EN.605.652.81.FA24
Megan Nguyen

Course Project Report

I. Introduction

Residue-residue interactions can be inferred from residue-residue contacts (inter-residue contacts within the same protein). Protein interactions may involve different structural elements, such as side chains, alpha-carbons (Cα), or beta-carbons (Cβ). Determining alpha-carbon and beta-carbon interactions provides insight into the structural organization of proteins and how residues are positioned to form stable interactions. In contrast, interactions via side chains help define the specific functional roles of the protein. A residue is considered in contact with another if the pair of residues are spatially close within the protein. Contacts are defined using a distance threshold based on the 3D structure of the protein, expressed by the x, y, and z coordinates of the amino acids' atoms found in a PDB or mmCIF file, with the Euclidean distance used for calculation. A distance cutoff between 5-8 Å is often strict enough to identify contacts between the alpha-carbon and beta-carbon atoms of residues [1]. Many papers will generally specify a cutoff between 5-8 Å. A protein contact map can be derived, which is a binary matrix where each cell (i, j) indicates whether residue i is in contact with residue j. Essentially, it represents the distance between all possible residue pairs of a 3D protein structure.

The objective of this system is to store information about the residues of a protein and predict residue-residue interactions by calculating residue contacts based on spatial distance. Specifically, the contacts are determined by using the alpha-carbon atoms and beta-carbon atoms of different residues (inter-residue contacts). Users should be able to access data regarding a protein's residues and its predicted residue-residue interactions by querying the database with the UniProt ID or the PDB ID. The user will be able to access data on the general residue proximity based on the alpha-carbon and beta-carbon atoms of a residue. Users will also be displayed general information about the protein they query.

Because users can enter either a UniProt ID or a PDB ID, the theme I will focus on is the reliability of data resources, specifically data provenance. This is important because not only could the mapping of PDB IDs to UniProt IDs lead to conflicting information, but there are also multiple data resources involved. To address this, I have included some data elements in my database such as the date the information was created and the source version used to generate the data. This information helps track the data history and adds to its reliability, as knowing the AlphaFold version, for example, provides insight into how the data was created.

II. Related Work

I plan to expand this project in the future to focus on protein-protein interactions or protein structure prediction. Residue interactions provide important insights into protein structure and functionality, ultimately helping researchers understand the causes of disease and developing new drugs, such as identifying drug targets. In protein-protein interactions (PPI), residue interactions can also help identify potential binding sites on protein surfaces, offering further insights into protein structures and drug design. While the role of charged residues is not yet

fully understood, their conservation patterns have been linked to protein-protein interactions and are often found at protein interaction interfaces [2]. Initially, I intended to include information about residue charges; however, due to time constraints, I decided to include this in a later update.

Residue contacts that are spatially close together within the 3D structure of a protein have been found to play a significant role in protein folding. Although the concept of predicting residue-residue contact maps for protein structure prediction has been around since the early 2000s, accurate predictions have only become possible recently due to advancements in machine learning [4]. Various machine learning algorithms have since been developed to predict protein residue contacts and protein contact maps, with many implementing deep learning approaches. Some of these methods, using neural networks such as convolutional neural networks (CNNs) and graph neural networks (GNNs), have resulted in high accuracy by incorporating chemicophysical and evolutionary information [5]. I focus on a subset of the problem, if you will, which is first predicting inter-residue interactions within the same the protein, and on a particular type of atoms.

III. Requirements

A few requirements are necessary for the system to function:

1. Develop a GUI that has a search box
    - The user will enter a valid UniProt ID or PDB ID and be able to retrieve predicted inter-residue interactions within the same protein from searching a protein of their choice
    - Accept case-insensitive inputs
    - Deploy tool on production server
2. Determine residue contacts
    - The user will be able to obtain calculated residue contacts from the Cartesian atomic coordinates of Cα and Cβ atoms
    - Error checks are set up in the case structural data is not available
3. Data storage
    - A database must be implemented to store the data queried by users
    - The user will need to be able to directly access the database
    - The contact map images generated need to be housed somewhere (due to the time constraint, I opted to just store the images in Django's static or media directory)

At a high-level, the tool works as intended. However, there are a few requirements that are not met discussed in later sections.

IV. Logical Database Design

This database is designed to support the storage of protein sequence data, residue-level information, protein-structure mapping, residue-residue interactions and metadata about data sources. It will be used for storing large-scale protein data and will need to efficiently handle queries related to protein structures, residue interactions and related metadata. I consider this

database design to be in 3rd Normal Form (3NF), because of the normalization techniques implemented to reduce data duplication and improve data integrity. All non-key attributes are functionally dependent on the primary key and no transitive dependencies exist (i.e., all attributes depend on the key and not on other non-key attributes).

1. **Normalization and Data Integrity**
   Normalization:
   - The schema follows normalization principles to reduce redundancy and optimize data organization. For example, the Protein_Identifier table stores unique mappings between UniProt IDs and PDB IDs, preventing duplication, while the Residue_Interactions table records interactions between residues only once for each unique pair."

   Data Integrity:
   - Foreign key constraints maintain proper relationships between tables, such as the Residue table's foreign key reference to the Protein table, ensuring that only valid proteins can have associated residues. Additionally, uniqueness constraints, like the unique uniprot_id in the Protein_Identifier table, prevent data duplication. Finally, NOT NULL constraints ensure that essential columns, such as uniprot_id and pdb_id, always contain valid data.

