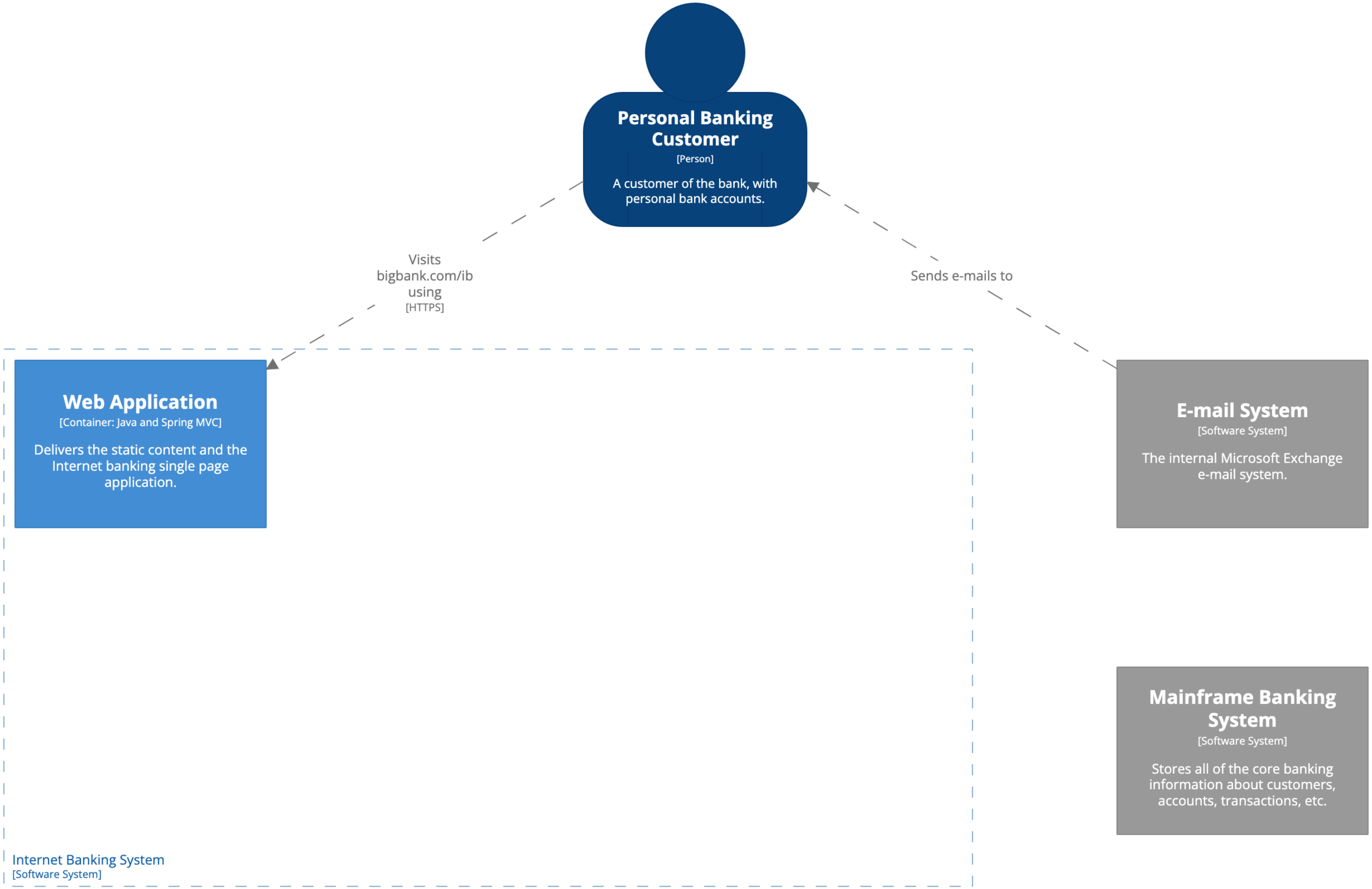
Internet Banking System

[Software System]

[Container] Internet Banking System

The container diagram for the Internet Banking System.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Visits bigbank.com/ib using
[HTTPS]

Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

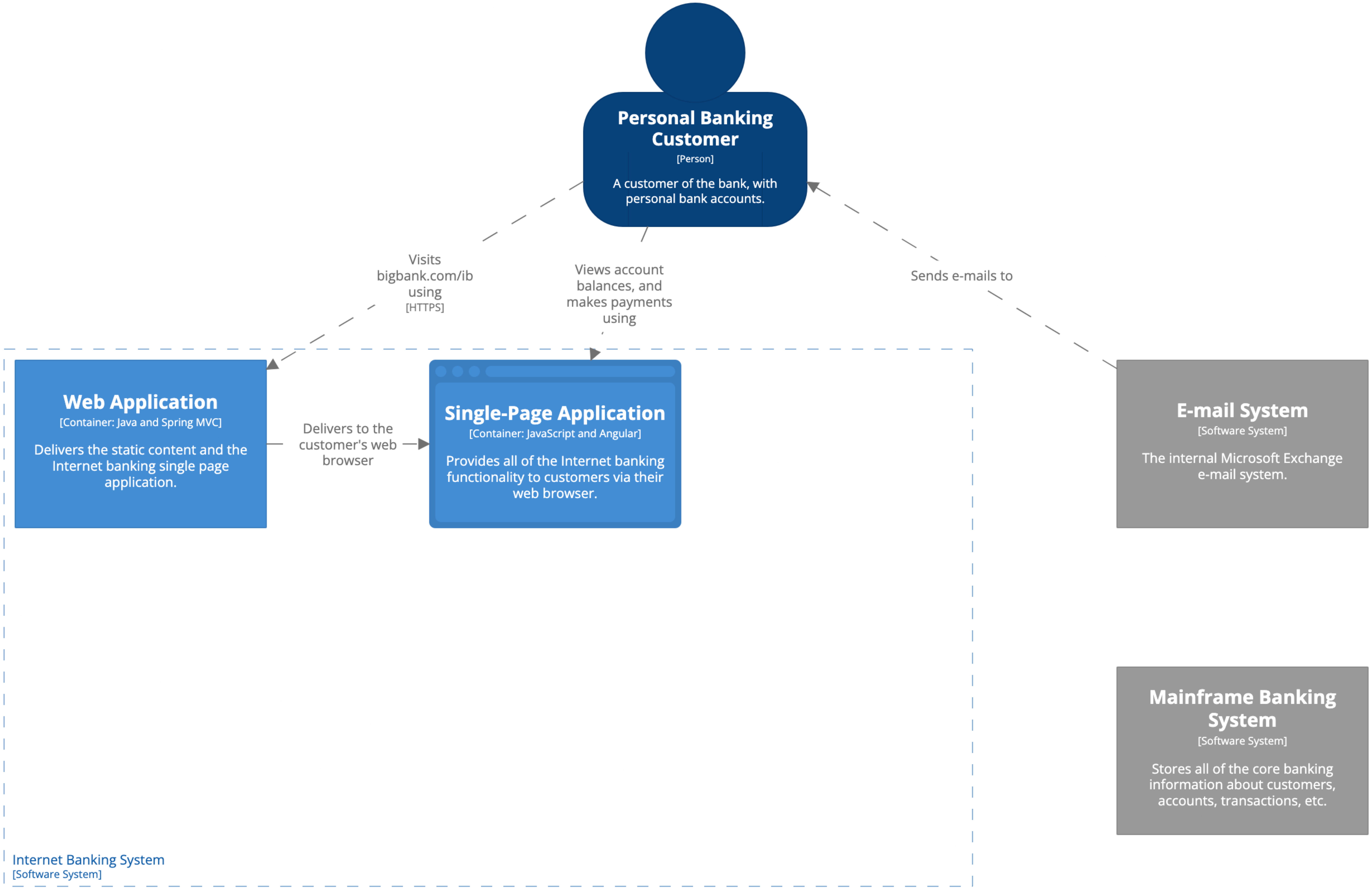
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

[Container] Internet Banking System

The container diagram for the Internet Banking System.
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Visits bigbank.com/ib using
[HTTPS]

Views account balances, and makes payments using

Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Delivers to the customer's web browser

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

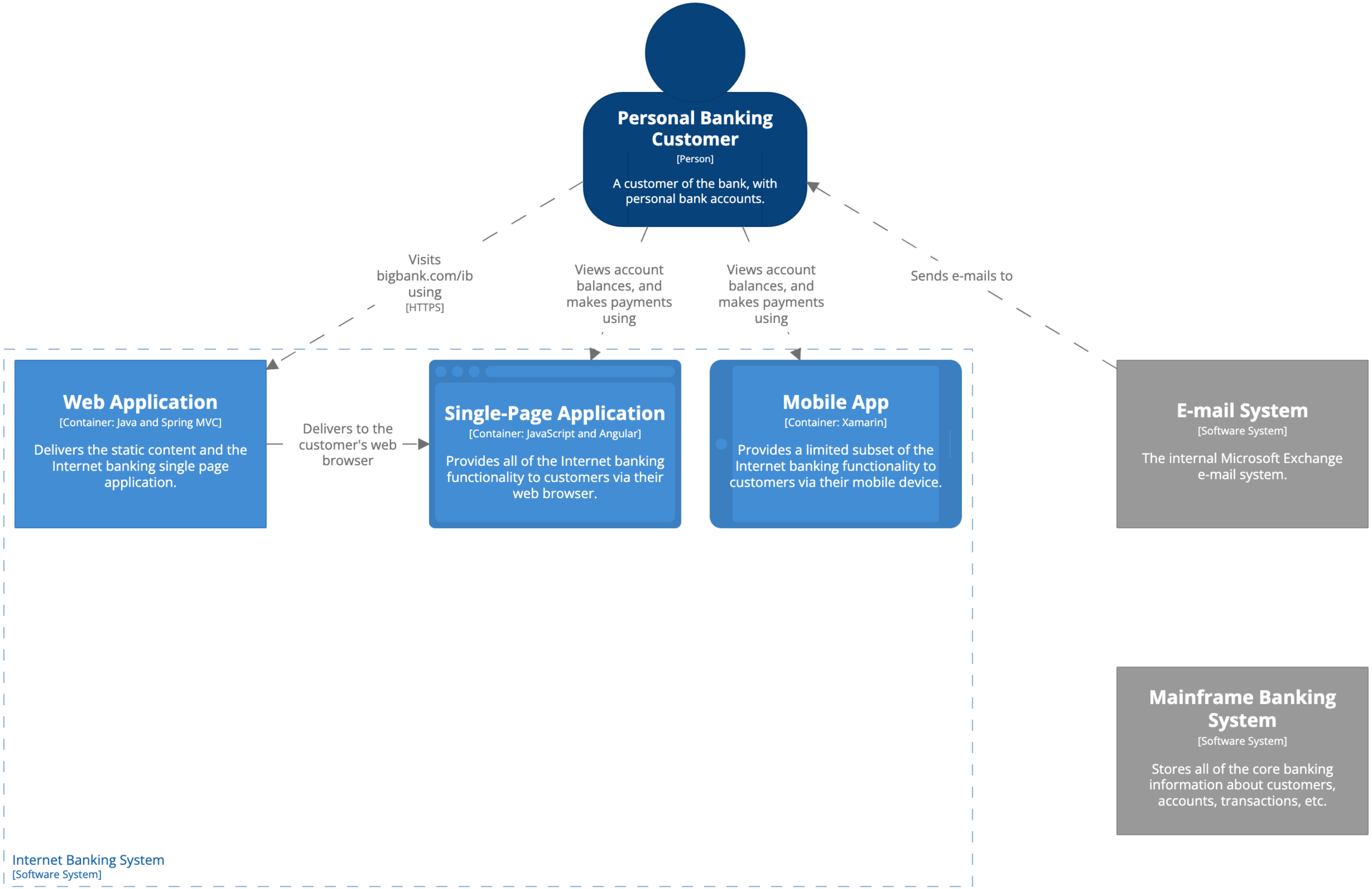
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

[Container] Internet Banking System

The container diagram for the Internet Banking System.
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Visits bigbank.com/ib using
[HTTPS]

Views account balances, and makes payments using

Views account balances, and makes payments using

Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Delivers to the customer's web browser

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

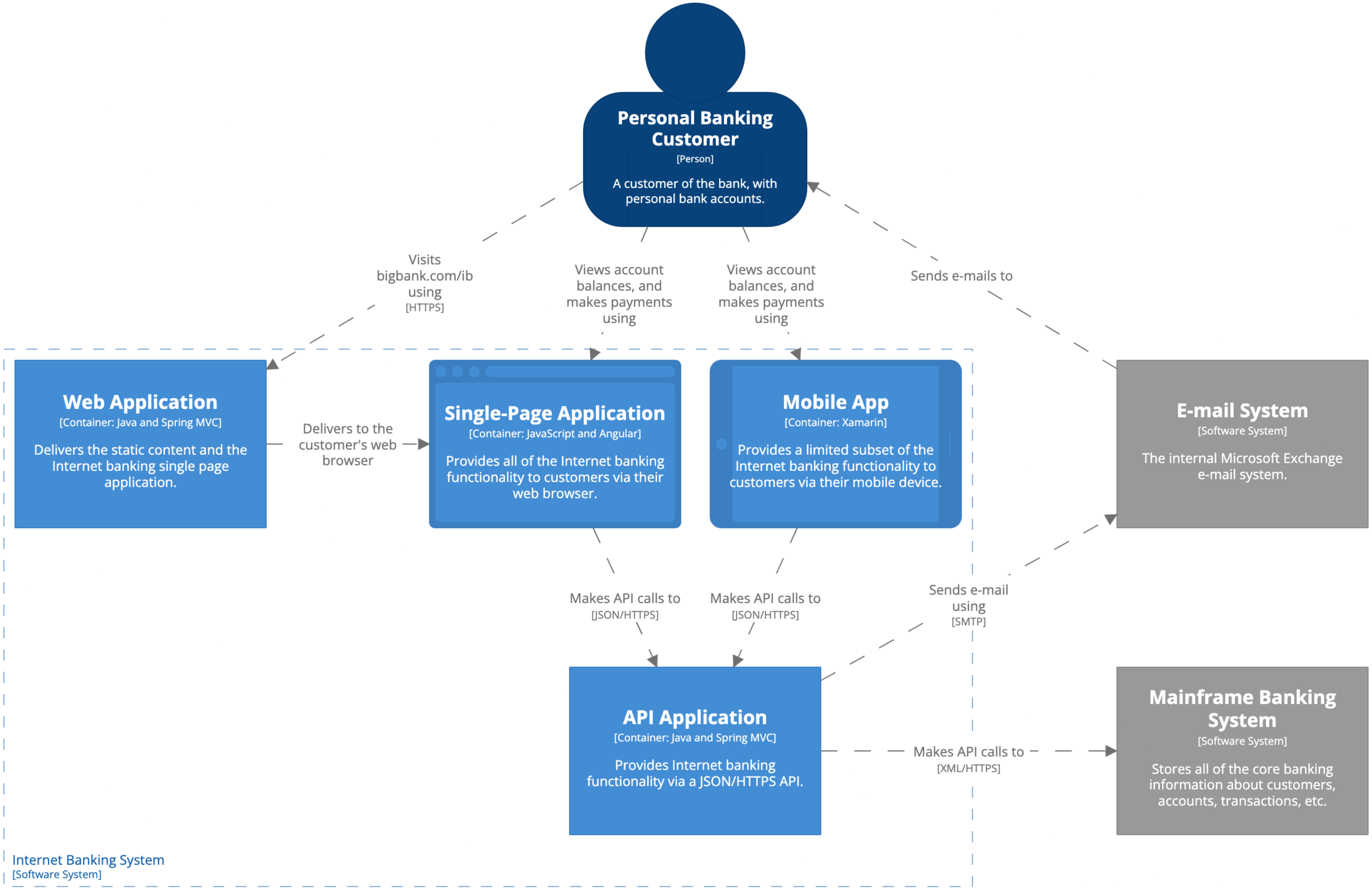
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

[Container] Internet Banking System

The container diagram for the Internet Banking System.
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Visits bigbank.com/ib using
[HTTPS]

Views account balances, and makes payments using

Views account balances, and makes payments using

Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Delivers to the customer's web browser

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sends e-mail using
[SMTP]

API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

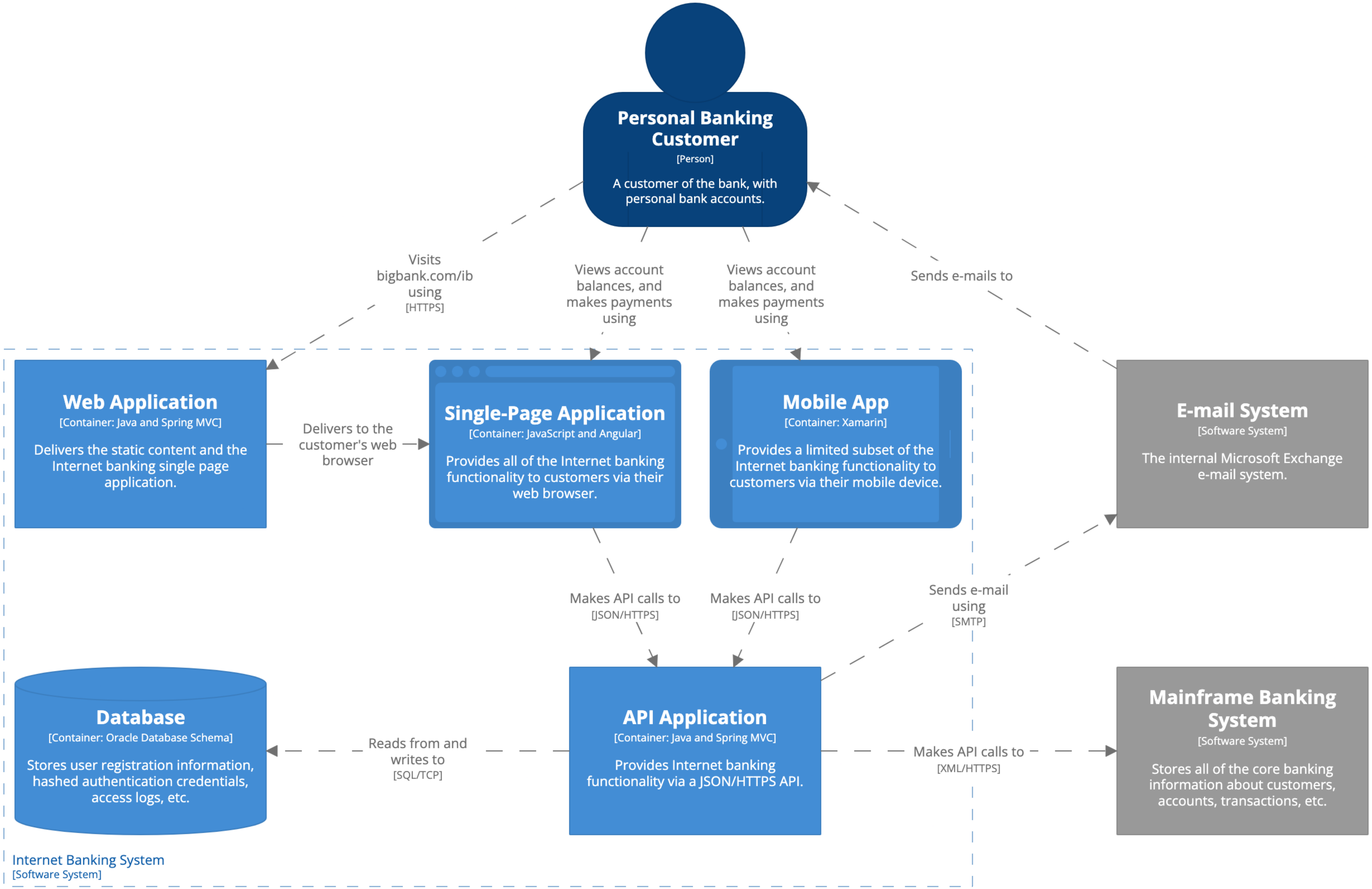
Makes API calls to
[XML/HTTPS]

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

[Container] Internet Banking System

The container diagram for the Internet Banking System.
Monday, 27 February 2023 at 15:36 Greenwich Mean Time


Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Visits bigbank.com/ib using
[HTTPS]
Views account balances, and makes payments using
Views account balances, and makes payments using
Sends e-mails to
Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.
Delivers to the customer's web browser
Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.
Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Sends e-mail using
[SMTP]
Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Reads from and writes to
[SQL/TCP]
API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
Internet Banking System
[Software System]

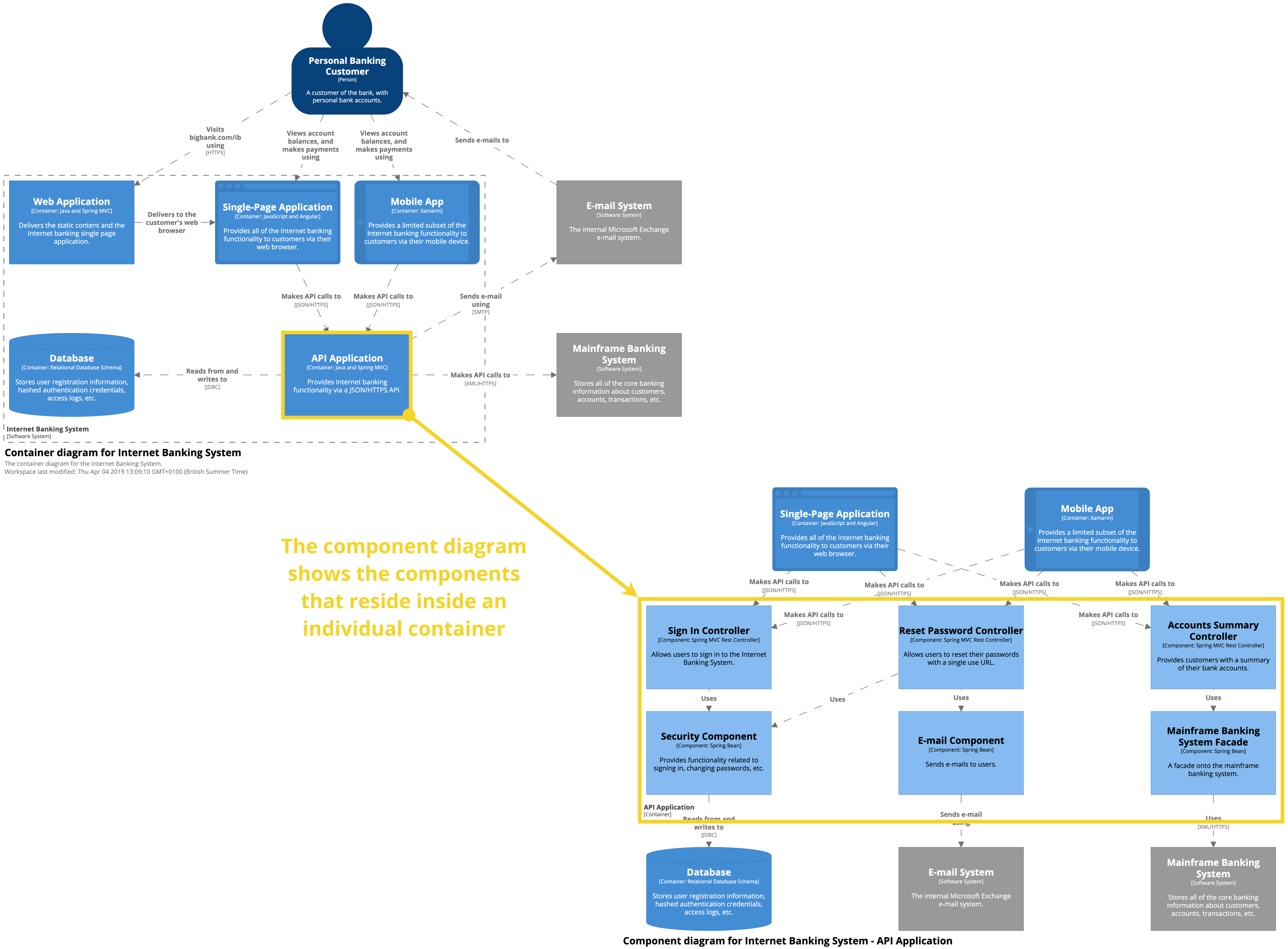
[Container] Internet Banking System

The container diagram for the Internet Banking System.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time

Level 3

Component diagram



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Visits bigbank.com/ib using
[HTTPS]

Views account balances, and makes payments using

Views account balances, and makes payments using

Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Delivers to the customer's web browser

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sends e-mail using
[SMTP]

Database
[Container: Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

Reads from and writes to
[JDBC]

API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

Makes API calls to
[XML/HTTPS]

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

Container diagram for Internet Banking System

The container diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)

The component diagram shows the components that reside inside an individual container

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.

Reset Password Controller
[Component: Spring MVC Rest Controller]
Allows users to reset their passwords with a single use URL.

Accounts Summary Controller
[Component: Spring MVC Rest Controller]
Provides customers with a summary of their bank accounts.

Uses

Uses

Uses

Uses

Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

E-mail Component
[Component: Spring Bean]
Sends e-mails to users.

Mainframe Banking System Facade
[Component: Spring Bean]
A facade onto the mainframe banking system.

API Application
[Container]

Reads from and writes to
[JDBC]

Sends e-mail using

Uses
[XML/HTTPS]

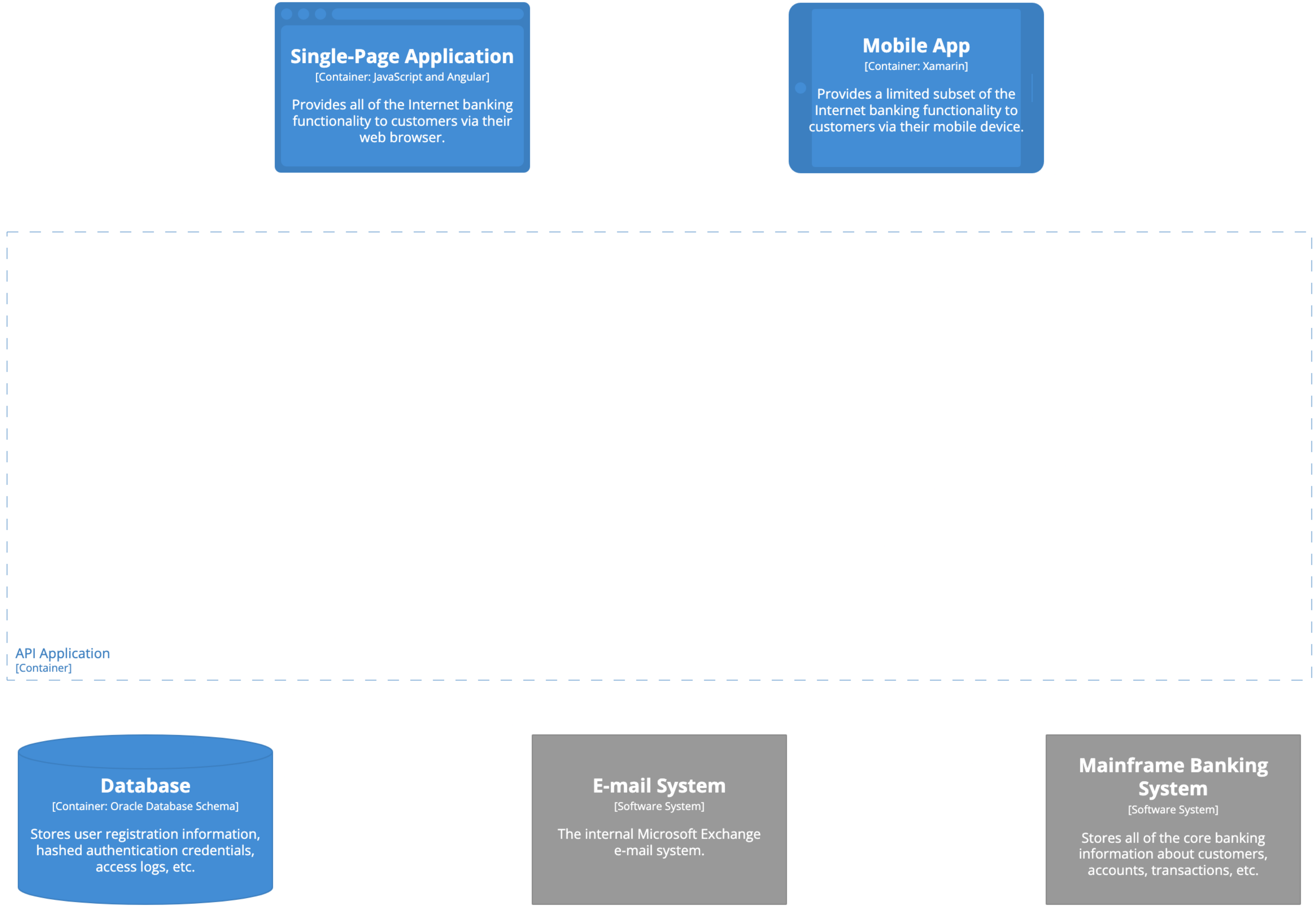
Database
[Container: Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Component diagram for Internet Banking System - API Application

The component diagram for the API Application.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)



Single-Page Application

[Container: JavaScript and Angular]

Provides all of the Internet banking functionality to customers via their web browser.

Mobile App

[Container: Xamarin]

Provides a limited subset of the Internet banking functionality to customers via their mobile device.

API Application

[Container]

Database

[Container: Oracle Database Schema]

Stores user registration information, hashed authentication credentials, access logs, etc.

E-mail System

[Software System]

The internal Microsoft Exchange e-mail system.

Mainframe Banking System

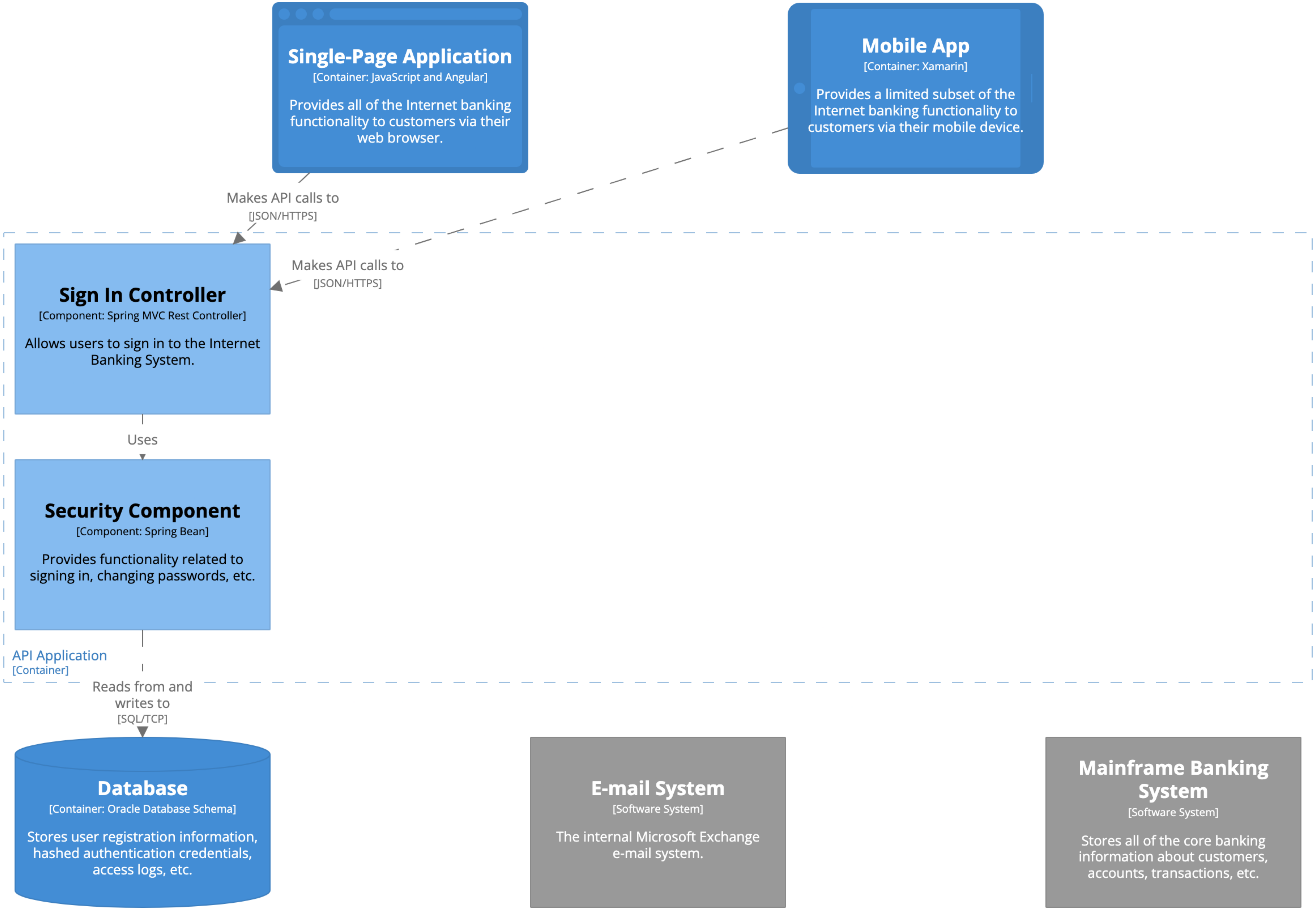
[Software System]

Stores all of the core banking information about customers, accounts, transactions, etc.

[Component] Internet Banking System - API Application

The component diagram for the API Application.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.

Uses

Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

API Application
[Container]

Reads from and writes to
[SQL/TCP]

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

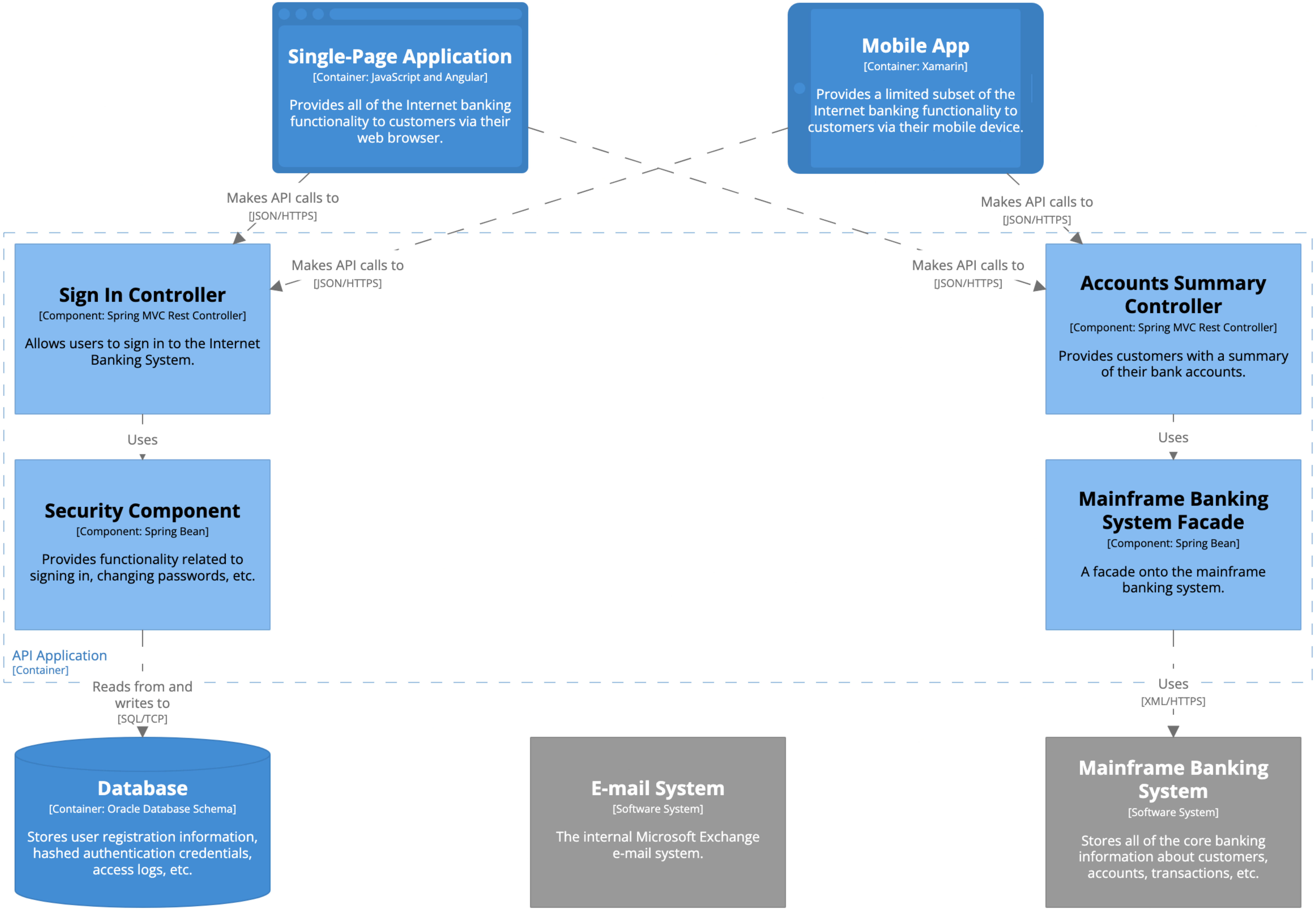
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

[Component] Internet Banking System - API Application

The component diagram for the API Application.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.

Uses

Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

Accounts Summary Controller
[Component: Spring MVC Rest Controller]
Provides customers with a summary of their bank accounts.

Uses

Mainframe Banking System Facade
[Component: Spring Bean]
A facade onto the mainframe banking system.

API Application
[Container]

Reads from and writes to
[SQL/TCP]

Uses
[XML/HTTPS]

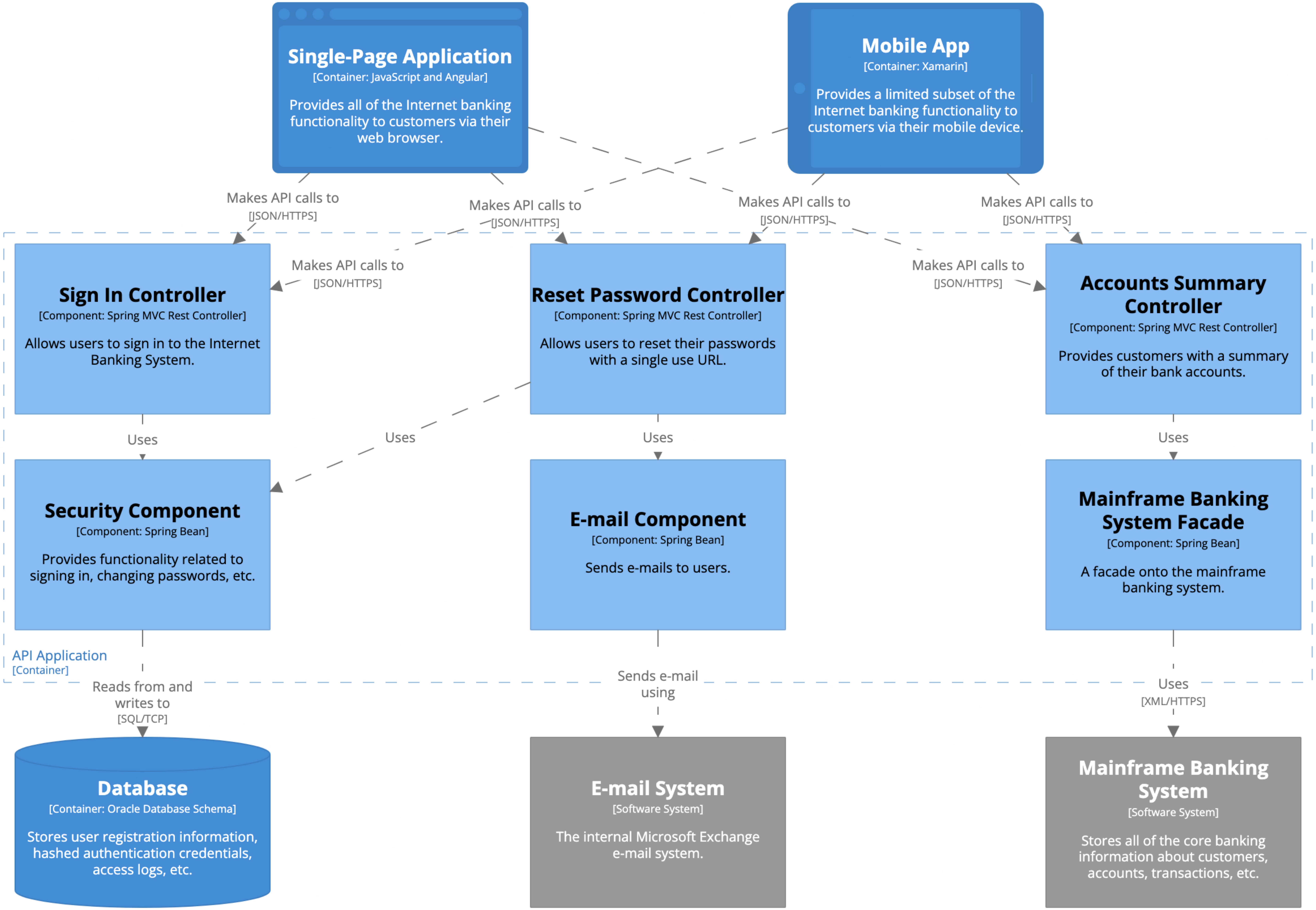
Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

[Component] Internet Banking System - API Application

The component diagram for the API Application.
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.

Reset Password Controller
[Component: Spring MVC Rest Controller]
Allows users to reset their passwords with a single use URL.

Accounts Summary Controller
[Component: Spring MVC Rest Controller]
Provides customers with a summary of their bank accounts.

Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

E-mail Component
[Component: Spring Bean]
Sends e-mails to users.

Mainframe Banking System Facade
[Component: Spring Bean]
A facade onto the mainframe banking system.

API Application
[Container]

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Makes API calls to [JSON/HTTPS]
Makes API calls to [JSON/HTTPS]
Makes API calls to [JSON/HTTPS]
Makes API calls to [JSON/HTTPS]
Makes API calls to [JSON/HTTPS]
Makes API calls to [JSON/HTTPS]
Uses
Uses
Uses
Uses
Reads from and writes to [SQL/TCP]
Sends e-mail using
Uses [XML/HTTPS]

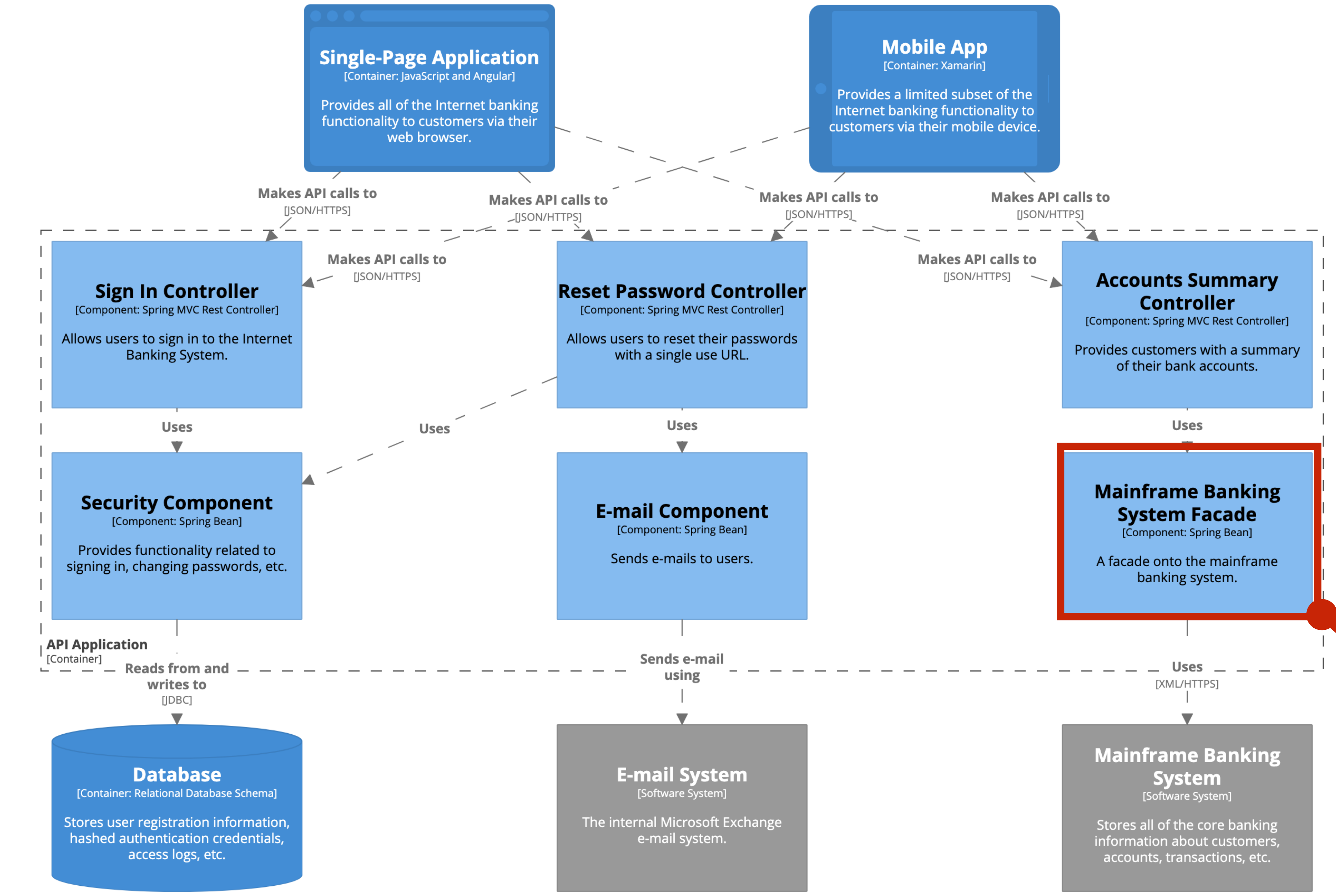
[Component] Internet Banking System - API Application

The component diagram for the API Application.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time

Level 4

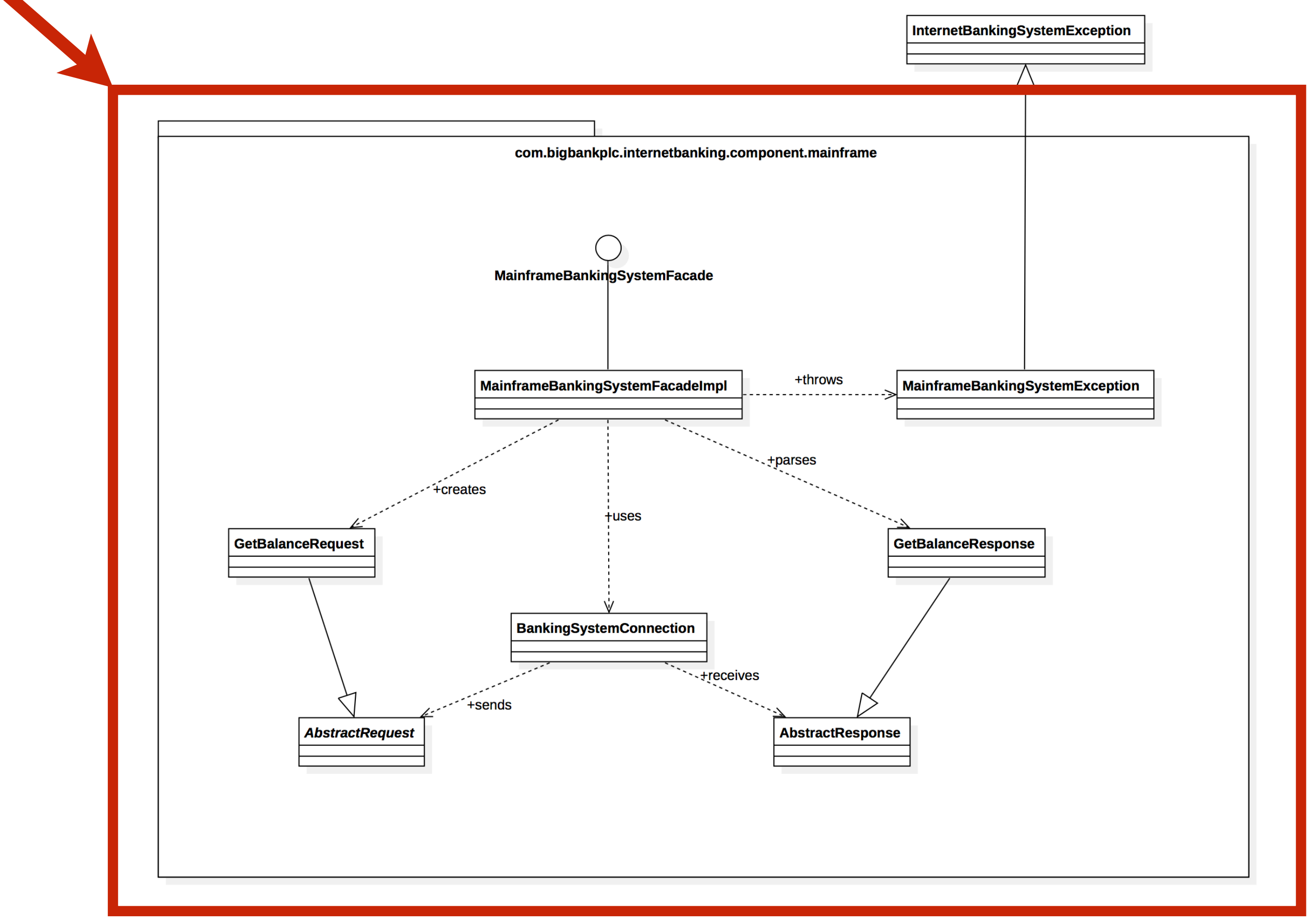
Code diagram

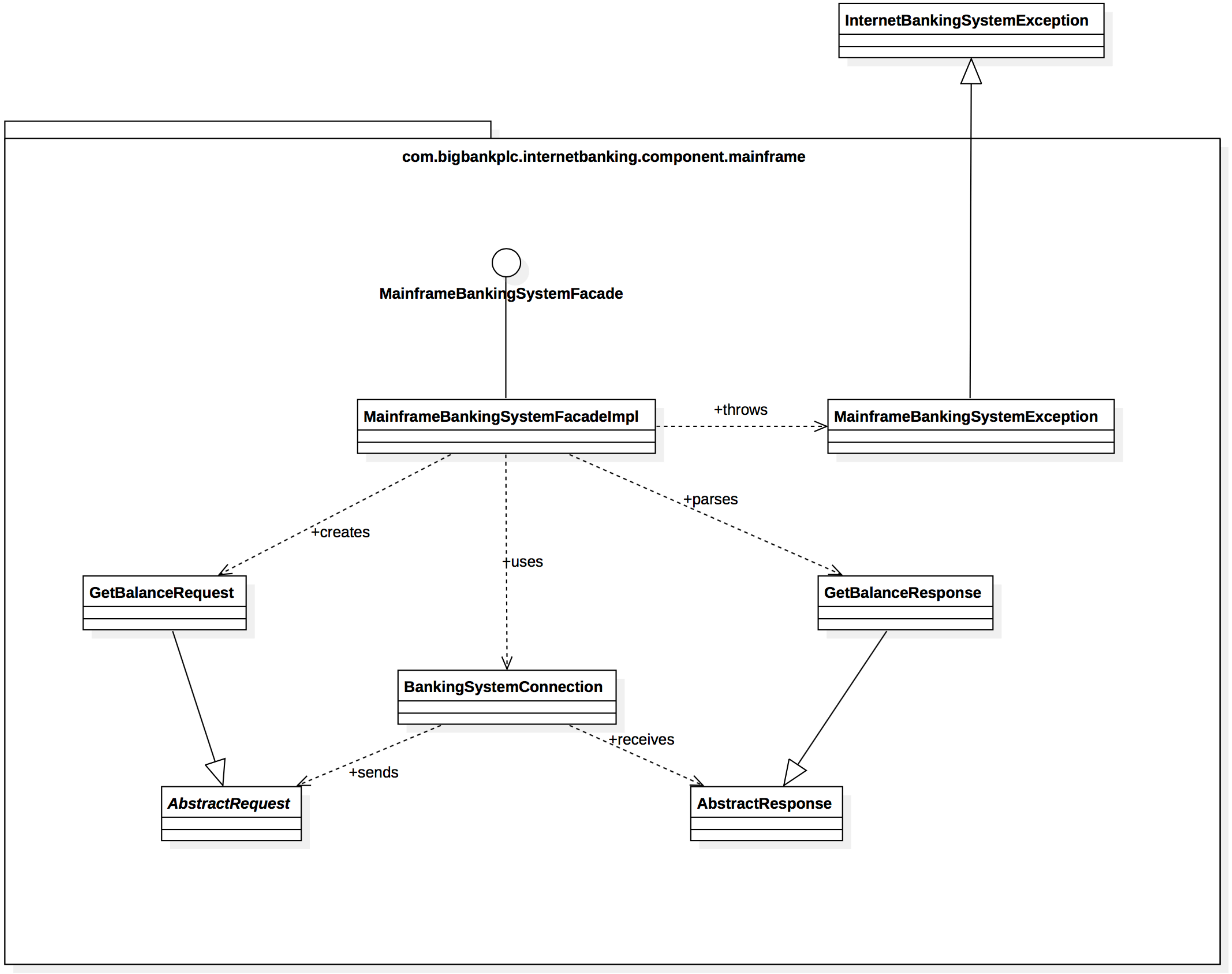

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.
Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.
Reset Password Controller
[Component: Spring MVC Rest Controller]
Allows users to reset their passwords with a single use URL.
Accounts Summary Controller
[Component: Spring MVC Rest Controller]
Provides customers with a summary of their bank accounts.
Uses
Uses
Uses
Uses
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
E-mail Component
[Component: Spring Bean]
Sends e-mails to users.
Mainframe Banking System Facade
[Component: Spring Bean]
A facade onto the mainframe banking system.
API Application
[Container]
Reads from and writes to
[JDBC]
Sends e-mail using
Uses
[XML/HTTPS]
Database
[Container: Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Component diagram for Internet Banking System - API Application

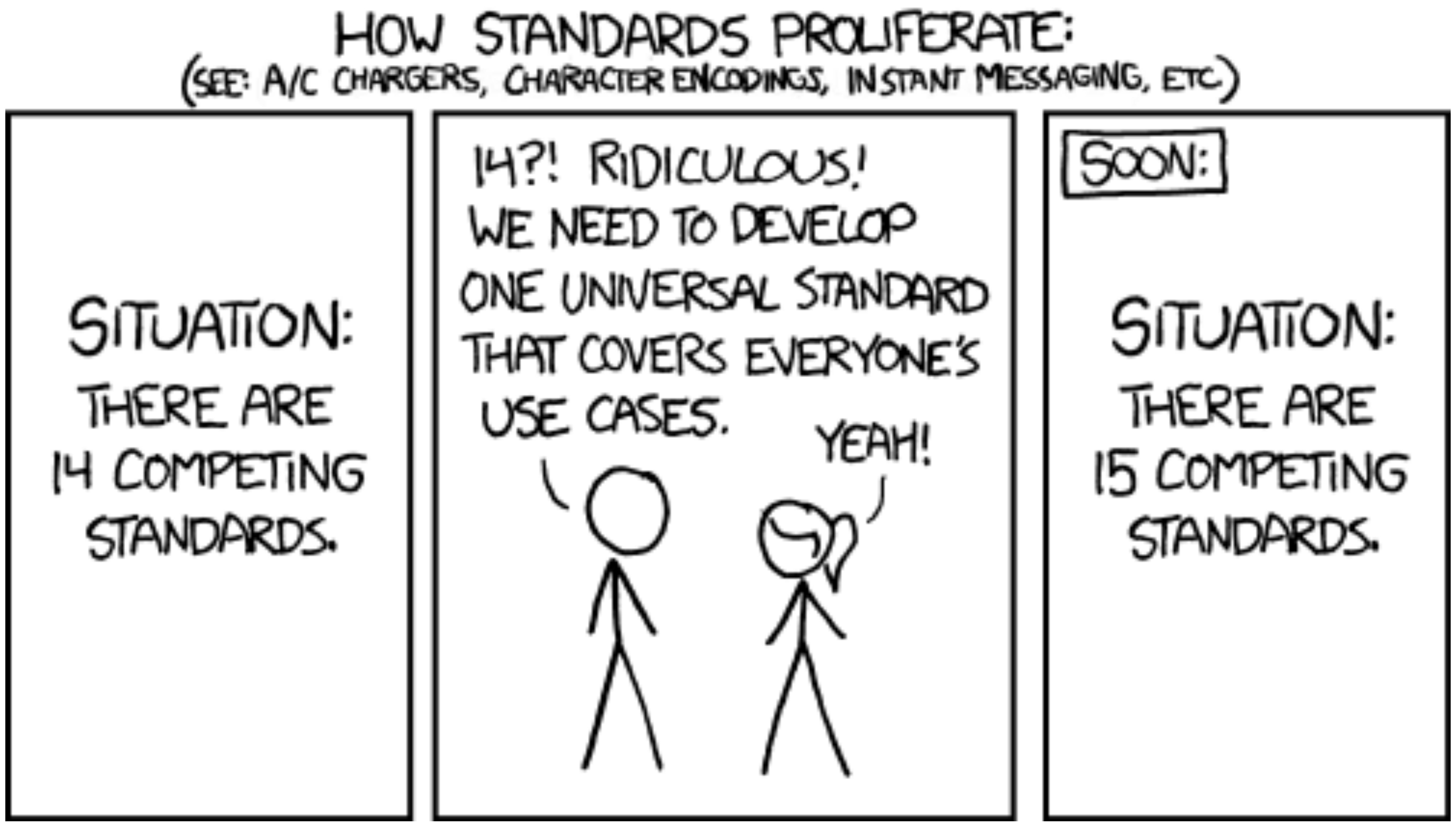
The component diagram for the API Application.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)

The code level diagram shows the code elements that make up a component

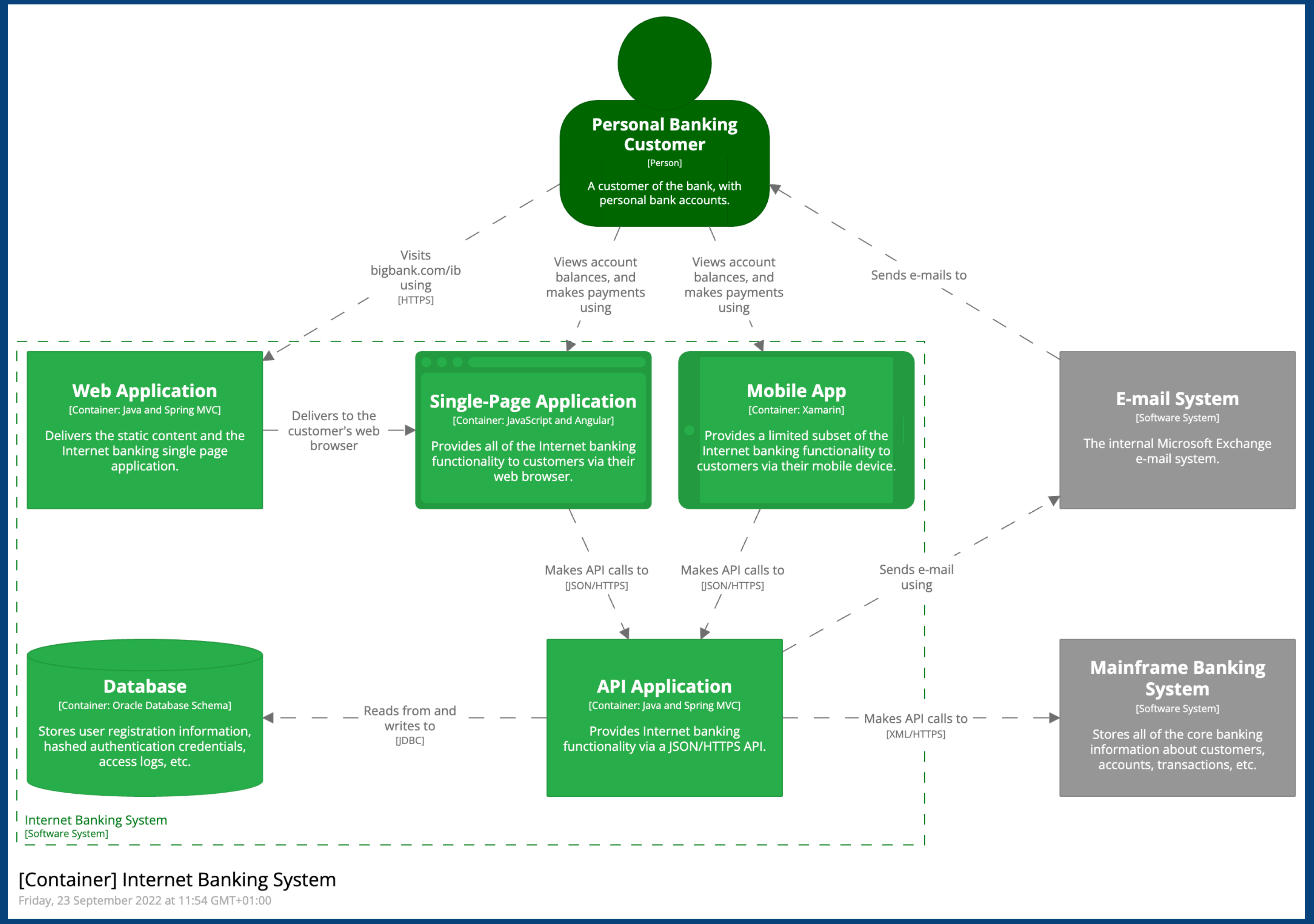

InternetBankingSystemException
com.bigbankplc.internetbanking.component.mainframe
MainframeBankingSystemFacade
MainframeBankingSystemFacadeImpl
+throws
MainframeBankingSystemException
+creates
+uses
+parses
GetBalanceRequest
GetBalanceResponse
BankingSystemConnection
+sends
+receives
AbstractRequest
AbstractResponse


InternetBankingSystemException
com.bigbankplc.internetbanking.component.mainframe
MainframeBankingSystemFacade
MainframeBankingSystemFacadeImpl
+throws
MainframeBankingSystemException
+creates
+uses
+parses
GetBalanceRequest
GetBalanceResponse
BankingSystemConnection
+sends
+receives
AbstractRequest
AbstractResponse

Notation

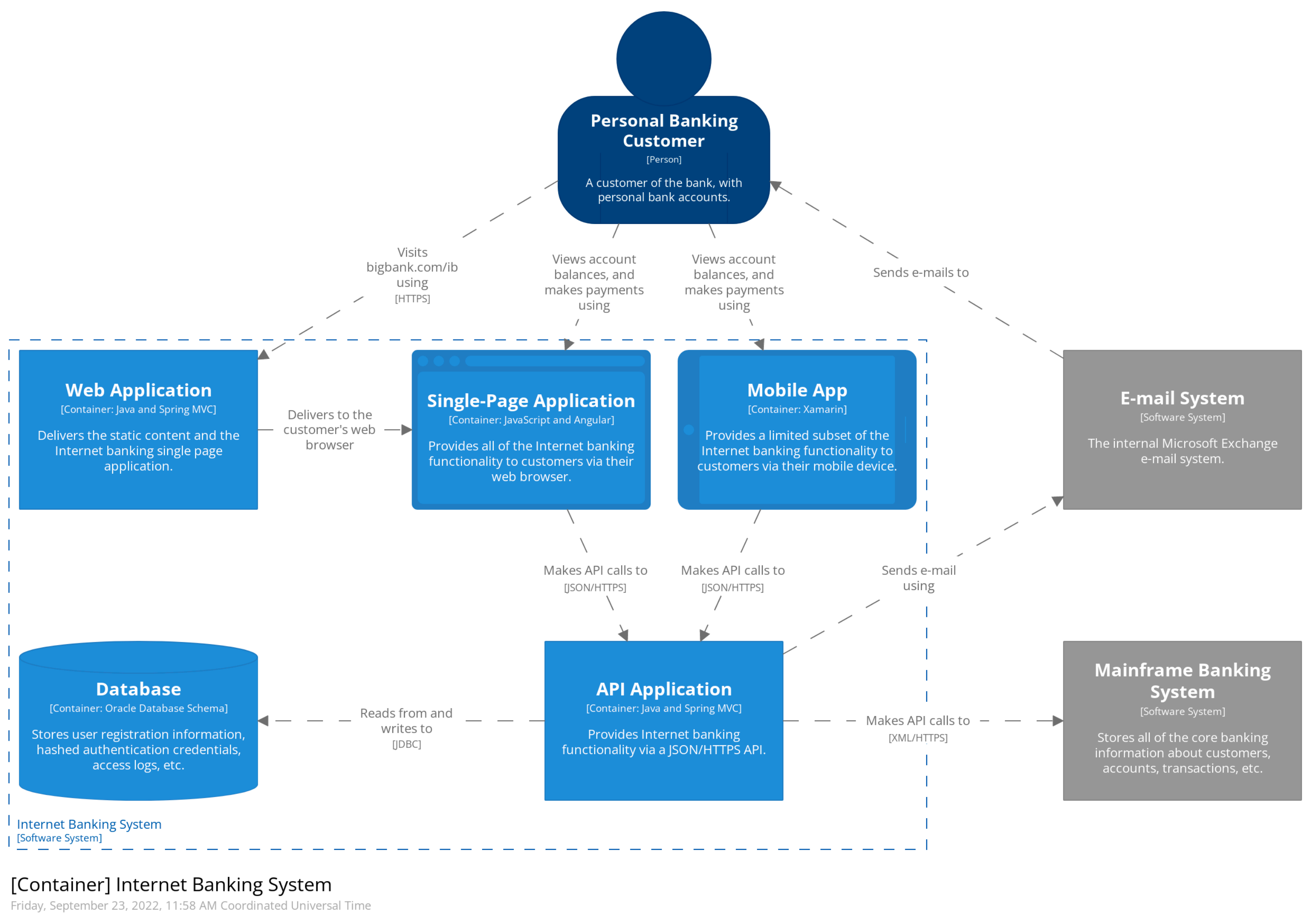

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION:
THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION:
THERE ARE 15 COMPETING STANDARDS.

