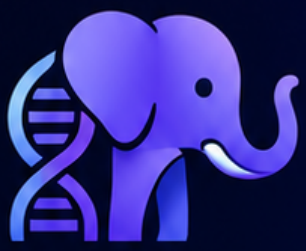

BioPHP

DOCUMENTATION

Documentation de BioPHP

BioPHP est une bibliothèque PHP dédiée à la bioinformatique.

Elle fournit des outils simples et puissants pour manipuler, analyser et visualiser des données biologiques.

SIMPLE
et intuitif

SPÉCIALISÉ
pour la bioinformatique

PUISSANT
et performant

L'écosystème BioAPI, BioPHP & BioTools

Guide de démarrage pour le biologiste

Amélie Duvernet

Version 1

Table des matières

Introduction

Ce guide s'adresse à un biologiste qui connaît déjà son domaine scientifique mais découvre pour la première fois l'écosystème logiciel composé de trois briques complémentaires :

— **BioAPI** — une API REST (API Platform / Symfony) qui expose en lecture les données biologiques de référence : acides aminés, nucléotides, codons, enzymes de restriction, échelles de pKa, matrices de substitution, etc. C'est la source de vérité des données.
— **BioPHP** (`amelaye/biophp`) — une bibliothèque PHP / bundle Symfony qui consomme cette API, manipule des séquences (ADN, ARN, protéines) et sait *parser* les principaux formats de bases de données biologiques (GenBank, Swiss-Prot, EMBL, PDB, PROSITE, ExPASy ENZYME, Entrez, TRANSFAC, KEGG…).
— **BioTools** (`amelaye/biotools`) — un bundle Symfony qui habille les « mini-outils » historiques de biophp.org (traduction ADN → protéine, recherche de palindromes, température de fusion, digestion par enzymes de restriction…) sous forme de formulaires et de services prêts à l'emploi. Il s'appuie entièrement sur BioPHP pour les données et les calculs.

La dépendance va donc dans un seul sens : **BioTools** → **BioPHP** → **BioAPI**. On peut utiliser BioAPI seul, BioPHP seul (avec ou sans BioAPI selon ce qu'on lui donne), ou les trois ensemble.

Chaque chapitre qui suit couvre l'une de ces trois briques, avec des exemples concrets orientés « je cherche la fonction qui fait X ».

Prérequis techniques

— Une version de PHP raisonnablement récente (la contrainte exacte évolue avec le projet, on ne la fige pas ici).
— Composer.
— Pour un usage en bundle Symfony : rien de particulier à prévoir à l'avance — BioTools installe lui-même ce dont il a besoin. On verra les détails concrets dans les chapitres qui leur sont consacrés.
— Un accès réseau sortant vers `api.amelayes-biophp.net`, sauf si vous hébergez votre propre instance de BioAPI.

Installation dans une application Symfony

C'est le mode d'emploi normal si vous partez d'une application Symfony existante (ou d'un nouveau projet créé avec `symfony new`).

```
# La bibliotheque de parsing et de sequences
composer require amelaye/biophp

# Les mini-outils prets a l'emploi (optionnel)
```

```
composer require amelaye/biotools
```

Ces deux commandes installent aussi tout ce dont BioPHP et BioTools ont besoin pour fonctionner. On verra comment s'en servir concrètement, étape par étape, dans les chapitres qui leur sont consacrés.

Note

Une application de démonstration Symfony complète est consultable en ligne (https://demo.amelayes-biophp.net) si vous voulez voir l'ensemble câblé dans de vraies vues Twig.

Installation « vanilla » (sans Symfony)

BioPHP est distribué comme un bundle Symfony, mais les classes qui parlent à l'API et qui manipulent les séquences n'ont aucune dépendance dure au framework : elles peuvent être instanciées directement dans un script PHP classique, à condition de fournir vous-même un client HTTP (Guzzle) et un sérialiseur (JMS Serializer).

```
composer require amelaye/biophp guzzlehttp/guzzle jms/serializer
```

```php
<?php
require_once 'vendor/autoload.php';

use Amelaye\BioPHP\Domain\Sequence\Service\SequenceManager;

$client = new GuzzleHttp\Client([
    'base_uri' => 'https://api.amelayes-biophp.net',
]);
$serializer = JMS\Serializer\SerializerBuilder::create()->build();

$aminoApiManager  = new \Amelaye\BioPHP\Api\AminoApi($client,
    $serializer);
$nucleotidManager = new \Amelaye\BioPHP\Api\NucleotidApi($client,
    $serializer);
$elementManager   = new \Amelaye\BioPHP\Api\ElementApi($client,
    $serializer);

$sequenceManager = new SequenceManager(
    $aminoApiManager,
    $nucleotidManager,
    $elementManager
);

$aMirrors = $sequenceManager->findMirror(
    "AGGGAATTAAGTAAATGGTAGTGG",
    6,
    8,
    'E'
);

var_dump($aMirrors);
```

Attention

BioTools, lui, a besoin d'un minimum de Symfony pour fonctionner tel quel. En usage totalement autonome, on passe plutôt par BioPHP seul. On verra dans le chapitre qui lui est consacré comment utiliser BioTools malgré tout, même en dehors d'une vraie application Symfony.

Chapitre 1

L'API (BioAPI)

1.1 Vue d'ensemble

BioAPI est une API REST bâtie avec API Platform sur Symfony. Elle expose, en **lecture seule**, un ensemble de tables de référence en biologie moléculaire : acides aminés, nucléotides, codons, enzymes de restriction, échelles de pKa, matrices de substitution PAM250, alphabets réduits de protéines, paramètres de température de fusion, et fournisseurs de réactifs. C'est la base de données que consomment BioPHP et, indirectement, BioTools.

— URL de base publique : `https://api.amelayes-biophp.net`
— Format : JSON-LD / Hydra (standard API Platform) — chaque collection est renvoyée dans une clé `hydra:member`.
— Toutes les ressources n'exposent que les opérations `GET` (collection et élément) : il s'agit d'une API de consultation, pas d'édition.
— La pagination est **désactivée** : une requête sur une collection renvoie *tous* les enregistrements en une fois. Pratique pour charger une table de référence complète en mémoire, à garder en tête si vous interrogez la ressource en boucle.

BioAPI publie aussi une interface Swagger interactive, directement sur son URL de base, qui liste toutes les ressources et permet de tester une requête sans écrire une ligne de code :

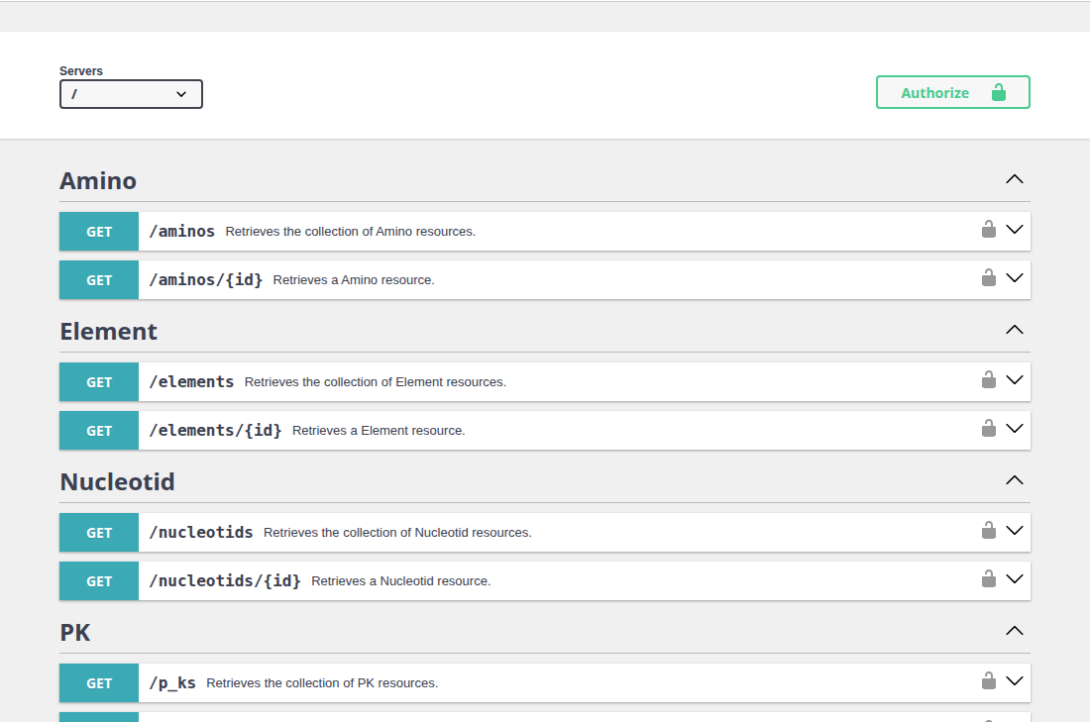

FIGURE 1.1 – Interface Swagger générée automatiquement par BioAPI, listant les ressources disponibles.

1.2 Authentification

La documentation Swagger de BioAPI déclare une clé d'API, transmise via l'en-tête HTTP :

```
X-AUTH-TOKEN: <votre_jeton>
```

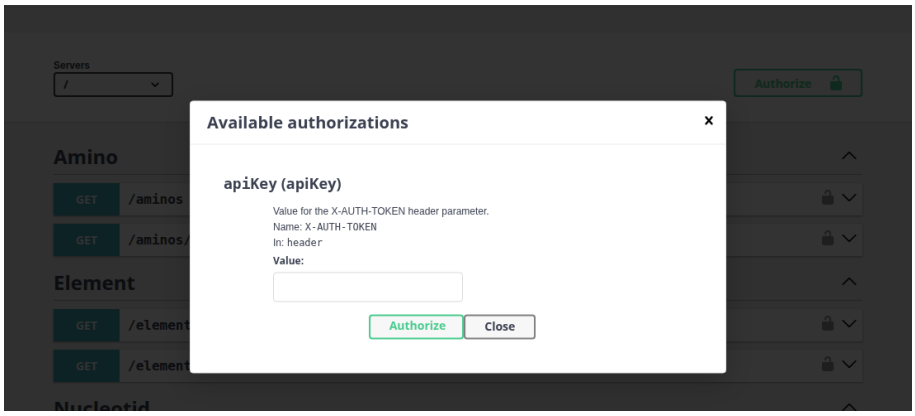

FIGURE 1.2 – Fenêtre « Available authorizations » de l'interface Swagger de BioAPI, avec le nom exact de l'en-tête attendu.

Les adaptateurs PHP de BioPHP (voir chapitre 2) acceptent ce jeton en troisième argument optionnel de leur constructeur.

Attention

Test effectué depuis l'interface Swagger de BioAPI : une requête `GET /aminos` envoyée **sans** en-tête `X-AUTH-TOKEN` renvoie bien un code `200` avec les données complètes. En l'état, l'authentification ne semble donc pas être imposée en lecture sur l'instance publique. Ce comportement n'étant documenté nulle part comme volontaire, il pourrait changer sans préavis : envoyez le jeton dès que vous en possédez un, par prudence.

À compléter

La procédure d'obtention d'un jeton pour l'instance publique n'est pas documentée dans le code — à préciser dans une prochaine passe si elle est nécessaire pour votre usage, ou à ignorer si vous hébergez votre propre instance sans contrôle d'accès.

1.3 Ressources disponibles

Le tableau suivant sert d'index rapide : pour chaque ressource, le chemin HTTP, et les champs exposés. C'est le point d'entrée à consulter quand vous cherchez « où est telle donnée ».

Ressource	Chemin	Champs
Acides aminés	`/aminos`	`id`, `name`, `name1Letter`, `name3Letters`, `weight1`, `weight2`, `residueMolWeight`
Éléments chimiques	`/elements`	`id`, `name` (ex. : *water*, *carbone*), `weight`
Nucléotides	`/nucleotids`	`id`, `letter` (A/T/G/C…), `complement`, `nature` (ADN/ARN), `weight`
pKa (EMBOSS)	`/p_ks/<id>` (élément seul)	`id` (nom de l'échelle, ex. *EMBOSS*), `nTerminus`, `cTerminus`, et les pKa des chaînes latérales `k`, `r`, `h`, `d`, `e`, `c`, `y`
Matrice PAM250	`/pam250_matrix_digits`	`id` (index), `value` (score de substitution)
Alphabets réduits	`/protein_reductions`	`id`, `alphabet` (ex. *Murphy*), `letters`, `pattern`, `nature`, `reduction`, `description`
Tm — empilement de bases	`/tm_base_stackings`	`id`, `temperatureEnthalpy`, `temperatureEnthropy`
Codons (triplets)	`/triplets`	`id`, `triplet`
Codons par espèce	`/triplet_species`	`id`, `nature`, `triplets`, `tripletsGroups`
Enzymes type II	`/type_i_i_endonucleases`	`id` (nom de l'enzyme), `samePattern`, `recognitionPattern`, `computingPattern`, `lengthRecognitionPattern`, `cleavagePosUpper`, `cleavagePosLower`, `nbNonNBases`
Enzymes type IIb	`/type_i_ib_endonucleases`	mêmes champs que type II
Enzymes type IIs	`/type_i_is_endonucleases`	mêmes champs que type II

Ressource	Chemin	Champs
Fournisseurs	`/vendors`	`id`, `vendor`
Liens fournisseurs	`/vendor_links`	`id`, `name`, `link`

Note

Les trois ressources d'enzymes de restriction ont des chemins qui reflètent littéralement le nom de classe PHP (`TypeIIEndonuclease` → `/type_i_i_endonucleases`), dû à la convention de pluralisation automatique d'API Platform sur un nom contenant des majuscules consécutives. C'est un détail à ne pas deviner : mieux vaut copier ces chemins tels quels.

1.4 Exemples de requêtes

1.4.1 En ligne de commande

Récupérer la liste complète des acides aminés :

```
curl -H "Accept: application/ld+json" \
     https://api.amelayes-biophp.net/aminos
```

Récupérer un enregistrement précis d'échelle de pKa :

```
curl -H "Accept: application/ld+json" \
     https://api.amelayes-biophp.net/p_ks/EMBOSS
```

Une réponse de collection a la forme suivante (extrait réel, format Hydra — notez la première entrée `STOP`, qui représente les codons stop et non un acide aminé à proprement parler) :

```
{
  "@context": "/contexts/Amino",
  "@id": "/aminos",
  "@type": "hydra:Collection",
  "hydra:totalItems": 26,
  "hydra:member": [
    {
      "@id": "/aminos/*",
      "@type": "Amino",
      "id": "*",
      "name": "STOP",
      "name1Letter": "*",
      "name3Letters": "STP",
      "weight1": 0,
      "weight2": 0
    },
    {
      "@id": "/aminos/A",
      "@type": "Amino",
      "id": "A",
      "name": "Alanine",
      "name1Letter": "A",
      "name3Letters": "Ala",
      "weight1": 89.09,
      "weight2": 89.09,
      "residueMolWeight": 71.07
```

```
    }
  ]
}
```

Le même aller-retour vu depuis l'interface Swagger de BioAPI (bouton « Try it out » puis « Execute ») :

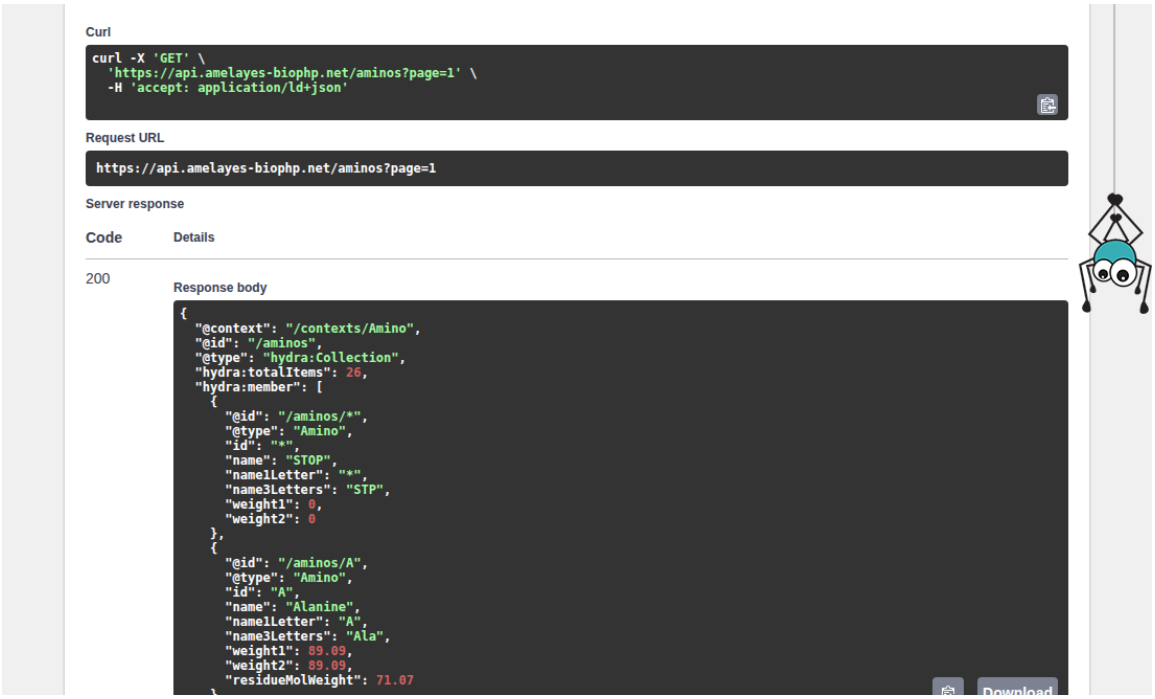

FIGURE 1.3 – Requête `GET /aminos` exécutée depuis l'interface Swagger de BioAPI, avec sa réponse réelle.

1.4.2 En PHP, avec Guzzle

Sans passer par les adaptateurs de BioPHP — utile si vous voulez consommer l'API depuis un script totalement indépendant :

```php
<?php
require_once 'vendor/autoload.php';

$client = new GuzzleHttp\Client([
    'base_uri' => 'https://api.amelayes-biophp.net',
]);

$response = $client->get('/nucleotids', [
    'headers' => ['Accept' => 'application/ld+json'],
]);

$data = json_decode($response->getBody()->getContents(), true);

foreach ($data['hydra:member'] as $nucleotid) {
    printf(
        "%s (%s) complement: %s, poids: %s\n",
        $nucleotid['letter'],
        $nucleotid['nature'],
        $nucleotid['complement'],
```

```
        $nucleotid['weight']
    );
}
```

1.4.3 Exemple « biologiste » : repérer les sites de coupure d'une enzyme de restriction

Un exemple concret de « parsing » côté API : aller chercher les caractéristiques d'une enzyme de restriction via `/type_i_i_endonucleases` (son site de reconnaissance, ses positions de coupure) et les utiliser pour repérer à la main les sites de coupure dans une séquence, sans passer par l'outil de digestion déjà prêt de BioTools (voir §3.4.15 pour la version « clé en main »).

```php
<?php
require_once 'vendor/autoload.php';

$client = new GuzzleHttp\Client([
    'base_uri' => 'https://api.amelayes-biophp.net',
]);

// 1. On recupere les enzymes de type II pour verifier un site de coupure
$response = $client->get('/type_i_i_endonucleases/EcoRI', [
    'headers' => ['Accept' => 'application/ld+json'],
]);
$ecoRI = json_decode($response->getBody()->getContents(), true);

printf(
    "EcoRI reconnait %s, coupe en position %d/%d\n",
    $ecoRI['recognitionPattern'],
    $ecoRI['cleavagePosUpper'],
    $ecoRI['cleavagePosLower']
);

// 2. On recherche ce site dans une sequence
$sequence = "GGAATTCGGAATTCAAGCTT";
$site = $ecoRI['recognitionPattern']; // ex. "GAATTC"
$positions = [];
$offset = 0;
while (($pos = strpos($sequence, $site, $offset)) !== false) {
    $positions[] = $pos;
    $offset = $pos + 1;
}

printf("Sites %s trouves aux positions : %s\n", $site, implode(', ', $positions));
```

Note

L'identifiant exact à utiliser dans l'URL (`EcoRI` ci-dessus) correspond au champ `id` de l'entité — à vérifier via `GET /type_i_i_endonucleases` si le nom exact de l'enzyme qui vous intéresse n'est pas connu à l'avance.

Chapitre 2

BioPHP

2.1 Vue d'ensemble

BioPHP se découpe en quatre familles d'outils, que l'on retrouve dans les sections qui suivent :

— **Lire des fichiers de bases de données biologiques** — un lecteur par format (GenBank, Swiss-Prot, PROSITE, TRANSFAC, KEGG…), 31 au total.
— **Manipuler des séquences** — traduction, complément, recherche de motifs, alignement, enzymes de restriction…
— **Une petite boîte à outils** — quelques fonctions de biologie et de statistiques qui reviennent souvent.
— **Parler à BioAPI** — les classes qui récupèrent les données de référence présentées au chapitre 1 (acides aminés, nucléotides, enzymes…), utilisées en coulisses par les trois familles précédentes.

La première famille — la lecture de fichiers — occupe la majeure partie de ce chapitre : c'est elle qui compte le plus de classes (31 lecteurs), et c'est généralement le premier arrêt d'un biologiste qui a un fichier téléchargé sur son disque et veut en extraire l'information.

Note

Toute cette famille repose sur deux dossiers qui se répondent : `Domain/Database` porte le mécanisme générique (la fabrique qui choisit le bon lecteur, le gestionnaire de collections pour les gros fichiers), et `Domain/Parser` porte les 31 lecteurs eux-mêmes, plus les quelques objets que certains d'entre eux renvoient en plus de la séquence. Les deux dossiers partagent la même organisation interne : un sous-dossier `Service` pour les classes qui font le travail, `Entity` pour les objets qu'elles manipulent ou renvoient, et `Interfaces` pour le contrat de chacun. Vous n'avez pas besoin de retenir cette organisation pour utiliser BioPHP — un seul point d'entrée suffit, voir section suivante — mais elle explique le chemin complet des classes citées dans les exemples de code.

2.2 Installation et configuration

2.2.1 Installation

```
composer require amelaye/biophp
```

En Symfony, on enregistre le bundle dans `config/bundles.php` :

```php
return [
    // ...
    JMS\SerializerBundle\JMSSerializerBundle::class => ['all' => true],
    Amelaye\BioPHP\AmelayeBioPHPBundle::class => ['all' => true],
];
```

Note

Le bundle active lui-même l'ORM Doctrine et enregistre ses propres entités (`Domain/Database/Entity` et `Domain/Sequence/Entity`) auprès de lui, via un `prepend()` de dependency injection : vous n'avez donc *rien* à déclarer côté `doctrine.yaml` pour que ce mapping soit pris en compte, du moment que `doctrine/orm` et `doctrine/doctrine-bundle` sont installés — ils le sont déjà, en tant que dépendances de BioPHP.

En installation « vanilla » (sans Symfony), vous instanciez directement les classes de `Domain/` dont vous avez besoin. Pour la lecture simple d'un fichier (§2.3.1), aucune base de données n'est nécessaire ; pour indexer de gros fichiers ou pour sauvegarder ce qu'un lecteur a extrait, voir la section suivante.

2.2.2 Persistance : installer les tables dans une base MySQL existante

BioPHP ne réclame une base de données que pour deux usages : indexer un gros fichier pour y retrouver un enregistrement précis sans tout relire (`DatabaseManager`, §2.3.1), et sauvegarder ce qu'un lecteur a extrait pour le relire ensuite sans repasser par le fichier d'origine (`RecordManager`, §2.3.2). Si vos fichiers tiennent en mémoire et que vous les relisez à chaque fois, vous pouvez ignorer cette section entièrement : rien dans ce chapitre n'oblige à utiliser une base de données.

Quand vous en avez besoin, BioPHP mappe douze entités Doctrine, qui se traduisent en autant de tables :

Table	Rôle
`collection`	Un ensemble nommé de fichiers indexés ensemble (un lot importé).
`collection_element`	L'index d'un enregistrement dans un fichier de la collection : son identifiant, le fichier et la ligne où il commence. Sert à `DatabaseManager::fetch()`.
`parsed_record`	Ce qu'un lecteur a extrait d'un enregistrement, conservé en JSON. Sert à `RecordManager` (§2.3.2).
`sequence`, `feature`, `reference`, `author`, `accession`, `keyword`, `gb_sequence`, `sp_databank`, `src_form`	Les neuf objets communs de §2.3.3, mappés en entités Doctrine pour que le schéma soit complet et testé — **mais aucun code de BioPHP n'écrit dedans**, voir l'encadré Attention ci-dessous.

Attention

Les tables `sequence`, `feature`, `reference` etc. existent dans le schéma mais restent vides tant que vous ne les remplissez pas vous-même : les 31 lecteurs de ce chapitre renvoient ces objets en mémoire, pour que vous les lisiez avec leurs « getters », mais ne les persistent jamais eux-mêmes. La seule persistance automatique offerte par BioPHP passe par `RecordManager` et la table `parsed_record` (§2.3.2). Ne soyez donc pas surpris de trouver ces neuf tables vides après avoir lu des dizaines de fichiers : c'est le comportement normal.

En Symfony

Renseignez la variable d'environnement `DATABASE_URL` pour pointer vers votre base MySQL existante :

```
# .env.local
DATABASE_URL="mysql://utilisateur:motdepasse@127.0.0.1:3306/ma_base?
    serverVersion=8.0"
```

Puis créez les tables qui manquent :

```
# base neuve, ou base existante sans aucune des tables ci-dessus
$ bin/console doctrine:schema:create

# base existante qui a deja d'autres tables (les votres, ou une
# installation precedente de BioPHP a mettre a niveau) :
# ne cree/modifie que ce qui manque, sans toucher au reste
$ bin/console doctrine:schema:update --force
```

Note

`doctrine:schema:update -force` compare le mapping des entités au schéma réellement présent dans la base et n'applique que la différence : c'est la commande à préférer dès que la base contient déjà des tables qui ne sont pas les vôtres (celles de votre application, ou d'un autre bundle). Elle ne supprime jamais de table qui n'est plus mappée, sauf si vous ajoutez explicitement `-complete`.

En vanilla

Sans Symfony, vous construisez l'`EntityManager` vous-même, en pointant le mapping par attributs vers les deux dossiers d'entités de BioPHP, puis vous demandez à Doctrine de créer ou mettre à jour le schéma :

```
1  <?php
2  require_once 'vendor/autoload.php';
3
4  use Doctrine\DBAL\DriverManager;
5  use Doctrine\ORM\EntityManager;
6  use Doctrine\ORM\Mapping\UnderscoreNamingStrategy;
7  use Doctrine\ORM\ORMSetup;
8  use Doctrine\ORM\Tools\SchemaTool;
9
10 $config = ORMSetup::createAttributeMetadataConfiguration(
11     [
12         __DIR__ . '/vendor/amelaye/biophp/Domain/Database/Entity',
```

```
        __DIR__ . '/vendor/amelaye/biophp/Domain/Sequence/Entity',
    ],
    true // mode developpement ; passer a false en production
);
$config->setNamingStrategy(new UnderscoreNamingStrategy(CASE_LOWER, true
    ));

$connection = DriverManager::getConnection([
    'driver'   => 'pdo_mysql',
    'host'     => '127.0.0.1',
    'port'     => 3306,
    'dbname'   => 'ma_base',
    'user'     => 'utilisateur',
    'password' => 'motdepasse',
], $config);

$entityManager = new EntityManager($connection, $config);

// a executer une fois pour creer les tables manquantes dans la base
// existante ('ma_base' ci-dessus), sans toucher a ses tables actuelles
(new SchemaTool($entityManager))->updateSchema(
    $entityManager->getMetadataFactory()->getAllMetadata()
);
```

Attention

BioPHP ne fournit pas de bibliothèque de migrations (`doctrine/migrations` n'est pas une dépendance) : c'est `doctrine:schema:update` (Symfony) ou `SchemaTool::updateSchema()` (vanilla) qui tient lieu d'outil d'installation et de mise à niveau du schéma. Si votre projet utilise déjà `doctrine/migrations`, rien ne vous empêche d'y intégrer les entités de BioPHP comme les vôtres : elles sont mappées par attributs de la même manière.

Attention

Si vous mettez à niveau une installation BioPHP antérieure au 20 septembre 2026 : le champ `organism` de `Sequence` est passé d'un stockage PHP sérialisé (type `array`) à du JSON, et plusieurs clés étrangères et noms d'index ont changé (les contraintes nommées `prim_acc` sur six tables différentes entraient en collision sur SQLite et PostgreSQL, où les noms d'index sont globaux). Ces tables doivent être recréées : `doctrine:schema:update -force` s'en charge, mais sauvegardez d'abord les données que vous teniez à garder.

2.3 Lire un fichier de base de données

2.3.1 Le mécanisme de lecture, une fois pour toutes

Quel que soit le format, la lecture se fait toujours de la même manière : on charge le fichier ligne par ligne, on donne ces lignes à `DatabaseReaderFactory` en précisant le format, et on récupère en retour un objet — le *lecteur* — dont les « getters » donnent accès aux informations lues.

```
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('mon_fichier.gb'); // un tableau, une ligne par entree
$lecteur = DatabaseReaderFactory::readDatabase('GENBANK', $aLignes);
```

C'est le seul appel nécessaire dans la grande majorité des cas. Le nom du format (`'GENBANK'`, `'SWISSPROT'`, `'PROSITE'`…) est toujours celui qui apparaît dans la colonne « Format » de l'index de la section 2.5.

En coulisses, `DatabaseReaderFactory` délègue tout le travail à `DatabaseParserFactory` : c'est cette seconde classe qui connaît la liste des 31 formats supportés (`getFormats()`), qui sait associer un nom de format à la classe qui le lit (`getParserClass()`), et qui construit une instance fraîche de cette classe (`createParser()`). C'est également le point d'entrée à modifier pour qui voudrait ajouter un 32e format : une classe dans `Domain/Parser/Service`, et une ligne d'enregistrement dans cette fabrique.

Note

Les lecteurs eux-mêmes ne sont pas des services Symfony — il n'y a pas de déclaration à faire pour eux dans le conteneur d'injection de dépendances. C'est ce qui permet à l'appel ci-dessus de fonctionner à l'identique en Symfony et en installation « vanilla » : dans les deux cas, on appelle simplement `DatabaseReaderFactory::readDatabase()`.

Ce que tous les lecteurs ont en commun, au fond, c'est de savoir répondre à quatre questions sur leur format : où commence un enregistrement, où il se termine, quel est son identifiant, et comment lire les lignes qui le composent. C'est ce contrat commun (l'interface `ParseDatabaseInterface`) qui permet à `DatabaseManager` (ci-dessous) de découper n'importe quel fichier sans rien connaître du format qu'il contient.

Note

Un fichier de base de données contient généralement plusieurs enregistrements à la suite, et l'exemple ci-dessus suppose que `mon_fichier.gb` n'en contient qu'un seul. Pour un fichier qui en contient des milliers, c'est `DatabaseManager` qui prend le relais : sa méthode `recording()` parcourt le fichier une première fois pour repérer où commence et où finit chaque enregistrement, et indexe cette information (via Doctrine, dans les tables `Collection` et `CollectionElement`). Sa méthode `fetch($id)` retrouve ensuite un enregistrement précis par son identifiant, sans avoir à relire tout le fichier depuis le début, et vous rend le même genre de lecteur que `DatabaseReaderFactory::readDatabase()`. C'est la seule partie de BioPHP qui réclame une base de données : si vos fichiers tiennent en mémoire et que vous savez déjà quel fichier contient l'enregistrement qui vous intéresse, vous n'en avez pas besoin.

