C4 Model

Adrien CAUBEL

Références :
- https://c4model.com/
- https://www.youtube.com/watch?v=x2-rSnhpw0g&t=1257s

C4 MODEL

« Model for visualising software architecture »

Le problème

- Si on demande à des architectes/développeurs de nous dessiner une architecture logiciel
  - On va avoir des boxes,
  - Des couleurs,
  - Des traits dans tous les sens !

- MAIS
  - La notation sera incohérente (couleur, format, trait, etc …)
  - Il y aura des nom ambiguë, des fois des annotations ou mot complet
  - Un mix d’abstraction

Solution : UML ?

- Il y a le langage UML qui essaie de répondre à ses problématique
- Tout le monde peut le comprendre
  - PO, SM, devs, client, utilisateur, business manager, etc …

- MAIS
  - Des équipes ont abandonné ce système, au profit de diagramme plus simple
  - Car les notations UML sont très lourdes

  - Ceci n’est pas forcément une très bonne chose …, il faut revenir au VISUEL

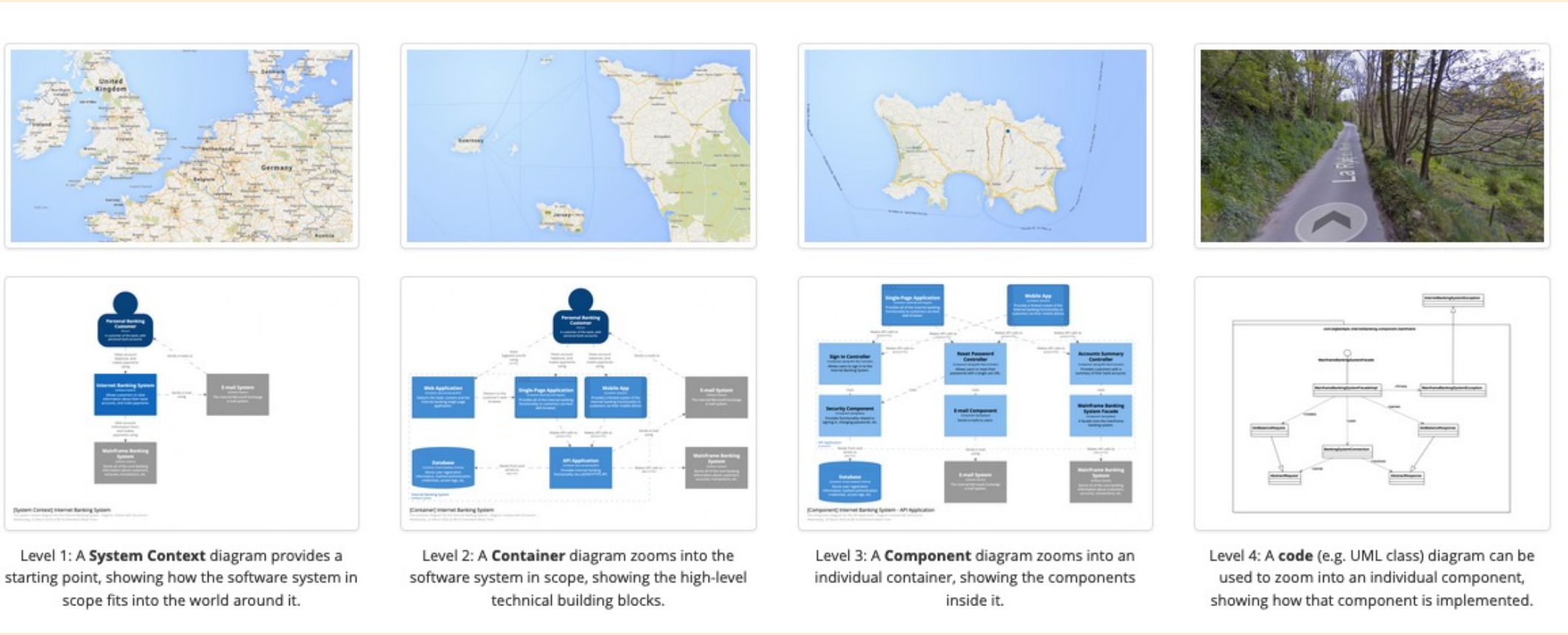
C4 Model

Objectif

Représenter notre système avec différents niveau de détail

Métaphore

Le C4 Model permet de visualiser du code à différentes échelles (zoom / dézoom)
=> On crée une carte (map) de notre code à différent niveau