The C4 model is **notation independent**


Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Visits bigbank.com/ib using
[HTTPS]
Views account balances, and makes payments using
Views account balances, and makes payments using
Sends e-mails to
Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.
Delivers to the customer's web browser
Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.
Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Sends e-mail using
Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Reads from and writes to
[JDBC]
API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
Internet Banking System
[Software System]

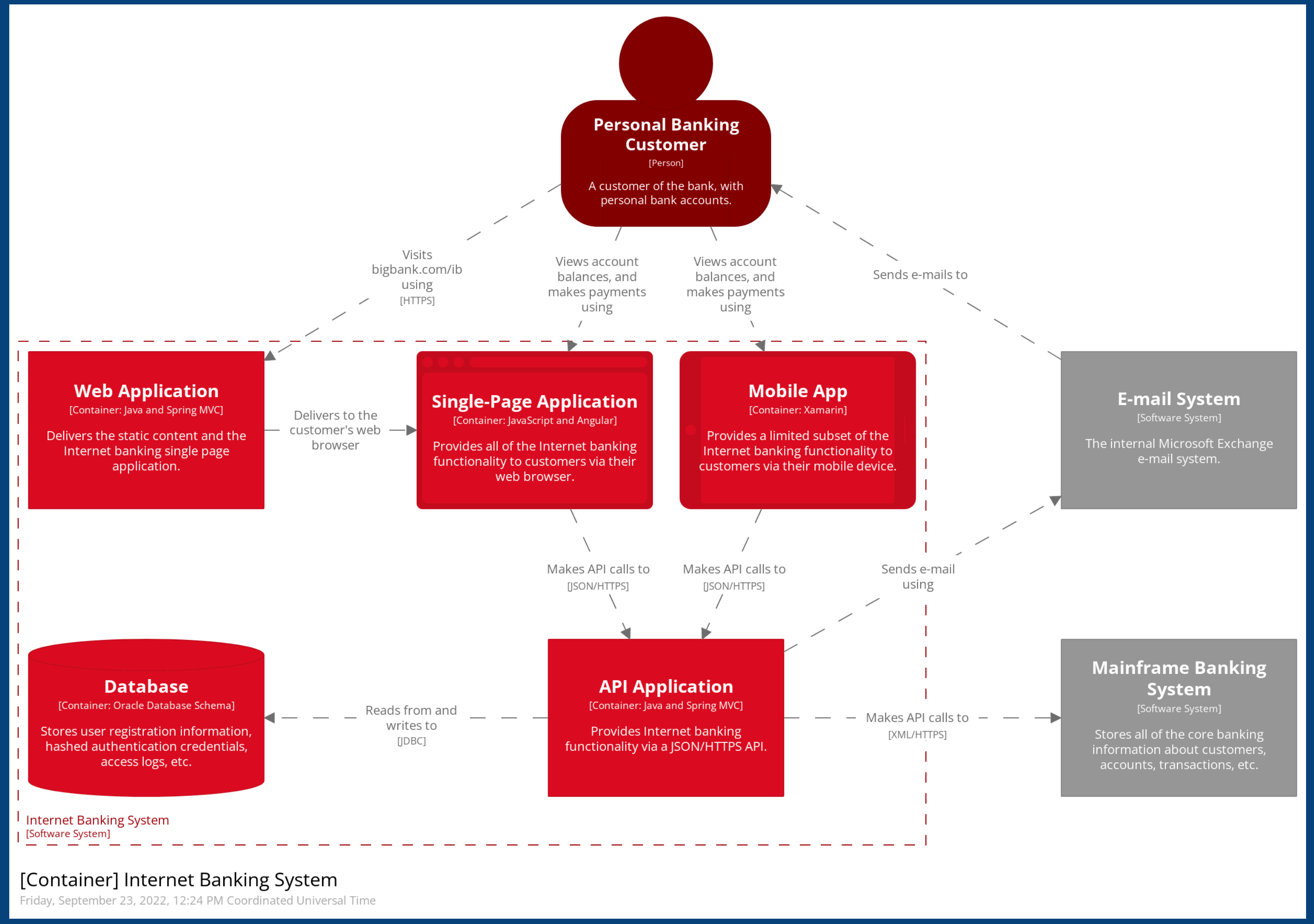
[Container] Internet Banking System

Friday, 23 September 2022 at 11:54 GMT+01:00


Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Visits bigbank.com/ib using
[HTTPS]
Views account balances, and makes payments using
Views account balances, and makes payments using
Sends e-mails to
Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.
Delivers to the customer's web browser
Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.
Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Sends e-mail using
Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Reads from and writes to
[JDBC]
API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
Internet Banking System
[Software System]

[Container] Internet Banking System

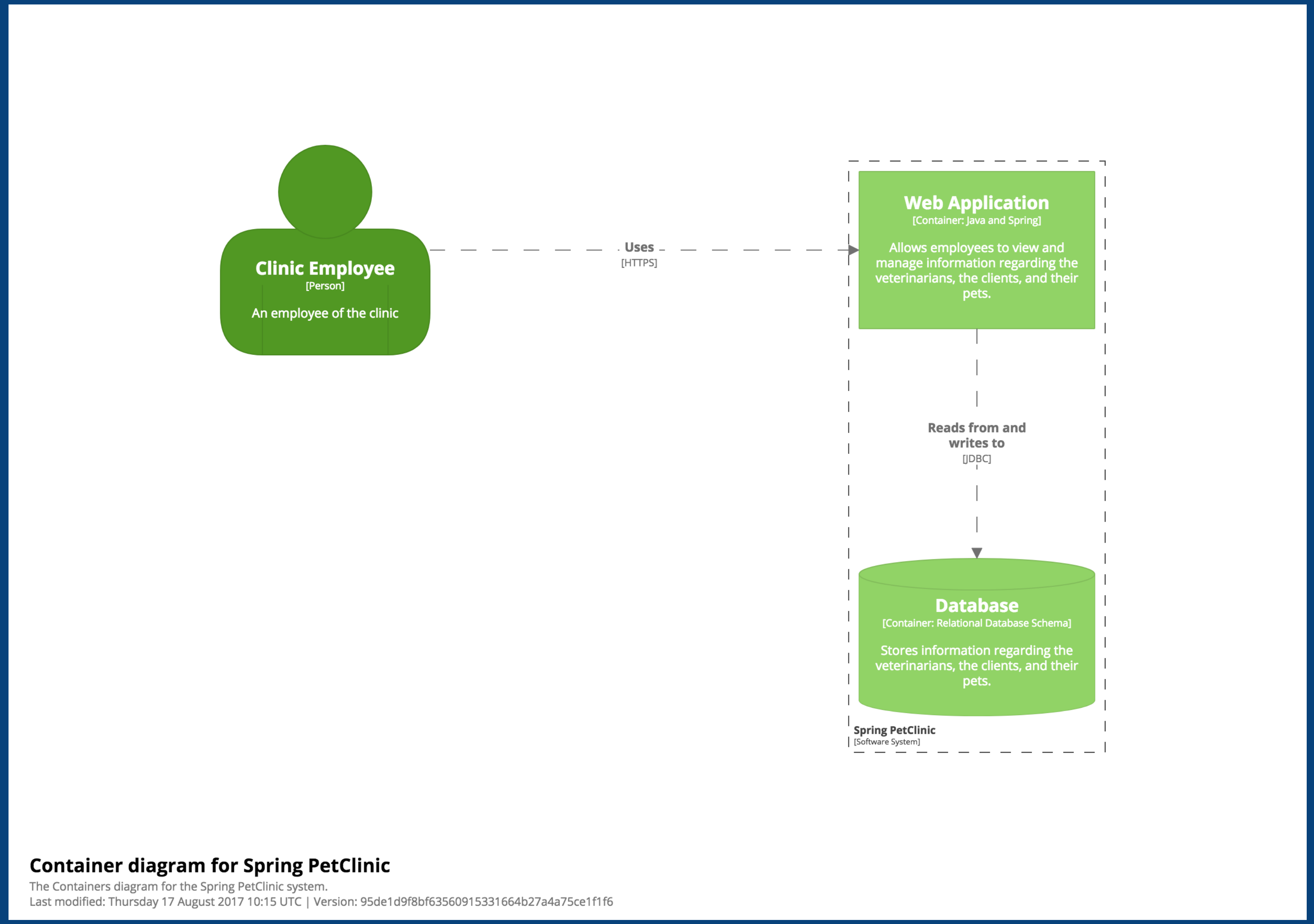
Friday, September 23, 2022, 11:58 AM Coordinated Universal Time


Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Visits bigbank.com/ib using
[HTTPS]
Views account balances, and makes payments using
Views account balances, and makes payments using
Sends e-mails to
Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.
Delivers to the customer's web browser
Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.
Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Sends e-mail using
Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Reads from and writes to
[JDBC]
API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
Internet Banking System
[Software System]

[Container] Internet Banking System

Friday, September 23, 2022, 12:24 PM Coordinated Universal Time

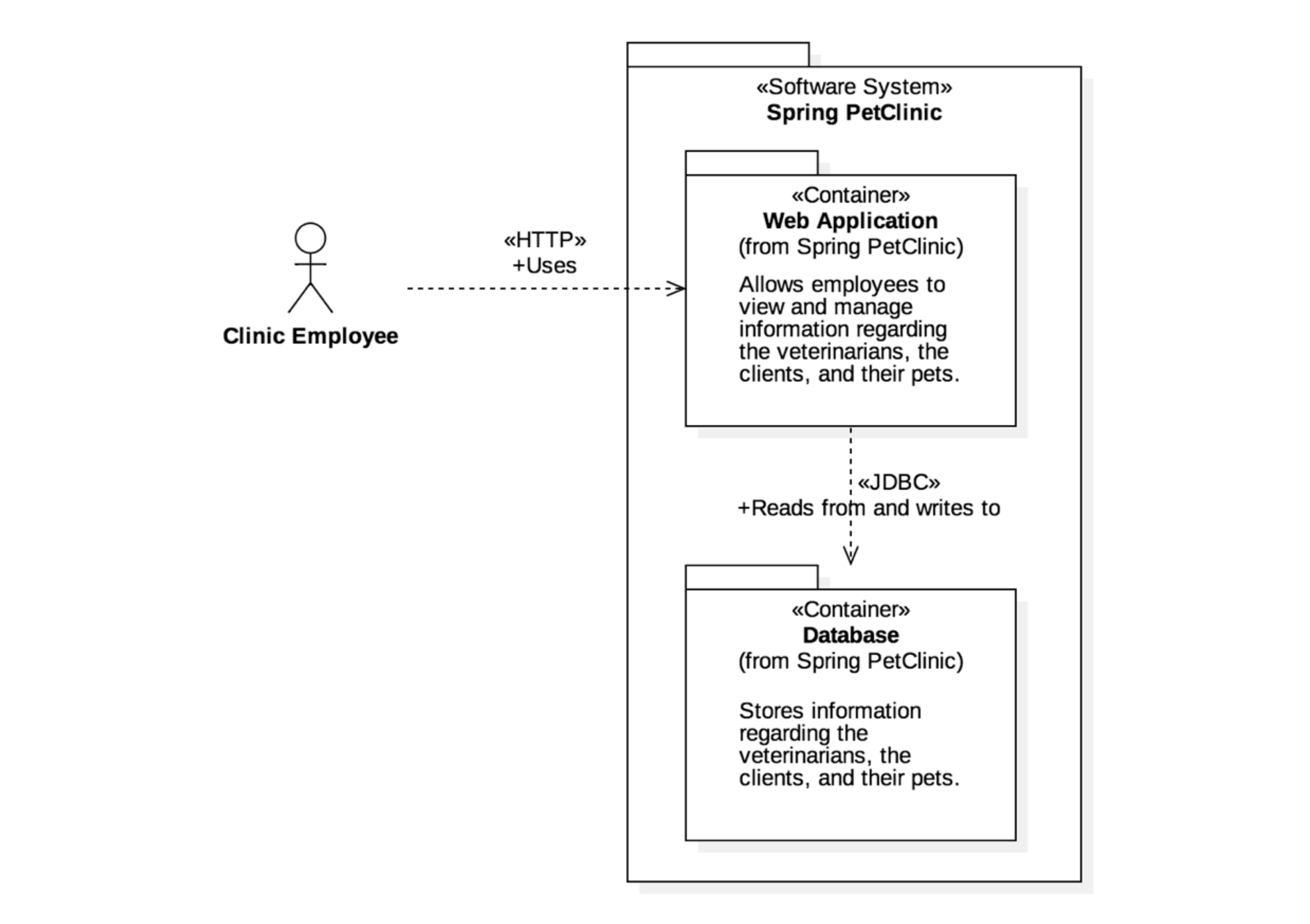
The C4 model is **notation independent**

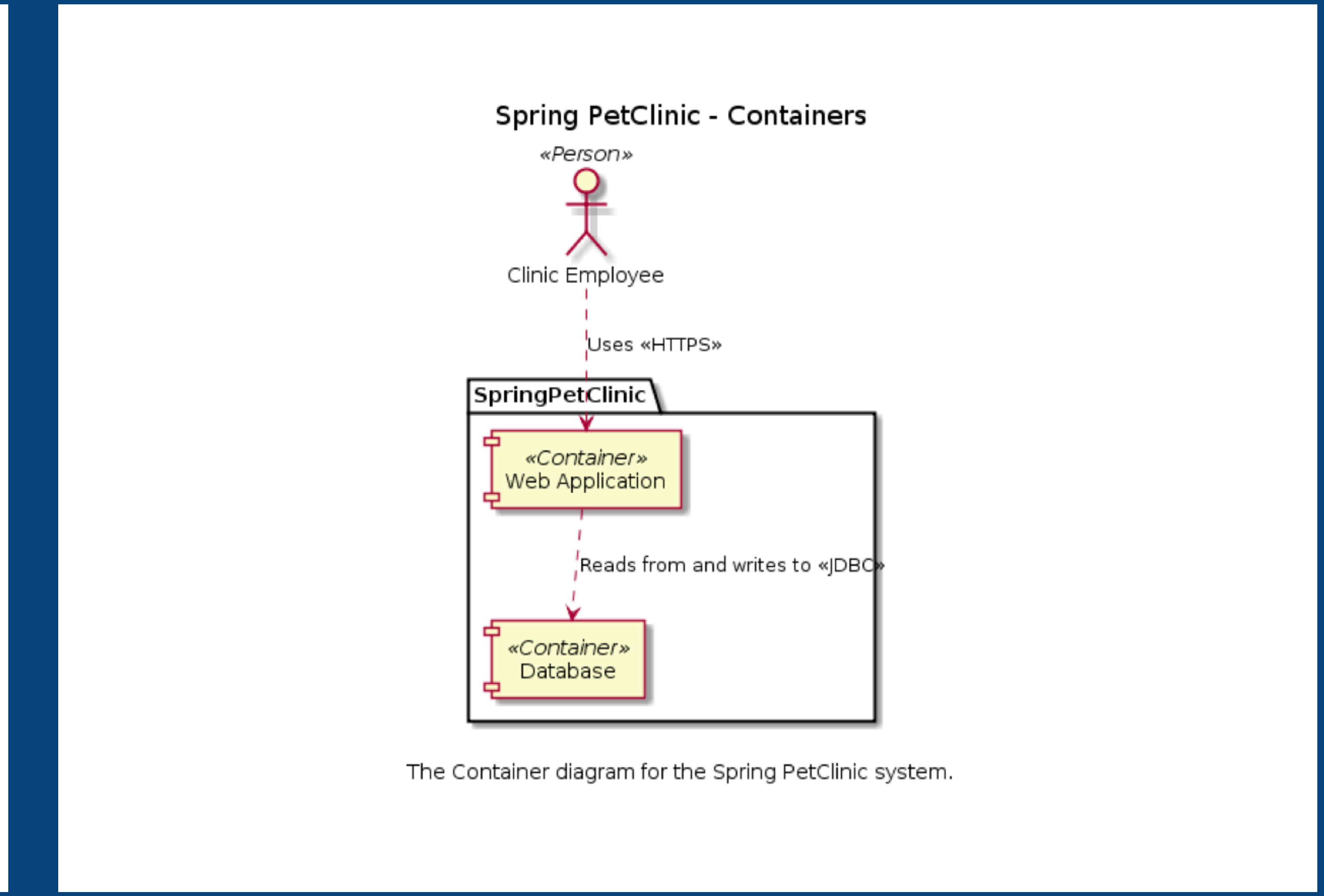

Clinic Employee
[Person]
An employee of the clinic
Uses
[HTTPS]
Web Application
[Container: Java and Spring]
Allows employees to view and manage information regarding the veterinarians, the clients, and their pets.
Reads from and writes to
[JDBC]
Database
[Container: Relational Database Schema]
Stores information regarding the veterinarians, the clients, and their pets.
Spring PetClinic
[Software System]

Container diagram for Spring PetClinic

The Containers diagram for the Spring PetClinic system.

Last modified: Thursday 17 August 2017 10:15 UTC | Version: 95de1d9f8bf63560915331664b27a4a75ce1f1f6


«Software System»
Spring PetClinic
Clinic Employee
«HTTP»
+Uses
«Container»
Web Application
(from Spring PetClinic)
Allows employees to view and manage information regarding the veterinarians, the clients, and their pets.
«JDBC»
+Reads from and writes to
«Container»
Database
(from Spring PetClinic)
Stores information regarding the veterinarians, the clients, and their pets.



The Container diagram for the Spring PetClinic system.

Titles

Short and meaningful, include the **diagram type**, numbered if diagram order is important; for example:

System Context diagram for Financial Risk System

[**System Context**] Financial Risk System

Visual consistency

Try to be consistent with notation
and element positioning across diagrams

Acronyms

Be wary of using acronyms, especially those related to the business/domain that you work in

Boxes

Start with simple boxes containing the element name, type, technology (if appropriate) and a description/responsibilities

Personal Banking Customer

[Person]

A customer of the bank, with personal bank accounts.

Internet Banking System

[Software System]

Allows customers to view information about their bank accounts, and make payments.

API Application

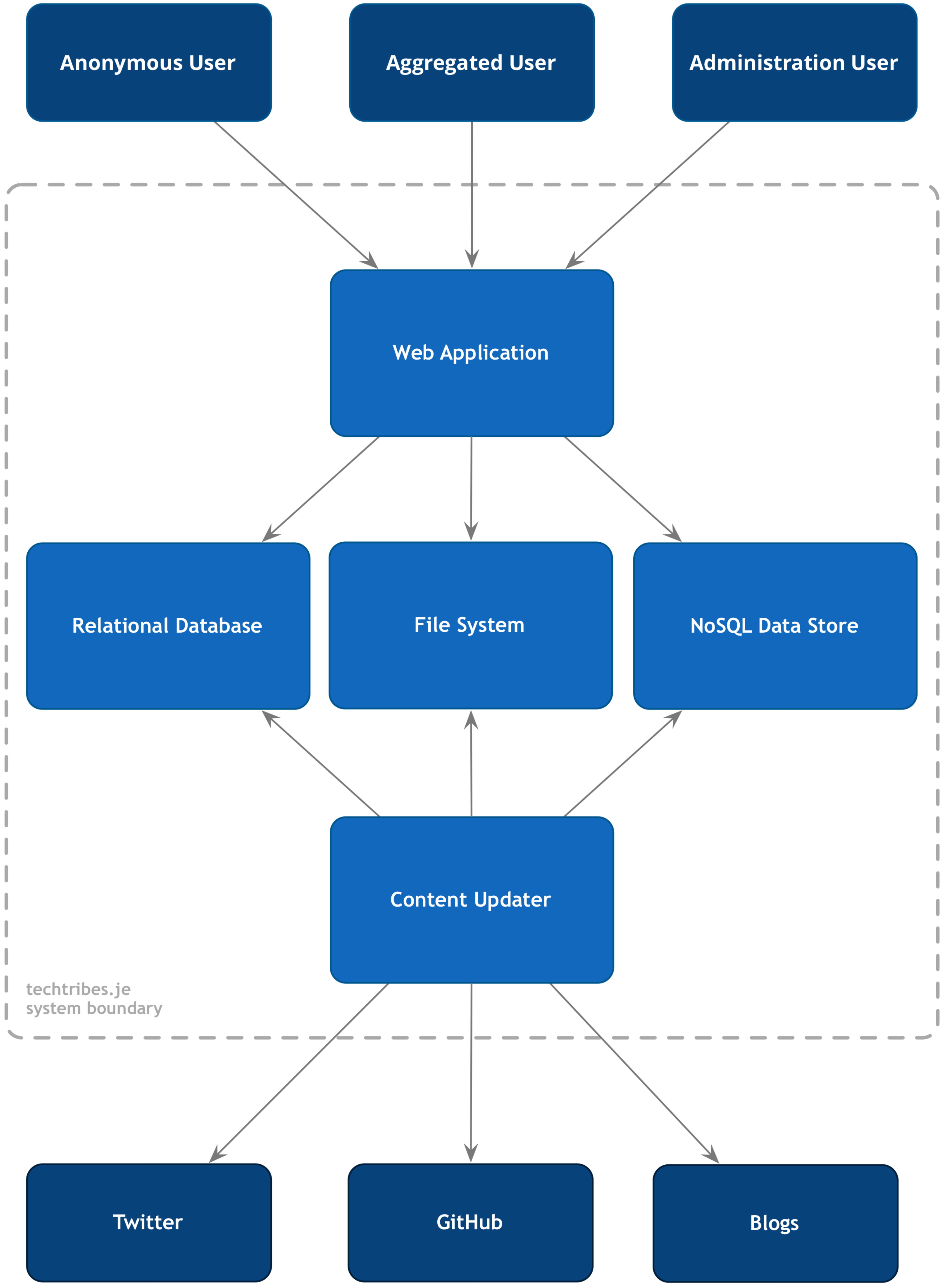
[Container: Java and Spring MVC]

Provides Internet banking functionality via a JSON/HTTPS API.

Mainframe Banking System Facade

[Component: Spring Bean]

A facade onto the mainframe banking system.



Anonymous User

Aggregated User

Administration User

Web Application

Relational Database

File System

NoSQL Data Store

Content Updater

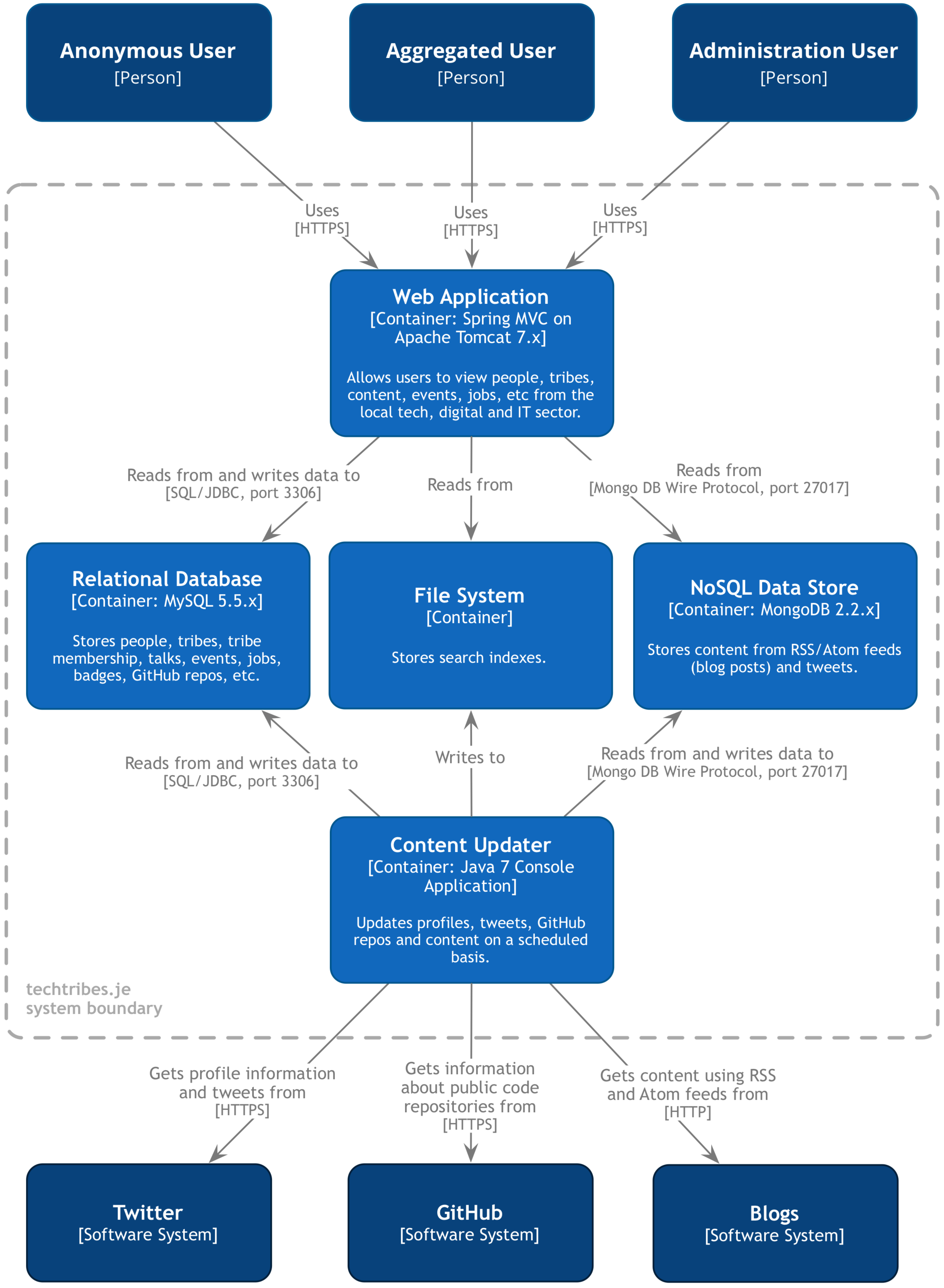
techtribes.je
system boundary

Twitter

GitHub

Blogs

[Containers] techtribes.je



Anonymous User
[Person]

Aggregated User
[Person]

Administration User
[Person]

Uses
[HTTPS]

Uses
[HTTPS]

Uses
[HTTPS]

Web Application
[Container: Spring MVC on Apache Tomcat 7.x]

Allows users to view people, tribes, content, events, jobs, etc from the local tech, digital and IT sector.

Reads from and writes data to
[SQL/JDBC, port 3306]

Reads from

Reads from
[Mongo DB Wire Protocol, port 27017]

Relational Database
[Container: MySQL 5.5.x]

Stores people, tribes, tribe membership, talks, events, jobs, badges, GitHub repos, etc.

File System
[Container]

Stores search indexes.

NoSQL Data Store
[Container: MongoDB 2.2.x]

Stores content from RSS/Atom feeds (blog posts) and tweets.

Reads from and writes data to
[SQL/JDBC, port 3306]

Writes to

Reads from and writes data to
[Mongo DB Wire Protocol, port 27017]

Content Updater
[Container: Java 7 Console Application]

Updates profiles, tweets, GitHub repos and content on a scheduled basis.

techtribes.je
system boundary

Gets profile information and tweets from
[HTTPS]

Gets information about public code repositories from
[HTTPS]

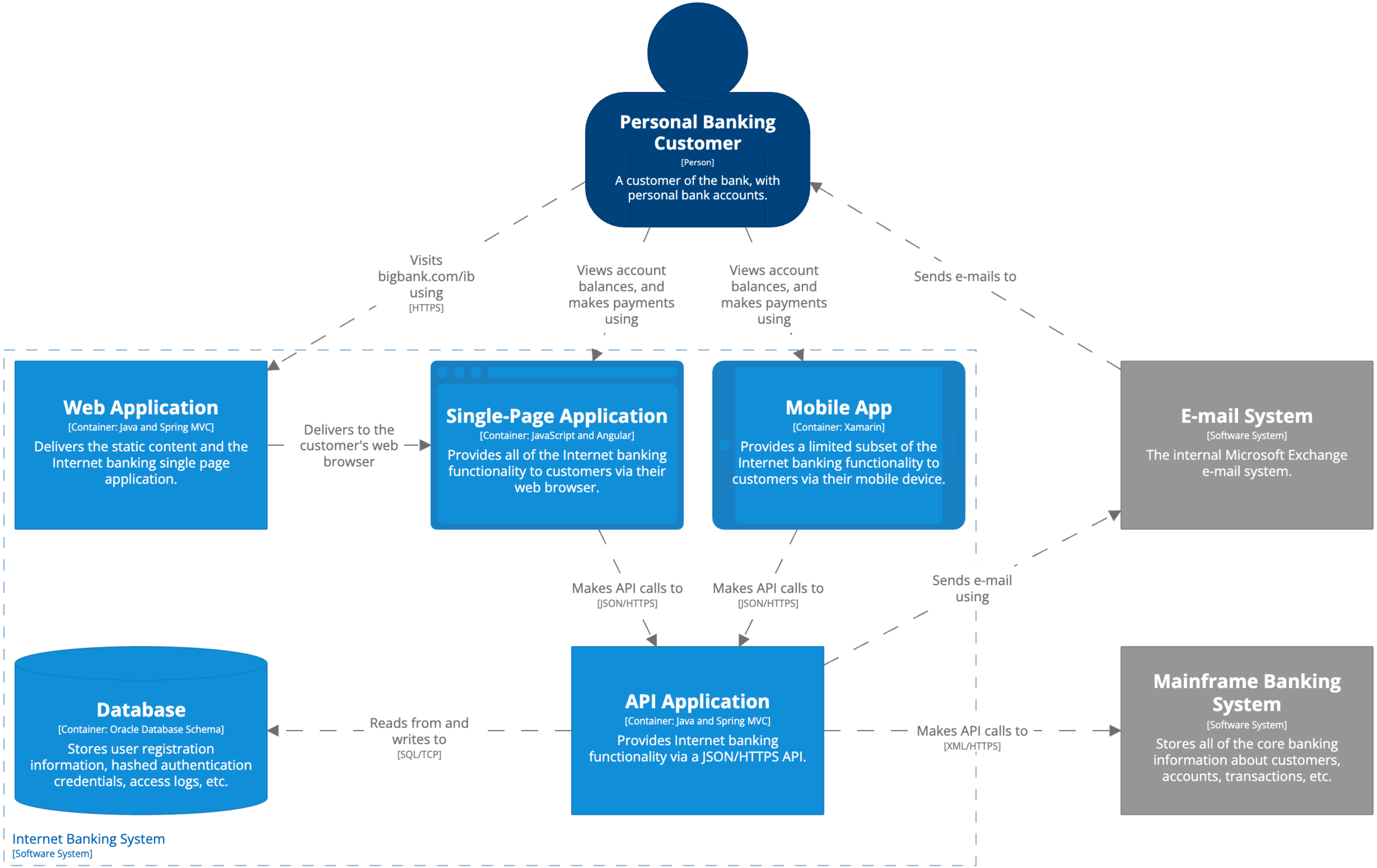
Gets content using RSS and Atom feeds from
[HTTP]

Twitter
[Software System]

GitHub
[Software System]

Blogs
[Software System]

[Containers] techtribes.je



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Visits bigbank.com/ib using
[HTTPS]

Views account balances, and makes payments using

Views account balances, and makes payments using

Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Delivers to the customer's web browser

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sends e-mail using

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

Reads from and writes to
[SQL/TCP]

API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

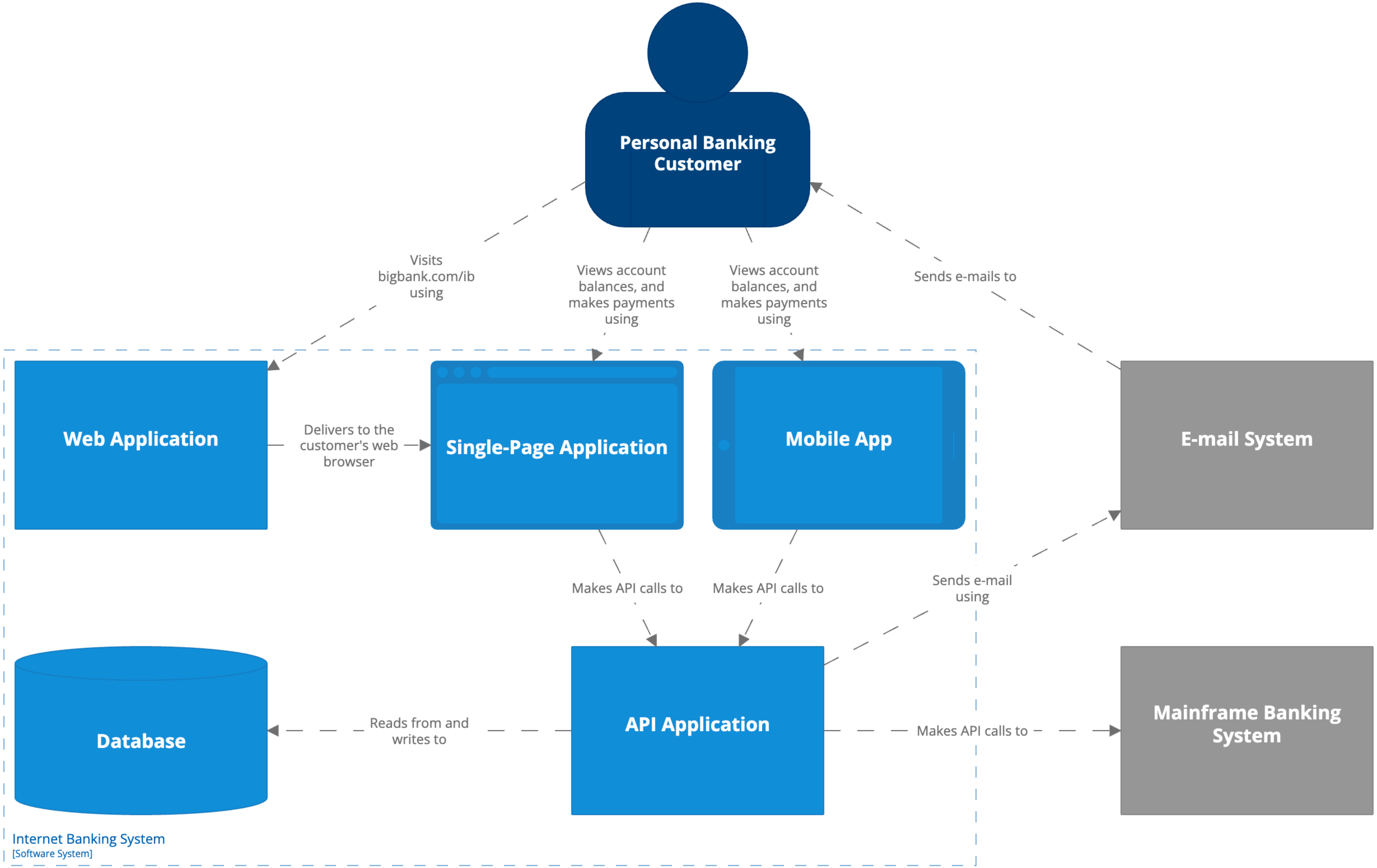
Makes API calls to
[XML/HTTPS]

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

[Container] Internet Banking System

The container diagram for the Internet Banking System - diagram created with Structurizr.
Wednesday, 22 March 2023 at 08:16 Greenwich Mean Time

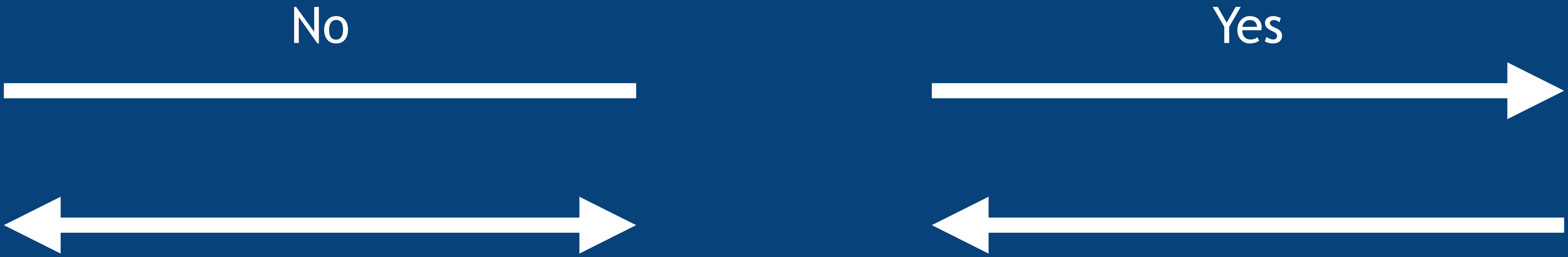


[Container] Internet Banking System

The container diagram for the Internet Banking System - diagram created with Structurizr.
Wednesday, 22 March 2023 at 08:16 Greenwich Mean Time

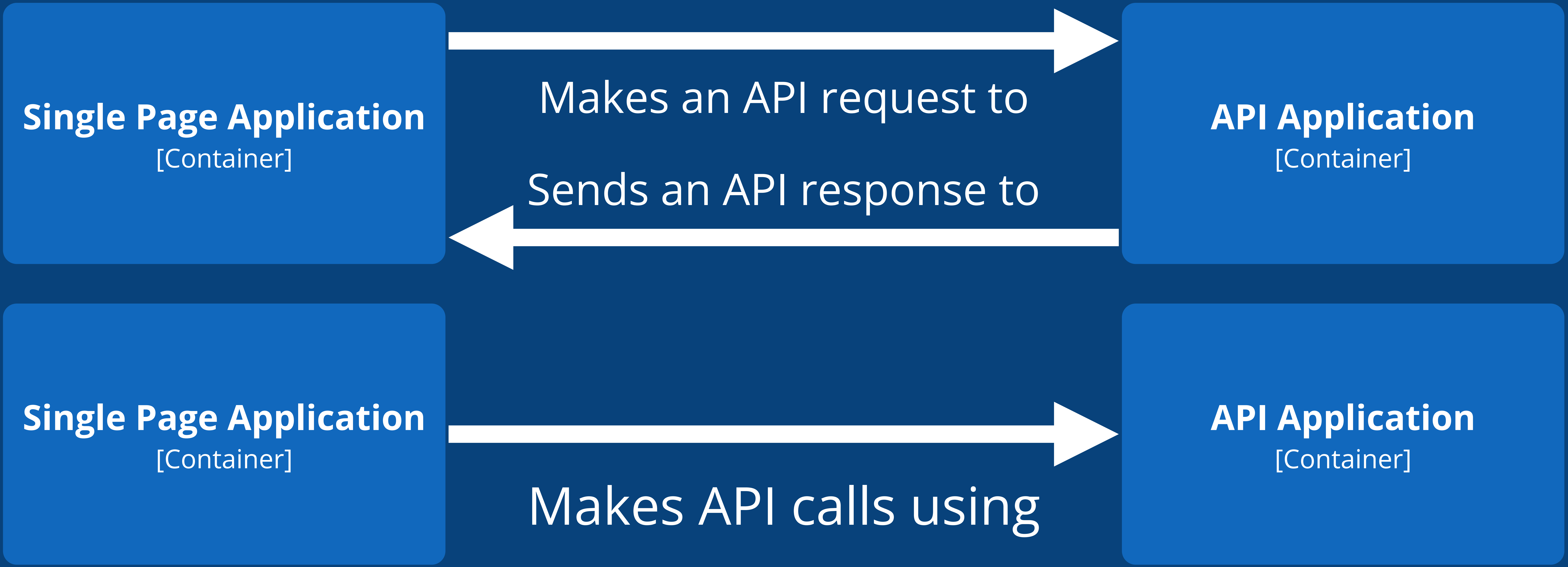
Lines

Favour uni-directional lines showing the most important dependencies or data flow, with an annotation to be explicit about the purpose of the line and direction



No

Yes



Single Page Application
[Container]

Makes an API request to

Sends an API response to

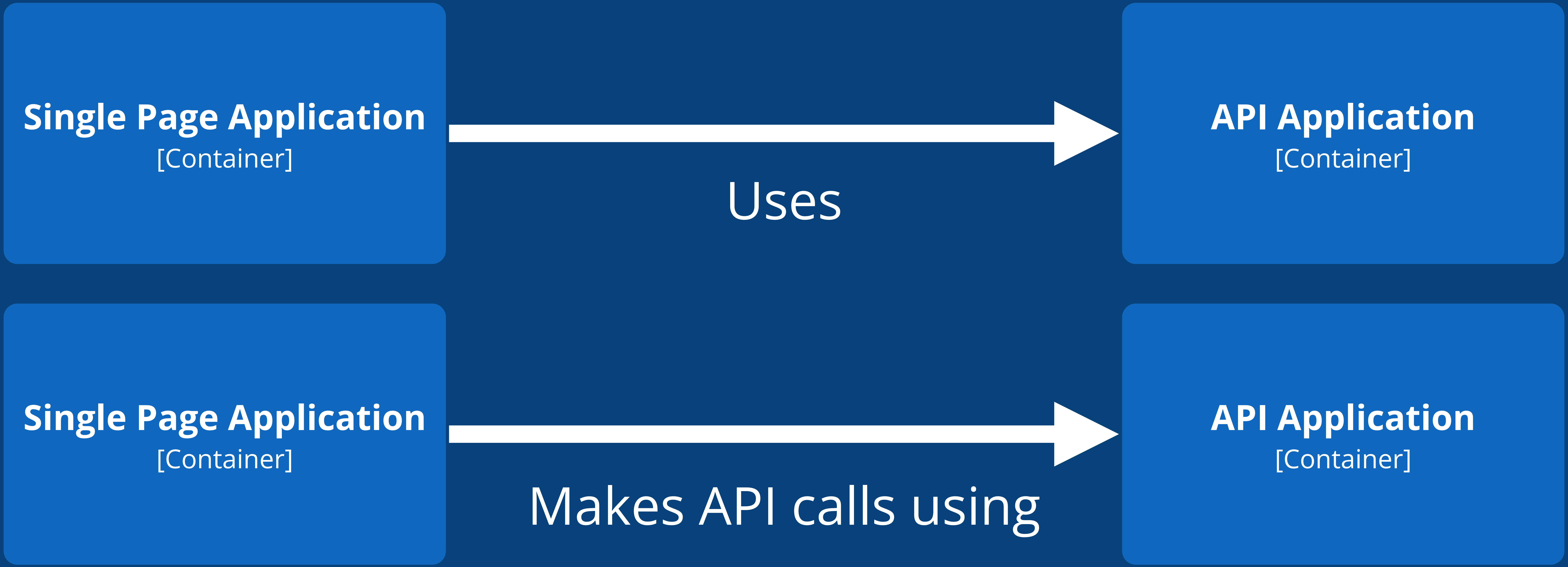
API Application
[Container]

Single Page Application
[Container]

Makes API calls using

API Application
[Container]

Summarise the intent of the relationship



Single Page Application
[Container]

Uses

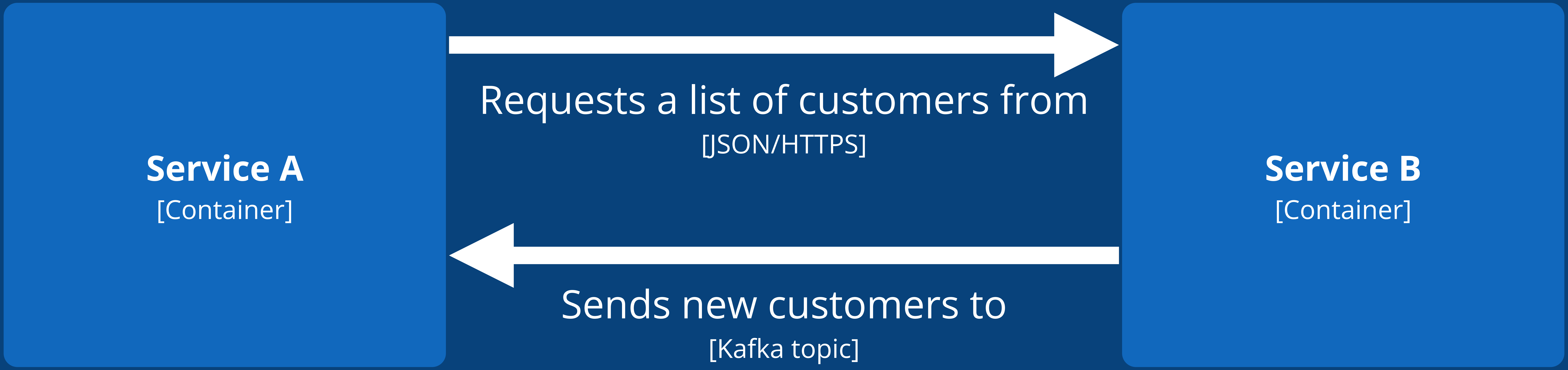
API Application
[Container]

Single Page Application
[Container]

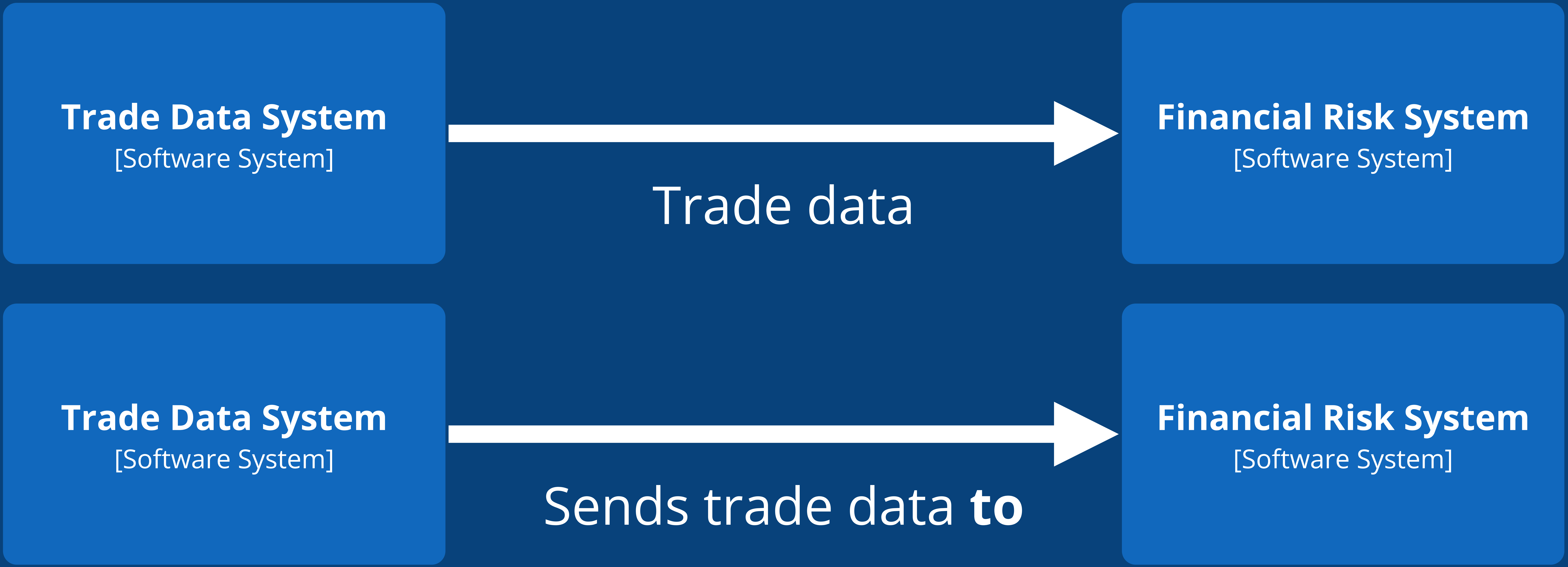
Makes API calls using

API Application
[Container]

Summarise, yet be specific



Show both directions when the intents are different



Trade Data System
[Software System]

Trade data

Financial Risk System
[Software System]

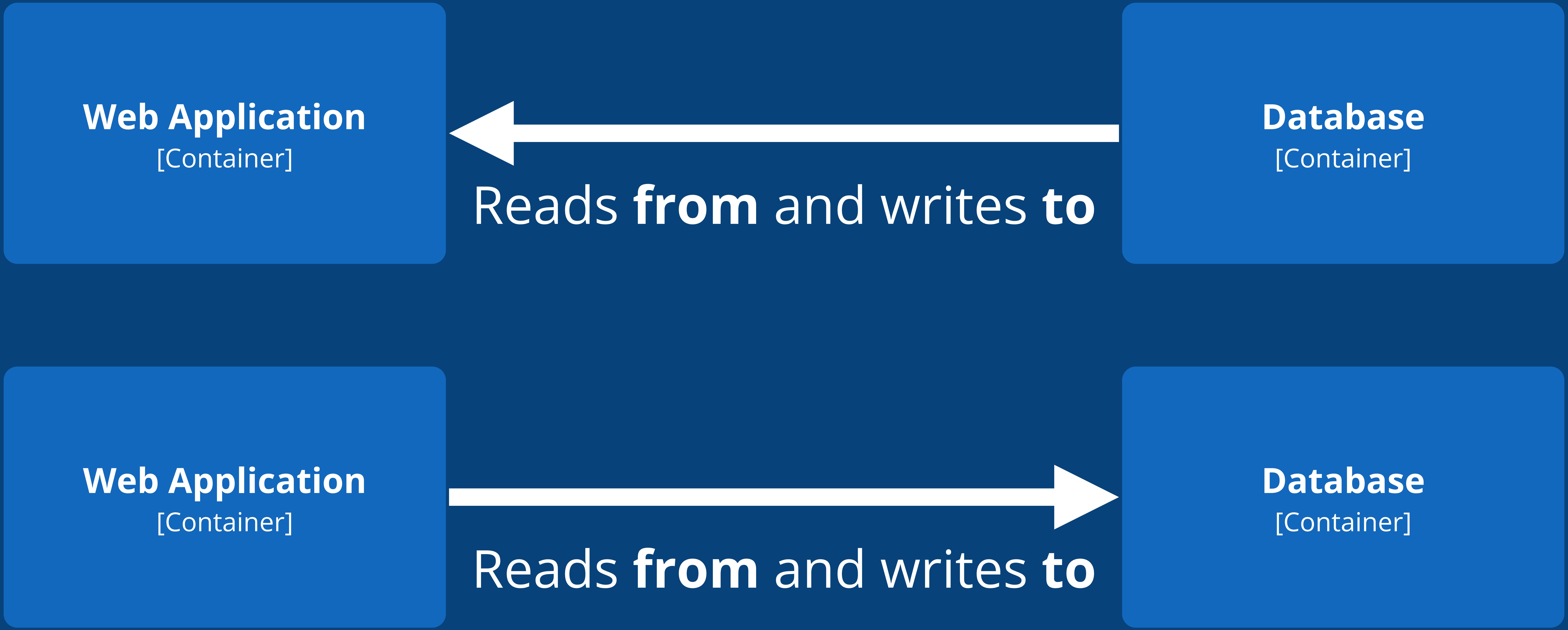
Trade Data System
[Software System]

Sends trade data **to**

Financial Risk System
[Software System]

Add more words to make the intent explicit

If in doubt, read the relationship



Web Application
[Container]

Reads **from** and writes **to**

Database
[Container]

Web Application
[Container]

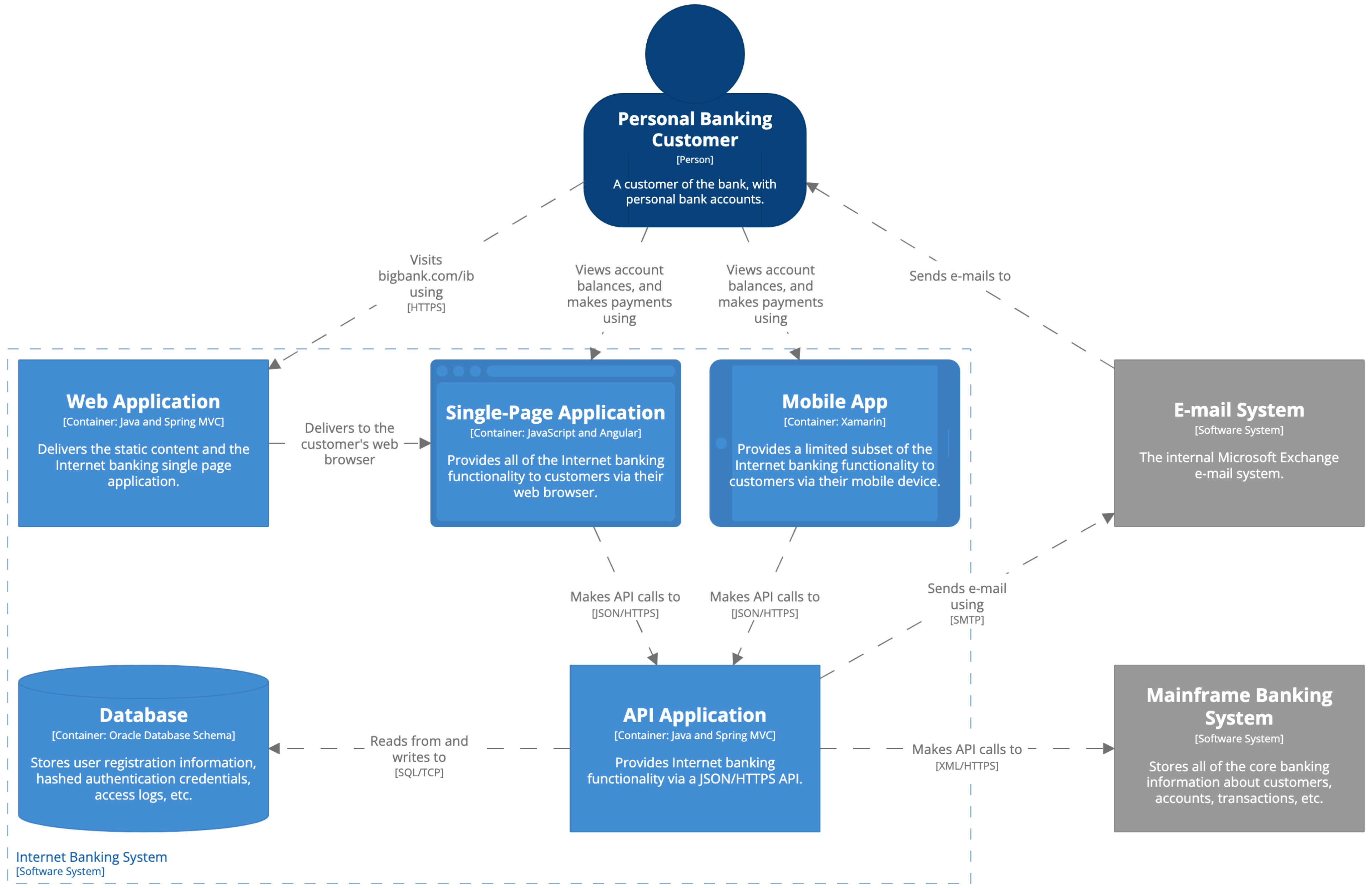
Reads **from** and writes **to**

Database
[Container]

Key/legend

Explain shapes, line styles, colours, borders, acronyms, etc

… even if your notation seems obvious!