2.3.2 Sauvegarder et relire les enregistrements

Le mécanisme décrit ci-dessus (`recording()` / `fetch()`) indexe un fichier pour y retrouver un enregistrement, mais retourne à ce fichier pour le relire et le reparser à chaque appel. BioPHP peut aussi conserver directement ce qu'un lecteur a extrait, une bonne fois pour toutes, grâce à `Amelaye\BioPHP\Domain\Database\Service\RecordManager` (voir son installation en §2.2.2).

Ce mécanisme est entièrement générique : il fonctionne à l'identique pour les 31 formats de ce chapitre, y compris un futur 32e format que vous ajouteriez vous-même, sans code supplémentaire — il lit simplement les « getters » publics de ce que le lecteur a renvoyé. Trois opérations, groupées derrière l'interface `RecordStorageInterface` :

Méthode	Rôle
`store($format, $lignes)`	Parse un enregistrement isolé (le même tableau de lignes que `readDatabase()`) et l'enregistre en base. Un enregistrement déjà stocké sous le même identifiant est mis à jour plutôt que dupliqué.
`storeFile($nom, $format, ...$fichiers)`	Parse tous les enregistrements d'un ou plusieurs fichiers et les enregistre, groupés sous une collection nommée $nom (créée si elle n'existe pas encore). Retourne le nombre d'enregistrements traités.
`find($format, $id)`	Retrouve un enregistrement déjà stocké, sans repasser par le fichier d'origine. Retourne `null` s'il n'existe pas.

Ce qui est stocké est un objet `ParsedRecord`, dont `getData()` rend un tableau associatif : les mêmes informations que les « getters » du lecteur, mais à plat et prêtes pour du JSON (`getGeneNames()` devient la clé `geneNames`, `isTruePositive()` devient `truePositive`, et un objet rencontré au passage — une référence, un atome…— est déroulé de la même manière, récursivement).

```php
<?php
use Amelaye\BioPHP\Domain\Database\Service\RecordManager;

// $recordManager est autowire en Symfony (RecordStorageInterface) ;
// en vanilla : new RecordManager($entityManager, './data/');

// Stocker tout un fichier Swiss-Prot d'un coup
$nbEnregistres = $recordManager->storeFile('uniprot_sample', 'SWISSPROT'
    , 'sprot.dat');
echo "$nbEnregistres enregistrements stockes.\n";

// Retrouver un enregistrement precis, sans relire le fichier
$enregistrement = $recordManager->find('SWISSPROT', 'TNFA_HUMAN');
if ($enregistrement !== null) {
    $donnees = $enregistrement->getData();
    echo $donnees['sequence']['description'] ?? '';
    foreach ($donnees['geneNames'] ?? [] as $synonymes) {
        echo implode(' / ', $synonymes) . "\n";
    }
}
```

Chaque fiche qui suit rappelle, dans son paragraphe « Persistance », le nom de format à passer à `store()` / `storeFile()` / `find()` — c'est le même nom que celui donné à `readDatabase()`, dans la même colonne de l'index de la section 2.5.

Attention

`storeFile()` parse et écrit chaque enregistrement individuellement : pour un très gros fichier (des dizaines de milliers d'enregistrements), préférez l'exécuter en ligne de commande ou en tâche de fond plutôt que dans une requête HTTP, même si un flush groupé par lots de 100 en limite le coût mémoire.

2.3.3 Les objets communs à toute lecture

La plupart des 31 lecteurs — tous ceux qui décrivent une séquence annotée (GenBank, Swiss-Prot, EMBL en tête) — héritent d'une même classe de base, `ParseDbAbstractManager`, et rendent donc

leurs informations à travers le même jeu d'objets, tous dans `Amelaye\BioPHP\Domain\Sequence\Entity`. Ce tableau sert de référence commune à toutes les fiches qui suivent : chacune se contente d'indiquer quels de ces objets elle remplit réellement.

Objet	Rendu par	Champs principaux
`Sequence`	`getSequence()`	`getPrimAcc()` (accession principale), `getEntryName()`, `getSeqLength()`, `getMolType()`, `getDate()`, `getSource()`, `getOrganism()` (tableau, de la lignée taxonomique au règne), `getDescription()`, `getSequence()` (la séquence brute), `getFragment()`
`Feature`	`getFeatures()` (tableau)	`getFtKey()` (ex. `"CDS"`, `"gene"`), `getFtFrom()`, `getFtTo()`, `getStrand()` (`"+"` ou `"-"`), `getFtQual()` / `getFtValue()` (nom et valeur du qualificatif), `getFtDesc()`
`Reference`	`getReferences()` (tableau)	`getRefno()`, `getTitle()`, `getJournal()`, `getMedline()`, `getPubmed()`, `getBaseRange()`, `getRemark()`, `getComments()`
`Author`	`getAuthors()` (tableau)	`getRefno()` (la référence à laquelle il est rattaché), `getAuthor()`
`Accession`	`getAccession()` (tableau)	`getAccession()` — les accessions *secondaires* d'un enregistrement (la principale est déjà dans `Sequence::getPrimAcc()`)
`Keyword`	`getKeywords()` (tableau)	`getKeywords()` — un mot-clé par objet
`GbSequence`	`getGbSequence()`	`getVersion()`, `getNcbiGiId()`, `getStrands()`, `getTopology()`, `getDivision()`, `getSegmentNo()` / `getSegmentCount()` — pas toujours rempli, voir chaque fiche
`SrcForm`	`getSrcForm()`	`getEntry()` — peu utilisé, présent pour compatibilité
`SpDatabank`	`getSpDatabank()` (tableau)	`getDbName()`, `getPid1()`, `getPid2()` — renvois vers d'autres banques (champ `DR` de Swiss-Prot)

2.4 Les objets rendus par certains parsers

Cinq formats — PDB, PROSITE, ExPASy ENZYME, ENTREZ et GENOME — décrivent des informations que les objets ci-dessus ne prévoient pas (des coordonnées atomiques, une liste de références PDB, une maladie liée à un déficit enzymatique…), à travers sept objets au total (PDB à lui seul en fournit trois). Chacun a donc son propre objet, dans `Domain/Parser/Entity` :

Objet	Rendu par	Champs
`PdbAtom`	`PDB` (`getAtoms()`, `getHetAtoms()`)	`getSerial()`, `getName()`, `getAltLoc()`, `getResName()`, `getChainId()`, `getResSeq()`, `getX()` / `getY()` / `getZ()`, `getOccupancy()`, `getTempFactor()`, `getElement()`

Objet	Rendu par	Champs
`PdbHelix`	`PDB` (`getHelices()`)	`getHelixId()`, `getInitResName()`, `getInitChainId()`, `getInitSeqNum()`, `getEndResName()`, `getEndChainId()`, `getEndSeqNum()`, `getHelixClass()`, `getLength()`
`PdbSheet`	`PDB` (`getSheets()`)	`getSheetId()`, `getStrand()`, `getInitResName()`, `getInitChainId()`, `getInitSeqNum()`, `getEndResName()`, `getEndChainId()`, `getEndSeqNum()`
`PrositeDbRef`	`PROSITE` (`getDbRefs()`)	`getAccession()`, `getEntryName()`, `isTruePositive()`
`ExpasyDisease`	`EXPASY_ENZYME` (`getDiseases()`)	`getDisease()`, `getReference()`
`EntrezReference`	`ENTREZ` (`getReferences()`)	`getRefNo()`, `getBaseRange()`, `getAuthors()`, `getTitle()`, `getJournal()`, `getMedline()`, `getPubmed()`, `getRemark()`
`GenomeReference`	`GENOME` (`getReferences()`)	`getType()`, `getAuthors()`, `getTitle()`, `getJournal()`, `getVolume()`, `getPages()`, `getYear()`

Note

Chacun de ces objets a son interface (`PdbAtomInterface`, `PrositeDbRefInterface`...) dans `Domain/Parser/Interfaces`. Vous n'en avez pas besoin pour lire un fichier, mais elle permet, si vous en avez l'usage, de substituer votre propre implémentation à l'objet fourni par BioPHP sans toucher au lecteur lui-même.

2.5 Les parsers, un par un

Chaque fiche qui suit se lit indépendamment des autres : allez directement à celle du format qui vous intéresse. L'index ci-dessous sert de table de correspondance rapide entre le nom d'un format et ce qu'il décrit, pour retrouver le bon si votre fichier ne correspond à aucun des exemples déjà rencontrés.

TABLE 2.5: Index des 31 formats supportés

Format	Décrit
`SWISSPROT`	Séquence protéique annotée (UniProt/Swiss-Prot)
`GENBANK`	Séquence annotée (NCBI)
`EMBL`	Séquence annotée (EMBL)
`PDB`	Structure 3D (coordonnées atomiques, hélices, feuillets)
`PROSITE`	Motif / signature de famille de protéines
`EXPASY_ENZYME`	Enzyme par numéro EC
`PDBSTR`	Famille structurale dérivée de PDB
`UNIGENE`	Cluster de séquences expriment le même gène
`PRINTS`	Empreinte (motifs multiples) de familles de protéines
`BLOCKS`	Motifs conservés de familles de protéines
`AAINDEX`	Indices physico-chimiques des acides aminés

Table 2.5 – suite

Format	Décrit
PRODOM	Familles de domaines protéiques
NCBI_LIT	Référentiel des revues scientifiques (NCBI)
PMD	Mutations de protéines (Protein Mutant Database)
HGBASE	Variants génétiques humains bi-alléliques
PRF	Séquences protéiques (Protein Research Foundation)
PIR	Séquences protéiques annotées (Protein Information Resource)
EPD	Promoteurs eucaryotes (Eukaryotic Promoter Database)
GENOME	Statistiques de séquençage d'un génome
ENTREZ	Enregistrement génomique (NCBI Entrez)
TRANSFAC_MATRIX	Matrice de poids-position (facteurs de transcription)
TRANSFAC_GENE	Gène régulé
TRANSFAC_CLASS	Classe de facteurs de transcription
TRANSFAC_CELL	Type cellulaire
TRANSFAC_FACTOR	Facteur de transcription
TRANSFAC_SITE	Site de fixation d'un facteur
KEGG_COMPOUND	Composé chimique (métabolisme)
KEGG_REACTION	Réaction chimique
KEGG_ENZYME	Enzyme (vue métabolisme)
KEGG_ORTHOLOG	Groupe de gènes orthologues
KEGG_GENOME	Génome (vue métabolisme)

2.5.1 Swiss-Prot

Un enregistrement Swiss-Prot (aujourd'hui UniProtKB/Swiss-Prot) décrit une protéine : sa séquence, ses accessions, ses caractéristiques structurales et fonctionnelles annotées à la main, ses références bibliographiques et ses renvois vers d'autres banques. C'est la base la mieux annotée des trois grands formats de séquences (avec GenBank et EMBL) — celle vers laquelle se tourner en priorité quand la qualité de l'annotation compte plus que l'exhaustivité.

Format à passer à `readDatabase()` : `'SWISSPROT'`

Un enregistrement s'ouvre sur une ligne `ID` et se ferme sur une ligne `//` :

```
ID   TNFA_HUMAN     STANDARD;      PRT;   233 AA.
AC   P01375;
DT   01-NOV-1990 (REL. 16, CREATED)
DE   TUMOR NECROSIS FACTOR PRECURSOR (TNF-ALPHA).
GN   TNF.
OS   HOMO SAPIENS (HUMAN).
OC   EUKARYOTA; METAZOA; CHORDATA; MAMMALIA; PRIMATES.
RN   [1]
RP   X-RAY CRYSTALLOGRAPHY.
RA   ECK M.J., SPRANG S.R.;
RL   J. BIOL. CHEM. 264:17595-17605(1989).
DR   PDB; 1TNF; X-RAY.
KW   CYTOKINE; GLYCOPROTEIN.
FT   DOMAIN       1     35       CYTOPLASMIC.
SQ   SEQUENCE   233 AA;  25644 MW;
     MSTESMIRDV ELAEEALPKK TGGPQGSRRC LFLSLFSFLI VAGATTLFCL LHFGVIGPQR
//
```

Ce que le lecteur rend : les objets communs de la section 2.3.3 (`getSequence()`, `getFeatures()`, `getReferences()`, `getAuthors()`, `getAccession()`, `getKeywords()`), plus :

Méthode	Contenu
`getSpDatabank()`	Tableau de `SpDatabank` — les renvois vers d'autres banques (champ `DR`)
`getGeneNames()`	Tableau des noms de gènes, groupés par synonymes (champ `GN`)
`getSequpdDate()`	Date de la dernière mise à jour de la séquence (champ `DT`)
`getNotupdDate()`	Date de la dernière mise à jour de l'annotation (champ `DT`)

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('tnfa_human.txt');
$lecteur = DatabaseReaderFactory::readDatabase('SWISSPROT', $aLignes);

$sequence = $lecteur->getSequence();
echo $sequence->getEntryName();      // "TNFA"
echo $sequence->getDescription();    // "TUMOR NECROSIS FACTOR PRECURSOR ..."
echo $sequence->getSequence();       // la sequence proteique elle-meme

// Les noms de genes, groupes par synonymes
foreach ($lecteur->getGeneNames() as $aSynonymes) {
    echo implode(' / ', $aSynonymes) . "\n";
}

// Les sites et domaines annotes (FT)
foreach ($lecteur->getFeatures() as $feature) {
    printf(
        "%s : de %d a %d\n",
        $feature->getFtKey(),    // ex. "DOMAIN", "ACT_SITE", "BINDING"
        $feature->getFtFrom(),
        $feature->getFtTo()
    );
}

// Les renvois vers d'autres banques (DR), par exemple pour retrouver
// les structures PDB associees a cette proteine
foreach ($lecteur->getSpDatabank() as $renvoi) {
    if ($renvoi->getDbName() === 'PDB') {
        echo "Structure PDB : " . $renvoi->getPid1() . "\n";
    }
}

echo "Derniere mise a jour de la sequence : " . $lecteur->getSequpdDate();
```

À quoi ça sert concrètement : récupérer la séquence protéique et sa description en une ligne ; lister les synonymes d'un gène ; parcourir les sites fonctionnels annotés à la main (domaines, sites actifs, sites de fixation) pour les positionner sur la séquence ; retrouver rapidement, via `getSpDatabank()`, les identifiants PDB, InterPro ou Pfam liés à une protéine sans avoir à interroger ces banques séparément ; connaître la fraîcheur de l'annotation avant de s'y fier.

Persistance : `$recordManager->find('SWISSPROT', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'SWISSPROT', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.2 GenBank

Un enregistrement GenBank décrit lui aussi une séquence annotée, mais avec un accent différent de Swiss-Prot : là où Swiss-Prot documente une protéine en détail, GenBank documente le contexte génomique d'une séquence nucléique — où se trouvent les gènes, les exons, les CDS sur le brin séquencé. C'est le format historique du NCBI, et le plus courant pour l'ADN et l'ARN.

Format à passer à `readDatabase()` : `'GENBANK'`

Un enregistrement s'ouvre sur une ligne `LOCUS` et se ferme sur une ligne `//` :

```
LOCUS       AB012345                233 bp    mRNA    linear   PRI 01-
   JAN-2020
DEFINITION  Homo sapiens mRNA for tumor necrosis factor, complete cds.
ACCESSION   AB012345
VERSION     AB012345.1  GI:123456789
KEYWORDS    TNF.
SOURCE      Homo sapiens (human)
  ORGANISM  Homo sapiens
            Eukaryota; Metazoa; Chordata; Mammalia; Primates.
REFERENCE   1  (bases 1 to 233)
  AUTHORS   Eck M.J., Sprang S.R.
  TITLE     The structure of TNF
  JOURNAL   J. Biol. Chem. 264:17595-17605(1989)
FEATURES             Location/Qualifiers
     source          1..233
                     /organism="Homo sapiens"
     gene            1..233
                     /gene="TNF"
     CDS             1..233
                     /product="tumor necrosis factor"
ORIGIN
        1 atgagcacag aaagcatgat ccgggacgtg gagctggccg
//
```

Ce que le lecteur rend : les objets communs de la section 2.3.3, plus `getGbSequence()` — un `GbSequence` avec, pour GenBank, la version complète (`getVersion()`), l'identifiant `GI` (`getNcbiGiId()`), la nature des brins, la topologie et la division.

Attention

La table `FEATURES` peut décrire des dizaines de types de blocs différents dans un fichier GenBank réel. Le lecteur BioPHP n'en reconnaît explicitement que cinq : `source`, `gene`, `exon`, `CDS` et `misc_feature`. Les autres types (`mRNA`, `intron`, `polyA_site`...) sont traversés sans erreur mais n'apparaissent pas dans `getFeatures()`.

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('sequence.gb');
$lecteur = DatabaseReaderFactory::readDatabase('GENBANK', $aLignes);

$sequence = $lecteur->getSequence();
```

```php
echo $sequence->getPrimAcc();        // le numero d'accession, ex. "AB012345"
echo $sequence->getDescription();    // la description en une ligne (DEFINITION)
echo $sequence->getOrganism();       // la classification de l'organisme
echo $sequence->getSequence();       // la sequence elle-meme (A, T, G, C ...)

$gb = $lecteur->getGbSequence();
echo $gb->getVersion();              // "AB012345.1"
echo $gb->getNcbiGiId();             // "123456789"

foreach ($lecteur->getFeatures() as $feature) {
    printf(
        "%s : de %d a %d (%s)\n",
        $feature->getFtKey(),    // "source", "gene", "exon", "CDS" ou "misc_feature"
        $feature->getFtFrom(),
        $feature->getFtTo(),
        $feature->getStrand()    // "+" ou "-"
    );
}

foreach ($lecteur->getReferences() as $reference) {
    echo $reference->getTitle() . " (" . $reference->getJournal() . ")\n";
}
```

À quoi ça sert concrètement : extraire la séquence et situer l'organisme dont elle provient ; lister les gènes, CDS et exons avec leurs positions et leur brin, pour par exemple découper la région codante et la traduire (voir `translate()`, section suivante) ; comparer l'identifiant de version (`VERSION`/`GI`) entre deux téléchargements du même enregistrement pour détecter une mise à jour ; récupérer la bibliographie associée à une séquence.

Persistance : `$recordManager->find('GENBANK', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'GENBANK', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.3 EMBL

Le format EMBL (banque européenne, aujourd'hui ENA) décrit le même genre d'information que GenBank — séquence annotée, organisme, table de caractéristiques, bibliographie — mais avec la grammaire à étiquettes sur deux lettres de Swiss-Prot (`ID`, `AC`, `DE`…) plutôt que les mots-clés complets de GenBank. Utile sur des fichiers qui proviennent d'ENA plutôt que du NCBI, pour le même usage que GenBank.

Format à passer à `readDatabase()` : `'EMBL'`

Un enregistrement s'ouvre sur une ligne `ID` et se ferme sur une ligne `//` :

```
ID   AB012345; SV 1; linear; mRNA; STD; HUM; 233 BP.
AC   AB012345;
DT   01-JAN-2020 (Rel. 1, Created)
DE   Homo sapiens mRNA for tumor necrosis factor, complete cds.
KW   TNF.
OS   Homo sapiens (human)
OC   Eukaryota; Metazoa; Chordata; Mammalia; Primates.
RN   [1]
```

```
RP   1-233
RX   PUBMED; 2547453.
RA   Eck M.J., Sprang S.R.;
RT   "The structure of TNF";
RL   J. Biol. Chem. 264:17595-17605(1989).
FT   source          1..233
FT                   /organism="Homo sapiens"
FT   CDS             1..233
FT                   /product="tumor necrosis factor"
SQ   Sequence 233 BP;
     atgagcacag aaagcatgat ccgggacgtg gagctggccg
        40
//
```

Ce que le lecteur rend : les mêmes objets communs que Swiss-Prot et GenBank (section 2.3.3). Le lecteur EMBL remplit également `getGbSequence()` — topologie, division et version y sont renseignées, mais pas l'identifiant `GI` (propre à GenBank) ni le nombre de brins.

Note

Contrairement à GenBank, le lecteur EMBL n'impose aucune liste fermée de types de `FEATURES` : tout bloc `FT` rencontré est lu et apparaît dans `getFeatures()`, quel que soit son type. C'est l'inverse de la limitation signalée dans l'encadré « Attention » de la fiche GenBank.

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('sequence.embl');
$lecteur = DatabaseReaderFactory::readDatabase('EMBL', $aLignes);

$sequence = $lecteur->getSequence();
echo $sequence->getPrimAcc();        // "AB012345"
echo $sequence->getDescription();
echo $sequence->getSequence();

$gb = $lecteur->getGbSequence();
echo $gb->getTopology();             // "LINEAR"
echo $gb->getDivision();             // "HUM"

foreach ($lecteur->getFeatures() as $feature) {
    echo $feature->getFtKey() . " : " . $feature->getFtQual()
        . " = " . $feature->getFtValue() . "\n";
}

foreach ($lecteur->getReferences() as $reference) {
    echo $reference->getPubmed() . " - " . $reference->getJournal() . "\
       n";
}
```

À quoi ça sert concrètement : le même usage que GenBank (extraire séquence, organisme, caractéristiques, bibliographie), appliqué à des fichiers issus d'ENA ; retrouver directement l'identifiant PubMed d'une référence, lu depuis la ligne `RX` sans traitement supplémentaire.

Persistance : `$recordManager->find('EMBL', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'EMBL', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.4 PDB

Un enregistrement PDB (Protein Data Bank) décrit une structure tridimensionnelle résolue expérimentalement (cristallographie aux rayons X, RMN…) : les coordonnées atomiques de chaque résidu, l'organisation en hélices et feuillets, la ou les chaînes qui composent la structure. C'est un format très différent des trois précédents — on n'y lit pas une annotation de séquence, mais une géométrie.

Format à passer à `readDatabase()` : `'PDB'`

Un enregistrement s'ouvre sur une ligne `HEADER` et se ferme sur une ligne `END` :

```
HEADER    CYTOKINE                                01-JAN-95   1TNF
TITLE     TUMOR NECROSIS FACTOR-ALPHA
COMPND    MOL_ID: 1; MOLECULE: TUMOR NECROSIS FACTOR; CHAIN: A, B, C;
SOURCE    MOL_ID: 1; ORGANISM_SCIENTIFIC: HOMO SAPIENS;
KEYWDS    CYTOKINE, INFLAMMATION
EXPDTA    X-RAY DIFFRACTION
AUTHOR    M.J.ECK,S.R.SPRANG
SEQRES   1 A  157  VAL ARG SER SER SER ARG THR PRO SER ASP LYS PRO VAL
HELIX    1   1 SER A  109  ASN A  113  1
                                      5
SHEET    1   A 4 THR A  77  MET A  83  0
CRYST1   66.100   66.100  116.700  90.00  90.00 120.00 P 32 2 1      6
ATOM      1  N   VAL A   1      33.611  33.560  20.041  1.00 25.00
             N
END
```

Ce que le lecteur rend : pas d'objets communs ici — PDB décrit une géométrie, pas une séquence annotée, et n'hérite donc pas de `ParseDbAbstractManager`.

Méthode	Contenu
`getIdCode()`	Le code à 4 caractères de la structure (ex. `"1TNF"`)
`getClassification()` / `getTitle()` / `getDepositionDate()`	Classification, titre et date de dépôt
`getCompounds()` / `getSources()`	Tableaux de blocs (un par molécule) — champs `MOL_ID`, `MOLECULE`, `CHAIN` pour l'un, `ORGANISM_SCIENTIFIC` pour l'autre
`getKeywords()` / `getAuthors()`	Tableaux de chaînes
`getExperimentalTechnique()`	Ex. `"X-RAY DIFFRACTION"`
`getSeqRes()`	Tableau `chaîne => séquence` — la séquence protéique de chaque chaîne, reconstituée à partir des codes à 3 lettres
`getHelices()` / `getSheets()`	Tableaux de `PdbHelix` / `PdbSheet` (section 2.4)
`getCryst1()`	Tableau associatif — paramètres de la maille cristalline (`a`, `b`, `c`, `alpha`, `beta`, `gamma`, `spaceGroup`, `z`)
`getAtoms()` / `getHetAtoms()`	Tableaux de `PdbAtom` (section 2.4) — atomes de la chaîne principale / hétéro-atomes (ligands, ions, eau)

Attention

Seuls les types d'enregistrements les plus utiles à la bio-informatique structurale sont lus (`HEADER`, `TITLE`, `COMPND`, `SOURCE`, `KEYWDS`, `EXPDTA`, `AUTHOR`, `SEQRES`, `HELIX`, `SHEET`, `CRYST1`, `ATOM`, `HETATM`). Les nombreux autres types que compte le format PDB complet (`CONECT`, `ANISOU`, `MASTER`…) ne sont pas repris.

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('1tnf.pdb');
$lecteur = DatabaseReaderFactory::readDatabase('PDB', $aLignes);

echo $lecteur->getIdCode();                  // "1TNF"
echo $lecteur->getExperimentalTechnique();   // "X-RAY DIFFRACTION"

// La sequence de chaque chaine
foreach ($lecteur->getSeqRes() as $chaine => $sequence) {
    echo "Chaine $chaine : $sequence\n";
}

// Les helices
foreach ($lecteur->getHelices() as $helice) {
    printf(
        "Helice %s : residus %d a %d\n",
        $helice->getHelixId(),
        $helice->getInitSeqNum(),
        $helice->getEndSeqNum()
    );
}

// Distance entre deux atomes (calcul geometrique simple a partir des
// coordonnees x, y, z)
$aAtomes = $lecteur->getAtoms();
$premier = $aAtomes[0];
$dernier = end($aAtomes);
$distance = sqrt(
    ($premier->getX() - $dernier->getX()) ** 2
    + ($premier->getY() - $dernier->getY()) ** 2
    + ($premier->getZ() - $dernier->getZ()) ** 2
);
```

À quoi ça sert concrètement : récupérer la séquence réellement observée dans la structure, chaîne par chaîne (à comparer à la séquence théorique issue de GenBank ou Swiss-Prot) ; situer les hélices et feuillets pour comprendre le repliement ; exploiter les coordonnées atomiques pour des calculs géométriques (distances, contacts, centre de masse d'une chaîne) ; retrouver rapidement la technique expérimentale utilisée pour juger de la fiabilité des coordonnées.

Persistance : `$recordManager->find('PDB', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'PDB', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.5 PROSITE

Un enregistrement PROSITE ne décrit pas une séquence mais un *motif* (un « pattern », ou une matrice) — le schéma qui caractérise une famille de protéines ou un site fonctionnel. C'est l'outil

de référence pour retrouver rapidement la signature qui définit une famille connue et la confronter à une séquence.

Format à passer à `readDatabase()` : `'PROSITE'`

Un enregistrement s'ouvre sur une ligne `ID` et se ferme sur une ligne `//` :

```
ID   ATP_GTP_A; PATTERN.
AC   PS00017;
DT   APR-1990 (CREATED); DEC-2004 (DATA UPDATE); DEC-2004 (INFO UPDATE).
DE   ATP/GTP-binding site motif A (P-loop).
PA   [AG]-x(4)-G-K-[ST].
NR   /RELEASE=20.24,146629;   /TOTAL=602(578); /POSITIVE=583(559);
CC   /TAXO-RANGE=??????; /MAX-REPEAT=1;
3D   1YIC; 1TNF;
DR   P01375, TNFA_HUMAN, T; P00533, EGFR_HUMAN, T;
DO   PDOC00017;
//
```

Ce que le lecteur rend : pas d'objets communs — PROSITE décrit un motif, pas une séquence annotée.

Méthode	Contenu
`getEntryName()` / `getEntryType()` / `getAccession()`	Ex. `"ATP_GTP_A"`, `"PATTERN"`, `"PS00017"`
`getDates()`	Tableau événement (`CREATED`, `DATA UPDATE`…) => `date`
`getDescription()`	Description en une phrase
`getPattern()`	Le motif lui-même (champ `PA`)
`getMatrix()`	Champ `MA`, gardé en texte brut
`getNumericalResults()` / `getComments()`	Tableaux associatifs (`qualificatif => valeur`), champs `NR` et `CC`
`getRule()`	Champ `RU` — règle supplémentaire, le cas échéant
`getPdbXrefs()`	Tableau des codes PDB associés (champ `3D`)
`getDbRefs()`	Tableau de `PrositeDbRef` (section 2.4) — les protéines Swiss-Prot recensées comme correspondant (ou non) au motif
`getDocXref()`	Renvoi vers la documentation détaillée (champ `DO`)

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('ps00017.prosite');
$lecteur = DatabaseReaderFactory::readDatabase('PROSITE', $aLignes);

echo $lecteur->getEntryName();  // "ATP_GTP_A"
echo $lecteur->getDescription();
echo $lecteur->getPattern();    // "[AG]-x(4)-G-K-[ST]."

// Separer vrais et faux positifs parmi les proteines associees
foreach ($lecteur->getDbRefs() as $renvoi) {
    if ($renvoi->isTruePositive()) {
        echo "Vrai positif : " . $renvoi->getEntryName() . "\n";
    }
```

```
}
```

À quoi ça sert concrètement : récupérer le motif d'une famille pour le rechercher dans une séquence à soi (avec `SequenceMatchManager`, voir plus loin) ; croiser un motif avec les structures PDB qui l'illustrent ; séparer, parmi les protéines référencées, celles qui correspondent vraiment au motif de celles listées pour comparaison (faux positifs).

Persistance : `$recordManager->find('PROSITE', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'PROSITE', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.6 ExPASy ENZYME

Un enregistrement ExPASy ENZYME décrit une enzyme par son numéro EC (*Enzyme Commission*) — sa réaction catalysée, ses cofacteurs, les maladies liées à un déficit de cette enzyme, et ses renvois vers PROSITE et Swiss-Prot.

Format à passer à `readDatabase()` : `'EXPASY_ENZYME'`

Un enregistrement s'ouvre sur une ligne `ID` et se ferme sur une ligne `//` :

```
ID   3.1.1.3
DE   Triacylglycerol lipase.
AN   Triglyceride lipase.
CA   (1) Triacylglycerol + H2O = diacylglycerol + a carboxylate.
CF   Ca(2+).
CC   -!- Wide specificity.
DI   Lipase deficiency, congenital; MIM: 246650.
PR   PROSITE; PDOC00110;
DR   P00590, LIP_HUMAN;
//
```

Ce que le lecteur rend : pas d'objets communs — là encore, une enzyme décrite par sa réaction n'est pas une séquence annotée.

Méthode	Contenu
`getId()`	Le numéro EC (ex. `"3.1.1.3"`)
`getDescription()`	Nom recommandé de l'enzyme
`getAlternateNames()`	Tableau — synonymes (champ `AN`, un par ligne)
`getCatalyticActivities()`	Tableau — une entrée par réaction catalysée (une enzyme à plusieurs substrats en a plusieurs)
`getCofactors()`	Tableau (champ `CF`)
`getComments()`	Texte libre (champ `CC`)
`getDiseases()`	Tableau de `ExpasyDisease` (section 2.4)
`getPrositeRefs()`	Tableau des identifiants de documentation PROSITE liés
`getSwissprotRefs()`	Tableau `accession => nom d'entrée` — les protéines Swiss-Prot portant cette activité

Attention

Ne confondez pas ce lecteur avec `RestrictionEnzymeManager` (section 2.6.4) : ExPASy ENZYME couvre les enzymes du métabolisme en général (numérotation EC), alors que `RestrictionEnzymeManager` s'occupe spécifiquement des enzymes de restriction — celles

qui découpent l'ADN, déjà rencontrées au chapitre 1 avec `EcoRI`. Deux bases de données complètement différentes, malgré le nom proche.

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('3.1.1.3.txt');
$lecteur = DatabaseReaderFactory::readDatabase('EXPASY_ENZYME', $aLignes);

echo $lecteur->getId();                    // "3.1.1.3"
echo $lecteur->getDescription();
print_r($lecteur->getCatalyticActivities());

foreach ($lecteur->getDiseases() as $maladie) {
    echo $maladie->getDisease() . " (" . $maladie->getReference() . ")\n";
}
```

À quoi ça sert concrètement : retrouver une enzyme par son numéro EC et lire sa réaction ; lister ses cofacteurs nécessaires ; remonter aux maladies liées à un déficit de cette enzyme ; retrouver les protéines Swiss-Prot qui la portent, sans avoir à chercher séparément.

