



**GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND  
TECHNOLOGY**

**(Approved by AICTE, Autonomous under JNTUH, Hyderabad)**

**Department of Computer Science and Engineering**

**CERTIFICATE**

This is to certify that the Industrial major project entitled “**MOLECULAR  
SEARCH SYSTEM FOR CSIR-ICT MOLBANK**” is submitted by

<b>SASIREKHA KAMBHAMPATY</b>	<b>14241A0544</b>
<b>VINAY GUDA</b>	<b>14241A05D1</b>
<b>Y. SAI KRISHNA VIVEK</b>	<b>14241A05H4</b>
<b>CH SAI BHUVAN CHANDRA</b>	<b>14241A05N5</b>

In partial fulfillment of the award of degree in BACHELOR OF TECHNOLOGY in Computer Science and Engineering during academic year 2017-2018.

INTERNAL GUIDE

**Asst. Prof. Raghuram Kunsoth**

HEAD OF THE DEPARTMENT

**Prof. Dr. Ch. Mallikarjuna Rao**

EXTERNAL EXAMINER

## ACKNOWLEDGEMENT

There are many people who helped us directly and indirectly to complete our project successfully. We would like to take this opportunity to thank one and all. First, we would like to express our gratitude towards our internal guide **Mr. Raghuram Kunsoth, Asst. Professor**, Department of CSE for his support in the completion of our dissertation. We wish to express our sincere thanks to **Ch. Mallikarjuna Rao, HOD, Department of CSE** and also to our principal **Dr. J Praveen** for providing the facilities to complete the dissertation. We would like to thank all our faculty and friends for their help and constructive criticism during the project period. Finally, we are greatly indebted to our parents for their moral support and encouragement to achieve goals.

**SASIREKHA KAMBHAMPATY**

14241A0544

**VINAY GUDA**

14241A05D1

**Y. SAI KRISHNA VIVEK**

14241A05H4

**CH SAI BHUVAN CHANDRA**

14241A05N5

## **DECLARATION**

I hereby declare that the industrial major project entitled “**MOLECULAR SEARCH SYSTEM FOR CSIR-IICT MOLBANK**” is the work done during the period from **4th December, 2017 to 6th April, 2018** and is submitted in the partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering from Gokaraju Rangaraju Institute of Engineering and Technology (Autonomous under Jawaharlal Nehru Technology University, Hyderabad). The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

**SASIREKHA KAMBHAMPATY**

14241A0544

**VINAY GUDA**

14241A05D1

**Y. SAI KRISHNA VIVEK**

14241A05H4

**CH SAI BHUVAN CHANDRA**

14241A05N5

## ABSTRACT

At the Indian Institute of Chemical Technology, there is a requirement for a chemical database that stores physical and chemical properties of samples submitted for record keeping. These molecules have properties such as structure, molecular weight, formula, SMILES, IUPAC name, sample code etc. We have developed a database, as well as a website, that will allow permitted users to upload new molecules, search existing molecules by structure and query string, and be informed of updates made to the database. The Molecular Search System is also able to generate a report and is secure throughout the entire session. We have opted to build the web application using PostgreSQL and Django to satisfy the above requirement. We have used an open-source RDBMS data cartridge, called Bingo, to store and operate on molecular representations in the database. Additionally, we have used Indigo toolkit to generate 2D coordinates of SMILES IDs, and Kekule.js to render these 2D coordinates onto the browser. All the technologies used in this project are open-source and free of cost.

# CONTENTS

CERTIFICATE.....	i
ACKNOWLEDGEMENT .....	ii
DECLARATION .....	iii
ABSTRACT .....	iv
CONTENTS .....	v
1. Introduction.....	1
1.1. Chemical Databases.....	1
1.2. Background.....	2
1.2.1. Search .....	2
1.2.2. Structure Representation .....	2
1.3. Project Overview .....	3
1.3.1. Existing System .....	3
1.3.2. Problems with the existing System.....	3
1.3.3. Proposed System.....	4
1.3.4. Scope of Proposed System .....	4
2. System Requirements .....	6
2.1 Software Requirements.....	6
2.2 Hardware Requirements .....	6
3. Technology .....	7
3.1. Django.....	7
3.1.1. MVC Architecture .....	7
3.1.2. MTV framework.....	7
3.1.3. Forms .....	8
3.1.4. Settings .....	8
3.1.5. Admin .....	9
3.1.6. URL Routing .....	9
3.2. Bingo.....	10
3.2.1. Why Bingo as Your Molecular Search Cartridge?.....	10

3.2.2. Features.....	11
3.3. Indigo Toolkit.....	12
3.3.1. Indigo as Molecular SDK.....	12
3.3.2. Features.....	12
3.4. Kekule.js.....	13
3.4.1. Features.....	13
3.5. PostgreSQL.....	13
3.6. HTML.....	14
3.6.1 HTML5.....	14
3.6.2. Functions.....	15
3.7. CSS.....	16
3.7.1. CSS3.....	16
3.8. Bootstrap.....	18
3.8.1. Why Bootstrap.....	18
3.8.2. Bootstrap Package.....	18
3.8.3. Grid System.....	19
3.9. Python3.6.....	19
4. System Design.....	21
4.1 System Architecture.....	21
4.2 Introduction to UML.....	22
4.3 UML Diagrams.....	23
4.3.1 Use-Case Diagram.....	23
4.3.2 Activity Diagram.....	24
4.3.3 Sequence Diagram.....	26
5. Implementation.....	27
5.1 Authentication Module.....	27
5.1.1 Creating users.....	27
5.2.3 Limiting Access to Users.....	28
5.2.4 Cross Site Request Forgery (CSRF) Protection.....	29

5.2 Upload Module .....	30
5.3 Search Module .....	31
5.3.1 The Search Form .....	32
5.4 Report Module .....	33
5.5 Code Snippets .....	34
5.5.1 Models.py .....	34
5.5.2 Views.py – major functions .....	36
5.5.3 Urls.py .....	42
5.5.4 Search.html .....	43
6. Testing .....	45
6.1 Levels of Testing .....	45
6.1.1 Unit Testing .....	45
6.1.2 Integration Testing .....	45
6.1.3 System Testing .....	45
6.1.4 Acceptance Testing .....	46
6.1.5 Regression Testing .....	46
7. Screenshots .....	47
Conclusion .....	56
Bibliography .....	57

# **1. Introduction**

## **1.1. Chemical Databases**

Chemical databases are computer-aided molecular information storage systems, in chemo-informatics. They contain a wide range of information and features of the chemical compounds. They are often used for storing different types of structures of a chemical compound. The information stored is primarily obtained by experimentation and research. Chemical databases store physical, chemical and biological properties apart from vast information obtained through research. Chemical databases help scientists stay updated with the latest discoveries and inventions in the field of chemistry.

They emphasize on the need for a standardized chemical data that can be accessed at the onset of any research and development process. This in turn increases the efficiency and eases the workflow of research-oriented activities. Such chemical databases are much needed in today's world. They pave the way for radical drug discovery and testing of the drug. On the other hand, the availability of vast information at one place can lead to path breaking inventions in any field pertaining to chemistry. They have been instrumental in providing new opportunities in the development, application and knowledge discovery over the years. Simple visualization and representation of complex molecules eases the burden on scientists. They help in understanding the countless possible reactions with a single click. Through chemo informatics it is possible to analyze the available data to arrive at significant conclusions which otherwise would take a large amount of time. Chemo informatics ensures data accuracy for most part.

There are a variety of chemical databases which store peculiar data about the compounds. The most important among them is the structure database. It has become increasingly difficult for chemists to represent complex structures on paper. The development of structure based chemical databases has abated the burden drastically. The most important outcome of structure based chemical databases has been the 3-D structure representation. Other widely used types of chemical databases are literature database, crystallographic database, spectral databases and reaction databases.



## **1.2. Background**

The properties of molecules beyond their structure can be broadly classified into a wide range of categories based on the requirement. Over the years such information and categories have seen a voluminous increase, making it hard to handle such large-scale data. There came a need to make use of the computer servers. Thus, chemical databases have become very popular and widely used ever since they could be remotely accessed. This has led to many inventions and discoveries in numerous fields. Computer databases have made it easy to handle data of any scale.

### **1.2.1. Search**

The need to have a computerized database for chemical compounds has come from the difficulties in information retrieval through search. The search functionality is the heart of any chemical database. Given the variety of information a chemical compound has, the search has to be very efficient and accurate. The search functionality can comprise of different parameters like the structure search, IUPAC search, isomorphic search, similarity search etc.

### **1.2.2. Structure Representation**

The advent of chemical databases led to another interesting and time saving feature i.e. the chemical structure representation. It made it very simple to represent a chemical structure in different varieties. It has completely eliminated the need to have them drawn on a paper. Chemical databases are particularly different from other general-purpose databases in their support for sub-structure search. This kind of search is achieved by looking for sub graph isomorphism and is a widely studied application of Graph theory. Any chemical compound can be represented or drawn with the help of chemical structure editor tools. The structures can later be morphed into three dimensional structures. These editors take care of the atom-atom mapping, automatic coordinate generation, obtaining layouts and generating the SMILES id for every compound.

Thus, chemical databases have proved to be very handy in providing fast, scalable and efficient storage and searching solution for cheminformatics.

### **1.3. Project Overview**

At the Indian Institute of Chemical Technology (IICT), a research laboratory under the Council for Scientific and Industrial Research, there is a molecular bank consisting of information about nearly sixty thousand patented molecules developed in the IICT labs. This molecular bank is a database consisting of data predominantly in the form of excels sheets.

The scientists at IICT need a portal to interact with the database for storing, searching and accessing information about the chemical molecules. The portal helps them in research and drug discovery. Each molecule has a wide range of information associated with it. The information related to a molecule has close to fifty fields.

#### **1.3.1. Existing System**

The existing database has a portal that lets the authorized users to store, search and access information from the database. The existing portal has been developed by a private software development firm, which charges a certain amount annually for its maintenance. The existing portal has been developed using the .NET framework with MySQL as the relational database system.

This portal has proven to be highly inefficient for various technical reasons. The reasons listed below have been identified as the ones causing major trouble.

#### **1.3.2. Problems with the existing System**

##### *1.3.2.1. Structure Search Algorithm*

The search functionality which happens to be the core of any chemical database has to be very optimized and efficient. In the existing system the structure search for a particular molecule is based on image overlapping. The structure to be searched for is converted into an image and is overlapped with all the molecular structures present in the database making it very time consuming.

##### *1.3.2.2. User Interface*

The user interface is not on par with that of internationally renowned chemical database applications. The data retrieval has quite a lot of unhandled errors, which do not show up. After a particular type of search is performed the search results do not get displayed in the required format. The most important feature for any chemical database application is the chemical

structure editor. The editor on the existing system doesn't provide requirements such as copy/pasting a molecular structure, opening an existing molecular structure file and saving a structure in the required file format.

#### *1.3.2.3. Report Generation*

The search function provides with the relevant search results. There needs to be an option for generating a report for every molecule in the search results. The report typically consists of entire information about that molecule. This key feature is lacking in the existing system.

#### *1.3.2.4. User Authentication*

The admin needs to approve any data upload into the application. The data is primarily uploaded by three scientists consecutively, namely 1. The Chemist 2. The Biologist and 3. The Spectral Data scientist. This has to be finally verified and approved by the admin.

### **1.3.3. Proposed System**

The proposed system is a portal which has many extra features besides overcoming the above problems. The entire application is based on the Django web Framework with PostgreSQL as the relational database management system. An open source RDBMS cartridge called BINGO has been used on top of PostgreSQL for the ease of handling vast chemical functionalities. Django is an ideal web framework to use as its integration with other python modules for chemo-informatics is seamless.

The entire view of the proposed system is written in HTML 5 and is styled using CSS3 and Bootstrap3.6.7.

### **1.3.4. Scope of Proposed System**

The proposed system consists of a multilevel search functionality which is based on several parameters like SMILES, IUPAC, molecular weight, plate number, spectral values etc. This search has been further extended to the structure search.

The structure search is based on the SMILES generation through the BINGO cartridge. Simplified molecular-input line-entry system abbreviated as SMILES is a specification in form of a line notation for describing the structure of chemical species using short ASCII strings. Since every graph is usually stored in its depth first search traversal or a breadth first search traversal form, it is extremely optimal to base the search on them. This gives

a landslide victory over the image-based search algorithm in terms of efficiency and optimization.

The inbuilt authentication system of Django eliminates the need to handle authentication and authorization separately. It makes sure that a particular user has access to only specific functionalities which the user is entitled to.

The proposed system also eliminates the necessity to store the structure of the molecules in the database. To retrieve the structure each time for every molecule from the database is a very expensive operation. In the proposed system the structure is being rendered from the SMILES on the screen using KEKULE.JS, an open source JavaScript library to represent, draw, edit and compare molecular structures on the web-browser.

The other key feature that has been considered in the proposed system is the automatic email generator whenever an update in the database takes place. The four central users of the application receive an email whenever they update any data pertaining to the molecules.

The entire user interface is made very simple and user friendly for performing any kind of operations. The report generation functionality has been designed on par with the renowned chemical applications. The search results can be exported into an excel sheet for later reference. A standardized template is put in place for uploading any new information, thus making is very clean and efficient.

