The lost art of software modelling

Simon Brown

@simonbrown

Over the past decade, many teams have thrown away big design up front

Unfortunately, architectural thinking, documentation, diagramming and modelling were also often discarded

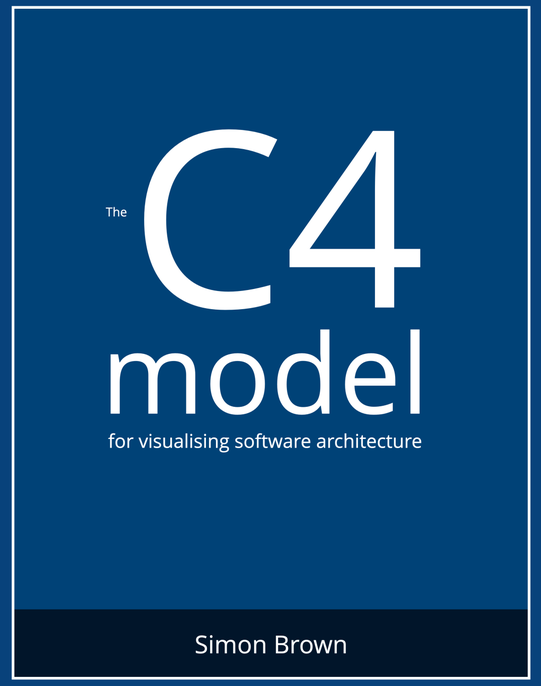
Big design up front is dumb.
Doing no design up front
is even dumber.

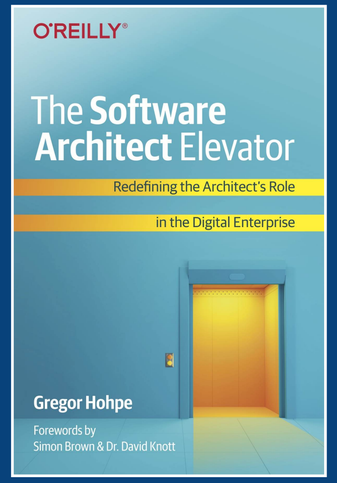
Dave Thomas

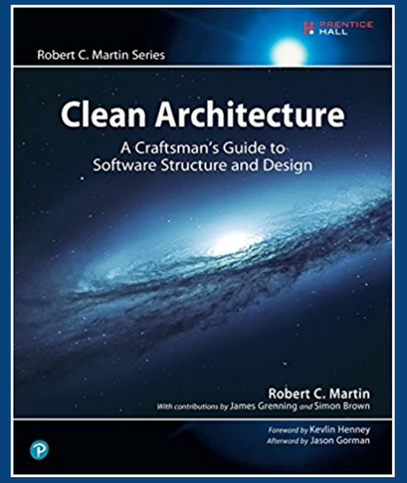
Simon Brown

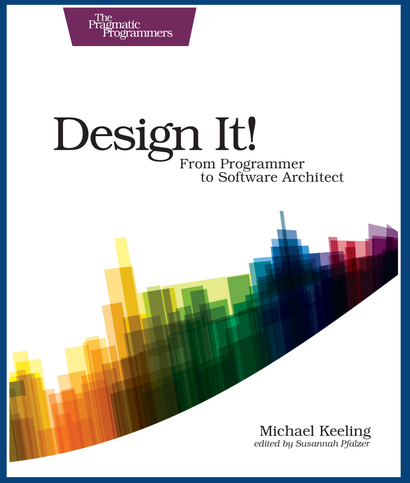
Independent consultant specialising in software architecture,
plus the creator of the C4 model and Structurizr

@simonbrown









Architecture meets agile

“we’re about to start our agile transformation … we need help making our architecture/design processes more agile”

vs

Agile meets architecture

“we’ve been on our agile journey for X years … our software lacks structure, we have no documentation, etc”

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.

3. Non-functional Requirements

a. Performance

- Risk reports must be generated before 9am the following business day in Singapore.

b. Scalability

- The system must be able to cope with trade volumes for the next 5 years.
  - The Trade Data System export includes approximately 5000 trades now and it is anticipated that there will be slow but steady growth of 10 additional trades per day.
  - The Reference Data System export includes approximately 20,000 counterparties and growth will be negligible.
- There are 40-50 business users around the world that need access to the report.

c. Availability

- Risk reports should be available to users 24x7, but a small amount of downtime (less than 30 minutes per day) can be tolerated.

d. Failover

- Manual failover is sufficient, provided that the availability targets can be met.

e. Security

- This system must follow bank policy that states system access is restricted to authenticated and authorised users only.
- Reports must only be distributed to authorised users.
- Only a subset of the authorised users are permitted to modify the parameters used in the risk calculations.
- Although desirable, there are no single sign-on requirements (e.g. integration with Active Directory, LDAP, etc).
- All access to the system and reports will be within the confines of the bank’s global network.

f. Audit

- The following events must be recorded in the system audit logs:
  - Report generation.
  - Modification of risk calculation parameters.

g. Fault Tolerance and Resilience

- The system should take appropriate steps to recover from an error if possible, but all errors should be logged.
- Errors preventing a counterparty risk calculation being completed should be logged and the process should continue.

h. Internationalization and Localization

- All user interfaces will be presented in English only.
- All reports will be presented in English only.
- All trading values and risk figures will be presented in US dollars only.

i. Monitoring and Management

- A Simple Network Management Protocol (SNMP) trap should be sent to the bank’s Central Monitoring Service in the following circumstances:
  - When there is a fatal error with the system.
  - When reports have not been generated before 9am Singapore time.

j. Data Retention and Archiving

- Input files used in the risk calculation process must be retained for 1 year.

k. Interoperability

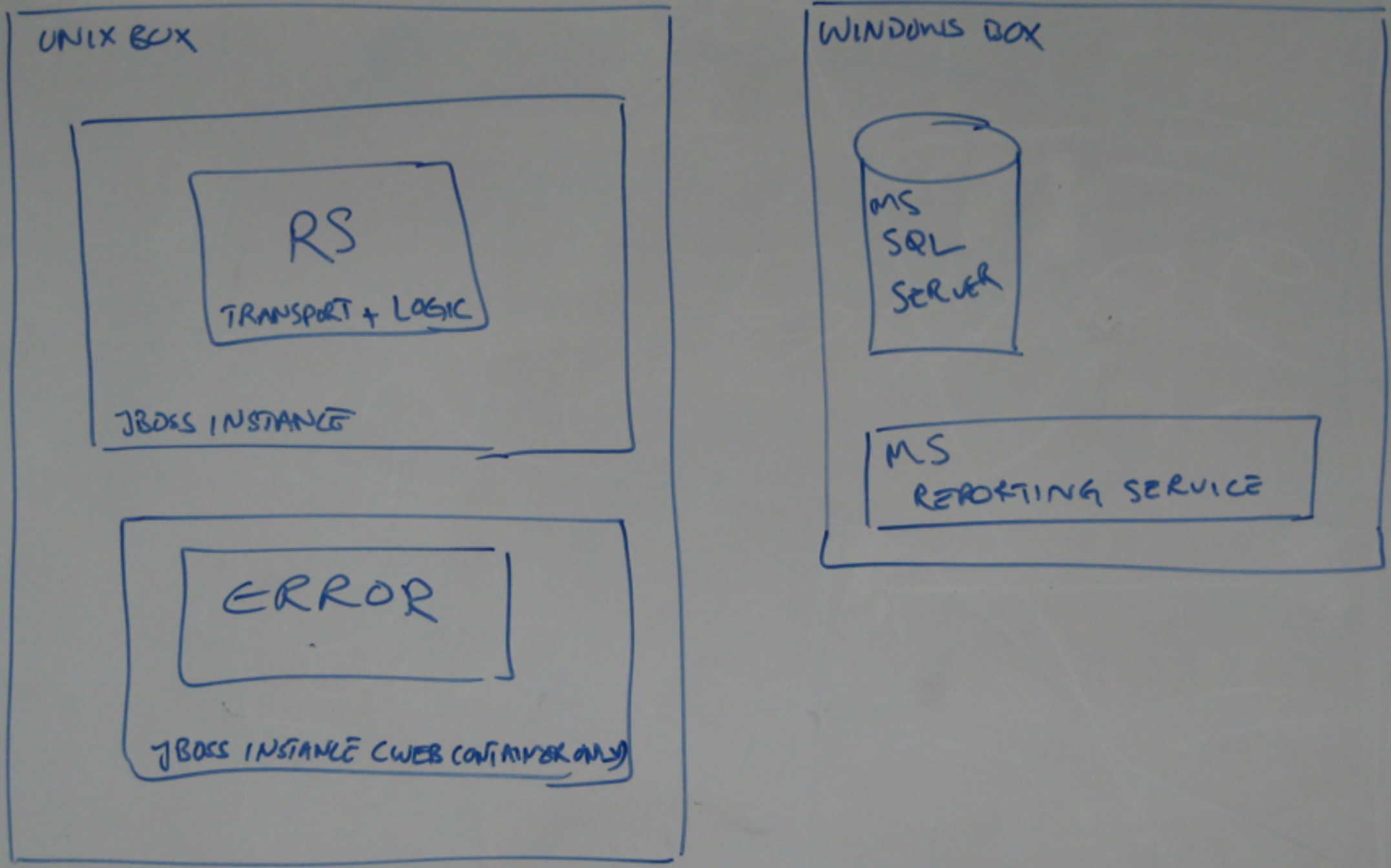
- Interfaces with existing data systems should conform to and use existing data formats.

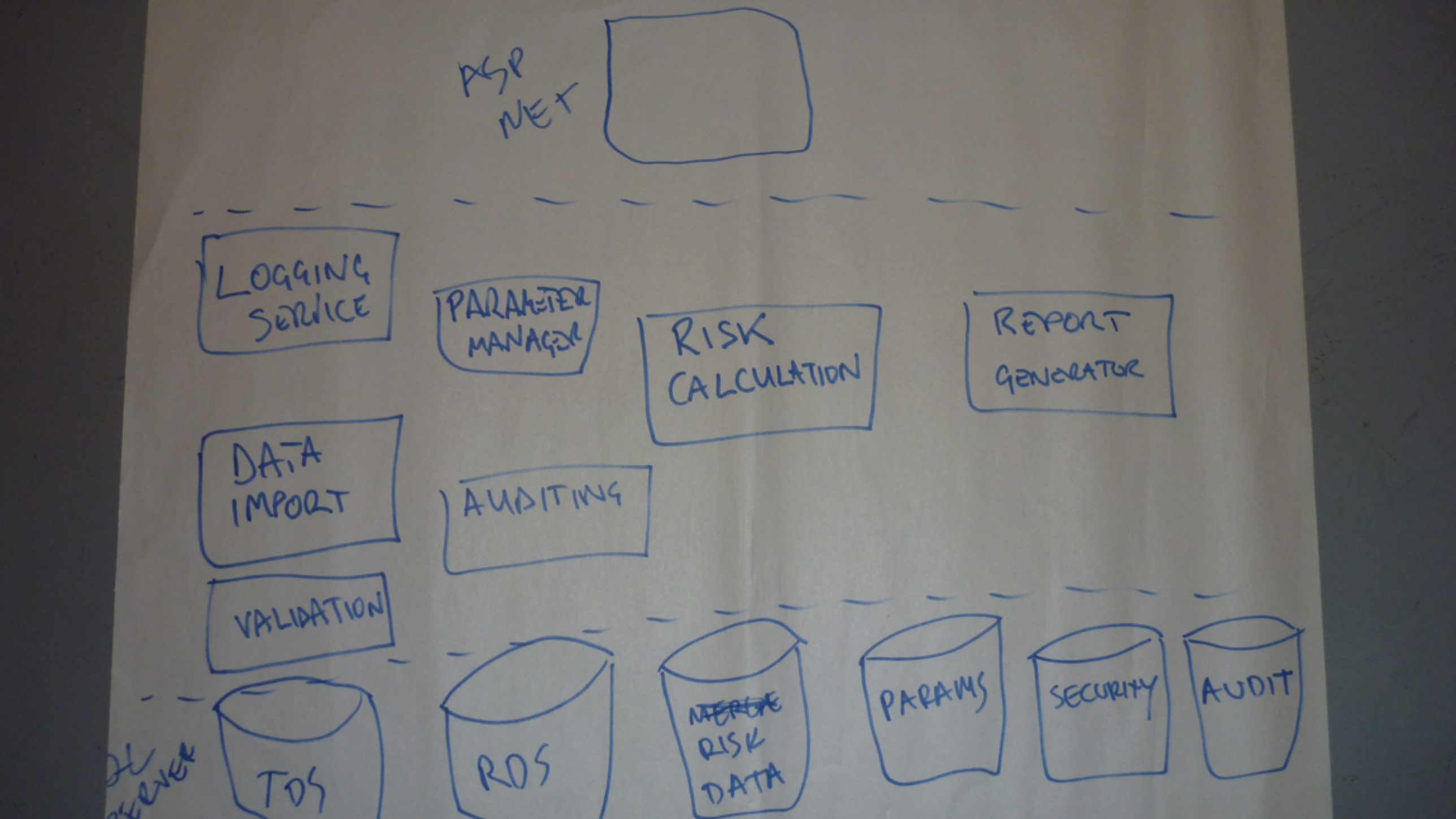
“Financial Risk System” architecture kata
Simon Brown | @simonbrown

CC BY NC SA

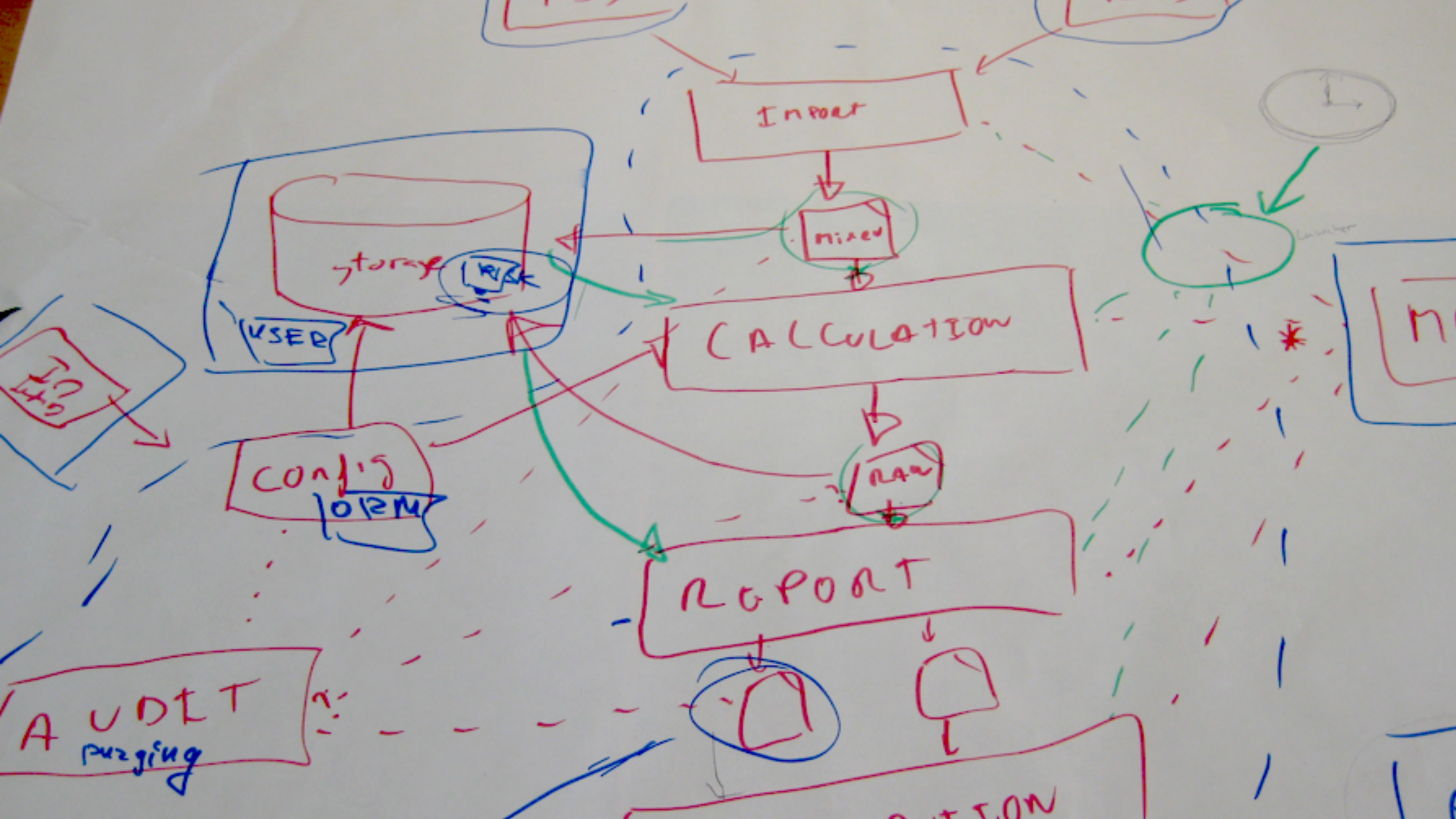
Design a software solution for the ”Financial Risk System”, and **draw** one or more architecture diagrams to describe your solution

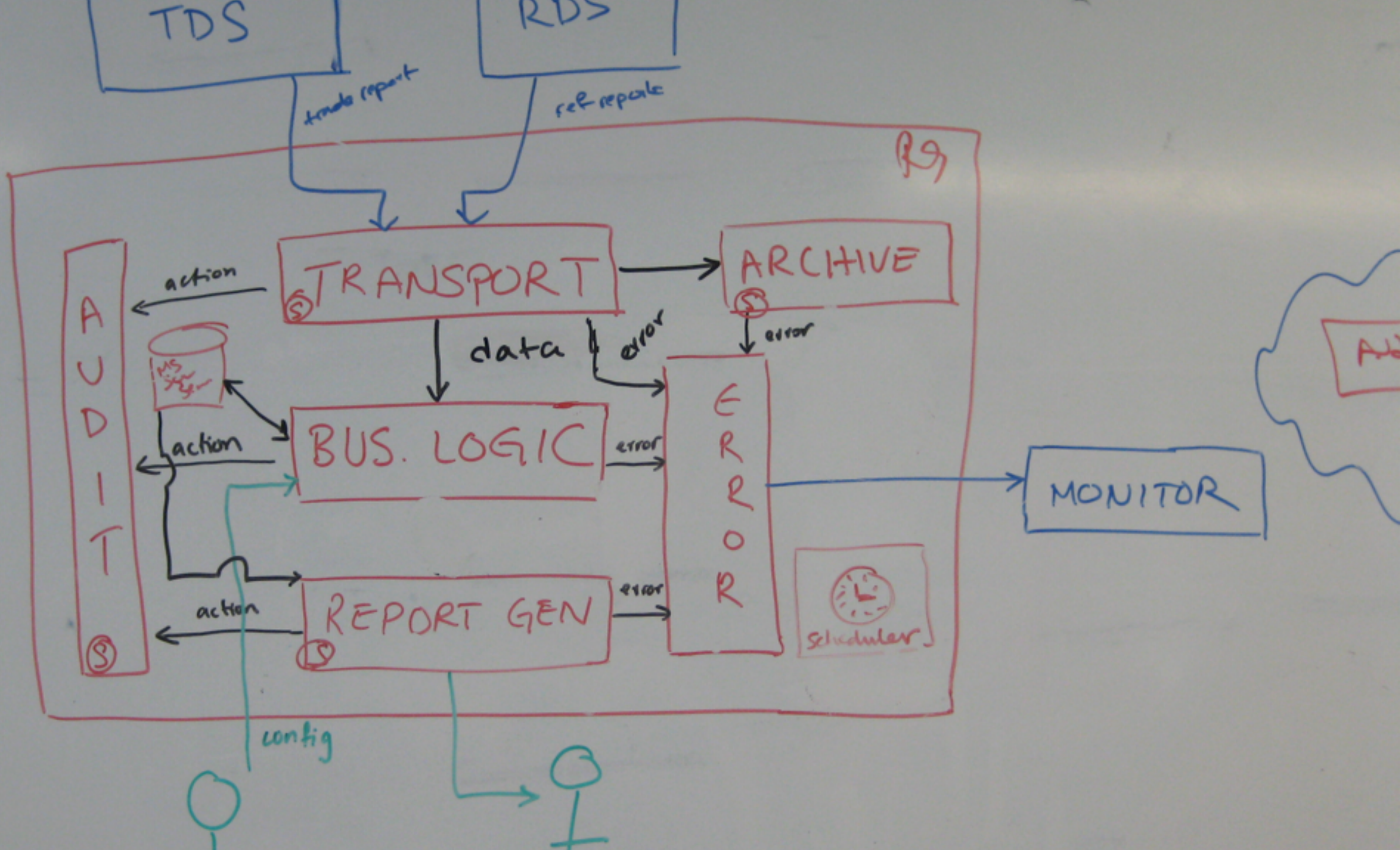
60-90 minutes

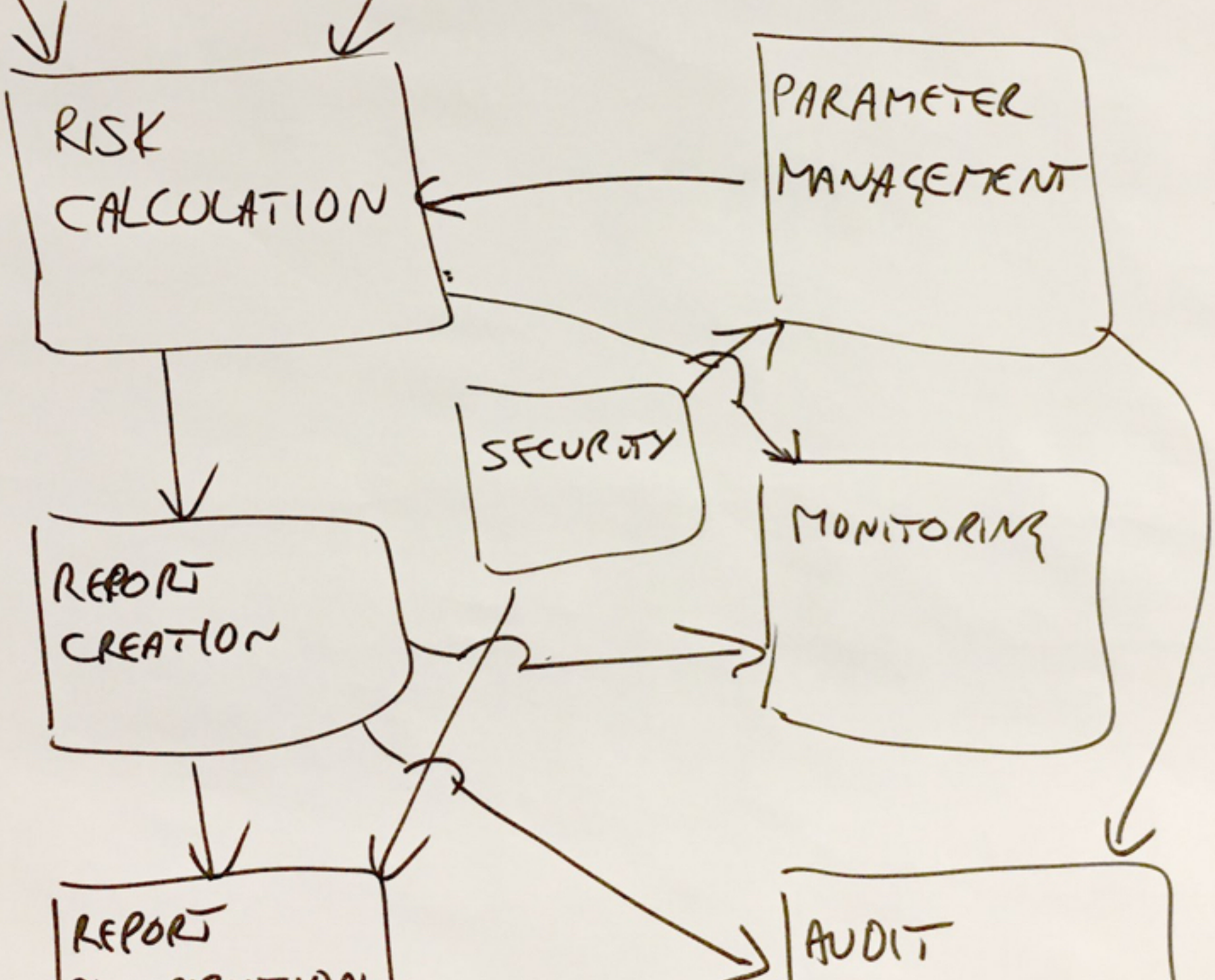

UNIX BOX
RS
TRANSPORT + LOGIC
JBOSS INSTANCE
ERROR
JBOSS INSTANCE (WEB CONTAINER ONLY)
WINDOWS BOX
MS
SQL
SERVER
MS
REPORTING SERVICE

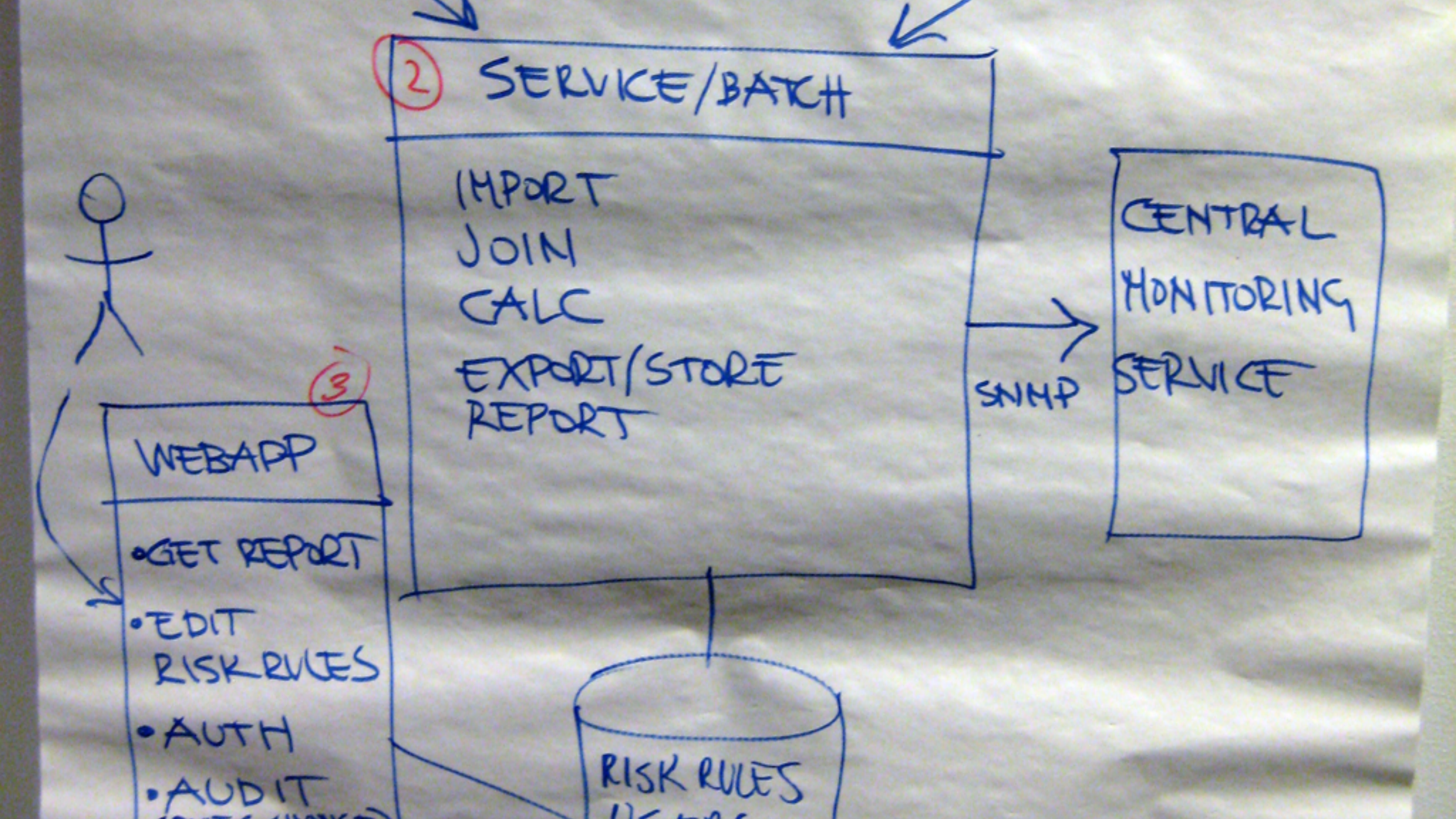

ASP NET
LOGGING SERVICE
PARAMETER MANAGER
RISK CALCULATION
REPORT GENERATOR
DATA IMPORT
AUDITING
VALIDATION
TDS
RDS
RISK DATA
PARAMS
SECURITY
AUDIT

FUNCTIONAL VIEW


Import
storage
risk
USER
config
mixed
CALCULATION
plan
REPORT
AUDIT
purging


TDS
RDS
trade report
ref repos
R9
TRANSPORT
ARCHIVE
action
data
error
error
AUDIT
BUS. LOGIC
action
error
ERROR
MONITOR
action
REPORT GEN
error
Scheduler
config


RISK
CALCULATION
PARAMETER
MANAGEMENT
SECURITY
MONITORING
REPORT
CREATION
AUDIT

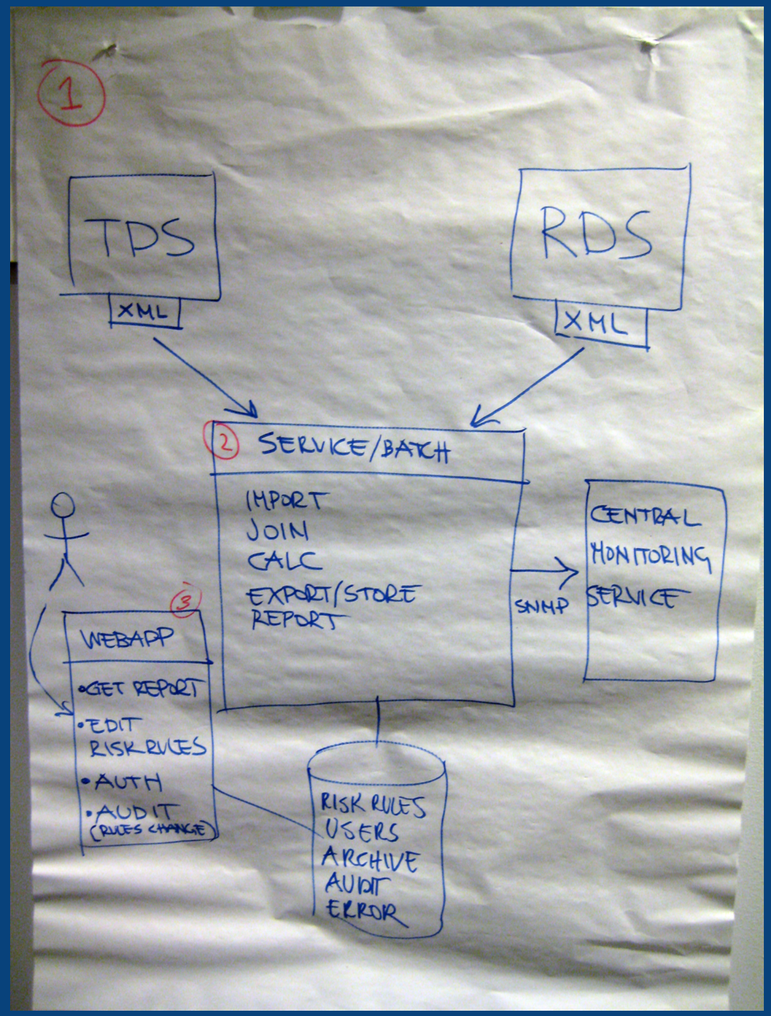

2
SERVICE/BATCH
IMPORT
JOIN
CALC
EXPORT/STORE
REPORT
SNMP
CENTRAL
MONITORING
SERVICE
3
WEBAPP
•GET REPORT
•EDIT
RISK RULES
•AUTH
•AUDIT
RISK RULES

The producer-consumer conflict of software architecture diagrams

I don’t want to put technology choices on the diagrams…

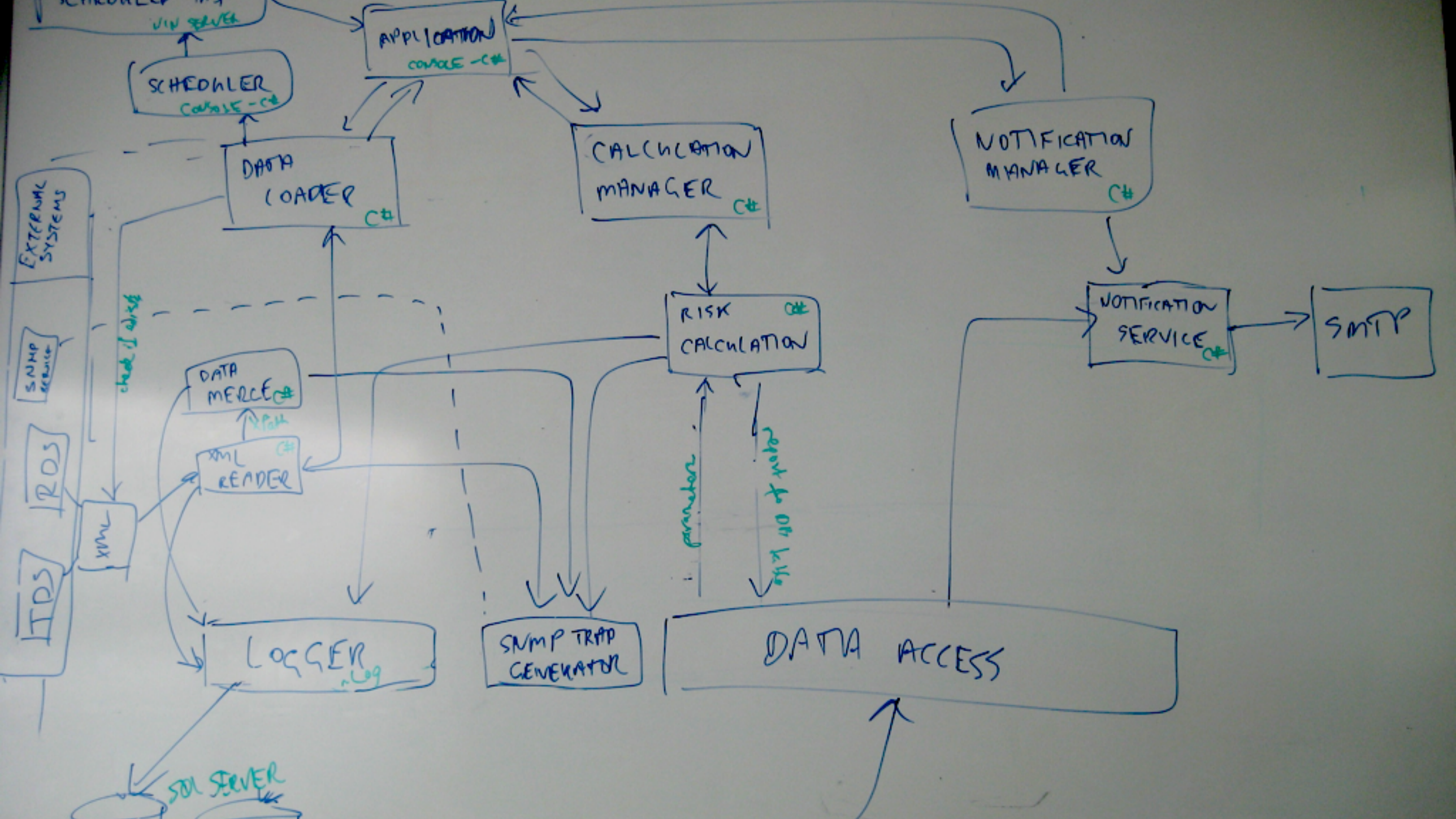
Software design should be technology independent…