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Visits bigbank.com/ib using
[HTTPS]

Views account balances, and makes payments using

Views account balances, and makes payments using

Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Delivers to the customer's web browser

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sends e-mail using
[SMTP]

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

Reads from and writes to
[SQL/TCP]

API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

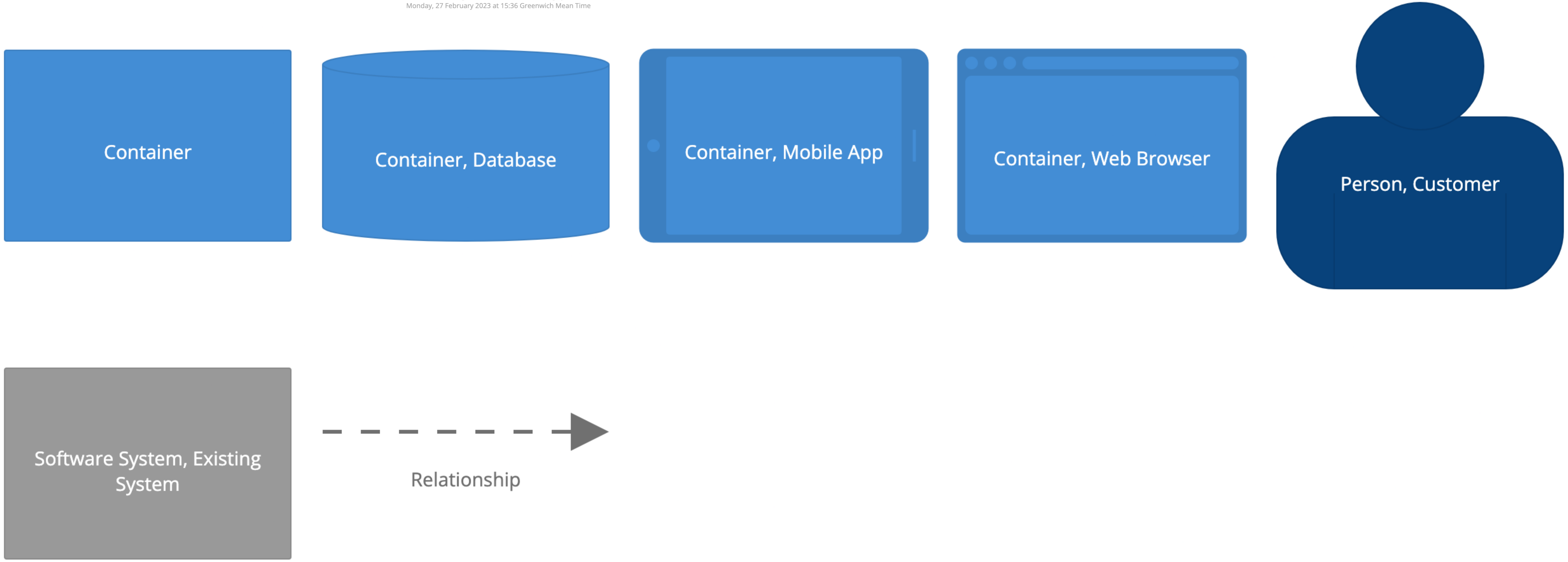
Makes API calls to
[XML/HTTPS]

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

[Container] Internet Banking System

The container diagram for the Internet Banking System.
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Container

Container, Database

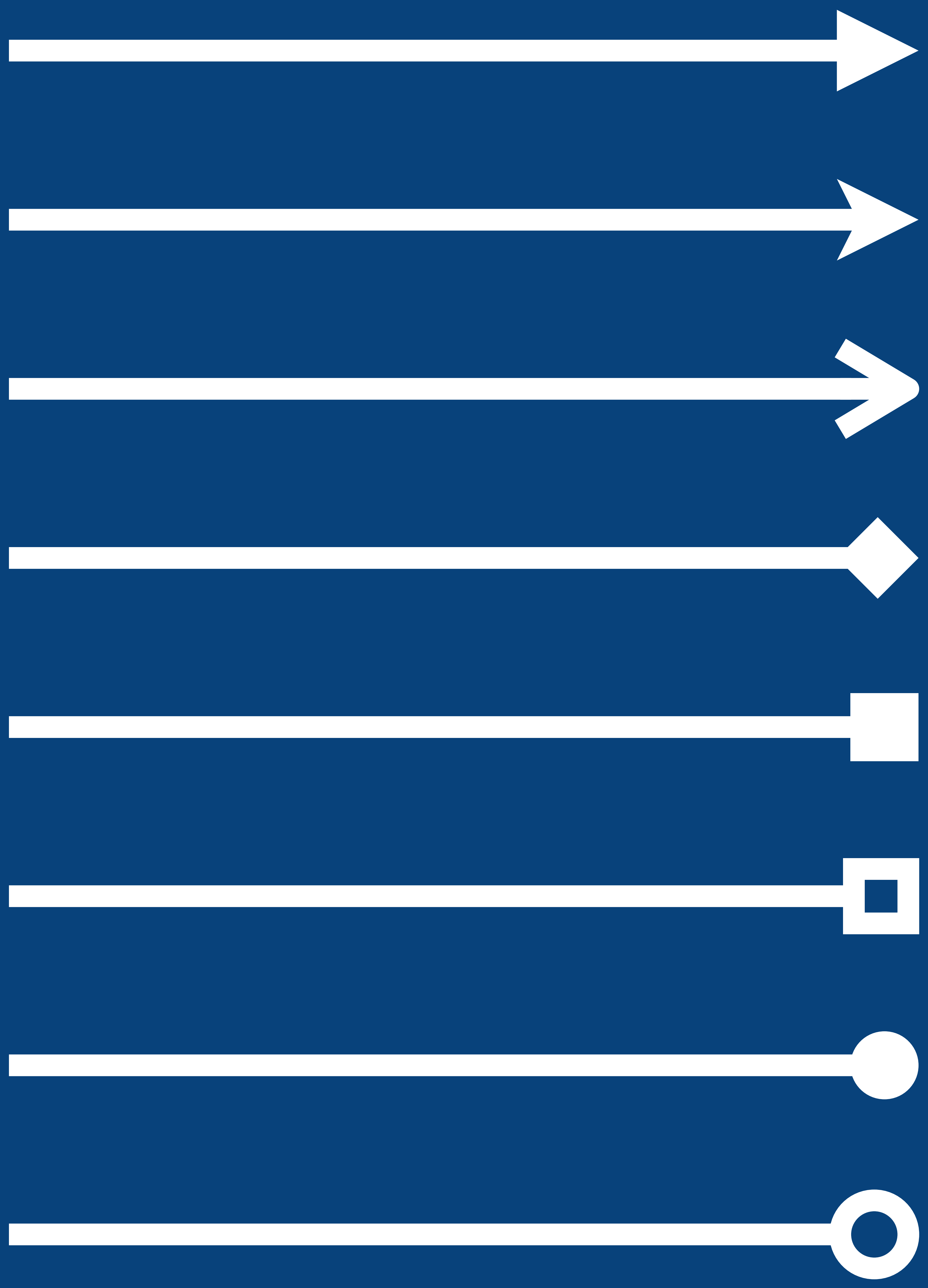
Container, Mobile App

Container, Web Browser

Person, Customer

Software System, Existing System

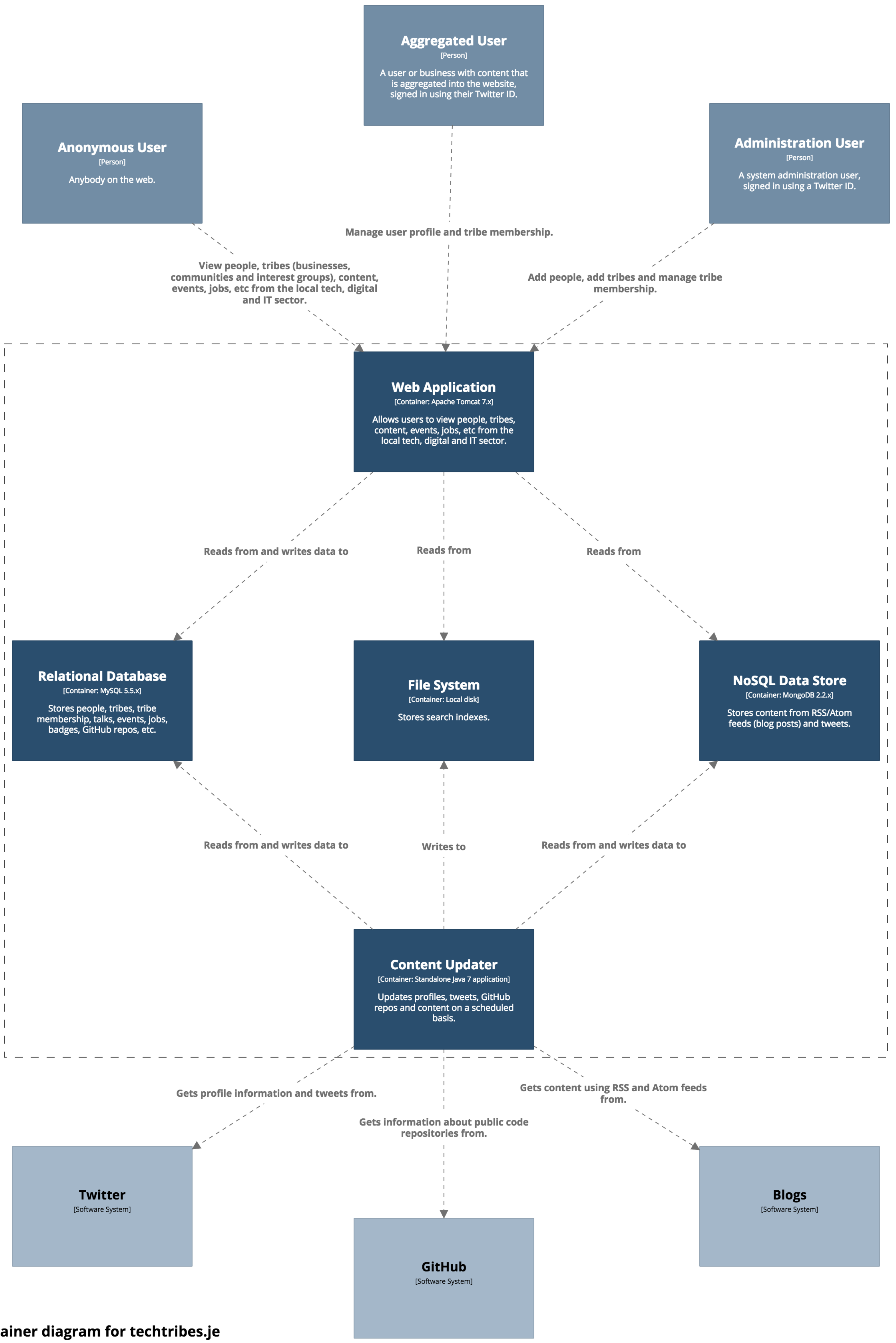
Relationship



Arrowheads

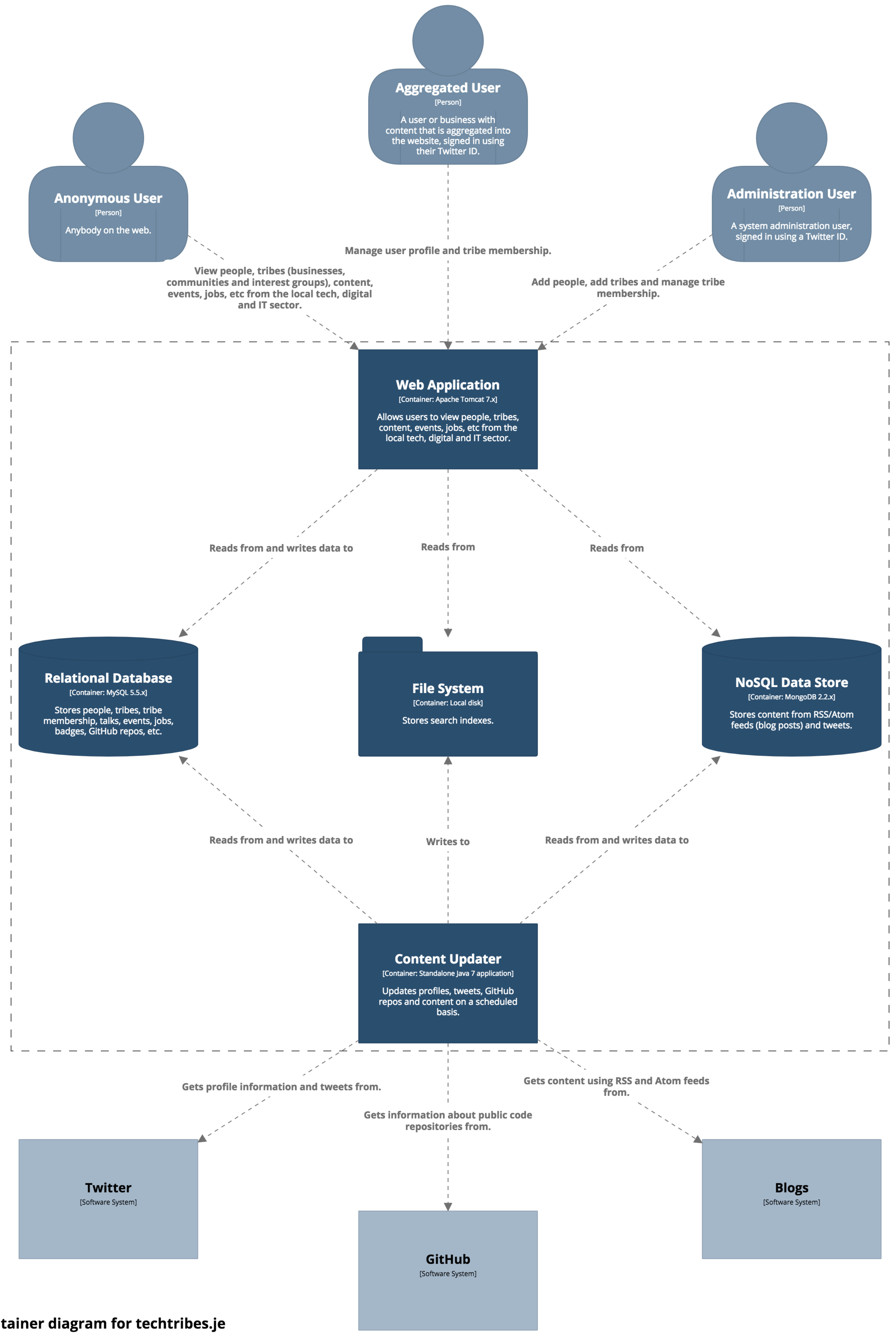
Be careful, using different arrowheads is very subtle; readers may miss them

Use shape, colour and size to **complement** a diagram that already makes sense


Aggregated User
[Person]
A user or business with content that is aggregated into the website, signed in using their Twitter ID.
Anonymous User
[Person]
Anybody on the web.
Administration User
[Person]
A system administration user, signed in using a Twitter ID.
Manage user profile and tribe membership.
View people, tribes (businesses, communities and interest groups), content, events, jobs, etc from the local tech, digital and IT sector.
Add people, add tribes and manage tribe membership.
Web Application
[Container: Apache Tomcat 7.x]
Allows users to view people, tribes, content, events, jobs, etc from the local tech, digital and IT sector.
Reads from and writes data to
Reads from
Reads from
Relational Database
[Container: MySQL 5.5.x]
Stores people, tribes, tribe membership, talks, events, jobs, badges, GitHub repos, etc.
File System
[Container: Local disk]
Stores search indexes.
NoSQL Data Store
[Container: MongoDB 2.2.x]
Stores content from RSS/Atom feeds (blog posts) and tweets.
Reads from and writes data to
Writes to
Reads from and writes data to
Content Updater
[Container: Standalone Java 7 application]
Updates profiles, tweets, GitHub repos and content on a scheduled basis.
Gets profile information and tweets from.
Gets information about public code repositories from.
Gets content using RSS and Atom feeds from.
Twitter
[Software System]
GitHub
[Software System]
Blogs
[Software System]

Container diagram for techtribes.je

Friday 12 May 2017 10:42 UTC


Aggregated User
[Person]
A user or business with content that is aggregated into the website, signed in using their Twitter ID.
Anonymous User
[Person]
Anybody on the web.
Administration User
[Person]
A system administration user, signed in using a Twitter ID.
Manage user profile and tribe membership.
View people, tribes (businesses, communities and interest groups), content, events, jobs, etc from the local tech, digital and IT sector.
Add people, add tribes and manage tribe membership.
Web Application
[Container: Apache Tomcat 7.x]
Allows users to view people, tribes, content, events, jobs, etc from the local tech, digital and IT sector.
Reads from and writes data to
Reads from
Reads from
Relational Database
[Container: MySQL 5.5.x]
Stores people, tribes, tribe membership, talks, events, jobs, badges, GitHub repos, etc.
File System
[Container: Local disk]
Stores search indexes.
NoSQL Data Store
[Container: MongoDB 2.2.x]
Stores content from RSS/Atom feeds (blog posts) and tweets.
Reads from and writes data to
Writes to
Reads from and writes data to
Content Updater
[Container: Standalone Java 7 application]
Updates profiles, tweets, GitHub repos and content on a scheduled basis.
Gets profile information and tweets from.
Gets information about public code repositories from.
Gets content using RSS and Atom feeds from.
Twitter
[Software System]
GitHub
[Software System]
Blogs
[Software System]

Container diagram for techtribes.je

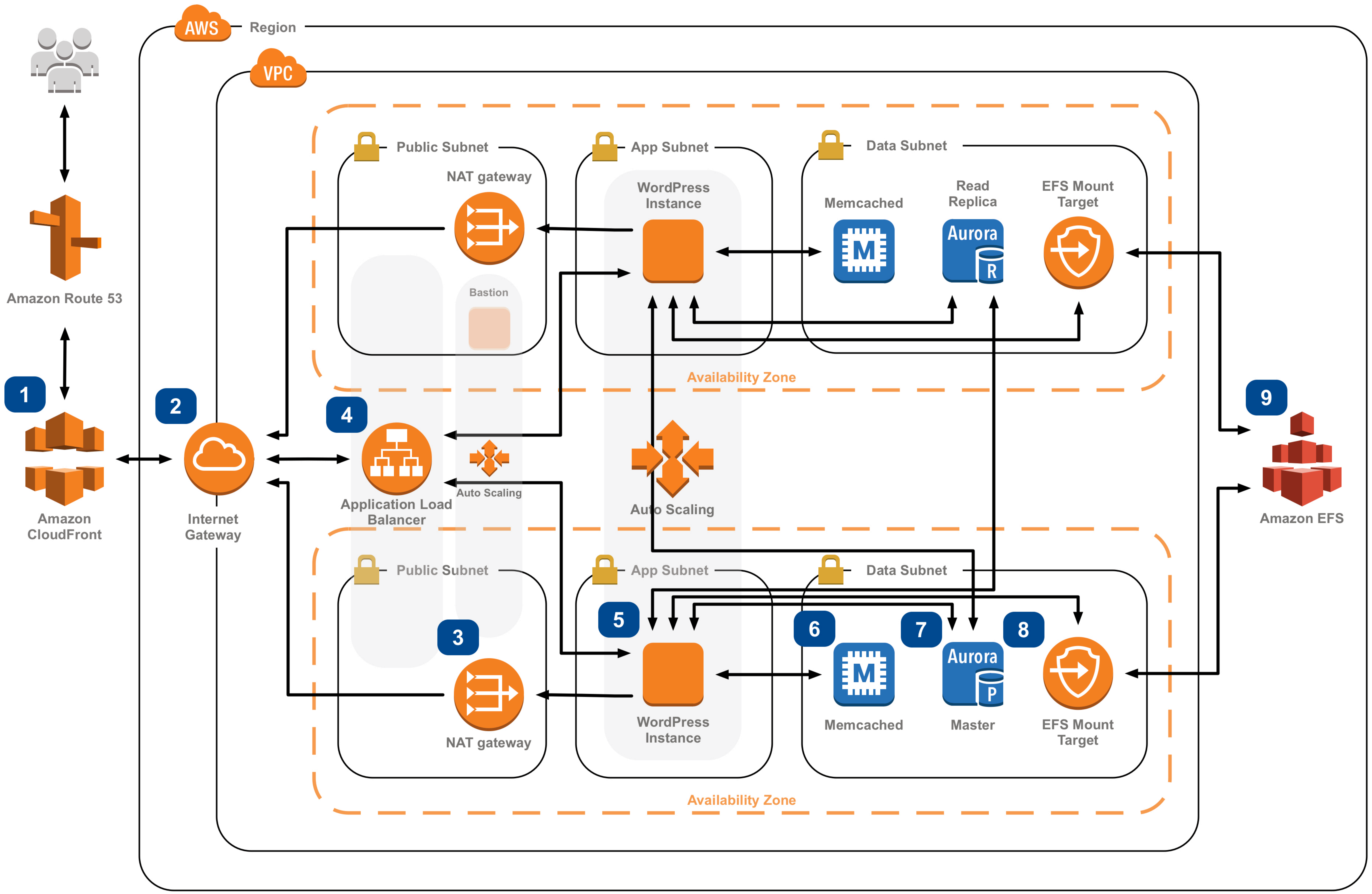
Monday 27 February 2017 09:39 UTC

Be careful with **icons**

WordPress Hosting

How to run WordPress on AWS

WordPress is one of the world's most popular web publishing platforms, being used to publish 27% of all websites, from personal blogs to some of the biggest news sites. This reference architecture simplifies the complexity of deploying a scalable and highly available WordPress site on AWS.



1 Static and dynamic content is delivered by **Amazon CloudFront**.

2 An **Internet gateway** allows communication between instances in your VPC and the Internet.

3 **NAT gateways** in each public subnet enable Amazon EC2 instances in private subnets (App & Data) to access the Internet.

4 Use an **Application Load Balancer** to distribute web traffic across an Auto Scaling Group of Amazon EC2 instances in multiple AZs.

5 Run your WordPress site using an **Auto Scaling group** of **Amazon EC2 instances**. Install the latest versions of WordPress, Apache web server, PHP 7, and OPcache and build an Amazon Machine Image that will be used by the Auto Scaling group launch configuration to launch new instances in the Auto Scaling group.

6 If database access patterns are read-heavy, consider using a WordPress plugin that takes advantage of a caching layer like **Amazon ElastiCache** (**Memcached**) in front of the database layer to cache frequently accessed data.

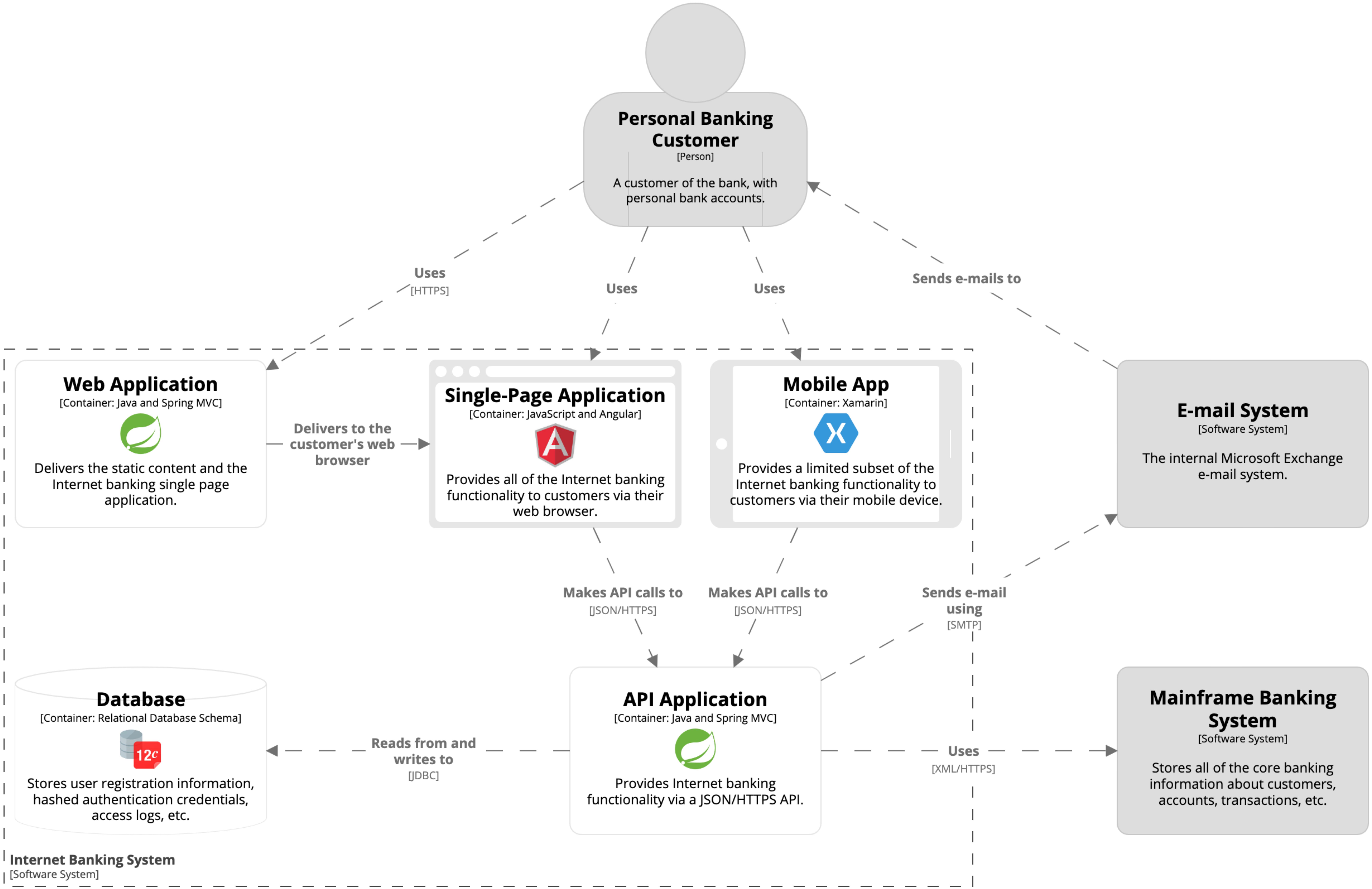
7 Simplify your database administration by running your database layer in **Amazon RDS** using either Aurora or MySQL.

8 Amazon EC2 instances access shared WordPress data in an Amazon EFS file system using **Mount Targets** in each AZ in your VPC.

9 Use **Amazon EFS**, a simple, highly available, and scalable network file system so WordPress instances have access to your shared, unstructured WordPress data, like php files, config, themes, plugins, etc.

amazon web services

AWS Reference Architectures



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Database
[Container: Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

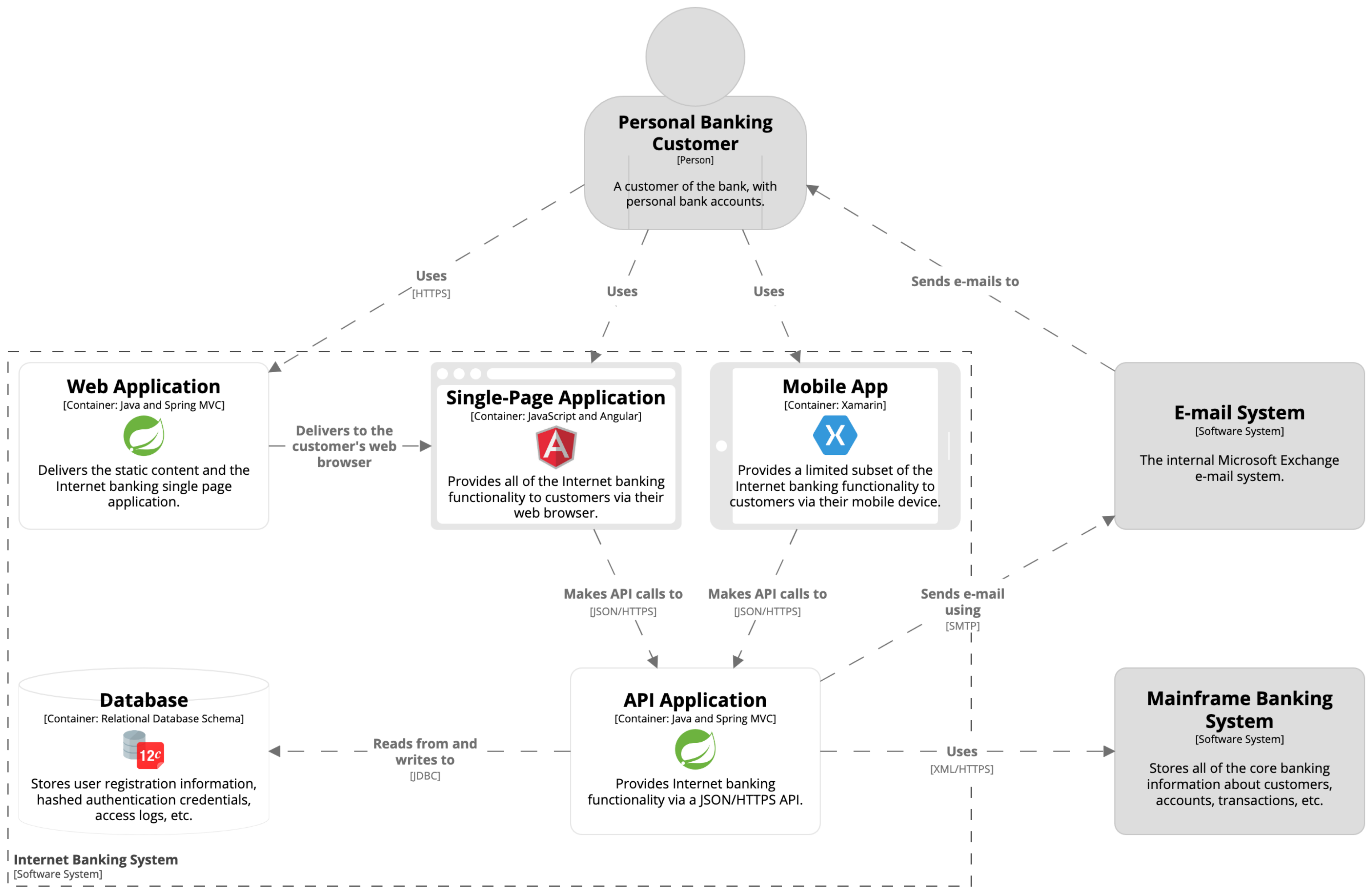
API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

Container diagram for Internet Banking System

The container diagram for the Internet Banking System.
Workspace last modified: Sat Jan 11 2020 14:47:20 GMT+0000 (Greenwich Mean Time)



Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.

Uses
[HTTPS]

Uses

Uses

Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Delivers to the customer's web browser

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Makes API calls to
[JSON/HTTPS]

Makes API calls to
[JSON/HTTPS]

Sends e-mail using
[SMTP]

Database
[Container: Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

Reads from and writes to
[JDBC]

API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

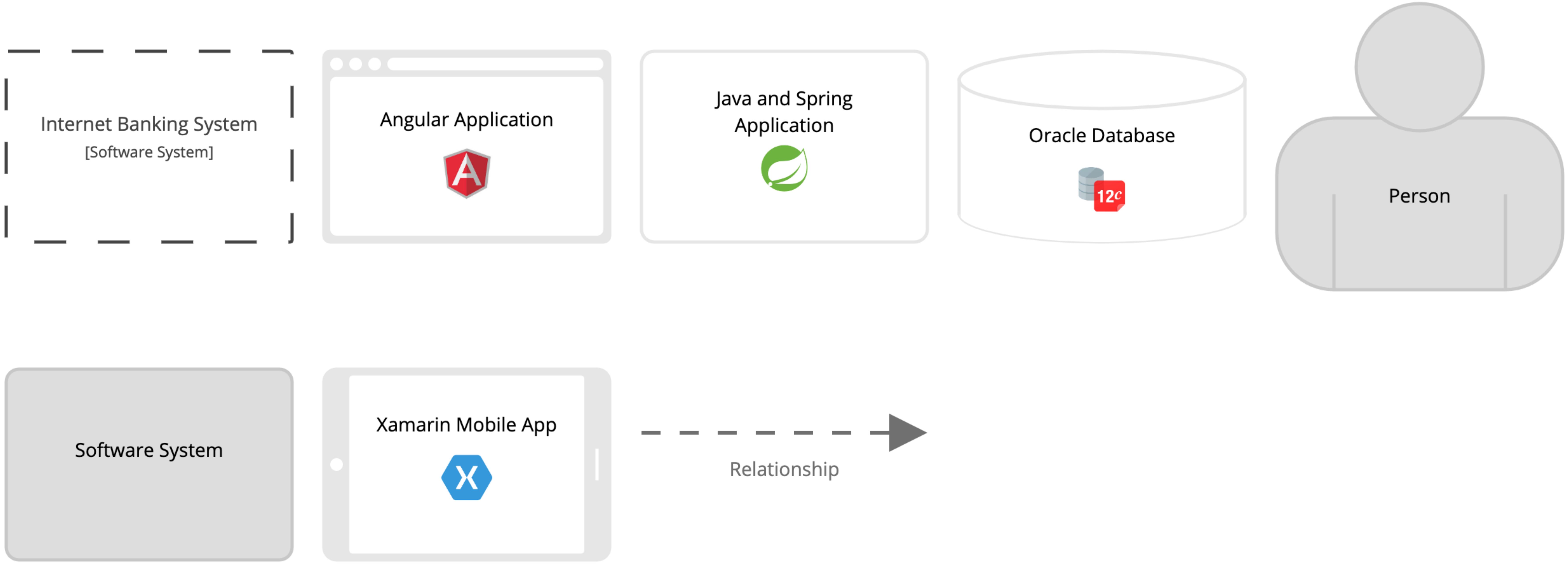
Uses
[XML/HTTPS]

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Internet Banking System
[Software System]

Container diagram for Internet Banking System

The container diagram for the Internet Banking System.
Workspace last modified: Sat Jan 11 2020 14:47:20 GMT+0000 (Greenwich Mean Time)



Internet Banking System
[Software System]

Angular Application

Java and Spring Application

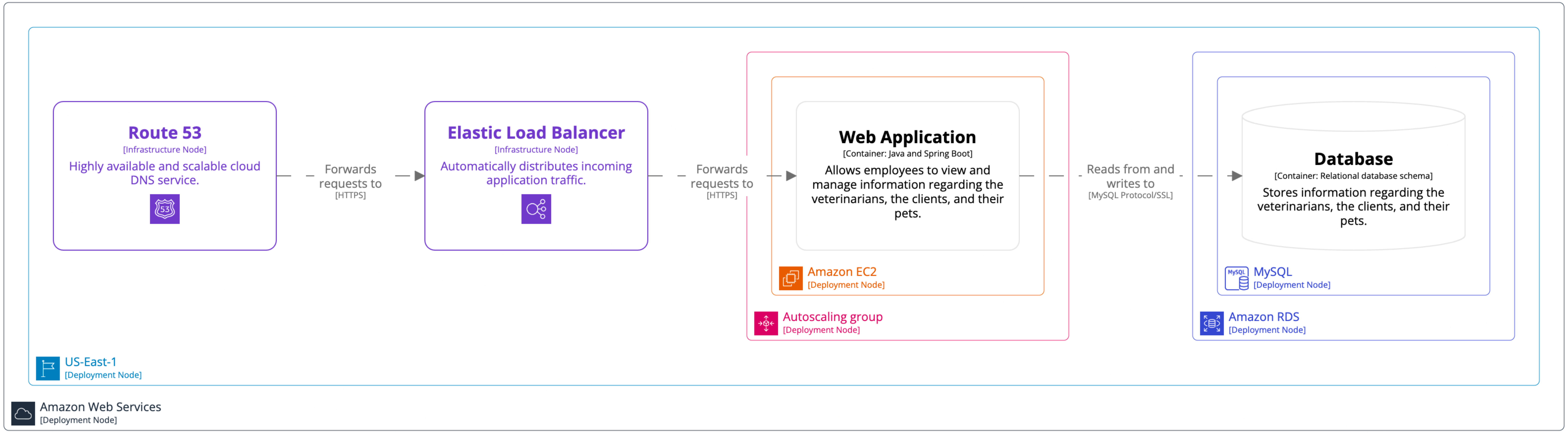
Oracle Database

Person

Software System

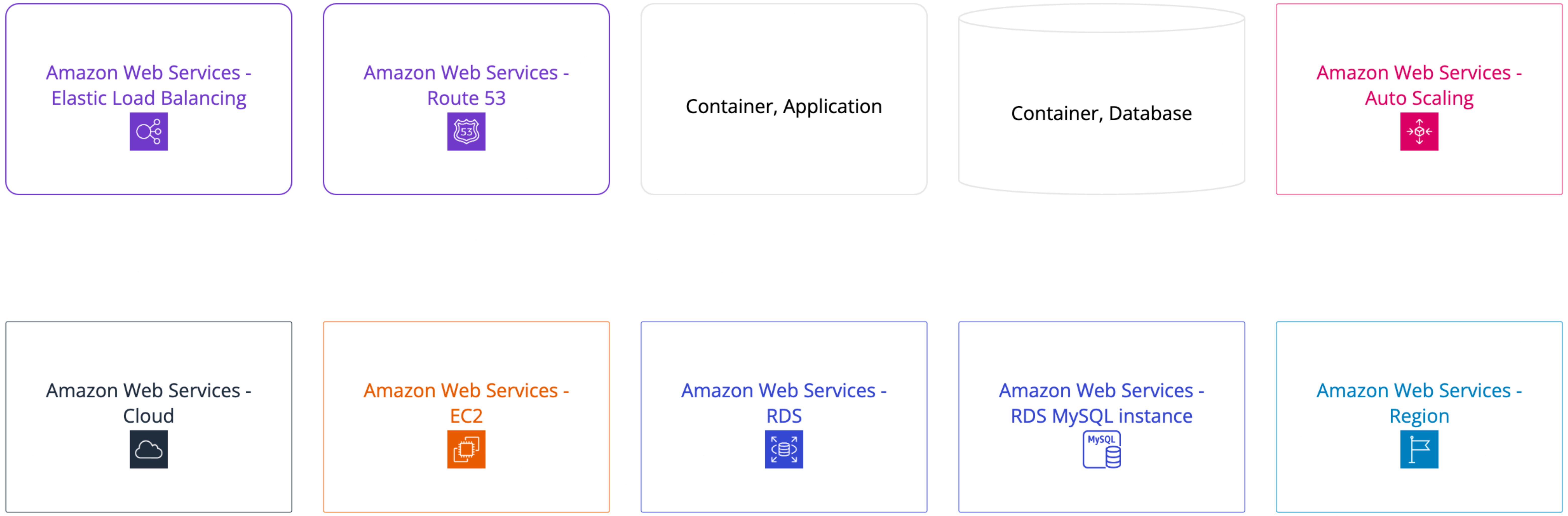
Xamarin Mobile App

Relationship



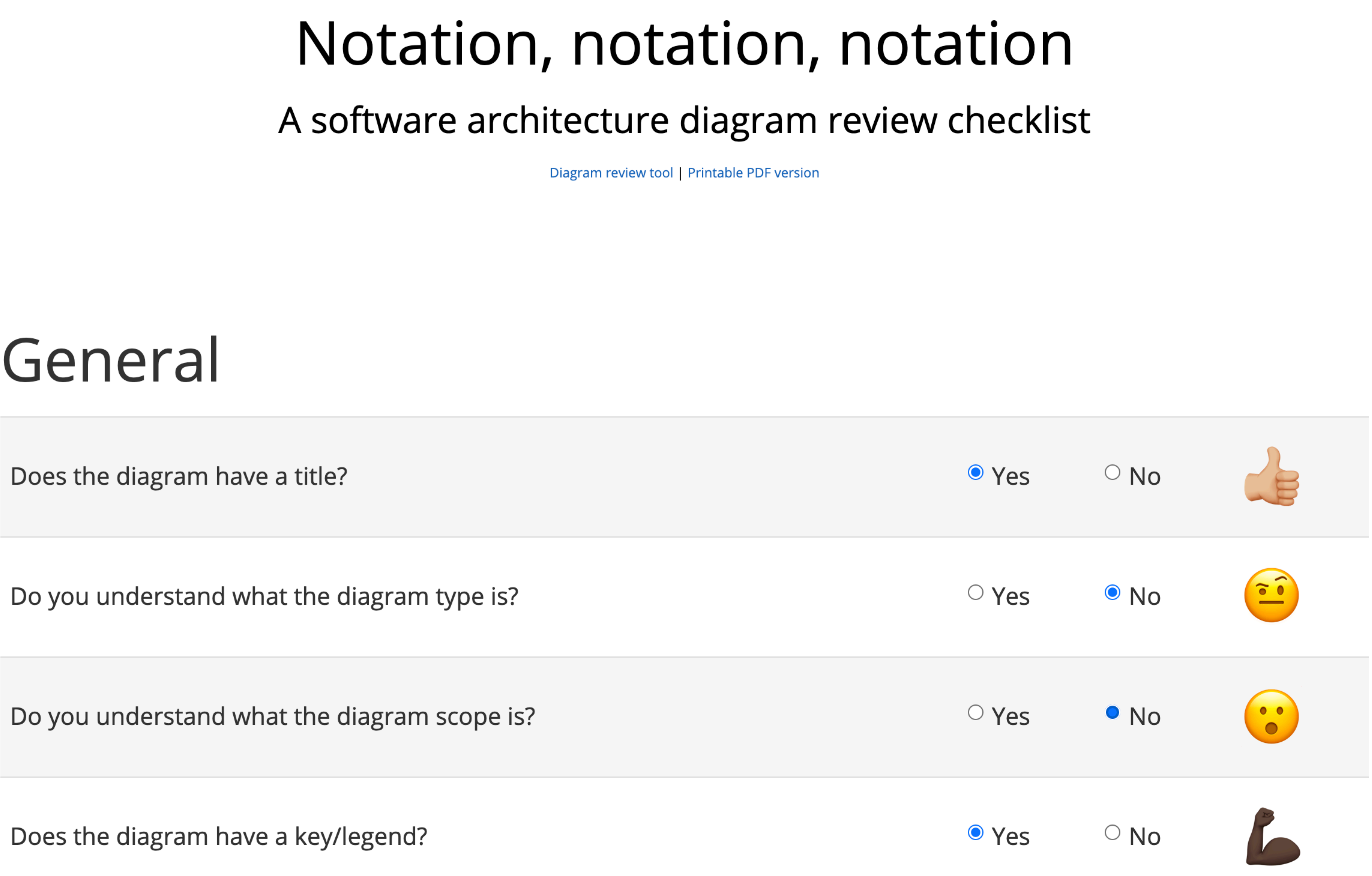
[Deployment] Spring PetClinic - Live

Sunday, 5 March 2023 at 09:41 Greenwich Mean Time


Amazon Web Services - Elastic Load Balancing
Amazon Web Services - Route 53
Container, Application
Container, Database
Amazon Web Services - Auto Scaling
Amazon Web Services - Cloud
Amazon Web Services - EC2
Amazon Web Services - RDS
Amazon Web Services - RDS MySQL instance
Amazon Web Services - Region

Increase the **readability** of software architecture diagrams, so they can **stand alone**

Any narrative should **complement** the diagram rather than explain it



Notation, notation, notation

A software architecture diagram review checklist

Diagram review tool | Printable PDF version

General

Does the diagram have a title?	● Yes	○ No	👍
Do you understand what the diagram type is?	○ Yes	● No	🤨
Do you understand what the diagram scope is?	○ Yes	● No	😯
Does the diagram have a key/legend?	● Yes	○ No	💪

c4model.com

Abstractions first, notation second

Ensure that your team has a ubiquitous language to describe software architecture

The C4 model is...

A set of hierarchical abstractions

(software systems, containers, components, and code)

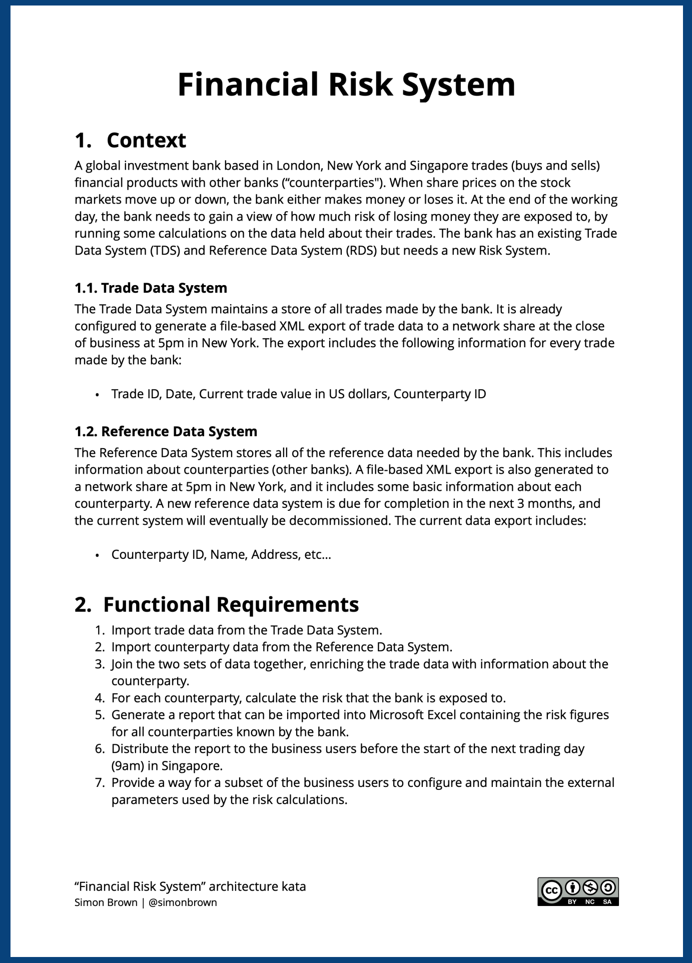
A set of hierarchical diagrams

(system context, containers, components, and code)

Notation independent

Tooling independent

Draw **System Context** and **Container** diagrams to describe a solution for the "Financial Risk System"



Financial Risk System

1. Context

A global investment bank based in London, New York and Singapore trades (buys and sells) financial products with other banks ("counterparties"). When share prices on the stock markets move up or down, the bank either makes money or loses it. At the end of the working day, the bank needs to gain a view of how much risk of losing money they are exposed to, by running some calculations on the data held about their trades. The bank has an existing Trade Data System (TDS) and Reference Data System (RDS) but needs a new Risk System.

1.1. Trade Data System

The Trade Data System maintains a store of all trades made by the bank. It is already configured to generate a file-based XML export of trade data to a network share at the close of business at 5pm in New York. The export includes the following information for every trade made by the bank:

- Trade ID, Date, Current trade value in US dollars, Counterparty ID

1.2. Reference Data System

The Reference Data System stores all of the reference data needed by the bank. This includes information about counterparties (other banks). A file-based XML export is also generated to a network share at 5pm in New York, and it includes some basic information about each counterparty. A new reference data system is due for completion in the next 3 months, and the current system will eventually be decommissioned. The current data export includes:

- Counterparty ID, Name, Address, etc…

2. Functional Requirements

1. Import trade data from the Trade Data System.
2. Import counterparty data from the Reference Data System.
3. Join the two sets of data together, enriching the trade data with information about the counterparty.
4. For each counterparty, calculate the risk that the bank is exposed to.
5. Generate a report that can be imported into Microsoft Excel containing the risk figures for all counterparties known by the bank.
6. Distribute the report to the business users before the start of the next trading day (9am) in Singapore.
7. Provide a way for a subset of the business users to configure and maintain the external parameters used by the risk calculations.

"Financial Risk System" architecture kata
Simon Brown | @simonbrown

simonbrown.je

Designing software is where
the complexity should be,
not communicating it!

Similar levels of abstraction provide a way to easily **compare** solutions

The diagrams should spark
meaningful questions

No

“What does that arrow mean?”

“Why are some boxes red?”

“Is that a Java application?”

“Is that a monolithic application, or a collection of microservices?”

“How do the users get their reports?”

Yes

“What protocol are your two Java applications using to communicate with each other?”

“Why do you have two separate C# applications instead of one?”

“Why are you using MongoDB?”

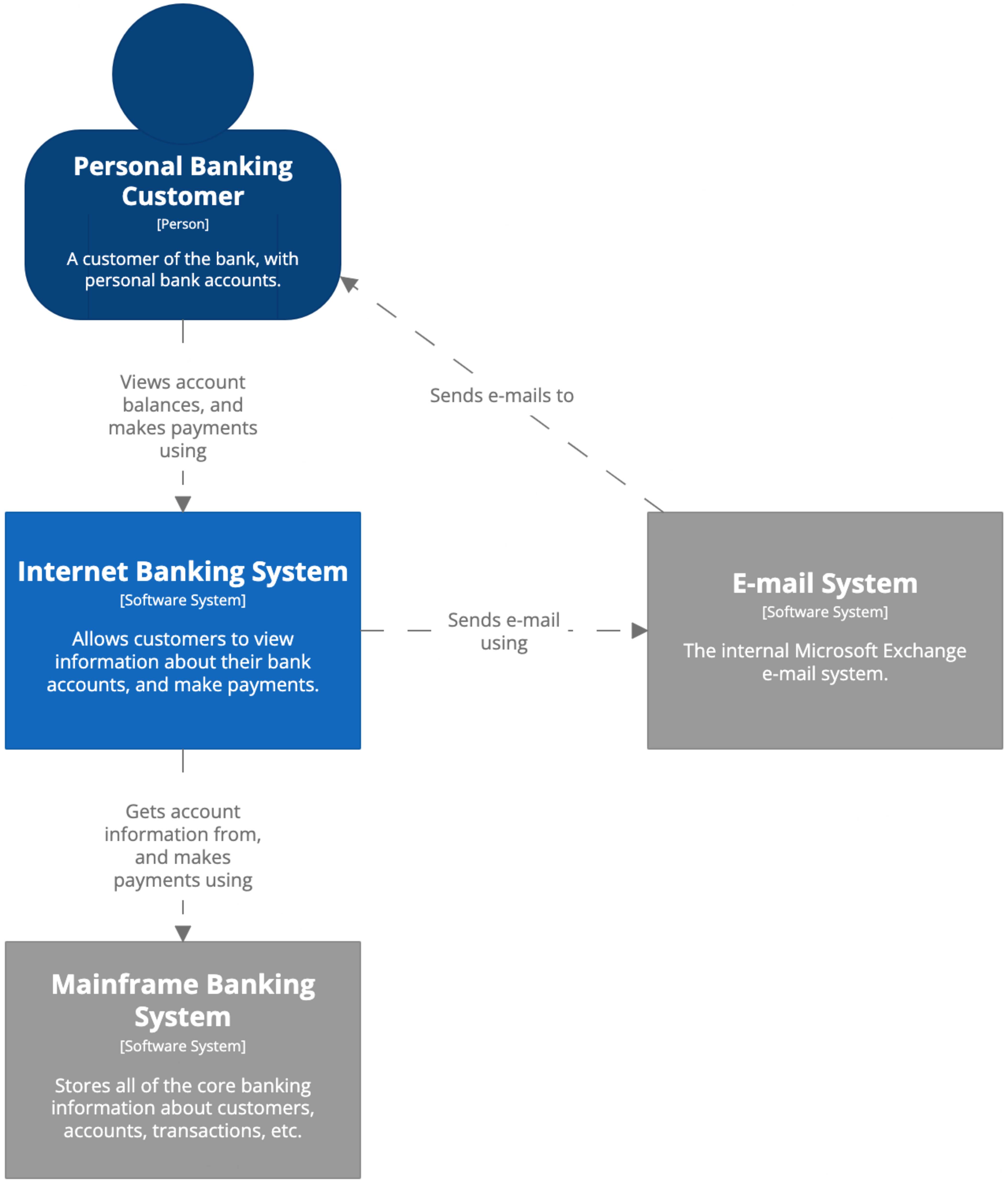
“Why are you using MySQL when our standard is Oracle?”

“Should we really build new applications with .NET Framework rather than .NET Core?”

Richer diagrams lead to richer **design discussions**

Richer diagrams lead to
better communication,
making it easier to scale teams

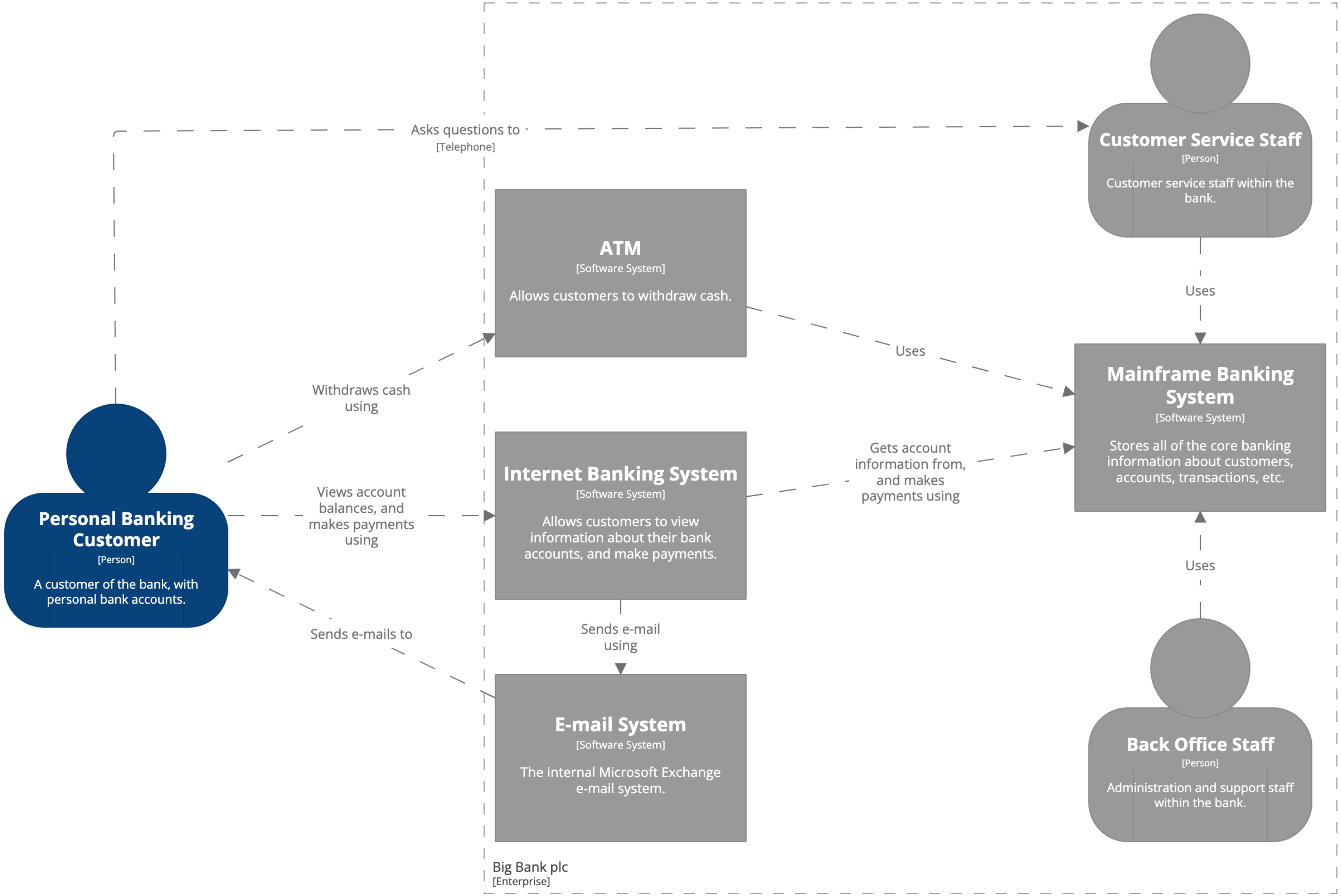
System landscape diagrams


Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Views account balances, and makes payments using
Sends e-mails to
Internet Banking System
[Software System]
Allows customers to view information about their bank accounts, and make payments.
Sends e-mail using
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Gets account information from, and makes payments using
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

[System Context] Internet Banking System

The system context diagram for the Internet Banking System.