Persistance : `$recordManager->find('EXPASY_ENZYME', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'EXPASY_ENZYME', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.7 PDBSTR

Un enregistrement PDBSTR décrit un membre d'une famille structurale dérivée de PDB — c'est un regroupement de structures similaires, pas la structure elle-même (pour ça, voir `PDB` ci-dessus).

Format à passer à `readDatabase()` : `'PDBSTR'`

Un enregistrement tient sur une seule ligne `MEMBER` et se ferme sur une ligne `//` :

```
MEMBER      1YIC_01       108     PROTEIN       1YIC   97/02/18   97/07/23
//
```

Ce que le lecteur rend :

Méthode	Contenu
`getMemberId()`	Identifiant du membre (ex. `"1YIC_01"`)
`getLength()`	Longueur, en résidus
`getMolType()`	Type de molécule (ex. `"PROTEIN"`)
`getEntryGroup()`	Code PDB du groupe auquel appartient le membre
`getCreateDate()` / `getUpdDate()`	Dates de création et de dernière mise à jour

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('membre.pdbstr');
$lecteur = DatabaseReaderFactory::readDatabase('PDBSTR', $aLignes);

```

```php
echo $lecteur->getMemberId();     // "1YIC_01"
echo $lecteur->getEntryGroup();   // "1YIC"
echo $lecteur->getUpdDate();      // "97/07/23"
```

À quoi ça sert concrètement : regrouper les structures d'une même famille sous un groupe commun ; suivre la date de dernière mise à jour d'un membre pour savoir si une structure plus récente existe.

Persistance : `$recordManager->find('PDBSTR', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'PDBSTR', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.8 PRINTS

Un enregistrement PRINTS décrit une *empreinte* (« fingerprint ») de famille de protéines — un ensemble de plusieurs motifs conservés qui, pris ensemble, identifient la famille avec plus de précision qu'un motif unique (comme celui de PROSITE).

Format à passer à `readDatabase()` : `'PRINTS'`

Un enregistrement s'ouvre sur une ligne `gc;` :

```
gc; TNFALPHA
gn; TNFALPHA
ga; 16-NOV-1995; UPDATE 06-JUN-1999
gd; Tumor necrosis factor alpha fingerprint.
```

Attention

PRINTS ne porte aucun marqueur de fin d'enregistrement : un flux qui en contiendrait plusieurs ne peut pas être découpé automatiquement, et la lecture s'arrête à la fin du fichier. En pratique, traitez un fichier PRINTS à la fois par appel.

Ce que le lecteur rend :

Méthode	Contenu
`getEntryName()` / `getEntryType()`	Champs `gc;` et `gn;`
`getCreateDate()` / `getUpdDate()`	Lues sur le champ `ga;`
`getDescription()`	Champ `gd;`

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('tnfalpha.prints');
$lecteur = DatabaseReaderFactory::readDatabase('PRINTS', $aLignes);

echo $lecteur->getEntryName();    // "TNFALPHA"
echo $lecteur->getDescription();
echo $lecteur->getUpdDate();      // "06-JUN-1999"
```

À quoi ça sert concrètement : identifier rapidement une empreinte de famille par son nom et sa description ; vérifier la fraîcheur de l'entrée avant de s'y fier.

Persistance : `$recordManager->find('PRINTS', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'PRINTS', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.9 BLOCKS

Un enregistrement BLOCKS décrit un *bloc* — une région conservée sans trou (« ungapped ») d'une famille de protéines, positionnée à une certaine distance du bloc précédent de la même famille.

Format à passer à `readDatabase()` : `'BLOCKS'`

Un enregistrement s'ouvre sur une ligne `ID` et se ferme sur une ligne `//` :

```
ID   TNF_FAMILY; BLOCK
AC   IPB002128C; distance from previous block=(30,31)
DE   Tumor necrosis factor family signature
BL   THR; width=15; seqs=12; 99.5%;
//
```

Ce que le lecteur rend :

Méthode	Contenu
`getId()` / `getAccession()`	Nom et accession du bloc
`getDistMin()` / `getDistMax()`	Distance minimale et maximale, en résidus, par rapport au bloc précédent de la famille
`getDescription()`	Champ `DE`
`getAaTriplet()`	Premier élément du champ `BL` (ex. `"THR"`)

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('tnf_family.blocks');
$lecteur = DatabaseReaderFactory::readDatabase('BLOCKS', $aLignes);

echo $lecteur->getId();                          // "TNF_FAMILY"
printf("Distance au bloc precedent : %d-%d\n", $lecteur->getDistMin(),
    $lecteur->getDistMax());
```

À quoi ça sert concrètement : repérer les blocs conservés d'une famille de protéines et leur espacement, pour reconstituer l'organisation d'ensemble de la famille.

Persistance : `$recordManager->find('BLOCKS', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'BLOCKS', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.10 ProDom

Un enregistrement ProDom décrit une famille de *domaines* protéiques — une région qui se retrouve, répétée ou non, dans plusieurs protéines différentes.

Format à passer à `readDatabase()` : `'PRODOM'`

Un enregistrement s'ouvre sur une ligne `ID` et se ferme sur une ligne `//` :

```
ID 20167 p2002.1                              10 seq.
AC PD000001
KW   FADR(2) Y586(1) // COMPLETE PROTEOME DNA-BINDING FATTY
//
```

Ce que le lecteur rend :

Méthode	Contenu
`getEntryNo()` / `getAccession()`	Numéro d'entrée et accession
`getRelease()`	Version de la base (le `"p"` de tête est retiré)
`getDomainCount()`	Nombre de séquences dans la famille
`getFreqNames()`	Tableau `nom => occurrences` — les noms de protéines les plus fréquents dans la famille
`getKeywords()`	Tableau — mots-clés (seconde moitié du champ `KW`, après le `//`)

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('pd000001.prodom');
$lecteur = DatabaseReaderFactory::readDatabase('PRODOM', $aLignes);

echo $lecteur->getAccession();        // "PD000001"
echo $lecteur->getDomainCount();      // 10

// Les noms de proteines les plus frequents de la famille
arsort($noms = $lecteur->getFreqNames());
foreach ($noms as $nom => $occurrences) {
    echo "$nom : $occurrences\n";
}
```

À quoi ça sert concrètement : compter les domaines d'une famille et voir les noms de protéines les plus fréquents pour identifier rapidement de quoi il s'agit ; retrouver les mots-clés associés.

Persistance : `$recordManager->find('PRODOM', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'PRODOM', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.11 AAINDEX

Un enregistrement AAINDEX (AAINDEX1) décrit une propriété physico-chimique numérique des vingt acides aminés — hydrophobicité, volume, charge…— mesurée ou calculée dans une publication donnée.

Format à passer à `readDatabase()` : `'AAINDEX'`

Un enregistrement s'ouvre sur une ligne `H` et se ferme sur une ligne `//` :

```
H ARGP820101
D Hydrophobicity index (Argos et al., 1982)
A Argos, P., Rao, J.K.M. and Hargrave, P.K.
T Structural prediction of membrane-bound proteins
J Eur. J. Biochem. 128, 565-575 (1982)
```

```
R  LIT:1810048b PMID:1575719
I     A/L     R/K     N/M     D/F     C/P     Q/S     E/T     G/W     H/Y
         I/V
     0.61    0.60    0.06    0.46    1.07    0.00    0.47    0.07    0.61
             2.22
//
```

Ce que le lecteur rend :

Méthode	Contenu
`getAccession()`	Identifiant de l'indice (champ `H`)
`getDescription()`	Champ `D`
`getAuthor()` / `getTitle()` / `getJournal()`	Publication d'origine (champs `A`, `T`, `J`)
`getLitRefs()`	Tableau `base => identifiant` (champ `R`, ex. `"PMID" => "1575719"`)

Note

Les valeurs numériques de l'indice lui-même (champ `I`, vingt valeurs à raison d'une par acide aminé) ne sont pas décomposées : c'est un tableau fixe de vingt nombres que le lecteur ne fait pas apparaître en `getter` dédié.

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('argp820101.aaindex');
$lecteur = DatabaseReaderFactory::readDatabase('AAINDEX', $aLignes);

echo $lecteur->getAccession();     // "ARGP820101"
echo $lecteur->getDescription();   // "Hydrophobicity index (Argos et al
    ., 1982)"
echo $lecteur->getJournal();

print_r($lecteur->getLitRefs());   // ["LIT" => "1810048b", "PMID" =>
    "1575719"]
```

À quoi ça sert concrètement : retrouver un indice physico-chimique des acides aminés par son identifiant ; remonter à la publication qui l'a mesuré ou calculé, pour en vérifier la méthodologie avant de l'utiliser dans une analyse.

Persistance : `$recordManager->find('AAINDEX', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'AAINDEX', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.12 PIR

Un enregistrement PIR (Protein Information Resource) décrit une protéine annotée, dans une grammaire particulière : chaque champ qualifie ses données avec des paires `#clé valeur`, plutôt que d'utiliser un champ dédié par information comme le font GenBank ou Swiss-Prot.

Format à passer à `readDatabase()` : `'PIR'`

Un enregistrement s'ouvre sur une ligne `ENTRY` et se ferme sur une ligne `//` :

```
ENTRY            RHTDTO  #type complete
TITLE            tumor necrosis factor alpha precursor - human
ORGANISM         #formal_name Homo sapiens #common_name human
DATE             15-Jun-2001 #sequence_revision 15-Jun-2001 #text_change
    15-Jun-2001
ACCESSIONS       A12345; B67890
KEYWORDS         cytokine; glycoprotein
SUMMARY          #length 233  #molecular-weight 25644  #checksum 4657
//
```

Ce que le lecteur rend :

Méthode	Contenu
`getEntryName()` / `getEntryType()`	Lus sur `ENTRY` (nom, puis qualificatif `#type`)
`getTitle()`	Champ `TITLE`
`getAccessions()`	Tableau, champ `ACCESSIONS`
`getOrganism()` / `getSpecies()`	Noms commun et scientifique (qualificatifs `#common_name` / `#formal_name` de `ORGANISM`)
`getCreateDate()` / `getSeqrevDate()` / `getTxtchgDate()`	Trois dates du champ `DATE` (création, révision de séquence, changement de texte)
`getLength()` / `getMolwt()` / `getChecksum()`	Qualificatifs du champ `SUMMARY`
`getKeywords()`	Tableau, champ `KEYWORDS`

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('rhtdto.pir');
$lecteur = DatabaseReaderFactory::readDatabase('PIR', $aLignes);

echo $lecteur->getEntryName();   // "RHTDTO"
echo $lecteur->getOrganism();    // "human"
echo $lecteur->getLength();      // 233
echo $lecteur->getChecksum();    // "4657"
```

À quoi ça sert concrètement : lire une entrée PIR malgré sa grammaire particulière aux qualificatifs ; récupérer masse et longueur d'une protéine sans calcul ; vérifier l'intégrité d'un fichier téléchargé via la somme de contrôle (`checksum`).

Persistance : `$recordManager->find('PIR', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'PIR', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.13 PRF

Un enregistrement PRF (Protein Research Foundation, aussi appelé SEQDB) décrit une protéine publiée : sa séquence, sa taxonomie, et la référence bibliographique qui l'a rendue publique.

Format à passer à `readDatabase()` : `'PRF'`

Un enregistrement s'ouvre sur une ligne `CODE` et se ferme sur une ligne `///` (trois barres, comme PMD — pas deux comme la famille GenBank) :

```
CODE        0901234A
NAME        TUMOR NECROSIS FACTOR ALPHA
SOURCE      HUMAN
cname       HOMO SAPIENS
taxon       EUKARYOTA; METAZOA; CHORDATA; MAMMALIA
JOURNAL     J. Biol. Chem.
AUTHOR      Eck,M.J., Sprang,S.R.
TITLE       The structure of TNF
KEYWORD     cytokine  inflammation
CROSSREF    PIR=ICHU2;PIR=ICGI2
SEQUENCE    MSTESMIRDV ELAEEALPKK TGGPQGSRRC
///
```

Ce que le lecteur rend :

Méthode	Contenu
`getEntryCode()` / `getEntryName()`	Champs `CODE` et `NAME`
`getSource()` / `getCommonName()`	Champs `SOURCE` et `cname`
`getTaxonomy()`	Tableau, champ `taxon`
`getJournal()` / `getAuthors()` / `getTitle()`	Référence bibliographique
`getKeywords()`	Tableau, champ `KEYWORD`
`getComment()`	Champ `COMMENT`
`getCrossRefs()`	Tableau de paires `["db" => ..., "id" => ...]`, champ `CROSSREF`
`getSequence()`	La séquence protéique elle-même

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('0901234a.prf');
$lecteur = DatabaseReaderFactory::readDatabase('PRF', $aLignes);

echo $lecteur->getEntryName();  // "TUMOR NECROSIS FACTOR ALPHA"
echo $lecteur->getSequence();
print_r($lecteur->getTaxonomy());

foreach ($lecteur->getCrossRefs() as $renvoi) {
    echo $renvoi['db'] . " : " . $renvoi['id'] . "\n";
}
```

À quoi ça sert concrètement : récupérer la séquence d'une protéine publiée, sa taxonomie et sa référence bibliographique ; retrouver les identifiants équivalents dans d'autres banques via les renvois croisés.

Persistance : `$recordManager->find('PRF', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'PRF', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.14 PMD

Un enregistrement PMD (Protein Mutant Database) décrit une mutation rapportée dans la littérature pour une protéine donnée, avec l'article qui la documente.

Format à passer à `readDatabase()` : `'PMD'`

Un enregistrement s'ouvre sur une ligne `ENTRY` et se ferme sur une ligne `///` (comme PRF) :

```
ENTRY            A000300 - Artificial                        2607383
AUTHORS          Eck,M.J. & Sprang,S.R.
MEDLINE          89123456
JOURNAL          J. Biol. Chem.
TITLE            Engineering TNF variants
///
```

Ce que le lecteur rend :

Méthode	Contenu
`getEntryType()` / `getEntryNo()`	Premier caractère et six suivants du champ `ENTRY`
`getMutationType()` / `getArticleNo()`	Reste du champ `ENTRY`
`getAuthors()`	Tableau, champ `AUTHORS`
`getMedlineNo()`	Champ `MEDLINE`
`getJournal()` / `getTitle()`	Référence bibliographique

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('a000300.pmd');
$lecteur = DatabaseReaderFactory::readDatabase('PMD', $aLignes);

echo $lecteur->getEntryNo();        // le numero d'entree
echo $lecteur->getMutationType();
echo $lecteur->getTitle();
```

À quoi ça sert concrètement : retrouver les mutations décrites pour une protéine et l'article scientifique qui les rapporte, pour une étude de structure-fonction par exemple.

Persistance : `$recordManager->find('PMD', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'PMD', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.15 ENTREZ

Un enregistrement Entrez décrit un génome dans le système NCBI Entrez, avec les mêmes colonnes fixes que GenBank pour l'en-tête (`LOCUS`, `DEFINITION`, `ACCESSION`…) mais sans table `FEATURES` ni séquence `ORIGIN` : c'est de l'annotation pure.

Format à passer à `readDatabase()` : `'ENTREZ'`

Un enregistrement s'ouvre sur une ligne `LOCUS` et se ferme sur une ligne `//` :

```
LOCUS       NC_001416              48502 bp    DNA     linear   PHG 01-
   JAN-2020
DEFINITION  Enterobacteria phage lambda, complete genome.
ACCESSION   NC_001416
VERSION     NC_001416.1  GI:9626243
KEYWORDS    complete genome.
SOURCE      Escherichia phage lambda
  ORGANISM  Escherichia phage lambda
            Viruses; Duplodnaviria; Heunggongvirae.
REFERENCE   1  (bases 1 to 48502)
  AUTHORS   Sanger,F., Coulson,A.R., Hong,G.F. and Petersen,G.B.
  TITLE     Nucleotide sequence of bacteriophage lambda DNA
  JOURNAL   J. Mol. Biol. 162:729-773(1982)
//
```

Ce que le lecteur rend : pas les objets communs de la section 2.3.3 (ce lecteur n'hérite pas de `ParseDbAbstractManager`, faute de table `FEATURES`) mais un jeu de méthodes très proche :

Méthode	Contenu
`getEntryName()` / `getMolType()` / `getLength()`	Lus sur la ligne `LOCUS`
`getEntryDate()` / `getDivision()` / `getTopology()` / `getStrands()`	Idem, autres colonnes de `LOCUS`
`getDefinition()`	Champ `DEFINITION`
`getPrimAcc()` / `getAccession()`	Première et toutes les accessions (champ `ACCESSION`)
`getVersion()` / `getNcbiGiId()`	Champ `VERSION`
`getKeywords()`	Tableau, champ `KEYWORDS`
`getSource()` / `getOrganism()` / `getTaxonomy()`	Champs `SOURCE` et `ORGANISM`
`getReferences()`	Tableau de `EntrezReference` (section 2.4)

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('nc_001416.entrez');
$lecteur = DatabaseReaderFactory::readDatabase('ENTREZ', $aLignes);

echo $lecteur->getEntryName();    // "NC_001416"
echo $lecteur->getDefinition();
echo $lecteur->getLength();       // 48502

foreach ($lecteur->getReferences() as $reference) {
    echo $reference->getTitle() . " - " . $reference->getJournal() . "\n
        ";
}
```

Note

Un fichier Entrez ressemble beaucoup à un fichier GenBank (mêmes colonnes d'en-tête), mais les deux lecteurs rendent des objets différents : `ENTREZ` n'a pas de table `FEATURES` ni de séquence, là où `GENBANK` (section 2.5.2) a les deux. Choisissez le format selon ce que contient réellement votre fichier, pas seulement selon son apparence.

À quoi ça sert concrètement : lire l'annotation d'un enregistrement génomique NCBI Entrez quand le fichier ne porte pas de table `FEATURES` ni de séquence à proprement parler ; récupérer rapidement définition, organisme et bibliographie.

Persistance : `$recordManager->find('ENTREZ', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'ENTREZ', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.16 GENOME

Un enregistrement GENOME résume l'avancement du séquençage d'un organisme : taille du génome, nombre d'entrées GenBank associées, degré d'achèvement, et les publications qui en rendent compte.

Format à passer à `readDatabase()` : `'GENOME'`

Un enregistrement s'ouvre sur une ligne `ORGANISM` et se ferme sur une ligne `//`. Les lignes de continuation sont indentées d'une tabulation :

```
ORGANISM        Escherichia coli
COMMON_NAME     E. coli
CLASSIFICATION  Bacteria; Proteobacteria; Gammaproteobacteria
COMPLETED       Yes
GB_RELEASE      250
GB_ENTRIES      12
GB_BASEPAIRS    4641652
GENOME_SIZE     4641652
REF_TYPE        Journal Article
REF_AUTHOR      Blattner,F.R.; Plunkett,G.
REF_TITLE       The complete genome sequence of Escherichia coli K-12
REF_JOURNAL     Science
REF_VOLUME      277
REF_PAGES       1453-1462
REF_YEAR        1997
//
```

Ce que le lecteur rend :

Méthode	Contenu
`getOrganism()` / `getCommonName()`	Noms scientifique et commun
`getTaxClass()`	Tableau, champ `CLASSIFICATION`
`getIsComplete()`	Champ `COMPLETED`, tel qu'écrit
`getGbRelease()` / `getGbEntries()` / `getGbBasepairs()`	État de la couverture GenBank
`getSize()`	Taille du génome haploïde, en paires de bases

Méthode	Contenu
`getReferences()`	Tableau de `GenomeReference` (section 2.4) — un enregistrement peut en compter plusieurs

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('ecoli.genome');
$lecteur = DatabaseReaderFactory::readDatabase('GENOME', $aLignes);

echo $lecteur->getOrganism();      // "Escherichia coli"
echo $lecteur->getIsComplete();    // "Yes"
echo $lecteur->getSize();          // 4641652

foreach ($lecteur->getReferences() as $reference) {
    echo $reference->getTitle() . " (" . $reference->getYear() . ")\n";
}
```

À quoi ça sert concrètement : comparer l'avancement du séquençage de plusieurs organismes ; retrouver rapidement la publication de référence d'un génome.

Persistance : `$recordManager->find('GENOME', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'GENOME', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.17 HGBase

Un enregistrement HGBase décrit un variant génétique humain bi-allélique (une mutation ponctuelle) et sa fréquence dans une population donnée.

Format à passer à `readDatabase()` : `'HGBASE'`

Un enregistrement s'ouvre sur une ligne `HAPLOTYPEID` et se ferme sur une ligne `//` :

```
HAPLOTYPEID      HGVBASE:12345
ALLELE           A/G
ISINBLOCK        Yes
POPULATIONID     POP0042
POPULATION       Caucasian (USA) (216 individuals)
FREQUENCY        2% (1039 individuals)
SOURCEID         SRC0099
CITATION         Smith et al. 2001
SUBMITTER        Jan. W. Koper (SUB0001234)
SOURCECOMMENT    Genotyped by TaqMan assay
//
```

Ce que le lecteur rend :

Méthode	Contenu
`getHaplotypeId()` / `getAllele()` / `getIsInBlock()`	Identification du variant

Méthode	Contenu
`getPopulationId()` / `getPopName()` / `getPopIndiv()`	Population observée et son effectif
`getFreqPerc()` / `getFreqIndiv()`	Fréquence de l'allèle, en pourcentage et en effectif
`getSourceId()` / `getCitation()`	Référence de la source
`getSubmitterName()` / `getSubmissionId()`	Qui a soumis l'entrée, et sous quel numéro
`getSourceComment()`	Commentaire libre sur la source

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('hgvbase_12345.txt');
$lecteur = DatabaseReaderFactory::readDatabase('HGBASE', $aLignes);

printf(
    "%s : frequence %.1f%% sur %d individus, population %s\n",
    $lecteur->getAllele(),
    $lecteur->getFreqPerc(),
    $lecteur->getFreqIndiv(),
    $lecteur->getPopName()
);
```

À quoi ça sert concrètement : lire les fréquences alléliques d'un variant par population ; remonter au soumetteur et à la méthode de génotypage utilisée.

Persistance : `$recordManager->find('HGBASE', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'HGBASE', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.18 EPD

Un enregistrement EPD (Eukaryotic Promoter Database) décrit un promoteur eucaryote, dans une grammaire à étiquettes sur deux lettres proche de celle d'EMBL.

Format à passer à `readDatabase()` : `'EPD'`

Un enregistrement s'ouvre sur une ligne `ID` et se ferme sur une ligne `//` :

```
ID   HS_ACHA_1      standard; single; HUM.
AC   EP73001;
DT   15-JUN-2001 (REL. 75, CREATED)
DT   20-JUL-2010 (REL. 90, LAST SEQUENCE UPDATE)
DE   Human acetylcholine receptor alpha subunit promoter.
CC   TATA-less promoter.
//
```

Ce que le lecteur rend :

Méthode	Contenu
`getEntryName()` / `getDataClass()` / `getInsiteType()` / `getTaxDiv()`	Lus sur la ligne `ID`
`getAccessions()`	Tableau, champ `AC`
`getCreateDate()` / `getCreateRel()`	Date et version de création
`getSequpdDate()` / `getSequpdRel()`	Date et version de la dernière mise à jour de séquence
`getNotupdDate()` / `getNotupdRel()`	Date et version de la dernière mise à jour d'annotation
`getDescription()` / `getComments()`	Champs `DE` et `CC`

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('ep73001.epd');
$lecteur = DatabaseReaderFactory::readDatabase('EPD', $aLignes);

echo $lecteur->getEntryName();     // "HS_ACHA_1"
echo $lecteur->getDataClass();     // "standard"
echo $lecteur->getSequpdDate();    // "20-JUL-2010"
```

À quoi ça sert concrètement : retrouver un promoteur eucaryote, sa classe et sa taxonomie ; connaître sa fraîcheur avant de s'y fier, comme pour Swiss-Prot.

Persistance : `$recordManager->find('EPD', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'EPD', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.19 UniGene

Un enregistrement UniGene regroupe les séquences que l'on pense provenir d'un même gène (un *cluster*) : dans quels tissus il s'exprime, et à quelles protéines il ressemble.

Format à passer à `readDatabase()` : `'UNIGENE'`

Un enregistrement s'ouvre sur une ligne `ID` et se ferme sur une ligne `//` :

```
ID          Hs.1
TITLE       Tumor necrosis factor
EXPRESS     liver; kidney; brain
PROTSIM     ORG=Mus musculus; PROTGI=12345; PROTID=NP_001.1; PCT=87; ALN
   =200
SCOUNT      42
//
```

Ce que le lecteur rend :

Méthode	Contenu
`getClusterId()` / `getTitle()`	Identifiant et titre du cluster
`getExpression()`	Tableau des tissus (champ `EXPRESS`)
`getProtSims()`	Tableau, une ligne `PROTSIM` brute par entrée (protein similarities)
`getSeqCount()`	Nombre de séquences du cluster (champ `SCOUNT`)

Note

Les lignes `PROTSIM` sont gardées telles quelles (texte brut) plutôt que décomposées champ par champ : leur mise en forme varie d'un fichier à l'autre.

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('hs.1.unigene');
$lecteur = DatabaseReaderFactory::readDatabase('UNIGENE', $aLignes);

echo $lecteur->getTitle();           // "Tumor necrosis factor"
print_r($lecteur->getExpression()); // ["liver", "kidney", "brain"]
echo $lecteur->getSeqCount();        // 42
```

À quoi ça sert concrètement : voir dans quels tissus un cluster de gène s'exprime ; savoir combien de séquences le composent, et à quelles protéines d'autres organismes il ressemble.

Persistance : `$recordManager->find('UNIGENE', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'UNIGENE', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.5.20 NCBI_LIT

Un enregistrement NCBI_LIT décrit une revue scientifique référencée par le NCBI — son titre complet, ses abréviations, ses numéros ISSN/ESSN.

Format à passer à `readDatabase()` : `'NCBI_LIT'`

Un enregistrement s'ouvre sur une ligne `JrId:` et se ferme sur une ligne de tirets — le seul format de la bibliothèque à ne pas fermer sur `//` ni sur `///` :

```
JrId: 12345
JournalTitle: Journal of Biological Chemistry
MedAbbr: J Biol Chem
ISSN: 0021-9258
ESSN: 1083-351X
IsoAbbr: J. Biol. Chem.
NlmId: 2985121R
----------
```

Ce que le lecteur rend :

Méthode	Contenu
`getId()`	Champ `JrId`
`getTitle()`	Titre complet de la revue

Méthode	Contenu
`getMedAbbr()` / `getIsoAbbr()`	Deux abréviations différentes (médicale, ISO)
`getIssn()` / `getEssn()`	Numéros ISSN imprimé et électronique
`getNlmId()`	Identifiant à la National Library of Medicine

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('journaux.txt');
$lecteur = DatabaseReaderFactory::readDatabase('NCBI_LIT', $aLignes);

echo $lecteur->getTitle();     // "Journal of Biological Chemistry"
echo $lecteur->getMedAbbr();   // "J Biol Chem"
echo $lecteur->getIssn();      // "0021-9258"
```

À quoi ça sert concrètement : résoudre une abréviation de revue rencontrée dans une référence bibliographique (par exemple issue d'un champ `getJournal()` de GenBank ou Swiss-Prot) vers son titre complet et ses identifiants normalisés.

2.5.21 La famille TRANSFAC

TRANSFAC décrit la régulation des gènes : facteurs de transcription, leurs sites de fixation sur l'ADN, les gènes qu'ils régulent. La banque publie un fichier par type d'information (`matrix.dat`, `gene.dat`, `class.dat`, `cell.dat`, `factor.dat`, `site.dat`), tous écrits dans la même grammaire : une étiquette sur deux lettres en début de ligne, sa donnée à partir de la cinquième colonne, et une ligne `//` qui ferme l'enregistrement. BioPHP donne donc à cette famille un lecteur par fichier, tous bâtis sur une base commune : chacun connaît déjà l'accession (`getAccession()`), l'identifiant (`getId()`) et les deux dates de création et de mise à jour (`getDateCreated()`/`getDateUpdated()`) — seuls les champs propres à chaque fichier sont à découvrir fiche par fiche.

Persistance : `$recordManager->find('NCBI_LIT', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'NCBI_LIT', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

TRANSFAC_MATRIX

Décrit la matrice de poids-position d'un facteur de transcription — pour chaque position du site de fixation, combien de fois chaque nucléotide (A, C, G, T) a été observé.

Format : `'TRANSFAC_MATRIX'` — ouvre sur `AC`, ferme sur `//`.

```
AC  M00001
ID  V$MYOD_01
DT  20.06.1990 (created); ewi.
NA  MyoD
DE  Myoblast determination protein
01      1       2       2       0       N
02      0       0       4       0       C
BA  15 sequences
CC  quality: A
//
```

Méthode	Contenu
`getName()`	Champ `NA`
`getDescription()`	Champ `DE`
`getMatrix()`	Tableau — une ligne par position, chacune `["A"=>..., "C"=>..., "G"=>..., "T"=>..., "consensus"=>...]`
`getBasis()`	Champ `BA` — base de calcul de la matrice
`getComments()`	Champ `CC`

```php
<?php
use Amelaye\BioPHP\Domain\Database\Factory\DatabaseReaderFactory;

$aLignes = file('m00001.transfac');
$lecteur = DatabaseReaderFactory::readDatabase('TRANSFAC_MATRIX',
    $aLignes);

echo $lecteur->getName();  // "MyoD"
foreach ($lecteur->getMatrix() as $position => $ligne) {
    printf("Position %d : A=%d C=%d G=%d T=%d\n",
        $position, $ligne['A'], $ligne['C'], $ligne['G'], $ligne['T']);
}
```

À quoi ça sert : récupérer une matrice de poids-position pour rechercher des sites de fixation potentiels dans une séquence (scan par glissement de fenêtre, en dehors de BioPHP).

Persistance : `$recordManager->find('TRANSFAC_MATRIX', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'TRANSFAC_MATRIX', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

TRANSFAC_GENE

Décrit un gène dont TRANSFAC documente la régulation.

Format : `'TRANSFAC_GENE'` — ouvre sur `AC`, ferme sur `//`.

```
AC  G000123
ID  G_TNF
SD  TNF
DE  Tumor necrosis factor gene
OS  human, homo sapiens
OC  Eukaryota; Metazoa; Chordata; Mammalia
CO  CE0000123
//
```

Méthode	Contenu
`getShortDescription()` / `getDescription()`	Champs `SD` et `DE`
`getOrganism()` / `getSpecies()`	Noms commun et scientifique (champ `OS`)
`getTaxClass()`	Tableau, champ `OC`
`getCompelAccessions()`	Tableau des accessions COMPEL liées (éléments composites, champ `CO`)

```php
<?php
$lecteur = DatabaseReaderFactory::readDatabase('TRANSFAC_GENE', file('
    g000123.transfac'));
echo $lecteur->getShortDescription();   // "TNF"
echo $lecteur->getOrganism();           // "human"
```

À quoi ça sert : identifier rapidement le gène régulé derrière un identifiant TRANSFAC, et sa classification taxonomique.

Persistance : `$recordManager->find('TRANSFAC_GENE', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'TRANSFAC_GENE', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

TRANSFAC_CLASS

Décrit une classe structurale de facteurs de transcription (doigts de zinc, glissières à leucine, hélice-tour-hélice…) et liste les facteurs qui en font partie.