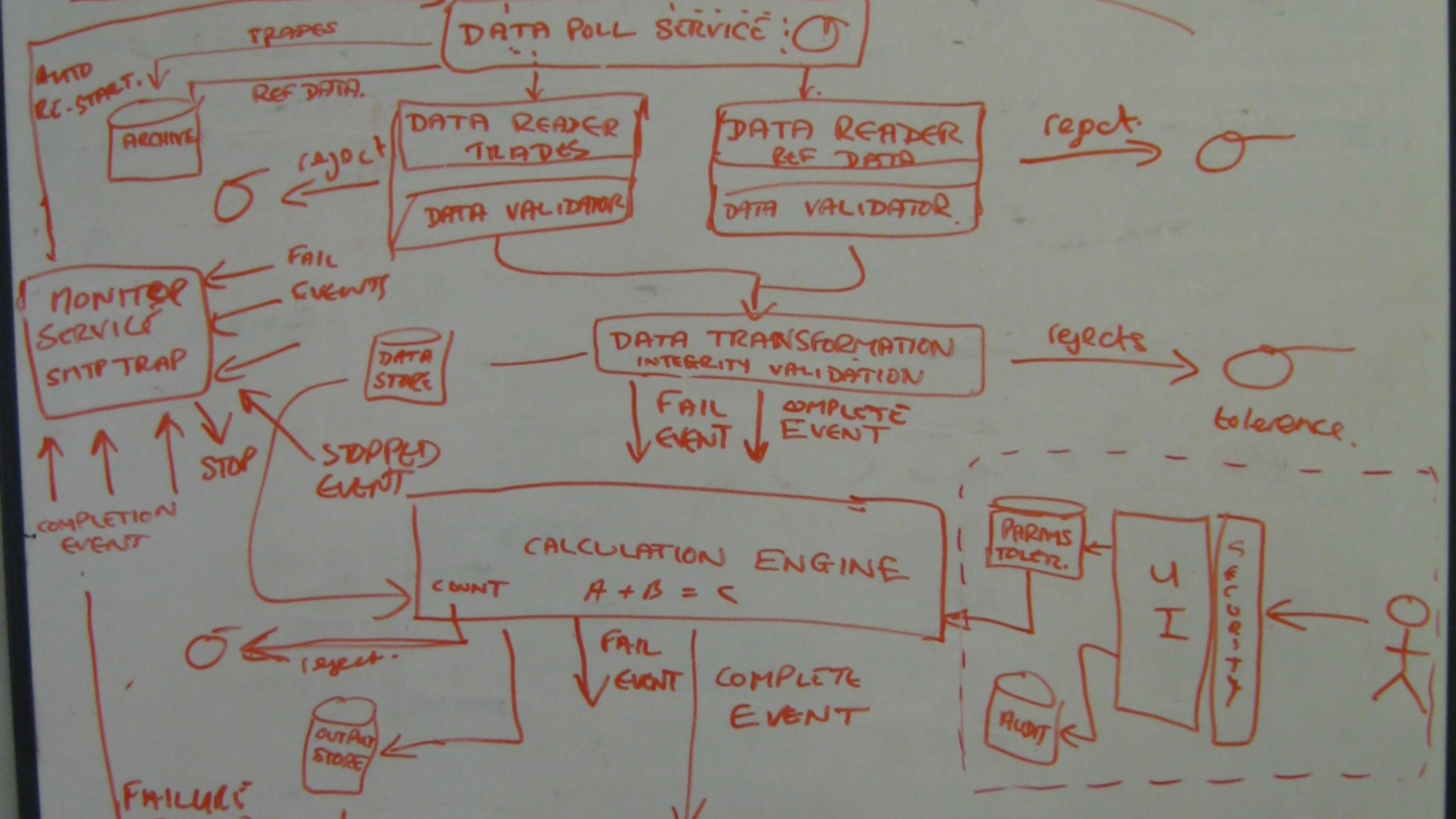
Producer

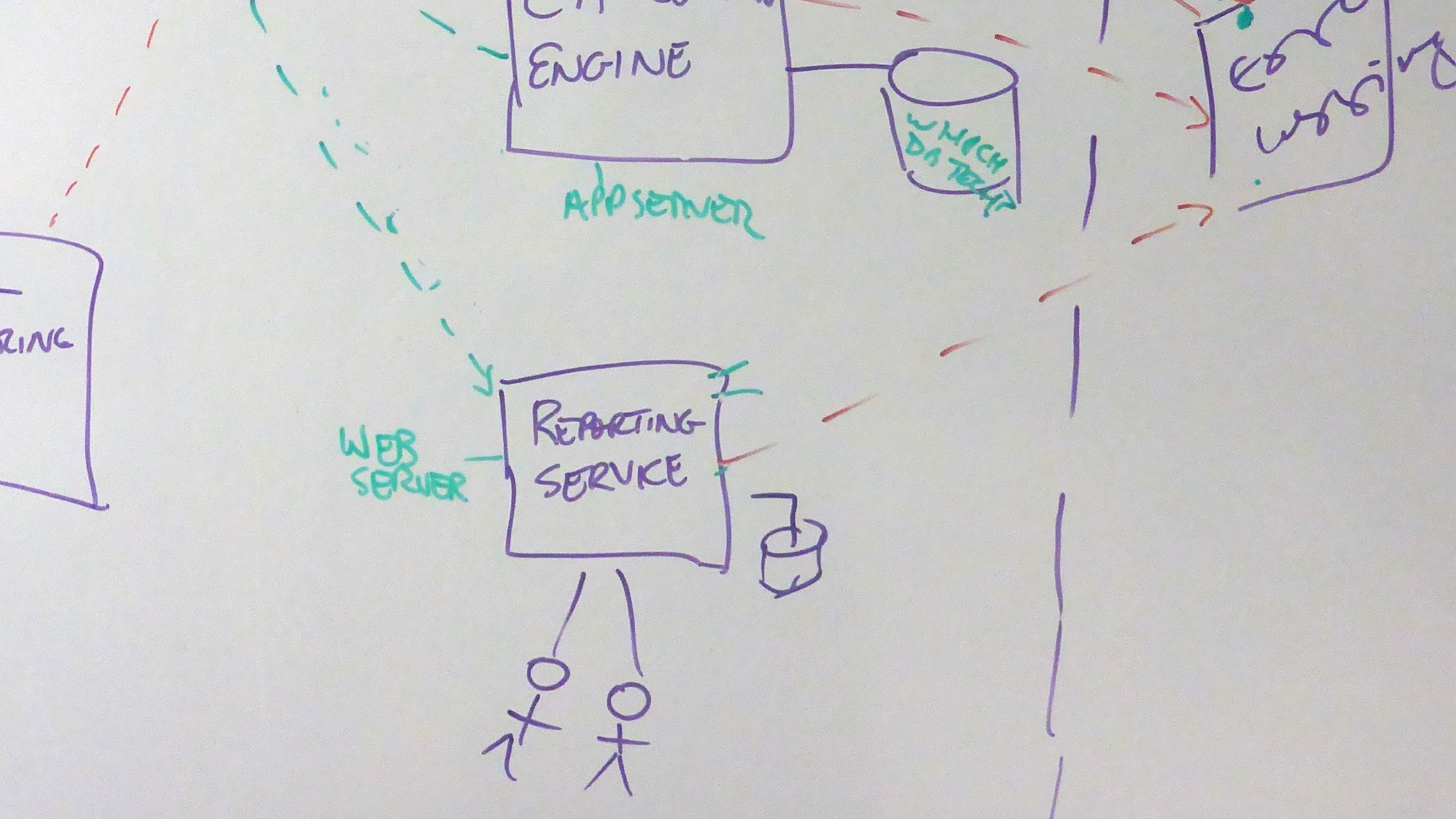


I wish these diagrams included technology choices…

Consumer

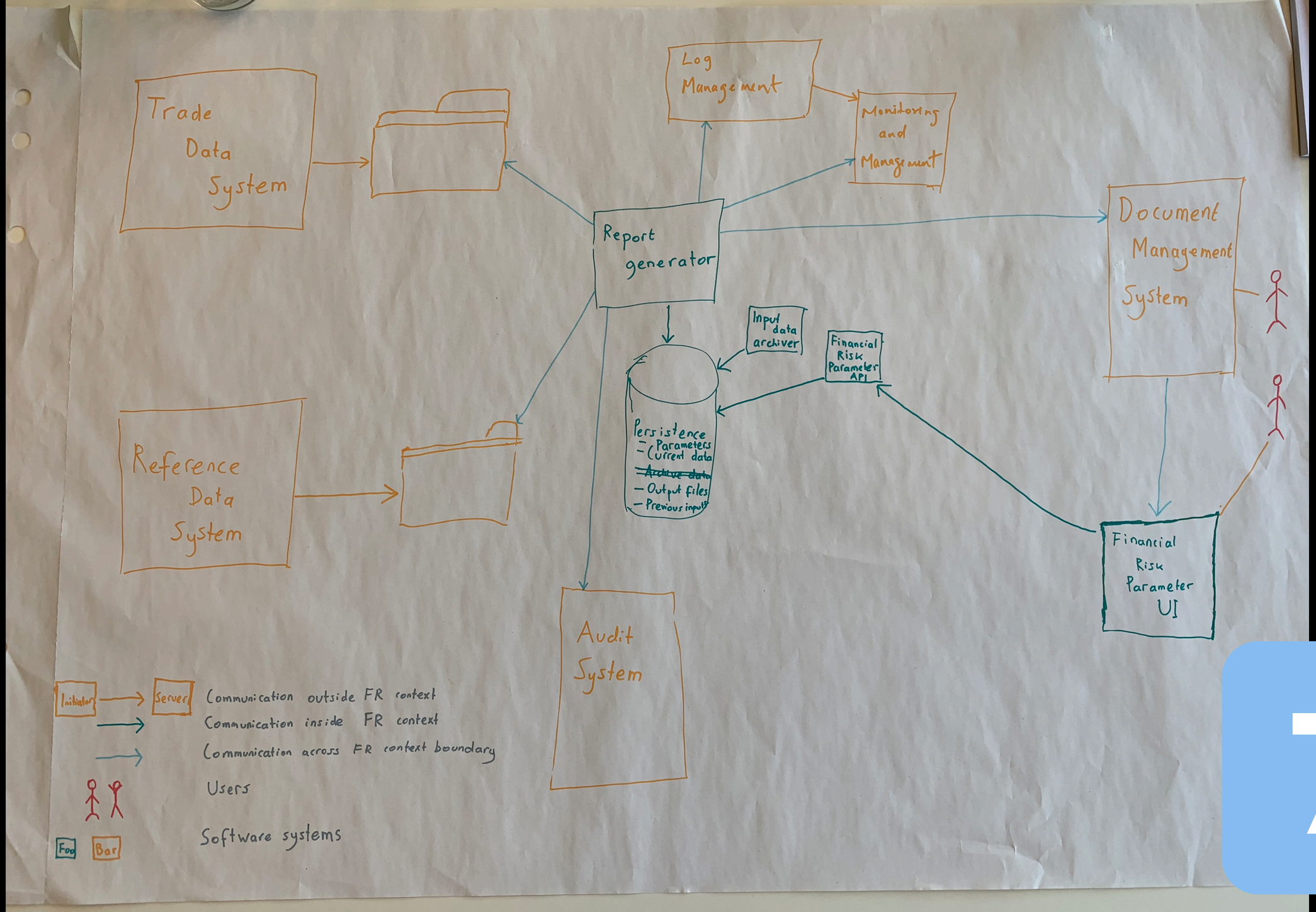

APPLICATION
console - C#
SCHEDULER
console - C#
DATA LOADER
C#
CALCULATION MANAGER
C#
NOTIFICATION MANAGER
C#
EXTERNAL SYSTEMS
SNMP
IRO3
ITD3
check if exist
DATA MERGE
C#
XPath
XML READER
C#
XML
RISK CALCULATION
C#
parameters
save to DB
NOTIFICATION SERVICE
C#
SMTP
LOGGER
.log
SNMP TRAP GENERATOR
DATA ACCESS
SQL SERVER


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

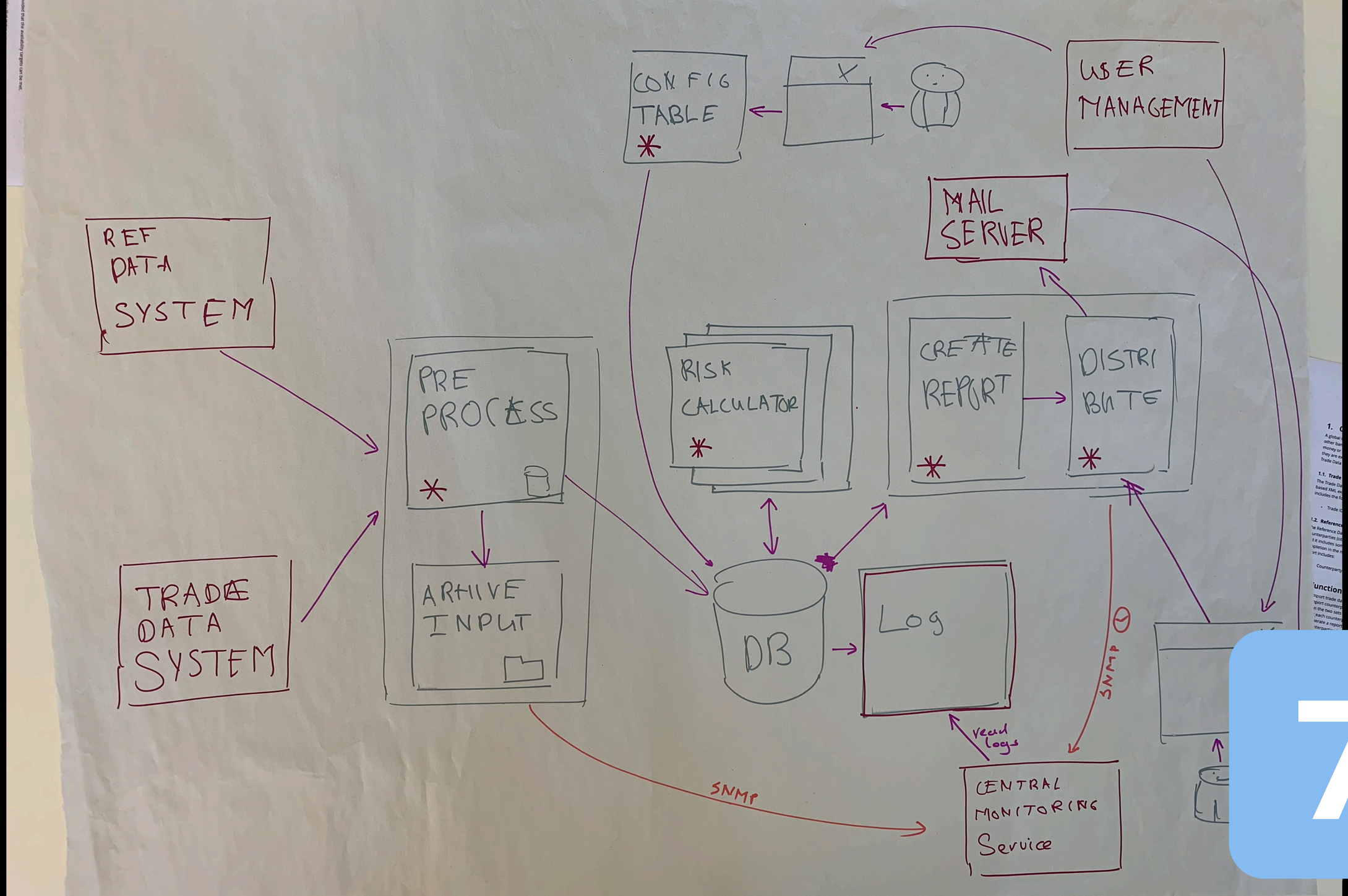

ENGINE
APP SERVER
RING
WEB SERVER
REPORTING SERVICE

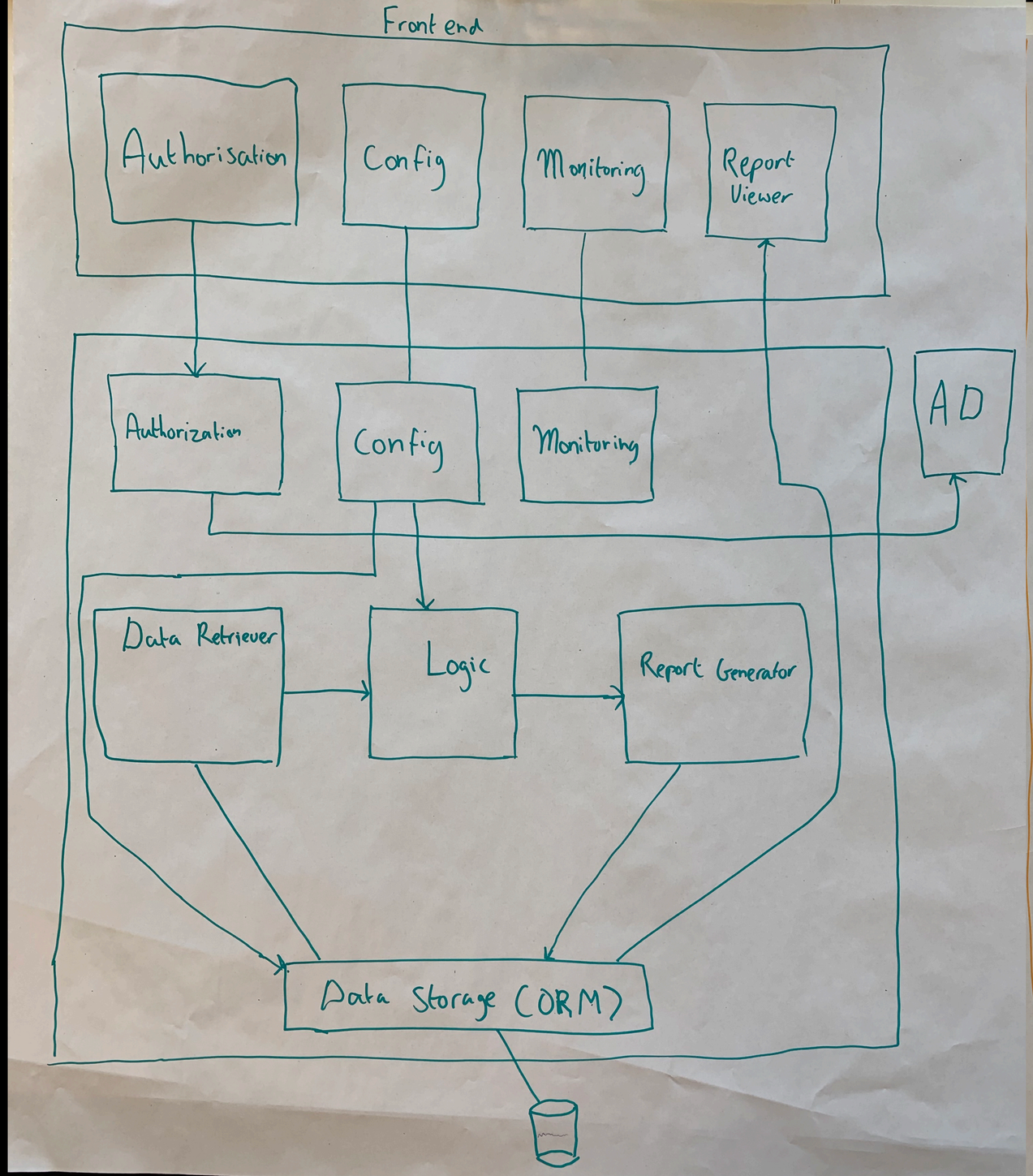

Report Monitor
(other machine)
Data Normalizer
Risk Calculator
Aggregator
CP Risk Calc
Report Generator
Auditing Service
Report Publisher
Report Repository


Params
Calcs
EH?
Batch
Date
- RiskCal
- calcs
- Par


Trade 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 inputs
Reference Data System
Audit System
Financial Risk Parameter UI
Initiator
Server
Communication outside FR context
Communication inside FR context
Communication across FR context boundary
Users
Foo
Bar
Software systems

7

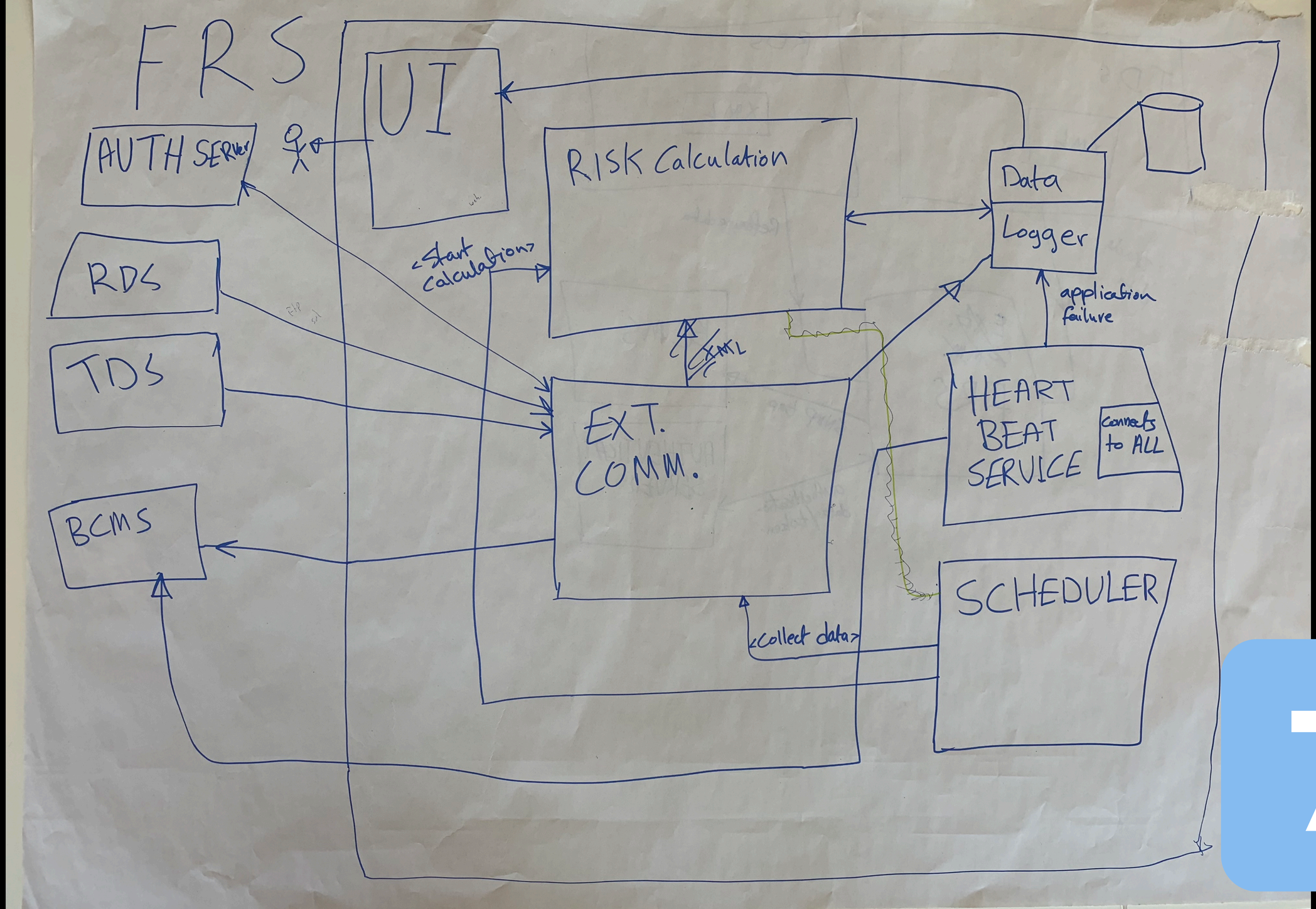

CONFIG TABLE
USER MANAGEMENT
MAIL SERVER
REF DATA SYSTEM
PRE PROCESS
RISK CALCULATOR
CREATE REPORT
DISTRI BUTE
TRADE DATA SYSTEM
ARHIVE INPUT
DB
Log
SNMP
read logs
SNMP
CENTRAL MONITORING Service
7

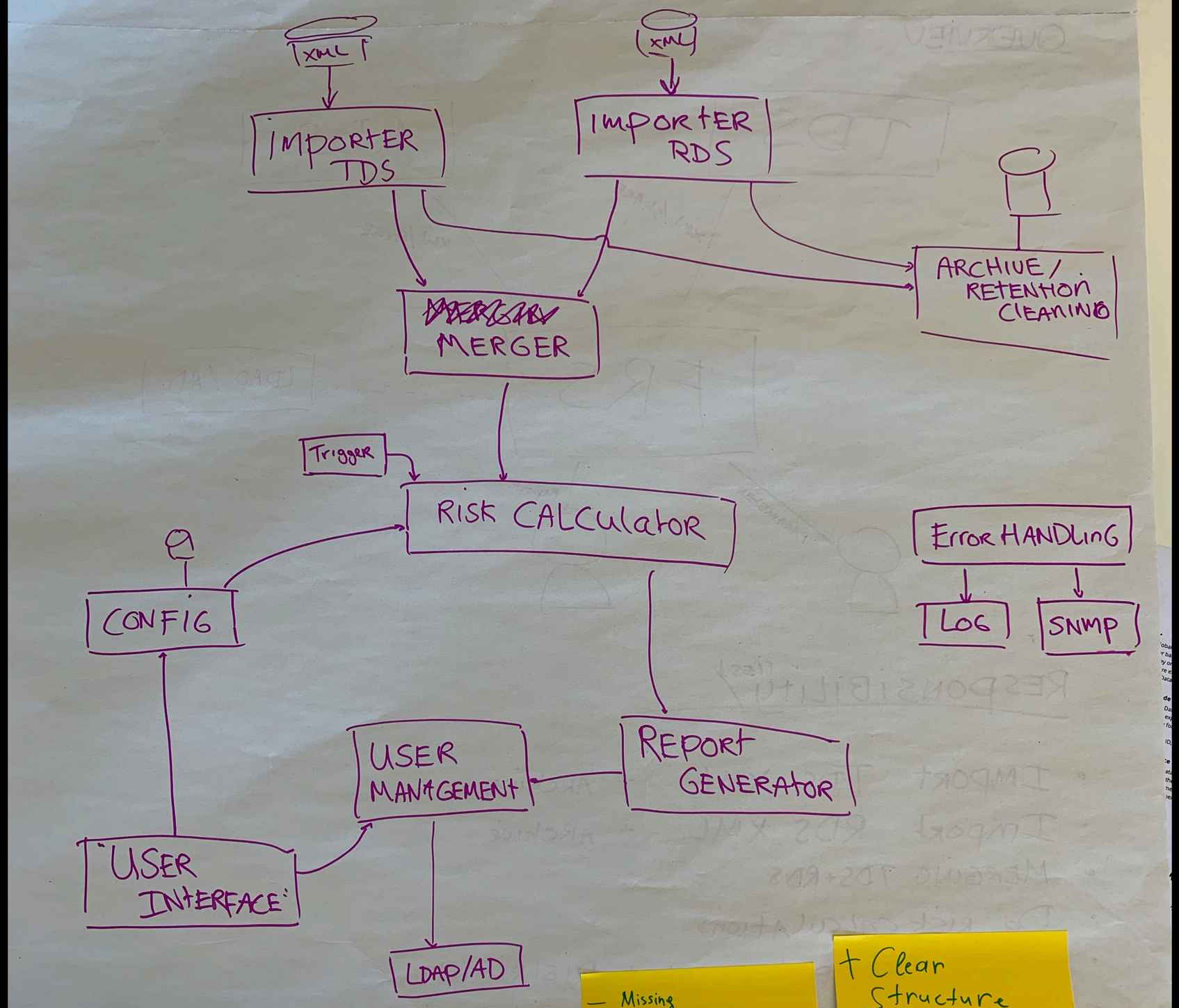


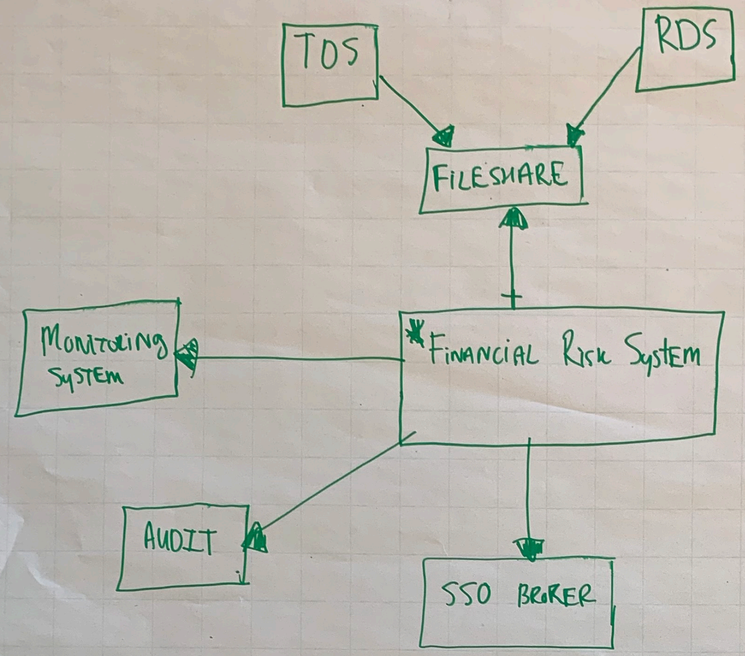
Significant decisions

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

7


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


XML
XML
IMPORTER TDS
IMPORTER RDS
ARCHIVE / RETENTION CLEANING
MERGER
Trigger
RISK CALCULATOR
CONFIG
ERROR HANDLING
LOG
SNMP
USER MANAGEMENT
REPORT GENERATOR
USER INTERFACE
LDAP/AD
+ Clear Structure