## **2. System Requirements**

### **2.1 Software Requirements**

- Supported Operating System: Windows 7 onwards (64-bit architecture)
- Supported Development Environment
  - Python 3.6
  - HTML 5
  - CSS 3
  - Bootstrap 3
- Required Web Framework: Django 2.0
- Database: PostgreSQL 10.
- RDBMS Cartridge: BINGO by EPAM.
- JavaScript Libraries: Kekule.js licensed by MIT.
- Script Editor: Sublime Text
- Web Server: Apache Lounge
- WSGI: Apache mod\_wsgi distribution

### **2.2 Hardware Requirements**

- Processor: Intel Core i7
- RAM: 8GB
- Hard Disk: 1TB

## 3. Technology

### 3.1. Django

Django is an open source web application written in python. It was designed to make common Web-Development and Application Development tasks fast, easy and in a simple understandable manner. Since the advent of the Internet, the best way to design a client-server application has been following the concept of MVC and Django is no exception.

#### 3.1.1. MVC Architecture

- The **Model** is a single, definitive source of information about your data. It contains the field information that the user wants to be in his database and all properties that are specific to the fields. Even the behaviors of the data that the user wants to store can be defined in models. Generally, each model corresponds to a single database table.
- The **View** is what the user sees on the screen when he enters the URL. It corresponds to the presentation layer for the model that you specify. View also provides the user an interface to input the user data.
- The **Controller** controls the flow of information between models and the views. The python scripts, that include the programmed logic, decide what information has to be pulled from the database with the help of model and what information is passed to the view. As mentioned earlier, it also has the ability to get information from the user via View and implement the business logic on it.

#### 3.1.2. MTV framework

Django follows the MVC pattern closely, however there are minor changes in the implementation as it uses its own logic. The Controller functionality is handled by the Django Framework itself so the entire Django model can now be conceptualized as MTV framework.

- The **Model**, responsible for the data access layer, contains all the information about the data, how to access it, how to validate it, relationships between the data entities and their behaviors.
- The **Template**, responsible for the presentation layer, contains presentation-related decisions: how information has to be handled on the Web page and how it should be displayed.

- The **View**, responsible for the business logic layer, acts as a bridge between models and templates and it contains the logic that accesses the model and defers to the appropriate template.

### 3.1.3. Forms

Django provides a rich framework to facilitate the creation, manipulation of forms and form data. The Django **Form** class is the heart of the system of components in Django's **Form** class. As we describe the structure of an object, its behavior, and the way it has to be represented in Model, a **Form** class describes a form and determines how it looks and works. Form's fields are themselves classes, they can be self-managed and self-validated.

### 3.1.4. Settings

A Django settings file consists of all the configuration of Django installation.

Some of the available configuration settings are:

- BUILD PATHS
  - SITE\_ROOT
  - BASE\_DIR
- SECURITY
  - ALLOWED\_HOSTS
  - SECRET\_KEY
  - DEBUG
- APPLICATION DEFINITION
  - INSTALLED\_APPS
  - MIDDLEWARE
  - ROOT\_URLCONF
  - TEMPLATES
  - WSGI\_APPLICATION
- DATABASE
  - DATABASES
- PASSWORD VALIDATION
  - AUTH\_PASSWORD\_VALIDATORS
- INTERNATIONALIZATION
  - LANGUAGE\_CODE
  - TIME\_ZONE
  - USE\_I18N

- USE\_L10N
  - USE\_TZ
- LOGS
  - LOGS\_ROOT
  - LOGGING
- STATIC
  - STATIC\_URL
  - STATIC\_ROOT
- EMAIL SETTINGS
  - EMAIL\_HOST\_USER
  - EMAIL\_HOST\_PASSWORD
  - EMAIL\_HOST
  - EMAIL\_USE\_TLS
  - EMAIL\_PORT

### 3.1.5. Admin

In Django, one of the most powerful parts is the automatic admin interface. Users can allow the admin to read the models metadata to provide quick, model-centric interface where only trusted users and admins can manage content that is hosted on the site. Admin tool is very useful when the application is under development stage as all the changes can be tracked, revoked with just few clicks. Users have to register the model with the `admins.py` so that the model can be monitored by admin.

### 3.1.6. URL Routing

A URL is a simple web address and it is visible in your browser's address bar. Every page on the internet has a URL associated with it. **URLS.PY** is a file in the project site folder in Django which has all the possible URL patterns that the application can allow.

```
urlpatterns = [
    path('', views.home, name='home'),
    path('/home', views.loadhome, name='loadhome'),
]
```

As you can see in the above code snippet, the first argument is the URL pattern and second argument is the response function associated with the URL. So, whenever a user enters the URL in the address bar, the corresponding views function is called and some actions are performed over there which are specified in the programmed logic and the response is brought



back to the web page in the form of HTTPResponse or a simple REDIRECT request.

## **3.2. Bingo**

Bingo is an open-source RDBMS data cartridge that provides fast, scalable, and efficient storage and searching solutions for chemical information.

Bingo can be easily integrated with almost all the database technologies like Oracle, Microsoft SQL Server, PostgreSQL, and MySQL. Bingo also provides extendible hashing on the data columns for fast retrieval and quick search.

Bingo sets the industry standard in structure and reaction registration and retrieval. It implements state-of-the-art indexing algorithms within the database server to allow fast and reliable chemical searching. Users who choose bingo to be their chemical database manager can seamlessly combine chemical substructures, reaction, and exact structure searching in their own understandable SQL terminology.

### **3.2.1. Why Bingo as Your Molecular Search Cartridge?**

Bingo has all the necessary features required by modern cheminformatics applications such as advanced tautomer search, resonance substructure search, and fast updating of the index when adding new structures. It provides an interface for the Oracle cost-based optimizer where possible, particularly in SMILES queries.

Bingo is reliable as it has no legacy code to maintain, and it is extensively tested.

Bingo is fast and has succeeded in achieving the best performance in the industry for a search cartridge for both the screening phase and the matching phase of various types of searches, especially substructure search. It has very effective memory management with no unnecessary re-allocations. Communication with the underlying database, especially LOB handling, is optimized as well. During substructure searches, molecules and reactions are stored in shared memory to speed up the access.

### 3.2.2. Features

- **Searches in a variety of ways on the molecule structures:** For molecule structure searching, Bingo supports 2D and 3D exact and substructure searches, as well as SMARTS searches. It also supports similarity, tautomer, Markush, formula, molecular weight, and flexmatch searches. Canonical SMILES, with isomeric information included, are available as well. For reaction searches, Bingo supports reaction substructure search (RSS) with the optional automatic generation of atom-to-atom mapping. All of these techniques are available through extensions to the SQL syntax.

A chemistry column in a table being searched can contain Molfile (V2000 or V3000) or SMILES data. Similarly, the query can be in Molfile or SMILES format.

Different indexing options are available to optimize storage space requirements versus the speed of registration. For example, by switching off the tautomer fingerprints, the indexing will become faster. Indexes can be updated or rebuilt with no downtime for maintenance. The addition of new molecules/reactions does not require the rebuilding of the index.

- **Flexibility and scalability of a true data cartridge:** Fully compliant with the Oracle and MS SQL Server, Bingo allows structure and reaction tables to be placed anywhere in the database, and the data can be accessed by any SQL compliant application.

Bingo supports a large database operation. It is successfully operating on the *PubChem database*, with up to 27 million structures.

- **Integration:** With Bingo, you can build a fully relational registration system: Insert, delete, and update records in a relational structure or reaction database using data cartridge technology in Oracle, or automatically generated triggers in MS SQL Server.

With Bingo, you can develop new applications that manage proprietary structure and reaction information. Since the cartridge is data model independent, it allows great flexibility in the design of applications and the management of proprietary structure and reaction information.

### 3.3. Indigo Toolkit

Indigo is an open-source, universal molecular toolkit that can be used for molecular fingerprinting, substructure search, and molecular visualization developed by EPAM. It is also capable of performing a molecular similarity search and provides enhanced stereochemistry support for end users.

Indigo is based on a cheminformatics library that incorporates a number of unique algorithms developed by EPAM, as well as some standard algorithms well-known in the cheminformatics world. Indigo is used by many corporations and institutions.

#### 3.3.1. Indigo as Molecular SDK

Indigo SDK allows developers of all platforms to seamlessly integrate into their application and all the problems such as loading binary modules, threading issues, stack overflows are taken care of by Indigo itself. Indigo SDK is highly configurable and extensible. You can write C/C++/C#/Java/Python plugins for it and distribute them independently.

#### 3.3.2. Features

- Input formats support: Molfiles/Rxnfiles v2000 and v3000, SDF, RDF, CML, SMILES, SMARTS.
- Portability: Pre-built binary packages are provided for Linux and Windows (both 32-bit and 64-bit), and also for Mac OS X systems (both 10.5 and 10.6).
- Automatic layout for SMILES-represented molecules and reactions.
- Canonical (isomeric) SMILES computation.
- Exact matching, substructure matching, SMARTS matching.
- Matching of tautomer and resonance structures.
- Molecule fingerprinting, molecule similarity computation.
- Fast enumeration of SSSR rings, subtrees, and edge subgraphs.
- Molecular weight, molecular formula computation.
- R-Group deconvolution and scaffold detection. Pioneer work in computing the exact maximum common substructure for an arbitrary amount of input structures.
- Combinatorial chemistry
- Plugins support in the API. As a reference, please see the Renderer plugin distributed together with the Indigo API.

### 3.4. Kekule.js

Kekule.js is an open source JavaScript library for chemo-informatics released under MIT license. Currently, it is molecule-centric, focusing on providing the ability to represent, draw, edit, compare and search molecule structures on web browsers. The whole library is divided into several modules, each provides different features and functions. Users can use the combination of modules for specific purpose.

#### 3.4.1. Features

- Core: Representation of chemical concepts such as atom, bond.
- IO: Read/write different format of chemical data. Now including:
  - Kekule JSON/XML format
  - CML
  - MDL MOL2000/3000
  - SMILES
- Render: Provides low-level cross-browser render methods to draw molecule (and other chemical objects) in web browser context
- Widget: Animation system to show/hide those widgets and actions associated with those widgets.
- Chem-Widget:
  - Periodical table
  - 2D or 3D viewer for molecules
- Algorithm:
  - Ring perception
  - Aromatic ring recognition
  - Molecule comparison
  - Sub-structure search

### 3.5. PostgreSQL

PostgreSQL is an open-source, powerful object-relational database system. It runs on all major operating systems and its proven strong architecture has earned it a lot of reputation for reliability, data integrity and correctness. It is fully ACID compliant. It includes most of the SQL data types, including INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, and TIMESTAMP.

PostgreSQL is highly customizable as it can run stored procedures in more than a dozen programming languages, including Java, Perl, Python,

Ruby, C/C++, and its own PL/pgSQL, which is similar to Oracle's PL/SQL. Triggers and stored procedures can be written in C and loaded into the database as a library, allowing great flexibility in extending its capabilities. Similarly, PostgreSQL includes a framework that allows developers to define and create their own custom data types along with supporting functions and operators that define their behavior.

Postgres also provides PGADMIN tool to allow modifications to the databases and tables through user interface. PGADMIN is the most popular and feature rich Open Source administration and development platform for PostgreSQL

### **3.6. HTML**

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript it forms a triad of cornerstone technologies for the World Wide Web. Web browsers receive HTML documents from a web server or from local storage and render them into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects, such as interactive forms, may be embedded into the rendered page. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as <img /> and <input /> introduce content into the page directly. Others such as <p>...</p> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

#### **3.6.1 HTML5**

HTML5 is the latest evolution of the standard that defines HTML. The term represents two different concepts. It is a new version of the language HTML, with new elements, attributes, and behaviors, and a larger set of technologies that allows the building of more diverse and powerful Web sites and applications. This set is sometimes called HTML5 & friends and often shortened to just HTML5.

HTML5 is a natural evolution of earlier versions of HTML and strives to reflect the needs of both current and future Web sites. It inherits the vast majority of features from its predecessors, meaning that if you coded HTML before HTML5 came on the scene. This also means that much of HTML5 works in both old and new browsers; being backward compatible is a key design principle of HTML5. HTML5 also adds a bevy of new features. Many are straightforward, such as additional elements (article, section, figure, and many more) that are used to describe content

Designed to be usable by all Open Web developers, this reference page links to numerous resources about HTML5 technologies, classified into several groups based on their function.

### 3.6.2. Functions

- **Semantics** – allowing you to describe more precisely what your content is.
- **Connectivity** – allowing you to communicate with the server in new and innovative ways.
- **Offline and storage** – allowing web pages to store data on the client-side locally and operate offline more efficiently.
- **Multimedia** – making video and audio first-class citizens in the Open Web.
- **2D/3D graphics and effects** – allowing a much more diverse range of presentation options.
- **Performance and integration** – providing greater speed optimization and better usage of computer hardware.
- **Device access** –allowing for the usage of various input and output devices.
- **Styling** – letting authors write more sophisticated themes.
- **Forms 2.0** –Improvements to HTML web forms where new attributes have been introduced for <input> tag.
- **Web Socket** – A next-generation bidirectional communication technology for web applications.
- **Server-Sent Events** – HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).

- **Canvas** – This supports a two-dimensional drawing surface that you can program with JavaScript.
- **Audio & Video** – You can embed audio or video on your webpages without resorting to third-party plugins.
- **Geolocation** – Now visitors can choose to share their physical location with your web application.
- **Microdata** – This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.
- **Drag and drop** – Drag and drop the items from one location to another location on the same webpage.

### 3.7. CSS

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable. It handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs, variations in display for different devices and screen sizes as well as a variety of other effects.

CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.

#### 3.7.1. CSS3

Cascading Style Sheets have taken a huge leap forward with its latest version, CSS3. The fact that the new version enhances the look of a design project, but that's just the beginning of what designers can do when using CSS3.

