Improving Authentication and Authorization on SynBioHub

Zachary M. Zundel
University of Utah

UUCS-19-008

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

8 December 2019

Abstract

Synthetic Biology is an emerging discipline which uses engineering principles to shape biological behavior. The Synthetic Biology Open Language (SBOL) is a standard for describing biological constructs which enables engineering workflows that previous formats, such as GenBank and FASTA, could not. SynBioHub is an online repository for storing and sharing genetic designs. It uses the SBOL standard and an RDF triplestore to store designs, as well as supporting file attachment and external links. Several research efforts in synthetic biology have adopted SynBioHub and SBOL. These research efforts have revealed key areas for improvement in SynBioHub. Improving user sharing and permissioning is a primary target for improvement. The existing system has basic support for sharing with different privilege levels. Unfortunately, its architecture makes it difficult to extend and improve. Due to this difficulty, many features which would make SynBioHub more collaborative have not been implemented. This work aims to make synthetic biology more collaborative by providing a better foundation for experimentation and innovation in user sharing and permissioning. The existing authentication and authorization (auth) system is not centralized; it mixes concerns between page rendering and permissions management. The new system separates auth into its own software layer, separate entirely from page rendering. This new layer is itself split into separate authentication and authorization steps. New feature development and refinement will be made easier by the strong separations between the different components of SynBioHub.

IMPROVING AUTHENTICATION AND AUTHORIZATION ON SYNBIOHUB

by

Zachary M. Zundel

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

Bachelors of Science in Computer Science

School of Computing

The University of Utah

Dec 2019

Copyright © Zachary M. Zundel 2019
All Rights Reserved

The University of Utah

STATEMENT OF THESIS APPROVAL

The thesis of **Zachary M. Zundel**
has been approved by:

_____ **Chris J. Myers** , Chair **6 Dec 2019**

by **Tom C. Henderson** , Undergraduate Chair of
the Department of **Computer Science**

ABSTRACT

Synthetic Biology is an emerging discipline which uses engineering principles to shape biological behavior. The *Synthetic Biology Open Language* (SBOL) is a standard for describing biological constructs which enables engineering workflows that previous formats, such as GenBank and FASTA, could not. SynBioHub is an online repository for storing and sharing genetic designs. It uses the SBOL standard and an RDF triplestore to store designs, as well as supporting file attachment and external links. Several research efforts in synthetic biology have adopted SynBioHub and SBOL. These research efforts have revealed key areas for improvement in SynBioHub.

Improving user sharing and permissioning is a primary target for improvement. The existing system has basic support for sharing with different privilege levels. Unfortunately, its architecture makes it difficult to extend and improve. Due to this difficulty, many features which would make SynBioHub more collaborative have not been implemented. This work aims to make synthetic biology more collaborative by providing a better foundation for experimentation and innovation in user sharing and permissioning.

The existing authentication and authorization (auth) system is not centralized; it mixes concerns between page rendering and permissions management. The new system separates auth into its own software layer, separate entirely from page rendering. This new layer is itself split into separate authentication and authorization steps. New feature development and refinement will be made easier by the strong separations between the different components of SynBioHub.

To my Moth.

CONTENTS

ABSTRACT . . . . . **iii**

LIST OF FIGURES . . . . . **vii**

ACKNOWLEDGEMENTS . . . . . **ix**

CHAPTERS

1. INTRODUCTION **1**

1.1 Synthetic Biology 1
1.2 Synthetic Biology Open Language 2
1.2.1 Resource Description Framework 4
1.2.2 Data Model 4
1.3 SynBioHub 4
1.4 Web Applications 6
1.4.1 HTTP Requests and Responses 6
1.4.2 Middleware 7
1.5 Security 8
1.5.1 Authentication 8
1.5.2 Authorization 9
1.5.3 Sharing 9
1.6 Contributions 9
1.7 Overview 10

2. EXISTING AUTHENTICATION AND AUTHORIZATION **11**

2.1 Authentication 11
2.1.1 Basic Credentials 11
2.1.2 User Tokens 12
2.1.3 Sessions 13
2.2 Authorization 13
2.2.1 Public and Private Graphs 13
2.2.2 Ownership 14
2.2.3 Share Links 16
2.2.4 Non-SBOL Authorization 16

3. NEW AUTHENTICATION AND AUTHORIZATION **17**

3.1 Authentication 17
3.1.1 Basic Credentials 17
3.1.2 User Tokens 17
3.1.3 Sessions 18
3.1.4 Share Links 19

3.2 Authorization 19
3.2.1 Public and Private Graphs 20
3.2.2 Ownership 20
3.2.3 Share Links 21
3.2.4 Non-SBOL Authorization 22

4. VERIFICATION AND VALIDATION 23

4.1 Verifying the Auth System 23
4.2 Validating the Auth System 24

5. SUMMARY AND DISCUSSION 25

5.1 Summary 25
5.2 Availability of Work 25
5.3 Future Work 25

REFERENCES 27

LIST OF FIGURES

1.1 A comparison of standards for biological data. FASTA can express raw sequences. GenBank can annotate sequences with functional information. SBOL 1 allows for abstraction and composition. SBOL 2 allows for description of the functional interactions with non-DNA components. 3

1.2 A block diagram of SynBioHub's architecture. The majority of the figure represents one or many servers configured to run SynBioHub's components. The bottom right shows different users (software tools or researchers) interacting with the components. 6

1.3 The SynBioHub homepage. This is from `synbiohub.org`, the reference instance. 7

1.4 A prototypical example of middleware. A request comes in and is inspected by the middleware. This can involve annotating the request with additional data and sending it on to later processing steps, or rejecting the request. 8

2.1 A block diagram of the current auth system. If authentication data is part of either a request's session or its headers, they go through the `express-session` middleware or `authenticate` function, respectively. Authenticated requests and requests with no authentication data are then sent on to the page rendering routines, optionally via the `requireUser` or `requireAdmin` functions. Checking to ensure the request is properly authorized happens in these functions as well as embedded in the page rendering routines. 12

2.2 SynBioHub's interface for viewing a user's submissions. Constructs residing in their private graph are shown on the left. Constructs that have been moved to the public graph are shown on the right. 14

2.3 SynBioHub's current interface for granting ownership permissions to a user. This view is accessed from a button on the part page. 15

2.4 SynBioHub's interface for viewing parts which you have been given ownership of. This view is accessed by clicking the 'Shared with Me' button in the top navigation bar. 15

2.5 SynBioHub's current interface for displaying share links. This view is accessed via a button on the part page. 16

3.1 A block diagram for the new auth system. The authentication step remains the same, though the authentication routine is modified in the new authentication system. Now, however, all requests go through a single authorization step before being passed further down the line. This authorization step serves as a single pane of glass which all requests must pass through. 18

3.2 The new interface for managing a construct's sharing. Users and anonymous share links are shown separately, and each can be revoked. 19

3.3 The proposed workflow for sharing with a specific user. This step now allows selection of the permission level to grant to this user. 21

3.4 The proposed first step in generating a share link. This step now allows for description of the share link's purpose to allow for future management, as well as selection of the permission level to grant those who use the share link. 22

3.5 The proposed second step in generating a share link. The generated link is unique each time one is generated and encodes information about the permissions granted with the share link. 22

ACKNOWLEDGEMENTS

This thesis would not have been possible without Dr. Chris Myers, my advisor. I would like to thank him for the opportunities he has provided me to grow and learn as a student and a person. All the work I have done is a result of his patience, guidance, and willingness to argue me into a sensible path. Thank you, Chris.

SynBioHub would not exist without the work of James McLaughlin at Newcastle University. Jake Beal, Oliver Flatt, Adam Krusic, Jet Mante, Tramy Nguyen, Ethan Ransom, Meher Samineni, James Scholz, Anil Wipat, Michael Zhang, and countless others provided suggestions, bug reports, and valuable feedback. The thesis document was tremendously improved by the help of Heather Palmer.

This work was conducted under funding from Google Summer of Code, National Science Foundation grants DBI-1356041 and CCF-1748200, and DARPA award FA8750-17-C-0229 via MIT.

CHAPTER 1

INTRODUCTION

1.1 Synthetic Biology

Synthetic biology is a research discipline which applies engineering techniques to biological designs to produce novel behavior [5]. One way to produce this novel behavior is through the design of *genetic circuits*. Instead of using electrons and semiconductors to represent information, a genetic circuit uses DNA, proteins, and small molecules. The cellular mechanisms of protein synthesis, chemical activity, and genetic regulation arise from the interactions of these components and produce a behavior. One example of a circuit is the genetic toggle switch [8]. The genetic toggle switch represents a single bit of digital memory which can be toggled by manipulating extracellular concentrations of specific small molecules, and its current state is represented by a fluorescent protein. To create the toggle switch, a small DNA fragment which contained the necessary components was synthesized and inserted into a cell.