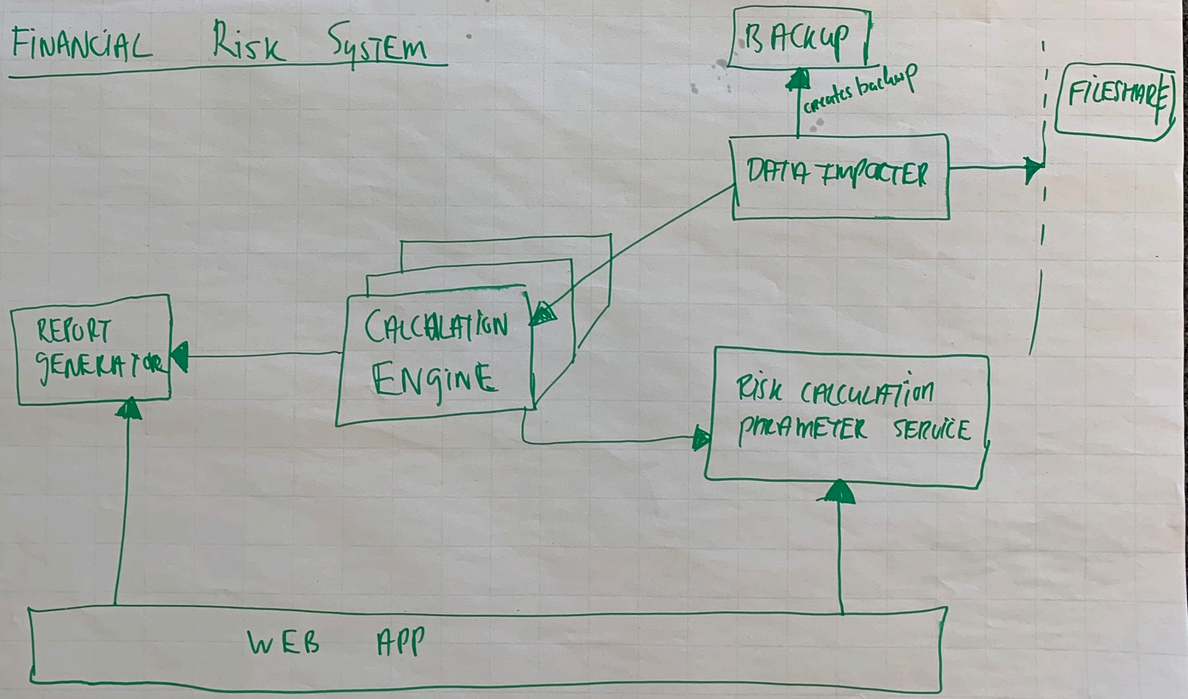


BACKUP

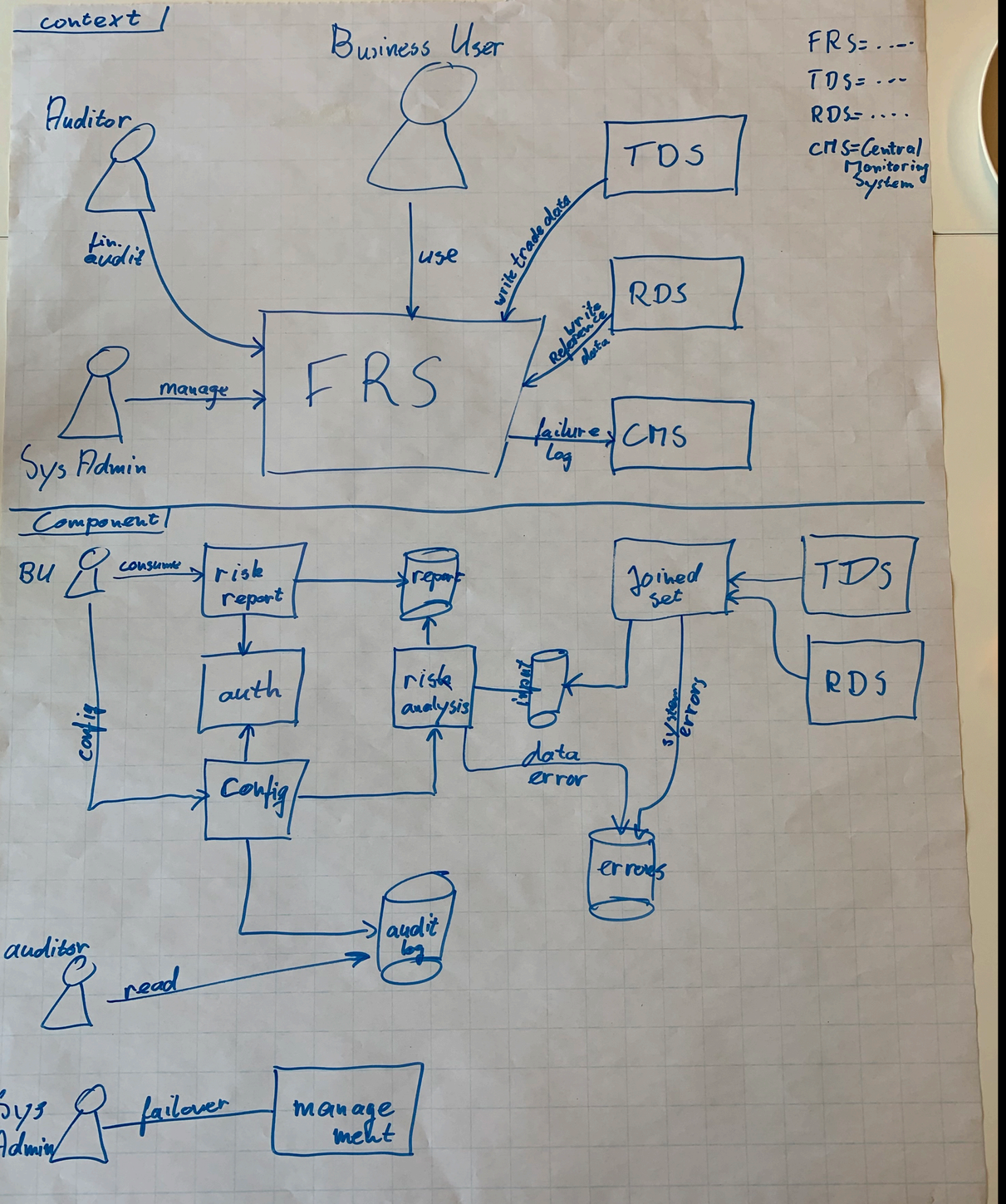
DECISIONS:

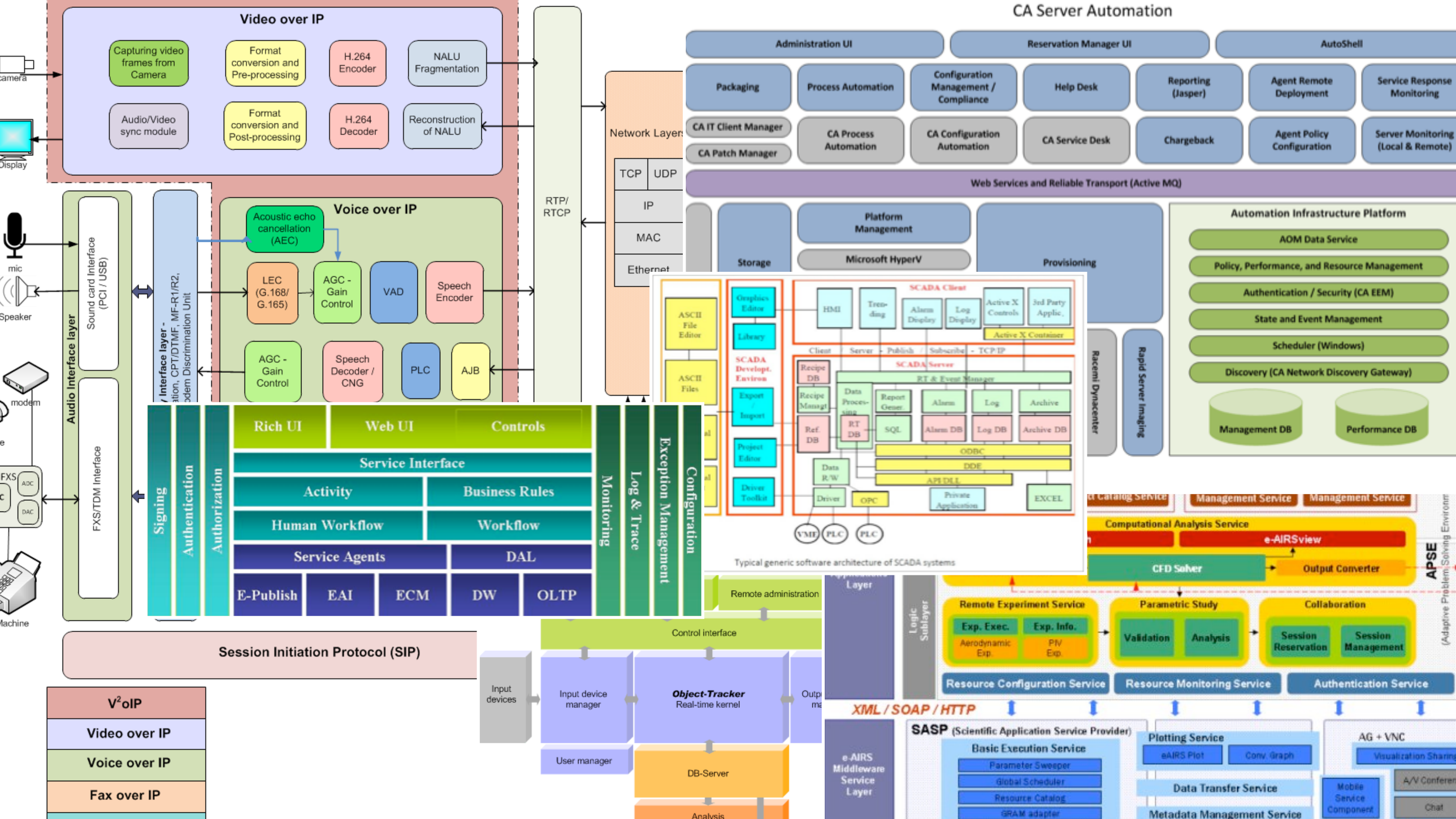
* CALCULATIONS ARE A JOB TRIGGERED ON SCHEDULE
* EXECUTE CALCULATIONS IN PARALLEL FOR EACH COUNTERPARTY
* WEBUI FOR VIEWING REPORTS AND MODIFYING RISK PARAMETERS.
* AUTHENTICATE AND AUTHORIZE USERS BASED ON SSO
* SINGLE POINT OF ENTRY

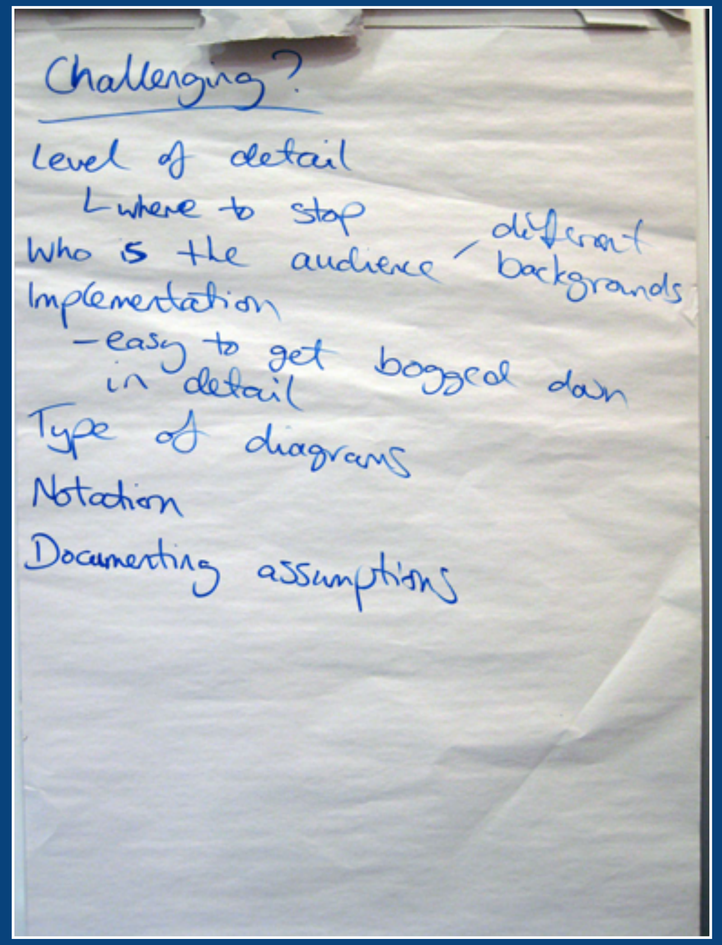
FINANCIAL RISK SYSTEM

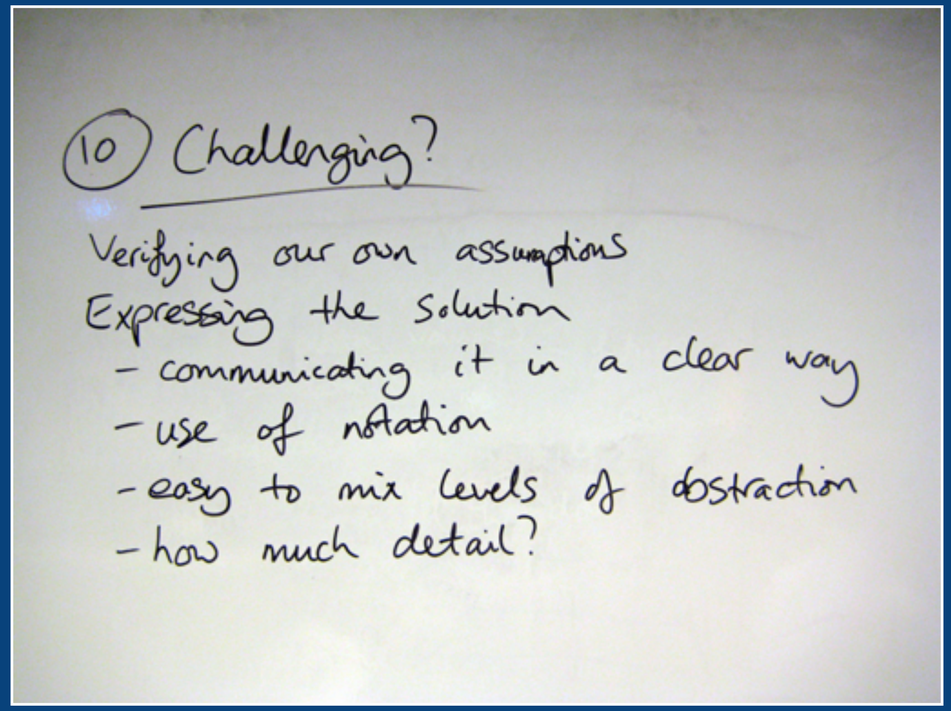


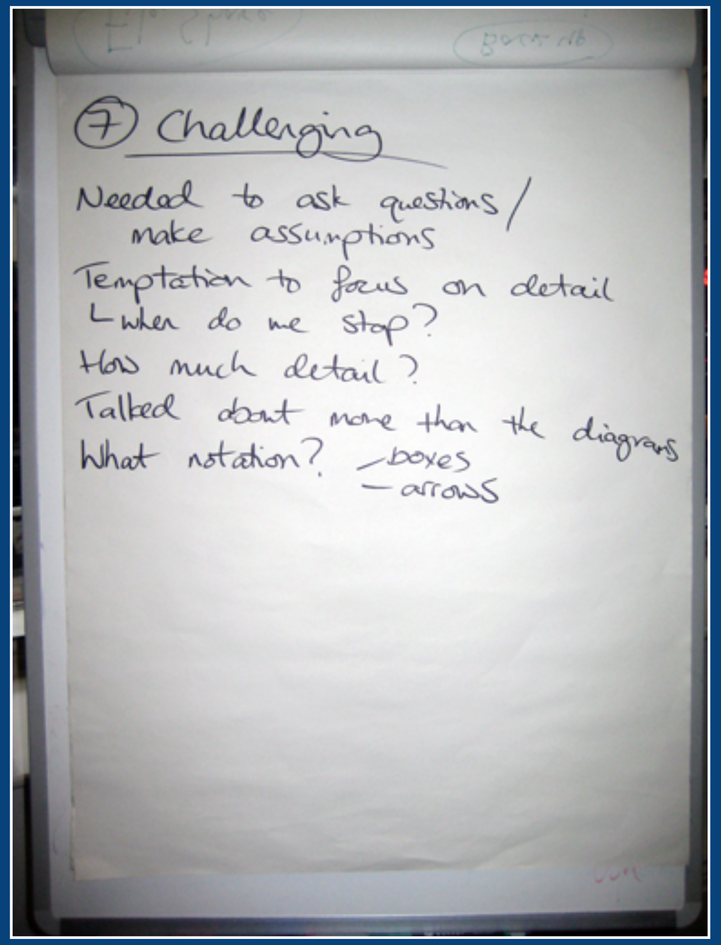
7


context
Business User
Auditor
fin. audit
use
write trade data
TDS
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
RDS
auth
risk analysis
input
config
Config
data error
system errors
errors
audit log
auditor
read
Sys Admin
failover
manage ment

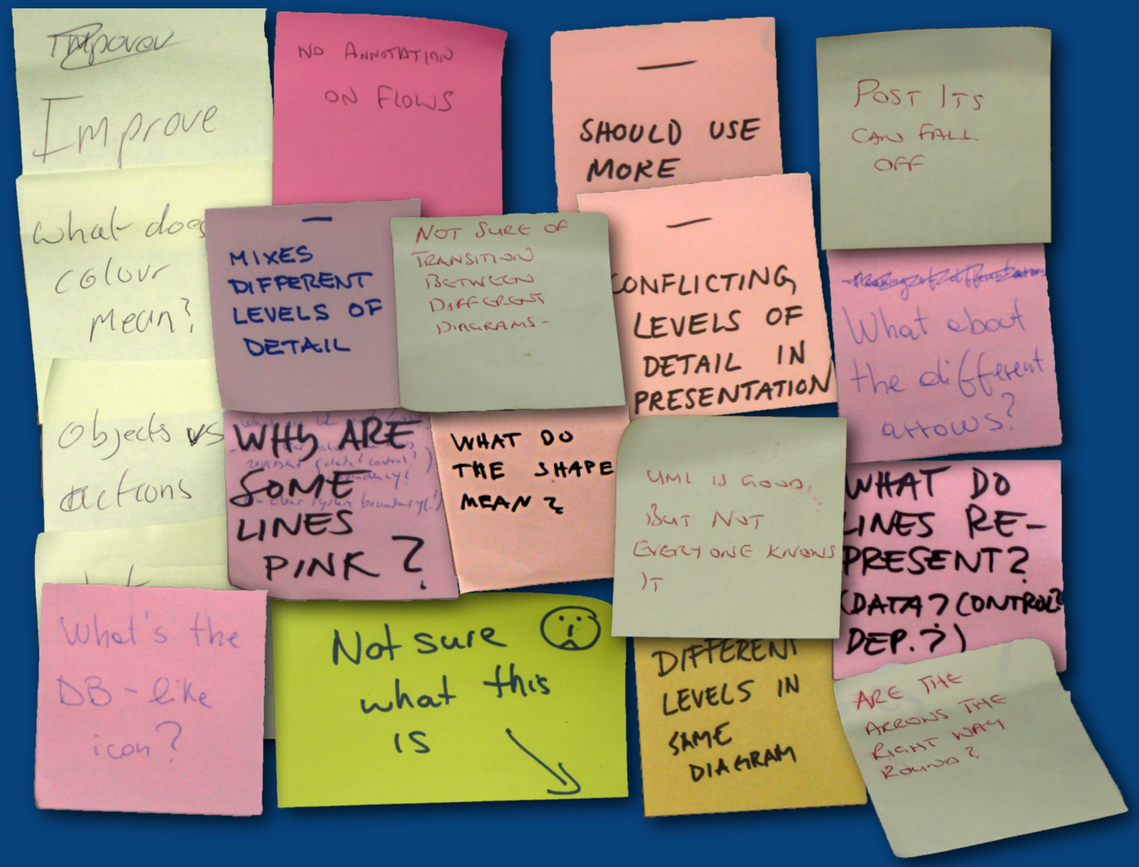

Video over IP
Capturing video frames from Camera
Format conversion and Pre-processing
H.264 Encoder
NALU Fragmentation
Audio/Video sync module
Format conversion and Post-processing
H.264 Decoder
Reconstruction of NALU
camera
Display
Voice over IP
Acoustic echo cancellation (AEC)
LEC (G.168/ G.165)
AGC - Gain Control
VAD
Speech Encoder
AGC - Gain Control
Speech Decoder / CNG
PLC
AJB
mic
Speaker
modem
Machine
Audio Interface layer
Sound card Interface (PCI / USB)
FXS/TDM Interface
Interface layer - tion, CPT/DTMF, MF-R1/R2, odem Discrimination Unit
RTP/ RTCP
Network Layers
TCP
UDP
IP
MAC
Ethernet
Signing
Authentication
Authorization
Rich UI
Web UI
Controls
Service Interface
Activity
Business Rules
Human Workflow
Workflow
Service Agents
DAL
E-Publish
EAI
ECM
DW
OLTP
Monitoring
Log & Trace
Exception Management
Configuration
Session Initiation Protocol (SIP)
V²oIP
Video over IP
Voice over IP
Fax over IP
CA Server Automation
Administration UI
Reservation Manager UI
AutoShell
Packaging
Process Automation
Configuration Management / Compliance
Help Desk
Reporting (Jasper)
Agent Remote Deployment
Service Response Monitoring
CA IT Client Manager
CA Patch Manager
CA Process Automation
CA Configuration Automation
CA Service Desk
Chargeback
Agent Policy Configuration
Server Monitoring (Local & Remote)
Web Services and Reliable Transport (Active MQ)
Storage
Platform Management
Microsoft HyperV
Provisioning
Automation Infrastructure Platform
AOM Data Service
Policy, Performance, and Resource Management
Authentication / Security (CA EEM)
State and Event Management
Scheduler (Windows)
Discovery (CA Network Discovery Gateway)
Management DB
Performance DB
Racemi Dynacenter
Rapid Server Imaging
ASCII File Editor
ASCII Files
Graphics Editor
Library
SCADA Develop. Environ
Export
Import
Project Editor
Driver Toolkit
SCADA Client
HMI
Trending
Alarm Display
Log Display
Active X Controls
3rd Party Applic.
Active X Container
Client - Server - Publish / Subscribe - TCP/IP
SCADA Server
Recipe DB
Recipe Managt
Ref. DB
Data Process-ing
RT & Event Manager
Report Gener.
Alarm
Log
Archive
RT DB
SQL
Alarm DB
Log DB
Archive DB
ODBC
DDE
API DLL
Data R/W
Driver
OPC
Private Application
EXCEL
VME
PLC
PLC
Typical generic software architecture of SCADA systems
Management Service
Management Service
Computational Analysis Service
e-AIRSview
CFD Solver
Output Converter
APSE
(Adaptive Problem Solving Environment)
Remote Experiment Service
Exp. Exec.
Exp. Info.
Aerodynamic Exp.
PIV Exp.
Parametric Study
Validation
Analysis
Collaboration
Session Reservation
Session Management
Resource Configuration Service
Resource Monitoring Service
Authentication Service
XML / SOAP / HTTP
Logic Sublayer
e-AIRS Middleware Service Layer
SASP (Scientific Application Service Provider)
Basic Execution Service
Parameter Sweeper
Global Scheduler
Resource Catalog
GRAM adapter
Plotting Service
eAIRS Plot
Conv. Graph
Data Transfer Service
Metadata Management Service
AG + VNC
Visualization Sharing
Mobile Service Component
A/V Conferen
Chat
Remote administration
Control interface
Input devices
Input device manager
Object-Tracker
Real-time kernel
User manager
DB-Server
Analysis


Challenging?
Level of detail
└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
└when do we stop?
How much detail?
Talked about more than the diagrams
What notation? - boxes
- arrows

- 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?

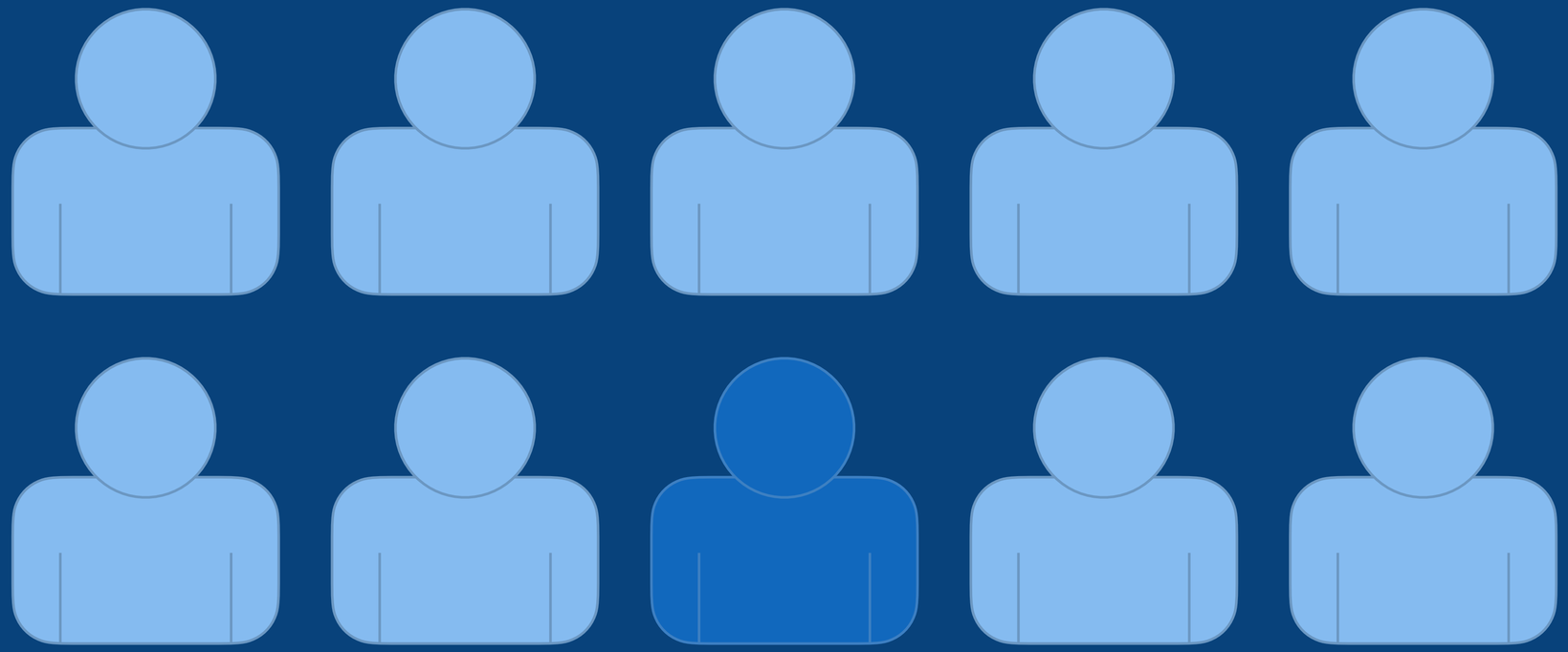


If you’re going to use “boxes & lines”, at least do so in a **structured way**, using a **self-describing notation**

Moving fast in the same direction
as a team requires

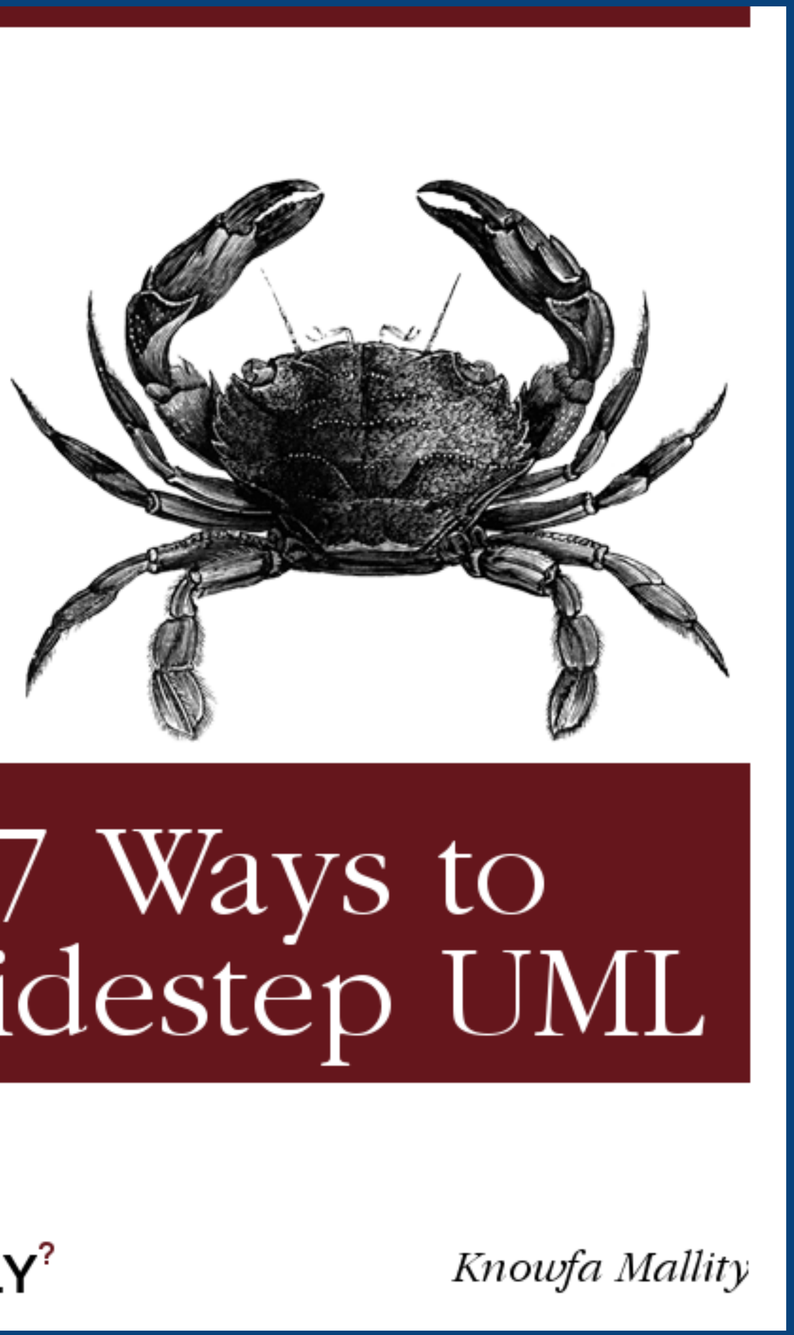
good communication

UML?



In my experience, optimistically,

1 out of 10 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 that!

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**

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

Grady Booch

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.

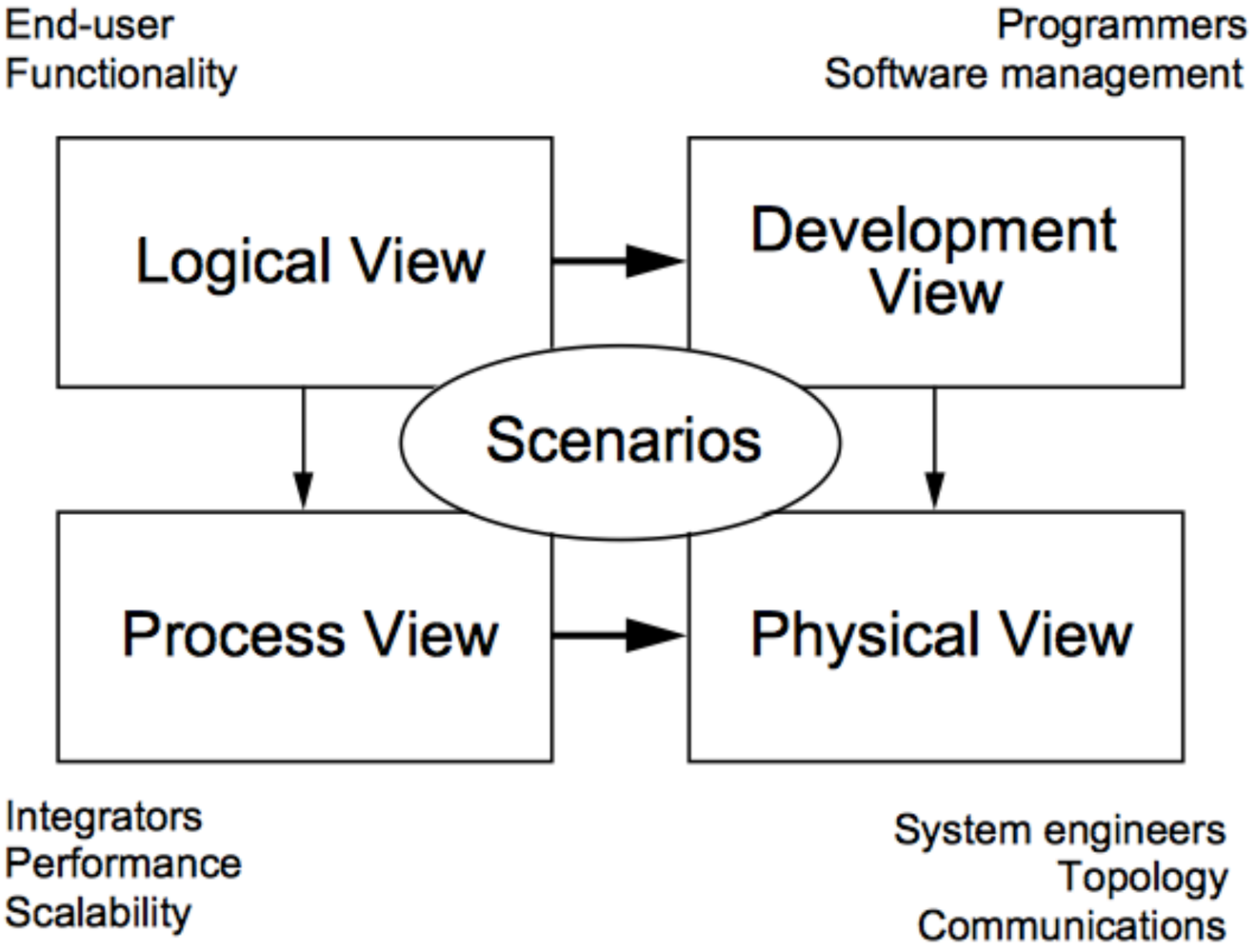
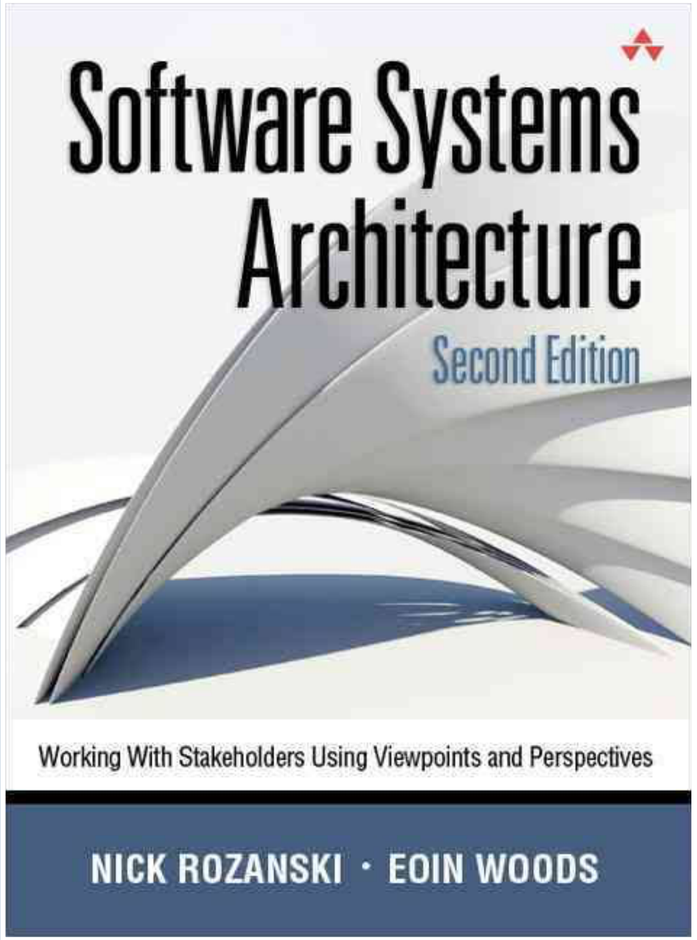
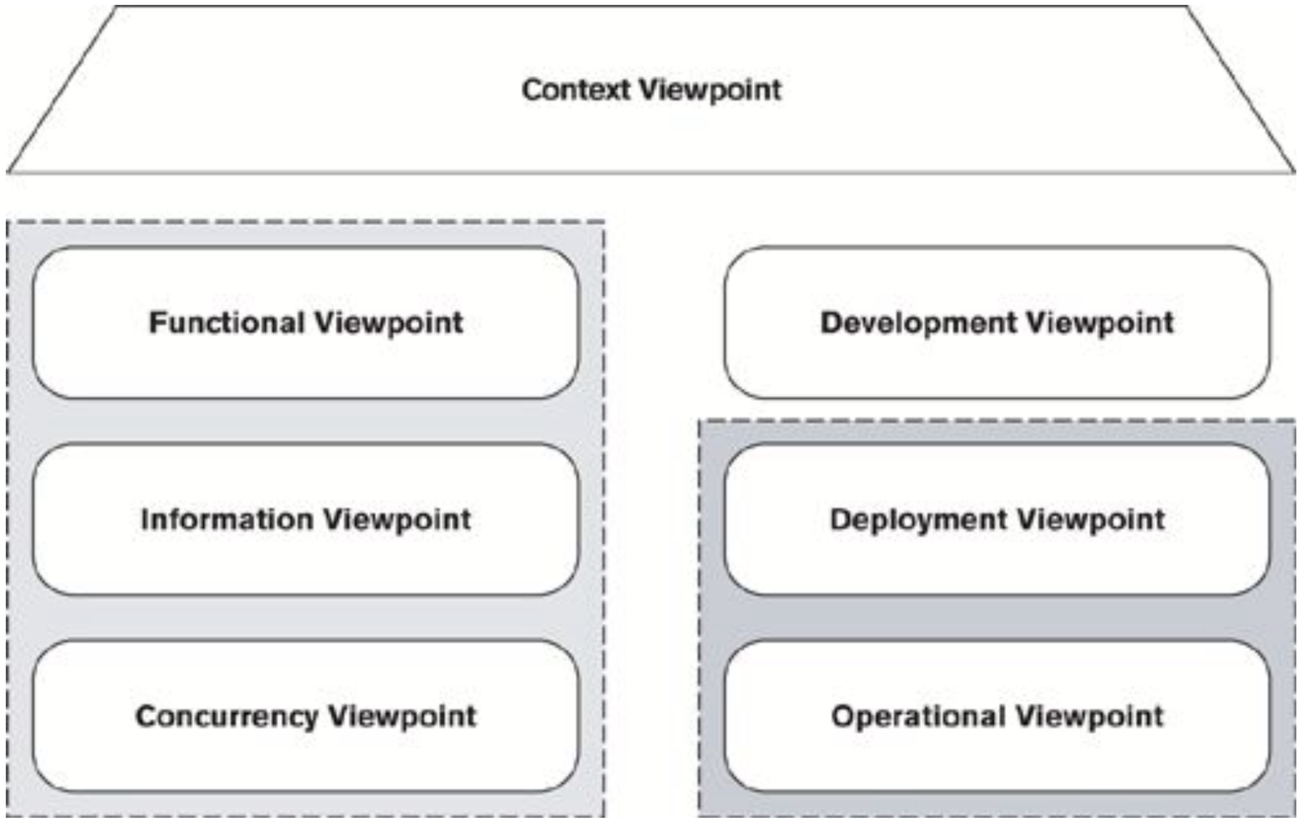


Figure 1 — The "4+1" view model

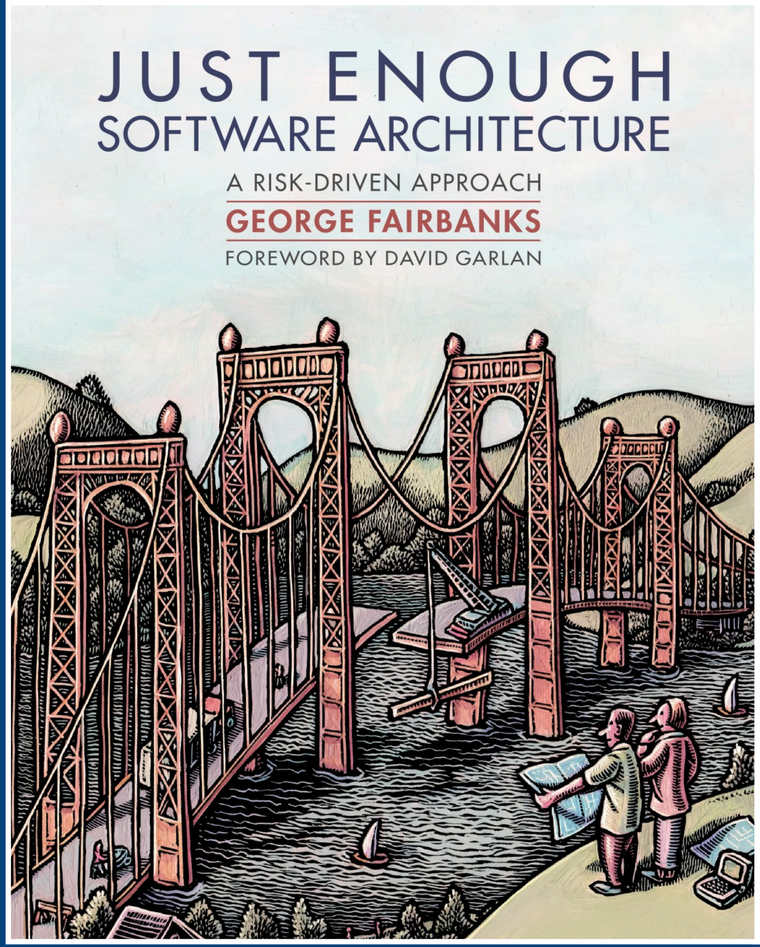




“Viewpoints and Perspectives”

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.

