

# **A universal and RESTful web-based metabolites database with pathway design functionality**

**Group 23**

# Content

1. Introduction .....	3
1.1 Task Assignment.....	4
2. Implementation.....	5
2.1 Tech Stack and System Architecture .....	5
2.2 Dataset .....	6
2.3 ER Model.....	7
2.4 Relational Schema and Functional Dependencies .....	8
2.5. Implementation of Metabolic Web Application. ....	10
2.6 Evaluation/Testing.....	12
3. Lessons Learned.....	14
3.1 Languages, Frameworks and API's .....	14
3.2 Creating the Database .....	14
4. Conclusion .....	15
References .....	16
Appendix .....	17

# 1. Introduction

Bioinformatics has been an emerging and promising research field since the late 20th century. The use of bioinformatics has appeared in many different fields of scientific research. Scientists have realized that it is vital to build an efficient database to store the data associated with the field.

Among the numerous databases in bioinformatics, the metabolic database is one of the most important ones. It focuses on data of metabolites, metabolic pathways, metabolite related genes, enzymes, etc. This information greatly helps researchers understand the metabolic process of the different organisms ranging from bacteria to a human cell, which facilitate disease detection, bioproduct production, etc. However, the network of the metabolic pathway is pretty complicated and also contains the cyclic graph, as well as the metabolic reactions involved in the many reactants and products, making it harder to search a metabolic pathway from the database. Also the metabolic databases are usually curated by the genome-scale model. This database design helps to find the pathway or calculate the flux balance analysis in some certain organism, it is less efficient to find the de novo pathway across different organisms.

Therefore, it is crucial to reorganize the traditional genome-scale metabolic model without the model-based constraints. It is also promising to develop a web-based metabolic database that could achieve the efficient pathway finding and redesign. In this project, we build a brand-new bioinformatics database integrating 20 BiGG genome-scale models[1] using the Spring Boot Java framework to create a RESTful web-service with an embedded MySQL database. Our web application includes fundamental functionalities such as the querying and modification of the metabolic database, as well as having the option of leaving comments on the website. We have also added a functionality to search and retrieve the metabolite pathways using an algorithm that combines Breadth First Search (BFS) and MySQL queries.

In the remainder of the paper, we will discuss the steps used to create our web application by first discussing the data used to create a database for our metabolic database and the ER diagram and relational schema derived from this data. Next is the technology used to create our web application (Frontend and Backened) and how that technology was used to implement our website. Evaluation of our website with JUnit will be discussed next. And lastly the lessons learned while creating our application.

## 1.1 Task Assignment

When designing our implementation plan we decided that everyone was going to work on every aspect of the project. We all wanted to gain experience developing the backend, database and frontend parts of our web application.

Team Member	Tasks
Everyone	Designing the Database(ER Model, Relational Model) Contributing to Checkpoints Communicating with Slack Using Git For Version Control Populating the Database Testing Frontend Functionalities Helped each other debug Project Manager role switched week to week
Hui	<ul style="list-style-type: none"><li>• Cleaned that data</li><li>• Worked on backend/frontend for the pathway functionalities and the reaction search page</li><li>• Created the comment functionality</li></ul>
Charles	<ul style="list-style-type: none"><li>• Worked on backend/frontend for all modification pages</li><li>• Add, Update and Delete: Reactions, Genes, Metabolites</li><li>• Created Login Functionality and added Security features</li></ul>
Sam	<ul style="list-style-type: none"><li>• Worked on backend/frontend for genes and metabolite search pages</li><li>• Cleaned up Frontend: Made sure frontend pages flowed well together and added styling (CSS). Created homepage.</li><li>• Wrote JUnit Test</li></ul>

## 2. Implementation

### 2.1 Tech Stack and System Architecture

**Backend:**

- Java
- Java Persistence API and Hibernate - Java API used to interact with relational database
- Spring Boot- Framework for creating Java based applications
- MySQL - Relational Database Management System

**Frontend:**

- HTML, CSS, JavaScript
- Thymeleaf- Java template engine made to map Java objects to HTML and JavaScript.