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

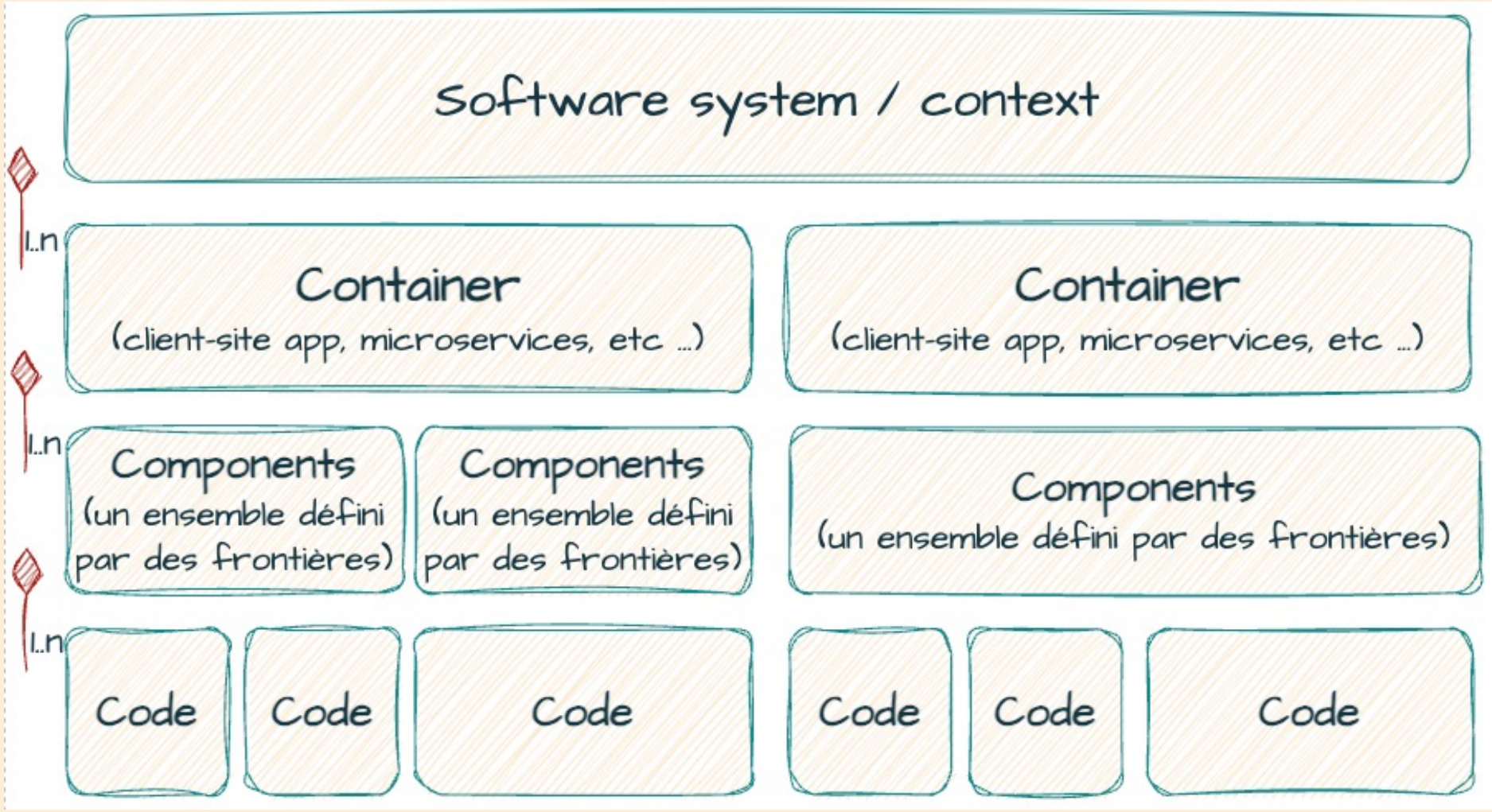
Level 2: A **Container** diagram zooms into the software system in scope, showing the high-level technical building blocks.