*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

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.

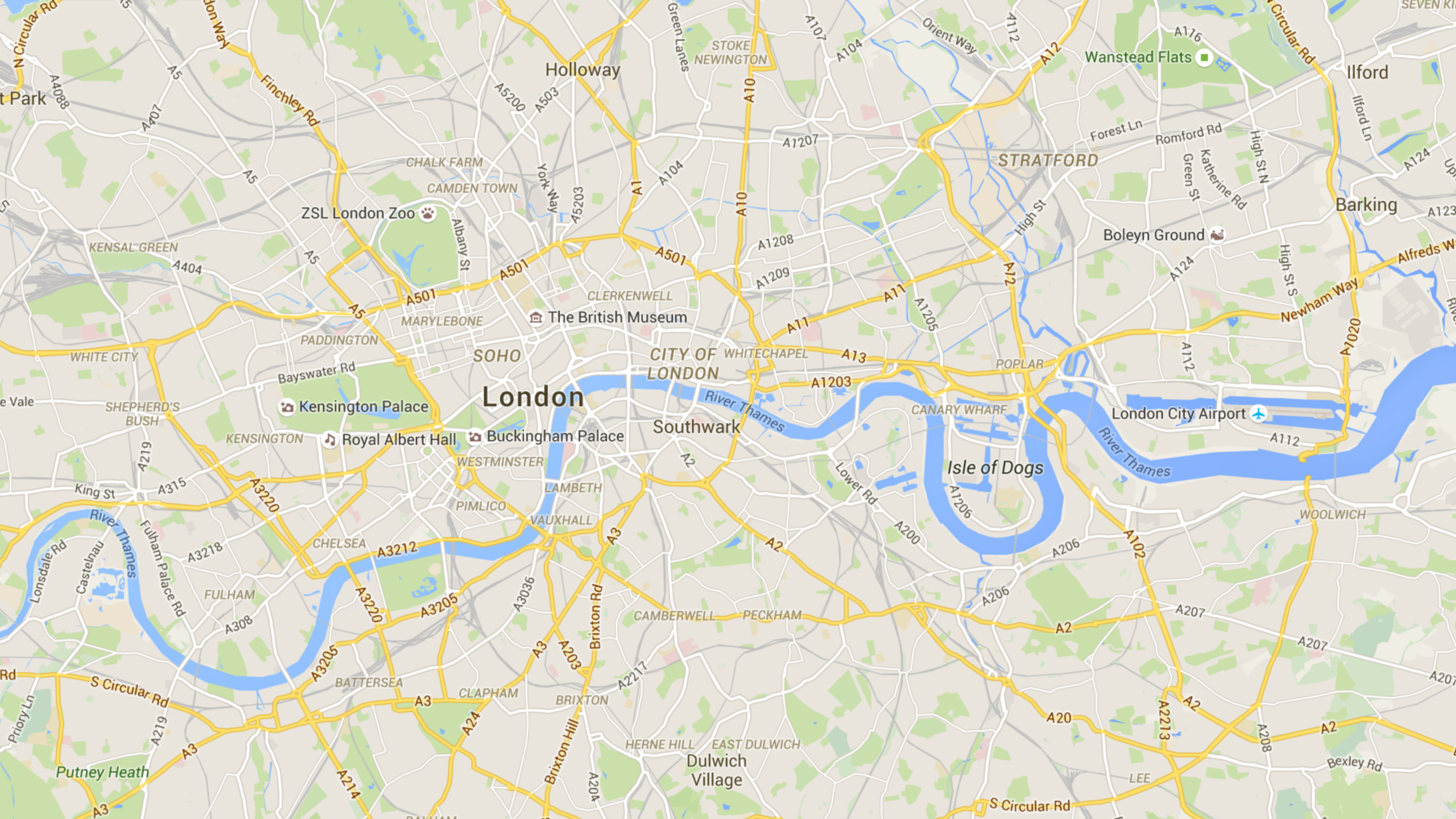
[1]The old English spelling differentiates our use of "reflexion" from the field of reflective computing [Smi84].

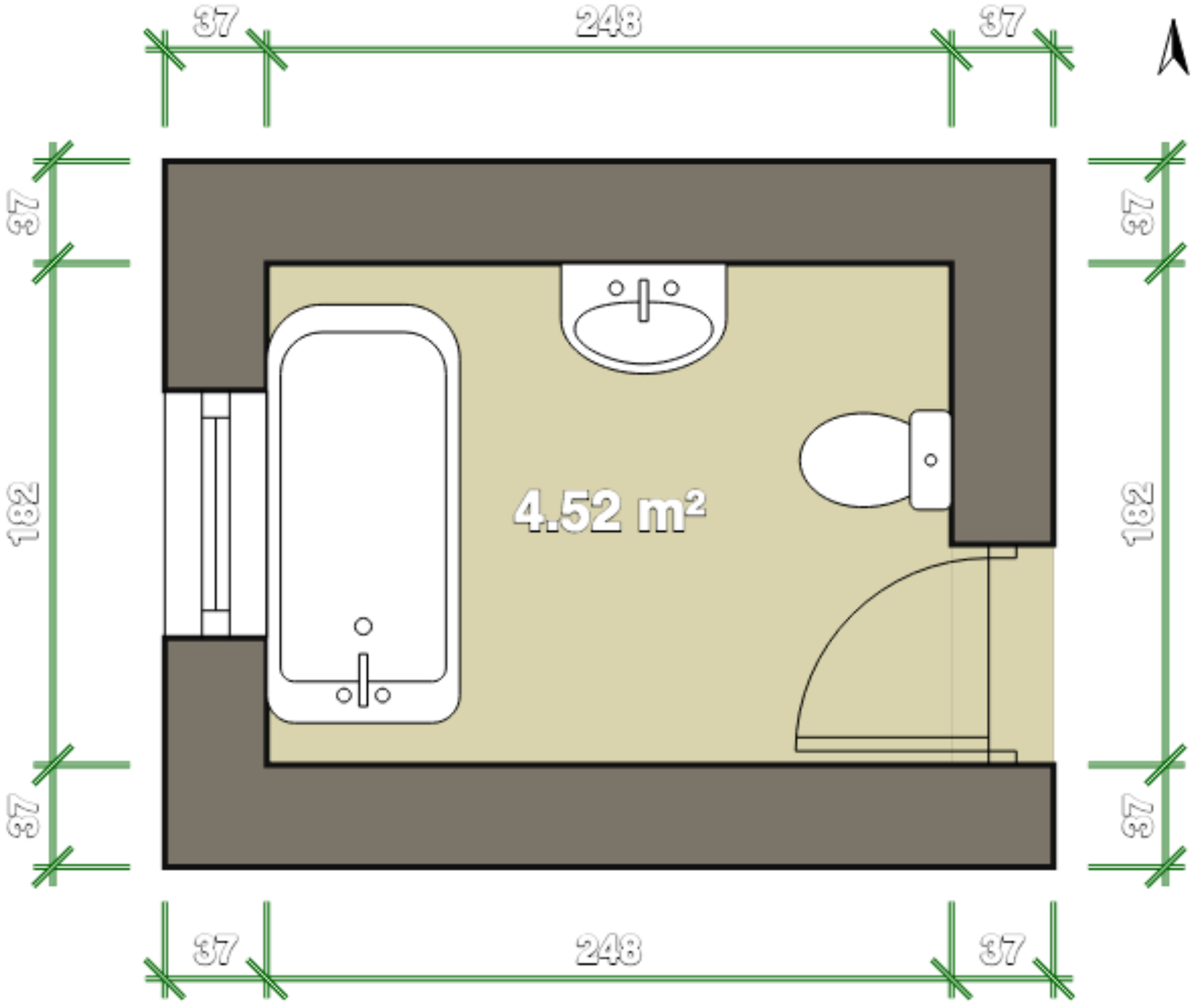
1 Introduction

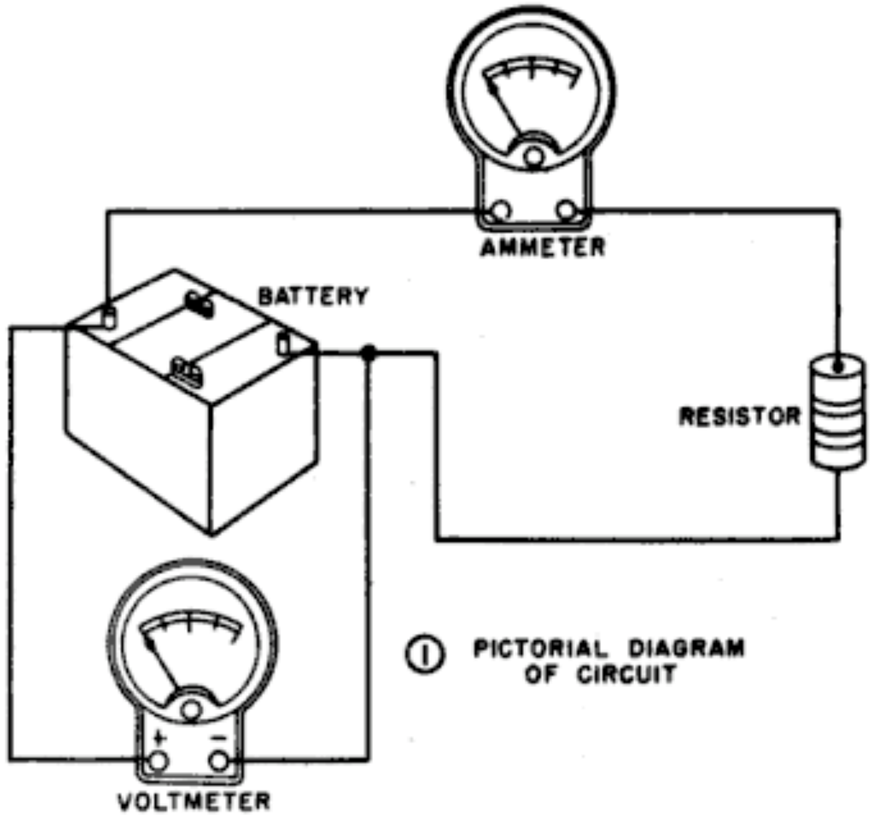
Software engineers often think about an existing software system in terms of high-level models. Box and arrow sketches of a system, for instance, are often found on engineers' whiteboards. Although these models are commonly used, reasoning about the system in terms of such models can be dangerous because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


London
Holloway
STOKE NEWINGTON
Wanstead Flats
Ilford
Barking
STRATFORD
CHALK FARM
CAMDEN TOWN
ZSL London Zoo
Boleyn Ground
KENSAL GREEN
MARYLEBONE
The British Museum
CLERKENWELL
PADDINGTON
WHITE CITY
SOHO
CITY OF LONDON
WHITECHAPEL
POPLAR
Bayswater Rd
Kensington Palace
SHEPHERD'S BUSH
KENSINGTON
Royal Albert Hall
Buckingham Palace
WESTMINSTER
Southwark
River Thames
CANARY WHARF
London City Airport
Isle of Dogs
LAMBETH
PIMLICO
VAUXHALL
CHELSEA
FULHAM
CAMBERWELL
PECKHAM
BATTERSEA
CLAPHAM
BRIXTON
HERNE HILL
EAST DULWICH
Dulwich Village
Putney Heath
WOOLWICH
LEE
Bexley Rd
Finchley Rd
Albany St
York Way
Green Lanes
Orient Way
Forest Ln
Romford Rd
Green St
Katherine Rd
High St
High St N
High St S
Ilford Ln
Newham Way
Alfreds W
Lower Rd
Brixton Rd
Brixton Hill
Fulham Palace Rd
Lonsdale Rd
Castelnau
Priory Ln
King St
S Circular Rd
N Circular Rd
A1
A2
A3
A5
A10
A11
A12
A13
A20
A24
A102
A104
A107
A112
A116
A124
A200
A201
A203
A204
A206
A207
A208
A214
A219
A308
A315
A404
A407
A501
A1020
A1203
A1205
A1206
A1207
A1208
A1209
A2213
A2217
A3036
A3205
A3212
A3218
A3220
A4088
A5200
A5203


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



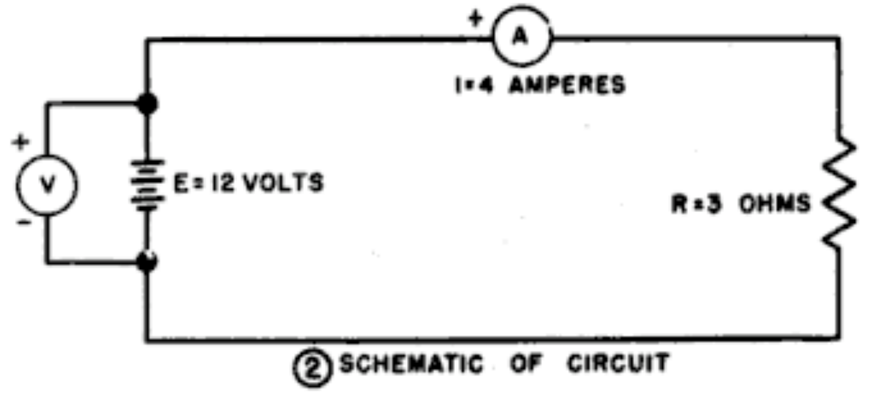
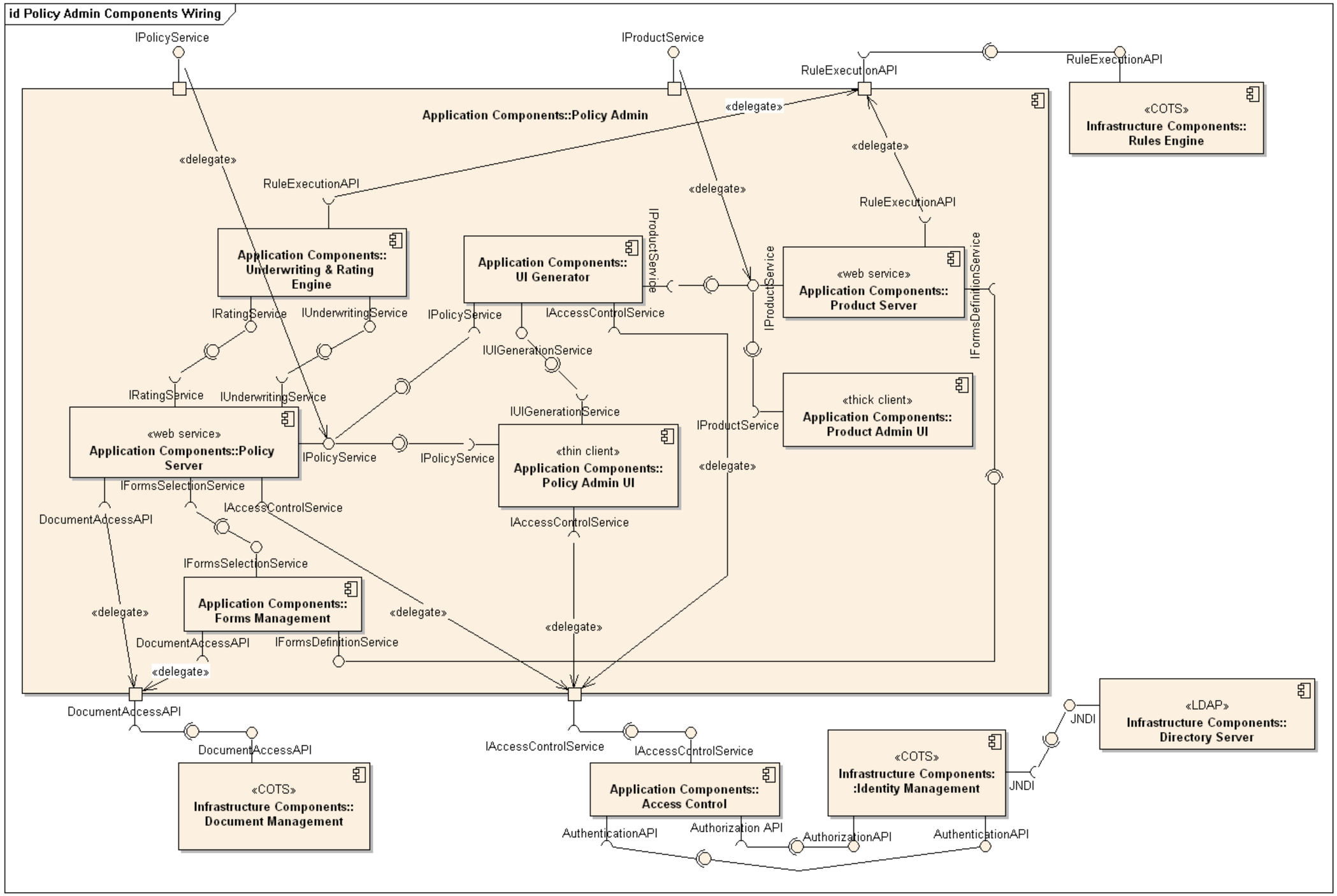
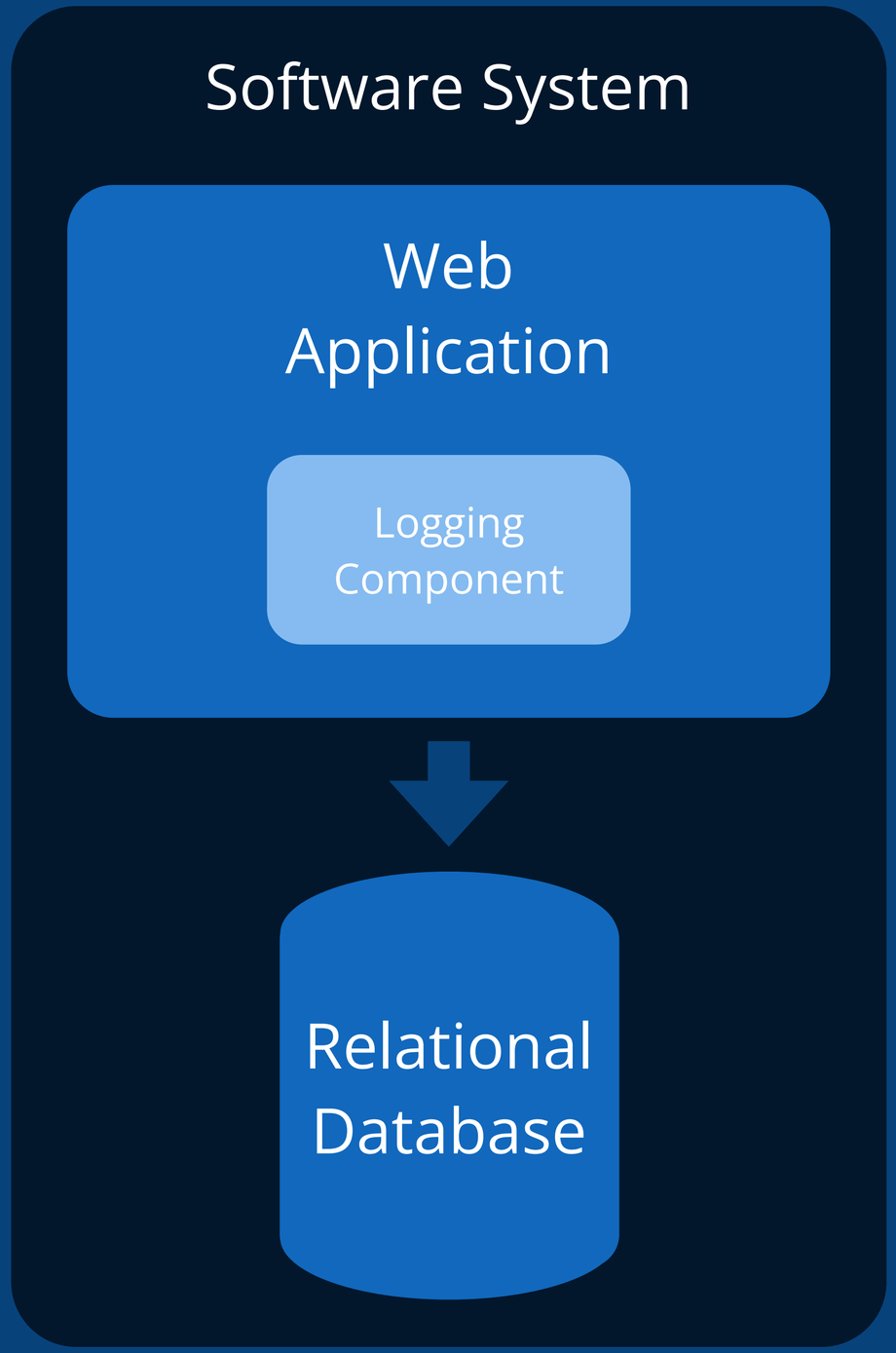


Figure 48. Diagram of a basic circuit.

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



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



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

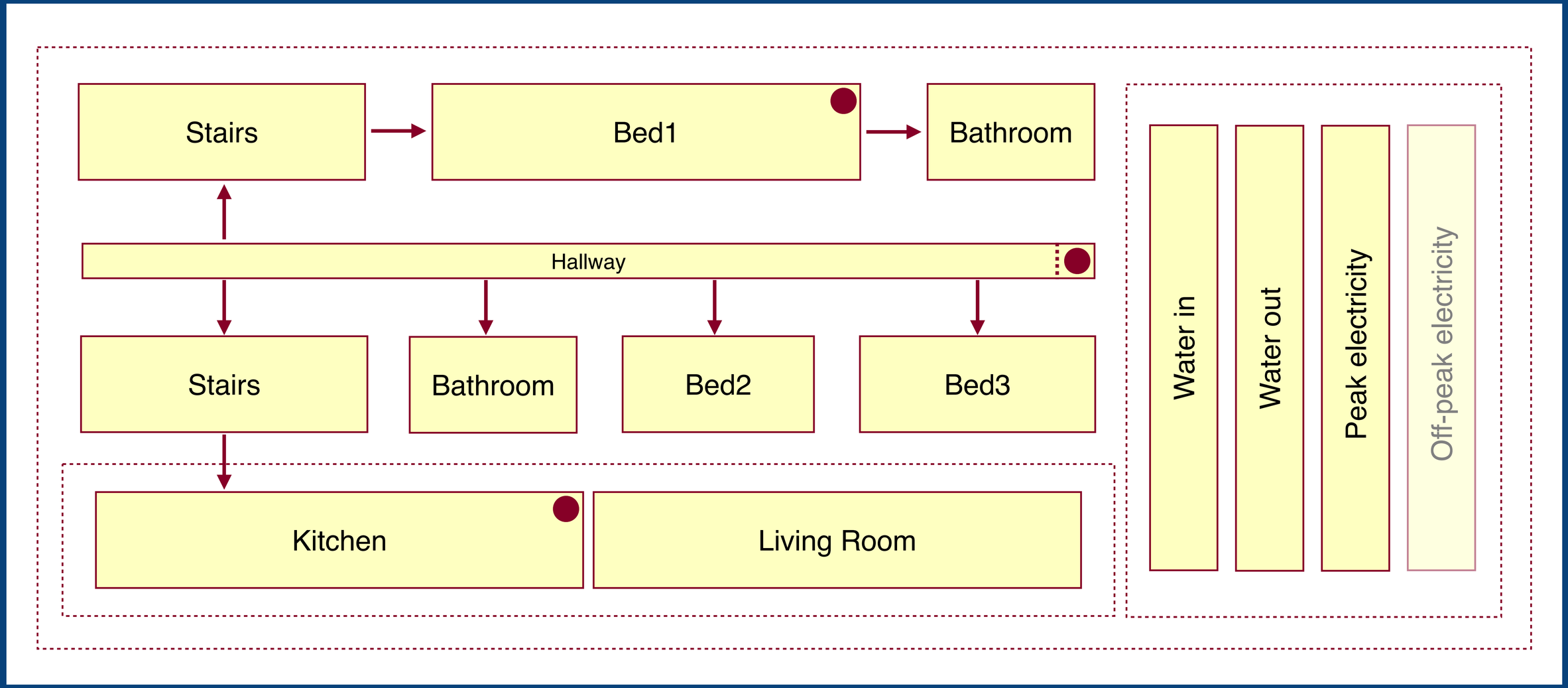
Ubiquitous language

Would you code it that way?

(ensure that your diagrams reflect
your implementation intent)

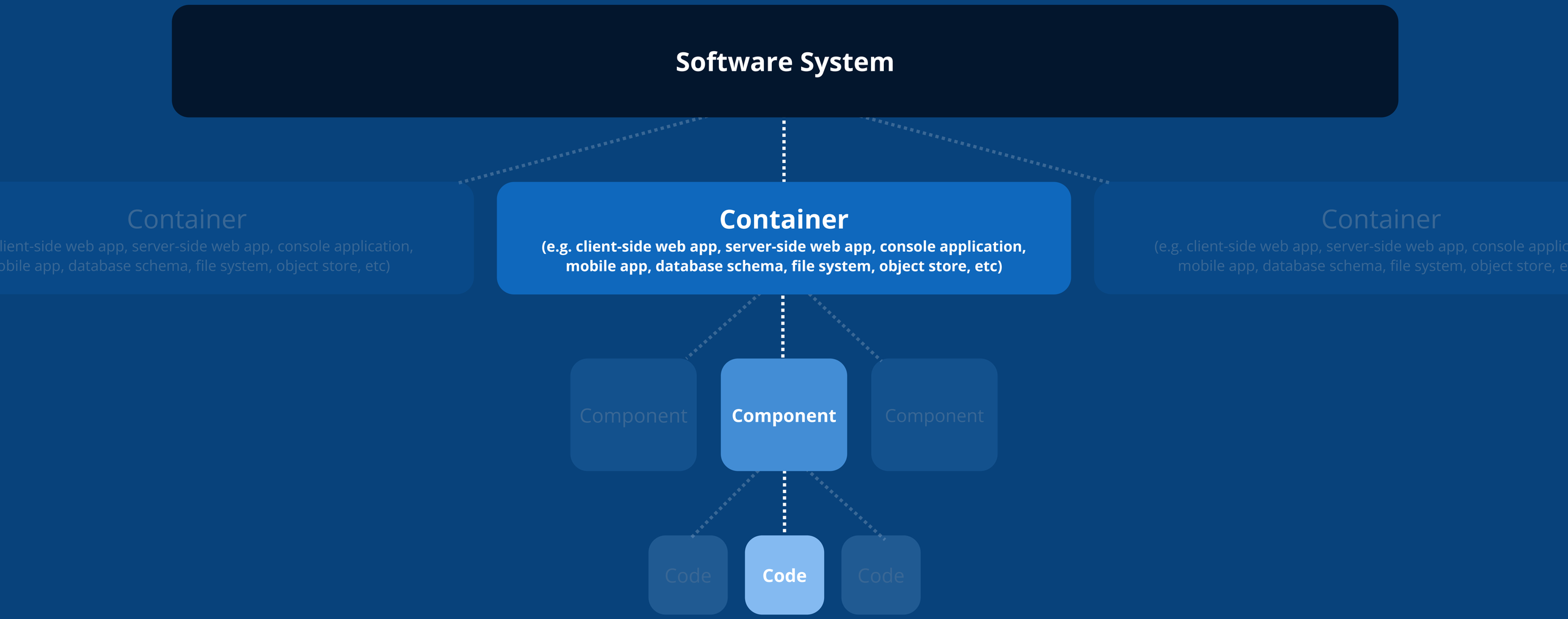
When drawing software architecture diagrams, think like a software developer

If software developers created building architecture diagrams...



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

Abstractions



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).

