Article
Shintaro Kaminaka · May 1, 2020 14m read

仕様ファースト(APIファースト)アプローチによるREST APIの開発

この記事では、REST API開発への仕様ファーストアプローチについて説明します。

従来のコードファーストREST API開発は次のようになります。

- コードを書く
- RESTを有効にする
- ドキュメント化（REST APIとして）

仕様ファーストのアプローチでは同じ手順を行いますが、順序が逆になります。 ドキュメントを兼ねた仕様書を作成し、そこからRESTアプリの定型文を生成して、最後にビジネスロジックを書きます。

これは、次の理由でメリットがあります。

- REST APIを使用したいと思っている外部開発者またはフロントエンド開発者向けの関連性のある有用なドキュメントが常に入手できます。
- OAS（Swagger）で作成された仕様をさまざまなツールにインポートして、編集、クライアント生成、API管理、ユニットテスト、その他の多くのタスクの自動化または簡略化を行うことができます。
- 改善されたAPIアーキテクチャ。コードファーストアプローチではAPIはメソッドごとに開発されるため、開発者はAPIアーキテクチャ全体を簡単に見落としてしまう可能性があります。これに対し、仕様ファーストの開発者は通常、APIの消費者の立場としてAPIと対話するように強制されます。これは、よりクリーンなAPIアーキテクチャの設計に役立ちます。
- 開発の迅速化。すべての定型的なコードが自動的に生成されるため、コードを記述する必要はありません。後は、ビジネスロジックの開発をするだけです。
- より高速なフィードバックループ。消費者はAPIをすぐに見ることができ、仕様を修正するだけで簡単に提案を行うことができます。

仕様ファーストのアプローチでAPIを開発しましょう！

計画

1. Swaggerで仕様を開発する

- Docker
- ローカルで
- オンラインで

2. IRISに仕様を読み込む

- API管理REST API
- ^%REST
- クラス

3. IRISに読み込んだ仕様を確認する
4. 実装
5. さらなる開発
6. 考慮事項

- 特別なパラメータ
- CORS

7. IAMに仕様を読み込む

仕様を作成する

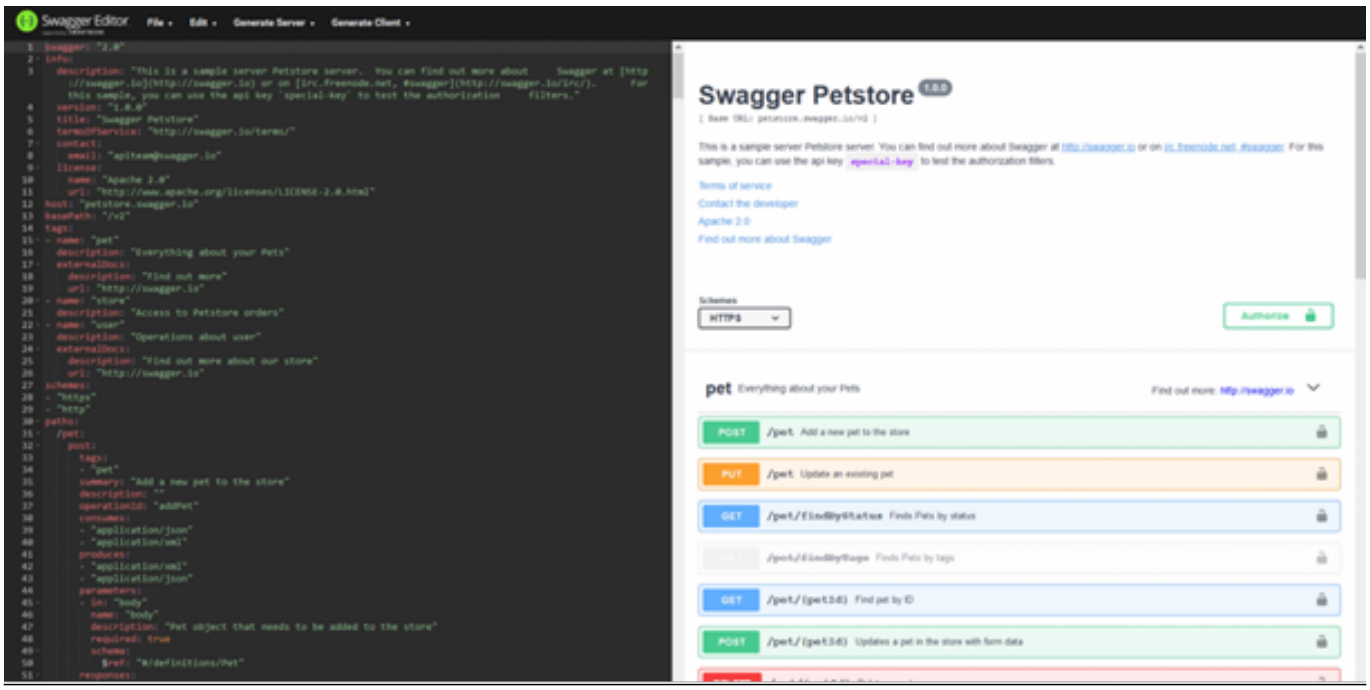
最初のステップは、当然のことながら仕様を作成することです。 InterSystems IRISはOpen API仕様（OAS）をサポートしています。

