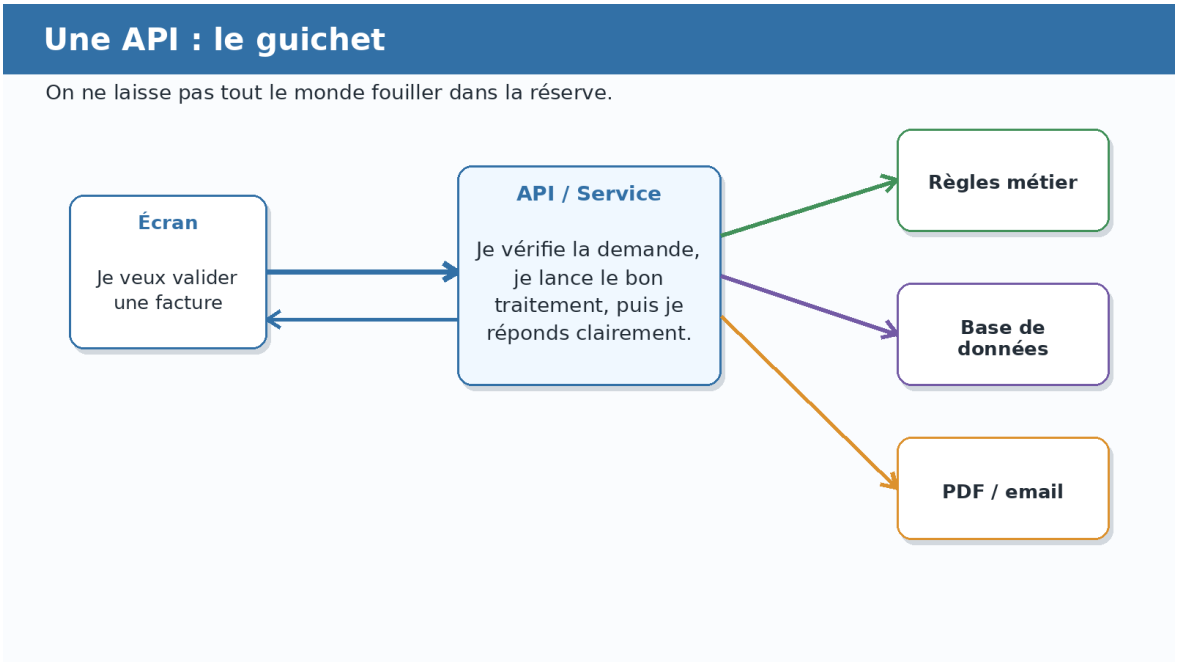
MASTERCLASS 3

API et architecture

Transformer ton application WINDEV en système migrable

Guide pratique pour développeurs



L'API vue comme un guichet : simple à comprendre, très utile en migration.

Objectif : apprendre à découper une application WINDEV existante pour la rendre plus ouverte, plus propre et plus facile à migrer progressivement.

Avant-propos

Ce support accompagne la masterclass n°3 sur la migration d'applications WINDEV. Il est écrit comme un guide pratique : on part d'une application existante, souvent ancienne, et on apprend à la rendre plus migrable sans tout casser.

Le ton est volontairement simple. L'idée est d'expliquer les notions d'architecture comme on les expliquerait à un jeune de 10 ans : avec des images, des analogies et des étapes concrètes. Mais le contenu reste fait pour des développeurs.

Idée centrale
On ne migre pas facilement un gros bloc. On migre mieux des morceaux. Le travail consiste donc à créer des morceaux propres : services, règles métier séparées, accès aux données isolés, puis éventuellement API.

Comment utiliser ce guide

- Lis une première fois pour comprendre la logique générale.
- Reviens ensuite avec ton propre projet WINDEV ouvert à côté.
- Choisis une fenêtre importante, mais pas la plus dangereuse.
- Utilise les tableaux et checklists pour faire ton diagnostic.
- Commence par extraire un petit traitement, pas par réécrire l'application complète.

Ce que tu dois obtenir à la fin

- Une carte simple de ton application : interface, métier, données, services techniques.
- Une liste de traitements candidats à une couche de services.
- Une première idée de points d'entrée API.
- Un plan de transition qui évite le "grand remplacement".

Table des matières

1. Le gros bloc : pourquoi le monolithe bloque la migration

Dans beaucoup d'applications WINDEV, tout est au même endroit. Une fenêtre affiche les champs, lit la base, contrôle les droits, calcule les montants, imprime les états, envoie les emails et affiche les messages.

Au départ, c'est confortable. On développe vite. On clique sur un bouton, on écrit le code, on obtient le résultat. Mais après plusieurs années, le projet ressemble parfois à une chambre où l'on aurait rangé les vêtements, les outils, les factures, les câbles et les casseroles dans le même placard.

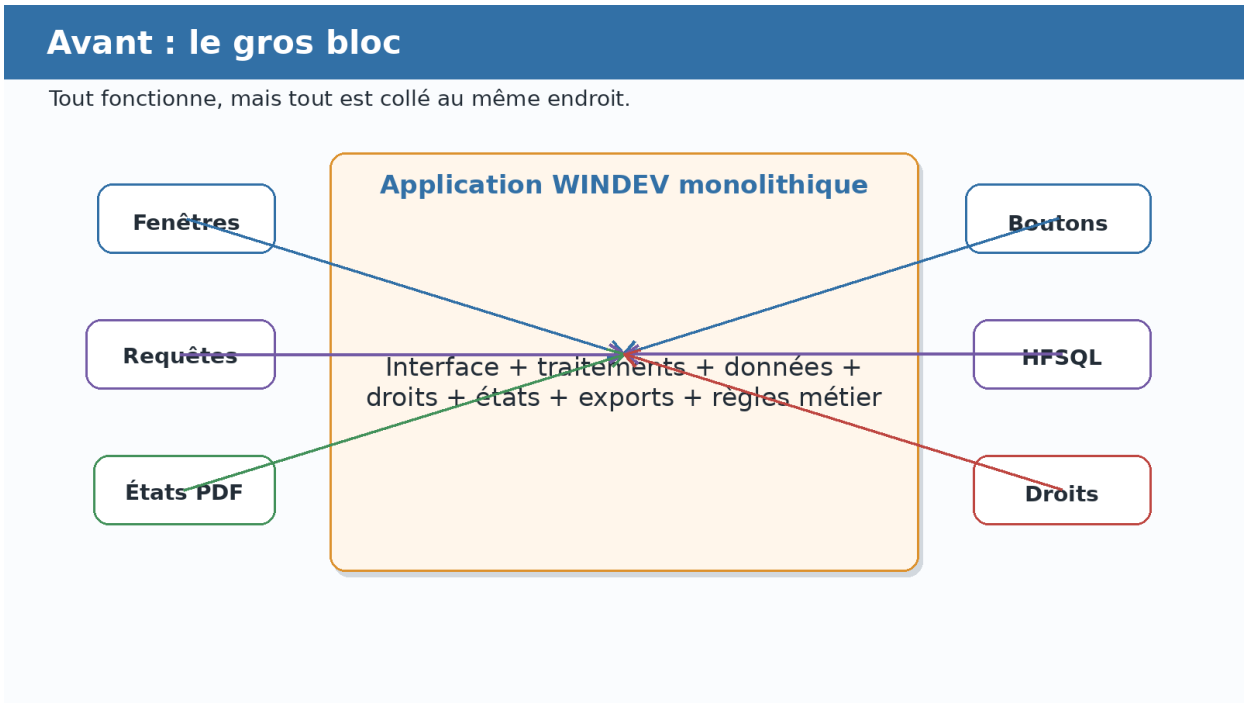


Figure 1 - Le monolithe : tout est collé, donc tout devient difficile à déplacer.