For our backend MySQL was used for our RDBMS. MySQL was chosen since it was the RDBMS being used in class and works well with the SpringBoot Framework. Java was chosen for our backend programming language since it's the programming language we are most familiar with. To be able to use Java to retrieve and send data from and to our database, as well as send that data to our frontend we used the Spring Boot framework.

We utilized the architectural style called REST(Representational State Transfer) when making our web application. REST is the fundamental architecture that the internet uses. It allows for systems to communicate with textual representations of web resources by using a set of predefined rules. We utilized this architecture to be able to have our frontend and backend communicate with each other efficiently and effectively using post and get requests. Data flows through our app following the logic defined in the Model View Controller software design pattern. The Model manages the data and logic of our application. The model uses the Java Persistence API and Hibernate to use Java to communicate with our MySQL database. The View is the frontend of our application. The Controller responds to the user input and interacts with both the model and view.

Our frontend was created with HTML, CSS and Javascript. The frontend is not directly connected to the database and all user requests must go through the controller first. This allows us to strictly control how the user interacts and manipulates our database. When data in the form of an Java object is sent to the frontend we use "Thymeleaf" a Java template engine made to map objects to HTML or Javascript.

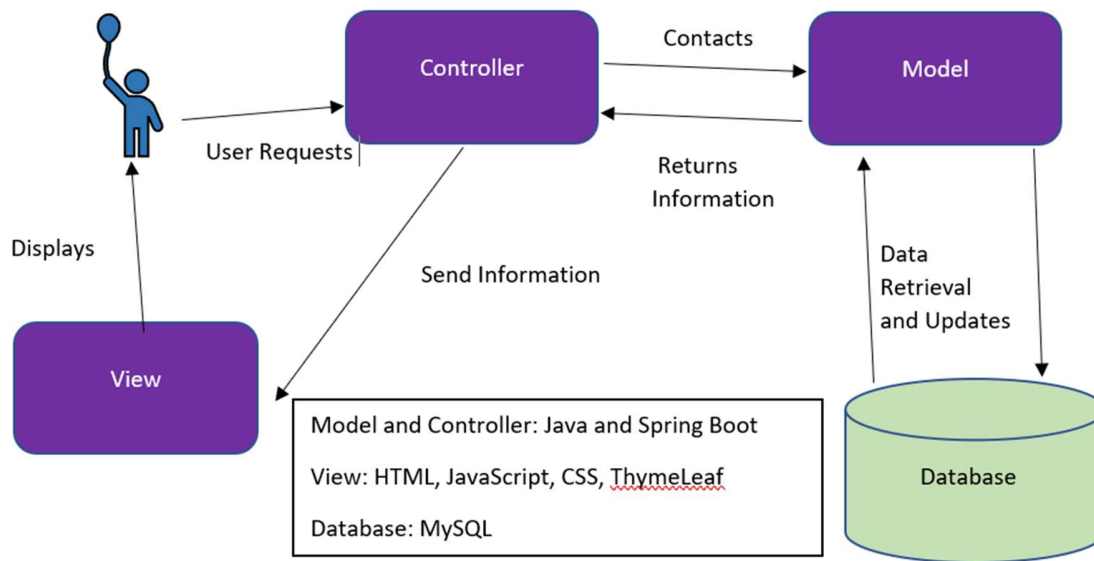


Figure 1. Model View Controller Diagram .