> **OpenAPI仕様** （以前のSwagger仕様）は、REST APIのAPI記述形式です。
> OpenAPIファイルを使用すると、以下を含むAPI全体を記述することができます。
>
> - 利用可能なエンドポイント（/users）および各エンドポイントでの操作（GET /users, POST /users)
> - 各操作の操作パラメータ入出力
> - 認証方法
> - 連絡先情報、ライセンス、使用条件およびその他の情報。
>
> API仕様はYAMLまたはJSONで記述できます。 この形式は学びやすく、人間と機械の両方が読み取れるようになっています。 完全OpenAPI仕様はGitHubにあります。OpenAPI 3.0仕様
>
> - Swagger docs.から

Swaggerを使用してAPIを記述します。 Swaggerを使用する方法はいくつかあります。

- オンライン
- Docker：docker run -d -p 8080:8080 swaggerapi/swagger-editor
- ローカルインストール

Swaggerをインストールまたは実行すると、次のウィンドウがWebブラウザーに表示されます。



左側では、API仕様を編集し、右側では、レンダリングされたAPIドキュメント／テストツールをすぐに確認できます。

最初のAPI 仕様をその中にロードします（YAML）。
これは、1つのGETリクエストを使用したシンプルなAPIで、指定された範囲の乱数を返します。

数学API仕様

構成は次のとおりです。

APIおよび使用されているOASバージョンに関する基本情報。

```
swagger: "2.0"
info:
  description: "Math"
  version: "1.0.0"
  title: "Math REST API"
```

サーバーホスト、プロトコル（http、https）、およびWebアプリケーション名は次のとおりです。

```
host: "localhost:52773"
basePath: "/math"
schemes:
  - http
```

次に、パス（完全なURLは http://localhost:52773/math/random/:min/:max
）とHTTPリクエストメソッド（get、post、put、delete）を指定します。

```
paths:
  /random/{min}/{max}:
    get:
```

その後、リクエストに関する情報を指定します。

```
     x-ISC_CORS: true
      summary: "Get random integer"
      description: "Get random integer between min and max"
      operationId: "getRandom"
      produces:
      - "application/json"
      parameters:
      - name: "min"
        in: "path"
        description: "Minimal Integer"
        required: true
        type: "integer"
        format: "int32"
      - name: "max"
        in: "path"
        description: "Maximal Integer"
        required: true
```

```
        type: "integer"
        format: "int32"
      responses:
        200:
          description: "OK"
```

この部分では、リクエストを定義します。

- CORSに対してこのパスを有効にします（詳細は後述します）
- 概要と説明を提供します。
- operationId は、仕様内の参照を許可します。これはまた、実装クラスで生成されたメソッド名でもあります。
- produces - 応答フォーマット（text、xml、 jsonなど）
- parametersは、入力パラメータを指定します（URLまたは本文で）が、この場合は、2つのパラメータ、すなわち乱数ジェネレータの範囲を指定します。
- responsesは、サーバーから可能な応答をリストします。