- **Compatible with older versions** – One of the best features of CSS3 is the fact that it is compatible with older versions of the language. Designers won't have to give up all their previous work with the predecessors of CSS3. The new language can be reworked on old modules too. However, there might be some speed issues during the conversion.
- **Simple and Independent** – Unlike CSS2 (which came as one big package), CSS3 is made of small modules which makes the application

easier and simpler to use. Selectors, Color, Box Model, Backgrounds and Borders, Text Effects, 2D/3D Transformations and user interface are some of the most useful modules that CSS3 offers.

- **View and Change Friendly** – Since it is broken up into small modules, it's simpler to change the parts individually without massively affecting the other components which don't require any major changes. Another great advantage is the fact that it is compatible with all kinds of platforms and can be viewed with similar ease in mobile as well as PCs and tablets.
- **Speedy Development** – Being an independent language which is not dependent on JavaScript, CSS3 loads a lot faster than its precursors. The language is pretty well compatible with all the browsers available. The individual modules also help in saving a lot of time during the development, implementation and end of production.
- **Platform Independent and Cross Browser Compatible** – When it comes to being platform independent, nothing can beat CSS3. It takes independence to a new level. It also offers designers with special processes which allow them to design components in a very easy manner. Although not under the W3C standards, CSS3 shows compatibility with most of the browsers available which helps because the end users will have a huge range of browsers to choose from.
- **Attractive Backgrounds** – Backgrounds are made exciting by the help of CSS3. It allows designers to choose from multiple backgrounds which can be applied with ease. The best part is that the background can be resized to fit the requirements of the project.
- **Borders and Texts** – CSS3 allows designers to apply beautiful borders on the page or site they are creating. This helps in enhancing the appearance of the site. It also allows the application of various texts for the content. This helps in highlighting important information.
- **Images and Animations** – Aside from easy coding, one of the best qualities of CSS3 is that it helps in enhancing the look of a website or pages immensely. It allows easy integration of the various images, which includes 3D, in the project. It also allows easy inclusion of videos and animations as well. Customization of the images can also be done with ease.



- **Testing the Features** – Previous versions required a lot of time to test to detect issues and bugs. But with CSS3's smaller modules, testing of the individual components becomes easy. Users can then integrate and run a quick compatibility test of all the parts individually. This saves a lot of time and hassle. This in turn requires lower turn out time and client satisfaction.
- **Easy maintenance** – To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.

### 3.8. Bootstrap

Bootstrap is the most popular front-end framework in the recent time. It is sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development. It uses HTML, CSS and JavaScript.

#### 3.8.1. Why Bootstrap

- **Mobile first approach** – Bootstrap 3, framework consists of Mobile first styles throughout the entire library instead them of in separate files.
- **Browser Support** – It is supported by all popular browsers.
- **Easy to get started** – With just the knowledge of HTML and CSS anyone can get started with Bootstrap. Also, Bootstrap's official site has good documentation.
- **Responsive design** – Bootstrap's responsive CSS adjusts to Desktops, Tablets and Mobiles.

It provides a clean and uniform solution for building an interface for developers and contains beautiful and functional built-in components which are easy to customize. It also provides web-based customization and best of all it is an open source.

#### 3.8.2. Bootstrap Package

- **Scaffolding** – Bootstrap provides a basic structure with Grid System, link styles, and background.
- **CSS** – Bootstrap comes with the feature of global CSS settings, fundamental HTML elements styled and enhanced with extensible classes, and an advanced grid system.

- **Components** – Bootstrap contains over a dozen reusable components built to provide iconography, dropdowns, navigation, alerts, pop-overs, and much more.

### 3.8.3. Grid System

In graphic design, a grid is a structure made up of a series of interesting straight lines used to structure the content. It is widely used to design layout and content structure in print design. In web design, it is a very effectively using HTML and CSS.

The Bootstrap grid system has four classes:

- xs (for phones)
- sm (for tablets)
- md (for desktops)
- lg (for larger desktops)

The classes above can be combined to create more dynamic and flexible layouts.

## 3.9. Python3.6

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Website and may be freely distributed. The same site also contains distributions of and pointers to many free third-party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas the other languages use punctuations. It has fewer syntactical constructions than other languages

Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

**Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

**Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects

**Python is a Beginner's Language:** Python is a great language for the beginner level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

## 4. System Design

### 4.1 System Architecture

The interaction between the web application, database, and client are shown in the figure below.

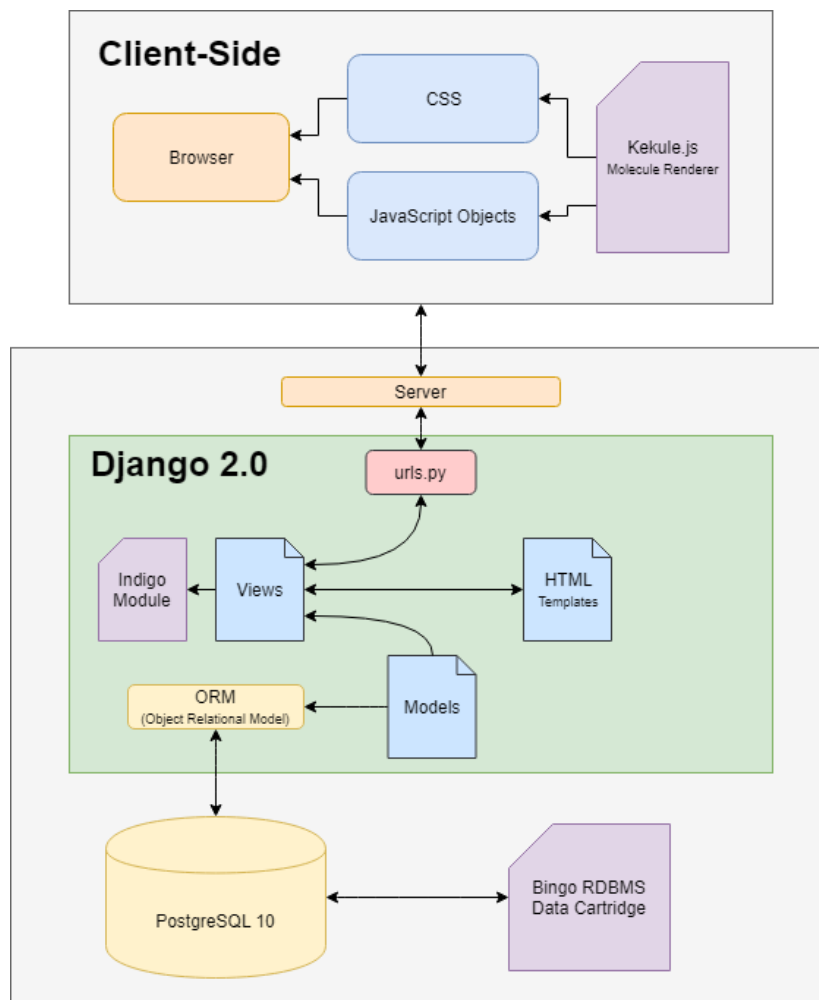


Figure 1: System Architecture

When the client makes a request for a substructure search based on a query structure, for instance, then the server directs the request based on its URL in *urls.py* to the corresponding controller function in *views.py*. The controller then generates an SQL query based on the preferences fetched using the POST request and queries the table through Django's Object Relational Model (ORM). The structure of the table is defined in *models.py*. The ORM

retrieves the results into an object in *views.py*, which then generates the 2D coordinates for each result molecule using the *Indigo Module*. Finally, *views.py* generates an *HTML Template file* to be rendered on to the client's browser. *Kekule.js* renders the 2D coordinates received through a context variable into a *Kekule* widget.

This process is better elucidated using UML Diagrams.

## 4.2 Introduction to UML

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.

The creation of UML was originally motivated by the desire to standardize the disparate notational systems and approaches to software design. It was developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in 1994–1995, with further development led by them through 1996.

UML offers a way to visualize a system's architectural blueprints in a diagram, including elements such as:

- any activities (jobs);
- individual components of the system;
- and how they can interact with other software components;
- how the system will run;
- how entities interact with others (components and interfaces);
- external user interface.

Although originally intended for object-oriented design documentation, UML has been extended to a larger set of design documentation (as listed above), and been found useful in many contexts.

### *Software development methods*

UML is not a development method by itself; however, it was designed to be compatible with the leading object-oriented software development methods of its time, for example OMT, Booch method, Objectory and especially RUP that it was originally intended to be used with when work began at Rational Software.

## *Modeling*

It is important to distinguish between the UML model and the set of diagrams of a system. A diagram is a partial graphic representation of a system's model. The set of diagrams need not completely cover the model and deleting a diagram does not change the model. The model may also contain documentation that drives the model elements and diagrams (such as written use cases).

UML diagrams represent two different views of a system model:

- Static (or structural) view: emphasizes the static structure of the system using objects, attributes, operations and relationships. It includes class diagrams and composite structure diagrams.
- Dynamic (or behavioral) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

## **4.3 UML Diagrams**

### **4.3.1 Use-Case Diagram**

Use case diagrams are a set of use cases, actors, and their relationships. They represent the use case view of a system. A use case represents a particular functionality of a system. Hence, use case diagram is used to describe the relationships among the functionalities and their internal/external actors.

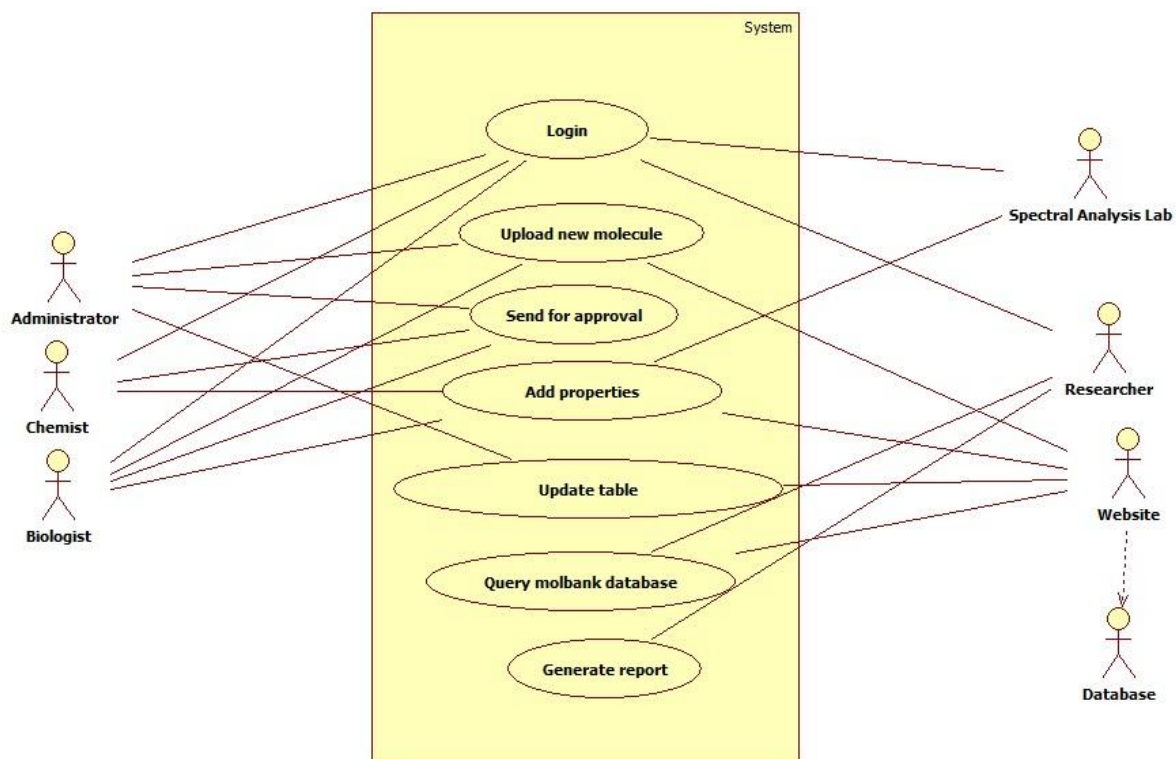


Figure 2: Use Case Diagram

### 4.3.2 Activity Diagram

Activity diagram describes the flow of control in a system. It consists of activities and links. The flow can be sequential, concurrent, or branched. Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system. Activity diagrams are used to visualize the flow of controls in a system. This is prepared to have an idea of how the system will work when executed.

