Development of Search APIs for FlavorDB2: BTech Project in API Creation and Data Management

Student's Name: Sarvajeeth U K, Lakshay Goel, Nabh Rajput

Roll Number: 2021417,2021,2021

BTP report submitted in partial fulfillment of the requirements
for the Degree of B.Tech. in CSE

on ...(27th November 2024)...

BTP Track: Engineering Track

BTP Advisor

Dr Ganesh Bagler

Indraprastha Institute of Information Technology

New Delhi

Student’s Declaration

*We hereby declare that the work presented in the report entitled “**Development of Search APIs for FlavorDB2: BTech Project in API Creation and Data Management**”, submitted by us for the partial fulfillment of the requirements for the degree of Bachelor of Technology at Indraprastha Institute of Information Technology, Delhi, is an authentic record of our work carried out under the guidance of Dr. Ganesh Bagler. Due acknowledgments have been given in the report to all material used. This work has not been submitted anywhere else for the award of any other degree.*

Place & Date: IIIT Delhi, 25th November 2024
Sarvajeeth U K
Lakshay Goel
Nabh Rajput

Certificate

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

....
Place & Date: IIIT Delhi, 27th November 2024
Dr. Ganesh Bagler
BTP Advisor

Abstract

This report presents the development of robust search APIs for FlavorDB2, an advanced database of flavor molecules and their associated properties, aimed at enhancing data accessibility and usability for researchers in computational gastronomy and food science. The project involved migrating raw CSV datasets to a MongoDB database, followed by the implementation of RESTful APIs using the Spring Boot framework.

The APIs enable complex and multidimensional queries, allowing users to retrieve data based on molecular identifiers, regulatory statuses, food categories, and synthesis processes. Advanced features like query filtering and aggregation ensure precise results and scalability.

Table of Contents

1. **Introduction**
    - 2.1 Context
    - 2.2 Problem Statement
    - 2.3 Project Objectives
2. **Methodology**
    - 3.1 Data Preparation
    - 3.2 API Design and Implementation
    - 3.3 Performance Optimization
    - 3.4 Testing and Validation
    - 3.5 Documentation
3. **System Architecture**
    - 4.1 Layered Architecture
    - 4.2 Workflow
    - 4.3 Technologies Used
    - 4.4 Architecture Diagram
4. **Results and Discussion**
    - 5.1 Key Achievements
    - 5.2 User Experience Enhancements
    - 5.3 Challenges and Solutions
5. **Conclusion**
    - 6.1 Summary of Contributions
    - 6.2 Impact
    - 6.3 Closing Remarks
6. **References**
7. **Appendix**
    - Appendix A: Data
    - Appendix B: API Endpoints
        - i. B.1 Molecule Data Endpoints
        - ii. B.2 More Properties Endpoints
        - iii. B.3 Functional Properties Endpoints
        - iv. B.4 Entity Endpoints
    - Appendix C: Example Dataset Snippets
    - Appendix D: Postman Query and Response

Introduction

Context

FlavorDB2 is a cutting-edge database that provides extensive insights into flavor molecules, including their chemical properties, sensory attributes, and natural or synthetic origins. It serves as a critical resource for researchers in computational gastronomy, food science, and related disciplines. Despite its comprehensive data, the usability of FlavorDB2 was limited by the lack of efficient mechanisms to perform dynamic and multidimensional searches.

Problem Statement

With over 25,000 molecules and 900+ food categories, retrieving specific data from FlavorDB2 posed significant challenges, especially for advanced queries that combine molecular, sensory, and regulatory dimensions. Existing tools lacked the depth and flexibility required for such operations, leading to a demand for a scalable and intuitive API-based solution.

Project Objectives

The primary goal of this project was to design and implement a robust system of search APIs for FlavorDB2 to improve its accessibility and functionality. This involved:

1. **Data Migration**:
    - Transforming raw datasets from CSV files into a structured MongoDB database for efficient storage and querying.
2. **API Development**:
    - Creating RESTful APIs using Spring Boot to support advanced search functionalities, such as filtering by molecular identifiers, food categories, and synthesis methods.
3. **Performance Optimization**:
    - Ensuring the APIs deliver quick, accurate responses while handling large datasets and complex queries.

4. **Testing and Documentation**:
    - Validating API functionality and performance under varying scenarios and providing clear documentation for developers and end-users.

This report highlights the methodologies, achievements, and engineering challenges addressed during the development of search APIs, which bridge the gap between raw database resources and user-friendly interfaces.

Methodology

The project followed a systematic approach, ensuring that each phase contributed to the development of efficient and scalable search APIs for FlavorDB2. The methodology is detailed below:

1. Data Preparation

- **Dataset Conversion**:
    - Raw data provided in CSV format was reviewed for inconsistencies such as missing or duplicate values.
    - Pre-processing steps, including data cleaning and transformation, were performed using Python libraries like `pandas` and `NumPy`.
- **MongoDB Schema Design**:
    - Data was structured into collections aligned with logical categories, such as `molecules_data`, `entities_data`, and `properties_data`.
    - Schemas were designed to support complex queries, indexing key fields like CAS numbers and molecular weights to enhance retrieval performance.