Format : `'TRANSFAC_CLASS'` — ouvre sur `AC`, ferme sur `//`.

```
AC  C0001
ID  Basic domains
CL  1; Basic domains; 1.1; Leucine zipper factors
SD  bZIP transcription factors
BF  T00001 T00002 T00003
//
```

Méthode	Contenu
`getClassification()`	Tableau — la classification, du niveau le plus large au plus précis (champ `CL`)
`getStructuralDescription()`	Champ `SD`
`getMemberFactors()`	Tableau des identifiants de facteurs membres (champ `BF`)
`getComments()`	Champ `CC`

```php
<?php
$lecteur = DatabaseReaderFactory::readDatabase('TRANSFAC_CLASS', file('
    c0001.transfac'));
foreach ($lecteur->getMemberFactors() as $facteur) {
    echo $facteur . "\n";
}
```

À quoi ça sert : situer un facteur dans sa classe structurale ; lister tous les facteurs partageant la même famille de domaine de liaison à l'ADN.

Persistance : `$recordManager->find('TRANSFAC_CLASS', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'TRANSFAC_CLASS', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

TRANSFAC_CELL

Décrit le type ou la lignée cellulaire dont provient un facteur de transcription — ce qui relie une observation de fixation au tissu dans lequel elle a été faite.

Format : `'TRANSFAC_CELL'` — ouvre sur `AC`, ferme sur `//`.

```
AC  T00012
ID  HeLa
OS  human
SO  cervical carcinoma
CD  Immortalized cervical cancer cell line
//
```

Méthode	Contenu
`getOrganism()`	Champ `OS`
`getFactorSource()`	Champ `SO` — origine du facteur
`getDescription()`	Champ `CD`

```php
<?php
$lecteur = DatabaseReaderFactory::readDatabase('TRANSFAC_CELL', file('
    t00012.transfac'));
echo $lecteur->getOrganism() . " - " . $lecteur->getFactorSource();
```

À quoi ça sert : savoir de quel type cellulaire provient un facteur observé, pour interpréter correctement un site de fixation décrit dans TRANSFAC_SITE.

Persistance : `$recordManager->find('TRANSFAC_CELL', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'TRANSFAC_CELL', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

TRANSFAC_FACTOR

La fiche la plus complète de la famille : décrit un facteur de transcription — son nom, ses synonymes, l'organisme dont il provient, sa classe structurale et ses homologues chez d'autres espèces.

Format : `'TRANSFAC_FACTOR'` — ouvre sur `AC`, ferme sur `//`.

```
AC  T00001
ID  MyoD
FA  MyoD1
SY  bHLHc1; MYOD1
OS  human, homo sapiens
OC  Eukaryota; Metazoa; Chordata; Mammalia
HO  mouse MyoD, rat MyoD
CL  C0001; Basic domains; 1.1
SQ  MELLSPPLRDVDLTAPDGSLCSFATTDDFYD
//
```

Méthode	Contenu
`getFactorName()`	Champ `FA`
`getSynonyms()`	Tableau, champ `SY`

Méthode	Contenu
`getOrganism()` / `getSpecies()`	Champ `OS`
`getTaxClass()`	Tableau, champ `OC`
`getHomologs()`	Tableau, champ `HO`
`getClassAccession()` / `getClassId()` / `getClassDecimalNo()`	Renvoi vers la classe structurale (champ `CL`, à rapprocher de TRANSFAC_CLASS)
`getSequence()`	Champ `SQ` — séquence du facteur, si fournie

```php
<?php
$lecteur = DatabaseReaderFactory::readDatabase('TRANSFAC_FACTOR', file('t00001.transfac'));
echo $lecteur->getFactorName();  // "MyoD1"
print_r($lecteur->getSynonyms());
print_r($lecteur->getHomologs());
```

À quoi ça sert : obtenir la fiche d'un facteur de transcription, ses synonymes (utiles pour recouper avec une autre banque) et ses homologues chez d'autres espèces.

Persistance : `$recordManager->find('TRANSFAC_FACTOR', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'TRANSFAC_FACTOR', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

TRANSFAC_SITE

Décrit un site de fixation observé — la portion d'ADN reconnue par un facteur, sa position par rapport au début de la transcription du gène, et la méthode expérimentale qui l'a mis en évidence.

Format : `'TRANSFAC_SITE'` — ouvre sur `AC`, ferme sur `//`.

```
AC  R00001
ID  MyoD_site_1
TY  D
DE  MyoD binding site in the myogenin promoter
RE  myogenin promoter
SQ  CACCTGT
SF  -120
BF  T00001
OS  human
MM  gel shift assay
//
```

Méthode	Contenu
`getSeqType()`	Champ `TY` (ex. `"D"` pour un élément ADN issu d'un gène)
`getDescription()` / `getGeneRegion()`	Champs `DE` et `RE`
`getSequence()`	Champ `SQ`
`getFirstPosition()`	Champ `SF` — position, comptée depuis le site d'initiation de la transcription (négative si en amont)

Méthode	Contenu
`getBindingFactors()`	Tableau des facteurs qui s'y fixent (champ `BF`, renvoie vers TRANSFAC_FACTOR)
`getOrganism()` / `getMethod()` / `getComments()`	Champs `OS`, `MM`, `CC`

```php
<?php
$lecteur = DatabaseReaderFactory::readDatabase('TRANSFAC_SITE', file('
    r00001.transfac'));
printf(
    "%s : position %s, facteurs : %s\n",
    $lecteur->getSequence(),
    $lecteur->getFirstPosition(),
    implode(', ', $lecteur->getBindingFactors())
);
```

À quoi ça sert : lister les sites de fixation connus autour d'un gène, les facteurs concernés et la méthode qui les a mis en évidence — pour construire, par exemple, une carte de régulation d'un promoteur.

2.5.22 La famille KEGG

KEGG décrit le métabolisme : voies métaboliques, molécules, réactions, gènes, génomes. Comme TRANSFAC, elle publie un fichier par type d'information, tous écrits dans la même grammaire : une étiquette sur douze colonnes, sa donnée à partir de la treizième, et une ligne `///` (trois barres, pas deux) qui ferme l'enregistrement. Tous les lecteurs de cette famille connaissent déjà l'identifiant (`getEntry()`) et les noms/synonymes (`getNames()`) — les champs propres à chaque fichier sont à découvrir fiche par fiche.

Persistance : `$recordManager->find('TRANSFAC_SITE', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'TRANSFAC_SITE', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

KEGG_COMPOUND

Décrit un composé chimique du métabolisme : sa formule, les réactions où il intervient, les enzymes qui agissent sur lui, les voies métaboliques auxquelles il appartient.

Format : `'KEGG_COMPOUND'` — ouvre sur `ENTRY`, ferme sur `///`.

```
ENTRY       C00031                          Compound
NAME        D-Glucose; Grape sugar; Dextrose
FORMULA     C6H12O6
REACTION    R00299 R00301 R01555
ENZYME      1.1.1.47  2.7.1.1  5.3.1.9
PATHWAY     PATH: map00010  Glycolysis / Gluconeogenesis
DBLINKS     CAS: 50-99-7
///
```

Méthode	Contenu
`getFormula()`	Champ `FORMULA`
`getReactions()`	Tableau d'identifiants de réactions (champ `REACTION`)

Méthode	Contenu
`getEnzymes()`	Tableau de numéros EC (champ `ENZYME`)
`getPathways()`	Tableau de paires `[identifiant, nom]` (champ `PATHWAY`)
`getDbLinks()`	Tableau de paires `[banque, identifiant]` (champ `DBLINKS`)

```php
<?php
$lecteur = DatabaseReaderFactory::readDatabase('KEGG_COMPOUND', file('
    c00031.kegg'));
echo $lecteur->getFormula();    // "C6H12O6"
foreach ($lecteur->getPathways() as [$id, $nom]) {
    echo "$id : $nom\n";
}
```

À quoi ça sert : partir d'une molécule et remonter aux réactions et voies métaboliques qui la concernent, sans avoir à connaître d'avance sa formule.

Persistance : `$recordManager->find('KEGG_COMPOUND', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'KEGG_COMPOUND', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

KEGG_REACTION

Décrit une réaction biochimique : son équation (écrite avec les identifiants de composés) et les enzymes qui la catalysent.

Format : `'KEGG_REACTION'` — ouvre sur `ENTRY`, ferme sur `///`.

```
ENTRY        R00299                         Reaction
NAME         ATP:D-glucose 6-phosphotransferase
DEFINITION   ATP + D-Glucose <=> ADP + D-Glucose 6-phosphate
EQUATION     C00002 + C00031 <=> C00008 + C00092
ENZYME       2.7.1.1
///
```

Méthode	Contenu
`getDefinition()`	La réaction écrite avec les noms des composés
`getEquation()`	La réaction écrite avec les identifiants des composés
`getEnzymes()`	Tableau de numéros EC catalysant la réaction
`getPathways()`	Tableau de paires `[identifiant, nom]`

```php
<?php
$lecteur = DatabaseReaderFactory::readDatabase('KEGG_REACTION', file('
    r00299.kegg'));
echo $lecteur->getEquation();
print_r($lecteur->getEnzymes());
```

À quoi ça sert : lire l'équation d'une réaction et identifier les enzymes qui la catalysent — utile pour reconstituer une voie métabolique pas à pas.

Persistance : `$recordManager->find('KEGG_REACTION', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'KEGG_REACTION', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

KEGG_ENZYME

La fiche la plus riche de la famille : rassemble tout ce que KEGG sait d'un numéro EC — classification, substrats, produits, gènes qui codent l'enzyme, maladies liées, motifs et structures connues.

Format : `'KEGG_ENZYME'` — ouvre sur `ENTRY`, ferme sur `///`.

```
ENTRY        EC 2.7.1.1                  Enzyme
NAME         hexokinase
CLASS        Transferases; Transferring phosphorus-containing groups
SYSNAME      ATP:D-hexose 6-phosphotransferase
REACTION     R00299
SUBSTRATE    ATP; D-glucose
PRODUCT      ADP; D-glucose 6-phosphate
GENES        HSA: 3098 3099 3101
DISEASE      H00409  Hexokinase deficiency
STRUCTURES   PDB: 1HKB 1HKC 1IG8
DBLINKS      ExplorEnz: 2.7.1.1
///
```

Méthode	Contenu
`getClassification()`	Tableau, champ `CLASS`
`getSysname()`	Nom systématique de l'enzyme
`getReactions()` / `getSubstrates()` / `getProducts()`	Ce que catalyse l'enzyme
`getGenes()`	Tableau, champ `GENES` (un item par organisme)
`getDiseases()`	Tableau, champ `DISEASE`
`getMotifs()`	Tableau, champ `MOTIF`
`getStructures()`	Tableau des identifiants PDB liés
`getComment()` / `getPathways()` / `getOrthologs()` / `getDbLinks()`	Compléments

```php
<?php
$lecteur = DatabaseReaderFactory::readDatabase('KEGG_ENZYME', file('
    2.7.1.1.kegg'));
echo $lecteur->getSysname();
print_r($lecteur->getSubstrates());
print_r($lecteur->getDiseases());
print_r($lecteur->getStructures());  // les codes PDB, pour aller les
    lire avec le parser PDB
```

À quoi ça sert : obtenir en un seul appel la fiche complète d'une enzyme — substrats, produits, gènes, maladies, structures — et enchainer, par exemple, sur une lecture PDB des structures listées.

Persistance : `$recordManager->find('KEGG_ENZYME', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'KEGG_ENZYME', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

KEGG_ORTHOLOG

Regroupe les gènes de différentes espèces qui descendent d'un même gène ancestral et remplissent la même fonction — ce qui permet de transposer une voie métabolique d'une espèce à une autre.

Format : `'KEGG_ORTHOLOG'` — ouvre sur `ENTRY`, ferme sur `///`.

```
ENTRY        K00844                      KO
NAME         HK; hexokinase
DEFINITION   hexokinase [EC:2.7.1.1]
CLASS        Metabolism; Carbohydrate metabolism; Glycolysis
GENES        HSA: 3098 3099 3101
             MMU: 15275 15277
DBLINKS      GO: 0004396
///
```

Méthode	Contenu
`getDefinition()`	Champ `DEFINITION`
`getClassification()`	Tableau, champ `CLASS`
`getGenes()`	Tableau, une entrée par organisme (ex. `"HSA: 3098 3099 3101"`)
`getDbLinks()`	Tableau de paires `[banque, identifiant]`

```php
<?php
$lecteur = DatabaseReaderFactory::readDatabase('KEGG_ORTHOLOG', file('
    k00844.kegg'));
foreach ($lecteur->getGenes() as $ligne) {
    echo $ligne . "\n";  // "HSA: 3098 3099 3101", "MMU: 15275 15277"...
}
```

À quoi ça sert : comparer un groupe de gènes orthologues entre espèces — par exemple pour vérifier qu'un gène humain a bien un équivalent connu chez la souris avant de concevoir une expérience.

Persistance : `$recordManager->find('KEGG_ORTHOLOG', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'KEGG_ORTHOLOG', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

KEGG_GENOME

Nomme un organisme séquencé pour lequel KEGG tient des voies métaboliques, et le relie à la taxonomie NCBI.

Format : `'KEGG_GENOME'` — ouvre sur `ENTRY`, ferme sur `///`.

```
ENTRY        T01001                      Genome
NAME         hsa, Homo sapiens (human)
DEFINITION   Homo sapiens (human)
TAXONOMY     TAX:9606
LINEAGE      Eukaryota; Metazoa; Chordata; Mammalia; Primates
///
```

Méthode	Contenu
`getDefinition()`	Champ `DEFINITION`
`getTaxonomy()`	Identifiant de taxonomie NCBI, sans le préfixe `TAX:`
`getLineage()`	Tableau, champ `LINEAGE`

```php
<?php
$lecteur = DatabaseReaderFactory::readDatabase('KEGG_GENOME', file('
    t01001.kegg'));
echo $lecteur->getDefinition();  // "Homo sapiens (human)"
echo $lecteur->getTaxonomy();    // "9606"
```

Note

Ne confondez pas ce lecteur avec `GENOME` (section 2.5.16) : `KEGG_GENOME` identifie un organisme pour la vue « métabolisme » de KEGG (voies, taxonomie), tandis que `GENOME` résume l'avancement du séquençage d'un organisme côté GenBank. Deux informations différentes sur le même organisme, dans deux banques indépendantes.

À quoi ça sert : situer un génome dans sa lignée taxonomique avant d'aller chercher ses voies métaboliques ou ses groupes orthologues.

Persistance : `$recordManager->find('KEGG_GENOME', $id)` retrouve un enregistrement déjà stocké ; `storeFile($nom, 'KEGG_GENOME', ...$fichiers)` importe un fichier entier dans la table `parsed_record` (§2.3.2).

2.6 Manipuler des séquences

2.6.1 Le service principal : SequenceManager

C'est le service que l'on retrouve le plus souvent : il regroupe toutes les opérations courantes sur une séquence ADN/ARN/protéine. Le tableau suivant sert d'index par besoin.

Besoin	Méthodes
Transformer	`complement()`, `halfSequence()`, `expandNa()` (développe les symboles IUPAC ambigus), `subSeq()`
Traduire	`translate()`, `translateCodon()`, `getCodon()`, `countCodons()`
Chercher un motif	`patPos()`, `patPoso()`, `patFreq()`, `findPattern()`, `symFreq()`
Propriétés physico-chimiques	`molwt()` (masse moléculaire), `charge()`, `chemicalGroup()`
Palindromes et miroirs	`isMirror()`, `findMirror()`, `isPalindrome()`, `findPalindrome()`

Exemple : traduire une séquence codante en protéine (même `$sequenceManager` que dans l'installation « vanilla » de l'introduction) :

```php
<?php
$proteine = $sequenceManager->translate(
    "ATGGCCATTGTAATGGGCCGCTGA",
    0,  // cadre de lecture : 0, 1 ou 2
```

```
    1   // format : 1 = lettres simples (M), 3 = lettres triples (Met)
);

echo $proteine; // ex. "MAIVMGR*"
```

Note

`translate()` et les autres méthodes qui manipulent des acides aminés interrogent BioAPI en coulisses (via les adaptateurs vus au chapitre 1) pour connaître la table des codons — c'est pour cette raison que `SequenceManager` se construit avec les trois adaptateurs `AminoApi`, `NucleotidApi` et `ElementApi`, comme montré en introduction.

`SequenceBuilder` est une couche fine autour de `SequenceManager` : elle retient la séquence en cours (à partir d'un objet `Sequence`, typiquement celui renvoyé par un lecteur du chapitre précédent) pour éviter de la repasser en argument à chaque appel. Toutes les méthodes ci-dessus existent aussi dessus, sans le premier argument `$sequence`.

2.6.2 Des séquences qui se valident elles-mêmes

En complément, BioPHP propose des petits objets immuables qui enveloppent une chaîne de caractères et vérifient, dès leur création, qu'elle ne contient que des symboles valides pour leur alphabet : `DnaSequence`, `RnaSequence`, `AminoAcidSequence`.

```
<?php
use Amelaye\BioPHP\Domain\Sequence\ValueObject\DnaSequence;

$adn = new DnaSequence("ATGGCCATTGTAATGGGCCGCTGA");
echo $adn->getLength();
echo $adn->reverse();
$arn = $adn->toRna(); // conversion ADN -> ARN

$proteine = new \Amelaye\BioPHP\Domain\Sequence\ValueObject\
    AminoAcidSequence("MAIVMGR*");
var_dump($proteine->hasStop());          // true : contient un codon
    stop
echo $proteine->truncateAtStop();        // "MAIVMGR", coupee au stop
```

`MolecularSequenceFactory` sait construire automatiquement le bon objet (`DnaSequence`, `RnaSequence`…) à partir d'une séquence issue d'un lecteur du chapitre précédent, en regardant le type de molécule qu'il a lu (`fromEntity()`).

2.6.3 Les protéines : ProteinManager

Un service volontairement léger, pour les deux informations les plus demandées sur une protéine : sa longueur (`seqlen()`) et sa masse moléculaire (`molwt()`).

2.6.4 Les enzymes de restriction : RestrictionEnzymeManager

On peut maintenant refaire proprement l'exemple « EcoRI » du chapitre 1, sans reconstruire soi-même la recherche de motif :

```
<?php
// $typeIIEndonucleaseApi : l'adaptateur du chapitre 1 (Api\
    TypeIIEndonucleaseApi)
$gestionnaire = new \Amelaye\BioPHP\Domain\Sequence\Service\
    RestrictionEnzymeManager(
```

```
    $typeIIEndonucleaseApi,
    new \Amelaye\BioPHP\Domain\Sequence\Entity\Enzyme()
);

$gestionnaire->parseEnzyme("EcoRI");
$gestionnaire->setSequenceManager($sequenceManagerDeLaSequence);
$fragments = $gestionnaire->cutSeq();

print_r($fragments); // les morceaux d'ADN obtenus apres decoupe
```

`findRestEn()` fait l'inverse : on lui donne un motif (et, optionnellement, une position de coupe ou une longueur de motif), et il renvoie les enzymes connues qui correspondent — pratique quand on a identifié un site de coupure mais qu'on ne sait pas encore quelle enzyme l'a produit.

2.6.5 Aligner des séquences : SequenceAlignmentManager

Lit un fichier d'alignement multiple déjà produit par un outil comme ClustalW (formats FASTA ou CLUSTAL supportés), puis permet de l'explorer :

```
<?php
$alignement = new \Amelaye\BioPHP\Domain\Sequence\Service\
   SequenceAlignmentManager($sequenceBuilder);
$alignement->setFilename('mon_alignement.aln');
$alignement->setFormat('CLUSTAL');
$alignement->parseFile();

echo $alignement->consensus(80);  // sequence consensus (seuil de 80%)
print_r($alignement->resVar(80)); // positions variables sous ce seuil
```

2.6.6 Comparer deux séquences : SequenceMatchManager

Pour mesurer la ressemblance entre deux séquences : distance de Hamming (`hamdist()`, séquences de même longueur), distance de Levenshtein (`levdist()`, tolère des insertions/suppressions), ou une comparaison lettre à lettre pilotée par une matrice de substitution (`match()`, `setSubMatrix()`) — la même idée que la matrice PAM250 exposée par BioAPI au chapitre 1.

2.7 La boîte à outils

Quelques fonctions plus modestes, regroupées par thème.

Classe	Ce qu'elle offre
`GeneticsFunctions`	`CountACGT()` (compte les bases d'une séquence), `CountCG()`, `CreateInversion()` (brin complémentaire inversé), `RemoveNonCodingProt()`
`Mathematics Functions`	`Mean()`, `Median()`, `Variance()` — des statistiques simples sur un tableau de valeurs
`OligosManager`	`findOligos($sequence, $longueur)` compte la fréquence des oligonucléotides d'une longueur donnée dans une séquence ; `findZScore()` calcule des z-scores pour comparer ces fréquences

2.8 Les adaptateurs vers BioAPI

Enfin, le dossier `Api/` contient les 14 classes qui vont chercher les données de référence sur BioAPI — une par ressource du chapitre 1 (`AminoApi`, `NucleotidApi`, `ElementApi`, `TypeIIEndonucleaseApi`…). Ce sont elles qui alimentent en coulisses `SequenceManager` et `RestrictionEnzymeManager` présentés dans ce chapitre. Leur fonctionnement est celui déjà montré en introduction : on leur donne un client HTTP et un sérialiseur, elles renvoient des objets simples (un par ligne de donnée) plutôt que le JSON brut de l'API.

Chapitre 3

BioTools

3.1 Vue d'ensemble

Là où BioPHP (chapitre 2) vous donne des briques bas niveau — manipuler une séquence, lire un fichier SWISSPROT, appeler l'API de référence — BioTools vous donne **19 outils complets**, chacun prêt à devenir une page de votre application : un formulaire de saisie déjà validé, et le calcul biologique qui va derrière. Ce sont les mini-outils bien connus de biophp.org (traduction ADN/protéine, digestion par enzymes de restriction, température de fusion, recherche de palindromes, représentation du jeu du chaos, etc.), réécrits sous forme de composants Symfony.

Note

Chaque outil est fait de deux classes : un `FormType` (dans `Form/`) qui décrit les champs de saisie et leurs contraintes, et un `Manager` (dans `Service/`) qui contient le calcul lui-même, indépendant de Symfony. Le tableau du §3.6 fait correspondre les 19 couples.

Le point qui structure tout ce chapitre : BioTools ne fournit **ni contrôleur, ni template**. C'est un choix assumé — vous n'êtes pas forcé d'adopter les routes, le gabarit HTML ou la structure de page de biophp.org, vous branchez chaque formulaire là où ça vous arrange dans votre propre application. La contrepartie, c'est que pour chacun des 19 outils, vous avez à écrire vous-même l'action de contrôleur et la vue qui l'affiche. C'est précisément ce que chaque fiche de ce chapitre vous montre : une implémentation Symfony complète (calquée sur celle du site de démonstration), et une variante « vanilla » pour qui n'utilise pas Symfony.

Attention

La plupart des managers ont besoin des adaptateurs d'API vus au chapitre 1 (`NucleotidApiAdapter`, `AminoApiAdapter`, `VendorApiAdapter`…) injectés dans leur constructeur. En Symfony, l'autowiring s'en charge ; en « vanilla », vous les construisez à la main. Dans les deux cas, **un accès réseau vers l'API de référence est nécessaire** au moment de l'appel — c'est le premier réflexe à avoir si un outil échoue sans raison apparente.

3.2 Installation et configuration

3.2.1 Installation

```
composer require amelaye/biotools
```

BioTools dépend de BioPHP, qui dépend lui-même de deux bundles tiers. En Symfony, on enregistre les trois bundles dans `config/bundles.php` :

```
return [
    // ...
    JMS\SerializerBundle\JMSSerializerBundle::class => ['all' => true],
    Amelaye\BioPHP\AmelayeBioPHPBundle::class => ['all' => true],
    Amelaye\BioTools\AmelayeBioToolsBundle::class => ['all' => true],
];
```

Note

En installation « vanilla » (sans Symfony), il n'y a pas de bundle à enregistrer : vous instanciez directement les classes de `Form/` et `Service/` dont vous avez besoin, comme montré dans chaque fiche. Le formulaire Symfony (`FormType`) n'a alors plus de rôle — c'est votre propre code qui lit les champs reçus et les passe au manager.

3.2.2 Configuration

Le bundle fonctionne sans aucune configuration — des valeurs par défaut raisonnables sont fournies. Deux réglages peuvent être surchargés sous la clé `amelaye_biotools` :

```
# config/packages/amelaye_biotools.yaml
amelaye_biotools:
    nucleotids_graphs:
        path_graphs: 'created_files/'   # dossier ou sont ecrits les SVG
            generes
    protein_colors:
        # palettes de couleurs pour les alphabets reduits de proteines
```

Attention

`path_graphs` doit exister et être accessible en écriture par le processus PHP, **et** être servi publiquement si vous comptez afficher les images générées dans une page web. C'est un point de configuration facile à oublier au premier déploiement — une erreur 500 à la soumission d'un des quatre outils qui dessinent un graphique (§3.5) en est souvent le symptome.

3.3 Le pattern commun

Les 19 outils suivent tous le même schéma, qu'il est plus simple de comprendre une fois pour toutes plutôt que de le redécouvrir à chaque fiche :

1. Le `FormType` décrit les champs (avec, souvent, une valeur par défaut d'exemple) et leurs contraintes de validation.
2. Le contrôleur crée le formulaire, appelle `handleRequest()`, et si la soumission est valide, récupère les données avec `$form->getData()`.
3. Ces données sont passées telles quelles à la méthode principale du `Manager` correspondant, qui fait le calcul et renvoie un résultat structuré (tableau, chaîne formatée, ou chemin d'un fichier SVG généré).
4. Le template affiche ce résultat.

3.3.1 Les deux contraintes de validation maison

Deux `Constraint` Symfony reviennent sur plusieurs formulaires, dans `Validator/` :

Contrainte	Ce qu'elle vérifie
`SequenceRecognition`	La chaîne ne contient que des codes NC-IUBMB valides (bases A/C/G/T, protéines, ou nucléotides dégénérés Y/R/W/S/K/M/D/V/H/B/N) une fois les caractères non codants retirés. Rejette une séquence vide ou contenant un caractère inconnu.
`MeltingTemperature`	Même vérification de composition, plus une contrainte de longueur : entre 6 et 50 paires de bases (uniquement imposée si le champ n'est pas vide).

3.3.2 L'extension Twig

Deux outils (traduction ADN/protéine et alignement de séquences) exposent leur mise en forme via une extension Twig, `BioToolsExtension`, plutôt que de la recalculer dans le contrôleur :

```
{{ sequence_alignee|compare_alignment(autre_sequence) }}
{{ show_translations_aligned(sequence, frame) }}
{{ show_translations_aligned_complementary(frame) }}
```

Note

Ce filtre et ces deux fonctions appellent respectivement `SequenceAlignmentManager::compareAlignment()` et `DnaToProteinManager::showTranslationsAligned(Complementary)()`. Vous pouvez tout aussi bien appeler ces méthodes directement depuis le contrôleur si vous n'utilisez pas Twig — c'est ce que montrent les fiches correspondantes.

3.4 Les 19 fiches outils

Chaque fiche suit le même gabarit : à quoi sert l'outil, capture du formulaire, tableau des champs (avec leur traduction française, les libellés de l'interface étant en anglais), capture du résultat, implémentation Symfony puis « vanilla », et un encadré le cas échéant.

3.4.1 Chaos Game Representation (CGR et FCGR)

Décrit par Jeffrey (1990). En partant du centre d'un carré dont chaque sommet représente une base (A, C, G, T), chaque base de la séquence déplace le point courant à mi-distance de son sommet, et cette nouvelle position est tracée. Après un grand nombre d'itérations, des motifs fractals caractéristiques de la séquence apparaissent. La variante **FCGR**, décrite par Deschavanne *et al.* (1999), ne trace pas les positions successives mais les **fréquences d'oligonucléotides** sous forme de damier.

Note

CGR et FCGR partagent le même `FormType` (`ChaosGameRepresentationType`) et le même manager. Seuls les champs affichés par le template diffèrent : le CGR n'a besoin que du nom, de la séquence et de la taille de l'image ; le FCGR ajoute le choix de la longueur des oligonucléotides et du nombre de brins.

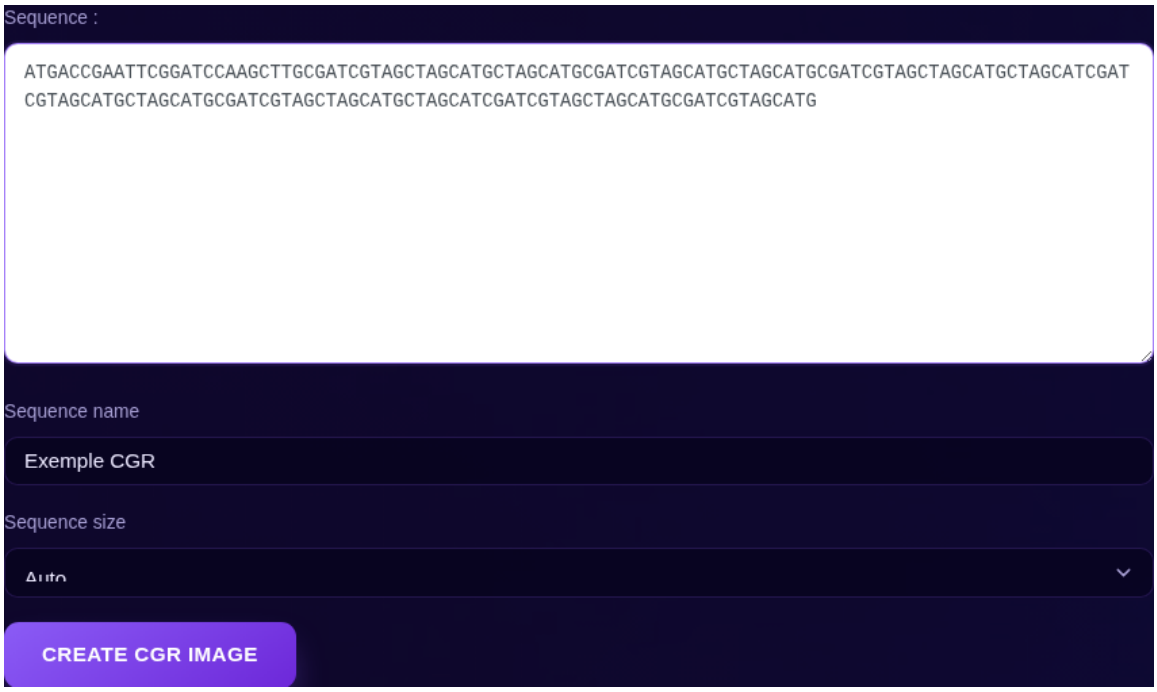

FIGURE 3.1 – Formulaire CGR.

Champ	**Libellé / traduction**	**Type**	**Valeurs, défaut, rôle**
`seq_name`	Sequence name *Nom de la séquence*	texte	Libre ; sert de titre à l'image générée.
`seq`	Sequence *Séquence*	texte long	ADN uniquement ; entre 50 et 5 000 000 pb. Un exemple d'ADN est proposé par défaut.
`size`	Sequence size *Taille de l'image*	choix	`auto` / 1024×1024 / 512×512 / 256×256.