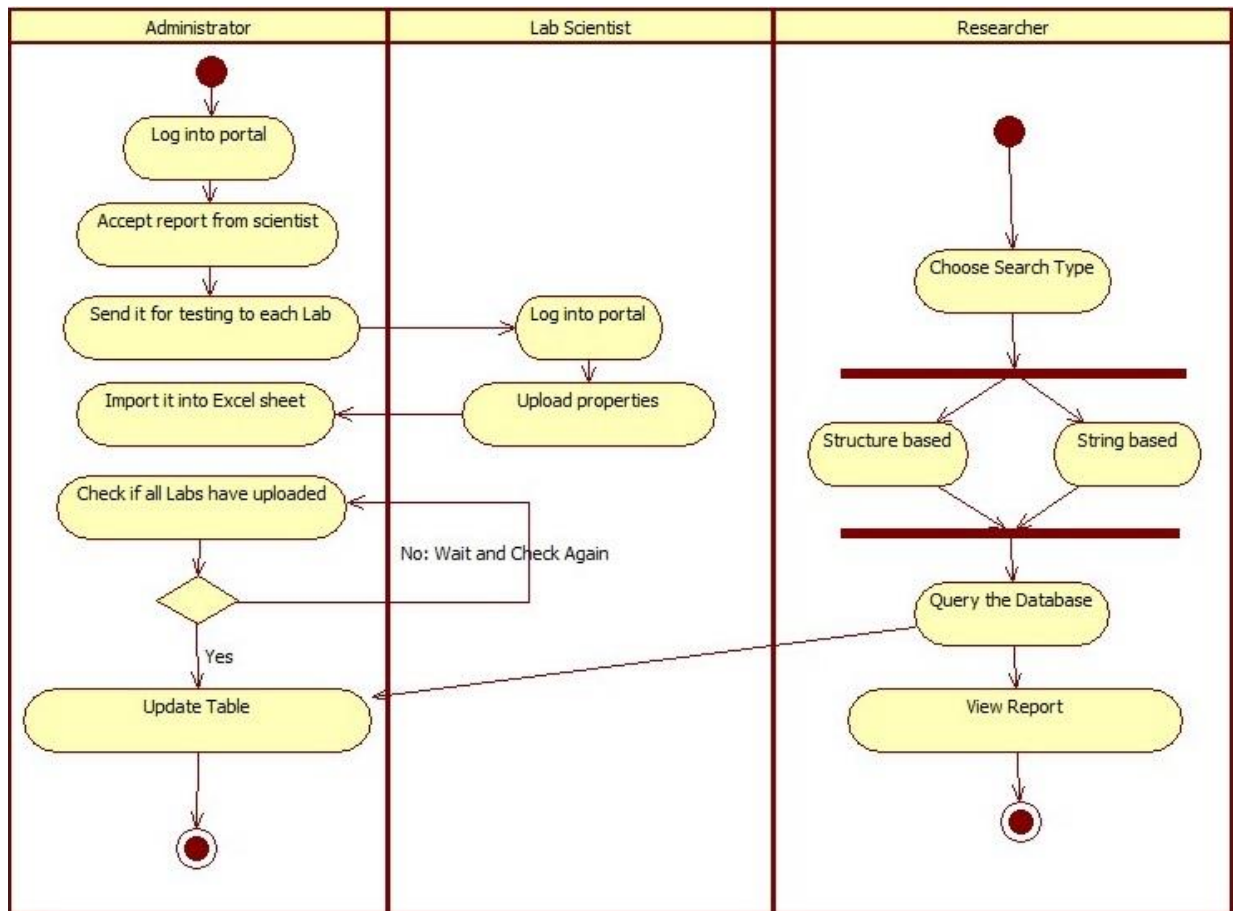


Figure 3: Activity Diagram with Swimlanes



### 4.3.3 Sequence Diagram

A sequence diagram is an interaction diagram. From the name, it is clear that the diagram deals with some sequences, which are the sequence of messages flowing from one object to another. Interaction among the components of a system is very important from implementation and execution perspective. Sequence diagram is used to visualize the sequence of calls in a system to perform a specific functionality.

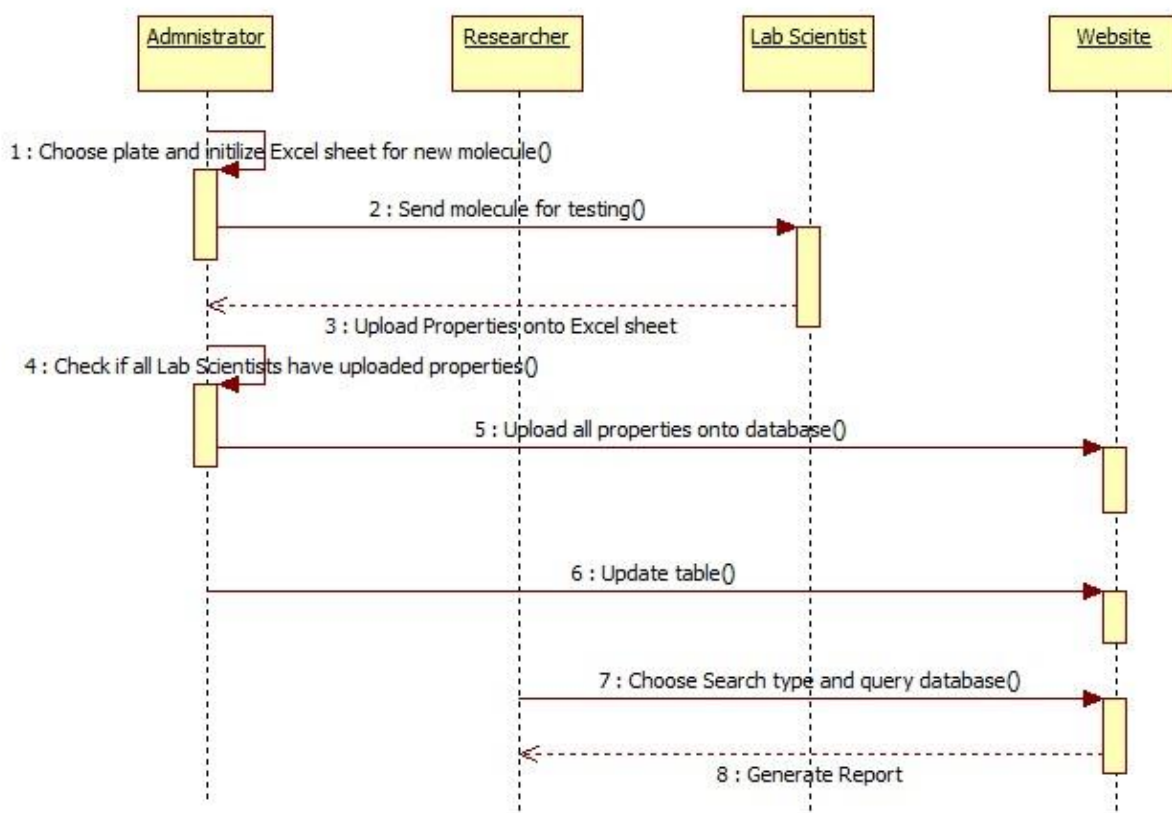
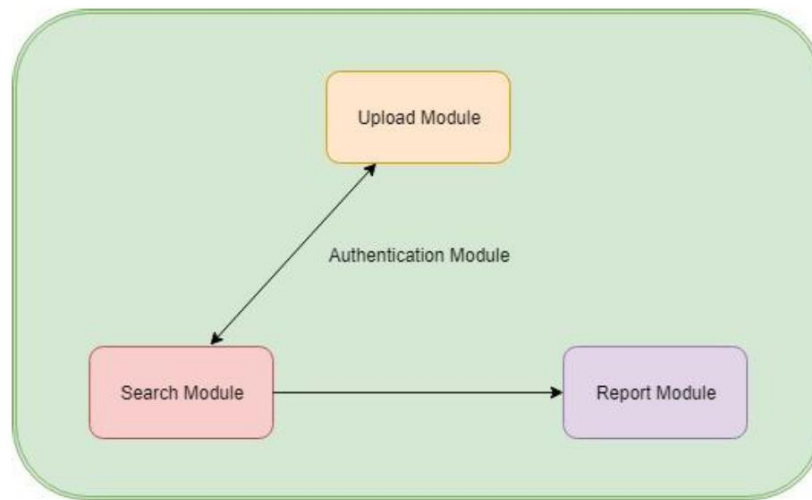


Figure 4: Sequence Diagram

## 5. Implementation

The entire web application can be broadly divided into four basic modules:

1. Authentication module
2. Upload module
3. Search module
4. Report module



*Figure 5: Module Architecture*

### 5.1 Authentication Module

This module contains rules for logging in and for accessing pages based on user permissions.

#### 5.1.1 Creating users

Django allows users to create an admin user using the `createsuperuser` command as below:

```
$ python manage.py createsuperuser
```

Once the admin user is created, the admin can log into the Django Administration page while the server is running the application. This page can be used to monitor the tables available in the `models.py` page in a user-friendly GUI. Also, the admin can create new permissions, and assign these permissions each time a new user is created.

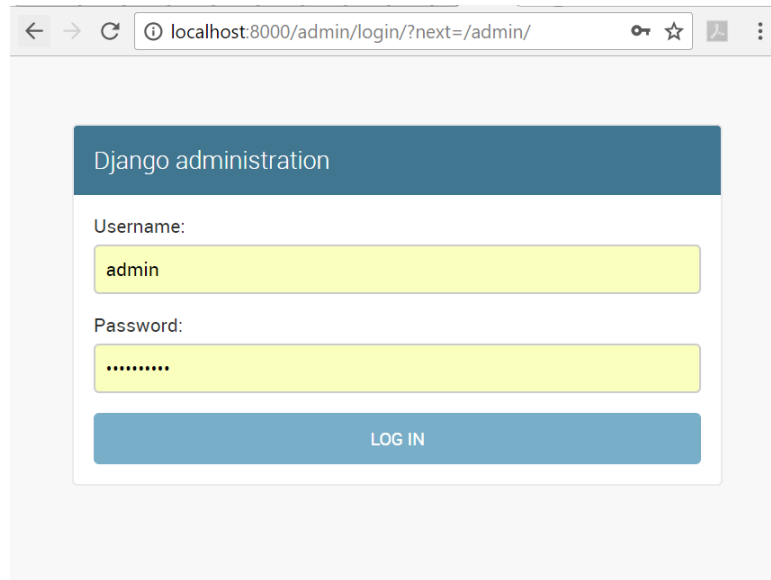


Figure 6: Django Administration Page

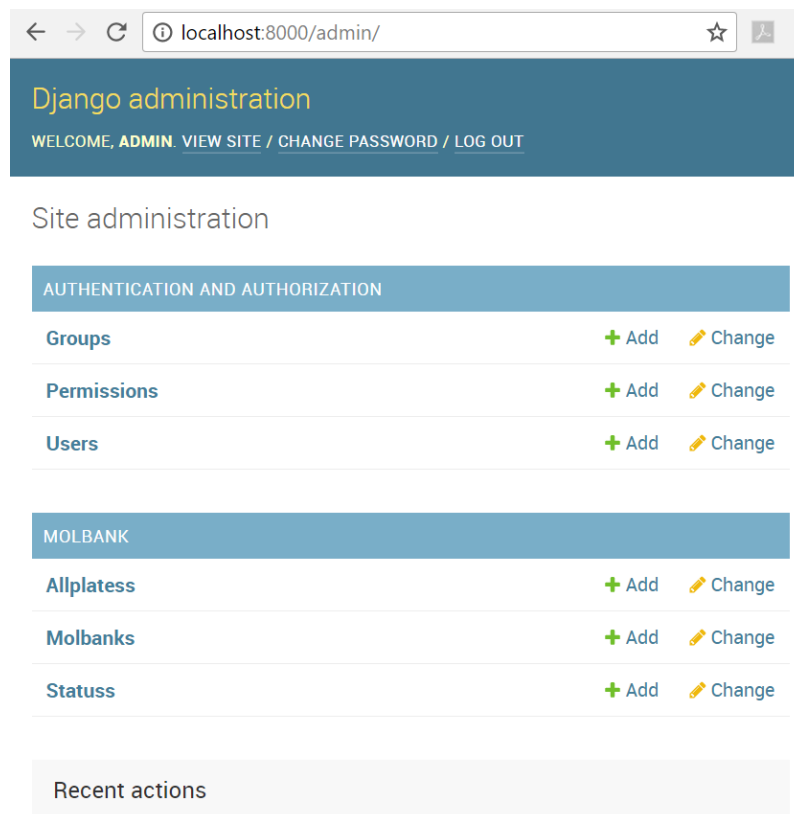


Figure 7: UI based control over users and tables in Models.py

### 5.2.3 Limiting Access to Users

In order to ensure that a mere change of URL does not direct a user to a page that they are not permitted to view, Django introduce the

`@login_required()` decorator. We can also display a particular response page, based on the permissions of the user.

```
@login_required()
def logoutHTML(request):
    logout(request)
    return redirect("login")

@login_required()
def uploadCSV(request):
    sample_func(request)
    if request.method == 'POST' and request.POST.get('myfile')!='' and
request.FILES['myfile']:
        myfile = request.FILES['myfile']
        if request.user.has_perm('molbank.is_admin'):
            fs = FileSystemStorage()
            if fs.exists(myfile):
                print(myfile, " updated")
                fs.delete(myfile)
            else:
                f = status.create(myfile.name)
                f.save()
            filename = fs.save(myfile.name, myfile)
            uploaded_file_url = fs.url(filename)
            return render(request, 'molbank/uploadCSV.html',
                {'uploaded_file_url': '/molbank' + uploaded_file_url,
'pill': 'uploadCSV', 'username': request.user.username})
```

Thus, the application is secure from unauthorized users.

#### 5.2.4 Cross Site Request Forgery (CSRF) Protection

CSRF attacks allow a malicious user to execute actions using the credentials of another user without that user's knowledge or consent. Django has built-in protection against most types of CSRF attacks. CSRF protection works by checking for a secret in each POST request. This ensures that a malicious user cannot simply “replay” a form POST to your website and have another logged in user unwittingly submit that form. The malicious user would have to know the secret, which is user specific (using a cookie).

