

RAPPORT DE STAGE

Au sein de l'Institut de Biologie et Chimie des Protéines
(IBCP)



Institution des Chartreux
Valentin RYCKAERT, Aurélien ROSAT

Table des matières

Remerciements	1
Glossaire.....	2
Introduction	3
L'IBCP	3
Objectifs du stage	4
Notre travail	4
Architecture du système informatique	5
Les outils utilisés	5
Ubuntu	5
FastAPI	5
Svelte / SvelteKit.....	6
MkDocs	6
GitLab	6
MariaDB.....	6
Tchap.....	6
Le déroulement du stage	7
La conduite du projet	7
Le code de NPSA-NG	7
Le CLI pour communiquer avec l'API	9
Transfert de compétence	10
La documentation	10
La formation	11
Les tests de déploiement sur un serveur	11
Clonage du code et lancement en mode dev	11
Ajout de l'HTTPS	11
Mise en place d'un mode production	12
Les résultats	14
Conclusion.....	19

Remerciements

Nous souhaitons exprimer tous nos remerciements à Alexis Michon et Emmanuel Bettler, qui nous ont accueilli et guidé durant toute la durée de ce stage. Grâce au temps qu'ils nous ont consacré, nous avons pu délivrer un travail efficace dans d'excellentes conditions.

Nous tenons à remercier également notre professeur, Monsieur Aurelio Lourenco, qui nous a permis d'entrer en contact avec l'équipe informatique de l'IBCP.

Enfin, nous souhaitons exprimer toute notre gratitude à l'ensemble du site de l'IBCP pour son accueil chaleureux et qui a fait en sorte que notre stage soit une réussite.

Glossaire

NPS@ (NPSA) : Le site que nous devons refaire datant de 1998 et écrit en Perl.

NPSA-NG : NG pour New Generation, le site que nous avons codé pendant le stage.

Système de calcul : programme permettant d'analyser et effectuer des calculs sur des éléments chimiques (ici, des protéines)

Job : Un job est un script Bash contenant une commande exécutant un système de calcul (avec différents inputs / options).

Queue : Une queue est la liste des jobs à exécuter suivant leur niveau de priorité et leur temps d'exécution estimé.

Cluster de Calcul : Le Cluster est un ensemble de machines situées dans la salle serveur de l'IBCP, spécialisé dans les calculs de protéines. Il héberge les différents systèmes de calcul comme ClustalO, BlastP et Alphafold 3. De plus, il réparties les jobs en différentes queues afin d'optimisé la demande d'exécutions des demandes.

Alignement de Séquence : L'alignement de séquences de protéines est une méthode bio-informatique qui permet de comparer et d'aligner plusieurs séquences protéiques afin d'identifier des similitudes, des différences et des motifs conservés.

Webservice : Un Webservice est un système de calcul accessible depuis NPSA / NPSA-NG. Quand nous parlons d'implémenter / coder un webservice, il s'agit d'écrire le code permettant de prendre les inputs et paramètres, de construire la commande bash permettant d'exécuter le service de calcul et de l'envoyer au cluster.

Introduction

Dans le cadre de notre BTS, nous devions effectuer un stage de huit semaines en entreprise pour notre deuxième année. Tous les deux en spécialité développement (SLAM), nous cherchions une entreprise qui avait besoin d'un profil comme le nôtre.

Notre professeur, Monsieur Aurelio Lourenco, nous a alors proposé le contact d'Alexis Michon, qui avait besoin de deux développeurs pour une application web. L'objectif était de refaire de zéro une vieille application en Perl, l'application NPSA, qui met à disposition des systèmes de calcul pour des protéines. Celle-ci était en effet datait du XXe siècle (1998) et était difficilement maintenable.

De plus, nos maîtres de stage (l'un administrateur systèmes-réseaux, l'autre bio-informaticien), ne possède que peu de compétences en développement. Il était donc important pour nous d'assurer un transfert de compétence.

Durant ces huit semaines, nous avons pu être guidé par les compétences réseaux de Monsieur Michon et celles en Bio-informatique de Monsieur Bettler, qui nous ont été d'une grande aide pour mener à bien ce projet. L'application à la fin de notre stage n'est pas pleinement terminée, l'objectif étant de laisser à nos deux maîtres de stage une base solide et documentée pour qu'ils puissent se former et faire évoluer l'application sans difficulté.

(Pour des soucis de sécurité et de propriété intellectuelle, les bouts de code montrés ici ne sont que des exemples pour expliquer le fonctionnement. Aucun n'est extrait de l'application finale.)