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

Les 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)*

#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 une 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

Note : on ne parle pas de containers Docker

#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)

#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 …)

Notation

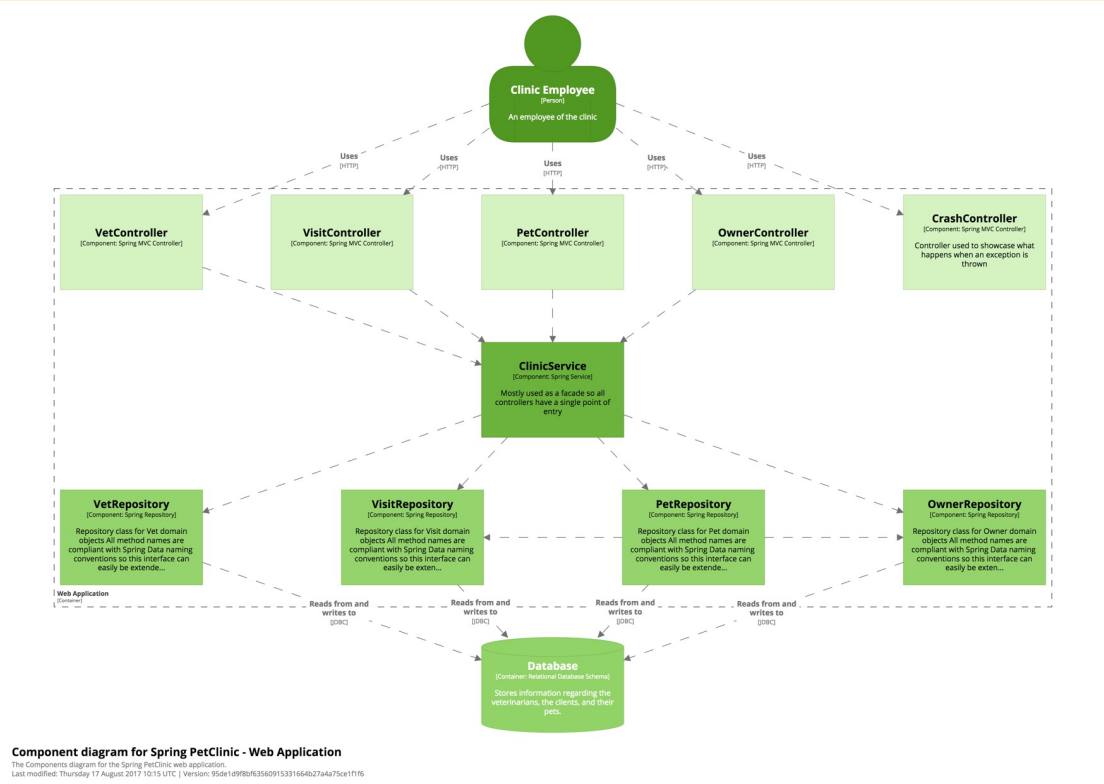
Pas de notation

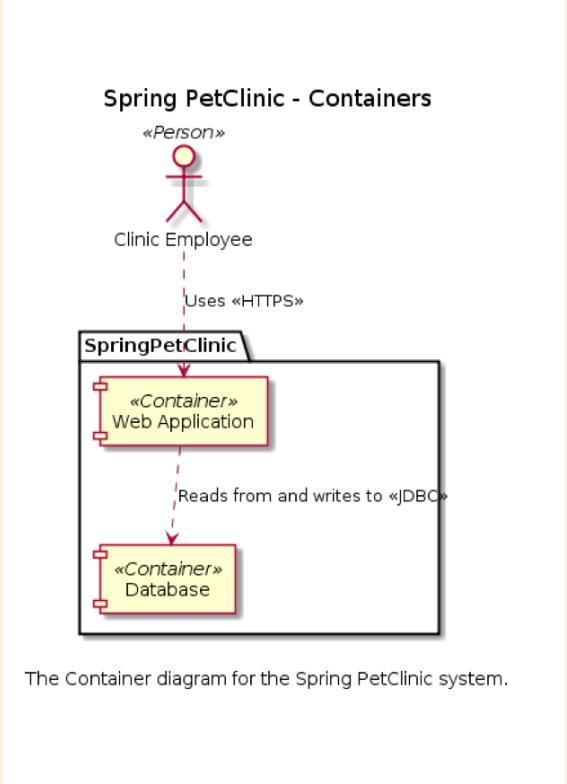
« Le C4 Model est indépendant de toute notation »

