iBelieve 2021
Présent et Futur de l'IBM i

Evènement
on-line
18 Nov 21

UNE API POSÉE SUR UN MONOLITHE NE POURRA JAMAIS VOLER TRÈS LOIN !

Un évènement organisé par :

itheis & IBM

Avec :

FRESCHE GAIA GROUPE

Avec :

FRESCHE GAIA Gamma soft GROUPEHISI REMAIN M81 Resiliane XCAST


UNE API POSÉE SUR UN
MONOLITHE NE POURRA JAMAIS
VOLER TRÈS LOIN !
(Mais, si il suit « Les trois règles de la softwaristique » ...)

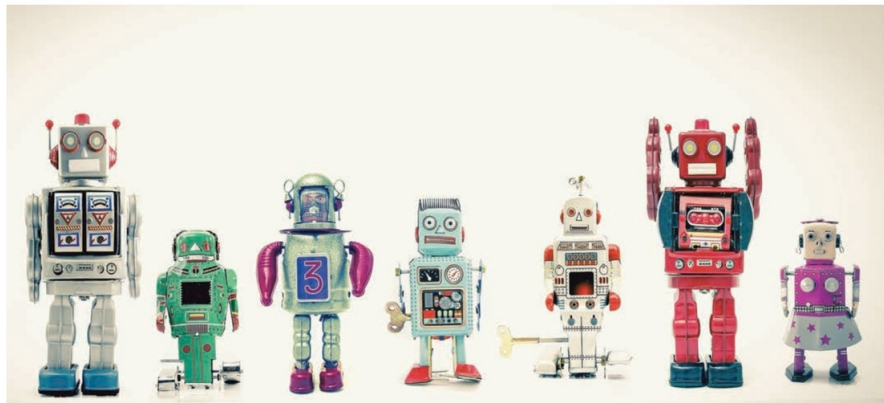
Intervenant

Champion IBM, Pascal Polverini possède une solide expérience dans les architectures logicielles de bout en bout, avec une expertise sur les bases de données, les langages de programmation, les protocoles de communication et l'interface graphique. Co-auteur de l'IBM i Modernization Redbook, il est un expert en cross-technologies.

Expert en stratégie de solutions et informatique

Pascal Polverini

Redbooks Author - IBM Champion



THREE LAWS of Softwaristics

Building on the fundamentals means easier evolution

In 1942, Isaac Asimov introduced the sci-fi literature world to the Three Laws of Robotics:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

movement of its electrons.

Pascal Polverini is a software architect at Fresche Legacy and co-authored the latest IBM i modernization Redbooks publication.

The Three Laws of Softwaristics

Thinking about software application development, debugging, maintenance, deployment and overall application evolution, I propose that the same law structure be applied to software:

1. Software may only use open standards or, if nonexistent, use interoperable formats.
2. Software must use a multitier architecture, except where such architecture would conflict with the First Law.
3. Software must centralize its logic, as long as such centralization does not conflict with the First or Second Law.

Law 1: Open Standards

This law applies to software languages as well as data formats. If the technical specification for the language or the format is public, then it's open.

Ruby and PHP are open-standard programming languages, XML and HTML are open-standard formats, and HTTP is an open-standard protocol. And while not truly open, Java* and RPG follow standards as well, rather than introducing cost and risk by reinventing the wheel. This also introduces risk for the sustainability of the software, as whoever uses it after you may not be able to maintain it. However, when standards are followed, you don't have to think about it anymore, as the standard secures the sustainability of the software as technology evolves. The technological evolution will either change the standard and provide a means of conversion from the old version to a new one or offer new products that will integrate the existing standard.

A good example of this would be Open Office. To make it widely used by the community, the developers created a converter from Microsoft* Word. The new RPG IV standard is similar, with the command CVTRPGSRC to convert RPG III to RPG IV.

In the absence of open standards, you may use interoperable formats—not so much for programing languages, but for data storage and exchange. Programing languages can be platform- or device-specific or cross-platform or -device, and that doesn't mean you have to choose between integration and deployment capacity. What is determinant is mainly the purpose of your app. If it's destined for the cloud, you may look for a multitenant back-end system and options to also enable offline syncing from your front-end device. In any case, even if you choose platform- and device-specific languages, you may use a suitable file format for data preservation and sharing. Therefore, deployment won't be an issue now or later.

Law 2: Multitier

Tiers could differ in naming, but the standard multitier model comprises three core tiers in three respective blocks: The data tier, which stores the data; the application tier, which processes the data; and the presentation tier, which presents the data and enables input.

For example, Excel spreadsheets include all three tiers in one single block, which is bad from an application perspective. This is why the .csv format was invented. It represents a data tier block separated from the presentation tier block. Text messaging uses two tiers in one block: presentation and data. More sophisticated software, like ERP or CRM, will have all three tiers in different blocks for databases, business processes and different presentation channels.

Having an app divided into multitier blocks allows each tier to be separately developed, tested, executed, reused or substituted.

A tier could be logical or physical. Physical tiers are entire blocks (e.g., a database file or a cascading style sheet), and it's easy to substitute a physical tier with another one.

Logical tiers can be considered as a layout within their environment. For example, DDS and RPG each comprise two tiers, but they have a layout in common: the buffer definition. Within the DDS, this layout is mixed with the UI presentation layout (another layout). While this can be seen as advantageous, as you get a centralized buffer description for both the RPG (application tier) and the file device (presentation tier), it would be better to have two distinct blocks—one for the buffer description and another for the presentation layout—even if they are within the same container. This way it becomes equivalent to a true tier block that can be easily substituted. (Do you wish DDS were XML? Read "Life After DDS" ibmsystemsmag.com/ibmi/developer/rpg/oamos-intro.)

A well-designed app will use different physical tiers, but also logical tiers.

For instance, in the application tier, a logical tier might have a distinct function for a validation process and, furthermore, this distinct function could also be placed in a separated physical module. The same principle can be used for data in the database—a distinct file or aggregate for NoSQL could be created.

Finally, a multitier architecture will increase the agility and sustainability of changing your app with market technology or device evolutions.

Law 3: Central Logic

This law concerns logical rules, (aka business rules). You can have absolute or relative rules.

An absolute rule can contemplate the type of a data field, its validation or its correlation to another field. For instance, a field can be set to always display a date or a number, or to always be validated in a unique way. It can also be set to not exist if it's not defined in a header file.

A relative rule can depend on the user login, contextual data or the environment. For example, if I must show a grid made of five columns on a GUI, I may show the entire grid on a browser desktop but only one column on a smartphone.

What matters in our open-standard and multitier premises is that absolute rules are defined at the data tier or, if not possible, at the application tier. Relative rules are defined at the application tier and, if not applicable, at the presentation tier.

The goal is to write any business rule once and on a single place. But if, for application reasons like speed transaction, you need to replicate the business rule in different places, you can still integrate a procedure that inherits (or references) the rule from a central place.

Finding Meaning in the Fundamentals

When you use fundamentals, you gain coherent results—but more than that, you gain durability. Because your model won't be the consequence of a casual "context," it's in harmony with the overall ecosystem; therefore it persists through evolutions. Actually, we can say that the fundamentals are the ecosystem. IT is not a natural science; therefore its ecosystem can change, but within it we can still look for fundamentals. P

Les Trois Lois: Robotique - Softwaristique

1. Un robot ne peut porter atteinte à un être humain ni, restant passif, laisser cet être humain exposé au danger.

2. Un robot doit obéir aux ordres donnés par les êtres humains, sauf si de tels ordres entrent en contradiction avec la première loi.