Un monolithe, ce n'est pas forcément un mauvais programme

Un monolithe est une application qui contient beaucoup de choses dans un même ensemble. Ce n'est pas automatiquement une catastrophe. Beaucoup de logiciels métiers très utiles sont monolithiques et fonctionnent très bien.

Le problème apparaît quand on veut changer une partie sans toucher au reste. Par exemple : refaire seulement l'écran de saisie en web, remplacer la base de données, créer une appli mobile, ou exposer quelques fonctions à un outil externe.

Pourquoi c'est dur à migrer

Ce qui est mélangé	Ce que ça provoque en migration
Code métier dans les boutons	On ne peut pas le réutiliser facilement dans une autre interface.
Requêtes dans les fenêtres	La logique d'accès aux données est dispersée partout.
Droits utilisateurs dans chaque écran	La sécurité devient difficile à centraliser.
États et exports appelés directement	On ne sait plus ce qui dépend de quoi.
Règles non documentées	On découvre les cas particuliers trop tard.

Image simple
Imagine que tu veux déménager une maison. Si les assiettes, les vêtements et les vis sont tous dans le même carton, tu vas perdre du temps. Une migration, c'est pareil : il faut faire des cartons propres.

Le but de cette masterclass

Le but n'est pas de tout réécrire. Le but est de rendre l'application plus respirable. On veut créer des séparations simples pour pouvoir changer une partie sans faire tomber toute la maison.

2. Le principe du découpage : ranger la maison avant le déménagement

Avant de parler API, il faut parler rangement. Une API posée sur un code emmêlé ne résout pas le problème. Elle le rend parfois visible depuis l'extérieur.

Découper, c'est répondre à une question simple : "Qui doit faire quoi ?".

Les grandes familles à séparer

Famille	Rôle simple	Exemple WINDEV
Interface	Montrer les choses à l'utilisateur et récupérer ses actions.	Fenêtre, champ de saisie, bouton, table, onglet.
Métier	Appliquer les règles importantes du logiciel.	Calculer une remise, valider une facture, bloquer une commande.
Données	Lire, écrire, modifier, supprimer les informations.	HLitRecherche, HAjoute, HModifie, requête SQL.
Services techniques	Rendre un service autour du métier.	Créer un PDF, envoyer un email, exporter Excel.
Sécurité	Dire qui a le droit de faire quoi.	Profil utilisateur, droits, journalisation.

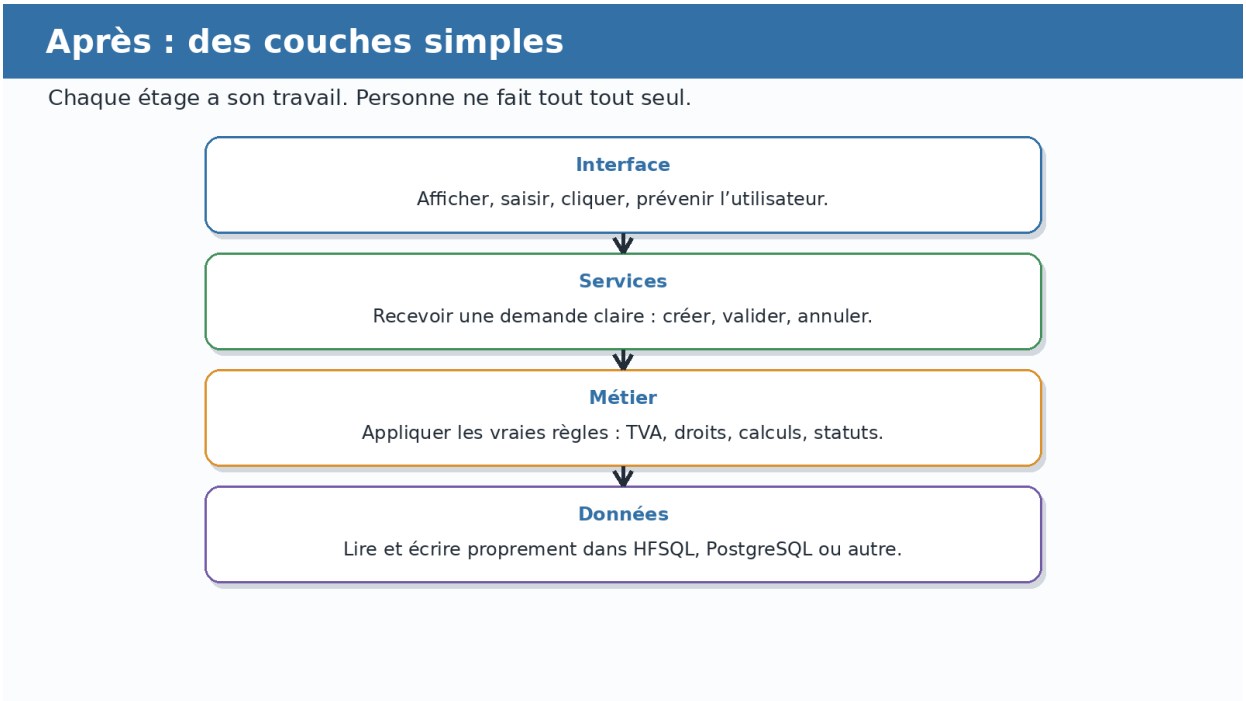


Figure 2 - Une architecture en couches : chaque étage a son travail.

La règle du cartable

Un enfant ne met pas son goûter, son cahier de maths, ses chaussures de sport et son poisson rouge dans la même poche du cartable. Sinon, le soir, tout est collé et personne n'est content.

Dans une application, c'est pareil. Le bouton ne devrait pas tout porter. Il peut déclencher une action, mais il ne devrait pas contenir toute l'intelligence du logiciel.

Ce qu'on veut obtenir

```
Fenêtre WINDEV
    -> récupère les valeurs affichées
    -> appelle un service
    -> affiche le résultat

Service métier
    -> vérifie les règles
    -> appelle la couche données
    -> appelle les services techniques si besoin

Couche données
    -> lit et écrit dans la base
```

Phrase à retenir

Le bouton demande. Le service réfléchit. La couche données range. Cette phrase suffit souvent à comprendre toute la logique.

3. API : le guichet entre l'ancien et le nouveau

Une API est un point d'entrée propre. Elle permet à un écran, un module web, une application mobile ou un autre logiciel de demander quelque chose sans connaître tout l'intérieur de l'application.