2. **Efficient Data Retrieval**
   Note: While I am able to perform these queries myself (see Section VIII. Implementation), I was unable to include the feature for users to query the database directly in the web interface due to time constraints.
   Indexing:
   - Indexes should be created on commonly queried fields, such as uniprot_id, pdb_id, and protein_id, to improve query performance. This will make searching for protein identifiers, residue information, and interactions much faster, especially when the database grows in size.

   Optimized Joins:
   - The design ensures that data can be retrieved through efficient joins across related tables. Retrieving all residues and their interactions for a specific protein can be done through a join between the Protein, Residue, and Residue_Interactions tables and information about the source of data for a protein can be linked through a join between Protein and Source tables.

   Optimized Queries:
   - Database views streamline complex queries, such as retrieving complete interaction data for a specific protein.

3. **Scalability and Extensibility**
   Handling Large Datasets:
   - The database is designed to handle large volumes of data, such as protein sequences and structural data. The Residue_Interactions table, which could

contain a large number of records due to the pairwise nature of residue interactions, is normalized to ensure that no duplicate interactions are stored.

Futureproofing:

- As new structural data becomes available, new tables or fields can be added without major changes, allowing integration of additional residue interaction types or metadata.

4. **Historical Tracking**

   Data Provenance:

   - The Source table tracks data provenance by storing metadata such as the source name, version, and creation date. This enables historical tracking and ensures the origins of datasets can be accurately traced.

5. **Key Relationships and ER Diagram**

   Protein-Residue Relationships:

   - A protein is composed of residues and the database structure is designed to reflect this relationship by linking residues to specific proteins via protein_id. This enables easy retrieval of all residues for a given protein.

   Residue-Interaction Relationships:

   - Residue interactions are stored once per unique pair, linked by residue positions and names.

   Protein-Structure Relationships:

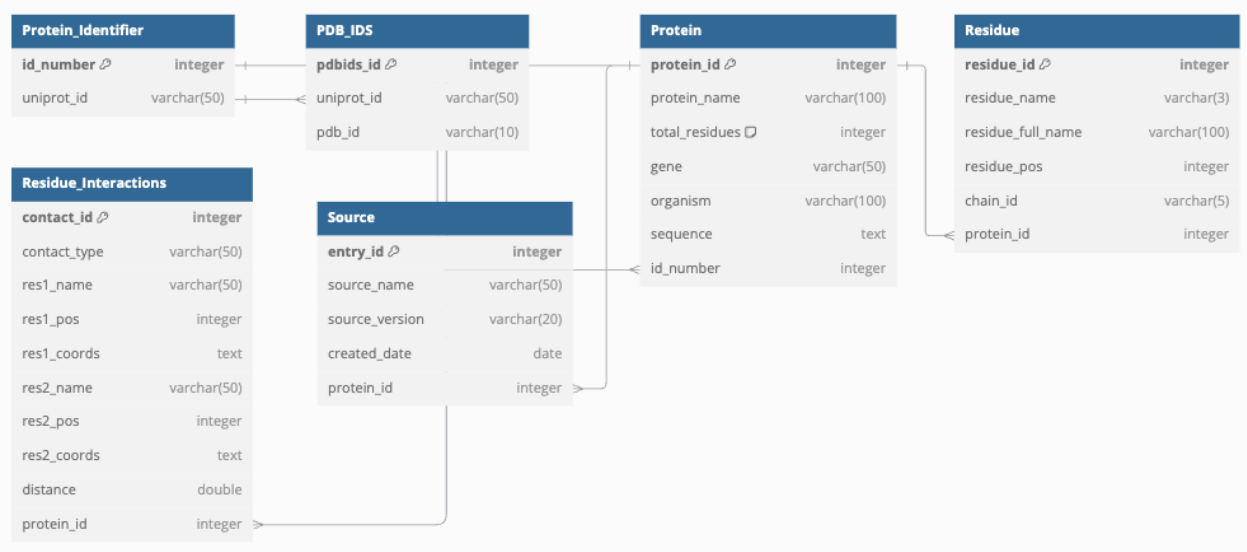
   - Proteins are mapped to PDB entries via uniprot_id, connecting biological sequences to structural data. These relationships help to tie together the biological sequence data with 3D structural data, which is essential for protein modeling and drug discovery.

Below is my Entity-Relationship Diagram (ER Diagram) and the relationship types briefly summarized. There are five entities:
Protein_Identifier, Protein, Residue, Residue_Interactions, Source

Relationship Types:

1. One-to-One:
   - Protein_Identifier ↔ Protein (via id_number)
2. One-to-Many:
   - Protein_Identifier ↔ PDB_IDS (via uniprot_id)
   - Protein ↔ Residue (via protein_id)
   - Protein ↔ Residue_Interactions (via protein_id)
   - Protein ↔ Source (via protein_id)
3. Many-to-Many:
   - Residue ↔ Residue_Interactions (via interactions between residues)

EN.605.652.81.FA24
Megan Nguyen



Generated from https://dbdiagram.io/d

V. Database Type

The database is a relational database (RDB) implemented using PostgreSQL and Python. PostgreSQL is an ideal choice for this project due to its scalability, robust support for complex queries, and ability to handle large datasets efficiently. Additionally, PostgreSQL's native support for JSON data types allows for seamless integration of unstructured data for future expansion. Its transactional table migrations further ensure data integrity, making it well-suited for a highly structured database environment.