2. API Design and Implementation

- **Framework Selection**:
    - Spring Boot was chosen for its robust support for building RESTful APIs and seamless integration with MongoDB.
- **Endpoint Development**:

    - Designed API endpoints to support diverse queries, categorized as:
        - **Molecular Search**: By identifiers such as CAS number, IUPAC name, or SMILES representation.
        - **Food Category Search**: By categories like fruits, spices, or beverages.
        - **Regulatory Search**: Based on approval statuses from organizations like FDA and IOFI.
        - **Synthesis Search**: By chemical processes or molecular structures.
- **Advanced Query Capabilities**:
    - Implemented filtering, logical operators (AND, OR), and range-based searches for precision.
    - Enabled multidimensional queries, allowing users to combine criteria for refined results.
- **Error Handling**:
    - Included mechanisms to handle invalid inputs and missing parameters, ensuring robust API functionality.

3. Performance Optimization

- **Indexing**:
    - Indexed frequently queried fields in MongoDB to reduce lookup times for large datasets.
    - Used compound indexing to handle complex queries involving multiple parameters.
- **Query Aggregation**:
    - Leveraged MongoDB's aggregation framework to optimize query execution and support advanced filtering.
- **Pagination**:
    - Implemented pagination to divide large result sets into manageable chunks, reducing server load and improving response times.

4. Testing and Validation

- **Functional Testing**:
  - Verified API endpoints using Postman, ensuring correct responses for valid inputs and robust handling of edge cases.
- **Performance Testing**:
  - Conducted load testing using JMeter to measure query response times under varying loads.
  - Achieved sub-second response times for most typical queries, validating system scalability.
- **Integration Testing**:
  - Tested seamless interaction between APIs and the MongoDB backend, ensuring consistency in data retrieval and formatting.

5. Documentation

- Detailed API documentation was created, including:
  - Endpoint descriptions, required parameters, and example queries.
  - Deployment and testing guidelines for developers and end-users.
- Provided sample use cases to demonstrate API capabilities.

System Architecture

The architecture of this project follows the standard **Spring Boot MVC (Model-View-Controller)** pattern, ensuring modularity, scalability, and maintainability. It is tailored to efficiently handle the requirements of robust search APIs for FlavorDB2.

1. Layered Architecture

1. **Controller Layer**:
   - Handles HTTP requests from users.
   - Maps API endpoints to corresponding service methods.
   - Validates user inputs before passing them to the service layer.

   - Example: `EntityController.java` and `MoleculeController.java`.
2. **Service Layer**:
   - Implements the core business logic.
   - Processes user queries and interacts with the repository layer.
   - Performs data transformations, aggregations, and validations as needed.
   - Example: `EntityService.java` and `MoleculeService.java`.
3. **Repository Layer**:
   - Provides a data access abstraction for MongoDB.
   - Executes database queries using MongoDB's native query language and aggregation framework.
   - Example: `EntityRepository.java` and `MoleculeRepository.java`.
4. **Database**:
   - MongoDB serves as the backend database.
   - Collections are designed to support efficient data retrieval and accommodate complex queries with features like indexing.

2. Workflow

1. **User Request**:
   - A client sends an API request (e.g., `GET /api/entities/by-name-and-category`).
2. **Controller Processing**:
   - The Controller layer validates the request and routes it to the appropriate service.
3. **Service Logic**:
   - The Service layer constructs the query logic and interacts with the repository.
4. **Database Query**:

    - The Repository executes the MongoDB query, leveraging indexes and the aggregation framework for efficiency.
5. **Response Handling**:
    - The retrieved data is formatted into JSON by the Controller and sent back to the user.

3. Technologies Used

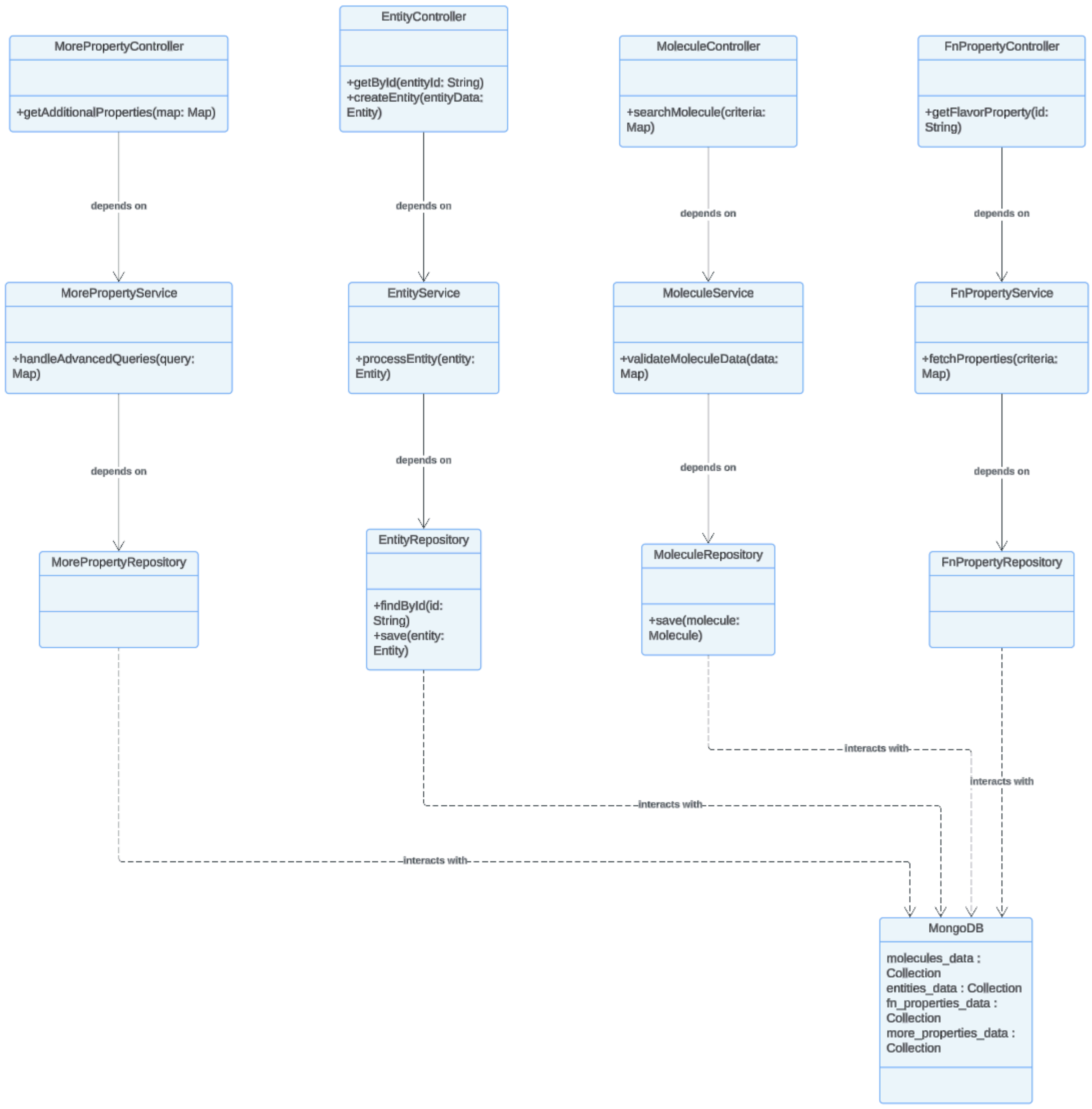
1. **Backend**:
    - **Spring Boot**: Framework for building RESTful APIs with robust dependency injection and modular design.
    - **Java**: Primary programming language for the implementation.
2. **Database**:
    - **MongoDB**: NoSQL database chosen for its flexibility and support for large datasets.
3. **API Testing**:
    - **Postman**: Used for functional testing of endpoints.
4. **Development Tools**:
    - **Maven**: For dependency management and build automation.
    - **MongoDB Compass**: For database schema visualization and query debugging.