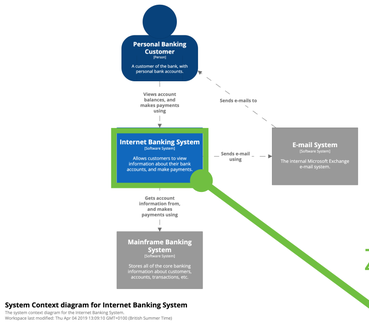
Static structure diagrams

C4

c4model.com

The C4 model for visualising software architecture

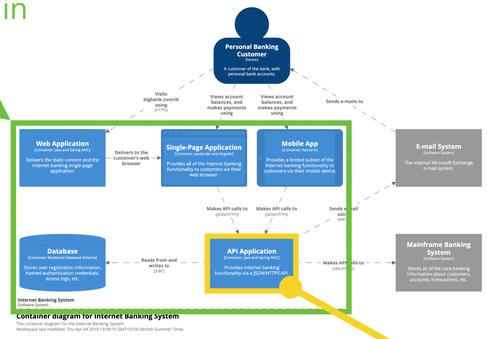
c4model.com



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)

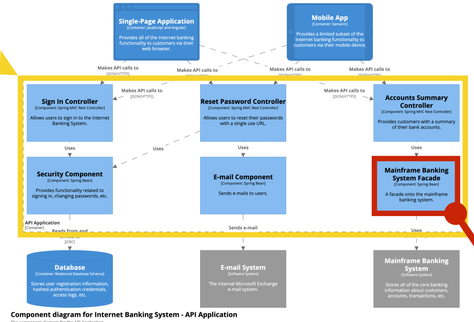
Zoom in



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)

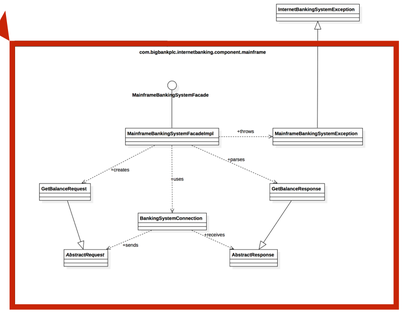
Zoom in



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



Level 1
Context

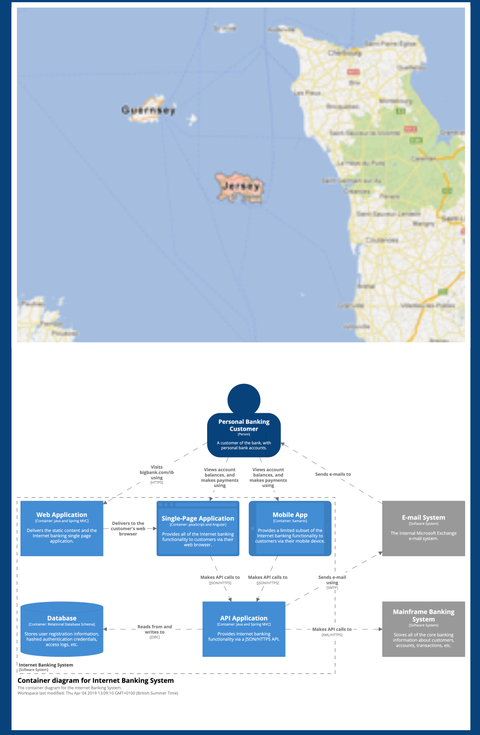
Level 2
Containers

Level 3
Components

Level 4
Code

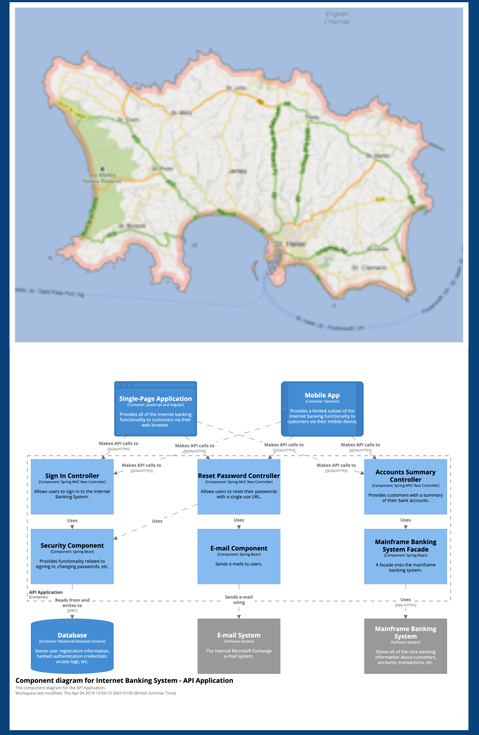
System Context diagram for Internet Banking System

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



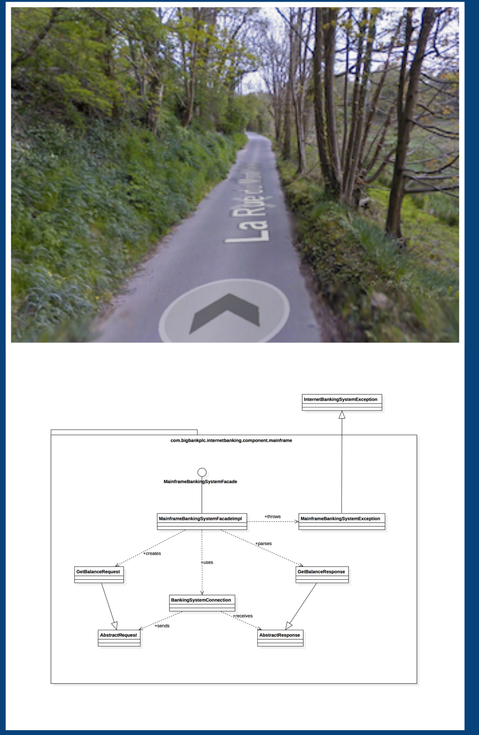
Container diagram for Internet Banking System

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



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)



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.

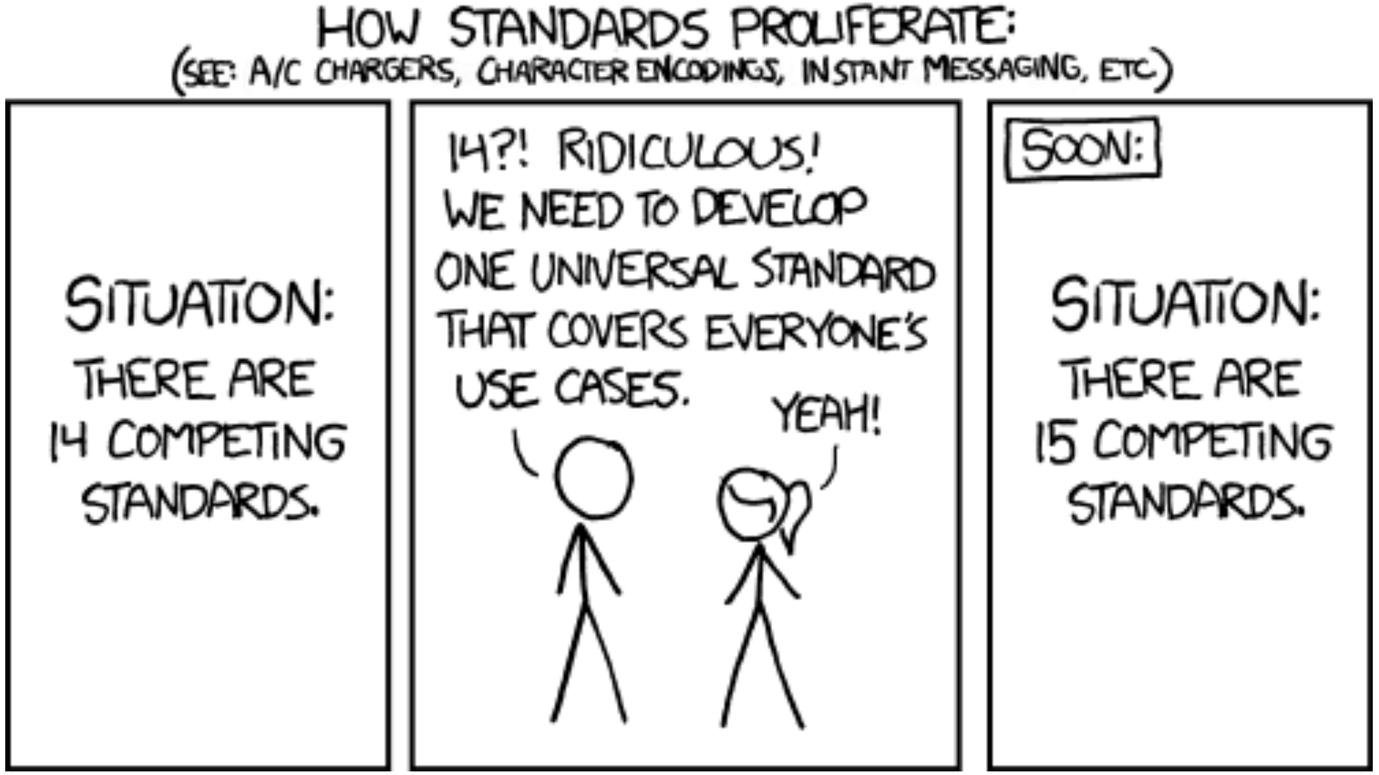
4. Code (e.g. classes)

Component implementation details.

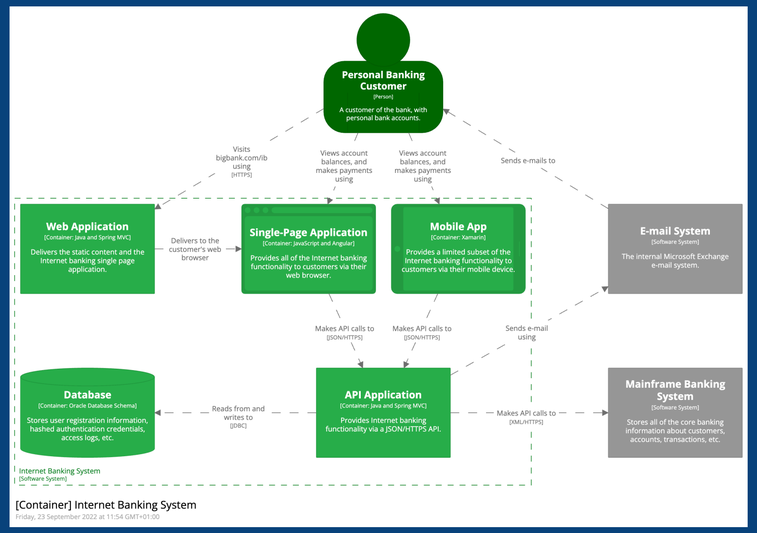
Overview first

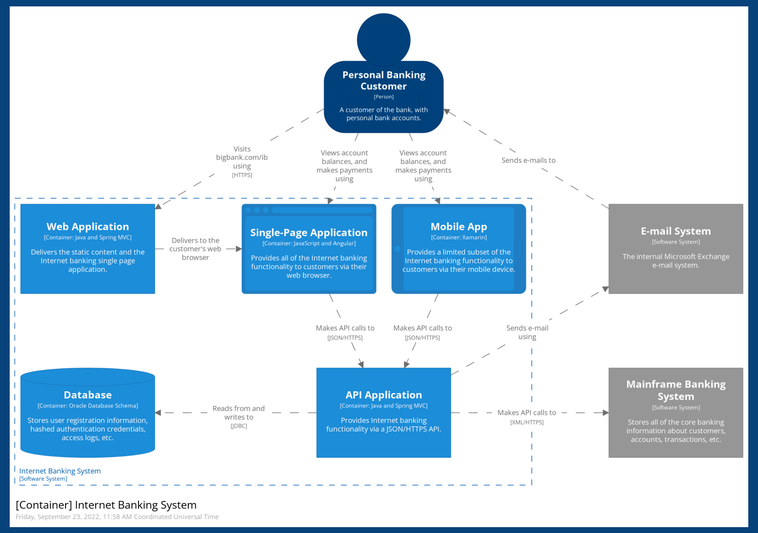
Zoom & filter

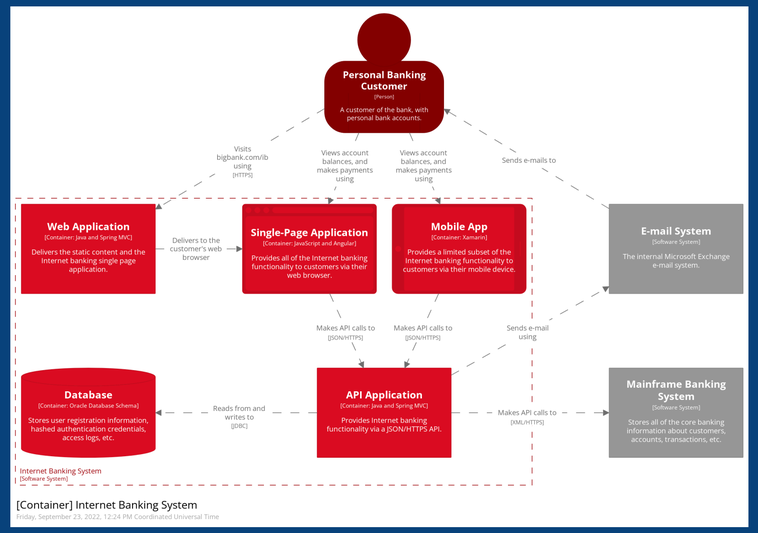
Details on demand


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.

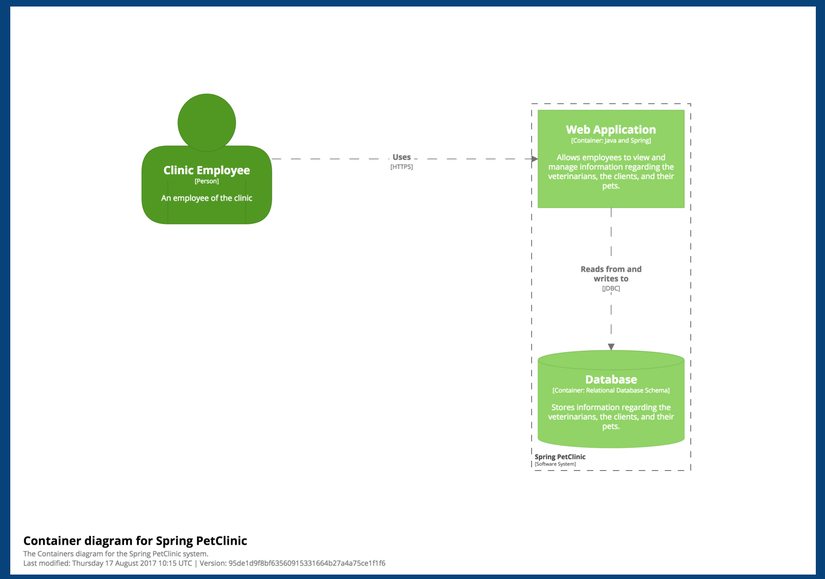
The C4 model is **notation independent**

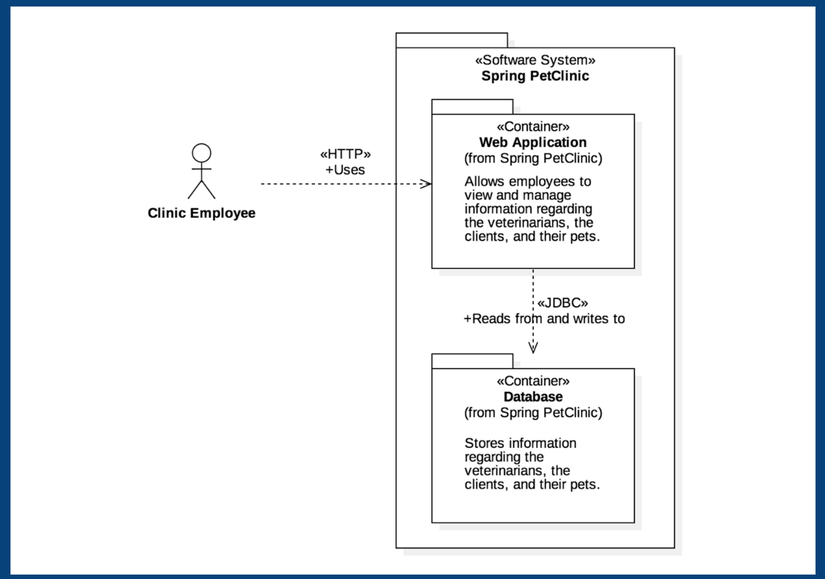


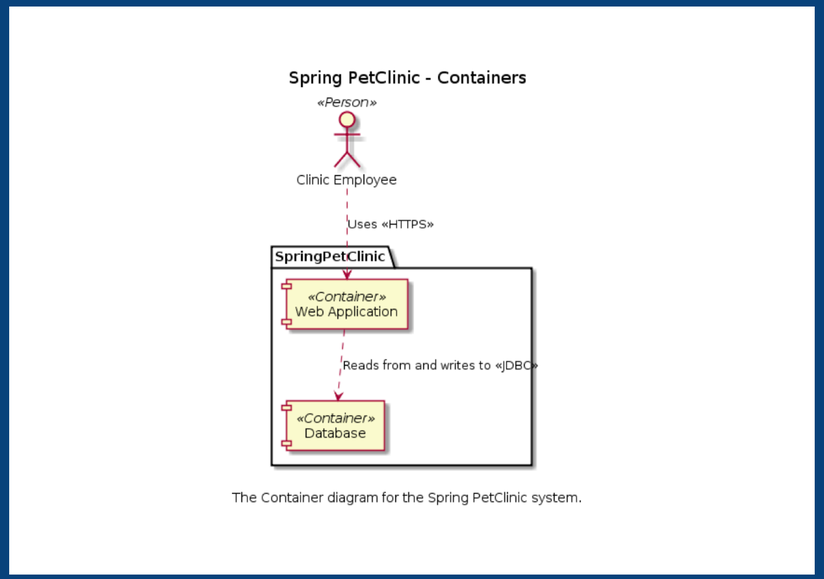




The C4 model is **notation independent**







Example

(Internet Banking System)

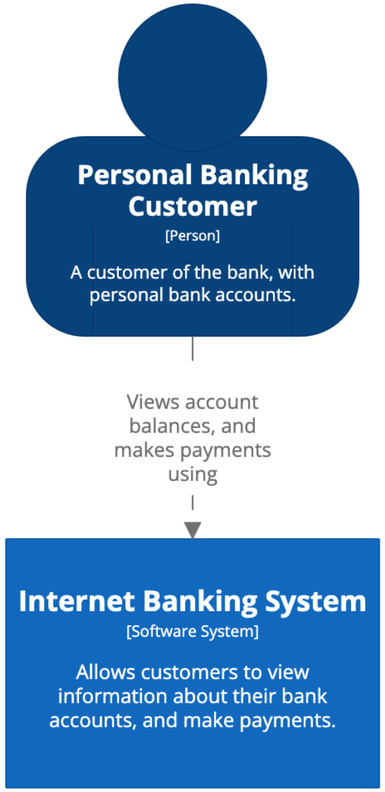
Level 1

System Context diagram

[System Context] Internet Banking System

The system context diagram for the Internet Banking System.

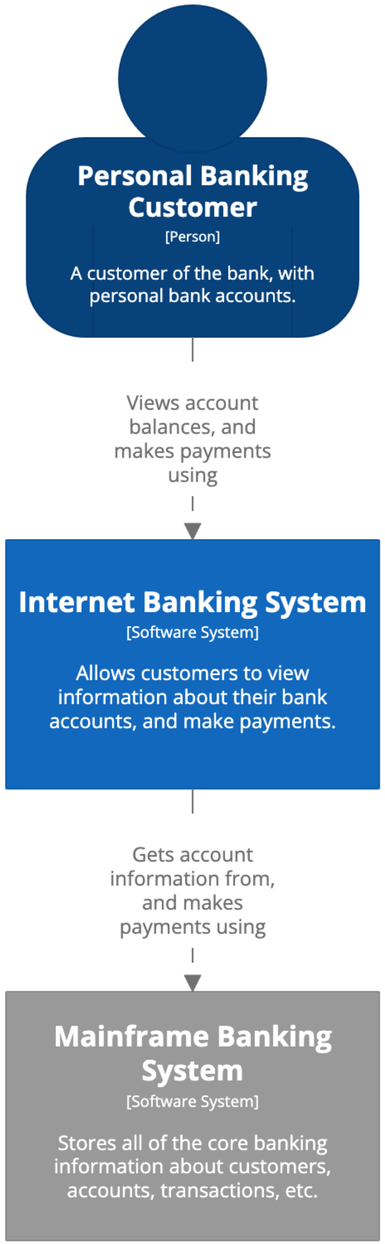
Monday, 27 February 2023 at 15:25 Greenwich Mean Time



[System Context] Internet Banking System

The system context diagram for the Internet Banking System.

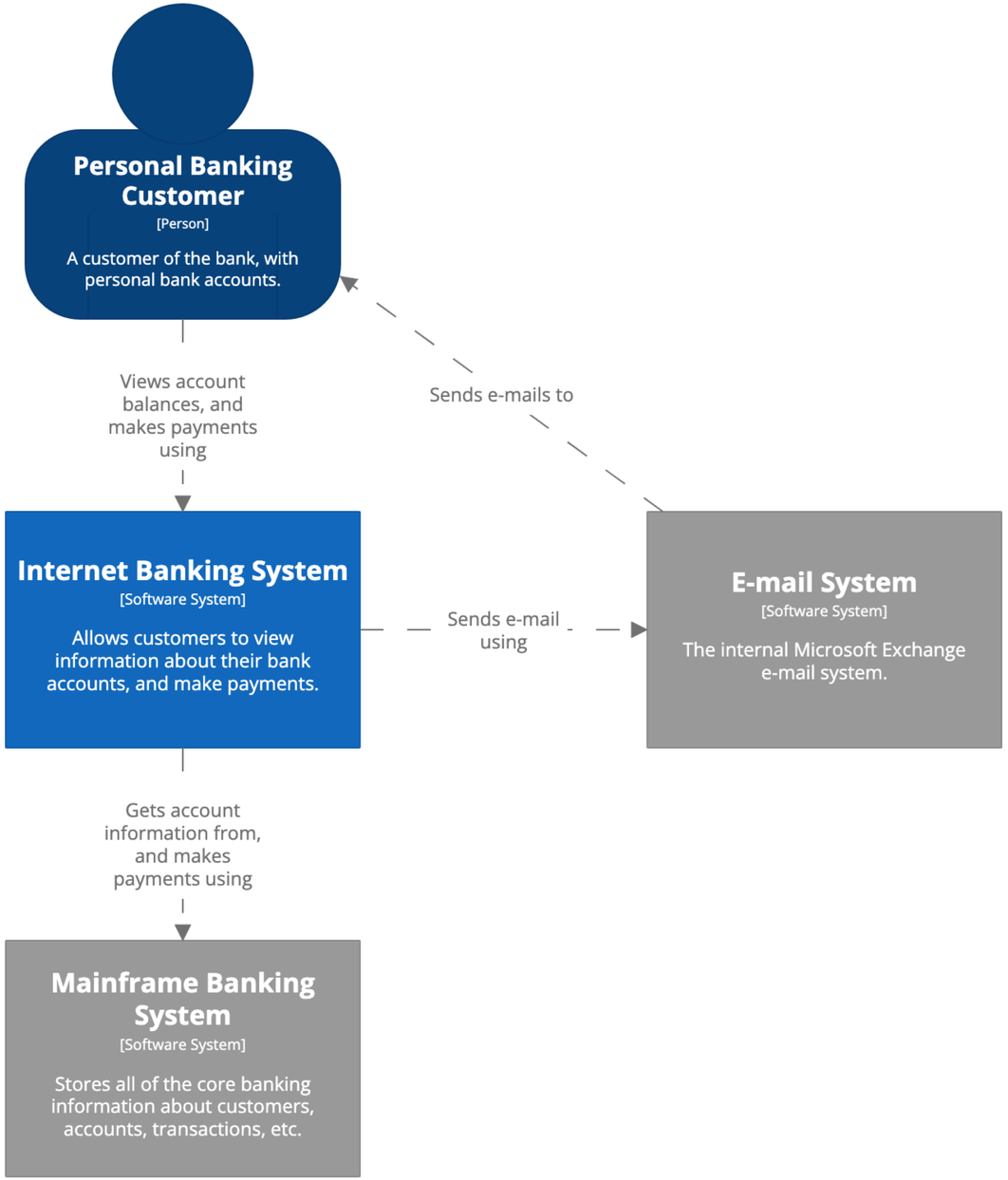
Monday, 27 February 2023 at 15:25 Greenwich Mean Time



[System Context] Internet Banking System

The system context diagram for the Internet Banking System.

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

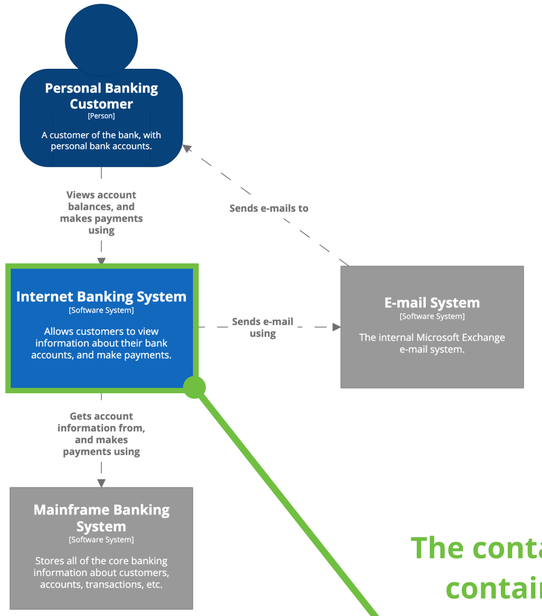


[System Context] Internet Banking System

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

Level 2

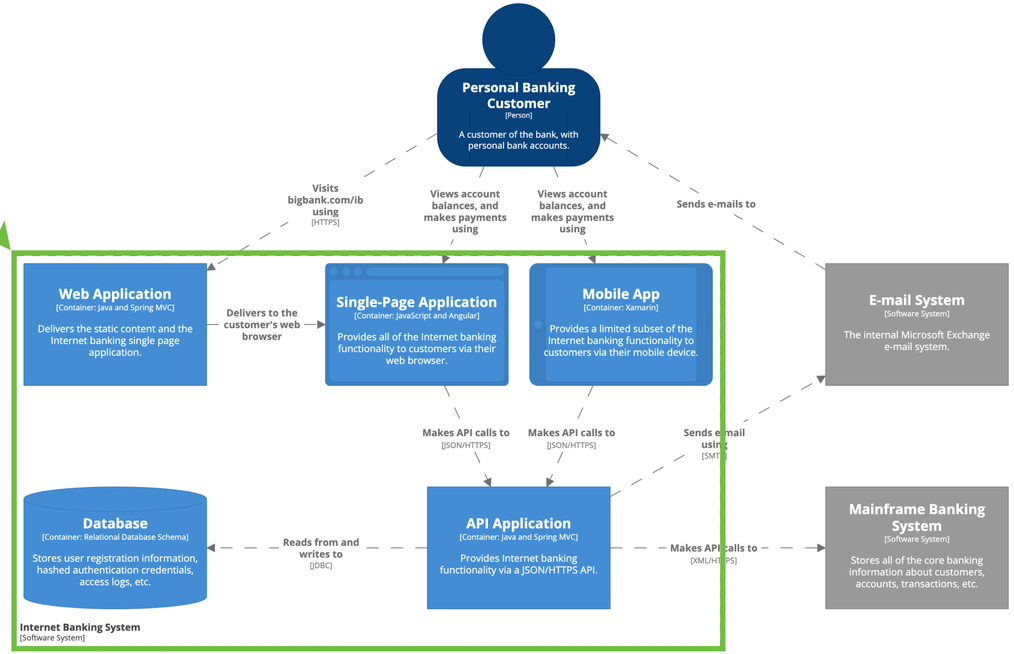
Container diagram



System Context diagram for Internet Banking System

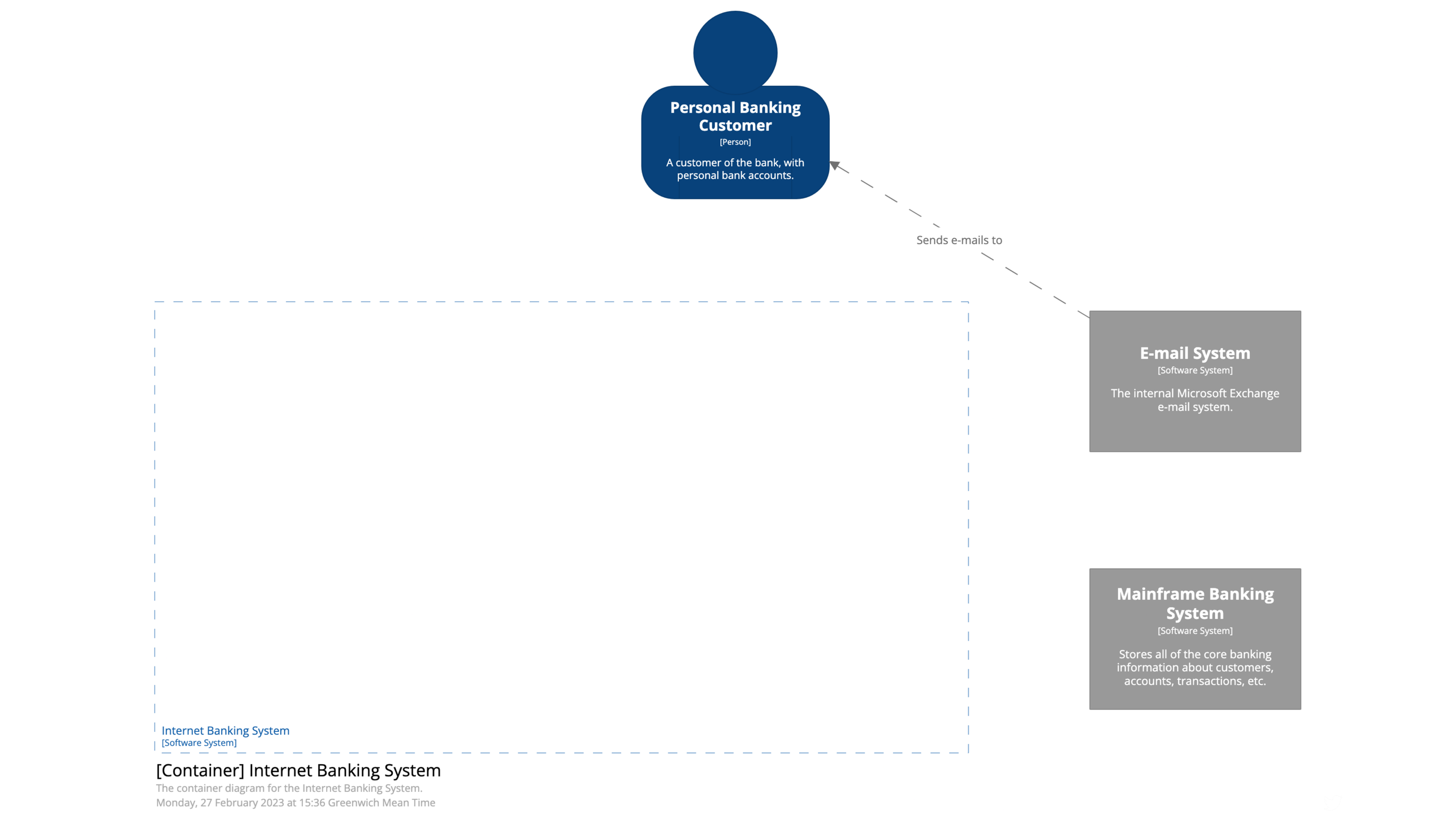
The system context diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)

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



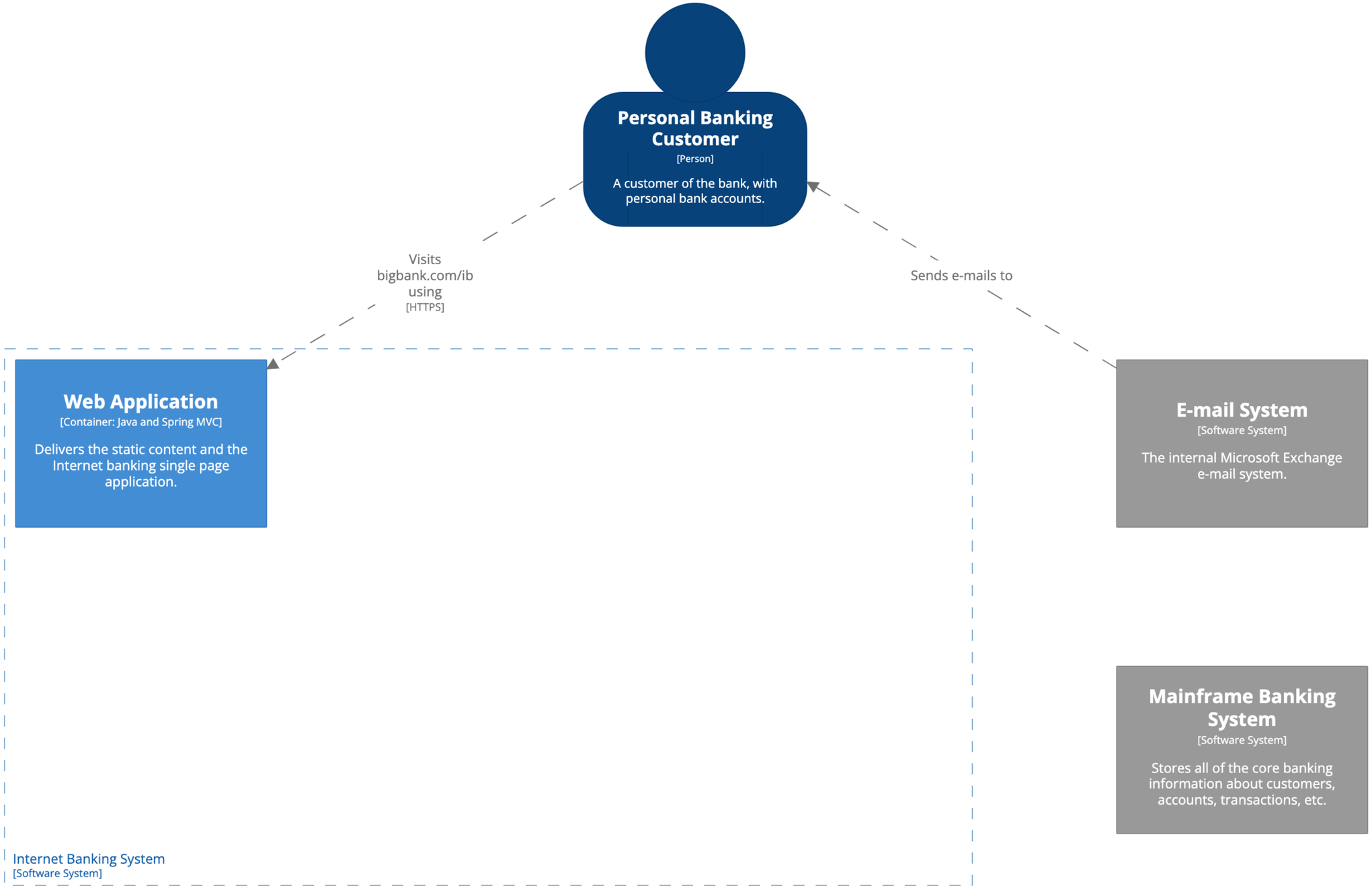
Container diagram for Internet Banking System