ご覧のとおり、このフォーマットはそれほど難しいものではありません。利用可能な機能は他にもたくさんありますが、仕様は次のとおりです。

最後に、定義をJSONとしてエクスポートします。 ファイルに移動→ 変換してJSON形式で保存します。
仕様は次のようになります。

数学API仕様

IRISに仕様を読み込む

仕様が完成したので、InterSystems IRISでこのREST APIの定型的なコードを生成できます。

この段階に進むには、次の3つが必要です。

- RESTアプリケーション名：生成されたコードのためのパッケージ（たとえば math）
- JSON形式のOAS仕様：前のステップで作成しました
- WEBアプリケーション名：REST APIにアクセスするための基本パス（ /mathの場合）

コード生成に仕様を使用するには3つの方法があります。それらは基本的に同じ方法ですが、同じ機能にアクセスするためのさまざまな方法を提供します

1. ^%RESTルーチン (インタラクティブターミナルセッションで Do ^%RESTを実行する）を呼び出す、ドキュメント。
2. %RESTクラス（Set sc = ##class(%REST.API).CreateApplication(applicationName, spec)、非インタラクティブ)を呼び出す、 ドキュメント。
3. API管理のREST APIを使用する、ドキュメント。

ドキュメントには必要なステップが適切に記述されていると思うので、1つだけ選択してください。
2つの注意事項を追加します。

- (1) と (2) の場合、動的オブジェクトにファイル名またはURLを渡すことができます。
- (2) と
  (3)の場合、Webアプリケーションを作成するため、次のように追加の呼び出しをする必要があります。

  ```
  set sc = ##class(%SYS.REST).DeployApplication(restApp, webApp, authenticationTy
  pe)
  ```

  なので、今回のケースでは以下のようになります。

  ```
  set sc = ##class(%SYS.REST).DeployApplication("math", "/math")
  ```

  authenticationType引数の値を%sySecurityインクルードファイルから取得、関連エントリは$$$Authe*なので、認証されていない アクセスパスの場合は $$$AutheUnauthenticated。
  省略した場合、パラメーターはデフォルトでパスワード認証になります。

IRISに読み込んだ仕様を確認する

アプリを正常に作成した場合、次の3つのクラスをもつ新しいmathパッケージが作成されているはずです。

- Spec - 仕様をそのまま保存します
- Disp - RESTサービスが呼び出されたときに直接呼び出されます。
  REST処理をラップし、実装メソッドを呼び出します。
- Impl - RESTサービスの実際の内部実装を保持します。 このクラスのみを編集する必要があります。

クラスの詳細に関するドキュメント 。

実装

最初は、実装クラスmath.implには/random/{min}/{max}操作に対応する1つのメソッドのみが含まれます。

```
/// Get random integer between min and max<br/>
/// The method arguments hold values for:<br/>
///     min, Minimal Integer<br/>
///     max, Maximal Integer<br/>
ClassMethod getRandom(min As %Integer, max As %Integer) As %DynamicObject
{
    //(Place business logic here)
    //Do ..%SetStatusCode(<HTTP_status_code>)
    //Do ..%SetHeader(<name>,<value>)
    //Quit (Place response here) ; response may be a string, stream or dynamic object
}
```