L'IBCP

Nous avons eu l'opportunité d'effectuer notre stage au sein du Centre National de la Recherche Scientifique (CNRS), l'un des principaux organismes de recherche en France. Au sein de cette institution, nous étions affiliés à l'Institut de Biologie et Chimie des Protéines (IBCP), situé à Lyon. L'IBCP est spécialisé dans l'étude de la chimie des protéines et accueille de nombreuses équipes de recherche ainsi que des étudiants en doctorat.

Le site de l'IBCP héberge également deux autres laboratoires de recherche. Le premier, le Laboratoire de Biologie et Technologie de l'Informatique (LBTI), se concentre sur l'intégration de la biologie et de l'informatique. Il développe des outils et des méthodes pour analyser des données biologiques complexes, facilitant ainsi le traitement et l'interprétation des données liées aux protéines et aux systèmes biologiques.

Le second laboratoire, le Laboratoire de Modélisation et Simulation en Biologie (MMSB), se spécialise dans la modélisation et la simulation des systèmes biologiques. Il utilise des approches computationnelles pour étudier les interactions entre les molécules et les processus biologiques, en développant des modèles mathématiques et des simulations numériques.

Durant notre stage, nous avons été intégrés à l'équipe IBCP. Notre projet consistait à remettre au goût du jour l'application web NPS@ du PRABI (Pôle Rhône-Alpes de Bio-Informatique – structure mettant à disposition des outils informatiques aux bio-informaticiens de la région), accessible à l'adresse <https://prabi.ibcp.fr/>. Ce site est une ressource importante pour les chercheurs, leur permettant d'effectuer des calculs de protéines et d'accéder à divers outils bio-informatiques. Notre travail a donc visé à recréer l'application avec des outils plus modernes, afin d'améliorer l'accessibilité et l'efficacité de cette plateforme, facilitant ainsi le travail des scientifiques dans leurs recherches.

Objectifs du stage

Notre travail

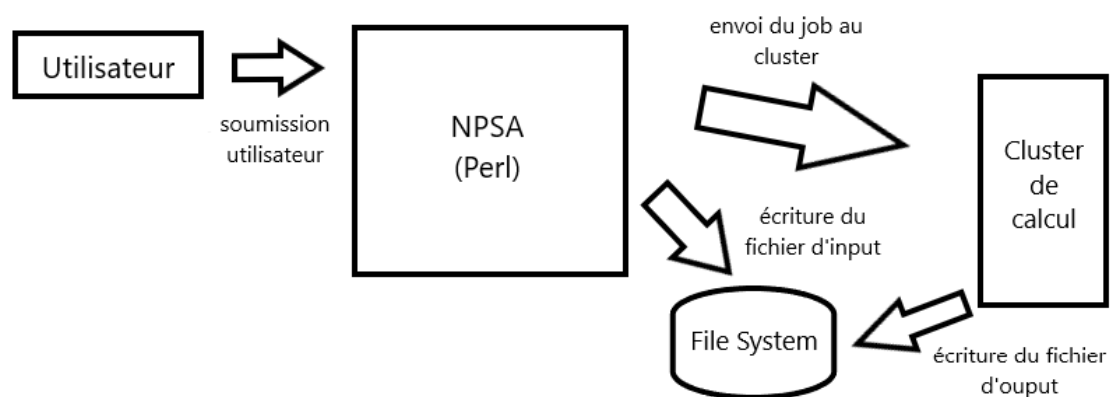
L'objectif de ce stage était double. D'une part, il s'agissait de créer une base solide pour l'application, respectant les contraintes de cybersécurité et offrant une ébauche fonctionnelle. D'autre part, il était essentiel de réaliser un travail de documentation et de formation afin que nos maîtres de stage, qui ne sont pas développeurs, puissent assurer la continuité du projet.

Ce projet, est en perpétuelle évolution. Nous n'avions pas d'objectif précis en termes de nombre de systèmes de calcul à implémenter, car le nombre n'est pas défini. L'application accueillera au fil du temps de plus en plus de services, chacun avec des spécificités rendant leur code unique. L'objectif était de moderniser cet outil pour faciliter la vie des chercheurs en leur offrant une solution simple d'utilisation, moderne et efficace.

