記事
Mihoko Iijima · 2021年6月17日 39m read

【GettingStarted with IRIS】チュートリアルを始めよう！その1：Full Stack チュートリアル

開発者のみなさん、こんにちは！

2023/2/21追記

チュートリアルページが新しくなり「Developer Hub」に変わりました！

Full Stackチュートリアルの開始方法や他のチュートリアルについて詳しくは、「InterSystems Developer Hub：クリック1回で開始できるチュートリアル（4種）のご紹介」をご参照ください。

この記事では、GettingStarted ページの無料体験環境（Sandbox）で試せるチュートリアルの中から、「Full Stack Tutorial」の使い方をご紹介します。

> GettingStarted ページでできることについては、こちらの記事でご紹介しています。
>
> 無料体験環境（Sandbox）の開始手続きについては
> 、こちらの記事でご紹介しています。ぜひご参照ください。

この記事では、チュートリアルを GettingStarted ページの Sandbox でご体験いただく流れを記載しています。Full Stack Tutorial のパート1 を開き、ログインいただくと Sandbox へのアクセス情報が表示されますので、Sandbox 用 IDE のリンクなどはお手元の環境でご確認ください。

また、チュートリアルの流れの中で、IRIS への接続情報を Sandbox 用 IDE で修正する内容があります。Full Stack Tutorial の対象ページを開き、お使いの Sandbox の情報をご確認ください。

Full Stack Tutorial 概要

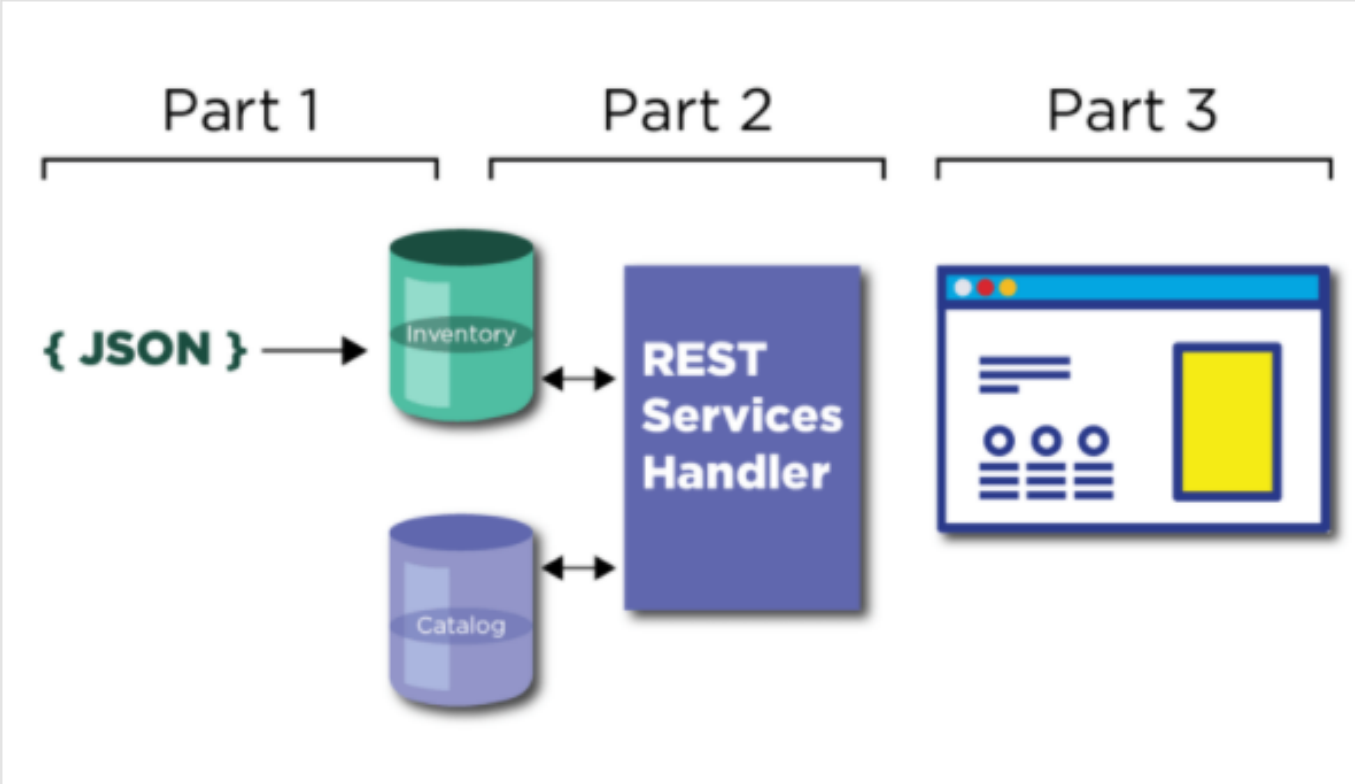
（オリジナルはこちらから　https://gettingstarted.intersystems.com/full-stack/）

InterSystems IRIS data platform は、マルチモデル（SQL、NoSQL）や、マルチワークロード（トランザクションとリアルタイム分析）が行える DBMS 機能、システム統合、変換、API管理、ビジネスロジックを操作できるプラットフォームで、組み込みの統合、分析機能、BI、機械学習、自然言語処理を含む強力で様々な機能を提供しています。これら機能を利用するために、別製品を追加することも、データを別の構成に移動することも不要で、全て 1 つのプラットフォームで操作いただけます。

フルスタックチュートリアルでは、小さな製造会社（焙煎したてのコーヒーを販売する会社）の基本的な情報管理インフラを作成していくテーマを用意しています。

この会社では、焙煎したばかりのおいしいコーヒー豆を焙煎、包装、販売しています。

このチュートリアルを通して、InterSystems IRIS data platform が IT
アーキテクチャのバックボーンとしてどのように機能するかを学習いただけます。



このチュートリアルは、3つのパートに分かれています。

コーヒーメーカーとして、生豆の在庫管理からオンラインポータルでの販売までを設定するためのプロセスをご紹介します。

パート1

架空会社「IRIS コーヒーカンパニー」で使用する、簡単な在庫処理システムを作成します。

IRIS が他の多くのデータベースと同様に、テーブルの作成やデータの読み込みに標準的な SQL
を使用できることをチュートリアルを通して学習します。また、Python を使用して JSON
形式で送付されてくる注文情報を処理します。

パート1の終わりには、新しいコーヒー豆の納品を会社の在庫として処理できるようになります。

パート2

在庫管理、焙煎、販売など、ビジネスのさまざまな処理を RESTful サービスを介して通信できるようにします。

コーヒーの焙煎所は在庫から豆を要求し、焙煎と包装の後、REST
サービスを使って最終製品をカタログに載せ、オンラインで販売します。

これらの処理は全て、チュートリアルで作成する RESTful サービスを利用して実行されます。

パート3

人気のある JavaScript フレームワーク Vue.js

を使用して、焙煎職人が作ったコーヒー豆を販売するオンラインストアを作成します。

---

1) パート1：SQLを使用してデータベースを作成する

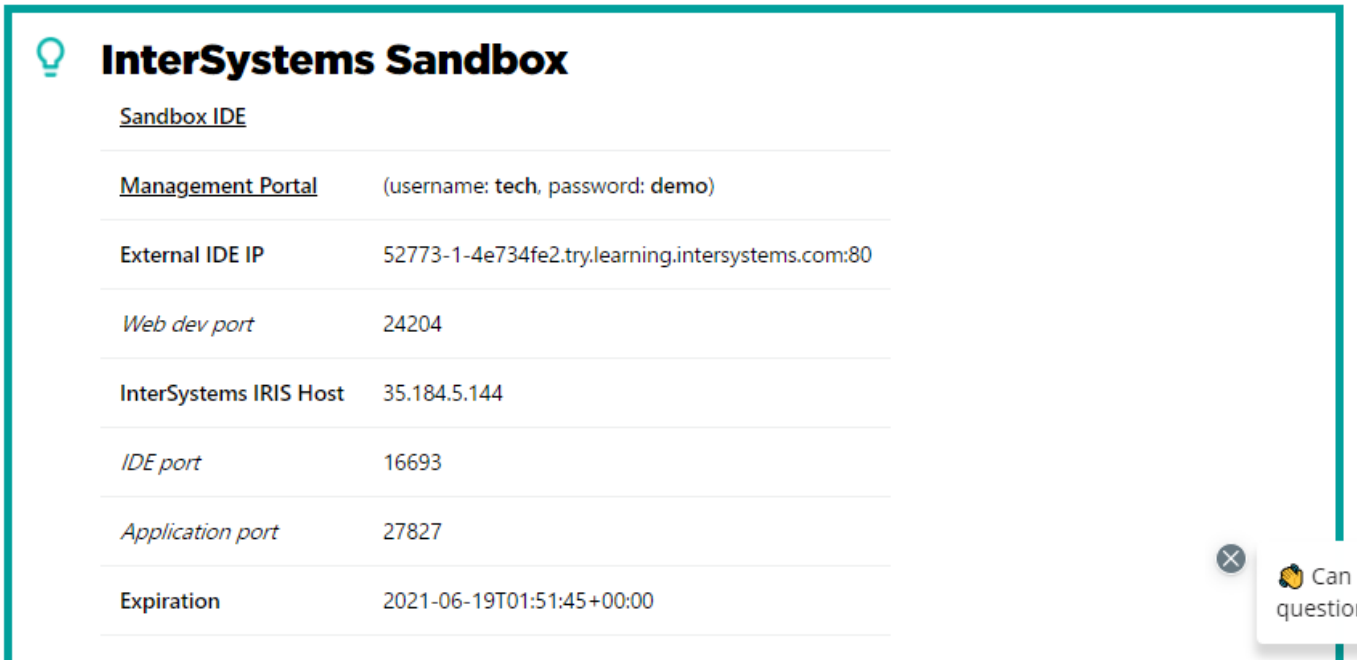
このチュートリアルを完成させるためには、無料体験環境の Sandbox の準備が必要です。

こちらのページにアクセスし　https://gettingstarted.intersystems.com/full-stack/full-stack-part-one/
）まだログインされていない場合は、ログインを行ってください。

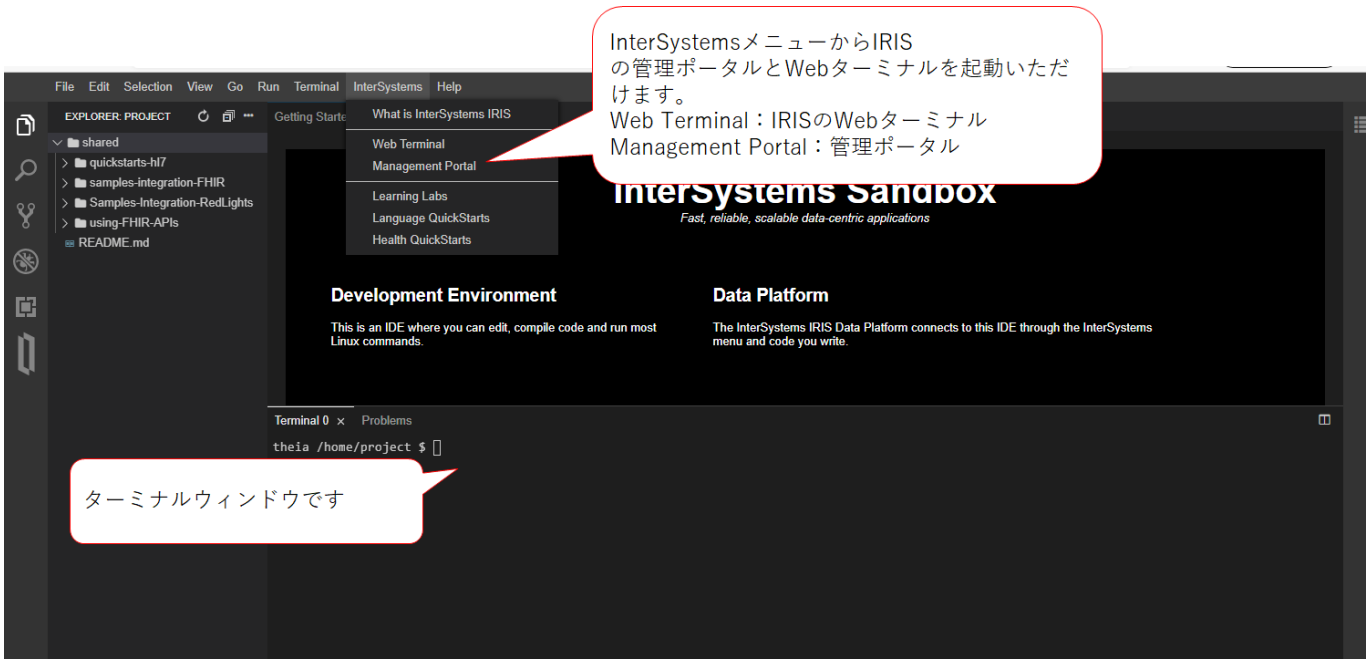
ユーザ登録がまだの場合は、こちらの記事をご覧いただき、ユーザ登録後、ログインを行ってください。

ログイン後、

ボタンをクリックします。既にクリックされている場合は、アクセス情報が以下のように表示されます。



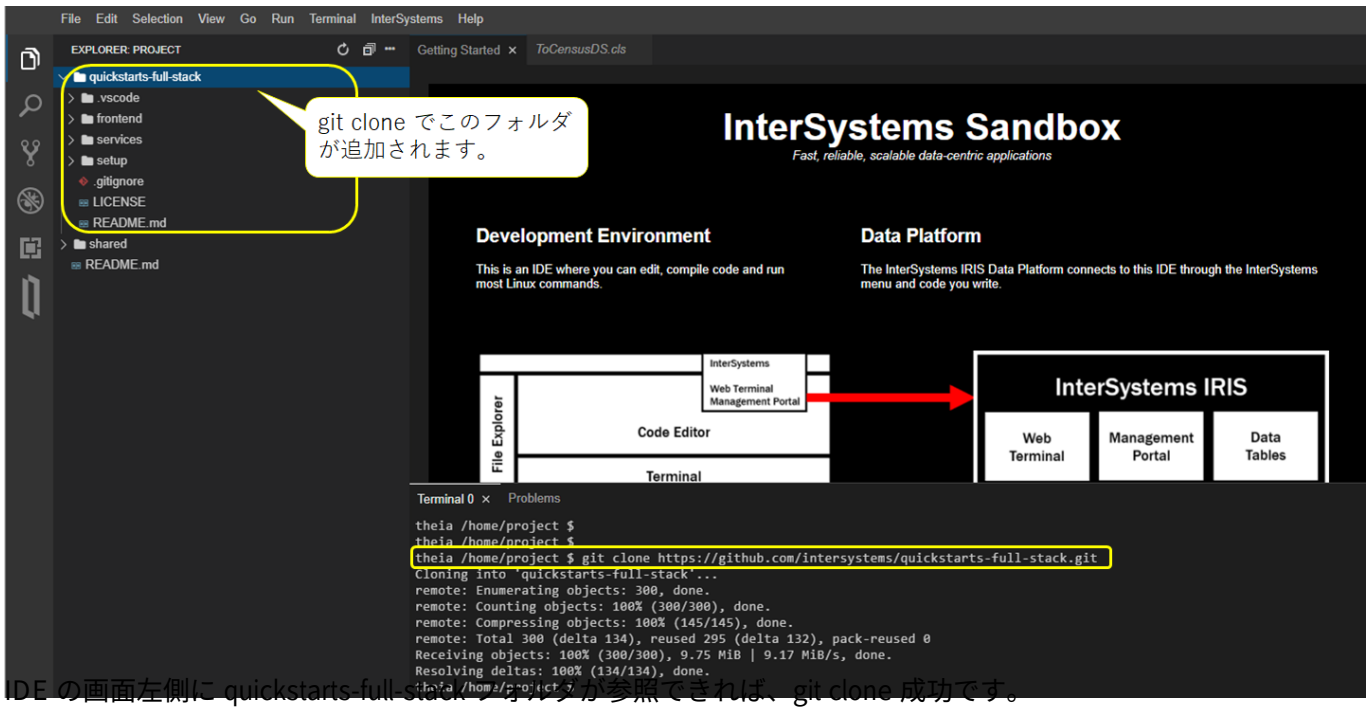
情報表示の一番上に「Sandbox
IDE」と書かれたリンクがあります。こちらをクリックするとブラウザでアクセスできる Sandbox 専用 IDE が開きます（下図）。



チュートリアルは、この IDE を使用します。

準備ができたら、フルスタックチュートリアル用ソースコードを git clone するため、画面中央下のターミナルウィンドウで以下実行します。

```
git clone https://github.com/intersystems/quickstarts-full-stack.git
```

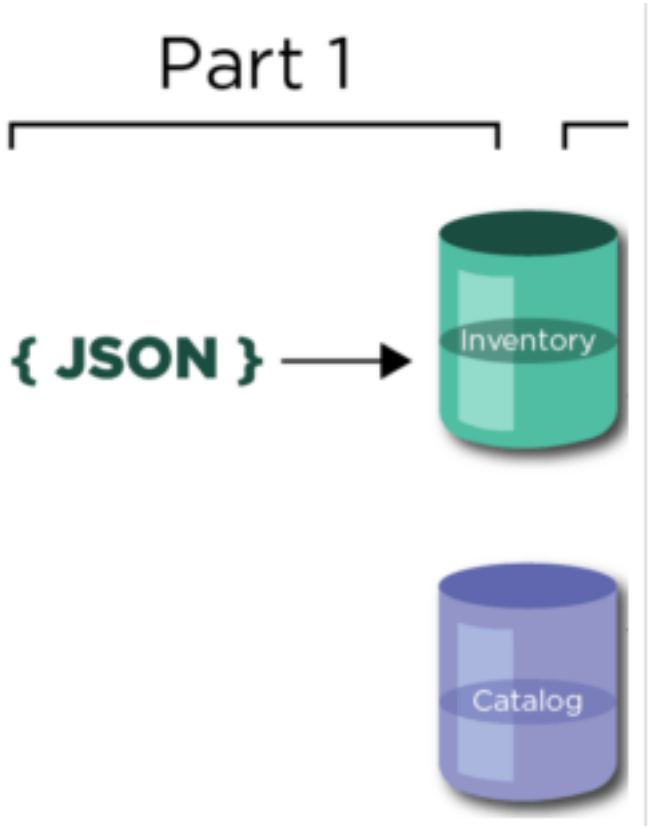


IDE の画面左側に quickstarts-full-stack フォルダが参照できれば、git clone 成功です。

この IDE は Theia で、Visual Studio Code とよく似た機能を持っています。左側にファイルエクスプローラーがあります。右側にはコード編集パネルがあり、編集パネルの下にはターミナル・ウィンドウがあります。

いよいよチュートリアルの開始です！

データベースを作成します。



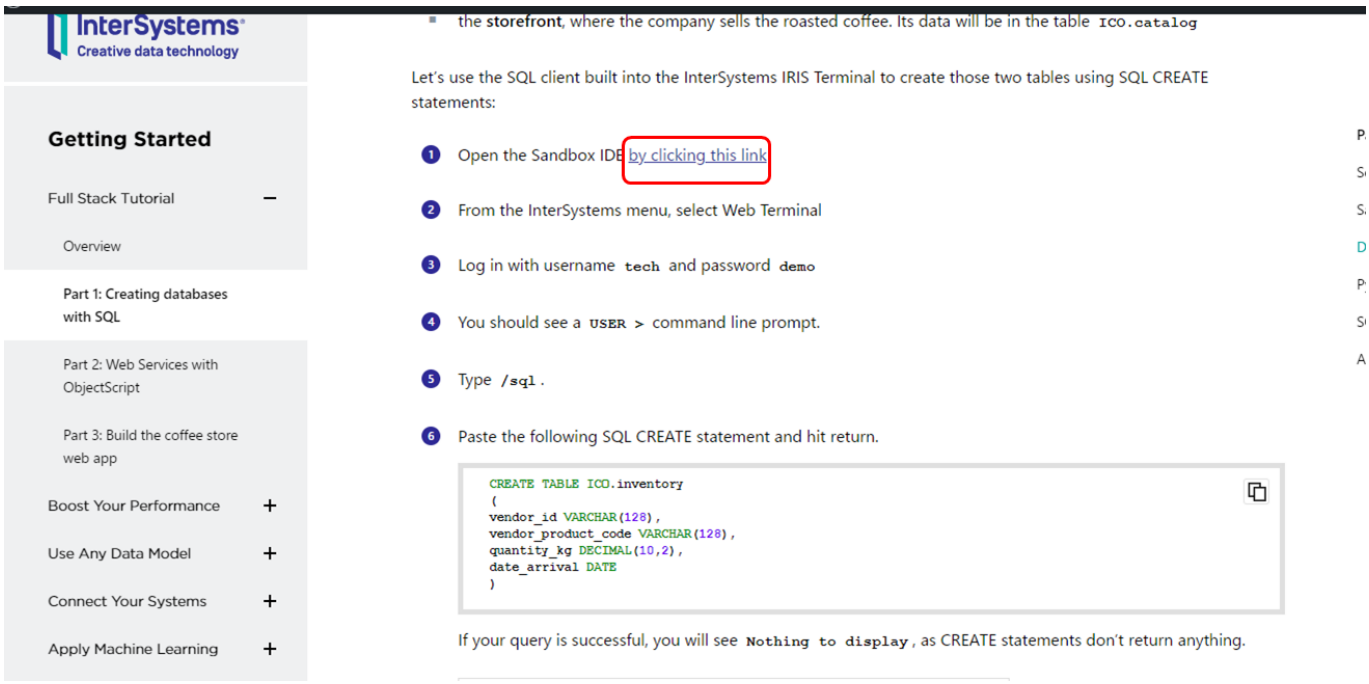
IRIS
コーヒーカンパニーは、3つの部門に分かれています。

- **倉庫**にはコーヒーの生豆の在庫が保管されています。テーブル名：ICO.inventory にデータが格納されます。
- **焙煎所**は、コーヒー豆を焙煎する部門で、データを保存する必要がない部門です。
- **Storefront** は、焙煎したコーヒーを販売する店舗です。販売データはテーブル名：ICO.catalog に格納されます。