The genetic circuit model is extremely useful because many techniques from electronic circuit design can serve as springboards for biological behavior [17]. For example, asynchronous circuit design involves the creation of systems which allow for communication and computation without a global clock regulating the exchange of information [15]. This paradigm is applicable to biology because of the difficulty of creating a biological clock with well-timed cycles that can communicate with all parts of a biological system. Another important concept borrowed from electronic design is that of decoupling abstract behavior from concrete implementation. This means that a designer begins with a high-level description of their desired functionality, and then works from a library of parts with known behavior to implement it. In electronic design, this functionality can be specified in a descriptive language such as Verilog, and the implementation can be automatically

generated by different design tools [14, 16, 25, 28]. Though this level of design automation is not yet present in the synthetic biology space, several efforts are currently under way to reach it. Libraries of parts are being developed and characterized, so that they can be automatically selected according to a specification [18]. Biological contexts, such as specific bacteria or even cell-free environments, are being characterized and standardized so that they can be used as platforms for engineered circuits with minimized interference. Tools are being developed which apply techniques such as technology mapping and logical synthesis to biological contexts [25]. Finally, data standards are being developed to enable portable and powerful expression of biological designs at all stages of the design workflow [22].

1.2 Synthetic Biology Open Language

Often, synthetic biology involves the synthesis of genetic sequences according to designed specifications [19]. These specifications are currently encoded in a wide variety of formats [13]. **Figure 1.1** on the following page describes some of these formats and the information they contain.

The simplest format, FASTA, simply encodes a sequence alongside a simple textual comment. The sequence is typically a DNA nucleotide sequence, but FASTA has support for RNA and protein sequences. A related format, FASTQ, allows for the inclusion of quality information for each element of the sequence. Since FASTA and FASTQ do not allow for annotating a sequence with specific functional information, it is not particularly well-suited to synthetic biology, which relies heavily on selecting and integrating parts based on their function.

Another popular format, GenBank, is also based around DNA sequences. Unlike FASTA, GenBank allows for the communication of significantly more information about the sequence, including annotating specific regions with their function. Unfortunately, GenBank is not a data standard. There is no single canonical resource to determine what a 'GenBank file' should and should not contain, or how common features should be described. This leads to the proliferation of several different 'dialects' of GenBank, which makes communicating between tools a complex problem. Each tool must be able to ingest and export their 'dialect' of GenBank, as well as any other 'dialects' they wish to communicate with.

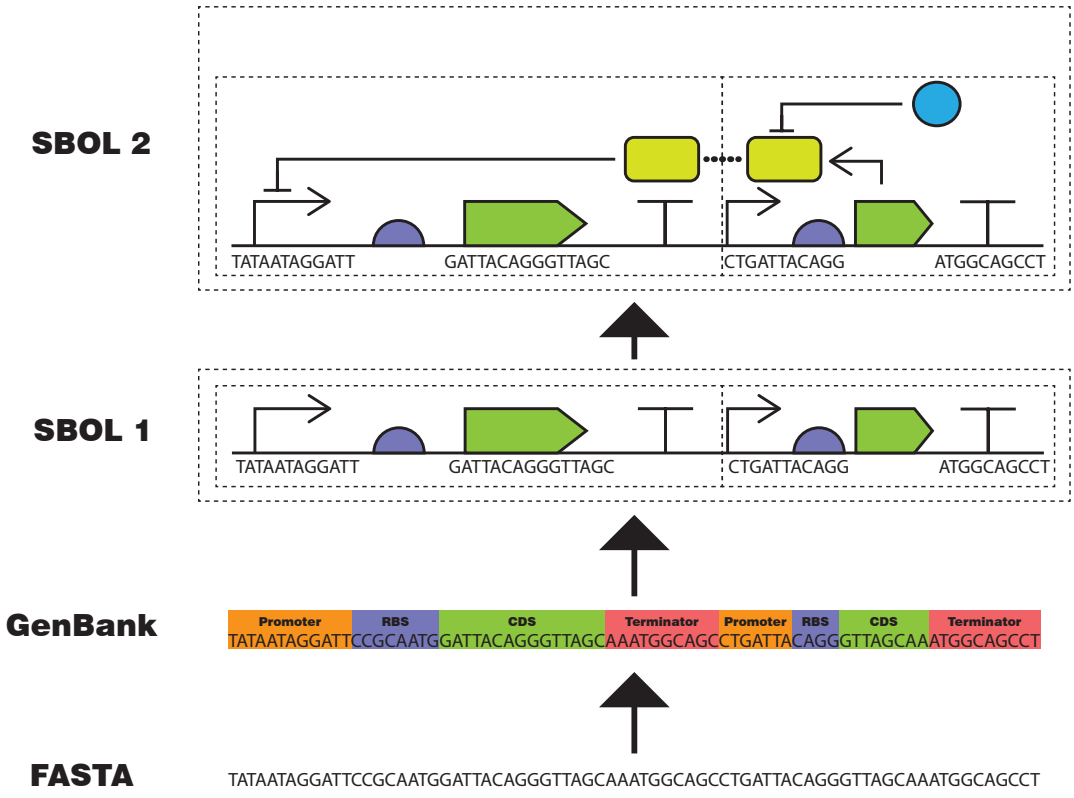

Figure 1.1. A comparison of standards for biological data. FASTA can express raw sequences. GenBank can annotate sequences with functional information. SBOL 1 allows for abstraction and composition. SBOL 2 allows for description of the functional interactions with non-DNA components.

Finally, GenBank is hampered by its fundamental link to the DNA sequence. Engineering design typically involves defining a high-level function, which is then implemented by searching through libraries of building-block parts with known functionality and interfaces. This means that a synthetic biologist may wish to communicate information about designs which have not been fully implemented. GenBank is fundamentally unable to communicate about these designs, so it is also a poor platform for synthetic biology.

Since the existing formats for representing genetic designs are inadequate, the *Synthetic Biology Open Lanugage* (SBOL) was developed [2, 6, 7, 22, 23]. The Synthetic Biology Open Language is a **data standard**. This means that there is a document describing in precise detail what comprises valid SBOL and how common elements should be described.

SBOL supports many useful engineering workflows, such as visualizing designs [3, 20] and combinatorial designs [21]. SBOL version 1 supports only structural information, while SBOL version 2 supports both structual and functional information. There are also several software libraries for interacting with data encoded in SBOL [1, 11, 27]

1.2.1 Resource Description Framework

SBOL leverages the *Resource Description Framework* (RDF) to describe data [12]. This means that any set of SBOL data is represented as a graph, with nodes identified by *Uniform Resource Identifiers* (URIs). Nodes are directionally linked by *predicates*. The source node in a link is the *subject*. The sink node in a link is the *object*. Predicates are, themselves, objects and may link to other nodes which provide more information about the predicate. A subject-predicate-object group is called a *triple*. For this reason, databases which store RDF data are often called *triplestores*.

1.2.2 Data Model

SBOL has a complex *data model*, which describes the structure of SBOL data on a higher level than raw triples. The data model defines several classes, each of which represents a type of data and contain both fields and links to other classes. The specifics of the SBOL data model are not pertinent to this work, but there are a few pieces of information which are necessary to understand. A subset of the classes defined in the SBOL data model are *top-level* classes, meaning that they can stand on their own as a piece of information. Other classes must be linked to at least one top-level class, they do not carry meaning on their own. One of these top-level classes is **Collection**, which represents a group of other top-levels. Some other common top-level classes are **ComponentDefinition** and **ModuleDefinition**, which represent structural and functional units. These top-levels are commonly referred to as *parts* or *modules*.

1.3 SynBioHub

SynBioHub is an application which allows for the storage and sharing of designs encoded in SBOL [9, 10]. SynBioHub is developed in collaboration with the University of Utah and Newcastle University. It is composed of many different components which interact in a variety of ways. **Figure 1.2** on page 6 gives a high-level view of SynBioHub's

architecture. Designed as a web application, the various components which make up SynBioHub can reside on one or many web servers. Most users interact with the core SynBioHub application, at the center of the figure, which is written in JavaScript using node.js. This SynBioHub core relies on a few components. User, session, and some configuration data is stored in a SQL database managed by SQLite, top right. The SynBioHub core communicates with the SQL database using the Sequelize.js library. When SBOL data are uploaded, they are processed by the SBOL processor (top center), which is written in Java. The SynBioHub core communicates with the SBOL processor using JSON sent over the processor's standard input and output channels. SBOL data stored in SynBioHub are saved in the RDF triplestore Virtuoso, bottom left. Virtuoso is third-party software developed by OpenLink Software, and SynBioHub uses version 7.2. SynBioHub and Virtuoso communicate over an HTTP interface. Optionally, SynBioHub's search can be improved by using SBOLExplorer (left middle) [26]. SBOLExplorer provides an HTTP API for searching, directly interacts with Virtuoso to ingest data to search, and stored information in ElasticSearch (top left).