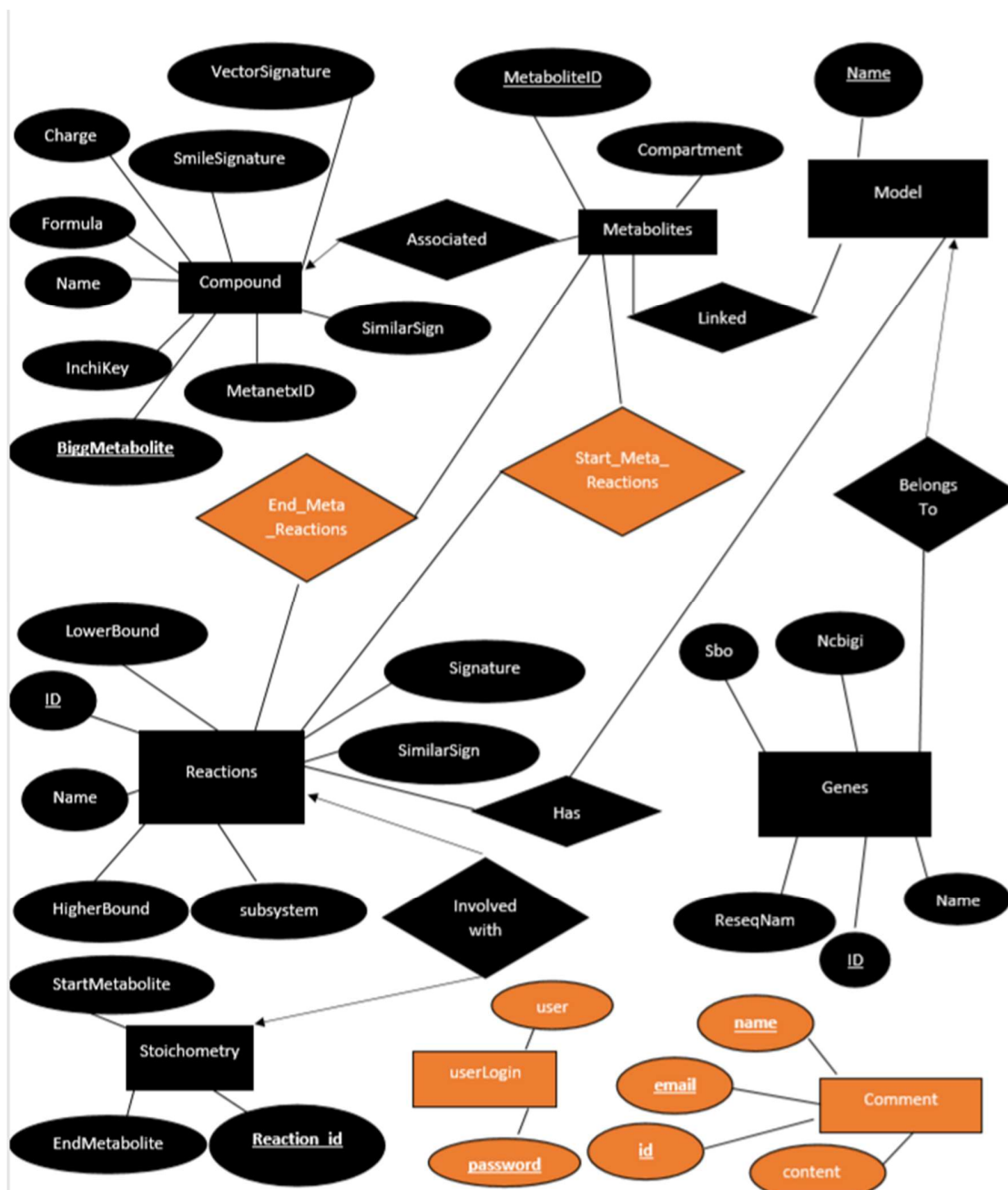
## 2.2 Dataset

Our data is derived from BiGG, a Systems Biology Research Group at the University of California, San Diego.(URL: <http://bigg.ucsd.edu/>)[1]. Their database contains more than 70 published genome-scale metabolic networks with a set of standardized identifiers called BiGG IDs. The data is stored in JSON format.

In our project, We selected the 20 most complete and frequent models to build our database based on their information. These 20 models contain 14069 distinct metabolites, 25193 distinct metabolic reactions, 10584 distinct genes, and 8002 distinct compounds in total. Each of them could be regarded as an independent entity, another entity is the model. There are also relationships between the entities that are shown in detail in the ER model below.

