

C4 Model

<https://c4model.com/>

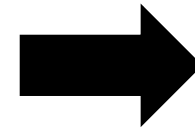
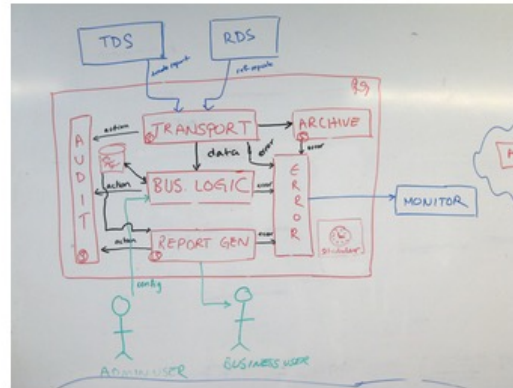
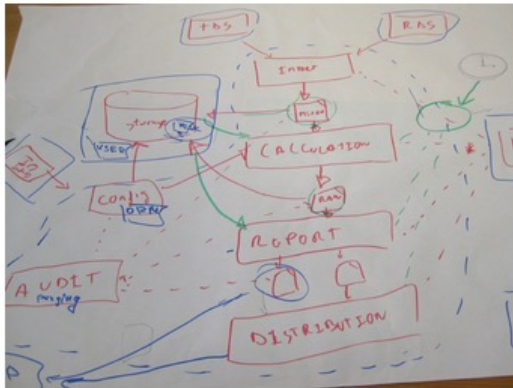
Sommaire

1. Objectifs du C4 Model
2. Comment
3. Les 4 niveaux
 1. Software System
 2. Container
 3. Components
 4. Code

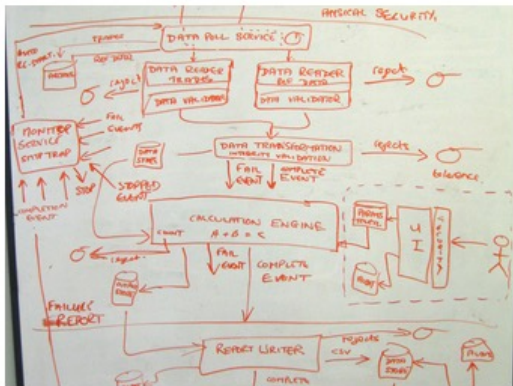
1. Objectifs C4 Model

Objectif

The C4 model was created as a way to help software development teams describe and communicate software architecture, both during up-front design sessions and when retrospectively documenting an existing codebase.



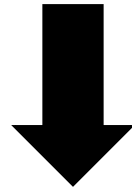
Eviter les schémas incompréhensifs ...



2. Comment

Différents niveaux de détails

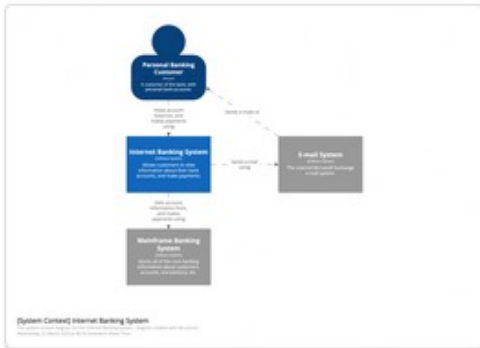
Représenter notre système avec différents niveau de détail



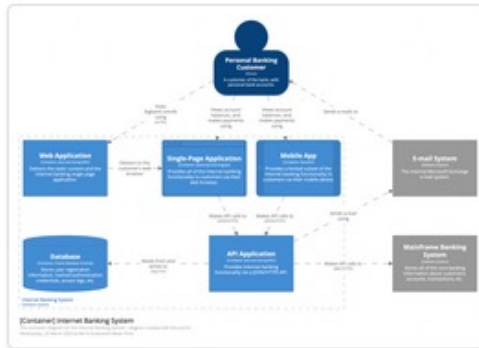
On s'adapte à l'interlocuteur

Métaphore

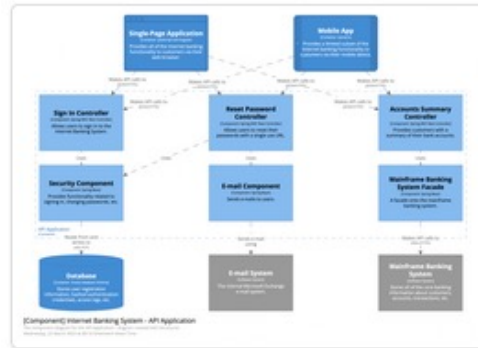
Le C4 Model permet de visualiser du code à différentes échelles (zoom / dézoom)