SynBioHub can be installed on a web server for private use and there is a reference instance maintained for public use (**Figure 1.3** on page 7). Several research efforts, such as the Living Computing Project and DARPA SD2 have integrated SynBioHub into their research workflows. Though it is proving useful in these environments, they have exposed several areas of weakness which merit additional work.

One particular pain point is around sharing and permissions. Because SynBioHub is designed to be used by a research group, a single instance typically has many users. These users often need to collaborate, sharing parts, editing them, and publishing them, and SynBioHub currently has limited capabilities in these areas [24]. SynBioHub currently has functionality to grant unlimited, irrevocable read-only access to a part, or revocable ownership of a part. These are useful, but do not represent the full scope of what is useful for collaboration. Additionally, new features such as user groups are frequently requested. Sharing and permissions are currently managed in several places. Some of these are small functions which modify or audit requests before they are rendered. A significant amount of the functionality is in the same places as pages are rendered. Because there are many different pages, which are rendered by different functions, modifying the behavior of each

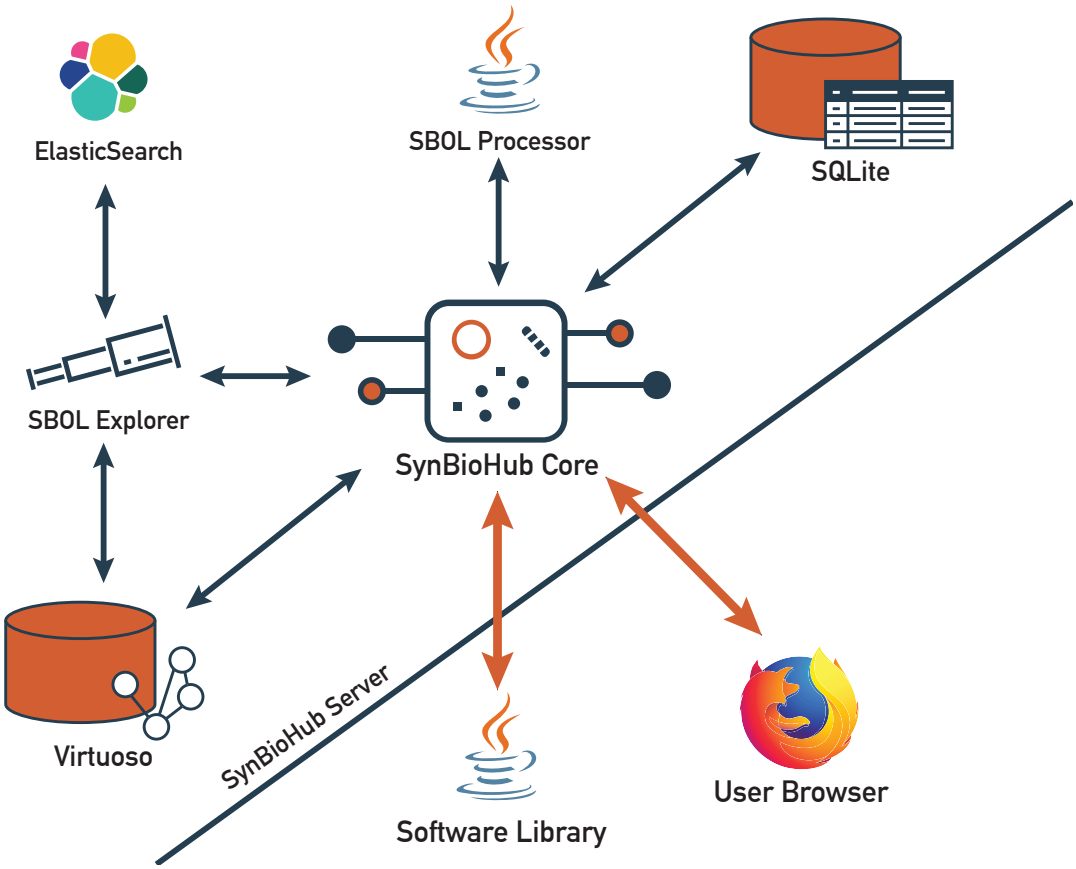

Figure 1.2. A block diagram of SynBioHub's architecture. The majority of the figure represents one or many servers configured to run SynBioHub's components. The bottom right shows different users (software tools or researchers) interacting with the components.

one is time-consuming and error-prone. This has led to very slow iteration on existing functionality, and completely hampered adding new functionality.

1.4 Web Applications

SynBioHub is a web application. This means that it appears to users as a web site with dynamically generated content. The content is generated by the SynBioHub application in response to *Hypertext Transfer Protocol* (HTTP) requests.

1.4.1 HTTP Requests and Responses

HTTP is the standard protocol for serving web pages on the Internet [4]. An HTTP client will send a request to a server, which returns a response. An HTTP request is made to a server and contains several pieces of information. First, it will specify the specific

Figure 1.3. The SynBioHub homepage. This is from `synbiohub.org`, the reference instance.

function to be invoked or information requested by use of the *path*, which specifies a specific *HTTP endpoint*. Additionally, a request can contain several *HTTP headers* which can specify many things, such as the types of information the client can handle, authentication information from the client, or how a response should be encoded. Finally, a request can contain *parameters* either encoded in the URL (for *GET* requests) or in the body of the request (for *POST* requests).

When a server receives a request, it will generate a response. The response will have an *HTTP status code*, which is a numerical code with a defined meaning. Codes range from 100-599, with each group of 100 representing a broad category. The 100s are informational codes, 200s are successful responses, 300s result in redirection, 400s mean that there was some client error, and 500s mean there was some server error. The response can also contain HTTP headers, much like the request. Finally, the response will contain a *body*, which indicates the result of performing the request.

1.4.2 Middleware

Because processing HTTP requests is often a multi-step process, many web applications use the *middleware* design to separate concerns. A middleware is a piece of code which

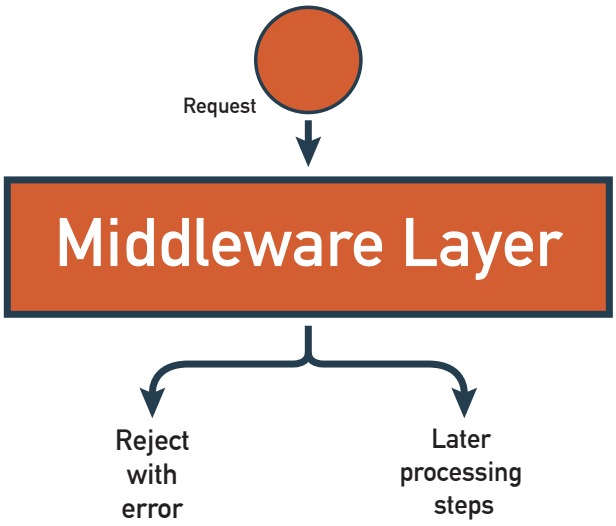

Figure 1.4. A prototypical example of middleware. A request comes in and is inspected by the middleware. This can involve annotating the request with additional data and sending it on to later processing steps, or rejecting the request.

is designed to perform a single function on an HTTP request. Typically, middleware is regarded as separate from the core logic of a web application. **Figure 1.4** on the current page shows a prototypical example of how a middleware functions.

When a request reaches a middleware, it is inspected. The middleware can decide that the request should not be allowed to continue for some reason, and send an HTTP response indicating this to the client. In the normal case, however, the middleware will modify the request in some way (such as annotating it with additional data) and then pass it to the next handler. This handler may be another middleware, or it may be the core response logic for the request.

1.5 Security

A primary concern with modifying how sharing works on SynBioHub is the security of uploaded parts. Since things uploaded to SynBioHub often represent active research, it is important that they are not improperly accessed.

1.5.1 Authentication

Frequently, the actions that someone can perform on a part (viewing it, editing it, sharing it) depend on who they are. Thus, the first step in an access control pipeline

is often *authentication*. Authentication is the proccess of identifying someone based on some credential they provide. Frequently, this is a username and password, though there are a few ways which a user can authenticate. Authentication typically involves cross-referencing the provided credential with some known information (a user database, for example). Any new information linked to the credential is then provided to the rest of the pipeline.