Monday, 27 February 2023 at 15:25 Greenwich Mean Time


Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Asks questions to
[Telephone]
Withdraws cash using
Views account balances, and makes payments using
Sends e-mails to
ATM
[Software System]
Allows customers to withdraw cash.
Internet Banking System
[Software System]
Allows customers to view information about their bank accounts, and make payments.
Sends e-mail using
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Uses
Gets account information from, and makes payments using
Customer Service Staff
[Person]
Customer service staff within the bank.
Uses
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
Uses
Back Office Staff
[Person]
Administration and support staff within the bank.
Big Bank plc
[Enterprise]

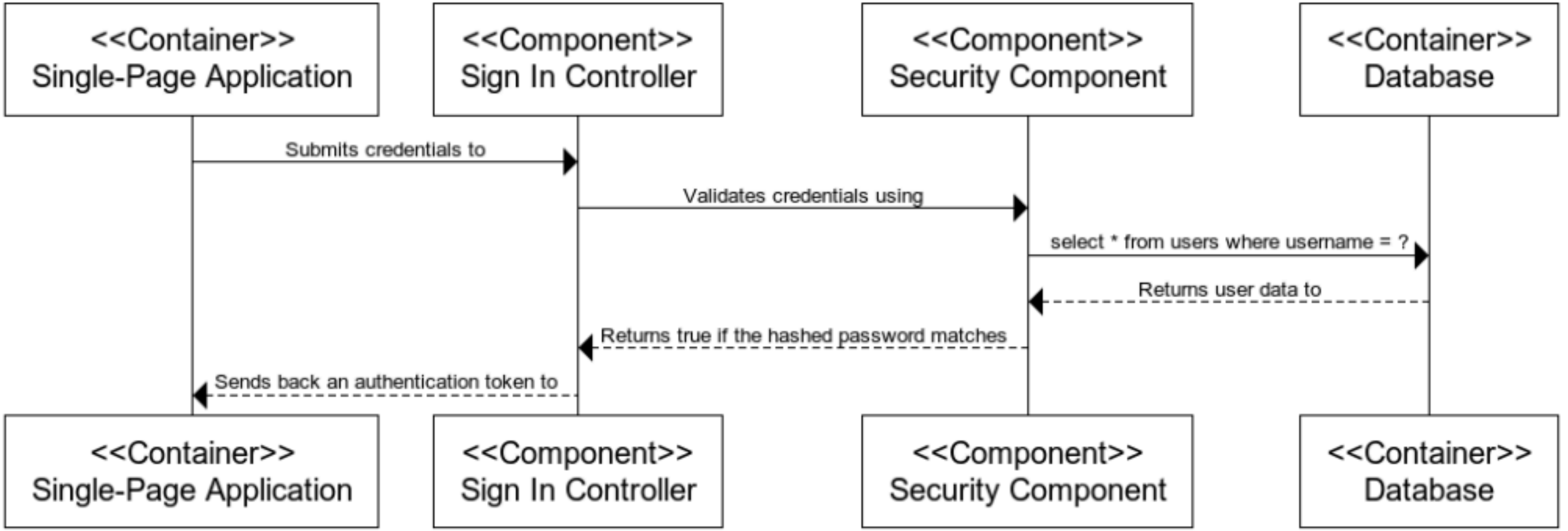
[System Landscape] Big Bank plc

Monday, 31 January 2022 at 08:56 Greenwich Mean Time

Runtime/behavioural diagrams

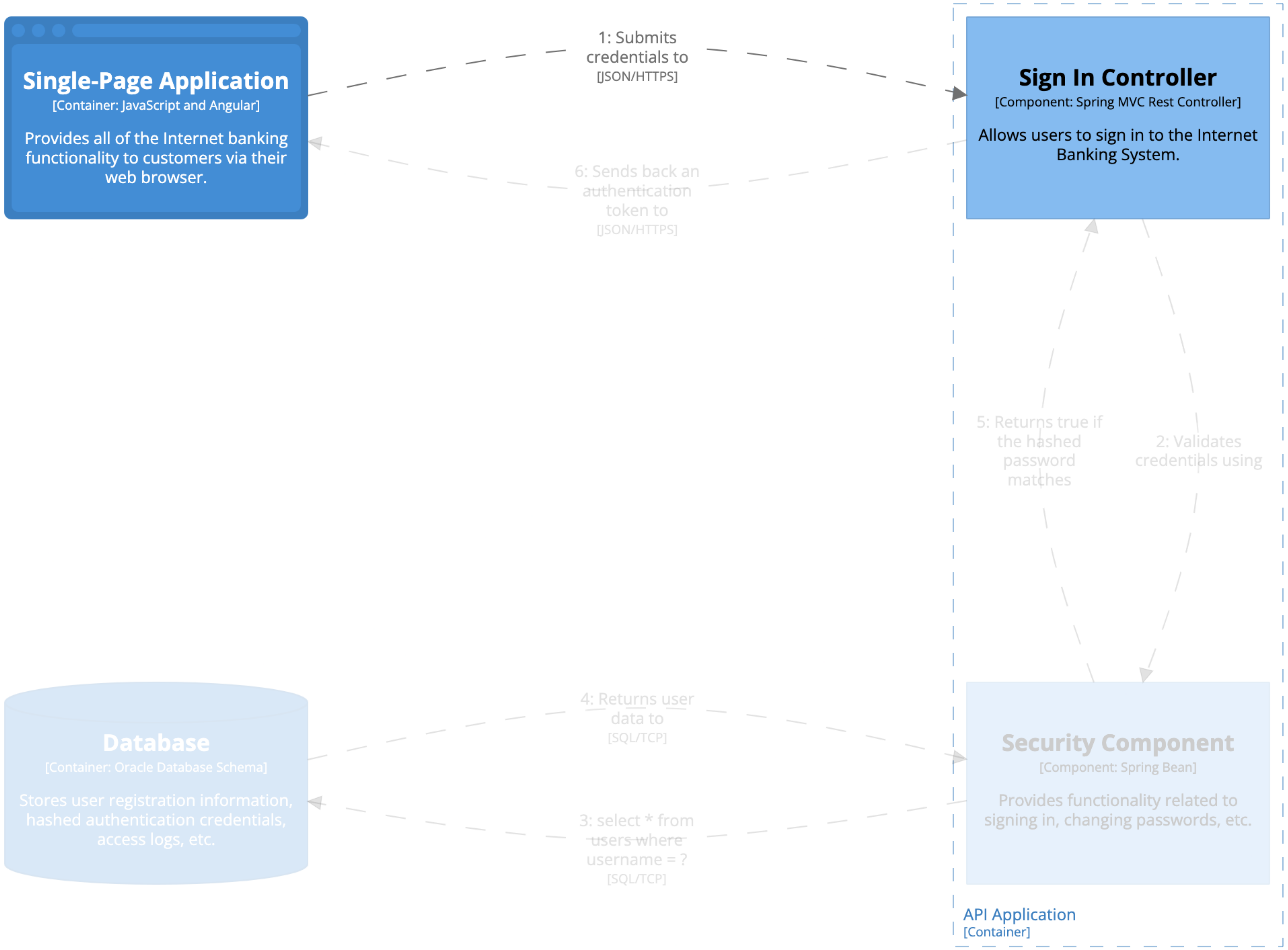
Static structure diagrams are very useful, but they don’t tell the whole story

API Application - Dynamic - SignIn



<<Container>> Single-Page Application
<<Component>> Sign In Controller
<<Component>> Security Component
<<Container>> Database
Submits credentials to
Validates credentials using
select * from users where username = ?
Returns user data to
Returns true if the hashed password matches
Sends back an authentication token to
<<Container>> Single-Page Application
<<Component>> Sign In Controller
<<Component>> Security Component
<<Container>> Database

www.websequencediagrams.com



Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

1: Submits credentials to
[JSON/HTTPS]

2: Validates credentials using

3: select * from users where username = ?
[SQL/TCP]

4: Returns user data to
[SQL/TCP]

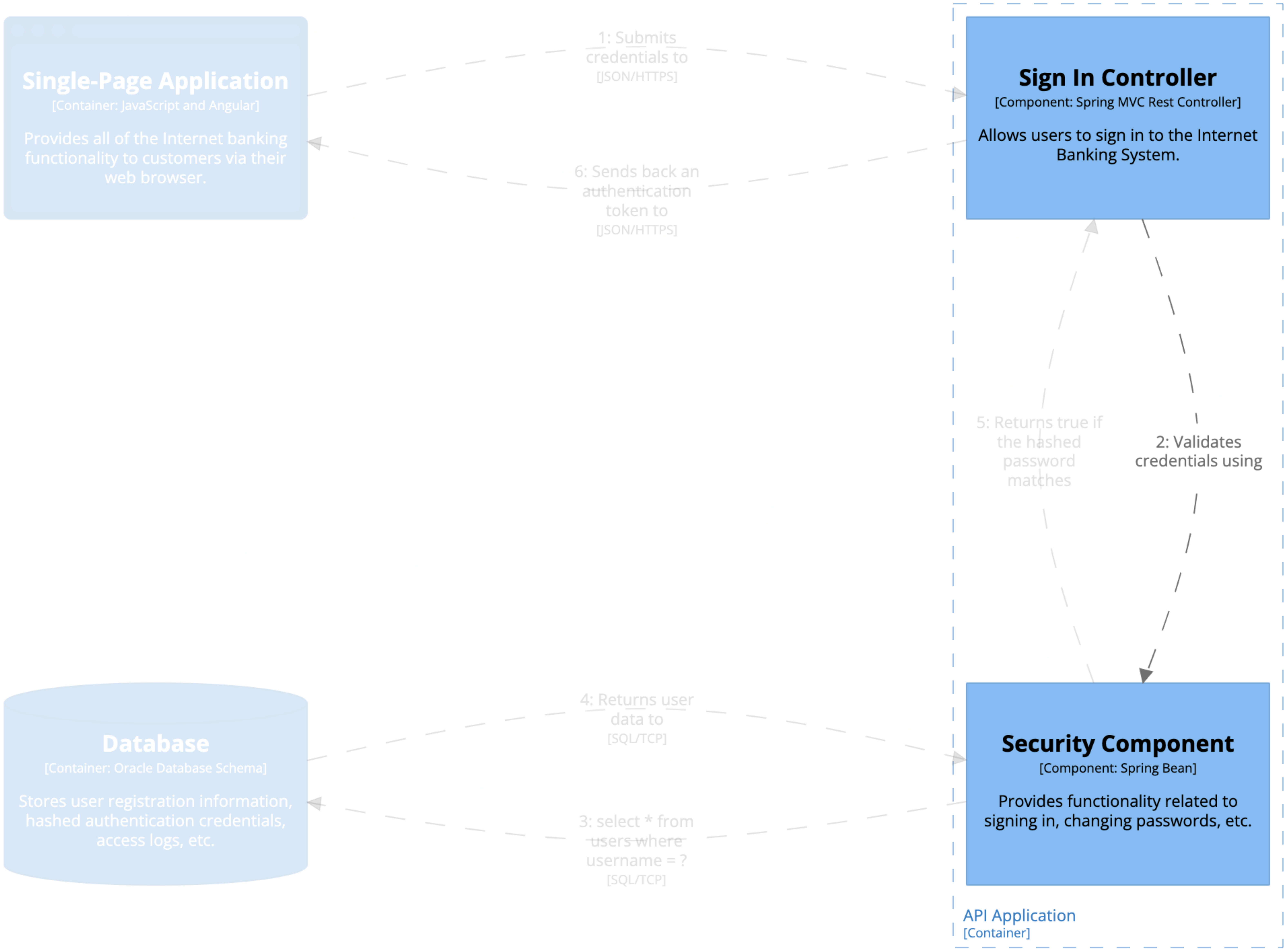
5: Returns true if the hashed password matches

6: Sends back an authentication token to
[JSON/HTTPS]

API Application
[Container]

[Dynamic] Internet Banking System - API Application

Summarises how the sign in feature works in the single-page application.
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

1: Submits credentials to
[JSON/HTTPS]

6: Sends back an authentication token to
[JSON/HTTPS]

Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.

5: Returns true if the hashed password matches

2: Validates credentials using

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

4: Returns user data to
[SQL/TCP]

3: select * from users where username = ?
[SQL/TCP]

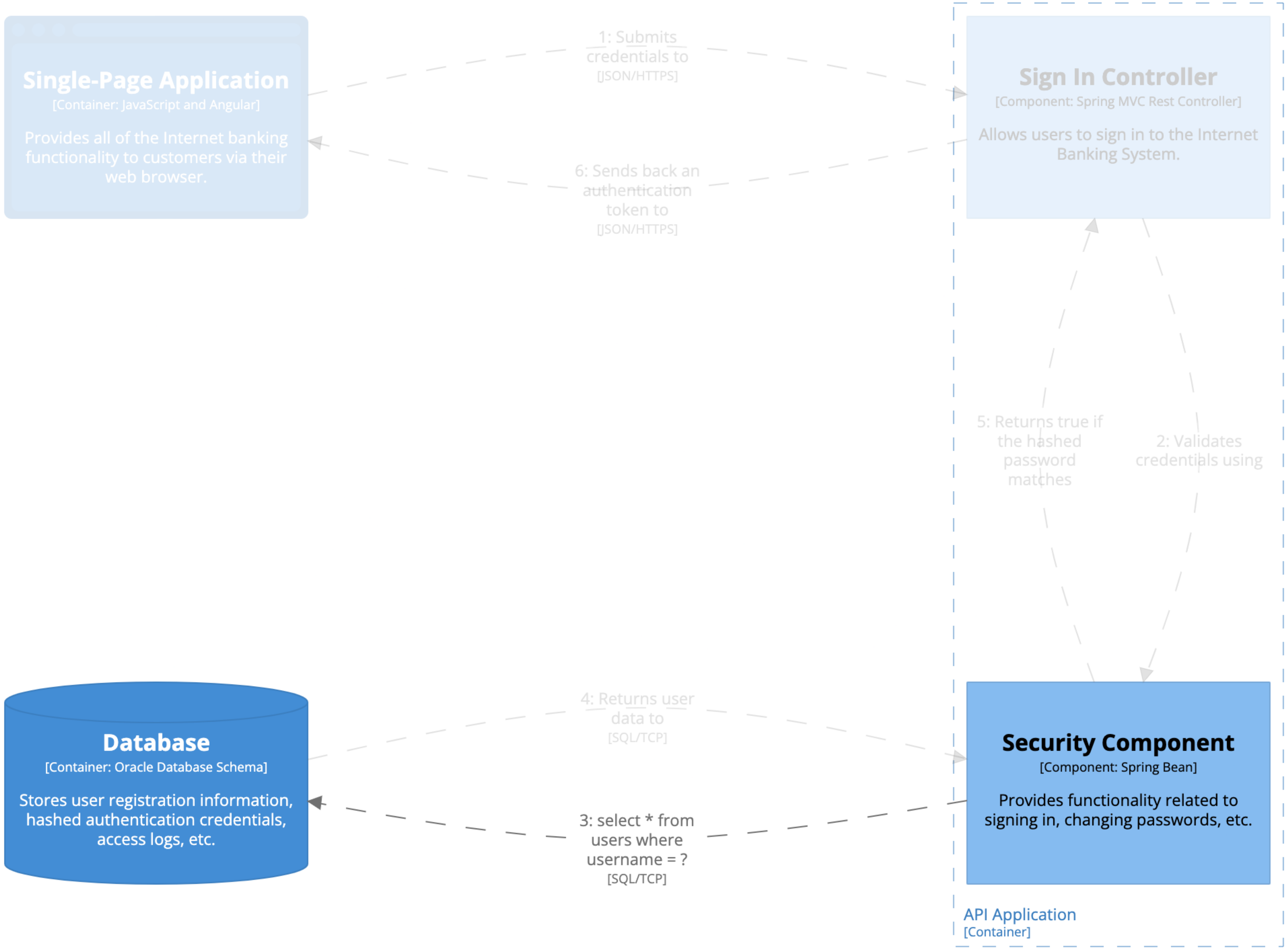
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

API Application
[Container]

[Dynamic] Internet Banking System - API Application

Summarises how the sign in feature works in the single-page application.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

1: Submits credentials to
[JSON/HTTPS]

6: Sends back an authentication token to
[JSON/HTTPS]

Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.

5: Returns true if the hashed password matches

2: Validates credentials using

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

4: Returns user data to
[SQL/TCP]

3: select * from users where username = ?
[SQL/TCP]

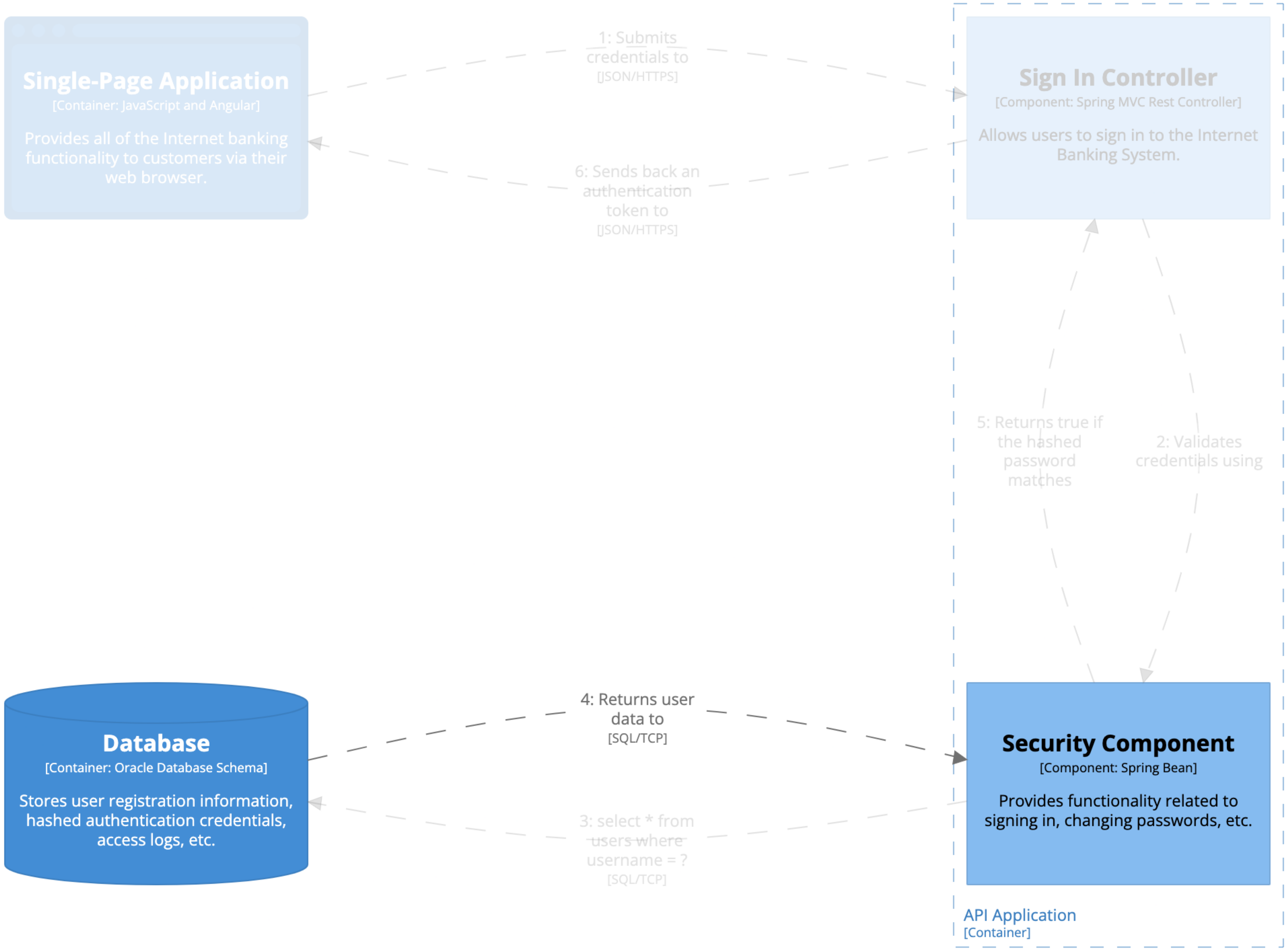
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

API Application
[Container]

[Dynamic] Internet Banking System - API Application

Summarises how the sign in feature works in the single-page application.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

1: Submits credentials to
[JSON/HTTPS]

6: Sends back an authentication token to
[JSON/HTTPS]

Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.

5: Returns true if the hashed password matches

2: Validates credentials using

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

4: Returns user data to
[SQL/TCP]

3: select * from users where username = ?
[SQL/TCP]

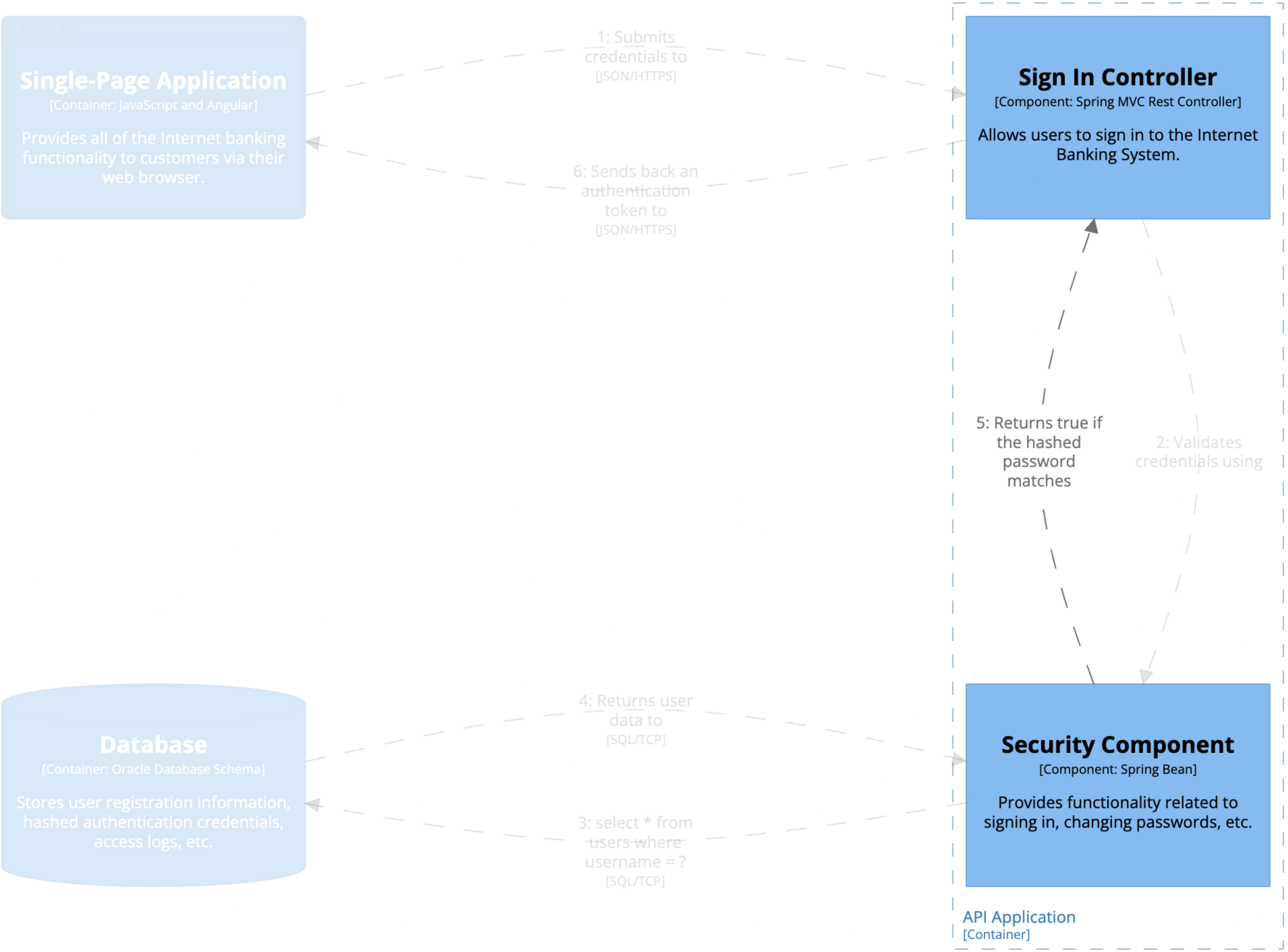
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

API Application
[Container]

[Dynamic] Internet Banking System - API Application

Summarises how the sign in feature works in the single-page application.

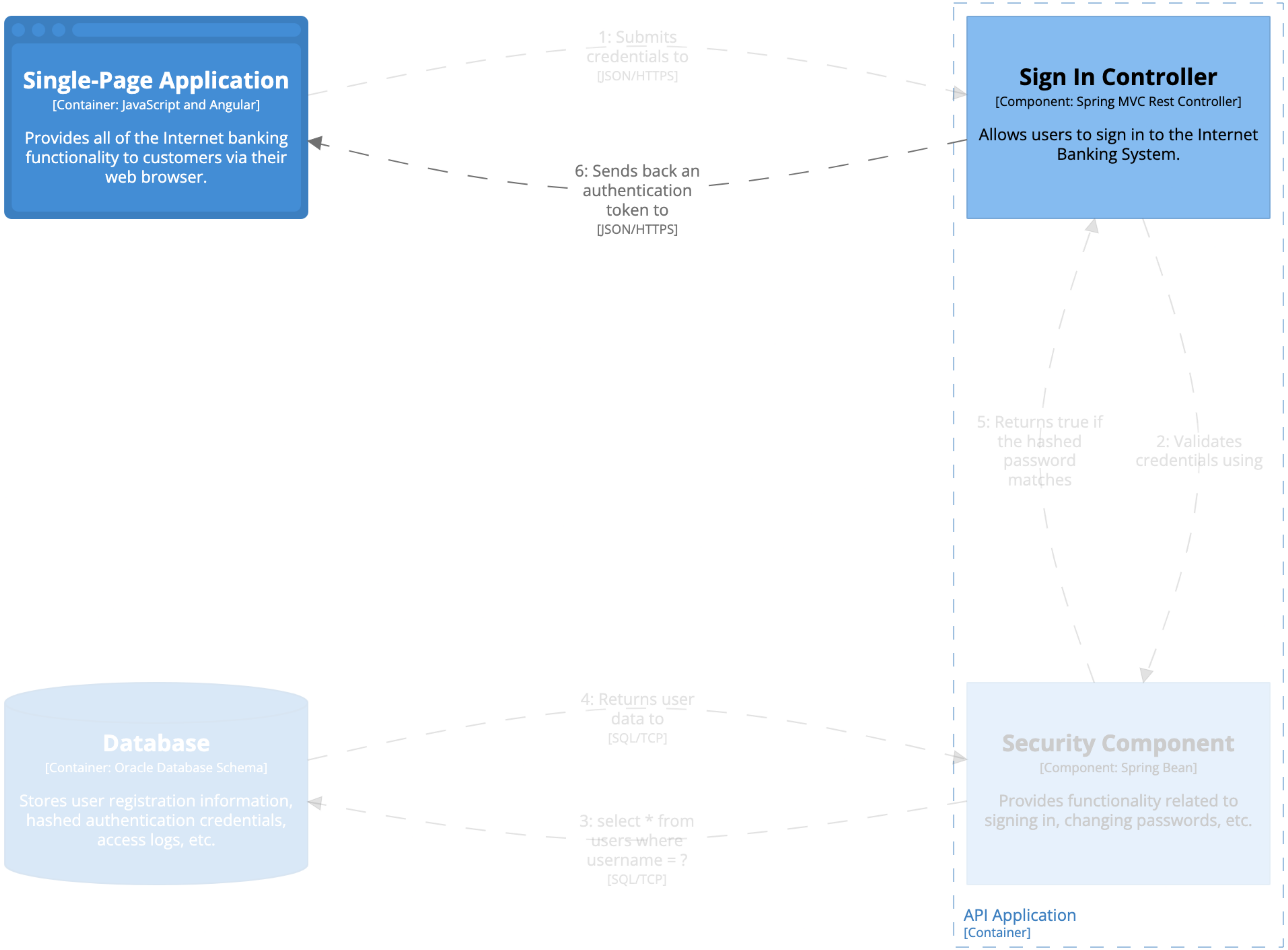
Monday, 27 February 2023 at 15:36 Greenwich Mean Time


Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.
1: Submits credentials to
[JSON/HTTPS]
6: Sends back an authentication token to
[JSON/HTTPS]
Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.
5: Returns true if the hashed password matches
2: Validates credentials using
Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
4: Returns user data to
[SQL/TCP]
3: select * from users where username = ?
[SQL/TCP]
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
API Application
[Container]

[Dynamic] Internet Banking System - API Application

Summarises how the sign in feature works in the single-page application.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

1: Submits credentials to
[JSON/HTTPS]

6: Sends back an authentication token to
[JSON/HTTPS]

Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.

5: Returns true if the hashed password matches

2: Validates credentials using

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

4: Returns user data to
[SQL/TCP]

3: select * from users where username = ?
[SQL/TCP]

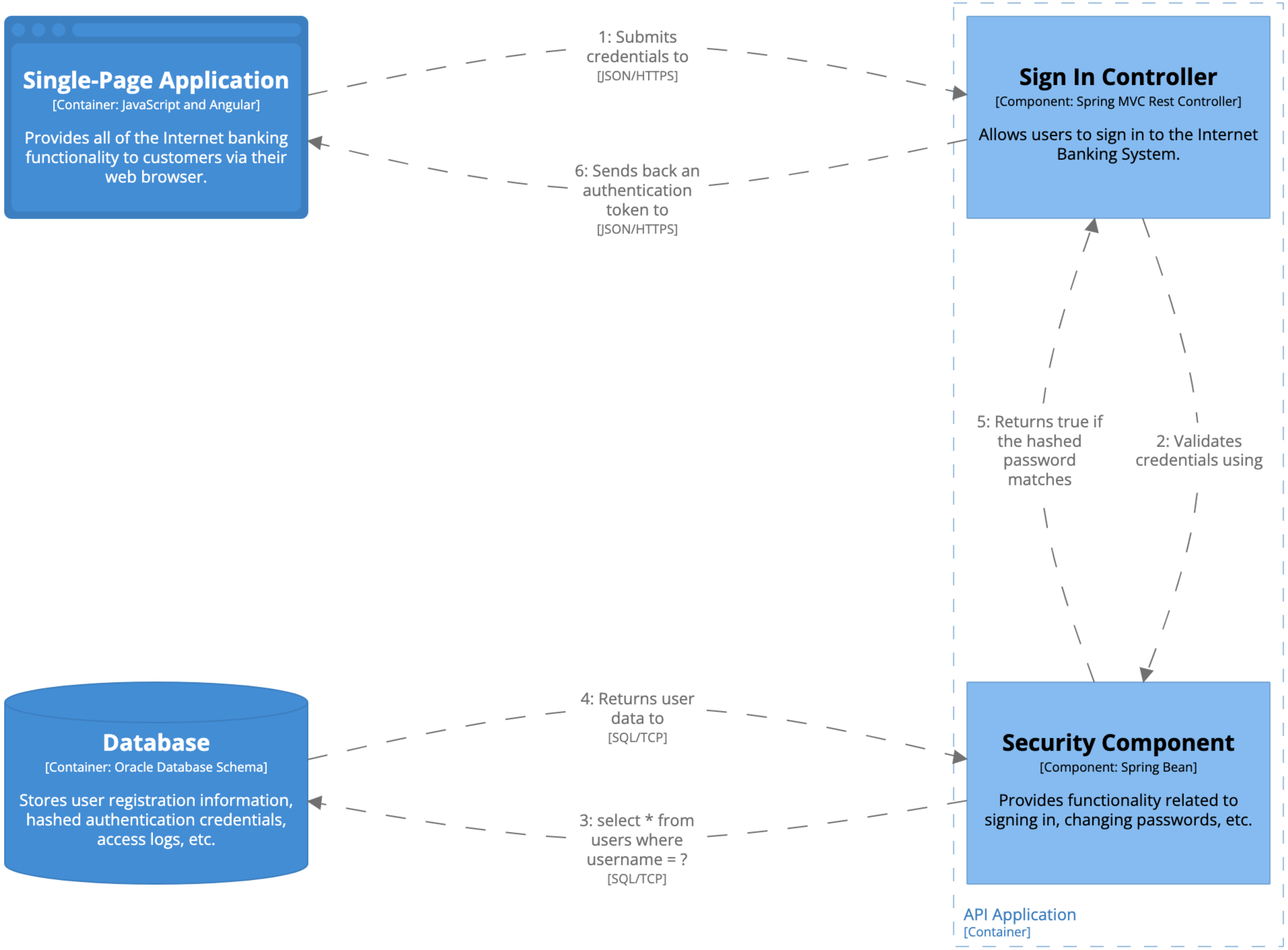
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.

API Application
[Container]

[Dynamic] Internet Banking System - API Application

Summarises how the sign in feature works in the single-page application.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time



Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.
1: Submits credentials to
[JSON/HTTPS]
6: Sends back an authentication token to
[JSON/HTTPS]
Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.
5: Returns true if the hashed password matches
2: Validates credentials using
Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
4: Returns user data to
[SQL/TCP]
3: select * from users where username = ?
[SQL/TCP]
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
API Application
[Container]

[Dynamic] Internet Banking System - API Application

Summarises how the sign in feature works in the single-page application.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time

Use dynamic diagrams to describe **patterns** or **complex interactions**

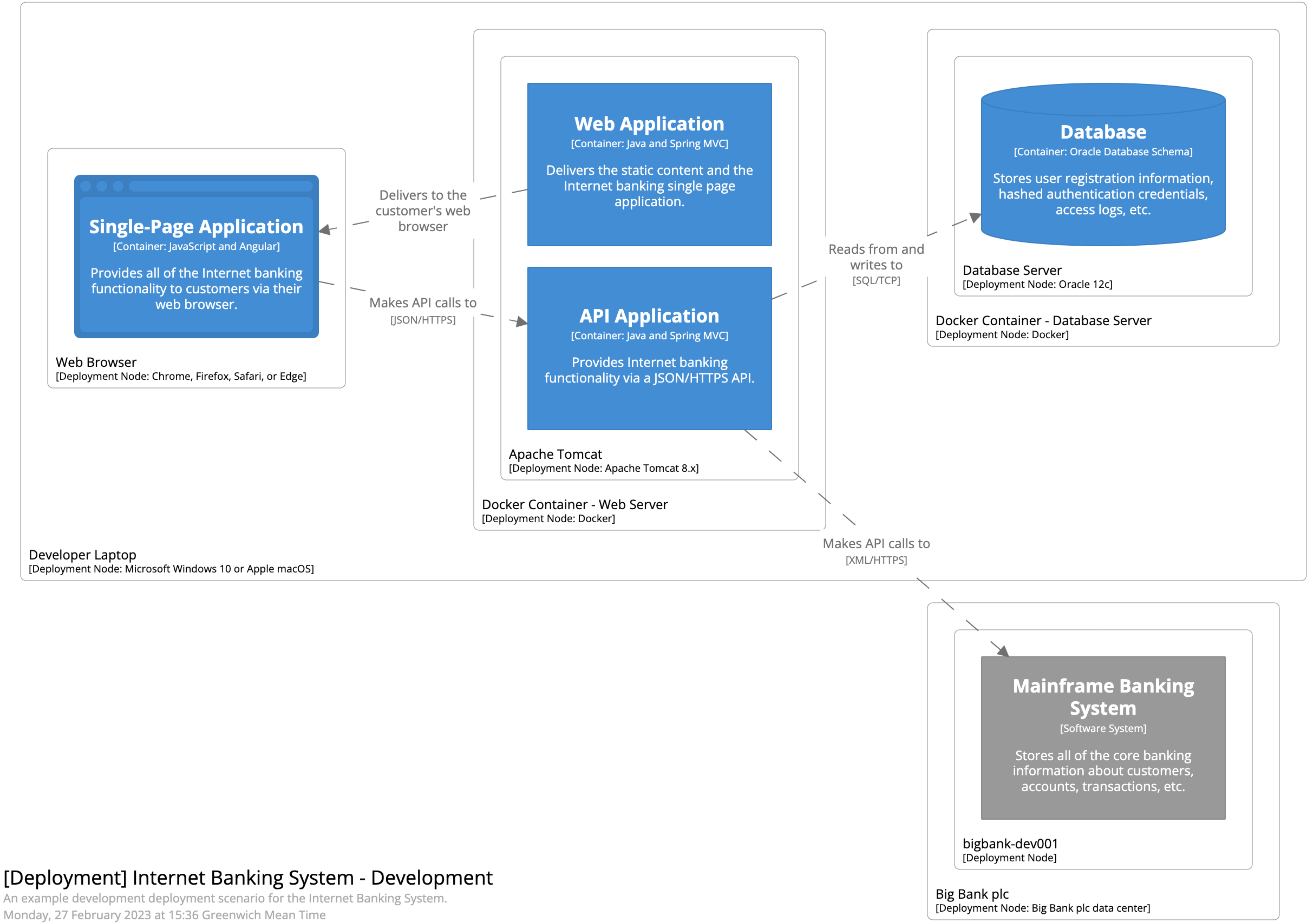
Deployment diagrams

Deployment is about the mapping of containers to infrastructure

Deployment Node

Physical infrastructure (a physical server or device), virtualised infrastructure (IaaS, PaaS, a virtual machine), containerised infrastructure (a Docker container), database server, Java EE web/application server, Microsoft IIS, etc

A deployment node can contain other **deployment nodes** or software system/container **instances**



Developer Laptop
[Deployment Node: Microsoft Windows 10 or Apple macOS]

Web Browser
[Deployment Node: Chrome, Firefox, Safari, or Edge]

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Docker Container - Web Server
[Deployment Node: Docker]

Apache Tomcat
[Deployment Node: Apache Tomcat 8.x]

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

Docker Container - Database Server
[Deployment Node: Docker]

Database Server
[Deployment Node: Oracle 12c]

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

Big Bank plc
[Deployment Node: Big Bank plc data center]

bigbank-dev001
[Deployment Node]

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Delivers to the customer's web browser

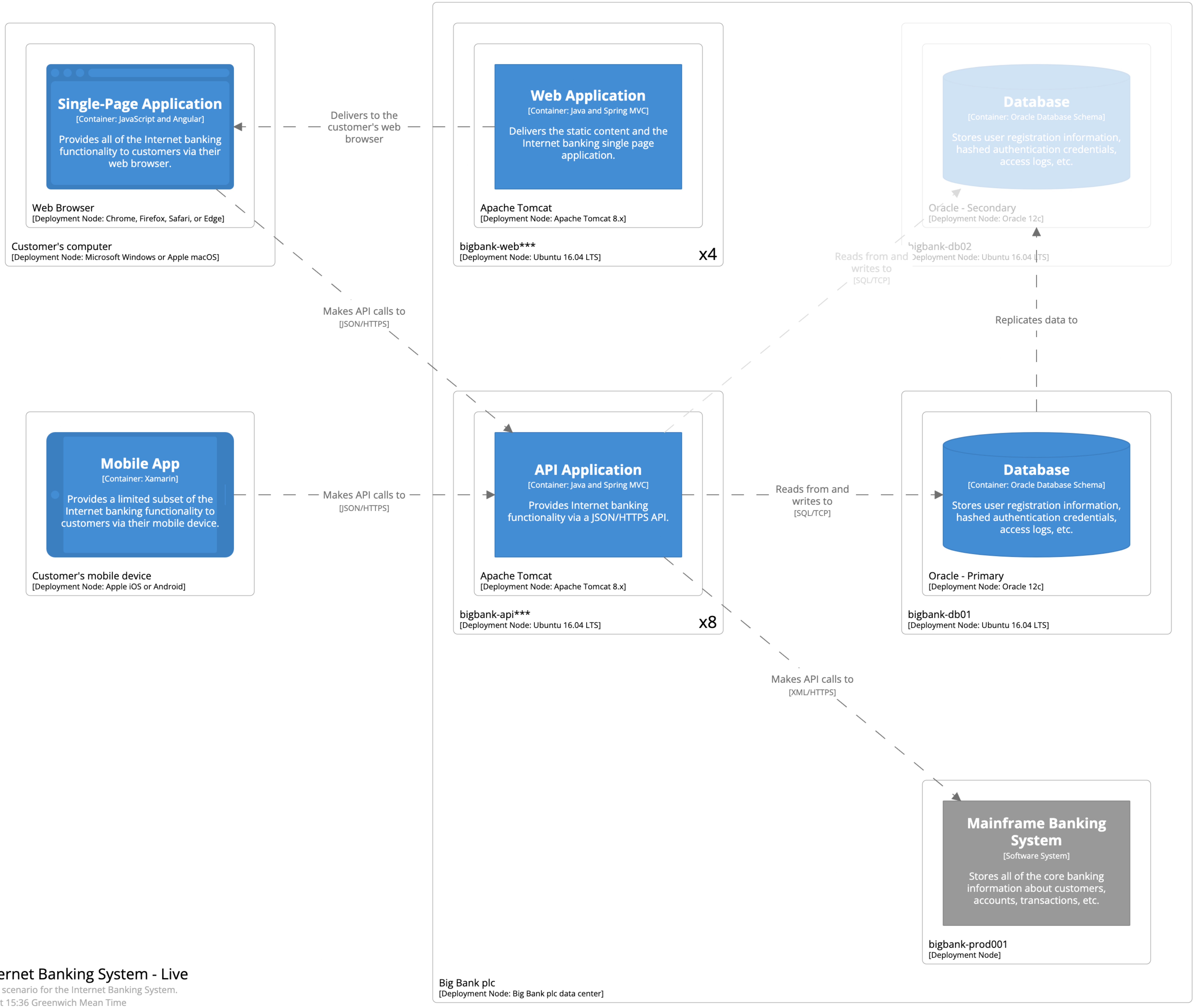
Makes API calls to
[JSON/HTTPS]

Reads from and writes to
[SQL/TCP]

Makes API calls to
[XML/HTTPS]

[Deployment] Internet Banking System - Development

An example development deployment scenario for the Internet Banking System.
Monday, 27 February 2023 at 15:36 Greenwich Mean Time


Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.
Web Browser
[Deployment Node: Chrome, Firefox, Safari, or Edge]
Customer's computer
[Deployment Node: Microsoft Windows or Apple macOS]
Delivers to the customer's web browser
Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.
Apache Tomcat
[Deployment Node: Apache Tomcat 8.x]
bigbank-web***
[Deployment Node: Ubuntu 16.04 LTS]
x4
Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Oracle - Secondary
[Deployment Node: Oracle 12c]
bigbank-db02
[Deployment Node: Ubuntu 16.04 LTS]
Reads from and writes to
[SQL/TCP]
Replicates data to
Makes API calls to
[JSON/HTTPS]
Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.
Customer's mobile device
[Deployment Node: Apple iOS or Android]
Makes API calls to
[JSON/HTTPS]
API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.
Apache Tomcat
[Deployment Node: Apache Tomcat 8.x]
bigbank-api***
[Deployment Node: Ubuntu 16.04 LTS]
x8
Reads from and writes to
[SQL/TCP]
Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Oracle - Primary
[Deployment Node: Oracle 12c]
bigbank-db01
[Deployment Node: Ubuntu 16.04 LTS]
Makes API calls to
[XML/HTTPS]
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
bigbank-prod001
[Deployment Node]
Big Bank plc
[Deployment Node: Big Bank plc data center]

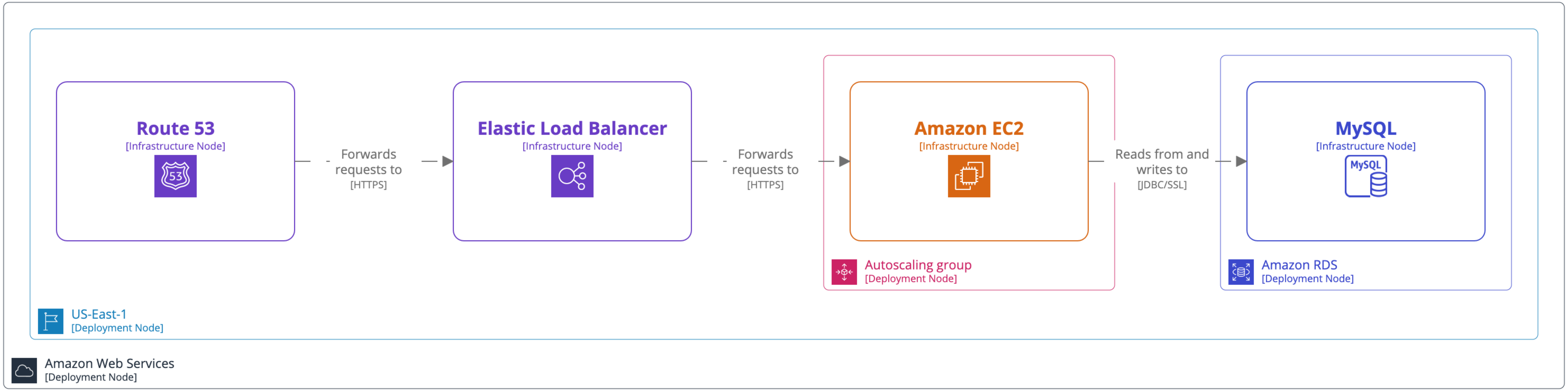
[Deployment] Internet Banking System - Live

An example live deployment scenario for the Internet Banking System.

Monday, 27 February 2023 at 15:36 Greenwich Mean Time

Infrastructure Node

Routers, firewalls, load balancers,
DNS providers, edge caches, etc



Route 53
[Infrastructure Node]

Forwards requests to
[HTTPS]

Elastic Load Balancer
[Infrastructure Node]

Forwards requests to
[HTTPS]

Amazon EC2
[Infrastructure Node]

Autoscaling group
[Deployment Node]

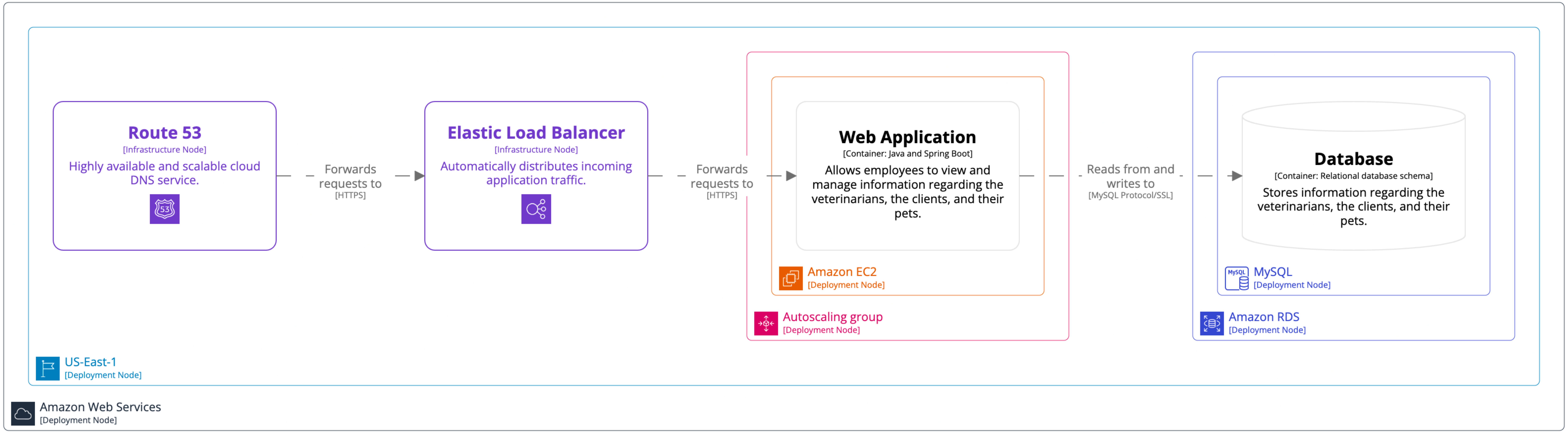
Reads from and writes to
[JDBC/SSL]

MySQL
[Infrastructure Node]

Amazon RDS
[Deployment Node]

US-East-1
[Deployment Node]

Amazon Web Services
[Deployment Node]



Route 53
[Infrastructure Node]
Highly available and scalable cloud DNS service.

Forwards requests to
[HTTPS]

Elastic Load Balancer
[Infrastructure Node]
Automatically distributes incoming application traffic.

Forwards requests to
[HTTPS]

Web Application
[Container: Java and Spring Boot]
Allows employees to view and manage information regarding the veterinarians, the clients, and their pets.

Reads from and writes to
[MySQL Protocol/SSL]

Database
[Container: Relational database schema]
Stores information regarding the veterinarians, the clients, and their pets.

Amazon EC2
[Deployment Node]

Autoscaling group
[Deployment Node]

MySQL
[Deployment Node]

Amazon RDS
[Deployment Node]

US-East-1
[Deployment Node]

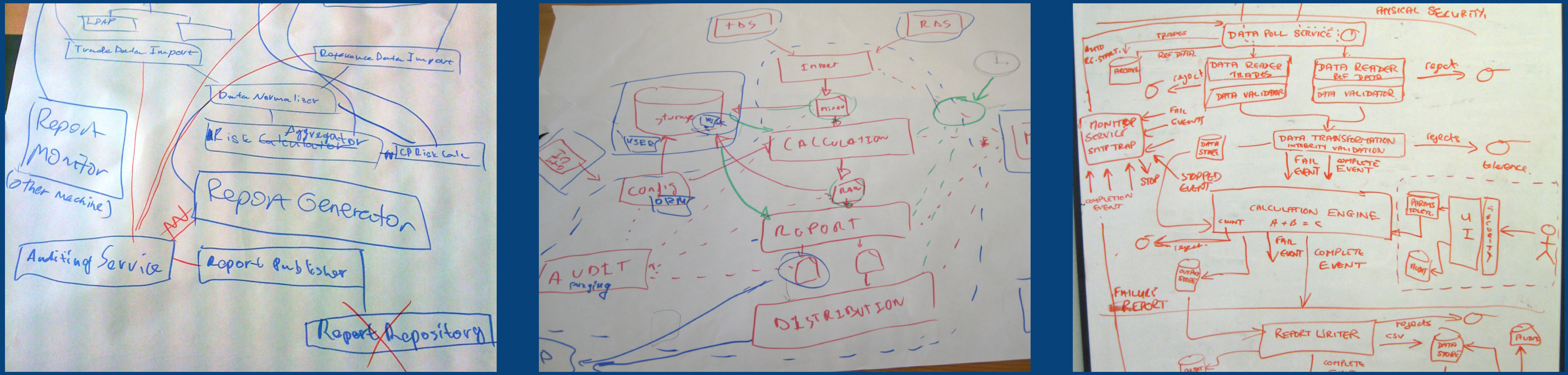
Amazon Web Services
[Deployment Node]

FAQ

Part 1

> C4 has been around over a decade - if it was truly useful, it would have replaced UML in most teams

C4 wasn't designed to replace UML



C4 was designed to bring structure to the typical ad hoc "boxes and arrows" diagrams teams typically create because they are no longer using UML

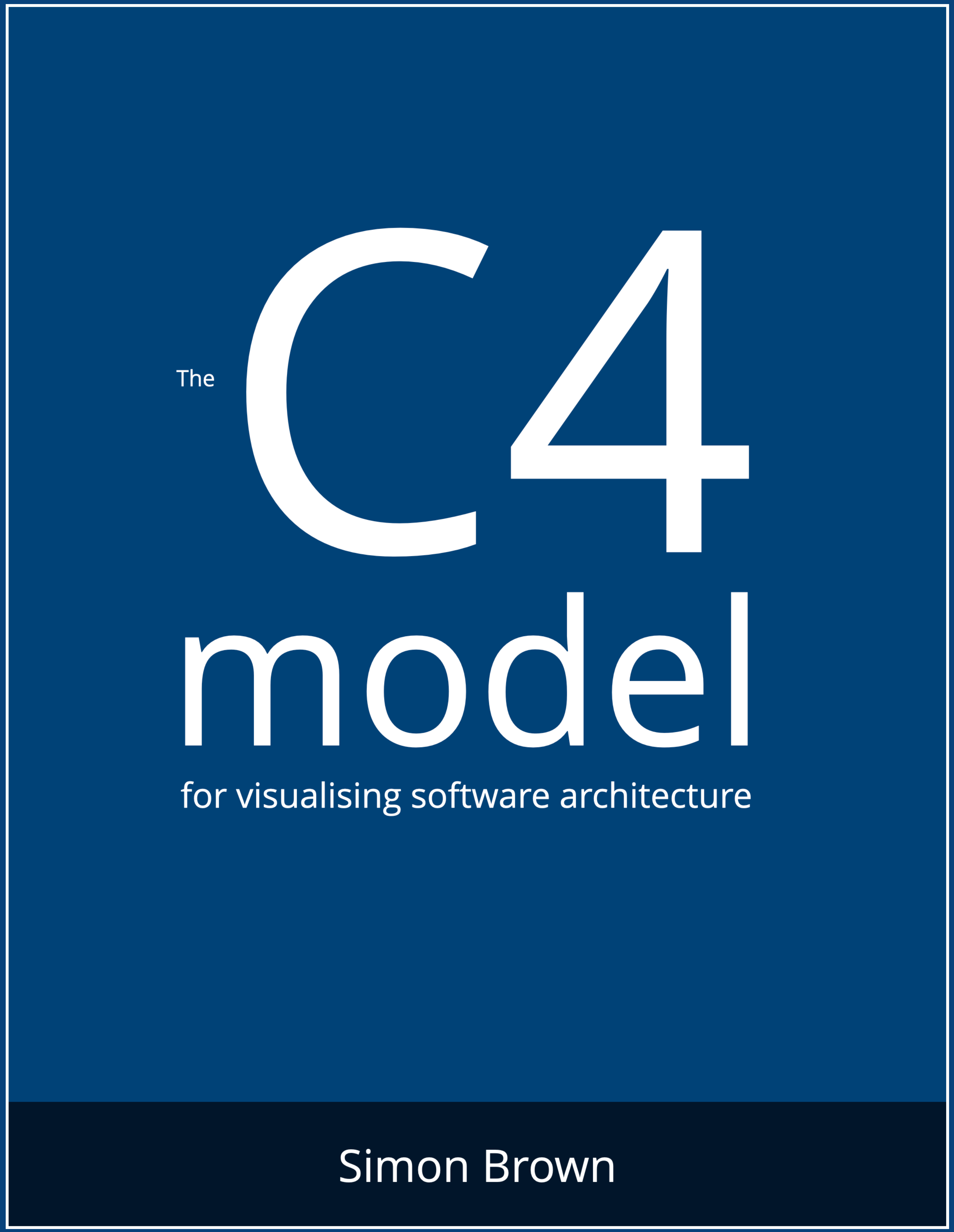
I've seen more interest than ever in C4 over the past few years; many organisations have adopted it as their preferred approach for software architecture diagramming

I’ve run software architecture
workshops
in **30+ countries**
for **10,000+ people**
across most industry sectors

Academic establishments

A free subscription is available for students and staff at academic establishments, **for teaching purposes** (e.g. preparation of teaching material, use in assignments, etc). It's based upon the regular cloud service subscription with 5 workspaces, and is granted automatically to users who sign up with an e-mail address from the following 80 academic establishments:

- Facultad de Ingeniería de la Universidad de Buenos Aires, Argentina (`@fi.uba.ar`)
- Universidad Tecnológica Nacional, Argentina (`@ca.frre.utn.ed.ar`, `@alu.frt.utn.edu.ar`, `@frt.utn.edu.ar`, `@doc.frt.utn.edu.ar`)
- University of Queensland, Australia (`@uq.edu.au`, `@uq.net.au`, `@student.uq.edu.au`)
- University of Tasmania, Australia (`@utas.edu.au`)
- Howest University of Applied Sciences, Belgium (`@howest.be`, `@student.howest.be`)
- PXL University of Applied Sciences and Arts, Belgium (`@pxl.be`, `@student.pxl.be`)
- Universidade Federal do Pará, Brazil (`@ig.ufpa.br`, `@icen.ufpa.br`)
- Universidade federal de Pernambuco, Brazil (`@ufpe.br`, `@cin.ufpe.br`)
- Université de Sherbrooke, Canada (`@usherbrooke.ca`)
- École de Technologie Supérieure, Canada (`@etsmtl.ca`, `@ens.etsmtl.ca`)
- Duoc UC, Chile (`@duoc.cl`, `@alumnos.duoc.cl`)
- Universidad de Chile, Chile (`@dcc.uchile.cl`)



The C4 model

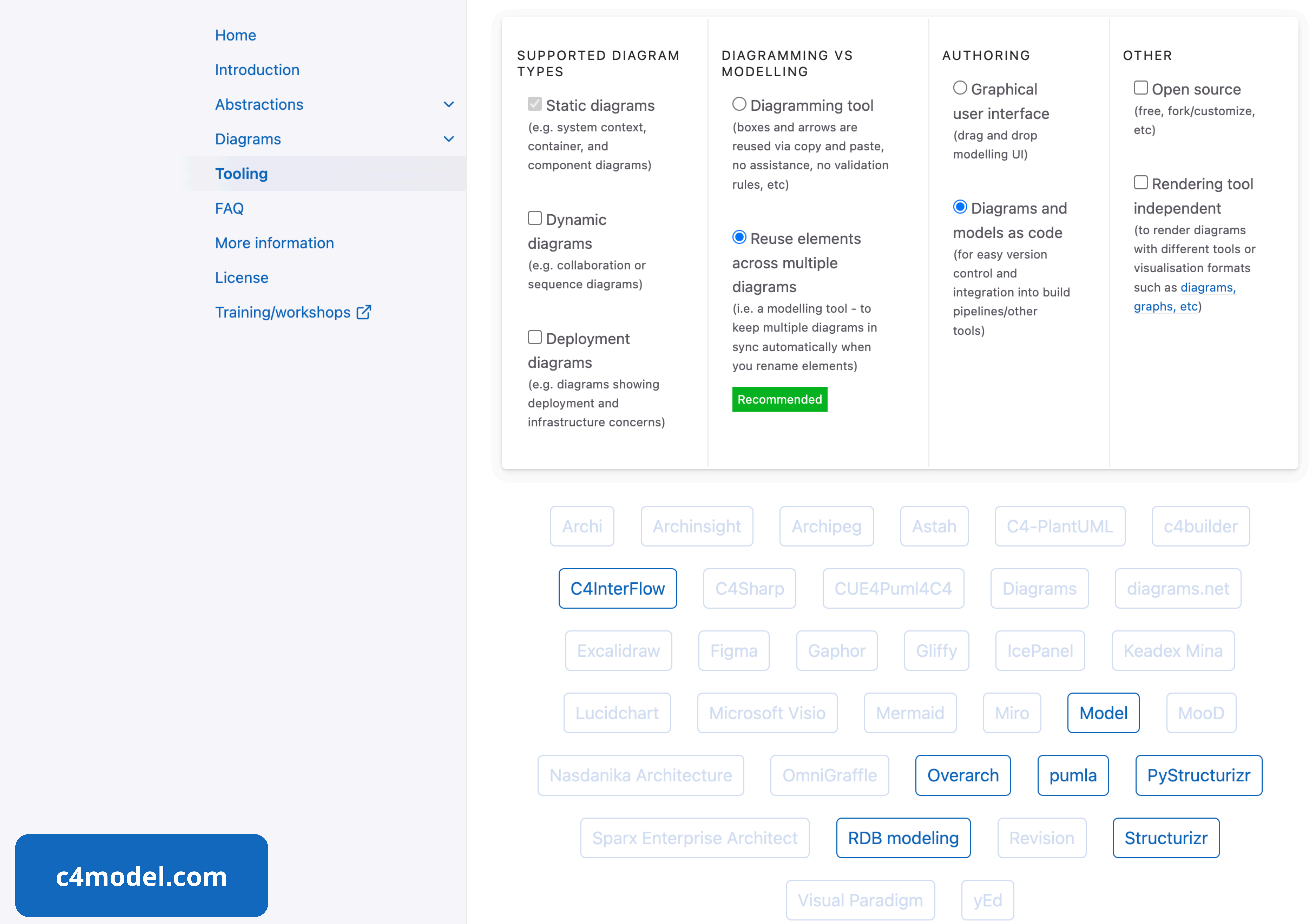
for visualising software architecture

Simon Brown

My C4 model book is also used as course material in many other universities

Tooling?