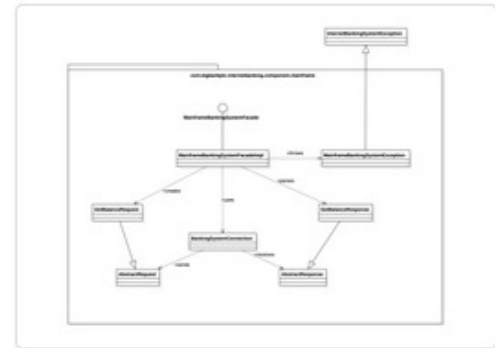
Level 1: A **System Context** diagram provides a starting point, showing how the software system in scope fits into the world around it.



Level 2: A **Container** diagram zooms into the software system in scope, showing the high-level technical building blocks.



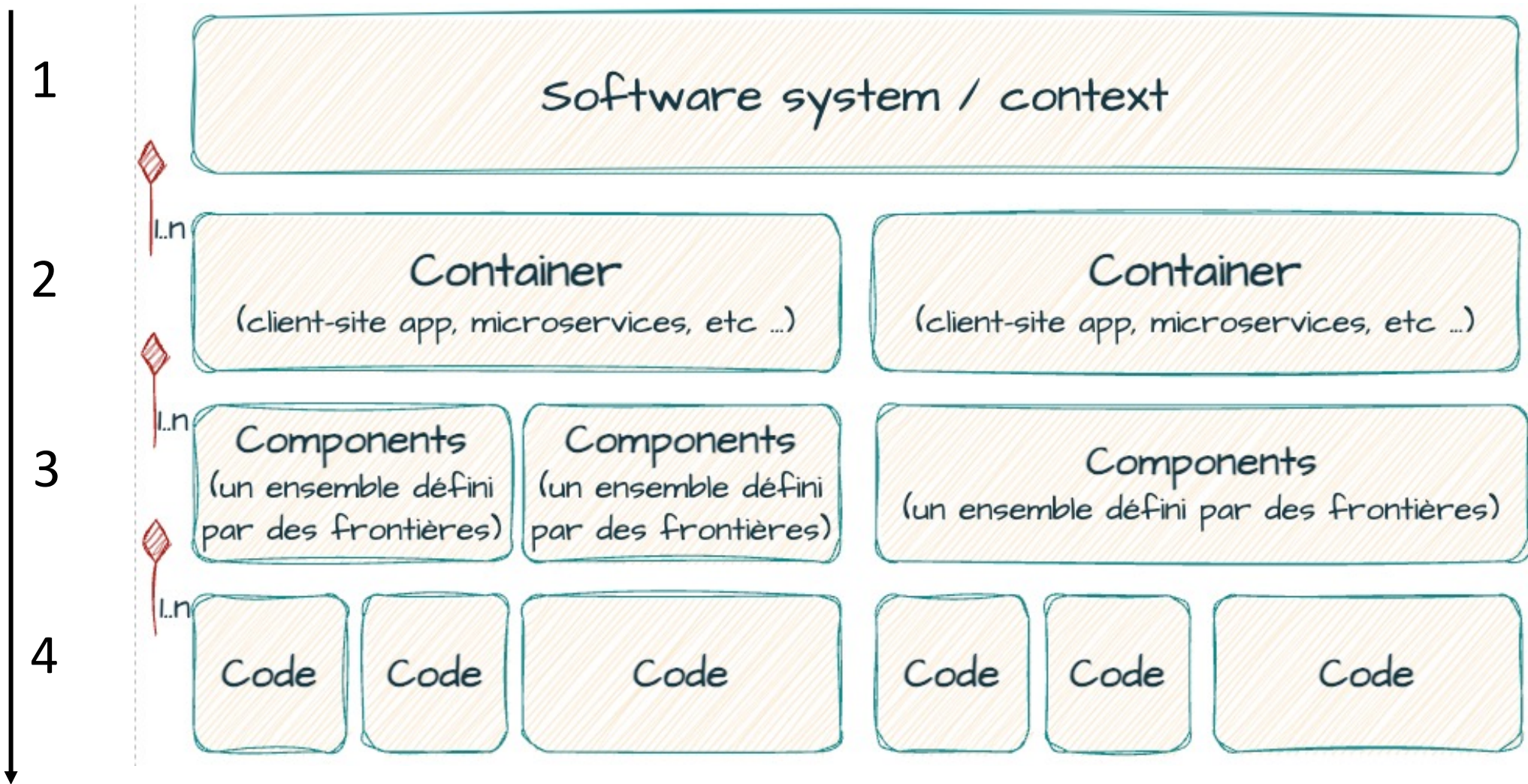
Level 3: A **Component** diagram zooms into an individual container, showing the components inside it.