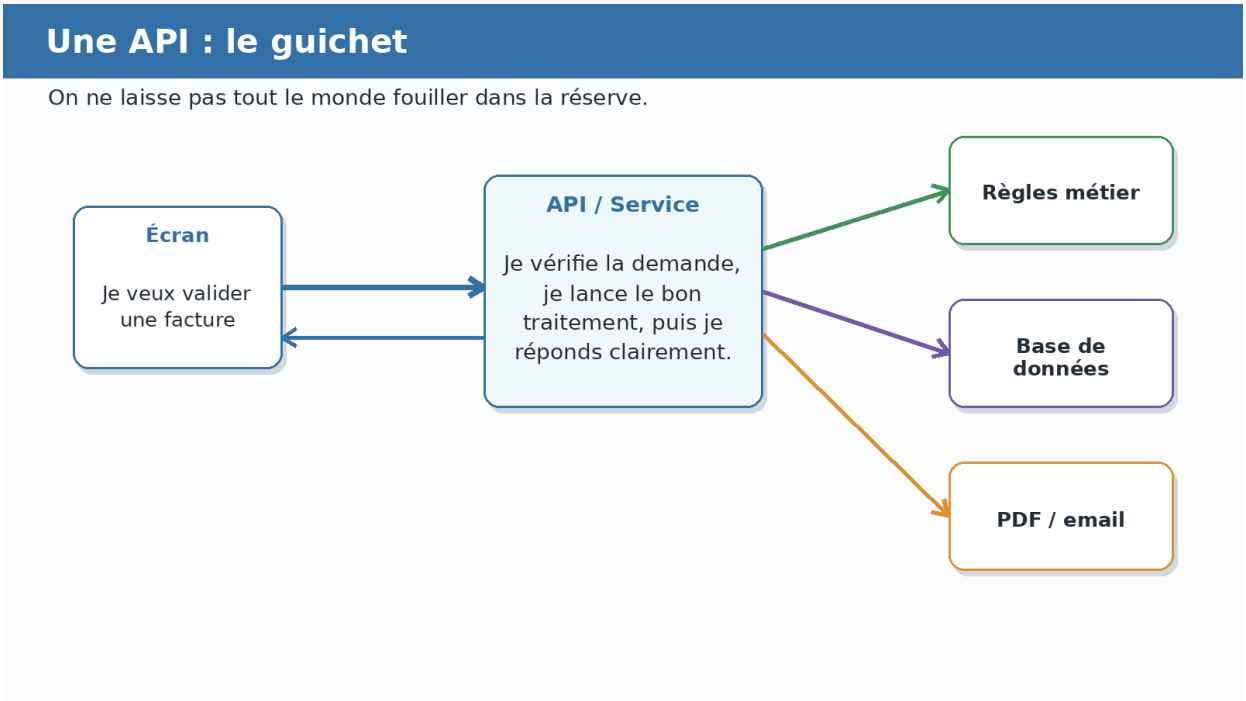


Figure 3 - L'API comme guichet : elle reçoit, contrôle, transmet et répond.

Définition très simple

Une API, c'est comme un guichet dans une mairie. Tu ne rentres pas dans tous les bureaux pour chercher un dossier. Tu vas au guichet et tu demandes : "Je veux un acte de naissance". Le guichet vérifie, transmet au bon service, puis te donne une réponse.

Dans une application métier

Au lieu de laisser chaque fenêtre faire ce qu'elle veut directement dans la base, on peut créer des demandes claires :

```
CréerClient
LireClient
ValiderCommande
CréerFacture
EnvoyerFactureParEmail
ObtenirSoldeClient
```

Pourquoi l'API aide à migrer

Sans API	Avec API
Le nouveau module doit comprendre l'ancien code.	Le nouveau module appelle un point d'entrée clair.
Chaque écran peut faire différemment.	Les règles passent par le même service.
La base est souvent touchée directement.	L'accès aux données peut être mieux contrôlé.
Le remplacement est brutal.	La transition peut être progressive.

API REST en quelques mots

Dans le monde actuel, beaucoup d'API sont des API REST. Elles utilisent des requêtes HTTP, comme le web. On peut avoir par exemple :

```
GET    /clients/42
POST   /factures
POST   /factures/123/valider
GET    /factures/123/pdf
```

Dans l'écosystème PC SOFT, la documentation indique qu'un Webservice REST est un programme hébergé sur un serveur avec des fonctions accessibles via des requêtes HTTP. La fonction RESTEnvoie permet d'envoyer une requête REST et d'attendre la réponse du serveur. Voir les références en annexe.

> **Attention**
> Une API n'est pas magique. Si elle appelle un vieux traitement confus, elle donnera juste un accès moderne à un vieux bazar. Le vrai travail reste le découpage.

4. Faire l'inventaire des traitements dans WINDEV

Avant de modifier le code, il faut observer. Le premier travail n'est pas de coder. Le premier travail est de faire une carte.

Choisir une fenêtre de départ

Ne commence pas par la fenêtre la plus dangereuse du projet. Choisis une fenêtre utile, connue, mais pas explosive. Par exemple : une fiche client, une commande, une facture, une intervention, un stock.

Méthode en 5 étapes

1. Ouvre la fenêtre dans WINDEV.
2. Liste les boutons et traitements principaux.
3. Repère les accès à la base de données.
4. Repère les calculs et règles métier.
5. Repère les services techniques : PDF, email, export, impression, API externe.

Exemple : FEN_Facture

Action trouvée	Famille	Question à poser
Vérifier que le client existe	Métier + données	Est-ce une règle commune à plusieurs écrans ?
Calculer la TVA	Métier	Ce calcul doit-il être réutilisable ailleurs ?
Ajouter la facture	Données	Est-ce toujours fait de la même façon ?
Générer le PDF	Service technique	Peut-on appeler ce service depuis un autre module ?
Envoyer l'email	Service technique	Faut-il tracer l'envoi ?
Afficher "Facture validée"	Interface	Ce message appartient-il seulement à l'écran ?

La fiche d'inventaire

Traitement	Où est le code ?	Famille	Risque	Candidat service ?
Valider facture	BTN_Valider	Métier	Fort	Oui
Calcul TVA	BTN_Valider	Métier	Moyen	Oui
Imprimer facture	PROC locale	Service technique	Moyen	Oui
Afficher erreur	Fenêtre	Interface	Faible	Non

Petit conseil
Ne cherche pas la perfection au premier passage. Mets déjà des mots sur ce que fait le code. Tu verras ensuite ce qui mérite d'être extrait.

5. Séparer interface, métier et données

C'est le cœur de la masterclass. Quand l'interface, le métier et les données sont séparés, l'application devient plus facile à tester, à comprendre et à migrer.

