

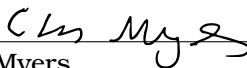
SynBioHub 3 - An Improved Synthetic Biology Repository

Eric Yu


April 2022

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of


Bachelor's of Computer Science



Chris Myers
Advisor



H. James de St. Germain
Director of Undergraduate Studies
School of Computing



Mary Hall
Director
School of Computing

3-May-2022

Contents

1 Introduction	3
1.1 Contributions	5
1.2 Overview	6
2 Background	6
2.1 Synthetic Biology Open Language	6
2.2 Triplestores	7
2.3 Representational State Transfer	8
2.4 Related Works	8
3 Methods	9
3.1 Libraries	9
3.1.1 Spring Boot	9
3.1.2 libSBOLj	10
3.1.3 Apache Jena	10
3.1.4 Hibernate	10
3.2 Data Storage	11
3.2.1 Virtuoso	11
3.2.2 User Data	11
3.3 Design Patterns	13
3.3.1 Controllers	14
3.3.2 Data Transfer Objects	14
3.3.3 Repositories	15
3.3.4 Entities	16
4 Results	17
4.1 Search	18
4.2 Downloads	18
4.3 Authorization and Authentication	21
5 Conclusions	21
5.1 Summary	21
5.2 Further Work	22
5.2.1 Other Endpoints	22
5.2.2 Improved Authentication and Authorization	22
5.2.3 Plugin Support	23
6 Acknowledgement	23
References	24

1 Introduction

One of the primary goals of the field of Synthetic Biology is to facilitate reuse of genetic parts and designs. To meet this goal, there must be an efficient way to store, share, and find these modular genetic designs. This is especially critical for a rapidly growing field such as Synthetic Biology, with publications per year growing nearly tenfold since the year 2000 [1]. To address this, various data repositories have been created, such as the Joint BioEnergy Institute's public Inventory of Composable Elements (JBEI-ICE) [5] and the *International Genetically Engineered Machine* (iGEM) Registry of Standard Biological Parts (<http://parts.igem.org>). SynBioHub was developed in 2017 as a collaboration between multiple universities. It was designed as a repository for not only biological parts, but a wider ecosystem of Synthetic Biology tools, supporting standards like SBOL [3] to accomplish this.

Although SynBioHub is already utilized by many synthetic biologists and organizations, further development is necessary to meet the needs of large-scale synthetic biology projects. The first and most major issue is SynBioHub's use of server-side rendered pages. Figure 1 shows the user-facing pages being rendered by the server before being sent to the user. While this may be easier to develop, as it is part of the same codebase, it eventually becomes harder to maintain as the project scales. The functions to render a page may become intertwined with the data processing functions, making simple tasks such as designing new pages carrying a risk of affecting the backend's behavior. Ultimately, making simple changes may become time consuming, harder to understand, and cause further coupling between the frontend and backend. This also makes keeping the application up to date difficult, as adding or upgrading libraries in one end of the code risks conflicting or breaking parts of the code in the other end of the application. Any vulnerability in the frontend code could also put the backend at risk, making security a concern. Additionally, server side rendering puts the strain of rendering user pages on the server, which can cause degraded performance for the entire SynBioHub instance.

There were other general concerns as well. SynBioHub does not currently support SBOL 3, which supports a much wider range of file types for serialization and deserialization. In the current SynBioHub, the use

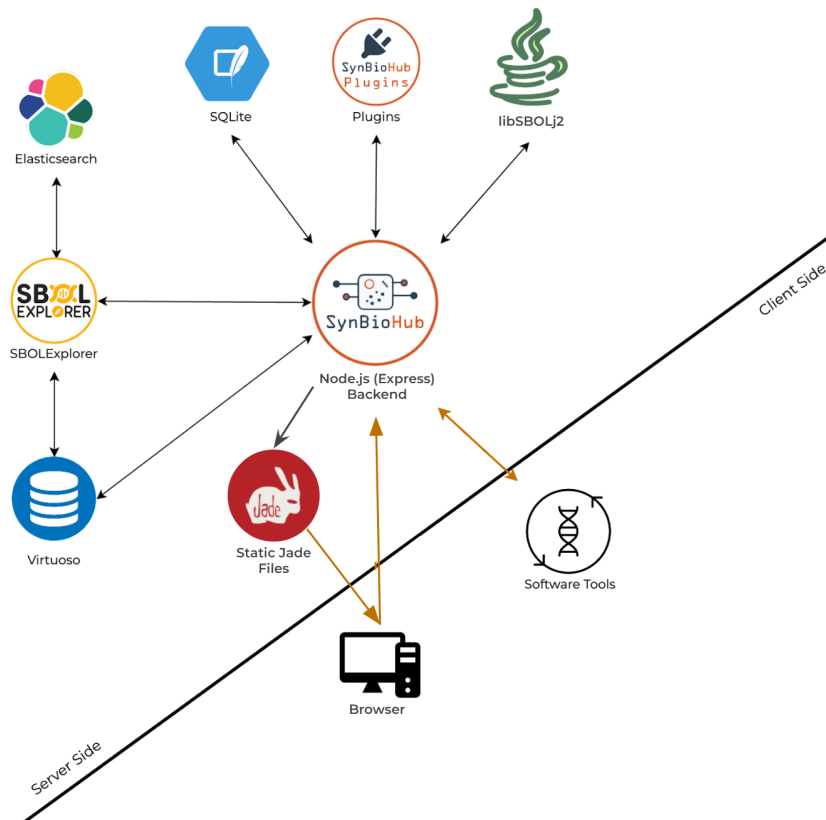


Figure 1: Architecture diagram of the original SynBioHub. Note the usage of Jade files, which are webpage templates statically rendered by the backend before being sent to the frontend.