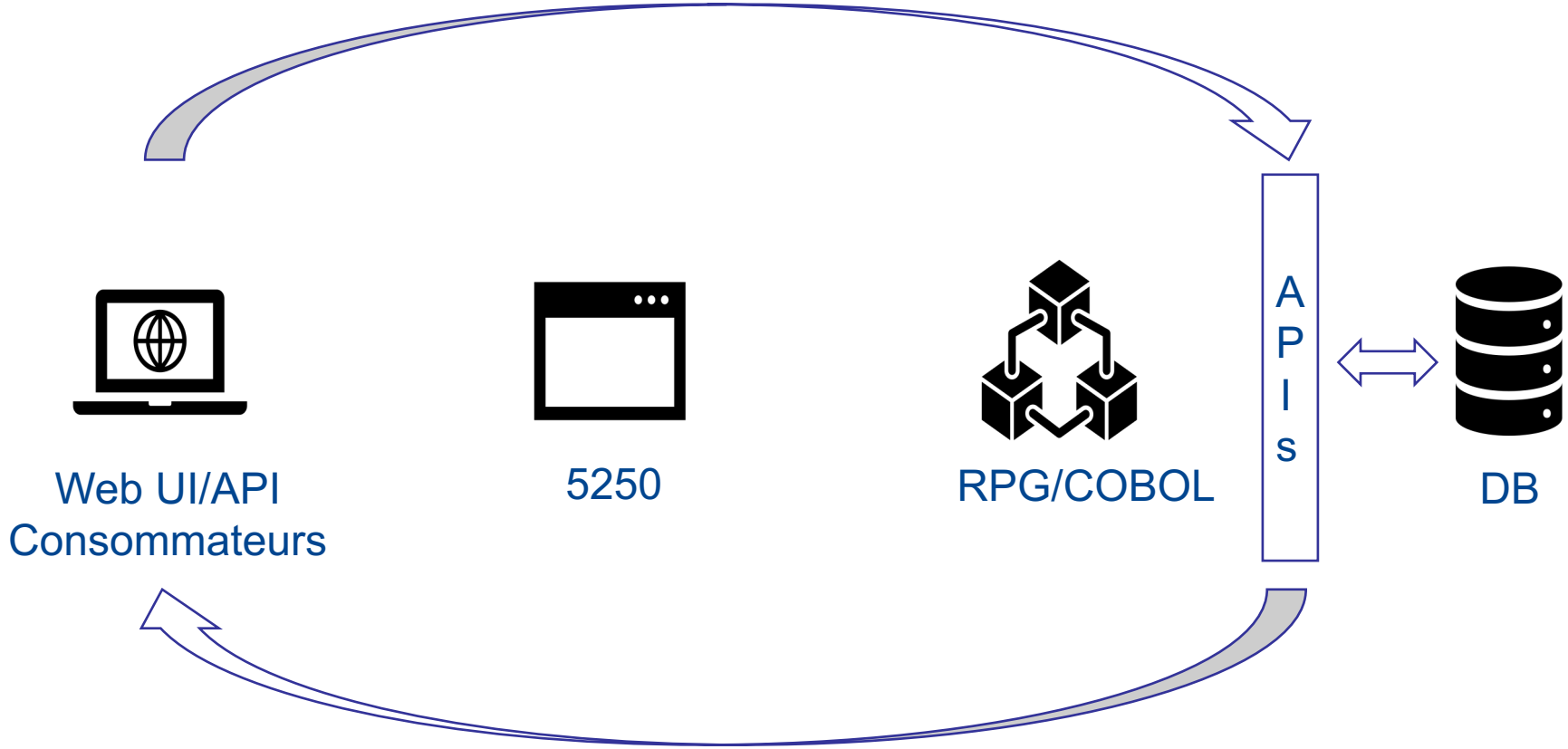
3. Un robot doit protéger son existence dans la mesure où cette protection n'entre pas en contradiction avec la première ou la deuxième loi.

1. Un logiciel ne peut utiliser que des standards ouverts ou, s'ils sont inexistants, utiliser des formats interopérables.

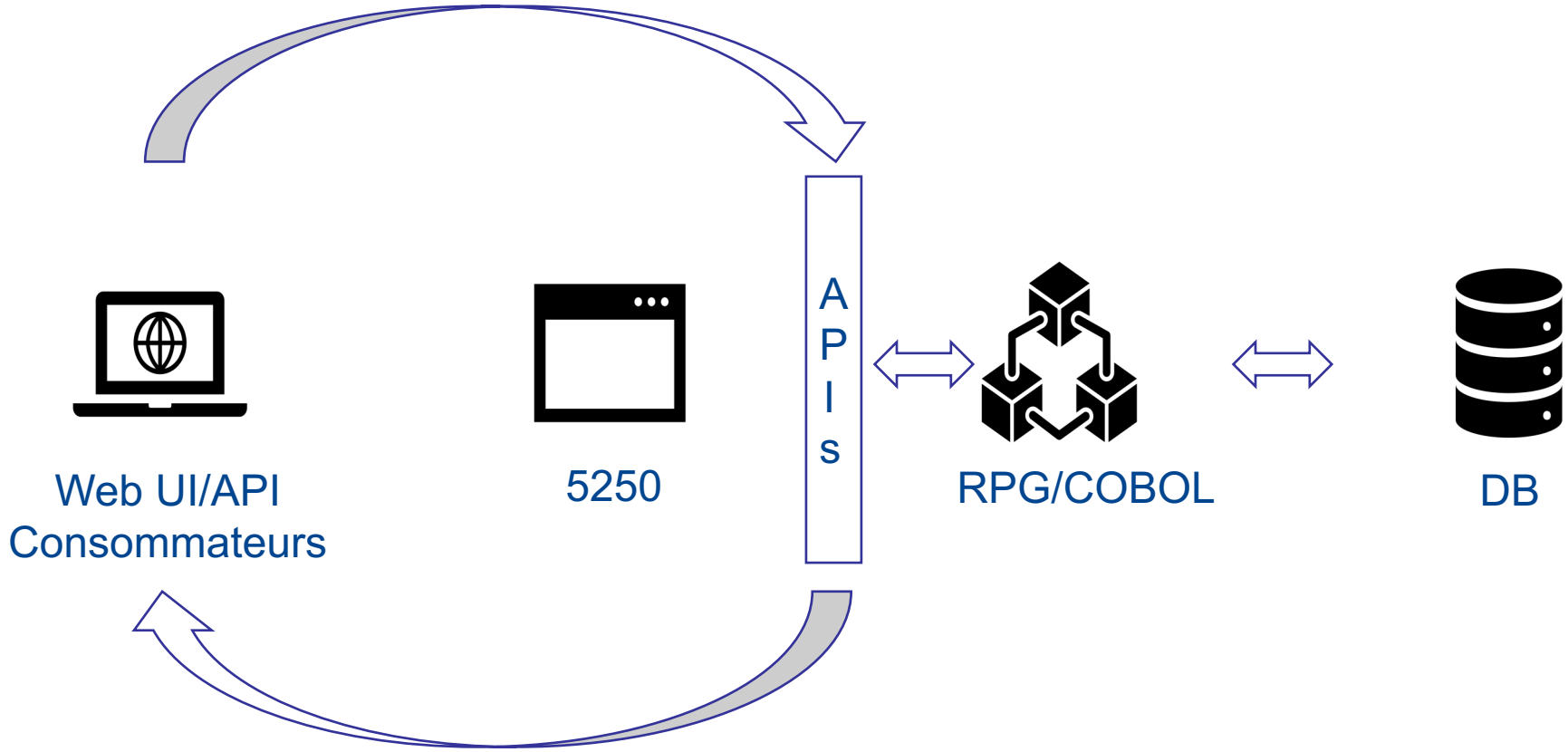
2. Un logiciel doit utiliser une architecture à plusieurs niveaux, sauf lorsqu'une telle architecture entrerait en contradiction avec la première loi.

3. Un logiciel doit centraliser sa logique, tant qu'une telle centralisation n'entre pas en contradiction avec la première ou la deuxième loi.