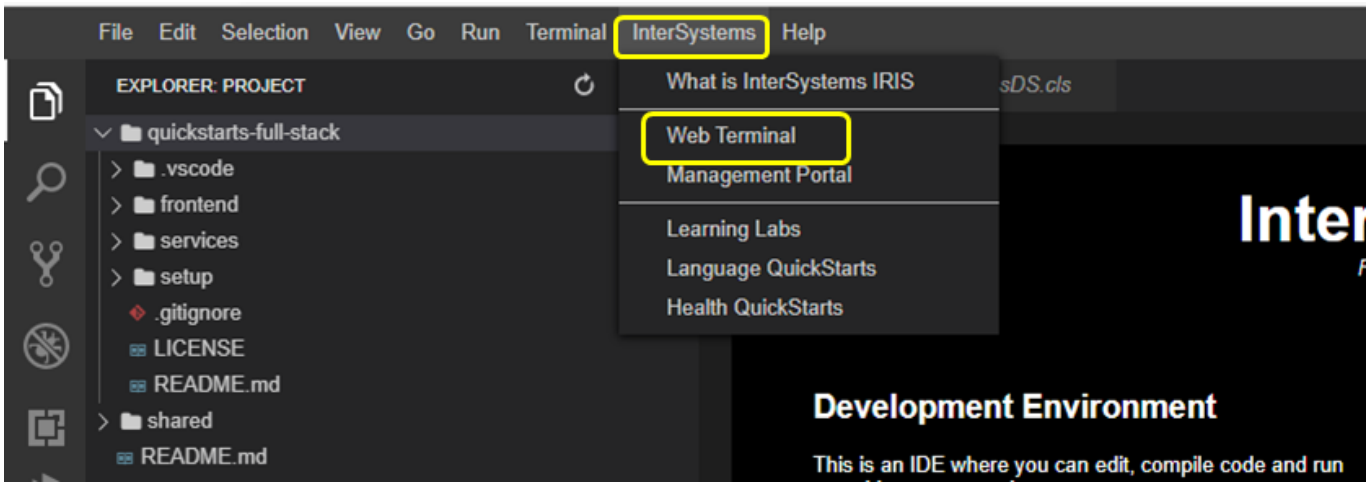
IRIS ターミナルをSQL用モードに切り替え、CREATE文を使って2つのテーブルを作成します。

手順は以下の通りです（IDEからアクセスできます）。

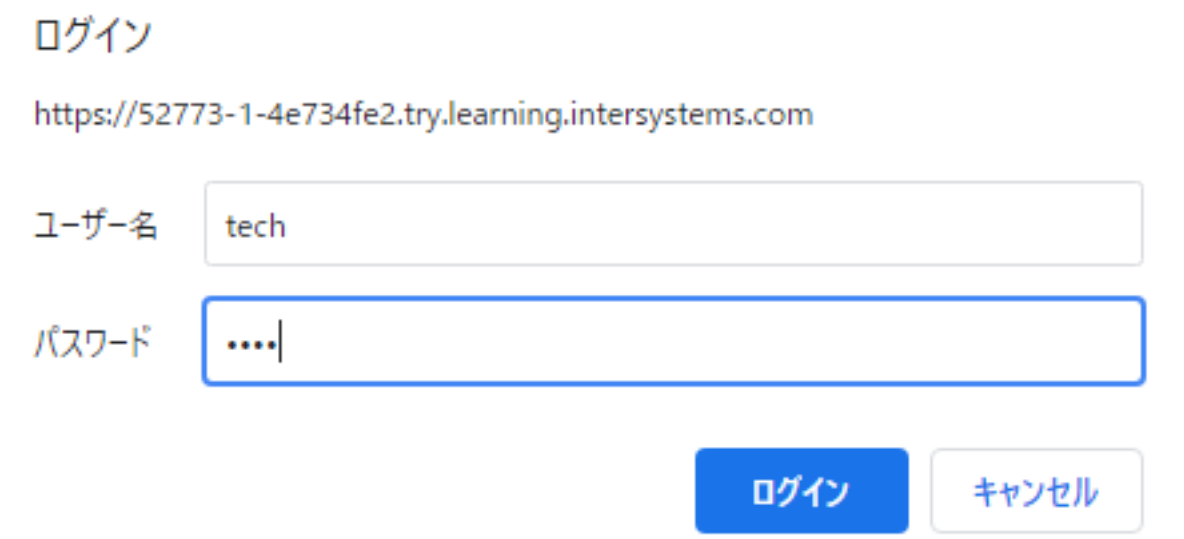
(1) Sandbox の IDE
を開きます（パート1：https://gettingstarted.intersystems.com/full-stack/full-stack-part-one/#database-creation
を開き、ログインしていると以下のように IDE へのリンクが表示されます）。



(2) IDE のメニューから、InterSystems > Web Terminal を選択します。別タブで Web ターミナルが開きます。



(3) ログイン画面が出てきたら、ユーザ名：tech パスワード：demo を入力してログインします。



(4) プロンプトに USER> と表示される画面が開きます。

(5) Web ターミナルのモードを SQL 実行モードに変えるため、**/sql** を入力します。

（Web ターミナル専用コマンドです）。

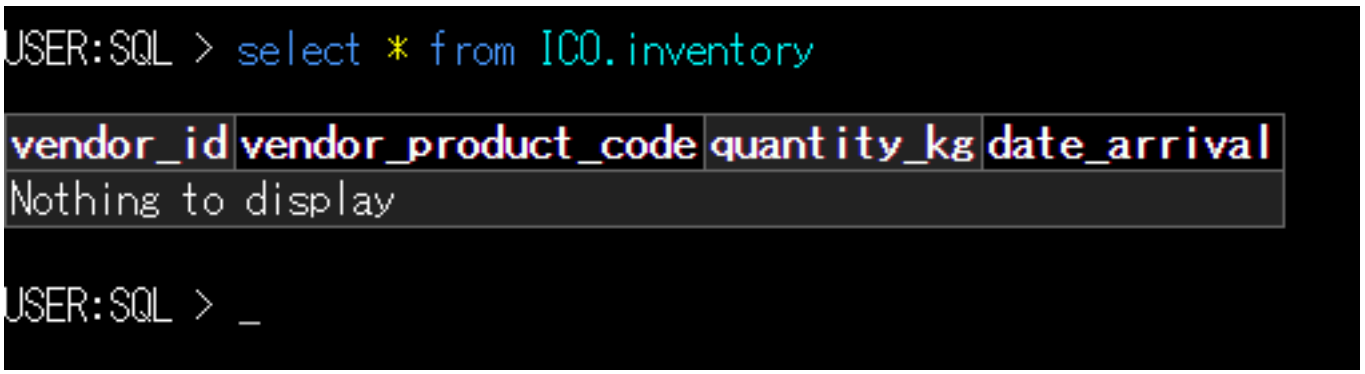
(6) 以下のSQL文をコピーして、Web ターミナルに貼り付けます（Ctrl +v でペーストできます）。

```
CREATE TABLE ICO.inventory
(
vendor_id VARCHAR(128),
vendor_product_code VARCHAR(128),
quantity_kg DECIMAL(10,2),
date_arrival DATE
)
```

プロンプトが戻ってくるまでお待ちください。

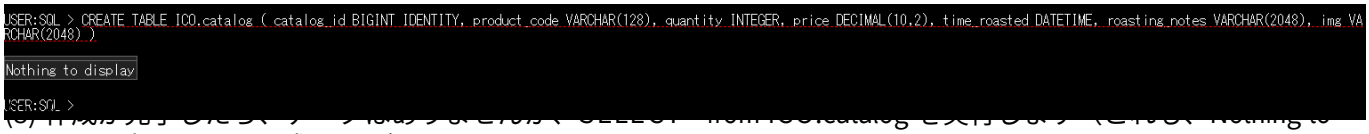
テーブルが作成できたか確認のため、SELECT * from ICO.inventory
を実行してみます（まだレコードがないので、「Nothing to display」と表示されます）。

```
SELECT * from ICO.inventory
```



7) 次に、ICO.catalog テーブルを作成するため、以下 SQL をコピーし、Web ターミナルに貼り付け実行します。

```
CREATE TABLE ICO.catalog
(
catalog_id BIGINT IDENTITY,
product_code VARCHAR(128),
quantity INTEGER,
price DECIMAL(10,2),
time_roasted DATETIME,
roasting_notes VARCHAR(2048),
img VARCHAR(2048)
)
```



(8) 作成が完了したら、テーブルはちゃんとできたかどうか、SELECT * from ICO.catalog を実行しましょう（これも、Nothing to display と表示されれば成功です）。

```
SELECT * from ICO.catalog
```

(9) Web ターミナルで、 **/sql** を実行すると、元のターミナルプロンプトに戻ります（＝IRIS の ObjectScript が実行できるターミナルに戻ります）。

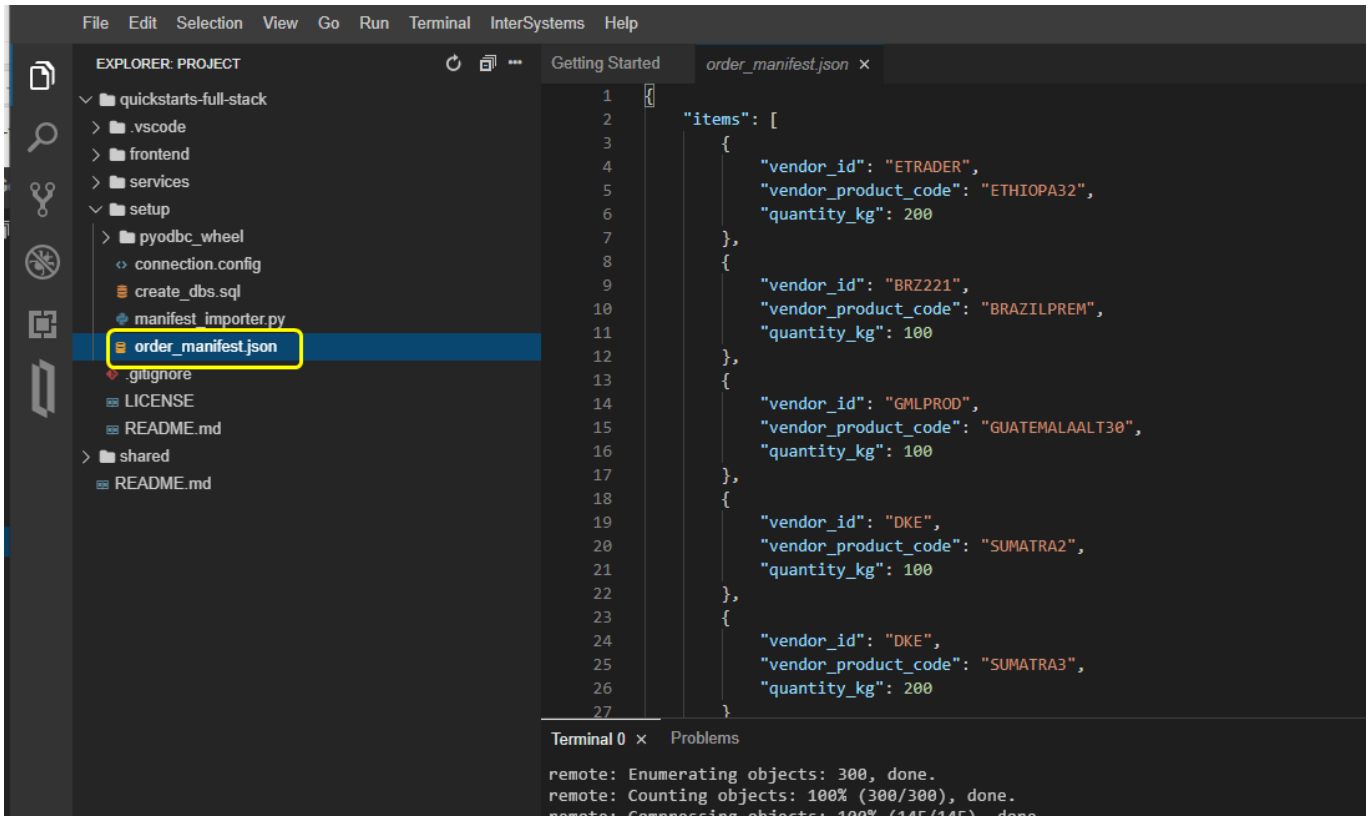
Python を利用してデータをロードします。

チュートリアルでは、Python でデータをロードする処理を作成しています。

大まかな処理の流れは以下の通りです。

世界中のベンダーから生豆の出荷依頼が来ると想定し、生豆の注文をデータベースに入力できるようにします。出荷情報は JSON 形式とし、単一の注文マニフェストファイルで送付される仕様としています。

注文マニフェストの例は、quickstarts-full-stack/setup/ordermanifest.json にあります。IDE の左画面を利用してファイルを開いてみましょう。



データロードには、Python のプログラムを使用します。プログラム内では、注文マニフェストの JSON ファイルを解析し、データベースに接続し INSERT を実行しています（データ登録時に SQL が使用されますが、今回はWeb ターミナルではなく、Python のプログラム内で実行しています）。

注文マニフェストをデータベースにインポートする処理を作成します。

Python の標準ライブラリを使用して、指定ディレクトリから JSON ファイルを読み込みます。その後 ODBC 経由でデータベースに接続し、SQL の INSERT をし湯押して、JSON データを IRIS に登録します。

> 次の Python コードの解説が不要な場合は、サンプルスクリプトの実行に移動してください。

Python コードの中身解説

スクリプトは setup/manifestimporter.py にあります。

以下、データロード用スクリプトで行っている重要な要素について解説します。

main() 関数では、JSON 注文マニフェストファイルをインポートし、検証を行い、inventory テーブルに登録するために必要な構造をチェックしています。

```
def main():
    with open('./order_manifest.json') as f:
    data = json.load(f)
    data, status, exp = validate_manifest(data)
```

次に、connection.config ファイルにあるデータベースの接続情報を読み込んでいます。

```
connection_detail = get_connection_info("connection.config")
ip = connection_detail["ip"]
port = int(connection_detail["port"])
namespace = connection_detail["namespace"]
username = connection_detail["username"]
password = connection_detail["password"]
driver = "{InterSystems ODBC}"
```

ODBCドライバーを使ってデータベースへの接続を設定します。

```
connection_string = 'DRIVER={};SERVER={};PORT={};DATABASE={};UID={};PWD={}' \
.format(driver, ip, port, namespace, username, password)
connection = pyodbc.connect(connection_string)
```

JSON データ（data）とデータベース接続オブジェクト（connection）を使って loadmanifest() 関数を呼び出します。

```
msg = load_manifest(data, connection)
```

loadmanifest() 関数では、JSON ファイル内のすべてのアイテムを繰り返し取得しながら INSERT 文を組み立て、前の手順で作成した ICO.inventory テーブルに各アイテムを挿入します。

方法は以下の通りです。最初に、INSERT 文を作成しています。

```
fieldnamesql = "INSERT INTO ICO.inventory (vendor_id, vendor_product_code, quantity_k
g, date_arrival)"
```

次の2行では、現在の日付を使用して「2021-06-16」の形式の日付文字列を作成します。この日付は、製品が倉庫に到着した日付に使用します。

```
today = date.today()
mydate = today.strftime("%Y-%m-%d")
```

このコードは、JSON ファイル内の各アイテムのオブジェクトをループし、取得したデータを使用して INSERT 文の VALUES の部分を作成しています。

その結果、次のような INSERT 文が完成します。

```
INSERT INTO ICO.inventory (vendor_id, vendor_product_code, quantity_kg, date_arrival)
 VALUES (ETRADER, ETHIOPIA32, 200, ‘2021-06-16’)
```

次に、ここまでの解説で作成した SQL 文（変数 valsql）を実行します。

loadmanifest() 関数の1行目でデータベースカーソルを定義しているので、カーソルに SQL 文を入力して execute() を実行しています。

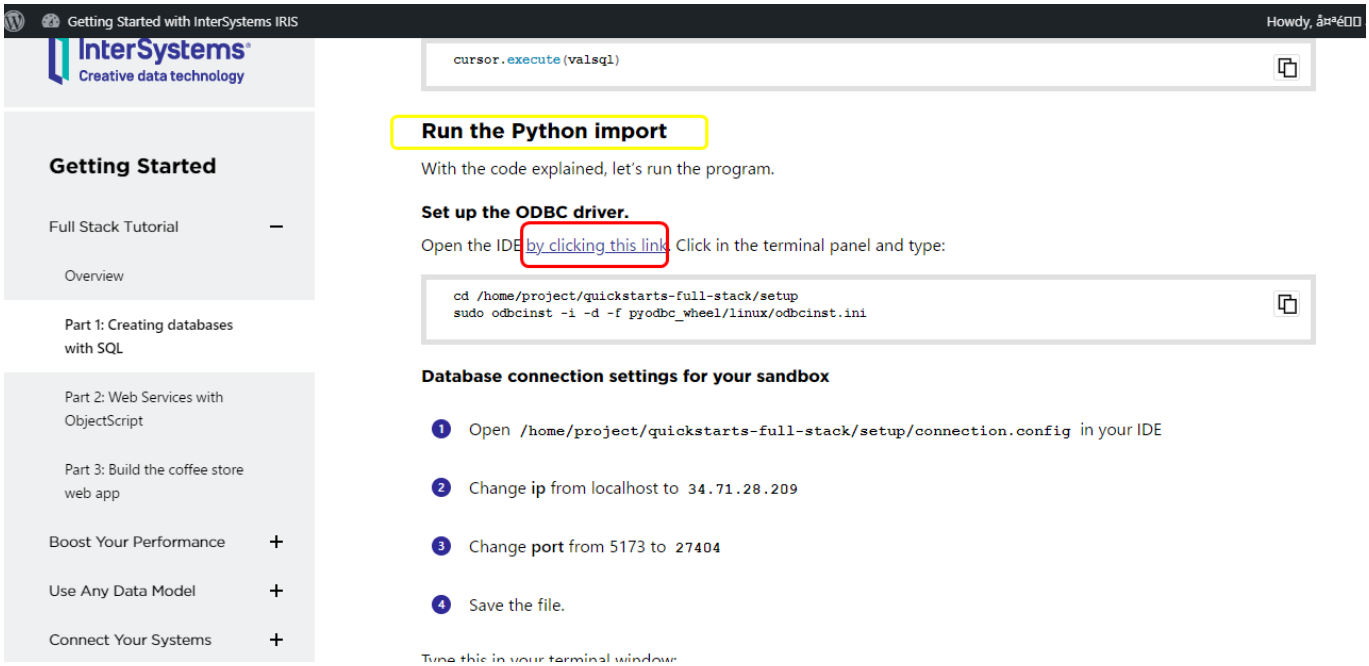
```
cursor.execute(valsql)
```

Pythonのデータローダスクリプトを実行する

コードの解説が終了しましたので、プログラムを実行してみましょう。

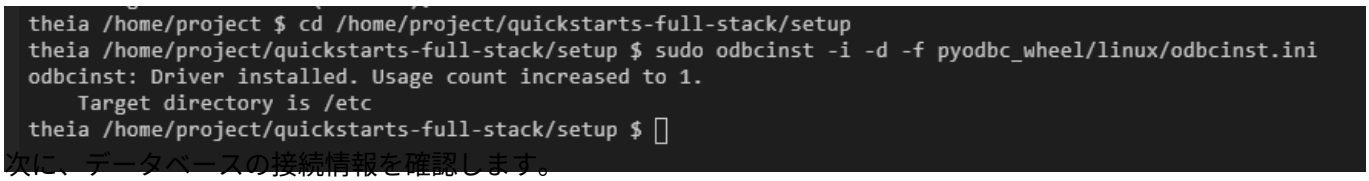
最初に、ODBCドライバのインストールを行います。

Sandbox の IDE を開きます。IDEを開く URL は https://gettingstarted.intersystems.com/full-stack/full-stack-part-one/#jsondataimport を開き、「**Run the Python import**」近くに表示されます。表示されない場合は、ログインを行ってください。