What tooling do you recommend for long-lived diagrams?



- Home
- Introduction
- Abstractions
- Diagrams
- **Tooling**
- FAQ
- More information
- License
- Training/workshops

SUPPORTED DIAGRAM TYPES

☑ Static diagrams
(e.g. system context, container, and component diagrams)

□ Dynamic diagrams
(e.g. collaboration or sequence diagrams)

□ Deployment diagrams
(e.g. diagrams showing deployment and infrastructure concerns)

DIAGRAMMING VS MODELLING

○ Diagramming tool
(boxes and arrows are reused via copy and paste, no assistance, no validation rules, etc)

● Reuse elements across multiple diagrams
(i.e. a modelling tool - to keep multiple diagrams in sync automatically when you rename elements)

Recommended

AUTHORING

○ Graphical user interface
(drag and drop modelling UI)

● Diagrams and models as code
(for easy version control and integration into build pipelines/other tools)

OTHER

□ Open source
(free, fork/customize, etc)

□ Rendering tool independent
(to render diagrams with different tools or visualisation formats such as diagrams, graphs, etc)

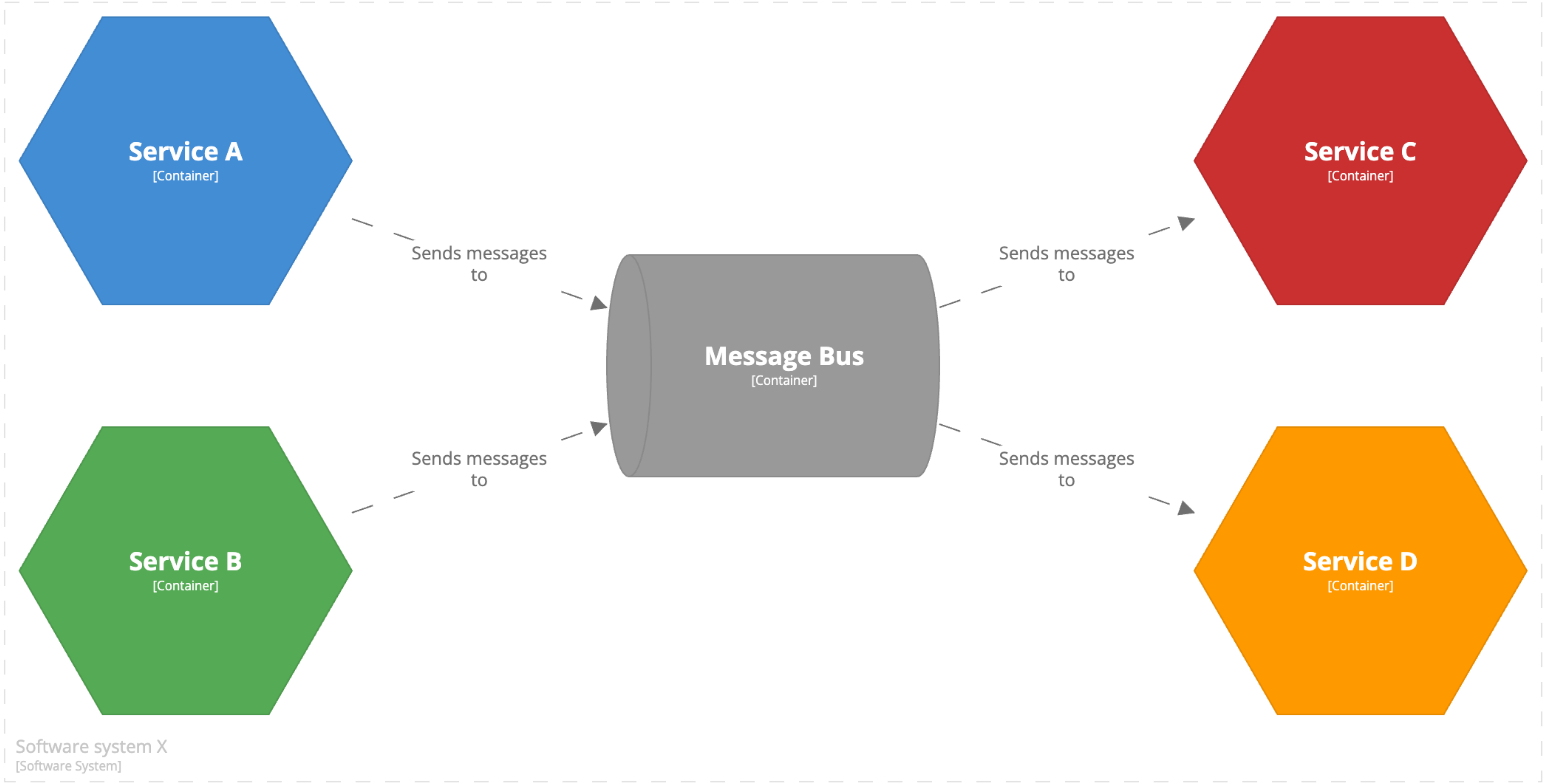
Archi | Archinsight | Archipeg | Astah | C4-PlantUML | c4builder
C4InterFlow | C4Sharp | CUE4PumI4C4 | Diagrams | diagrams.net
Excalidraw | Figma | Gaphor | Gliffy | IcePanel | Keadex Mina
Lucidchart | Microsoft Visio | Mermaid | Miro | Model | MooD
Nasdanika Architecture | OmniGraffle | Overarch | pumla | PyStructurizr
Sparx Enterprise Architect | RDB modeling | Revision | Structurizr
Visual Paradigm | yEd

c4model.com

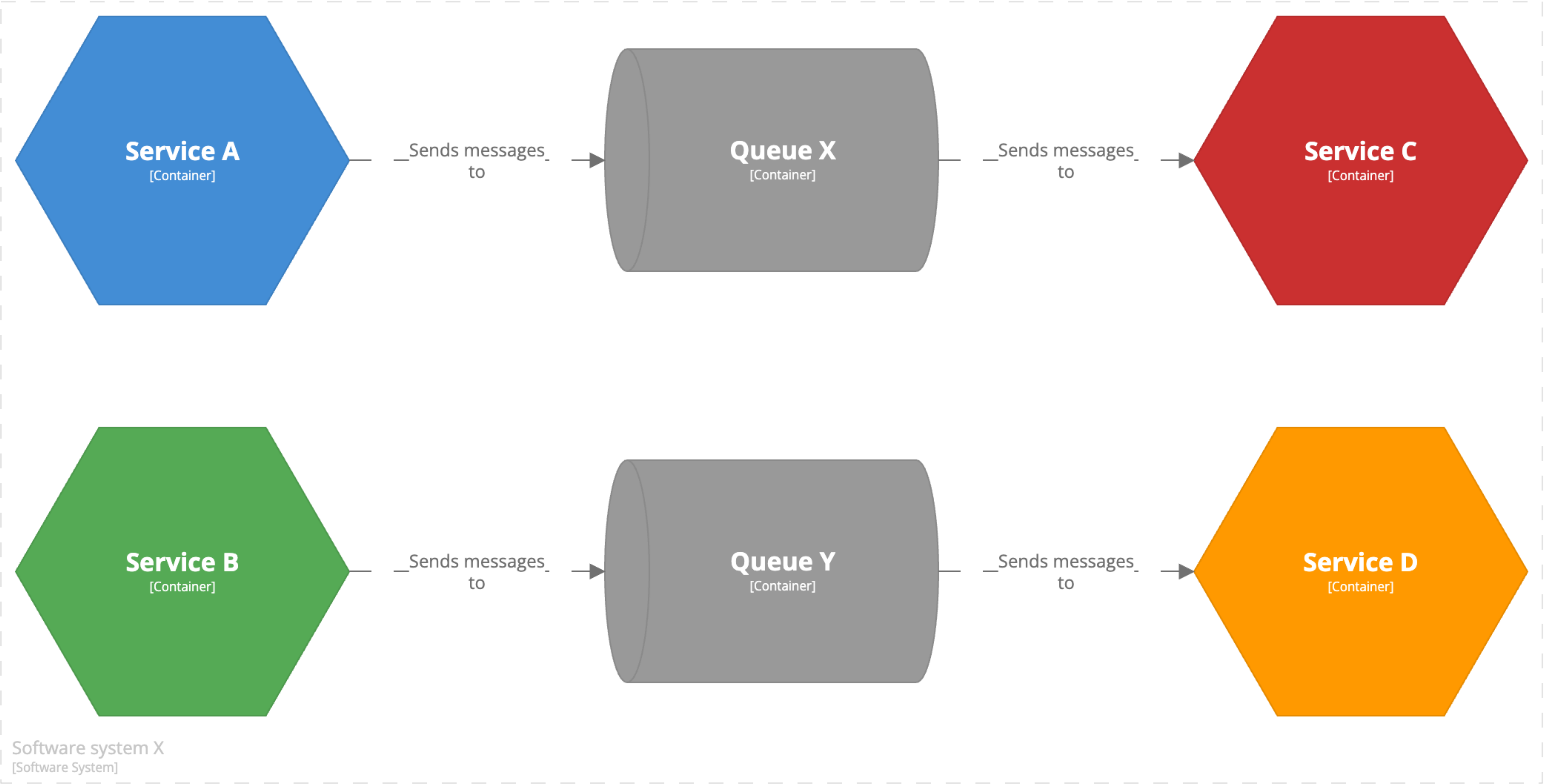
FAQ

Part 2

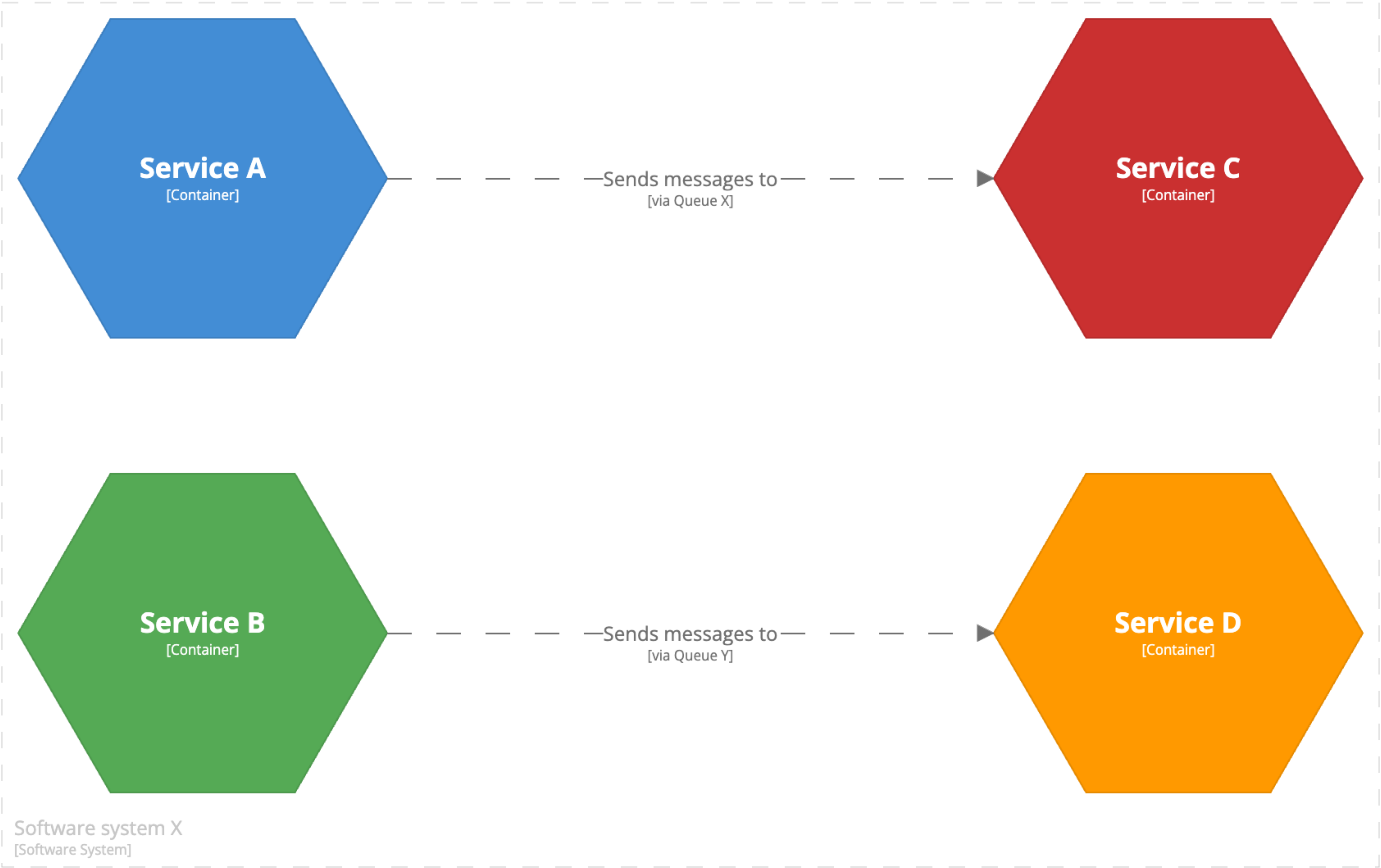
Message-driven architectures



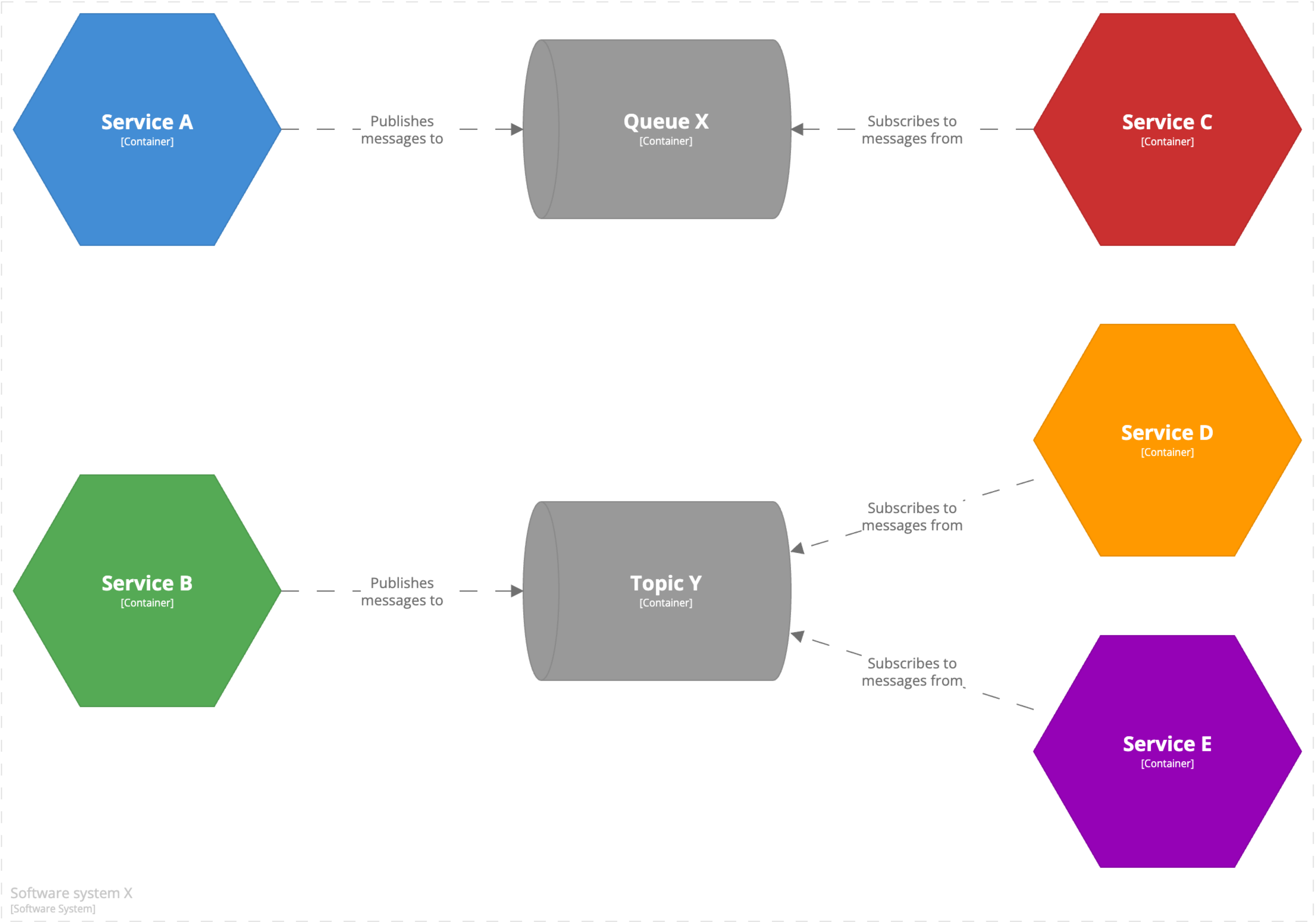
[Container] Software system X



[Container] Software system X

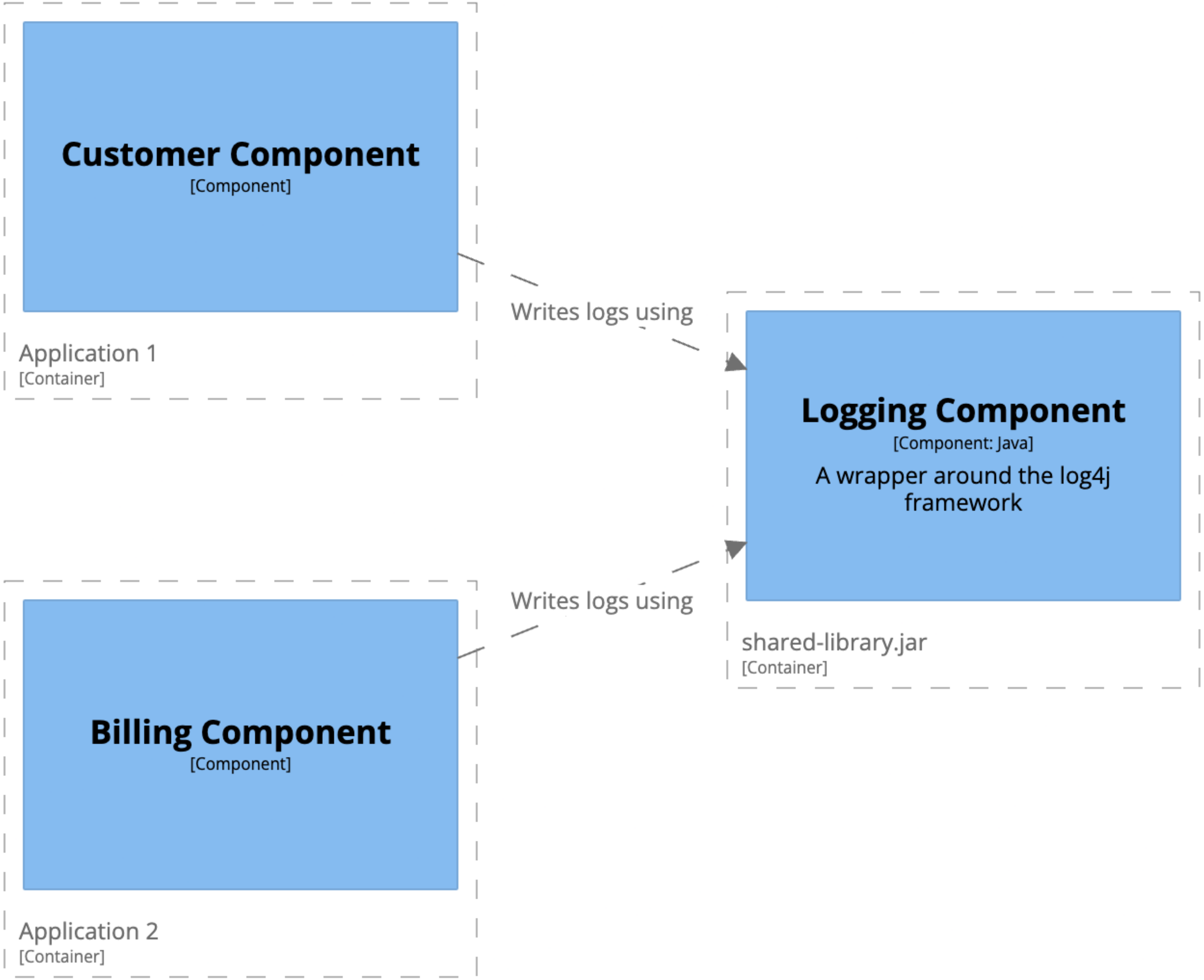


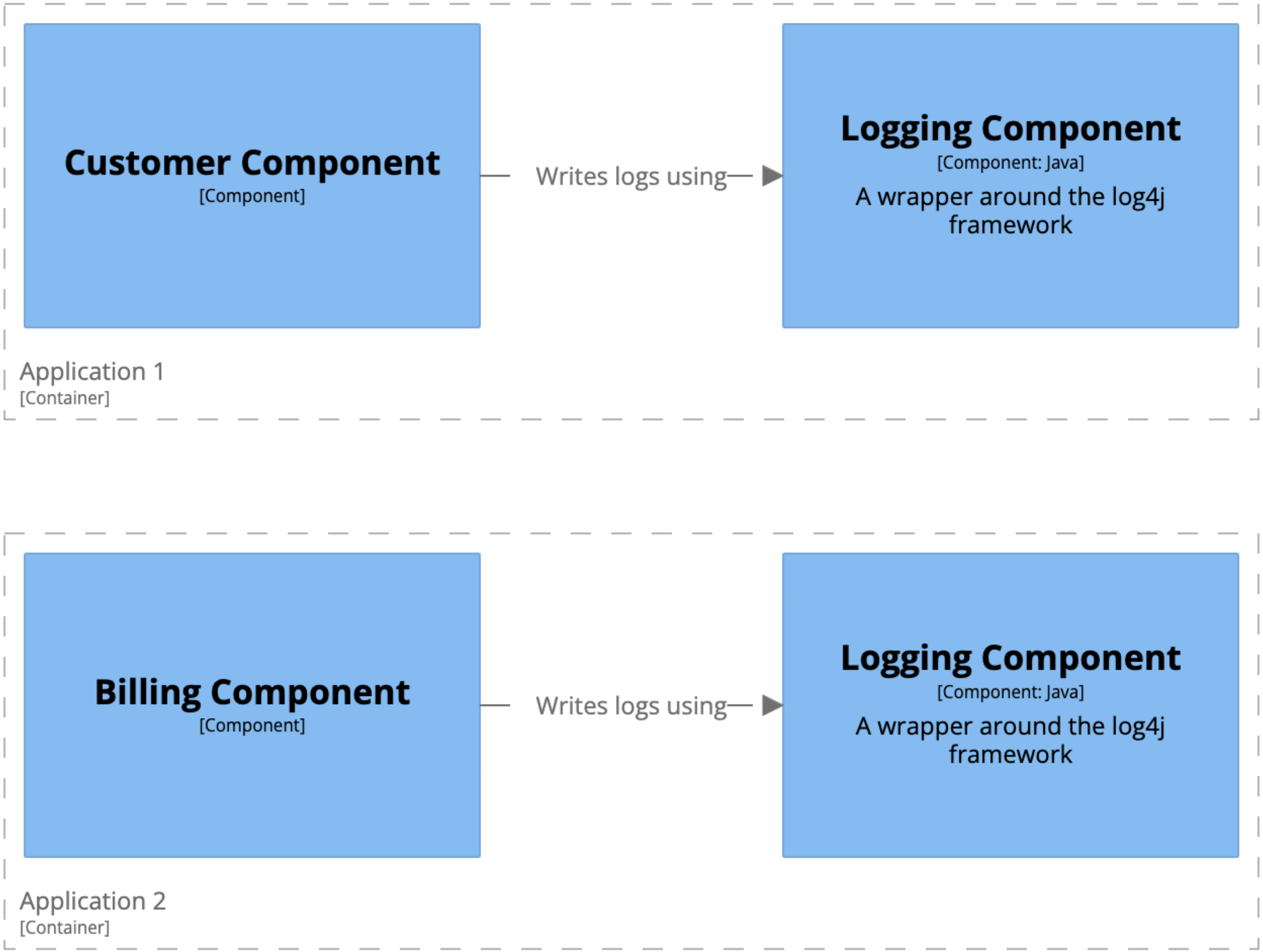
[Container] Software system X

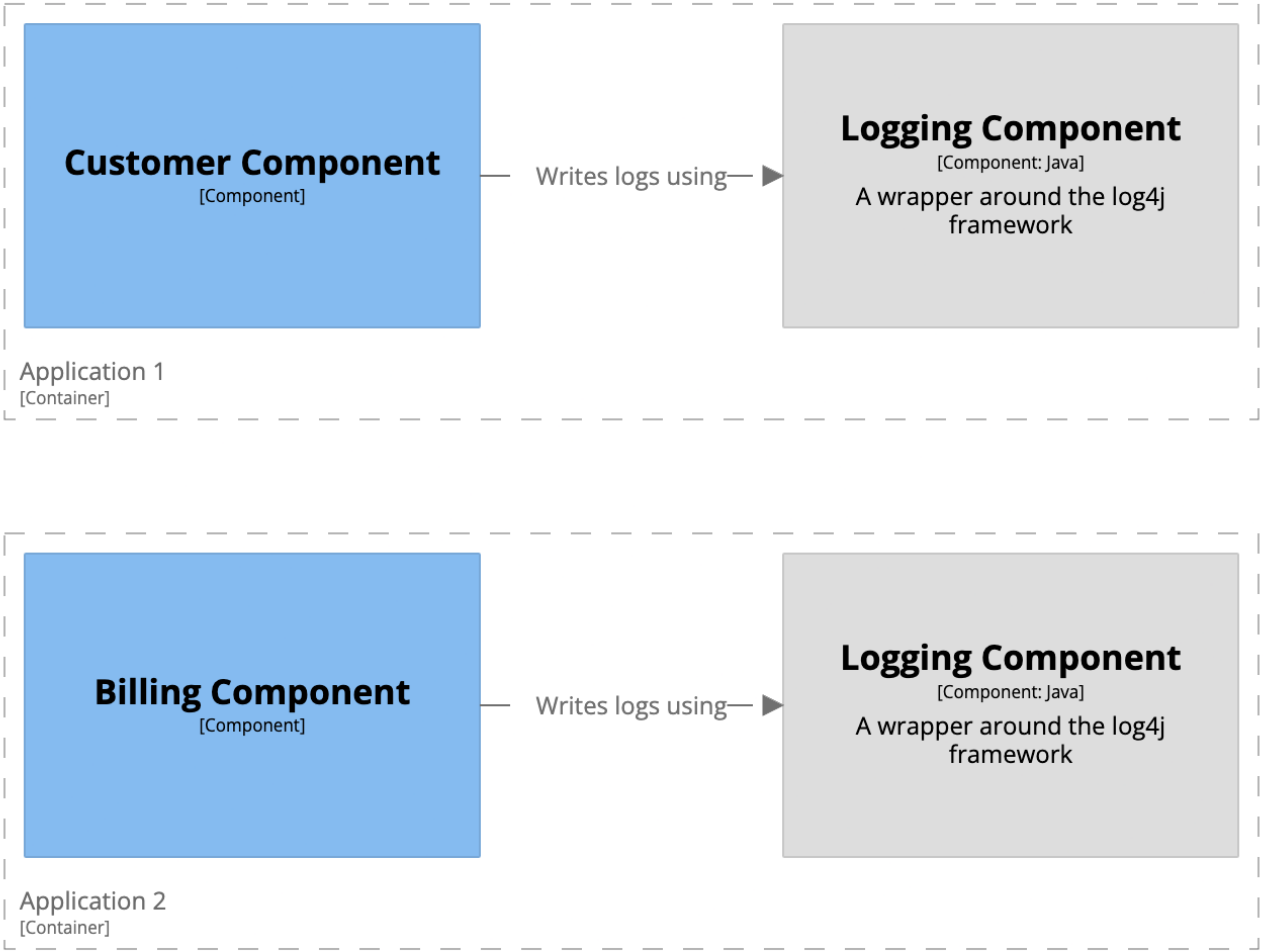


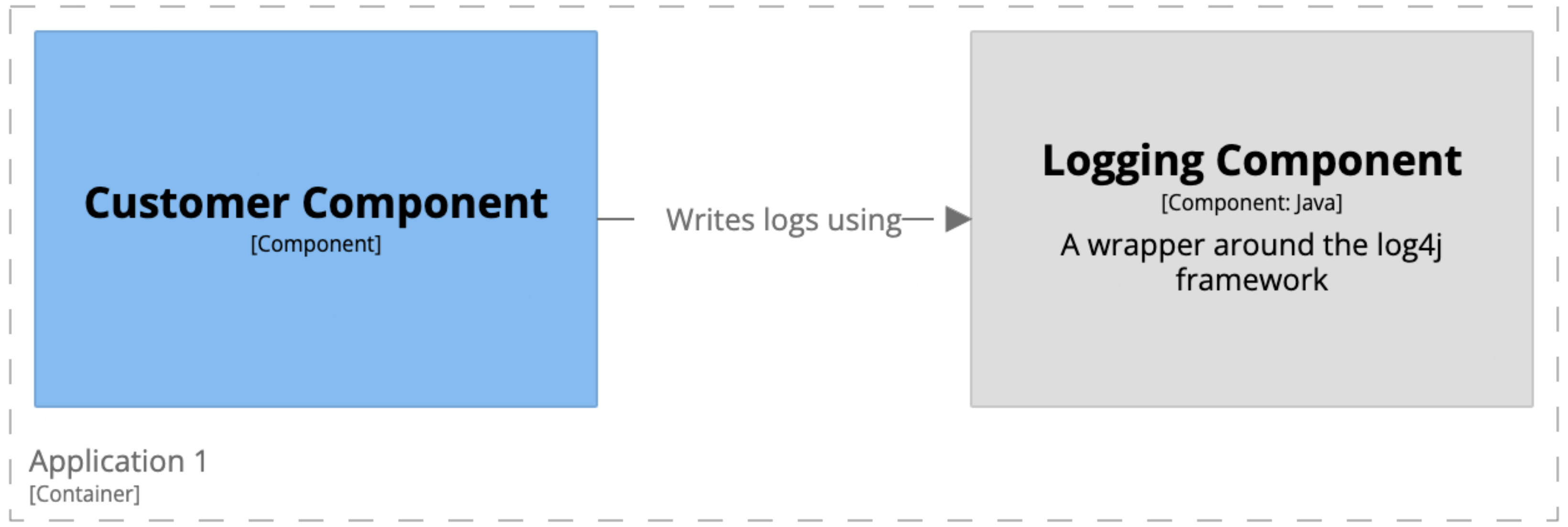
[Container] Software system X

Shared libraries


Customer Component
[Component]
Application 1
[Container]
Writes logs using
Logging Component
[Component: Java]
A wrapper around the log4j framework
shared-library.jar
[Container]
Writes logs using
Billing Component
[Component]
Application 2
[Container]


Customer Component
[Component]
Writes logs using
Logging Component
[Component: Java]
A wrapper around the log4j framework
Application 1
[Container]
Billing Component
[Component]
Writes logs using
Logging Component
[Component: Java]
A wrapper around the log4j framework
Application 2
[Container]


Customer Component
[Component]
Writes logs using
Logging Component
[Component: Java]
A wrapper around the log4j framework
Application 1
[Container]
Billing Component
[Component]
Writes logs using
Logging Component
[Component: Java]
A wrapper around the log4j framework
Application 2
[Container]

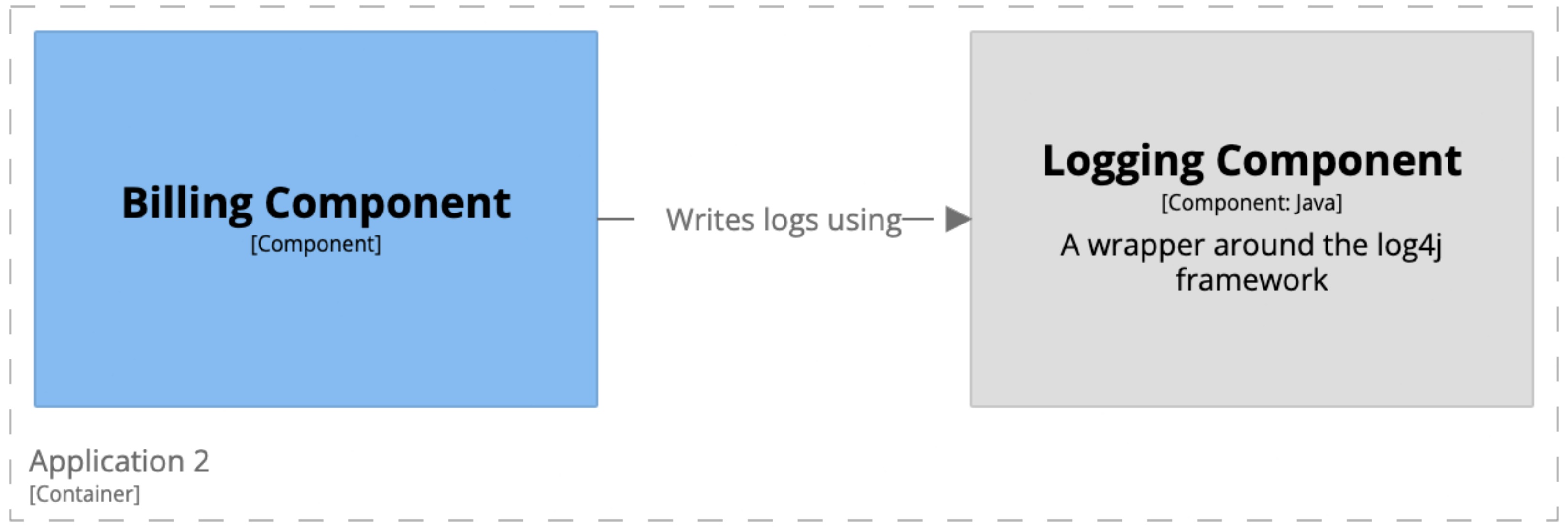


Customer Component
[Component]

Writes logs using

Logging Component
[Component: Java]
A wrapper around the log4j framework

Application 1
[Container]

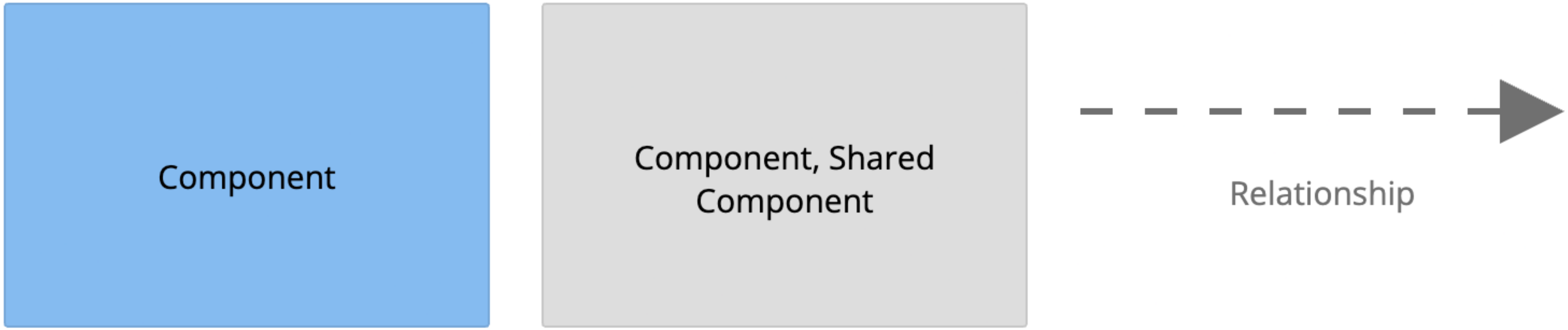


Billing Component
[Component]

Writes logs using

Logging Component
[Component: Java]
A wrapper around the log4j framework

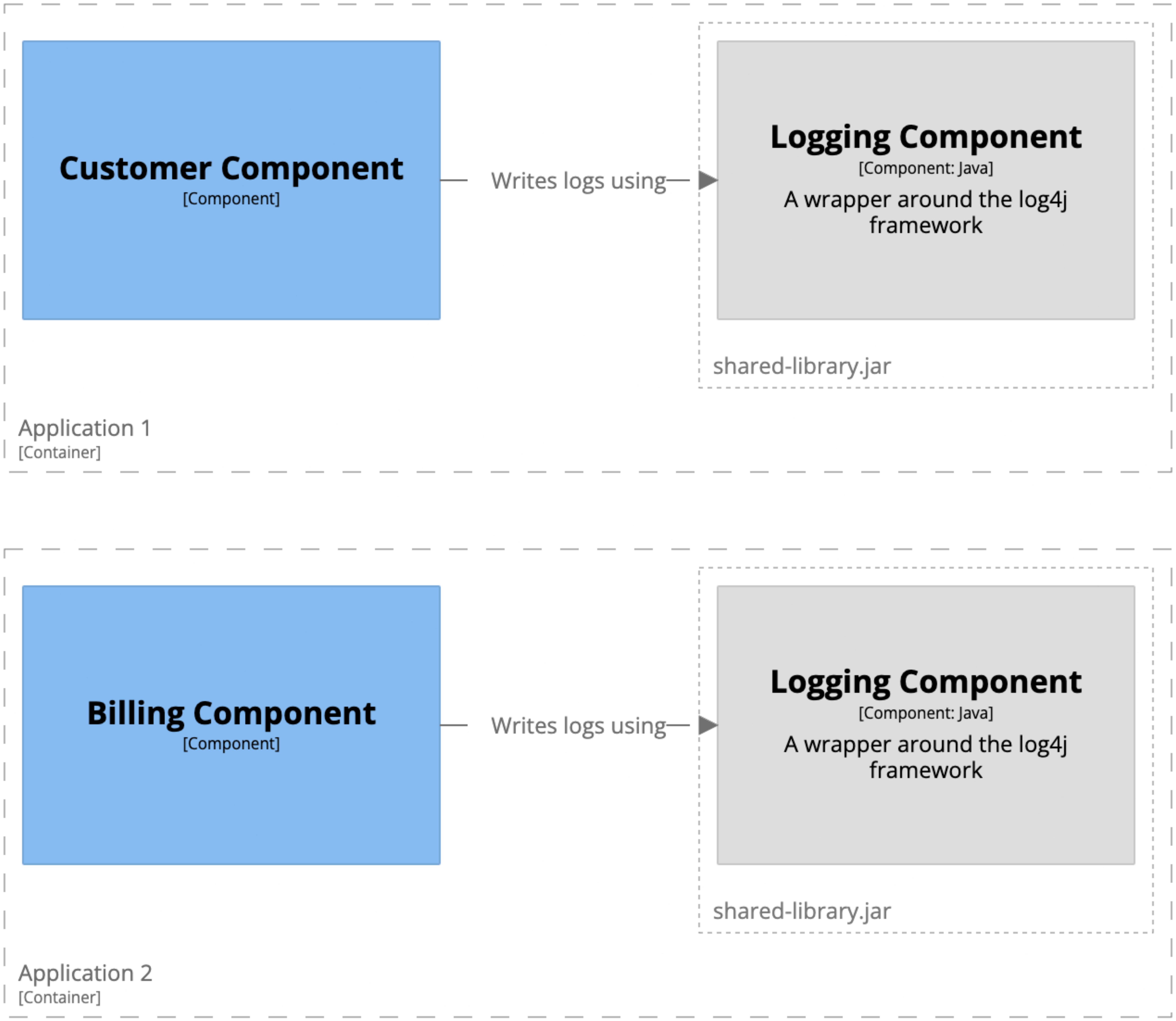
Application 2
[Container]



Component

Component, Shared Component

Relationship


Customer Component
[Component]
Writes logs using
Logging Component
[Component: Java]
A wrapper around the log4j framework
shared-library.jar
Application 1
[Container]
Billing Component
[Component]
Writes logs using
Logging Component
[Component: Java]
A wrapper around the log4j framework
shared-library.jar
Application 2
[Container]

Microservices

> C4 is more suited to monolithic architectures, and doesn’t support distributed architectures well

We’re modelling microservices as containers, with APIs and database schemas as components

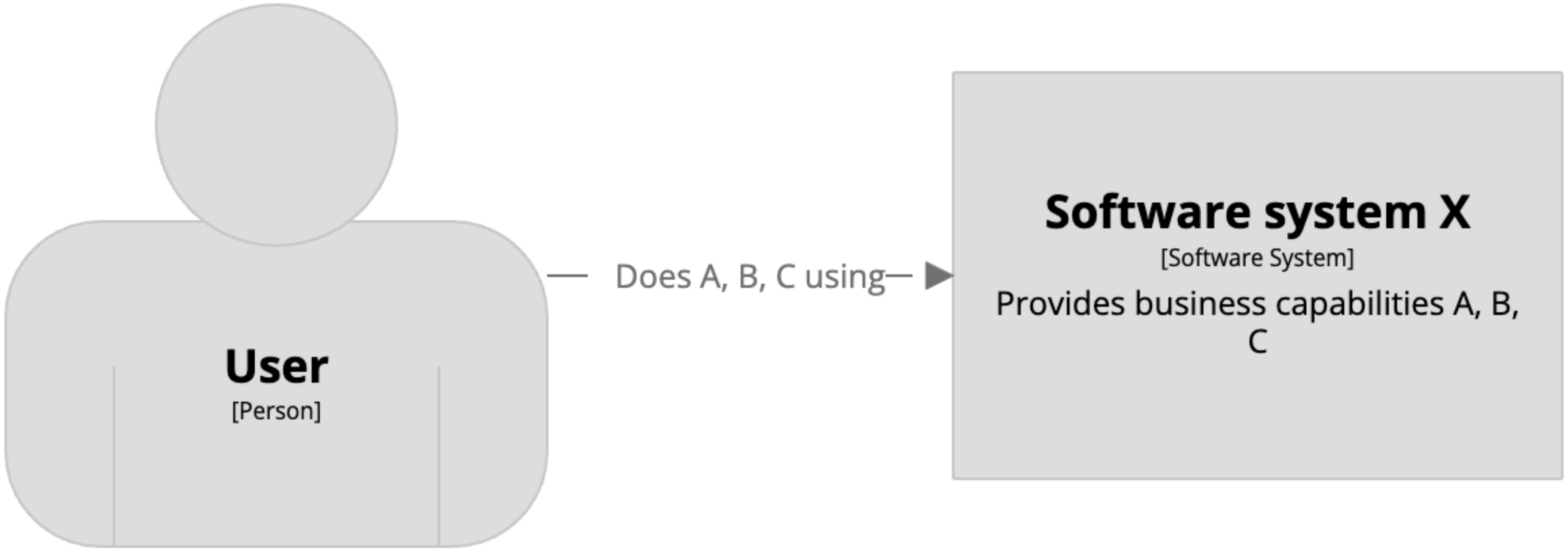
A microservice should be modelled as one of the following:

1. A software system
2. A container
3. A group of containers

What is a “microservice”?

Stage 1: 💵

(monolithic architecture)



User

[Person]

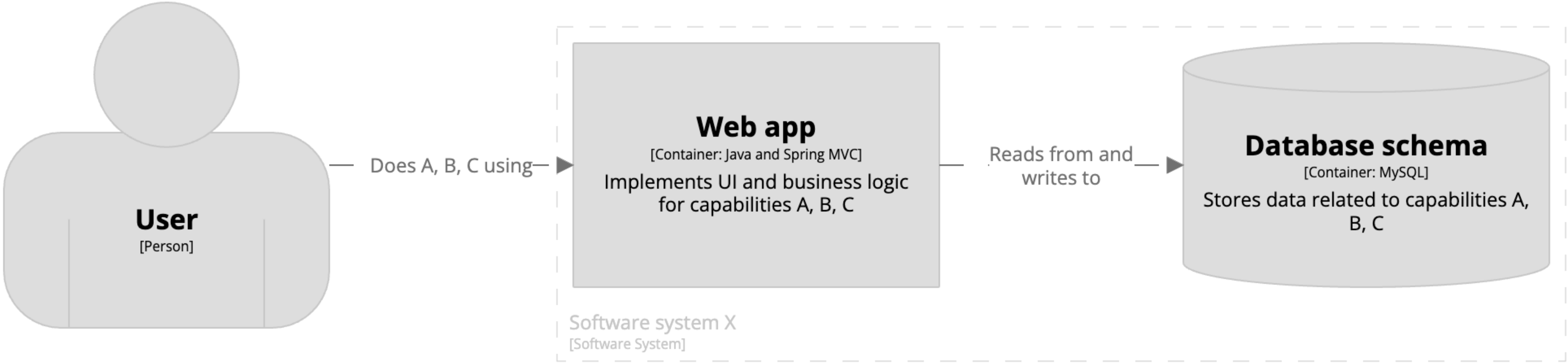
Does A, B, C using

Software system X

[Software System]

Provides business capabilities A, B, C

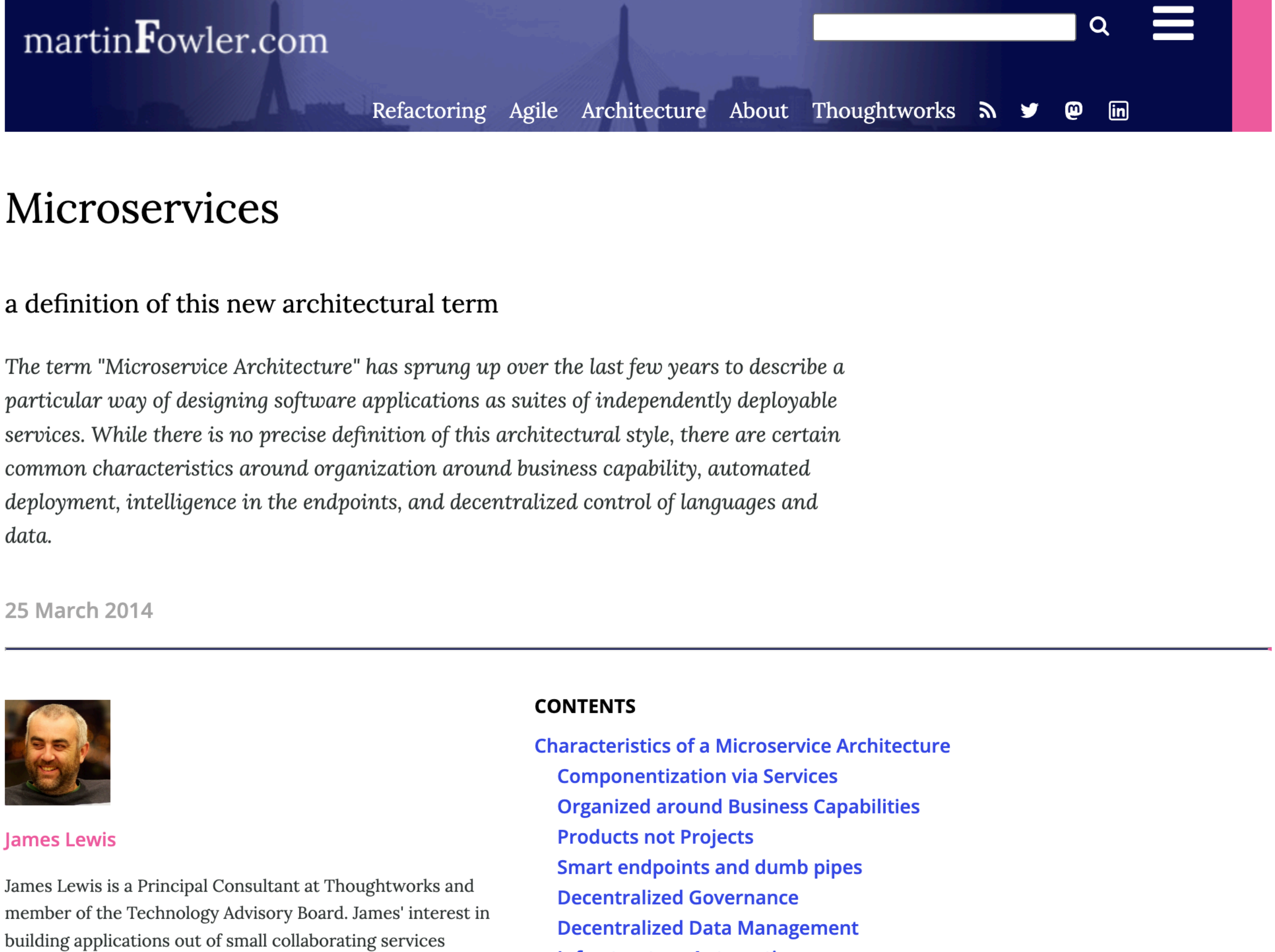
[System Context] Software system X


User
[Person]
Does A, B, C using
Web app
[Container: Java and Spring MVC]
Implements UI and business logic for capabilities A, B, C
Reads from and writes to
Database schema
[Container: MySQL]
Stores data related to capabilities A, B, C
Software system X
[Software System]

[Container] Software system X

Stage 2: 💵 💵

(microservices)



Microservices

a definition of this new architectural term

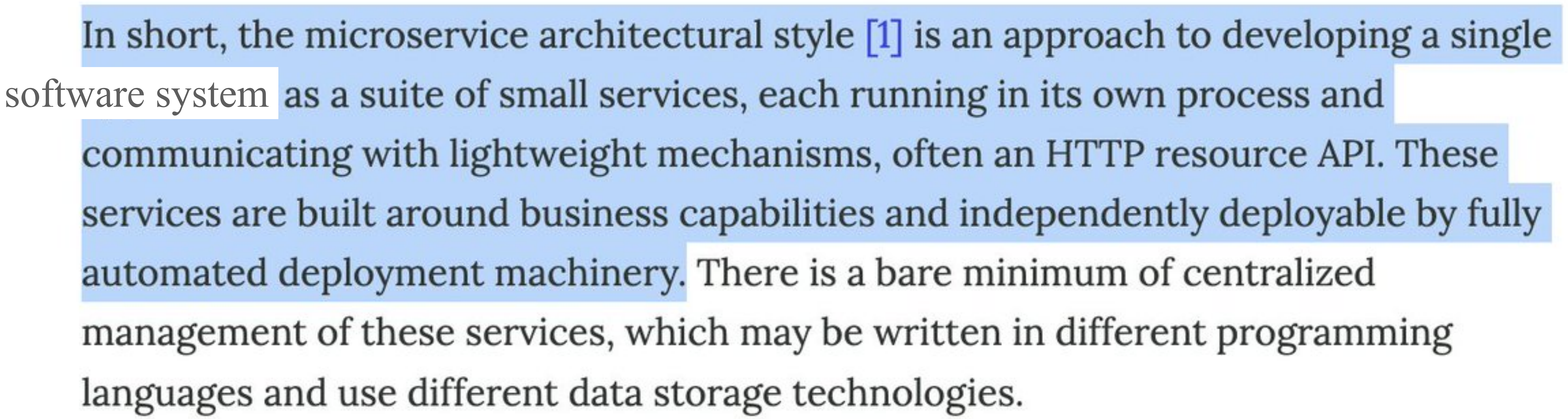
The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

25 March 2014

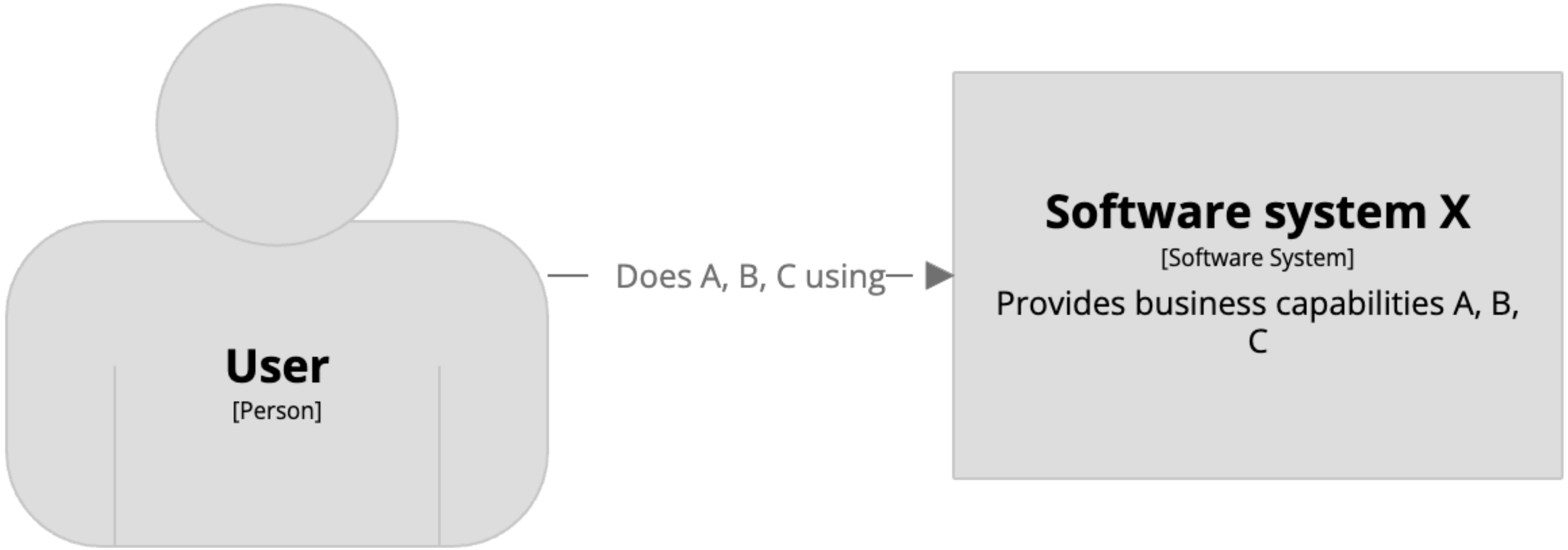
James Lewis

James Lewis is a Principal Consultant at Thoughtworks and member of the Technology Advisory Board. James' interest in building applications out of small collaborating services

CONTENTS



In short, the microservice architectural style [1] is an approach to developing a single software system as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.