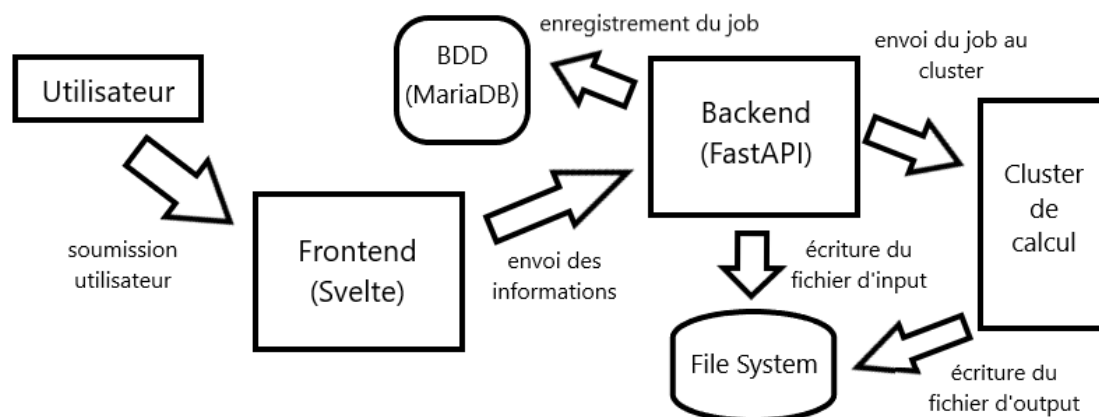
Le code devait être compréhensible, maintenable, sécurisé, documenté et testé, afin de faciliter le travail des futurs contributeurs. Il ne s'agissait pas seulement de rendre une solution fonctionnelle, mais de livrer une sorte de version "beta" solide.

Architecture du système informatique

Architecture actuelle de NPSA



Architecture de NPSA-NG à la fin de notre stage



Les outils utilisés

Ubuntu

Ubuntu est une distribution Linux basée sur Debian, développée par Canonical Ltd. Elle est conçue pour être facile d'utilisation, ce qui en fait l'une des distributions Linux les plus populaires. L'IBCP nous a fourni pour ce stage deux machines Ubuntu Desktop 22.04.5, sur lesquelles nous avons pu développer.

FastAPI

FastAPI est un Framework web léger et rapide pour construire des APIs avec Python. Il est conçu pour être simple à utiliser tout en offrant des performances élevées, grâce à son utilisation de Starlette pour la gestion des requêtes et Pydantic pour la validation des données. L'IBCP nous demandait en effet de construire un backend en python, et Il nous paraissait être un très bon choix pour obtenir un code lisible et maintenable.

Svelte / SvelteKit

Svelte est un Framework JavaScript frontend. Contrairement à d'autres Frameworks comme React ou Vue, Svelte compile les composants en code JavaScript pur au moment de la construction, ce qui permet d'obtenir des applications plus légères et plus rapides. Ici aussi, le choix de Svelte est surtout basé sur sa simplicité et la lisibilité du code final.

SvelteKit est une surcouche de Svelte qui permet de faire du code côté serveur. Il nous aura servi à gérer des requêtes GET entre différentes pages du frontend ainsi que les URL.

MkDocs

MkDocs est un générateur de documentation statique codé en python, conçu pour créer des sites web de documentation à partir de fichiers Markdown. Il est particulièrement apprécié pour sa simplicité d'utilisation et sa capacité à produire des documents bien structurés et esthétiques. Nous avons choisi cet outil, encore une fois, pour sa simplicité d'utilisation.

GitLab

GitLab est une plateforme de gestion de code source et de collaboration pour le développement logiciel, qui intègre des fonctionnalités de gestion de projet, d'intégration continue (CI), de déploiement continu (CD) et de suivi des problèmes. GitLab est basé sur Git, un système de contrôle de version décentralisé, et est disponible en tant que service cloud ou en version auto-hébergée. L'IBCP possède son propre serveur GitLab.

MariaDB

MariaDB est un système de gestion de base de données relationnelle open source, un fork de MySQL. Il a été créé par les développeurs originaux de MySQL après l'acquisition de MySQL par Oracle. MariaDB nous a été imposé par l'IBCP.

Tchap

Tchap est une plateforme de communication sécurisée et open source, développée par l'État français. Elle est basée sur le protocole Matrix, qui permet la messagerie instantanée, les appels audio et vidéo, ainsi que le partage de fichiers. Tchap a été conçu pour répondre aux besoins de communication des agents publics. L'IBCP, faisant partie du CNRS, utilise donc Tchap en tant qu'organisme public. Nous l'utilisons lorsque nos maîtres de stage étaient en déplacement ou en télétravail, notamment pour nos comptes rendus quotidiens.

Le déroulement du stage

La conduite du projet

Pendant ce stage, nous avons mis en place une méthodologie agile adaptée à notre projet, une version un peu ramassée de la méthodologie Scrum. En effet, tous les matins, notre maître de stage nous attendait dans son bureau pour revenir sur le code de la veille (sprint backlog) et nous donner les tâches à accomplir pour le prochain sprint qui se déroulait sur une journée ou deux, selon la quantité de tâches à réaliser. Nous étions davantage dans un format de Daily Scrum.