1.5.2 Authorization

The next step in the pipeline is *authorization*. This involves verifying that the authenticated user is able to perform the function they are attempting. They may be authorized because of who they are (they are manipulating an object they own) or because of rights afforded to them (manipulating an object shared with them). Authorization has levels; it is possible to be authorized to view something, but not edit it. These levels are typically referred to as *privileges*.

1.5.3 Sharing

Sharing is the act of changing another user's authorization on an object. Sharing inherently includes a notion of permissions. For example, you can share an object with another user by giving them access to view it.

1.6 Contributions

This thesis draws inspiration from common tools such as Google Docs to create a more powerful model for sharing and permissions on SynBioHub. Additionally, users such as the SD2 research effort were consulted to ascertain which parts of the existing model are causing collaborative friction. Once the target behavior was understood, it was implemented in SynBioHub. Existing authentication and authorization functionality are consolidated to a single location the the SynBioHub source tree. Much of the page rendering routines are simplified based on the assumption that the authorization concern is separated from the rendering concern. Finally, to validate the usefulness of this work, new sharing modes are added. These included augmenting share links to allow them to grant custom permissions, as well as granular sharing on a per-user basis.

1.7 Overview

Chapter 2 describes the current state of SynBioHub's auth system. Implementation details for both authentication and authorization, as well as the current system's limitations, are discussed. chapter 3 details the proposed new system. Each element discussed in chapter 2 is re-examined, and its role and implementation for the new system are defined. Chapter 4 is an analysis of the system's ability to replace its predecessor, as well as its extensibility for new use cases.

CHAPTER 2

EXISTING AUTHENTICATION AND AUTHORIZATION

The existing authorization and authentication (together called auth) system is distributed across SynBioHub. While many of the authentication functions are somewhat neatly separated from the rest of the code, most of the authorization checking is embedded in routines to render pages. This makes changing the authorization behavior of SynBioHub difficult. As a second-order effect, because authentication and authorization are closely linked, the changes that can be made in authentication are also limited.

Figure 2.1 on the following page provides an overview of the current authentication system. Users are authenticated using small middleware functions, and then most access control is done in the page rendering routines.

2.1 Authentication

SynBioHub uses three main methods to authenticate users. They can be authenticated based on basic credentials, user tokens, or user sessions.

2.1.1 Basic Credentials

Basic credentials are a username and password. Both user tokens and user sessions rely on an initial authentication using basic credentials. This method of authentication involves sending a username and password to the `login` endpoint. The login code hashes the password with a secret string, and then queries the SQL database for users which match the username and password hash. If any are found, that user is then authenticated.

If the login request is sent over a browser, SynBioHub creates a new session. The session has some metadata, including which user is authenticated for that session. The metadata is stored in the SQL database, and a random key for the session is generated. The key is sent back to the user to use as authentication for future requests.

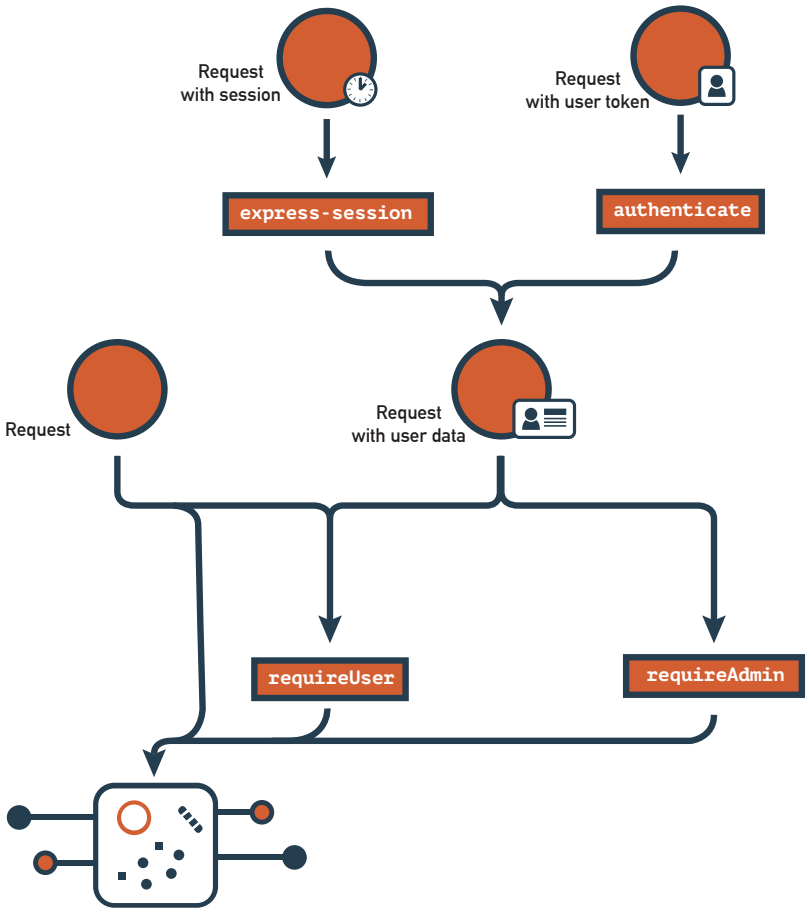

Figure 2.1. A block diagram of the current auth system. If authentication data is part of either a request's session or its headers, they go through the `express-session` middleware or `authenticate` function, respectively. Authenticated requests and requests with no authentication data are then sent on to the page rendering routines, optionally via the `requireUser` or `requireAdmin` functions. Checking to ensure the request is properly authorized happens in these functions as well as embedded in the page rendering routines.

If the login request is sent programmatically, the `Accept` HTTP header must have type `text/plain`, and SynBioHub instead responds with the authenticated user's token. The user token is valid for the lifetime of that SynBioHub, and can be used to authenticate future requests.

2.1.2 User Tokens

To authenticate a request using a session token, the token obtained from basic authentication should be included in the `X-authorization` HTTP header. SynBioHub inspects

requests for this header, and if it is included, it looks up the token in an in-memory structure. If the token is valid, and associated with a user, that user is successfully authenticated for the request.

2.1.3 Sessions

Sessions are primarily handled by the `express-session` library. The library looks in the `Cookie` HTTP header for a session key. The session key is then matched in the SQL database to find session data. This session data can include information about an authenticated user. If so, the user is then authenticated for the remainder of the request.

2.2 Authorization

Once a user's request has been authenticated, its authorization should be checked. There are a few different types of authorization checks that can be performed. There are two main categories of checks: SBOL-related checks, and SynBioHub-related checks. SBOL-related checks are performed on requests which interact with SBOL data, and involve checking graph permissions, validating share links, and checking for ownership. SynBioHub-related checks are performed on requests which have SynBioHub-related functionality, such as modifying instance settings. There is one type of interaction which does not neatly fit into one classification or another: in order to make a part publicly accessible, both an SBOL-related check and a SynBioHub-related check are performed.

2.2.1 Public and Private Graphs

SynBioHub segments SBOL data uploaded to SynBioHub into different graphs. These graphs are the default method for managing access control on individual SBOL items. **Figure 2.2** on the next page shows how graphs are presented to the user in SynBioHub.

The public graph contains parts which are viewable by any user, including unauthenticated users simply browsing a SynBioHub. Parts in the public graph are typically uploaded by a user, and then curated and moved to the public graph by a different user known as a curator. Generally, parts contained in the public graph cannot be changed, though there are a few fields which can be. These fields can only be edited by the part's owner, which is generally the original uploader.