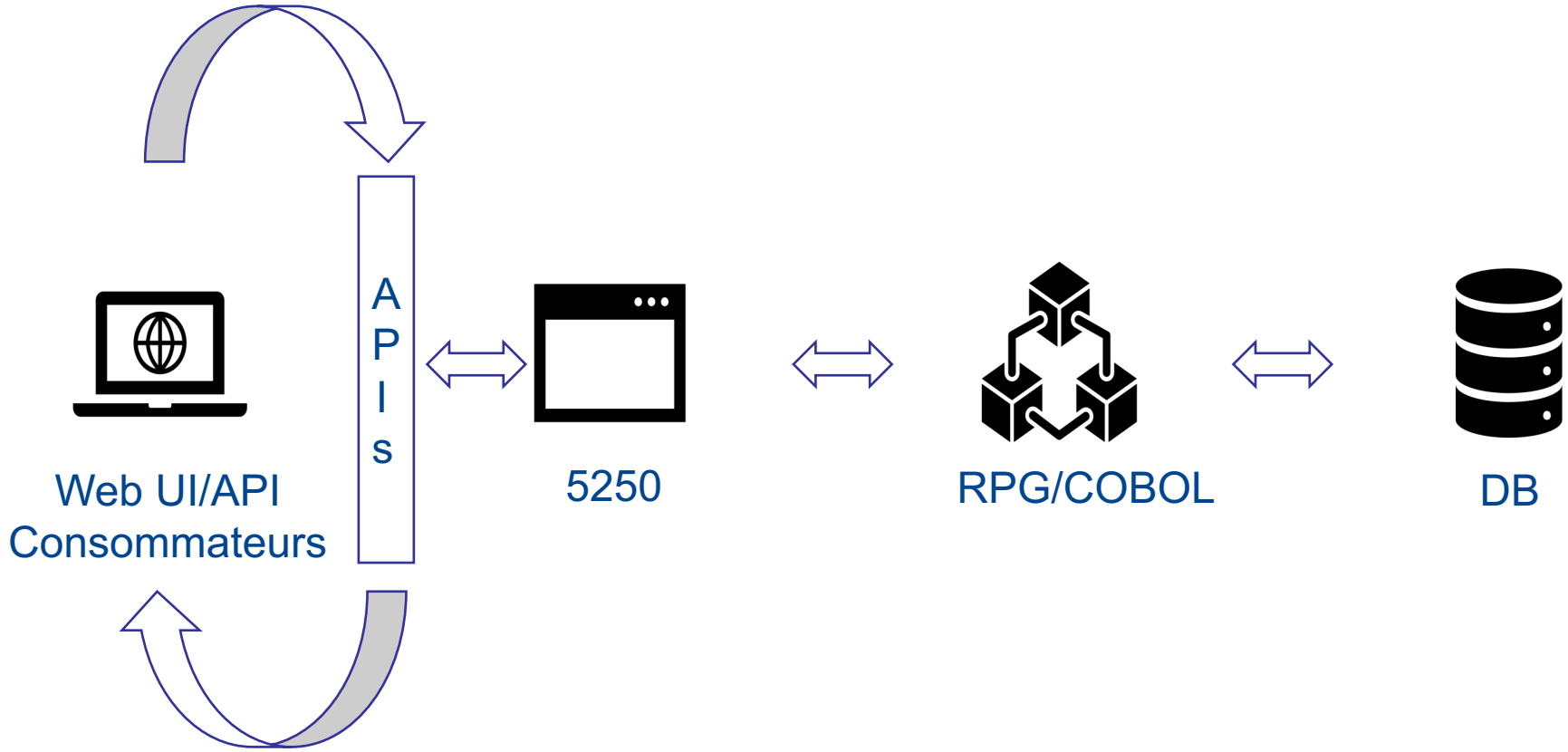
API



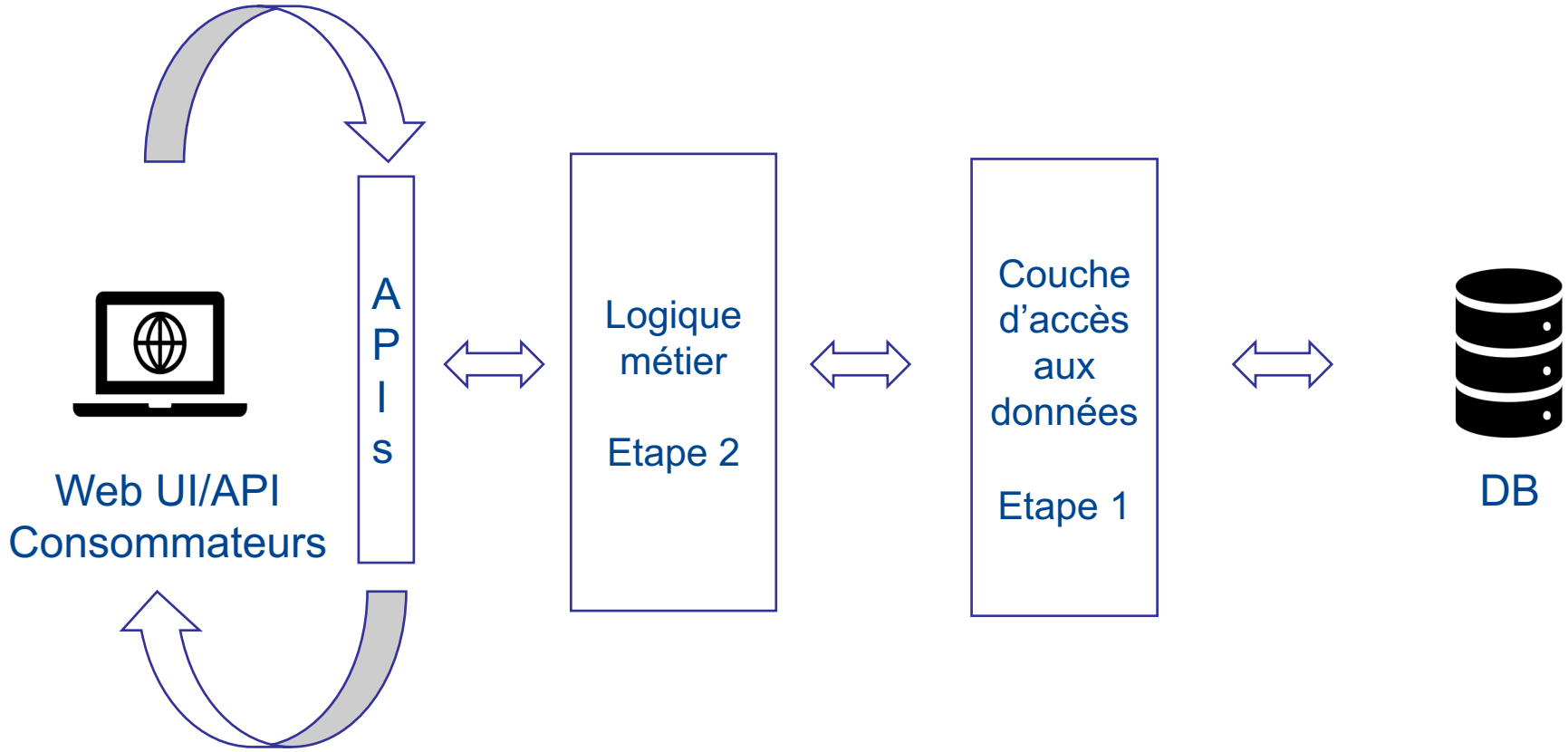
API



API

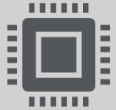


API


A
P
I
s
Web UI/API
Consommateurs
Logique
métier
Etape 2
Couche
d'accès
aux
données
Etape 1
DB