For cleaning up the dataset, we used Python scripts to prescreen the JSON dataset acquired from BiGG database. We carefully selected the core attributes that would be useful for building the database, which will show in the ER model part. Then, we extracted the metabolites, reactions, genes information from these 20 JSON files and integrated them together into a JSON file separately. In this process, we added the model information into the metabolites, reactions, genes entities. We then parsed these JSON files using Java and loaded the data into our database.

## 2.3 ER Model



\*Updated made since Checkpoint 2

## 2.4 Relational Schema and Functional Dependencies

**Compounds**(BiggMetaboliteID: String, MetanixID:String, Name:String,  
InchiKey:String, Formula:String, Charge:String, KeggCompound: String,  
SmileSignature:String, VectorSignature:String, SimilarSign: String)  
Functional Dependencies: BiggMetaboliteID  $\rightarrow$  { MetanixID, Name, InchiKey, Formula,  
Charge, SmileSignature, VectorSignature, SimilarSign}  
Primary Key: BiggMetaboliteID  
No other keys.

**Metabolites**(MetaboliteID: String, Compartment: String)  
Function Dependencies: MetaboliteID  $\rightarrow$  Compartment  
Primary Key: MetaboliteID  
No other keys.

**Associated**(MetaboliteID: String, BiggMetaboliteID: String)  
Function Dependencies: MetaboliteID  $\rightarrow$  BiggMetaboliteID  
Primary Key: MetaboliteID  
Foreign key: BiggMetaboliteID

Can combine the Metabolite and Associated Relation into one relation

**Metabolites**(MetaboliteID:String, Compartment:String, BiggMetaboliteID: String)  
Functional Dependencies: MetaboliteID  $\rightarrow$  {Compartment, BiggMetaboliteID}  
Primary Key: MetaboliteID  
Foreign key: BiggMetaboliteID

**Models**(ModelName: String)  
Primary Key: ModelName

**Linked**(MetaboliteID:String, ModelName:String)  
Primary Key: MetaboliteID and ModelName

**Genes**(GeneID:String, Name:String, RefseqNam:String, Sbo:String, Ncbigi:String)  
Functional Dependencies: GeneID  $\rightarrow$  {Name, RefseqName, Sbo, Ncbigi}  
Primary Key: GeneID  
No other keys.

**BelongsTo**(GeneID:String, ModelName:String)  
Functional Dependencies: GeneID  $\rightarrow$  ModelID  
Primary Key: GeneID and ModelName



Can combine the two relation Genes and BelongsTo into one relation

**Genes**(GeneID:String, Name:String, RefseqNam:String, Sbo:String, Ncbigi:String, ModelName: String)

Functional Dependencies: GeneID  $\rightarrow$  {Name, Refseq, Sbo, Ncbigi, ModelName}

Primary Key: GeneID

**Reactions**(ReactionID:String, Name:String, HigherBound:String, LowerBound:String, Subsystem:String, SimilarSign: String)

Functional Dependencies: ReactionID  $\rightarrow$  {Name, HigherBound, LowerBound, Subsystem, SimilarSign}

Primary Key: ReactionID

No other keys.

**Has**(ReactionID:String, ModelName:String)

No non trivial functional dependencies

Primary Key: Reaction\_ID and ModeName

**Stoichiometry**(ReactionID:String, StartMetabolite:String, EndMetabolite:String)

Functional dependencies: ReactionID  $\rightarrow$  {StartMetabolite, EndMetabolite}

Primary Key: ReactionID

No other keys.

**Userlogin**(User: String, Passport: String)

Functional Dependencies: User  $\rightarrow$  {Passport}

Primary Key: User

No other keys.