User

[Person]

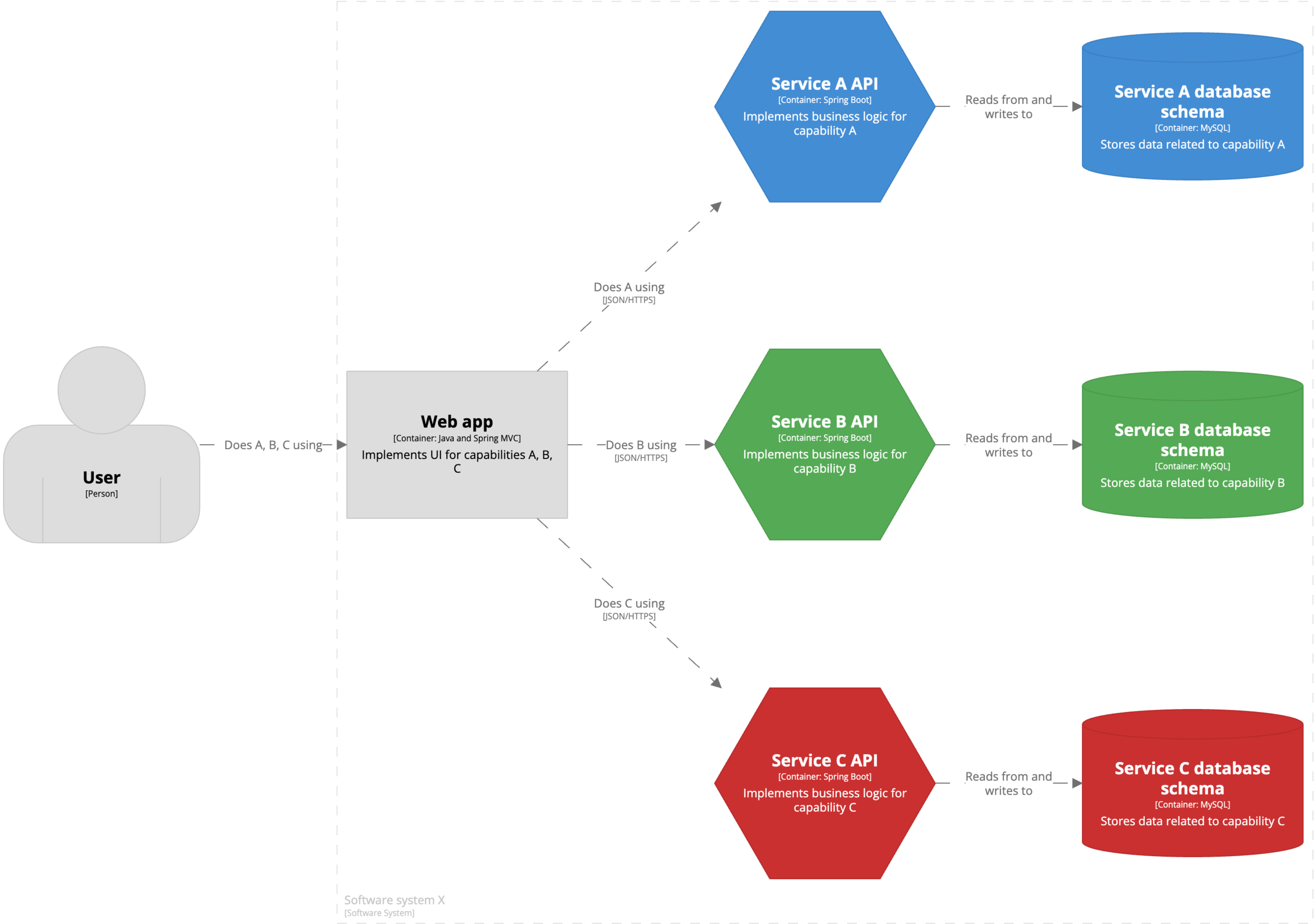
Does A, B, C using

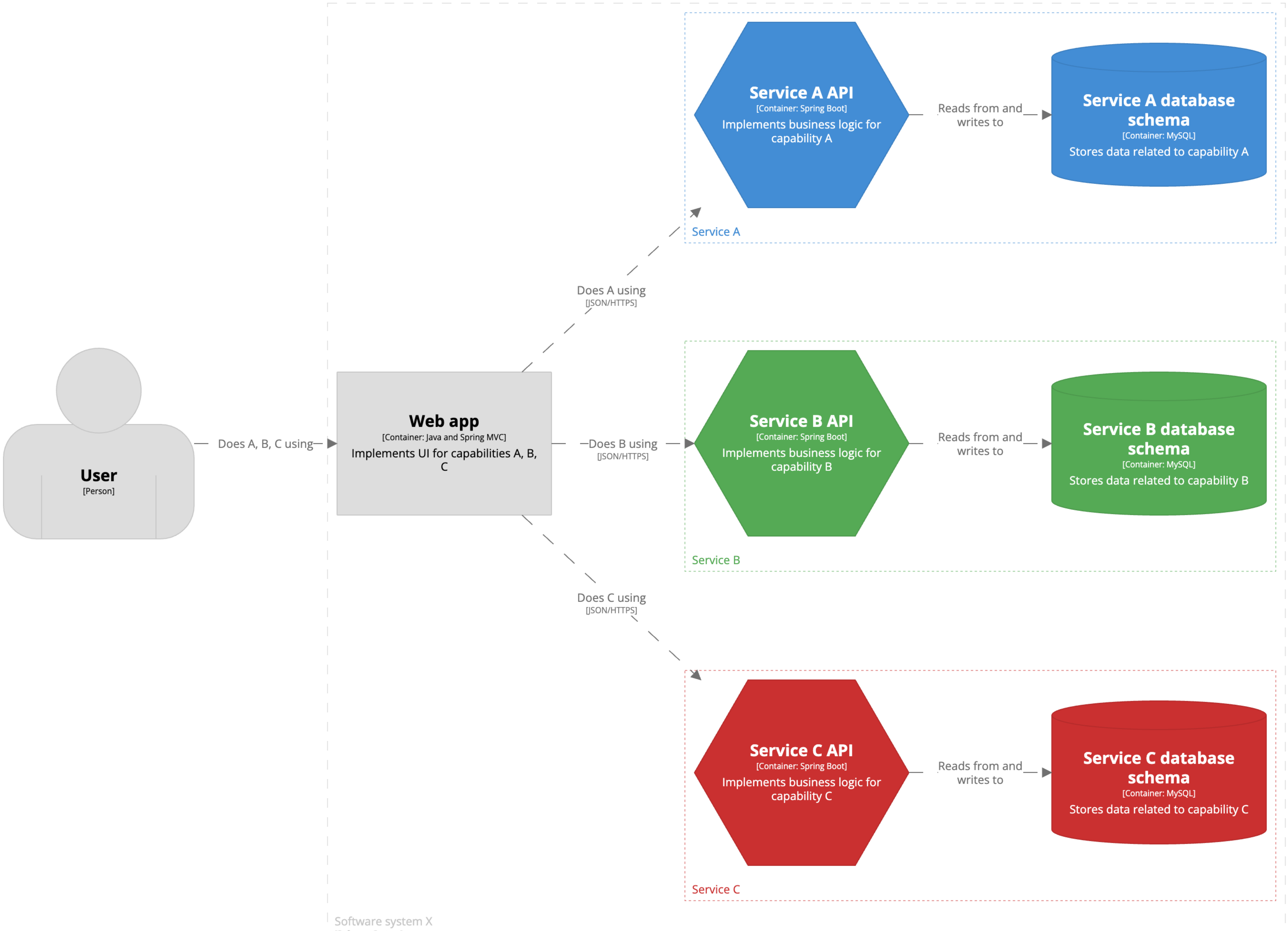
Software system X

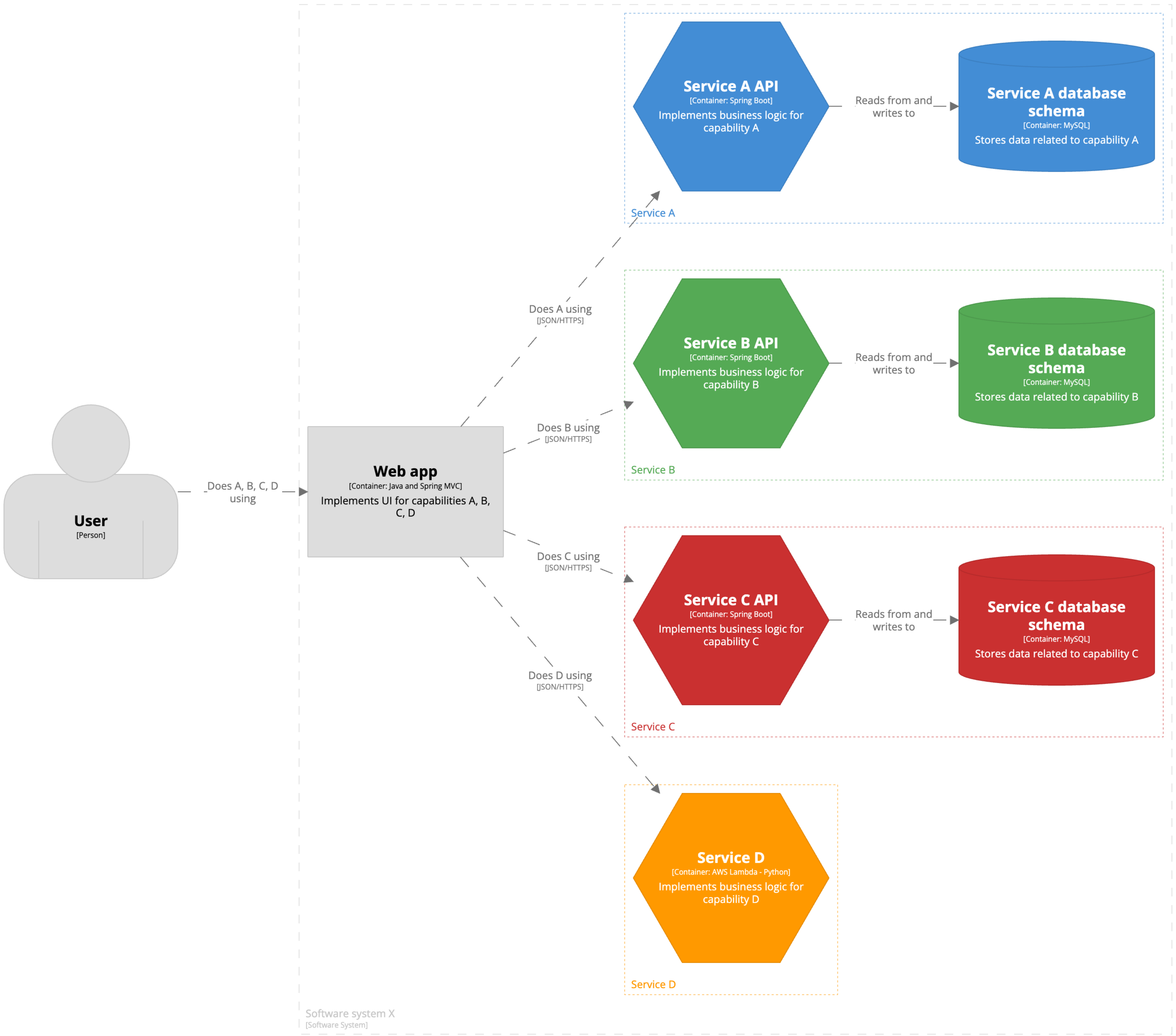
[Software System]

Provides business capabilities A, B, C

[System Context] Software system X


User
[Person]
Does A, B, C using
Web app
[Container: Java and Spring MVC]
Implements UI for capabilities A, B, C
Does A using
[JSON/HTTPS]
Does B using
[JSON/HTTPS]
Does C using
[JSON/HTTPS]
Service A API
[Container: Spring Boot]
Implements business logic for capability A
Service B API
[Container: Spring Boot]
Implements business logic for capability B
Service C API
[Container: Spring Boot]
Implements business logic for capability C
Reads from and writes to
Reads from and writes to
Reads from and writes to
Service A database schema
[Container: MySQL]
Stores data related to capability A
Service B database schema
[Container: MySQL]
Stores data related to capability B
Service C database schema
[Container: MySQL]
Stores data related to capability C
Software system X
[Software System]


User
[Person]
Does A, B, C using
Web app
[Container: Java and Spring MVC]
Implements UI for capabilities A, B, C
Does A using
[JSON/HTTPS]
Does B using
[JSON/HTTPS]
Does C using
[JSON/HTTPS]
Service A API
[Container: Spring Boot]
Implements business logic for capability A
Reads from and writes to
Service A database schema
[Container: MySQL]
Stores data related to capability A
Service A
Service B API
[Container: Spring Boot]
Implements business logic for capability B
Reads from and writes to
Service B database schema
[Container: MySQL]
Stores data related to capability B
Service B
Service C API
[Container: Spring Boot]
Implements business logic for capability C
Reads from and writes to
Service C database schema
[Container: MySQL]
Stores data related to capability C
Service C
Software system X



User
[Person]

Does A, B, C, D using

Web app
[Container: Java and Spring MVC]
Implements UI for capabilities A, B, C, D

Does A using
[JSON/HTTPS]

Does B using
[JSON/HTTPS]

Does C using
[JSON/HTTPS]

Does D using
[JSON/HTTPS]

Service A API
[Container: Spring Boot]
Implements business logic for capability A

Reads from and writes to

Service A database schema
[Container: MySQL]
Stores data related to capability A

Service A

Service B API
[Container: Spring Boot]
Implements business logic for capability B

Reads from and writes to

Service B database schema
[Container: MySQL]
Stores data related to capability B

Service B

Service C API
[Container: Spring Boot]
Implements business logic for capability C

Reads from and writes to

Service C database schema
[Container: MySQL]
Stores data related to capability C

Service C

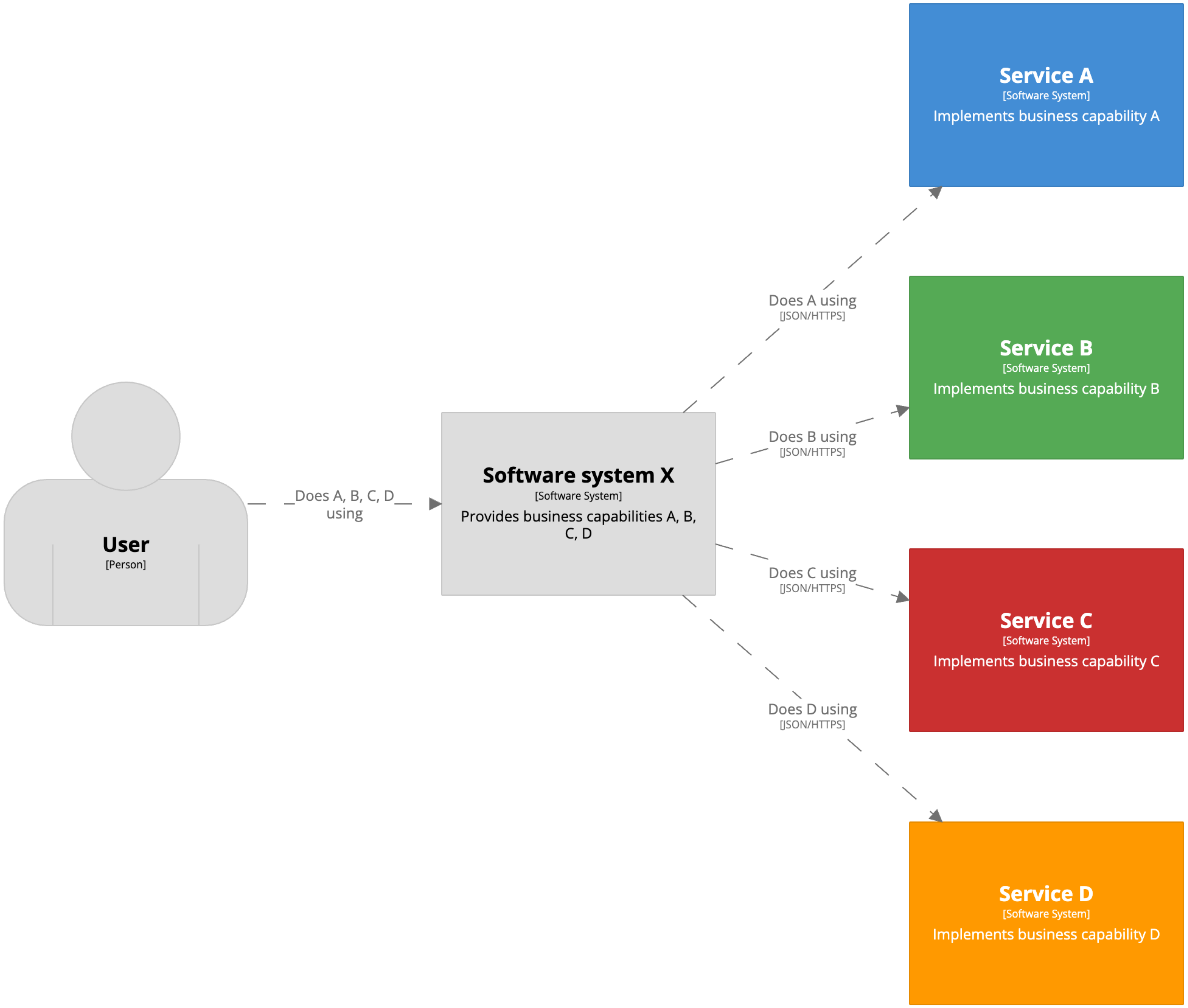
Service D
[Container: AWS Lambda - Python]
Implements business logic for capability D

Service D

Software system X
[Software System]

Stage 3: 💵 💵 💵

(Conway’s Law)



User
[Person]

_Does A, B, C, D__ using

Software system X
[Software System]
Provides business capabilities A, B, C, D

Does A using
[JSON/HTTPS]

Does B using
[JSON/HTTPS]

Does C using
[JSON/HTTPS]

Does D using
[JSON/HTTPS]

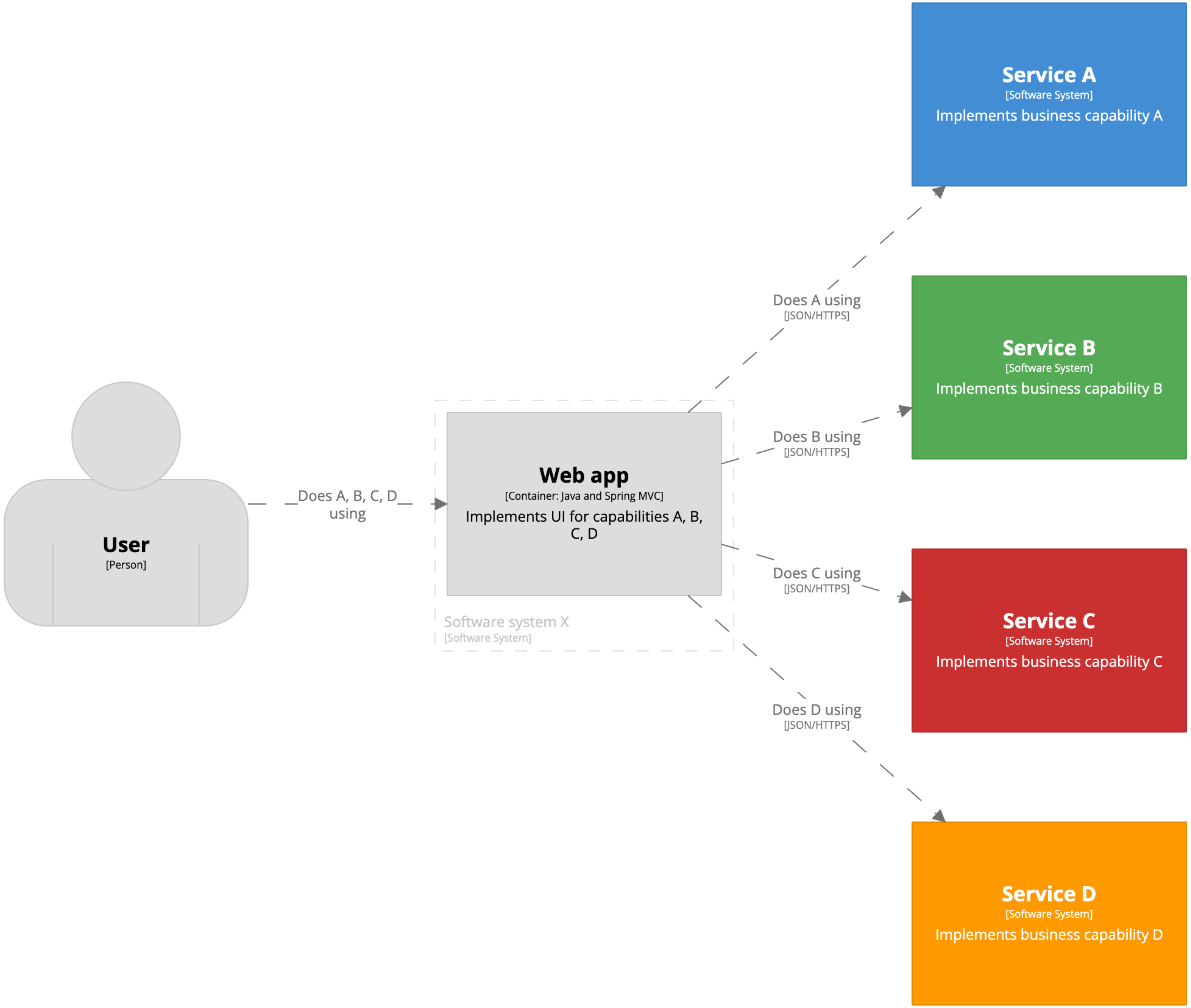
Service A
[Software System]
Implements business capability A

Service B
[Software System]
Implements business capability B

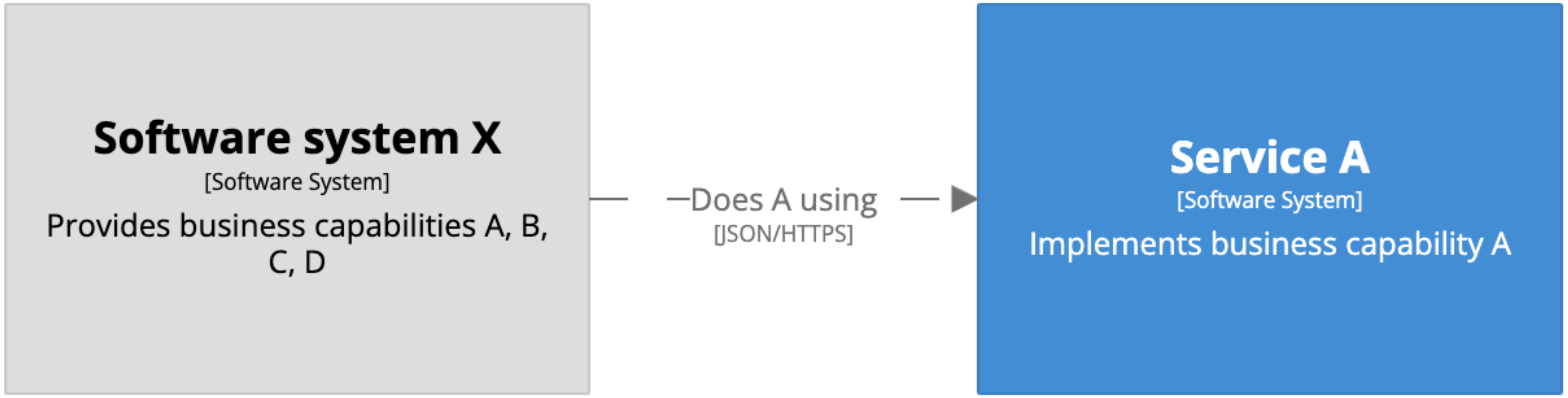
Service C
[Software System]
Implements business capability C

Service D
[Software System]
Implements business capability D

[System Context] Software system X



[Container] Software system X



Software system X

[Software System]

Provides business capabilities A, B, C, D

Does A using

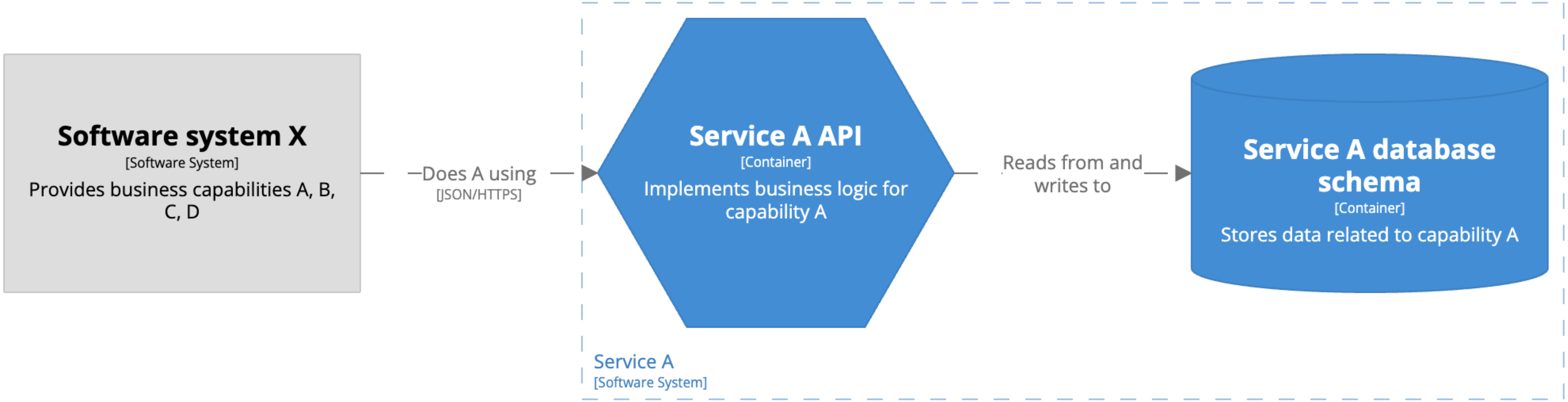
[JSON/HTTPS]

Service A

[Software System]

Implements business capability A

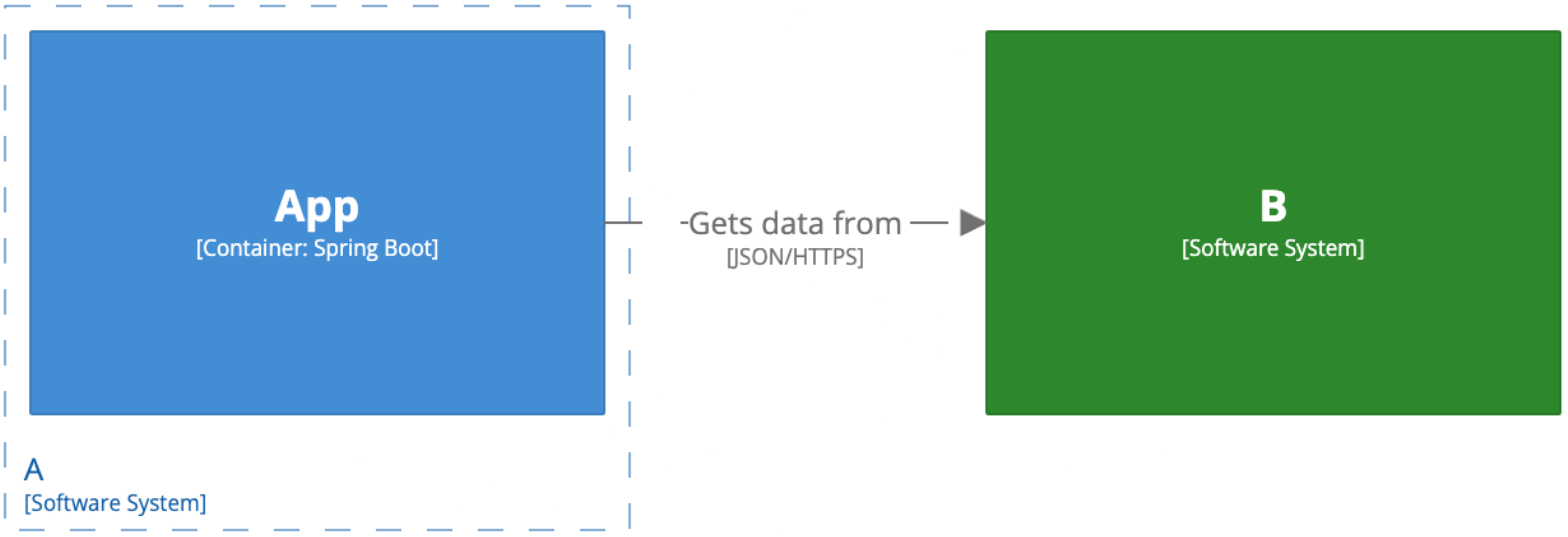
[System Context] Service A


Software system X
[Software System]
Provides business capabilities A, B, C, D
Does A using
[JSON/HTTPS]
Service A API
[Container]
Implements business logic for capability A
Reads from and writes to
Service A database schema
[Container]
Stores data related to capability A
Service A
[Software System]

[Container] Service A

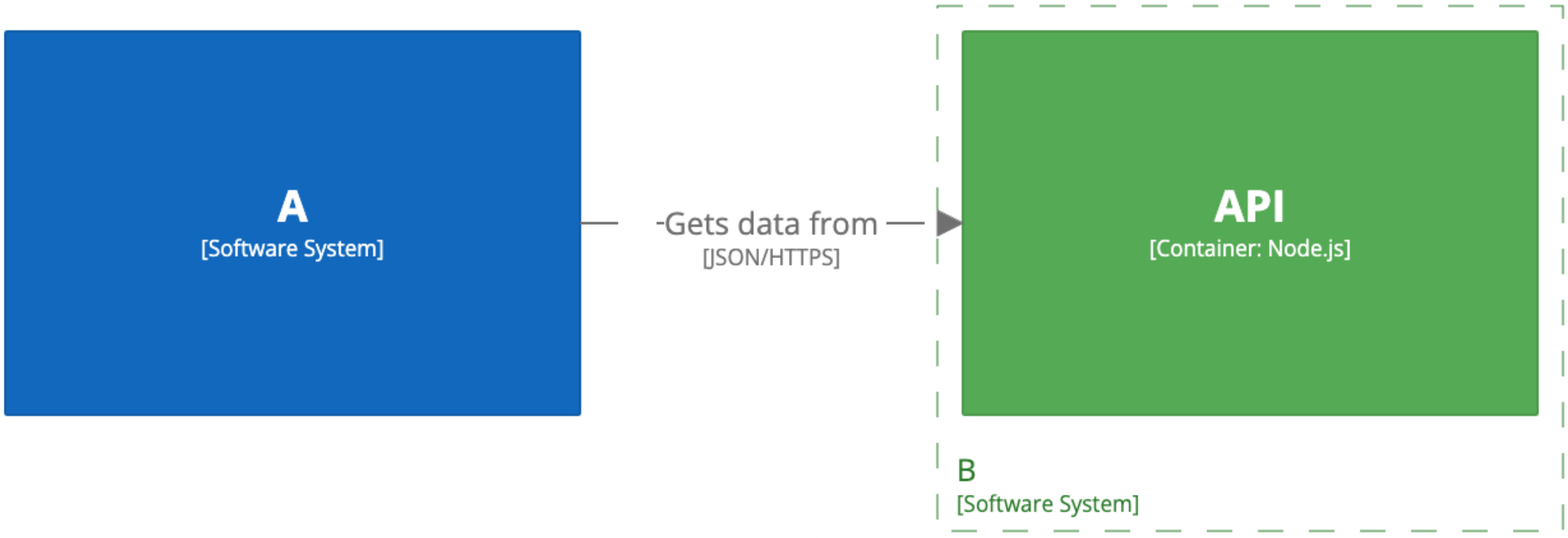
Dependencies to “external” containers

My recommendation is that container diagrams only show containers inside the software system that is the scope of the diagram



Container diagram for software system A

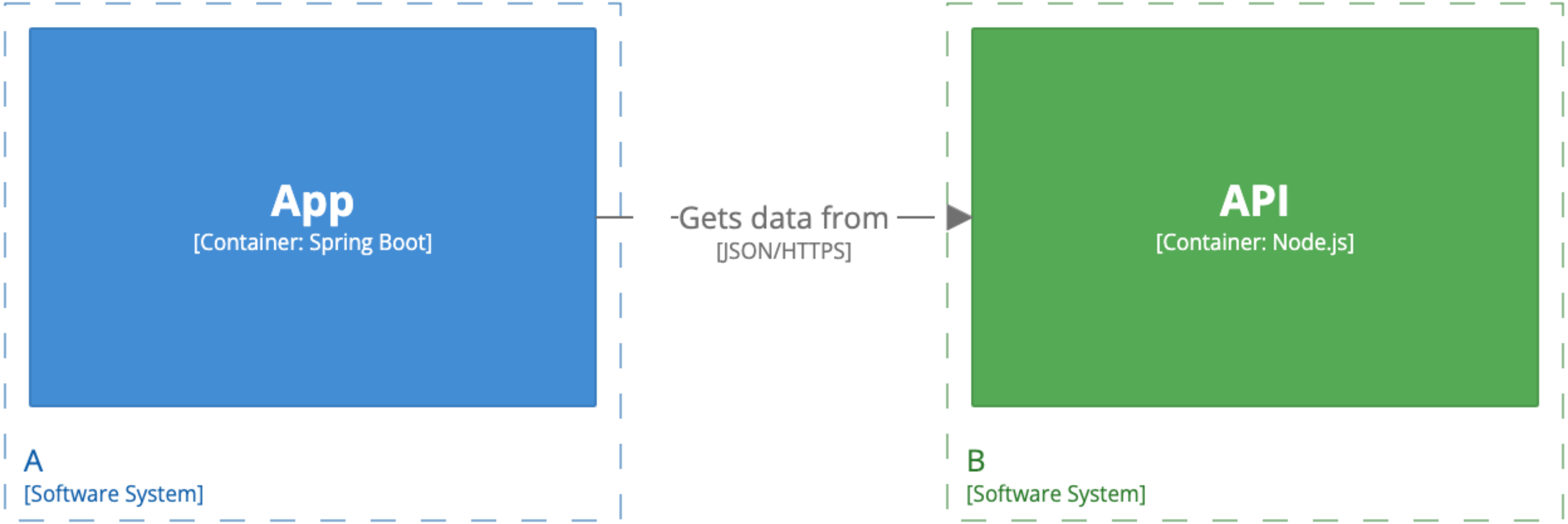
```
container a {
    include *
}
```



Container diagram for software system B

```
container b {
    include *
}
```

I don’t recommend showing “external” containers



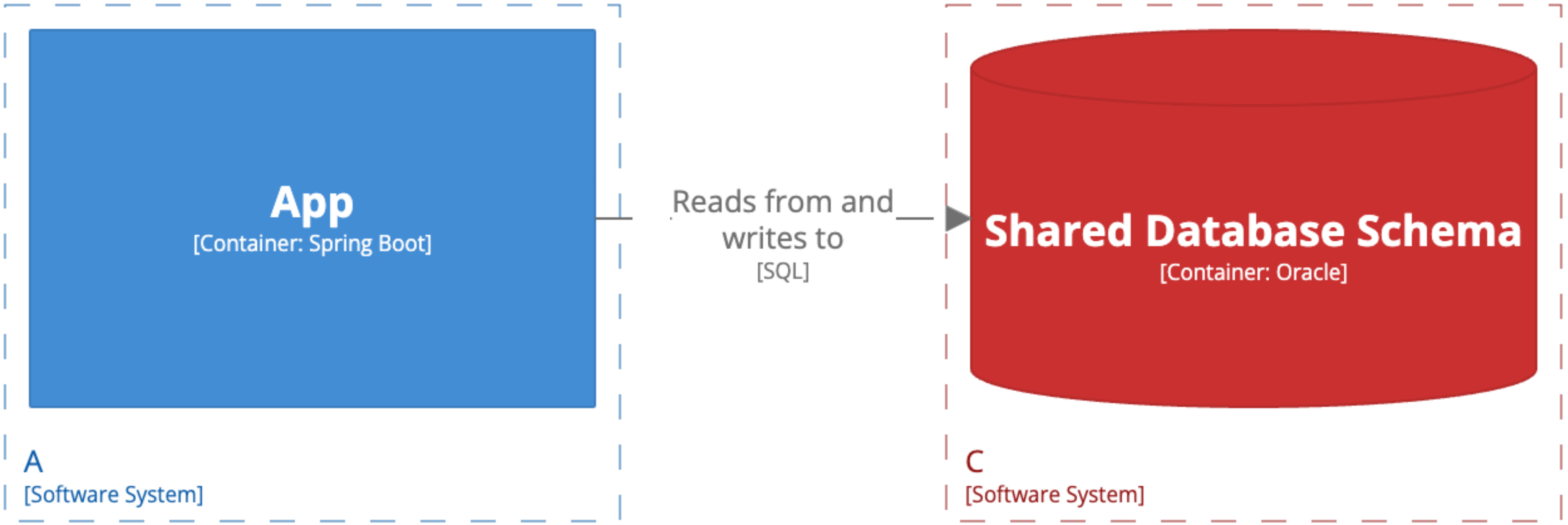
Container diagram for software systems A and B

```
container a {
    include a.app b.api
}
```

Showing “external” containers implies some understanding of implementation details, which makes the diagrams more volatile to change

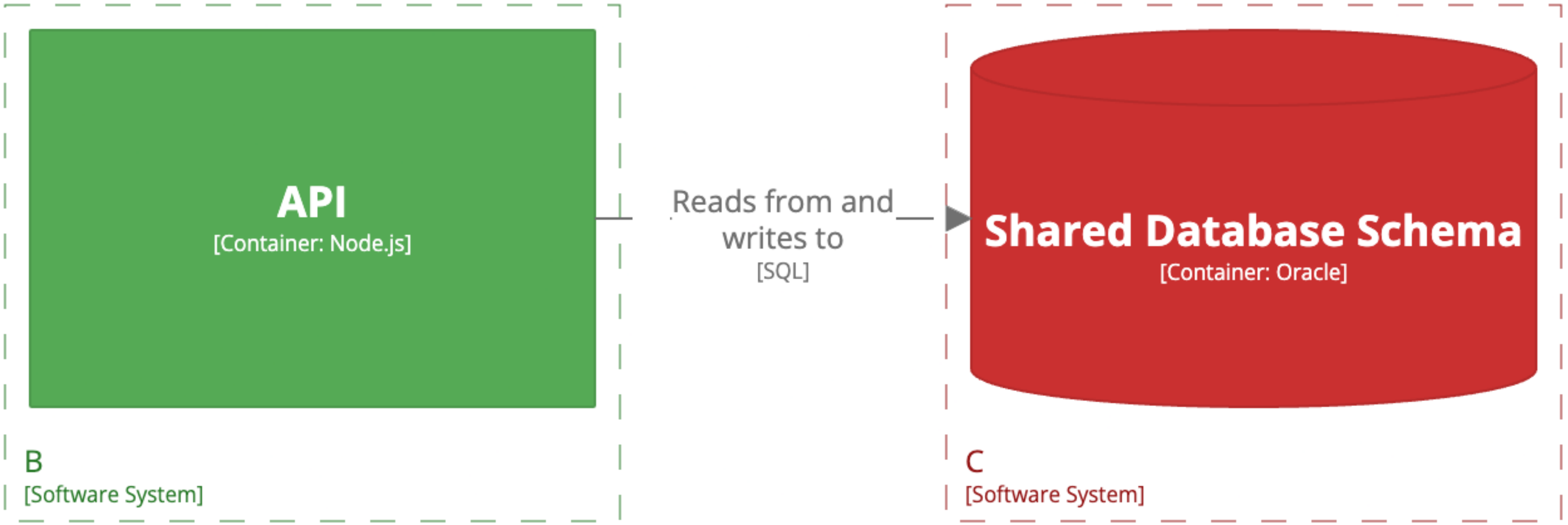
This is a form of coupling

There may some useful exceptions to this guidance...



Container diagram for software system A, showing a shared DB

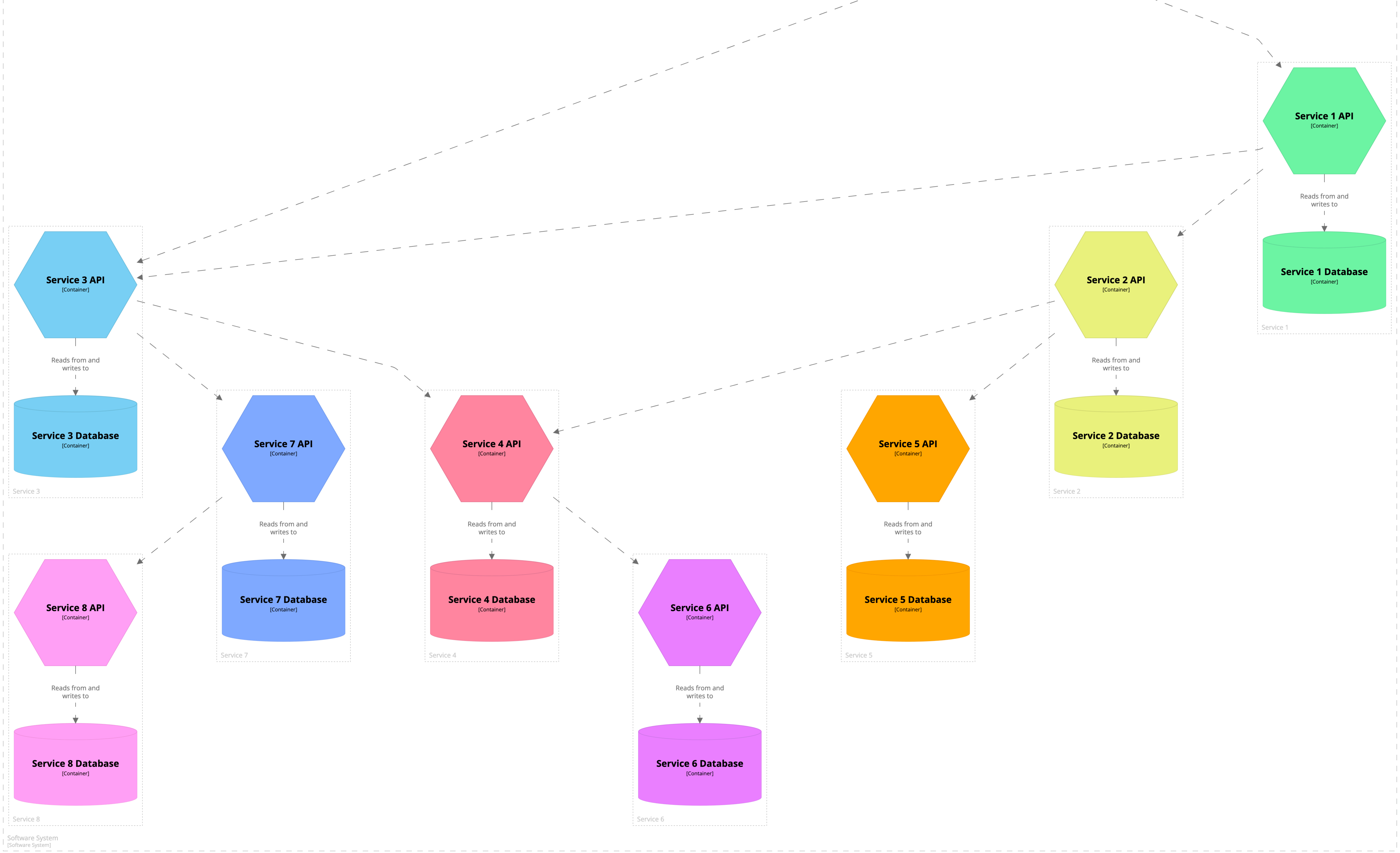
```
container a {
    include a.app c.db
}
```

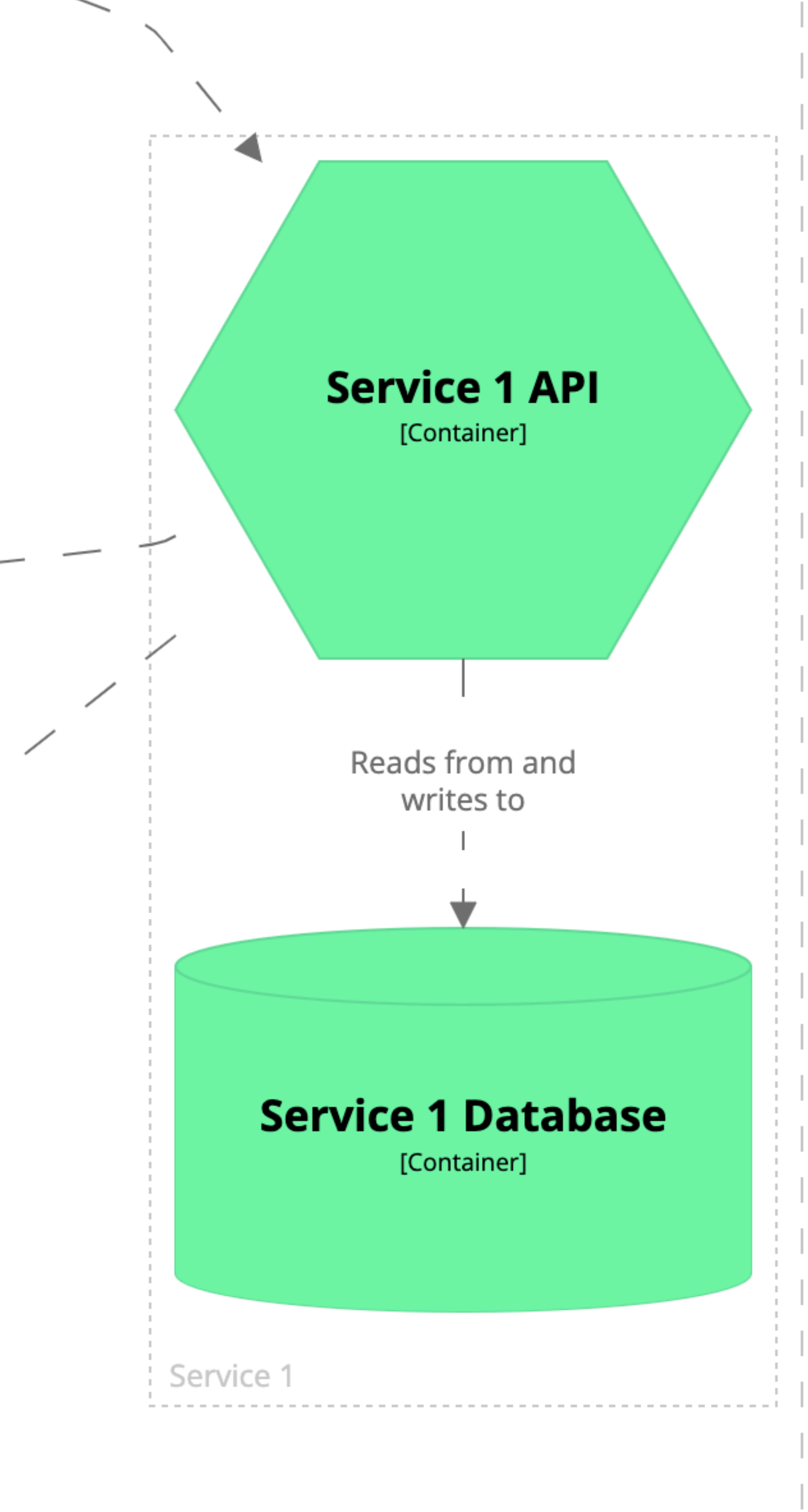


Container diagram for software system B, showing a shared DB

```
container b {
    include b.api c.db
}
```

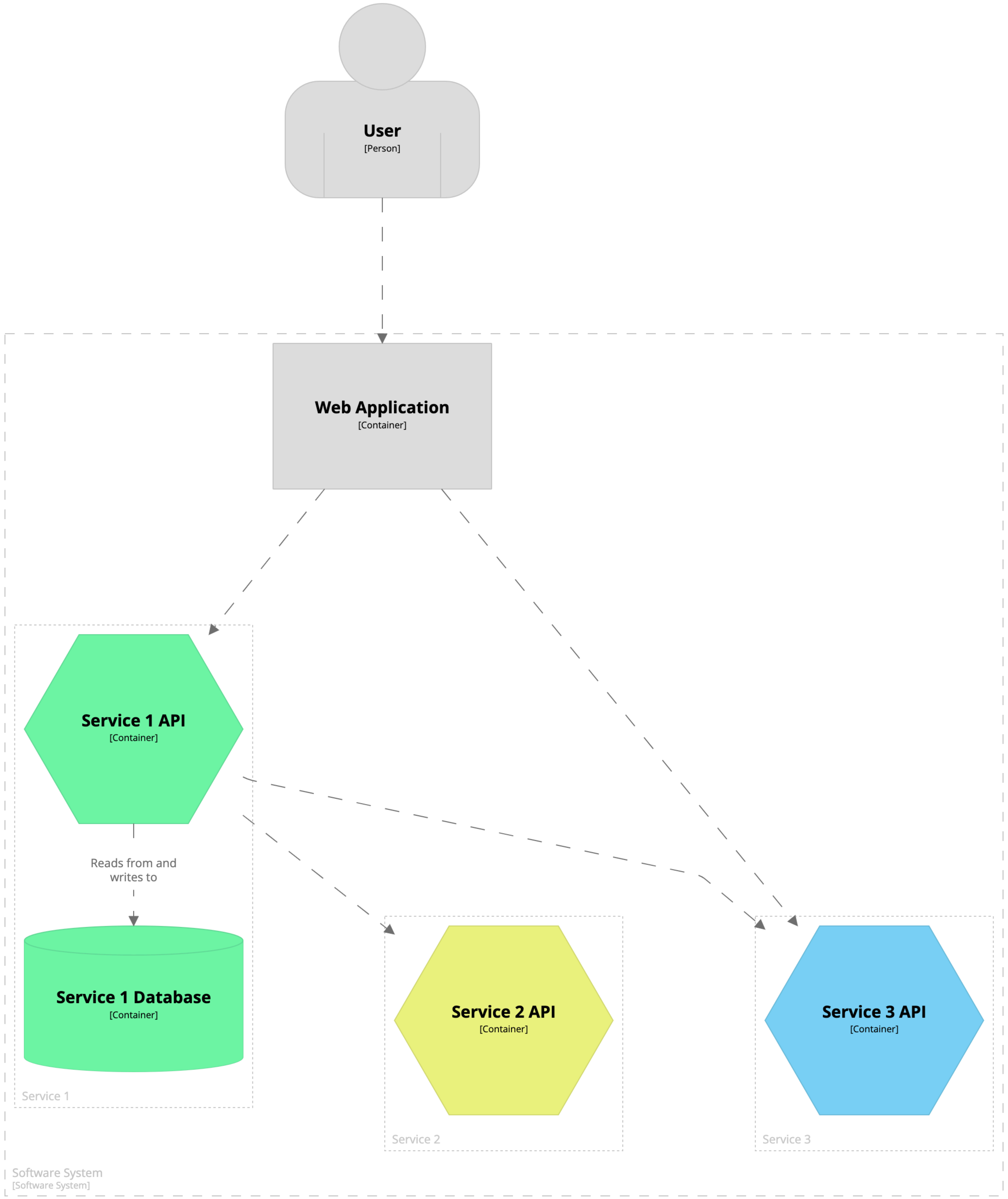
> C4 doesn't scale


Service 1 API
[Container]
Reads from and writes to
Service 1 Database
[Container]
Service 1
Service 3 API
[Container]
Reads from and writes to
Service 3 Database
[Container]
Service 3
Service 2 API
[Container]
Reads from and writes to
Service 2 Database
[Container]
Service 2
Service 7 API
[Container]
Reads from and writes to
Service 7 Database
[Container]
Service 7
Service 4 API
[Container]
Reads from and writes to
Service 4 Database
[Container]
Service 4
Service 5 API
[Container]
Reads from and writes to
Service 5 Database
[Container]
Service 5
Service 8 API
[Container]
Reads from and writes to
Service 8 Database
[Container]
Service 8
Service 6 API
[Container]
Reads from and writes to
Service 6 Database
[Container]
Service 6
Software System
[Software System]

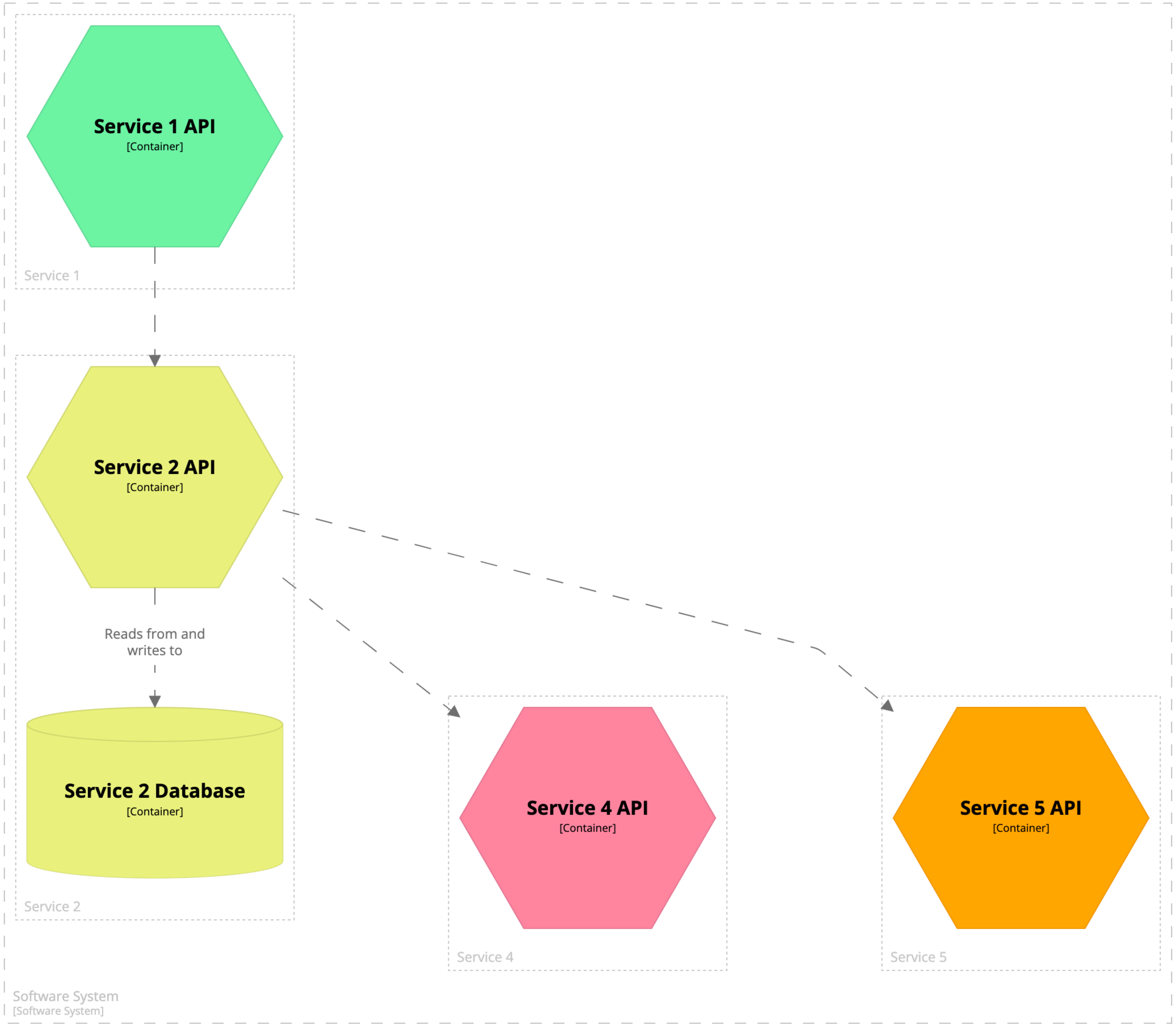


In this example,
a microservice is
a combination of
an API and
a database schema

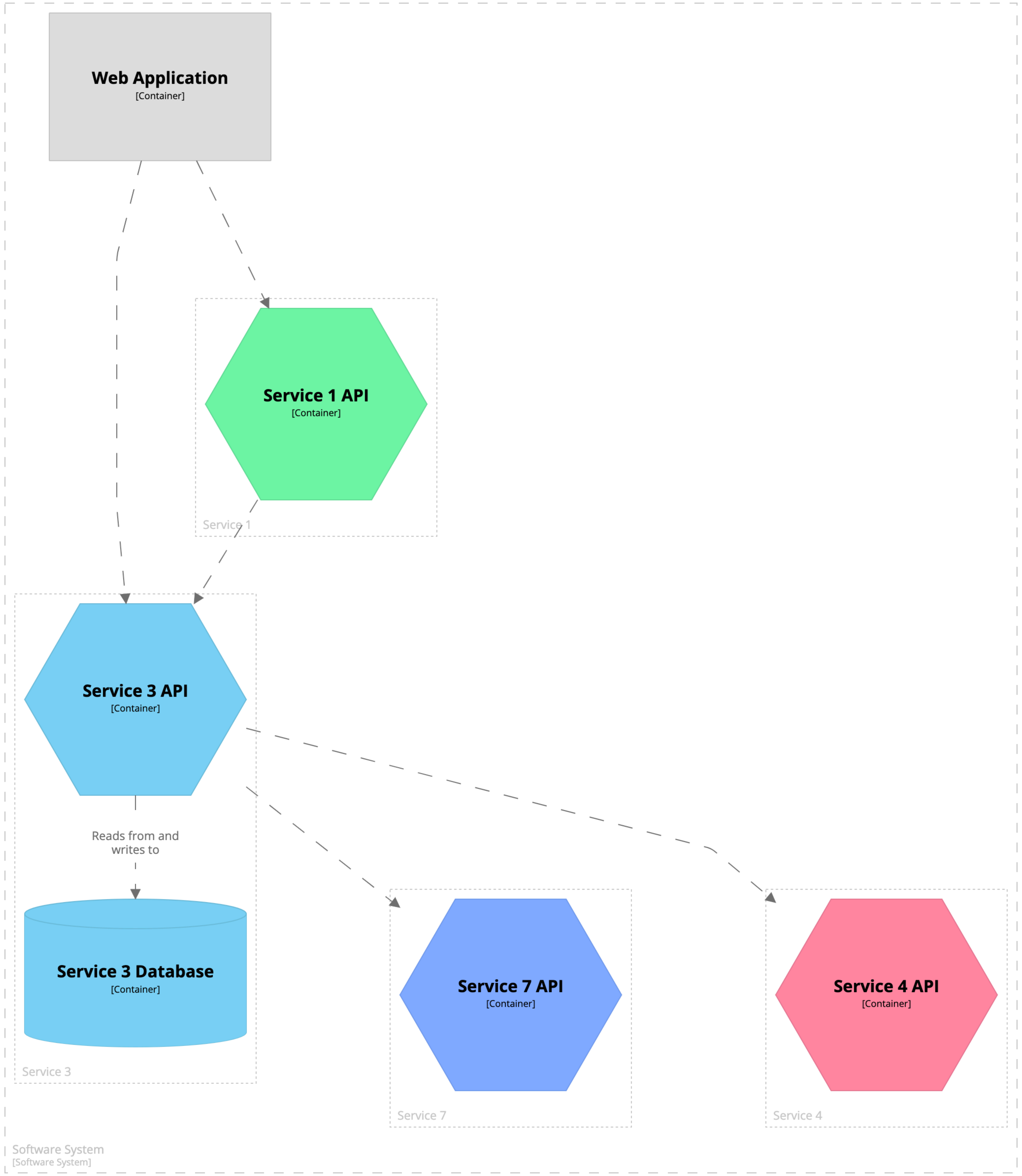
```
container softwareSystem {
    include user
    Include ->service1->
}
```