Le rôle de l'interface

L'interface doit parler à l'utilisateur. Elle affiche des champs, récupère des clics, montre des messages. Elle ne devrait pas porter toute la logique du logiciel.

```
Bouton Valider :
    - récupérer les valeurs de l'écran
    - appeler le service de validation
    - afficher le résultat
```

Le rôle du métier

Le métier, ce sont les règles qui font que ton application est utile. C'est souvent la partie la plus précieuse du logiciel. Elle doit être protégée, clarifiée et documentée.

```
FactureService.ValiderFacture :
    - vérifier le client
    - vérifier les lignes
    - calculer les montants
    - changer le statut
    - demander l'enregistrement
```

Le rôle des données

La couche données sait comment lire et écrire. Elle peut utiliser HFSQL aujourd'hui, PostgreSQL demain, ou autre chose plus tard. Plus les accès sont isolés, plus la migration de base sera maîtrisable.

```
FactureRepository.Enregistrer :
    - créer l'enregistrement facture
    - créer les lignes
    - gérer les erreurs de base
    - retourner un résultat clair
```

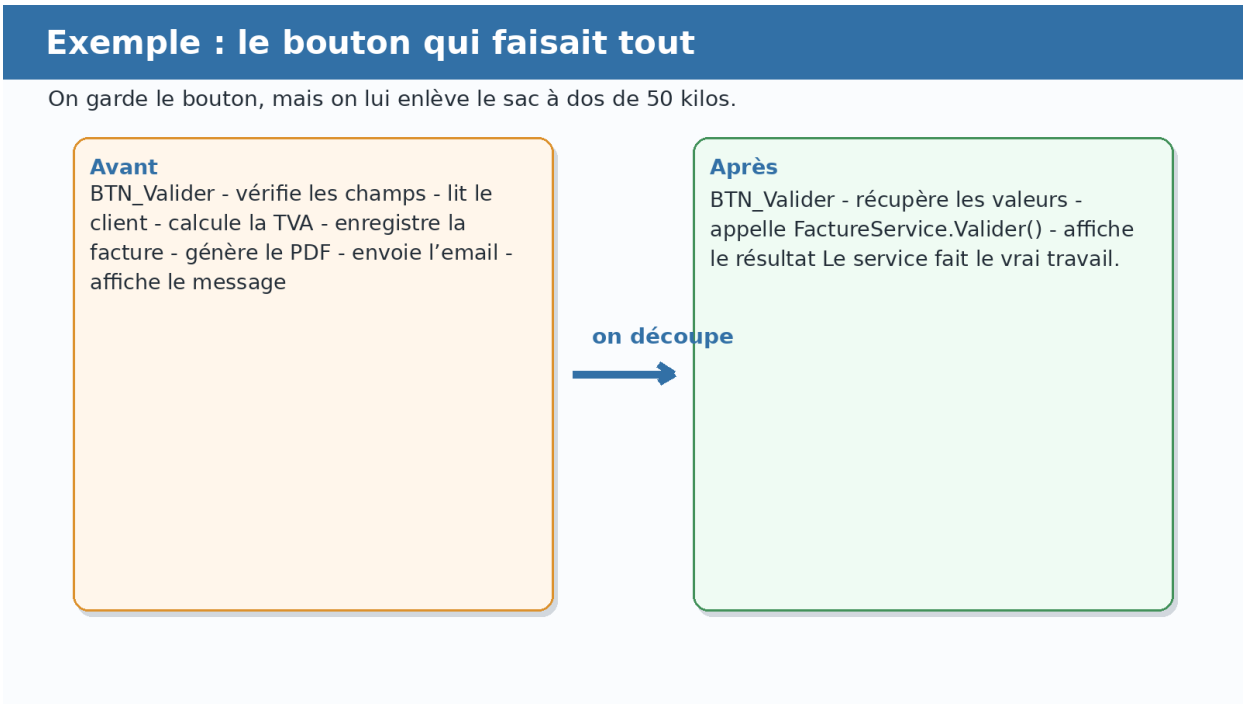


Figure 4 - Le bouton reste simple ; le service porte la logique.

Pourquoi cette séparation aide vraiment

Bénéfice	Explication simple
Réutilisation	Le même service peut être appelé par une fenêtre, une API ou un module web.
Tests	On peut tester une règle sans ouvrir toute l'interface.
Migration	On peut migrer un morceau sans refaire tout le logiciel.
Lisibilité	Un développeur comprend plus vite où chercher.
Sécurité	Les règles importantes peuvent être centralisées.

6. Créer une première couche de services internes

Avant de créer une API publique ou externe, commence par créer des services internes. C'est une étape plus simple et souvent plus sûre.

Qu'est-ce qu'un service interne ?

Un service interne est un morceau de code qui rend un service précis à l'application. Il peut être une classe, un ensemble de procédures, un composant interne, ou une organisation claire dans ton projet. Le nom exact dépend de ton style de développement, mais l'idée reste la même.

Exemples de services

Service	Responsabilité
ClientService	Créer, modifier, lire, vérifier un client.
FactureService	Créer, valider, annuler, calculer une facture.
StockService	Contrôler les quantités, réserver, sortir du stock.
DocumentService	Générer PDF, documents, impressions.
EmailService	Préparer et envoyer des emails.
UtilisateurService	Gérer les droits et profils.

Règle d'or : entrée claire, sortie claire

Un service doit recevoir des informations précises et retourner une réponse compréhensible. Il ne doit pas aller chercher discrètement tout dans l'écran.

```
Mauvais réflexe :
    Le service lit directement les champs de FEN_Facture.

Meilleur réflexe :
    La fenêtre prépare une demande.
    Le service reçoit cette demande.
    Le service retourne un résultat.
```

Modèle simple de demande/réponse

```
DemandeValidationFacture
    id_facture
    id_utilisateur
    date_validation
    options

ResultatValidationFacture
    succes : vrai/faux
    message : texte
    id_facture
    erreurs : liste
```

Image simple

Un service, c'est comme un cuisinier. Tu lui donnes une commande claire. Il ne vient pas fouiller dans ton assiette pour deviner ce que tu veux.

7. Transformer certains services en API

Une fois qu'un service interne est clair, il peut devenir un candidat API. Cela ne veut pas dire que tous les services doivent être exposés. On expose seulement ce qui a une vraie utilité.