of the Java SBOL 2 library is currently implemented as a workaround to the Node.js backend, meaning there is both Java and Javascript code running concurrently, which is not optimal. This further increases complexity and may be something that affects the performance of SynBioHub, which is another concern. There have been many instances of SBH freezing or running slowly, such as during high periods of use by concurrent users at the same time.

SynBioHub2 was designed as a temporary solution to some of these problems. A new frontend was built using ReactJS, communicating with SynBioHub using its REST API as its backend. While this solved the issue of the frontend being coupled with the backend, the performance was still reliant on the original SynBioHub. SynBioHub3 aims to address the previously mentioned concerns, by using SynBioHub2's newly developed frontend, coupled with a newly developed backend. First and foremost, SynBioHub3's architecture adheres to a strict separation of concerns between the frontend and backend. Communication between the two ends is done solely through a public REST API. This means that the frontend can be built as a standalone application that acts as a graphical user interface for the API, removing all coupling between the two applications. The API is written following the existing documentation such that existing applications that rely on SynBioHub such as SBOLCanvas [12] or SBOLDesigner [15] are able to use SynBioHub3 with little configuration needed. SynBioHub3 will be written to use many of the same backend applications, as there is no strong case to change these applications, however it will be written in a way such that switching triplestores is no harder than changing a line within the configuration file.

1.1 Contributions

This thesis will describe the major changes in the complete rewrite of SynBioHub3.

Redesigned Architecture

SynBioHub3 has been completely rewritten in Java following a different design pattern. This redesigned architecture is an improvement to SynBioHub's server-side rendering method. By reducing the coupling

between the client-side and server-side code, SynBioHub3 is made more modular, meaning the backend does not need to rely on a scripting language such as JavaScript to render frontend pages. This has led to an increase in performance, with tasks such as downloading large parts being much faster.

Improved Downloads

The entire download algorithm has been rewritten in SynBioHub3. The downloading algorithm in SynBioHub was inefficient, resulting in slower performance. SynBioHub3 redesigns the data gathering and filtering process, resulting in a massive speed gain as well as reducing the load that the triplestore must handle.

Authentication and Authorization

Authentication and Authorization have been implemented using the Spring Security library, replacing the previously hand-written library in use. This allows SynBioHub3 to take advantage of the already available functions for more secure basic authentication, as well as lay the groundwork for further development.

1.2 Overview

First, this thesis will describe the background of standards such as the Synthetic Biology Open Language, and other core concepts used in SynBioHub. Next, it will go over the methods used, such as various libraries, tools, and design standards used in the development of SynBioHub3. Finally, it will describe what has been currently implemented in SynBioHub3, as well as discuss what features are to be added in the future.

2 Background

2.1 Synthetic Biology Open Language

The Synthetic Biology Open Language (SBOL) was created as a response to the issue of unreproducible sequence information in papers, which hurts reproducibility and limits reuse of past work^[4]. Standardized

languages can help to improve communication by decreasing ambiguity and increasing the amount of information that can be conveyed, all while increasing the amount of flexibility both by the compatibility with many software tools and the adoption of one core standard. The first version of the SBOL standard defined a simple data model for genetic parts, and has since evolved to store more information. Version 2.0 of the SBOL standard [3], released in July of 2015, introduced a host of new features, including a new core data model. Version 3.0 of the SBOL standard, released in September of 2020 [7], refines the previous version by simplifying the SBOL model to reduce its complexity which was limiting future development.

The driving design goal for SynBioHub is to achieve all FAIR (findable, accessible, interoperable, reusable) principles for data sharing [13]. To achieve this goal, SynBioHub (unlike other repositories) natively supports a community-developed standardized data format—the *Synthetic Biology Open Language* (SBOL) [3]. This native support helps SynBioHub to meet the principles of FAIR in a number of ways. First, SBOL’s rich metadata and use of ontologies for identifiers enables standardized search algorithms that make designs more findable [14] [16]. Second, public instances of SynBioHub provides easy accessibility without authentication to static published genetic design information either through its web-based interface or REST *application programmers interface* (API). Third, the use of the SBOL standard promotes interoperability, since it is supported by a growing number of *genetic design automation* (GDA) tools, such as Cello [9], SBOLCanvas [12], VisBOL [6], SBOLDesigner [15], and more (<https://sbolstandard.org>). Finally, SynBioHub can be used with a curation workflow that ensures that the SBOL representation is rich with metadata and provenance information to promote genetic design reuse.