- L'équipe choisira sa notation
- Elle propose néanmoins un point de départ

C4 et UML

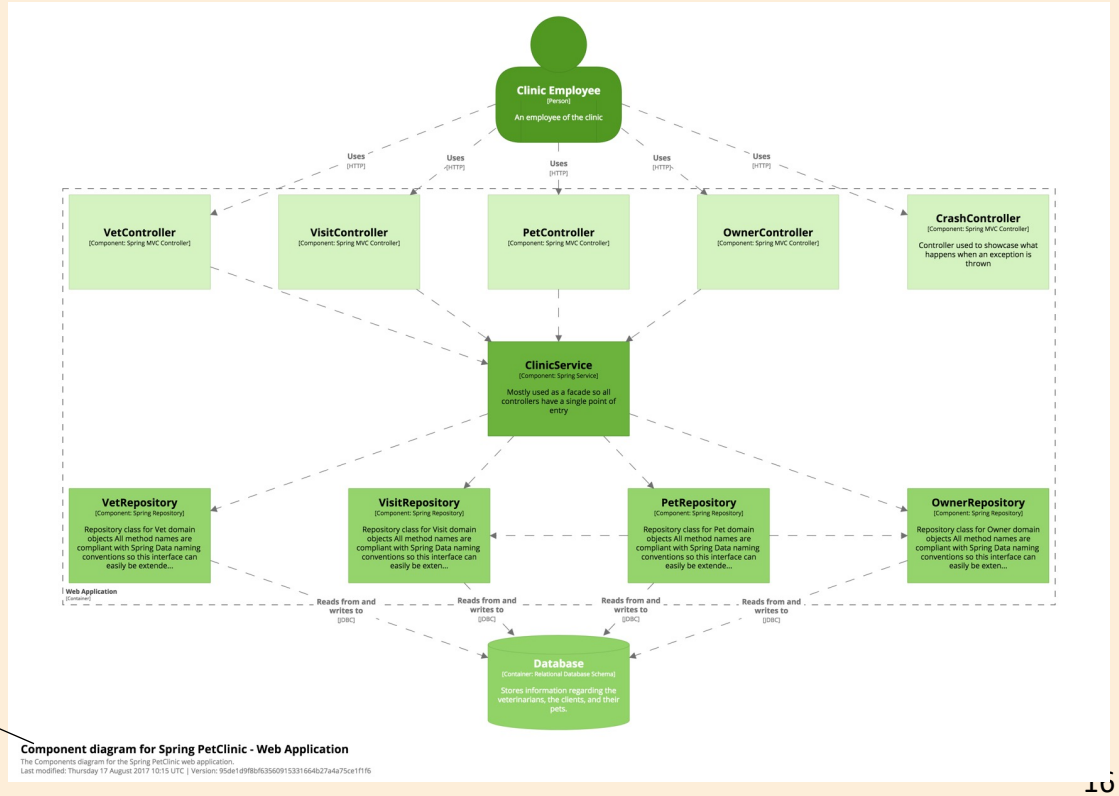
- On peut reprendre les éléments de UML
- Mais le C4 Model se veut bien plus simple (moins de notation)





Mettre les TITRE

- Un diagramme sans titre on comprend pas
  - Ni le niveau (System, Container, Components)
  - Ni l'objectif de la représentation



Component diagram for Spring PetClinic - Web Application

TP 1