Chaque nouvelle fonctionnalité était envoyée sur GitLab afin de mettre le code à disposition de notre maître de stage, qui testait l'application et nous faisait part de ses retours. Chaque jour, en plus des commits, nous devions rédiger un compte rendu des tâches effectuées durant la journée, afin que notre maître de stage sache quelles tâches restaient en suspens, lesquelles étaient terminées et celles qui devaient être vérifiées, le tout via Tchap. Cet outil nous servait également pour nos échanges quotidiens et pour résoudre les problèmes techniques que nous pouvions rencontrer.

Il nous arrivait parfois de planifier des réunions pour nous recentrer sur les objectifs et/ou former notre maître de stage aux outils que nous utilisions, comme GitLab.

Le code de NPSA-NG

Durant l'ensemble du stage, notre objectif aura été d'écrire du code fonctionnel, mais aussi lisible. Nos maîtres de stage n'ayant que peu de compétences dans le code, nous devions nous assurer qu'ils comprennent comment fonctionne l'application, comment elle est organisée et comment l'information passe de l'input de l'utilisateur au cluster de calcul.

FastAPI

L'Application NPSA-NG est composée d'une API python, codée avec le Framework FastAPI. Celle-ci est divisée en plusieurs « routeurs », qui regroupe un ensemble d'endpoints. Ces routeurs sont ensuite tous rattachés à l'API principale. Nous avons pour cette API :

- Un routeur par webservice (/run, /getResult)
- Un routeur pour la gestion des jobs (/getStatus, /updateStatus...)
- Un routeur pour des données particulières (liste des BDD de protéines, statistiques...)

L'API est reliée à une base de données MariaDB, qui enregistre les jobs soumis (utile principalement pour les statistiques).

```

from fastapi import FastAPI
import subprocess
from database import SessionDep, Job
from uuid import uuid4

app = FastAPI()

@app.get("/run")
def run_job(input: str, session: SessionDep):

    if not is_fasta(input):
        raise HTTPException(status_code=422, detail="input not fasta")

    with open("./input.fasta", "w") as f:
        f.write(input)

    new_job = Job(uuid4())
    session.add(new_job)
    session.commit()
    session.refresh(new_job)
    subprocess.run(['sh', 'clustalo -inputfile ./input.fasta'])
    return {"uuid": new_job.uuid}

```

Svelte

Svelte est un Framework « File-based routing », ce qui veut dire que les URL suivent l'arborescence de dossiers. Parce que les webservices ont beaucoup de différence entre eux (inputs / paramètres différents, outputs différents), nous avons choisi d'avoir deux composants Svelte par service :

- Un composant pour l'affichage de l'input (à l'URL /webservices/[nom_du_service]). Tous ces composants ont pour base le

composant principal `Webservice.svelte`, qui décrit l'architecture html / css dans laquelle on intègre les éléments d'inputs.

```
<script>
  import Webservice from "../Webservice.svelte"
</script>

<Webservice webservice_name="test">
  {#snippet inputContent()}
    <input type="text">
    <input type="mail">
  {/snippet}
  {#snippet OptionContent()}
    <input type="text">
    <input type="checkbox">
  {/snippet}
</Webservice>
```

- Un composant pour traiter l'output (`/job/[job_uuid]/result`). Ce composant est situé dans le même dossier que le composant d'input, mais est utilisé dans une page spécifique pour afficher les résultats de job (`/jobs/[job_uuid]/result`).

```
<script>
  import Service_Result from "../../Service_Result.svelte"
  import getResult from "../../api.ts"
  import { page } from '$app/stores'

  let service_name = $page.url.searchParams.get('webservice_name')
  job_result = await getResult()
</script>

<div>
  {#if service_name === 'test'}
    <Service_Result job_result={job_result} />
  {/if}
</div>
```

Le CLI pour communiquer avec l'API

Bien que cela n'ait pas été le centre de notre stage, on nous a demandé de construire un CLI en python permettant de communiquer avec l'API. Celui-ci sera utilisé par les bio-informaticiens de l'IBCP qui ont besoin d'envoyer de nombreux jobs de manière programmée. Ce CLI ne sera accessible qu'au sein de l'IBCP.

Pour réaliser ce projet, nous avons utilisé le Framework Typer, créé par le même développeur que FastAPI et ayant une structure de code quasi identique. Il est simple d'utilisation et son code est lisible.