4. Architecture Diagram

To visually represent this architecture, the diagram can illustrate the interaction flow:

1. **Client → Controller → Service → Repository → MongoDB → Response**.

UML Diagram


MorePropertyController
+getAdditionalProperties(map: Map)
EntityController
+getById(entityId: String)
+createEntity(entityData: Entity)
MoleculeController
+searchMolecule(criteria: Map)
FnPropertyController
+getFlavorProperty(id: String)
depends on
MorePropertyService
+handleAdvancedQueries(query: Map)
EntityService
+processEntity(entity: Entity)
MoleculeService
+validateMoleculeData(data: Map)
FnPropertyService
+fetchProperties(criteria: Map)
MorePropertyRepository
EntityRepository
+findById(id: String)
+save(entity: Entity)
MoleculeRepository
+save(molecule: Molecule)
FnPropertyRepository
interacts with
MongoDB
molecules_data : Collection
entities_data : Collection
fn_properties_data : Collection
more_properties_data : Collection

Results and Discussion

The development and implementation of search APIs for FlavorDB2 has laid a strong foundation for enhancing data accessibility, query performance, and user experience. While the APIs are not yet deployed, the completed stages demonstrate significant potential for improving interaction with the database. This section summarizes the progress and discusses its implications.

1. Key Achievements

- **Comprehensive API Functionality**:
    - Multiple RESTful APIs were implemented to support diverse search functionalities, including:
        - **Molecular Search**: By identifiers like CAS number, IUPAC name, and SMILES representation.
        - **Food Category Search**: By categories such as fruits, spices, or beverages.
        - **Regulatory Search**: Based on approval statuses from regulatory bodies like FDA and IOFI.
        - **Synthesis Search**: By chemical processes or molecular properties.
    - Advanced query capabilities enabled users to combine multiple search criteria with logical operators for refined results.
- **Efficient Data Migration**:
    - Raw datasets were successfully migrated from CSV files to a structured MongoDB database.
    - Collections were indexed to improve query performance and organized to facilitate complex searches.

2. User Experience Enhancements

- **Planned Improvements**:
    - The APIs are expected to provide an intuitive and seamless experience for researchers by:
        - Enabling quick access to specific molecular data with minimal effort.
        - Supporting complex analyses through multidimensional queries with logical operators.
        - Returning data in lightweight JSON formats suitable for integration with downstream applications.

3. Challenges and Solutions

- **Challenge**: Handling large datasets and complex queries without compromising performance.
  - **Solution**: Utilized MongoDB’s aggregation framework, efficient schema design, and indexing techniques.
- **Challenge**: Ensuring APIs are robust and handle diverse user queries gracefully.
  - **Solution**: Implemented thorough input validation and error handling mechanisms.

Conclusion

The development of search APIs for FlavorDB2 marks a significant advancement in enhancing the accessibility and usability of one of the most comprehensive flavor molecule databases. This project addressed critical challenges in data management and query performance, delivering a scalable and efficient solution tailored to the needs of researchers and professionals in computational gastronomy and food science.

Summary of Contributions

1. **Robust Search Capabilities**:
   - APIs were developed to handle diverse and complex queries, enabling users to retrieve data across molecular, sensory, regulatory, and synthesis dimensions.
2. **Efficient Data Migration**:
   - Migrated raw CSV datasets into a well-structured MongoDB database, optimizing data storage and query efficiency.
3. **Performance Optimization**:
   - Leveraged MongoDB indexing, query aggregation, and pagination to achieve sub-second response times for most use cases.
4. **Comprehensive Documentation**:
   - Provided clear API documentation, enabling seamless adoption by developers and end-users.