Level 4: A **code** (e.g. UML class) diagram can be used to zoom into an individual component, showing how that component is implemented.

Différentes échelles pour différents usages pour s'adresser à différentes personnes

4 niveaux

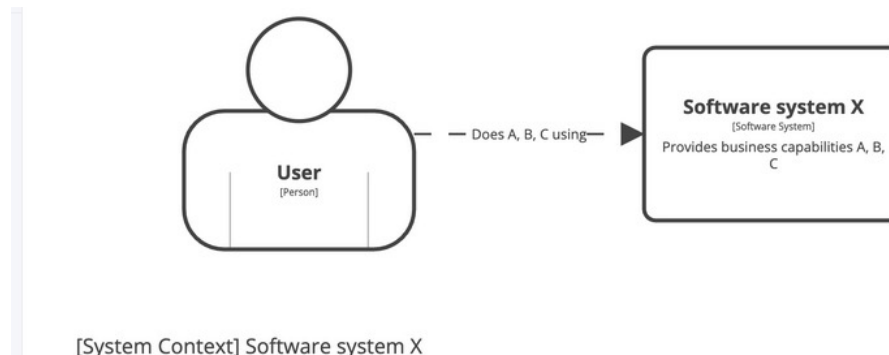


A **software system** is made up of one or more **containers** (applications and data stores), each of which contains one or more **components**, which in turn are implemented by one or more **code elements** (classes, interfaces, objects, functions, etc)