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



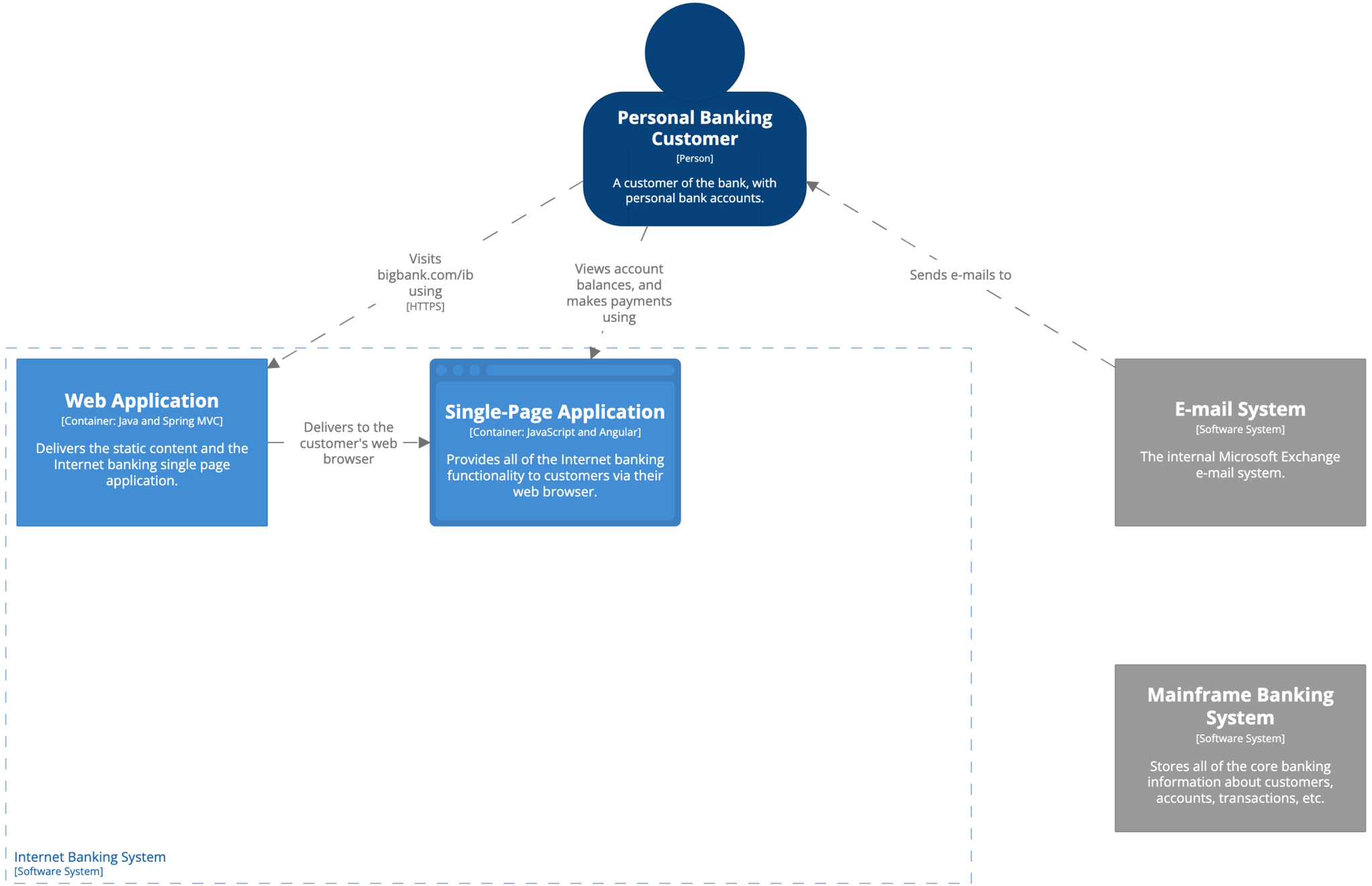
[Container] Internet Banking System

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



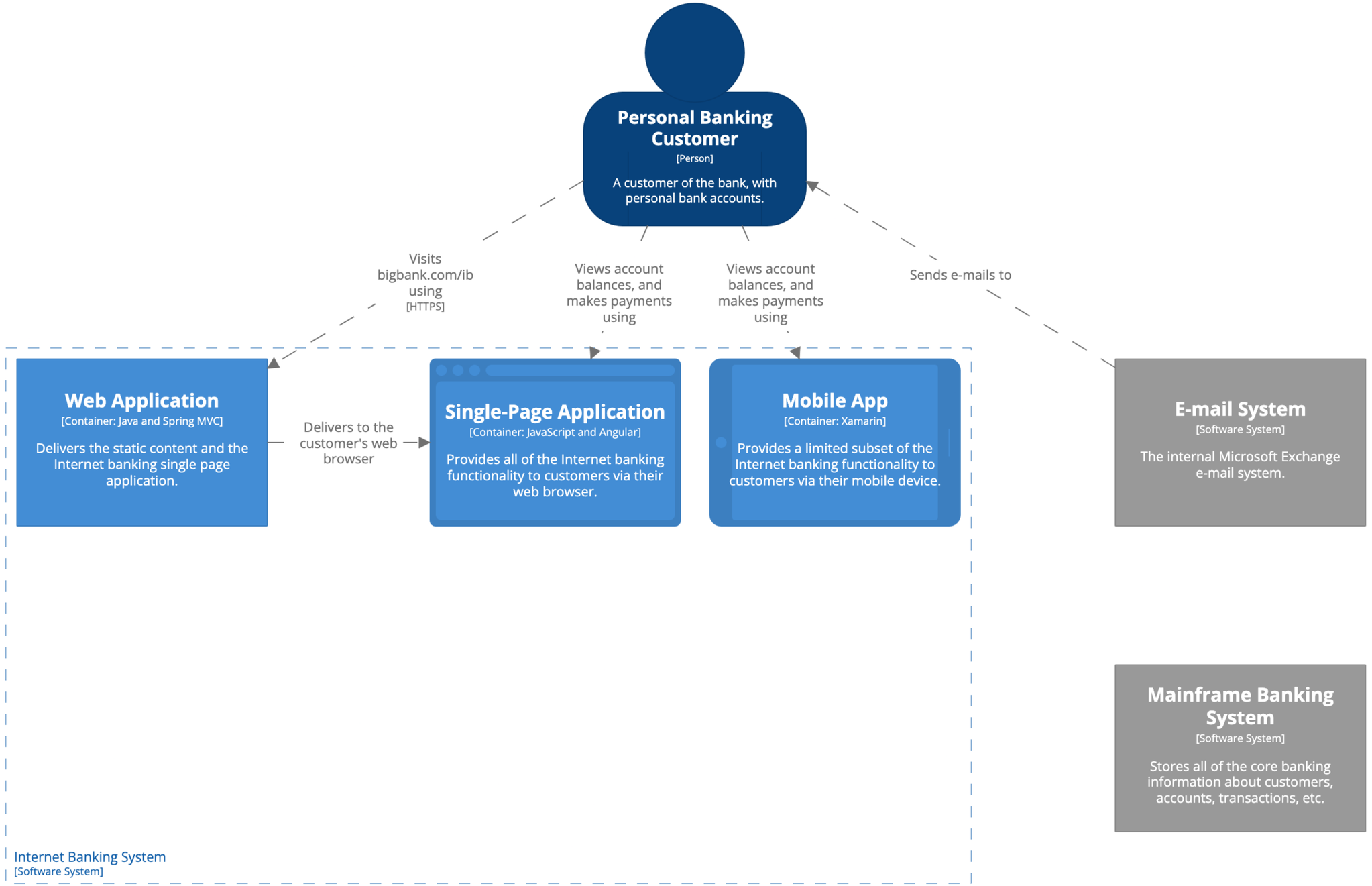
[Container] Internet Banking System

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



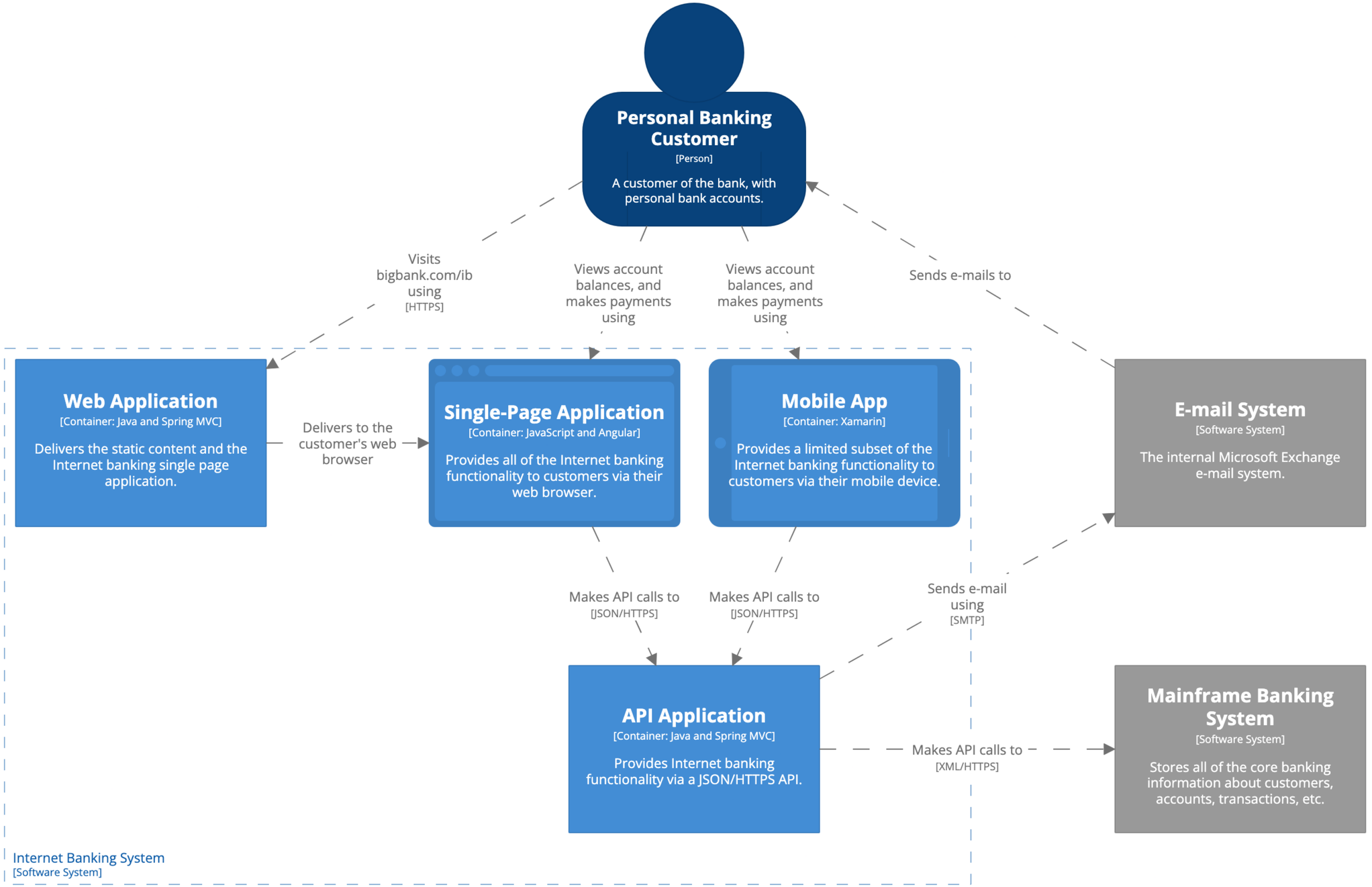
[Container] Internet Banking System

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



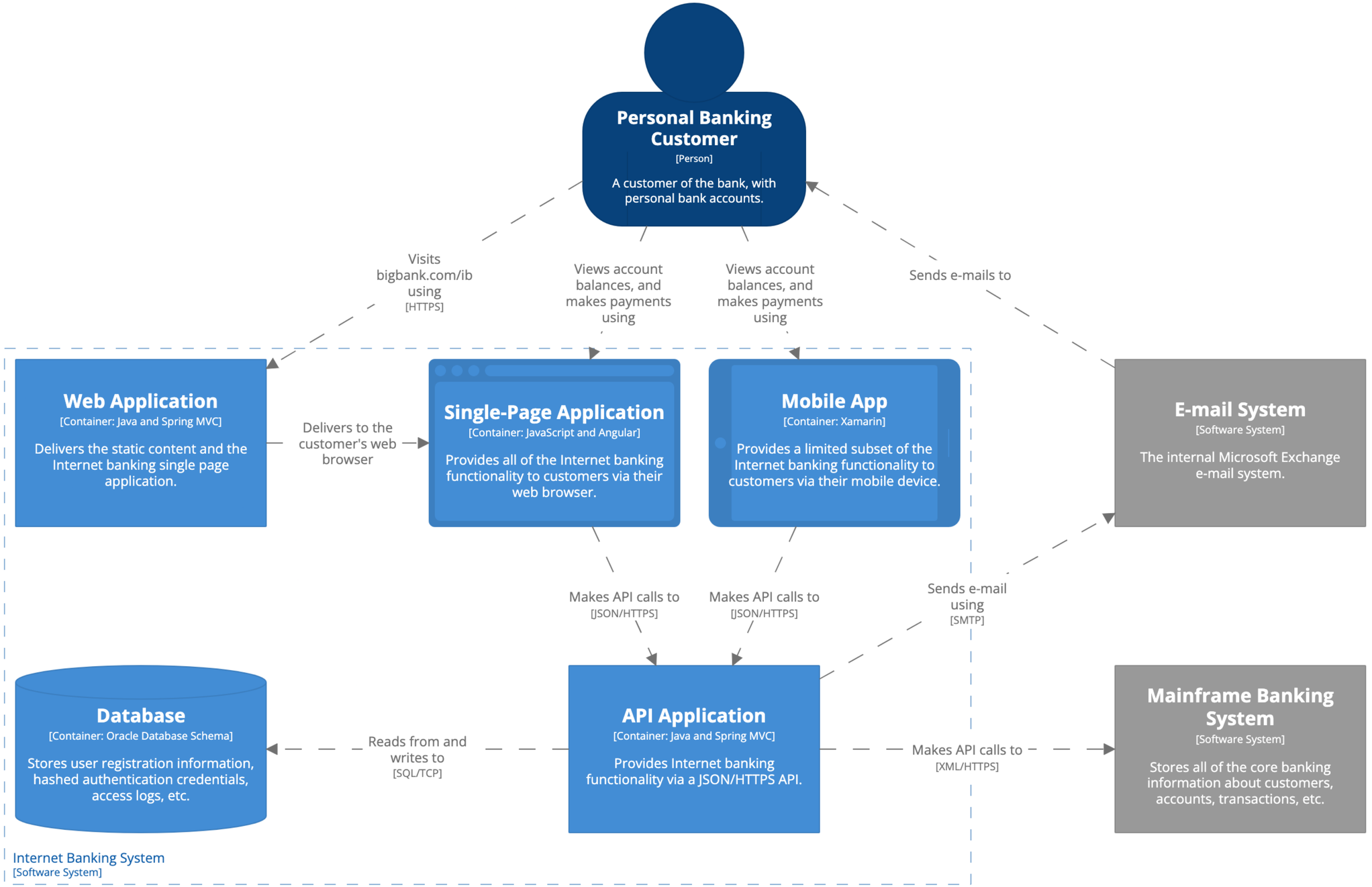
[Container] Internet Banking System

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



[Container] Internet Banking System

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

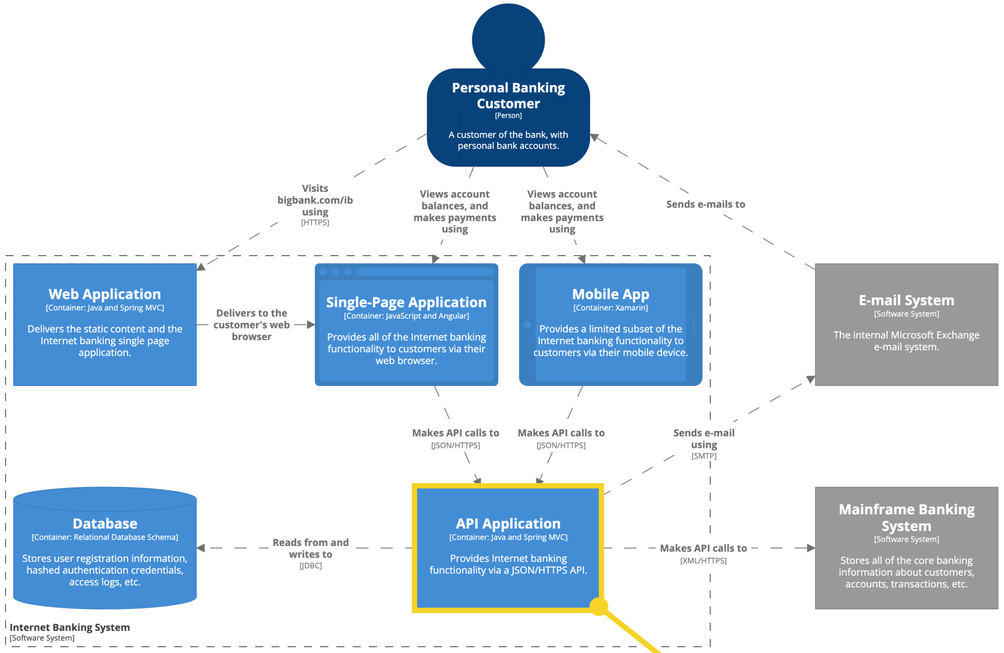


[Container] Internet Banking System

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

Level 3

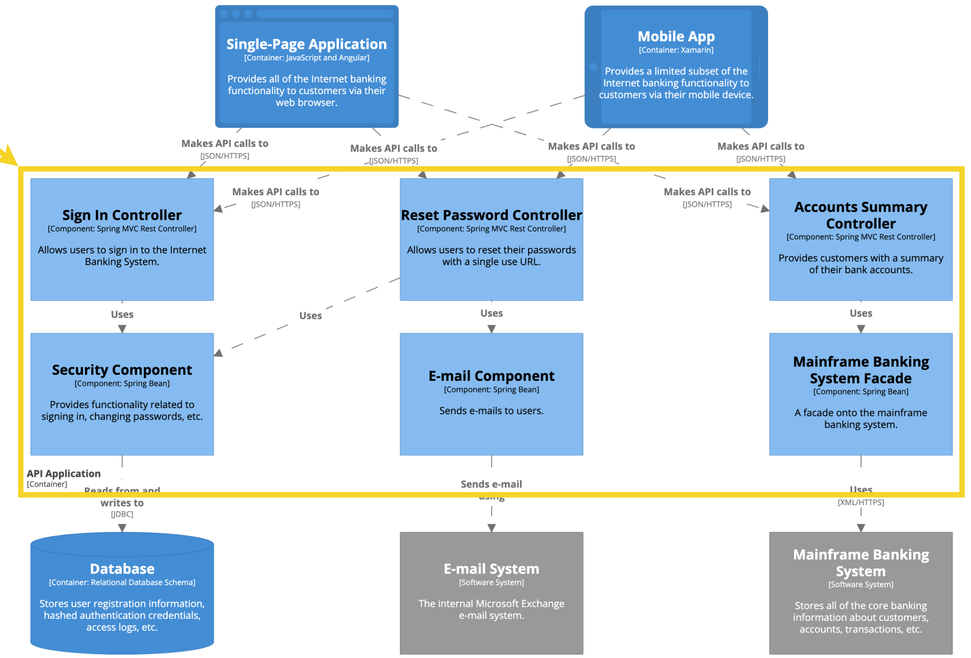
Component diagram



Container diagram for Internet Banking System

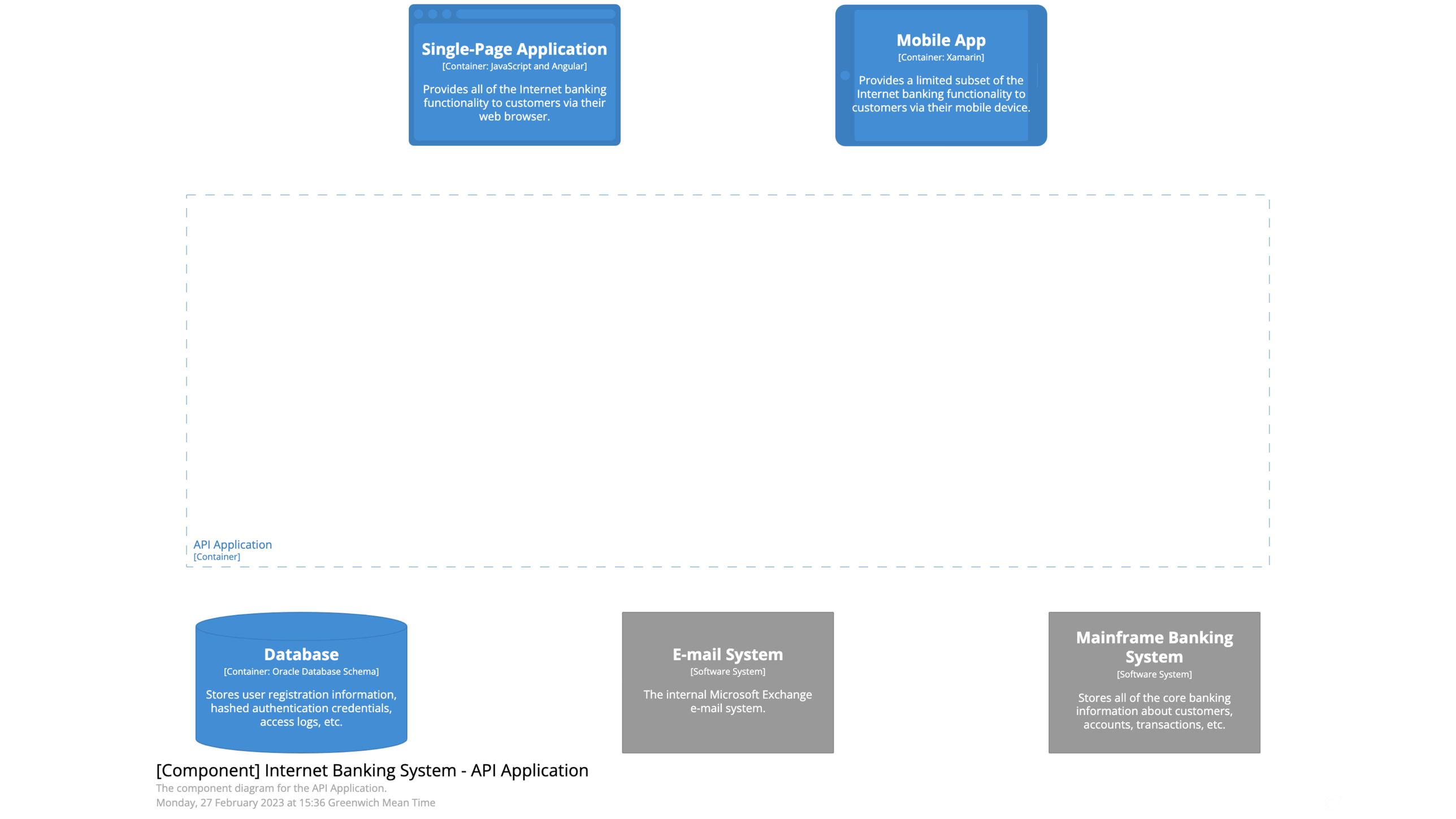
The container diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)

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



Component diagram for Internet Banking System - API Application

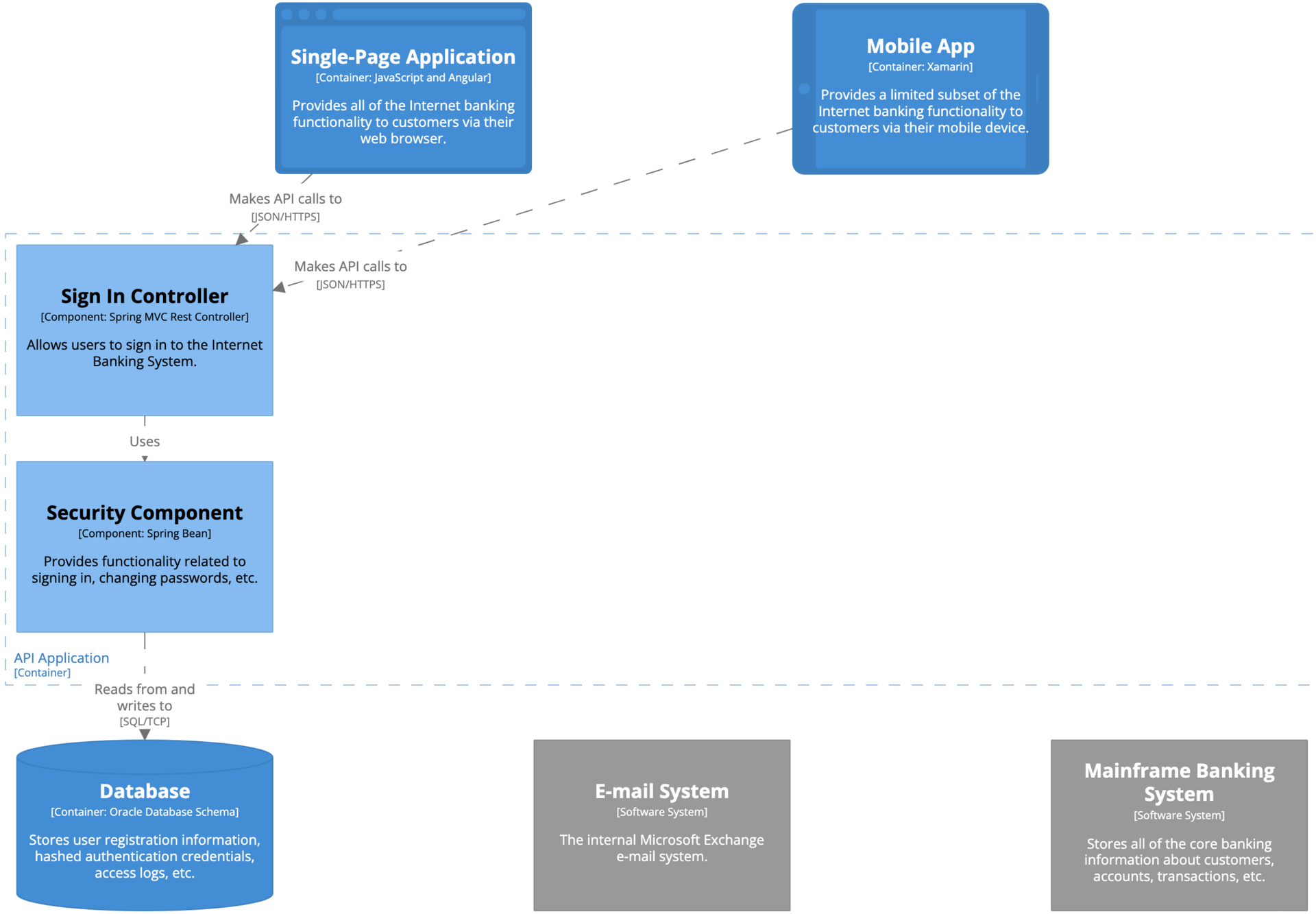
The component diagram for the API Application.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)



[Component] Internet Banking System - API Application

The component diagram for the API Application.

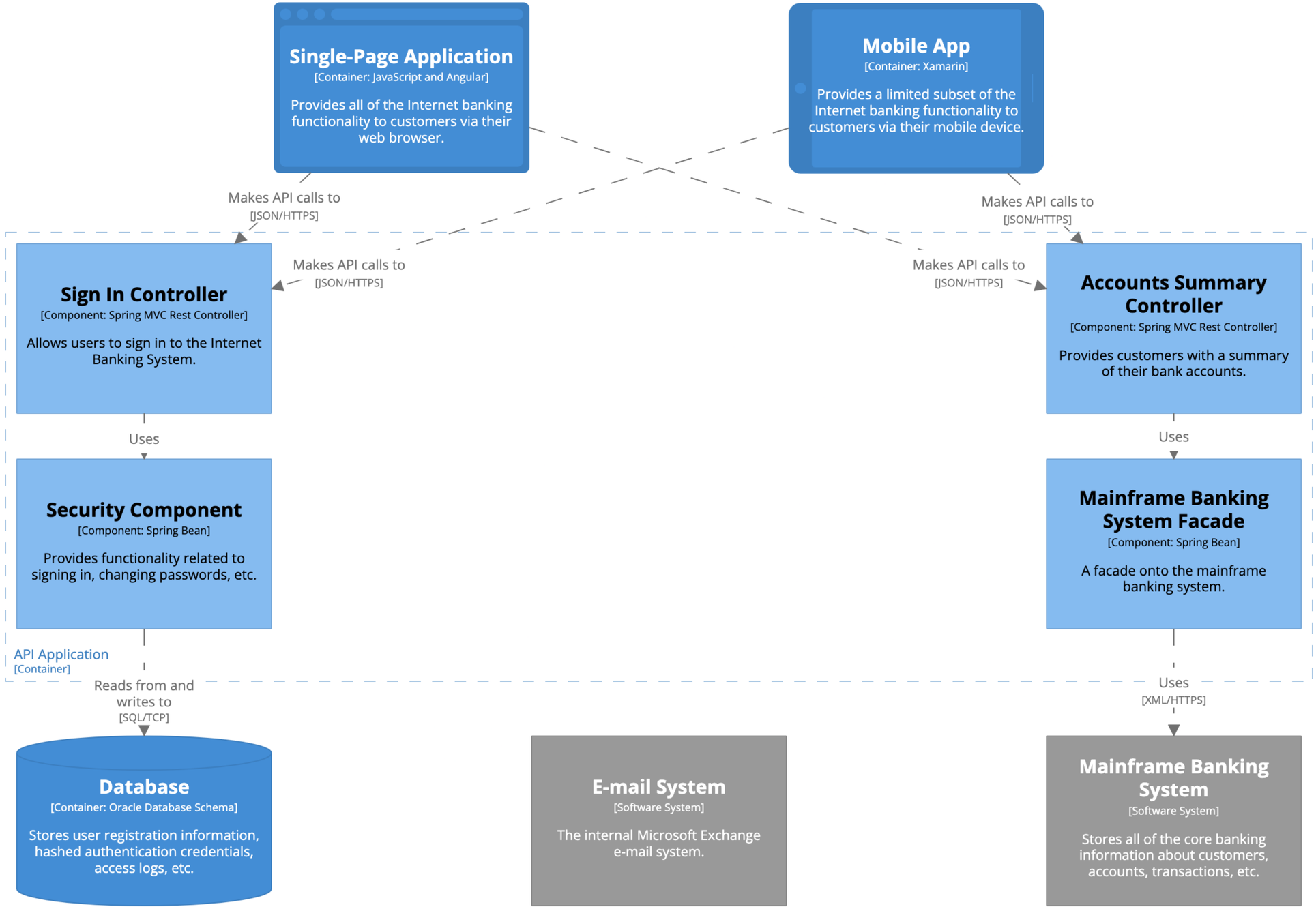
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



[Component] Internet Banking System - API Application

The component diagram for the API Application.

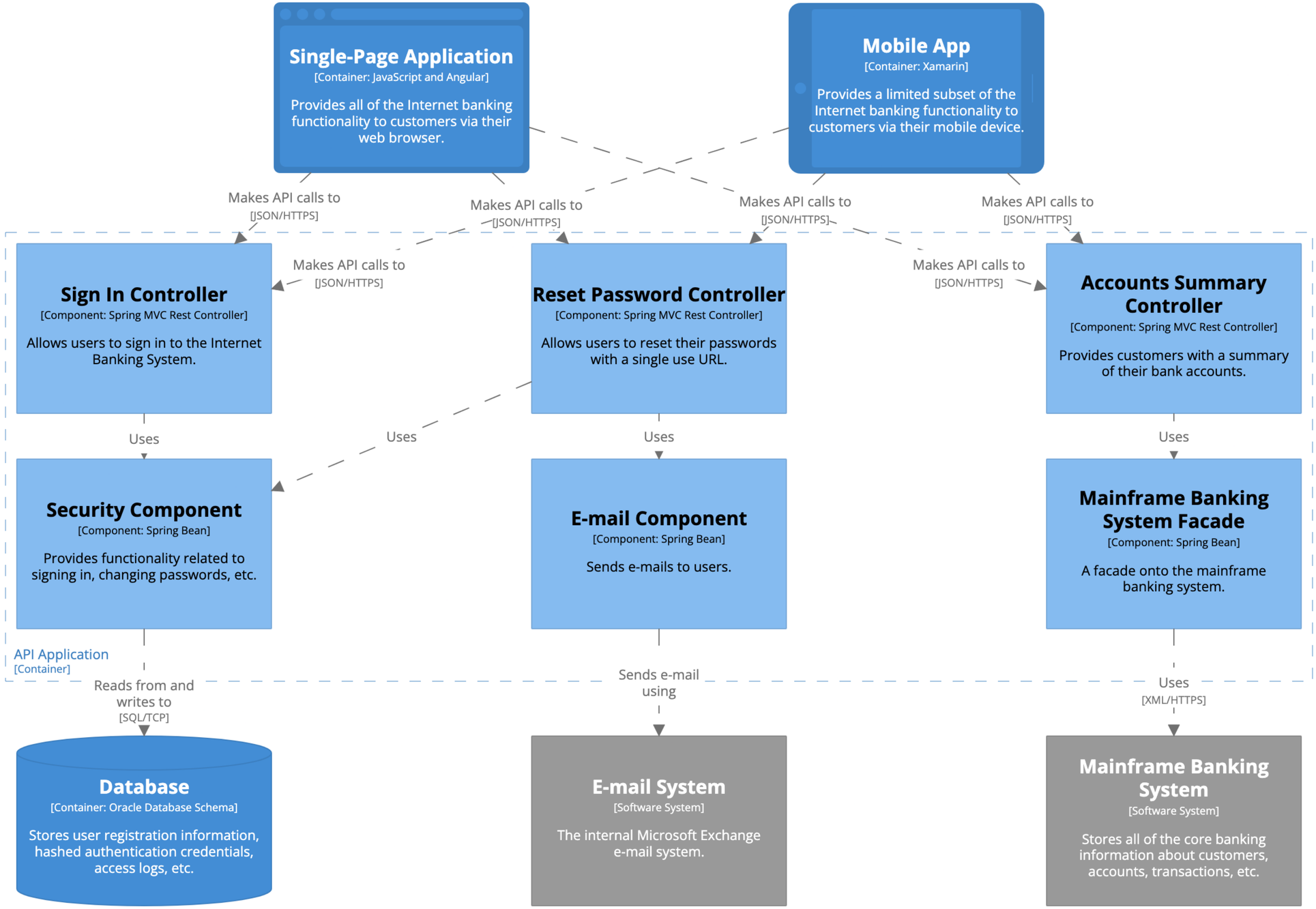
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



[Component] Internet Banking System - API Application

The component diagram for the API Application.