X-Elevate – Développez rapidement un RPG moderne

Architecture multiniveaux d'API



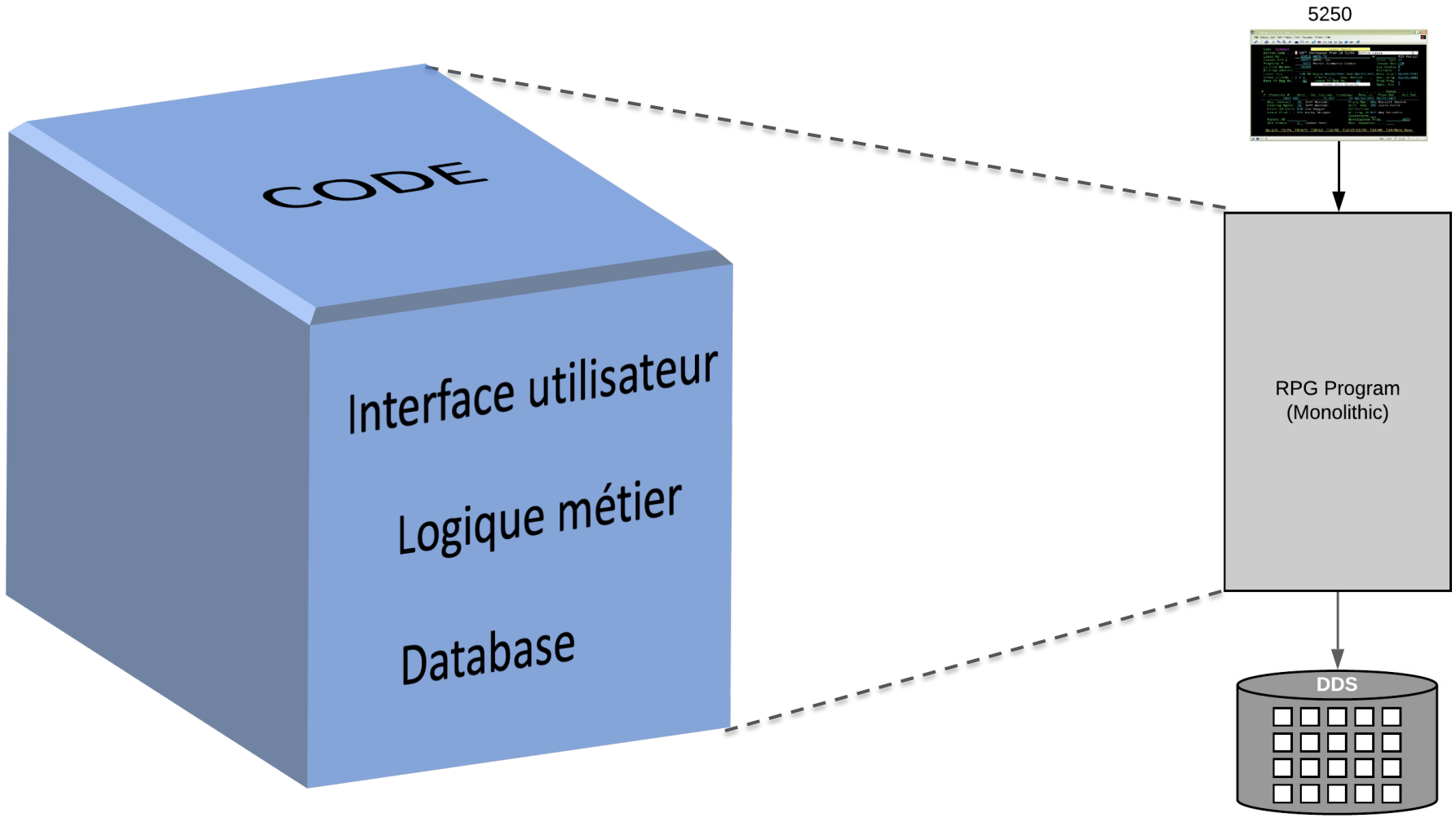
Template-based – Accélère le développement

Améliore l'agilité de l'entreprise

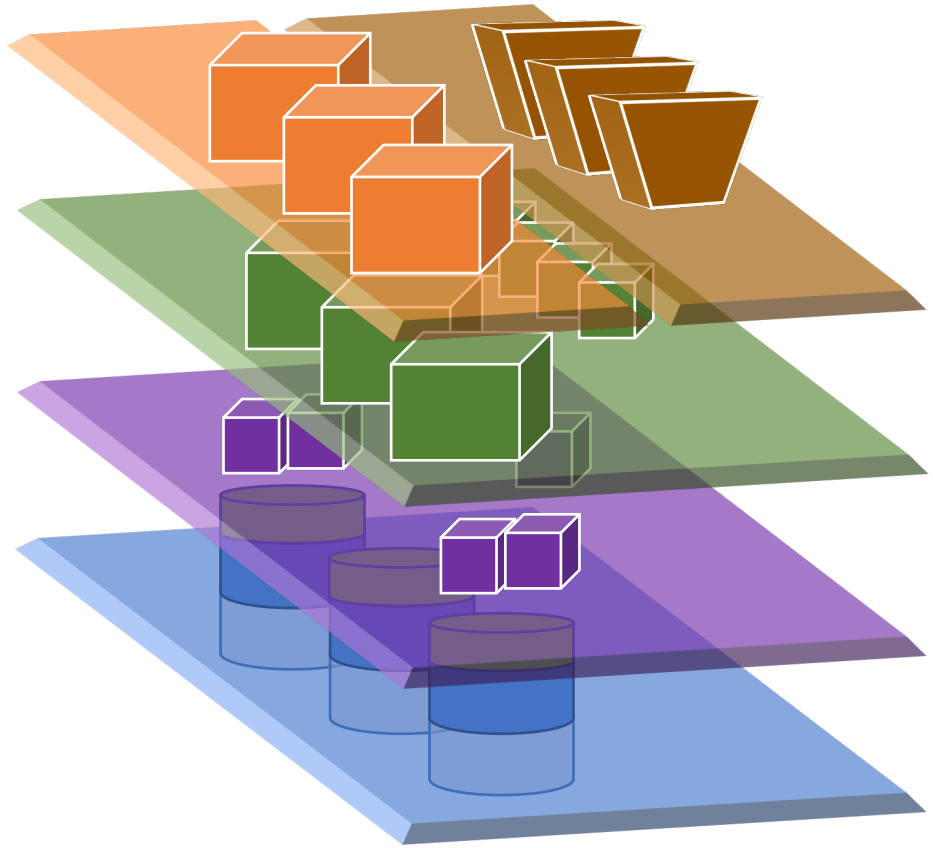
Construire du nouveau ou refactoriser

FRESCHE SOLUTIONS

Monolithe – IBM i “vieille” architecture



X-Elevate - Architecture de référence IBM i



Web Services (RESTful APIs)