2.2 Triplestores

The RDF, or Resource Description Framework, is a data model created by the World Wide Web Consortium (W3C) in 1999, designed to encapsulate metadata. More specifically, it aims to store relationships between data - subject, predicate, object. For example, if we want to represent the sentence “the sky is blue”, we could store this in a triple,

with the subject being “sky”, predicate being “color” or “has color”, and object being “blue”. The SBOL [3] standard was built on top of this data standard to allow for standardized storage of biological data (see Section 2.1).

Triplestores are often used in Synthetic Biology because of the graph-like structure of parts. These relationships are not well represented in relational databases, as fetching data would require many join queries traversing along the foreign-primary key path, which can be computationally expensive. A triplestore is similar to a normal relational database, except that each entry, or triple, must follow the RDF model described above.

2.3 Representational State Transfer

Representational State Transfer (REST) is a set of guiding principles designed in 2000 for use in distributed web systems. Over 75% of web services were found to adhere to these principles in 2019, compared to other design principles such as SOAP or GraphQL [8]. There are a few main guiding principles that a REST architecture must satisfy. First, there must be a uniform interface, meaning that each interface must be self-descriptive and have a uniform response. Second, it must follow the Client-Server architecture, which helps to enforce separation of concerns and improve portability of user data across multiple interfaces. Third, requests must be stateless, meaning that each request must contain all information necessary to resolve the request. Finally, the system must be designed in a layered hierarchy such that one component does not extend its functionality beyond its current layer. This places a bound on system complexity as changes made on one level will not be reflected on another.

2.4 Related Works

SBOExplorer [16] is a search program used to improve the quality and sorting of search results in SynBioHub. SynBioHub uses SBOExplorer for keyword and sequence queries [16][14]. SBOExplorer uses an inverted-index to return better fuzzy search results, which is implemented using Elasticsearch. For example, if the user erroneously

types the query “GPF” whilst intending to type “GFP”, SBOLEplorer is able to predict the user’s intentions and serve them the correct search results, similar to search engines on the web. Additionally, SBOLEplorer implements the PageRank algorithm [10], which is used to sort genetic design records by their popularity. This is achieved by querying the triplestore for a graph of each part, associating a higher number of links to a higher pagerank, which translates to greater popularity.

3 Methods

This section will go over the implementation-level details in the rewrite of SynBioHub3. First, various libraries used in the backend are introduced, describing their functionality and importance. Next, we will discuss the design patterns of SynBioHub3 and how they contribute to a more modular application.

3.1 Libraries

3.1.1 Spring Boot

The libraries used in SynBioHub3’s core backend development is the Spring Boot framework. This is a framework that encompasses many different libraries, such as security, database management, and REST request routing. By using this package of libraries, it allows for a more consistent code style, as the encapsulated functions will be standardized in regards to naming, documentation, and return types. This improves readability, as well as allowing developers less familiar with SynBioHub3 to pick up development quicker. These libraries have been used and tested by many large enterprises, which will eliminate many concerns as SynBioHub3 scales in the future. Because Spring boot applications like SynBioHub3 are written in Java, it can take advantage of the Java Virtual Machine. This allows us to compile the app into a single, standalone JAR file and run it on any platform, which reduces the complexity of fetching multiple dependencies, building, and deploying SynBioHub3 to platforms such as Docker.

3.1.2 libSBOLj

libSBOLj is a library that provides an implementation of the SBOL [3] standard in Java. Parts are abstracted as objects, and various functions are provided to manipulate these objects in code. An I/O module is also provided for serialization of RDF/XML files, and deserialization into a wide range of commonly used formats. A validator is available to check that SBOL models follow the standard correctly. libSBOLj is available for both versions 2 and 3 of the SBOL standard, with SynBioHub only supporting the former. SynBioHub3 will support both versions of the SBOL standard, in order to be backwards compatible with existing applications, as well as having full support for newer applications that can take advantage of the new SBOL version.

3.1.3 Apache Jena

Apache Jena is a library used to create and read RDF graphs. It provides classes to create and model RDF graphs, as well as functions to serialize and deserialize data in a wide range of formats. The most important functions used in SynBioHub3 (and not available in libSBOLj) reside within its graph manipulation abilities; functions such as unions and intersections are supported, and are much faster than its respective queries in Virtuoso (a triplestore). This allows SynBioHub3 to simplify the queries being sent to Virtuoso, reducing the load it must handle, speeding up download queries in the process.

3.1.4 Hibernate

An Object Relational Mapping (ORM) is required if one wants to be able to use a relational database (RDBMS) within an object-oriented application such as SynBioHub3. This is because of a few difficulties arising translating from an object-oriented paradigm to a relational paradigm, also known as an object-relational impedance mismatch. First, objects may be more granular than their corresponding database table, meaning not all class fields may exist in the database. Second, the notion of subtypes is common in objects, however this paradigm does not exist in RDBMSs. Third, equality can be defined by both value and object identity in object-oriented programs (i.e. $a==b$ versus $a.equals(b)$), while

RDBMSs only understand the notion of a key. Fourth, accessing data in an object-oriented program is done by following relations between objects. This is inefficient in an RDBMS, as it requires multiple queries, which is computationally expensive. Finally, objects are not guaranteed to follow Atomicity, Consistency, Isolation, and Durability (ACID) principles, as they can be mutated, which can cause conflicts in the database.