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



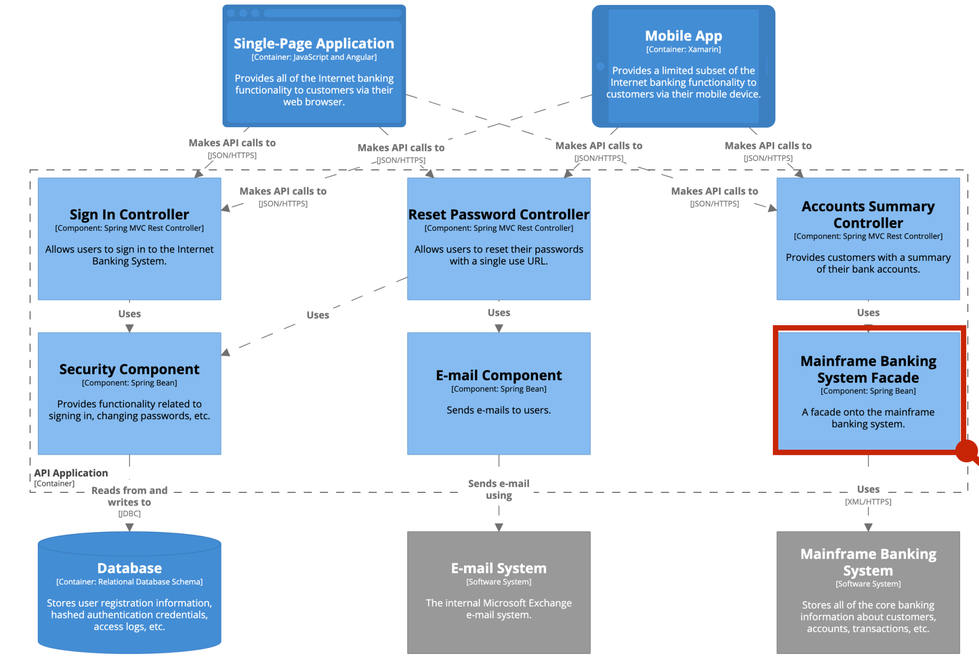
[Component] Internet Banking System - API Application

The component diagram for the API Application.

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

Level 4

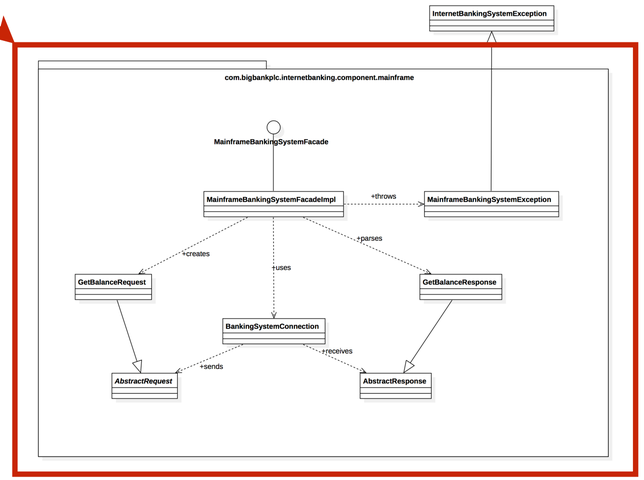
Code diagram

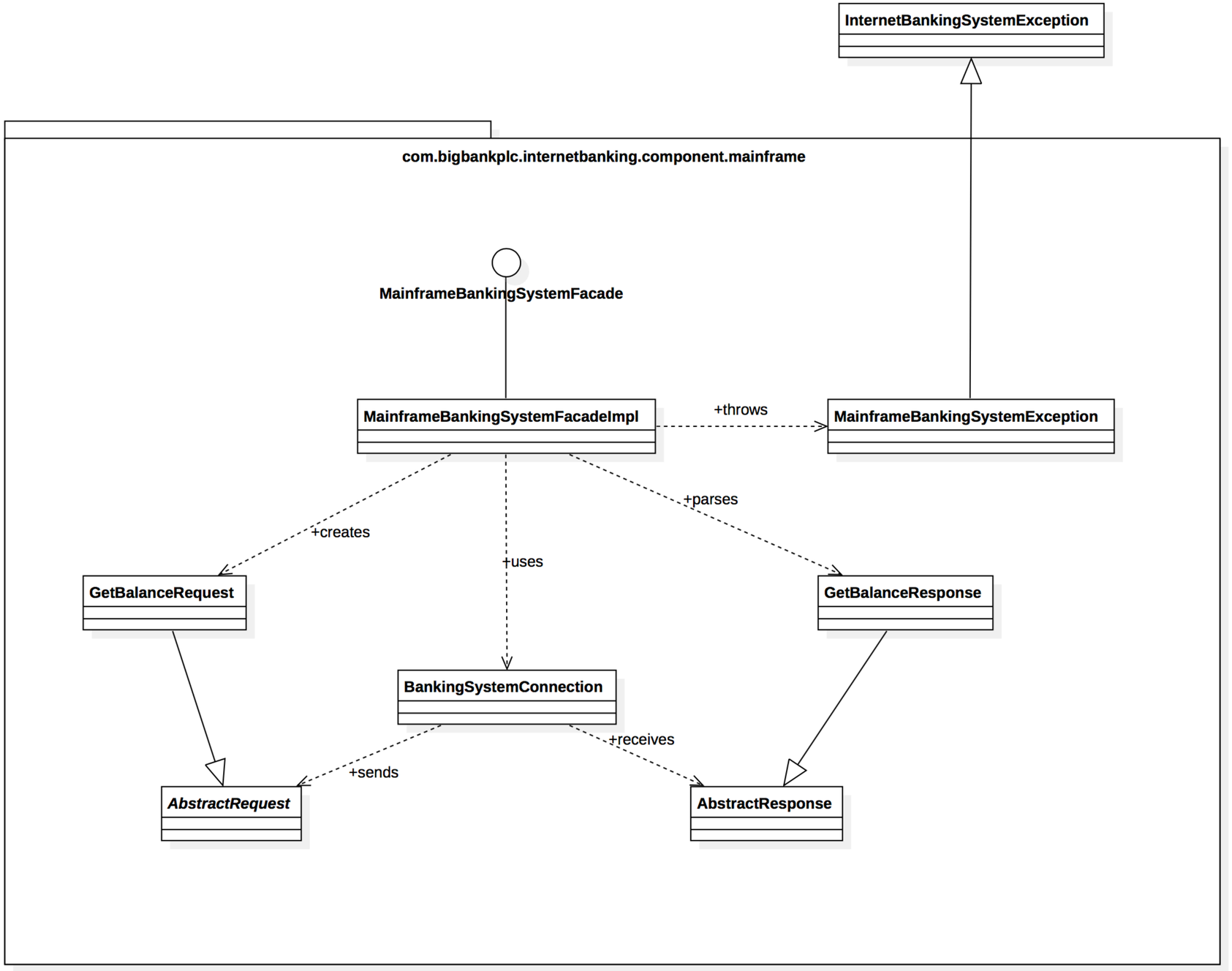


Component diagram for Internet Banking System - API Application

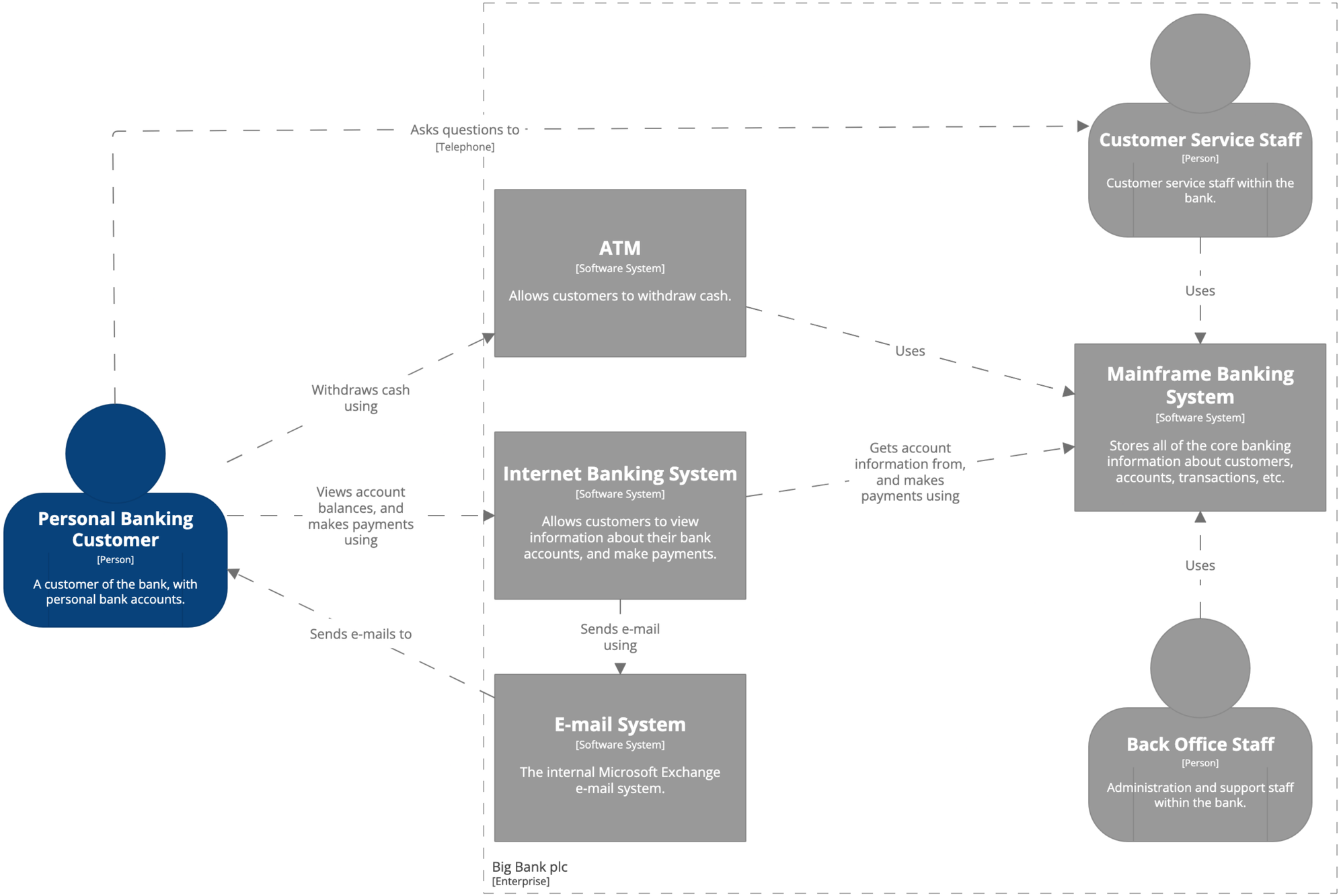
The component diagram for the API Application.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)

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
+receives
+sends
AbstractRequest
AbstractResponse

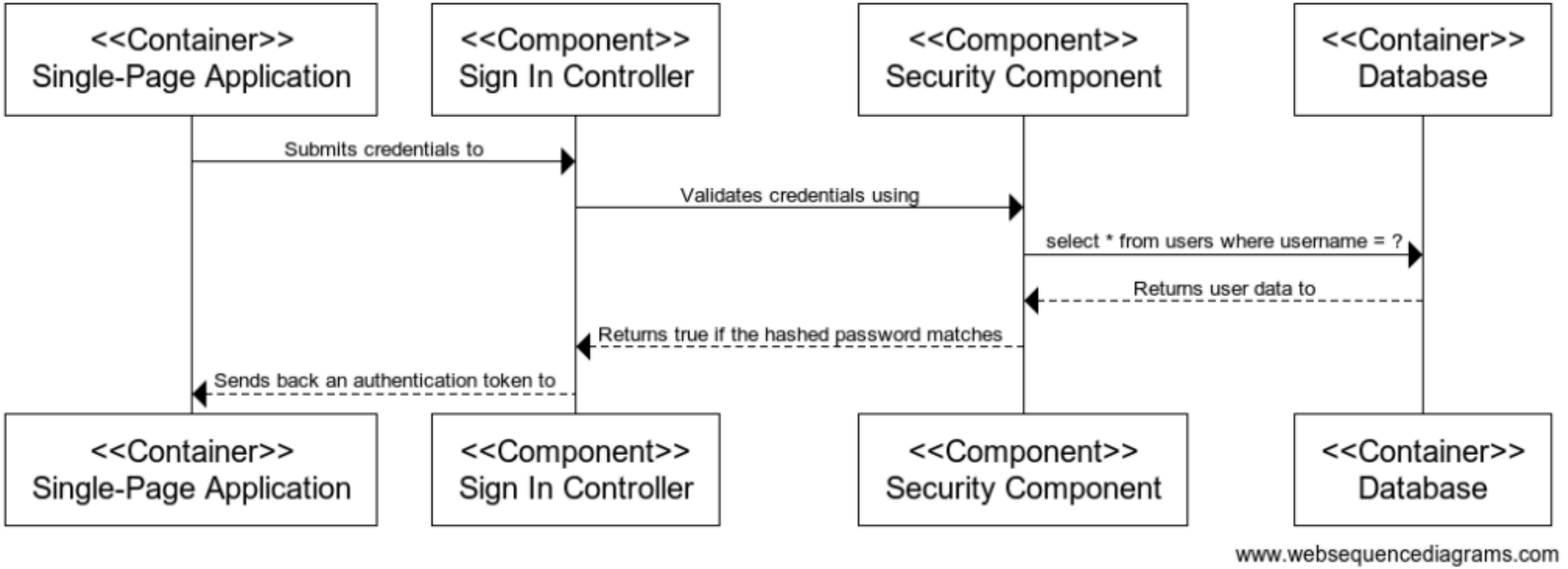
Other diagram types…

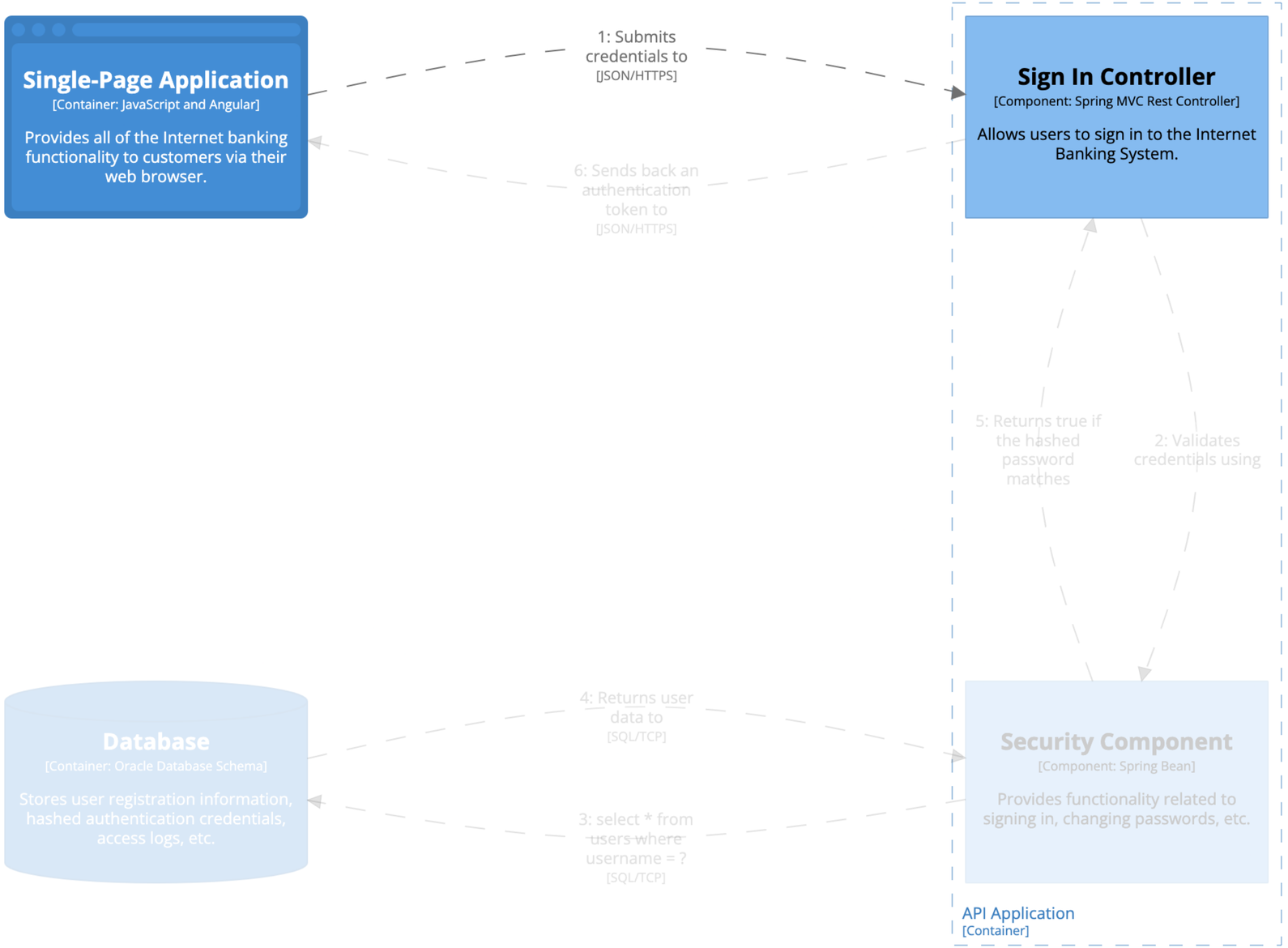


[System Landscape] Big Bank plc

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

API Application - Dynamic - SignIn

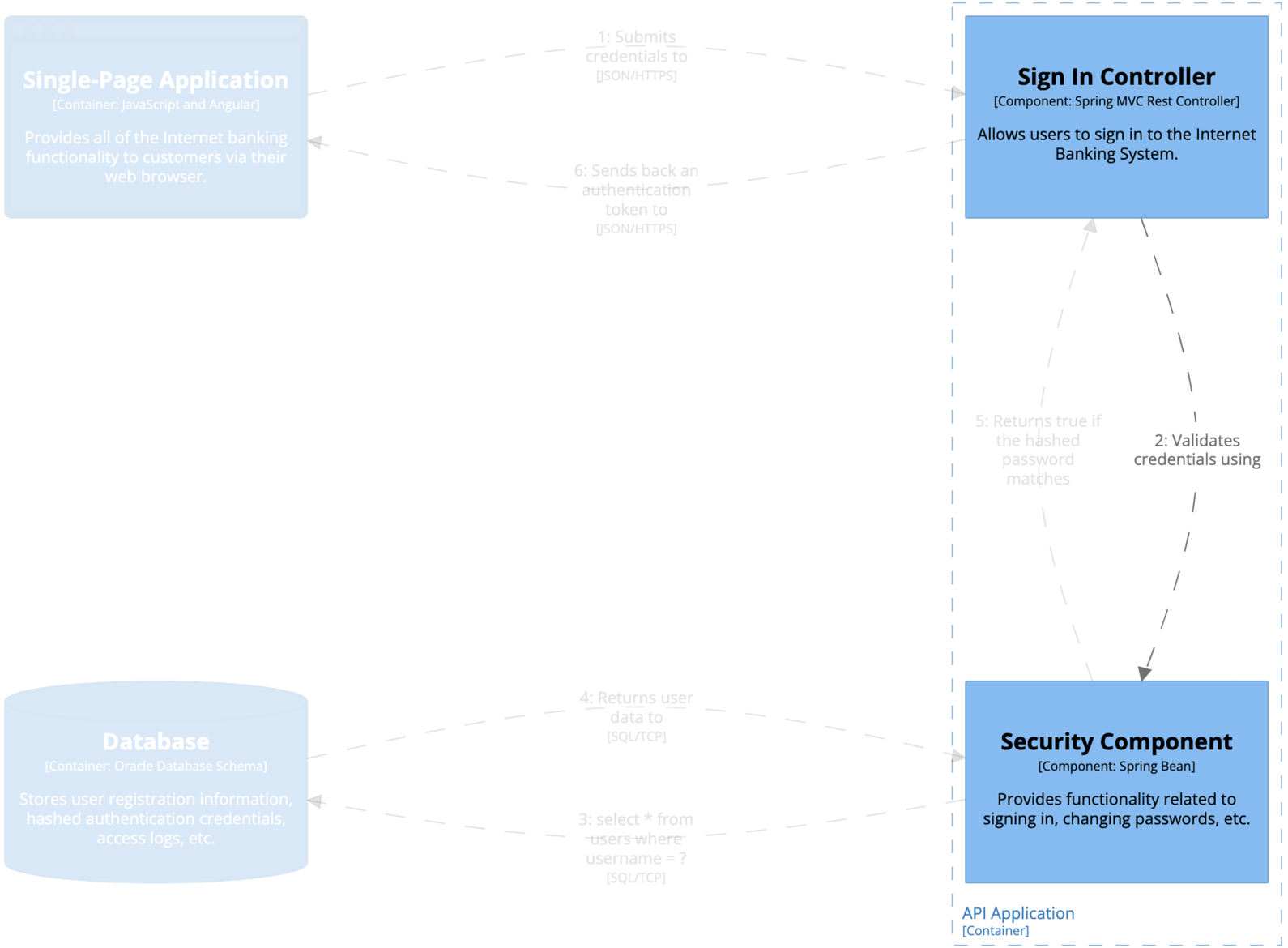




[Dynamic] Internet Banking System - API Application

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

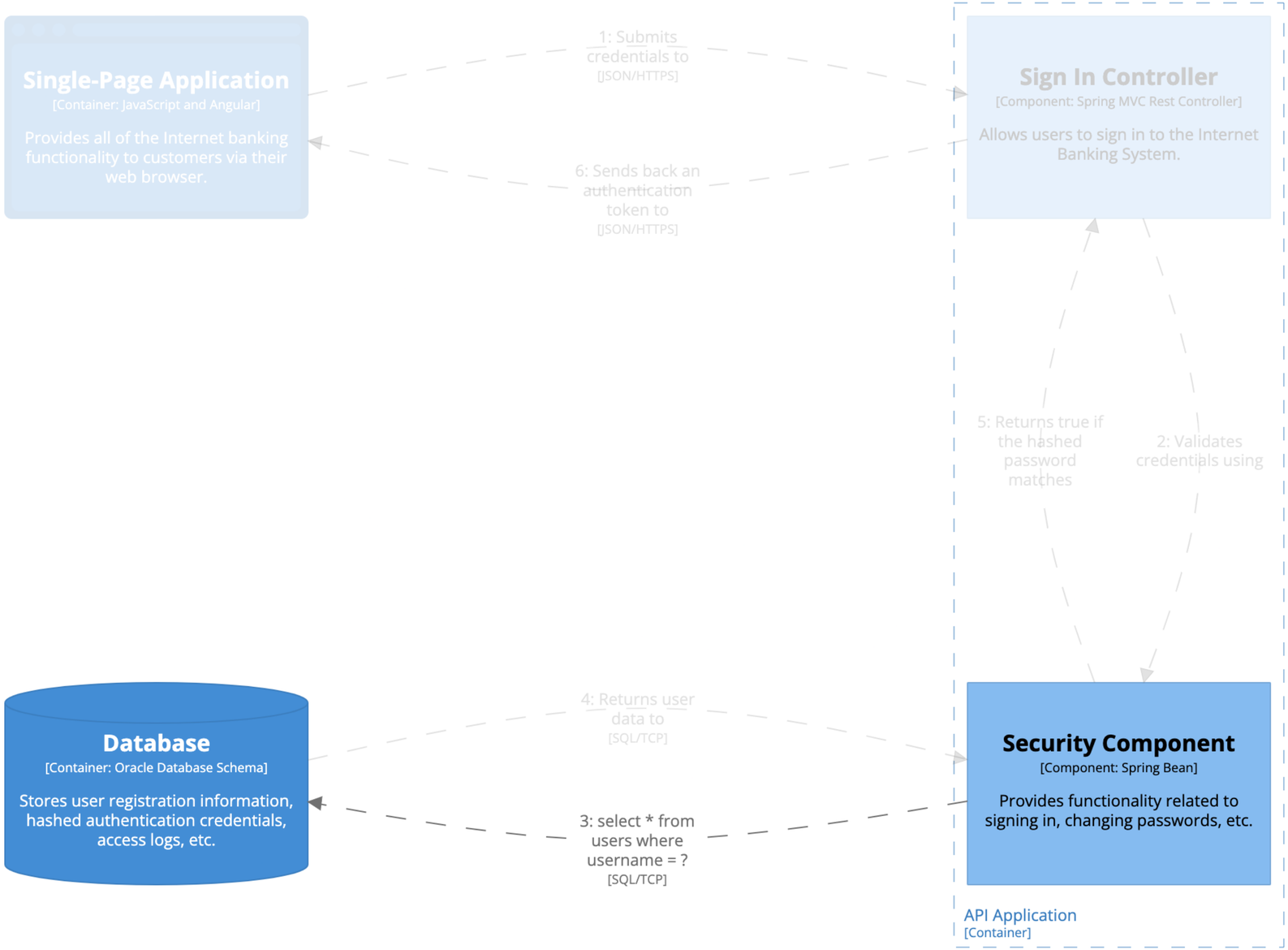
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



[Dynamic] Internet Banking System - API Application

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

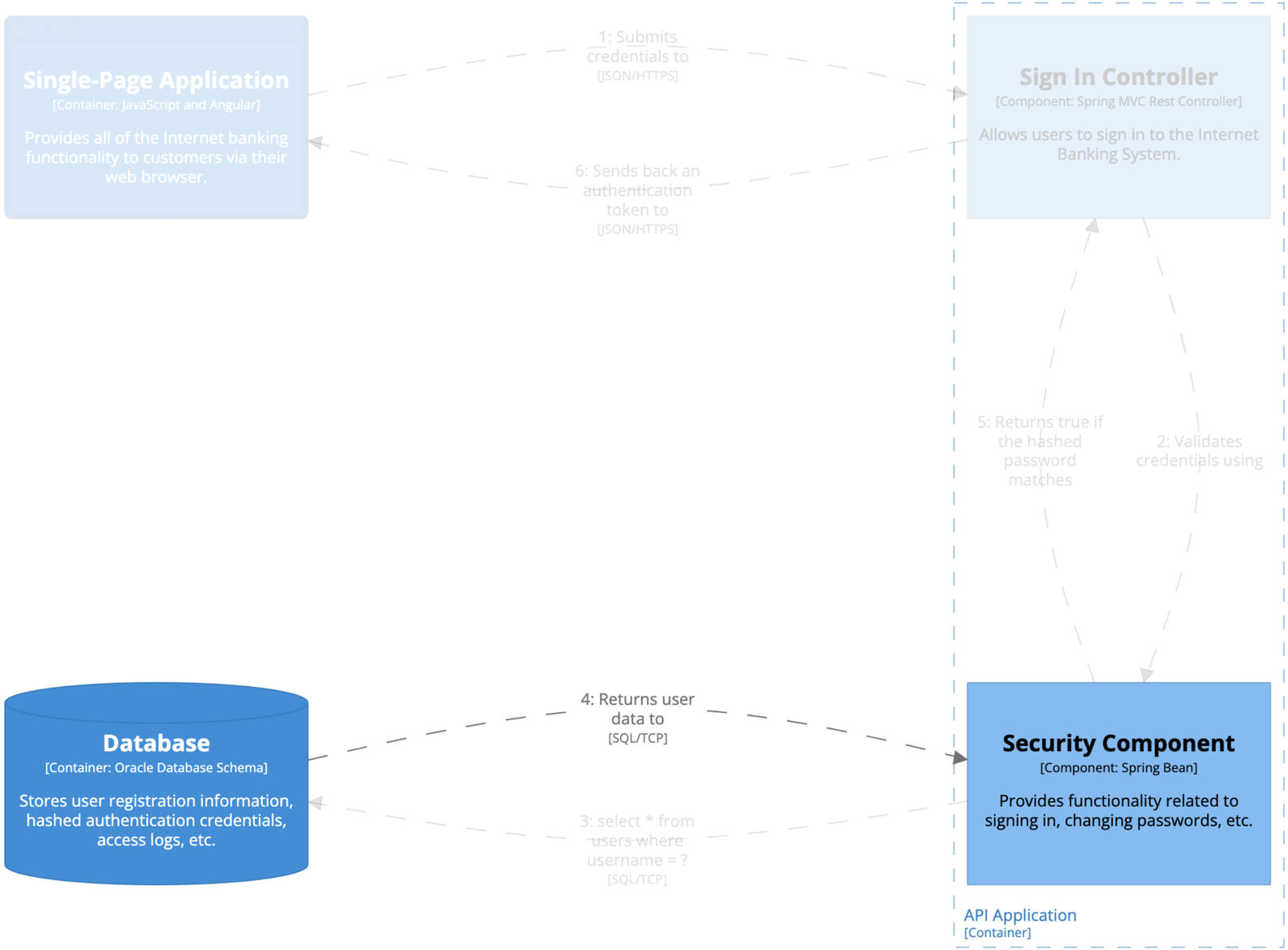
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



[Dynamic] Internet Banking System - API Application

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

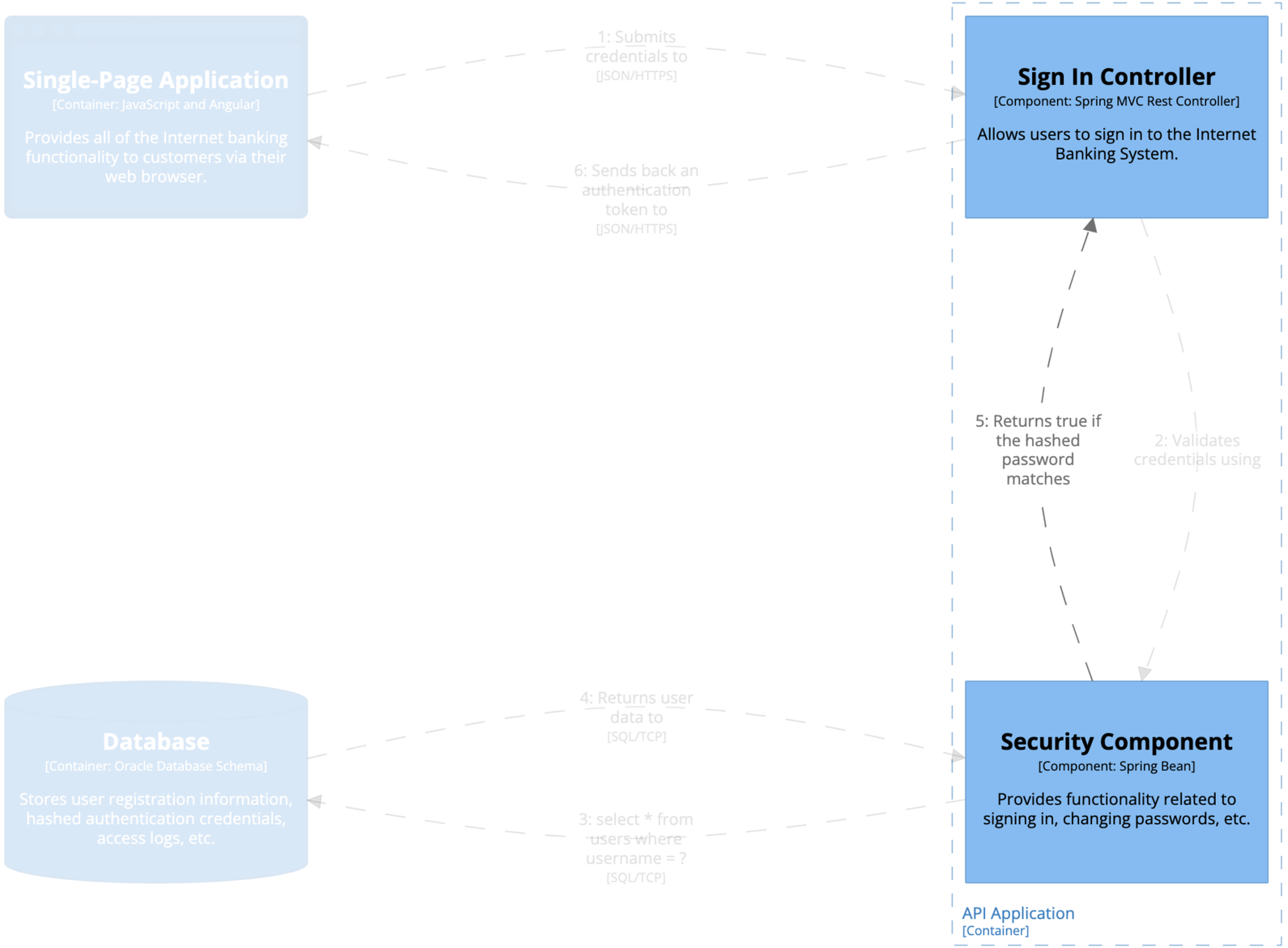
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



[Dynamic] Internet Banking System - API Application

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

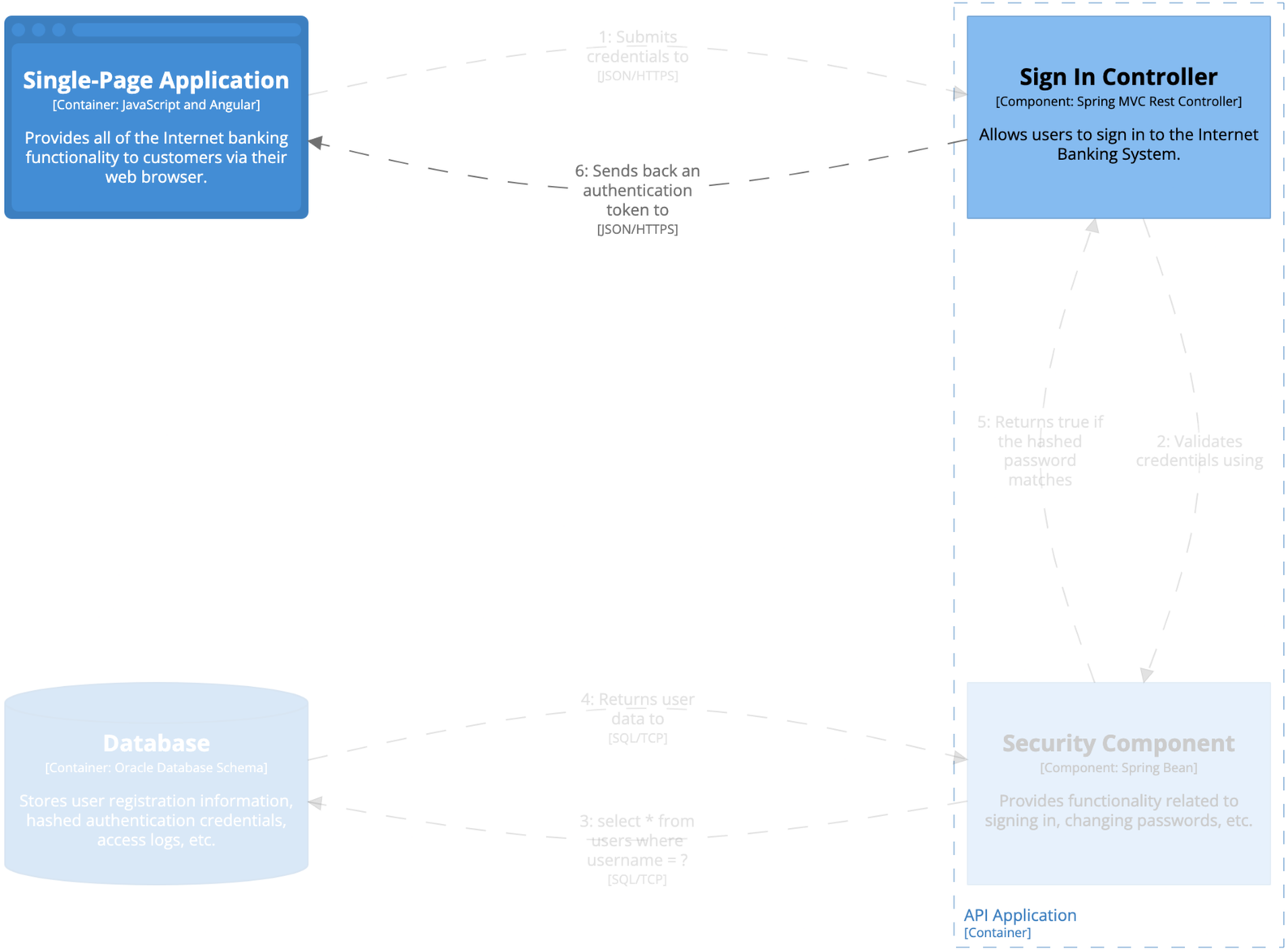
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