Ici aussi, pour des soucis d'organisation de code et de lisibilité, nous avons un CLI par webservice.

```
from typer import Typer
import requests

app = Typer()

def runJob(inputPath: str, guidetree_out: bool = False):
    with open(inputPath, 'r') as f:
        input = f.read()
    requests.post("http://localhost:8000", {
        'sequence': input,
        'guidetree-out': guidetree_out
    })

if __name__ == "__main__":
    app()

# utilisation : python main.py runJob --guidetree-out ./input.fasta
```

Transfert de compétence

La documentation

La fin de notre stage s'est essentiellement concentrée sur la documentation de l'application (backend et frontend). Pour cela, nous avons mis en place un MkDocs qui présentait chaque composant du frontend, l'architecture des deux codes et des tutoriels pour rajouter un service web :

- Un tutoriel « ajouter un service web » (backend et frontend)
- Des notes sur chaque composants Svelte
- Une note de conseils pour bien coder
- Une partie « référence » pour les liens vers les différentes documentations (Svelte, Vite, SvelteKit, FastAPI, unitTests...)

A noter que Monsieur Michon a une expérience en Python, ce qui fait que la documentation s'est concentrée majoritairement sur la partie Svelte et Vite.

La formation

Dans le cadre du transfert de compétence, nous devons expliquer à Monsieur Michon le fonctionnement de Git ainsi que les principales commandes. Pour cela, nous avons organisé une matinée de formation :

- Système de dépôt (code, dépôt local, dépôt remote)
- Arborescence de Git (branche dev, branche personnelle...)
- Commandes principales (add, commit, push, pull, switch, merge)

Les tests de déploiement sur un serveur

Clonage du code et lancement en mode dev

Bien que l'application ne soit pas terminée, nous avons pu durant la seconde moitié de notre stage commencer des tests de déploiement. Pour cela, Monsieur Michon nous a mis à disposition une machine Debian 12, sur laquelle nous avons pu cloner le code. Une fois les dépendances installées, nous pouvions lancer les deux applications en mode dev, accessibles depuis le réseau ('fastapi run' et 'npm run dev -- --host').

Ajout de l'HTTPS

Pour l'ajout du HTTPS, L'IBCP possède des certificats signés que nous avons pu utiliser. Nous avons dû modifier le code des deux applications :

FastAPI : utilisation d'une commande uvicorn, sur lequel FastAPI est basé.

```
if __name__ == '__main__':
    uvicorn.run(
        app,
        host="0.0.0.0",
        port=8432,
        ssl_keyfile="./localhost+4-key.pem",
        ssl_certfile="./localhost+4.pem"
    )
```

Svelte : modification de la configuration Vite

```
import fs from 'fs';

server: {
  https: {
    key: fs.readFileSync('./app/ssl/server.key'),
    cert: fs.readFileSync('./app/ssl/server.crt')
  }
}
```

Mise en place d'un mode production

Pour les essais de mise en production, l'objectif était de pouvoir lancer l'API python en mode daemon (avec systemctl), et que le code svelte soit 'build' en JavaScript natif, comme en production.

FastAPI

Ajout d'une configuration gunicorn :

```
from multiprocessing import cpu_count

# Socket Path
bind = 'unix:/home/demo/fastapi_demo/gunicorn.sock'

# Worker Options
workers = cpu_count() + 1
worker_class = 'uvicorn.workers.UvicornWorker'

# Logging Options
loglevel = 'debug'
accesslog = '/home/demo/fastapi_demo/access_log'
errorlog = '/home/demo/fastapi_demo/error_log'
```

Ajout d'une configuration dans '/etc/systemd/system/fastapi_demo.service' :

```
[Unit]
Description=Gunicorn Daemon for FastAPI Demo Application
After=network.target

[Service]
User=demo
Group=www-data
WorkingDirectory=/home/demo/fastapi_demo
ExecStart=/home/demo/fastapi_demo/venv/bin/gunicorn -c gunicorn_conf.py app:app

[Install]
WantedBy=multi-user.target
```

On lance ensuite le daemon avec les commandes suivantes :

```
sudo systemctl start fastapi_demo
```

```
sudo systemctl enable fastapi_demo
```