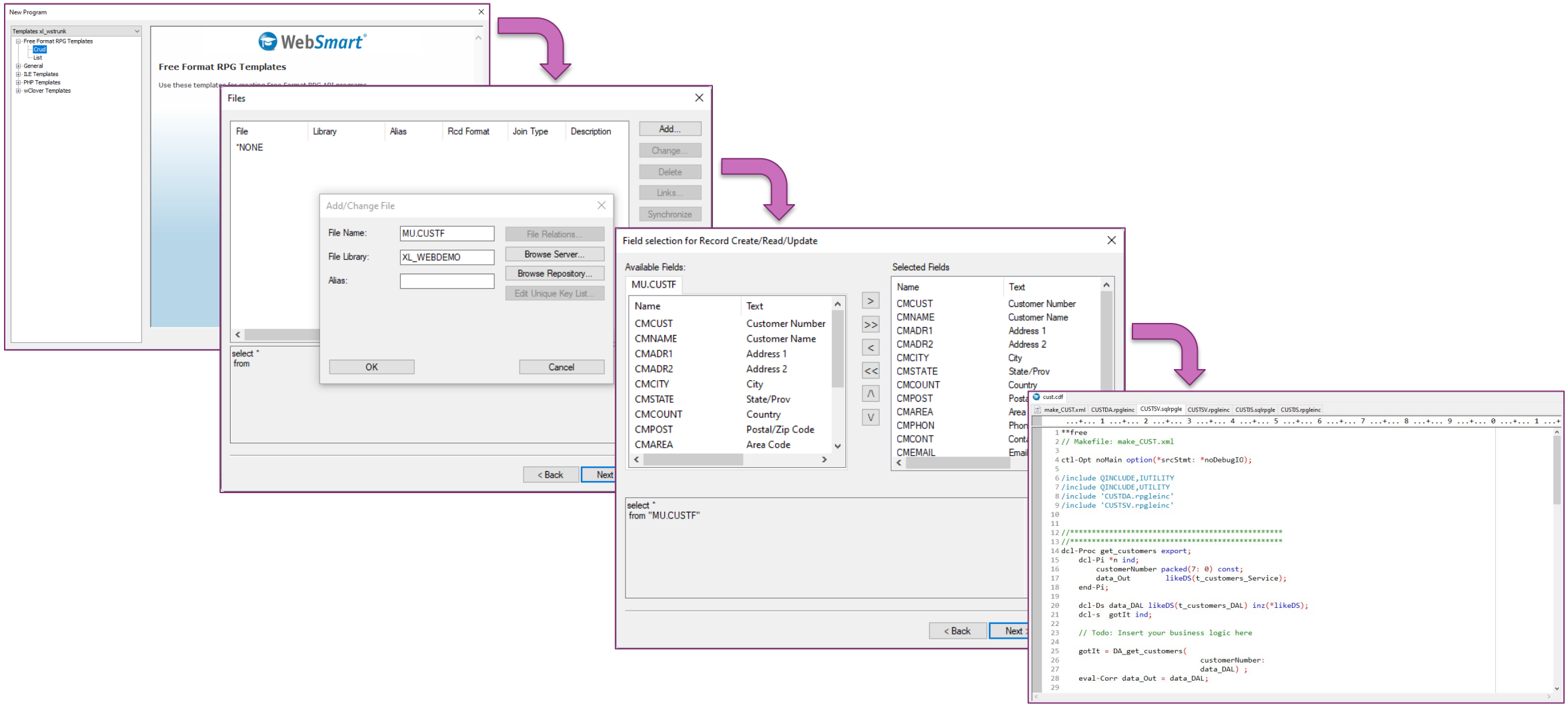
Interface utilisateur (ouverte au client)

Services de processus métier

Database Services

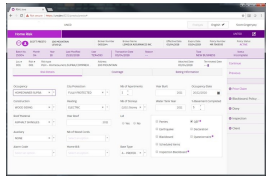
Tables, Index, Vues, etc.

Construction facile basée sur un assistant

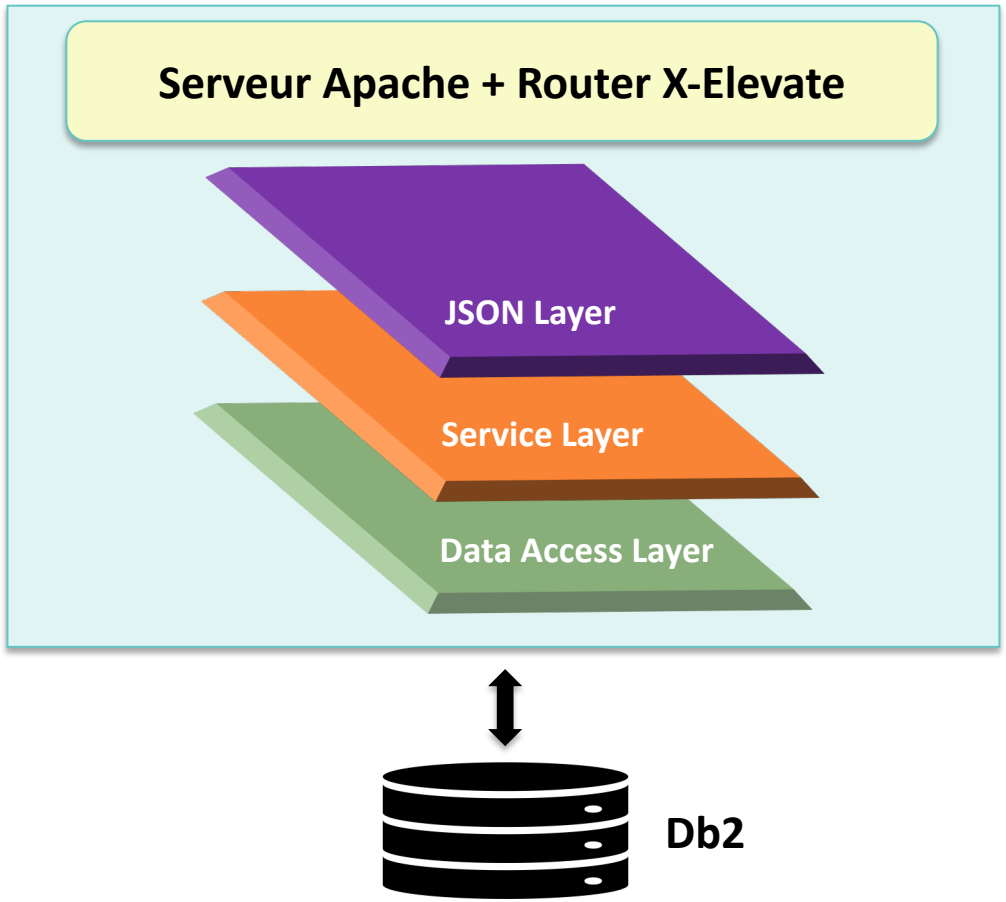


FRESCHE SOLUTIONS

Architecture modulaire à 3 niveaux



Niveau de présentation
Angular, Vue.js

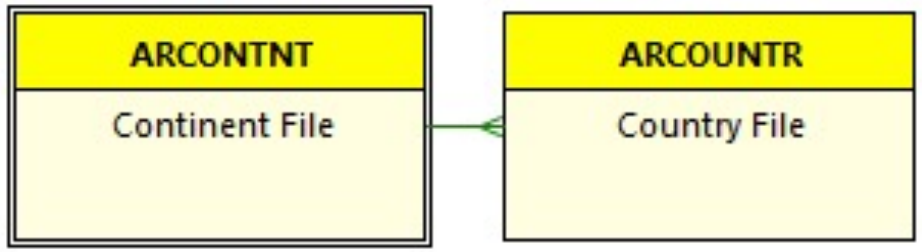


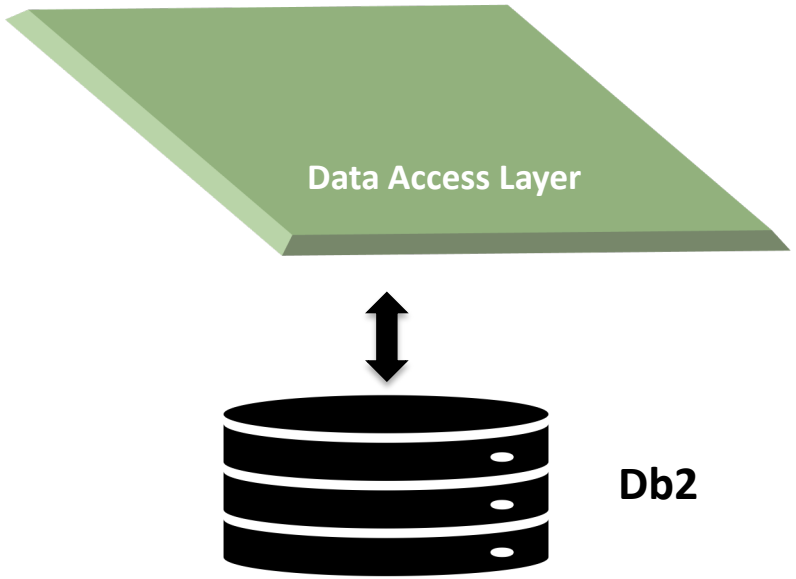
Niveau d'application
X-Elevate

- Applications web
- Code modulaire
- Architecture N-tier
- Capacité Agile et DevOps

FRESCHE SOLUTIONS

Abstraction des bases de données





Services de base de données

Tables, Index, Vues, etc.