[Dynamic] Internet Banking System - API Application

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

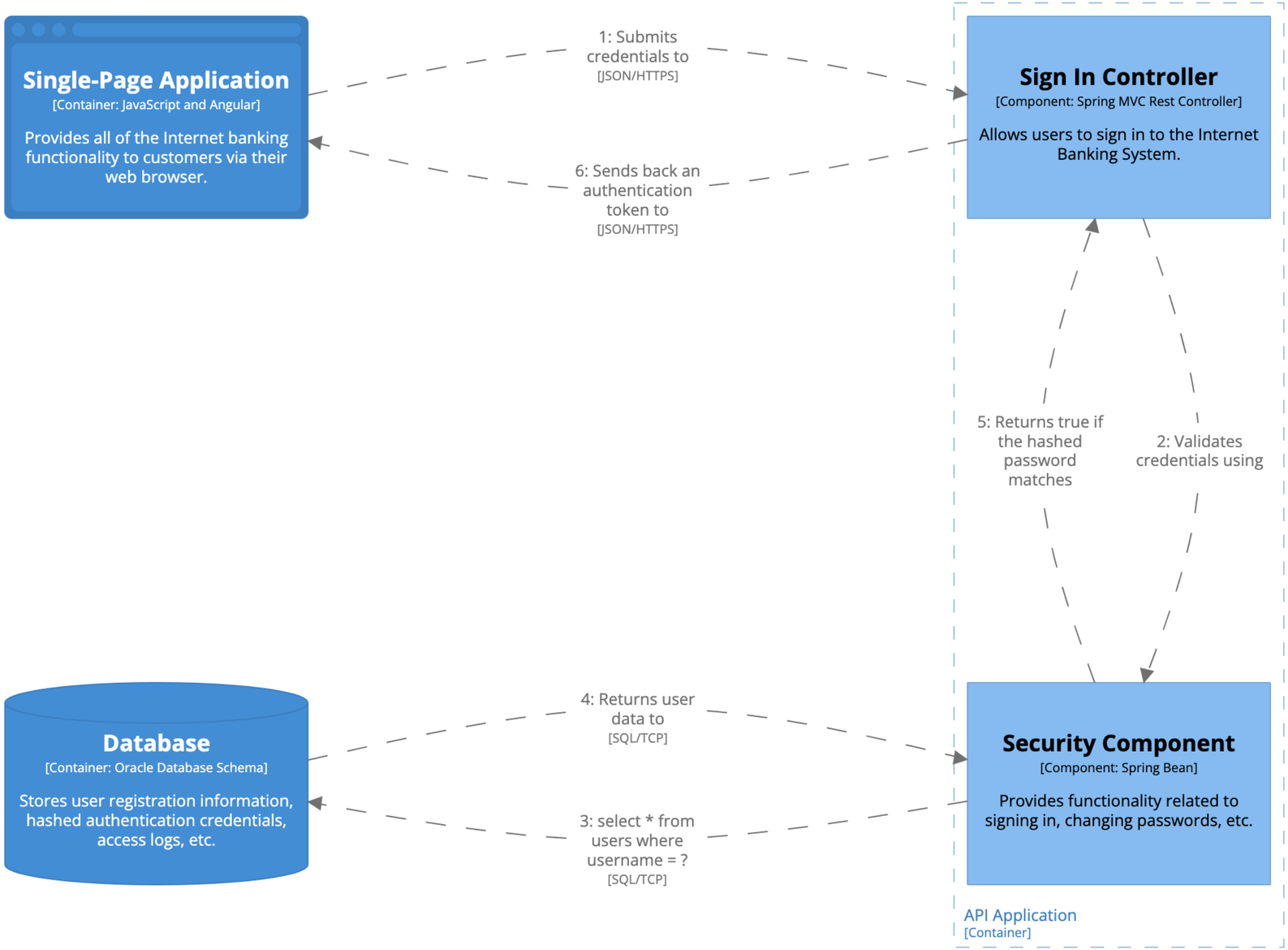
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



[Dynamic] Internet Banking System - API Application

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

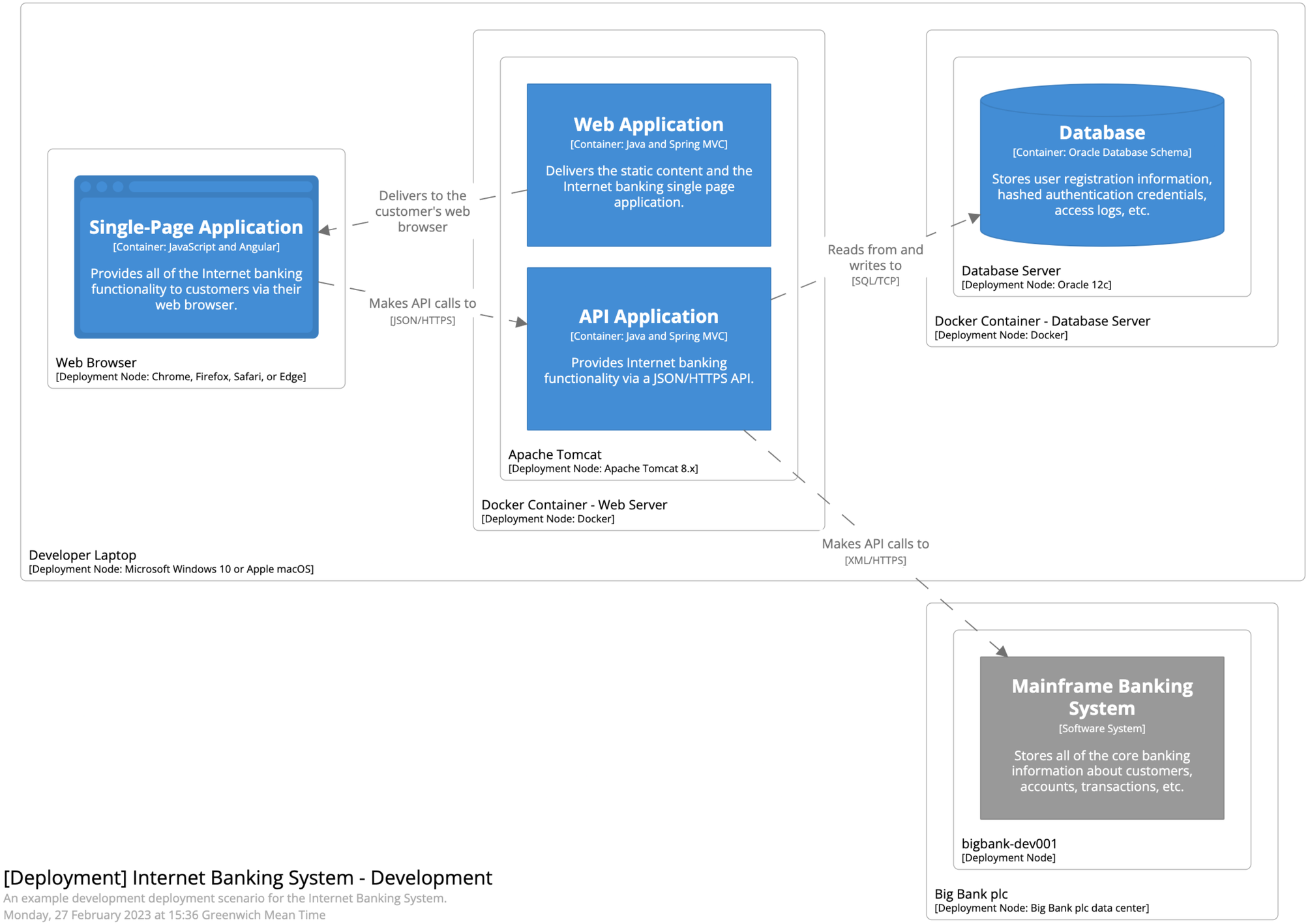
Monday, 27 February 2023 at 15:36 Greenwich Mean Time



[Dynamic] Internet Banking System - API Application

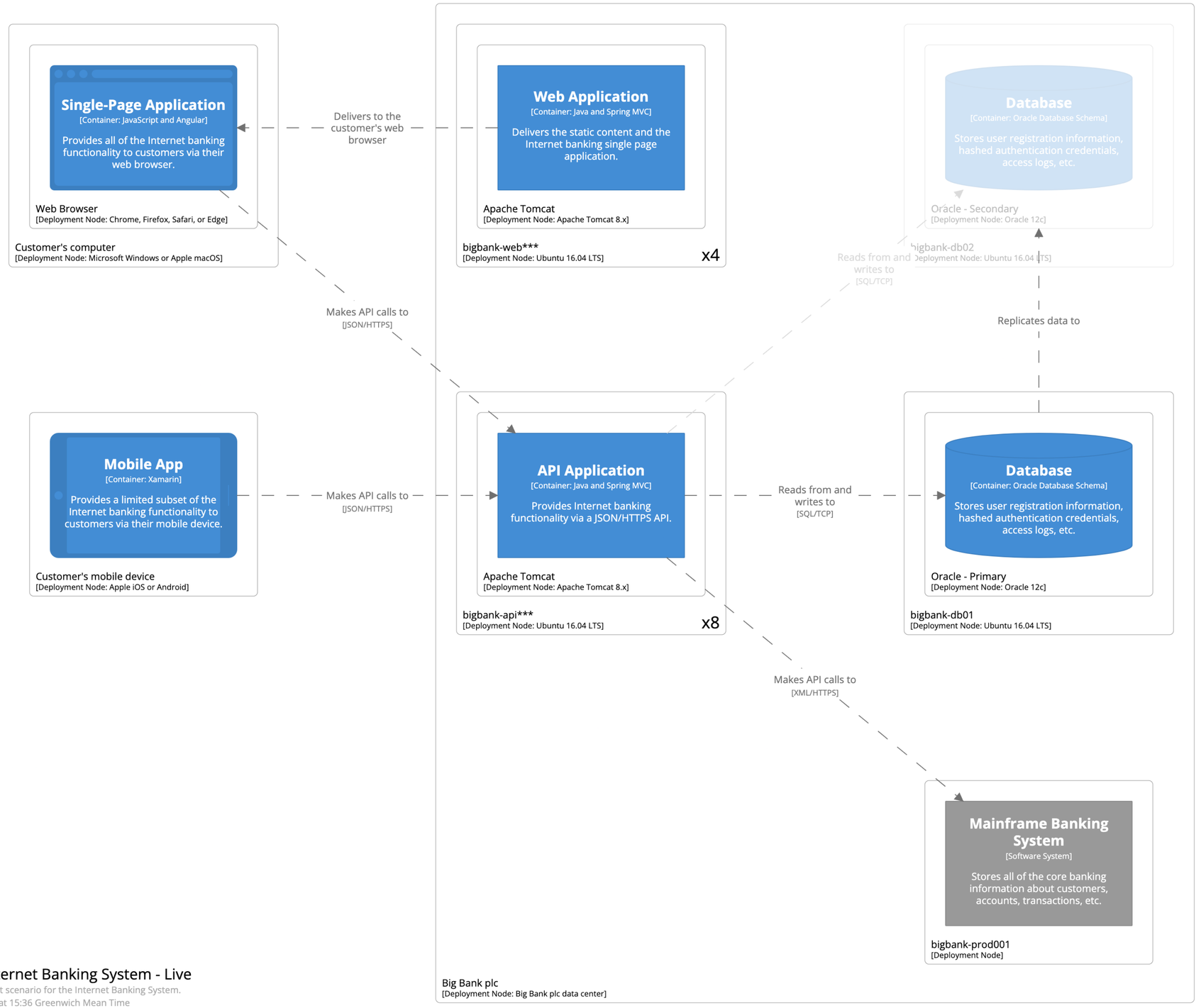
Summarises how the sign in feature works in the single-page application.

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



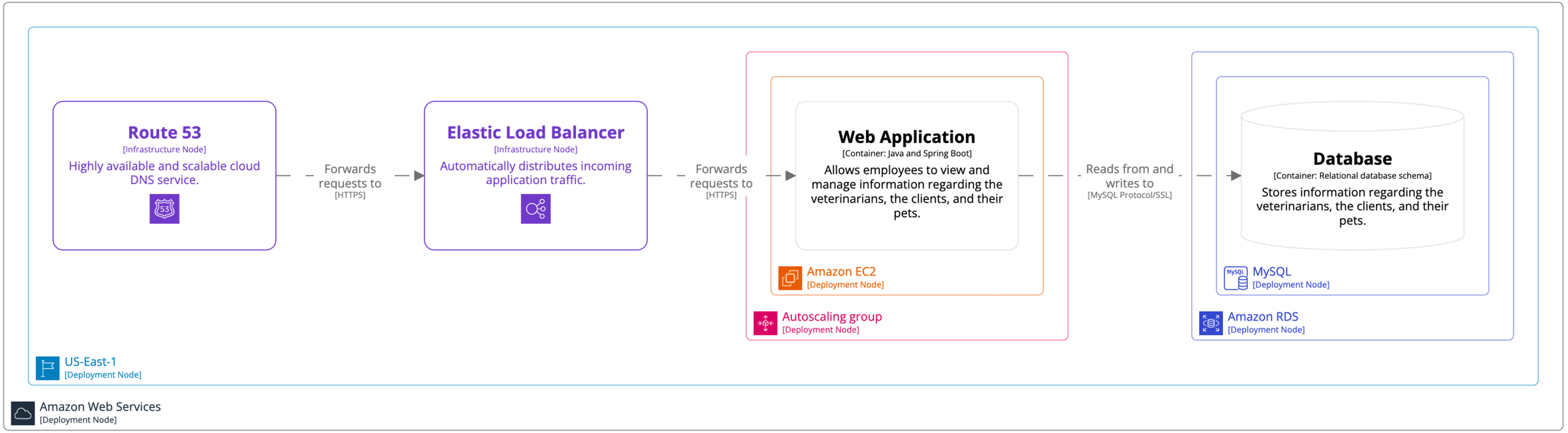
[Deployment] Internet Banking System - Development

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



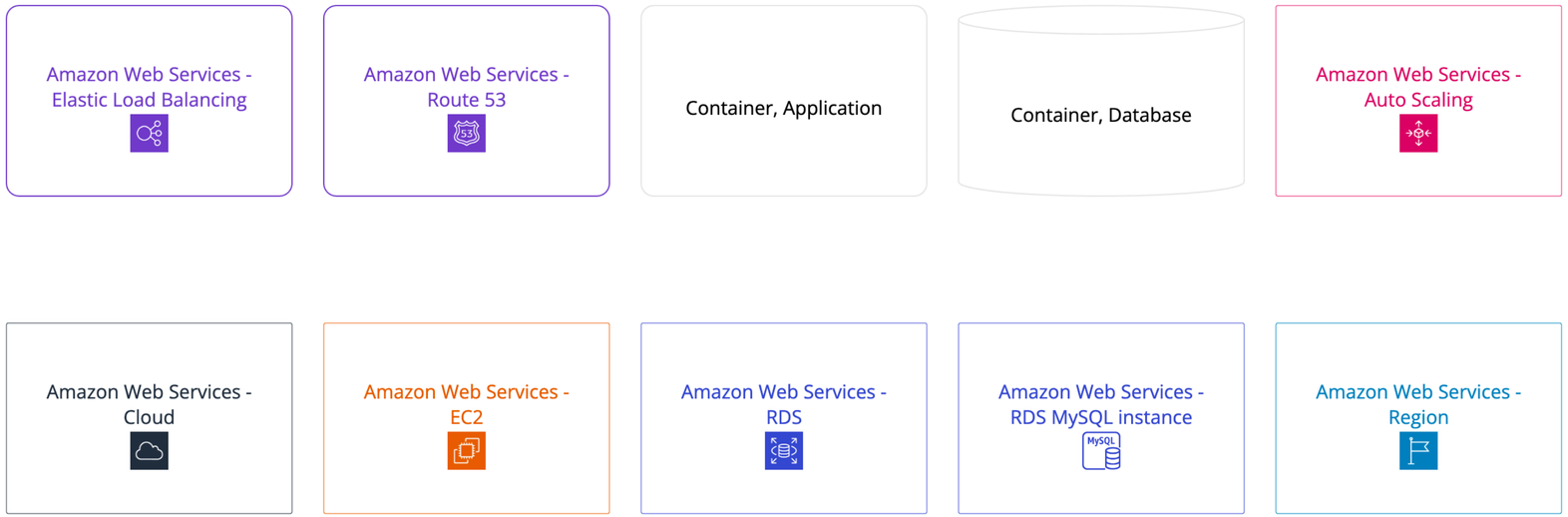
[Deployment] Internet Banking System - Live

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



[Deployment] Spring PetClinic - Live

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



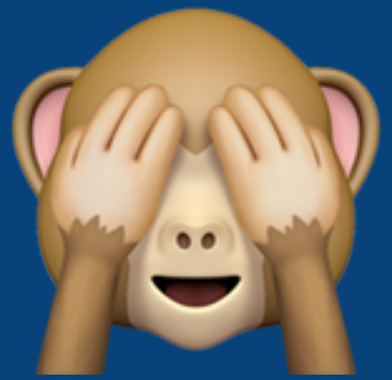
The lost art of software modelling?

Most teams use general purpose diagramming tools

(Visio, diagrams.net, Lucidchart, Gliffy, etc)

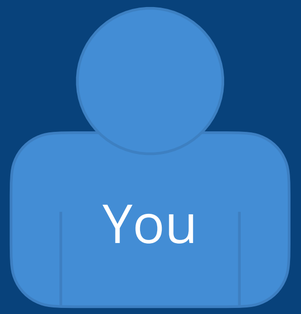
How can we avoid copy-pasting elements across diagrams?

Stop using Visio!



Structurizr DSL

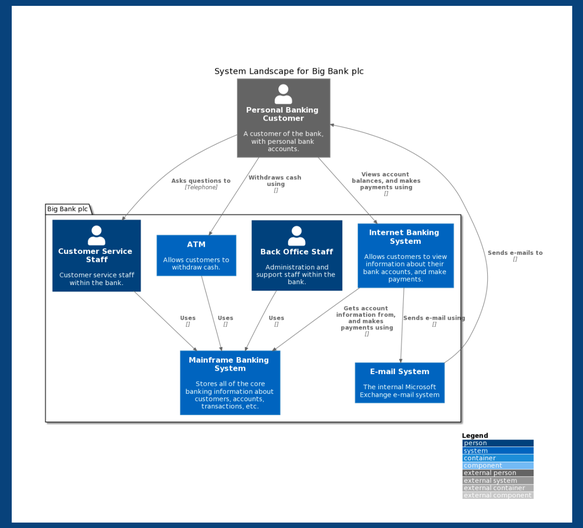
An open source, text-based domain specific language (DSL),
to create software architecture diagrams
based upon the C4 model

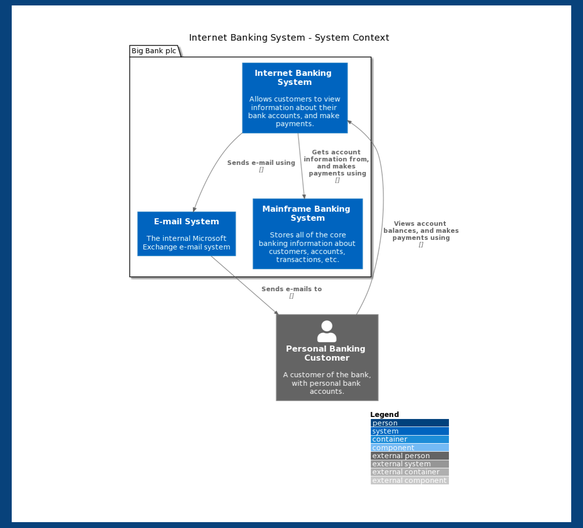


Create and maintain

```
workspace "Big Bank plc" "This is an example workspace to illustrate the key features of Structurizr, via the DSL, based around a fictional online banking system." {

    model {
        customer = person "Personal Banking Customer" "A customer of the bank, with personal bank accounts."

        enterprise "Big Bank plc" {
            supportStaff = person "Customer Service Staff" "Customer service staff within the bank." "Bank Staff"
            backoffice = person "Back Office Staff" "Administration and support staff within the bank." "Bank Staff"

            mainframe = softwaresystem "Mainframe Banking System" "Stores all of the core banking information about customers, accounts, transactions, etc." "Existing System"
            email = softwaresystem "E-mail System" "The internal Microsoft Exchange e-mail system" "Existing System"
            atm = softwaresystem "ATM" "Allows customers to withdraw cash." "Existing System"

            internetBankingSystem = softwaresystem "Internet Banking System" "Allows customers to view information about their bank accounts, and make payments." {
                singlePageApplication = container "Single-Page Application" "Provides all of the Internet banking functionality to customers via their web browser." "JavaScript and Angular" "Web Browser"
                mobileApp = container "Mobile App" "Provides a limited subset of the Internet banking functionality to customers via their mobile device." "Xamarin" "Mobile App"
                webApplication = container "Web Application" "Delivers the static content and the Internet banking single page application." "Java and Spring MVC"
                apiApplication = container "API Application" "Provides Internet banking functionality via a JSON/HTTPS API." "Java and Spring MVC" {
                    signinController = component "Sign In Controller" "Allows users to sign in to the Internet Banking System." "Spring MVC Rest Controller"
                    accountsSummaryController = component "Accounts Summary Controller" "Provides customers with a summary of their bank accounts." "Spring MVC Rest Controller"
                    resetPasswordController = component "Reset Password Controller" "Allows users to reset their passwords with a single use URL." "Spring MVC Rest Controller"
                    securityComponent = component "Security Component" "Provides functionality related to signing in, changing passwords, etc." "Spring Bean"
                    mainframeBankingSystemFacade = component "Mainframe Banking System Facade" "A facade onto the mainframe banking system." "Spring Bean"
                    emailComponent = component "E-mail Component" "Sends e-mails to users." "Spring Bean"
                }
                database = container "Database" "Stores user registration information, hashed authentication credentials, access logs, etc." "Oracle Database Schema" "Database"
            }

        }

        # relationships between people and software systems
        user -> customer -> internetBankingSystem "Views account balances, and makes payments using"
        internetBankingSystem -> mainframe "Gets account information from, and makes payments using"
        internetBankingSystem -> email "Sends e-mails using"
        email -> customer "Sends e-mails to"
        customer -> supportStaff "Asks questions to" "Telephone"
        supportStaff -> mainframe "Uses"
        customer -> atm "Withdraws cash using"
        atm -> mainframe "Uses"
        backoffice -> mainframe "Uses"

        # relationships to/from containers
        customer -> webApplication "Visits bigbank.com/ib using" "HTTPS"
        customer -> singlePageApplication "Views account balances, and makes payments using"
        customer -> mobileApp "Views account balances, and makes payments using"
        webApplication -> singlePageApplication "Delivers to the customer's web browser"

        # relationships to/from components
        singlePageApplication -> signinController "Makes API calls to" "JSON/HTTPS"
        singlePageApplication -> accountsSummaryController "Makes API calls to" "JSON/HTTPS"
        singlePageApplication -> resetPasswordController "Makes API calls to" "JSON/HTTPS"
        mobileApp -> signinController "Makes API calls to" "JSON/HTTPS"
        mobileApp -> accountsSummaryController "Makes API calls to" "JSON/HTTPS"
        mobileApp -> resetPasswordController "Makes API calls to" "JSON/HTTPS"
        signinController -> securityComponent "Uses"
        accountsSummaryController -> mainframeBankingSystemFacade "Uses"
        resetPasswordController -> securityComponent "Uses"
        resetPasswordController -> emailComponent "Uses"
        securityComponent -> database "Reads from and writes to" "JDBC"
        mainframeBankingSystemFacade -> mainframe "Makes API calls to" "XML/HTTPS"
        emailComponent -> email "Sends e-mail using"
    }

}
```

Automatically generates

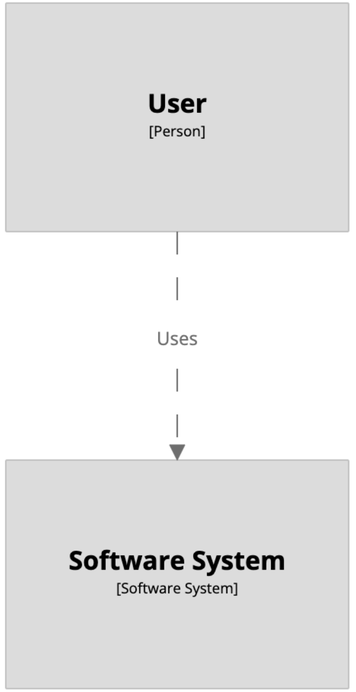




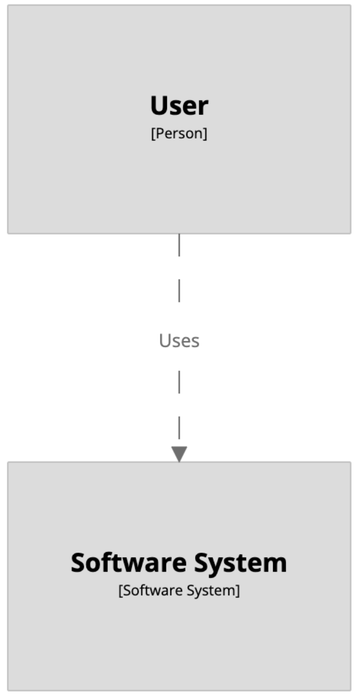
Diagrams as code 2.0

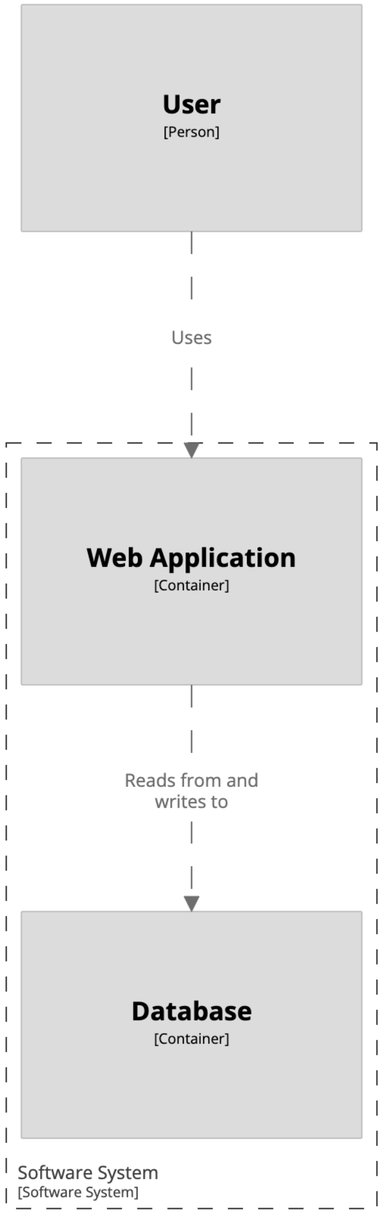
You create and maintain a single model, and the tool generates multiple diagrams, automatically keeping them all in sync whenever you change the model

```
workspace {

    model {
        user = person "User"
        softwareSystem = softwareSystem "Software System"



        user -> softwareSystem "Uses"

    }

    views {
        systemContext softwareSystem {
            include *
            autoLayout
        }



    }

}
```



```
workspace {

    model {
        user = person "User"
        softwareSystem = softwareSystem "Software System" {
            webapp = container "Web Application"
            database = container "Database"
        }

        user -> webapp "Uses"
        webapp -> database "Reads from and writes to"
    }

    views {
        systemContext softwareSystem {
            include *
            autoLayout
        }

        container softwareSystem {
            include *
            autolayout
        }
    }

}
```





Structurizr - C4 models as code

Overview | Repositories 25 | Projects | Packages | People

Find a repository... | Type | Language | Sort

java

Public

Structurizr for Java

software-architecture c4model architecture-diagrams structurizr

Java Apache-2.0 264 898 0 (4 issues need help) 2 Updated 27 minutes ago

dsl

Public

Structurizr DSL

dsl software-architecture structurizr c4model architecture-diagrams

Java Apache-2.0 256 1,100 0 2 Updated 33 minutes ago

cli

Public

A command line utility for Structurizr.

markdown plantuml asciidoc software-architecture architecture-doc architecture-decision-records structurizr

Java Apache-2.0 62 410 0 0 Updated yesterday

lite Public

Structurizr Lite

structurizr c4model softwarearchitecture c4-model

Java MIT 4 73 0 0 Updated yesterday

themes Public

14 13 0 1 Updated yesterday

ui Public

JavaScript MIT 5 16 5 0 Updated 2 days ago

onpremises Public

Structurizr on-premises installation

structurizr c4model softwarearchitecture c4-model

Java MIT 11 19 5 0 Updated 2 days ago

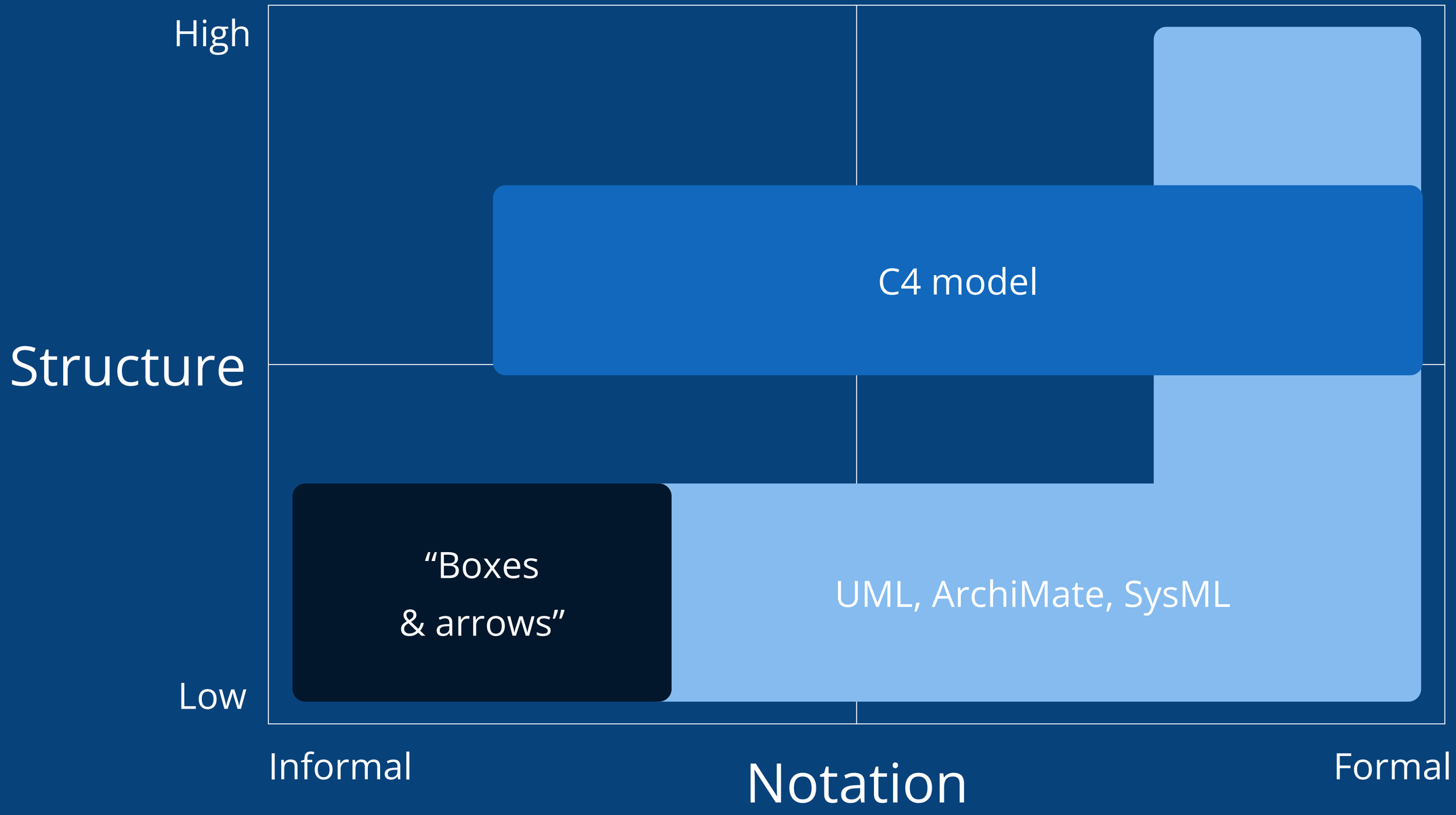
export Public

Export models and views to external formats.

Java Apache-2.0 21 12 1 2 Updated last week

Do you have any empirical evidence for any of this?

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


High
Structure
Low
C4 model
“Boxes
& arrows”
UML, ArchiMate, SysML
Informal
Notation
Formal

Summary

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

Thank you!

Simon Brown
@simonbrown