When deployed with HTTPS, `CsrfViewMiddleware` will check that the HTTP referer header is set to a URL on the same origin (including subdomain and port). Because HTTPS provides additional security, it is

imperative to ensure connections use HTTPS where it is available by forwarding insecure connection requests and using HSTS for supported browsers. A CSRF token is used as shown below:

```
<form name="myform" method="POST" action="/molbank/advancedSearch"
enctype="multipart/form-data">
    {% csrf_token %}
```

## 5.2 Upload Module

This module contains methods that handle requests to add/update files to the server filesystem. It also contains automatic email generator to inform the users that a change has been made to the uploaded files. The upload module allows the admin to upload a file into the database so that the other users can update and reupload it. It keeps track of each upload using a model in *models.py*. The function that updates the database on final upload is given below:

```

@login_required()
def updateDB(request):
    if request.method == 'POST' and request.FILES['myfileDB']:
        myfileDB = request.FILES['myfileDB']
        if request.user.has_perm('molbank.is_admin'):
            conn = psycopg2.connect("host=localhost dbname=vinay user=postgres
password=Sasi1Harish2!")
            cur = conn.cursor()
            cur.execute('SET search_path TO bingo')
            og_name = myfileDB.name
            name = og_name.replace(' ', '_')
            name = name.replace('-', '_')
            fs = FileSystemStorage()
            filename = fs.save(name, myfileDB)
            f = open(os.path.join(settings.MEDIA_ROOT, filename))
            cur.copy_expert(sql="""COPY molbank_allplates FROM STDIN WITH CSV
HEADER DELIMITER AS ','""", file=f)
            conn.commit()
            cur.close()
            print("og_name=", og_name.strip('.csv')+'.xlsx')
            obj = status.objects.filter(name=str(og_name.strip('.csv')+'.xlsx'))
            print(obj)
            if obj is not None:
                obj.delete()
                print(obj, "deleted.")
            print(obj)
            all = status.objects.all()
            conn.closed
            return render(request, 'molbank/AdminUploadCSV.html',
                           {'flag': True, 'pill': 'updateDB', 'allstat':
all, 'fname': myfileDB.name, 'username': request.user.username})
            return render(request, 'molbank/uploadCSV.html', {'error': 'You are not
authorized to upload files', 'pill': 'updateDB', 'username': request.user.username})

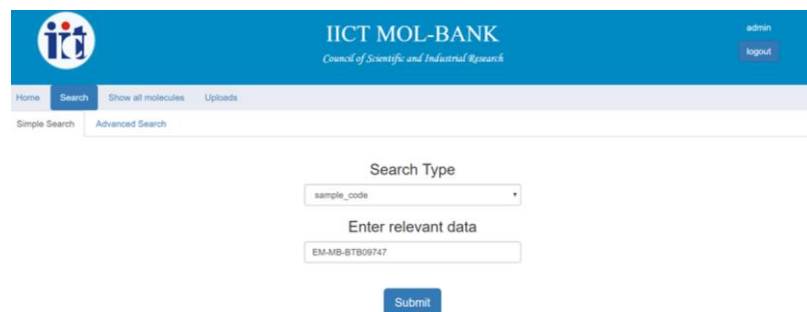
```

### 5.3 Search Module

This module contains queries, UI, and request handling methods to perform structure search. There are two types of searches that we have enabled:

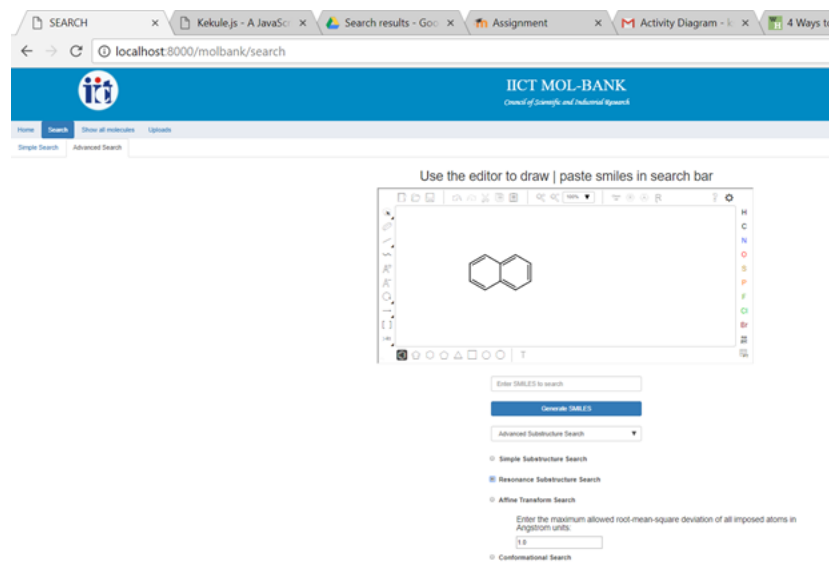
- Simple Search: Enables users to search the database using a string such as Formula, SMILES, Barcode, Sample code, etc.
- Advanced Search: Enables users to search by drawing a structure of the chemical module on a molecule sketcher.

### 5.3.1 The Search Form



The screenshot shows the 'Simple Search' form on the IICT MOL-BANK website. The header is blue with the IICT logo and the text 'IICT MOL-BANK Council of Scientific and Industrial Research'. Below the header, there are navigation links: 'Home', 'Search' (highlighted), 'Show all molecules', and 'Uploads'. Under 'Search', there are two tabs: 'Simple Search' and 'Advanced Search'. The main form area has a 'Search Type' dropdown menu set to 'sample\_code'. Below it is a text input field labeled 'Enter relevant data' containing the text 'EM-MB-BTB09747'. At the bottom of the form is a blue 'Submit' button.

Figure 8: Simple Search Form



The screenshot shows the 'Advanced Search' form on the IICT MOL-BANK website. The header is the same as in Figure 8. The 'Advanced Search' tab is selected. The main form area has a text input field labeled 'Enter SMILES to search' containing the text 'c1ccc2ccccc2c1'. Below this is a blue 'Generate SMILES' button. Underneath is a dropdown menu labeled 'Advanced Substructure Search'. Below the dropdown are four radio buttons: 'Simple Substructure Search', 'Resonance Substructure Search' (selected), 'Affine Transform Search', and 'Conformational Search'. Below the radio buttons is a text input field labeled 'Enter the maximum allowed root-mean-square deviation of all imposed atoms in Angstrom units:' containing the text '0.5'.

Figure 9: Advanced Search Form

Advanced search is implemented using JavaScript code to retrieve the SMILES ID of the molecule:

```
function getFromKetcher(){
    var ketcherFrame = document.getElementById('encoder_iframe');
    console.log('ketcher frame entered xD',ketcherFrame);
    var ketcher = null;

    if ('contentDocument' in ketcherFrame)
        ketcher = ketcherFrame.contentWindow.ketcher;
    else
        ketcher = document.frames['ifKetcher'].window.ketcher;
    mol=ketcher.getSmiles();
    document.getElementById('text_smiles').value=mol;
}
```

The search controller generates the query string, once the user fills up the form, as follows:

```
mollist: <RawQuerySet: select "ID", "SMILES", "SAMPLE_CODE", bingo.Molfile("SMILES") as structure from molbank_molbank
where "SMILES" operator(public.@) ('C1=CC=CC2=CC=CC=C12',''):bingo.sub ORDER BY getsimilarity("SMILES",'C1=CC=CC2=CC=CC
=C12','Tanimoto')>
```

Figure 10: Raw Query after translation by Django

## 5.4 Report Module

The report module makes a query to the database to retrieve all columns of the molecule selected in a list of search results.

```
@login_required()
def report(request, report):
    conn = psycopg2.connect("host=localhost dbname=vinay user=postgres
password=Sasi1Harish2!")
    cur = conn.cursor()
    cur.execute('SET search_path TO bingo')
    cur.execute("select * from molbank_allplates where sample_code =
'%s'"%str(report))
    mol = namedtuplefetchall(cur)
    indigo = Indigo()
    mol_obj = indigo.loadMolecule(mol[0].Smiles)
    mol_obj.layout()
    line = mol_obj.cml()
    line = line.replace('\n', '')
    cml = line
    conn.closed
    print(mol[0])
    return render(request, 'molbank/report.html', {'report': report, 'mol':
mol[0], 'cml': cml, 'col_list': col_list, 'username':request.user.username})
```



## 5.5 Code Snippets

### 5.5.1 Models.py

```
from django.db import models

# Create your models here.
class molbank(models.Model):
    ID = models.CharField(default='DUMMY_ID',primary_key=True,max_length=10)
    SMILES = models.CharField(default='DUMMY_SMILES',max_length=150)
    SAMPLE_CODE = models.CharField(default='DUMMY_SAMPLE_CODE',max_length=20)
    class Meta:
        permissions = (
            ('is_admin','MB: ADMIN'),
            ('is_chem','MB: CHEM'),
            ('is_bio','MB: BIO'),
            ('is_slab','MB: SLAB'),
        )
class status(models.Model):
    name = models.CharField(default='NoName', max_length = 100)
    chemist = models.BooleanField(default=False)
    biologist = models.BooleanField(default=False)
    spectral = models.BooleanField(default=False)

    @classmethod
    def create(cls, name):
        file = cls(name=name)
        return file

class allplates(models.Model):
    Project = models.CharField(max_length=4)
    Position_in_Array = models.CharField(null=True, blank=True, max_length=4)
    sub_project = models.CharField(null=True, blank=True, max_length=4)
    MTP_Plate_No = models.CharField(null=True, blank=True, max_length=4)
    Structure = models.CharField(null=True, blank=True, max_length=1)
    Solution_PT_Barcode = models.IntegerField(null=True, blank=True)
    Array_PT_Barcode = models.CharField(null=True, blank=True, max_length=10)
    Tray_PT_Barcode = models.IntegerField(null=True, blank=True)
    Neat_sample_Barcode = models.IntegerField(null=True, blank=True)
    Array_NS_Barcode = models.IntegerField(null=True, blank=True)
    Tray_NS_Barcode = models.IntegerField(null=True, blank=True)
    Smiles = models.CharField(max_length=500)
```

```

Sample_code = models.CharField(db_column='sample_code', primary_key=True, max_length=20)
Molecular_Weight = models.DecimalField(decimal_places=3, max_digits=10)
Molecular_Formula = models.CharField(max_length=500)
CLogP = models.DecimalField(decimal_places=3, max_digits=10)
Name = models.CharField(max_length=500)
NMR = models.DecimalField(null=True, blank=True, decimal_places=3, max_digits=10)
Mass = models.DecimalField(null=True, blank=True, decimal_places=3, max_digits=10)
HPLC_or_LCMS = models.DecimalField(null=True, blank=True, decimal_places=3, max_digits=10)
Solubility = models.DecimalField(null=True, blank=True, decimal_places=3, max_digits=10)
No_of_Rings = models.IntegerField()
No_of_nitrogens = models.IntegerField()
No_of_Oxygens = models.IntegerField()
TPSA = models.DecimalField(decimal_places=3, max_digits=10)
No_of_Sulphur = models.IntegerField()
Sample_Weight_submitted = models.CharField(max_length=10)
Purity = models.DecimalField(null=True, blank=True, decimal_places=3, max_digits=10)
IUPAC_Name = models.CharField(null=True, blank=True, max_length=500)
Date_of_approval = models.DateField(null=True, blank=True)
Date_of_Submission = models.DateField(null=True, blank=True)
Scientist_name = models.CharField(max_length=30)
Reference_syn_or_bio = models.CharField(null=True, blank=True, max_length=4)
Known_bio_activity = models.CharField(null=True, blank=True, max_length=4)
Natural_or_Synthetic = models.CharField(null=True, blank=True, max_length=4)
Physical_state = models.CharField(null=True, blank=True, max_length=4)
Sample_in = models.CharField(null=True, blank=True, max_length=4)
Sample_out = models.CharField(null=True, blank=True, max_length=4)
Miscellaneous = models.CharField(null=True, blank=True, max_length=4)
Optical_Rotation = models.CharField(null=True, blank=True, max_length=4)
Institute = models.CharField(max_length=10)

```

## 5.5.2 Views.py – major functions

```
@login_required()
def simpleSearch(request):
    if request.method == 'POST':
        form1 = simpleSearchForm(request.POST)
        print(form1.is_valid())
        print(form1)
        if form1.is_valid():
            search_type=request.POST.get('simple_search_type')
            var = str(form1.cleaned_data['data'])
            conn = psycopg2.connect("host=localhost dbname=vinay user=postgres
password=SasiHarish2!")
            cur = conn.cursor()
            cur.execute('SET search_path TO bingo')
            if int(search_type) == 5:
                cur.execute("select \"Smiles\", sample_code from molbank_allplates where
\"Solution_PT_Barcode\" = '"+var+"'")
            elif int(search_type) == 6:
                cur.execute("select \"Smiles\", sample_code from molbank_allplates where
\"Neat_sample_Barcode\" = '"+var+"'")
            elif int(search_type) == 3:
                cur.execute("select \"Smiles\", sample_code from molbank_allplates where
sample_code = '"+var+"'")
            elif int(search_type) == 4:
                cur.execute("select \"Smiles\", sample_code from molbank_allplates where
\"Molecular_Formula\" = '"+var+"'")
            elif int(search_type) == 1:
                cur.execute("select \"Smiles\", sample_code from molbank_allplates where
\"Smiles\" = '"+var+"'")
            elif int(search_type) == 2:
                cur.execute("select \"Smiles\", sample_code from molbank_allplates where
\"Name\" = '"+var+"'")
            molecule_list = namedtuplefetchall(cur)
            cml_dict = {}
            indigo = Indigo()
            for mol in molecule_list:
                mol_obj = indigo.loadMolecule(mol.Smiles)
                mol_obj.layout()
                line = mol_obj.cml()
                line = line.replace('\n','')
                cml_dict[mol.sample_code] = line
            conn.closed
            return render(request, 'molbank/search_results.html', {'molecule_list':
molecule_list, 'cml_dict': cml_dict, 'mols': molecule_list, 'username': request.user.username})
            return render(request, 'molbank/search_results.html', {'username': request.user.username})
```

```

@login_required()
def advancedSearch(request):
    if request.method == 'POST' or request.GET.get('page'):
        form1 = advancedSearchForm(request.POST)
        # print(form1.is_valid())
        conn = psycopg2.connect("host=localhost dbname=vinay user=postgres
password=SasiHarish2!")
        cur = conn.cursor()
        cur.execute('SET search_path TO bingo')
        if form1.is_valid():
            var = str(form1.cleaned_data['text_smiles'])
            search_type=request.POST.get('search_type')
            exact = {
                '1': 'ELE',
                '2': 'MAS',
                '3': 'STE',
                '4': 'FRA',
            }
            similarity = {
                '1': 'Tanimoto',
                '2': 'Tversky',
                '3': 'Euclid-Sub',
            }
            parameter = ''
            func = ''
            if int(search_type) == 1 or int(search_type) == 2:
                SubStructureSearch_type = request.POST.get('SubStructureSearch_type')
                if int(SubStructureSearch_type) == 2:
                    parameter = 'RES'
                elif int(SubStructureSearch_type) == 3:
                    rms = request.POST.get('rmsa')
                    parameter = 'AFF ' + rms
                elif int(SubStructureSearch_type) == 4:
                    rms = request.POST.get('rmsc')
                    parameter = 'CONF ' + rms
                func = "('" + var + "', '" + parameter + "')::bingo.sub"

            elif int(search_type) == 3:
                selected_list = form1.cleaned_data['ExactSearch_type']
                flag = 0