PostgreSQL's indexing capabilities optimize queries for large datasets, making it particularly useful for protein data. Finally, PostgreSQL benefits from a large, active community, offering continuous updates and reliable support. Given that my data consists of well-defined, interconnected entities such as Protein, Residue, Residue_Interaction, Source, and Protein_Identifier, a relational database was a solid choice for this project. These entities have distinct relationships, including one-to-one, one-to-many, and many-to-many associations. Relational databases are highly effective at modeling structured, interconnected data, making them ideal for managing protein and residue information.

VI. Physical/Application Design

Since I implemented a relational database, it was easy for me to convert my ER diagram into tables. The schema is as follows:

Protein_Identifier: Stores mappings of proteins to their UniProt ID

Column	Description
id_number	Primary key - unique ID for this row
uniprot_id	UniProt ID associated with the protein

EN.605.652.81.FA24
Megan Nguyen

PDB_IDS: Stores mappings of proteins to their UniProt ID and PDB ID

Column	Description
pdbids_id	Primary key - unique ID for this row
uniprot_id	Foreign key to Protein_Identifier table (uniprot_id)
pdb_id	PDB ID associated with the protein

Protein: Stores general information about the protein

Column	Description
protein_id	Primary key - unique ID for this row
protein_name	Protein name
total_residues	Total number of residues in the protein
gene	Gene associated with the protein
organism	Organism name
sequence	Sequence of protein
id_number	Foreign key to Protein_Identifier table (id_number)

Residue: Stores general information about individual residues within a protein

Column	Description
residue_id	Primary key - unique ID for this row
residue_name	Residue name (the 3-letter amino acid code)
residue_full_name	The full name of the amino acid
residue_pos	Position of residue in sequence
chain_id	Chain identifier in the protein
protein_id	Foreign key to Protein_Identifier table (protein_id)

Residue_Interaction: Stores information about interactions between residues within a protein

Column	Description
contact_id	Primary key - unique ID for this row
contact_type	Cα or Cβ contact
res1_name	Name of first interacting residue
res1_pos	Position of residue 1
res1_coords	Coordinates of residue 1
res2_name	Name of second interacting residue
res2_pos	Position of residue 2
res2_coords	Coordinates of residue 2
distance	Distance between the two residues (Å)
protein_id	Foreign key to Protein table (protein_id)

Source: Stores information about the data source to track the history of the data

Column	Description
entry_id	Primary key - unique ID for this row
source_name	Name of the data source

source_version	Version of the source or data entry
created_date	Date the entry was created
protein_id	Foreign key to Protein table (protein_id)

Django's models.py was used to define the data models (tables) for the database backend supported by Django.

VII. Test Plan

The first test was to verify that my GUI displayed correctly. Homework 4 served as my initial trial in using Django to present information on the web locally. This included ensuring that the GUI provides an actual search field for the user and that results are properly retrieved and displayed.

To confirm that my system was working, I tested out the tool myself and entered various UniProt IDs and PDB IDs. These IDs were found via the UniProt and RCSB PDB databases. For UniProt IDs without a PDB ID, I ran an advanced query to exclude results missing an associated PDB ID. Similarly, I ran an advanced query on RCSB PDB, adding a filter for "Accession Code(s) – UniProt" set to null. to There are a few error checks for the user input.

UniProt ID	**PDB ID**	**Test**
O75626	3DAL	1:1 match
P13773	-	No PDB ID
-	4IB4	Multiple UniProt IDs
O12305		No entry in AlphaFold (no protein structure available)
-	100d	No UniProt ID

1. UniProt and PDB ID format
   - Users can enter case-insensitive inputs. For example, if they enter the PDB ID "3dal," the search will still return results for "3DAL." The code also checks for the correct ID format: UniProt IDs consist of 6 or 10 alphanumeric characters, while PDB IDs consist of 4 alphanumeric characters. This involves a basic character check for length and alphanumeric format.
2. PDB to UniProt ID mapping
   - Users have the option to enter a PDB ID instead of a UniProt ID. However, the limitation is that, if a PDB ID is entered, there could be multiple UniProt IDs associated with that PDB ID. A single PDB ID can have many UniProt IDs due to isoforms, multimeric complexes, proteins from different species, or sequence variations.
   - A function that maps the PDB ID to a UniProt ID was created to overcome this issue. If the PDB ID is not available in the mapping, the user sees an error message indicating that the search could not be performed. This is necessary because the information that is retrieved from the API requires a UniProt ID.

3. Available protein data
    - This test is simple: if no structural data is available for the protein, the user will not receive any residue results. This is a common limitation, as most known proteins have experimental data available, though some do not. There may be entries available on UniProt but not in AlphaFold, from which I am retrieving the mmCIF file.
4. Database (mydb)
    - As mentioned in the database design, several techniques were implemented to ensure data integrity and avoid redundant data (e.g., unique constraints, foreign keys). This test involved querying the same PDB ID or UniProt ID and verifying in the backend database that no duplication occurred.

Of course, some common defensive programming tests were implemented for failed API calls.