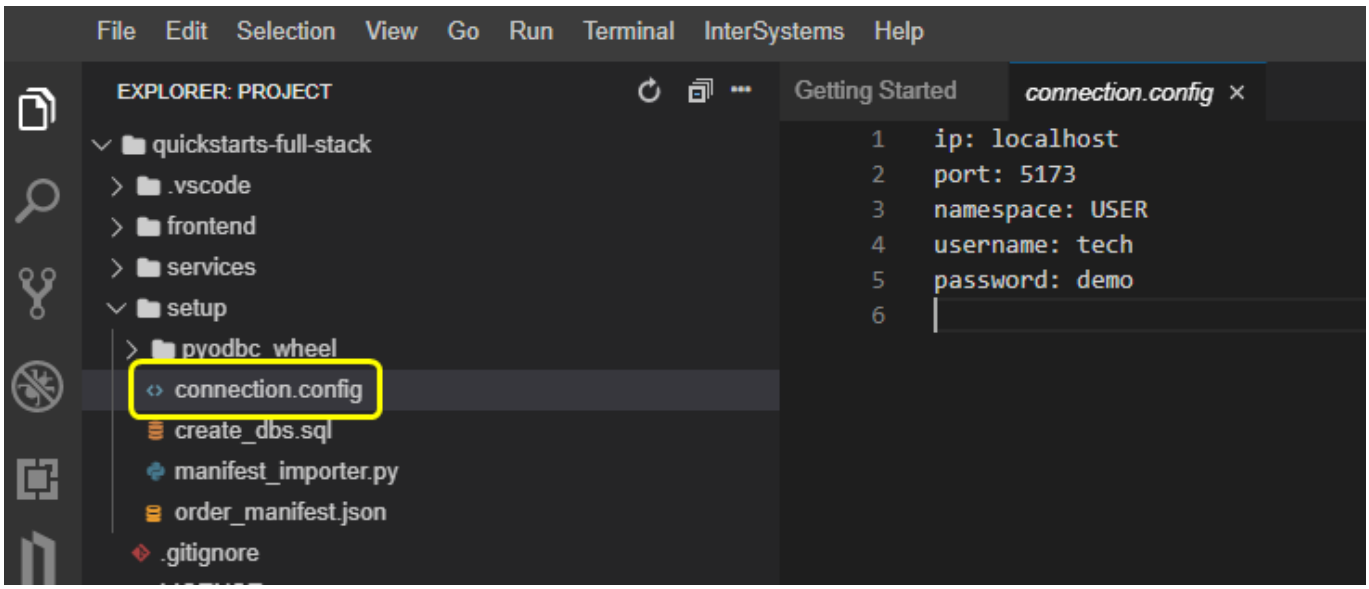
IDE を開いたら、ターミナルウィンドウで、以下実行してください。

```
cd /home/project/quickstarts-full-stack/setup
sudo odbcinst -i -d -f pyodbc_wheel/linux/odbcinst.ini
```



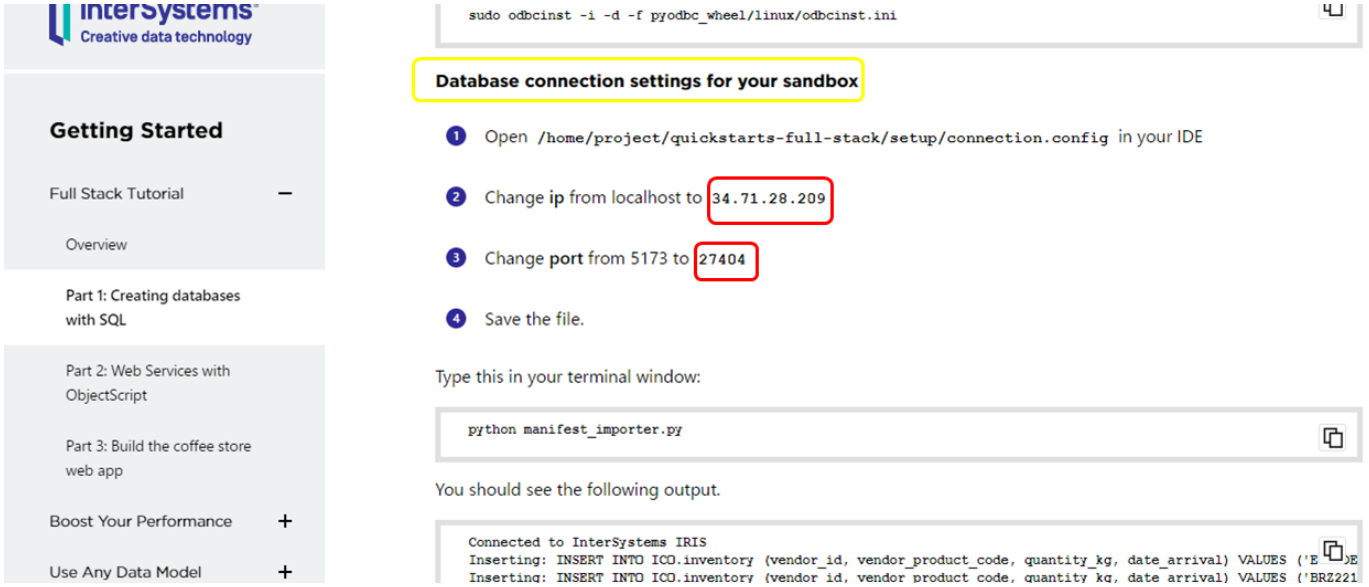
次に、データベースの接続情報を確認します。

(1) Sandbox の IDE で /home/project/quickstarts-full-stack/setup/connection.config を開きます。



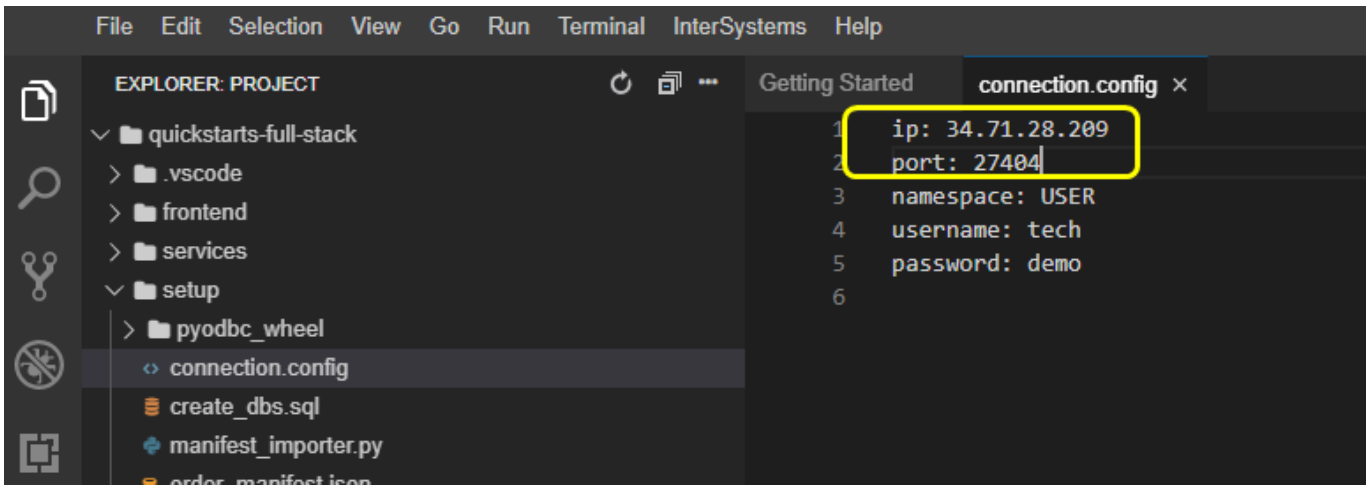
(2) IP アドレスに記載されている文字列を、修正します。

https://gettingstarted.intersystems.com/full-stack/full-stack-part-one/#jsondataimport を開き 「**Database connection settings for your sandbox**」の近くに変更対象の IP アドレスが表示されます。



図例だと、ip の設定に 34.71.28.209 を指定します。

port の設定も、27404 に変更します。



(3) 変更後、ファイルを保存します（Ctrl ＋ s）。

接続情報変更後、Sandbox の IDE のターミナルウィンドウで以下実行します。

```
python manifest_importer.py
```

図例にあるような、INSERT 文の実行ログを確認できると思います。

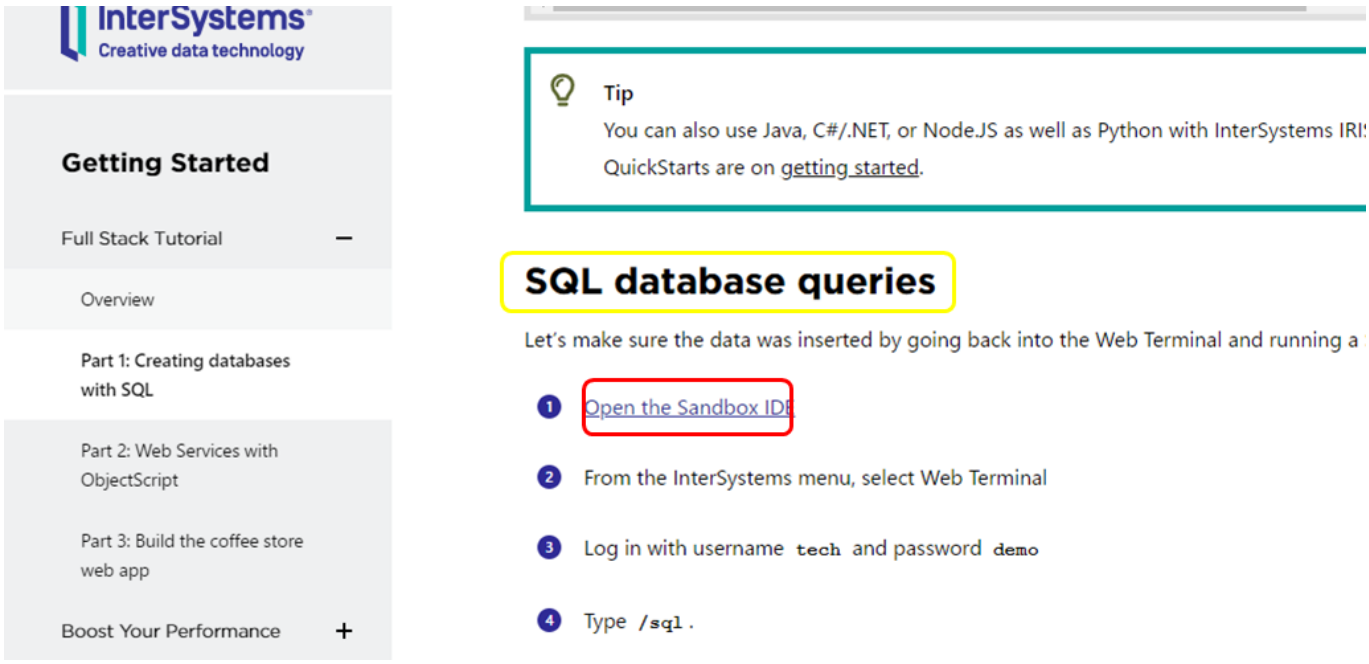
> IRIS は、Python だけでなく、Java、.NET、Node.js
> からもアクセスできます。言語別のアクセス方法の体験については、QuickStart をご参照ください。

SQLでデータを確認

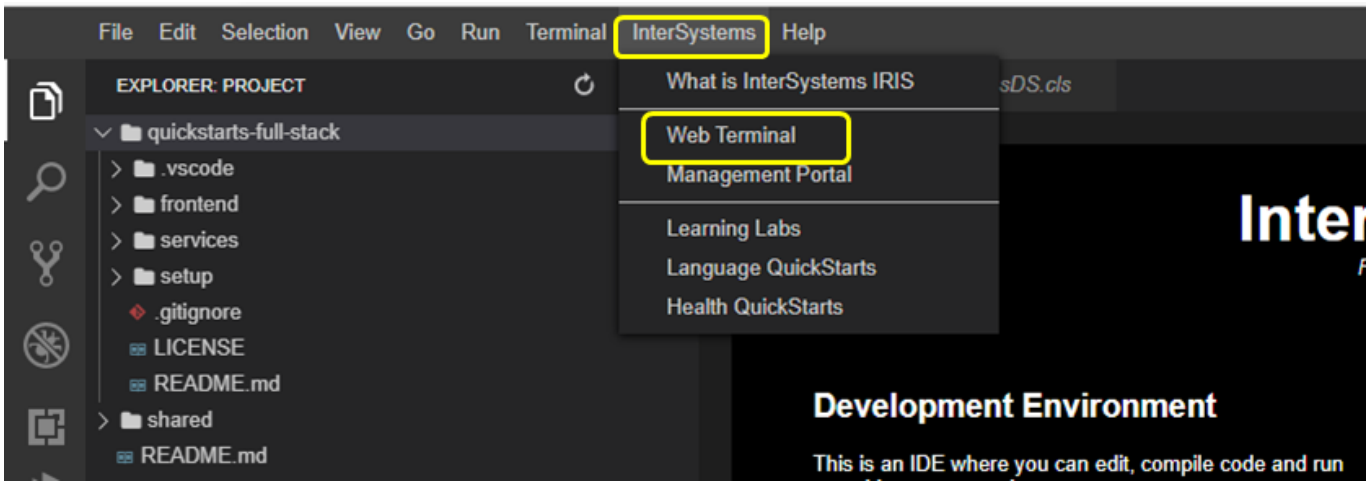
Python から登録したデータが正しくテーブルに登録できているかを、Web
ターミナルを使用して確認します。手順は以下の通りです。

(1) Sandbox 用 IDE 開きます。

https://gettingstarted.intersystems.com/full-stack/full-stack-part-one/#database-query を開き「**SQL database queries**」の近くに下図のように情報が表示されます。



(2) InterSystems > Web Terminal を開きます。

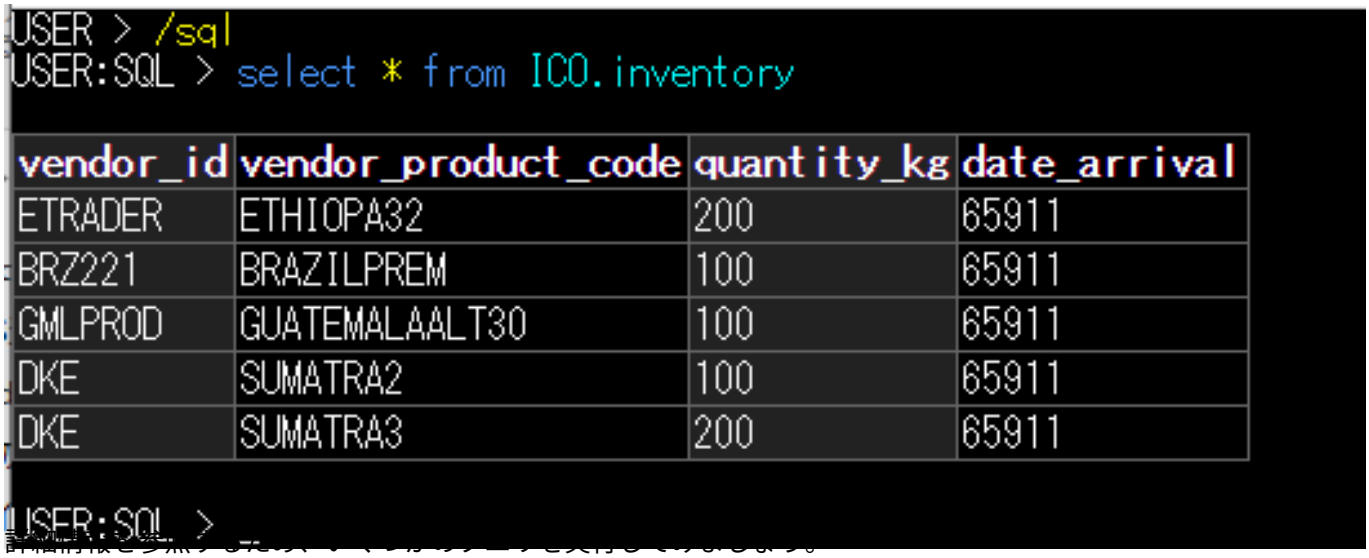


(3) ログイン画面が表示されたら、ユーザ名：tech パスワード:demo でログインしてください。

(4) **/sql** を入力し、SQL 実行モードに切り替えます。

以下実行します。

```
select * from ICO.inventory
```



100 kg 以上の大型商品を検索するクエリ

```
SELECT * FROM ICO.inventory WHERE quantity_kg > 100
```

特定のベンダー（例では、DKEの文字から始まるベンダー名を抽出）のすべての在庫を確認するクエリ

```
SELECT * FROM ICO.inventory WHERE vendor_id LIKE 'DKE'
```

独自の在庫を追加してみましょう！

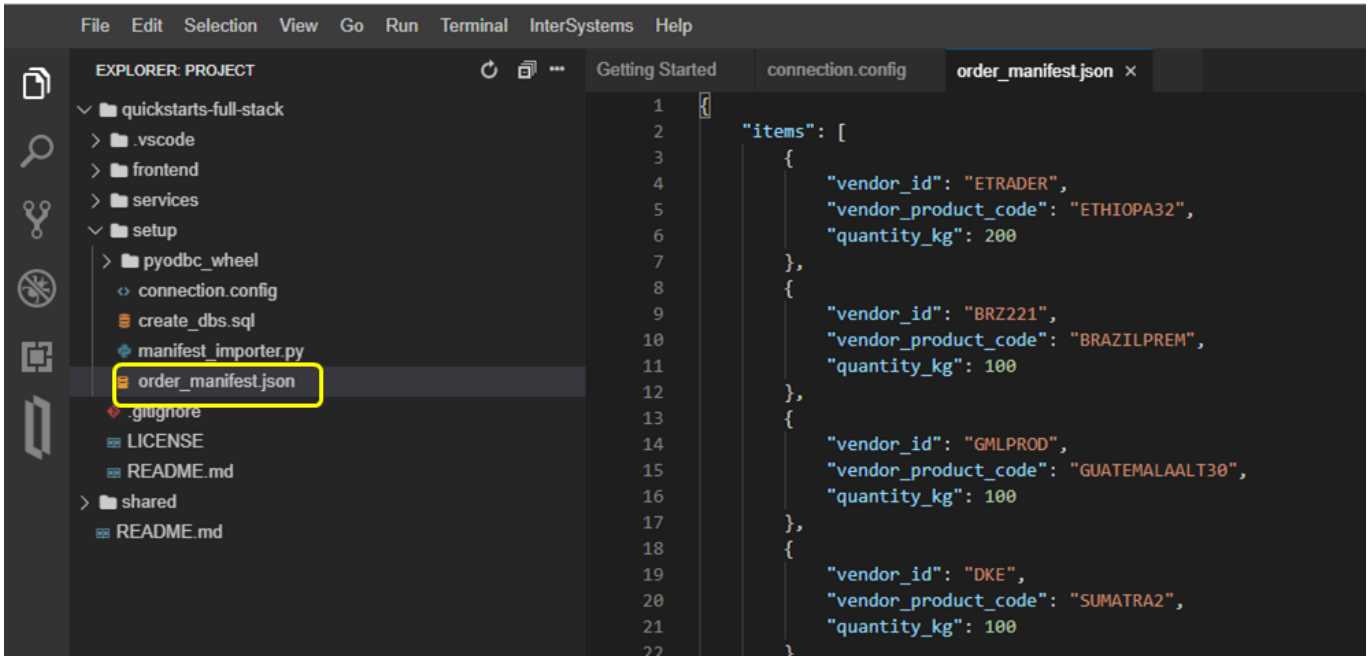
最後に、在庫を増やしてみましょう。vendor id、verdorproductcode、quantitykg に独自の値を登録した JSON マニフェストファイルを作成します。

(1) Sandbox の IDE の左画面を利用して、ordermanifest.json を開き、File > Save As... メニューを利用して、別名保存します（ordermanifest-original.json）。

注意：データロード用スクリプトでは、JSON マニフェストファイルに含まれるコメント行の対応を行っていません。コメントは使用しないでください。

(2) Sandbox の IDE で ordermanifest.json ファイルを開きます。

(3) 好きな値を登録します（英数字でご記入ください）。



(4) python manifestimporter.py を実行します。

パート1で確認できたこと

IRIS をリレーショナルデータベースとして使用できることが確認できました。

また、Python から SQL 文を実行できることを確認できました。

この後は、パート2 に進み、会社全体で使用する REST サービスを作成します。

パート2：ObjectScript を使用した REST サービスの開発

（オリジナルページはこちら　https://gettingstarted.intersystems.com/full-stack/part-two-rest-services/）

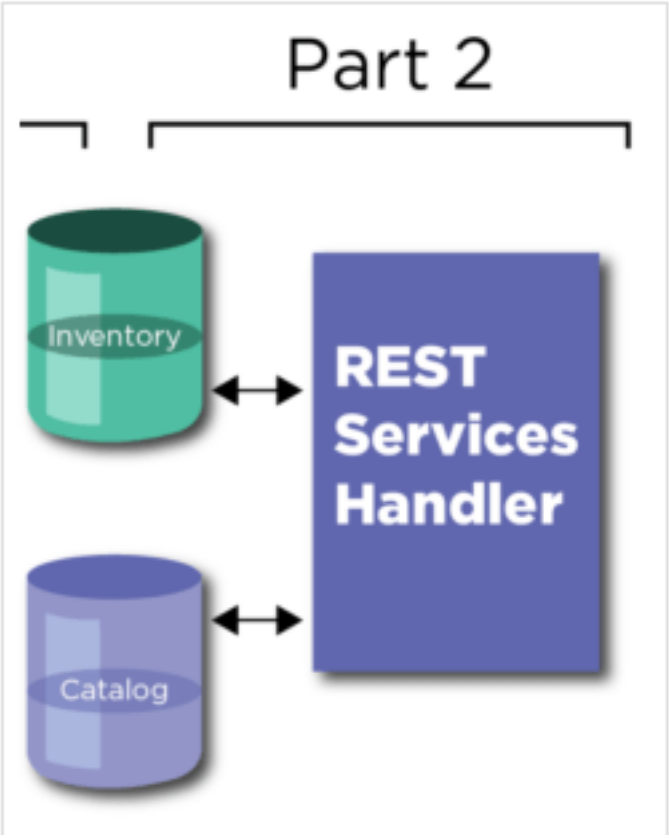
適切に設計されたシステムでは、ビジネスアプリケーションをデータベース上で直接操作することはありません。
その代わりに、サービスを介したアクセスを提供し、実行されるアクションを制御／監視できるようにします。