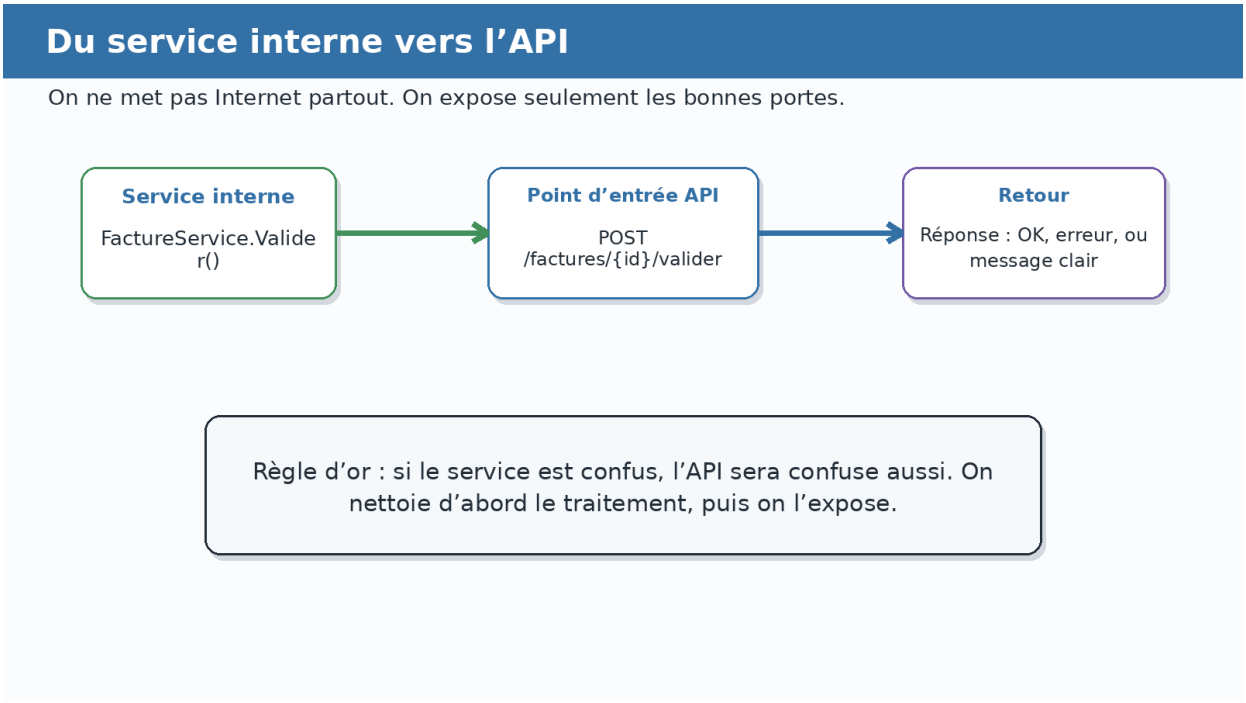


Figure 5 - On expose une porte propre, pas toute la maison.

La question à poser

Avant d'exposer un traitement, pose cette question : "Est-ce qu'un autre système a vraiment besoin de demander cela ?".

Exemples de bons points d'entrée

Besoin métier	Point d'entrée possible	Pourquoi c'est utile
Lire une fiche client	GET /clients/{id}	Utile pour web, mobile, support, automatisation.
Créer une commande	POST /commandes	Action métier centrale.
Valider une facture	POST /factures/{id}/valider	Règle métier importante à centraliser.
Télécharger un PDF	GET /factures/{id}/pdf	Utile depuis plusieurs interfaces.
Connaître le stock	GET /articles/{id}/stock	Utile pour consultation rapide.

Ce qu'il vaut mieux éviter au début

À éviter	Pourquoi
Exposer toute la base table par table	Tu risques de créer un accès direct déguisé à la base.
Exposer une énorme procédure de 2000 lignes	Tu rends le problème accessible à distance, mais tu ne le règles pas.
Commencer par le traitement le plus critique	Trop risqué pour une première étape.
Créer 80 endpoints dès la première semaine	Tu vas produire de la complexité avant d'avoir appris.

API orientée données ou orientée métier ?

Une API orientée données ressemble à une télécommande pour tables : ajouter client, modifier facture, supprimer ligne. Ce n'est pas forcément faux, mais cela peut exposer trop directement le modèle de données.

Une API orientée métier expose des actions qui ont du sens pour l'entreprise : valider une facture, transformer une commande, réserver un stock, clôturer une intervention.

```
Moins métier :
POST /facture/modifier-statut

Plus métier :
POST /factures/{id}/valider
POST /factures/{id}/annuler
POST /commandes/{id}/transformer-en-facture
```

8. Sécurité, erreurs et journalisation

Dès qu'on crée une API, on ouvre une porte. Une porte peut être très utile, mais elle doit être contrôlée.

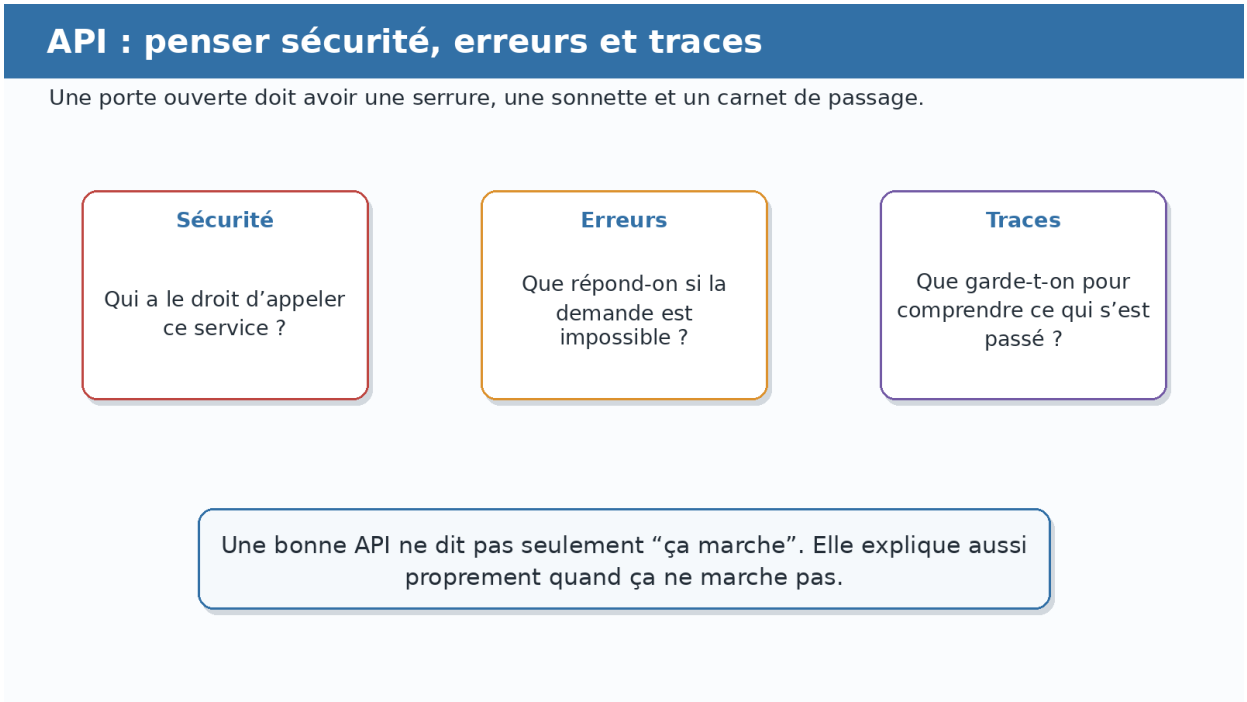


Figure 6 - Une API sérieuse prévoit sécurité, erreurs et traces.

Sécurité : qui a le droit ?

Une API ne doit pas répondre à n'importe qui. Il faut prévoir une authentification, des droits, des profils, et parfois des restrictions selon l'origine de l'appel.

Dans les Webservices REST WINDEV/WEBDEV, la documentation PC SOFT décrit notamment la sécurisation par jetons OAuth. Le détail exact dépendra de la version, du type de projet et du mode de déploiement. Voir les références en annexe.