```

```

        for string in selected_list:
            flag = 1
            parameter = parameter + exact[string] + ' '
            if flag == 0:
                parameter = 'NONE'
            func = "(" + var + ",'" + parameter + "'):bingo.exact"
    elif int(search_type) == 4:
        SimilaritySearch_type = request.POST.get('SimilaritySearch_type')
        parameter = similarity[SimilaritySearch_type]
        min_sim = request.POST.get('min_sim')
        func = "(" + min_sim + ", null, '" + var + ",'" + parameter + "'):bingo.sim"
    elif int(search_type) == 5:
        TautomerSearch_Switch = request.POST.get('TautomerSearch_Switch')
        parameter = 'TAU'
        selected_list = form1.cleaned_data['TautomerSearch_type']
        for string in selected_list:
            parameter = parameter + ' R' + string
            if int(TautomerSearch_Switch) == 1:
                func = "(" + var + ",'" + parameter + "'):bingo.sub"
            elif int(TautomerSearch_Switch) == 1:
                func = "(" + var + ",'" + parameter + "'):bingo.exact"
    #print(form1.cleaned_data['ExactSearch_type'])
    cur.execute("select sample_code, \"Smiles\", \"IUPAC_Name\" from molbank_allplates
where \"Smiles\" operator(public.%) %s ORDER BY getsimilarity(\"Smiles\",'%s','Tanimoto')\" %
(func,var))
    molecule_list = namedtuplefetchall(cur)
    cml_dict = {}
    indigo = Indigo()
    for mol in molecule_list:
        mol_obj = indigo.loadMolecule(mol.Smiles)
        mol_obj.layout()
        line = mol_obj.cml()
        line = line.replace('\n','')
        cml_dict[mol.sample_code] = line
    paginator = Paginator(molecule_list,12)
    page = request.GET.get('page')
    mols = paginator.get_page(page)
    request.session['func'] = func
    request.session['var'] = var
    conn.closed

```

```

return render(request, 'molbank/search_results.html', {
    'molecule_list': molecule_list,
    'cml_dict': cml_dict,
    'username': request.user.username,
    'mols': mols, 'flag': '1',
})

elif request.GET.get('page') and 'func' in request.session:
    func = request.session['func']
    var = request.session['var']
    cur.execute("select sample_code, \"Smiles\", \"IUPAC_Name\" from molbank_allplates
where \"Smiles\" operator(public.%) %s ORDER BY getsimilarity(\"Smiles\", '%s', 'Tanimoto')\" %
(func, var))
    molecule_list = namedtuplefetchall(cur)
    cml_dict = {}
    indigo = Indigo()
    for mol in molecule_list:
        mol_obj = indigo.loadMolecule(mol.Smiles)
        mol_obj.layout()
        line = mol_obj.cml()
        line = line.replace('\n', '')
        cml_dict[mol.sample_code] = line
    paginator = Paginator(molecule_list, 12)
    page = request.GET.get('page')
    mols = paginator.get_page(page)
    conn.close()
    return render(request, 'molbank/search_results.html', {'molecule_list':
molecule_list, 'cml_dict': cml_dict, 'username': request.user.username, 'mols': mols,})
    conn.close()
    return render(request, 'molbank/search_results.html', {'username': request.user.username})

def handler404(request, exception, template_name='404.html'):
    response = render_to_response('404.html', {},
                                context_instance=RequestContext(request))
    response.status_code = 404
    return response

def handler500(request, exception, template_name='500.html'):
    response = render_to_response('500.html', {},
                                context_instance=RequestContext(request))
    response.status_code = 500
    return response

```

```

@login_required()
def status_of_pending_files(request):
    if request.user.has_perm('molbank.is_admin'):
        user = 1
        #permissions
        print("PERMISSIONS:      { ")
        pprint.pprint(request.user.get_all_permissions())
        print("                  }")
        fs = FileSystemStorage()
        var = fs.listdir('')
        #listing of files
        print("FILES IN DIR :   { ")
        pprint.pprint(var[1])
        print("                  }")
        all = status.objects.all()
        return render(request, 'molbank/AdminUploadCSV.html', {'flag_pending': False, 'pending':
var[1], 'flag': True, 'pill': 'uploadCSV', 'allstat': all,
'user': user, 'username': request.user.username}) #flag is used for checking whether there are
any pending files or not
    elif request.user.has_perm('molbank.is_chem'):
        user = 2
        print("PERMISSIONS:      { ")
        pprint.pprint(request.user.get_all_permissions())
        print("                  }")
        fs = FileSystemStorage()
        var = fs.listdir('')
        # listing of files
        print("FILES IN DIR :   { ")
        pprint.pprint(var[1])
        print("                  }")
        all = status.objects.all()
        return render(request, 'molbank/uploadCSV.html',
                        {'flag_pending': False, 'pending': var[1], 'flag': True, 'pill':
'uploadCSV', 'allstat': all, 'user': user, 'username': request.user.username})
    elif request.user.has_perm('molbank.is_bio'):
        user = 3
        print("PERMISSIONS:      { ")
        pprint.pprint(request.user.get_all_permissions())
        print("                  }")
        fs = FileSystemStorage()
        var = fs.listdir('')
        # listing of files
        print("FILES IN DIR :   { ")
        pprint.pprint(var[1])
        print("                  }")

```

```

    all = status.objects.all()
    return render(request, 'molbank/uploadCSV.html',
                  {'flag_pending': False, 'pending': var[1], 'flag': True, 'pill':
'uploadCSV', 'allstat': all, 'username': request.user.username})
    elif request.user.has_perm('molbank.is_slab'):
        user = 4
        print("PERMISSIONS:      { ")
        pprint.pprint(request.user.get_all_permissions())
        print("                  }")
        fs = FileSystemStorage()
        var = fs.listdir('')
        # listing of files
        print("FILES IN DIR :    { ")
        pprint.pprint(var[1])
        print("                  }")
        all = status.objects.all()
        return render(request, 'molbank/uploadCSV.html',
                      {'flag_pending': False, 'pending': var[1], 'flag': True, 'pill':
'uploadCSV', 'allstat': all, 'user': user, 'username': request.user.username})
    return render(request, 'molbank/uploadCSV.html', {'flag':
False, 'pill': 'uploadCSV', 'username': request.user.username})

@login_required()
def updateDB(request):
    if request.method == 'POST' and request.FILES['myfileDB']:
        myfileDB = request.FILES['myfileDB']
        if request.user.has_perm('molbank.is_admin'):
            conn = psycopg2.connect("host=localhost dbname=vinay user=postgres
password=SasiHarish2!")
            cur = conn.cursor()
            cur.execute('SET search_path TO bingo')
            og_name = myfileDB.name
            name = og_name.replace(' ', '_')
            name = name.replace('-', '_')
            # name = name.strip('.csv')
            fs = FileSystemStorage()
            filename = fs.save(name, myfileDB)
            f = open(os.path.join(settings.MEDIA_ROOT, filename))
            cur.copy_expert(sql="""COPY molbank_allplates FROM STDIN WITH CSV HEADER DELIMITER
AS ','""", file=f)
            conn.commit()
            cur.close()
            print("og_name=", og_name.strip('.csv')+'.xlsx')
            obj = status.objects.filter(name=str(og_name.strip('.csv')+'.xlsx'))

```



```

        print(obj)
        if obj is not None:
            obj.delete()
            print(obj, "deleted.")
        print(obj)
        all = status.objects.all()
        conn.closed
        return render(request, 'molbank/AdminUploadCSV.html',
                       {'flag': True, 'pill': 'updateDB', 'allstat':
all, 'fname':myfileDB.name, 'username':request.user.username})
        return render(request, 'molbank/uploadCSV.html',{'error': 'You are not authorized to
upload files', 'pill': 'updateDB', 'username':request.user.username})

```

### 5.5.3 Urls.py

```

from django.urls import path, re_path, include
from . import views
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('', views.home, name='home'),
    path('', views.loadhome, name='loadhome'),
    path('search', views.search_mol, name='search_mol'),
    path('index', views.index, name='index'),
    path('advancedSearch', views.advancedSearch, name='advancedSearch'),
    path('simpleSearch', views.simpleSearch, name='simpleSearch'),
    path('add_mols', views.add_mols, name='add_mols'),
    path('add', views.add, name='add'),
    path('login', views.loginHTML, name='login'),
    path('logout', views.logoutHTML, name='logoutHTML'),
    path('uploadCSV', views.uploadCSV, name='uploadCSV'),
    path('updateDB', views.updateDB, name='updateDB'),
    path('status', views.status_of_pending_files, name='status'),
    path('spectral_files', views.spectral_files, name='spectral_files'),
    path('report/<str:report>', views.report, name='report'),

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

### 5.5.4 Search.html

```
{% extends "molbank/base.html" %}
{% load staticfiles%}
{%block title%}
IICT-MOLBANK
{%endblock%}
{%block heading%}
<p> ALL MOLECULES </p>
{% endblock %}
{% block script %}
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
    <script src="{% static '/molbank/kekule/extra/indigoAdapter.js' %}"></script>
    <script src="{% static '/molbank/ketcher/dist/raphael.min.js' %}"></script>
    <script src="{% static '/molbank/kekule/kekule.min.js' %}"></script>
    <script src="{% static '/molbank/jquery-3.3.1.min.js' %}"></script>
    <script>
        jQuery(document).ready(function($) {
            $(".clickable-row").click(function() {
                window.location = $(this).data("href");
            });
        });
    </script>
{%endblock%}
{%block style%}
<style type="text/css">
div.wid{
    max-width:100%;
    margin: auto;
}
</style>
{% endblock %}
{% block content %}
<center>
    {%if mols %}
    <div class="row wid">
    <table class="table table-hover" style="width: 90%;">
        <tr>
            <th>SAMPLE_CODE</th>
            <th>SMILES</th>
            <th>STRUCTURE</th>
        </tr>
```

```

{%for molecule in mols%}
<tr class='clickable-row' data-href="{% url 'report' molecule.sample_code %}">
  <td>{{molecule.sample_code}}</td>
  <td>{{molecule.Smiles}}</td>
  <td>
    <div id="chemViewer_{{molecule.sample_code}}" style="width:160px;height:110px"
data-widget="Kekule.ChemWidget.Viewer" data-chem-obj="url(#id_{{molecule.sample_code}})" data-
predefined-setting="static"></div>
    <script>
      var cmlData = '{% autoescape off
%}{{cml_dict|get_item:molecule.sample_code}}{% endautoescape %}'
      var myMolecule = Kekule.IO.loadFormatData(cmlData, 'cml');
      var chemViewer = new
Kekule.ChemWidget.Viewer(document.getElementById('chemViewer_{{molecule.sample_code}}'));
      chemViewer.setChemObj(myMolecule);
      chemViewer.setZoom(0.5);
    </script>
  </td>
</tr>
{%endfor%}
</table>
<div class="pagination">
<span class="step-links">
  {% if mols.has_previous %}
    <a href="?page=1">&laquo; first</a>
    <a href="?page={{ mols.previous_page_number }}">previous</a>
  {% endif %}