Svelte

Comme nous utilisons SvelteKit, la transformation de l'application en site statique ne fonctionnait pas : une partie de l'application constituait en un code à exécuter côté serveur. Nous avons donc dû choisir un 'adaptateur', c'est-à-dire une configuration vite permettant de transformer l'application en un code JavaScript comportant une partie serveur. Nous avons pris celui proposé par Svelte, qui s'exécute avec un serveur NodeJS :

```
import adapter from '@sveltejs/adapter-node';

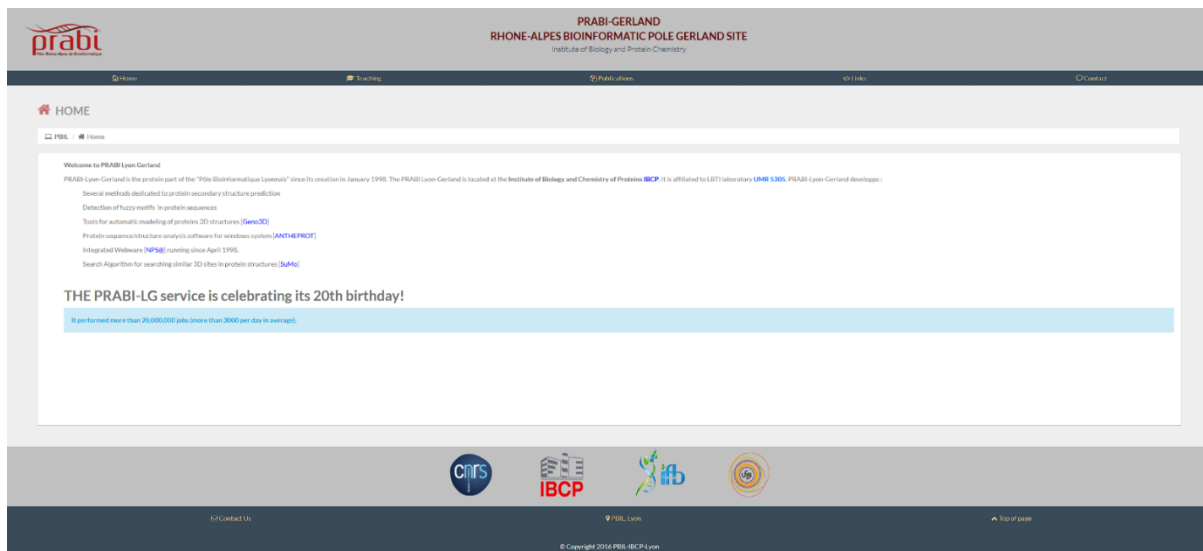
export default {
  kit: {
    adapter: adapter()
  }
};
```

Ce code est ajouté au fichier svelte.config.js.

Une fois la commande 'npm run build', on lance le frontend à l'aide de la commande 'node index.js' dans le dossier dist (celui où se situe le JS natif).

Les résultats

Le site que nous devons refaire de zéro datant de 1998 écrit en Perl (NPSA) :




Capture d'écran ci-dessous représentant le catalogue de Webservice proposé par NPSA :

- **Sequence homology search against proteic databases :**
 - [BLAST search](#) (protein (blastp) or nucleic (blastx) query sequence)
 - [PSI-BLAST search](#) (protein query sequence)
 - [FASTA search](#) (protein query sequence)
 - [SSEARCH search](#) (protein query sequence)
 - [HMMSEARCH](#) (protein query profile, hmmer format)
- **Sequence homology search against nucleic databases :**
 - [BLAST search](#) (protein (tblastn) or nucleic (blastn, tblastx) query sequence)
- **Patterns and signatures search :**
 - [PATTINPROT](#): scan a protein sequence or a protein database for one or several pattern(s)
 - [PROSCAN](#): scan a sequence for sites/signatures against PROSITE database
- **Profile building :**
 - [HMMBUILD](#): build a profile with HMMER (HMMER profile format)
- **Multiple alignment:**
 - [Clustal W Protein sequences](#) (Des Higgins, EBI, Hinxton Hall, UK)
 - [Clustal W DNA sequences](#) (Des Higgins, EBI, Hinxton Hall, UK)
 - [Multalin Protein sequences](#) (F.Corpet, INRA Toulouse, France)
 - [Multalin DNA sequences](#) (F.Corpet, INRA Toulouse, France)

Le catalogue de webservice de NPSA-NG à la fin de notre stage :

NPSA-NG


[Home](#) [Webservices](#) [My Jobs](#) [Stats](#) [Contact](#) [About](#)



IA

...


[AlphaFold 2](#) [AlphaFold 3](#)



Multiple Sequence Alignment

Identify conserved sequence patterns from multiple related sequences.


[ClustalO](#) [ClustalW](#)



Sequence Similarity Search

Find sequences in databases based on similarity.


[Blastp](#) [PsiBlast](#) [Fasta](#)



Secondary Structure Prediction

...





[SOPM](#) [SOPMA](#) [Gor IV](#)



Others

...

[Amphipaseek](#)



© 2025 Company, Inc

Voici un Webservice, c'est un formulaire à compléter avec des options et un Input. Cet Input est en fait une ou plusieurs séquences de protéine au format Fasta.

CLUSTALW

[\[Abstract\]](#) [\[NPS@ help\]](#) [\[Original server\]](#)

Paste a protein sequence databank in Pearson/Fasta format below : [help](#)

All sequence names must be different !