VIII. Implementation

The implementation for this tool was relatively straight forward and many resources across the web have detail a similar approach in predicting residue-residue interactions. The atomic coordinates are obtained from the mmCIF file of a queried protein, and contacts are calculated using these coordinates. The choice of the distance threshold varies across studies, and we cannot assume that commonly used distance thresholds in structural biology—such as 4 or 5 Å for defining interactions between two amino acids—are universally valid. According to the Critical Assessment of Techniques for Protein Structure Prediction (CASP) competition, a pair of residues is considered a contact if the distance between their atoms is less than or equal to 8 Å for beta-carbon (Cβ) atoms, and 7 Å for alpha-carbon (Cα) [1]. Nonetheless, many studies still consider a cutoff of 5 Å strict enough to define a contact, including studies on Protein Structure Networks (PSNs). Since proteins can adopt various conformations and interact internally due to the changing states of side chains, researchers use PSNs to study these changes. PSNs represent protein structures as networks, with specific points (often the centers of mass of side chains) serving as nodes. Researchers found that a cutoff of 5 Å provided stable results, meaning the networks remained consistent across different proteins and simulation force fields [7]. For this assignment, I hard-coded the threshold as 5 Å to simplify implementation and ensure the distance is close enough to determine contacts.

Generally, inter-residue contacts within a single protein structure are expected to occur within the same chain, as these contacts represent the spatial relationships or interactions between residues in that chain. Therefore, only the chain ID is displayed on the results page, and results for inter- or intra-chain contacts are not shown.

The major challenge with using Django was its ORM capabilities (Object-Relational Mapper). While Django's ORM is powerful and simple to implement, it can become pretty restrictive for complex queries or advanced database optimizations. For the purpose of this project, I used Django, which presented a bit of a learning curve. Scaling Django applications across multiple servers can be a bit of a hassle, but the biggest hurdle was deployment. I am not, by any means, a developer, so this took some time to figure out. Ultimately, I had to rely on deployment tools

like Docker to deploy my application, unlike simpler frameworks. However, I benefited from utilizing Django's atomic transactions, which are useful for multi-step processes involving the database. They also help roll back any changes if an error occurs during execution.

Currently, the major limitation of this tool is the inability for users to directly access the backend database. In other words, users cannot query the database directly—they can only query through a higher-level interface (e.g., API calls, predefined views). (This will be elaborated on in Section IX. Conclusion and Future Work.

Thus, the system can be summarized as such:

- **Search:**
  - Users can search for specific data (e.g., protein information, residue interactions)
- **API Calls:**
  - The system fetches relevant data from external sources using APIs
- **Calculations:**
  - The system performs necessary processing (e.g., calculating distances, interactions, etc.) based on the retrieved data
- **Display Results:**
  - The processed data is then displayed on the web interface, in a user-friendly format

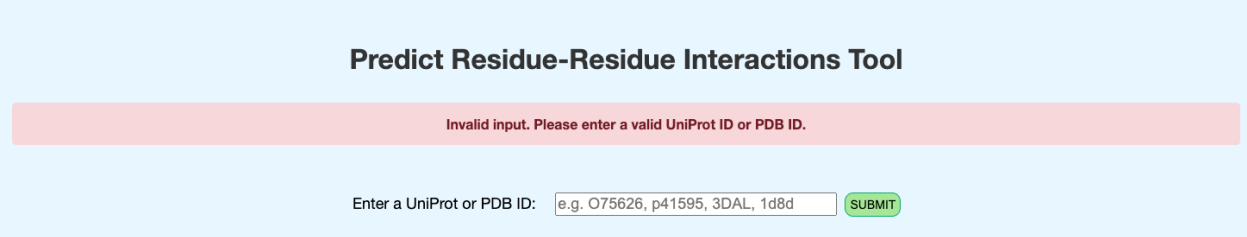
Below are the APIs I used for this project:
General protein information is obtained from the UniProt API.
https://www.uniprot.org/help/api_retrieve_entries
The protein structure was obtained from the AlphaFold API (mmCIF/PDB file).
https://alphafold.ebi.ac.uk/api-docs
The PDB ID to UniProt ID mapping was obtained from the PDBe REST API.
https://www.ebi.ac.uk/pdbe/api/

Because multiple APIs are involved, it was crucial to address this theme in the project. However, while I was able to gather some information regarding the sources of UniProt and AlphaFold, it was difficult to find this information for PDBe (where I made an API call for the PDB to UniProt ID mappings). It simply was not available.

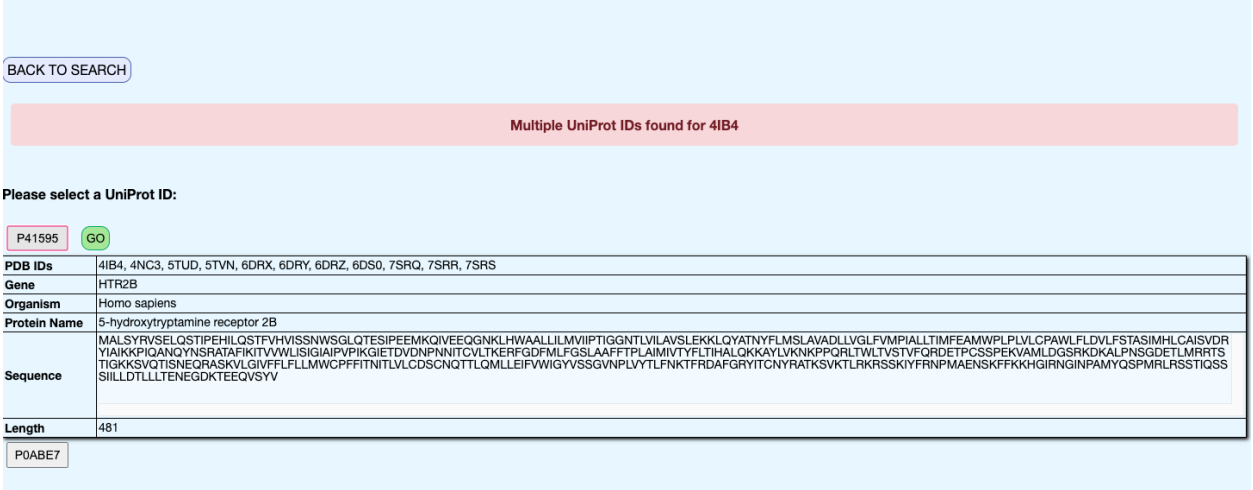
Below are error checks I discussed in Section VII: Test Plan. I used the IDs I listed in the table.