  <span class="current">
    Page {{ mols.number }} of {{ mols.paginator.num_pages }}.
  </span>
  {% if mols.has_next %}
    <a href="?page={{ mols.next_page_number }}">next</a>
    <a href="?page={{ mols.paginator.num_pages }}">last &raquo;</a>
  {% endif %}
</span>
</div>
{%else%}
<p> No Data found... </p>
<p><a href="index"> Try again </a></p>
</div>
{%endif%}
</center>
{% endblock %}

```

## **6. Testing**

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code.

### **6.1 Levels of Testing**

#### **6.1.1 Unit Testing**

While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

#### **6.1.2 Integration Testing**

Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updation etc.

#### **6.1.3 System Testing**

The software is compiled as product and then it is tested as a whole. This can be accomplished using one or more of the following tests:

- **Functionality testing** - Tests all functionalities of the software against the requirement.
- **Performance testing** - This test proves how efficient the software is. It tests the effectiveness and average time taken by the software to do desired task. Performance testing is done by means of load testing and stress testing where the software is put under high user and data load under various environment conditions.
- **Security & Portability** - These tests are done when the software is meant to work on various platforms and accessed by number of persons.

#### **6.1.4 Acceptance Testing**

When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.

- Alpha testing - The team of developer themselves perform alpha testing by using the system as if it is being used in work environment. They try to find out how user would react to some action in software and how the system should respond to inputs.
- Beta testing - After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose. This is not as yet the delivered product. Developers expect that users at this stage will bring minute problems, which were skipped to attend.

#### **6.1.5 Regression Testing**

Whenever a software product is updated with new code, feature or functionality, it is tested thoroughly to detect if there is any negative impact of the added code. This is known as regression testing.

## 7. Screenshots



Figure 11: Homepage

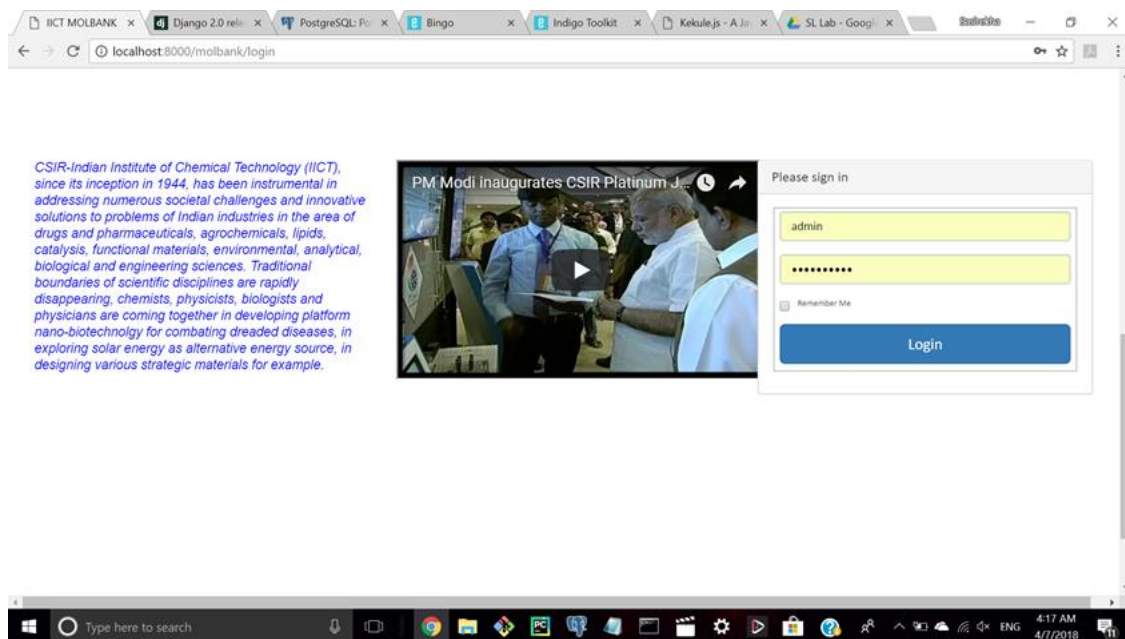


Figure 12: Admin Login Page

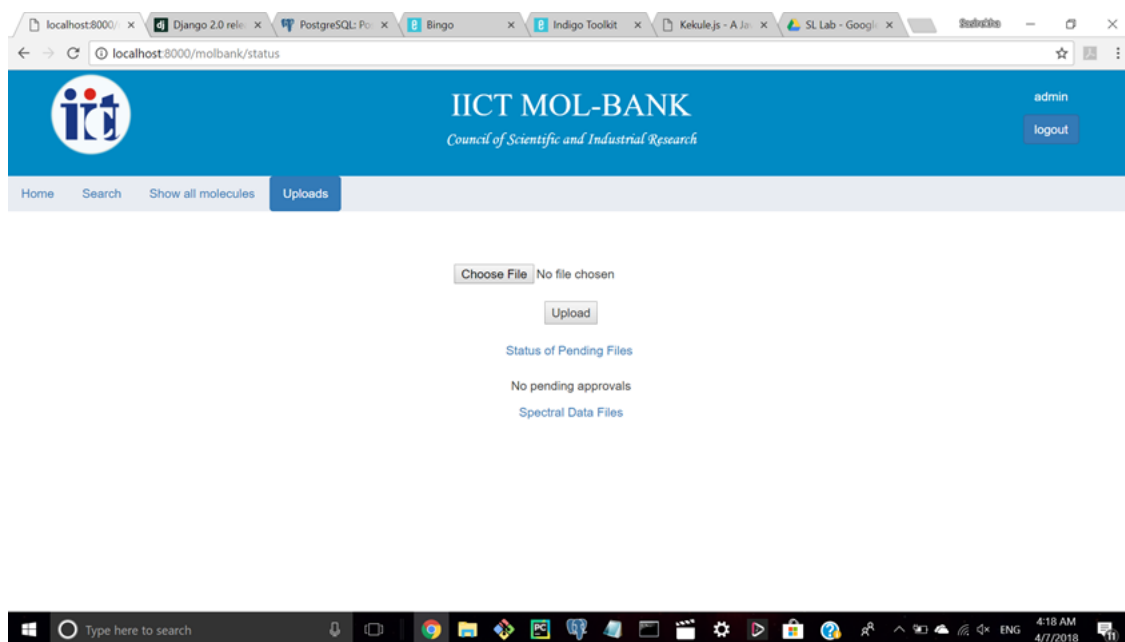


Figure 13: Admin Upload Page

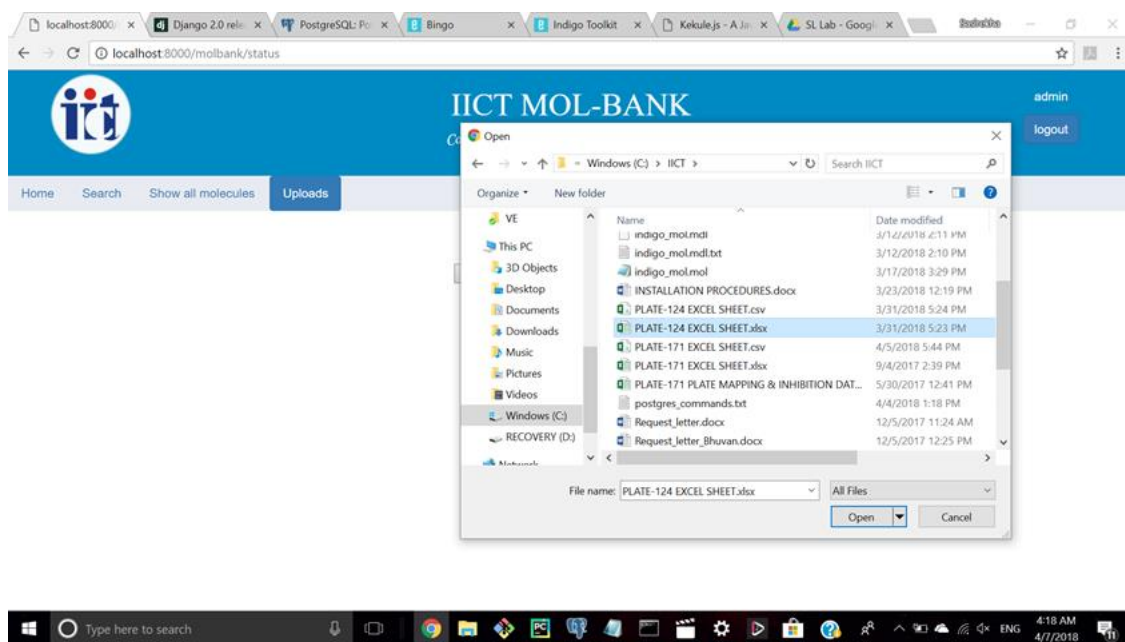


Figure 14: Selection of Template Excel Sheet to Pass to Lab Scientists

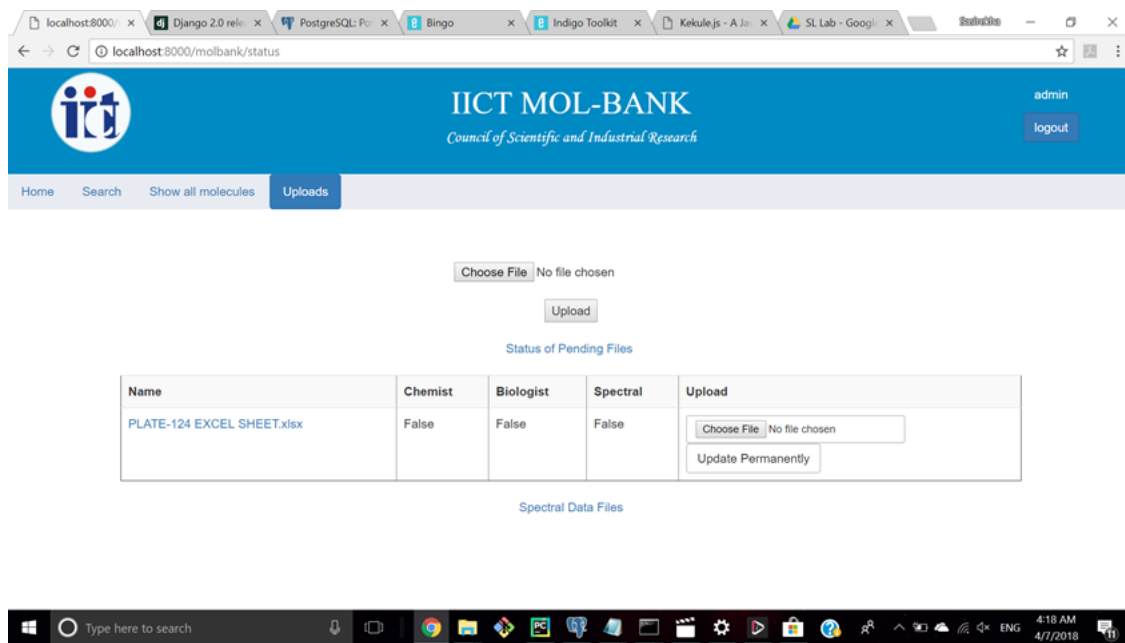


Figure 15: Upload Status (Visible to Admin Only)

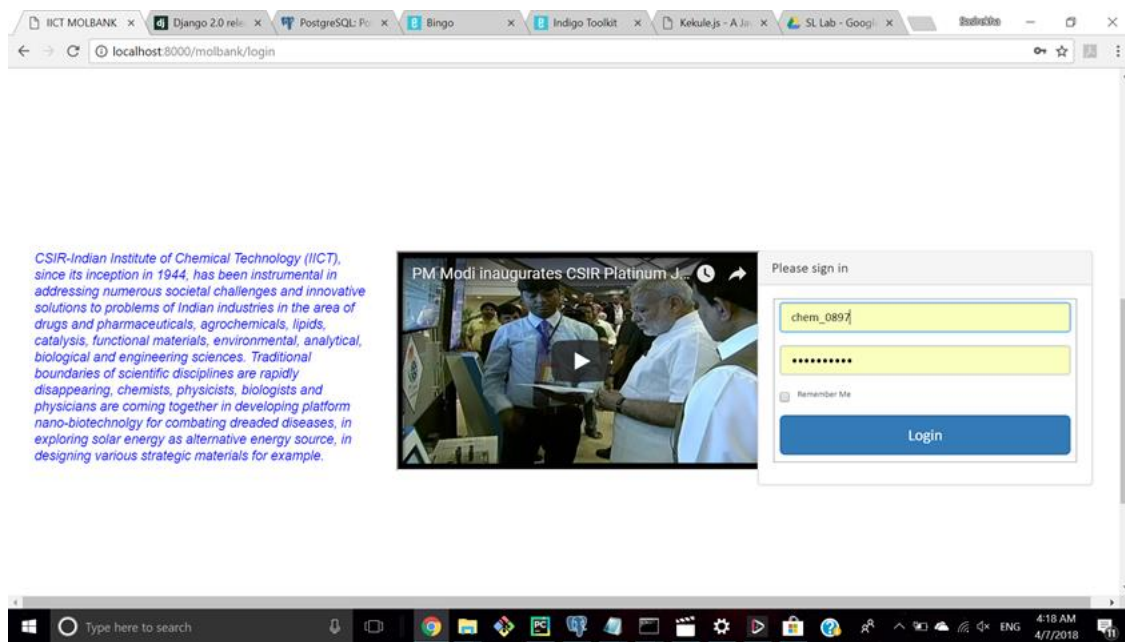


Figure 16: Chemist Upload Page



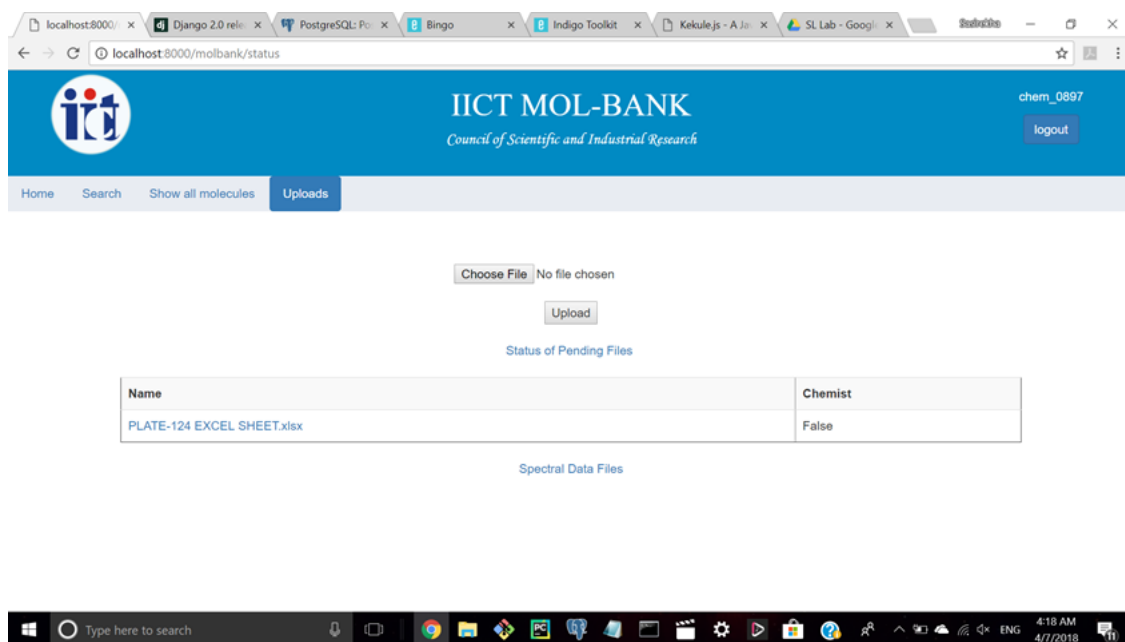


Figure 17: Chemist Status and Upload Page

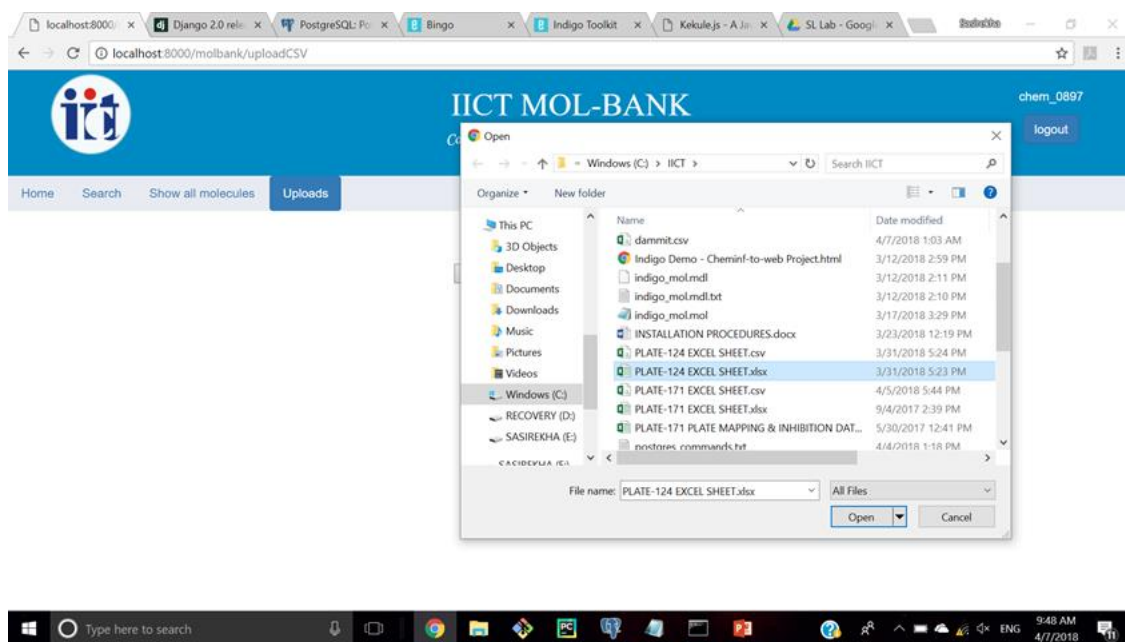


Figure 18: Selection of Excel Sheet with Chemical Properties for Upload

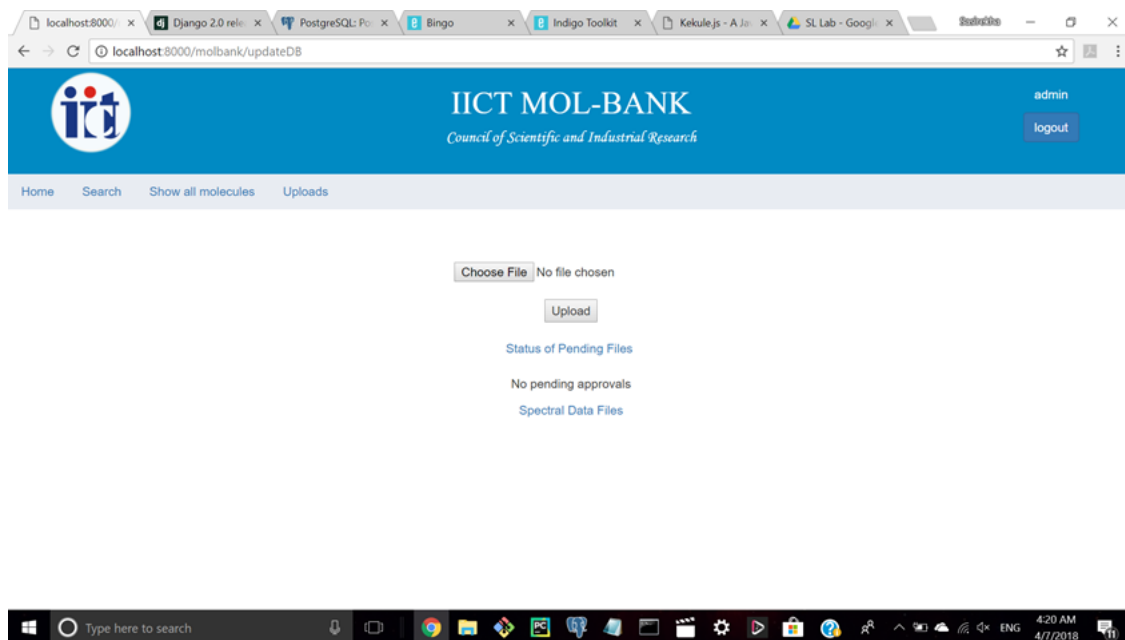


Figure 19: No More Pending Uploads for Chemist



Figure 20: All Molecules are Displayed with Pagination under 'Show All Molecules' Tab

**EM-MB-BTB09747**

**2-((4-(trifluoromethoxy)benzyl)oxy)-3a,4,7,7a-tetrahydro-1H-4,7-epoxyisoindole-1,3(2H)-dione**

SMILES: N1(C(=O)C2C3C=CC(C2C1=O)O3)OCc1ccc(cc1)OC(F)(F)F

Molecular Formula: C16H12F3NO5

Name: None

Molecular Weight: 355.270

Position In Array: A1

MTP Plate No: None

Barcodes Physical Properties Ownership Other Spectral Data

Figure 21: Results Page Retrieved Upon Selecting a Result Molecule

**SEARCH**

Home **Search** Show all molecules Uploads

Simple Search Advanced Search

Search Type

sample\_code

Enter relevant data