Output width :

CLUSTALW Parameters

Output format :

Output oder :

Pairwise alignment type :

Fast pairwise alignment parameters	Slow pairwise alignment parameters
K-tuple (word) size : <input type="text" value="1"/>	Protein weight matrix : <input type="text" value="GONNET"/>
Number of top diagonals : <input type="text" value="5"/>	Gap openig penalty : <input type="text" value="10.0"/>
Window size : <input type="text" value="5"/>	Gap extension penalty : <input type="text" value="0.1"/>
Gap penalty : <input type="text" value="3"/>	
Scoring method : <input type="text" value="Percentage"/>	

Multiple Alignment Parameters :

Weight matrix :

Gap opening penalty :

Gap extension penalty :

Residue-specific gap penalties OFF : ☐

Hydrophilic gaps OFF : ☐

Hydrophilic residues :

Percent of identity for delay :

Gap separation distance :

No end gap separation penalty : ☒

User : public Last modification time : Mon Mar 15 15:24:33 2021. Current time : Sun May 4 10:21:34 2025

Et voici un Webservice de NPSA-NG :

NPSA-NG Home Webservices My Jobs Stats Contact About

Clustalo

OR

Clear

Example

Parcourir...

Aucun fichier sélectionné.

submit

Every field with a * is required


Options

output-order

outfmt

dealign ☐

guidetree-out ☐



© 2025 Company, Inc

Plusieurs fonctionnalités ont été rajoutées notamment au niveau de la Cybersécurité, contrôle des inputs de séquence et autres failles XSS, l'input via un fichier pour faciliter la vie des chercheurs ainsi qu'un UX/UI design retravaillé.

Clustal est un ensemble de systèmes de calcul pour de l'alignement de séquence. Voici le résultat d'un job ClustalW sur NPSA :

CLUSTALW multiple alignment

[Abstract](#) Thompson, J.D., Higgins, D.G. and Gibson, T.J. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment

Residues repertoire

alignment with width:
 all residues.
 Conservation level of % (works with "using conservation level of" option in list above, negative value will hide).
 All prediction methods and secondary consensus ☐ with percentage of secondary structure.

☐ DSC (King and Sternberg, 1996) [\[HELP\]](#)
☐ DPM (Deleage and Roux, 1987) [\[HELP\]](#)
☐ GOR I (Garnier *et al.*, 1978) [Choose parameters](#) [\[HELP\]](#)
☐ GOR III (Gibrat *et al.*, 1987) [\[HELP\]](#)
☐ GOR IV (Garnier *et al.*, 1996) [\[HELP\]](#)
☐ HNN (Guermeur, 1997) [\[HELP\]](#)
☐ SIMPA96 (Levin *et al.*, 1996) [\[HELP\]](#)
☐ PHD (Rost *et al.*, 1994) [\[HELP\]](#)
☐ PREDATOR (Argos *et al.*, 1996) [Choose parameters](#) [\[HELP\]](#)
☐ SOPM (Geourjon and Deleage, 1994) [Choose parameters](#) [\[HELP\]](#)