3. Les 4 Niveaux

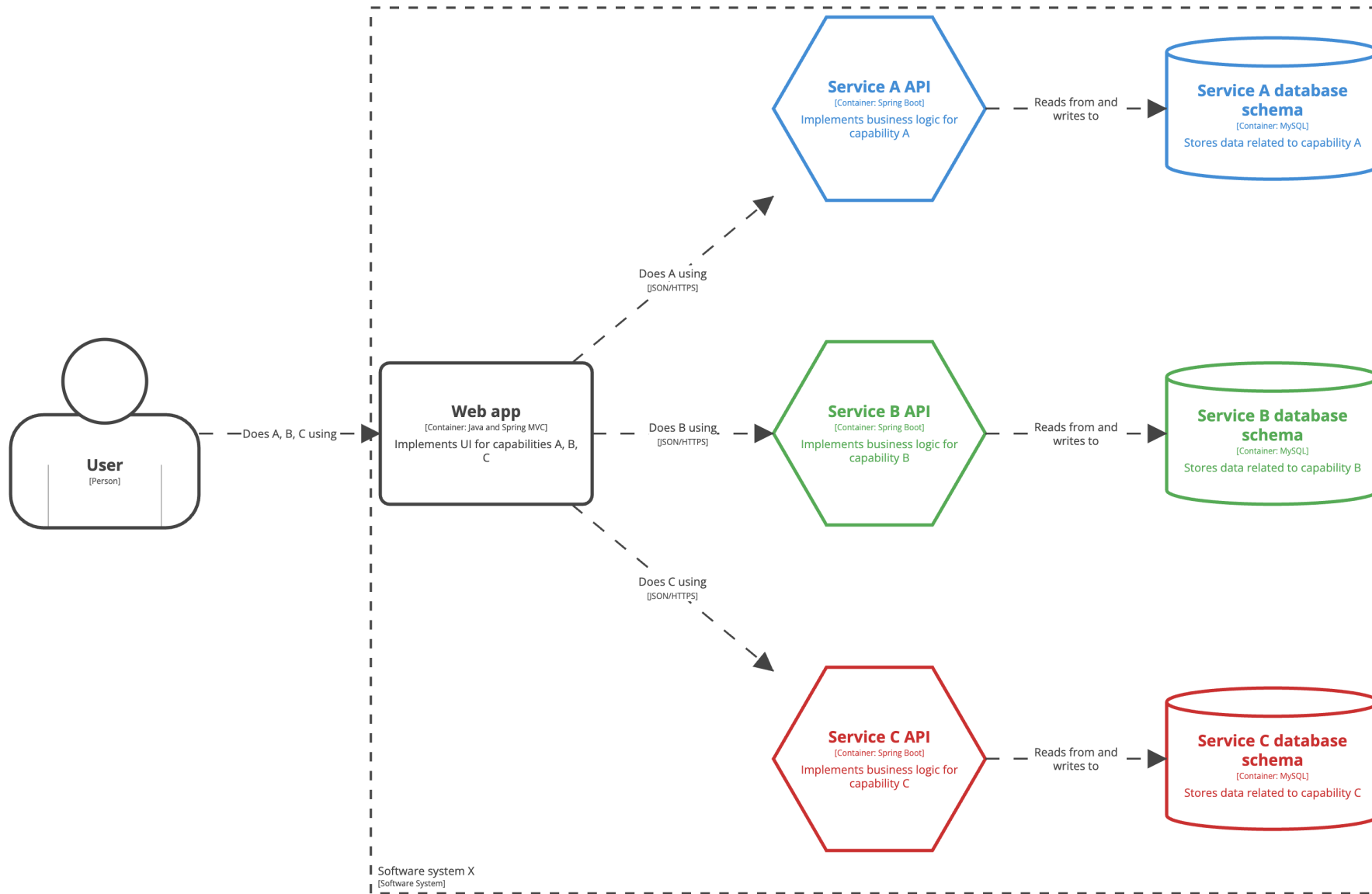
#1 Software System

- Décrit quelque chose qui apporte de la valeur au client
- On modélise
 - Notre software System
 - Et également ses interactions avec des autres Software System
 - E.g. dans le cas d'un SI on aura plusieurs logiciel qui vont communiquer ensemble pour pouvoir apporter la valeur