Test Screenshots (system: Apple M2 Pro, macOS: Sequoia 15.0):
Invalid input: “123”

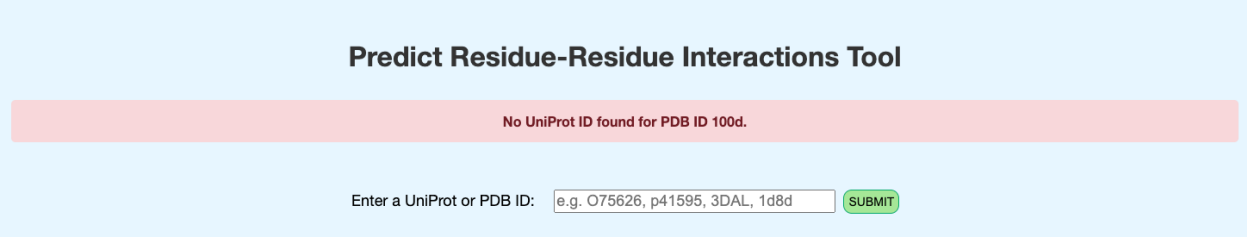
EN.605.652.81.FA24
Megan Nguyen



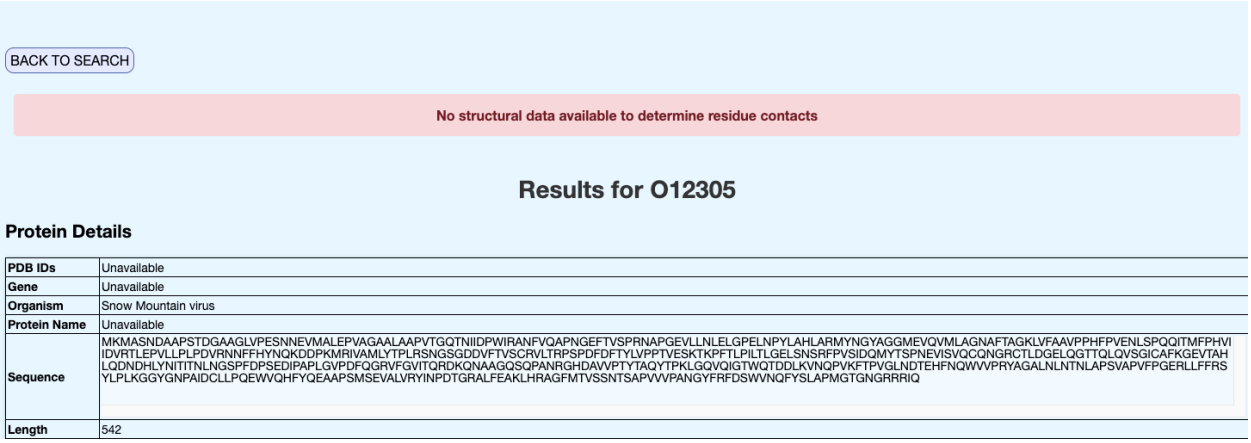
Multiple UniProt IDs associated with a PDB ID: “4IB4”



No UniProt ID associated with PDB ID: “100d”



No structural data available from AlphaFold: “O12305”



Typical results page would like this: “3DAL”

EN.605.652.81.FA24
Megan Nguyen

A successful search typically includes general information about the protein, residue interaction details (including both alpha-carbon and beta-carbon residue contacts), and the associated contact map.

BACK TO SEARCH

Results for O75626

Protein Details

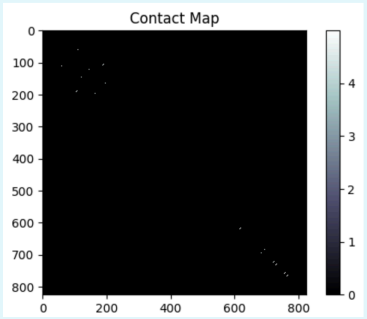
PDB IDs	3DAL
Gene	PRDM1
Organism	Homo sapiens
Protein Name	PR domain zinc finger protein 1
Sequence	MLDICLEKRVGTTLAAPKCNSSTVRFQGLAEGTKGTMKMDMEDADMTLWTEAEFEEKCTYIVNDHPWDSGADGGTSVQAEASLPRNLLFKYATNSEEVIGVMSKEYIPKGTRFGPLIGEIYTNDTVPKNANRKYFWRIYSRGELHHF IDGFNEEKSNWMRYVNPAHSPREQNLAACQNGMNIYFYTIKPIPANQELLVWYCRDFAERLHYPYPGELTMMNLTQTQSSLKQPSTEKNELCPKNVPKREYSVKEILKLDSNPSKGKDLYRSNISPLTSEKDLDDFRRRGSPEMPFY PRVVYPIRAPLPEDFLKASLAYGIERPTYITRSPIPSSTTPSPSARSSPDQSLKSSSPHSSPGNTVSPVGPGSQEHRDSYAYLNASYGTEGLGSYPGYAPLPHLPPAFIPSYNAHYPKFLLPPYGMNCNGLSAVSSMNGINNFGLFPRL CPVYSNLLGGGSLPHPMLNPTSLPSSLPSDGARRLLQPEHPREVLVPAPHSAFSFTGAAASMKDKACSPTSGSPTAGTAATAEHVVQPKATSAAMAAPSSDEAMNLIKNKRNMTGYKTLPYPLKKQNGKIKYECNVCAKTFGQLS NLKVHLRVHSGERPFKCQTCNKGFTQLAHLQKHYLVHTGEKPHECQVCHKRFSSTSNLKTHLRLHSGEKPYQCKVCPAKFTQFVHLKLHKRLHTRERPHKCSQCHKNYIHLCSLKVHLKGNCAAAPAPGLPLEDLTRINEEIEKFDI SDNADRLEDVEDDISVISVVEKEILAVVRKEKEETGLKVSLQRNMGNGLLSSGCSLYESSDLPLMKLPPSNPLPLVPVKVKQETVEPMDP
Length	825