Hibernate is an ORM library written in Java and used by Spring Boot. It aims to solve the aforementioned problems by providing classes and functions to model and manipulate an RDBMS from within the code itself. SynBioHub uses Hibernate to map user data within the SQL database. This allows actions such as reading row values or writing data such as passwords to be accomplished without having to write any SQL queries, reducing both time, complexity, and risk for common attacks such as SQL injection attacks.

3.2 Data Storage

3.2.1 Virtuoso

SynBioHub uses Virtuoso, a triplestore developed by OpenLink Software, to store RDF triples. The format that these triples are stored are follow the SBOL standard [3] [7], allowing a uniform data model that other applications can take advantage of as well. The SPARQL [1] query language, similar to SQL, is used by SynBioHub to interact with the triplestore. Figure 2 shows an example of one SPARQL query used by SynBioHub. Virtuoso provides a REST API, so SynBioHub or any other applications such as SBOExplorer [16] are able to query the triplestore without any configuration needed, simply by attaching a SPARQL query to a REST request.

3.2.2 User Data

In SynBioHub, user data was managed through the use of an embedded database, SQLite. The main advantage of such a database over traditional client-server databases such as MySQL or PostgreSQL is its ease of use. Traditional databases require a separate server process to manage the database, which needs to be installed and configured be-

```
PREFIX sbol: <http://sbols.org/v2#>
PREFIX dcterms: <http://purl.org/dc/terms/>

SELECT distinct ?uri WHERE {
  <$uri> sbol:member ?member .{
    ?member a ?uri .
  }
  UNION {
    ?member sbol:type ?uri .
    FILTER(STRSTARTS(str(?uri), 'http://www.biopax.org/release/biopax-level3.owl'))
  }
  UNION {
    ?member sbol:role ?uri .
    FILTER(STRSTARTS(str(?uri), 'http://identifiers.org/so/'))
  }
}
```

Figure 2: An example of a SPARQL query used to get all SBOL members of type “type” or “role”. Note the use of union operations, which can have an impact on the performance of Virtuoso, depending on the amount of resources is allocated to the process. SynBioHub3 aims to address this by moving all data filtering into the backend instead.

fore use, taking both disk space and time. With SQLite, the server is represented as a self-contained file, meaning that the database is both lightweight and portable, which is advantageous for the federated structure of SynBioHub, where multiple instances across different servers can often share the same user data.

SynBioHub3 uses H2, an embedded database written in Java. Similar to SQLite, it uses a database file instead of a dedicated server process, and is extremely lightweight, with the JAR file taking less than 3 MB of disk space. The decision to switch to H2 was twofold: first, SQLite only has four primitive data types: Integer, Real, Text, and Blob. Any other data types must fit into one of these four categories, which severely restricts what we are able to store without any compatibility workarounds. Second, Spring Boot (Section [3.1.1](#)) does not officially support SQLite, so there is no official Java Database Connectivity (JDBC) driver. This would mean that SynBioHub3 would need to rely on a third-party driver in order to support the database connection, which may have compatibility issues with the Spring Boot libraries.

3.3 Design Patterns

The backend utilizes a traditional Model-View-Controller (MVC) design pattern. Figure 3 shows how a user interacts with this design. The Model is what can be thought of as the “backend”, or the core of SynBioHub3’s logic. Here, functions reside to manipulate and parse input data, fetch data from other applications, and save and return prepared data, and control who can and cannot view certain data. The Controller is what the user interacts with; it is responsible for the user-facing API, and presenting this data to the Model. The View is the representation of data, most often graphical, that the user sees. For example, the SynBioHub3 front end or applications such as SBOLCanvas^[12] are views that present this structured data from SynBioHub3.

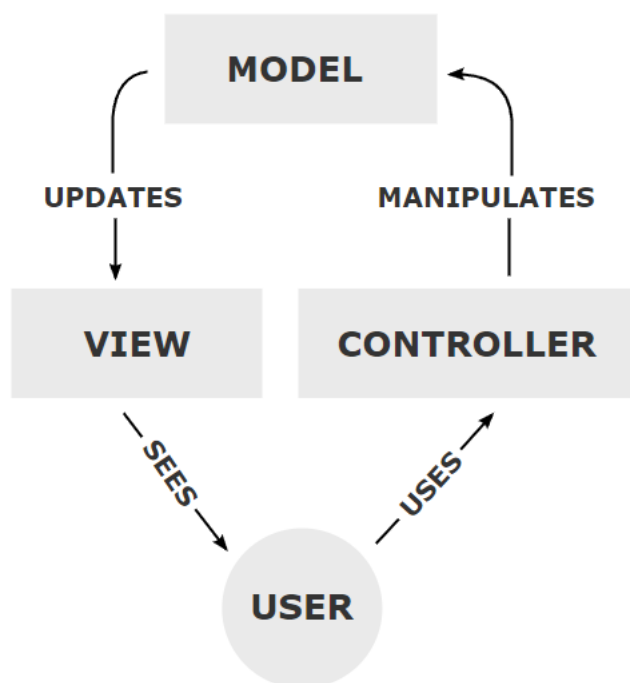


Figure 3: The Model-View-Controller design pattern.