#2 Containers

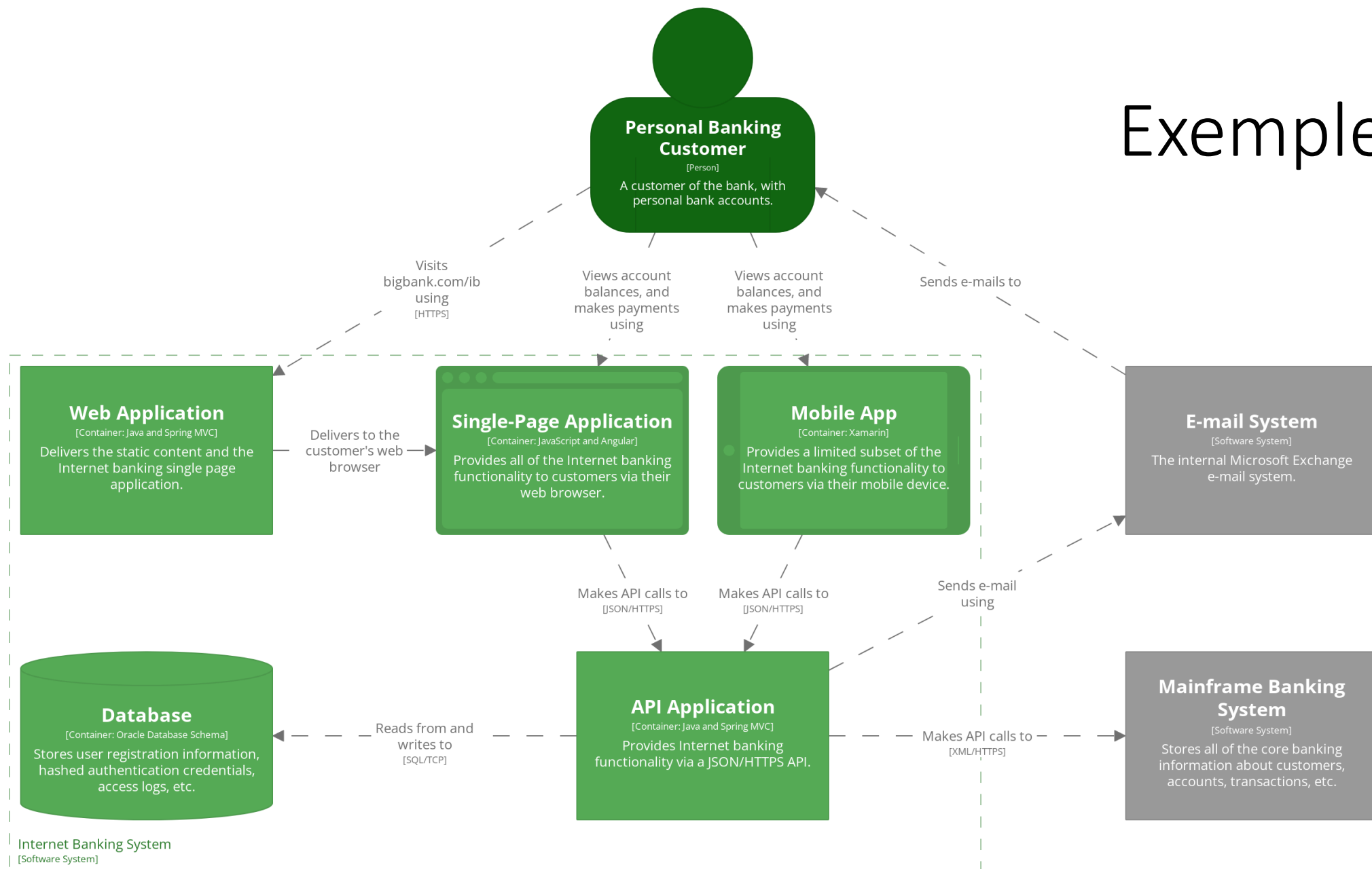
- Représente une application ou un endroit de stockage (e.g BDD)
- Chaque conteneur peut être déployable/exécuté de manière séparément (monolithique, microservices)
- Exemples :
 - Server-side Web App : Java EE sur serveur apache TomCat
 - Client-side Web App : JavaScript qui tourne sur un navigateur web
 - Client-side Desktop App : JavaFX
 - Database : SGBD, MongoDB



