

# INFORME TRABAJO PRÁCTICO FINAL

---

Lucas Avalos, María Belén Amat, Marcos Alvarenga, Fabrizio Britez

Para la realización del trabajo práctico realizamos una aplicación Web. Ambos backend y frontend se encuentran en el repositorio <https://github.com/leavalos/bioinformatica>

## Implementación del backend

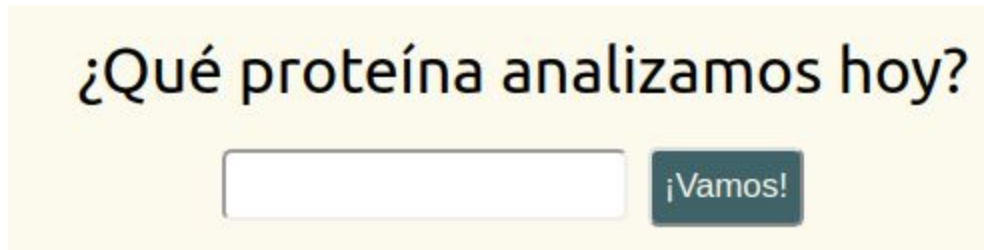
Para la implementación del backend decidimos usar Python junto con Flask, un framework para crear aplicaciones web. Nos conectamos con Blast, Clustal y DSSP de manera local ya que las consultas que enviábamos a las APIs a veces tardaban mucho tiempo y otras, nunca volvían.

La API que creamos cuenta con 2 endpoints principales que son los que consume nuestro frontend:

- **POST** /sequence  
Que contiene un campo **pdbservice** dentro del body. Retorna la lista de secuencias que matchean con el código pdb dado.
- **POST** /analyze  
Que recibe una secuencia y devuelve el análisis de haber alineado las secuencias homólogas a la secuencia query. Este análisis incluye estructuras primarias y secundarias.

## Implementación del frontend

Para la implementación del frontend utilizamos JavaScript con React. El usuario deberá ingresar el código PDB de la proteína que desea analizar.



¿Qué proteína analizamos hoy?

En caso de que el usuario ingrese un PDB vacío o un PDB inexistente se le informará que ese no es un código PDB válido.

Para conseguir todas las secuencias asociadas a ese código PDB, utilizaremos el endpoint del backend **/sequences** mencionado anteriormente. Como un código PDB puede tener múltiples secuencias, se le pedirá al usuario que seleccione cual de todas las posibles secuencias es la que desea analizar.



¿Qué proteína analizamos hoy?

Hace click en la secuencia para analizarla con estos parametros.

[2B0T\\_1 - NADP Isocitrate dehydrogenase - Corynebacterium glutamicum \(1718\)](#)

Chain A

MAKIWTRTDEAPLLATYSLKPVVEAFAATAGIEVETRDISLAGRILAQFPERLTEDQKVGNAELGELAKTPEANIILPNISASVPQLKAAIKELQDQGYDIPELPDN  
ATTDEEKDILARYNAVKGSAVNPVLRREGNSDRRAPIAVKNFVKKFPHRMGEWSADSKTNVATMDANDFRHNEKSIILDADEVQIKHIAADGTETILKDSLKLEGEV  
LDGTVLSAKALDAFLEQVARAKAEGILFSAHLKATMMKVSDPIIFGHVVRAYFADVFAQYGEQLLAAGLNGENGLAAILSGLESLDNGEEIKAAFEKGLDGPDLAM  
VNSARGITNLHVPSDVIVDASMPAMIRTSGHMWNKDDQEQLTLAIIPDSSYAGVYQTVIEDCRKNGAFDPTTMGTVPNVGLMAQKAEYGSHTKFRIEADGVVQ  
VVSSNGDVLIEHDVEANDIWRACQVKDAPIQDWVKLAVTRSRSLGMPAVFWLDPERAHDRNLASLVEKYLADHDTEGLDIQILSPVEATQLSIDRIRRGEDTISVTG  
NVLRDYNTDLFPILELGTSAKMLSVVPLMAGGGLFETGAGGSAPKHVQVQVEENHLRWDSLGEFLALAESFRHELNNNGNTKAGVLADALDKATEKLLNEEKSP  
SRKVGIEDNRGSHFWLTKFWADELAAQTEDADLAATFAPVAEALNTGAADIDAALLAVQGGATDLGGYYSPNEEKLTNIMRPVAQFNEVDALKK

Parametros Blast

E-value:  Coverage:

Gap open:  Gap extend:

matrix:

Una vez seleccionada la secuencia a analizar, utilizaremos el endpoint **/analyze** para conseguir la información de las estructuras primarias y secundarias de las secuencias homólogas a la secuencia query.

El usuario podrá ingresar un E-value y un coverage. Por el momento

utilizaremos un gap open de 11, un gap extend de 1 y la matriz BLOSUM62 como datos fijos no parametrizables.

## ¿Qué proteína analizamos hoy?

¡Vamos!

Ver alineamiento de secuencias

Ver estructura secundaria

Ver estructura terciaria

El usuario cuenta con 3 botones para poder ver la información obtenida: estructura primaria, secundaria y terciaria.

Para la visualización de la estructura primaria utilizamos una dependencia llamada **react-msa-viewer**, que nos permite ver el alineamiento de las estructuras primarias de las secuencias homólogas a nuestra secuencia query.

	147	149	151	153	155	157	159	161	163	165	167	169	171												
2NRL	D	F	D	A	V	L	K	C	W	G	P	V	E	A	D	-	Y	T	T	I	G	G	L	V	L
3QM5	D	F	D	A	V	L	K	C	W	G	P	V	E	A	D	-	Y	T	T	I	G	G	L	V	L
2NRM	D	F	D	A	V	L	K	X	W	G	P	V	E	A	D	-	Y	T	T	I	G	G	L	V	L

Luego nos dimos cuenta de que esta herramienta también nos servía para mostrar las estructuras secundarias.

Para poder visualizar la estructura terciaria utilizamos una dependencia llamada **ngl** que nos muestra la vista 3D de la estructura.