Each user has their own graph, and they have complete access to the parts in their

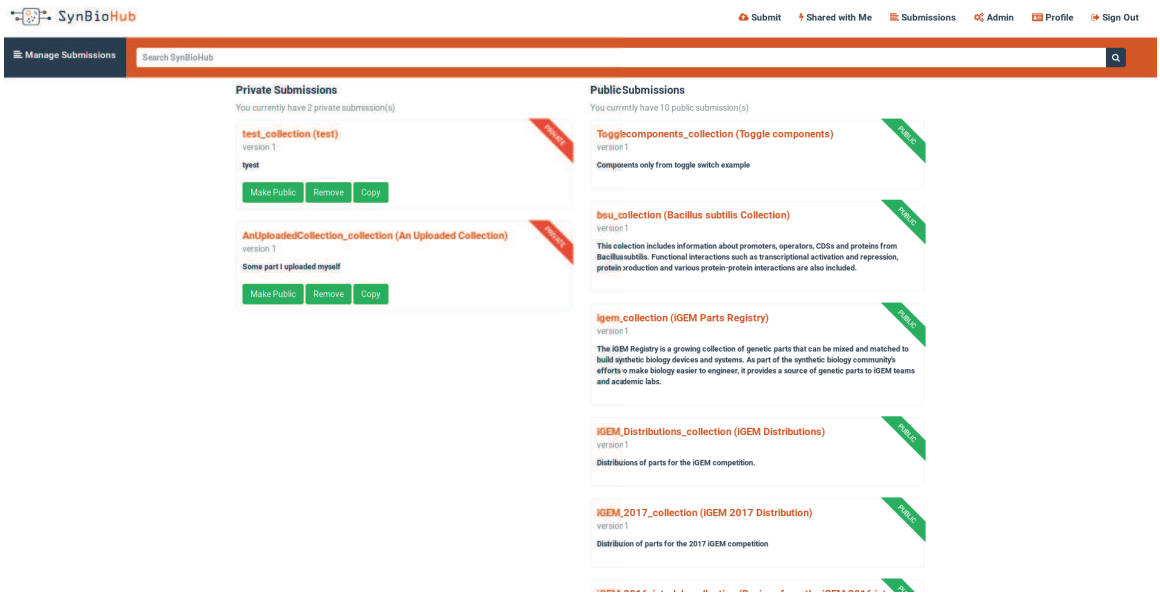

Figure 2.2. SynBioHub's interface for viewing a user's submissions. Constructs residing in their private graph are shown on the left. Constructs that have been moved to the public graph are shown on the right.

graph. When a request attempts to interact with a part in a private graph, the owner of the private graph and the request's authenticated user are compared. If they do not match, the request is not authorized to proceed.

When a user uploads new SBOL, it is deposited into their graph, and they are added as an owner. The part can then be modified, replaced, and shared with other users. When a user is ready for a curator to move their part into the public graph, they must share the part with the curator. The curator can then move the part into the public graph.

2.2.2 Ownership

To give a user editing privileges on a part, they must be added as an owner of the part. From a part page, a user can navigate to the page show in **Figure 2.3** on the following page to add an owner. This operation adds a triple to the SBOL top level for the part which links it to the new owner via the **ownedBy** predicate.

When a user has been given access to a part, they can view it using the Shared with Me page shown in **Figure 2.4** on the next page. To find all parts shared with a user, SynBioHub looks through all user graphs for triples with the **ownedBy** predicate and the desired user as a subject. These parts are rendered in the shown view, so they can be accessed by the

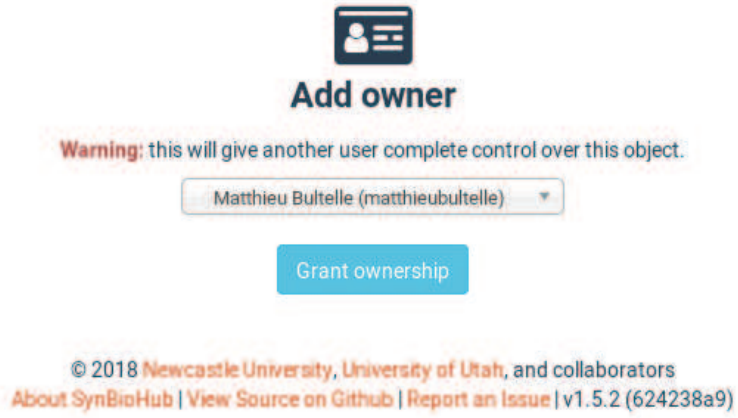

Figure 2.3. SynBioHub's current interface for granting ownership permissions to a user. This view is accessed from a button on the part page.

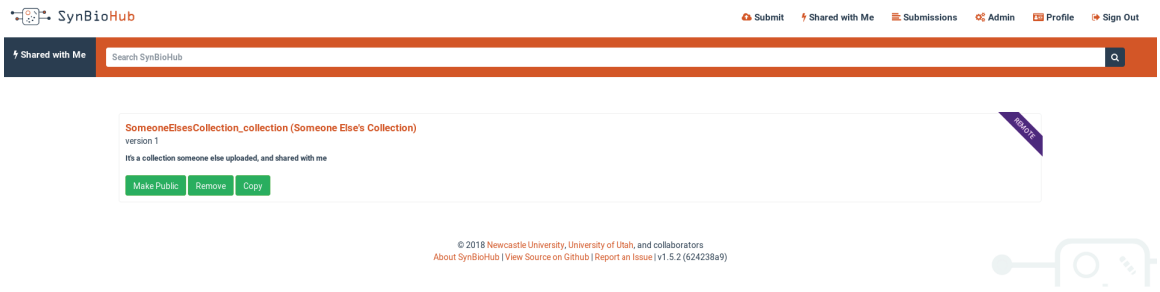

Figure 2.4. SynBioHub's interface for viewing parts which you have been given ownership of. This view is accessed by clicking the 'Shared with Me' button in the top navigation bar.

user.

When a user owns a part, they can perform any modifications as if the part is contained in their private graph. This includes modifying, deleting, and sharing the part. Ownership can be revoked, but there is no way to give read-only or other reduced access to a specific

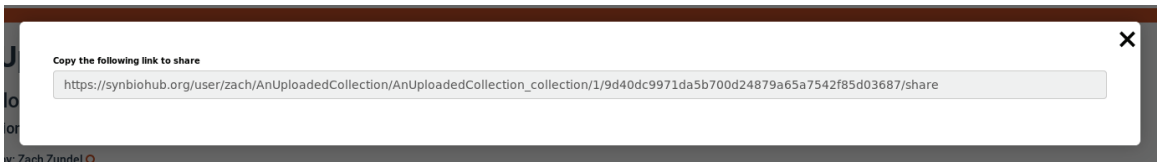

Figure 2.5. SynBioHub's current interface for displaying share links. This view is accessed via a button on the part page.

user.

2.2.3 Share Links

Share links are modified URLs that include a hash which grants read-only access to a part. Each part page has a button which generates the dialog seen in **Figure 2.5** on this page containing the share link.

These share links are useful, but have a few problems. First, since information about a part on SynBioHub is spread across many pages, it is important that all the links on a page work. This means that the rendering function must know if the page was accessed via a share link, and replace all the regular links with share links in the rendered page. Additionally, the share link hash is generated from the URI of the part which is shared. This means that it is impossible to change or invalidate the share link, so once a link has been shared, this access can never be revoked. Finally, share links only offer read-only access to a part. There is no way to change the privilege of a share link.

2.2.4 Non-SBOL Authorization

Some requests do not involve SBOL data. One category is requests having to do with administrating a SynBioHub. These can involve changing instance settings, registering the instance with the Web of Registries, and administering users. Another is curating parts, only some users have the necessary privileges to move parts into the public graph. Finally, some pages are only viewable to logged in users, like the profile modification page. These authorizations are checked by the `requireAdmin`, `requireCurator`, and `requireUser` functions, respectively, as shown in **Figure 2.1** on page 12. These functions are fairly simple: they check whether a request has an authenticated user, and if it does, they inspect it for the admin or curator flags.

CHAPTER 3

NEW AUTHENTICATION AND AUTHORIZATION

The new auth system is, to a first approximation, a refactor of the existing system to properly separate concerns between page rendering and authorization. The new authorization functionality, combined with slightly modified versions of the existing authentication, are the proposed auth system. **Figure 3.1** on the following page shows a flow diagram for how requests are processed in the new system. The primary change is that all requests transit through a new authorization middleware before ultimately arriving at the page rendering routines. This 'single pane of glass' approach allows for easier modification of authorization behavior.

3.1 Authentication

Authentication in the new module largely works the same way, but the functionality has been pulled out of the `app.js` file and into the auth source tree.

3.1.1 Basic Credentials

Looking up the username and password hash in the SQL database remains the same.

3.1.2 User Tokens

User tokens are still stored in an in-memory data structure. This means they do not persist between restarts of SynBioHub. Instead of creating a single token for each user as they log in, multiple tokens can correspond to a single user. Tokens can be invalidated or expire after a certain timeframe or number of uses. This means that tokens can be issues for single accessions, or only for short time frames, which is better for interacting with applications. For example, if a user would like to give a third-party application to their parts to edit, this gives them finer control over how the application can access their graph.