Exemple 1

- Unique monolithic UI
- Business logic dans différents ms

Example 2



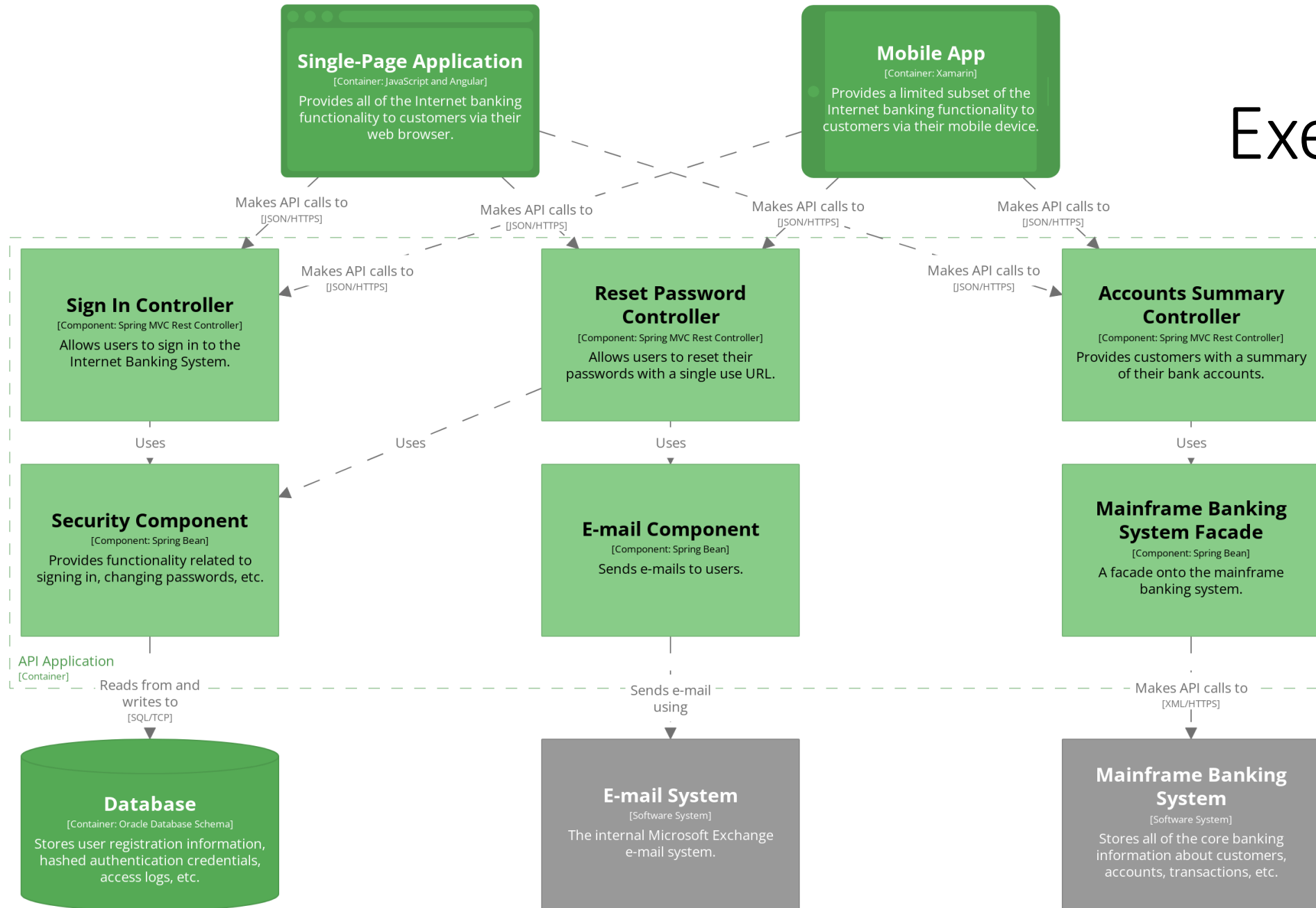
[Container] Internet Banking System

The container diagram for the Internet Banking System - diagram created with Structurizr.
Wednesday, March 22, 2023 at 8:16 AM Coordinated Universal Time

#3 Components

- Est un regroupement de fonctionnalités fortement liée (High coupled) cachée derrière une interface
- Dans cette couche on commence :
 - A parler techno : Spring Bean, Spring MVN, REST
 - Découpage en modules
 - Découpage en package / namespace
- Un component n'est pas une unité déployable séparément
 - Il me faut pls module (un sens l'autre pas de sens)
 - Ici regroupement fonctionnel (ce module va avec lui ... pour apporter de la valeur)

