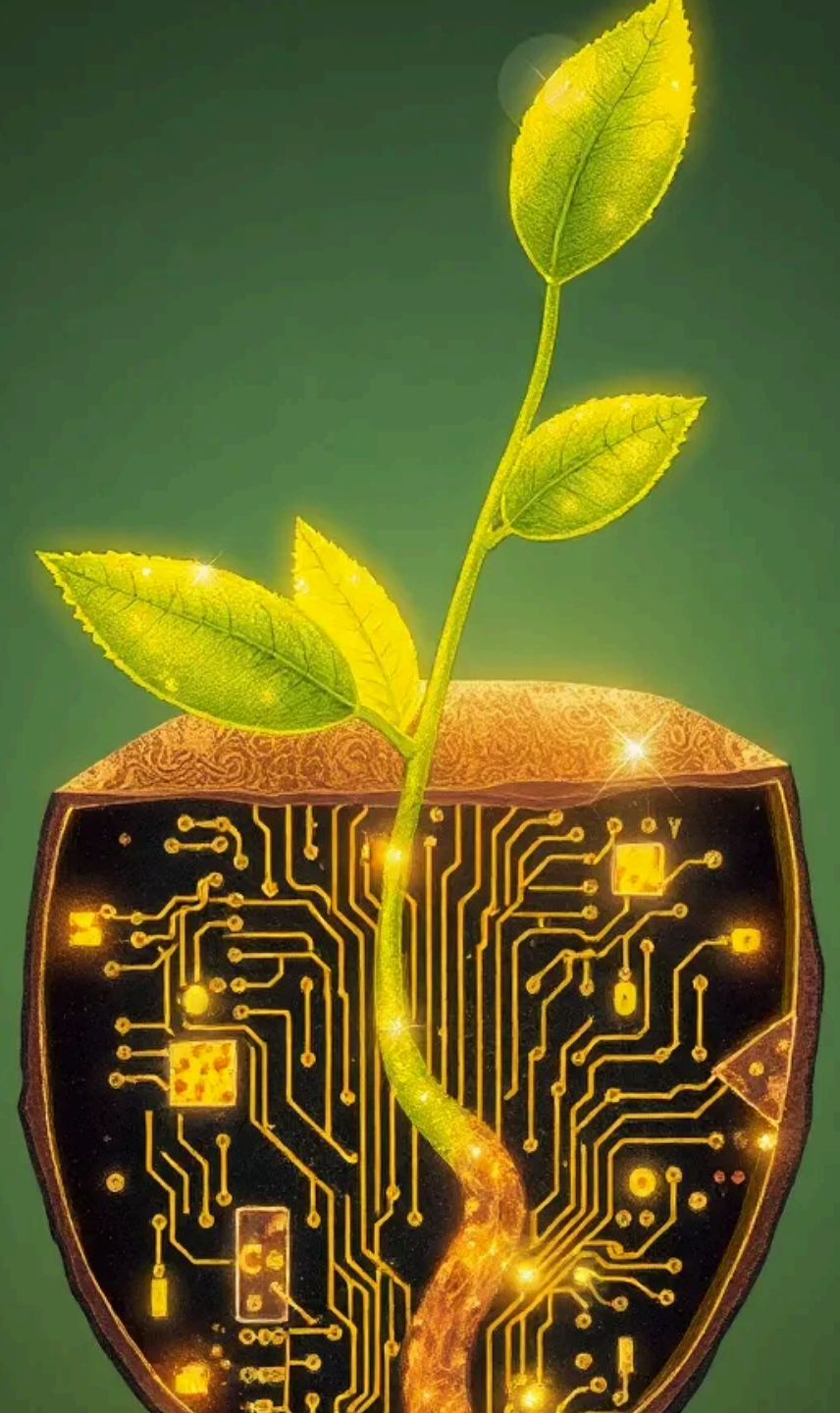


Architecture logicielle

Mise en route

Architecture logicielle — Florian Baune





Ouverture

Ouverture

Objectifs

- Cartographier la notion d'architecture logicielle
- Construire un vocabulaire commun
- Acquérir des outils méthodologiques

Une histoire

- Le cours suit **Panier Voisin** sur 10 ans
- Chaque chapitre représente une étape de vie du projet