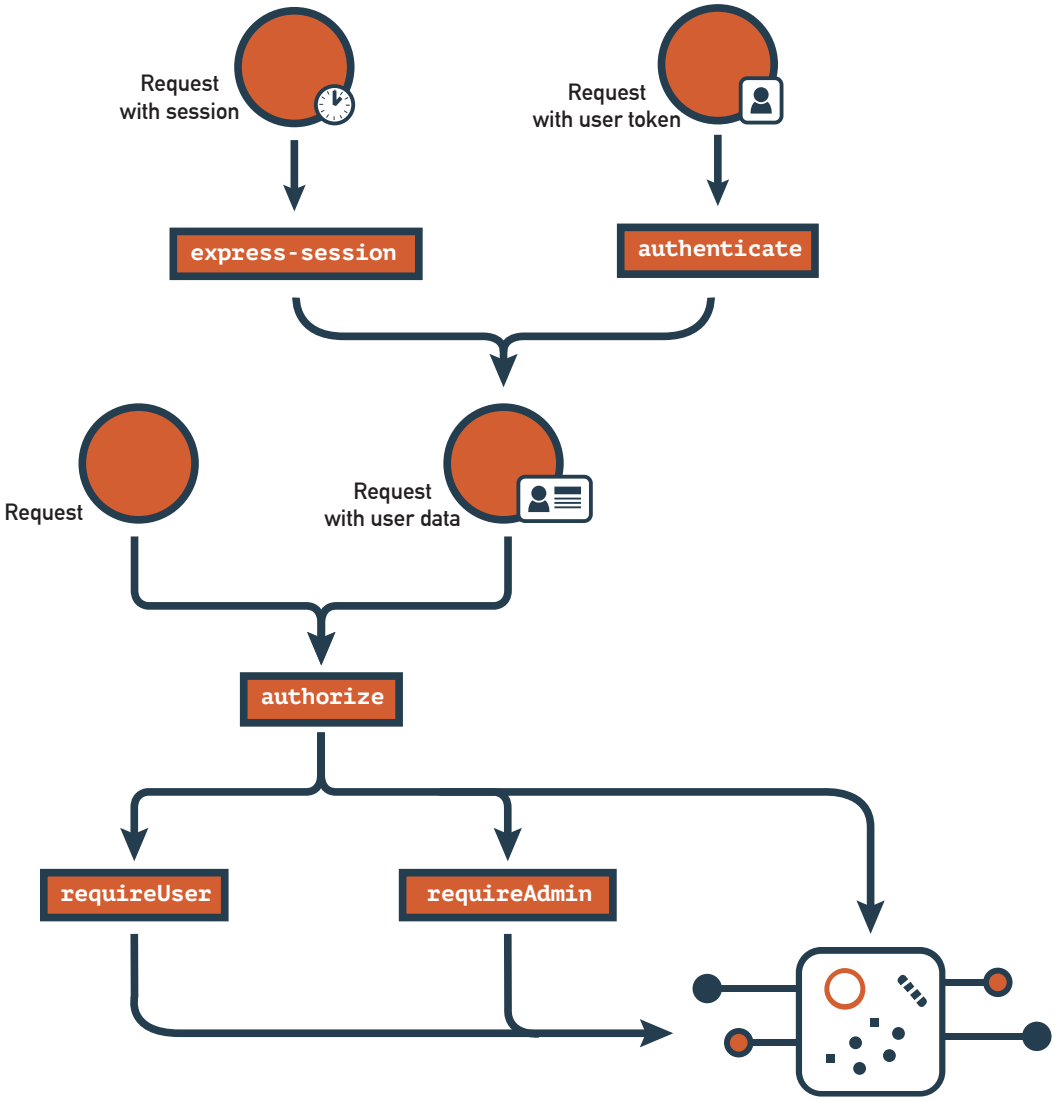

Figure 3.1. A block diagram for the new auth system. The authentication step remains the same, though the authentication routine is modified in the new authentication system. Now, however, all requests go through a single authorization step before being passed further down the line. This authorization step serves as a single pane of glass which all requests must pass through.

3.1.3 Sessions

Sessions remain largely the same between the old and new systems, in both they are managed by a third-party library. The primary difference is that a single session may have multiple authenticated users. This is because privileges for new-style share links are accorded to virtual users. One session may have zero or one complete users authenticated,

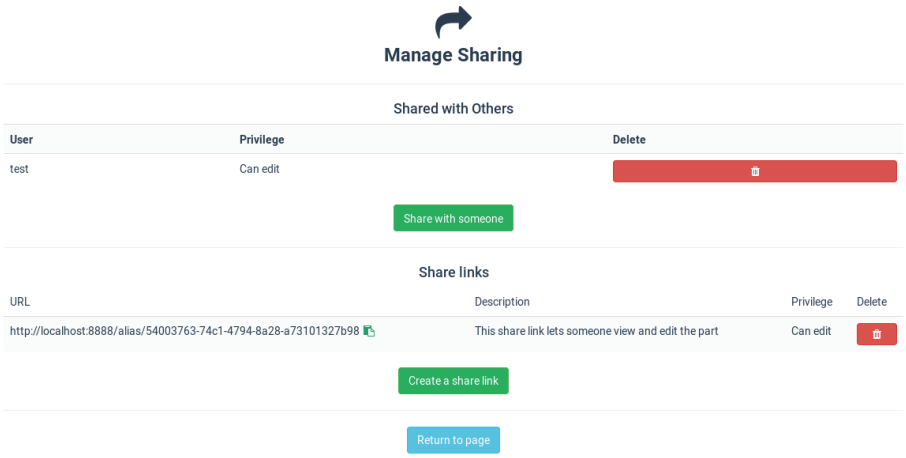

Figure 3.2. The new interface for managing a construct's sharing. Users and anonymous share links are shown separately, and each can be revoked.

and any number of virtual users.

3.1.4 Share Links

The most significant change to authentication in the new auth system is the new share links. Old-style share links worked by hashing a part URI with a secret string, which was then inserted into the URL. This meant that the access could never be revoked, because the URI of a part never changed. The new-style share links can authenticate additional users into a session. These 'virtual users' have access to the part for which the share link corresponds. This is done by searching the SQL database's new `Alias` for the share token in the link. An `Alias` entry contains information about the share link's virtual user, and the part it grants access to. The virtual user is authenticated into the session, and the request is redirected to the part's rendering routine.

3.2 Authorization

Authorization represents most of the new code in this work. **Figure 3.2** on the current page shows the new interface created for managing who is authorized to access a part. This is accessible for any part, and allows an owner to grant privileges to other specific users or generate share links which can grant privileges to anyone.

To support the new authorization scheme, an `Auth` table is added to the database. This table contains information about additionals rights afforded to users. When a user is given

some rights to an object, they are also given rights recursively to any objects which are accessible from the root object. Each URI to which a user is granted access is given a row in the `Auth` table, which also contains the user ID granted access, the level of privilege granted, and how the privilege was granted. For example, if access is given to an object because it is a child of another object, that is how the privilege was granted. This means that the `Auth` table holds representations of permission trees, one tree each time a user is given access to an object.

3.2.1 Public and Private Graphs

Public and private graph accession defaults are the same as the previous auth system. Now, when a request for a part passes through the authorize step, a few checks are performed.

For parts in the public graph, the default permission is assumed to be *read*. The `Auth` table is queried for any rows which contain both the URI of the part and a user ID which is added to the request in an authentication step. If any are found, the strongest permission is selected, and added to the request. The request is then passed to the page rendering code, which uses the permission in its rendering.

For parts in the private graph, the default permission is assumed to be *none*. The first check performed is if the user who owns the private graph is authenticated on the request. If so, the *owner* permission is added to the request, which is passed to the rendering code. If not, the `Auth` table is queried for any rows which give a user on the request access to the part. If any are found, the strongest permission is selected, added to the request, and the page is rendered. If none are found, the authorize module replies with a `404 Not Found` error, to prevent leaking information about parts in the private graphs.

3.2.2 Ownership

Ownership is implicit for parts in a user's own private graph, but the new sharing interface shown in **Figure 3.2** on the preceding page allows a user to grant permissions to other specific users. Unlike the previous system, these 'ownership' permissions do not imply complete access to the part. As shown in **Figure 3.3** on the next page, the specific privileges accorded to the user are now configurable.

When a user is granted ownership over the part, a new authorization tree is created

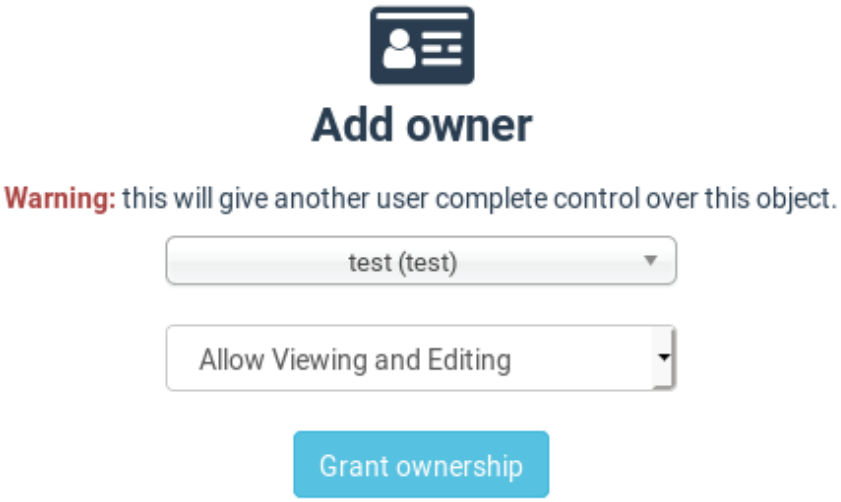

Figure 3.3. The proposed workflow for sharing with a specific user. This step now allows selection of the permission level to grant to this user.

using that part as the root. This authorization tree has a node for each child object of the root, and their children recursively. Each node has the same permissions as the root, the URI of the object, and a reference to its parent. Each node corresponds to a single row in the `Auth` table.