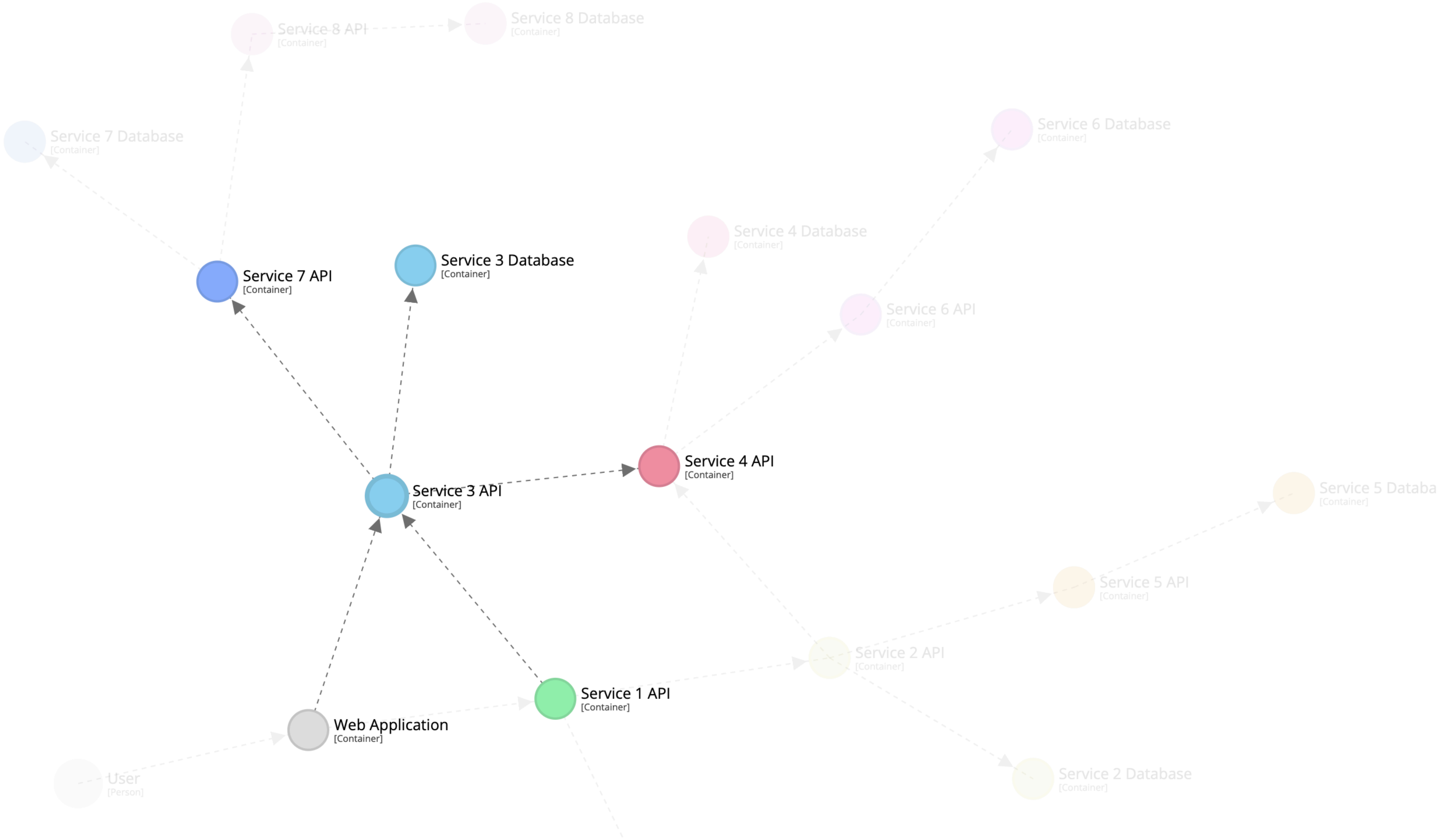
```
container softwareSystem {
    include ->service2->
}
```



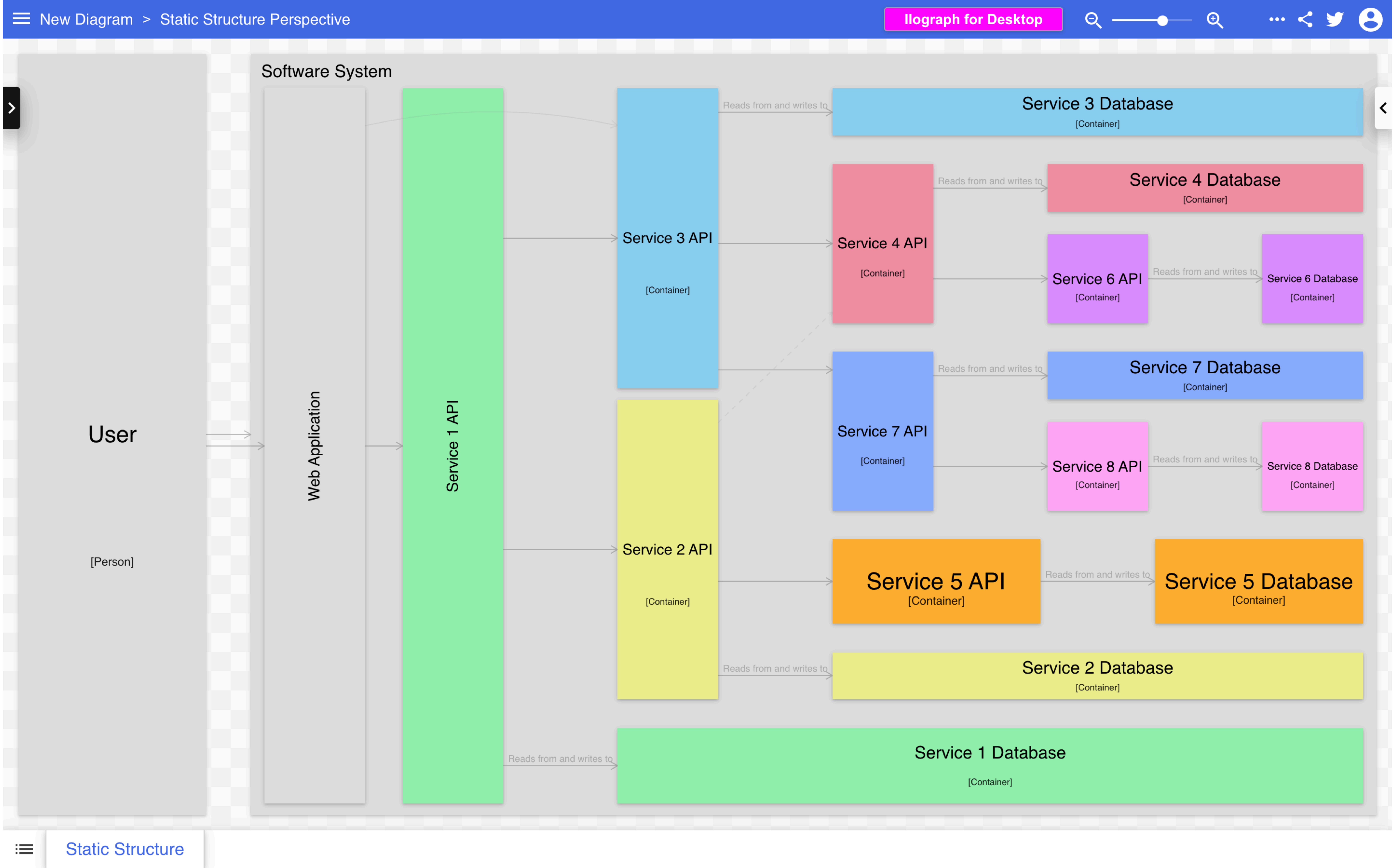
```
container softwareSystem {
    include ->service3->
}
```

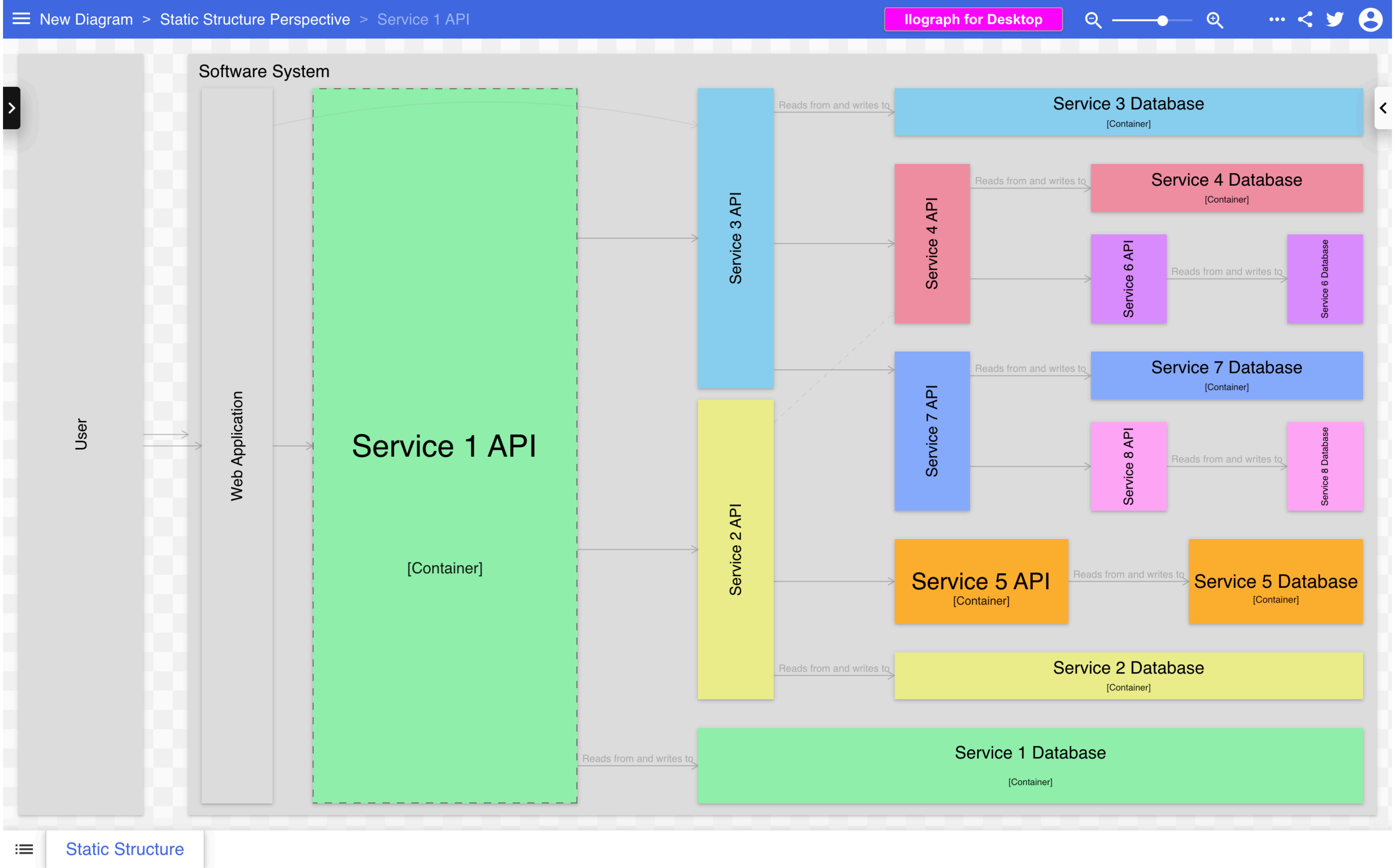


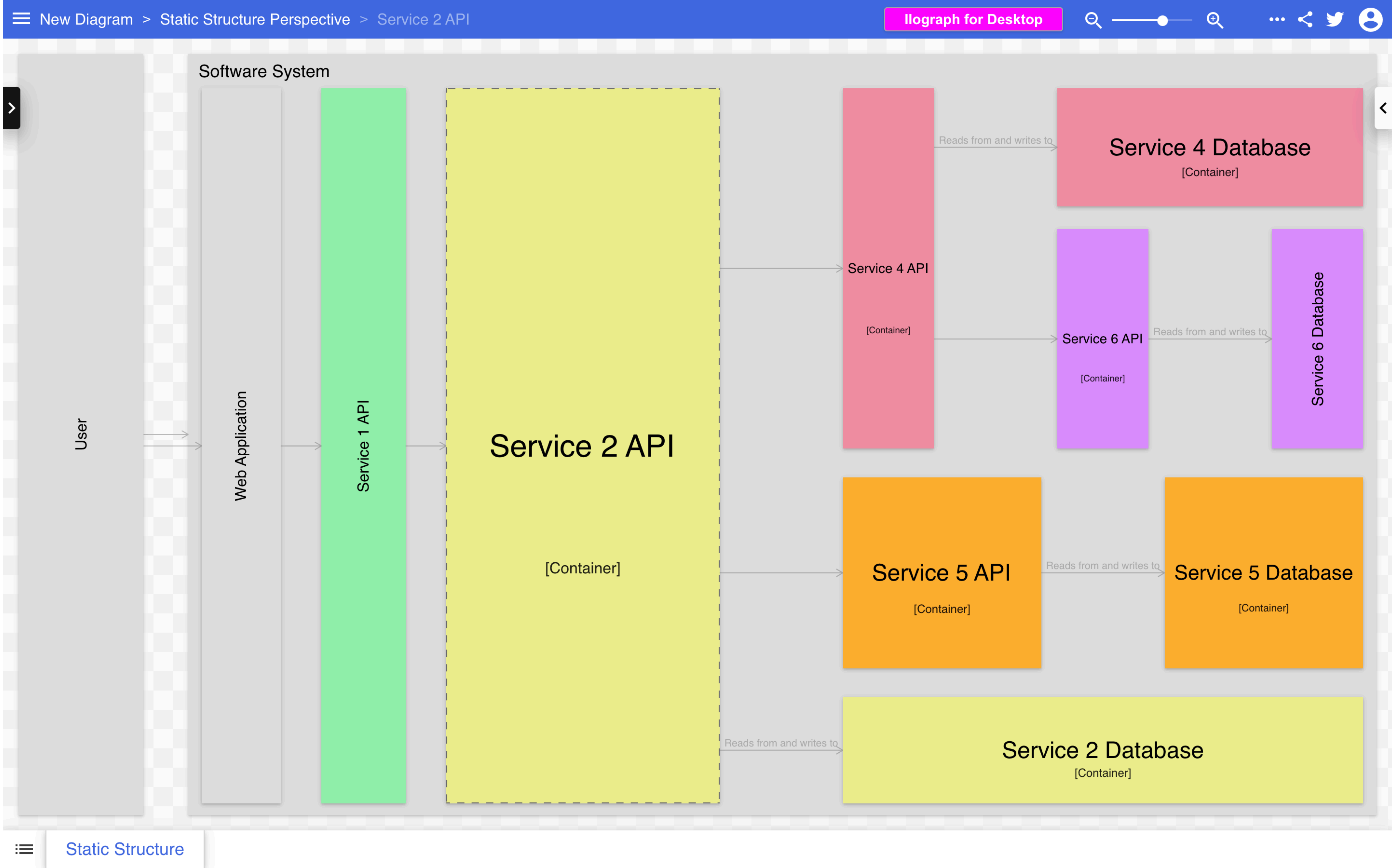
Service 8 API
[Container]
Service 8 Database
[Container]
Service 7 Database
[Container]
Service 6 Database
[Container]
Service 4 Database
[Container]
Service 3 Database
[Container]
Service 7 API
[Container]
Service 6 API
[Container]
Service 4 API
[Container]
Service 3 API
[Container]
Service 5 Databa
[Container]
Service 5 API
[Container]
Service 2 API
[Container]
Service 1 API
[Container]
Web Application
[Container]
User
[Person]
Service 2 Database
[Container]

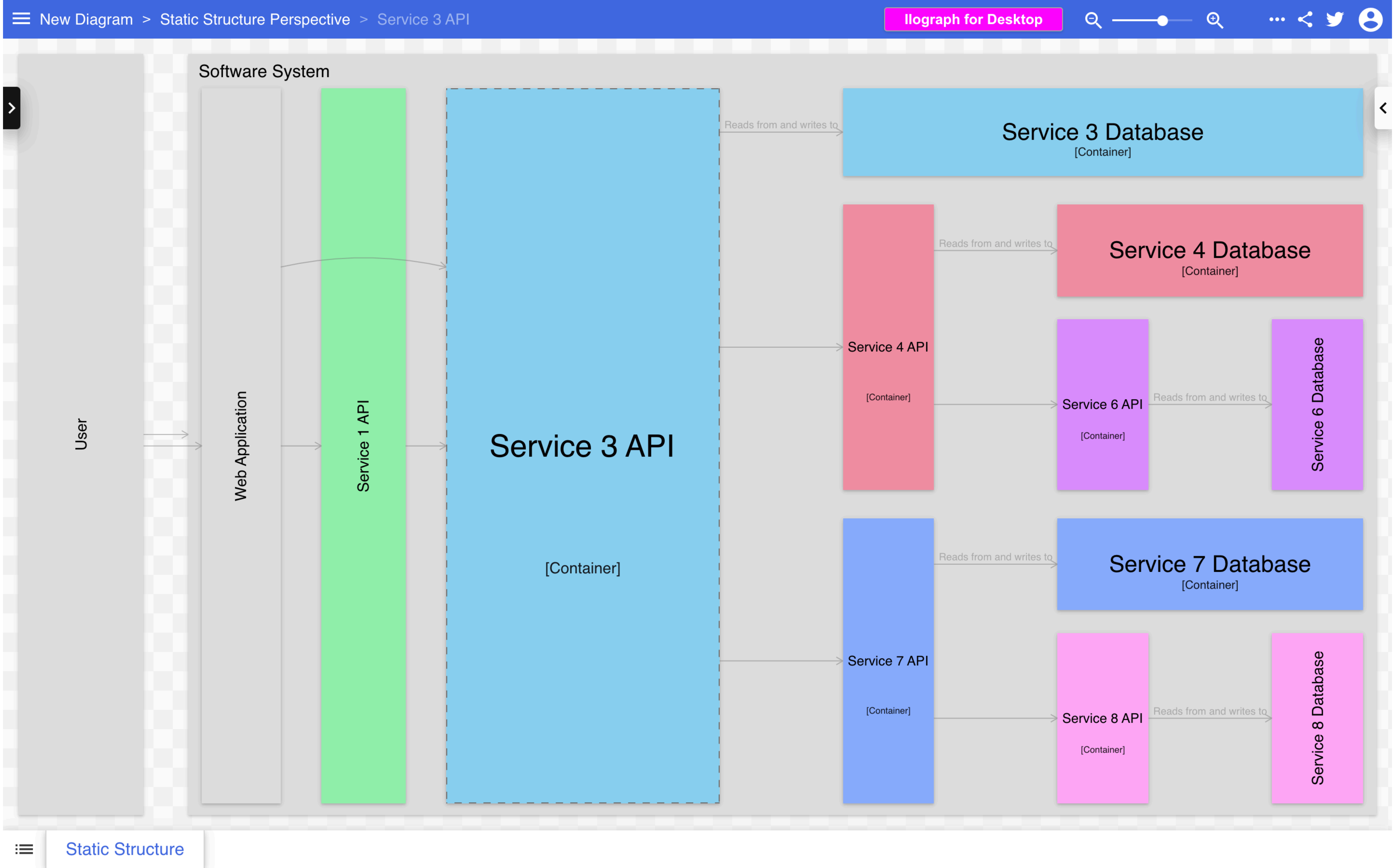

Service 8 API
[Container]
Service 8 Database
[Container]
Service 7 Database
[Container]
Service 6 Database
[Container]
Service 4 Database
[Container]
Service 7 API
[Container]
Service 3 Database
[Container]
Service 6 API
[Container]
Service 4 API
[Container]
Service 3 API
[Container]
Service 5 Databa
[Container]
Service 5 API
[Container]
Service 2 API
[Container]
Service 1 API
[Container]
Web Application
[Container]
User
[Person]
Service 2 Database
[Container]

Service 8 API
[Container]
Service 8 Database
[Container]
Service 7 Database
[Container]
Service 6 Database
[Container]
Service 4 Database
[Container]
Service 6 API
[Container]
Service 7 API
[Container]
Service 3 Database
[Container]
Service 4 API
[Container]
Service 3 API
[Container]
Service 5 Databa
[Container]
Service 3 API
from Software System [Software System]
[Container]
Element
Container
Service 3
Service API
Service 5 API
[Container]
Service 1 API
[Container]
Web Application
[Container]
User
[Person]
Service 2 Database
[Container]


New Diagram > Static Structure Perspective
Ilograph for Desktop
Software System
User
[Person]
Web Application
Service 1 API
Service 3 API
[Container]
Reads from and writes to
Service 3 Database
[Container]
Reads from and writes to
Service 4 Database
[Container]
Service 4 API
[Container]
Service 6 API
[Container]
Reads from and writes to
Service 6 Database
[Container]
Reads from and writes to
Service 7 Database
[Container]
Service 7 API
[Container]
Service 8 API
[Container]
Reads from and writes to
Service 8 Database
[Container]
Service 2 API
[Container]
Service 5 API
[Container]
Reads from and writes to
Service 5 Database
[Container]
Reads from and writes to
Service 2 Database
[Container]
Reads from and writes to
Service 1 Database
[Container]
Static Structure


New Diagram > Static Structure Perspective > Service 1 API
Ilograph for Desktop
Software System
User
Web Application
Service 1 API
[Container]
Service 3 API
Service 2 API
Reads from and writes to
Service 3 Database
[Container]
Service 4 API
Reads from and writes to
Service 4 Database
[Container]
Service 6 API
Reads from and writes to
Service 6 Database
Service 7 API
Reads from and writes to
Service 7 Database
[Container]
Service 8 API
Reads from and writes to
Service 8 Database
Service 5 API
[Container]
Reads from and writes to
Service 5 Database
[Container]
Reads from and writes to
Service 2 Database
[Container]
Reads from and writes to
Service 1 Database
[Container]
Static Structure


New Diagram > Static Structure Perspective > Service 2 API
Ilograph for Desktop
Software System
User
Web Application
Service 1 API
Service 2 API
[Container]
Service 4 API
[Container]
Reads from and writes to
Service 4 Database
[Container]
Service 6 API
[Container]
Reads from and writes to
Service 6 Database
Service 5 API
[Container]
Reads from and writes to
Service 5 Database
[Container]
Reads from and writes to
Service 2 Database
[Container]
Static Structure


New Diagram > Static Structure Perspective > Service 3 API
Ilograph for Desktop
Software System
User
Web Application
Service 1 API
Service 3 API
[Container]
Reads from and writes to
Service 3 Database
[Container]
Service 4 API
[Container]
Reads from and writes to
Service 4 Database
[Container]
Service 6 API
[Container]
Reads from and writes to
Service 6 Database
Service 7 API
[Container]
Reads from and writes to
Service 7 Database
[Container]
Service 8 API
[Container]
Reads from and writes to
Service 8 Database
Static Structure

A final note…

Level 1	Level 2	Level 3	Level 4	Level 5
Initial No software architecture diagrams.	**Ad hoc** Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	**Defined** Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	**Modelled** Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	**Optimising** - Model elements are shared between teams. - Centralised system landscape views are generated by aggregating decentralised team-based models. - Model elements are reverse-engineered from source code, deployment environment, logs, etc. - Alternative visualisations are used for different use cases (e.g. communication vs exploration). - Models are used as queryable datasets. ...

Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
Initial	**Ad hoc**	**Defined**	**Modelled**	**Optimising**
No software architecture diagrams.	Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	- Model elements are shared between teams. - Centralised system landscape views are generated by aggregating decentralised team-based models. - Model elements are reverse-engineered from source code, deployment environment, logs, etc. - Alternative visualisations are used for different use cases (e.g. communication vs exploration). - Models are used as queryable datasets. ...

Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
Initial No software architecture diagrams.	**Ad hoc** Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	**Defined** Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	**Modelled** Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	**Optimising** - Model elements are shared between teams. - Centralised system landscape views are generated by aggregating decentralised team-based models. - Model elements are reverse-engineered from source code, deployment environment, logs, etc. - Alternative visualisations are used for different use cases (e.g. communication vs exploration). - Models are used as queryable datasets. ...

Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
Initial No software architecture diagrams.	**Ad hoc** Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	**Defined** Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	**Modelled** Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	**Optimising** - Model elements are shared between teams. - Centralised system landscape views are generated by aggregating decentralised team-based models. - Model elements are reverse-engineered from source code, deployment environment, logs, etc. - Alternative visualisations are used for different use cases (e.g. communication vs exploration). - Models are used as queryable datasets. ...

Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
Initial No software architecture diagrams.	**Ad hoc** Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	**Defined** Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	**Modelled** Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	**Optimising** - Model elements are shared between teams. - Centralised system landscape views are generated by aggregating decentralised team-based models. - Model elements are reverse-engineered from source code, deployment environment, logs, etc. - Alternative visualisations are used for different use cases (e.g. communication vs exploration). - Models are used as queryable datasets. ...

Software architecture diagramming maturity model

Documenting software architecture

Enough detail to start **exploring**

Working software

over

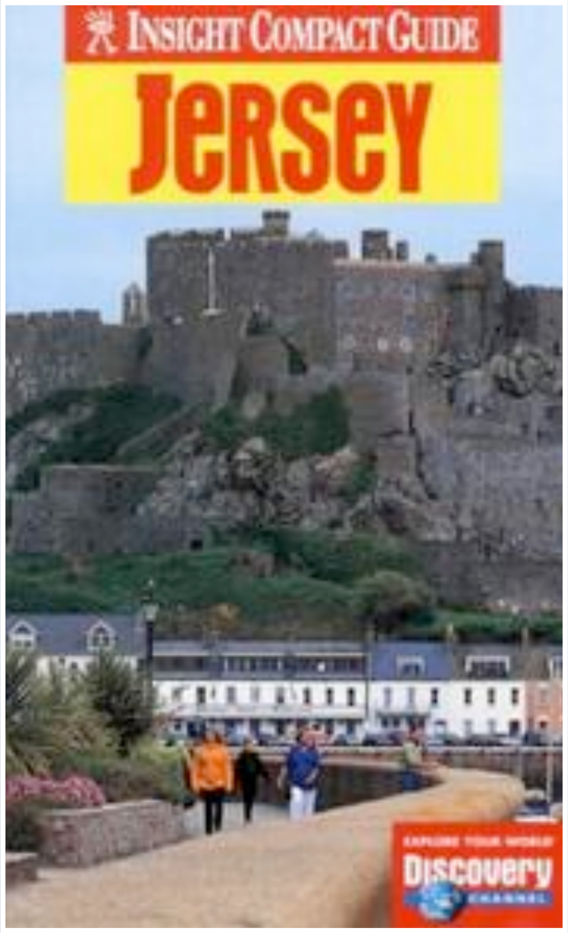
comprehensive documentation

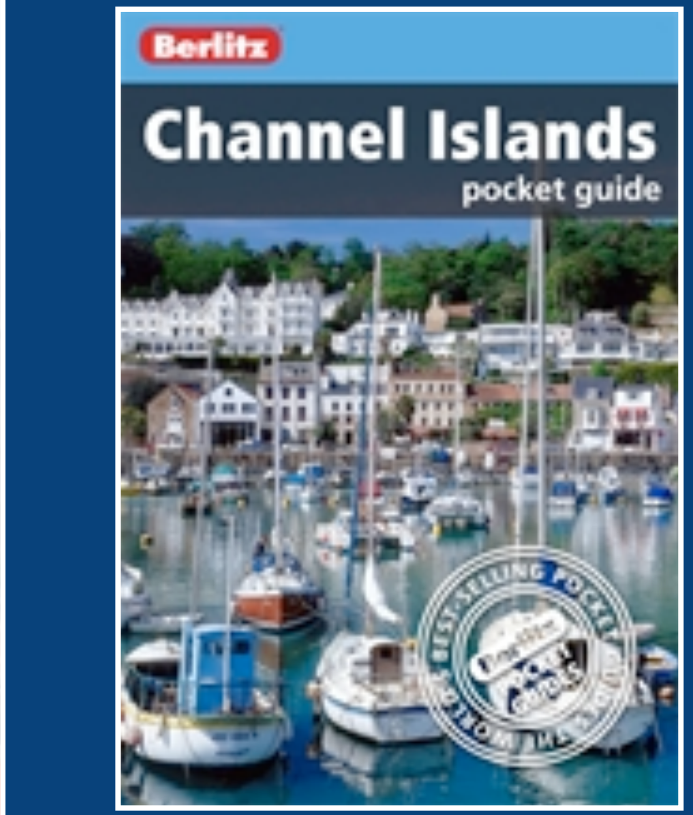
Manifesto for Agile Software Development

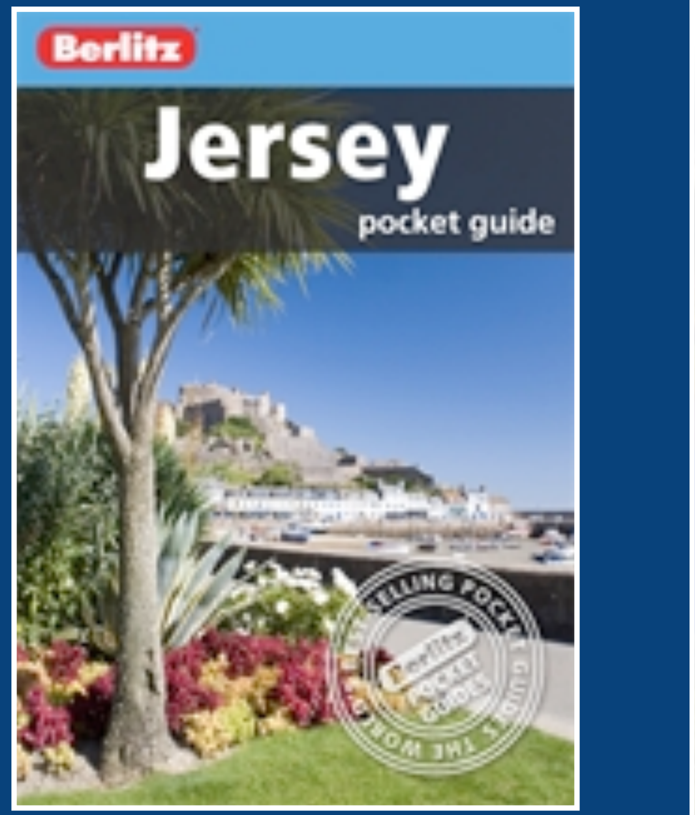
The code doesn’t tell the whole story

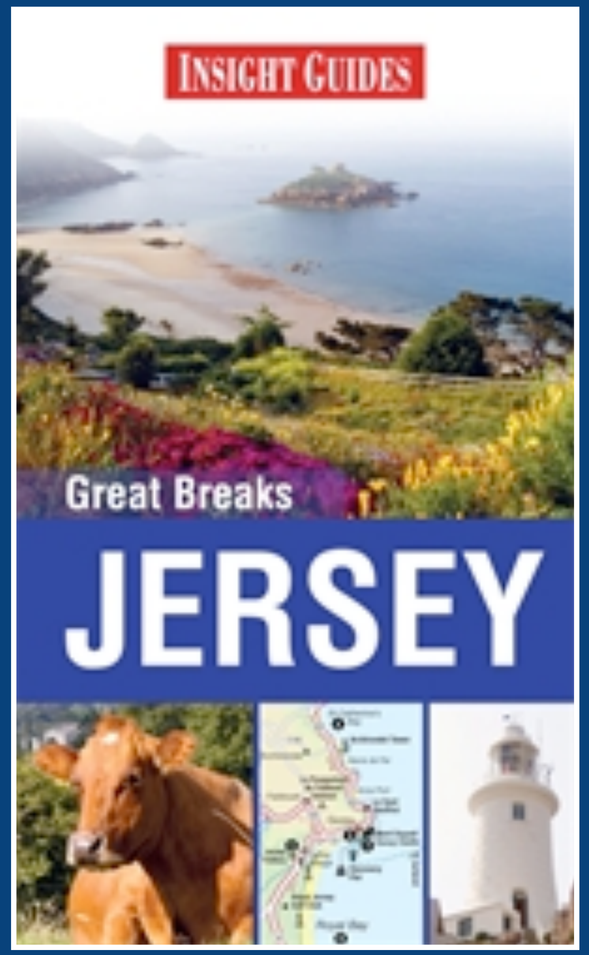
Software Architecture Document

Useful information spread across hundreds of pages; rarely read or updated



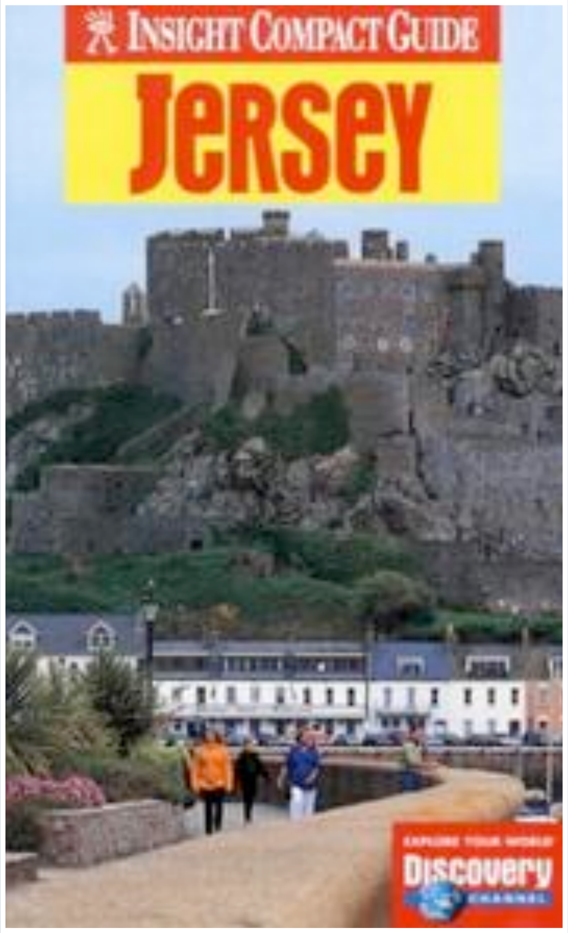


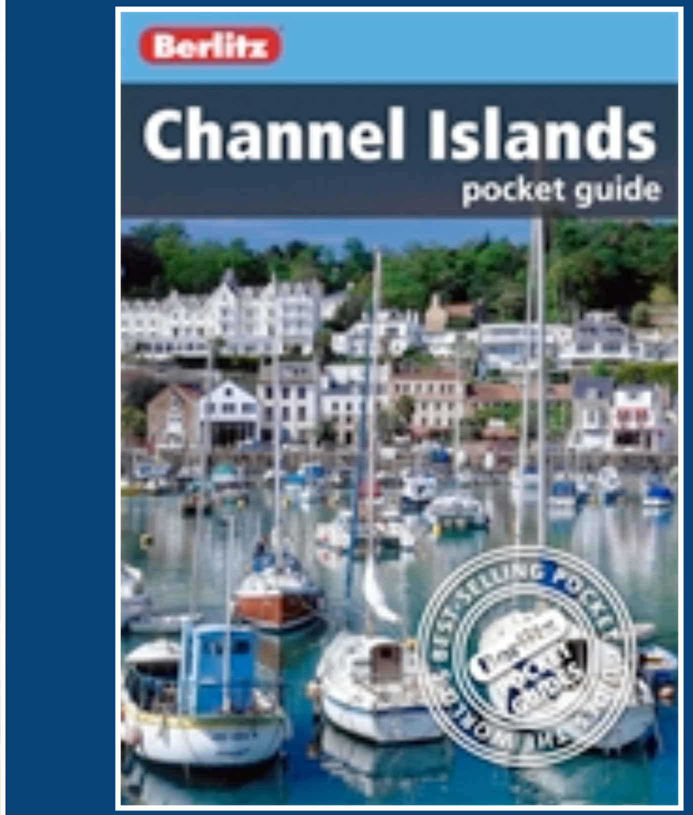


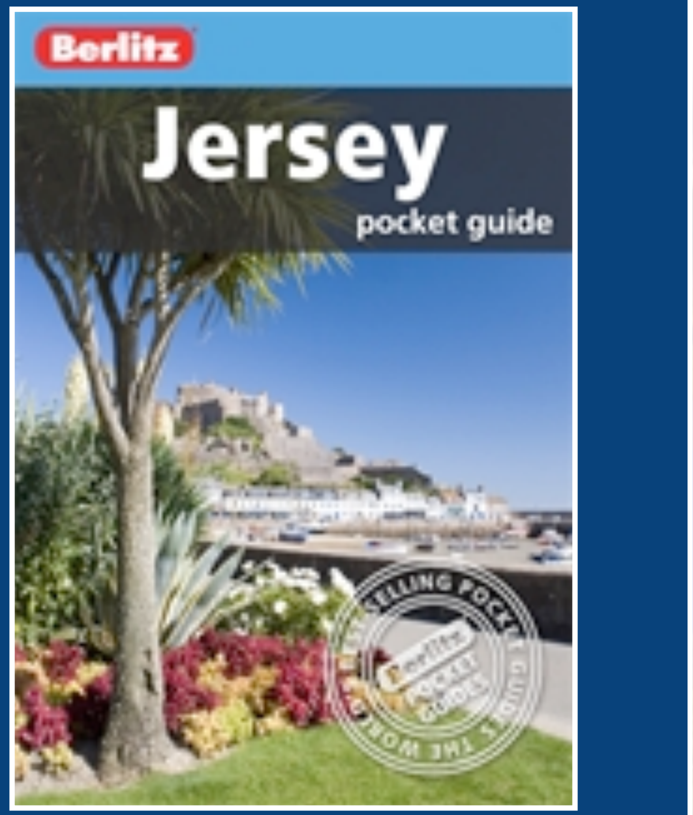


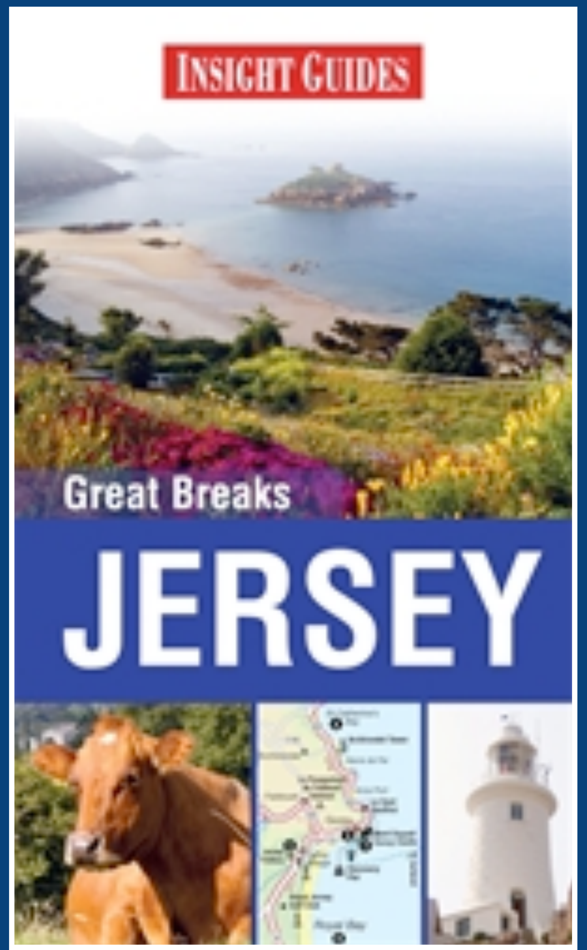
Travel Guidebook

(maps, points of interest, sights, itineraries, history, culture, practical information, etc)



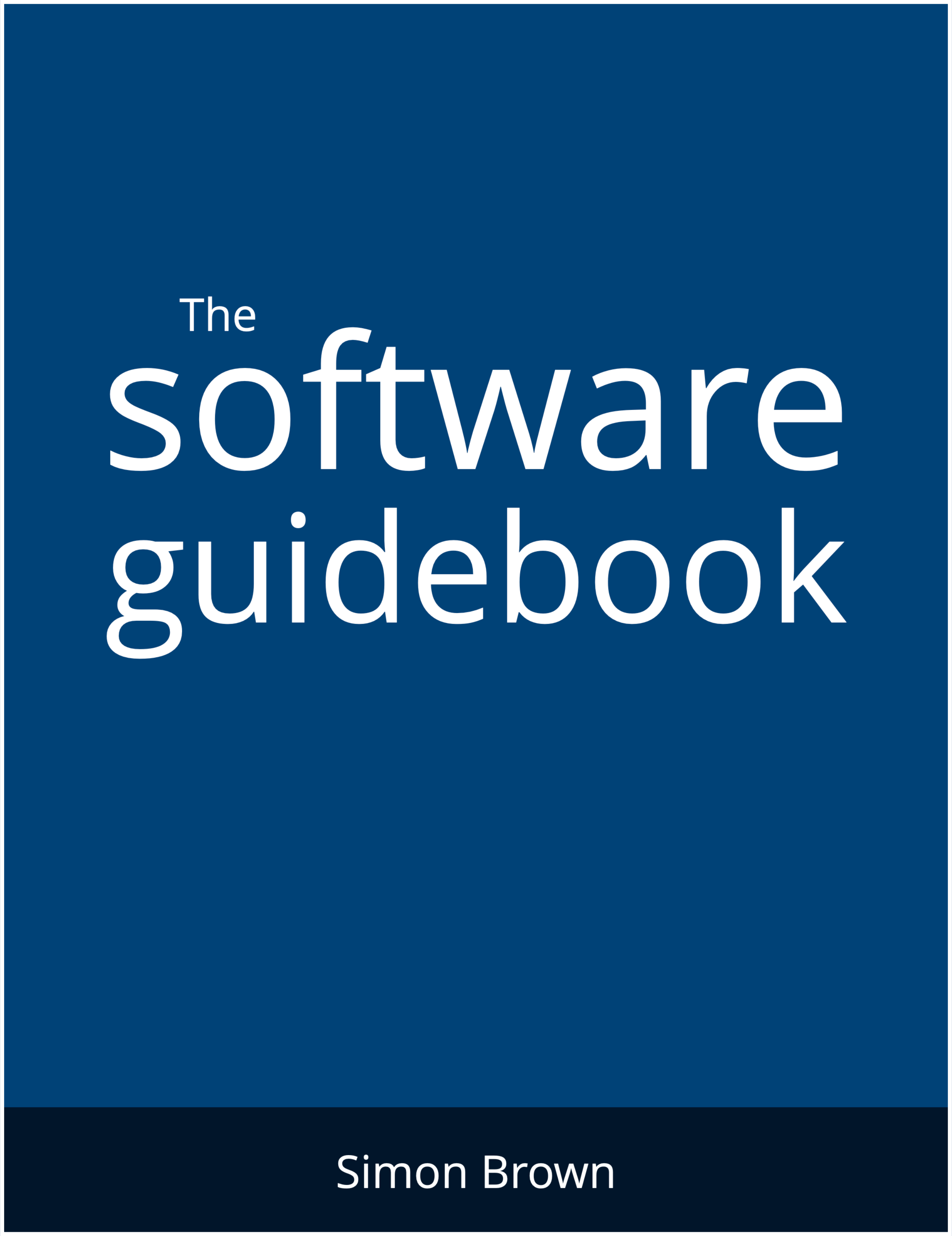






Software Guidebook

(maps, points of interest, sights, itineraries, history, culture, practical information, etc)



The
software
guidebook

Simon Brown

https://leanpub.com/documenting-software-architecture/c/free

The scope is a single
software system

Describe what you

can’t get from the code

Documentation should be **constantly evolving**

Context

A system context diagram, plus some narrative text to "set the scene".

Functional Overview

An overview of the software system; perhaps including wireframes, UI mockups, screenshots, workflow diagrams, business process diagrams, etc.

Quality Attributes

A list of the quality attributes (non-functional requirements; e.g. performance, scalability, security, etc).

Constraints

A list of the environmental constraints (e.g. timescales, budget, technology, team size/skills, etc).

Principles

A list of the development and architecture principles (e.g. coding conventions, separation of concerns, patterns, etc).

Software Architecture

A description of the software architecture, including static structure (e.g. containers and components) and dynamic/ runtime behaviour.

Code

A description of important or complicated component implementation details, patterns, frameworks, etc.

Data

Data models, entity relationship diagrams, security, data volumes, archiving strategies, backup strategies, etc.

This is a **starting point**; add and remove sections as necessary.

Infrastructure Architecture

A description of the infrastructure available to run the software system.

Deployment

The mapping of software (e.g. containers) to infrastructure.

Development Environment

A description of how a new developer gets started.

Operation and Support

An overview of how the software system is operated, supported, monitored, etc.

Decision Log

A log of the major decisions made; e.g. as free format text or a collection of "Architecture Decision Records".

arc42
Overview
Docs
FAQ
Examples
Training
Download
About

arc42 Template Overview

arc42 is a template for architecture communication and documentation.

Photo credit: **unsplash**

arc42 answers the following two questions in a pragmatic way, but can be tailored to your specific needs:

- *What* should we document/communicate about our architecture?
- *How* should we document/communicate?

Introduction and Goals
Required Functions
Quality Goals
Stakeholder

Goal	Description
...	...

Who?	Expectation?
...	...

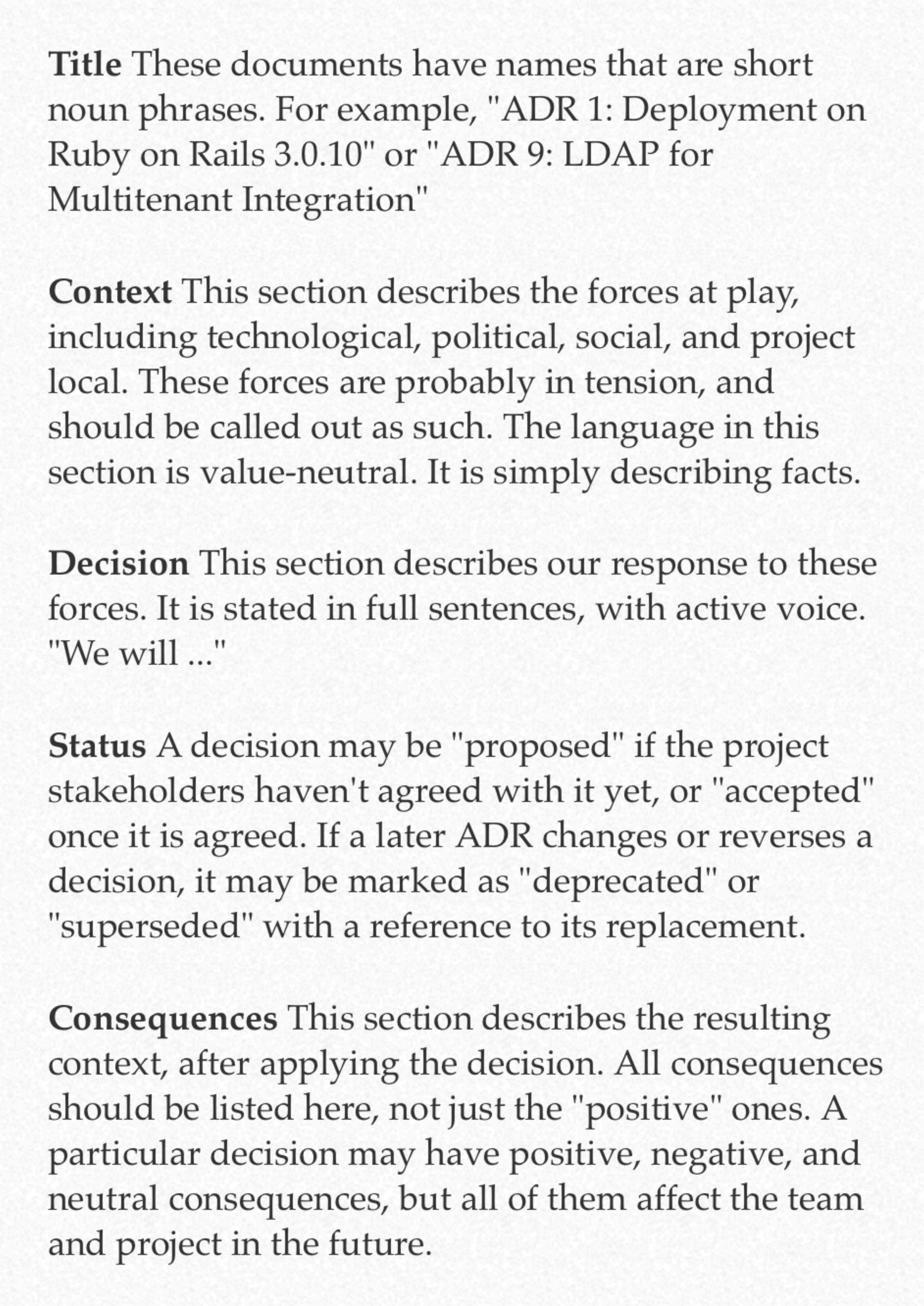
1. Introduction and Goals

Short description of the **requirements**, driving forces, extract (or abstract) of requirements. Top three (max five) **quality goals** for the architecture which have highest priority for the major stakeholders. A table of important **stakeholders** with their expectation regarding architecture.

“Architecture Decision Record”

A short description of an architecturally significant decision

http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions (Michael Nygard)



Title These documents have names that are short noun phrases. For example, "ADR 1: Deployment on Ruby on Rails 3.0.10" or "ADR 9: LDAP for Multitenant Integration"

Context This section describes the forces at play, including technological, political, social, and project local. These forces are probably in tension, and should be called out as such. The language in this section is value-neutral. It is simply describing facts.

Decision This section describes our response to these forces. It is stated in full sentences, with active voice. "We will ..."

Status A decision may be "proposed" if the project stakeholders haven't agreed with it yet, or "accepted" once it is agreed. If a later ADR changes or reverses a decision, it may be marked as "deprecated" or "superseded" with a reference to its replacement.

Consequences This section describes the resulting context, after applying the decision. All consequences should be listed here, not just the "positive" ones. A particular decision may have positive, negative, and neutral consequences, but all of them affect the team and project in the future.

Documentation format?

Microsoft Word, Microsoft SharePoint,
Atlassian Confluence, Markdown or AsciiDoc, etc

How long?

Something I can read in 1-2 hours;
a good starting point for exploring the code

How do you keep software architecture documentation up to date?

C4 model diagrams

+

software guidebook/arc42

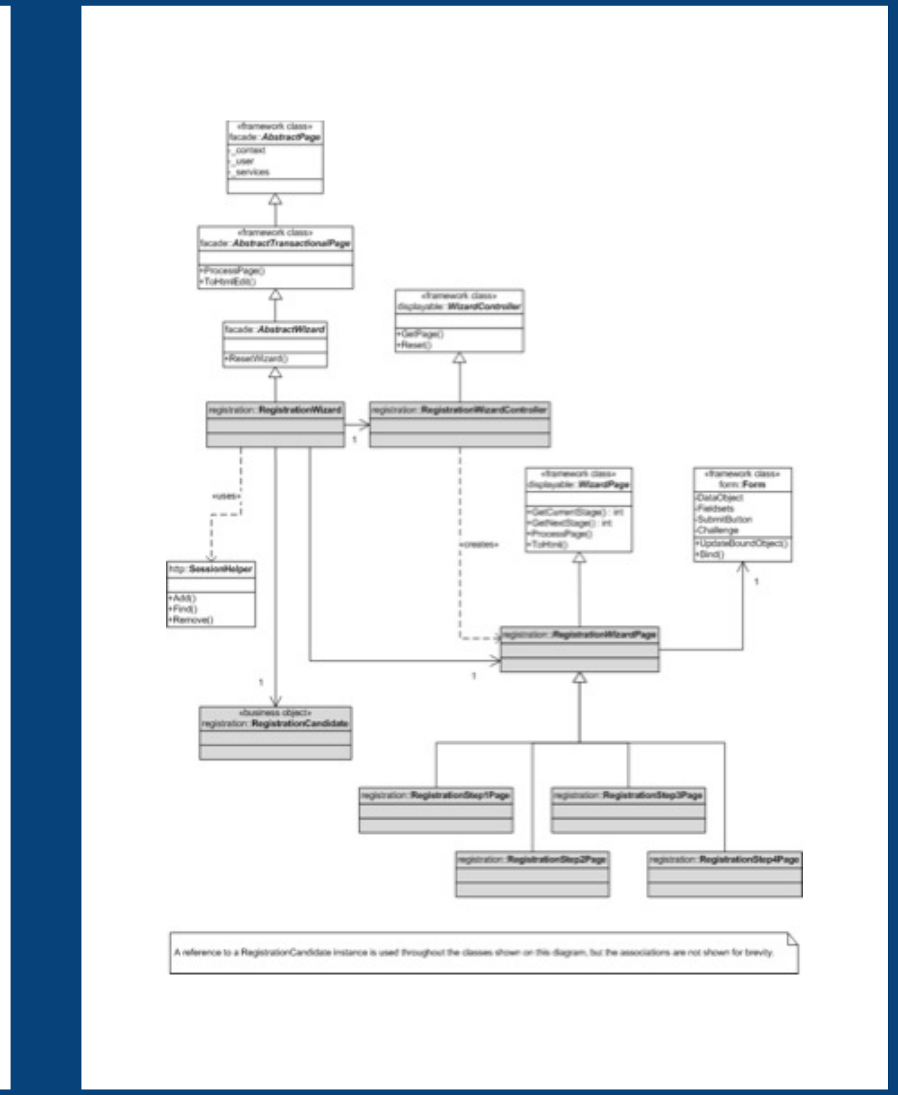
+

architecture decision records

Software architecture in practice

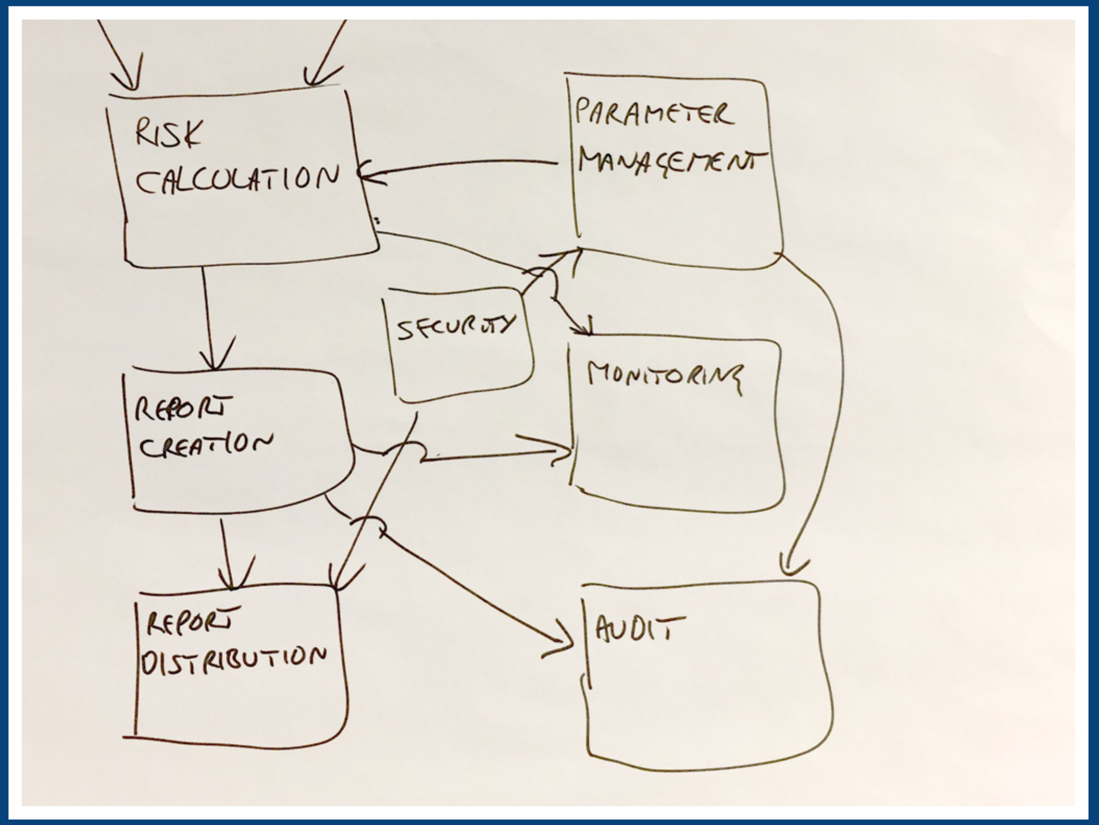
Big design up front

Software Architecture Document



A reference to a RegistrationCandidate instance is used throughout the classes shown on this diagram, but the associations are not shown for brevity.

vs



No design up front

> Big design up front is dumb. Doing no design up front is even dumber.

Dave Thomas

Evolutionary architecture

How much **up front design** should you do?