パート2では、ビジネスを機能させるために必要な RESTful Web サービスを作成します。

ほとんどのデータベースでは、Java Spring、Python Flask、Node.js Express
などのミドルウェアフレームワークを使用して、SQLでデータレイヤーとやり取りしています。IRIS
でもその方法を利用することはできますが、**より簡単で高性能な別の選択肢があります。**

- ObjectScriptでコードを記述する：ストアドプロシージャのパフォーマンスと、プログラミング言語の柔軟性、パワー、使いやすさを手に入れることができます。
- ミドルウェアが不要です：ミドルウェア層が組み込まれています。

コツさえつかめば、ObjectScript は Web
アプリケーションのバックエンドを構築するための最速の方法と言えます！



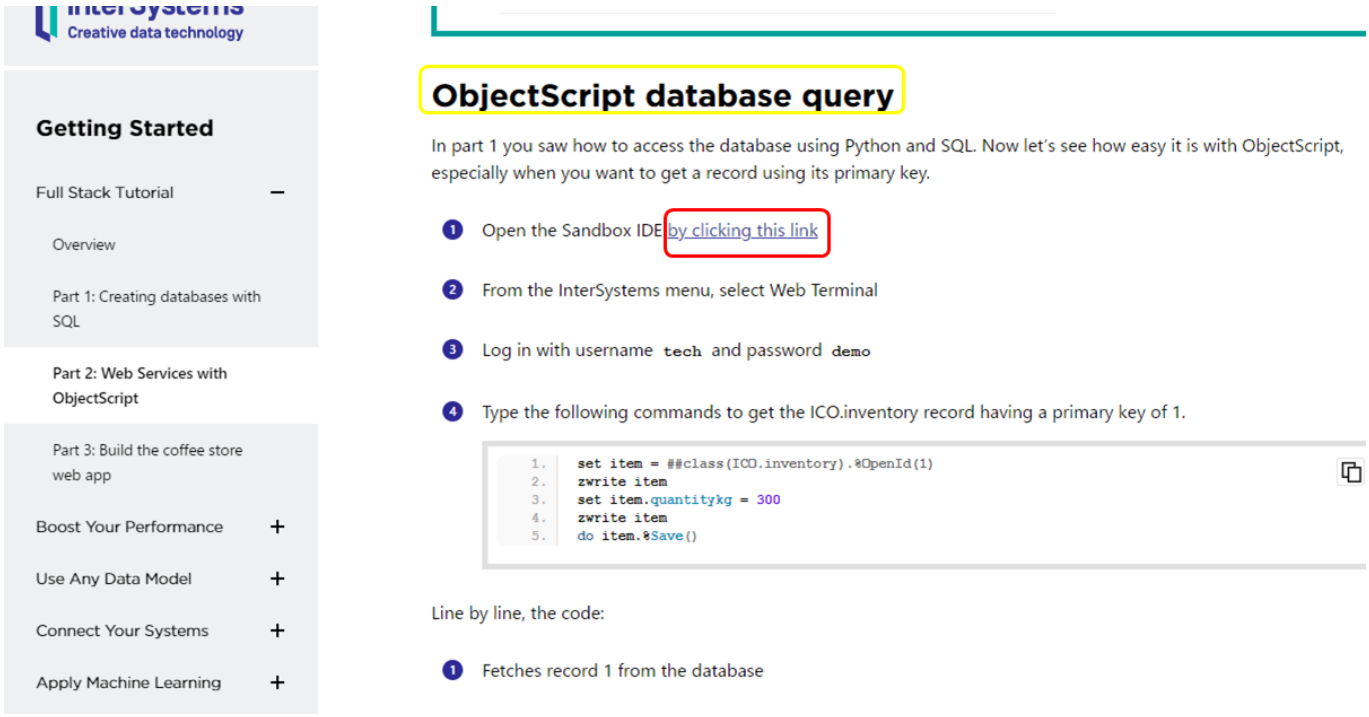
ObjectScript を使用したデータの操作

パート1 では、Python と SQL を使ってデータベースにアクセスする方法をご紹介しました。

パート2では、ObjectScript によるアクセスがいかに簡単か、特に主キーを使ってレコードを取得したい場合について見ていきたいと思います。

(1) Sandbox の IDE を開きます。

https://gettingstarted.intersystems.com/full-stack/part-two-rest-services/#query-cos を開き、「**ObjectScript database query**」の近くに IDE へのリンクが表示されます。



IDE
などのアクセス情報の表示が、 Open the IDE (setting requires sandbox - click here)
のように表示されていたら、ピンク色のリンクをクリックしてください。

(2) IDE のメニューから InterSystems > Web Terminal を選択します。

(3) ログイン画面が表示されたらユーザ名：tech パスワード：demo でログインしてください。

(4) 以下の ObjectScript のコマンドをコピーし、ターミナルに貼り付けて実行してください。

以下のコードは 主キーが 1 である ICO.Inventory のレコードを取得しています。

```
set item = ##class(ICO.inventory).%OpenId(1)
zwrite item
set item.quantitykg = 300
zwrite item
do item.%Save()
```

1行ずつコードを解説します。

1行目：データベースからレコードを 1 件ロードしています。

2行目：レコードデータをターミナルに表示しています。

3行目：在庫数が設定されている quantitykg の値を 300（kg） に変更しています。

4行目：変更を確認するため、画面に再度表示しています。

5行目：データベースに変更データを保存しています。

```
USER > set item = ##class(ICO.inventory).%OpenId(1) zwrite item set item.quantitykg = 300 zwrite item do item.%Save()
item=1@ICO.inventory  ; <OREF>
+----------------- general information ---------------
|      oref value: 1
|      class name: ICO.inventory
|           %%OID: $lb("1","ICO.inventory")
| reference count: 2
+----------------- attribute values ------------------
|       %Concurrency = 1  <Set>
|        datearrival = 65911
|         quantitykg = 200
|           vendorid = "ETRADER"
|  vendorproductcode = "ETHIOPA32"
+-----------------------------------------------------
item=1@ICO.inventory  ; <OREF>
+----------------- general information ---------------
|      oref value: 1
|      class name: ICO.inventory
|           %%OID: $lb("1","ICO.inventory")
| reference count: 2
+----------------- attribute values ------------------
|       %Concurrency = 1  <Set>
|        datearrival = 65911
|         quantitykg = 300
|           vendorid = "ETRADER"
|  vendorproductcode = "ETHIOPA32"
+-----------------------------------------------------

USER >
```

InterSystems Web Terminal
は、通常のコマンド・ライン・ターミナルと同様に動作しますが、ObjectScript
コマンドも実行できます。

ObjectScript と SQL によるデータベースの操作

それでは、ObjectScript を使用して SQL を使ったより複雑なクエリを実行してみましょう。

Web Terminal に戻って、次のように入力してください。

```
set sqlquery = "SELECT * FROM ICO.inventory"
set resultset = ##class(%SQL.Statement).%ExecDirect(,sqlquery)
while resultset.%Next() { Write !, resultset.%Get("vendor_id") }
```

1行ずつコードを解説します。

1行目：実行したい SELECT 文を変数に設定しています。

2行目：%SQL.Statement クラスを使用して、1 行目で設定した変数に設定した SQL 文を実行し、変数 resultset に検索結果のインスタンスを設定しています。

3行目：resultset 変数を使用して、検索結果を繰り返し行移動（%Next()）しながら vendorid プロパティの値を画面表示しています。

```
USER > set sqlquery = "SELECT * FROM ICO.inventory" set resultset = ##class(%SQL.Statement).%ExecDirect(,sqlquery) while resultset.%Next() { Write !, resultset.%Get("vendor_id") }

ETRADER
BRZ221
GMLPROD
DKE
DKE
TestVendor
USER >
```

ObjectScript のクラスを書いてみる

今まで試した Web Terminal
のスクリプトは、1行に沢山のコードを指定しているので、見やすいとは言えません。

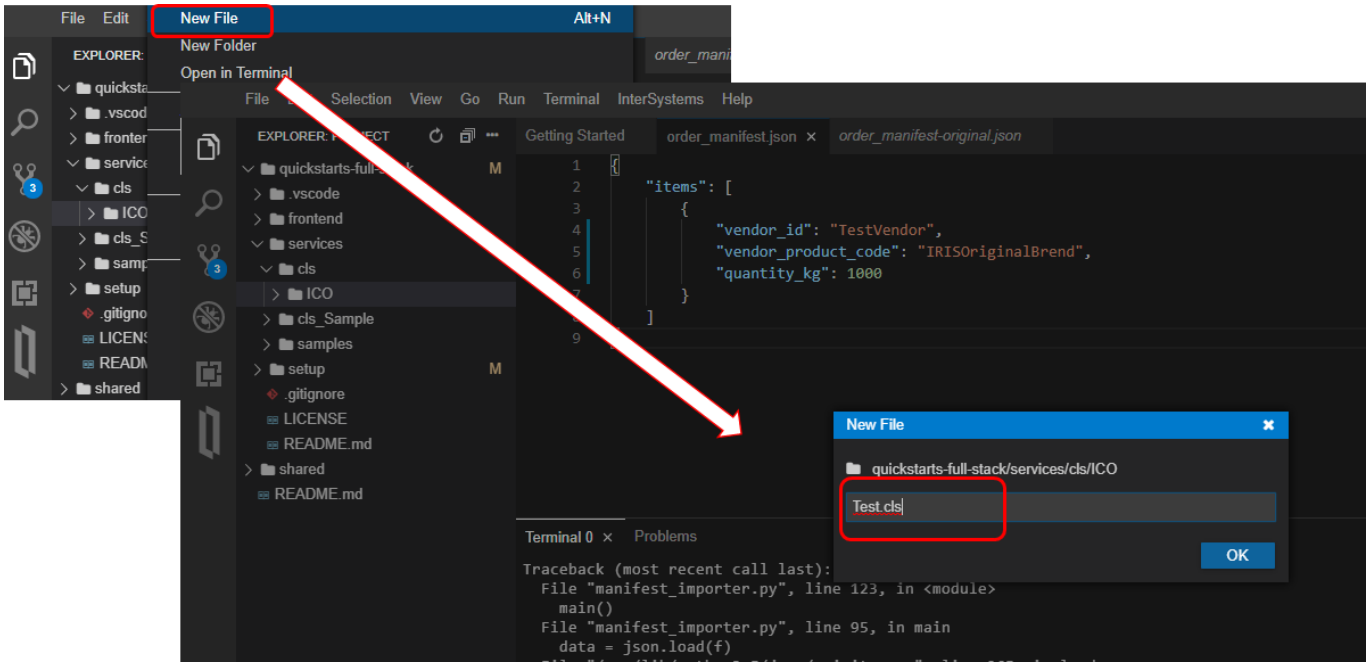
今度は、ObjectScript のコードをファイルにまとめてみましょう。

(1) Sandbox の IDE を開きます

(2) 画面左のフォルダから、quickstarts-full-stack > services > cls > ICO を開きます。

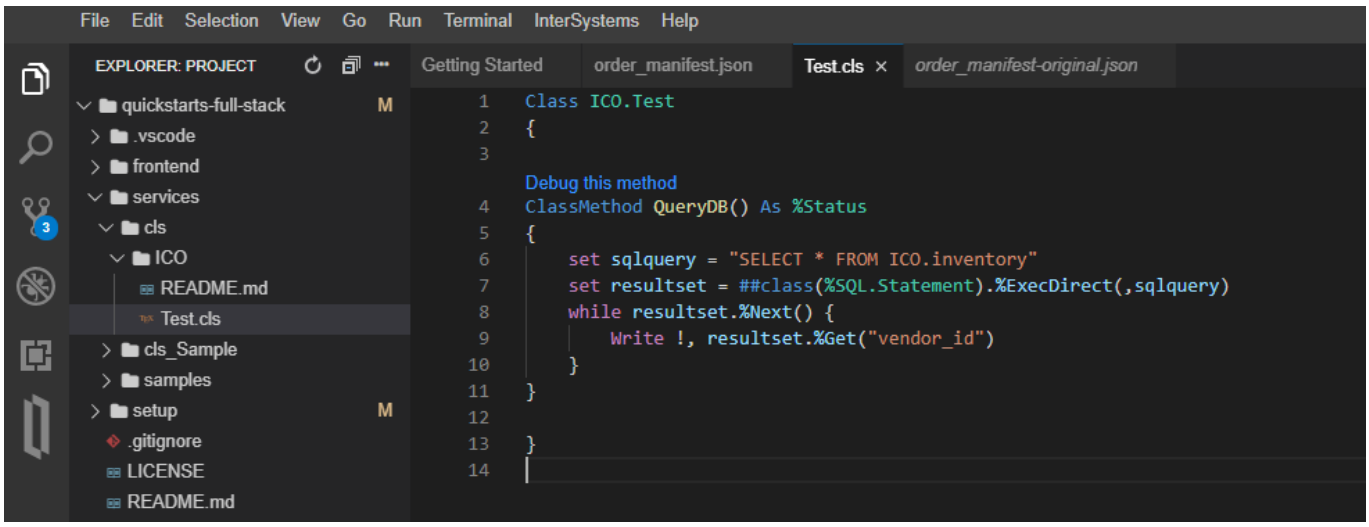
(3) ICO フォルダを右クリックし、New File メニューを選択します。

(4) ファイル名に Test.cls を指定し、OK ボタンをクリックします。



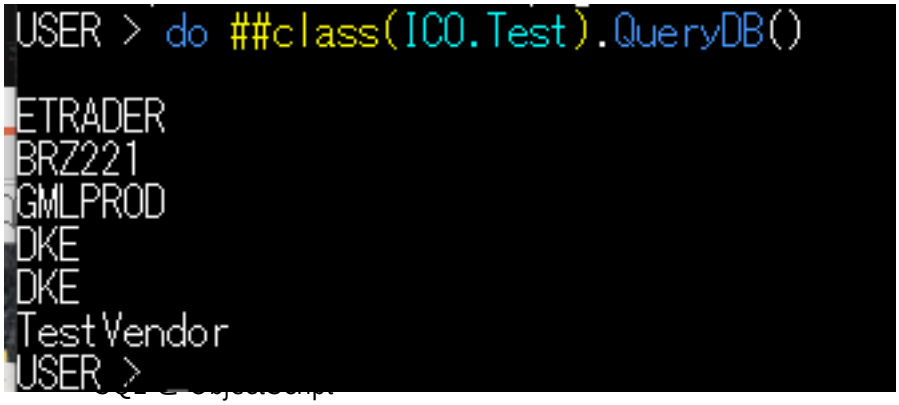
(5) ファイルに以下の文字列を記述します（ICO.Test クラスを作成しています）。

```
Class ICO.Test
{
/// Description
ClassMethod QueryDB() As %Status
{
  set sqlquery = "SELECT * FROM ICO.inventory"
  set resultset = ##class(%SQL.Statement).%ExecDirect(,sqlquery)
  while resultset.%Next() {
  Write !, resultset.%Get("vendorid")
  }
}
}
```



記述後、ファイルを保存し（Ctrl + s）Web Terminal で以下実行します。

```
do ##class(ICO.Test).QueryDB()
```



の使い分けや、マルチモデル機能については、開発者コミュニティの記事で詳しく紹介しています。

Web サービスの作成

ここまでの内容は、ObjectScript のプログラミング入門編でした。

ここからは、確認した内容を活かして、コーヒー焙煎ビジネスに必要な Web サービスを構築してみましょう。

作成概要は以下の通りです。

(1) 事前に作成されたコードをコピーしてコンパイルします。

(2) JSON の操作が行えるようにテーブル定義を変更します（JSONアダプタクラスの継承を追加します）。

(3) RESTful API がどのように構築されているか確認します。

(4) curl とブラウザを使い、Web サービスのデプロイとテストを行います。

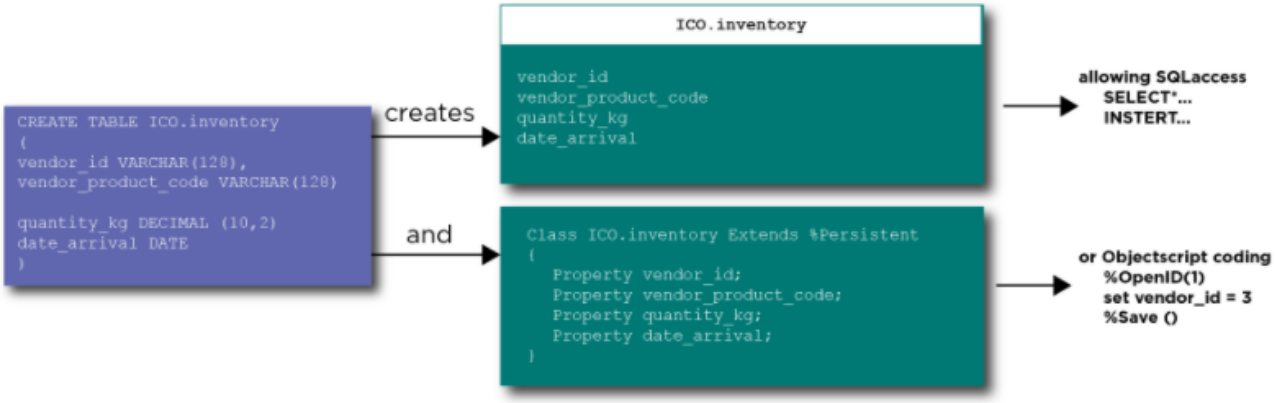
(5) Web サービスを利用して、コーヒーの在庫を店舗に移動させます。

(6) 販売情報を記録します。

パート1では、標準的な SQL を使用してデータベースのテーブルを作成しました。

実は、その作成の裏では、対応する ObjectScript クラスも作成されていることをご説明していませんでした。

テーブルを作成するとクラス定義（永続クラス）も用意される仕組みにより、開発者は目的に応じて、SQL とコードの記述が簡単に行えるようになっています。

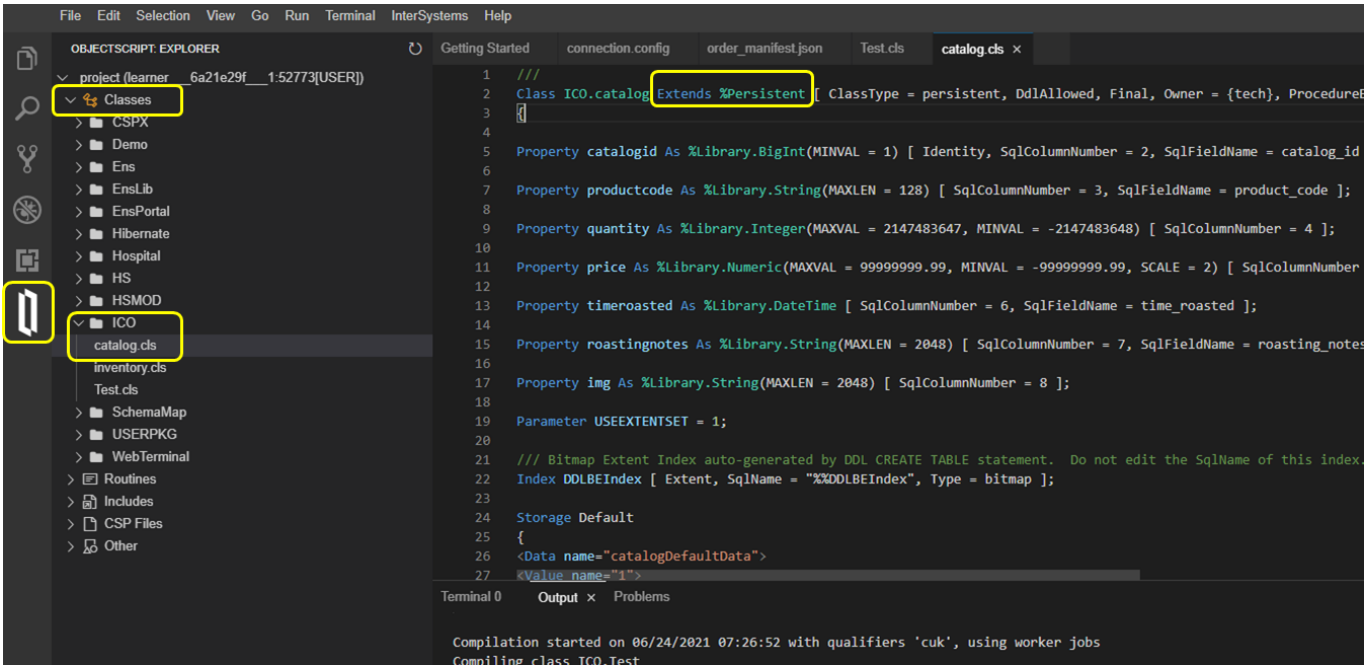


SQL で取得したデータを JSON としても出力できるようにするためには、データベースのテーブル定義に ObjectScript
のコードに少し加える必要があります。その前に、データベースに登録されているコードを見てみましょう。