**Start\_Meta\_Reaction**(MetaboliteID :String, ReactionID:String)

No non trivial functional dependencies

Primary Key:Reaction\_ID and MetaboliteID

**End\_Meta\_Reaction**(MetaboliteID :String, ReactionID:String)

No non trivial functional dependencies

Primary Key:Reaction\_ID and MetaboliteID

**Comment**(CommentID: String, Name: String, Email: String, Content: String)

Functional Dependencies {CommentID, Name, Email}  $\rightarrow$  {Passport}

Primary Key:CommentID and Name and Email

\*Updates made since Checkpoint 2

## Normalization Forms

Relations	Functional Dependencies	Best Normal Form Achieved
Compounds	FD: BiggMetaboliteID $\rightarrow$ {All Attributes}	BCNF: BiggMetaboliteID is a non-trivial candidate key
Metabolite	FD: MetaboliteID $\rightarrow$ {All Attributes}	BCNF: MetaboliteID is a non-trivial candidate key
Linked	No Functional Dependency	BCNF and 3NF: No FD relation is therefore 3NF and BCNF
Genes	FD: GeneID $\rightarrow$ {All Attributes}	BCNF: GeneID is a non-trivial candidate key
Reaction	FD: ReactionID $\rightarrow$ {All Attributes}	BCNF: ReactionID is a non-trivial candidate key
Has	No Functional Dependency	BCNF and 3NF: No FD relation is therefore 3NF and BCNF
Stoichiometry	FD: ReactionID $\rightarrow$ {All Attributes}	BCNF: ReactionID is a non-trivial candidate key
UserLogin	FD: User $\rightarrow$ {All Attributes}	BCNF: User is a non-trivial candidate key
Start_Meta_Reaction	No Functional Dependency	BCNF and 3NF: No FD relation is therefore 3NF and BCNF:
End_Meta_Reaction	No Functional Dependency	BCNF and 3NF: No FD relation is therefore 3NF and BCNF
Comment	FD: Comment, Name, and Email $\rightarrow$ {All Attributes}	BCNF: Comment, Name, and Email are non-trivial candidate key

## 2.5. Implementation of Metabolic Web Application.

The Model View Controller design pattern helped us define the structure of our project. The model and controller parts were created with Java. Our Java file can be separated into three categories Entity files, JPA Repositories, and Controller files. The Entity classes contained various Java objects that mapped to tables in our database, the

JPA Repositories contained functions that called SQL statements, and the controller files contained the code necessary for our backend and frontend to communicate.

We had one entity class for each of our tables in the database. The entity classes created were “Comment” to store user comments, “Compound” to store compound data, “EndMetaReaction” to store end meta reaction data, “Gene” to store gene data, “Has” to store has data, “Linked” to store linked data, “Metabolite” to store metabolite data, “Model” to store model data, “Reaction” to store reaction data, “StartMetaReaction” to store start meta reaction, “Stoichiometry” to store stoichiometry data, and “Userlogin” to store user information data. We then had JPA Repositories that contained the SQL statements to select, insert, update, and delete data. For example we had a “ReactionRepo” file that stored queries related to the Reaction table entity.

For the frontend to request and send data from and to the database, multiple controller classes were implemented to handle requests from the frontend. In the controller classes we have methods that continuously listen to requests from the frontend such as when a user navigates to a different html page or a user wants to search, insert, delete or modify our database.

The View was created with HTML, Javascript, CSS and Thymeleaf. We used these languages to create various different webpages in our application. We have different pages pertaining to different functionalities. For example we have a “gene.html” page to display information related to genes, “reaction.html” page to display information related to reactions, and so forth. We used Javascript to make our frontend dynamic. Get and post requests were used to send data between our frontend and backend. Thymeleaf is used to map Java objects to a HTML page.

Here is an example of what is happening behind the scenes when one of our webpages is loaded. Lets say a user navigates to our genes html page, a request will be sent to a controller class in Java that handles get/post request from this html. The controller class then calls the java functions containing the necessary SQL queries needed to display the information on the gene page. The Java code then sends the information to the frontend where Thymeleaf will be able to map the java objects to HTML so they are displayed on the page.

A user will be able to freely view information about the metabolites but to be able to update the database (insert, delete, etc..), the user will need to be an administrator. A security system was implemented for an administrator to be able to log in to our admin page and update the database.

## 2.6 Evaluation/Testing

For the backend evaluation of our web application we used the JUnit framework to write tests to verify the SQL queries and data modification SQL statements we created.