Impact

1. **For Computational Gastronomy**:
   - The APIs enable advanced analyses, such as flavor pairing, recipe generation, and molecular profiling.
2. **For Food Science**:
   - Researchers can now explore food-flavor relationships, regulatory statuses, and synthesis methods with greater ease and precision.

Closing Remarks

This project successfully bridges the gap between raw data and actionable insights, transforming FlavorDB2 into a user-friendly and scalable tool for research and innovation. It also sets the stage for future enhancements, such as machine learning-driven predictions, database expansions, and the development of interactive front-end tools.

References

Here is a list of references used throughout the project to support the development and understanding of FlavorDB2 and its associated APIs. Proper citations ensure the credibility of the work and acknowledge the contributions of prior research and tools.

1. Garg, N., Sethupathy, A., Tuwani, R., Rakhi, N. K., Dokania, S., Iyer, A., Gupta, A., et al. (2018). FlavorDB: A Database of Flavor Molecules. *Nucleic Acids Research, 46(D1),* D1210–D1216. [https://doi.org/10.1093/nar/gkx957]
2. Grover, N., Goel, M., Batra, D., Garg, N., Tuwani, R., Sethupathy, A., & Bagler, G. (2022). FlavorDB2: An Updated Database of Flavor Molecules. *Computational Gastronomy Repository.*[https://cosylab.iiitd.edu.in/flavordb2/]
3. MongoDB Documentation. Available at: [https://www.mongodb.com/docs/]
4. Spring Boot Documentation. Available at: [https://spring.io/projects/spring-boot]
5. Apache HTTP Server Documentation. Available at: [https://httpd.apache.org/]

Appendix - FlavourDB2

A. Molucule_Data

1. Fetch Molecules by Common Name

- Endpoint: /api/molecules_data/by-commonName
- HTTP Method: GET
- Inputs:
  - Query parameter:
    - commonName (String)
      Example: /api/molecules_data/by-commonName?commonName=glucose
- Outputs:

Output JSON Fields (mapped from fdb_molecules.csv):

```
{
 "id": "string",
 "pubchemId": int,
 "iupacName": "string",
 "commonName": "string",
 "smile": "string",
 "molecularWeight": float,
 "hbdCount": int,
 "hbaCount": int,
 "numRotatableBonds": int,
 "complexity": float,
 "topologicalPolarSurfaceArea": float,
 "monoisotopicMass": float,
 "exactMass": float,
 "xlogp": float,
 "charge": int,
 "heavyAtomCount": int,
 "atomStereoCount": int,
 "definedAtomStereoCenterCount": int,
 "undefinedAtomStereoCenterCount": int,
 "bondStereoCount": int,
 "definedBondStereoCenterCount": int,
 "undefinedBondStereoCenterCount": int,
 "isotopeAtomCount": int,
 "covalentlyBondedUnitCount": int,
 "casId": "string",
 "femaNumber": "string",
 "femaFlavorProfile": "string",
 "odor": "string",
 "taste": "string",
 "functionalGroups": "string",
 "inchi": "string",
 "volume3d": float,
 "foodDbFlavorProfile": "string",
 "superSweet": "string",
 "bitter": float,
 "superSweetDbId": "string",
 "bitterDbId": "string",
 "foodDbId": "string",
 "flavorNetId": int,
 "fenoroliAndOs": int,
 "natural": int,
 "unknownNatural": int,
 "synthetic": int,
 "flavorProfile": "string"
}
```

2. Fetch Molecules by Functional Groups

- **Endpoint**: /api/molecules_data/by-functionalGroups
- **HTTP Method**: GET
- **Inputs**:
  - Query parameter:
    - functionalGroups (String)
      Example:
      /api/molecules_data/by-functionalGroups?functionalGroups=alcohol
- **Outputs**:
  Same JSON structure as above.

3. Fetch Molecules by Flavor Profile

- **Endpoint**: /api/molecules_data/by-flavorProfile
- **HTTP Method**: GET
- **Inputs**:
  - Query parameter:

        - flavorProfile (String)
          Example: /api/molecules_data/by-flavorProfile?flavorProfile=sweet
- **Outputs**:
  Same JSON structure as above.

4. Fetch Molecules by FEMA Flavor Profile

- **Endpoint**: /api/molecules_data/by-femaFlavorProfile
- **HTTP Method**: GET
- **Inputs**:
    - Query parameter:
        - femaFlavorProfile (String)
          Example:
          /api/molecules_data/by-femaFlavorProfile?femaFlavorProfile=bitter
- **Outputs**:
  Same JSON structure as above.

5. Fetch Molecules by PubChem ID

- **Endpoint**: /api/molecules_data/by-pubchemId
- **HTTP Method**: GET
- **Inputs**:
    - Query parameter:
        - pubchemId (Integer)
          Example: /api/molecules_data/by-pubchemId?pubchemId=123456
- **Outputs**:
  Same JSON structure as above.

6. Fetch Molecules by Monoisotopic Mass

- **Endpoint**: /api/molecules_data/by-monoisotopicMass
- **HTTP Method**: GET
- **Inputs**:
    - Query parameter:
        - monoisotopicMass (Double)
          Example:
          /api/molecules_data/by-monoisotopicMass?monoisotopicMass=120.12
- **Outputs**:
  Same JSON structure as above.

Additional Notes

- The JSON response structure is consistent across all endpoints because the underlying **Molecule model** is the same.
- Filtering is based on the query parameter provided in the API request (e.g., common_name, functional_groups, etc.).
- Each API returns a **list of molecules**, and each molecule in the list has the fields outlined above.