(1) Sandbox の IDE を開き、InterSystems のアイコン  をクリックします。

(2) IDE の画面左側で Classes > ICO を展開します。catalog.cls と
inventory.cls の名称のファイルが確認できます。

パート1 で SQL 文を使ってデータベースにテーブルを作成した際に、それに対応する ObjectScript
クラスも同時に作成されていることが確認できます。



(3) catalog.cls をクリックします（上図）。カラムの定義は Property 定義として表示されています。また、データ型や文字長の設定は見覚えのある値が登録されているはずです。同様に、inventory.cls をご参照ください。

(4) 開いたファイルはすべて閉じておいてください。

JSON の操作が行えるテーブル定義に変更する

JSON の入出力を可能するため、確認したクラスを少し変更する必要があります。以下の手順で変更しましょう。

(1) Sandbox の IDE の  ファイルのアイコンをクリックします。

(2) quickstarts-full-stack > services > cls > ICO を開きます。

(3) ICO フォルダの上で右クリック > New File を選択し、catalog.cls の名称で保存します。

(4) ICO フォルダの上で右クリック > New File を選択し、inventory.cls の名称で保存します。

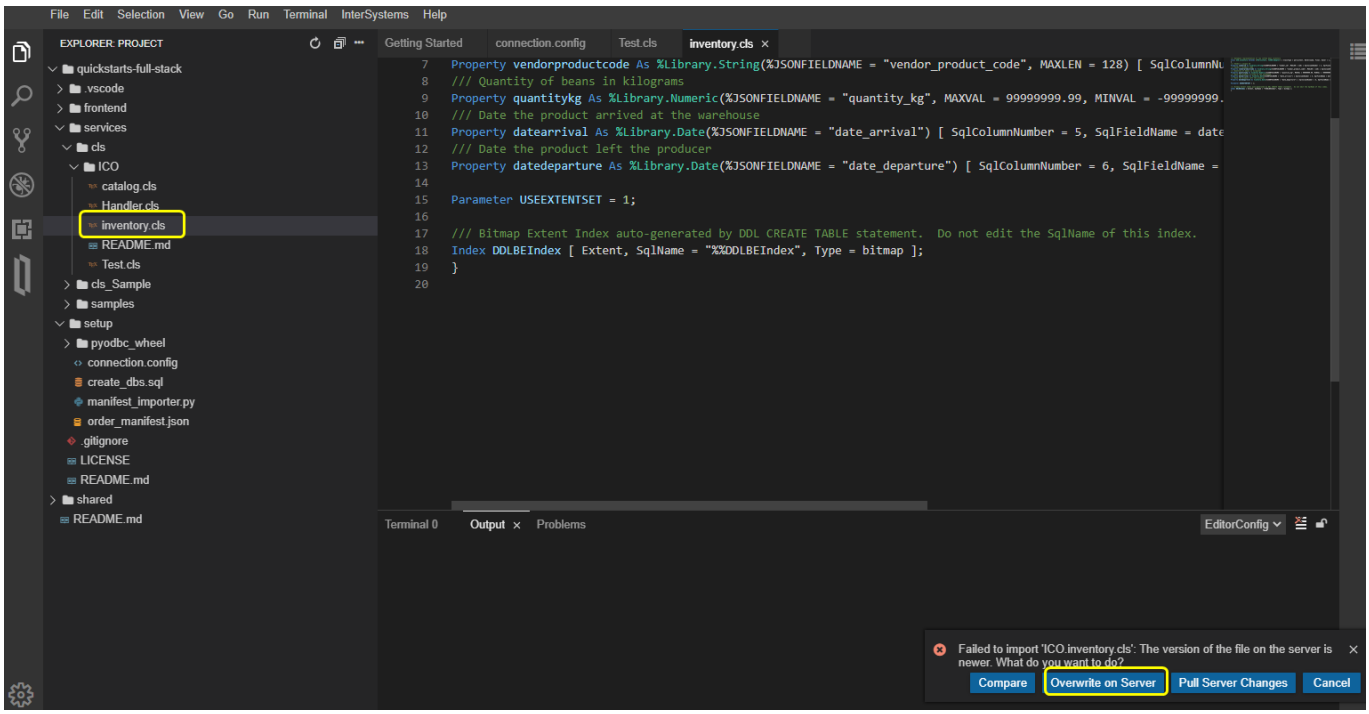
(5) quickstarts-full-stack > services > clsSample > ICO を開きます。

(6) catlog.cls を開き、ファイル内を全てコピーし、quickstarts-full-stack > services > cls > ICO > catalog.cls に貼り付けます。

(7) quickstarts-full-stack > services > clsSample > ICO > inventory.cls を開き、ファイル内を全てコピーし、quickstarts-full-stack > services > cls > ICO > inventory.cls に貼り付けます。

(8) 変更したファイル（quickstarts-full-stack > services > cls > ICO > catalog.cls と inventory.cls ）を保存します。

保存時、画面右下に選択欄が表示されるので **"Overwrite on Server"** を選択します。この操作により、%JSON.Adapter と プロパティのパラメータ：%JSONFIELDNAME を追加したクラスの新しいバージョンがサーバにアップロードされ、コンパイルされます。



(9) 2つのファイルに以下の操作を行いました。

A：%JSON.Adapter を継承し、テーブルのデータに対して自動的に JSON 出力が行えるように設定しました。

B：プロパティのパラメータ：%JSONFIELDNAME を追加し、JSON
出力時に使用するプロパティ名を変更しました。

ここまでの流れで、データベース用意したテーブル定義が、ObjectScript のクラス定義としても表現され、SQL と ObjectScript を使ってデータベースに対する操作ができることを確認できました。

これで、Web API を構築する準備が整いました！

RESTful サービスを作ってみよう！

テーブル定義（クラス定義）へ JSON アダプタを継承することで JSON の操作が行えるようになり、REST サービスを作成する準備が整いました！

先ほどと同様の手順で、クラス定義を追加します。

quickstarts-full-stack > services > cls > ICO 以下に Handler.cls を新規に作成します。手順は以下の通りです。

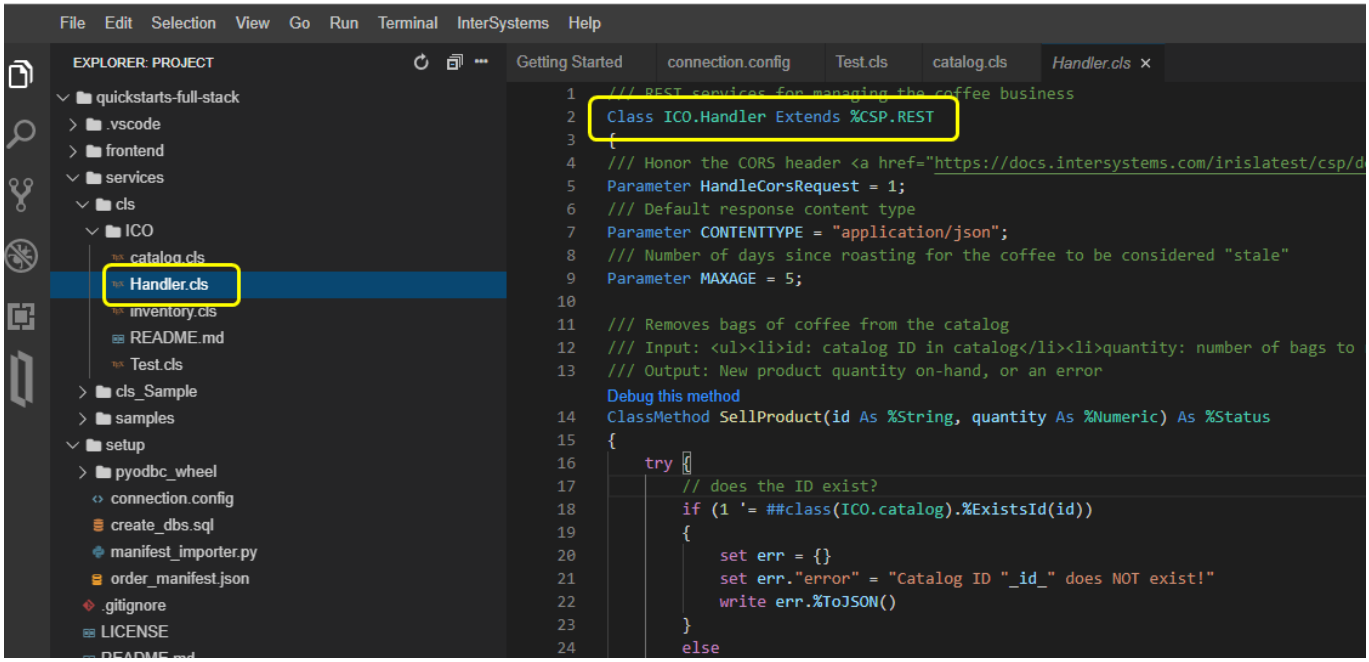
(1) Sandbox の IDE の  ファイルのアイコンをクリックします。

(2) quickstarts-full-stack > services > cls > ICO フォルダを開きます。

(3) ICO フォルダを右クリックし New File を選択し、クラス名に Handler.cls を設定します。

(4) quickstarts-full-stack > services > clsSamples > ICO フォルダを開きます。

(5) quickstarts-full-stack > services > clsSamples > ICO> Handler.cls
クラスを開き、クラス定義の中身を全部コピーし、(3) で作成した quickstarts-full-stack > services > cls > ICO > Handler.cls に貼り付けます。



(6) ファイルを保存します（今回は "Overwrite on Serve" のオプション表示は出てきません）。

Handler.cls の作成により、ミドルウェアとなる REST API が作成できました。

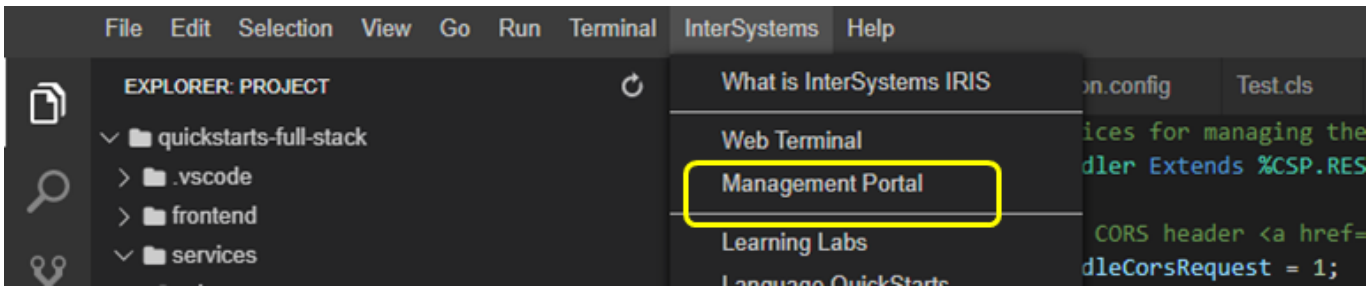
サービスを起動させるための最後の手順として、Web に公開するための設定を追加します。

IRIS では、Node.js の Express や Python の Flask と同じように、Web リクエストを ObjectScript コードにルーティングするツールを提供しています。

管理ポータルを使用して作成した ICO.Handler クラスを Web リクエスト時に呼び出されるように設定してみましょう。

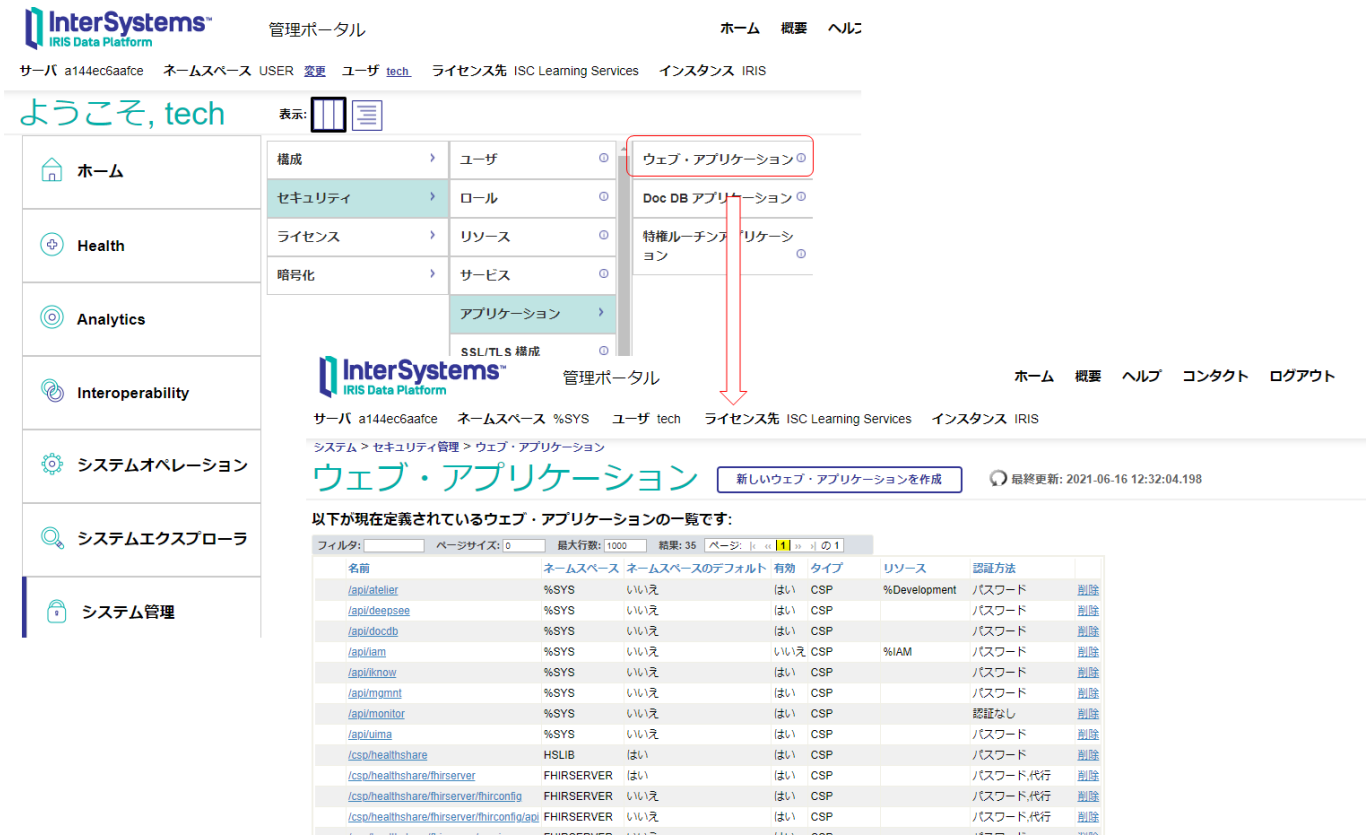
管理ポータルを開きます。

Sandbox の IDE のメニューから InterSystems > Management Portal を選択します。

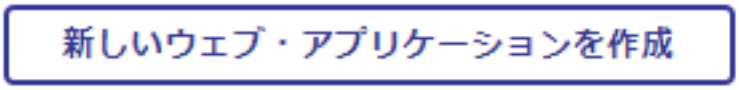


管理ポータルが開いたら、以下メニューにアクセスします。

システム管理 > セキュリティ > アプリケーション > ウェブ・アプリケーション



以下の手順で、REST のエンドポイントのパスを作成します。

(1) 

ボタンをクリックします。

(2) 名前 に **/api/coffeeco** を設定します。

(3) ネームスペースに **USER** を設定します。

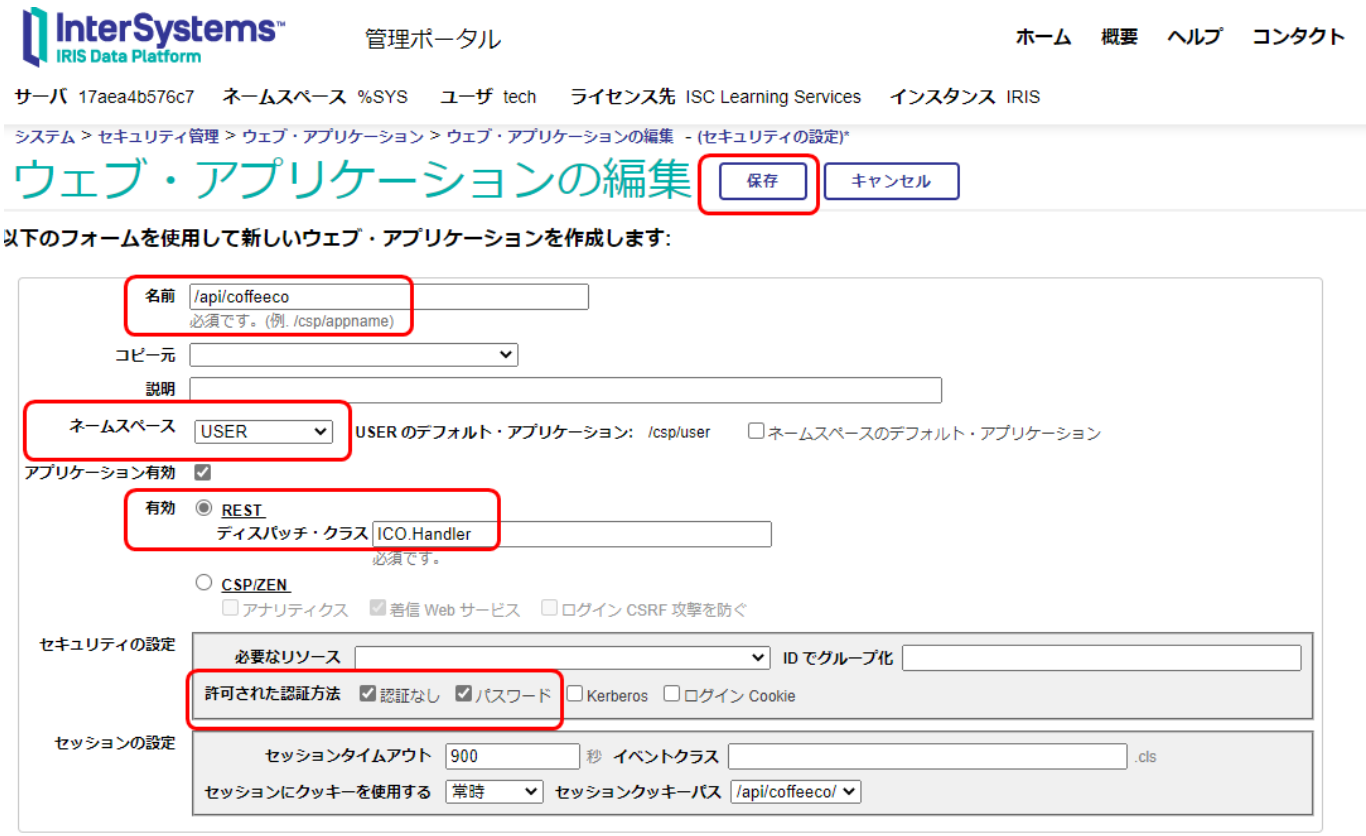
(4) アプリケーション有効 にチェックが入っていることを確認します。

(5) **REST** にチェックします。

(6) ディスパッチクラス に **ICO.Handler** を設定します。

(7) 許可された認証方法では、「**認証なし**」と「**パスワード**」にチェックを入れます。

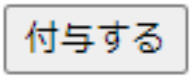
(8) **保存ボタン**をクリックし、設定を保存します。



保存後、「アプリケーション・ロール」タブが表示されるので、クリックし、「アプリケーションロール」に %All を設定します。

設定には、「利用可能」の一覧から %All を選択し、 

をクリックし、画面右側の「選択済み」に移動させます。

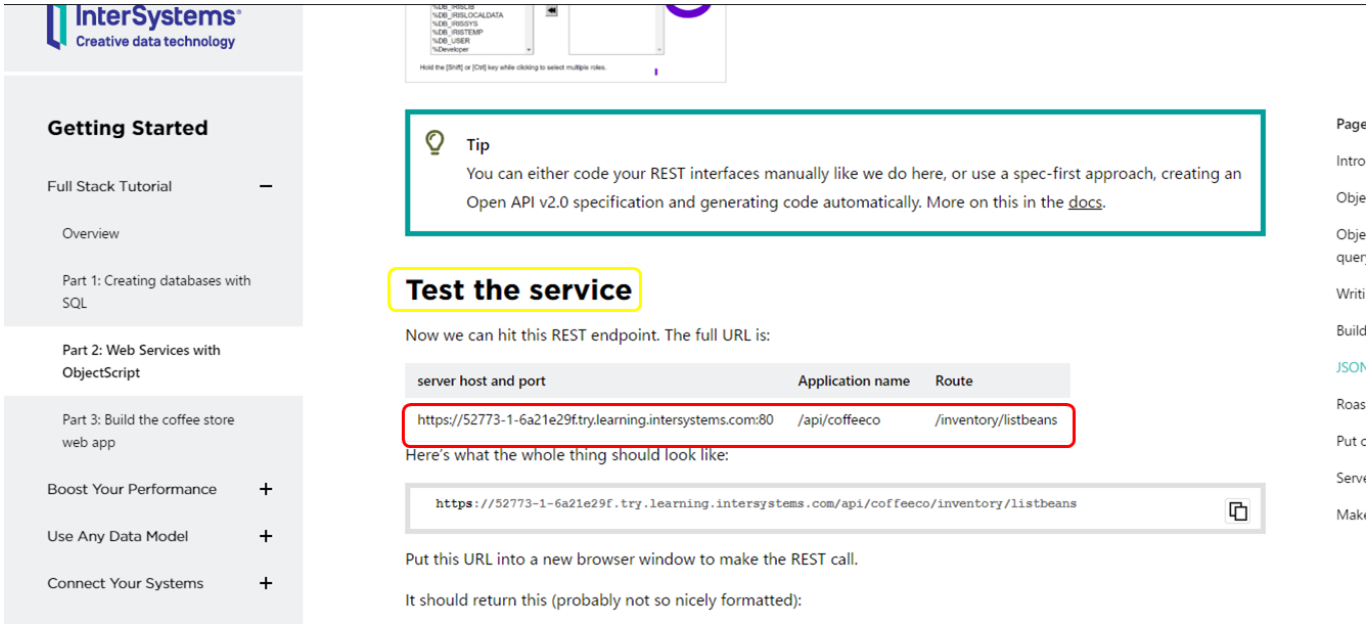
その後、 

ボタンをクリックすると、「アプリケーションロール」に %All が追加されます。

REST
インターフェースでは、ここで紹介しているように手動でコーディングすることもできますし、Open API v2.0 の仕様を作成して自動的にコードを生成する、API
ファーストのアプローチをとることもできます。詳しくはドキュメントをご覧ください。