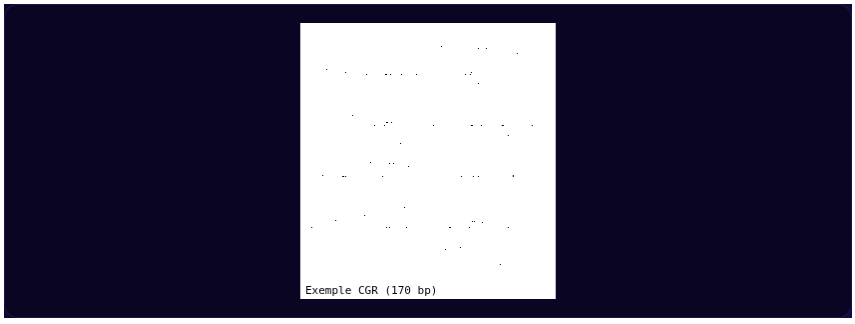

FIGURE 3.2 – Résultat CGR : chaque position successive de la séquence tracée dans le carré.

Pour le FCGR, deux champs supplémentaires apparaissent :

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`len`	Search oligos of length *Longueur des oligonucléotides*	choix	2 à 7.
`s`	Compute data for *Calculer sur*	choix	`2` = les deux brins (défaut) / `1` = brin supérieur seul.
`map`	Show as image map *Afficher en carte cliquable*	case	Décochée par défaut ; déconseillée pour de longs oligonucléotides.
`freq`	Show oligonucleotide frequencies *Afficher les fréquences*	case	Cochée par défaut.

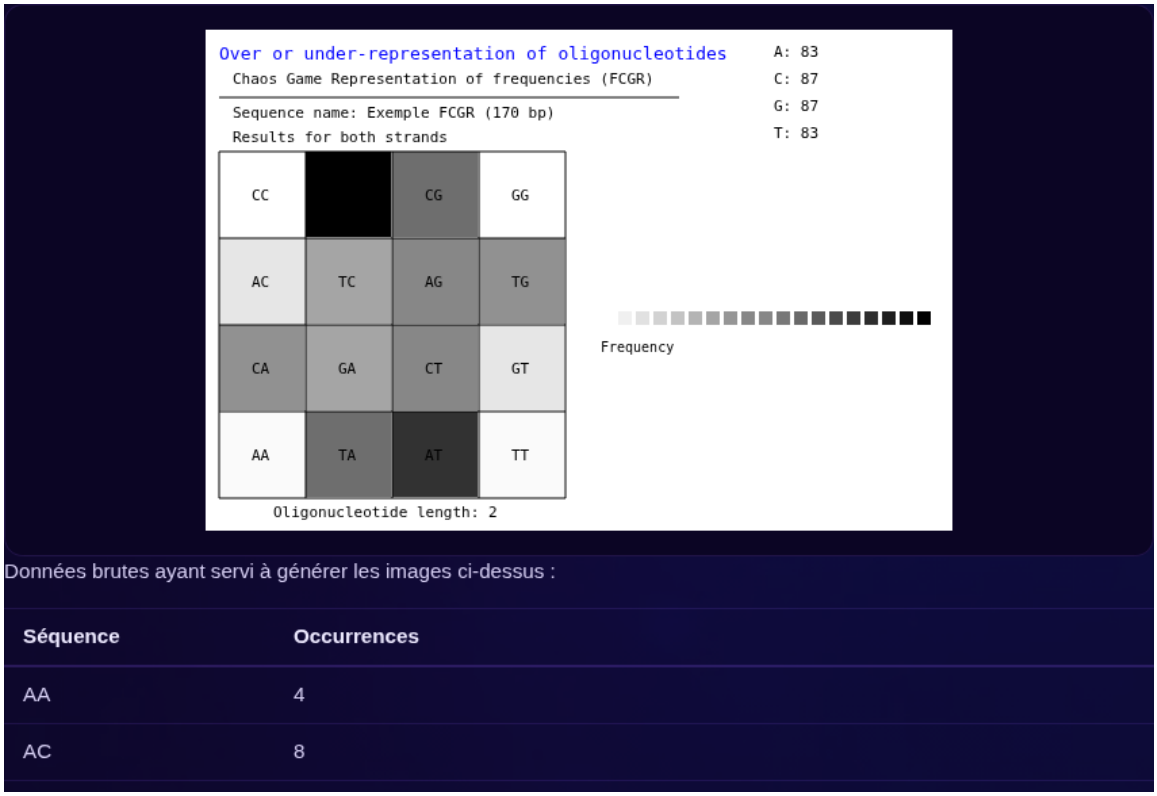

FIGURE 3.3 – Résultat FCGR : damier des fréquences d'oligonucléotides, en niveaux de gris.

Implémentation Symfony

```
1  public function cgrCompute(Request $request,
2      ChaosGameRepresentationManager $manager, $form)
3  {
4      $isComputed = false;
5      $image = null;
6
```

```
    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();
        $image = basename(
            $manager->CGRCompute($formData["seq_name"], $formData["seq"
                ], $formData["size"])
        );
        $isComputed = true;
    }

    return $this->render('minitools/chaosGameRepresentationCGR.html.twig
        ', [
        'form'       => $form->createView(),
        'image'      => $image,
        'isComputed' => $isComputed,
    ]);
}
```

Le template affiche ensuite l'image renvoyée par `CGRCompute()` avec un simple `<img src="{{ path_graphs }}/{{ image }}">`.

Implémentation vanilla

```
$manager = new ChaosGameRepresentationManager(
    $config['nucleotids_graphs'], // tableau de configuration (voir S3
        .2)
    new NucleotidApiAdapter($httpClient, $serializer)
);

$cheminSvg = $manager->CGRCompute('Ma sequence', $sequenceADN, 'auto');
// $cheminSvg pointe vers le fichier SVG ecrit dans path_graphs
```

Attention

`CGRCompute()` et `createCGRImage()` écrivent un fichier SVG sur disque (voir l'encadré de configuration du §3.2) : le dossier `path_graphs` doit exister et être accessible en écriture. Le FCGR (§3.5) écrit lui aussi son damier sur disque, exactement de la même manière — sa particularité n'est pas d'éviter le fichier, mais d'ajouter par-dessus des coordonnées de `<map>`/`<area>` HTML pour rendre l'image cliquable.

3.4.2 Distance entre séquences (oligonucléotides)

Compare plusieurs séquences entre elles à partir de leur composition en oligonucléotides, et produit un dendrogramme (arbre de parenté) groupé par la méthode UPGMA. Deux méthodes de distance sont proposées : la distance de Pearson sur des z-scores de tétranucléotides (adaptée aux séquences longues, plus de 20 000 pb), ou la distance euclidienne sur les fréquences d'oligonucléotides d'une longueur choisie.

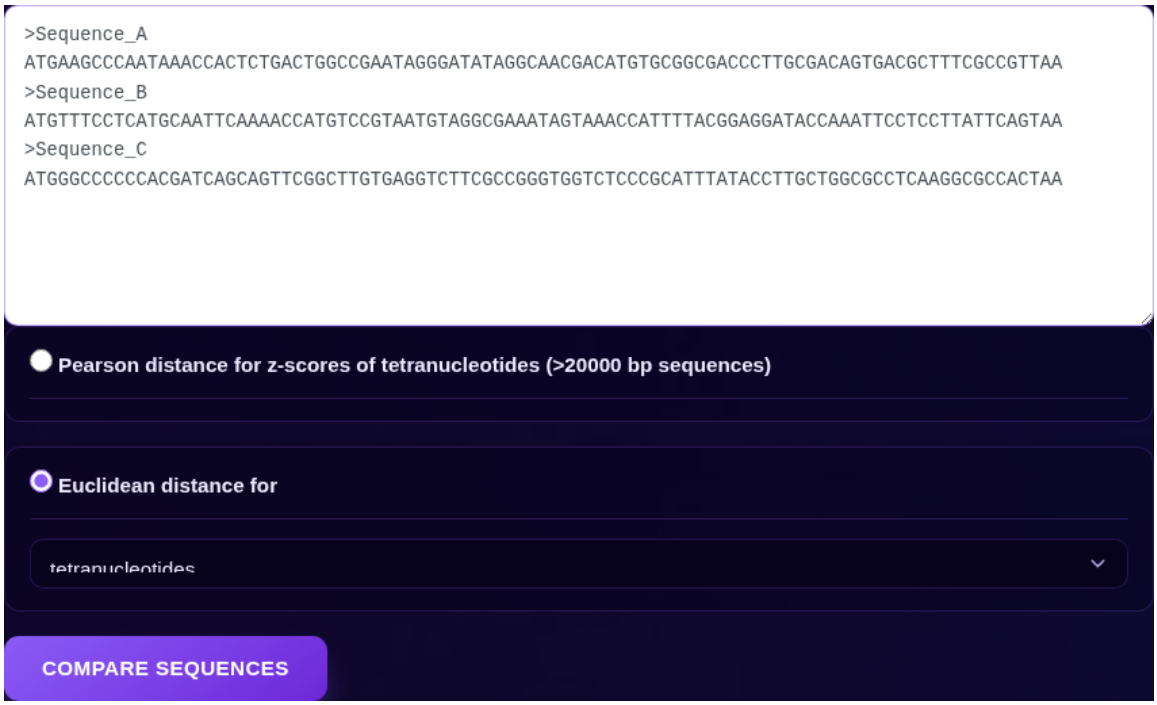

FIGURE 3.4 – Formulaire de comparaison de séquences.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`seq`	Sequence *Séquences*	texte long	Format FASTA multi-séquences (`>nom` suivi de la séquence) ; 2 000 000 pb maximum au total.
`method`	Pearson distance… / Euclidean distance for *Méthode de distance*	radio	`euclidean` (défaut) ou `pearson`.
`len`	(longueur des oligonucléotides) *Longueur des oligonucléotides*	choix	di- à hexanucléotides (2 à 6) ; défaut : tétranucléotides (4). Ignoré si méthode = Pearson.

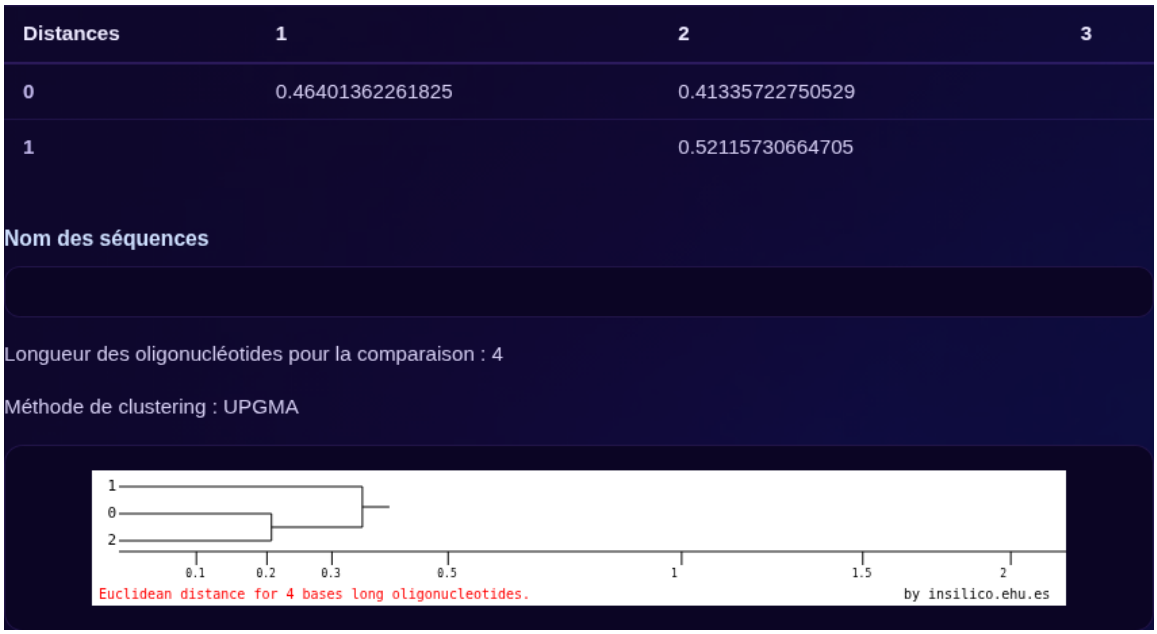

FIGURE 3.5 – Matrice des distances entre séquences.

Implémentation Symfony

```php
public function distanceAmongSequencesAction(Request $request,
    DistanceAmongSequencesManager $manager)
{
    $form = $this->createForm(DistanceAmongSequencesType::class);
    $data = [];

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();
        $length   = $formData["len"];
        $seqs     = $manager->formatSequences($formData["seq"]);

        if ($formData["method"] == "euclidean") {
            $oligoArray = $manager->
                computeOligonucleotidsFrequenciesEuclidean($seqs, $length
                );
            $data       = $manager->
                computeDistancesAmongFrequenciesEuclidean($seqs,
                $oligoArray, $length);
        } else {
            $oligoArray = $manager->computeOligonucleotidsFrequencies(
                $seqs);
            $data       = $manager->computeDistancesAmongFrequencies(
                $seqs, $oligoArray);
        }

        // le dendrogramme est ecrit en SVG a cote (voir S3.5)
        $chemin = $graphsDir . '/dendogram_' . bin2hex(random_bytes(4))
            . '.svg';
        $manager->upgmaClustering($data, $formData["method"], $length,
            $chemin);
    }
```

```
24
25      return $this->render('minitools/distanceAmongSequences.html.twig',
26          ['form' => $form->createView(), 'data' => $data]);
27  }
```

Implémentation vanilla

```
1  $manager = new DistanceAmongSequencesManager($oligosManager,
       $nucleotidApi);
2
3  $seqs = $manager->formatSequences($fastaMultiSequences);
4  $oligoArray = $manager->computeOligonucleotidsFrequenciesEuclidean($seqs
       , 4);
5  $distances = $manager->computeDistancesAmongFrequenciesEuclidean($seqs,
       $oligoArray, 4);
6
7  $manager->upgmaClustering($distances, 'euclidean', 4, '/chemin/vers/
       dendogramme.svg');
```

Attention

Comme le CGR, `upgmaClustering()` écrit le dendrogramme sur disque : même pré-requis sur `path_graphs` (existant, accessible en écriture, servi publiquement).

3.4.3 Traduction ADN vers protéine

Traduit une séquence d'ADN en protéine, dans un ou plusieurs cadres de lecture, avec un code génétique prédéfini (compilé par Elzanowski et Ostell, NCBI) ou personnalisé, et recherche optionnellement les cadres de lecture ouverts (ORF).

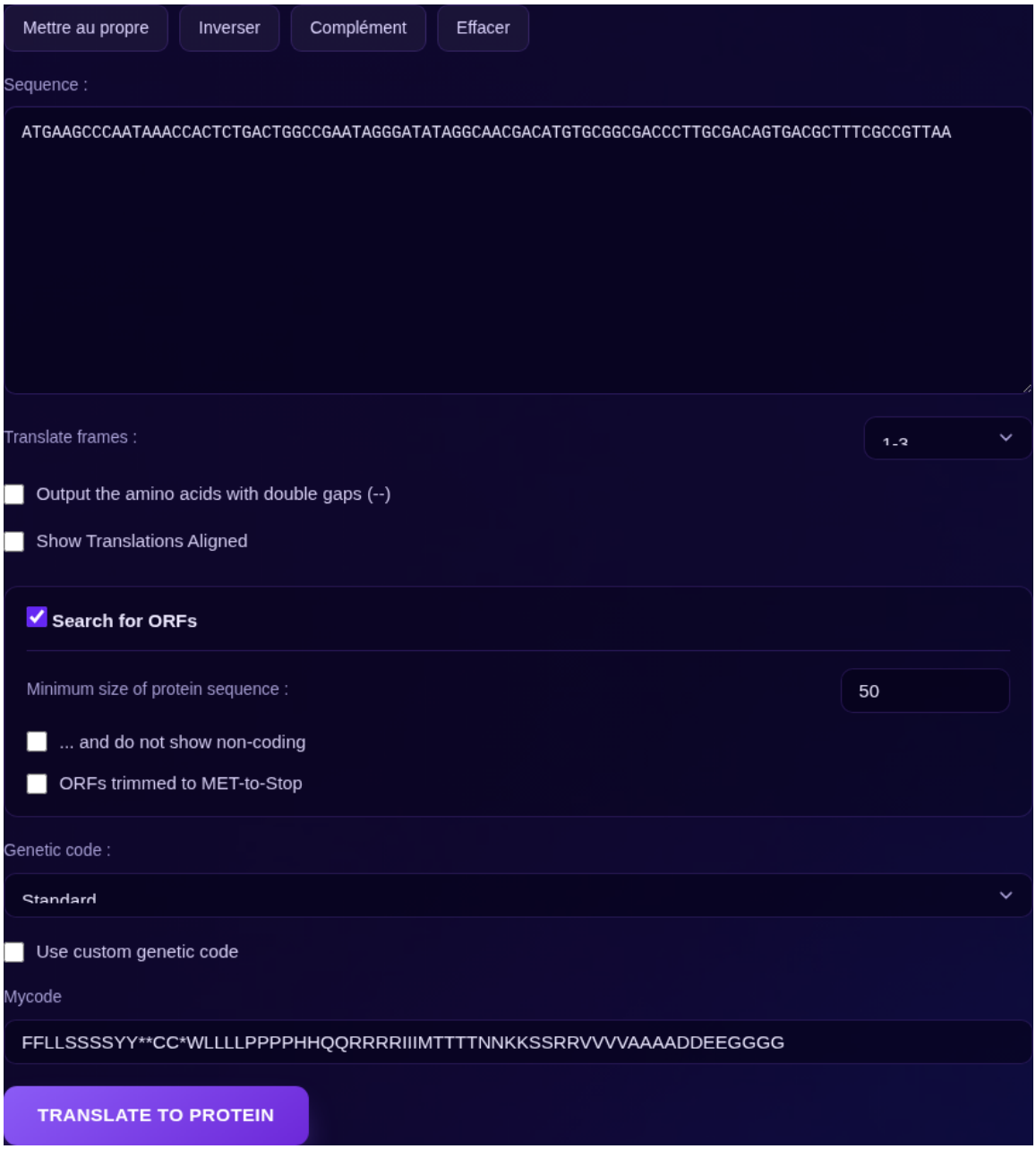

FIGURE 3.6 – Formulaire de traduction ADN vers protéine.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`sequence`	Sequence *Séquence*	texte long	ADN à traduire.
`frames`	(cadres de lecture) *Cadres de lecture*	choix	Ex. : cadre 1, 2, 3, ou les trois.
`genetic_code`	Genetic code *Code génétique*	choix	Standard par défaut ; liste des codes NCBI.
`usemycode`	Use my own code *Utiliser mon propre code*	case	Active le champ `mycode`.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`mycode`	Custom genetic code *Code génétique personnalisé*	texte	Chaîne de 64 caractères (un par triplet) ; un exemple (code standard) est pré-rempli.
`search_orfs`	Search ORFs *Rechercher les ORF*	case	
`protsize`	Minimum protein size *Taille protéique minimale*	texte	50 par défaut ; utilisé uniquement si recherche d'ORF.
`only_coding`	Only ORFs starting with Met *ORF commençant par Méthionine uniquement*	case	
`trimmed`	Trim ORFs from Met to Stop *Rogner de Méthionine à Stop*	case	
`show_aligned`	Show translations aligned *Afficher l'alignement des traductions*	case	
`dgaps`	Show gaps as dashes *Afficher les espaces comme des tirets*	case	

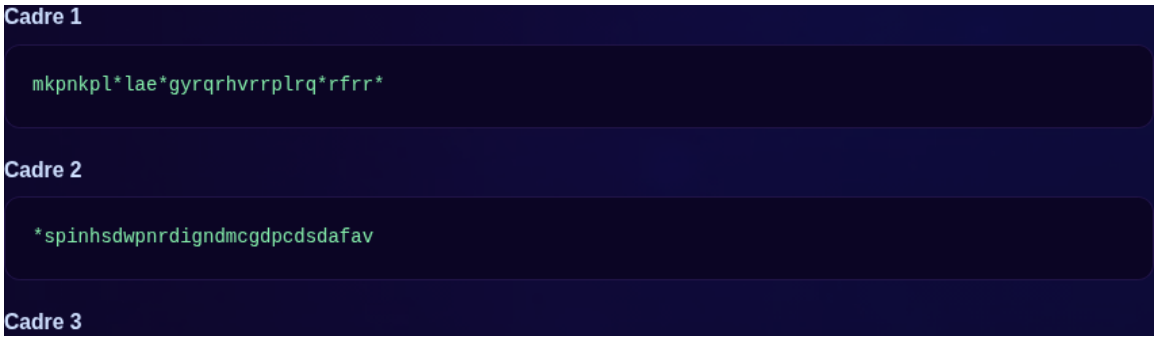

FIGURE 3.7 – Traductions obtenues, un cadre de lecture par bloc.

Implémentation Symfony

```
public function dnaToProteinAction(Request $request, DnaToProteinManager
    $manager)
{
    $form = $this->createForm(DnaToProteinType::class);
    $frames = [];

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();
        $sequence = $formData["sequence"];
```

```
9
10          if ($formData["usemycode"]) {
11              $frames = $manager->customTreatment($formData["frames"],
                    $sequence, $formData["mycode"]);
12          } else {
13              $frames = $manager->definedTreatment($formData["frames"],
                    $formData["genetic_code"], $sequence);
14          }
15
16          if ($formData["search_orfs"]) {
17              $frames = $manager->findORF($frames, $formData["protsize"],
18                  (bool) $formData["only_coding"], (bool) $formData["
                        trimmed"]);
19          }
20      }
21
22      return $this->render('minitools/dnaToProtein.html.twig',
23          ['form' => $form->createView(), 'frames' => $frames]);
24  }
```

Le template peut ensuite s'appuyer sur l'extension Twig (§3.3) plutôt que de précalculer l'alignement dans le contrôleur : `{{ show_translations_aligned(sequence, frame) }}`.

Implémentation vanilla

```
$manager = new DnaToProteinManager($aminoApi, $tripletApi,
    $tripletSpecieApi);

$frames = $manager->definedTreatment(1, 'Standard', $sequenceADN);
$frames = $manager->findORF($frames, 50, true, true);
```

3.4.4 Recherche de séquences palindromiques

Séquence d'ADN dont la lecture 5'→3' est identique sur les deux brins — autrement dit, une séquence égale à son propre complément inversé. C'est le cas des sites de reconnaissance de nombreuses enzymes de restriction.

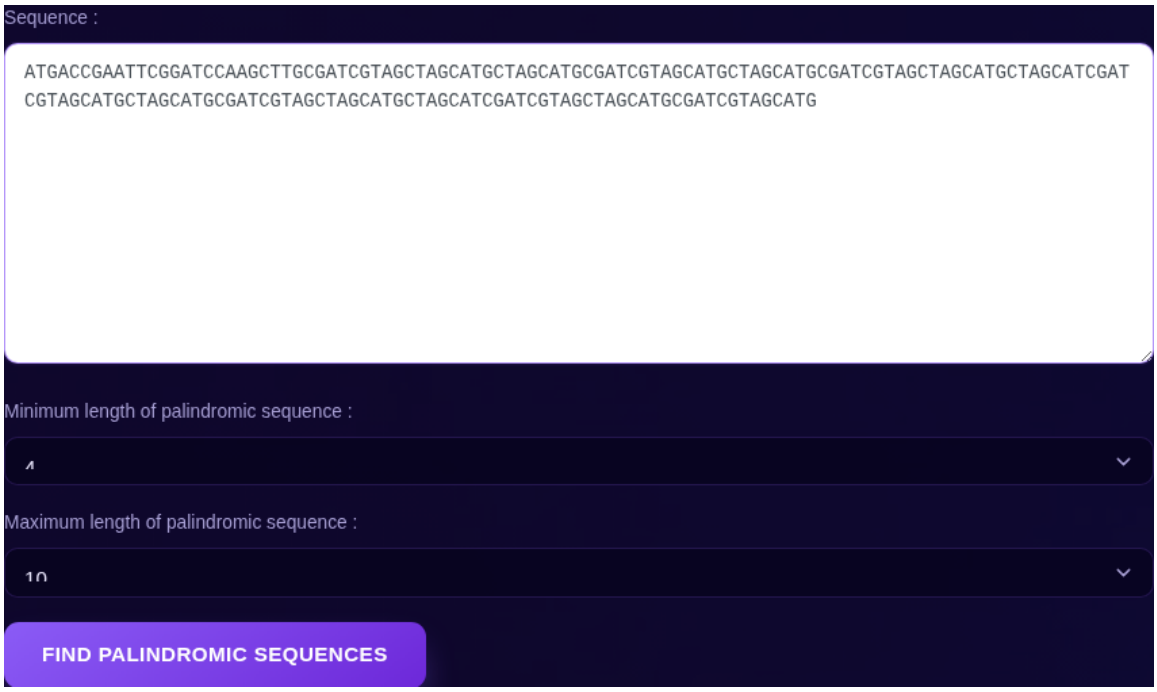

FIGURE 3.8 – Formulaire de recherche de palindromes.

Champ	**Libellé / traduction**	**Type**	**Valeurs, défaut, rôle**
`seq`	Sequence *Séquence*	texte long	ADN à analyser.
`min`	Minimum length *Longueur minimale*	choix	Longueur minimale d'un palindrome recherché.
`max`	Maximum length *Longueur maximale*	choix	Longueur maximale d'un palindrome recherché.

Position	Séquence
4	CCGAATTCGG
5	CGAATTCG
6	GAATTC
7	AATT
12	GGATCC
13	GATC
17	CAAGCTTG
18	AAGCTT
19	AGCT
25	CGATCG
26	GATC
31	TAGCTA
32	AGCT
33	GCTAGC
34	CTAG
35	TAGCATGCTA

FIGURE 3.9 – Positions et séquences palindromiques trouvées.

Implémentation Symfony

```
public function findPalindromesAction(Request $request,
    FindPalindromeManager $manager)
{
    $palindromes = [];
    $form = $this->createForm(FindPalindromesType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();
        $palindromes = $manager->findPalindromicSeqs(
            $formData["seq"], $formData["min"], $formData["max"]
        );
    }

    return $this->render('minitools/findPalindromes.html.twig',
        ['form' => $form->createView(), 'palindromes' => $palindromes]);
}
```

Implémentation vanilla

```php
$manager = new FindPalindromeManager();
$palindromes = $manager->findPalindromicSeqs($sequenceADN, 4, 10);
// ['position_dans_la_sequence' => 'sous-sequence_palindromique', ...]
```

Note

Seul outil du chapitre à ne dépendre d'aucun adaptateur d'API : `FindPalindromeManager` n'utilise que le `SequenceTrait` de BioPHP (complément inversé calculé localement). C'est la façon la plus simple de tester l'implémentation « vanilla » décrite ici.

3.4.5 Calcul du taux de GC (GC content finder)

Calcule la longueur et la composition en bases (A, C, G, T) d'une séquence fournie sous forme de fichier FASTA téléversé.

FIGURE 3.10 – Formulaire de dépôt d'un fichier FASTA.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`fasta`	FASTA File *Fichier FASTA*	fichier	100 000 ko maximum ; aucune vérification de type MIME (un fichier `.fasta`/`.fa` envoyé par le navigateur en `application/octet-stream` est accepté).

FIGURE 3.11 – Longueur et composition en bases du fichier déposé.

Implémentation Symfony

```php
public function fastaUploaderAction(Request $request,
    FastaUploaderManager $manager)
{
    $length = 0; $a = 0; $g = 0; $t = 0; $c = 0;
    $form = $this->createForm(FastaUploaderType::class);

```

```
    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();

        $var = $manager->createFiles($formData["fasta"], $this->
            getParameter('brochures_directory'));

        // createFiles() renvoie le contenu BRUT du fichier : l'en-tete
            ">..." et les
        // retours a la ligne doivent etre retires avant
            checkNucleotidSequence(),
        // sans quoi un vrai depot .fasta est rejete comme "non valide"
        $sequence = preg_replace('/^>.*$/m', '', $var);
        $sequence = preg_replace('/\s+/', '', $sequence);

        $manager->checkNucleotidSequence($sequence, $a, $g, $t, $c,
            strlen($sequence));
        $length = strlen($sequence);
    }

    return $this->render('minitools/gcContentFinder.html.twig',
        ['form' => $form->createView(), 'length' => $length, 'a' => $a,
            'g' => $g, 't' => $t, 'c' => $c]);
}
```

Implémentation vanilla

```
$manager = new FastaUploaderManager();

$contenuBrut = $manager->createFiles($fichierUploade, '/chemin/vers/
    uploads');
$sequence = preg_replace('/^>.*$/m', '', $contenuBrut);
$sequence = preg_replace('/\s+/', '', $sequence);

$manager->checkNucleotidSequence($sequence, $a, $g, $t, $c, strlen(
    $sequence));
```

Attention

`createFiles()` écrit le fichier téléversé sur disque, avant de le relire pour en extraire le contenu : même pré-requis de dossier accessible en écriture que pour les outils qui génèrent un SVG. C'est aussi l'un des deux seuls outils du chapitre (avec la distance entre séquences) à passer par une écriture disque.

Note

`checkNucleotidSequence()` rejette tout caractère qui n'est pas A/T/G/C strict (pas même les nucléotides dégénérés). Le retrait de la ligne d'en-tête `>` et des retours à la ligne avant l'appel, comme montré ci-dessus, est indispensable pour qu'un vrai fichier FASTA passe la validation.

3.4.6 Température de fusion (Tm)

Estime la température de fusion (Tm) d'un duplex d'ADN à partir de la séquence d'une amorce, selon deux approximations possibles : la formule de base (Wallace, selon la longueur de l'amorce) ou le calcul par empilement de bases voisines (*nearest-neighbor*, SantaLucia 1998 ; von Ahsen *et al.* 1999).

Note

Pour les séquences de moins de 14 nucléotides : $T_m = (w_A + x_T) \times 2 + (y_G + z_C) \times 4$. Au-delà : $T_m = 64,9 + 41 \times (y_G + z_C - 16,4)/(w_A + x_T + y_G + z_C)$. Les deux formules supposent une hybridation à 50 nM d'amorce, 50 mM de Na^+ et pH 7,0. Les nucléotides dégénérés (Y, R, W, S, K, M, D, V, H, B, N) sont substitués en interne avant le calcul, une fois pour la Tm minimale et une fois pour la Tm maximale.

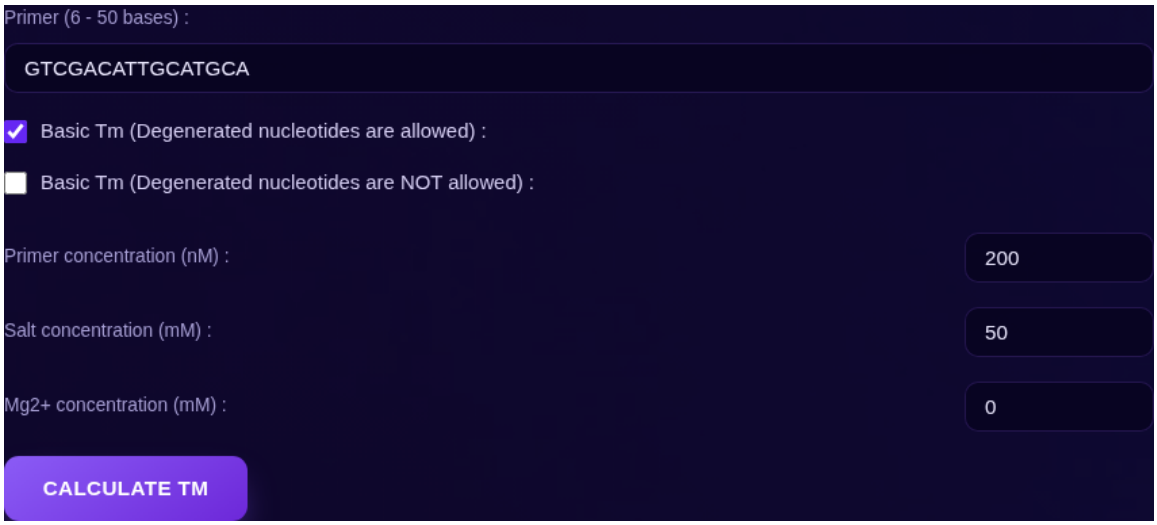

FIGURE 3.12 – Formulaire de calcul de la température de fusion.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`primer`	Primer (6 - 50 bases) *Amorce (6 à 50 bases)*	texte	Validée par la contrainte `MeltingTemperature` (composition NC-IUBMB, longueur 6-50 pb).
`basic`	Basic Tm (Degenerated nucleotides are allowed) *Tm de base (nucléotides dégénérés autorisés)*	case	
`nearestNeighbor`	Basic Tm (Degenerated nucleotides are NOT allowed) *Tm par empilement de bases (nucléotides dégénérés exclus)*	case	
`cp`	Primer concentration (nM) *Concentration en amorce (nM)*	texte	200 par défaut.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
cs	Salt concentration (mM) *Concentration en sel (mM)*	texte	50 par défaut.
cmg	Mg2+ concentration (mM) *Concentration en Mg^{2+} (mM)*	texte	0 par défaut.