Erreurs : que répond l'API ?

Une API doit répondre proprement. Elle ne doit pas seulement dire "erreur". Elle doit aider l'appelant à comprendre ce qui s'est passé.

Situation	Réponse utile
Client introuvable	client_introuvable + message compréhensible.
Droit insuffisant	acces_refuse + indication non sensible.
Facture déjà validée	facture_deja_validee + statut actuel.
Erreur technique	erreur_technique + identifiant de trace.

Journalisation : garder des traces

Quand une API est utilisée, il faut pouvoir répondre à ces questions : qui a appelé ? quand ? pour faire quoi ? avec quel résultat ?

```
Trace minimale recommandée :
- date et heure
- utilisateur ou application appelante
- endpoint appelé
- identifiant métier concerné
- succès ou échec
- message d'erreur technique si besoin
```

Important

Ne mets pas de secrets dans les logs : pas de mot de passe, pas de jeton complet, pas de données personnelles inutiles.

9. Faire cohabiter WINDEV et les nouveaux modules

Une migration sérieuse n'oblige pas à couper l'ancien système d'un coup. Le plus souvent, l'ancien et le nouveau doivent cohabiter pendant une période.

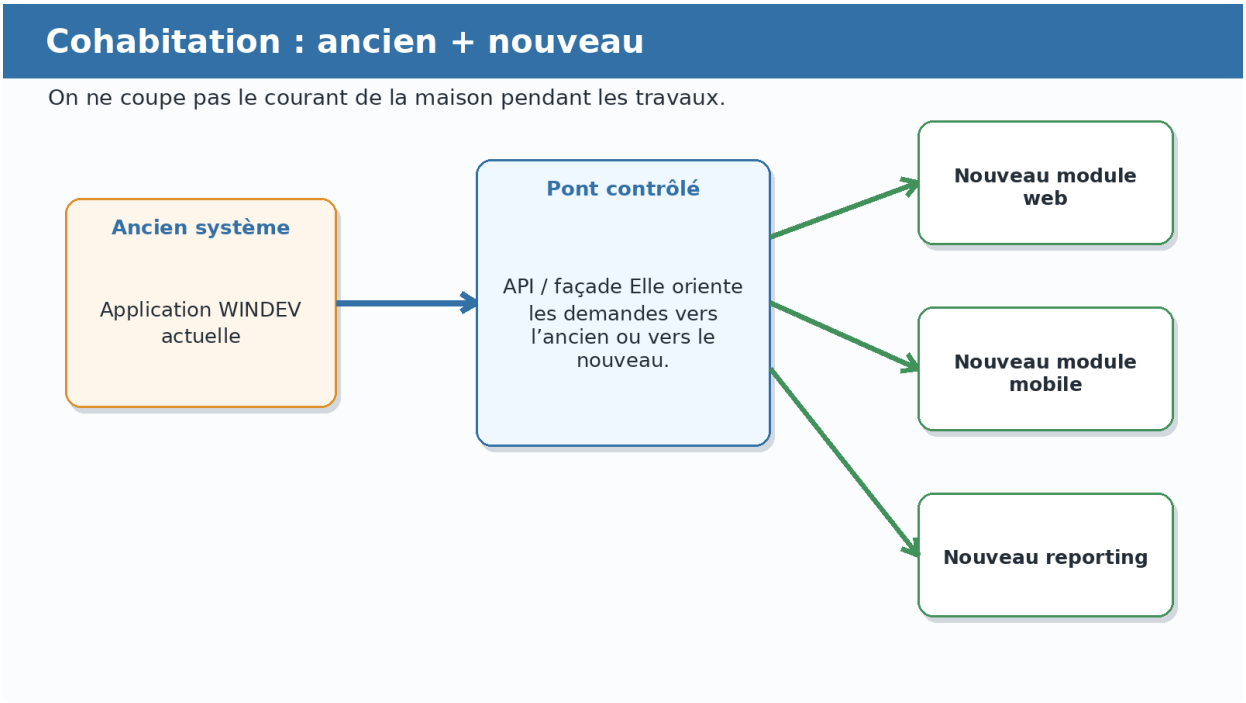


Figure 7 - L'ancien système continue de vivre pendant que les nouveaux modules arrivent.

Le scénario progressif

L'approche progressive consiste à remplacer petit à petit certaines fonctions par de nouveaux modules. Le système historique continue de fonctionner pendant la transition.

Cette logique est proche du "Strangler Fig pattern", une approche de modernisation où l'on remplace progressivement des fonctionnalités d'un système existant par de nouveaux services ou applications. Voir la référence Microsoft en annexe.

Exemple de transition

Étape	Ce qui se passe
1	L'application WINDEV reste l'application principale.
2	On crée une API de lecture clients.
3	Un petit module web consulte les clients via l'API.
4	On ajoute une API de commandes.
5	Les commerciaux utilisent un module mobile pour créer des commandes.
6	Certaines fonctions WINDEV deviennent moins utilisées.

Pourquoi cette méthode rassure

- Les utilisateurs ne perdent pas brutalement leurs outils.
- Le risque est limité à un petit périmètre.
- On peut tester sur un module avant d'aller plus loin.
- On découvre les règles cachées progressivement.
- On garde une solution de secours.

Image simple
C'est comme refaire une maison pièce par pièce. Tu ne dors pas dehors pendant les travaux. Tu gardes une pièce utilisable, puis tu avances.

10. Décider ce qui reste et ce qui migre en premier

Tout ne mérite pas d'être migré tout de suite. La bonne migration commence par un choix intelligent des priorités.