小さな実装から始めましょう。

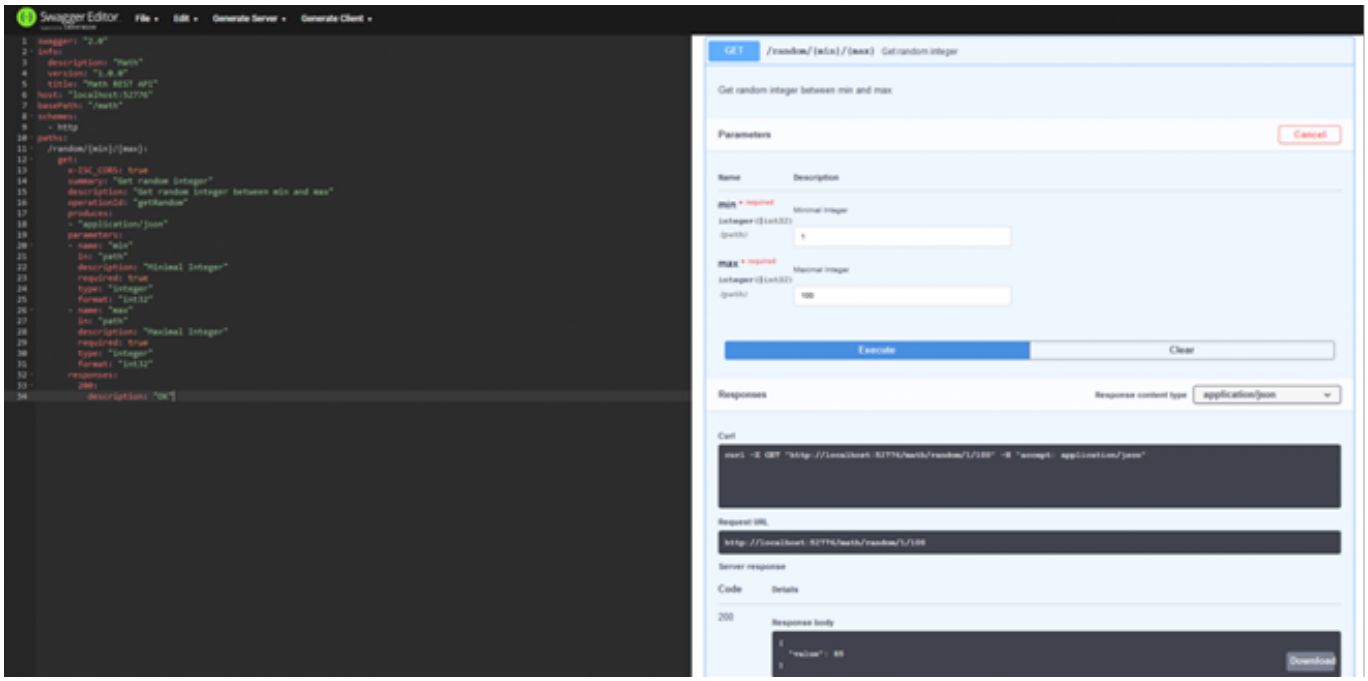
```
ClassMethod getRandom(min As %Integer, max As %Integer) As %DynamicObject
{
    quit {"value":($random(max-min)+min)}
}
```

そして最後に、ブラウザでこのページを開いてREST
APIを呼び出すことができます。 http://localhost:52773/math/random/1/100

出力は次のようになります。

```
{
    "value": 45
}
```

また、Swaggerエディターで試してみる
ボタンとリクエストパラメータを入力しても、同じリクエストが送信されます。



おめでとうございます！ 仕様ファーストアプローチで作成された最初のREST APIが公開されました。

さらなる開発

もちろん、APIは静的ではないので、新しいパスなどを追加する必要があります。 仕様ファースト開発では、先ず仕様を変更し、次にRESTアプリケーションを更新し（アプリケーションの作成と同じ呼び出し）、最後にコードを記述します。 仕様の更新は安全です。パスが仕様から削除されても、コードは影響を受けません。実装クラスでは、メソッドは削除されません。

考慮事項

追加の注意事項！

特別なパラメータ

InterSystems は、swagger仕様に特別なパラメータを追加しました。

名前	データ型	デフォルト	場所	説明
x-ISCDispatchParent	クラス名	%CSP.REST	情報	ディスパッチクラスのスーパークラス。
x-ISCCORS	ブーリアン	偽	操作	このエンドポイント/メソッドの組み合わせに対するCORSリクエ

				ストをサポートする必要があることを示すフラグ。
x-ISCRequiredResource	アレイ		操作	RESTサービスのこのエンドポイントへのアクセスに必要な定義済みリソースとそのアクセスモード（resource:mode）のカンマ区切りリスト。 例：["%Development:USE"]
x-ISCServiceMethod	文字列		操作	この操作を処理するためにバックエンドで呼び出されたクラスメソッドの名前。デフォルトはoperationIdで、これは通常適切です。