EM-MB-BTB09747

Submit

Figure 22: Simple Search Form

SEARCH RESULT x Django 2.0 rele PostgreSQL: Po x Bingo x Indigo Toolkit x Kekule.js - A J x SL Lab - Goog x

localhost:8000/molbank/simpleSearch

**IICT MOL-BANK**  
Council of Scientific and Industrial Research

admin  
logout

Home Search Show all molecules Uploads

SAMPLE_CODE	SMILES	STRUCTURE
EM-MB-BTB09747	<chem>N1(C(=O)C2C3C=CC(C2C1=O)O3)OCc1ccc(cc1)OC(F)(F)F</chem>	

Page of .

Type here to search

4:21 AM 4/7/2018

Figure 23: Simple Search Results Page

SEARCH x Django 2.0 rele PostgreSQL: Po x Bingo x Indigo Toolkit x Kekule.js - A J x SL Lab - Goog x

localhost:8000/molbank/search

**IICT MOL-BANK**  
Council of Scientific and Industrial Research

admin  
logout

Home Search Show all molecules Uploads

Simple Search Advanced Search

Chemical sketching interface showing a naphthalene structure.

Vertical element list: H, C, N, O, S, P, F, Cl, Br, Me, Ph, PT

Type here to search

4:21 AM 4/7/2018

Figure 24: Advanced Search Page for Structure Sketching



Figure 25: Selection of Substructure Search Type

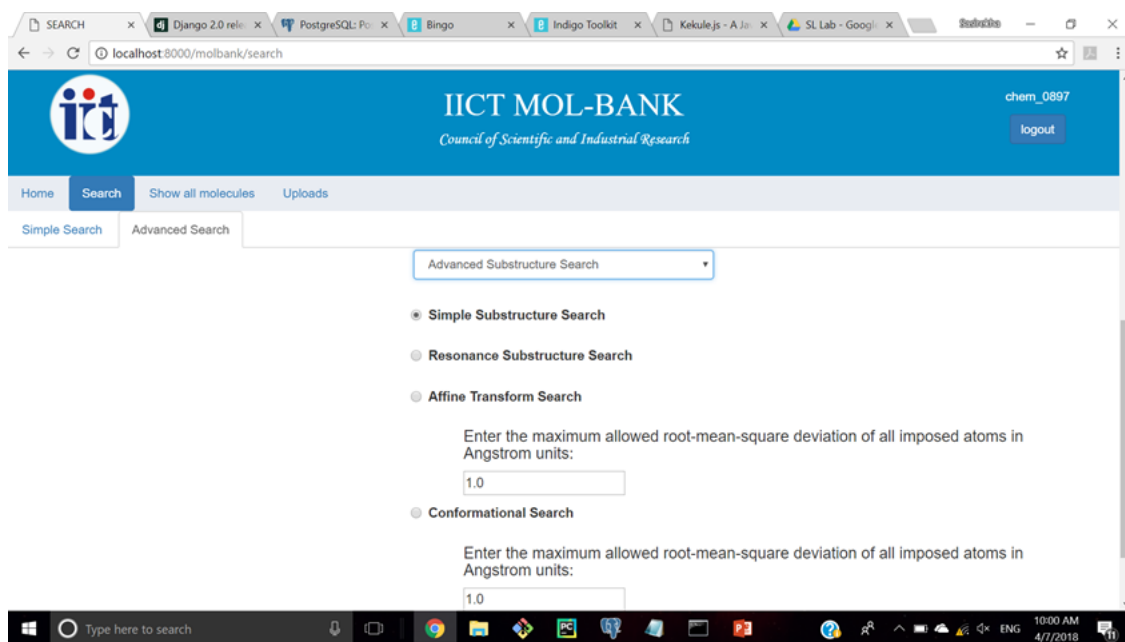


Figure 26: Advanced Substructure Search Form

SEARCH RESULT x Django 2.0 rele x PostgreSQL: Po x Bingo x Indigo Toolkit x Kekule.js - A J x SL Lab - Googl x

localhost:8000/molbank/advancedSearch

**IICT MOL-BANK**  
Council of Scientific and Industrial Research

admin  
logout

Home Search Show all molecules Uploads

SAMPLE_CODE	SMILES	STRUCTURE
EM-MB-BTB11533	<chem>S12(N(C(=O)c3cccc4cccc2c34)C)OC(=O)c2ccccc12</chem>	
EM-MB-BTB11546	<chem>[S+](c2cccc3cccc(c23)C(=O)O)N(C(=O)c2c1cccc2)C.[Cl-]</chem>	
EM-MB-BTB11675	<chem>N1(c2ccccc2)C(=O)c2c3c(ccc3c(cc2)N(C)C)C1=O</chem>	

Type here to search

4:22 AM  
4/7/2018

Figure 27: Search Results with Click-able Rows to Generate Reports

## Conclusion

In this project, we have used *Django* as our Server-side web framework. *PostgreSQL 10* is the underlying database used to store chemical molecules, in conjunction with the RDBMS data cartridge, *Bingo*. The same database is also used to store the application's authentication tables as well. To render molecules on a web browser, we have used *Indigo Toolkit*, which generates 2D coordinates, which are in turn rendered by *Kekule.js*. *Bootstrap* was also used with CSS to make the UI more appealing. Thus, our application provides a report UI that is easy to read and navigate. The search is highly optimized by using indexes to ensure quick retrieval. This is especially necessary when searching using structures over 60,000 molecules.

The research at IICT will benefit significantly from our website as it is faster, inexpensive, and more user-friendly than their current application.

## Bibliography

1. [https://en.wikipedia.org/wiki/Unified\\_Modeling\\_Language](https://en.wikipedia.org/wiki/Unified_Modeling_Language)
2. <https://docs.djangoproject.com/en/2.0/topics/auth/default/>
3. <https://docs.djangoproject.com/en/2.0/topics/security/#security-recommendation-ssl>
4. [https://www.tutorialspoint.com/uml/uml\\_building\\_blocks.htm](https://www.tutorialspoint.com/uml/uml_building_blocks.htm)
5. <http://partridgejiang.github.io/Kekule.js/documents/tutorial/index.html>
6. <http://lifescience.opensource.epam.com/indigo/>
7. <http://lifescience.opensource.epam.com/bingo/index.html>
8. <http://lifescience.opensource.epam.com/ketcher/developers-manual.html>
9. <https://www.postgresql.org/docs/10/static/index.html>
10. [https://www.tutorialspoint.com/software\\_engineering/software\\_testing\\_overview.htm](https://www.tutorialspoint.com/software_engineering/software_testing_overview.htm)