Example 1

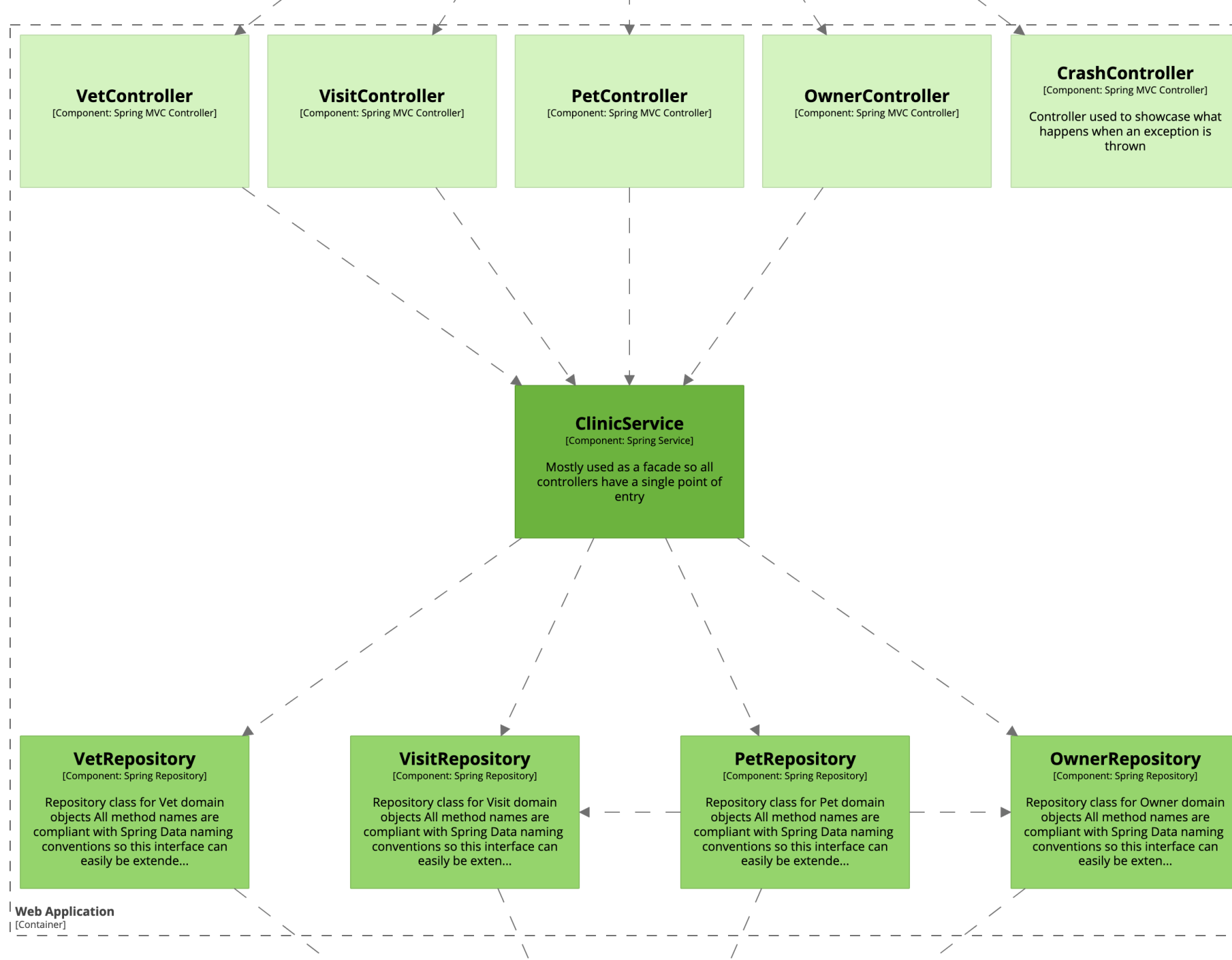


[Component] Internet Banking System - API Application

The component diagram for the API Application - diagram created with Structurizr.

Wednesday, March 22, 2023 at 8:16 AM Coordinated Universal Time

Example 2



#4 Code

- Pas recommandé de le faire (par les auteurs)
- Je dirais
 - Laisser cette partie aux développeurs
 - C'est à eux de faire en sorte que le code technique soit maintenable, évolutif (SOLID, Design Patterns, etc ...)