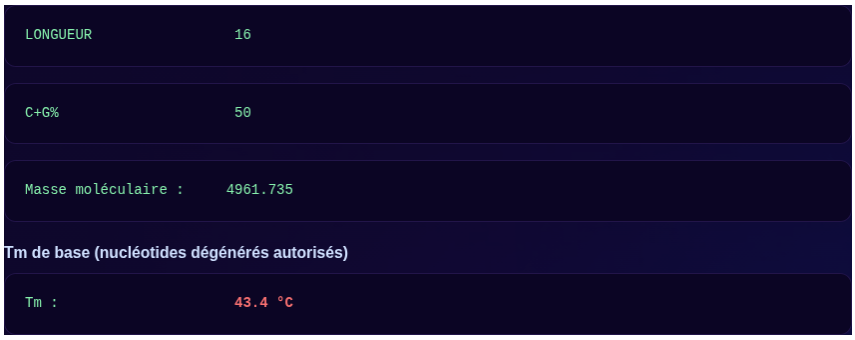

FIGURE 3.13 – Longueur, %GC, masse moléculaire et Tm calculés.

Implémentation Symfony

```
public function meltingTemperatureAction(Request $request,
    MeltingTemperatureManager $manager)
{
    $cg = $upperMwt = $lowerMwt = $countATGC = $tmMin = $tmMax = 0;
    $aTmBaseStacking = [];
    $form = $this->createForm(MeltingTemperatureType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();
        $primer   = $formData["primer"];

        $cg = $manager->calculateCG($primer);
        $manager->calculateMWT($upperMwt, $lowerMwt, $primer);
        $manager->basicCalculations($formData["basic"], $primer,
            $countATGC, $tmMin, $tmMax);
        $manager->neighborCalculations(
            $formData["nearestNeighbor"], $aTmBaseStacking, $primer,
            $formData["cp"], $formData["cs"], $formData["cmg"]
        );
    }

    return $this->render('minitools/meltingTemperature.html.twig',
        ['form' => $form->createView(), 'tmMin' => $tmMin, 'tmMax' =>
            $tmMax]);
}
```

Implémentation vanilla

```php
$manager = new MeltingTemperatureManager($sequenceManager,
    $tmBaseStackingApi);

$cg = $manager->calculateCG('GTCGACATTGCATGCA');
$manager->basicCalculations(true, 'GTCGACATTGCATGCA', $countATGC, $tmMin
    , $tmMax);
// $tmMin / $tmMax contiennent la Tm calculee
```

3.4.7 Analyse de données de puces à ADN (microarray)

Méthode de quantification adaptative avec soustraction locale du bruit de fond : pour chaque spot, la valeur (signal − bruit de fond) est ramenée à un pourcentage de la somme totale, sur les deux canaux ; les ratios ch1/ch2 et ch2/ch1 des réplicats d'un même gène sont ensuite résumés par leur médiane et le logarithme en base 10 de celle-ci. Un code couleur signale la sur- (rouge) ou sous-expression (bleu) dans le rendu de la démo.

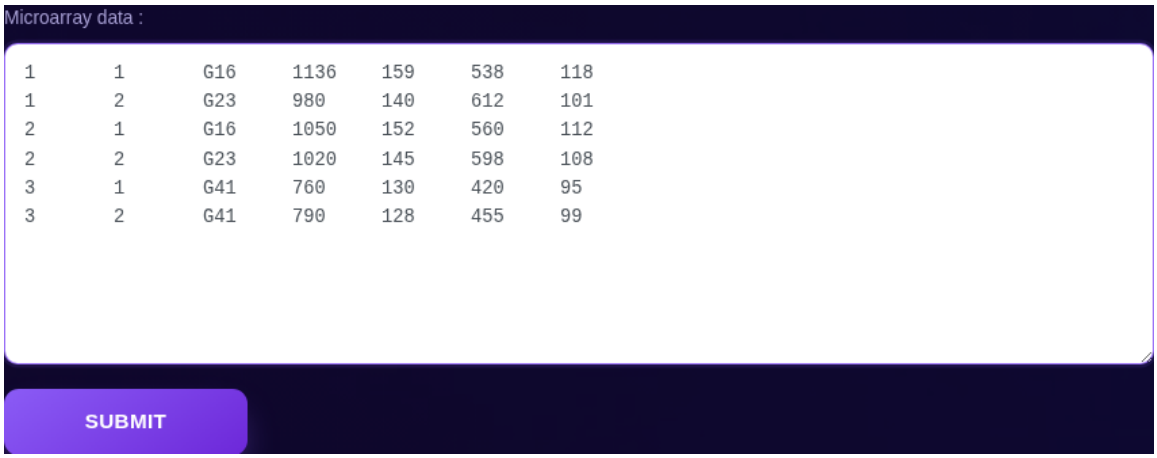

FIGURE 3.14 – Formulaire de dépôt des données de puce à ADN.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`data`	Microarray data *Données de la puce*	texte long	Tableau brut tabulé : colonne, ligne, nom du gène, signal et bruit de fond des deux canaux. Un jeu de données d'exemple (10 gènes) est pré-rempli.

Identification	Nb de données	Canal 1	log10	Canal 2	log10
G16	2	1.131	0.053	0.889	-0.051
G23	2	0.896	-0.048	1.118	0.049
G41	2	0.992	-0.004	1.009	0.004

FIGURE 3.15 – Effectif, médianes et logarithmes par gène.

Implémentation Symfony

```php
public function microArrayAnalysisAdaptiveQuantificationAction(Request $request,
    MicroarrayAnalysisAdaptiveManager $manager)
{
    $results = [];
    $form = $this->createForm(MicroArrayDataAnalysisType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->isValid()) {
        $formData = $form->getData();
        $results = $manager->processMicroarrayDataAdaptiveQuantificationMethod($formData["data"]);
    }

    return $this->render('minitools/microArrayAnalysisAdaptiveQuantification.html.twig',
        ['form' => $form->createView(), 'results' => $results]);
}
```

Implémentation vanilla

```php
$manager = new MicroarrayAnalysisAdaptiveManager();
$resultats = $manager->processMicroarrayDataAdaptiveQuantificationMethod($donneesBrutes);
// $resultats['NomDuGene'] = ['n_data' => .., 'median1' => .., 'medlog1' => .., ...]
```

Note

Le format attendu est strict : chaque ligne doit avoir au moins 7 colonnes séparées par des tabulations (position, position, nom, signal canal 1, bruit canal 1, signal canal 2, bruit canal 2) — les lignes plus courtes sont silencieusement ignorées. Un signal inférieur ou égal au bruit de fond (spot faible ou raté) est plafonné à une valeur positive minime plutôt que de produire un logarithme indéfini.

3.4.8 Recherche de répétitions microsatellites

Recherche les répétitions en tandem simples (di-, tri-, tétra-, pentanucléotidiques…) dans une séquence — aussi connues sous les noms de *Simple Tandem Repeats* (STR) ou *Simple Sequence Repeats* (SSR), généralement longues de moins de 100 bases.

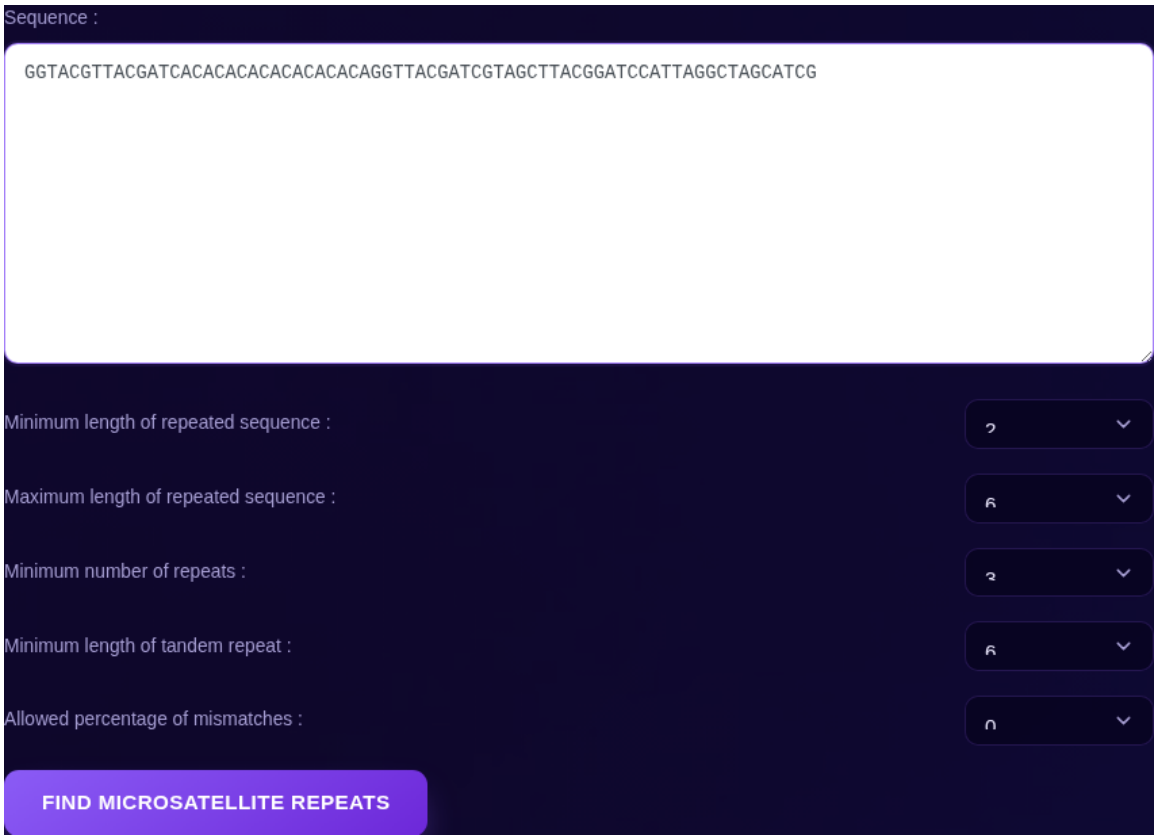

FIGURE 3.16 – Formulaire de recherche de microsatellites.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`sequence`	Sequence *Séquence*	texte long	ADN à analyser ; un exemple est pré-rempli.
`min`	Minimum length of repeated sequence *Longueur minimale du motif répété*	choix	2 à 6.
`max`	Maximum length of repeated sequence *Longueur maximale du motif répété*	choix	3 à 10 ; défaut : 6.
`min_repeats`	Minimum number of repeats *Nombre minimal de répétitions*	choix	2 à 6 ; défaut : 3.
`length_of_MR`	Minimum length of tandem repeat *Longueur minimale de la répétition en tandem*	choix	5 à 20 ; défaut : 6.
`mismatch`	Allowed percentage of mismatches *Pourcentage de mésappariements autorisé*	choix	0, 10, 20 ou 30 %.

Position	Cycle	Répétitions	Séquence
13	2	9	CACACACACACACACACA

FIGURE 3.17 – Position, période, nombre de répétitions et motif trouvés.

Implémentation Symfony

```
public function microsatelliteRepeatsFinderAction(Request $request,
    MicrosatelliteRepeatsFinderManager $manager)
{
    $results = [];
    $form = $this->createForm(MicrosatelliteRepeatsFinderType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();
        $results = $manager->findMicrosatelliteRepeats(
            $formData["sequence"], $formData["min"], $formData["max"],
            $formData["min_repeats"], $formData["length_of_MR"],
                $formData["mismatch"]
        );
    }

    return $this->render('minitools/microsatelliteRepeatsFinder.html.
        twig',
        ['form' => $form->createView(), 'results' => $results]);
}
```

Implémentation vanilla

```
$manager = new MicrosatelliteRepeatsFinderManager();
$resultats = $manager->findMicrosatelliteRepeats($sequenceADN, 2, 6, 3,
    6, 0);
```

3.4.9 Fréquence des (oligo)nucléotides

Compte la fréquence de chaque oligonucléotide d'une longueur donnée (1 à 8 bases) dans une séquence, sur un seul brin ou sur les deux.

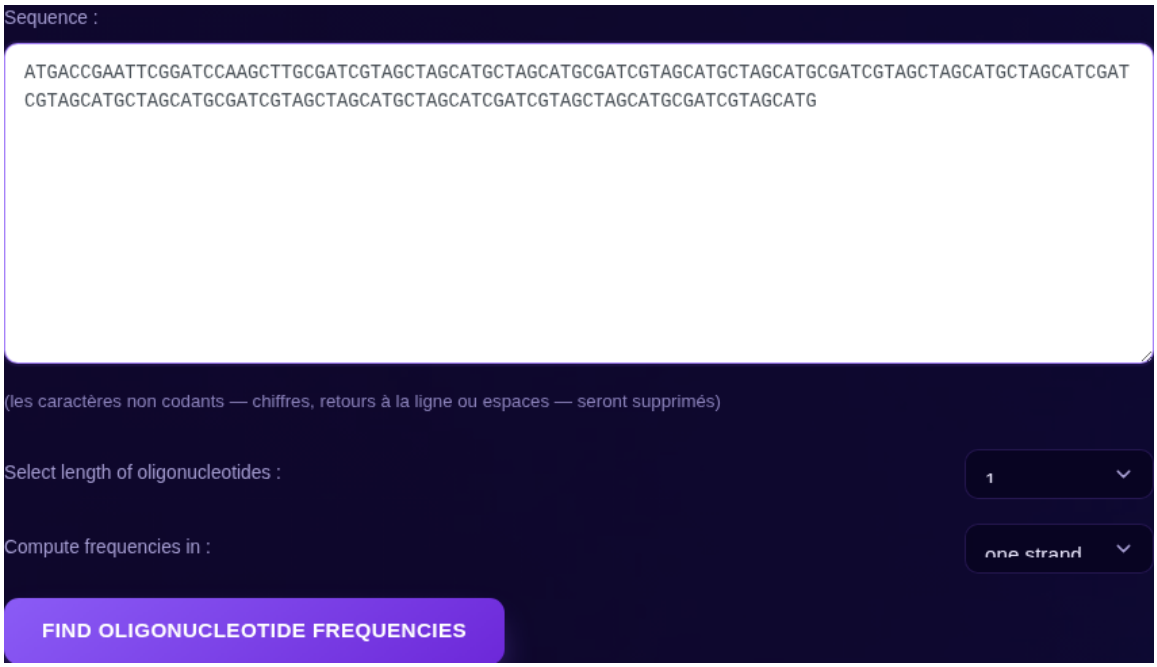

FIGURE 3.18 – Formulaire de calcul des fréquences d'oligonucléotides.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`sequence`	Sequence *Séquence*	texte long	ADN à analyser ; un exemple est pré-rempli. Doit compter au moins 4^{len} bases.
`len`	Select length of oligonucleotides *Longueur des oligonucléotides*	choix	1 à 8.
`strands`	Compute frequencies in *Calculer les fréquences sur*	choix	un brin ou les deux.

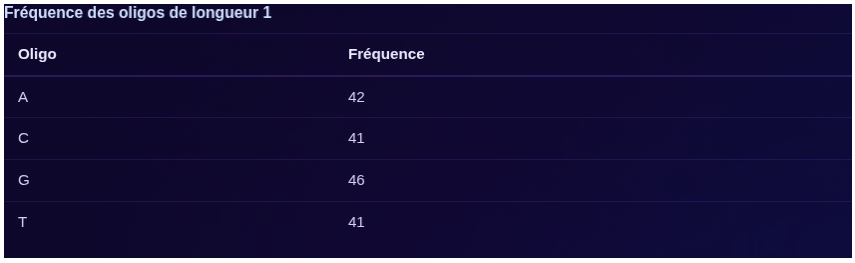

FIGURE 3.19 – Fréquence de chaque oligonucléotide trouvé.

Implémentation Symfony

```
public function oligonucleotideFrequencyAction(Request $request,
    NucleotidApiAdapter $nucleotidApi, OligosInterface $oligos)
```

```
{
    $results = []; $length = 0;
    $form = $this->createForm(OligoNucleotideFrequencyType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();
        $length   = $formData["len"];
        $sequence = $formData["sequence"];

        if ($formData["strands"] == 2) {
            $complements = $nucleotidApi::GetDNAComplement($nucleotidApi
                ->getNucleotids());
            GeneticsFunctions::createInversion($sequence, $complements);
        }

        $results = $oligos->findOligos($sequence, $length);
        ksort($results);
    }

    return $this->render('minitools/oligonucleotideFrequency.html.twig',
        ['form' => $form->createView(), 'results' => $results]);
}
```

Implémentation vanilla

```
$oligos = new OligosManager(); // depuis BioPHP, voir chapitre 2
$resultats = $oligos->findOligos($sequenceADN, 1);
// $resultats['A'] = 42, $resultats['C'] = 41, ...
```

Note

Seul outil du chapitre à ne pas avoir son propre manager : il réutilise directement `OligosManager` de BioPHP (chapitre 2, §2.7). Pour calculer sur les deux brins, le contrôleur construit lui-même le brin inversé complémentaire avant l'appel — ce n'est pas `findOligos()` qui s'en charge.

3.4.10 Amplification PCR *in silico*

Simule une amplification PCR : cherche dans une séquence toutes les positions où l'amorce 1 (sens) est suivie, à moins de la longueur maximale indiquée, du complément inversé de l'amorce 2, et liste les amplicons trouvés avec leur position et leur longueur.

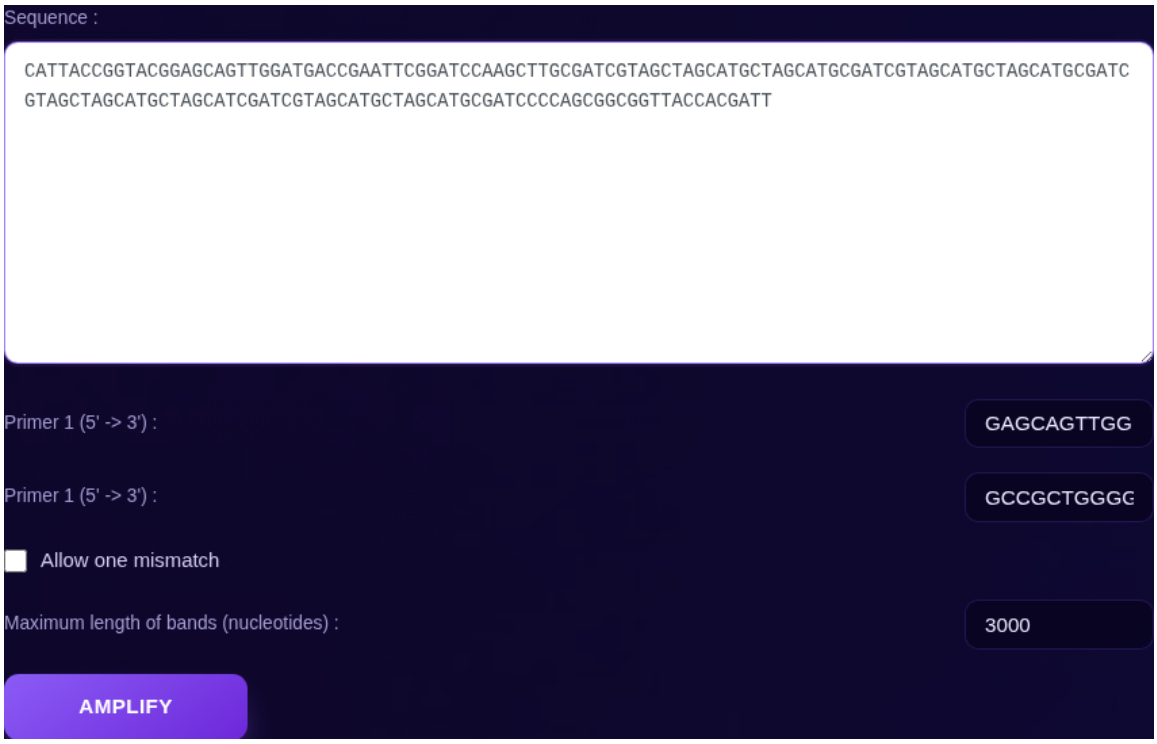

FIGURE 3.20 – Formulaire d'amplification PCR.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`sequence`	Sequence *Séquence*	texte long	ADN dans lequel chercher l'amplicon ; un exemple est pré-rempli.
`primer1`	Primer 1 (5' -> 3') *Amorce 1 (5' vers 3')*	texte	`GAGCAGTTGG` par défaut.
`primer2`	Primer 2 (5' -> 3') *Amorce 2 (5' vers 3')*	texte	`GCCGCTGGGG` par défaut.
`allowmismatch`	Allow one mismatch *Tolérer un mésappariement*	case	
`length`	Maximum length of bands (nucleotides) *Longueur maximale des bandes (nucléotides)*	texte	3 000 par défaut.

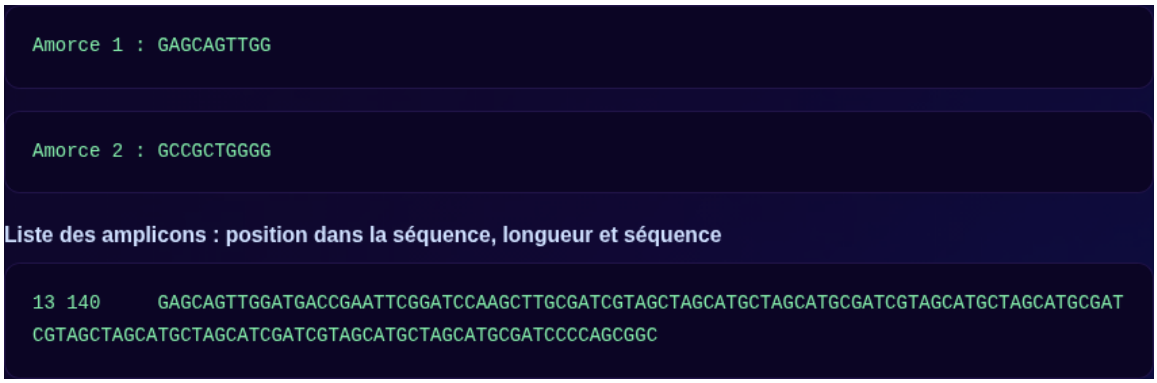

FIGURE 3.21 – Amplicons trouvés : position, longueur et séquence.

Implémentation Symfony

```php
public function pcrAmplificationAction(Request $request,
    PcrAmplificationManager $manager)
{
    $results = [];
    $form = $this->createForm(PcrAmplificationType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();

        $startPattern = $manager->createStartPattern(
            $formData["primer1"], $formData["primer2"], $formData["
                allowmismatch"]
        );
        $endPattern = $manager->createEndPattern($startPattern);

        $results = $manager->amplify($startPattern, $endPattern,
            $formData["sequence"], $formData["length"]);
    }

    return $this->render('minitools/pcrAmplification.html.twig',
        ['form' => $form->createView(), 'results' => $results]);
}
```

Implémentation vanilla

```php
$manager = new PcrAmplificationManager($nucleotidApi);

$startPattern = $manager->createStartPattern('GAGCAGTTGG', 'GCCGCTGGGG',
    false);
$endPattern   = $manager->createEndPattern($startPattern);
$amplicons    = $manager->amplify($startPattern, $endPattern,
    $sequenceADN, 3000);
// $amplicons[position_de_depart] = longueur_de_l_amplicon
```

Attention

Le schéma de recherche est contre-intuitif au premier abord : `createEndPattern()` ne travaille pas sur l'amorce 2 seule, mais sur le complément inversé de `startPattern` tout entier (les deux amorces séparées par un « ou » régulière). Dans la pratique, cela revient à chercher l'amorce 1 telle quelle en début d'amplicon, puis le complément inversé de l'amorce 2 en fin d'amplicon — exactement ce qu'on attend d'une PCR réelle, où l'amorce 2 s'hybride sur le brin complémentaire.

3.4.11 Propriétés d'une séquence protéique

Calcule, sur tout ou partie d'une séquence protéique : la composition en acides aminés, la masse moléculaire, le coefficient d'absorption molaire, le point isoélectrique, la charge à un pH donné, et présente la séquence en code à 3 lettres ou colorée par nature physico-chimique des résidus. Les codes NC-IUBMB servent de référence, et les caractères non codants sont supprimés par défaut.

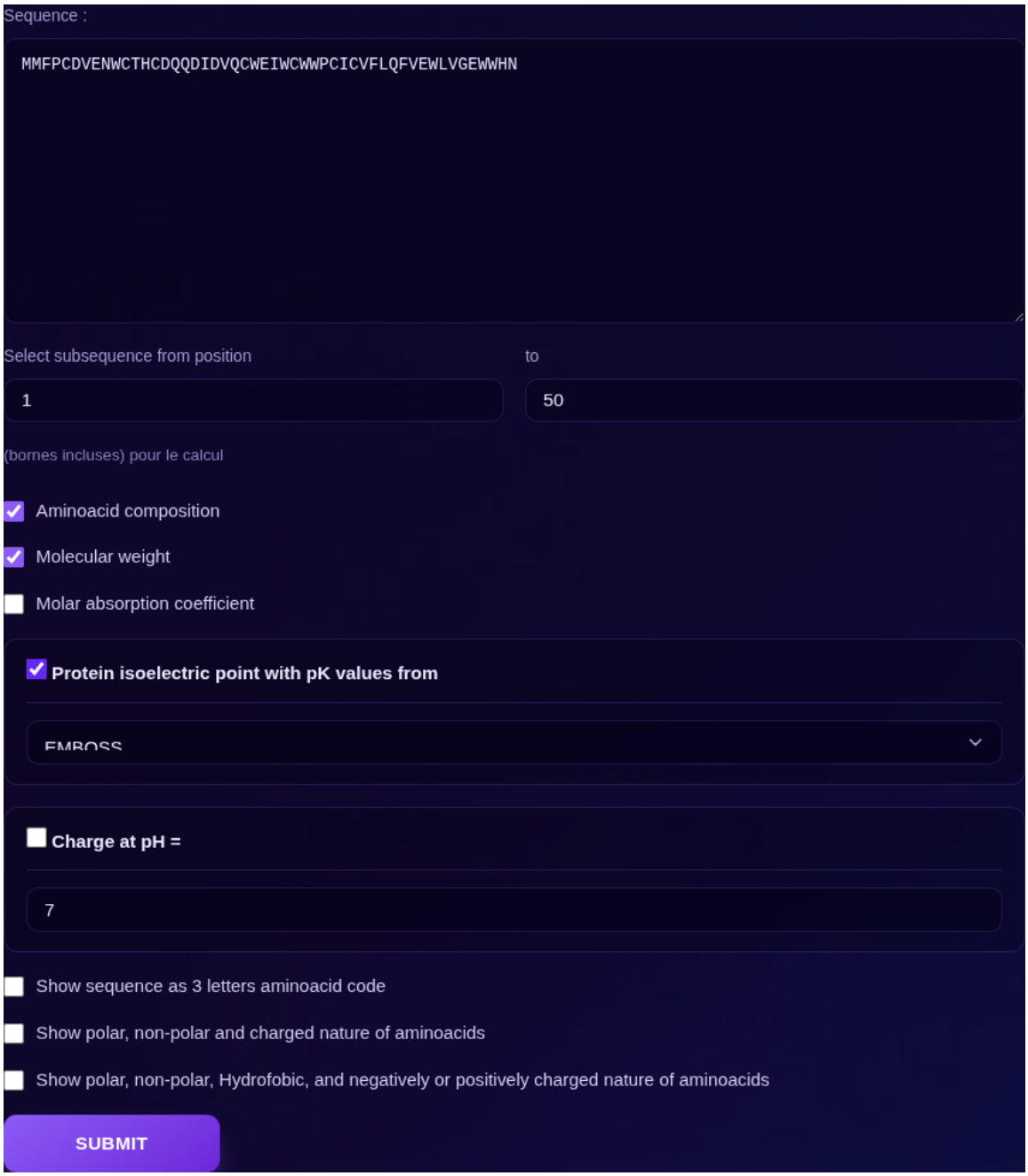

FIGURE 3.22 – Formulaire de calcul des propriétés d'une séquence protéique.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`seq`	Sequence *Séquence*	texte long	Protéine à analyser.
`start`	Select subsequence from position *Sous-séquence à partir de la position*	texte	Vide par défaut (voir encadré Attention ci-dessous).
`end`	to *jusqu'à la position*	texte	Vide par défaut.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`composition`	Aminoacid composition *Composition en acides aminés*	case	
`molweight`	Molecular weight *Masse moléculaire*	case	
`abscoef`	Molar absorption coefficient *Coefficient d'absorption molaire*	case	
`charge`	Protein isoelectric point with pK values from *Point isoélectrique, valeurs de pK selon*	case	Utilise le référentiel choisi dans `data_source`.
`data_source`	(référentiel de pK) *Référentiel de pK*	choix	EMBOSS / DTASelect / Solomon.
`charge2`	Charge at pH = *Charge au pH*	case	
`pH`	(valeur du pH) *pH*	texte	7 par défaut.
`three_letters`	Show sequence as 3 letters aminoacid code *Afficher en code 3 lettres*	case	
`type1`	Show polar, non-polar and charged nature *Nature polaire / apolaire / chargée*	case	Polaire : SCTNQHYW ; apolaire : GAPVILFM ; chargé : DEKR.
`type2`	Show polar, non-polar, hydrophobic... nature *Nature détaillée (5 catégories)*	case	

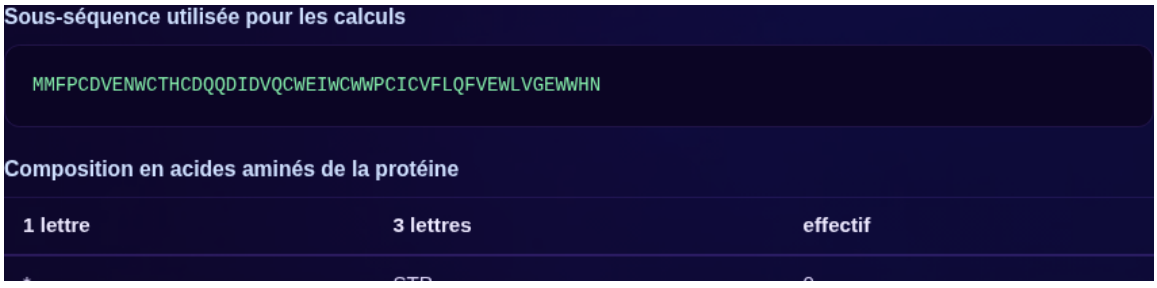

FIGURE 3.23 – Composition, masse moléculaire et autres propriétés calculées.