This separation of concerns addresses one of the major points in our redesign of SynBioHub. By separating the user-facing layer and logic layers, it is much easier to make changes to either side without interference, and changes will affect fewer parts of the entire project.

It also allows for easier development between different users, which is important as SynBioHub3 grows. Figure 4 shows the package structure of SynBioHub3.

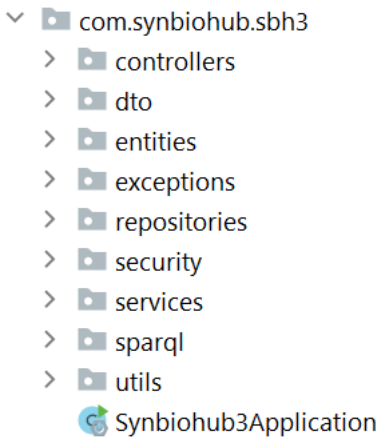


Figure 4: Overview of the packages in SynBioHub3. Each package aims to self-contain its code such that changes in one package do not affect behavior of other packages, with major changes being easy to implement through basic code refactoring.

3.3.1 **Controllers**

Controllers are the first layer in the API for REST request handling. Each controller corresponds to one endpoint in the API. Figure 5 shows an example of one of many endpoint in SynBioHub3. Each endpoints can be configured with the “mapping” annotation, which designates both the type of REST request (E.G. GET, POST, DELETE, etc.), as well as the endpoint URL itself. These endpoint structures are also highly configurable, with support for pattern matching, as well as regular expressions. All of these features are provided by Spring Boot (Section 3.1.1), reducing the time needed to manually route and handle these various request parameters.

3.3.2 **Data Transfer Objects**

Data transfer objects (DTOs) are objects in the backend that represent a piece of data. They hold no logical code; only minimal functions to

```
@GetMapping("/{visibility:.+}/{collectionID:.+}
                /{displayID:.+}/{version:.+}/subCollections")
public String getSubCollections(@PathVariable("visibility") String
    visibility, @PathVariable("collectionID") String collectionID,
    @PathVariable("displayID") String displayID,
    @PathVariable("version") String version,
    HttpServletRequest request) throws
    JsonProcessingException {

    String collectionInfo = String.format("%s/%s/%s/%s", visibility,
        collectionID, displayID, version);
    String sparqlQuery =
        searchService.getSubCollectionsSPARQL(collectionInfo);
    return
        searchService.collectionToOutput(searchService.SPARQLQuery(sparqlQuery));
}
```

Figure 5: A REST controller responsible for getting all sub-collections. Note that very little code exists here; data processing is instead delegated to various service functions in order to abstract the controller and service layers of SynBioHub3.

access and mutate the data. The main advantage of DTOs are their self-contained nature, as they were originally designed to aggregate multiple pieces of data into one object in order to avoid repeated network calls.

The use of DTOs allows SynBioHub3 to separate the data layer from the service layer. Figure 6 shows a DTO used by SynBioHub3 to receive a multipart login form. By abstracting the various pieces of data into an object, any changes to a DTO minimally impact how functions must parse this data, reducing the coupling that the data and service layers will have.

3.3.3 Repositories

Hibernate (Section 3.1.4) is used in order to map a RDBMS table's information to a Repository object. Because the database table has been abstracted into an object, queries to the database can be written simply by declaring a "findBy" method name, with values being passed in as parameters. Operations such as saving, updating, or deleting rows in the database are possible using built-in functions, and can take lists of Entities as arguments to perform batch saves. Figure 7 shows an

```
@Data
public class LoginDTO {

    @NotBlank
    private String email;

    @NotBlank
    private String password;
}
```

Figure 6: A DTO used to encapsulate a login query. Note that there are no methods for data processing; this is done in the service layer. Various annotations can be used in order to specify the required fields and types for a query. The `@Data` annotation is used by Spring Boot to automatically generate getters and setters for all fields, reducing the need to write these manually.

example of a Repository used in SynBioHub3 to store user data.

```
@Repository
public interface UserRepository extends JpaRepository<UserEntity,
    Integer> {
    Optional<UserEntity> findByUsername(String username);
    Optional<UserEntity> findByEmail(String email);
}
```

Figure 7: The “users” SQL table represented as an object in Java. Each query function will return a row from the database, which corresponds to the `UserEntity` object. The Java “Optional” container wraps the return type of the function, meaning that the query may return a null value.