CORS

CORSサポートを有効にする方法は3つあります。

1. x-ISCCORS を真としてルートごとに指定する方法。 これは、私たちがMath REST APIで実行した方法です。

2. APIごとに追加する方法

```
Parameter HandleCorsRequest = 1;
```

そして、クラスを再コンパイルします。 また、仕様の更新後も存続します。

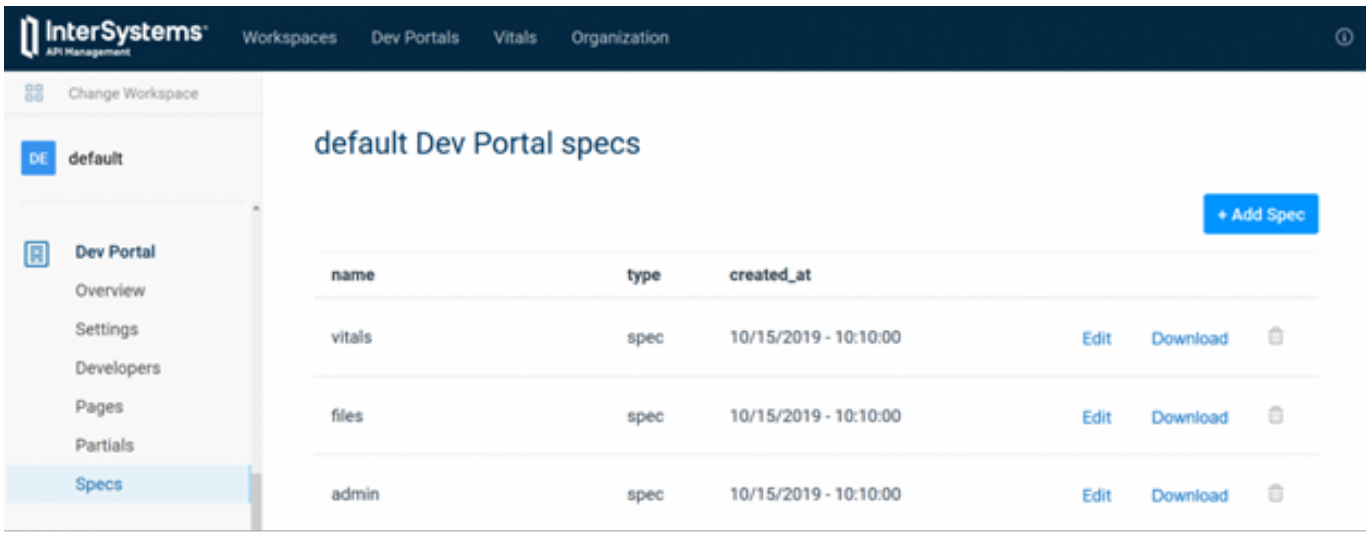
3. （推奨）APIごとにカスタムディスパッチャースーパークラス (%CSP.RESTを拡張する)
を実装し、CORS処理ロジックをそこに記述します。
このスーパークラスを使用するには、 x-ISCDispatchParent を仕様に追加します。