B. More Properties

1. Fetch Properties by Number of Aromatic Rings

- **Endpoint**: `/api/more_properties/by-aromaticRings`
- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameter:
    - `aromatic_rings` (Integer)
      - Example: `/api/more_properties/by-aromaticRings?aromatic_rings=2`
- **Outputs**:

```
[
  {
   "id": "string",
   "pubchemId": int,
   "numberOfAtoms": int,
   "molecularFormula": "string",
   "molecularComposition": "string",
   "molecularWeight": float,
   "molecularMass": float,
   "energy": float,
   "alogp": float,
   "logd": float,
   "molecularSolubility": float,
   "pka": "string",
   "numberOfAromaticBonds": int,
   "numberOfAromaticRings": int,
   "hbaCount": int,
   "hbdCount": int,
   "numberOfHBonds": int,
   "numRings": int,
   "numRotatableBonds": int,
   "surfaceArea": float,
   "molecularSasa": float,
   "radiusOfGyration": float,
   "molecular3dSasa": float
  }
]
```

2. Fetch Properties by Number of Aromatic Bonds

- **Endpoint**: `/api/more_properties/by-aromaticBonds`
- **HTTP Method**: `GET`
- **Inputs**:
    - Query parameter:
        - `aromatic_bonds` (Integer)
            - Example: `/api/more_properties/by-aromaticBonds?aromatic_bonds=6`
- **Outputs**: Same JSON structure as above.

3. Fetch Properties by Number of Rings

- **Endpoint**: `/api/more_properties/by-numRings`
- **HTTP Method**: `GET`
- **Inputs**:
    - Query parameter:
        - `num_rings` (Integer)
            - Example: `/api/more_properties/by-numRings?num_rings=3`
- **Outputs**: Same JSON structure as above.

4. Fetch Properties by Number of Rotatable Bonds

- **Endpoint**: `/api/more_properties/by-numRotatableBonds`

- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameter:
    - `num_rotatableBonds` (Integer)
      - Example: `/api/more_properties/by-numRotatableBonds?num_rotatableBonds=4`
- **Outputs**: Same JSON structure as above.

5. Fetch Properties by Surface Area

- **Endpoint**: `/api/more_properties/by-surfaceArea`
- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameter:
    - `surface_area` (Double)
      - Example: `/api/more_properties/by-surfaceArea?surface_area=150.5`
- **Outputs**: Same JSON structure as above.

C. FnProperty

1. Fetch Properties by CoE Number

- **Endpoint**: `/api/properties/by-coe`
- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameter:
    - `coe_no` (String)
      - Example: `/api/properties/by-coe?coe_no=1234`
- **Outputs**:

```
[
  {
   "id": "string",
   "name": "string",
   "botanicalName": "string",
   "botanicalFamily": "string",
   "synonyms": "string",
   "casNo": "string",
   "flNo": "string",
   "femaNo": float,
   "nasNo": "string",
   "coeNo": "string",
   "einecsNo": "string",
   "jecfaNo": "string",
   "eNo": "string",
   "description": "string",
   "consumption": "string",
   "individual": "string",
   "tradeAssociationGuidelines": "string",
   "iofi": "string",
   "specifications": "string",
   "reportedUsesPpm": "string",
   "foodCategoryUsualMax": "string",
   "aromaThresholdValues": "string",
   "tasteThresholdValues": "string",
   "empiricalFormulaMw": "string",
   "synthesis": "string",
   "naturalOccurrence": "string",
   "eafusNo": "string",
   "derivatives": "string",
   "derivativeNames": "string",
   "foreignNames": "string",
   "essentialOilComposition": "string",
   "genusSpecies": "string",
   "composition": "string",
   "note": "string",
   "insNo": "string",
   "animalFamily": "string",
   "physicalChemicalCharacteristics": "string"
  }
]
```

2. Fetch Properties by FEMA Number

- **Endpoint**: `/api/properties/by-fema`
- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameter:
    - `fema_no` (Double)
      - Example: `/api/properties/by-fema?fema_no=3965`
- **Outputs**: Same JSON structure as above.

3. Fetch Properties by FL Number

- **Endpoint**: `/api/properties/by-flNo`
- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameter:
    - `fl_no` (String)
      - Example: `/api/properties/by-flNo?fl_no=123456`
- **Outputs**: Same JSON structure as above.

4. Fetch Properties by NAS Number

- **Endpoint**: `/api/properties/by-nas`
- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameter:
    - `nas_no` (String)
      - Example: `/api/properties/by-nas?nas_no=7890`
- **Outputs**: Same JSON structure as above.

5. Fetch Properties by EINECS Number

- **Endpoint**: `/api/properties/by-einecs`
- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameter:
    - `einecs_no` (String)
      - Example: `/api/properties/by-einecs?einecs_no=5678`

- **Outputs**: Same JSON structure as above.

6. Fetch Properties by JECFA Number

- **Endpoint**: `/api/properties/by-jecfa`
- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameter:
    - `jecfa_no` (String)
      - Example: `/api/properties/by-jecfa?jecfa_no=3456`
- **Outputs**: Same JSON structure as above.