Implémentation Symfony

```
1 public function proteinPropertiesAction(Request $request,
      ProteinPropertiesManager $manager)
2 {
3     $aminoacids = [];
4     $molweight = 0;
```

```
    $form = $this->createForm(ProteinPropertiesType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();

        $seq = GeneticsFunctions::removeNonCodingProt($formData["seq"]);
        $subsequence = $manager->writeSubsequence($formData["start"],
            $formData["end"], $seq);
        $aminoacidContent = $manager->aminoacidContent($subsequence);

        $manager->setPk($formData["data_source"]);

        if ($formData["composition"]) {
            $aminoacids = $manager->formatAminoacidContent(
                $aminoacidContent);
        }
        if ($formData["molweight"]) {
            $molweight = $manager->proteinMolecularWeight(
                $aminoacidContent);
        }
    }

    return $this->render('minitools/proteinProperties.html.twig',
        ['form' => $form->createView(), 'aminoacids' => $aminoacids, '
            molweight' => $molweight]);
}
```

Implémentation vanilla

```
$manager = new ProteinPropertiesManager($pkApi);

$sous_sequence = $manager->writeSubsequence(1, strlen($proteine),
    $proteine);
$composition = $manager->aminoacidContent($sous_sequence);
$manager->setPk('EMBOSS');
$masse = $manager->proteinMolecularWeight($composition);
```

Attention

`writeSubsequence($start, $end, $seq)` ne renvoie la séquence entière que si **au moins un** des deux champs est renseigné ; si **`start` et `end`** sont tous les deux vides, la sous-séquence renvoyée est une **chaîne vide**, et tous les calculs qui en dépendent (composition, masse, etc.) sortent à zéro sans qu'aucune erreur ne soit levée. Toujours renseigner au moins `start=1` et `end=` longueur de la séquence pour l'analyser en entier.

3.4.12 Rétrotraduction protéine vers ADN

Retraduit une séquence protéique en séquence d'ADN, en choisissant à chaque position le triplet le plus probable pour l'espèce (ou le code génétique) choisi — les codes génétiques et leurs triplets viennent de BioAPI (chapitre 1).

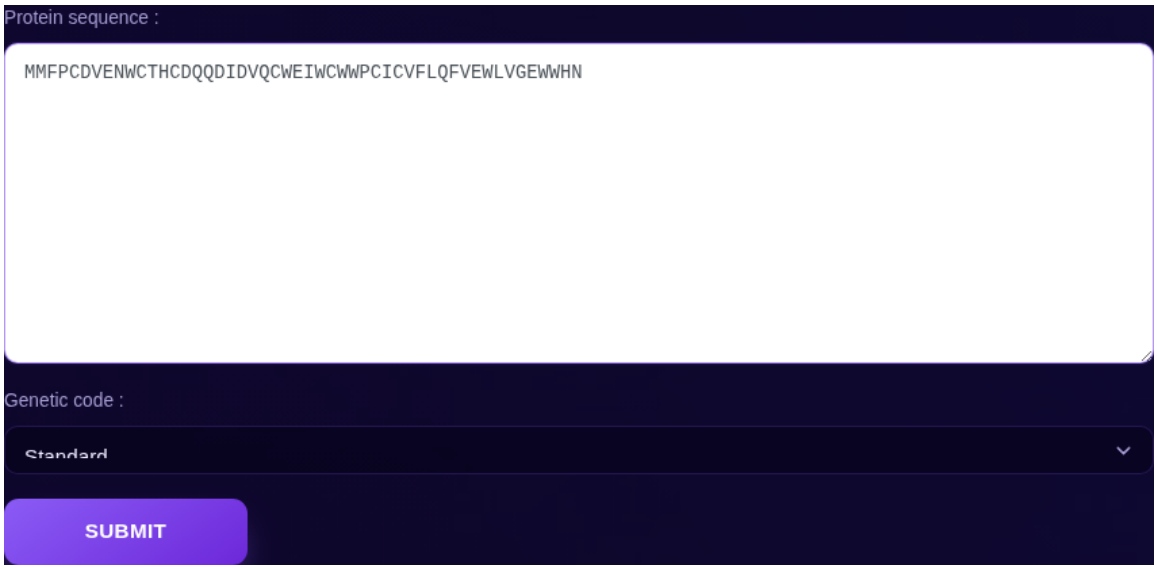

FIGURE 3.24 – Formulaire de rétrotraduction protéine vers ADN.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`sequence`	Protein sequence *Séquence protéique*	texte long	`FLIMVSPTAYHQNKDECWRGX*` par défaut (tous les codes 1-lettre valides).
`genetic_code`	Genetic code *Code génétique*	choix	Liste des espèces/codes disponibles sur BioAPI, récupérée dynamiquement au chargement du formulaire.

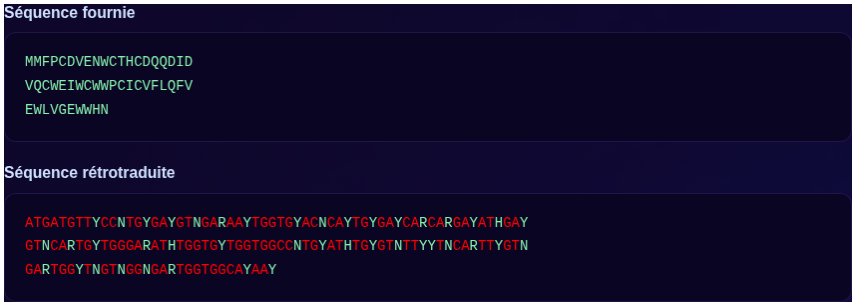

FIGURE 3.25 – Séquence d'ADN rétrotraduite.

Implémentation Symfony

```
public function proteinToDnaAction(Request $request, ProteinToDnaManager
    $manager)
{
    $dna = "";
    $form = $this->createForm(ProteinToDnaType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
         $formData = $form->getData();
```

```
        $dna = $manager->translateProteinToDNA($formData["sequence"],
            $formData["genetic_code"]);
    }

    return $this->render('minitools/proteinToDna.html.twig',
        ['form' => $form->createView(), 'dna' => $dna]);
}
```

Implémentation vanilla

```
$manager = new ProteinToDnaManager($tripletSpecieApi);
$adn = $manager->translateProteinToDNA('MAVLK', 'Standard');
```

Note

`ProteinToDnaType` interroge `TripletSpecieApiAdapter` dès sa construction pour peupler la liste des codes génétiques : en « vanilla », il faut faire cet appel réseau vous-même avant d'afficher les choix possibles à l'utilisateur.

3.4.13 Séquences aléatoires

Génère une séquence aléatoire selon trois procédés : en mélangeant aléatoirement les caractères d'une séquence existante, en tirant des nucléotides ADN selon des fréquences A/C/G/T choisies, ou en tirant des acides aminés selon les 20 fréquences (les valeurs par défaut sont celles du génome humain).

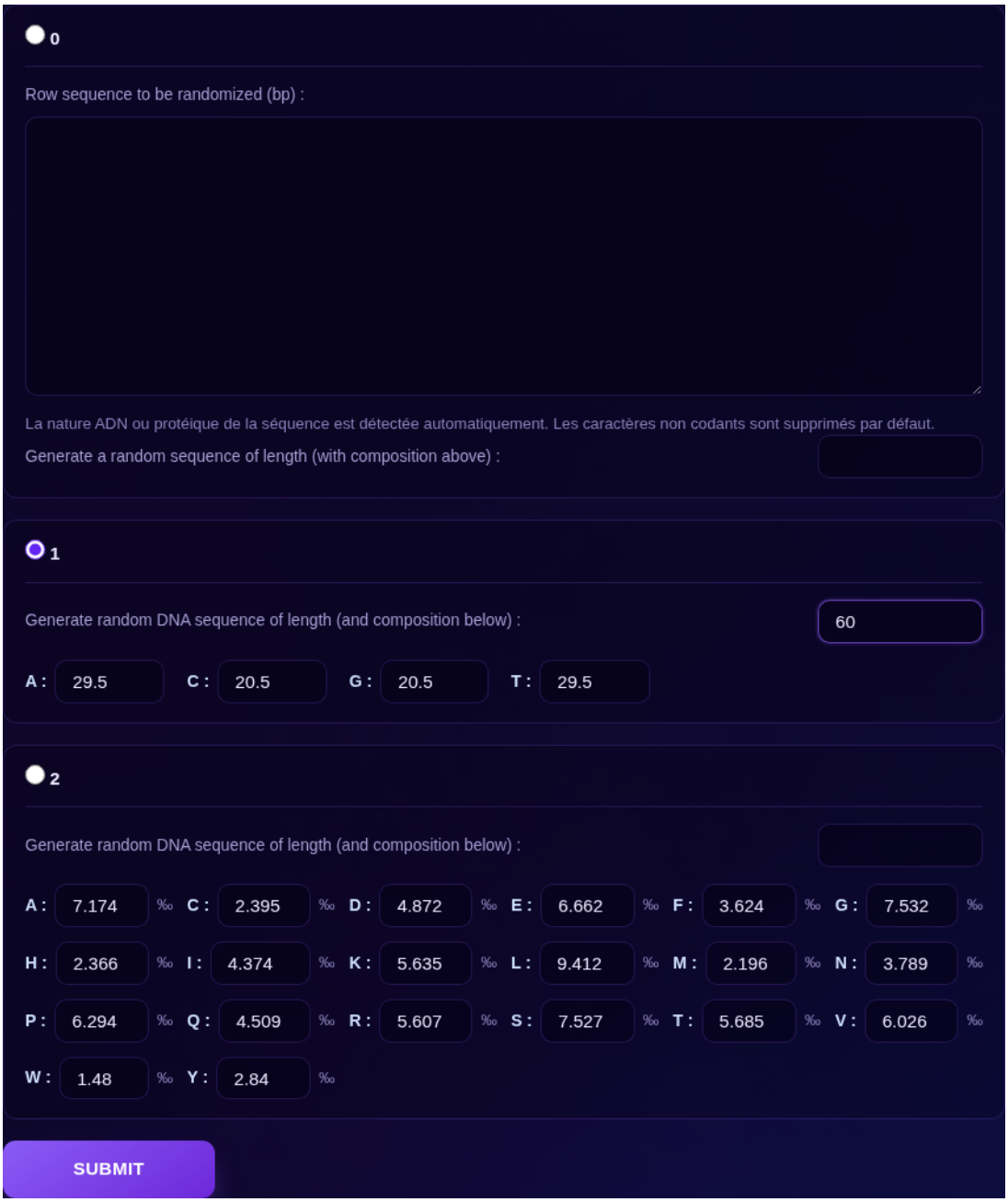

FIGURE 3.26 – Formulaire de génération de séquences aléatoires (28 champs, groupés en trois procédés).

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`procedure`	(choix du procédé) *Procédé*	radio	`fromseq` / `fromACGT` / `fromAA`.
`seq`	Row sequence to be randomized *Séquence à mélanger*	texte long	Utilisée avec `fromseq`.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`length1`	Generate a random sequence of length (with composition above) *Longueur (même composition)*	texte	Utilisé avec `fromseq`.
`length2`	Generate random DNA sequence of length (and composition below) *Longueur de la séquence ADN*	texte	Utilisé avec `fromACGT`.
`dnaA, dnaC, dnaG, dnaT`	A / C / G / T *Fréquences A/C/G/T (‰)*	texte	29.5 / 20.5 / 20.5 / 29.5 par défaut.
`length3`	(longueur pour fromAA) *Longueur de la séquence protéique*	texte	Utilisé avec `fromAA`.
`a, c, d, e, f, g, h, i, k, l, m, n, p, q, r, s, t, v, w, y`	(20 fréquences d'acides aminés) *Fréquences des 20 acides aminés (‰)*	texte	Valeurs par défaut fondées sur le protéome humain (ex. : L = 9.412, W = 1.48).

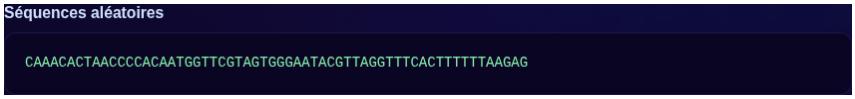

FIGURE 3.27 – Séquence aléatoire générée.

Implémentation Symfony

```
public function randomSeqsAction(Request $request,
    RandomSequencesManager $manager)
{
    $result = "";
    $form = $this->createForm(RandomSequencesType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();

        switch ($formData["procedure"]) {
            case "fromseq":
                $result = $manager->createFromSeq($formData["length1"],
                    $formData["seq"]);
                break;
            case "fromACGT":
                $freqs = ['A' => $formData["dnaA"], 'C' => $formData["
                    dnaC"],
                          'G' => $formData["dnaG"], 'T' => $formData["
                              dnaT"]];
```

```
                $result = $manager->createFromACGT($freqs, $formData["
                    length2"]);
                break;
            case "fromAA":
                $freqs = []; // les 20 frequences, cf. formData['a'] ..
                    formData['y']
                $result = $manager->createFromAA($freqs, $formData["
                    length3"]);
                break;
        }
    }

    return $this->render('minitools/randomSeqs.html.twig',
        ['form' => $form->createView(), 'results' => $result]);
}
```

Implémentation vanilla

```
$manager = new RandomSequencesManager($nucleotidApi, $aminoApi);

$sequence = $manager->createFromACGT(
    ['A' => 29.5, 'C' => 20.5, 'G' => 20.5, 'T' => 29.5],
    60
);
```

3.4.14 Alphabet protéique réduit

Cet outil regroupe les acides aminés d'une séquence protéique en un alphabet plus restreint, selon l'une des classifications publiées (Murphy, Wang & Wang, Li *et al.*, IMGT…), ou selon un alphabet personnalisé fourni par l'utilisateur.

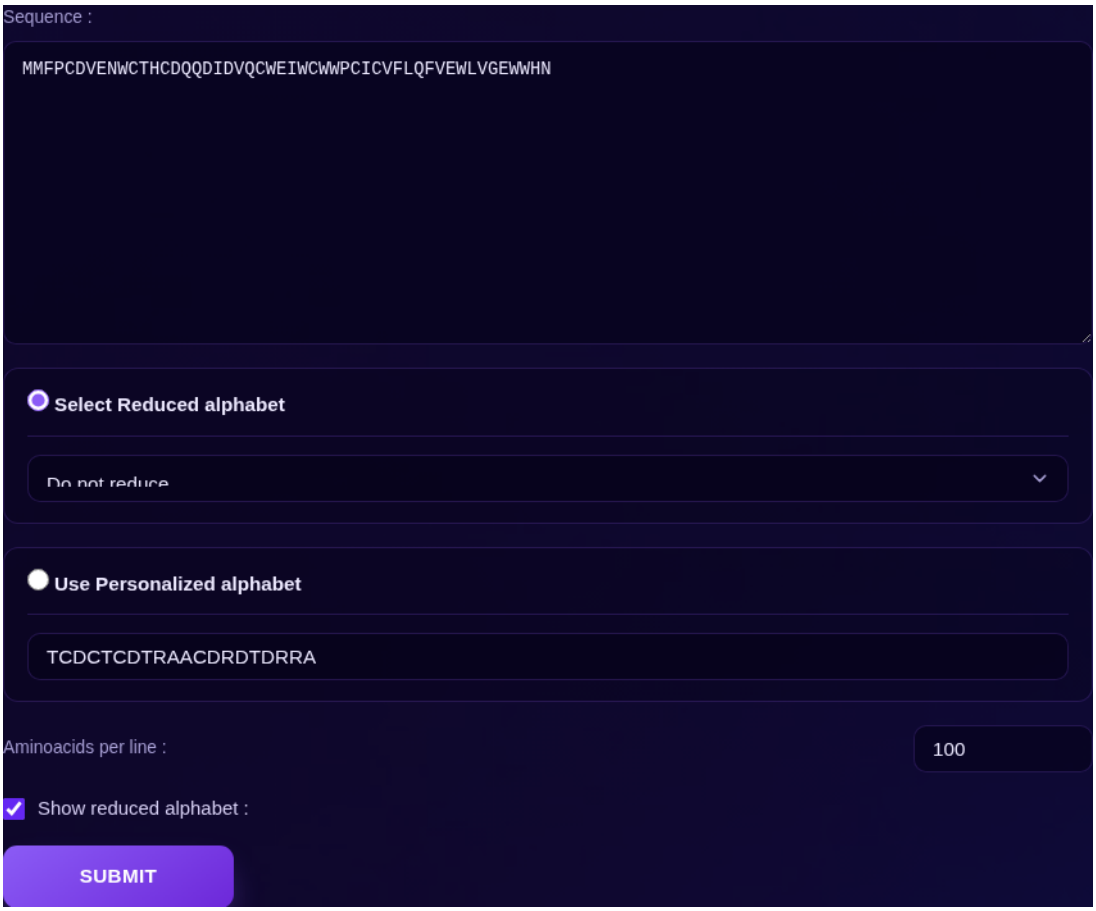

FIGURE 3.28 – Formulaire de réduction d'alphabet protéique.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`seq`	Sequence *Séquence protéique*	texte long	Alphabet `ARNDCEQGHILKMFPSTWYVX*` par défaut.
`mode`	Select Reduced alphabet / Use Personalized alphabet *Alphabet prédéfini ou personnalisé*	radio	`pre` ou `custom`.
`type`	(liste des alphabets réduits) *Type de réduction*	liste déroulante	Utilisé avec `mode=pre` : Murphy (2/4/8/10/15), Wang & Wang (2/3/5), Li *et al.* (3/4/5/10), IMGT (3/5/11), etc.
`custom_alphabet`	(chaine de 20 lettres) *Alphabet personnalisé*	texte	Utilisé avec `mode=custom` ; doit compter exactement 20 caractères, un par acide aminé de la série `ARNDCEQGHILKMFPSTWYV`.
`aaperline`	Aminoacids per line *Acides aminés par ligne*	texte	100 par défaut, pour la mise en forme de l'affichage.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`show_reduced`	Show reduced alphabet *Afficher l'alphabet réduit*	case à cocher	Affiche en plus la table de correspondance utilisée.

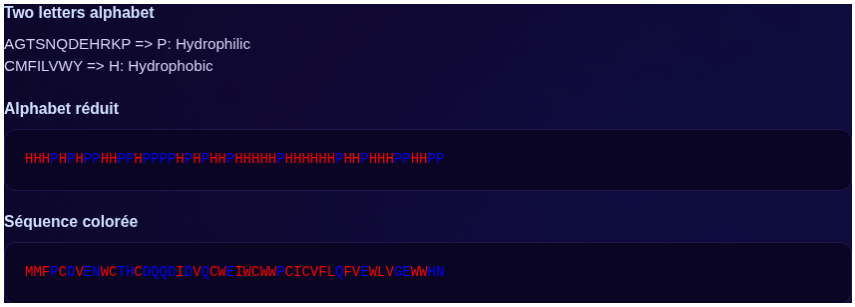

FIGURE 3.29 – Séquence réduite, avec coloration par lettre d'alphabet.

Implémentation Symfony

```php
public function reduceProteinAlphabetAction(Request $request,
    ReduceProteinAlphabetManager $manager)
{
    $sequence = $reducedSequence = "";
    $reduceCode = [];
    $form = $this->createForm(ReduceAlphabetType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();
        $sequence = $formData["seq"];

        if ($formData["mode"] === "pre") {
            $reducedSequence = $manager->reduceAlphabet($sequence,
                $formData["type"]);
            if ($formData["show_reduced"]) {
                $reduceCode = $manager->createReduceCode($formData["type
                    "]);
            }
        } else {
            $reducedSequence = $manager->reduceAlphabetCustom($sequence,
                $formData["custom_alphabet"]);
        }
    }

    return $this->render('minitools/reduceProteinAlphabet.html.twig', [
        'form'       => $form->createView(),
        'sequence'   => $reducedSequence,
        'aaperline' => $formData['aaperline'] ?? 100,
        'reduce_code' => $reduceCode,
    ]);
}
```

Implémentation vanilla

```php
$manager = new ReduceProteinAlphabetManager($proteinColors, $proteinReductionApi);

// alphabet predefini (Murphy et al., 10 groupes)
$reduced = $manager->reduceAlphabet('ARNDCEQGHILKMFPSTWYVX*', 'Murphy10');

// alphabet personnalise (20 lettres, une par acide amine de ARNDCEQGHILKMFPSTWYV)
$reducedCustom = $manager->reduceAlphabetCustom(
    'ARNDCEQGHILKMFPSTWYVX*',
    'TCDCTCDTRAACDRDTDRRA'
);
```

Note

La validation du champ `custom_alphabet` (exactement 20 caractères) n'est appliquée que lorsque `mode` vaut `custom` : choisir un alphabet prédefini dispense de fournir une chaine de 20 lettres correcte.

3.4.15 Digestion par enzymes de restriction

Cet outil recherche, pour une ou plusieurs séquences d'ADN, les sites de coupure de l'ensemble (ou d'un sous-ensemble filtré) des endonucléases de restriction connues, et retourne pour chacune le nombre et la position des coupures.

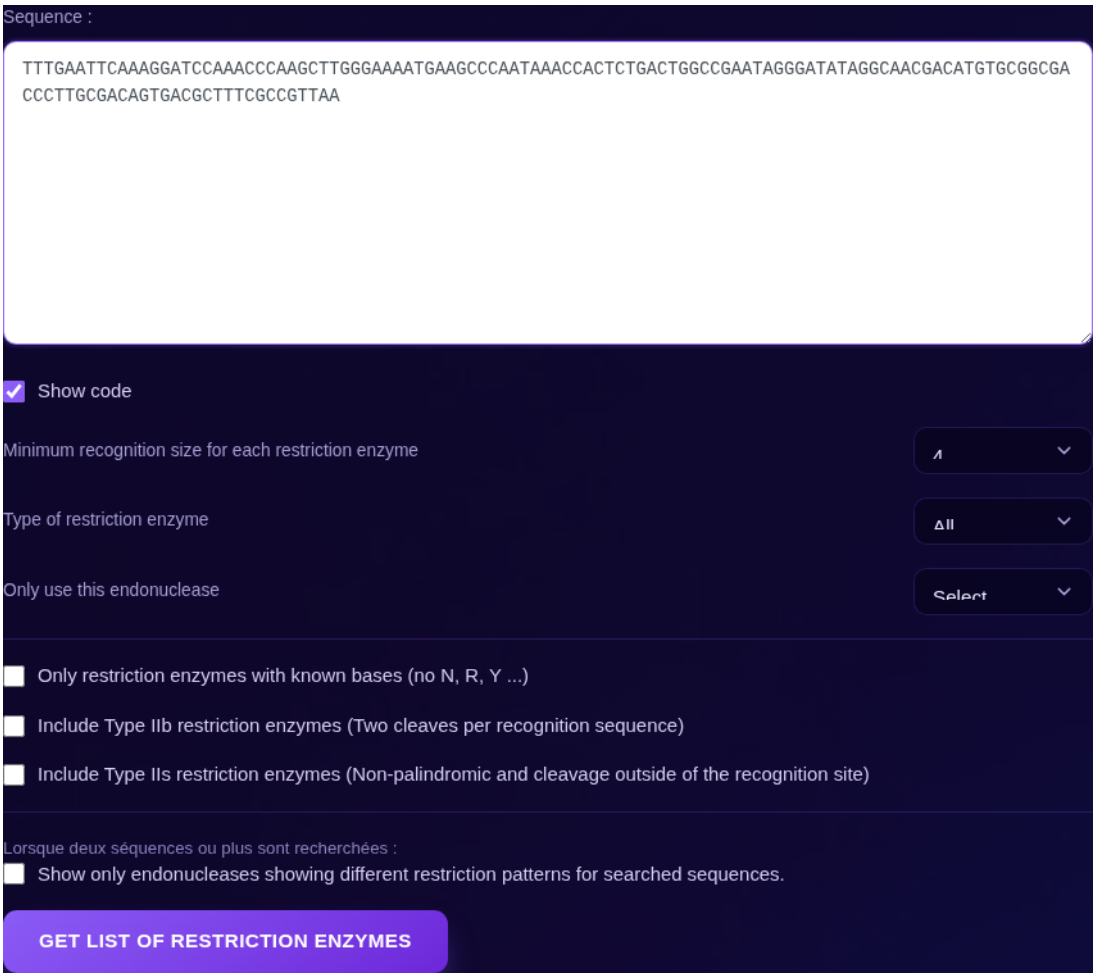

FIGURE 3.30 – Formulaire de digestion par enzymes de restriction.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`sequence`	Sequence *Séquence(s) à analyser*	texte long	Une séquence brute, ou plusieurs séquences au format FASTA (`>nom`). 1 000 000 caractères maximum.
`showcode`	Show code *Afficher le code*	case à cocher	Cochée par défaut.
`minimum`	Minimum recognition size for each restriction enzyme *Taille minimale du site de reconnaissance*	liste déroulante	1 à 8, 4 par défaut.
`retype`	Type of restriction enzyme *Type de coupure*	liste déroulante	`All` (0) / `Blunt ends` (1, bouts francs) / `Overhang end` (2, extrémités cohesives).
`wre`	Only use this endonuclease *Se limiter à cette enzyme*	liste déroulante	Vide (= toutes les enzymes) ou le nom d'une enzyme précise.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`defined`	Only restriction enzymes with known bases (no N, R, Y ...) *Enzymes à bases entièrement définies*	case à cocher	Exclut les motifs contenant des bases dégénérées.
`IIb`	Include Type IIb restriction enzymes *Inclure les enzymes de type IIb*	case à cocher	Deux coupures par site de reconnaissance.
`IIs`	Include Type IIs restriction enzymes *Inclure les enzymes de type IIs*	case à cocher	Coupure non palindromique, en dehors du site de reconnaissance.
`onlydiff`	Show only endonucleases showing different restriction patterns for searched sequences *N'afficher que les profils différents*	case à cocher	Pertinent uniquement avec plusieurs séquences en entrée.

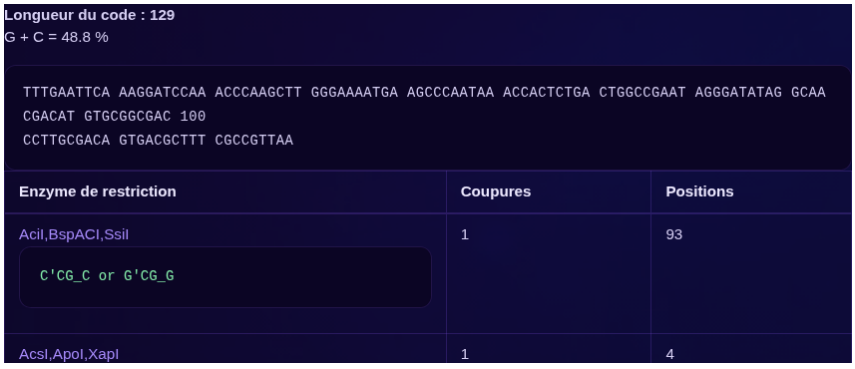

FIGURE 3.31 – Liste des enzymes et de leurs sites de coupure.

Implémentation Symfony

```
public function restrictionDigestAction(Request $request,
    RestrictionDigestManager $manager)
{
    $sequence = $digestion = $enzymes_array = $digestionMulti = [];
    $bShowCode = false;
    $form = $this->createForm(RestrictionEnzymeDigestType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData  = $form->getData();
        $bShowCode = $formData["showcode"];
        $sequence  = $manager->extractSequences($formData["sequence"]);

        $enzymes_array = $manager->getNucleolasesInfos(
            $formData["IIs"], $formData["IIb"], $formData["defined"]
        );
        $enzymes_array = $manager->reduceEnzymesArray(
            $enzymes_array, $formData["minimum"], $formData["retype"],
            $formData["defined"], $formData["wre"]
```

```
        );

        foreach ($sequence as $number => $val) {
            $digestion[$number] = $manager->restrictionDigest(
                $enzymes_array, $sequence[$number]["seq"]);
        }

        if (count($sequence) > 1) {
            $digestionMulti = $manager->enzymesForMultiSeq(
                $sequence, $digestion, $enzymes_array, $formData["
                    onlydiff"], $formData["wre"]
            );
        }
    }

    return $this->render('minitools/restrictionDigest.html.twig', [
        'form'            => $form->createView(),
        'show_code'       => $bShowCode,
        'sequence'        => $sequence,
        'digestion'       => $digestion,
        'digestion_multi' => array_unique($digestionMulti),
        'enzymes_array'   => $enzymes_array,
    ]);
}

// route annexe : /minitools/show-vendors/{enzyme}, appelee en Ajax
    depuis le
// template pour afficher les fournisseurs commerciaux d'une enzyme
    donnee
public function showVendorsAction($enzyme, RestrictionDigestManager
    $manager)
{
    $message = "";
    $enzyme_array = $manager->getVendors($message, $enzyme);
    return new JsonResponse(["message" => $message, "vendors" =>
        $enzyme_array]);
}
```

Implémentation vanilla

```
$manager = new RestrictionDigestManager(
    $vendorLinksApi, $typeIIEndonucleaseApi, $typeIIbEndonucleaseApi,
    $typeIIsEndonucleaseApi, $vendorApiAdapter
);

$sequence = $manager->extractSequences(">seq1\
    nACGTACGTACGTTAGCTAGCTAGCTAGC");
$enzymes  = $manager->getNucleolasesInfos(false, false, false);
$enzymes  = $manager->reduceEnzymesArray($enzymes, 4, 0, false, "");

$digestion = $manager->restrictionDigest($enzymes, $sequence[0]["seq"]);
```

Note

La route `/minitools/show-vendors/{enzyme}` est un cas particulier : c'est la seule route de BioTools qui ne rend pas un formulaire, mais répond en JSON. Dans la démo, elle est

appelée en arrière-plan (Ajax) lorsque l'utilisateur clique sur le nom d'une enzyme dans les résultats, pour afficher ses fournisseurs commerciaux sans recharger la page.

3.4.16 Alignement de séquences

Cet outil aligne deux séquences (ADN/ARN ou protéiques, détectées automatiquement) selon l'algorithme de Smith-Waterman, et affiche le résultat avec les brèches (*gaps*) nécessaires à l'alignement.