3.2.3 Share Links

Share links operate on a different model in the new system. Old-style share links verify their authenticity by providing a hash of the part's URI with a secret string. This means that the share link never changes, and can never be revoked. Additionally, only one share link can exist for a part.

The new system allows for both a privilege level and a reason to be associated with a share link using the interface shown in **Figure 3.4** on the following page. Because the share links are not associated with an actual user in the same way an ownership is, this given reason is used to remind why the share link was generated.

When a share link is created, first a 'virtual user' is created. Then, an authorization tree is created granting the chosen permission to virtual user for the object and its children. The authorization tree is saved in the `Auth` table. Finally, a random token is generated. This

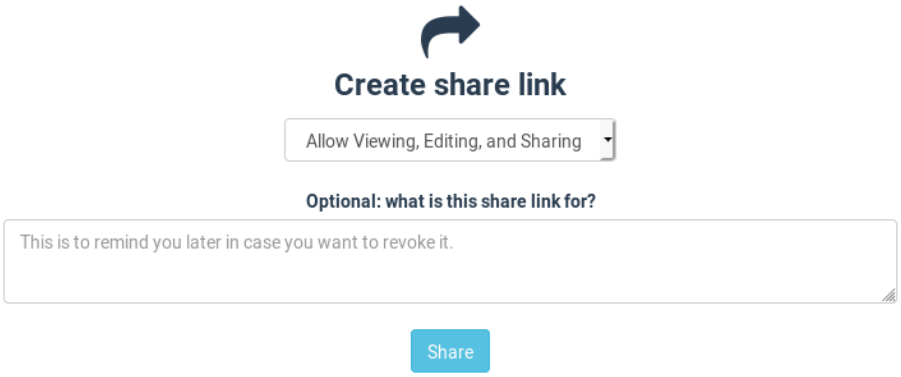

Figure 3.4. The proposed first step in generating a share link. This step now allows for description of the share link's purpose to allow for future management, as well as selection of the permission level to grant those who use the share link.

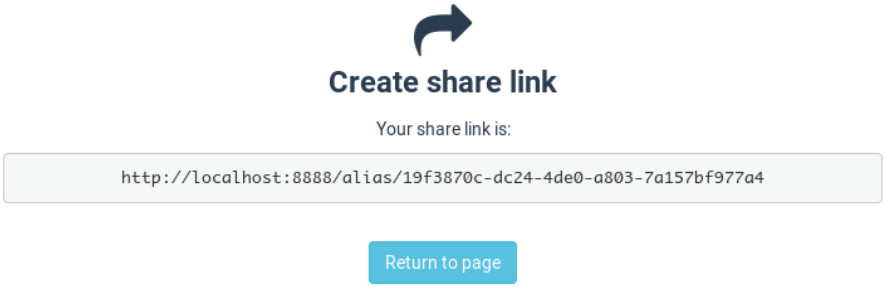

Figure 3.5. The proposed second step in generating a share link. The generated link is unique each time one is generated and encodes information about the permissions granted with the share link.

token is saved in the `Alias` table, alongside the ID of the virtual user and the URI for the root part of the authorization tree. The token is used to construct the share link, which is then given to the user as shown in **Figure 3.5** on the current page.

When a user accesses a part via the share link, the token is used to find the ID of the virtual user and the destination part URI in the `Alias` table. The virtual user is authenticated into the session, and the request is redirected to the destination part. Since the virtual user has some privilege to the part, the request completes successfully.

3.2.4 Non-SBOL Authorization

Non-SBOL authorizations have remained completely unchanged. A future work item could be to refactor them in with the rest of the authorization code.

CHAPTER 4

VERIFICATION AND VALIDATION

The work of this thesis has two main criteria to satisfy. First, no main functionality should be lost from the existing auth system. This is the *verification* criterion; it indicates that the new auth model is at least as powerful as the old one. The second criterion, *validation*, is that the new system is easier to extend than the old one.

4.1 Verifying the Auth System

Verification of the auth system was mostly done manually. The existing SynBioHub integration tests are run against SynBioHub to check that some normal use cases were unchanged. The rest of the verification was done by comparing old functionality to new, stepping through the workflows, and ensuring that the behavior remained unchanged. **Table 4.1** on this page shows how the new system fulfills the same purposes as the old system.

One regression in the new system is that share links do not have the same versatility as in the previous system. Because a share link gave access to all pages accessible from the shared page, accessing a child or related page was simple. One such child page could be the endpoint for downloading GenBank given an SBOL part. The share link itself could be

Table 4.1. Comparing old to new functionality in the auth systems. Each piece of old functionality has an analogue in the new system which is at least as powerful.

Functionality	Old System	New System
Share links	Irrevocable read-only access to anonymous user	Revocable access to anonymous with configurable permission level
Ownership	Revocable, complete access to specific user	Revocable access to specific user with configurable permission level

modified or appended to in order to view a child page. Because the new system is slightly more complex, access to child pages is not guaranteed. Additionally, the token in a share link is now derived from the virtual user for the share link, not from the shared construct. These facts together mean that derived pages can no longer be accessed by modifying the share link. A user who wishes to access derived pages must use the session token they are given upon authenticating with the share link. For those who access SynBioHub in a browser, this is automatically handled by modern browsers. For those who interact with SynBioHub using tha API, this introduces some extra complexity as they have to manage sessions.

4.2 Validating the Auth System

The old auth system was really two: share links and ownership. Both were handled mostly separately from each other, but integrated with page rendering. The new system works from the assumption that both systems were ultimately built with the same goals in mind, which is part of the motivation for uniting them with a common backend. Additionally, this means that making changes to the common backend enables them for both share links and ownership. This leads to a slightly different framing of the new system: anonymous and specified user permissions.

The two functionalities of the old system are available in the new system, and were able to be easily extended in the new one. They were so simple to add that they were in the first revision of the new auth system. In fact, it would have been more complex to introduce the same restrictions as the old system. This provides evidence that one of the more ephemeral goals of the new auth system, extensibility, has been satisfied.

CHAPTER 5

SUMMARY AND DISCUSSION

5.1 Summary

This work was motivated by the desire to extend SynBioHub's existing auth system. Issues with the current system were identified. These include feature deficits as well as structural issues which made modifying the current behavior difficult. To address these problems, a new auth system was developed. This involved defining a new model for authorization, implementing that model, and redesigning portions of SynBioHub to properly separate concerns. The new system was verified by ensuring that there was minimal feature loss in the migration to the new system. The value of the work was validated by extending the new system to allow for granular privilege selection within the existing share link/owner features.

5.2 Availability of Work

SynBioHub is available on GitHub at `https://github.com/SynBioHub/synbiohub`. The majority of the work in this thesis is embodied in pull request #1019. The reference instance of SynBioHub can be accessed at `https://synbiohub.org`.

5.3 Future Work

Immediately upon completing the new auth module, several avenues for future work are unblocked.

One feature request which will likely take immediate priority is groups. For several years, the ability to have group ownership and sharing of objects has been an open feature request. One way that this could be implemented is by extending the virtual user concept to include groups as a virtual user. When a user is authenticated, so would be all virtual users corresponding to their groups. These virtual users might be slightly more powerful

than those used in share links, as they could have their own graphs to which parts can be submitted.

Another feature could be federated authentication for SynBioHubs. Because SynBioHubs can federate with each other via the Web of Registries, it is common to interact with multiple different instances while browsing. It would be useful to have some way of tying users between SynBioHubs together, so that privileges could be managed across the entire ecosystem.

The work of this thesis represents a step forward in SynBioHub's extensibility, and will unblock development of several features which are key to collaboration. This collaboration will hopefully decrease the friction of performing standards-based synthetic biology research, leading to new and exciting research developments.

REFERENCES

[1] B. A. Bartley, K. Choi, M. Samineni, Z. Zundel, T. Nguyen, C. J. Myers, and H. M. Sauro, *pySBOL: A Python Package for Genetic Design Automation and Standardization*, ACS Synth. Biol., 8 (2019), pp. 1515–1518.