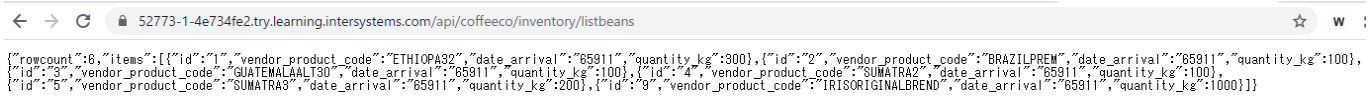
作成したサービスのテストをしましょう

ここまでの設定で、REST
用エンドポイントの作成が完了しました。エンドポイントは https://gettingstarted.intersystems.com/full-stack/part-two-rest-services/#json-enable-the-data-tables を開き「**Test the service**」の近くに表示されます（Sandbox 毎に情報は異なります）。



`https://Sandbox ??????Web???????/api/coffeeco/inventory/listbeans`

上記 URL（Webサーバアドレスはお試しいただいている環境に合わせてご変更ください）をコピーし、ブラウザのアドレスバーに入力し、REST サービスをテストすると、以下のような結果が返ります。



見易い表示に変えると、以下のような結果が返ります。

```
{
    "rowcount": 5,
    "items": [{
        "id": "1",
        "vendor_product_code": "ETHIOPA32",
        "date_arrival": "65541",
        "quantity_kg": 400
    }, {
        "id": "2",
        "vendor_product_code": "BRAZILPREM",
        "date_arrival": "65541",
        "quantity_kg": 200
    }, {
        "id": "3",
        "vendor_product_code": "GUATEMALAALT30",
        "date_arrival": "65559",
        "quantity_kg": 200
    }, {
        "id": "4",
        "vendor_product_code": "SUMATRA2",
        "date_arrival": "65559",
        "quantity_kg": 200
    }, {
        "id": "5",
```

```
        "vendor_product_code": "SUMATRA3",
        "date_arrival": "65559",
        "quantity_kg": 400
    }]
}
```

コーヒー豆を焙煎します

現在、ブラウザからのリクエストでレスポンスが返ってくることを確認できたので、サービスが動作していることがわかりました。
ここからは、コーヒービジネスを操作するためのサービスを作成します。

最初に、コーヒーの生豆を在庫から取り出し、焙煎します。この動作には、/api/coffeeco/inventory/getbeans の URL を使用します。

Handler.cls を開き、ファイル末尾に定義された XData UrlMap の <Routes> のタグ内部をご参照ください。

URL /api/coffeeco**/inventory/getbeans** に対して、クラスメソッド **GetRawBeans()**
が呼び出されるように定義されていることがわかります。

URL で値を渡すための URL に含めるパラメータの記載方法にも注目してください。

```
XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
{
<Routes>
    <Route Url="/inventory/listbeans" Method="GET" Call="ListRawBeans" />
    <Route Url="/inventory/getbeans/:id/:quantity" Method="POST" Call="GetRawBeans" />
    <Route Url="/catalog/catalogproduct" Method="POST" Call="CatalogProduct" />
    <Route Url="/catalog/getproducts" Method="GET" Call="GetProducts" />
    <Route Url="/catalog/getproducts/:fresh" Method="GET" Call="GetProducts" />
    <Route Url="/catalog/sellproduct/:id/:quantity" Method="POST" Call="SellProduct" />
</Routes>
}
```

クラス定義内で GetRawBeans()メソッドを参照します。

レコードの主キーとなる値が URL の **:id** で渡されます。GetRawBeans()
メソッドが実行されるとき、メソッドの第 1 引数に **:id** の情報が渡されます。

この値を %OpenId() メソッドの引数に指定することで、ObjectScript
を使用してデータベースから対象のデータをロードすることができます（とても簡単な方法です）。

残りのコードは以下の通りです。

(1) quantity に設定されている値が十分な量であるかの確認

(2) 在庫から要求された量を減らす処理

(3) 要求元に新しい在庫量を返す処理

```
Debug this method
ClassMethod GetRawBeans(id As %String, quantity As %Numeric) As %Status
{
    try {
        // does the vendor ID exist?
        if (1 '= ##class(ICO.inventory).%ExistsId(id))
        {
            set err = {}
            set err."error" = "ID "_id_" does NOT exist!"
            write err.%ToJSON()
        }
        else
        {
            // do we have enough quantity?
            set item = ##class(ICO.inventory).%OpenId(id)
            if (quantity > item.quantitykg)
            {
                set err = {}
                set err."error" = "You tried to get "_quantity_", but we only have "_item.quantitykg_" kilograms available."
                write err.%ToJSON()
            }
            else
            {
                // decrement the quantity and return the requested inventory
                set item.quantitykg = (item.quantitykg - quantity)
                set sc = item.%Save()
                do item.%JSONExportToString(.outstring)
                write outstring
            }
        }
    } catch (oException) {
        set expobj = {}
        set expobj."exception" = oException.%AsSystemError()
        write expobj.%ToJSON()
    }
    Quit $$$OK
}
```

在庫から豆を取り出し、焙煎を行うためのリクエストを実行してみます。

今回のリクエストは、ブラウザでテストすることはできません（ブラウザで、POST
要求を送信することができないためです）。

Sandbox の IDE のターミナルウィンドウでは、curl
コマンドを実行できるので、以下のように入力してみください。

注意：リクエストに使用する Web サーバアドレスは Sandbox
の利用環境により異なります。お使いの環境のサーバ名に修正して実行してください。

```
curl -X POST https://52773-1-4e734fe2.try.learning.intersystems.com/api/coffeeco/inventory/getbeans/1/2.4
```

```
theia /home/project/quickstarts-full-stack/setup $ curl -X POST https://52773-1-4e734fe2.try.learning.intersystems.com/api/coffeeco/inventory/getbeans/1/2.4
{"vendor_id":"ETRADER","vendor_product_code":"ETHIOPA32","quantity_kg":297.6,"date_arrival":"2021-06-16"}theia /home/project/quickstarts-full-stack/
theia /home/project/quickstarts-full-stack/setup $
```

お店にコーヒーを並べる

このチュートリアルではシンプルな例を利用しているため、リクエストされた生豆の量はどこにも記録されていません。

コーヒーを焙煎し、袋詰めし、店頭に並べて販売する処理についてはリクエストする Web
アプリケーション側で処理することとします。

その処理を追加するため、quickstarts-full-stack > services > samples ディレクトリに 2
つのスクリプトが用意されています。

- **createproducts.sh**

5 つのサンプルコーヒー製品の情報を作成するシェルです。最初の 3 つは本日焙煎されたもので、最後の 2 つは 6 日前に焙煎されたものです。

このシェルの実行で、鮮度の古い6日前の焙煎豆情報を作成しています。オンラインショップでは、鮮度が古い焙煎豆を割引して販売したいので、割引対象データとして準備しています。

- **loadproducts.sh**

curl コマンドを実行して、対象ディレクトリ内のすべての JSON ファイルを繰り返し参照し、用意した REST サービスを使用して JSON ファイル内のデータを ICO.catalog にロードします。

それではシェルを実行してみましょう。手順は以下の通りです。

(1) Sandbox の IDE のターミナルウィンドウで以下実行します。

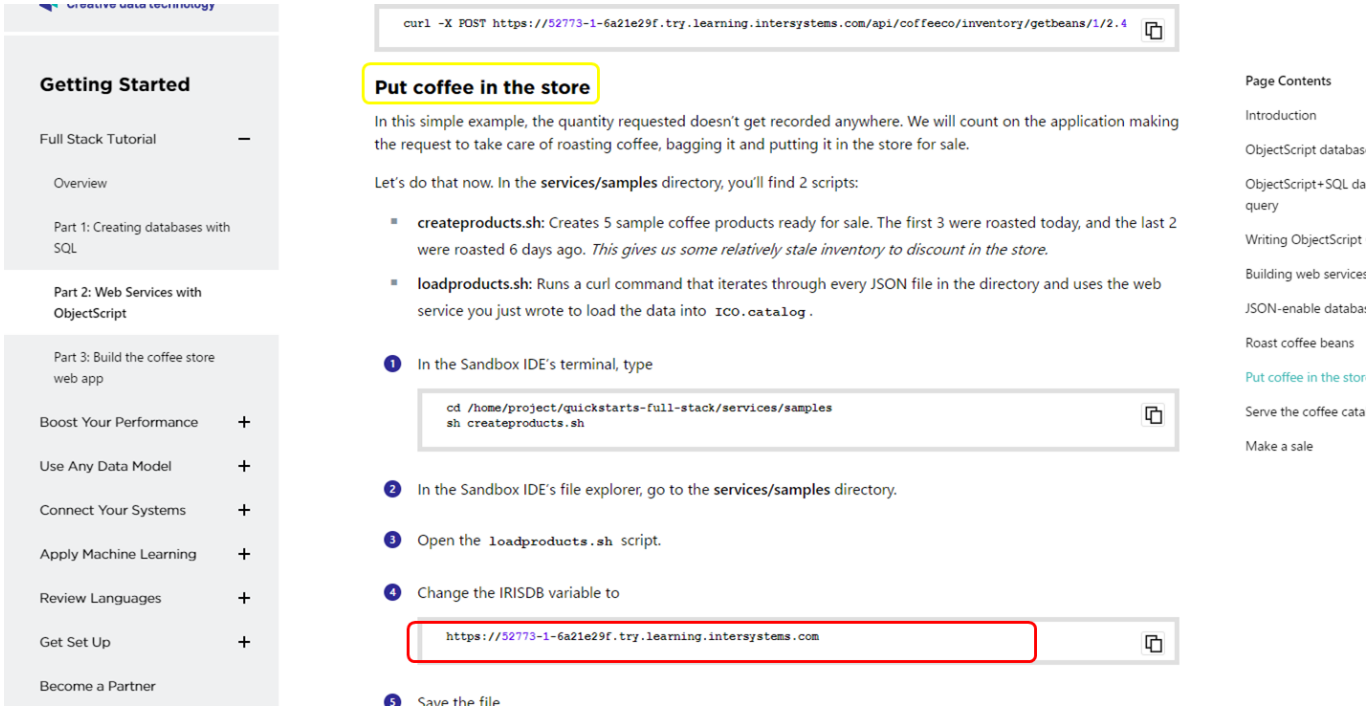
```
cd /home/project/quickstarts-full-stack/services/samples
sh createproducts.sh
```

(2) IDE で、quickstarts-full-stack > services > samples ディレクトリに 5 つの JSON ファイルが作成できたことを確認してください。

(3) loadproducts.sh を IDE で開きます。

(4) 環境変数 IRISDB の値を修正します（Sandbox のウェブサーバアドレスに修正します）。

https://gettingstarted.intersystems.com/full-stack/part-two-rest-services/#coffee-to-store を開き「**Put coffee in the store**」の近くに表示される情報をご利用ください。



```
IRISDB="https://52773-1-4e734fe2.try.learning.intersystems.com"

for jsonfile in *.json ;
do
    curl -d "@$jsonfile" -H "Content-Type: application/json" -X POST $IRISDB"/api/coffeeco/catalog/catalogproduct"
done;
```

(5) loadproducts.sh を保存します。

(6) Sandbox の IDE のターミナルウィンドウで以下実行します。

```
cd /home/project/quickstarts-full-stack/services/samples
sh loadproducts.sh
```

```
theia /home/project/quickstarts-full-stack/services/samples $ cd /home/project/quickstarts-full-stack/services/samples
theia /home/project/quickstarts-full-stack/services/samples $ sh loadproducts.sh
{"success":1}{"success":1}{"success":1}{"success":1}{"success":1}theia /home/project/quickstarts-full-stack/services/samples $
```

シェルを実行したくない場合は、curl コマンドを利用してデータを読み込むこともできます。

コマンド実行例は以下の通りです（Webサーバアドレスは Sandbox
ごとに異なります。実行環境に合わせて実行してください。）。

```
curl -d "@product_brazil_dark.json" -H "Content-Type: application/json" -X POST https
://52773-1-4e734fe2.try.learning.intersystems.com/api/coffeeco/catalog/catalogproduct
```

この時点で、生豆を出荷し在庫に入れ、一部は焙煎してパッケージ化し ICO.catalog
テーブルに保存したため、オンラインショップで販売できるようになりました。

コーヒーカタログを提供する

ここからは、フロントエンドのオンラインストアフロントに必要なサービスを考えていきます。Web
開発者は、以下の用途で利用できる一連のサービスを必要としています。

(1) 焙煎したばかりの新鮮なコーヒーを販売してほしい

(2) 値引きして販売する必要のある鮮度の古いコーヒーバック情報を入手したい

(3) コーヒーバックの販売記録を作りたい

最初の2項目は非常に簡単です。読み取り専用のサービスとなるので、GET要求を使用すれば取得できます。

両方の GET 要求を 1 つの GetProducts()メソッドで処理できるように、新鮮な在庫と古い鮮度の在庫のどちらを返すか指定できる入力引数を用意することもできます。

Handler.cls （quickstarts-full-stack > services > cls > ICO > Handler.cls）の下の方に定義されている UrlMap
定義をご参照ください。

/catalog/getproducts は、全て新しい鮮度のコーヒーを返すUrlです。

/catalog/getproducts/:fresh は、**/catalog/getproducts** に似ていますが、鮮度の古いコーヒーを取得できる追加のパラメータを用意しています（**fresh** が 0 の場合に鮮度が古い情報を返します）。

/catalog/sellproduct/:id/:quantity は、クライアントが特定の商品のバッグを販売したことを記録する処理の Url です。

```
XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
{
<Routes>
    <Route Url="/inventory/listbeans" Method="GET" Call="ListRawBeans" />
    <Route Url="/inventory/getbeans/:id/:quantity" Method="POST" Call="GetRawBeans" /
>
    <Route Url="/catalog/catalogproduct" Method="POST" Call="CatalogProduct" />
    <Route Url="/catalog/getproducts" Method="GET" Call="GetProducts" />
    <Route Url="/catalog/getproducts/:fresh" Method="GET" Call="GetProducts" />
    <Route Url="/catalog/sellproduct/:id/:quantity" Method="POST" Call="SellProduct"
/>
</Routes>
}
```

GetProducts() メソッドでは、SQL を使用してクエリを実行し、返されたレコードを繰り返し処理で取得し、取得できた情報を JSON オブジェクトに設定し、 JSON 配列に追加し、最後に呼び出し元に作成した JSON データを返送しています。

以上の流れで、Web 開発者が販売中の商品を紹介する素敵なサイトを構築するために必要な情報がすべて揃いました！

コーヒーを販売する

販売したコーヒーを記録するサービスの SellProduct() は、プロダクト ID と販売されるバッグの数量が引数として指定されます。

ここでは、エラーチェックや特別な支払い処理、発送などは行わず、非常にシンプルな処理を行っていて、カタログのコーヒーバッグの数量を減少させるだけの処理としています。

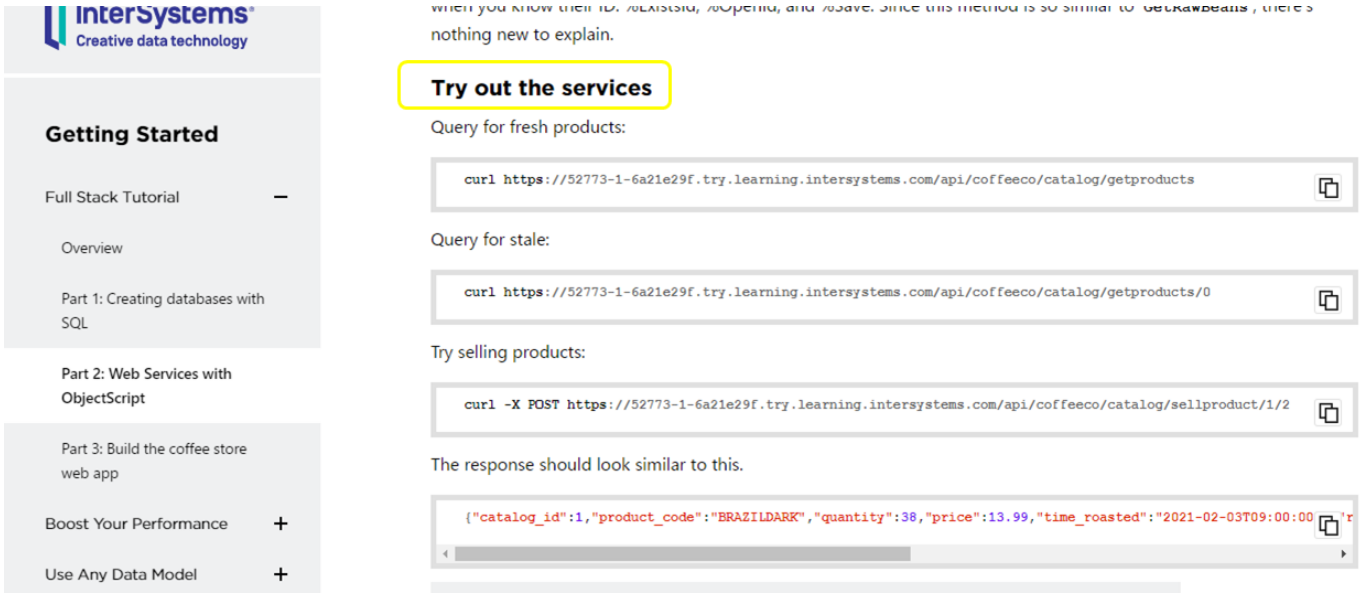
また、お客様が複数の商品を購入した場合、クライアントはそれぞれの商品に対して SellProduct リクエストを送信するものとします。

GetRawBeans() メソッドの処理と同様に、レコード ID がわかっている場合にレコードを照会する ObjectScript の便利なメソッド %ExistsId()、%OpenId()、%Save() を利用します。このメソッドは GetRawBeans() と非常によく似ているため、新たに追加する説明はありません。

サービスを試してみます。

実行に使用する URL の Web サーバアドレスは Sandbox 毎に異なります。

https://gettingstarted.intersystems.com/full-stack/part-two-rest-services/#catalog-services を開き、「**Try out the services**」の近くの表示をご確認ください。



- 新鮮なコーヒー豆を問い合わせるサービス

```
curl https://52773-1-4e734fe2.try.learning.intersystems.com/api/coffeeco/catalog/getproducts
```

- セール商品を問い合わせるサービス（末尾のパラメータに 0 を指定しています）

```
curl https://52773-1-4e734fe2.try.learning.intersystems.com/api/coffeeco/catalog/getproducts/0
```

- 商品を販売します

```
curl -X POST https://52773-1-4e734fe2.try.learning.intersystems.com/api/coffeeco/catalog/sellproduct/1/2
```

この実行結果の例は以下の通りです。

```
{"catalog_id":1,"product_code":"BRAZILDARK","quantity":38,"price":13.99,"time_roasted":"2021-02-03T09:00:00Z","roasting_notes":"Full bodied and low acidity. Thick, creamy, nutty and semi-sweet.","img":"brazil_dark.jpg"}
```

パート3：コーヒーストア用 Web アプリの構築

オリジナルページはこちら　https://gettingstarted.intersystems.com/full-stack/part-three-front-end/

Introduction

このチュートリアルの最後のパート 3 では、あなたが運営するコーヒーショップのオンライン storefront を作成します。

アプリケーションでは、Vue.js という JavaScript
フレームワークを使用して、シングルページウェブアプリケーション（SPA）を作成しています。

Vue.js についての説明はこのチュートリアルの範囲外ですが、Vue.js を使用してどのように Web アプリケーションが構築されるか、また、パート2 で作成した REST サービスがこのアプリでどのように使用されているかを確認することができます。