Residue Interaction Details

Chain ID: A

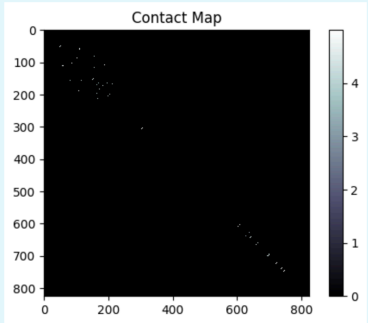
Alpha-carbon (Cα) Residue Contacts

Residue 1	Position	Coordinates	Residue 2	Position	Coordinates	Distance (Å)
LEU	14	[8.031 -20.702 34.239]	ALA	15	[10.571 -19.195 36.74]	3.87
PRO	17	[15.099 -17.666 41.14]	LYS	18	[14.88 -16.753 45.434]	4.4
LYS	18	[14.88 -16.753 45.434]	CYS	19	[14.132 -14.755 48.692]	3.89
CYS	19	[14.132 -14.755 48.692]	ASN	20	[11.591 -14.526 52.076]	4.24
LEU	29	[-18.271 -7.338 54.992]	ALA	30	[-20.309 -8.278 51.924]	3.8
ALA	30	[-20.309 -8.278 51.924]	GLU	31	[-22.346 -5.765 48.71]	4.56
MET	46	[-11.246 -9.688 15.791]	ILE	187	[-11.28 -10.978 11.863]	4.13
THR	47	[-13.861 -5.142 13.158]	LEU	48	[-12.219 -1.821 16.403]	4.92
TRP	49	[-8.189 -6.188 17.792]	GLU	53	[-5.712 -6.423 21.012]	4.07
THR	50	[-4.179 -2.495 19.162]	ALA	52	[-0.594 -4.707 21.295]	4.72
THR	50	[-4.179 -2.495 19.162]	GLU	53	[-5.712 -6.423 21.012]	4.6
GLU	51	[-0.638 -4.569 15.827]	CYS	176	[-1.325 -7.752 12.966]	4.33
GLU	51	[-0.638 -4.569 15.827]	ASP	203	[-2.655 -0.301 14.249]	4.98
PHE	54	[-3.867 -9.196 16.646]	CYS	176	[-1.325 -7.752 12.966]	4.7
PHE	54	[-3.867 -9.196 16.646]	TYR	185	[-5.055 -12.517 13.303]	4.86
GLU	55	[0.44 -10.317 19.432]	ASN	178	[3.363 -9.107 15.908]	4.74
GLU	56	[-2.471 -9.749 24.048]	LYS	57	[-6.715 -11.543 22.441]	4.88
CYS	58	[-3.532 -15.126 18.792]	ARG	112	[-1.529 -16.326 14.908]	4.53



Beta-carbon (Cβ) Residue Contacts

Residue 1	Position	Coordinates	Residue 2	Position	Coordinates	Distance (Å)
LEU	14	[8.031 -20.702 34.239]	ALA	15	[10.571 -19.195 36.74]	3.87
PRO	17	[15.099 -17.666 41.14]	LYS	18	[14.88 -16.753 45.434]	4.4
LYS	18	[14.88 -16.753 45.434]	CYS	19	[14.132 -14.755 48.692]	3.89
CYS	19	[14.132 -14.755 48.692]	ASN	20	[11.591 -14.526 52.076]	4.24
LEU	29	[-18.271 -7.338 54.992]	ALA	30	[-20.309 -8.278 51.924]	3.8
ALA	30	[-20.309 -8.278 51.924]	GLU	31	[-22.346 -5.765 48.71]	4.56
MET	46	[-11.246 -9.688 15.791]	ILE	187	[-11.28 -10.978 11.863]	4.13
THR	47	[-13.861 -5.142 13.158]	LEU	48	[-12.219 -1.821 16.403]	4.92
TRP	49	[-8.189 -6.188 17.792]	GLU	53	[-5.712 -6.423 21.012]	4.07
THR	50	[-4.179 -2.495 19.162]	ALA	52	[-0.594 -4.707 21.295]	4.72
THR	50	[-4.179 -2.495 19.162]	GLU	53	[-5.712 -6.423 21.012]	4.6
GLU	51	[-0.638 -4.569 15.827]	CYS	176	[-1.325 -7.752 12.966]	4.33
GLU	51	[-0.638 -4.569 15.827]	ASP	203	[-2.655 -0.301 14.249]	4.98
PHE	54	[-3.867 -9.196 16.646]	CYS	176	[-1.325 -7.752 12.966]	4.7
PHE	54	[-3.867 -9.196 16.646]	TYR	185	[-5.055 -12.517 13.303]	4.86
GLU	55	[0.44 -10.317 19.432]	ASN	178	[3.363 -9.107 15.908]	4.74
GLU	56	[-2.471 -9.749 24.048]	LYS	57	[-6.715 -11.543 22.441]	4.88
CYS	58	[-3.532 -15.126 18.792]	ARG	112	[-1.529 -16.326 14.908]	4.53