IAMに仕様を読み込む

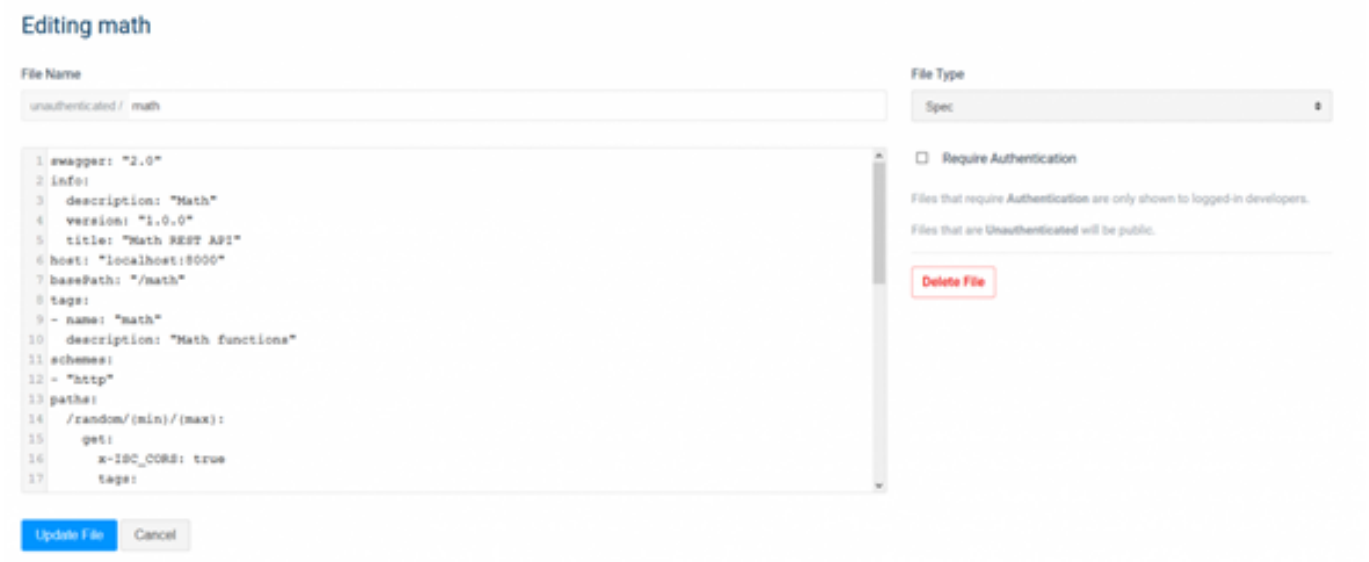
最後に、IAMに仕様を追加して、他の開発者向けに公開できるようにします。

IAMをまだ始めていない場合は、この記事を読んでみてください。 この記事はIAMを介したREST APIの提供についても説明しているため、ここでの説明は省きます。 InterSystems IRISインスタンスではなくIAMを指すようにするため、仕様のホストとベースパスパラメータを変更した方が良いかも知れません。

IAM管理者ポータルを開き、 関連するワークスペースで［仕様］タブに移動します。

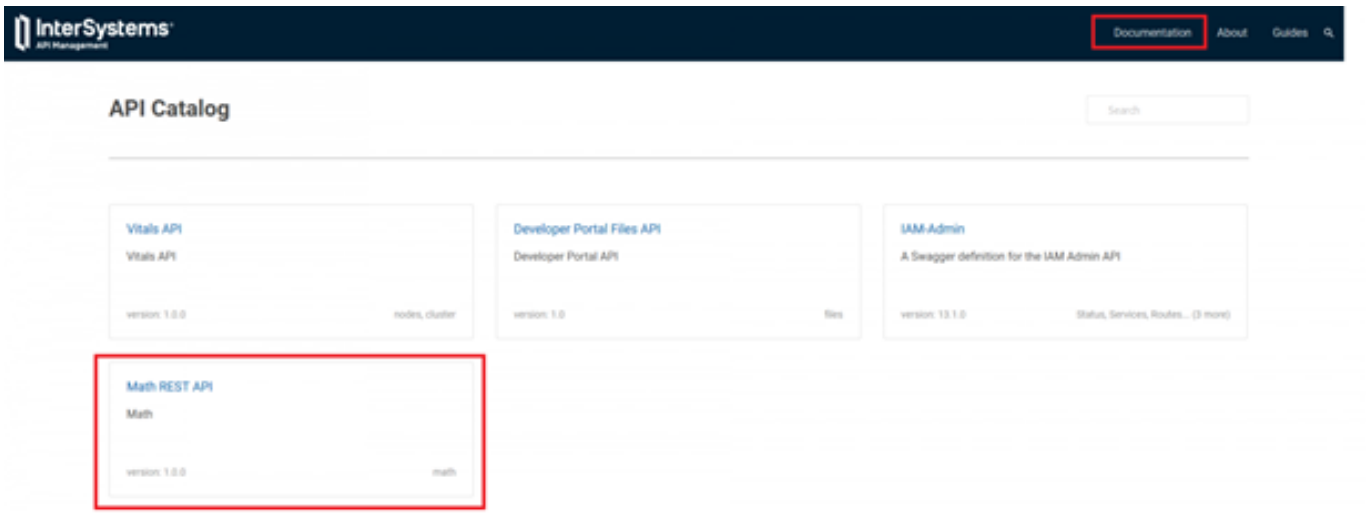


［仕様を追加］ボタンをクリックして、新しいAPIの名前を入力します(この例ではmath ）。
IAMで新しいSpecを作成した後、編集
をクリックして仕様コードを貼り付けます（JSONまたはYAML。IAMではどちらでも構いません）。