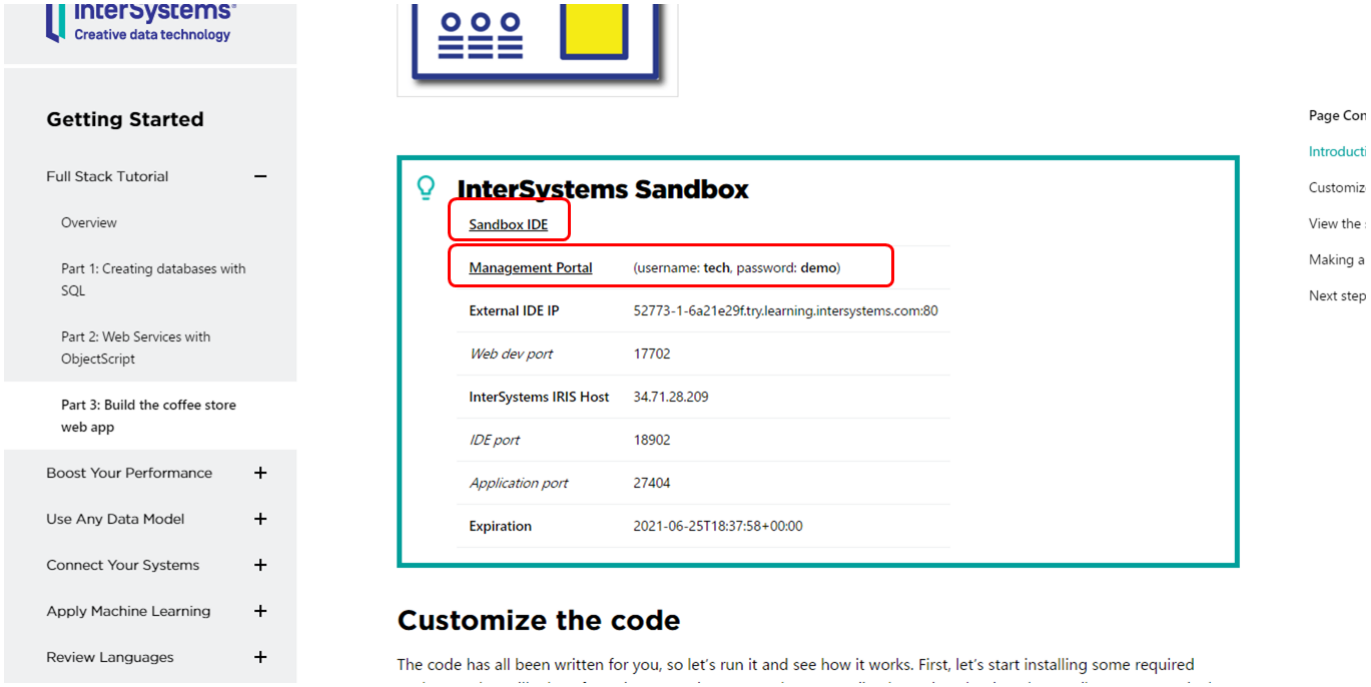
コードをカスタマイズする

Web アプリケーション用コードは全て準備済です。実際に動かして動作を確認してみましょう。

まず、必要なパッケージのインストールを行います。インストールには数分かかります。また、インストール後にいくつかの編集を行います。

Sandbox の IDE のターミナルウィンドウで以下のコマンドを入力してください。

IDE へのリンク情報は、https://gettingstarted.intersystems.com/full-stack/part-three-front-end/#customize-code を開き、ログインすると以下図のように表示されます。



IDE
などのアクセス情報の表示が、 Open the IDE (setting requires sandbox - click here)

のように表示されていたら、ピンク色のリンクをクリックしてください。

IDE のターミナルウィンドウで以下実行してください。インストールには数分時間がかかります。

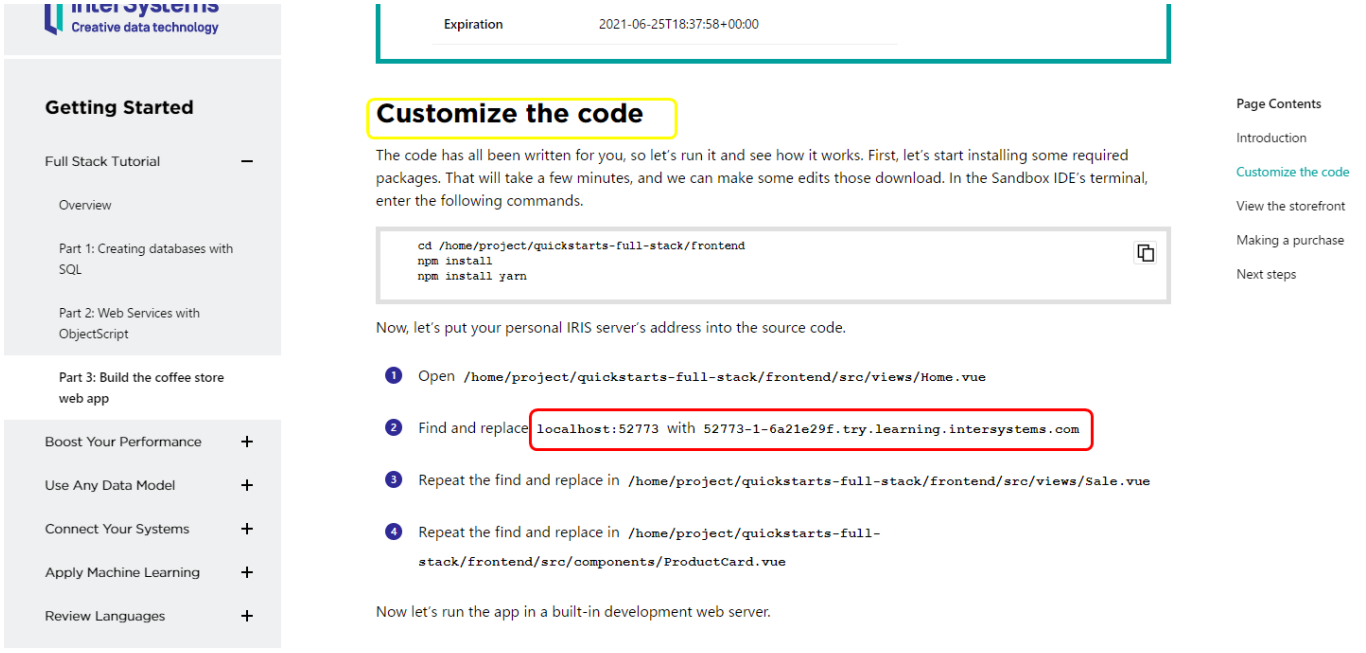
```
cd /home/project/quickstarts-full-stack/frontend
npm install
npm install yarn
```

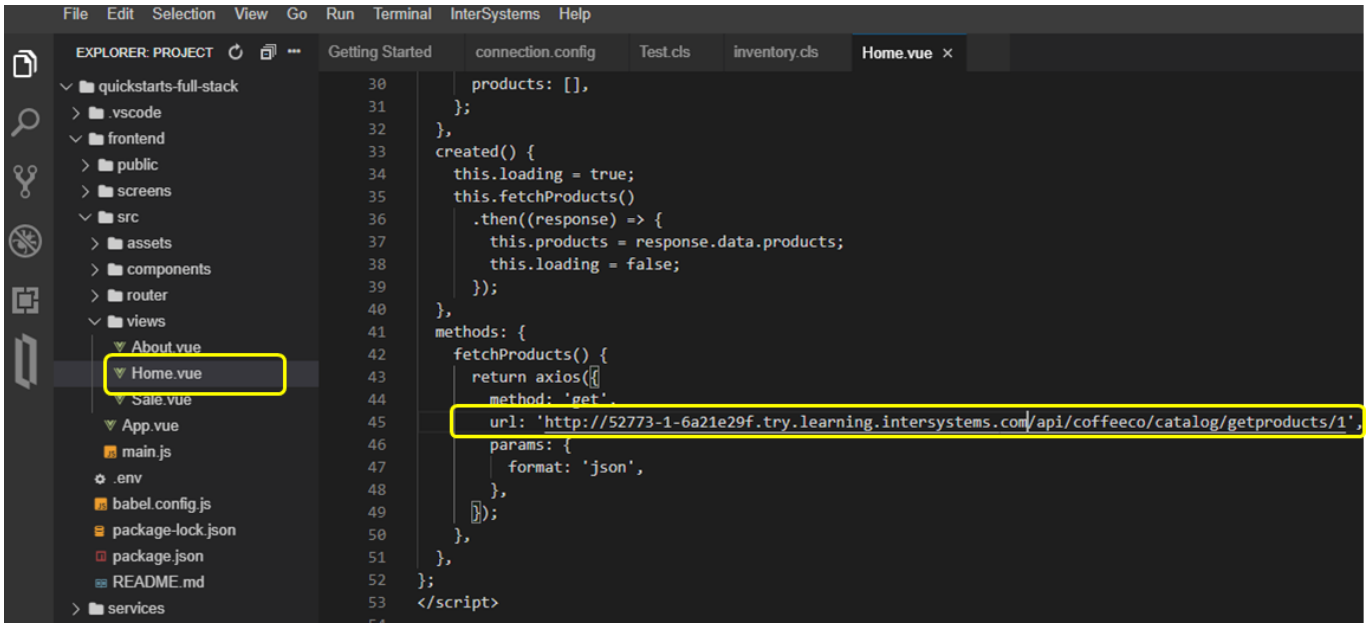
インストールが完了したら、お使いの IRIS サーバのアドレスをソースコードに指定します。

(1) IDE で quickstarts-full-stack > frontend > src > views > Home.vue を開きます。

(2) localhost:52773 と記載されている部分（url の設定）を、お使いの IRIS
用アドレスに書き換えます（例：52773-1-4e734fe2.try.learning.intersystems.com）。

https://gettingstarted.intersystems.com/full-stack/part-three-front-end/#customize-code を開き、「**Customize the code**」の近くにアクセス情報が表示されます。ご確認ください。





(3) IDE で quickstarts-full-stack > frontend > src > views > Sale.vue を開き、(2)と同様に、IRIS の接続情報を書き換えてください。

(4) IDE で quickstarts-full-stack > frontend > src > components > ProductCard.vue を開き、orderurl の設定を (2) と同様に、IRIS の接続情報を書き換えてください。

では、このアプリのテスト用の組み込み Web サーバで実行してみましょう。

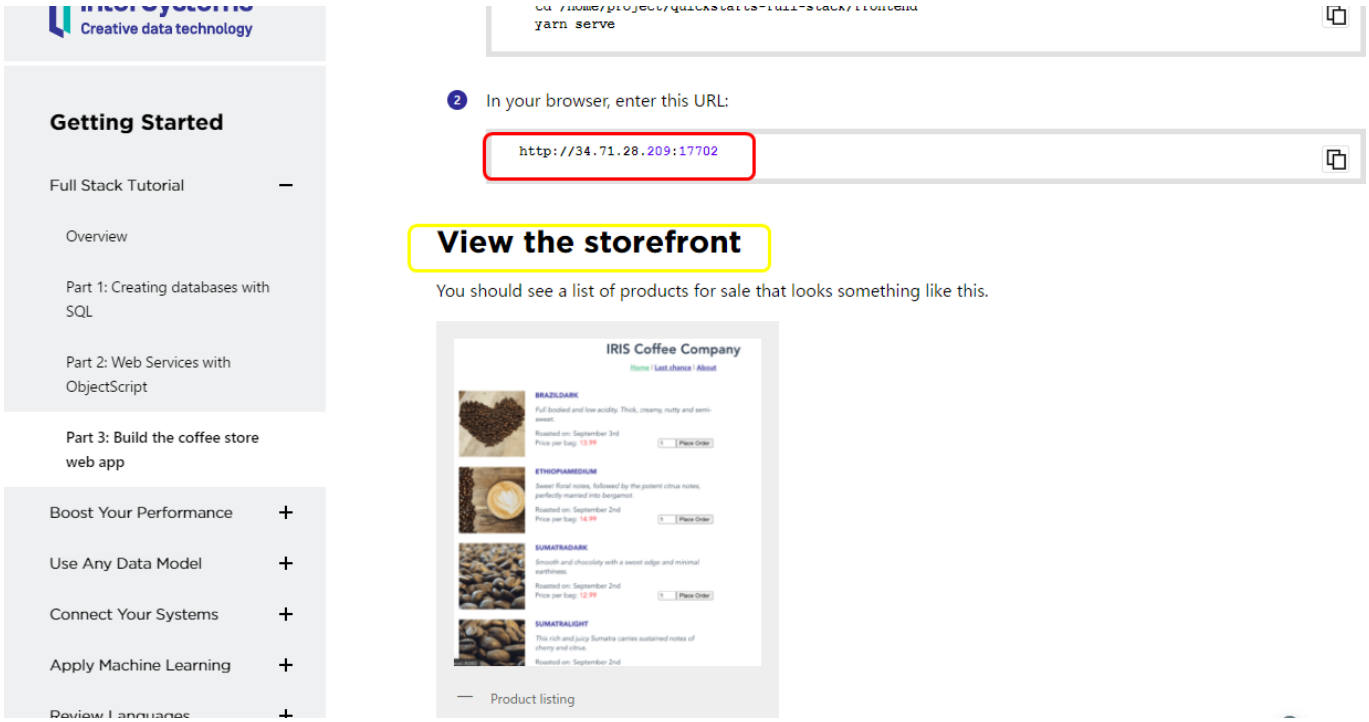
(1) Sandbox の IDE のターミナルウィンドウで以下実行します（実行には時間がかかります）。

```
cd /home/project/quickstarts-full-stack/frontend
yarn serve
```

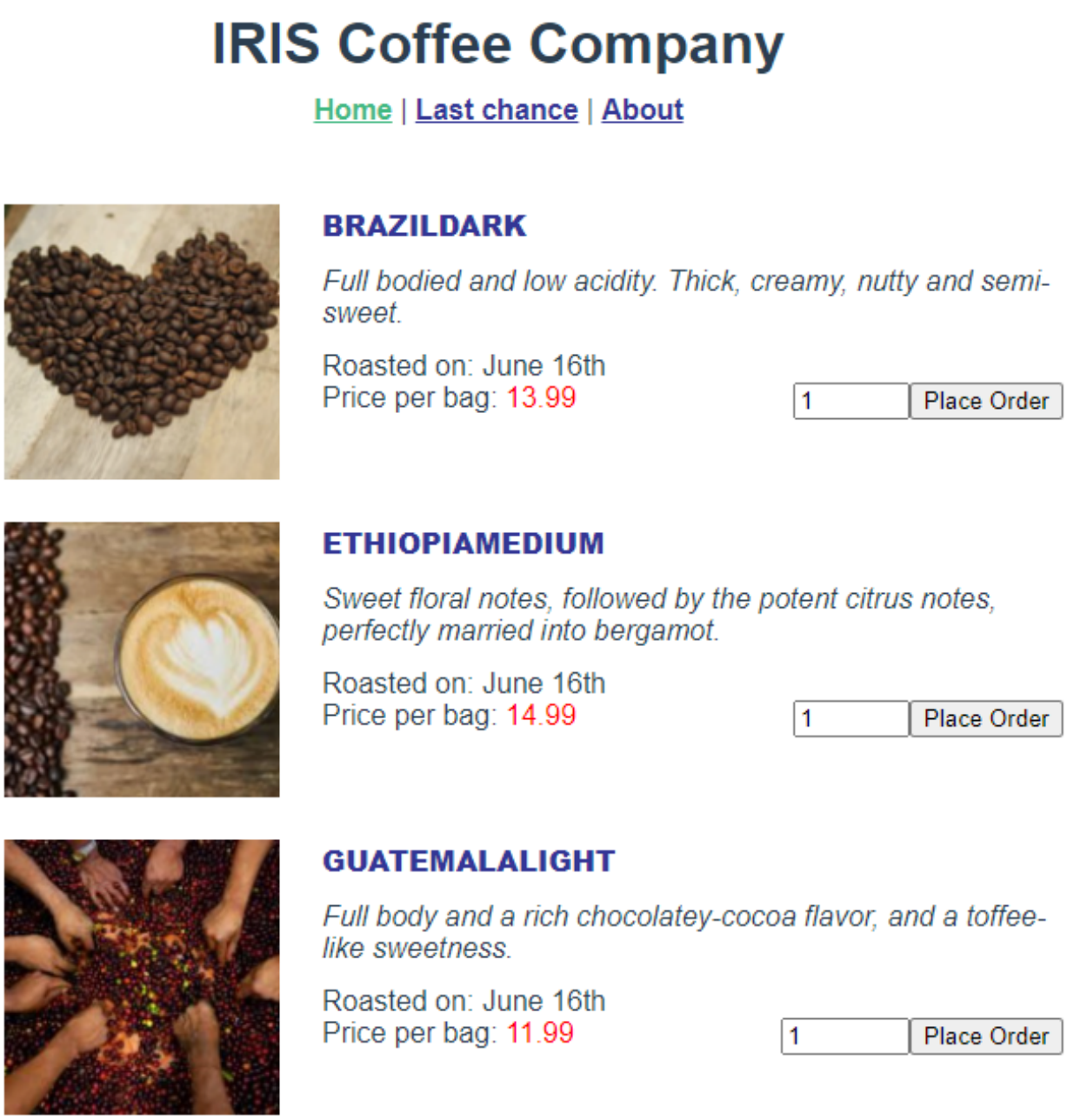
```
theia /home/project/quickstarts-full-stack/frontend $   cd /home/project/quickstarts-full-stack/frontend
theia /home/project/quickstarts-full-stack/frontend $   yarn serve
yarn run v1.22.4
$ vue-cli-service serve
 INFO  Starting development server...
98% after emitting CopyPlugin

 WARNING  Compiled with 1 warning

Module Warning (from ./node_modules/eslint-loader/index.js):

/home/project/quickstarts-full-stack/frontend/src/components/ProductCard.vue
  52:15  warning  Unexpected alert  no-alert
  54:15  warning  Unexpected alert  no-alert

✖ 2 problems (0 errors, 2 warnings)


You may use special comments to disable some warnings.
Use // eslint-disable-next-line to ignore the next line.
Use /* eslint-disable */ to ignore all warnings in a file.

  App running at:
  - Local:   http://localhost:4200/

  It seems you are running Vue CLI inside a container.
  Access the dev server via http://localhost:<your container's external mapped port>/

  Note that the development build is not optimized.
  To create a production build, run npm run build.
```

(2) ブラウザに、パート3の画面で指定された URL にアクセスしてください。

URL は https://gettingstarted.intersystems.com/full-stack/part-three-front-end/#store-view を開き、「**View the storefront**」の近くに表示されます。