[2] J. Beal, R. S. Cox, R. Grünberg, J. McLaughlin, T. Nguyen, B. Bartley, M. Bissell, K. Choi, K. Clancy, C. Macklin, C. Madsen, G. Misirli, E. Oberortner, M. Pocock, N. Roehner, M. Samineni, M. Zhang, Z. Zhang, Z. Zundel, J. H. Gennari, C. Myers, H. Sauro, and A. Wipat, *Synthetic Biology Open Language (SBOL) Version 2.1.0*, Journal of Integrative Bioinformatics, 13 (2016).

[3] J. Beal, T. Nguyen, T. Gorochowski, A. Goñi-Moreno, J. Scott-Brown, J. McLaughlin, C. Madsen, B. Aleritsch, B. Bartley, S. Bhakta, M. Bissell, S. Castillo-Hair, K. Clancy, A. Luna, N. Le Novere, Z. Palchick, M. Pocock, H. Sauro, J. Sexton, and A. Wipat, *Communicating structure and function in synthetic biology diagrams*, ACS Synthetic Biology, 8 (2019).

[4] M. Belshe, R. Peon, and M. Thomson, *Hypertext Transfer Protocol Version 2 (HTTP/2)*, Tech. Rep. RFC7540, RFC Editor, May 2015.

[5] S. A. Benner and A. M. Sismour, *Synthetic biology*, Nat Rev Genet, 6 (2005), pp. 533–543.

[6] R. S. Cox, C. Madsen, J. A. McLaughlin, T. Nguyen, N. Roehner, B. Bartley, J. Beal, M. Bissell, K. Choi, K. Clancy, R. Grünberg, C. Macklin, G. Misirli, E. Oberortner, M. Pocock, M. Samineni, M. Zhang, Z. Zhang, Z. Zundel, J. H. Gennari, C. Myers, H. Sauro, and A. Wipat, *Synthetic Biology Open Language (SBOL) Version 2.2.0*, Journal of Integrative Bioinformatics, 15 (2018).

[7] M. Galdzicki, K. P. Clancy, E. Oberortner, M. Pocock, J. Y. Quinn, C. A. Rodriguez, N. Roehner, M. L. Wilson, L. Adam, J. C. Anderson, B. A. Bartley, J. Beal, D. Chandran, J. Chen, D. Densmore, D. Endy, R. Grünberg, J. Hallinan, N. J. Hillson, J. D. Johnson, A. Kuchinsky, M. Lux, G. Misirli, J. Peccoud, H. A. Plahar, E. Sirin, G.-B. Stan, A. Villalobos, A. Wipat, J. H. Gennari, C. J. Myers, and H. M. Sauro, *The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology*, Nat Biotechnol, 32 (2014), pp. 545–550.

[8] T. S. Gardner, C. R. Cantor, and J. J. Collins, *Construction of a genetic toggle switch in Escherichia coli*, Nature, 403 (2000), pp. 339–342.

[9] C. Madsen, J. A. McLaughlin, G. Misirli, M. Pocock, K. Flanagan, J. Hallinan, and A. Wipat, *The sbol stack: a platform for storing, publishing, and sharing synthetic biology designs*, ACS synthetic biology, 5 (2016), pp. 487–497.

[10] J. A. McLaughlin, C. J. Myers, Z. Zundel, G. Misirli, M. Zhang, I. D. Ofiteru, A. Goñi-Moreno, and A. Wipat, *SynBioHub: A Standards-Enabled Design Repository for Synthetic Biology*, ACS Synth. Biol., 7 (2018), pp. 682–688.

[11] J. A. McLaughlin, C. J. Myers, Z. Zundel, N. Wilkinson, C. Atallah, and A. Wipat, *sboljs: Bringing the Synthetic Biology Open Language to the Web Browser*, ACS Synth. Biol., 8 (2019), pp. 191–193.

[12] E. Miller, *An introduction to the resource description framework*, Bulletin of the American Society for Information Science and Technology, 25 (1998), pp. 15–19.

[13] L. Mills, *Common File Formats*, Current Protocols in Bioinformatics, 00 (2003), pp. A.1B.1–A.1B.18.

[14] G. Misirli, T. Nguyen, J. A. McLaughlin, P. Vaidyanathan, T. S. Jones, D. Densmore, C. Myers, and A. Wipat, *A Computational Workflow for the Automated Generation of Models of Genetic Designs*, ACS Synth. Biol., 8 (2019), pp. 1548–1559.

[15] C. J. Myers, *Asynchronous circuit design*, Wiley, New York, NY, 2001. OCLC: 248544994.

[16] C. J. Myers, J. Beal, T. E. Gorochowski, H. Kuwahara, C. Madsen, J. A. McLaughlin, G. Misirli, T. Nguyen, E. Oberortner, M. Samineni, A. Wipat, M. Zhang, and Z. Zundel, *A standard-enabled workflow for synthetic biology*, Biochm. Soc. Trans., 45 (2017), pp. 793–803.

[17] T. Nguyen, T. Jones, P. Fontanarrosa, J. Mante, Z. Zundel, D. Densmore, and C. Myers, *Design of asynchronous genetic circuits*, Proceedings of the IEEE, PP (2019), pp. 1–13.

[18] A. A. K. Nielsen, B. S. Der, J. Shin, P. Vaidyanathan, V. Paralanov, E. A. Strychalski, D. Ross, D. Densmore, and C. A. Voigt, *Genetic circuit design automation*, Science, 352 (2016), pp. aac7341–aac7341.

[19] J. Peccoud, J. C. Anderson, D. Chandran, D. Densmore, M. Galdzicki, M. W. Lux, C. A. Rodriguez, G.-B. Stan, and H. M. Sauro, *Essential information for synthetic DNA sequences*, Nat Biotechnol, 29 (2011), pp. 22–22.

[20] J. Y. Quinn, R. S. Cox III, A. Adler, J. Beal, S. Bhatia, Y. Cai, J. Chen, K. Clancy, M. Galdzicki, N. J. Hillson, et al., *Sbol visual: a graphical language for genetic designs*, PLoS biology, 13 (2015), p. e1002310.

[21] N. Roehner, B. Bartley, J. Beal, J. McLaughlin, M. Pocock, M. Zhang, Z. Zundel, and C. Myers, *Specifying combinatorial designs with the synthetic biology open language (sbol)*, ACS Synthetic Biology, 8 (2019).

[22] N. Roehner, J. Beal, K. Clancy, B. Bartley, G. Misirli, R. Grünberg, E. Oberortner, M. Pocock, M. Bissell, C. Madsen, T. Nguyen, M. Zhang, Z. Zhang, Z. Zundel, D. Densmore, J. H. Gennari, A. Wipat, H. M. Sauro, and C. J. Myers, *Sharing Structure and Function in Biological Design with SBOL 2.0*, ACS Synth. Biol., 5 (2016), pp. 498–506.

[23] F. Schreiber, G. D. Bader, P. Gleeson, M. Golebiewski, M. Hucka, N. L. Novère, C. Myers, D. Nickerson, B. Sommer, and D. Waltemath, *Specifications of Standards in Systems and Synthetic Biology: Status and Developments in 2016*, Journal of Integrative Bioinformatics, 13 (2016), pp. 1–7.

[24] U. Urquiza-García, T. Zieliński, and A. J. Millar, *Better research by efficient sharing: evaluation of free management platforms for synthetic biology designs*, Synthetic Biology, 4 (2019), p. ysz016.

[25] L. Watanabe, T. Nguyen, M. Zhang, Z. Zundel, Z. Zhang, C. Madsen, N. Roehner, and C. Myers, *iBioSim 3: A Tool for Model-Based Genetic Circuit Design*, ACS Synth. Biol., 8 (2019), pp. 1560–1563.

[26] M. Zhang, Z. Zundel, and C. J. Myers, *SBOLExplorer: Data Infrastructure and Data Mining for Genetic Design Repositories*, ACS Synth. Biol., (2019), p. acssynbio.9b00089.

[27] Z. Zhang, T. Nguyen, N. Roehner, G. Misirli, M. Pocock, E. Oberortner, M. Samineni, Z. Zundel, J. Beal, K. Clancy, A. Wipat, and C. J. Myers, *libSBOLj 2.0: A Java Library to Support SBOL 2.0*, IEEE Life Sci. Lett., 1 (2015), pp. 34–37.

[28] Z. Zundel, M. Samineni, Z. Zhang, and C. J. Myers, *A Validator and Converter for the Synthetic Biology Open Language*, ACS Synth. Biol., 6 (2017), pp. 1161–1168.