JUnit Test Name	Functionality	Status
addGene()	Inserts new Gene to database	Passed
deleteGene()	Deletes Gene from database	Passed
updateGene()	Updates existing attributes of a Gene in database	Passed
addMetabolite()	Inserts new Metabolite to database	Passed
deleteMetabolite()	Deletes Metabolite from database	Passed
updateMetabolite()	Updates existing attributes of a Metabolite in database	Passed
addReaction()	Inserts new Reaction to database	Passed
deleteReaction()	Deletes Reaction from database	Passed
updateReaction()	Updates existing attributes of a Reaction in database	Passed
updateMetaForgeinKeyDoesNotExistsInCompounds()	Checking that a new Metabolite is not added to our database if the value for the bigg_compoundid is not present as the in the compound table	Passed
findGeneByID()	Searches for a Gene using the primary key	Passed
geneLikeSearch()	Searches for all genes with primary keys that start with	Passed

	a specified string	
getMetaID()	Searches for a Metabolite in the in our database using the primary key	Passed
joinCompoundMetabolitesMetaID()	Check to see if correct results are returned when joining the compound and metabolites, when searching for metabolites that start with a certain string	Passed
joinCompoundMetabolitesLikeSearchCompoundName()	Check to see if correct results are returned when joining the compound and metabolites, when searching for compounds names that contain a certain string	Passed
getReaction()	Check to see that the correct reaction data is found when searching by reactionid	Passed
reactionLIKESearch()	Searches for all reactions with primary keys that start with a specified string	Passed
getStoichiometryByID()	Check to see if correct tuple is returned when you search the stoichiometry table by its primary key	Passed
insertNewStoichimetry()	Insert a new Stoichiometry into database	Passed
updateStoichimetry()	Update an existing Stoichiometry	Passed
insertStoichiometryForgeinKeyDoesNotExitsInReactions()	Checking that a new Stoichiometry is not added to our database if the value for the reactionid is not present as the in the	Passed

	reaction table	
deleteReactionCheckStoichiometryDeletedAsWell()	Delete a reaction and make sure Stoichiometry tuple with same reactionid is deleted in Stoichiometry table	Passed
deleteReactionCheckHasDeletedAsWell()	Delete a Reaction and make sure tuple in Has with same reactionid is deleted in Has table	Passed

Given the time constraint of development time for the project we did not write any test for the frontend functionality. We understand the importance of testing the frontend, but since the focus of this class is the database we decided to focus on making sure all our queries with the database were working properly and then trusting ourselves to call the right queries depending on the user input. If we had more time we would have used a framework like Selenium, which you can use to write Java code to simulate user interaction with a website. So for the frontend evaluation part of our functionality we used manual testing to verify our html pages were functioning as expected.

## 3. Lessons Learned

### 3.1 Languages, Frameworks and API's

Our group had a mixture of experiences using the languages and frameworks we used to make this app before starting the project. We want to emphasize Java Persistence and Hibernate because this API relates to the focus of the course, the database. This API was wonderful, it helped us write efficient code and was fun to use as well. It was very satisfying having the contents of a SQL query automatically mapped to a Java object we made. After completing the project it is safe to say that we all have gained valuable knowledge and skills with various languages, frameworks, API's and software design principles.

### 3.2 Creating the Database

Actually designing and creating a relational database in MySQL gave us important hands on experience with interacting with a RDMS. It was also a great way to

apply the knowledge we learned in class, which helped solidify our understanding of the content of the class. Before this project the only experience anyone had with a database in our group was with Firebase which is a NoSQL database that stores data in a JSON format. The way you interact with a NoSQL database is very different from a relational database. Now that the project is complete and the semester is coming to an end we all feel very confident with our ability to interact with a relational database, which is great, because that's why we all signed up for this course.