Jour 0 — Panier Voisin

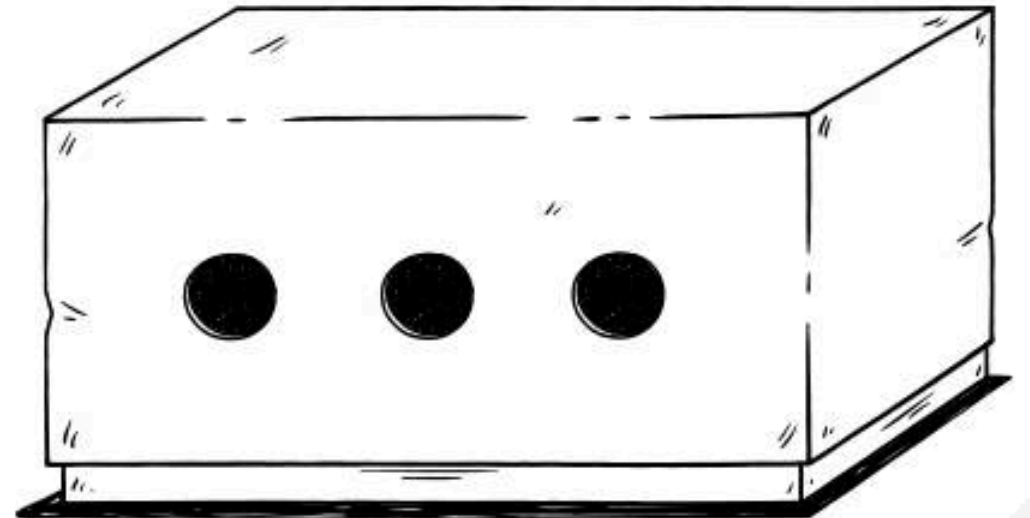
- Une plateforme de vente de paniers de producteurs locaux
- **Camille**, fondatrice
- **Hugo**, premier développeur
- **Vous**, l'équipe technique

*“Ça c'est la caisse.
Le mouton que tu veux est dedans.”*

“C'est tout à fait comme ça que je le voulais !”

CHAPITRE II - LE PETIT PRINCE DE SAINT EXUPÉRY

*Dessine moi une
architecture...*



Architecture Logicielle : Trois définitions



1. Des structures

“

Les structures qui permettent
de raisonner sur un système

”

- Éléments
- Relations
- Propriétés

Sur Panier Voisin

- **Éléments** : front web, API, base de données, service de paiement
- **Relations** : le front appelle l'API, l'API lit et écrit dans la base, l'API délègue le paiement
- **Propriétés** : l'API répond en moins d'une demi-seconde, la base est sauvegardée chaque nuit

2. Des décisions coûteuses à changer

“

Ce qui est difficile
à changer plus tard

”

Sur Panier Voisin

- Changer de base de données pour les commandes (relationnelle $\Leftrightarrow$ documentaire) : migration, réécriture, tests
- Changer la couleur du bouton « Commander » : 5 minutes

La première coûte cher. La seconde, non.

3. Des compromis justifiés

“

Tout est compromis.
Le pourquoi compte plus que le comment.

”

Sur Panier Voisin

« On utilise PostgreSQL » → ne dit rien.

« On utilise PostgreSQL **parce que** commandes et paiements ont besoin de transactions fiables, **et parce que** Hugo le maîtrise » → une décision d'architecture.

Synthèse

L'architecture logicielle, ce sont...

- des **structures**
- des **décisions** coûteuses à changer
- des **compromis** justifiés



Architecture ou design ?

Quiz Collaboratif

Une distinction essentielle

“

Toute architecture est du design,
mais tout design n'est pas de l'architecture.

”

Deux tests

- **Combien ça coûte à changer ?**
- **Jusqu'où la règle s'applique-t-elle ?**

C'est parti !

L'équipe se pose cette question

« Fait-on une seule application pour tout le monde, ou une application pour les clients et une autre pour les producteurs ? »

Réponse

« Une application, ou deux ? » → **Architecture**
Structure tout le système ; très coûteux à changer.

L'équipe se pose cette question

« Les frais changent selon le mode de retrait, et d'autres modes arriveront. Comment les calculer sans que le code devienne illisible ? »

Réponse

« Calcul des frais de retrait » → **Design**

Reste interne au calcul des frais.

L'équipe se pose cette question

« Développe-t-on le paiement nous-mêmes, ou passe-t-on par un prestataire ? »

Réponse

« Paiement maison ou prestataire ? » → **Architecture**
Sécurité, conformité, dépendance à un fournisseur ;
coûteux à défaire.

L'équipe se pose cette question

« Le code de l'interface a-t-il le droit de lire directement la base de données, ou tout doit-il passer par une couche métier ? »

Réponse

« Accès direct à la base, ou couche métier ? » →

Architecture

Règle qui s'applique partout et contraint tous les développeurs.