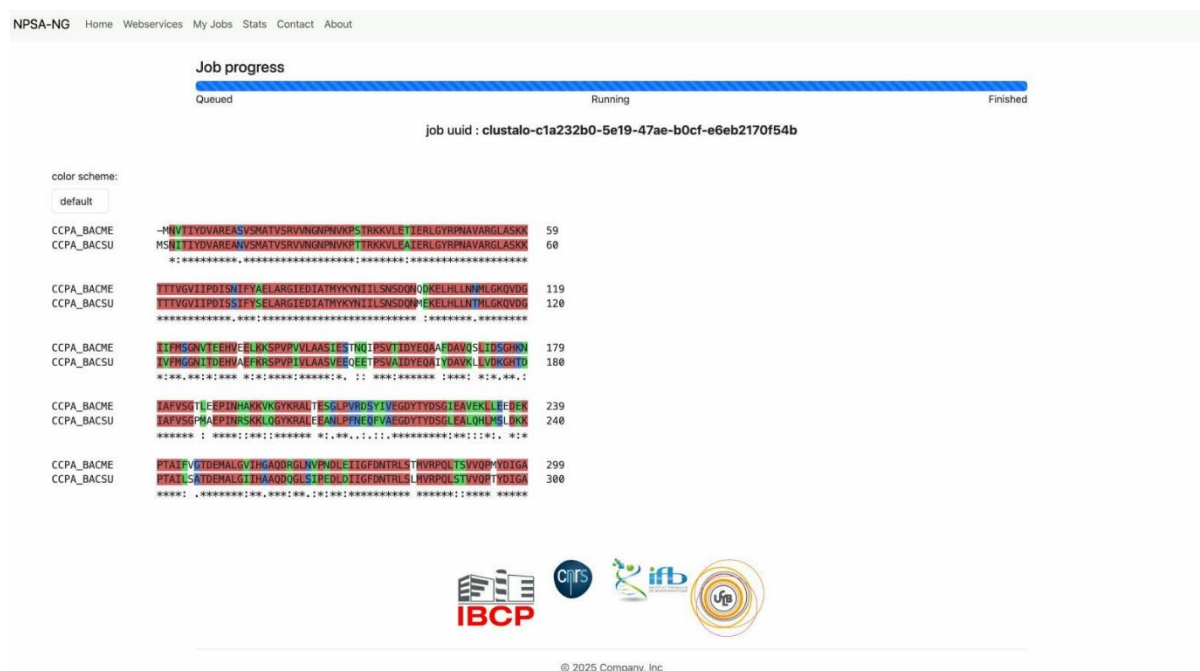
View CLUSTALW in: [\[AnTheProt \(PC\)\]](#), [\[Download...\]](#) [\[HELP\]](#)

```

      10      20      30      40      50      60
      |      |      |      |      |      |
CCPA_BACME  ---MNVTIYDVAREASVSMATVSRVNGNP---NVKPSSTRKKVLETIERLGYRPNNAVAR
CCPA_BACSU  ---MSNIITIIDVAREANVSMATVSRVNGNP---NVKPTTRKKVLEAIERLGYRPNNAVAR
CCPA_STRMU  MNTDDTITIIDVAREAGVSMATVSRVNGNK---NVKENTRKKVLEVIDRLDYRPNNAVAR
DEGA_BACSU  ---MKTITIDVAKAAGVSITTVSRVINNTG---RISDKTRQKVMNMVNMAYTPNVHAA
FRUR_ECOLI  -----MKLDEIARLAGVSRRTTASYVINGKAKQYRVSDKTVEKVMVAVREHNYHPNAVAA
RBTR_KLEAE  ---MKKITIIDLAELSGVSAVASAILNGNWKRRISAKLAEKVTRIAEEQGYAINRQAS
              .: .:*. .:*. .:*. .:*. .: .: .:*. .: .:*. .: .:*.
Prim.cons.  MNTMM2ITIIDVAREAGVSMATVSRVNGNPK222V22KTRKKVLEVIE2LGYPNNAVAR

      70      80      90     100     110     120
      |      |      |      |      |      |
CCPA_BACME  GLASKKTTTGVVGIIPDISNIFYAELARGIEDIATMYKYNILSNSDQNDKELHLLNNML
CCPA_BACSU  GLASKKTTTGVVGIIPDISNIFYSELARGIEDIATMYKYNILSNSDQNMELHLLNTML
CCPA_STRMU  GLASKKTTTGVVGIIPNIANAYFSILAKGIDDIAATMYKYNIVLASSDEDDKEVNVINTLF
DEGA_BACSU  ALTGKRTNMIALVAPDISNPFYGEAKSIEERADELGFQMLICSTDYDPKKEKTYFSVLK
FRUR_ECOLI  GLRAGRTRSIGLVIIPDLENTSYTRIANYLERQARQGYQLLIACSEDQPDNEMRCIEHLL
RBTR_KLEAE  MLRSKKSHVIGMIIPKYDNRYFGSIAERFEEMARERGLLPITCTRRRPLEIEAVKAML
              * .: .: .:*. .: .: .:*. .: .: .: .: .: .: .: .: .:
Prim.cons.  GLASKKTTT2GV2IPDISNIFY2ELA2GIEDIATMY2YNI223SDQ2PDKELHL2NT2L
  
```

Pour NPSA-NG de nouvelles choses étaient ajoutées notamment une barre de progression qui tient au courant l'utilisateur de la situation du Job qui l'a envoyé avant l'affichage de l'output. Plusieurs schémas de couleurs ont aussi été ajoutés pour montrer plusieurs facettes de l'alignement. Voici un job ClustalO (plus récent que ClustalW) sur NPSA-NG :



Conclusion

Notre stage au CNRS IBCP a été une expérience extrêmement bénéfique qui nous a permis de mettre en pratique les compétences acquises durant notre BTS SIO.

En travaillant sur ce projet, nous avons non seulement amélioré nos compétences en développement, mais également approfondi notre compréhension des API et de leur intégration dans des applications complexes. Notre mission de transfert de compétences nous a également beaucoup apporté, en nous forçant à structurer notre pensée et à délivrer une explication claire et simple.

Cette immersion dans un environnement professionnel nous a offert un aperçu précieux du fonctionnement d'un institut de recherche, renforçant notre motivation pour la suite de nos études en école d'ingénieur. Nous sommes convaincus que les connaissances et les compétences que nous avons développées durant cette période seront très utiles pour notre parcours académique et professionnel futur.