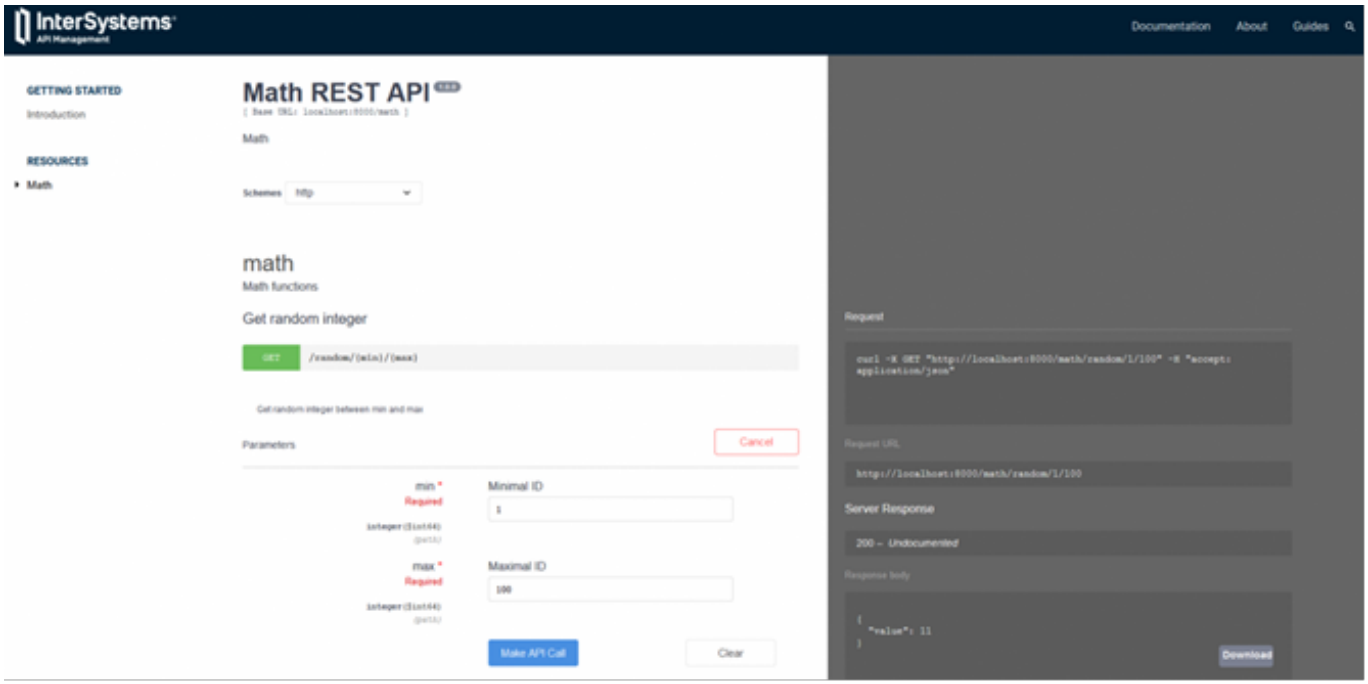


ファイルのアップデートをクリックするのを忘れないでください

APIが開発者向けに公開されました。 開発者ポータルを開いて、右上端にあるドキュメントをクリックします。
3つのデフォルトAPIに加えて、新しい Math REST APIが利用可能になっているはずです。



それを開きます。



開発者は、新しいAPIのドキュメントを参照して、同じ場所で試すことができます。

まとめ

InterSystems IRISは、REST APIの開発プロセスを簡素化し、仕様ファーストのアプローチにより、REST APIのライフサイクル管理をさらに迅速かつ容易にします。 このアプローチでは、クライアント生成、ユニットテスト、API管理など、さまざまな関連タスクに多様なツールを使用できます。

リンク

- OpenAPI 3.0 仕様
- RESTサービスの作成
- IAMで始める

- ICMドキュメント

また、前のパート「 InterSystems API管理を使用してAPIの負荷を分散する」も確認してください 。

#API #InterSystems API Manager (IAM) #REST API #InterSystems IRIS

Source URL:https://community.intersystems.com/node/476556