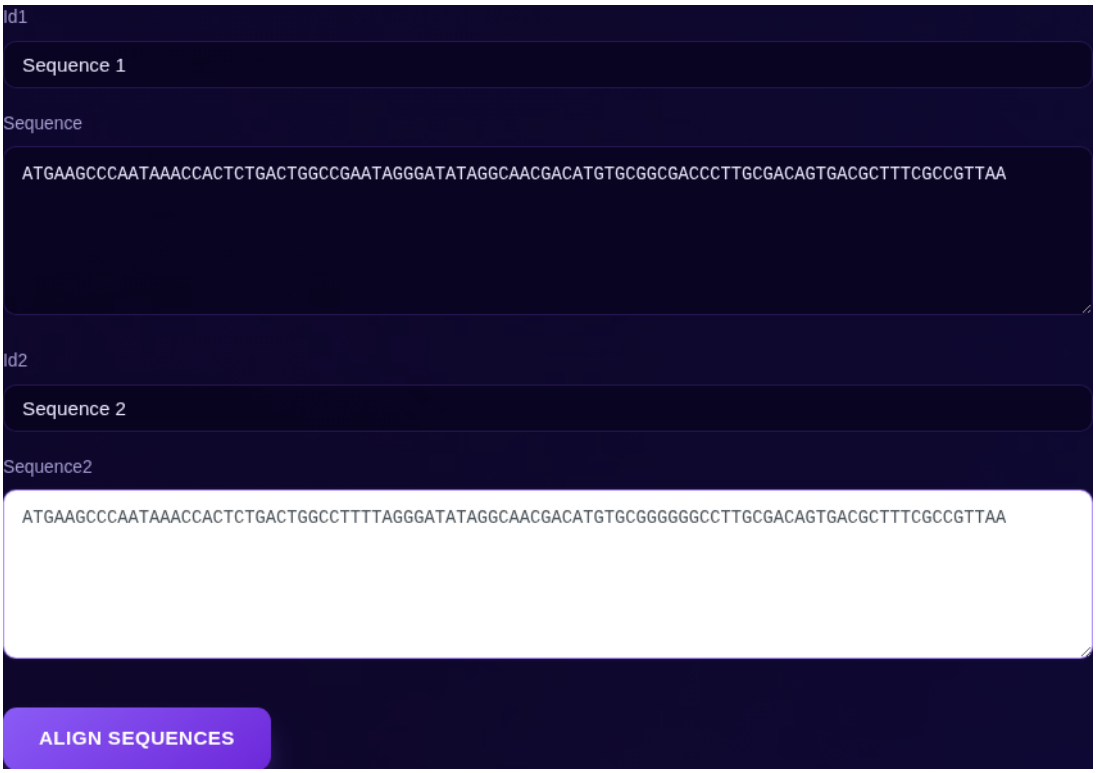

FIGURE 3.32 – Formulaire d'alignement de deux séquences.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`id1`	(nom de la première séquence) *Nom de la séquence 1*	texte	« Sequence 1 » par défaut.
`sequence`	(première séquence) *Séquence 1*	texte long	Nettoyée automatiquement (majuscules, U→T, X→N).
`id2`	(nom de la seconde séquence) *Nom de la séquence 2*	texte	« Sequence 2 » par défaut.
`sequence2`	(seconde séquence) *Séquence 2*	texte long	Même nettoyage que `sequence`.

Attention

La longueur cumulée des deux séquences est limitée (300 nucléotides dans la démo) : l'algorithme construit une matrice dont la taille croit avec le carré de la longueur des séquences, ce qui peut sinon dépasser la mémoire ou le temps d'exécution alloués à PHP.

FIGURE 3.33 – Séquences alignées, avec la ligne de comparaison (|) entre les positions identiques.

Implémentation Symfony

```php
public function seqAlignmentAction(Request $request,
    SequenceAlignmentManager $manager)
{
    $form = $this->createForm(SequenceAlignmentType::class);
    $sCompare = $sAlignSeqA = $sAlignSeqB = "";
    $id1 = $id2 = "";

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();
        $id1 = $formData["id1"];
        $id2 = $formData["id2"];

        // detection ADN/proteine : ADN si A+C+G+T >= la moitie de la
            sequence
        $countACGT = substr_count($formData["sequence"], "A")
            + substr_count($formData["sequence"], "C")
            + substr_count($formData["sequence"], "G")
            + substr_count($formData["sequence"], "T");

        if ($countACGT > (strlen($formData["sequence"]) / 2)) {
            $aAlignment = $manager->alignDNA($formData["sequence"],
                $formData["sequence2"]);
        } else {
            $aAlignment = $manager->alignProteins($formData["sequence"],
                $formData["sequence2"]);
        }

        $sAlignSeqA = $aAlignment["seqa"];
        $sAlignSeqB = $aAlignment["seqb"];
        $sCompare   = $manager->compareAlignment($sAlignSeqA,
            $sAlignSeqB);
    }

    return $this->render('minitools/seqAlignment.html.twig', [
        'form'       => $form->createView(),
        'compare'    => $sCompare,
        'align_seqa' => $sAlignSeqA,
```

```
        'align_seqb' => $sAlignSeqB,
        'sequence1'  => $id1,
        'sequence2'  => $id2,
    ]);
}
```

Implémentation vanilla

```
$manager = new SequenceAlignmentManager($pam250MatrixApi);

$result    = $manager->alignDNA($sequenceA, $sequenceB);
$compare   = $manager->compareAlignment($result["seqa"], $result["seqb"])
    ;
```

Note

Le filtre Twig `compare_alignment` présenté en §3.3 n'est qu'un habillage : il appelle directement `SequenceAlignmentManager::compareAlignment()`, la même méthode que celle utilisée ici côté contrôleur pour construire la ligne de comparaison avant le rendu.

3.4.17 Manipulation de séquences

Cet outil applique une opération simple à une séquence d'ADN/ARN (suppression des caractères non codants, séquence inversée, complément, complément inversé, double brin affiché, conversion en ARN), et peut en prime en calculer la composition nucléotidique et le taux de G+C.

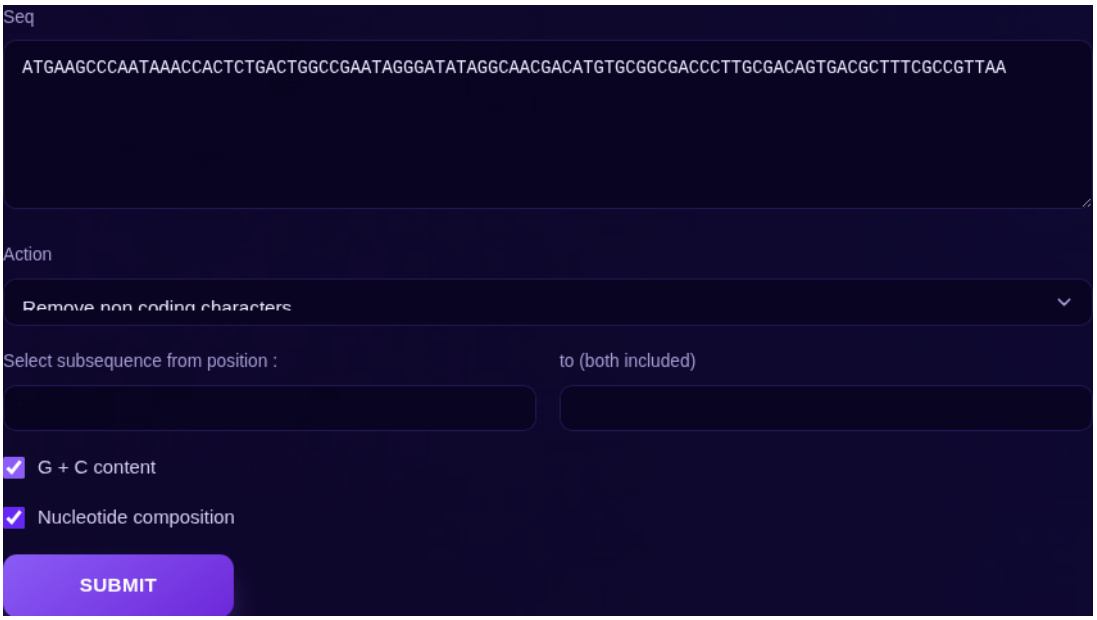

FIGURE 3.34 – Formulaire de manipulation de séquence.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`seq`	(séquence) *Séquence à traiter*	texte long	Nettoyée automatiquement à la soumission (majuscules, X→N, retrait des caractères non codants).
`action`	(liste des opérations) *Opération à appliquer*	liste à choix unique	`remove_non_coding` / `reverse` / `complement` / `reverse_and_complement` / `display_both_strands` / `toRNA`.
`start`	Select subsequence from position *Début de la sous-séquence*	texte	Position 1-indexée ; vide = début de la séquence.
`end`	to (both included) *Fin de la sous-séquence (incluse)*	texte	Vide = fin de la séquence.
`GC`	G + C content *Taux de G+C*	case à cocher	Ajoute le pourcentage de G+C au résultat.
`ACGT`	Nucleotide composition *Composition nucléotidique*	case à cocher	Ajoute le décompte de chaque base (y compris les bases dégénérées IUPAC présentes) au résultat.

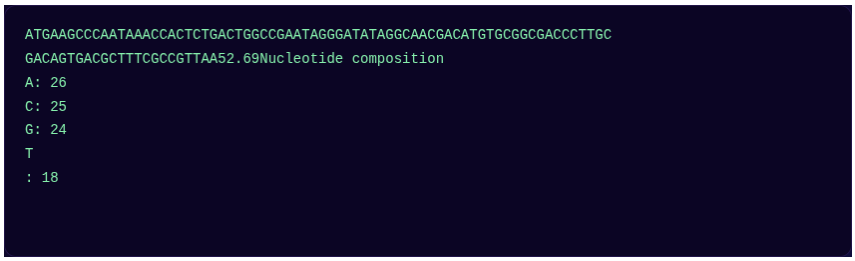

FIGURE 3.35 – Résultat de l'opération choisie.

Implémentation Symfony

```php
public function sequencesManipulationAndDataAction(Request $request,
    SequenceManipulationAndDataManager $manager)
{
    $result = "";
    $form = $this->createForm(SequenceManipulationType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->
        isValid()) {
        $formData = $form->getData();
        $seq = $formData["seq"];

        // sous-sequence eventuelle
        if (isset($formData["start"]) || isset($formData["end"])) {
            $start = $formData["start"] != "" ? ($formData["start"] - 1)
                : 0;
            $end   = $formData["end"] != "" ? $formData["end"] : strlen(
                $formData["seq"]);
```

```
            $seq   = substr($formData["seq"], $start, $end - $start);
        }

        switch ($formData["action"]) {
            case "reverse":
                $result = strrev($seq);
                break;
            case "complement":
                $result = $manager->compDNA($seq);
                break;
            case "reverse_and_complement":
                $result = $manager->compDNA(strrev($formData["seq"]));
                break;
            case "display_both_strands":
                $result = $manager->displayBothStrands($seq);
                break;
            case "toRNA":
                $result = $manager->toRNA($seq);
                break;
            default:
                $result = $seq; // remove_non_coding : le nettoyage a
                    deja eu lieu en PRE_SUBMIT
                break;
        }

        if ($formData["GC"]) {
            $result .= $manager->gcContent($seq);
        }
        if ($formData["ACGT"]) {
            $result .= $manager->acgtContent($seq);
        }
    }

    return $this->render('minitools/sequencesManipulationAndData.html.
        twig',
        ['form' => $form->createView(), 'result' => $result]);
}
```

Implémentation vanilla

```
$manager = new SequenceManipulationAndDataManager();

$reverseComplement = $manager->compDNA(strrev('ATGCGGATCC'));
$gc                = $manager->gcContent('ATGCGGATCC');
```

Note

Les opérations `reverse`, `complement` et `reverse_and_complement` ne sont pas des méthodes de `SequenceManipulationAndDataManager` : la première est un simple `strrev()` PHP, les deux autres passent par `compDNA()`, une méthode héritée du trait `SequenceTrait` de BioPHP (§2.7) et non propre à BioTools.

3.4.18 Générateur de *skews*

Cet outil parcourt une séquence à l'aide d'une fenêtre glissante et trace, pour chaque position, un ou plusieurs indicateurs de composition (GC-skew, AT-skew, KETO-skew, taux de G+C, oligo-skew) sous forme de graphique SVG.

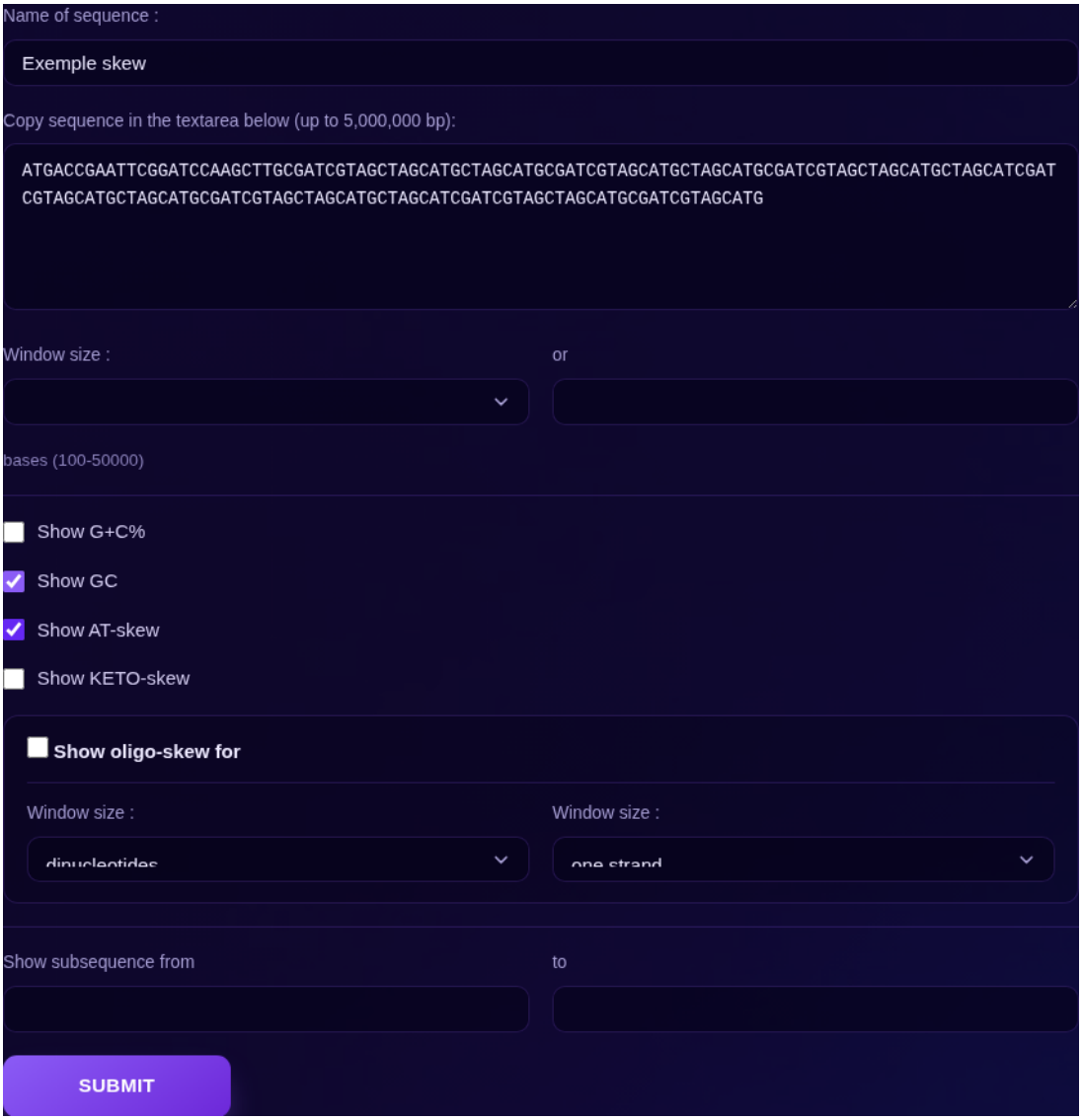

FIGURE 3.36 – Formulaire du générateur de skews.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`name`	Name of sequence *Nom de la séquence*	texte	Vide → « sequence » par défaut ; utilisé dans le titre du graphique.
`seq`	Copy sequence in the textarea below *Séquence à analyser*	texte long	Jusqu'à 10 000 000 caractères ; nettoyée automatiquement.
`window`	Window size *Taille de fenêtre (liste)*	liste déroulante	5 000 / 10 000 / 50 000 / 100 000.
`window2`	or *Taille de fenêtre (personnalisée)*	texte	100 à 100 000 ; prioritaire sur `window` si renseigné.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`GmC`	Show G+C% *Afficher le taux de G+C*	case à cocher	
`GC`	Show GC *Afficher le GC-skew*	case à cocher	
`AT`	Show AT-skew *Afficher le AT-skew*	case à cocher	
`KETO`	Show KETO-skew *Afficher le KETO-skew*	case à cocher	
`oskew`	Show oligo-skew for *Afficher l'oligo-skew*	case à cocher	Active le calcul, potentiellement long, de `oligo_len`/`strands`.
`oligo_len`	dinucleotides. . .hexanucleotides *Longueur des oligonucléotides*	liste déroulante	2 à 6.
`strands`	one strand / both strands *Un ou deux brins*	liste déroulante	Change la formule de distance utilisée (Almeida *et al.* si deux brins, Pearson standard sinon).
`from` / `to`	Show subsequence from / to *Sous-séquence à analyser*	texte	Optionnels ; restreignent l'analyse à un intervalle.

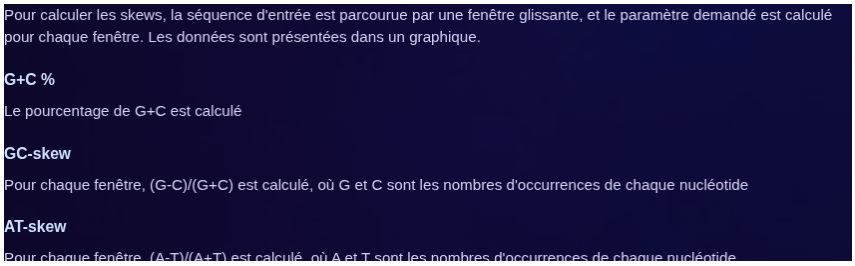

FIGURE 3.37 – Graphique SVG des skews demandés.

Implémentation Symfony

```php
public function skewsAction(Request $request, SkewsManager $manager)
{
    $imageResult = "";
    $form = $this->createForm(SkewsType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->isValid()) {
        $formData = $form->getData();
        $sequence = $formData["seq"];

        if ($formData["from"] || $formData["to"]) {
            if ($manager->strIsInt($formData["to"])) {
                $sequence = substr($formData["seq"], 0, $formData["to"]);
            }
            if ($manager->strIsInt($formData["from"])) {
                $sequence = substr($formData["seq"], $formData["from"]);
            }
```

```
        }

        $aOligoSkew = [];
        if ($formData["oskew"] && $manager->strIsInt($formData["
            oligo_len"])) {
            $aOligoSkew = $manager->oligoSkewArrayCalculation(
                $sequence, $formData["window"], $formData["oligo_len"],
                    $formData["strands"]
            );
        }

        $imageResult = basename($manager->createImage(
            $sequence, $formData["window"], $formData["GC"], $formData["
                AT"],
            $formData["KETO"], $formData["GmC"], $aOligoSkew,
            $formData["oligo_len"], $formData["from"], $formData["to"],
                $formData["name"]
        ));
    }

    return $this->render('minitools/skews.html.twig',
        ['form' => $form->createView(), 'image' => $imageResult]);
}
```

Implémentation vanilla

```
$manager = new SkewsManager($nucleotidsGraphsConfig, $oligosManager);

$image = $manager->createImage(
    $sequence, 5000, true, true, false, false,
    [], 2, '', '', 'ma_sequence'
);
// $image contient le chemin complet du fichier SVG ecrit sur disque
```

Attention

Comme pour la représentation en chaos et la distance entre séquences, `createImage()` écrit un fichier SVG dans le répertoire configuré par `path_graphs` (§3.2). Un dossier absent ou non accessible en écriture s'y traduit par une erreur serveur ; voir le détail du mécanisme d'écriture en §3.5.

Note

L'option `oskew` (oligo-skew) est la plus coûteuse en calcul : elle compare, fenêtre par fenêtre, les fréquences d'oligonucléotides de toute la séquence à celles de chaque fenêtre, selon la méthode d'Almeida *et al.* (2001) sur deux brins, ou une distance de Pearson classique sur un seul brin.

3.4.19 Formules utiles

Ce n'est pas un outil d'analyse de séquence à proprement parler, mais une boîte à formules de laboratoire courantes : poids moléculaire d'acides nucléiques, conversions pmol/µg, température, pression et centrifugation.

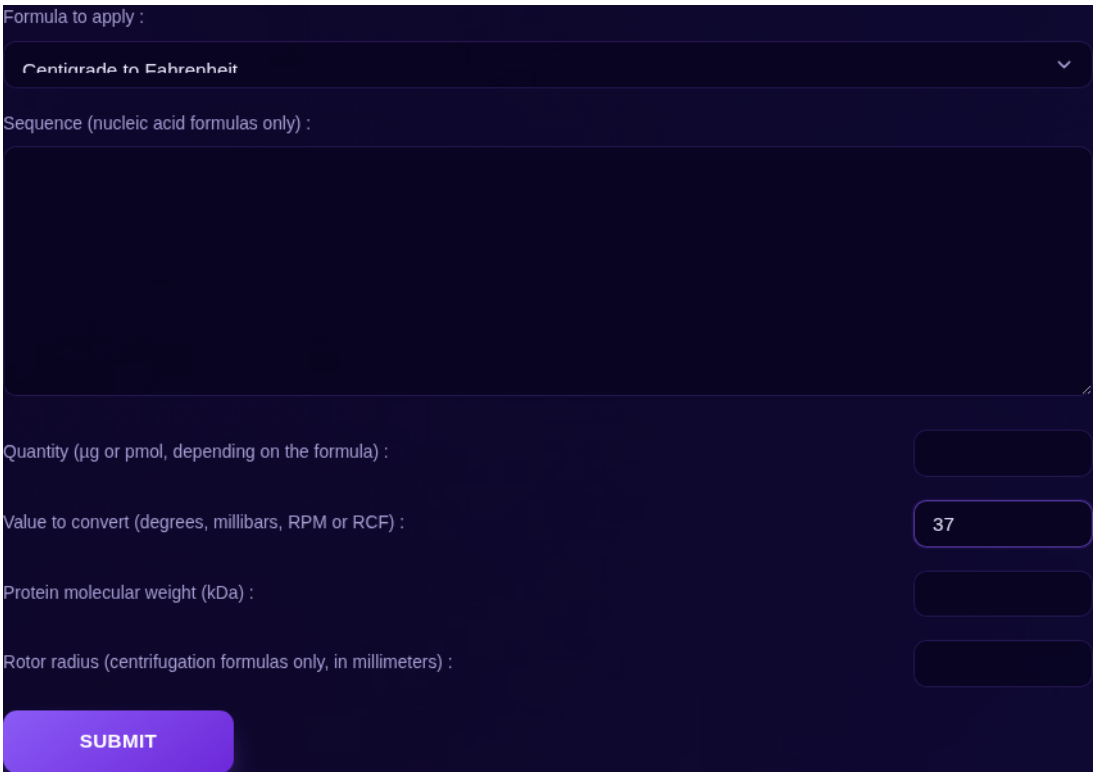

FIGURE 3.38 – Formulaire des formules utiles.

Champ	Libellé / traduction	Type	Valeurs, défaut, rôle
`formula`	Formula to apply *Formule à appliquer*	liste déroulante groupée	23 formules groupées en 5 catégories : ADN, ARN, températures/pression, protéines, centrifugation.
`sequence`	Sequence (nucleic acid formulas only) *Séquence (formules acide nucléique uniquement)*	texte long	Nettoyée automatiquement ; sa longueur sert de nombre de bases dans les formules ADN/ARN.
`quantity`	Quantity (µg or pmol, depending on the formula) *Quantité (µg ou pmol)*	texte	Utilisé par les formules de conversion pmol↔µg.
`value`	Value to convert (degrees, millibars, RPM or RCF) *Valeur à convertir*	texte	Utilisé par les formules de température, pression et centrifugation.
`molecularWeight`	Protein molecular weight (kDa) *Masse moléculaire protéique (kDa)*	texte	Utilisé par les formules protéiques.
`radius`	Rotor radius (centrifugation formulas only, in millimeters) *Rayon du rotor (mm)*	texte	Utilisé par les formules de centrifugation.

FIGURE 3.39 – Résultat de la formule choisie.

Implémentation Symfony

```php
public function usefulFormulasAction(Request $request, FormulasManager $manager)
{
    $result = $formula = null;
    $form = $this->createForm(FormulasType::class);

    if ($request->isMethod('POST') && $form->handleRequest($request)->isValid()) {
        $formData = $form->getData();
        $formula  = $formData["formula"];

        $result = match ($formula) {
            "mw_dsdna"       => $manager->mwOfDsDNA($formData["sequence"]),
            "centi_to_fahren" => $manager->centiToFahren($formData["value"]),
            "mbar_to_inchhg" => $manager->mbarToInchHg($formData["value"]),
            "rpm_to_rcf"     => $manager->rpmToRcf($formData["value"], $formData["radius"]),
            // ... 19 autres cas, un par entree de la liste deroulante
            default => null,
        };
    }

    return $this->render('minitools/usefulFormulas.html.twig',
        ['form' => $form->createView(), 'formula' => $formula, 'result' => $result]);
}
```

Implémentation vanilla

```php
$manager = new FormulasManager();

$fahrenheit = $manager->centiToFahren(37);        // 98.6
$inchHg     = $manager->mbarToInchHg(1013.25);    // pression atmospherique standard
```

Attention

Deux des 23 formules ont été corrigées par rapport au code de référence biophp.org, qui contenait une erreur mathématique :

— **Centigrade vers Fahrenheit** : le code de référence appliquait `32 + (C × 0.555)`, alors que 0.555 (≈5/9) est le facteur Fahrenheit→Centigrade utilisé dans le mauvais sens (100°C donnait 87.5°F au lieu de 212°F). BioTools applique `32 + (C × 1.8)`.

— **Millibars vers pouces de mercure** : le code de référence appliquait `mbar × 0.0394` (le facteur millimètre→pouce), surestimant le résultat d'environ 33 %. BioTools applique `mbar × 0.02953`.

Si un calcul fondé sur ces deux formules ne correspond pas à un résultat trouvé ailleurs sur le web, c'est très probablement cette correction qui en est la cause.

3.5 Les graphiques SVG

Trois outils du chapitre produisent un graphique : la représentation en chaos (§3.4.1, en CGR comme en FCGR), la distance entre séquences (§3.4.2, dendrogramme) et le générateur de skews (§3.4.18). Tous trois s'appuient sur la même brique, `Amelaye\BioTools\Service\Graphics\SvgCanvas`.

3.5.1 SvgCanvas

BioTools ne dépend pas de l'extension GD de PHP : `SvgCanvas` réécrit en SVG les quelques primitives de dessin dont les managers graphiques ont besoin, avec une correspondance directe vers les fonctions GD qu'elle remplace.

Méthode	Remplace (GD)	Rôle
`__construct($width, $height)`	`imagecreate()`	Crée une toile SVG de la taille donnée.
`rgb($r, $g, $b)` *statique*	`imagecolorallocate()`	Construit une couleur `#rrggbb` à partir de trois composantes 0–255.
`background($couleur)`	(allocation de la couleur de fond)	Remplit toute la toile.
`rect($x1, $y1, $x2, $y2, $couleur)`	`imagefilledrectangle()`	Rectangle plein ; les coordonnées peuvent être données dans n'importe quel ordre.
`line($x1, $y1, $x2, $y2, $couleur)`	`imageline()`	Trace un segment.
`pixel($x, $y, $couleur)`	`imagesetpixel()`	Un point isolé.
`pixelCloud($points, $couleur)`	(boucle de `imagesetpixel()`)	Regroupe une série de points en un seul tracé SVG au lieu d'un élément par point — c'est ce qui évite des fichiers démesurément gros pour les skews et le dendrogramme.
`text($police, $x, $y, $texte, $couleur)`	`imagestring()`	Écrit du texte ; les cinq tailles de police GD (1 à 5) sont approximées par des tailles et lignes de base SVG équivalentes.
`render()`	—	Retourne le document SVG complet sous forme de chaîne.
`save($chemin)`	`imagepng($im, $chemin)`	Écrit le SVG au chemin donné et retourne ce chemin.

Note

Le paramètre `$police` de `text()` n'est pas une vraie police : c'est l'entier 1 à 5 des cinq polices bitmap intégrées à GD (`imagestring($im, $police, ...)`), conservé pour que le code porté depuis GD n'ait pas à changer ses appels ; `SvgCanvas` traduit cet entier en une taille de police et un décalage de ligne de base SVG proches du rendu bitmap d'origine.

3.5.2 Où vont les fichiers

Les trois managers concernés (`ChaosGameRepresentationManager`, en mode CGR comme en mode FCGR, `DistanceAmongSequencesManager` et `SkewsManager`) appellent tous `SvgCanvas::save()` avec un chemin construit à partir du même paramètre de configuration, `amelaye_biotools.nucleotids_graphs.path_graphs` (§3.2). Le nom de fichier inclut un suffixe aléatoire (`bin2hex(random_bytes(4))`) pour que deux requêtes simultanées n'écrasent pas le même fichier.

Attention

Ce répertoire doit remplir deux conditions à la fois, faciles à manquer séparément :

— **accessible en écriture** par le processus PHP (l'erreur type, côté serveur, ressemble à `file_put_contents(): failed to open stream`) ;

— **servi publiquement** par le serveur web, puisque le contrôleur ne retourne que le *nom* du fichier (`basename(...)`) — c'est au template de construire l'URL publique vers ce même répertoire.

C'est exactement le problème rencontré et corrigé sur le site de démonstration pendant la production des captures de ce chapitre (§3.4.2, §3.4.1) : un répertoire non accessible en écriture se traduit côté client par une erreur 500 générique, sans autre indication.

3.6 Récapitulatif

Outil	Formulaire	Manager	Méthode principale
Chaos Game Representation (CGR/FCGR) §3.4.1	`ChaosGameRepresentationType`	`ChaosGameRepresentationManager`	`cgrCompute()` / `fcgrCompute()`
Distance entre séquences §3.4.2	`DistanceAmongSequencesType`	`DistanceAmongSequencesManager`	`upgmaClustering()`
Traduction ADN→protéine §3.4.3	`DnaToProteinType`	`DnaToProteinManager`	`translateDNAToProtein()`
Séquences palindromiques §3.4.4	`FindPalindromesType`	`FindPalindromeManager`	`findPalindromicSeqs()`
Taux de GC (upload FASTA) §3.4.5	`FastaUploaderType`	`FastaUploaderManager`	`createFiles()`
Température de fusion §3.4.6	`MeltingTemperatureType`	`MeltingTemperatureManager`	`basicCalculations()` / `neighborCalculations()`
Analyse de puces à ADN §3.4.7	`MicroArrayDataAnalysisType`	`MicroarrayAnalysisAdaptiveManager`	`processMicroarrayDataAdaptiveQuantificationMethod()`
Répétitions microsatellites §3.4.8	`MicrosatelliteRepeatsFinderType`	`MicrosatelliteRepeatsFinderManager`	`findMicrosatelliteRepeats()`
Fréquence des oligonucléotides §3.4.9	`OligoNucleotideFrequencyType`	`OligosManager` *(BioPHP)*	`findOligos()`
Amplification PCR §3.4.10	`PcrAmplificationType`	`PcrAmplificationManager`	`amplify()`
Propriétés protéiques §3.4.11	`ProteinPropertiesType`	`ProteinPropertiesManager`	`aminoacidContent()`, `proteinMolecularWeight()`…
Rétrotraduction protéine→ADN §3.4.12	`ProteinToDnaType`	`ProteinToDnaManager`	`translateProteinToDNA()`
Séquences aléatoires §3.4.13	`RandomSequencesType`	`RandomSequencesManager`	`createFromSeq()` / `createFromACGT()` / `createFromAA()`
Alphabet protéique réduit §3.4.14	`ReduceAlphabetType`	`ReduceProteinAlphabetManager`	`reduceAlphabet()` / `reduceAlphabetCustom()`

Outil	Formulaire	Manager	Méthode principale
Digestion par enzymes de restriction §3.4.15	`RestrictionEnzymeDigestType`	`RestrictionDigestManager`	`restrictionDigest()`
Alignement de séquences §3.4.16	`SequenceAlignmentType`	`SequenceAlignmentManager`	`alignDNA()` / `alignProteins()`
Manipulation de séquences §3.4.17	`SequenceManipulationType`	`SequenceManipulationAndDataManager`	`displayBothStrands()`, `gcContent()`...
Générateur de skews §3.4.18	`SkewsType`	`SkewsManager`	`oligoSkewArrayCalculation()`, `createImage()`
Formules utiles §3.4.19	`FormulasType`	`FormulasManager`	23 méthodes de conversion