Example queries:

These queries were performed by me using admin access to the database.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydb',
        'USER': 'nmegan',
        'PASSWORD': 'password',
        'HOST': '127.0.0.1',
        'PORT': '5432',

    }
}
```

EN.605.652.81.FA24
Megan Nguyen

The user can query the database directly via the command line interface (password in screenshot above):

psql -U nmegan -h localhost mydb

Retrieving protein name per UniProt ID:

```
SELECT p.protein_name, pi.uniprot_id
FROM Protein p
JOIN Protein_Identifier pi ON p.id_number = pi.id_number
JOIN PDB_IDS pdb ON pi.uniprot_id = pdb.uniprot_id
WHERE pdb.pdb_id = '1D8D';
```

```
                                protein_name                                 | uniprot_id
-----------------------------------------------------------------------------+------------
 Protein farnesyltransferase subunit beta                                    | Q02293
 Protein farnesyltransferase/geranylgeranyltransferase type-1 subunit alpha | Q04631
(2 rows)
```

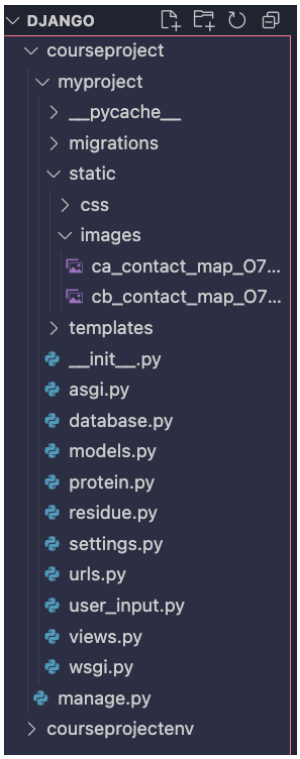
Retrieving the interacting residues (residue 1 and residue 2), their positions, and the distance between them for a UniProt ID:

```
SELECT p.protein_name, pi.uniprot_id, ri.res1_name, ri.res2_name, ri.res1_pos, ri.res2_pos,
ri.distance
FROM Protein p
JOIN Protein_Identifier pi ON p.id_number = pi.id_number
JOIN PDB_IDS pdb ON pi.uniprot_id = pdb.uniprot_id
JOIN Residue_Interactions ri ON p.protein_id = ri.protein_id
WHERE pdb.pdb_id = '1D8D'
ORDER BY ri.contact_id ASC
LIMIT 1;
```