L'équipe se pose cette question

« Dans la partie qui gère les commandes, comment prévenir le producteur par email aujourd'hui, et par SMS demain, sans tout réécrire ? »

Réponse

« Notifier le producteur (email, puis SMS) » → **Design**
Local à la gestion des commandes.

L'équipe se pose cette question

« Les différentes parties du système s'appellent-elles directement, ou communiquent-elles en s'envoyant des messages ? »

Réponse

« Appels directs, ou messages ? » → **Architecture**

Même idée qu'à la question précédente, mais appliquée à tout le système : c'est une architecture orientée événements.

L'équipe se pose cette question

« Quand un panier contient des produits de plusieurs producteurs, comment répartit-on la commande entre eux ? »

Réponse

« Comment répartit-on la commande ? » → **Design**

Piège : l'algorithme peut être compliqué, mais il reste enfermé dans une partie du système. Complexe ne veut pas dire architectural.

L'équipe se pose cette question

« Stocke-t-on les prix en centimes (nombres entiers) ou en euros avec des décimales ? »

Réponse

« Centimes ou décimales ? » → **Architecture**

Piège : la question paraît anodine, mais elle touche toutes les données et tout le code qui manipule des prix.
Anodin ne veut pas dire design.

L'équipe se pose cette question

« React ou Vue pour l'interface ? »

Réponse

« React ou Vue ? » → **À débattre**

Coûteux à changer pour 2 développeurs, donc plutôt architectural ici ; plus local dans un grand groupe où chaque équipe gère son morceau d'interface.

Synthèse

- Deux tests : coût du changement, portée de la règle
- Complexe ne veut pas dire architectural
- Anodin ne veut pas dire design
- Une même idée change de nature selon sa portée
- La frontière dépend du contexte

Architecture Logicielle : Les trois vues



Trois façons de lire un système

Vue	Question	On y voit
Code	Comment le code est découpé ?	Modules, dépendances
Exécution	Que se passe-t-il à l'exécution ?	Processus, appels, flux
Déploiement	Où est-ce que ça tourne ?	Machines, réseaux

Panier Voisin — vue code

- Modules : Catalogue, Commande, Paiement, Compte
- Commande dépend de Catalogue et de Paiement

Panier Voisin — vue exécution

- Navigateur → API
- API → base de données
- API → service de paiement
- Tâche planifiée → emails producteurs

Panier Voisin — vue déploiement

- Front : hébergeur externe
- API + base : un seul serveur sur site
- Service de paiement : externe

Relecture croisée

Avec le schéma d'un groupe voisin, à l'aide de 3 couleurs
(code, exécution, déploiement) :

- Boîte : du code, quelque chose qui tourne, ou un lieu ?
- Flèche : dépendance, appel ou flux de données ?
- Qu'est-ce qui manque (utilisateurs, services externes, légende) ?



Juste assez d'architecture

Juste assez

La question

Camille a 3 mois et 2 développeurs.

Quelle architecture est suffisante ?

Un MVP, c'est pour apprendre

Un MVP (*Minimum Viable Product*) sert à savoir si l'idée marche — pas à tenir 100 000 utilisateurs.

Le principe

**Investir sur ce qui sera coûteux à changer,
économiser sur tout le reste.**

Faire des choix...

Raccourcis raisonnables

- Site responsive plutôt qu'app mobile
- Un seul serveur
- Emails envoyés pendant la commande (synchrone)
- Pas de cache
- Catalogue saisi à la main
- Déploiement manuel
- Tests seulement sur les prix et le paiement

Bombes à retardement

- Stocker soi-même les numéros de carte
- Un modèle de données à un seul producteur
- Mots de passe en clair
- Aucune sauvegarde
- Règles de prix dupliquées (front et back)

En exergue

« Un seul serveur » est raisonnable.

« Aucune sauvegarde » ne l'est pas.

Synthèse

- Un MVP sert à apprendre
- On investit sur ce qui est coûteux à changer
- Un bon raccourci est **conscient**, avec une **échéance connue**

Clôture



À retenir (1/2)

- **Architecture logicielle** : structures, décisions et compromis coûteux à changer
- **Design** : décisions locales, qui ne contraignent qu'une partie du système
- **Décision d'architecture** : choix structurant justifié par un « pourquoi »

À retenir (2/2)

- **Vue** : angle de lecture du système (code, exécution, déploiement)
- **MVP** (*Minimum Viable Product*) : version minimale pour tester une idée
- **Raccourci conscient** : simplicité assumée, échéance connue

Architecture logicielle

Architecture logicielle — Florian Baune