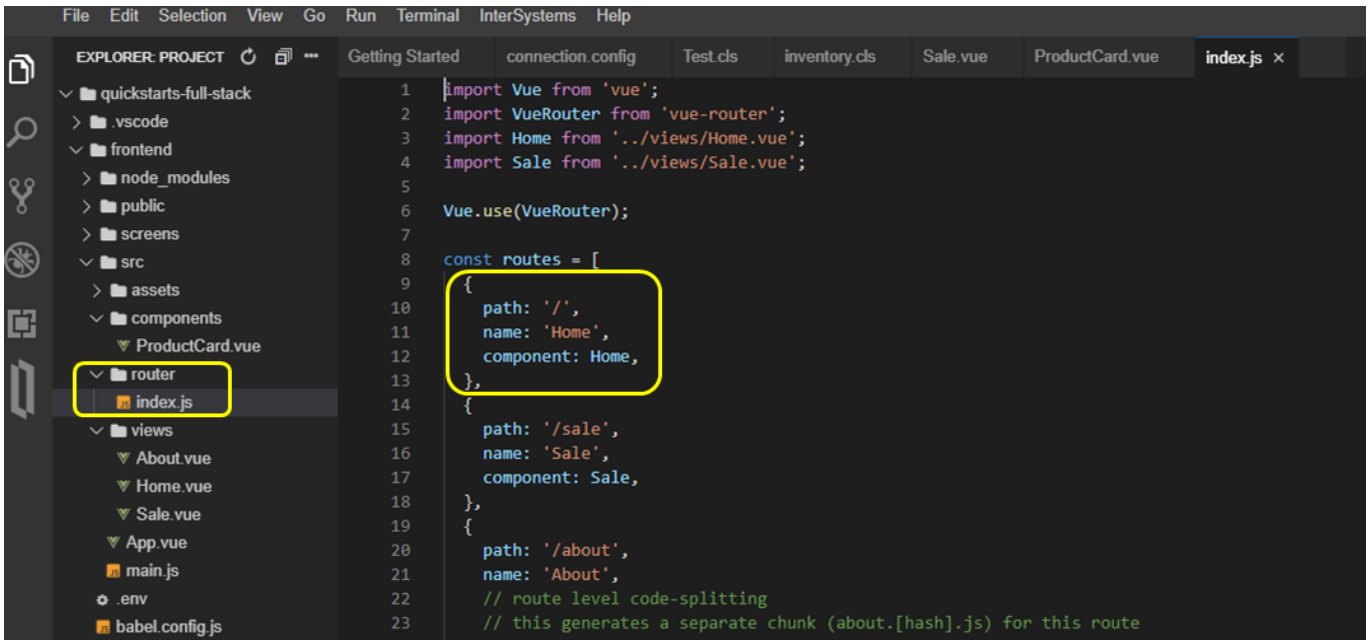
URL にアクセスすると、以下の画面が表示されます。



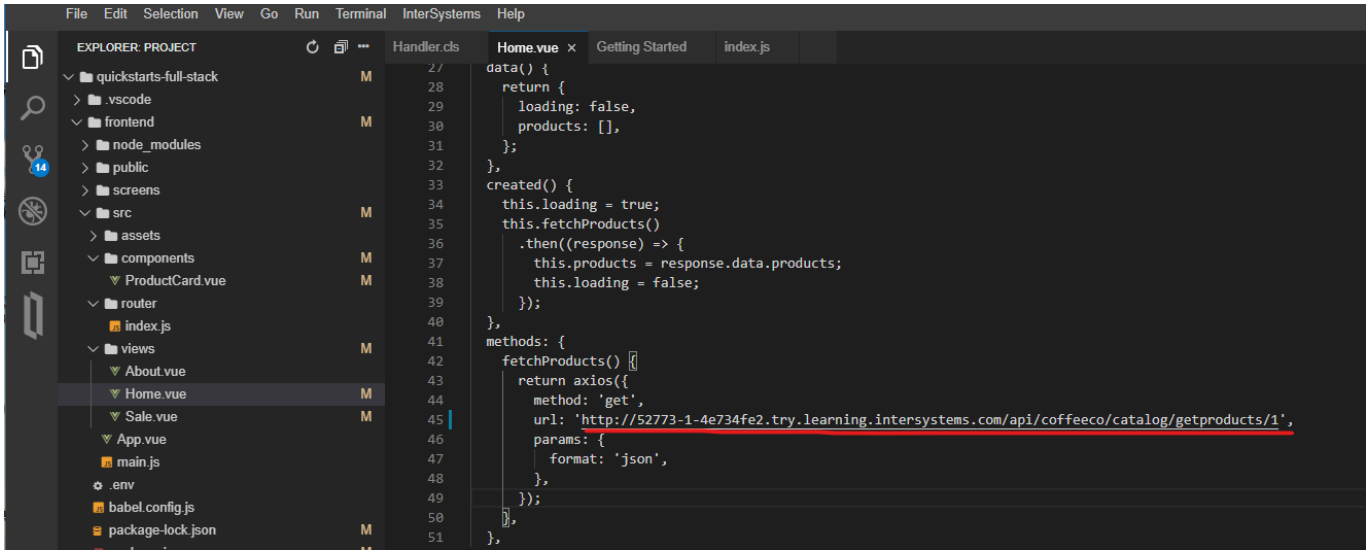
Vue.js の中で、どのように実行されているか少し解説します。

React や Angular などの多くのフレームワークと同様に、URLはコンポーネントに「ルーティング」されます。

IDE で、quickstarts-full-stack > frontend > src > router > index.js を参照すると、デフォルトの URL パスである「/」が Home コンポーネントにルーティングされています。



Webアプリケーションで「/」 が指定されると、quickstarts-full-stack > frontend > src > views > Home.vue が表示され、/catalog/getproducts/1 を GET 要求で実行しています。



IRIS では、quickstarts-full-stack > services > cls > ICO > Handler.cls の GetProducts() メソッドが引数に 1 を指定された状態で実行され（＝新鮮な豆を取得）
過去5日間に焙煎されたすべての販売商品のリストをJSONで返します。

URL のパラメータに 1 を渡すか、指定しない場合、焙煎されたばかりのバッグのみが要求されます

```
GET https://52773-1-4e734fe2.try.learning.intersystems.com/api/coffeeco/catalog/getproducts/1
```

実行結果例は以下の通りです。

```
{
    "rowcount": 5,
    "products": [{
        "catalog_id": "1",
```

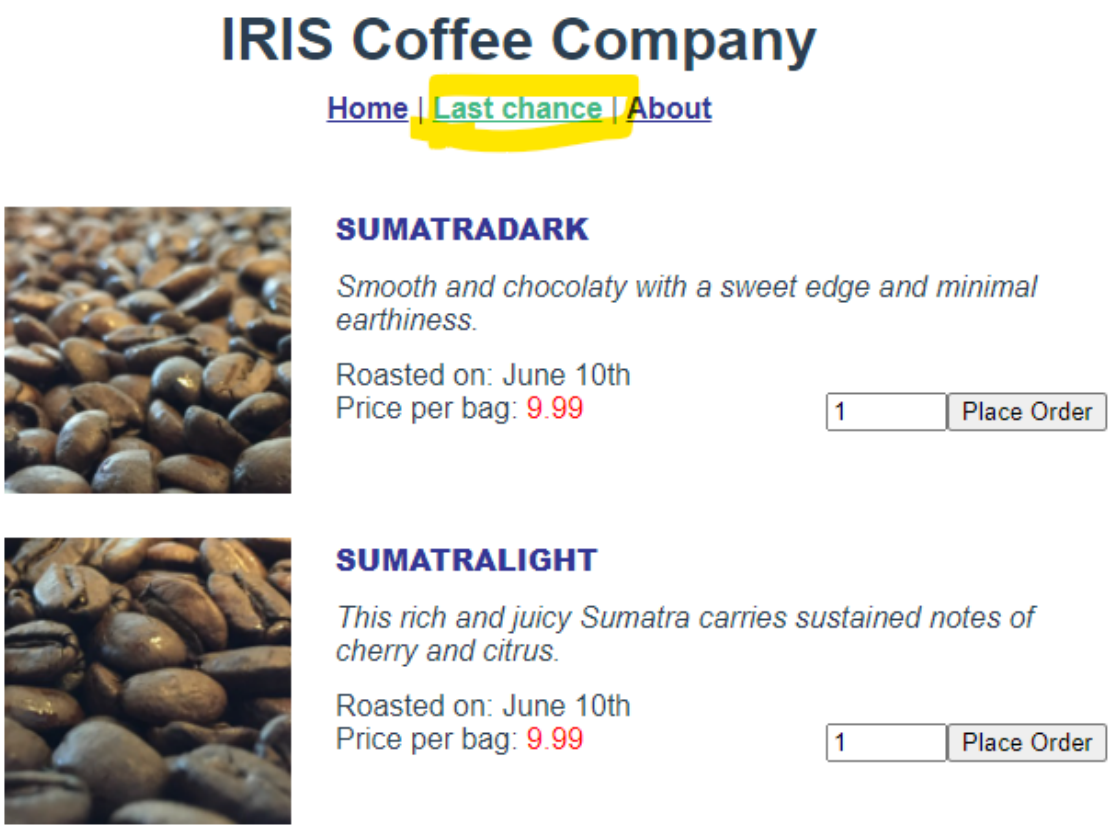
```
        "product_code": "BRAZILDARK",
        "quantity": 38,
        "time_roasted": "2021-02-09 09:00:00",
        "roasting_notes": "Full bodied and low acidity. Thick, creamy, nutty and semi-
sweet.",
        "img": "brazil_dark.jpg",
        "price": 13.99
    }, {
        "catalog_id": "2",
        "product_code": "ETHIOPIAMEDIUM",
        "quantity": 40,
        "time_roasted": "2021-02-08 09:00:00",
        "roasting_notes": "Sweet floral notes, followed by the potent citrus notes, p
erfectly married into bergamot.",
        "img": "ethiopia_medium.jpg",
        "price": 14.99
    }, {
        "catalog_id": "3",
        "product_code": "GUATEMALALIGHT",
        "quantity": 120,
        "time_roasted": "2021-02-09 17:30:00",
        "roasting_notes": "Full body and a rich chocolatey-cocoa flavor, and a toffee-
like sweetness.",
        "img": "guatemala_light.jpg",
        "price": 11.99
    }, {
        "catalog_id": "4",
        "product_code": "SUMATRADARK",
        "quantity": 80,
        "time_roasted": "2021-02-07 13:01:30",
        "roasting_notes": "Smooth and chocolaty with a sweet edge and minimal earthin
ess.",
        "img": "sumatra_dark.jpg",
        "price": 12.99
    }, {
        "catalog_id": "5",
        "product_code": "SUMATRALIGHT",
        "quantity": 40,
        "time_roasted": "2021-02-07 09:00:00",
        "roasting_notes": "This rich and juicy Sumatra carries sustained notes of che
rry and citrus.",
        "img": "sumatra_light.jpg",
        "price": 12.99
    }]
}
```


REST サービスの処理詳細については、quickstarts-full-stack > services > cls > ICO > Handler.cls の GetProducts() メソッドをご参照ください。

Home.vue は、JSON オブジェクトを繰り返し処理して、JSON 内の各アイテムに対して ProductCard (quickstarts-full-stack > frontend > src > components > ProductCard.vue） を作成します。

ProductCard.vue は、単一の商品を表示し、それを注文するための UI を作成する方法を知っているだけです。

では、Web ページの「**Last chance**」のリンクをクリックしてみましょう。



製品の短いリストが表示されるはずです（表示されない場合は、ICO.catalog テーブルの timeroasted の値をいくつか変更して、5日以上前の値にする必要があります）。

この表示は、同じバックエンドのサービスを呼び出していますが、5日以上前の焙煎珈琲を返すようにパラメータを指定しています（1 以外の任意の数字を渡すことで指定できます）。

REST の呼び出しは以下の通りです。

```
GET /api/coffeeco/catalog/getproducts/2
```

応答結果例は、以下の通りです。

```
{
    "rowcount": 1,
    "products": [{
        "catalog_id": "10",
        "product_code": "SUMATRALIGHT",
        "quantity": 40,
        "time_roasted": "2021-02-02 09:00:00",
        "roasting_notes": "This rich and juicy Sumatra carries sustained notes of che
rry and citrus.",
        "img": "sumatra_light.jpg",
        "price": 12.99
    }]
```

```
}
```

Home.vue と同様に、Sale.vue でも ProductCard
コンポーネントを使用して商品を表示していますが、fetchProducts()
関数の実行結果（＝RESTの応答）から ProductCard にデータを渡す前に価格を 3 ドル分値引きしています。

これは、コードを単純化するためにコンポーネントを再利用する簡単な例であり、価格設定と在庫管理を分離していることを示しています。

購入処理

いよいよチュートリアルの最後の項目です！コーヒーを買ってみましょう！

商品の数量を変更し「Place Order」ボタンをクリックします。注文が完了したことを示すポップアップが表示されるはずです（チュートリアルは架空サンプルです。実際のアプリであれば、ショッピングカートに商品を入れて、顧客が支払いを済ませるまで注文は行われないでしょう）。

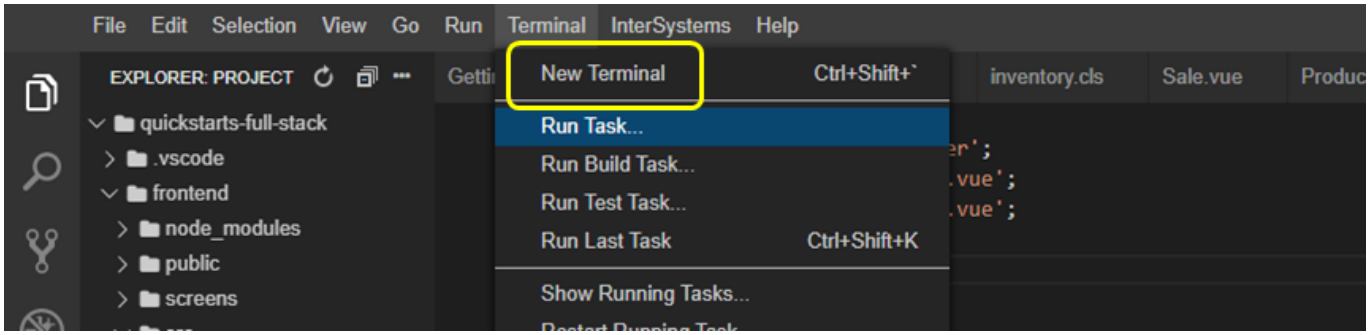
チュートリアルは簡単な例としているため、ProductCard が RESTサービスの SellProduct()
メソッドを呼び出し、注文されたコーヒー豆をカタログから取り出す様子を示しています。

REST 呼び出しは以下の通りです。

```
POST /api/coffeeco/catalog/sellproduct/????????ID/???????????
```

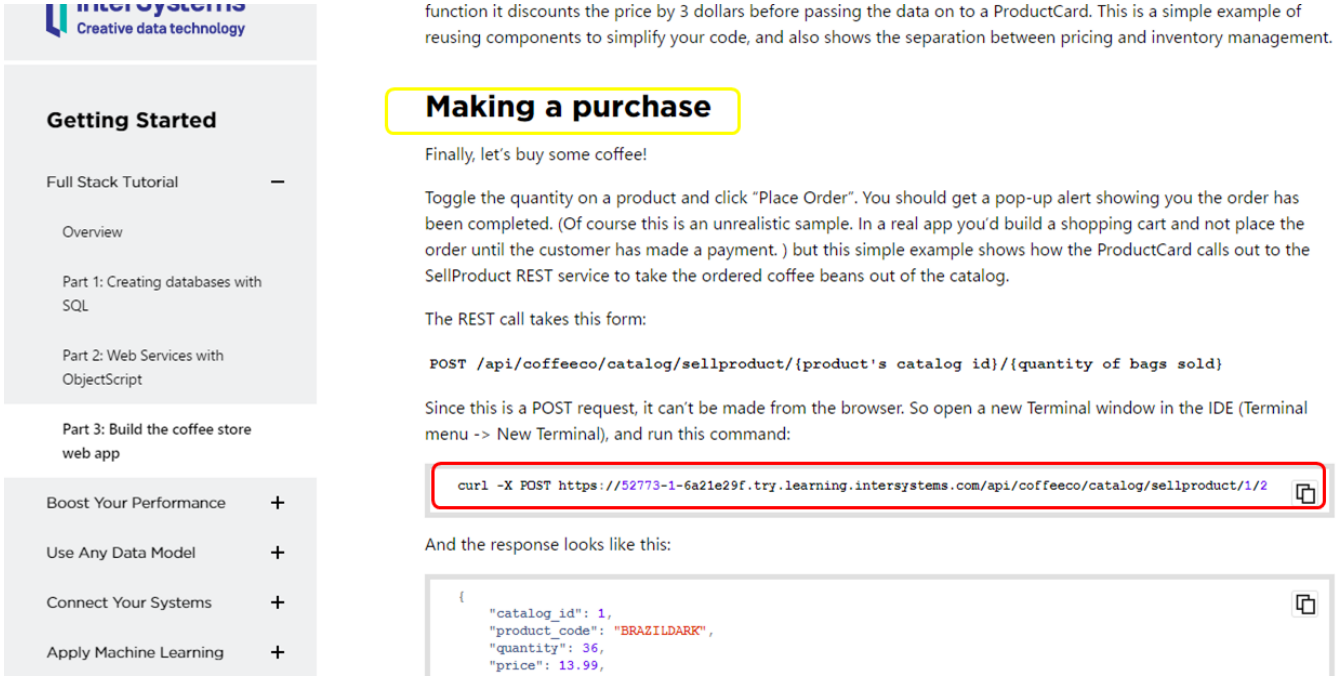
POST要求なので、ブラウザからは実行できません。

IDE のターミナルを新規に開きます（Terminal > New Terminal）。

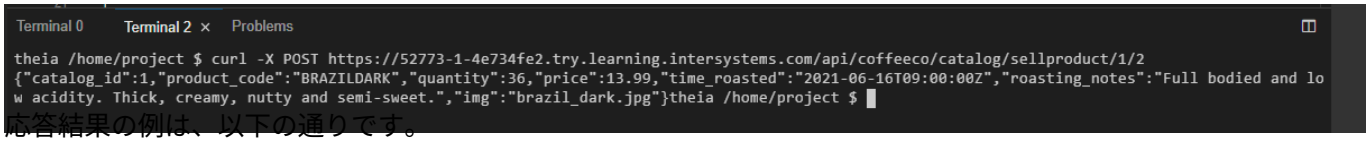


以下コマンドを新規ターミナルで実行してください。

コマンド実行例は、https://gettingstarted.intersystems.com/full-stack/part-three-front-end/#purchase
を開き、「**Making a purchase**」近くの表示をご確認ください。




```
curl -X POST https://52773-1-4e734fe2.try.learning.intersystems.com/api/coffeeco/catalog/sellproduct/1/2
```



応答結果の例は、以下の通りです。

```
{
    "catalog_id": 1,
    "product_code": "BRAZILDARK",
    "quantity": 36,
    "price": 13.99,
    "time_roasted": "2021-02-09T09:00:00Z",
    "roasting_notes": "Full bodied and low acidity. Thick, creamy, nutty and semi-
sweet.",
    "img": "brazil_dark.jpg"
}
```

ここでは、もう少しアプリケーションを使って遊んでいただく方法をご紹介します。

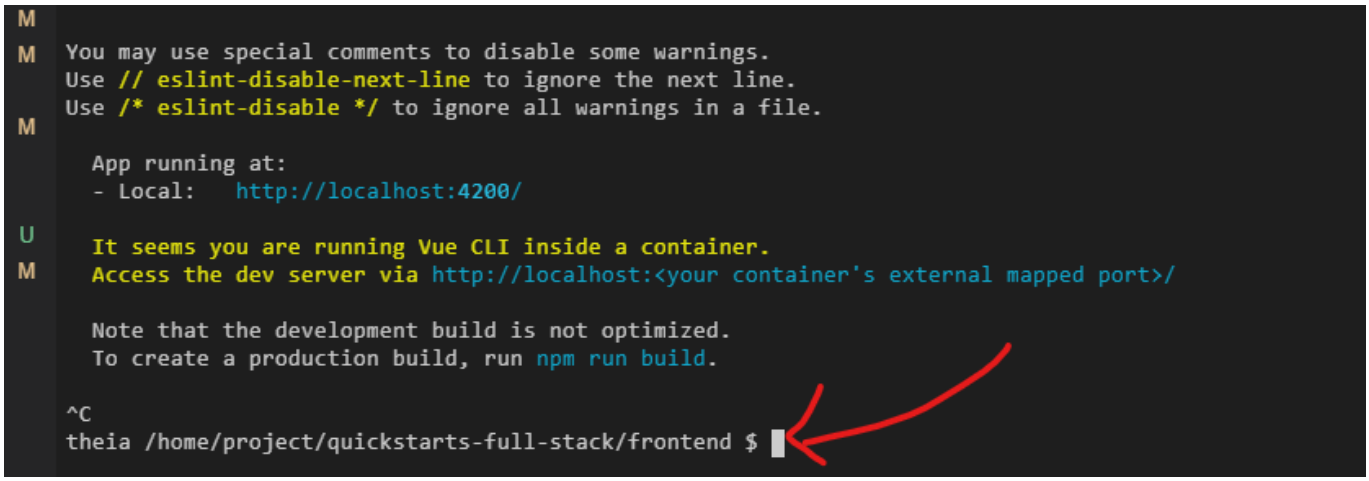
(1) 在庫が足りなくなるまで、コーヒーバッグを注文し続けます。最終的にはStorefront
から商品を消滅させることができる予定です。

(2) Web 開発の経験がある方は、quickstarts-full-stack > frontend > src > components >ProductCard.vue
コンポーネントの CSS を変更してみてください （ファイルの最後の <style> セクションにあります）。

(3) Vue.js の経験があれば、ProductCard.vue の processOrder() 関数で使われている基本的な JavaScript の警告メッセージを、もっと面白いものに変更してみましょう。また、独自のコンポーネントを作成して、ポップアップ・アラートの代わりにこの情報を表示することもできます。

最後に、Webアプリケーションの終了方法をご案内します。

Sandbox の IDE のターミナルウィンドウがプロンプトが戻っていない状態になっています。Web アプリケーションを終了して良い場合は、Ctrl ＋ C を実行し、元のプロンプトに戻してください。



Next Steps

InterSystems IRIS の基本的な機能をご紹介しましたが、いかがでしたでしょうか。

このチュートリアルで学習したことを振り返ります。

パート1では、テーブル作成やデータの読み込みに SQL を使用し、直接ターミナルから SQL を入力したり、Python プログラムから実行する方法を確認しました。

パート2では、高速で柔軟なデータベースプログラミング言語である ObjectScript のご紹介と、ObjectScript を使用して RESTサービスを構築する方法をご紹介します。

パート3では、人気の高い JavaScript フレームワークを使って、顧客向けのフロントエンド Web アプリを構築する方法を学習しました。

以上でフルスタックチュートリアルは終了です！

最後までお付き合いいただきありがとうございました！

https://gettingstarted.intersystems.com/
には、まだまだ他のチュートリアルをご用意しています。ぜひご体験ください！

#初心者 #InterSystems IRIS

ソースURL:https://jp.community.intersystems.com/post/%E3%80%90gettingstarted-iris%E3%80%91%E3%83%81%E3%83%A5%E3%83%BC%E3%83%88%E3%83%AA%E3%82%A2%E3%83%AB%E3%82%92%E5%A7%8B%E3%82%81%E3%82%88%E3%81%86%EF%BC%81%E3%81%9D%E3%81%AE1%EF%BC%9Afull-stack-%E3%83%81%E3%83%A5%E3%83%BC%E3%83%88%E3%83%AA%E3%82%A2%E3%83%AB