```
                          protein_name                           | uniprot_id | res1_name | res2_name | res1_pos | res2_pos |     distance
-----------------------------------------------------------------+------------+-----------+-----------+----------+----------+------------------
 Protein farnesyltransferase/geranylgeranyltransferase type-1 subunit alpha | Q04631     | MET       | ALA       |        1 |        2 | 3.869999885559082
(1 row)
```

Testing the tool:
A screenshot of the project folder containing the entire code.

EN.605.652.81.FA24
Megan Nguyen



I tested my web application via Django's built-in server and ran it on my local machine. First I changed directory to be located in my Django folder courseproject, which is the root folder for my project, containing everything related to my Django application.

cd ./courseproject
(courseprojectenv) megannguyen@Megans-MacBook-Pro courseproject %

Within courseproject, is the myproject folder, which is the core of the Django application. It includes all the logic, configurations, and files needed to run my application. A summary is below:

courseproject/

- The root directory for my project
- Contains the main Django application (myproject)

myproject/

- The main application folder housing all Django-specific files and logic for the web application

manage.py

- A command-line utility to interact with the Django project (e.g., running the server, migrations)

courseprojectenv/

- The virtual environment housing dependencies for my project

Key subdirectories and files

- **__pycache__/** (stores compiled Python bytecode for performance)
- **migrations/** (tracks database schema changes for the models in the application)
- **static/**
    - **css/styles.css** (contains styling for my application)
    - **images/** (contains static image assets used in the application)
- **templates/** (HTML templates rendering different views of the application)
    - **user_input.html** (user input view where input is entered)
    - **select_id.html** (user view to select a UniProt ID)
    - **results.html** (final results view that displays protein and residue information)

Python files:

The last three Python files form the core of the project's functionality, containing the logic for residue contact calculations and related data processing.

- **asgi.py** (configuration for ASGI (Asynchronous Server Gateway Interface) deployment)
- **wsgi.py** (configuration for WSGI (Web Server Gateway Interface) deployment)
- **database.py** (handles database interactions)
- **models.py** (defines the database schema using Django's ORM)
- **views.py** (contains the logic for rendering pages and interacting with models/templates)
- **urls.py** (maps URLs to views in my application)
- **settings.py** (contains project settings (e.g., database, installed apps, middleware))
- **protein.py** (processes protein-related data or logic, such as protein name, gene, etc.)
- **residue.py** (handles residue-related data or logic, such as identifying residue pairs)
- **user_input.py** (processes user input-related functionality, such as valid inputs)

Running the command below starts the server:
python manage.py runserver

IX. Conclusion and Future Work

As briefly mentioned in Section II: Related Work, I plan to expand this project to encompass broader protein studies. With that being said, there are some smaller tasks that will need to be incorporated into the tool before expanding into broader protein studies. I will need to account for residue charges at some point in time, as well as the calculation of side chains. I also would include some flexibility in the threshold cut off. Users would be able to adjust the distance cutoff themselves, since there is a bit of ambiguity in what is the "correct" distance cutoff is. Additionally, as mentioned in Section VIII. Implementation, the major limitation of this tool is the inability for users to directly access the backend database. In other words, users cannot query the database directly—they can only query through a higher-level interface. Although the UI was completed, I did not have enough time to incorporate direct user access to the database. In terms of storage for the contact map images, there definitely could have been a better method to save the user images. Finally, I plan to actually deploy my tool to a production sever. For the entirety of the project, I was running my web application via Django's built-in server and testing it on my local machine, but ultimately did not have the time to actual deploy the tool. Ideally, if additional time was permitted, I would use Docker for this.

The higher-level goal involves the expansion into broader protein studies. Ultimately, I am aiming to develop a machine learning model capable of accurately predicting protein-protein interactions (or protein structures), with the goal of contributing to drug design. PPIs as drug targets are challenging, as the interfaces between interacting proteins are typically highly hydrophobic, larger than the usual receptor-ligand contact areas, and consist of residues that bind with high affinity [6]. As a result, it is difficult for small molecules to inhibit PPIs, effectively making drug selectivity more difficult as well. Therefore, by accurately predicting PPIs, we could improve drug design and potentially accelerate the development of novel therapeutics.

X. References

[1] Adhikari B, Cheng J. Protein Residue Contacts and Prediction Methods. Methods Mol Biol. 2016;1415:463-76. doi: 10.1007/978-1-4939-3572-7_24. PMID: 27115648; PMCID: PMC4894841.

[2] Zhao, N., Pang, B., Shyu, C.-R. and Korkin, D. (2011), Charged residues at protein interaction interfaces: Unexpected conservation and orchestrated divergence. Protein Science, 20: 1275-1284. https://doi.org/10.1002/pro.655

[3] Adhikari B, Cheng J. Protein Residue Contacts and Prediction Methods. Methods Mol Biol. 2016;1415:463-76. doi: 10.1007/978-1-4939-3572-7_24. PMID: 27115648; PMCID: PMC4894841.

[4] Li, Y., Hu, J., Zhang, C., Yu, D.-J., & Zhang, Y. (2021). Deducing high-accuracy protein contact-maps from a triplet of coevolutionary matrices through deep residual convolutional networks. *PLOS Computational Biology, 17*(3), e1008865. https://doi.org/10.1371/journal.pcbi.1008865

[5] Fariselli P, Casadio R. A neural network based predictor of residue contacts in proteins. Protein Eng. 1999 Jan;12(1):15-21. doi: 10.1093/protein/12.1.15. PMID: 10065706.

[6] Oláh J, Szénási T, Lehotzky A, Norris V, Ovádi J. Challenges in Discovering Drugs That Target the Protein-Protein Interactions of Disordered Proteins. Int J Mol Sci. 2022 Jan 28;23(3):1550. doi: 10.3390/ijms23031550. PMID: 35163473; PMCID: PMC8835748.

[7] Salamanca Viloria, J., Allega, M.F., Lambrughi, M. *et al.* An optimal distance cutoff for contact-based Protein Structure Networks using side-chain centers of mass. *Sci Rep* **7**, 2838 (2017). https://doi.org/10.1038/s41598-017-01498-6

Online resources referenced:
https://docs.djangoproject.com/en/5.1/
https://www.w3schools.com/html/html_scripts.asp
https://www.digitalocean.com/community/tutorials/how-to-add-javascript-to-html
https://medium.com/@dillonf2/creating-a-very-simple-button-in-javascript-51b870133bc6
https://www.geeksforgeeks.org/third-normal-form-3nf/

EN.605.652.81.FA24
Megan Nguyen

XI. Appendix A

The data dictionary is listed below. In summary, I combined each table into one group, as this is mostly accurate (general protein data was obtained from UniProt via the UniProt ID, and residue information used to determine contacts was retrieved from AlphaFold). The data types are listed in the "Type" column. Each data group contains a mix of data types (as shown in the ER diagram).

Data	Description	Type	Source
Protein	General protein data such as protein name, gene, etc.). This is obtained from the UniProt API.	TEXT, INT, VARCHAR	https://rest.uniprot.org/uniprotkb/O75626.json and https://www.ebi.ac.uk/pdbe/api/ (PDB to UniProt ID mapping)
Residue/Residue Interaction	General residue data, including residue interaction data (interacting residues, their positions, and the calculated atoms). This is obtained from the AlphaFold API	TEXT, INT, VARCHAR	https://alphafold.ebi.ac.uk/api/prediction/O75626 and https://alphafold.ebi.ac.uk/files/AF-O75626-F1-model_v4.cif
Source	Source information obtained from UniProt and Alphafold	TEXT, DATE, VARCHAR	UniProt and Alphafold API