Base de données
Continent et pays
(code DDL)

```
create table CONTINENT_ARCONTNT for system name "ARCONTNT" (
       CONTINENTID for column "ADCNTNID" char(2) ccsid 37 not null default,
       CONTINENTNAME for column "ADCNTNAM" char(13) ccsid 37 not null default,
       CHANGEDTIMESTAMP for column "CHANGEDTMZ" timestamp default current_timestamp,
       primary key (CONTINENTID))
  rcdfmt ARCONTNTR;
```

```
create table COUNTRY_ARCOUNTR for system name "ARCOUNTR" (
       COUNTRYID for column "ACCNTID" char(3) ccsid 37 not null default,
       COUNTRYNAME for column "ACCNTNAM" char(40) ccsid 37 not null default,
       PRIMARYLANGUANGE for column "ACCNTPLN" char(3) ccsid 37 not null default,
       SECONDARYLANGUANGE for column "ACCNTSLN" char(3) ccsid 37 not null default,
       COUNTRY2LETTERID for column "ACCNTID2" char(2) ccsid 37 not null default,
       COUNTRYNUMERICID for column "ACCNTNUM" decimal(3, 0) not null default,
       CONTINENTID for column "ACCNTNID" char(2) ccsid 37 not null default,
       CHANGEDTIMESTAMP for column "CHANGEDTMZ" timestamp default current_timestamp,
       primary key (COUNTRYID))
  rcdfmt ARCOUNTRR;
```

```
--Foreing Keys
alter table COUNTRY_ARCOUNTR
   add constraint COUNTRY_ARCOUNTR_CONTINENTID_FK foreign key (CONTINENTID)
     references CONTINENT_ARCONTNT (CONTINENTID);
```

Niveau d'accès aux données - Data Access Layer (DAL)

DA.sqlrpgle (includes DA.rpgleinc)

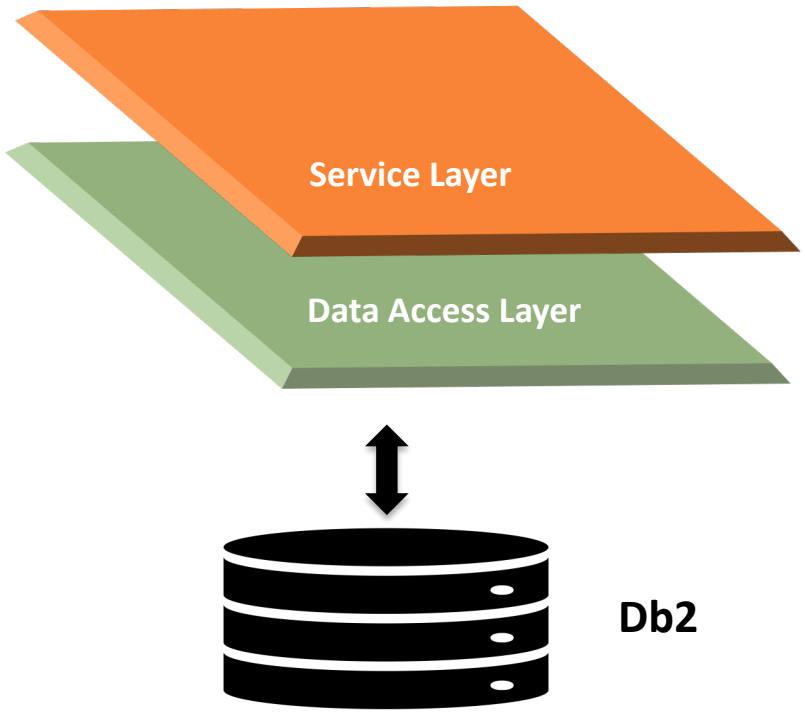
```
dcl-proc DA_add_countryCrud export;
    dcl-Pi *n;
        data_In    likeDS(t_countryCrud_DAL);
        commitInd  ind const;
    end-Pi;
    dcl-ds data likeds(w_countryCrud_DAL);
    // Copy data to table structure
    eval-Corr data = data_In;

    monitor;  // Insert row
        exec SQL
            insert into COUNTRY_ARCOUNTR
            (   COUNTRYID,
                COUNTRYNAME,
                PRIMARYLANGUANGE,
                SECONDARYLANGUANGE,
                COUNTRY2LETTERID,
                COUNTRYNUMERICID,
                CONTINENTID)
            values(:data)
            WITH NC;
        i_check_SQLState();
        if commitInd;
            exec SQL commit;
            i_check_SQLState();
        endif;
    on-Error;
        if commitInd;
            exec SQL rollBack;
        endif;
    endMon;
end-Proc;
```

DA.rpgleinc

```
dcl-DS t_countryCrud_DAL qualified template inz;
    countryid char(3);
    countryname        char(40);
    primarylanguange   char(3);
    secondarylanguange char(3);
    country2letterid   char(2);
    countrynumericid   packed(3: 0);
    continentid        char(2);
    changedTimeStamp  timestamp;
end-Ds;
```

```
dcl-Pr DA_add_countryCrud extProc(*dclCase);
    data_In    likeDS(t_countryCrud_DAL);
    commitInd  ind const;
end-Pr;
```

Abstraction des processus métier



Services de processus métier

Services de base de données

Tables, Index, Vues, etc.

Couche de règles métier

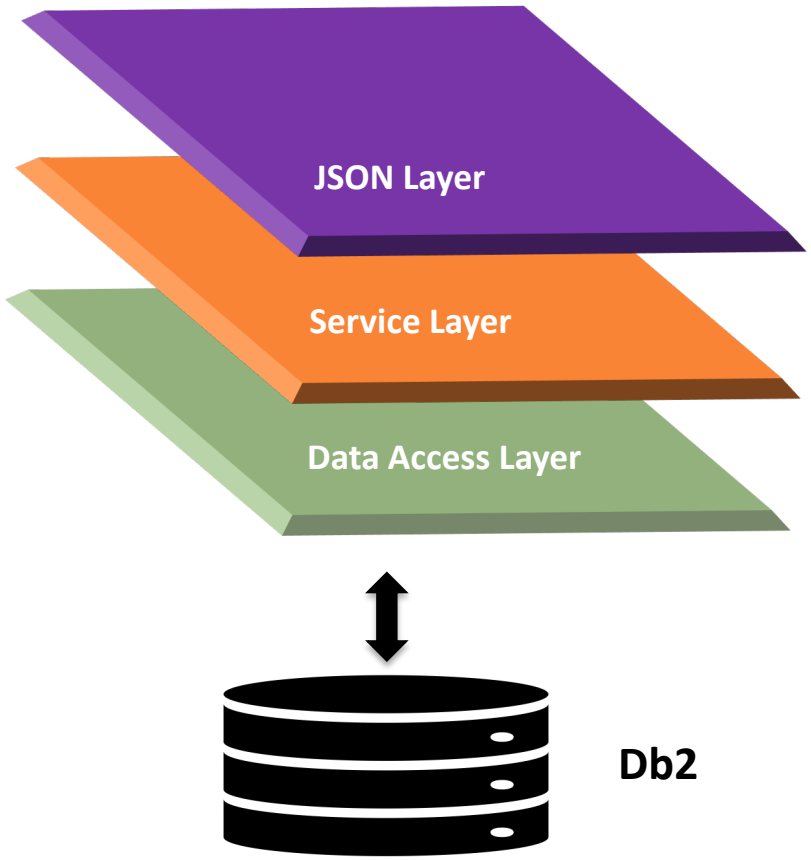
SV.sqlrple (includes DA&SV rpgleinc)

```
dcl-proc add_countryCrud export;
    dcl-Pi *n;
        data likeDS(t_countryCrud_Service);
    end-Pi;
    dcl-Ds data_DAL likeDS(t_countryCrud_DAL) inz(*likeDS);

    // Todo: Insert your business logic here

    validate_countryCrud(data);
    if (uget_messageCount() = *zero);
        eval-Corr data_DAL = data;
        DA_add_countryCrud(data_DAL: *on);
    endIF;
    return;
end-Proc;
```