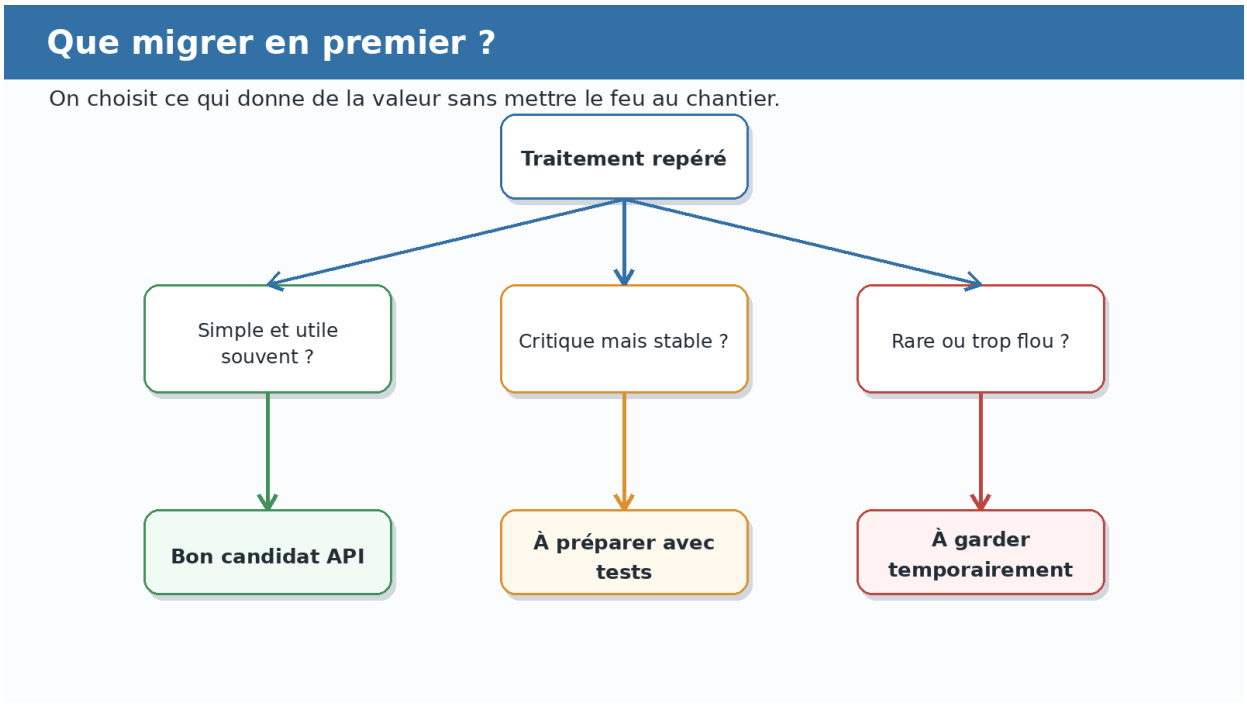


Figure 8 - Choisir les bons premiers candidats évite de bloquer le chantier.

Ce qui peut rester temporairement dans WINDEV

Élément	Pourquoi le garder au début
États très complexes	Les refaire peut être long et peu rentable au départ.
Écrans d'administration rares	Peu utilisés, donc pas prioritaires.
Fonctions anciennes mais stables	Aucun intérêt à casser ce qui marche sans raison.
Traitements très spécifiques	Ils doivent être compris avant d'être déplacés.

Ce qui peut migrer plus tôt

Élément	Pourquoi c'est souvent prioritaire
Consultation client	Simple, utile, peu risqué.
Création de demande terrain	Souvent utile en mobile ou web.
Validation métier centrale	Permet de centraliser une règle importante.
Exports stratégiques	Utile pour reporting et automatisation.
Lecture de référentiels	Bon candidat pour commencer une API.

La matrice de décision

Question	Si oui	Si non
Le traitement est-il souvent utilisé ?	Candidat intéressant.	Priorité plus basse.
Est-il stable ?	Plus simple à exposer.	À clarifier avant migration.
Est-il utile à d'autres modules ?	Bon candidat service/API.	Peut rester interne.
Est-il très risqué ?	Prévoir tests et garde-fous.	Bon exercice de départ.
Est-il bien compris ?	On peut avancer.	Documenter d'abord.

Phrase à retenir

Migrer proprement, ce n'est pas tout refaire. C'est choisir ce qu'on sort, dans quel ordre, et pourquoi.

11. Atelier pratique : découper une gestion de factures

Dans cet atelier, on part d'un cas très courant : une fenêtre WINDEV de facture avec un bouton Valider qui fait beaucoup trop de choses.

Situation de départ

```
BTN_Valider
    1. Vérifier que le client existe
    2. Vérifier les lignes de facture
    3. Calculer le total HT
    4. Calculer la TVA
    5. Appliquer une remise
    6. Enregistrer la facture
    7. Enregistrer les lignes
    8. Générer le PDF
    9. Envoyer l'email
    10. Afficher "Facture validée"
```

Étape 1 : classer les actions

Action	Interface	Métier	Données	Service technique	API possible
Vérifier client		Oui	Oui		Oui
Calculer TVA		Oui			Oui
Appliquer remise		Oui			Oui
Enregistrer facture			Oui		Oui
Générer PDF				Oui	Oui
Envoyer email				Oui	Oui
Afficher message	Oui				Non

Étape 2 : proposer les services

```
FactureService
    ValiderFacture
    CalculerTotaux
    AnnulerFacture

ClientService
    VerifierClientActif

FactureRepository
    EnregistrerFacture
    EnregistrerLignes

DocumentService
    GenererPDFFacture

EmailService
    EnvoyerFacture
```

Étape 3 : proposer les points d’entrée API

```
POST /factures/{id}/valider
GET  /factures/{id}
GET  /factures/{id}/pdf
POST /factures/{id}/envoyer-email
GET  /clients/{id}/statut
```

Étape 4 : définir une réponse claire

```
Réponse si tout va bien :
{
  "succes": true,
  "message": "Facture validée",
  "id_facture": 123,
  "statut": "validée"
}

Réponse si problème :
{
  "succes": false,
  "code_erreur": "client_inactif",
  "message": "Le client n’est pas actif."
}
```

Exercice à faire avec ton projet

6. Choisis une fenêtre qui valide ou crée quelque chose.
7. Liste toutes les actions déclenchées par le bouton principal.
8. Classe chaque action dans une famille.
9. Entoure les actions métier réutilisables.
10. Propose un service interne.
11. Imagine un endpoint API seulement si un autre module pourrait en avoir besoin.

12. Plan d’action en 30 jours

Ce plan permet de passer de la théorie à l’action sans bloquer ton développement quotidien.