0%

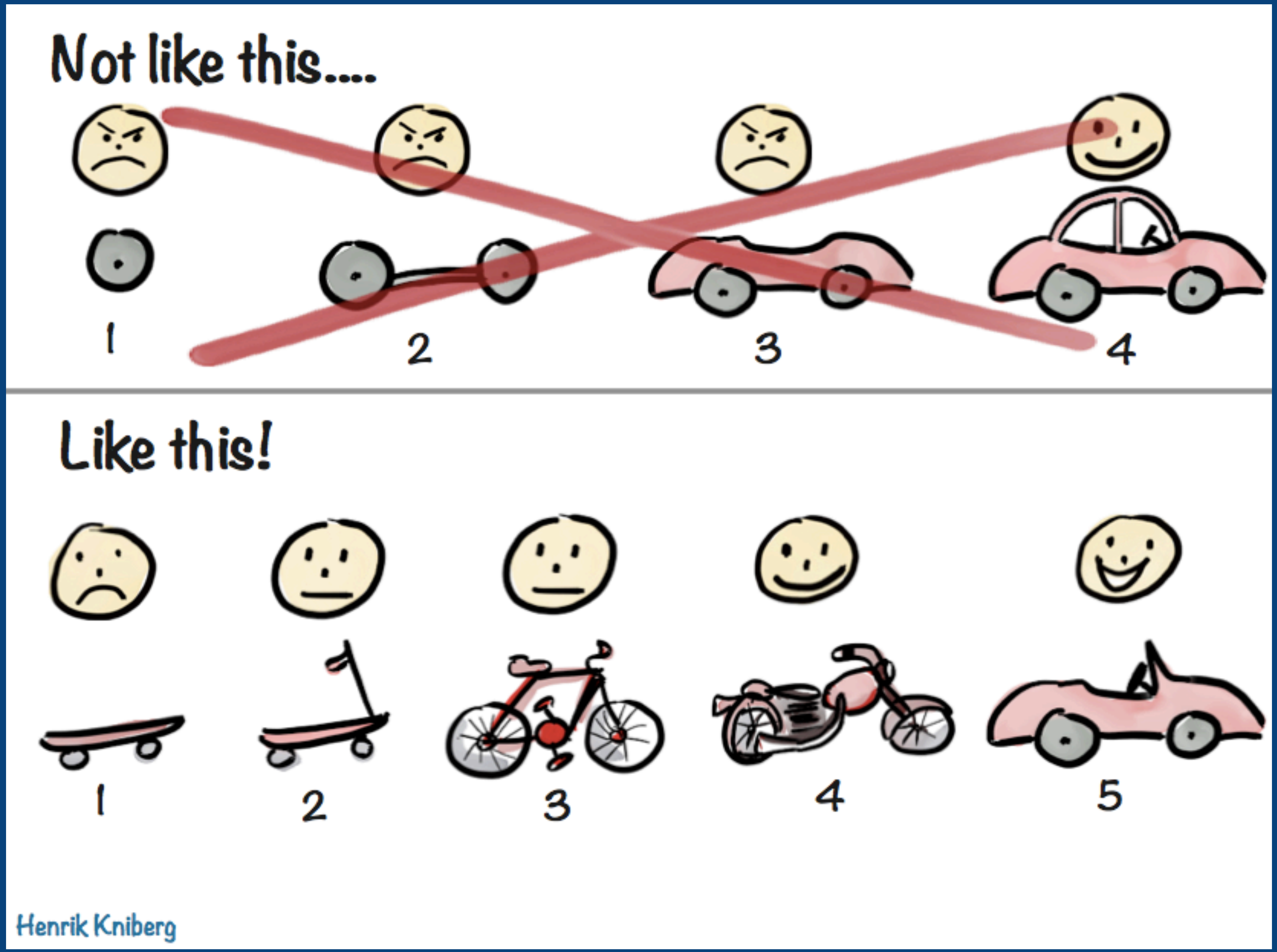
100%

“it depends”

Sometimes requirements are known, and sometimes they aren’t

(enterprise software development vs product companies and startups) “just enough”

Up front design is not necessarily about creating a perfect end-state or complete architecture



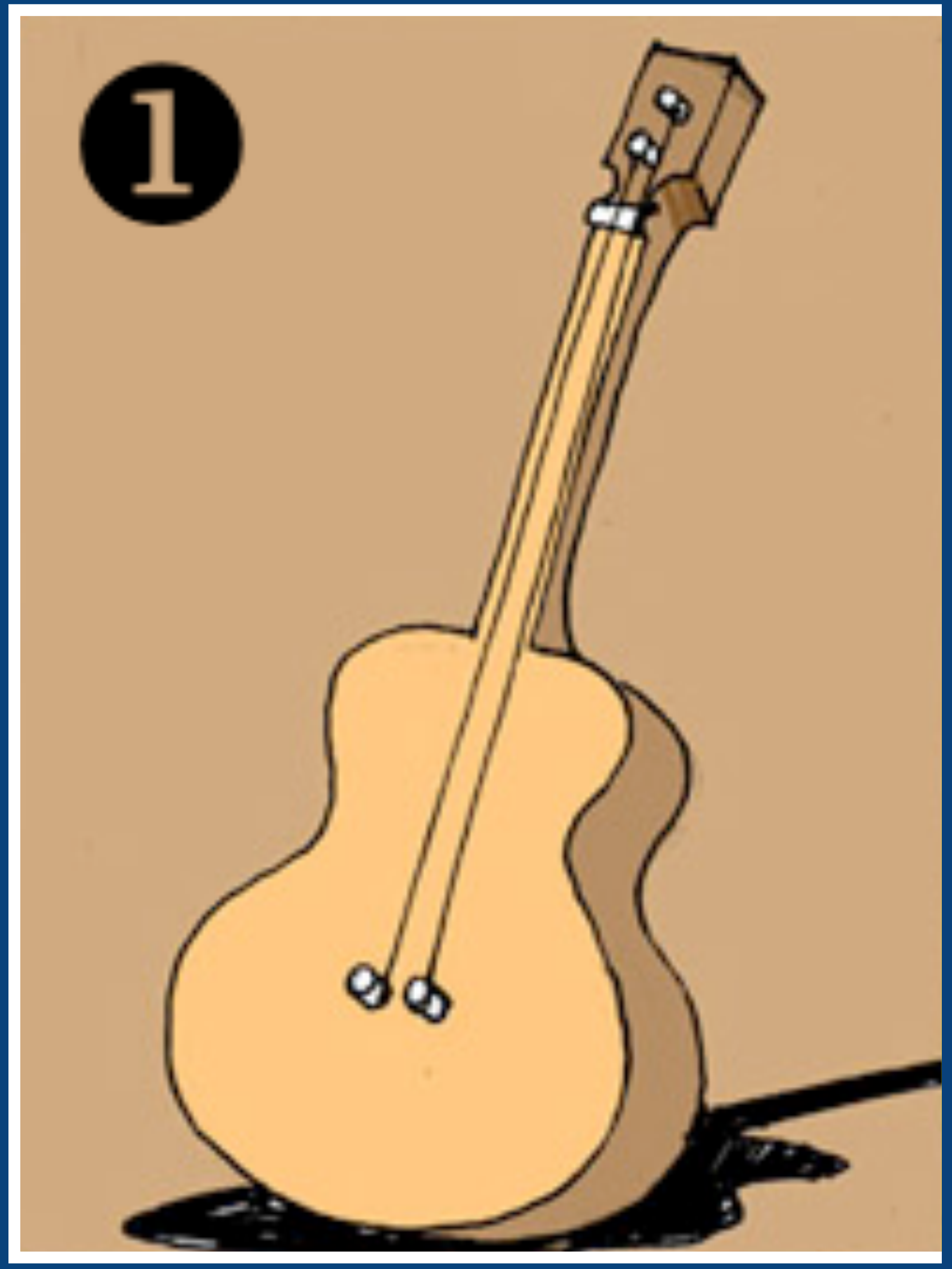
Iteration (via prototyping and experimentation) is great for product design but...

you don’t just “build the car”

❶
❷
❸
❹
Joshua Kerievsky
industrial logic

Evolutionary Design

Beginning With A Primitive Whole



Evolutionary Design

Beginning With A Primitive Whole

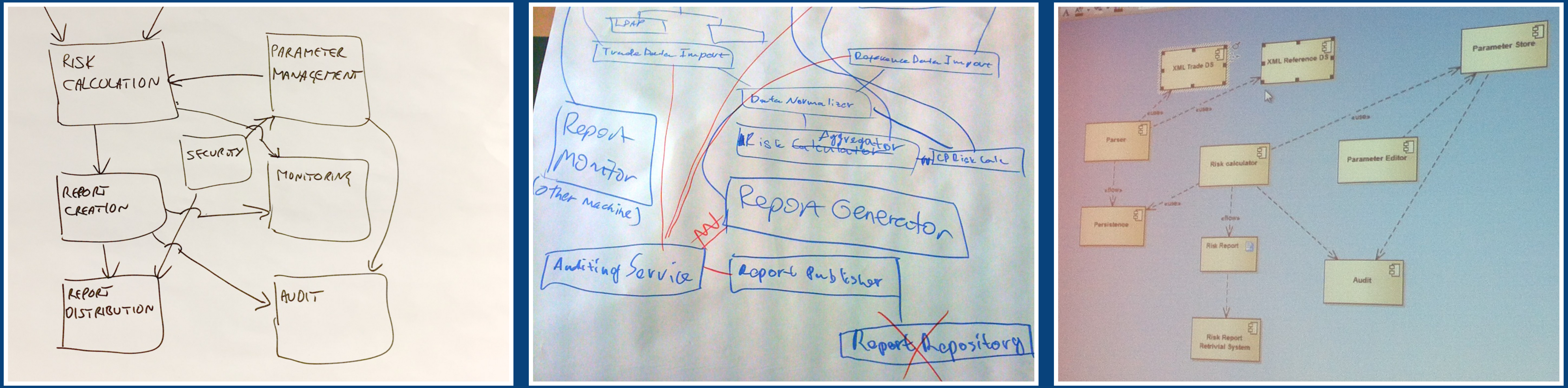
We’re not trying to make every decision

> I think there is a role for a broad starting point architecture. Such things as stating early on how to layer the application, how you'll interact with the database (if you need one), what approach to use to handle the web server.

Martin Fowler

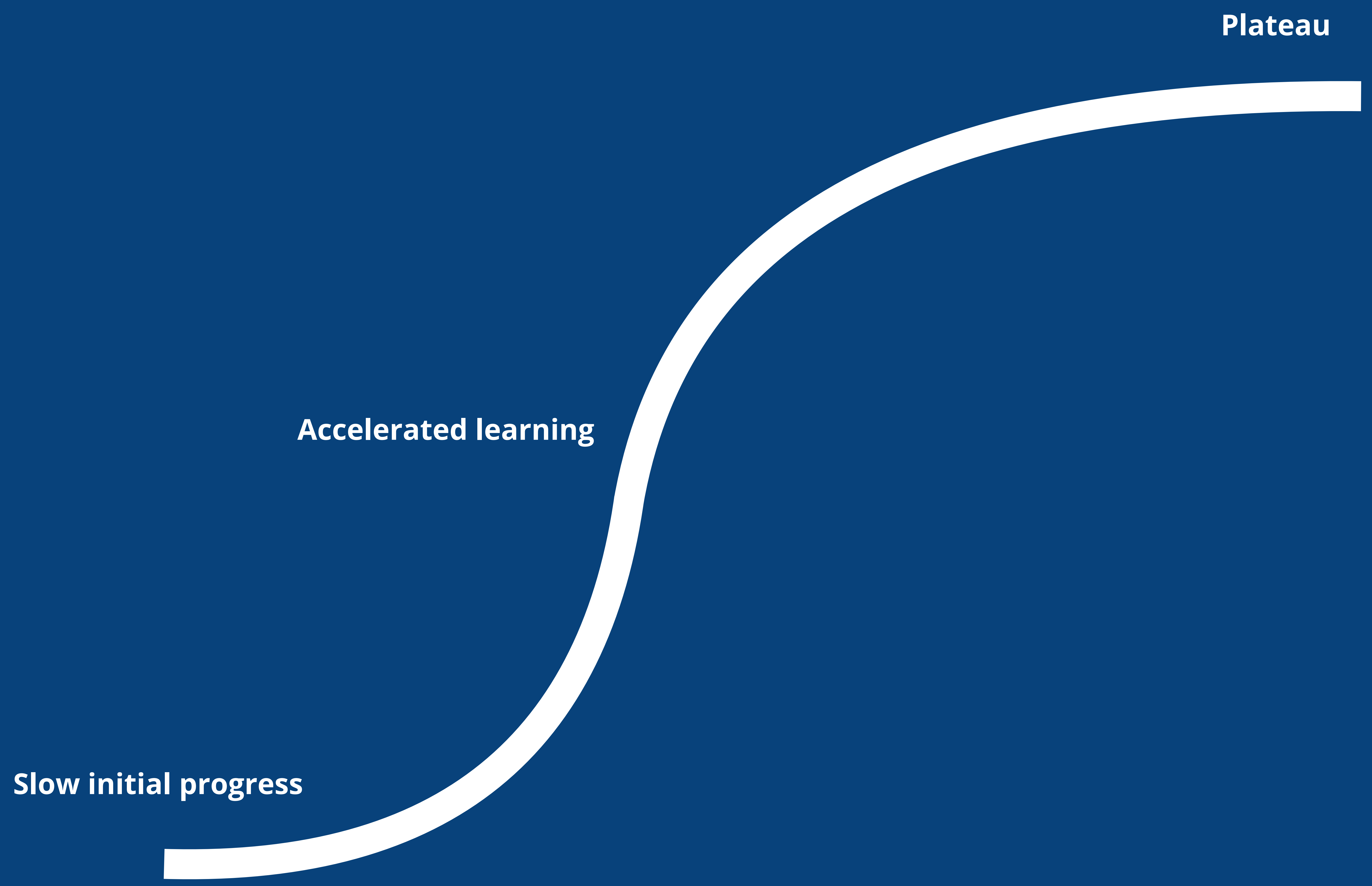
https://martinfowler.com/articles/designDead.html

A **starting point** adds value



If you don’t **engage** in the problem, you end up with a very simplified and superficial view of the solution

Part of the design activity is about discovering “unknown unknowns”



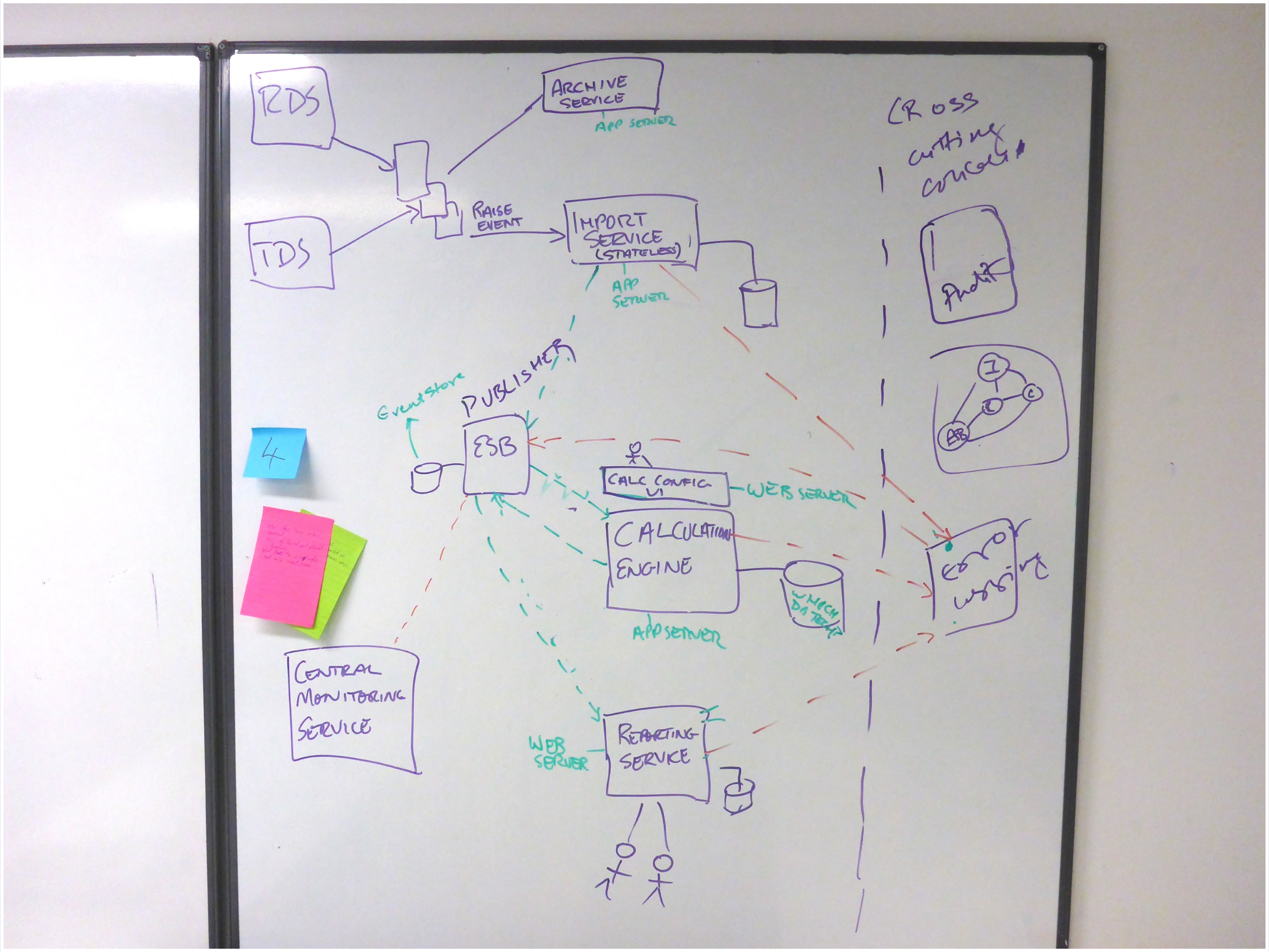
Plateau

Accelerated learning

Slow initial progress

The typical s-curve of learning

1. Is that what we're going to **build**?



2. Is it going to **work**?

Diagrams are a visual checklist for design decisions

System Context diagram

What is the scope of the software system we’re building?

Who is using it? What are they doing?

What system integrations does it need to support?

Financial Risk System: Context Diagram

USER
INTERACTION
PRE-EXISTING SYSTEM
PROPOSED SYSTEM
Business Users
(PERSON)
A CUSTOMER THAT ACCESSES RISK REPORTS
Business Users with Configuration Access
(PERSON)
A USER THAT CAN MODIFY RISK CALCULATION PARAMS
DELIVERS LINK TO REPORT TO
ACCESS REPORTS USING
CHANGES CONFIGS FOR
Financial Risk System
(SOFTWARE SYSTEM)
TAKES TRADE DATA AND REFERENCE DATA AND CALCULATES RISK; GENERATES RISK REPORT AND MAKES IT AVAILABLE TO BUSINESS USERS
Email
(SOFTWARE SYSTEM)
SENDS EMAILS TO BUSINESS USERS
Auth
(SOFTWARE SYSTEM)
AUTHENTICATES BUSINESS USERS
SENDS REPORT LINKS THROUGH
AUTHENTICATES AND CHECKS PERMISSIONS USING
MONITORED BY
USES DATA FROM
USE DATA FROM
Central Monitoring System
(SOFTWARE SYSTEM)
MONITORS HEALTH OF ALL SERVICES
Trade Data System
(SOFTWARE SYSTEM)
PROVIDES DATA ABOUT ALL TRADES BY THE BANK
Reference Data System
(SOFTWARE SYSTEM)
PROVIDES REFERENCE DATA ABOUT COUNTERPARTIES

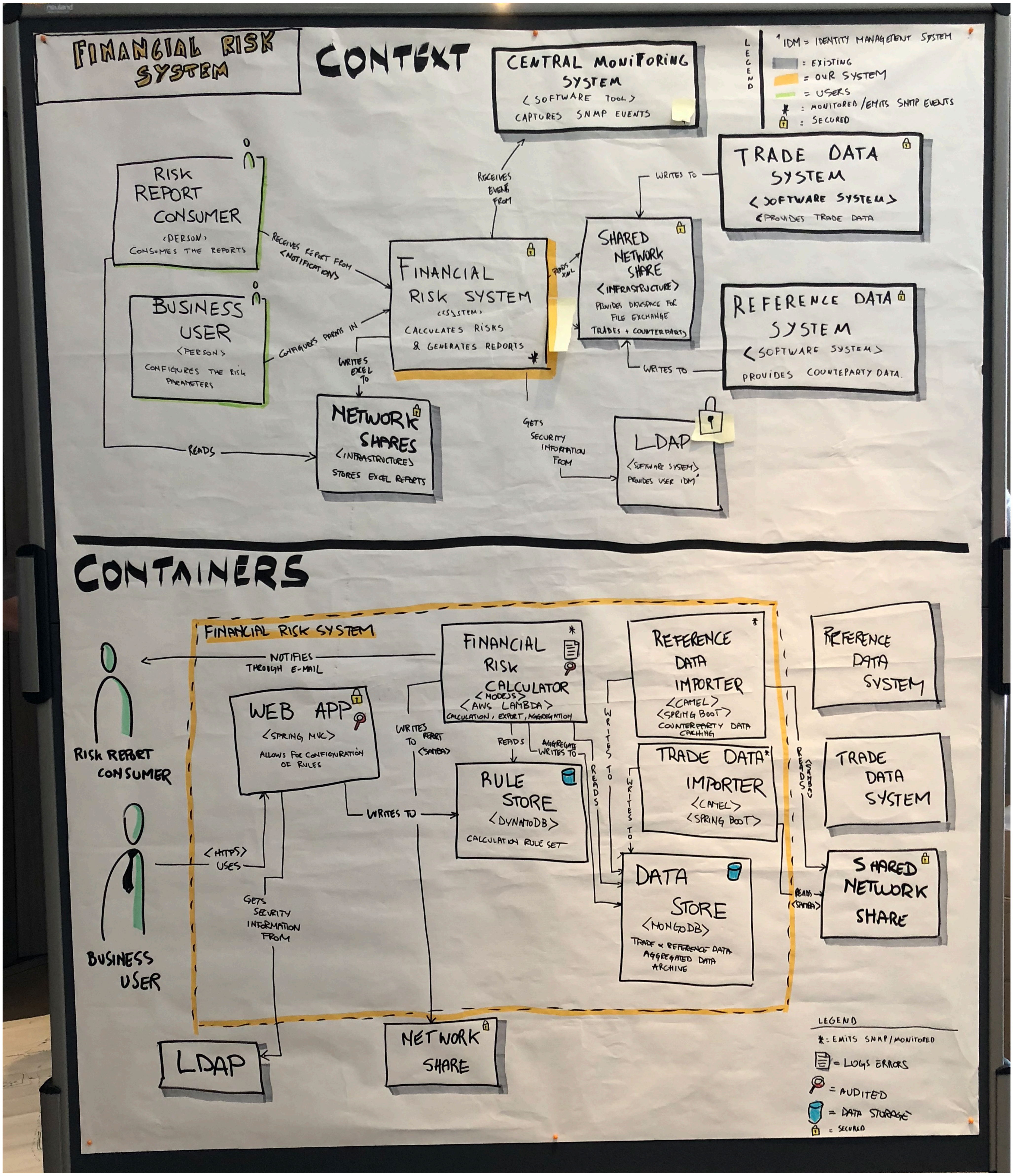
Container diagram

What are the major technology building blocks?

What are their responsibilities?

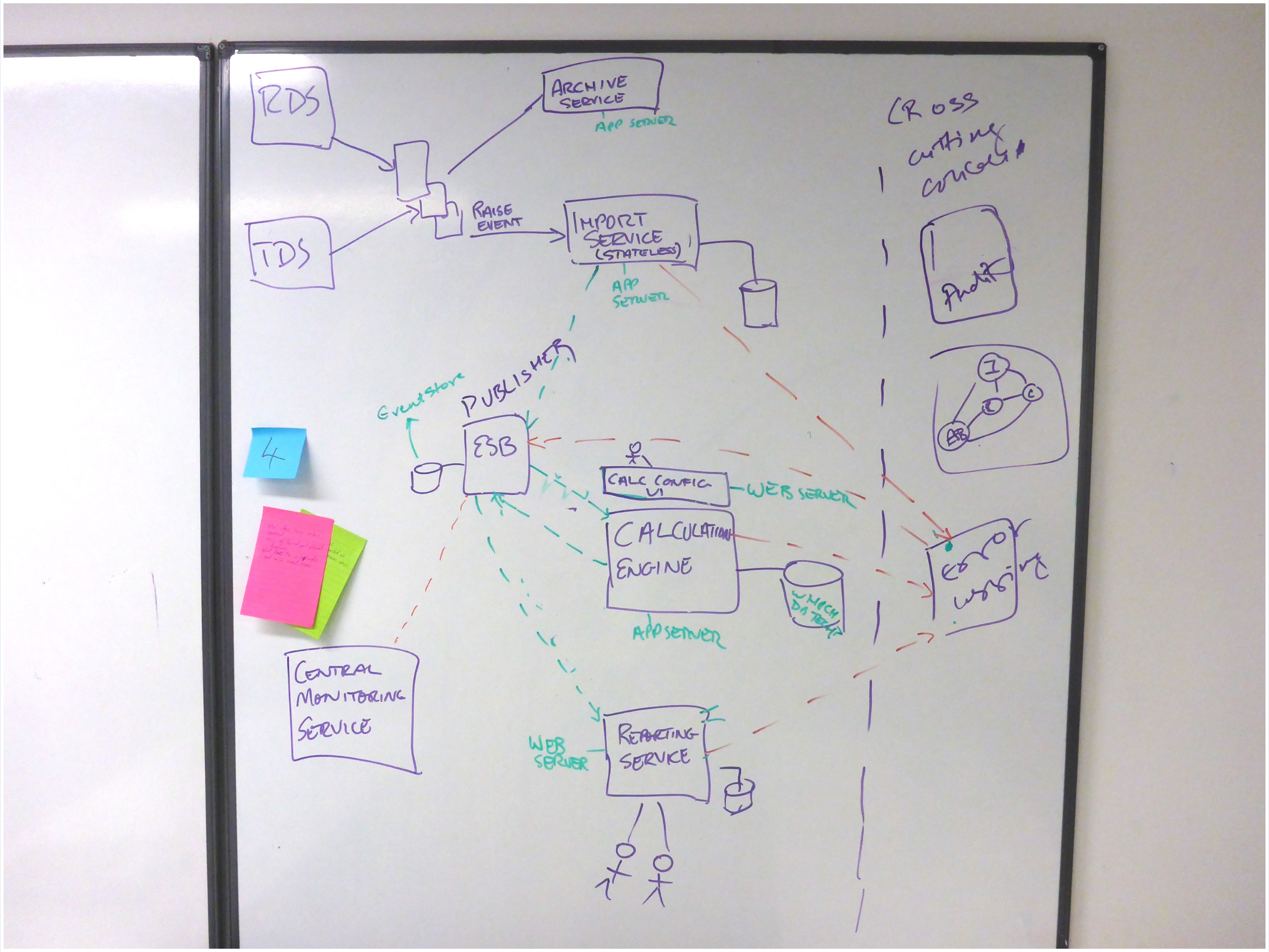
How do they communicate?

Financial Risk System: Container Diagram
USER
INTERACTION
CONTAINER
RISK SERVICE BOUNDARY
INDICATES THAT A COMPONENT IS MONITORED BY THE CENTRAL MONITORING SYSTEM
DATA STORE
Business User (PERSON)
A CUSTOMER THAT ACCESSES REPORTS
Business User with Config Access (PERSON)
A USER THAT CAN MODIFY CALCULATION PARAMS
ACCESS REPORTS USING
REQUESTS WEB APP (HTTPS)
REQUESTS WEB APP (HTTPS)
MODIFY CONFIGS USING
Risk Report Web App (JAVASCRIPT REACT)
PROVIDES USER INTERFACE TO ACCESS RISKS REPORTS
Web Server (JAVA SPRING)
SERVES SPAS AND PROVIDES API ENDPOINTS TO THE WEB APPLICATIONS
Configuration Web App (JAVASCRIPT REACT)
PROVIDES A SUBSET OF USERS AN INTERFACE TO CONFIGURE RISK CALCULATION PARAMS
ACCESS REPORTS USING (REST JSON)
MODIFIES CONFIGS USING (REST JSON)
WRITES CONFIGS TO
RETRIEVES REPORTS FROM
AUTHENTICATES USERS USING (OAUTH)
Auth (SOFTWARE SYSTEM)
AUTHENTICATES USERS
Report Store (CIFS NETWORK STORAGE)
STORES GENERATED REPORTS, AND THE INPUT DATA AND CONFIG USED
Input Store (CIFS NETWORK STORAGE)
STORES INPUT TRADE AND REFERENCE DATA AND CALCULATION CONFIGS
Config Store (POSTGRES DB)
STORES CONFIG PARAMS FOR RISK CALCULATIONS
STORES REPORTS TO
TAKES INPUT FROM
STORE INPUT DATA TO
Trade Data System (SOFTWARE SYSTEM)
PROVIDES TRADE DATA
Email (SOFTWARE SYSTEM)
SEND EMAILS TO USERS
SEND LINK TO REPORT USING
"Risk Cruncher" (JAVA COMMANDLINE)
CALCULATES RISK DATA + GENERATES REPORTS AND UPLOADS IT TO REPORT STORE
SCHEDULES JOB FOR
Scheduler (JAVA COMMAND LINE)
SCHEDULES CALCULATION JOBS AND COPIES INPUT DATA TO THE INPUT STORE
COPIES DATA FROM
Reference Data System (SOFTWARE SYSTEM)
PROVIDES REFERENC DATA



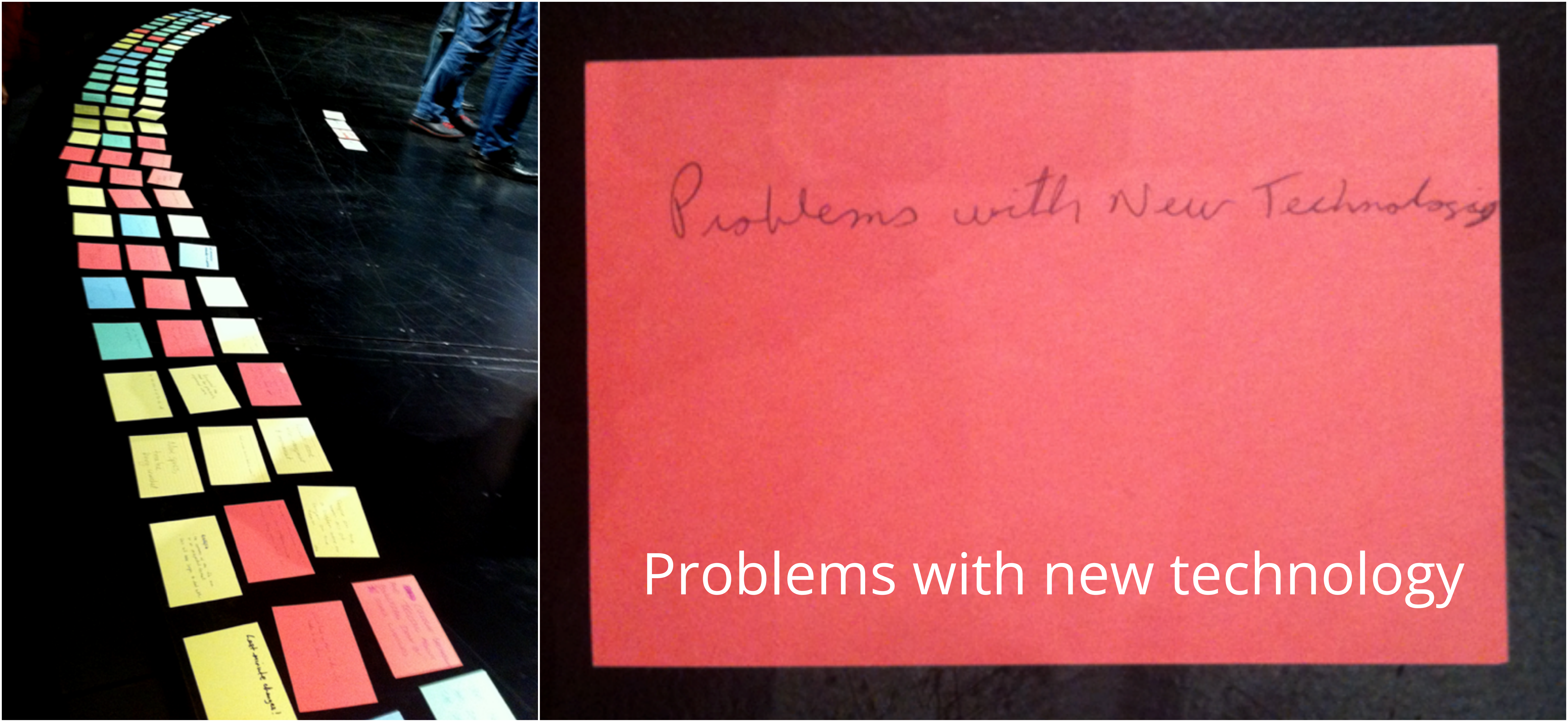
Understand the structure and create a shared vision

1. Is that what we're going to **build**?



2. Is it going to **work**?

Teams need to explicitly manage technical risk



Problems with new technology

An example timeline from “Beyond Retrospectives”
Linda Rising, GOTO Aarhus 2011

Identify and **mitigate**
your **highest priority risks**

Probability

Impact

	Low 1	Medium 2	High 3
Low 1	1	2	3
Medium 2	2	4	6
High 3	3	6	9

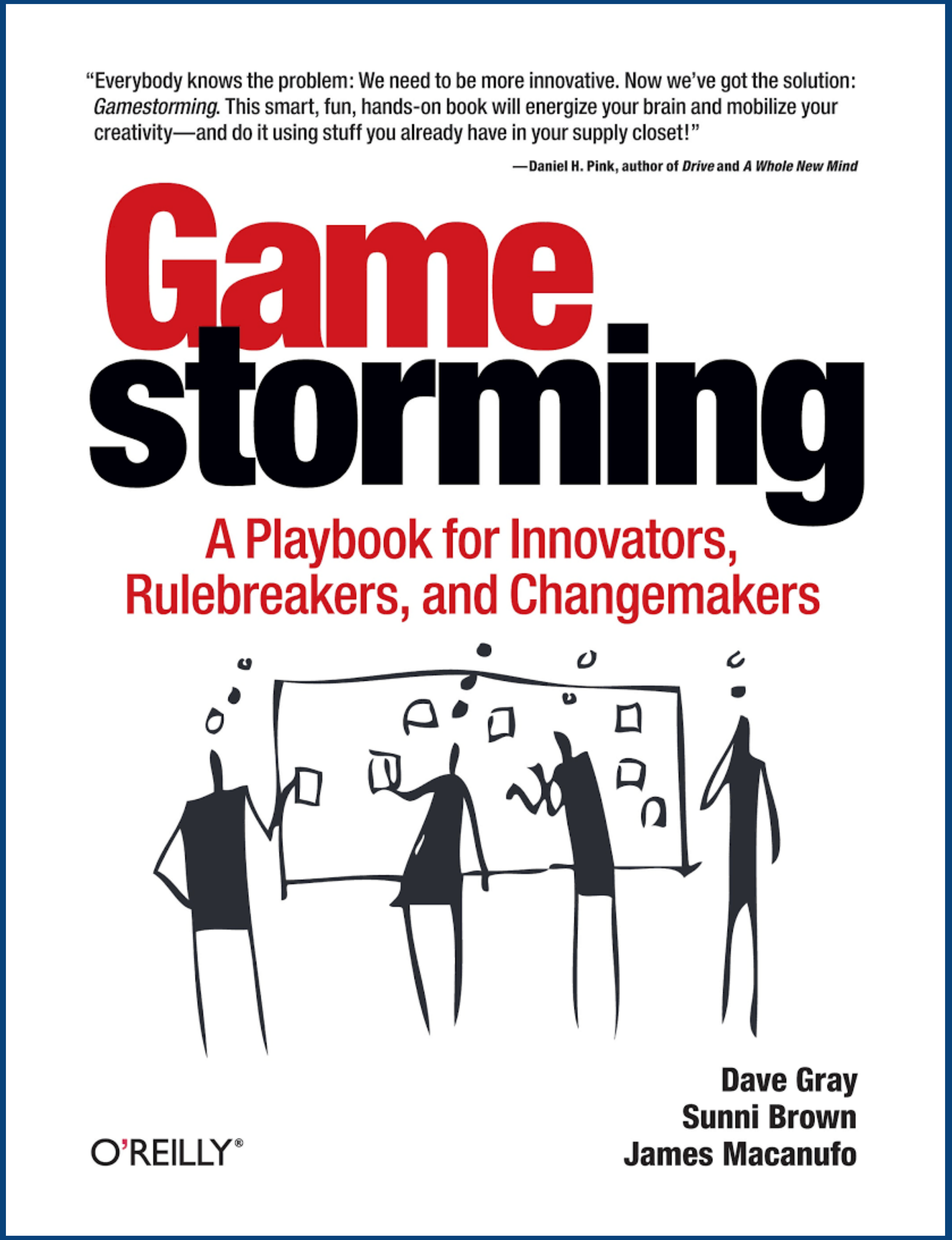
The software architecture role should own the technical risks

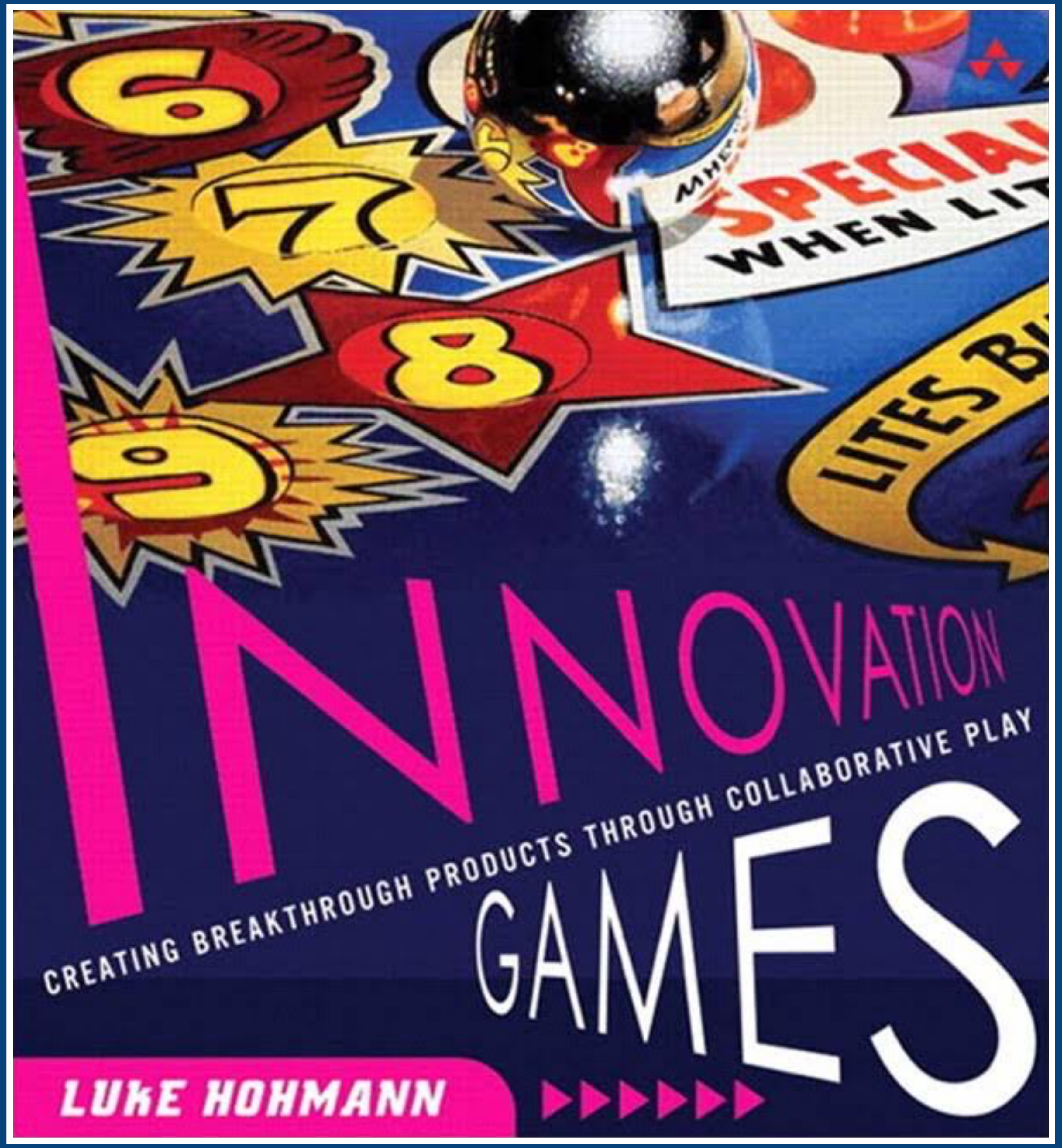
Architecturally significant?

costly to change | complicated | new

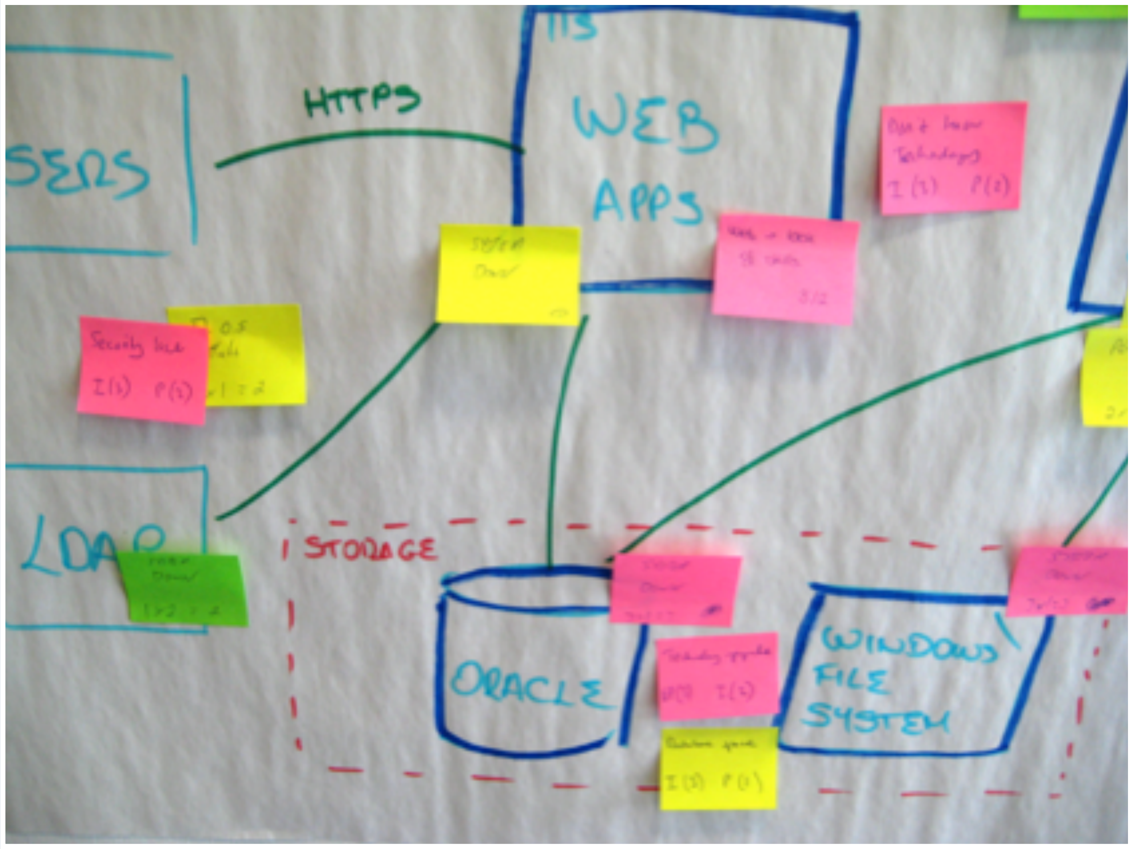
Like estimates,

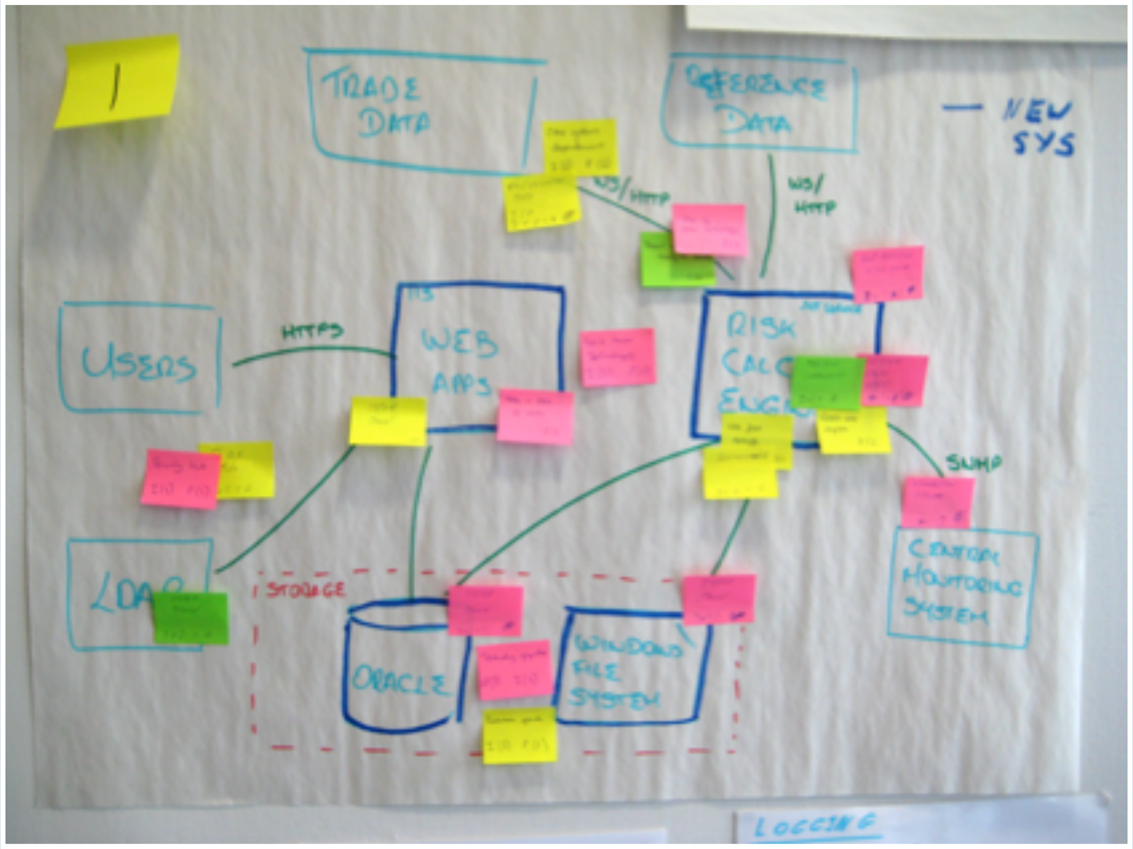
risks are subjective

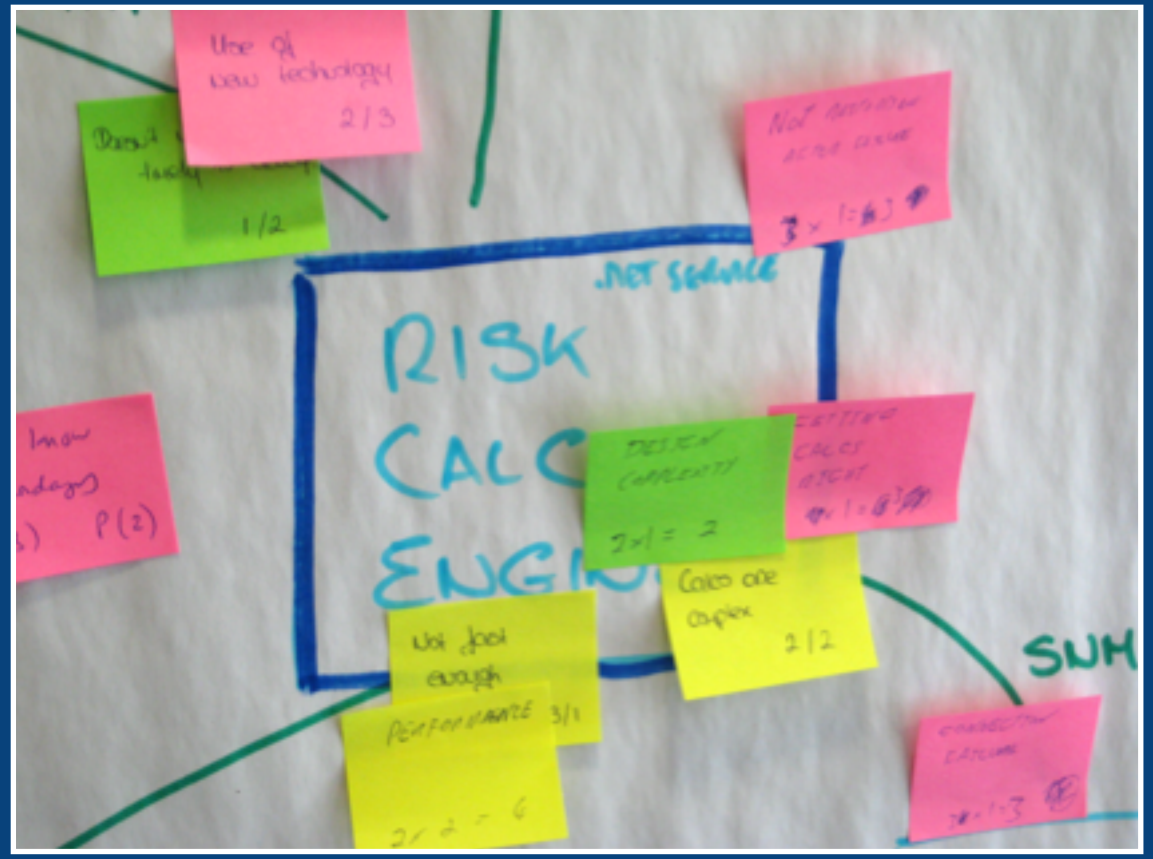

"Everybody knows the problem: We need to be more innovative. Now we've got the solution: Gamestorming. This smart, fun, hands-on book will energize your brain and mobilize your creativity—and do it using stuff you already have in your supply closet!"
—Daniel H. Pink, author of Drive and A Whole New Mind
Gamestorming
A Playbook for Innovators, Rulebreakers, and Changemakers
Dave Gray
Sunni Brown
James Macanufo
O'REILLY



Visual and collaborative "games"







Risk-storming

A visual and collaborative technique for identifying risk

Threat modelling

(STRIDE, LINDDUN, Attack Trees, etc)

Base your architecture on requirements, travel light and prove your architecture with concrete experiments.

Agile Architecture: Strategies for Scaling Agile Development

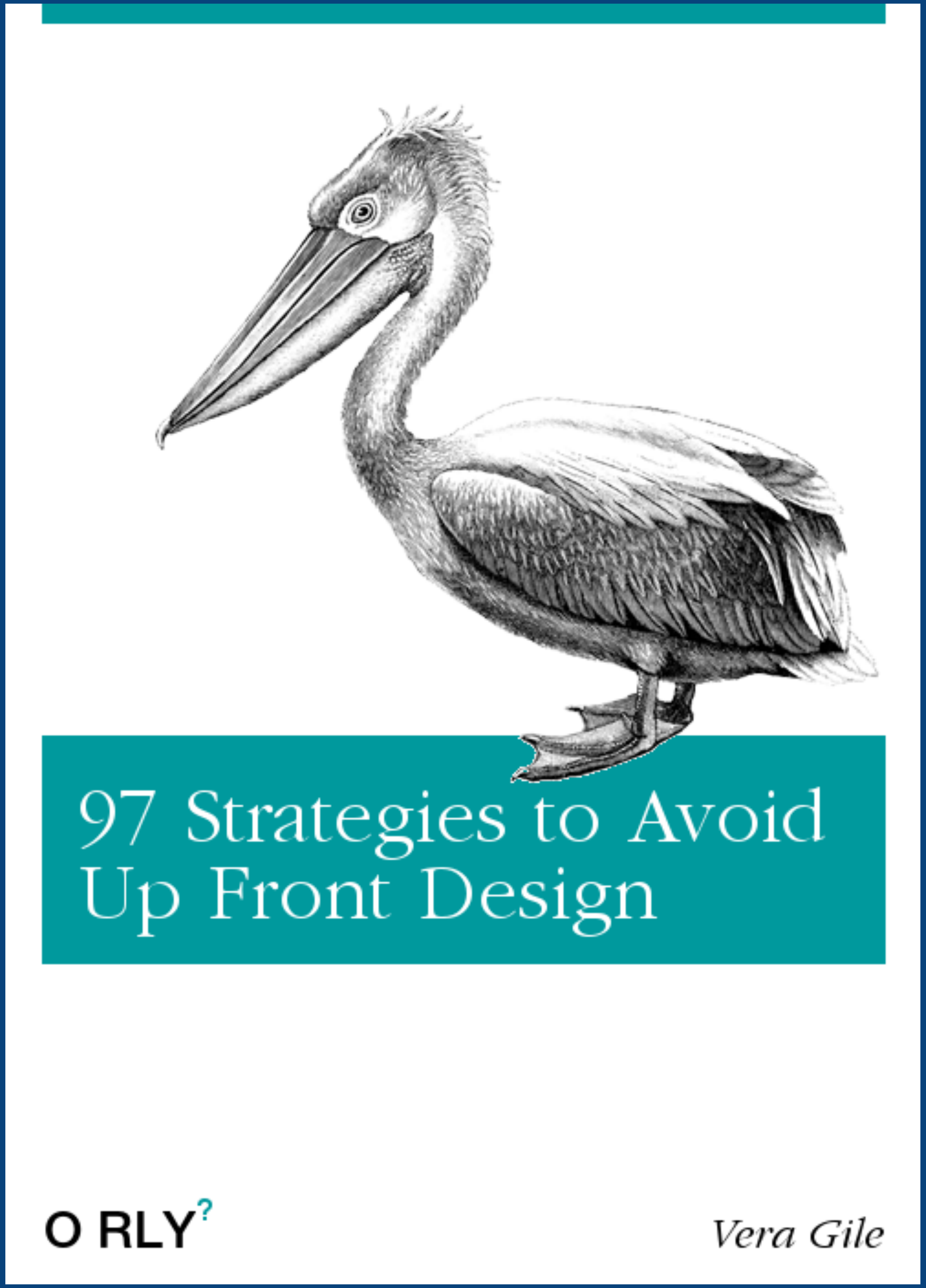
Scott Ambler

Concrete experiment

Proof of concept, prototype, spike, tracer, vertical slice, walking skeleton, executable reference architecture, …

Just enough up front design to create **firm and sufficient foundations**

How much up front design should you do?



#52

“I’m good with maybe a day for a one-year effort.”

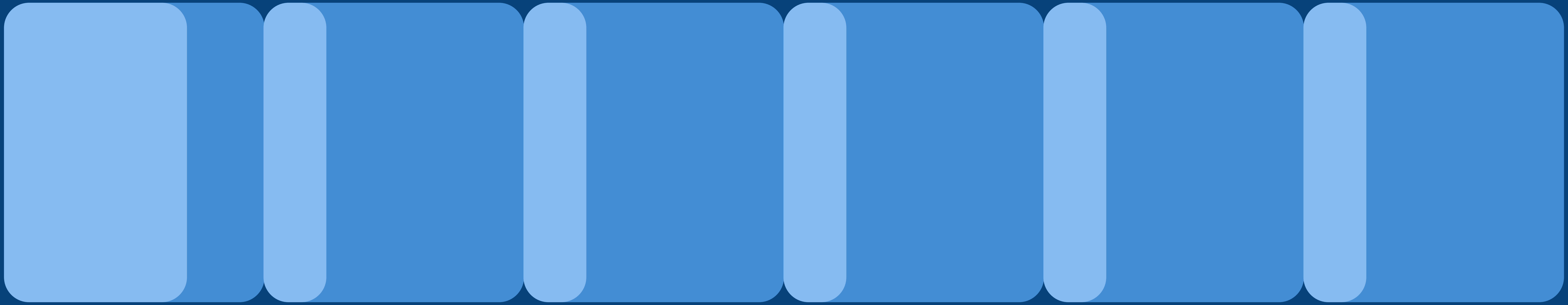
Up front design is an iterative and incremental process; stop when:

- ✓ You understand the significant architectural drivers (requirements, quality attributes, constraints).
- ✓ You understand the context and scope of what you’re building.
- ✓ You understand the significant design decisions (i.e. technology, modularity, etc).
- ✓ You have a way to communicate your technical vision to other people.
- ✓ You are confident that your design satisfies the key architectural drivers.
- ✓ You have identified, and are comfortable with, the risks associated with building the software.

Techniques: Workshops, interviews, Event Storming, Impact Mapping, domain modelling, OOAD, CRC, DDD, architecture reviews, ATAM, architecture dry runs, Risk-storming, concrete experiments, C4 model, ADRs, etc.

How long?

Hours, days or weeks … not months or years



Some Design Up Front
+ Evolutionary Design

Some up front design to create a **starting point** and **direction** for further **evolutionary design**

Estimates?

> we used to do things like this,
> it worked but we stopped doing
> it when we became agile

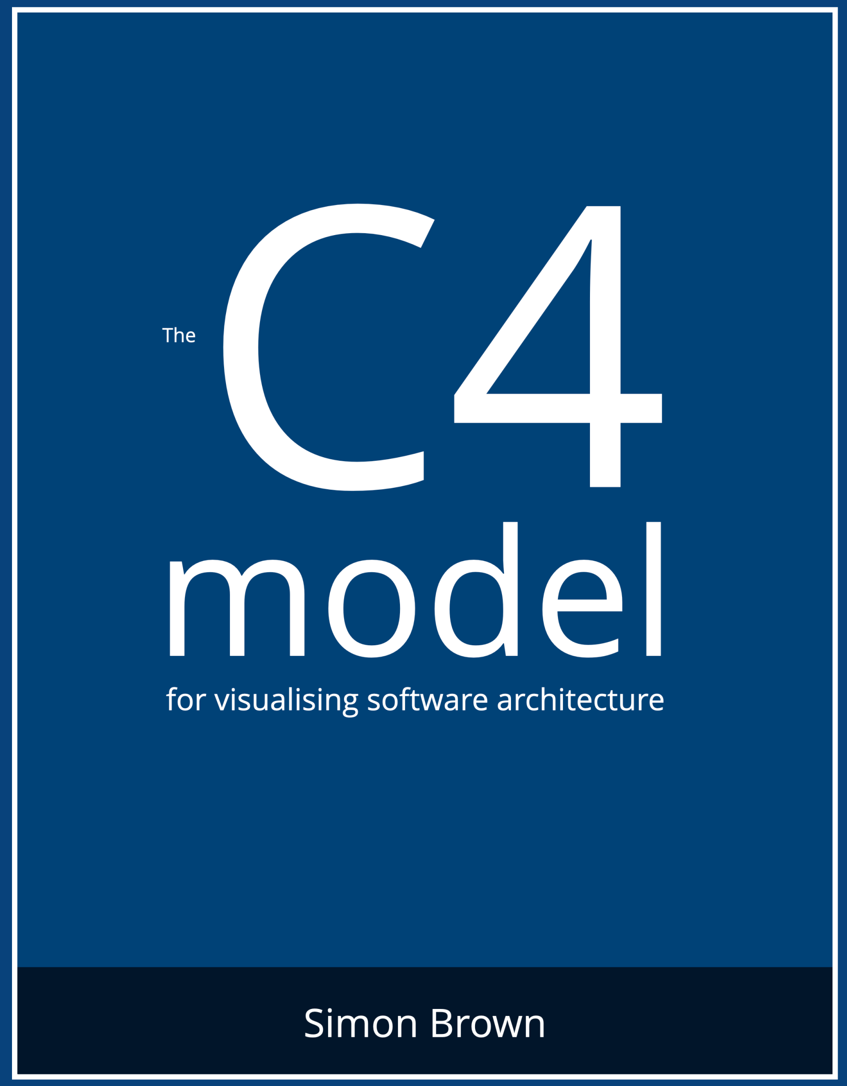
Adopt an agile mindset

Choose a starting point and continuously improve
to discover what works for you

Thank you!

Software architecture for developers

Simon Brown



The C4 model for visualising software architecture

Simon Brown

https://leanpub.com/b/software-architecture

Simon Brown