SV.rpgleinc

```
dcl-DS t_countryCrud_Service qualified template;
    countryid char(3);
    countryname         char(40);
    primarylanguange    char(3);
    secondarylanguange  char(3);
    country2letterid    char(2);
    countrynumericid    packed(3: 0);
    continentid         char(2);
    changedTimeStamp  timestamp;
end-Ds;


dcl-Pr get_countryCrud ind extProc(*dclCase);
    countryid char(3) const;
    data_Out        likeDS(t_countryCrud_Service);
end-Pr;


dcl-Pr add_countryCrud extProc(*dclCase);
    data_In likeDS(t_countryCrud_Service);
end-Pr;
```

Niveaux de services Web



Web Services (RESTful APIs)

Services de processus métier

Database Services

Tables, Index, Vues, etc.

Niveau JSON

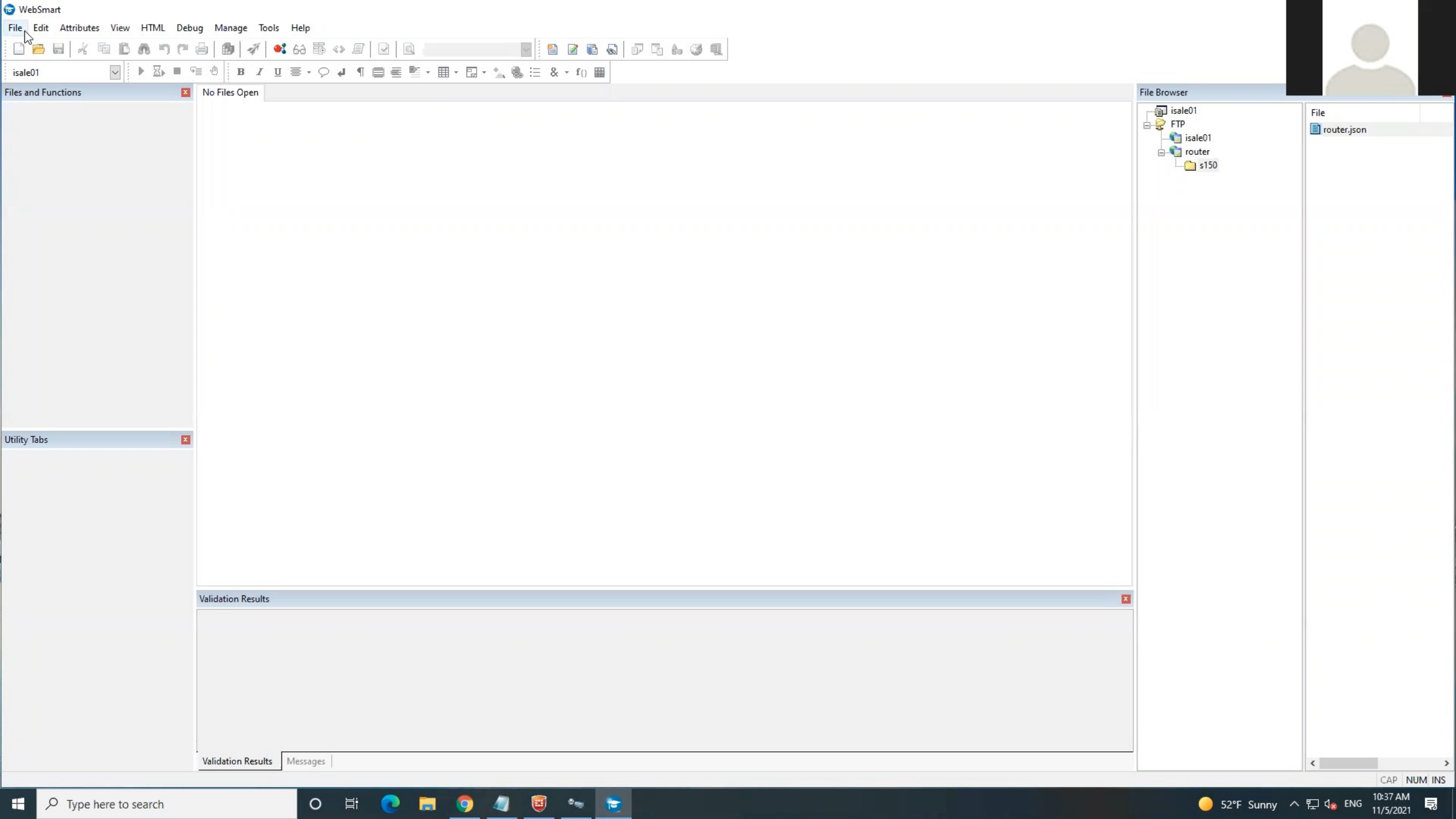
JS.sqlrpgle (includes SV and JS rpgleinc)

```
dcl-Proc json_add_countryCrud export;
    dcl-Pi *n;
        JSONIn   varChar(16000000);
        JSONOut  varChar(16000000);
    end-Pi;
    dcl-ds data likeDS(t_countryCrud_Service);
    // Parse JSON input to DS
    data = parse_JSON(JSONIn);
    // Any errors -> return
    // Call ADD procedure
    add_countryCrud(data);
    // Any errors -> return
    if uget_messageCount() <> *zero;
        clear JSONOut;
        return;
    endif;
    // Fetch the updated table data to return
    get_countryCrud(data.countryid:
                    data);
    // Any errors -> return
    if uget_messageCount() <> *zero;
        clear JSONOut;
        return;
    endif;
    // Set JSON for Output
    set_JSON(data);
    JSONOut = %subst(json_data: 1: json_len);
end-Proc;
```

JS.rpgleinc

```
dcl-Pr json_add_countryCrud extproc(*dclCase);
    JSONIn   varChar(16000000);
    JSONOut  varChar(16000000);
end-Pr;
```

Démonstration


WebSmart
File Edit Attributes View HTML Debug Manage Tools Help
isale01
Files and Functions
No Files Open
File Browser
isale01
FTP
isale01
router
s150
File
router.json
Utility Tabs
Validation Results
Validation Results
Messages
CAP NUM INS
Type here to search
52°F Sunny
ENG
10:37 AM
11/5/2021

Code généré - Exemples

```
**free
ctl-Opt noMain option(*srcStmt: *noDebugIO);

/define U_GLOBAL_USER

/include UTILITYP,UTILITY
/include IUTILITYP,IUTILITY

dcl-DS w_Supplier_DAL qualified template;
   vmvend                    zoned(7: 0);
   supplierName              char(26);
   supplierPhoneNumber       zoned(11: 0);
   address1                  char(26);
   address2                  char(26);
   city                      char(15);
   state                     char(2);
   email                     char(50);
   country                   char(10);
   postalCode                char(10);
   supplierApprovalStatus    char(15);
   manufacturingStatus       char(1);
   useridUpdated             char(10);
   supplierApprovalTimestamp timeStamp;
   supplierApprovalUser      char(10);
end-Ds;
```

```
dcl-proc add_Supplier export;
   dcl-PI *n;
      data likeDS(t_Supplier_Service);
   end-Pi;

   dcl-Ds data_DAL likeDS(t_Supplier_DAL) inz(*likeDS);

   validate_Supplier(data);
   if (uget_messageCount() = *zero);
      eval-Corr data_DAL = data;
      DA_add_Supplier(data_DAL);
   endIF;

   return;
end-Proc;
```