3.3.4 Entities

Entities are abstractions of each row within a RDBMS table. The Hibernate (Section [3.1.4](#)) library is used to map the relational row schema into an object. Figure [8](#) shows an example of an Entity in SynBioHub3. Relational-specific paradigms, such as columns, primary and foreign keys, uniqueness, and null constraints are all able to be represented using Hibernate’s annotations. These Entities are able to be manipulated in code like any other object, and operations such as inserting

multiple Entities into a List in order to perform batch SQL commands are all supported by Hibernate.

```
@Entity
@Data
public class UserEntity {
    @Id
    @SequenceGenerator(name= "CLIENT_SEQUENCE", sequenceName =
        "CLIENT_SEQUENCE_ID", initialValue=1, allocationSize = 1)
    @GeneratedValue(strategy=GenerationType.AUTO,
        generator="CLIENT_SEQUENCE")
    private Integer id;

    @Column(name = "name")
    private String name;

    @Column(name = "username")
    private String username;

    @Column(name = "password")
    private String password;

    @Column(name = "email")
    private String email;

    @Column(name = "affiliation")
    private String affiliation;

    @Column(name = "isAdmin")
    private Boolean isAdmin;

    @Column(name = "isCurator")
    private Boolean isCurator;
```

Figure 8: A class representing each row within the "users" table. The use of Hibernate allows us to represent database entries as Entities in code, as part its Object-Relational Mapping capabilities. Data types such as Strings, Integers, and Booleans are all mapped to the corresponding SQL data types automatically as well.

4 Results

This section will describe what has been implemented in SynBioHub3 thus far. Figure 9 shows the newly designed architecture at a high level,

with SynBioHub3's interaction with various libraries and services. Both search and download endpoints have been fully implemented, meaning existing users and applications can use SynBioHub3 immediately with no major changes. Basic Authentication and Authorization have been added, meaning users are able to login and receive an X-Authorization token for identity verification.

4.1 Search

All of the original search endpoints in SynBioHub have been successfully replicated in SynBioHub3. This means that apps will be able to use the search functionality without any major changes. Additionally, new standardized endpoints have been added to replace some inconsistencies in the old SynBioHub.

Many of the endpoints in the original SynBioHub do not strictly adhere to the Uniform Resource Locators (URL) standard [2], and as a result, may vary in their formatting. For example, some endpoints do not require a question mark before the query string, which is not ideal. Figure 10 shows examples of these inconsistencies. These did not affect the original SynBioHub as heavily, as URL routing and parsing was done manually. However, in Spring Boot, there already exist methods to parse valid URL's, and the inconsistent formatting does not allow for programmatic best practices in using available libraries. In SynBioHub3, we have introduced a new set of endpoints that will adhere to the URL standards. These will be added alongside the endpoints. In the future, we may choose to deprecate these URL's if SynBioHub3 achieves majority adoption, in order to achieve API standardization.

4.2 Downloads

Another set of endpoints that have been fully implemented are downloads. Users are able to download any part in a wide range of formats. This is accomplished by using libraries to read and model the raw data from Virtuoso (Section 3.1.3) as well as to serialize the data into a wide range of formats (Section 3.1.2).

Downloads have seen a significant boost in performance, due to two main reasons. First, the recursive algorithm used to fetch child