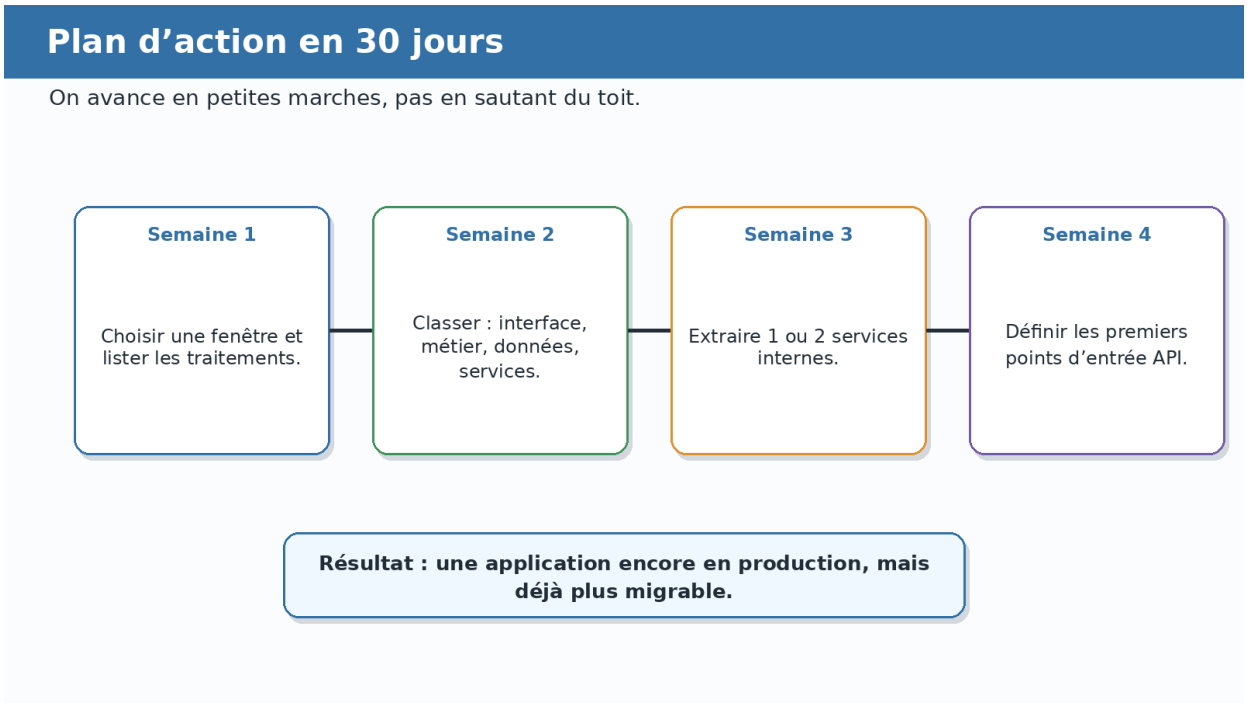


Figure 9 - Une progression simple pour obtenir un premier résultat en 30 jours.

Semaine 1 : observer

- Choisir une fenêtre importante mais maîtrisable.
- Lister les boutons et traitements principaux.
- Repérer les accès aux données.
- Repérer les règles métier.
- Noter les dépendances : états, emails, exports, modules externes.

Semaine 2 : classer

- Classer chaque traitement : interface, métier, données, service technique, sécurité.
- Repérer les règles répétées dans plusieurs fenêtres.
- Repérer les traitements utiles à d’autres interfaces.
- Choisir un premier traitement candidat.

Semaine 3 : extraire

- Créer un premier service interne.
- Déplacer un petit traitement dedans.
- Faire en sorte que la fenêtre appelle le service.
- Tester que le comportement reste identique.
- Documenter l’entrée, la sortie et les erreurs.

Semaine 4 : préparer l’API

- Décrire le premier point d’entrée possible.
- Définir les données d’entrée.

- Définir la réponse en succès et en erreur.
- Prévoir sécurité, logs et droits.
- Décider si l'API doit être créée maintenant ou plus tard.

Objectif réaliste
Au bout de 30 jours, tu n'as pas migré toute l'application. Mais tu as prouvé que tu pouvais extraire un morceau proprement. C'est énorme.

13. Annexes pratiques

Checklist 1 - Une fenêtre est-elle trop chargée ?

Question	Oui / Non
Le bouton principal contient-il plus de 100 lignes de code ?	
La fenêtre lit-elle directement plusieurs fichiers ou tables ?	
La fenêtre calcule-t-elle des règles métier importantes ?	
La fenêtre génère-t-elle des PDF ou exports ?	
La fenêtre envoie-t-elle des emails ?	
La fenêtre vérifie-t-elle des droits localement ?	
Le même calcul existe-t-il dans plusieurs fenêtres ?	

Checklist 2 - Un traitement peut-il devenir un service ?

Critère	Score 0 à 3
Il est utile à plusieurs endroits.	
Il a une règle métier claire.	
Il reçoit des données simples.	
Il peut retourner un résultat clair.	
Il peut être testé sans interface.	
Il est assez stable.	

Interprétation simple : plus le score est élevé, plus le traitement est un bon candidat pour devenir un service.

Checklist 3 - Avant d'exposer une API

Point à vérifier	OK ?
Le service interne existe déjà ou est clairement défini.	
Les données d'entrée sont connues.	
Les réponses succès/erreur sont décrites.	
Les droits sont définis.	
Les logs sont prévus.	
Le traitement est testé.	
On sait qui utilisera cette API.	

Mini vocabulaire

Mot	Explication simple
Monolithe	Application où beaucoup de choses sont dans un même bloc.
Couche	Zone de responsabilité : interface, métier, données, etc.
Service	Morceau de code qui rend un service précis.
API	Porte d'entrée propre pour demander une action ou une information.
Endpoint	Adresse précise d'une API, par exemple GET /clients/42.
Payload	Données envoyées dans une requête.
Token	Jeton utilisé pour prouver qu'on a le droit d'appeler.
Log	Trace écrite pour comprendre ce qui s'est passé.

Anti-pièges

Piège	Meilleur réflexe
Créer une API avant de comprendre le code.	Faire l'inventaire d'abord.
Exposer directement toutes les tables.	Exposer des actions métier utiles.
Tout migrer d'un coup.	Migrer morceau par morceau.
Oublier les droits.	Prévoir la sécurité dès le début.
Ne pas journaliser.	Tracer les appels importants.
Créer une architecture trop compliquée.	Rester simple et lisible.

14. Modèles réutilisables

Modèle de fiche de traitement

Champ	À remplir
Nom du traitement	
Fenêtre ou procédure actuelle	
Description simple	
Famille	Interface / métier / données / service technique / sécurité
Données d'entrée	
Résultat attendu	
Erreurs possibles	
Dépendances	
Candidat service ?	Oui / Non
Candidat API ?	Oui / Non
Priorité	Faible / moyenne / forte

Modèle de fiche API

Champ	Exemple
Nom	Valider une facture
Endpoint	POST /factures/{id}/valider
Utilisateur autorisé	Comptable, administrateur
Entrée	id_facture, id_utilisateur, options
Sortie succès	succes=true, statut=validée
Sortie erreur	code_erreur, message
Trace à garder	date, utilisateur, id_facture, résultat
Ancien traitement appelé	FactureService.ValiderFacture

Architecture cible minimale

```
[Interface WINDEV]
        |
        v
[Services internes]
        |
        +--> [Règles métier]
        +--> [Accès données]
        +--> [Services techniques]

Puis, quand c'est utile :

[Nouveau module web/mobile]
        |
        v
[API REST]
        |
        v
[Services internes]
```

Conclusion

Une application WINDEV ancienne n'est pas forcément une mauvaise application. C'est souvent une application qui a beaucoup travaillé, beaucoup évolué, beaucoup encaissé.

Mais à force de tout porter dans le même bloc, elle finit par manquer d'air.

L'objectif n'est pas de casser ce qui fonctionne. L'objectif est de créer des ouvertures : des services, des API, des frontières propres.

Bref : faire respirer l'application pour pouvoir la faire évoluer sans tout jeter.

15. Références techniques vérifiées

Ces références ont été utilisées pour vérifier les points techniques mentionnés dans le guide.

- PC SOFT - Créer un Webservice REST - https://doc.pcsoft.fr/fr-FR/?1000022792=
- PC SOFT - RESTEnvoie, fonction d'appel REST - https://doc.pcsoft.fr/fr-FR/?1000021476=
- PC SOFT - Webservice REST : appeler une fonction du Webservice - https://doc.pcsoft.fr/fr-FR/?1410091009=
- PC SOFT - Secure REST web service / OAuth tokens - https://doc.pcsoft.fr/en-US/?1410090997=&product=WB
- Microsoft Azure Architecture Center - Strangler Fig pattern - https://learn.microsoft.com/en-us/azure/architecture/patterns/strangler-fig