D. Entity

1. Fetch Entities by Alias and Category

- **Endpoint**: `/api/entities/by-name-and-category`
- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameters:
    - `entity_alias` (String)
      - Example: `/api/entities/by-name-and-category?entity_alias=glucose&category=chemical`
    - `category` (String)
      - Example: `/api/entities/by-name-and-category?entity_alias=glucose&category=chemical`
- **Outputs**:

```
[
  {
   "id": "string",
   "entityId": int,
   "category": "string",
   "categoryReadable": "string",
   "entityAlias": "string",
   "entityAliasBasket": "string",
   "entityAliasReadable": "string",
   "entityAliasSynonyms": "string",
   "entityAliasUrl": "string",
   "naturalSourceName": "string",
   "naturalSourceUrl": "string"
  }
]
```

2. Fetch Entities by Natural Source Name

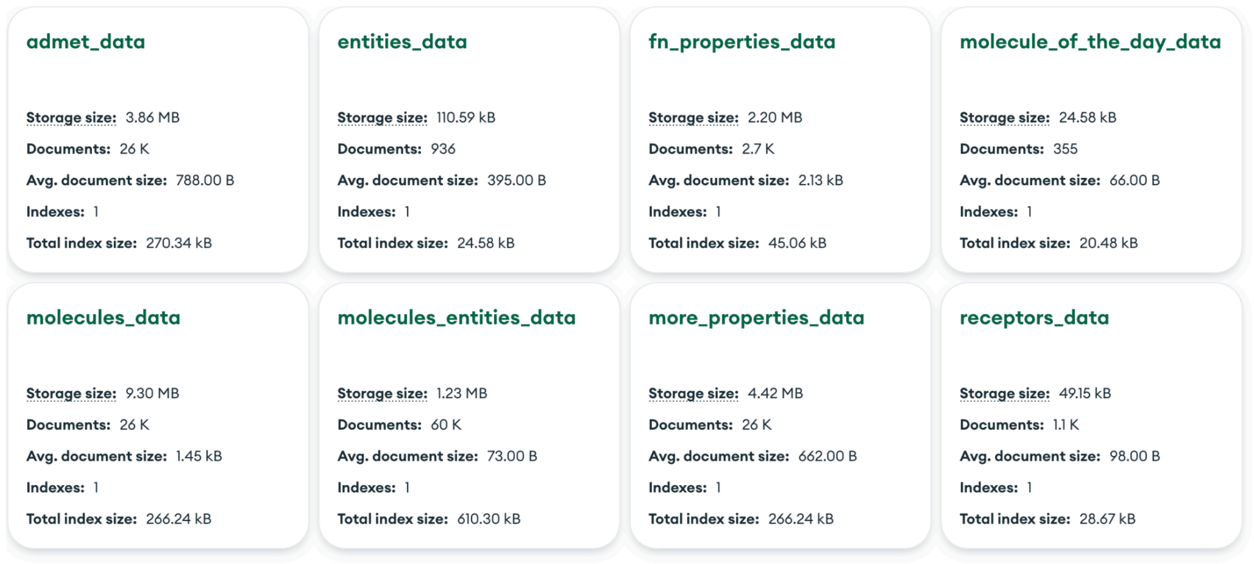
- **Endpoint**: `/api/entities/by-natural-source`
- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameter:
    - `natural_source_name` (String)
      - Example: `/api/entities/by-natural-source?natural_source_name=apple`
- **Outputs**: Same JSON structure as above.

3. Fetch Entities by Readable Alias

- **Endpoint**: `/api/entities/by-alias-readable`
- **HTTP Method**: `GET`
- **Inputs**:
  - Query parameter:
    - `entity_alias_readable` (String)
      - Example: `/api/entities/by-alias-readable?entity_alias_readable=Glucose`
- **Outputs**: Same JSON structure as above.