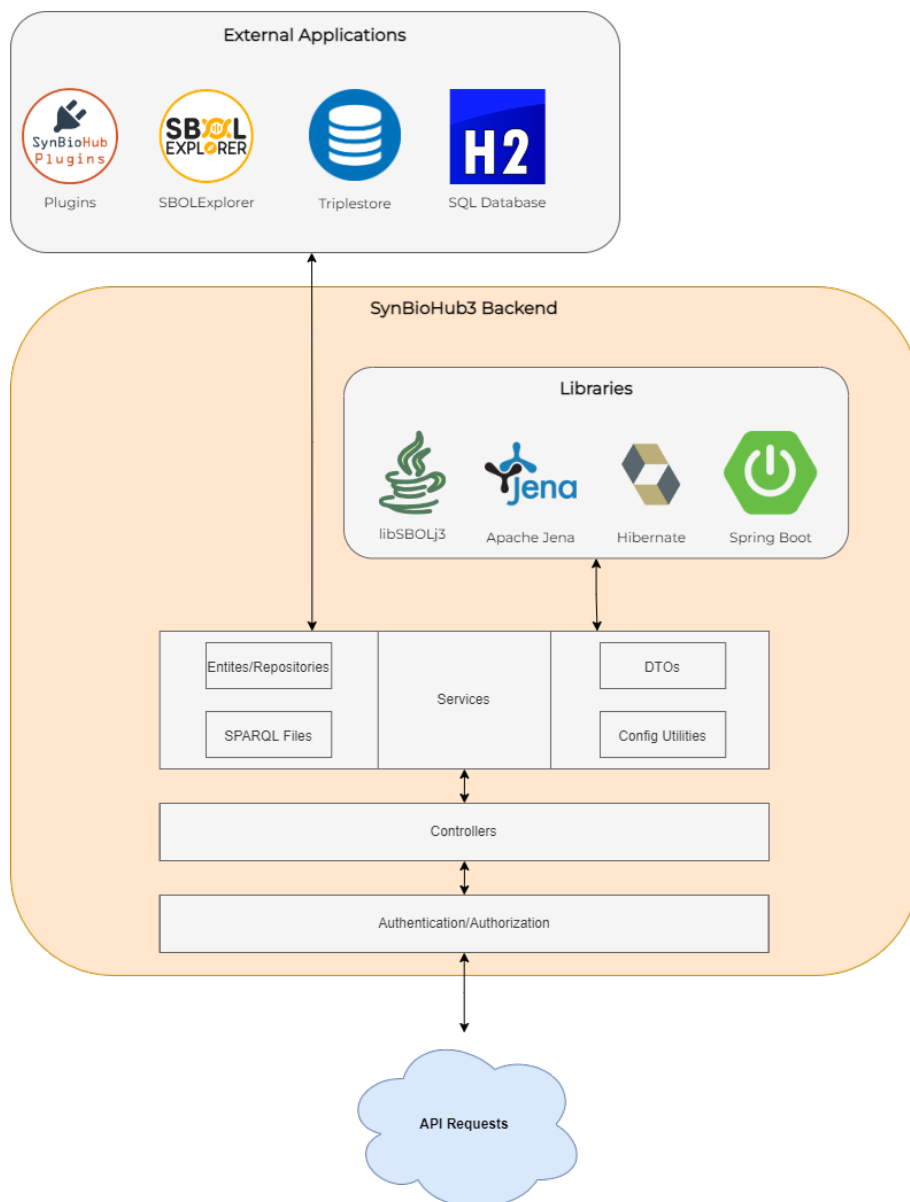


Figure 9: The architecture of SynBioHub3. Many backend applications used are similar, however, frontend pages are no longer rendered by the backend. The frontend is treated like any other application, and communicates solely through the public REST API.



Figure 10: Examples for querying SynBioHub found in the API documentation (<https://wiki.synbiohub.org/api-docs/>). For a search query, the first set of key/value pairs is separated by a single slash, and the offset and limit are separated by a slash and question mark, while the SPARQL query endpoint uses a question mark but no slash to separate the query. Similar inconsistencies within the API design exist; they are not ideal, as for end users it can be more confusing to use, and for developers it requires different, sometimes non-standard, methods of parsing URL's.

parts has been completely rewritten. The use of an RDF parsing library (Section [3.1.3](#)) allows us to model the RDF graph as a binary tree, allowing us to take advantage of a breadth-first search operation to fetch child parts. Secondly, the SPARQL query to fetch these parts have been rewritten as well. Unions have been removed from queries sent to Virtuoso, as all data processing and filtering is now moved into the backend. These union queries have had a significant impact on the time to query Virtuoso. To test the new download improvements, a download query to fetch the 2019 iGEM parts collection was sent to both SynBioHub and SynBioHub3 locally. The command `curl -s -w %{time_total} \n <uri>` was used to time and download each GET request. Table [1](#) shows these results. Alongside these two major improvements, SynBioHub3 benefits from not having to switch between Java and Javascript to complete download operations, although it is difficult to quantify the extent to which this had an impact on performance.

	iGEM Parts Registry
SynBioHub	Did not finish
SynBioHub3	22.77 sec

Table 1: Testing download speeds on local SynBioHub and SynBioHub3 instances. Times are averaged over 3 trials. SynBioHub could not finish the download.

4.3 Authorization and Authentication

Authentication and authorization is a critical part of SynBioHub. User accounts are used to manage uploads, collections, and shared parts. In the original SynBioHub, SQLite is used to store this basic user data basic access privileges, such as administrator or curator. In SynBioHub3, SQLite is still used to allow easy portability of user data. However, the tables are updated to store other data, such as user groups and more granular share links. This is accomplished by storing in the SQLite user tables specific record access rights on graphs, sub-graphs, and individual URI identified objects as described in [17].

In SynBioHub 3, basic authentication and authorization has been implemented. Users are able to register as a new user, and login to receive an x-authorization token. This has been achieved by using Spring security. The library provides a wide range of features within the authentication and authorization process. When a user logs in, the library will check whether such user exists, and if so, whether their password matches the hashed representation in the database. It will then get the user's roles, such as admin or curator, and return this as an authorization token. When a logged in user queries SynBioHub3, Spring Security (Section 3.1.1) will check to see whether a user has access to a certain subset of endpoints, which can be fine tuned in the backend.

5 Conclusions

5.1 Summary

SynBioHub3 aims to fix the issues that SynBioHub currently faces. Its architecture has been redesigned from the ground up, following a strict Model-View-Controller design pattern, removing the coupling be-

tween the backend and frontend. By using the Spring Boot framework, SynBioHub3 avoids the need to integrate a large number of external libraries, improving code consistency while reducing complexity. Authentication and authorization is handled by Spring Boot, which lays a consistent foundation for any future complex additions or modifications to this layer. By rewriting the algorithm for downloading parts, performance has been increased by shifting the data computation from the triplestore to the backend, as well as reducing the load that the triplestore must handle.

5.2 Further Work

The SBH3 backend is still currently in development, and thus is missing some features that exist within the original SynBioHub, as well as new features that do not currently exist.

5.2.1 Other Endpoints

There are multiple endpoints that rely on the authentication endpoints, such as submission, permission, and attachments, with submission being a non-trivial task to implement. These rely on the authentication endpoints in order to check user access to parts for download or modification. We hope to see speed-ups, specifically with submission, since this is implemented using Java in SynBioHub.