- Fully free-form RPG
- Data centric model
- Techniques modernes
- Services based

Code généré – Exemples

Error Checking

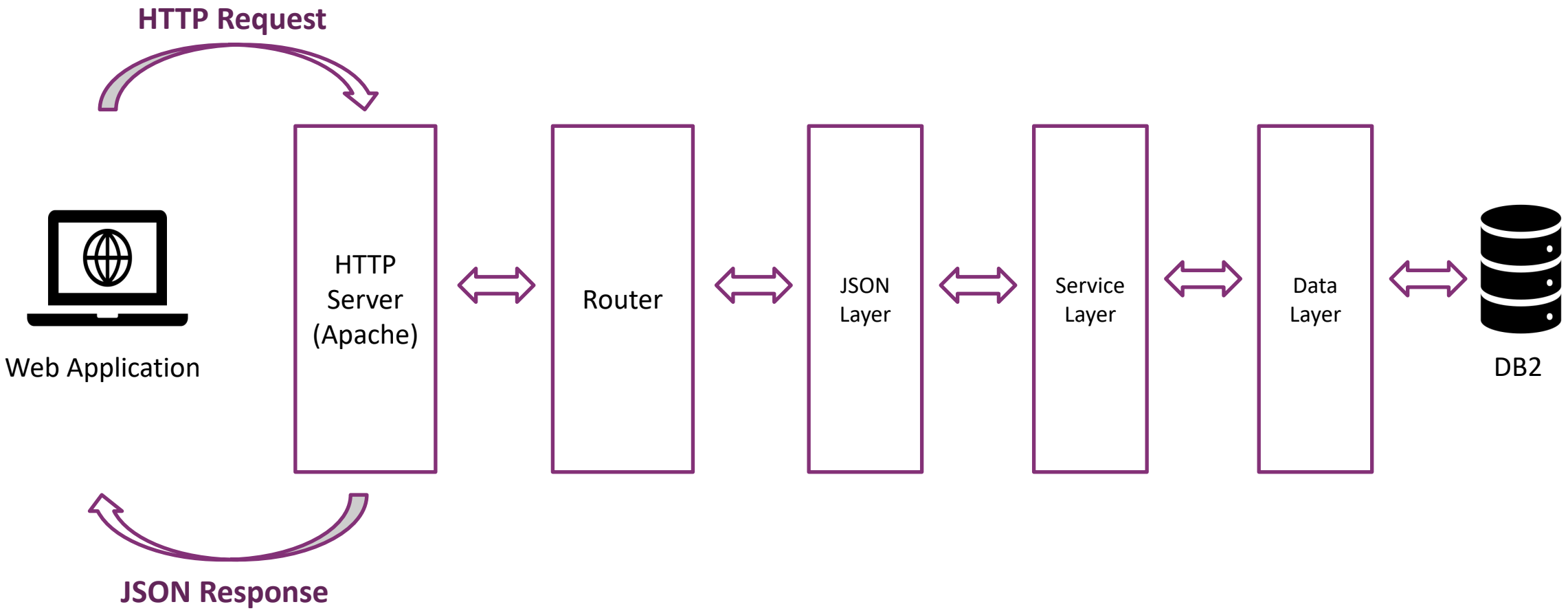
```
dcl-proc DA_add_countryCrud export;
    dcl-Pi *n;
        data_In    likeDS(t_countryCrud_DAL);
        commitInd  ind const;
    end-Pi;
    dcl-ds data likeds(w_countryCrud_DAL);
    // Copy data to table structure
    eval-Corr data = data_In;

    monitor;  // Insert row
        exec SQL
            insert into COUNTRY_ARCOUNTR
            (   COUNTRYID,
                COUNTRYNAME,
                PRIMARYLANGUANGE,
                SECONDARYLANGUANGE,
                COUNTRY2LETTERID,
                COUNTRYNUMERICID,
                CONTINENTID)
            values(:data)
            WITH NC;
        i_check_SQLState();
        if commitInd;
            exec SQL commit;
            i_check_SQLState();
        endif;
    on-Error;
        if commitInd;
            exec SQL rollBack;
        endif;
    endMon;
end-Proc;
```

Commitment Control

Meilleures pratiques de développement et d'architecture modernes

- Architecture API multiniveaux, modulaire et RESTful
  - Séparation des bloques
  - Code réutilisable
  - Code évolutif (scalability)

API X-Elevate - Flux de données



FRESCHE SOLUTIONS

X-Elevate – ces valeurs

Architecture multi-niveaux

Accélère la modernisation des applications RPG

Augmente Agilité

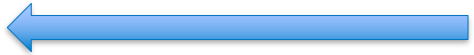
Adoption facile

Exposition de données via des API RESTful

ROI de X-Elevate

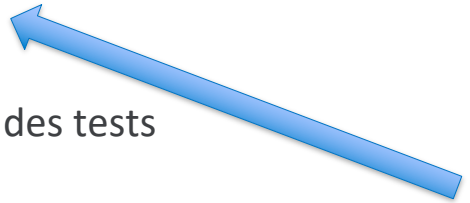
Le fondement des économies de temps et de coûts :

- Création plus rapide de programmes
- Qualité et cohérence du code
  - Réduction de l'effort de maintenance
  - Plus facile à déboguer
  - Des corrections et des modifications plus rapides
  - Moins d'erreurs humaines
- Architecture stratifiée et modulaire
  - Plus rapide à modifier / améliorer
  - Prêt pour DevOps et l'automatisation des tests
  - Prêt pour les microservices
  - Similitude architecturale avec d'autres langues
- Faciliter l'intégration des jeunes développeurs



Combien de temps faut-il pour créer, déboguer et tester un programme ?
(petit ?, moyen ?, grand ?)
(Combien de lignes de code par jour un développeur peut-il créer ?)

Combien de défauts dans 1 000 lignes de code ?
(Et combien de temps pour les déboguer, y remédier et les tester à nouveau ?)



Combien de temps vous faudrait-il pour créer une architecture n-tiers conforme aux meilleures pratiques ?
(Votre équipe a-t-elle l'expérience de l'architecture ?)

ROI de X-Elevate

PARAMÈTRES

Moyenne. Salaire	$140,000
Jours ouvrables par an	234

Économies après 100 programmes : $315K

Temps de conception, de prototypage et d'affinage d'une architecture similaire

Temps / Effort	12 mois
Est. Coût	**$140,000**

Erreurs humaines / Défauts

Moyenne industrielle de 15 à 50 défauts par KLOC (1 000 lignes de code)

Défauts par	25
Temps moyen	15 min
LOC moyenne	1,000

	X-Elevate	Manuellement
Coût de l'élimination des défauts	**$0**	**$46,740**

Temps pour créer de nouveaux programmes (composants et structure)

	X-Elevate	Manuellement	% Distribution
Petit programme	0.5 hrs	8 hrs	40%
Programme moyen	0.5 hrs	16 hrs	40%
Grand programme	0.5 hrs	40 hrs	20%
Nombre de programmes	100		
Effort			
Petit	20	320	
Moyen	20	640	
Grand	10	800	
Heures totales	50 hrs	1760 hrs	
		jour	
Total des jours-hommes	**6.25** jours	**220** s	
Coût basé sur la distribution	**$3,740**	**$131,620**	

FRESCHE SOLUTIONS

MERCI

info@freschesolutions.com
contact@itheis.com