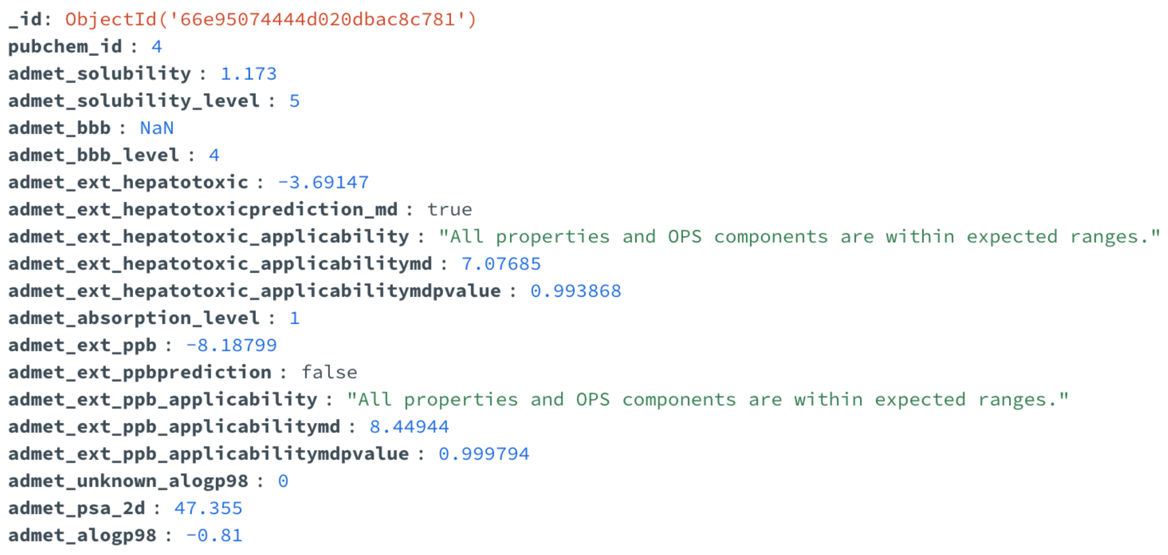
Example Dataset Snippet

1. Collection Snippet



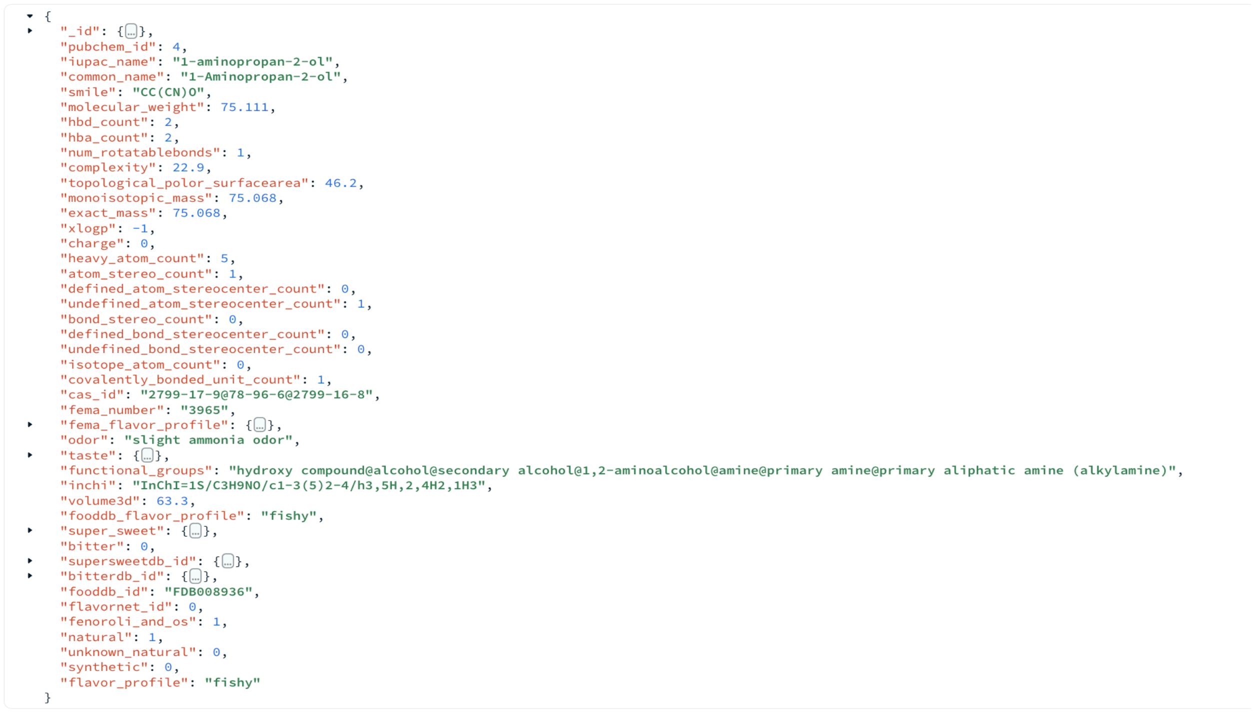
Collection	Storage size	Documents	Avg. document size	Indexes	Total index size
admet_data	3.86 MB	26 K	788.00 B	1	270.34 kB
entities_data	110.59 kB	936	395.00 B	1	24.58 kB
fn_properties_data	2.20 MB	2.7 K	2.13 kB	1	45.06 kB
molecule_of_the_day_data	24.58 kB	355	66.00 B	1	20.48 kB
molecules_data	9.30 MB	26 K	1.45 kB	1	266.24 kB
molecules_entities_data	1.23 MB	60 K	73.00 B	1	610.30 kB
more_properties_data	4.42 MB	26 K	662.00 B	1	266.24 kB
receptors_data	49.15 kB	1.1 K	98.00 B	1	28.67 kB

Data Snippet



```
_id: ObjectId('66e95074444d020dbac8c781')
pubchem_id : 4
admet_solubility : 1.173
admet_solubility_level : 5
admet_bbb : NaN
admet_bbb_level : 4
admet_ext_hepatotoxic : -3.69147
admet_ext_hepatotoxicprediction_md : true
admet_ext_hepatotoxic_applicability : "All properties and OPS components are within expected ranges."
admet_ext_hepatotoxic_applicabilitymd : 7.07685
admet_ext_hepatotoxic_applicabilitymdpvalue : 0.993868
admet_absorption_level : 1
admet_ext_ppb : -8.18799
admet_ext_ppbprediction : false
admet_ext_ppb_applicability : "All properties and OPS components are within expected ranges."
admet_ext_ppb_applicabilitymd : 8.44944
admet_ext_ppb_applicabilitymdpvalue : 0.999794
admet_unknown_alogp98 : 0
admet_psa_2d : 47.355
admet_alogp98 : -0.81
```

Flavor Molecules Collection:

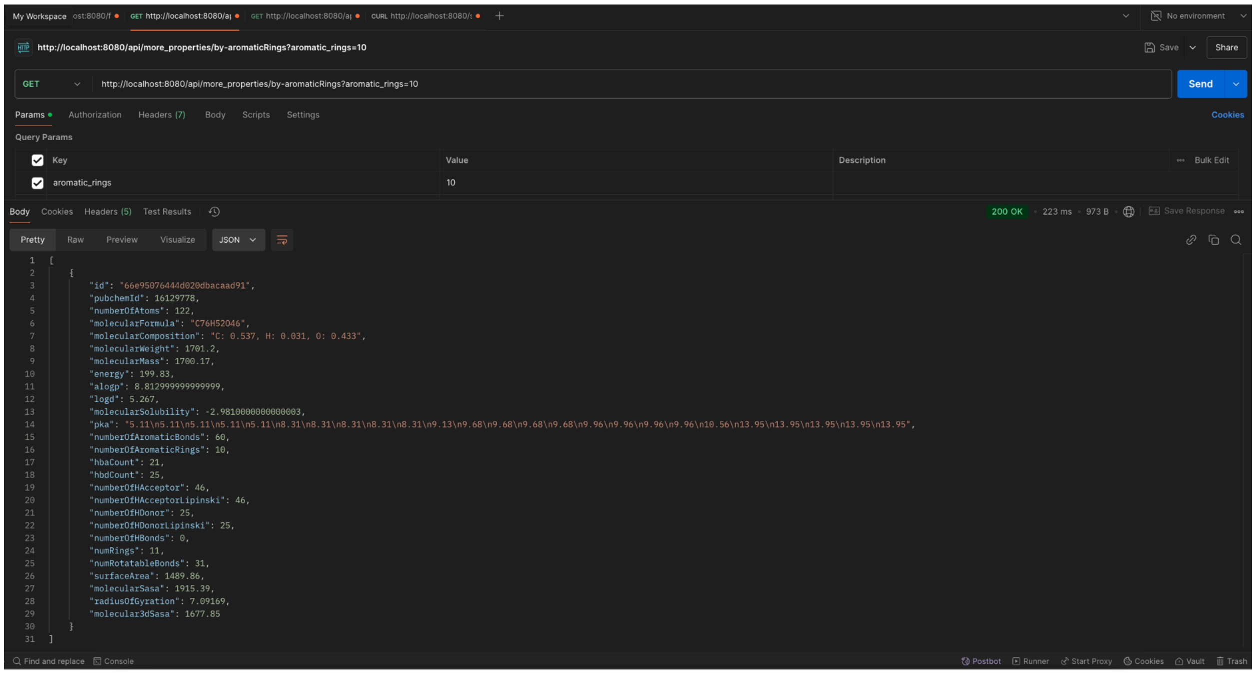


```
{
  "_id": {…},
  "pubchem_id": 4,
  "iupac_name": "1-aminopropan-2-ol",
  "common_name": "1-Aminopropan-2-ol",
  "smile": "CC(CN)O",
  "molecular_weight": 75.111,
  "hbd_count": 2,
  "hba_count": 2,
  "num_rotatablebonds": 1,
  "complexity": 22.9,
  "topological_polor_surfacearea": 46.2,
  "monoisotopic_mass": 75.068,
  "exact_mass": 75.068,
  "xlogp": -1,
  "charge": 0,
  "heavy_atom_count": 5,
  "atom_stereo_count": 1,
  "defined_atom_stereocenter_count": 0,
  "undefined_atom_stereocenter_count": 1,
  "bond_stereo_count": 0,
  "defined_bond_stereocenter_count": 0,
  "undefined_bond_stereocenter_count": 0,
  "isotope_atom_count": 0,
  "covalently_bonded_unit_count": 1,
  "cas_id": "2799-17-9@78-96-6@2799-16-8",
  "fema_number": "3965",
  "fema_flavor_profile": {…},
  "odor": "slight ammonia odor",
  "taste": {…},
  "functional_groups": "hydroxy compound@alcohol@secondary alcohol@1,2-aminoalcohol@amine@primary amine@primary aliphatic amine (alkylamine)",
  "inchi": "InChI=1S/C3H9NO/c1-3(5)2-4/h3,5H,2,4H2,1H3",
  "volume3d": 63.3,
  "fooddb_flavor_profile": "fishy",
  "super_sweet": {…},
  "bitter": 0,
  "supersweetdb_id": {…},
  "bitterdb_id": {…},
  "fooddb_id": "FDB008936",
  "flavornet_id": 0,
  "fenoroli_and_os": 1,
  "natural": 1,
  "unknown_natural": 0,
  "synthetic": 0,
  "flavor_profile": "fishy"
}
```

Postman Query and Response:

- **Query**: `/api/molecule-search?name=Vanillin`

Response:



GET http://localhost:8080/api/more_properties/by-aromaticRings?aromatic_rings=10

Query Params

Key	Value	Description
aromatic_rings	10	

200 OK · 223 ms · 973 B

```
[
    {
        "id": "66e95076444d020dbacaad91",
        "pubchemId": 16129778,
        "numberOfAtoms": 122,
        "molecularFormula": "C76H52046",
        "molecularComposition": "C: 0.537, H: 0.031, O: 0.433",
        "molecularWeight": 1701.2,
        "molecularMass": 1700.17,
        "energy": 199.83,
        "alogp": 8.812999999999999,
        "logd": 5.267,
        "molecularSolubility": -2.9810000000000003,
        "pka": "5.11\n5.11\n5.11\n5.11\n5.11\n8.31\n8.31\n8.31\n8.31\n8.31\n9.13\n9.68\n9.68\n9.68\n9.68\n9.96\n9.96\n9.96\n9.96\n10.56\n13.95\n13.95\n13.95\n13.95\n13.95",
        "numberOfAromaticBonds": 60,
        "numberOfAromaticRings": 10,
        "hbaCount": 21,
        "hbdCount": 25,
        "numberOfHAcceptor": 46,
        "numberOfHAcceptorLipinski": 46,
        "numberOfHDonor": 25,
        "numberOfHDonorLipinski": 25,
        "numberOfHBonds": 0,
        "numRings": 11,
        "numRotatableBonds": 31,
        "surfaceArea": 1489.86,
        "molecularSasa": 1915.39,
        "radiusOfGyration": 7.09169,
        "molecular3dSasa": 1677.85
    }
]
```