5.2.2 Improved Authentication and Authorization

More granularity in authentication and authorization, such as storing share links or multiple user groups, is something that has been discussed by members of the SBOL board during the planning phase of SynBioHub3. By taking advantage of the relational SQL database, which is currently only storing basic user information, we can use more complex relational methods such as foreign keys and joins to store data such as groups, graph permission, or share links with varying levels of granularity.

5.2.3 Plugin Support

Plugins are a crucial part of SynBioHub, as they have been the main method of adding new features due to their ease of development. There are currently 37 official plugins available for SynBioHub (<https://github.com/SynBioHub>), which handle features such as upload conversion, SBOL visualization, part curation, and search. We would like to reuse many of these plugins in SynBioHub3, as their usage has proven their helpfulness in many aspects in SynBioHub. Additionally, plugins are an excellent way for anyone who may want to contribute to SynBioHub3 without having to modify its source code, and we want to support the community-focused aspects of these plugins.

6 Acknowledgement

The author would like to thank Professor Myers for being the principal investigator of this research, as well as Jeanet Mante for revisions of this paper and Benjamin Hatch for contributions to the frontend of SynBioHub3. The author of this work is supported by DARPA FA8750-17-C-0229, the National Science Foundation Grant No. 1939892, and the National Institute of Standards and Technology (NIST), Microsoft Azure. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References

- [1] SPARQL 1.1 Query Language. Technical report, W3C, 2013.
- [2] Tim Berners-Lee, Larry Masinter, and Mark McCahill. Uniform resource locators (url), Dec 1994.
- [3] Robert Sidney Cox, Curtis Madsen, James Alastair McLaughlin, Tramy Nguyen, Nicholas Roehner, Bryan Bartley, Jacob Beal, Michael Bissell, Kiri Choi, Kevin Clancy, and et al. Synthetic biology open language (sbol) version 2.2.0. *Journal of Integrative Bioinformatics*, 15(1), 2018.
- [4] Michal Galdzicki, Kevin P Clancy, Ernst Oberortner, Matthew Pocock, Jacqueline Y Quinn, Cesar A Rodriguez, Nicholas Roehner, Mandy L Wilson, Laura Adam, J Christopher Anderson, and et al. The synthetic biology open language (sbol) provides a community standard for communicating designs in synthetic biology. *Nature Biotechnology*, 32(6):545–550, 2014.
- [5] Timothy Ham, Zinovii Dmytriv, Hector Plahar, Joanna Chen, Nathan Hillson, and Jay Keasling. Design, implementation and practice of jbei-ice: an open source biological part registry platform and tools. *Nucleic acids research*, 40(18):e141–e141, 2012.
- [6] Benjamin Hatch, Linhao Meng, Jeanet Mante, James A. McLaughlin, James Scott-Brown, and Chris J. Myers. Visbol2—improving web-based visualization for synthetic biology designs. *ACS Synthetic Biology*, 10(8):2111–2115, 2021. PMID: 34324811.
- [7] James Alastair McLaughlin, Jacob Beal, Göksel Mısırlı, Raik Grünberg, Bryan A. Bartley, James Scott-Brown, Prashant Vaidyanathan, Pedro Fontanarrosa, Ernst Oberortner, Anil Wipat, Thomas E. Goroehowski, and Chris J. Myers. The synthetic biology open language (sbol) version 3: Simplified data exchange for bioengineering. *Frontiers in Bioengineering and Biotechnology*, 8, 2020.
- [8] Andy Neumann, Nuno Laranjeiro, and Jorge Bernardino. An analysis of public rest web service apis. *IEEE Transactions on Services Computing*, 14(4):957–970, 2021.

- [9] Alec A K Nielsen, Bryan S Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth A Strychalski, David Ross, Douglas Densmore, and Christopher A Voigt. Genetic circuit design automation. *Science*, 352(6281):aac7341, April 2016.
- [10] Larry Page, Sergey Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. 1998.
- [11] Philip Shapira, Seokbeom Kwon, and Jan Youtie. Tracking the emergence of synthetic biology. *Scientometrics*, 112(3):1439–1469, 2017.
- [12] Logan Terry, Jared Earl, Sam Thayer, Samuel Bridge, and Chris J Myers. Sbolcanvas: A visual editor for genetic designs. *ACS Synthetic Biology*, 10(7):1792–1796, 2021.
- [13] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A.C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The fair guiding principles for scientific data management and stewardship. *Scientific Data*, 3(1):160018, Mar 2016.
- [14] Eric Yu, Jeanet Mante, and Chris J. Myers. Sequence-based searching for synbiohub using vsearch. *ACS Synthetic Biology*, 11(2):990–995, 2022. PMID: 35060706.
- [15] Michael Zhang, James Alastair McLaughlin, Anil Wipat, and Chris J Myers. Sboldesigner 2: an intuitive tool for structural genetic design. *ACS synthetic biology*, 6(7):1150–1160, 2017.

- [16] Zhang, Michael and Zundel, Zach and Myers, Chris. SBOExplorer: Data infrastructure and data mining for genetic design repositories. *ACS Synthetic Biology*, 8(10):2287–2294, 2019. PMID: 31532640.
- [17] Zach Zundel. *Improving Authentication and Authorization on SynBioHub*. B.S. Thesis, University of Utah, December 2019.