One key takeaway that we learned is that it's smart to really think hard about your planned functionalities and how they are going to interact with the database. In our project when we initially created an ER Model and Relational Model, they did not include all the relations that we needed to create all the functionalities that we wanted to implement. We then constructed our database with the original plans and later in the semester we had to add more relations to the database when we started to implement more functionalities, specifically the Admin Login, comment and pathways functionalities. This could have been avoided if we thought a little harder in the beginning about how the functionalities of our application were going to interact with the database, when we created our original ER Model and Relational Model. In the grand scheme of things this did not set us back to much, but we will definitely give this more thought in our future projects.

## **4. Conclusion**

In this project, we developed a web-based bioinformatics database. Besides the basic functionality of the web-based database, we also achieve an advance functionality of pathway redesign based on the combination of MySQL and Breadth-first-searching algorithm. It is an interesting project to achieve a basic graph function within the MySQL DBMS. This project was a great way to apply what we learned in class and also learn new languages, frameworks and APIs. As a group we worked very well together and had very good communication while working remotely. We started the project very early in the semester and everyone contributed to creating the app, which allowed us to create a quality, fully functional web application.

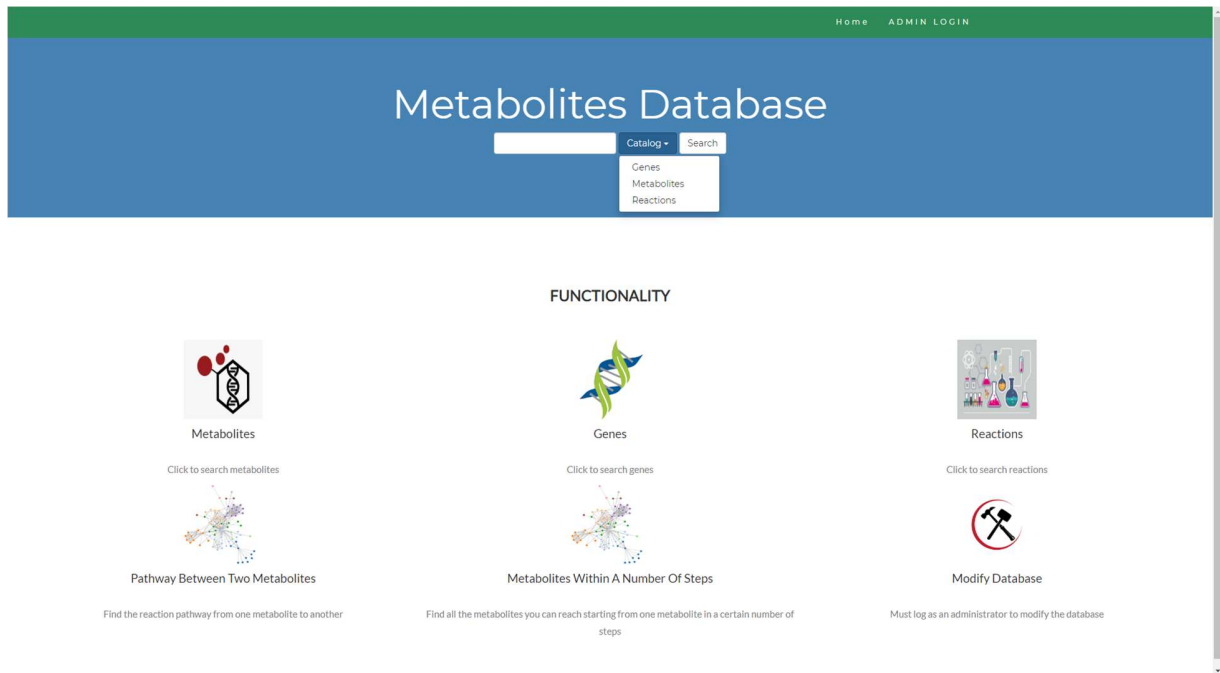
## References

[1] King ZA, Lu JS, Dräger A, Miller PC, Federowicz S, Lerman JA, Ebrahim A, Palsson BO, and Lewis NE. BiGG Models: A platform for integrating, standardizing, and sharing genome-scale models (2016) *Nucleic Acids Research* 44(D1):D515-D522. doi:10.1093/nar/gkv1049

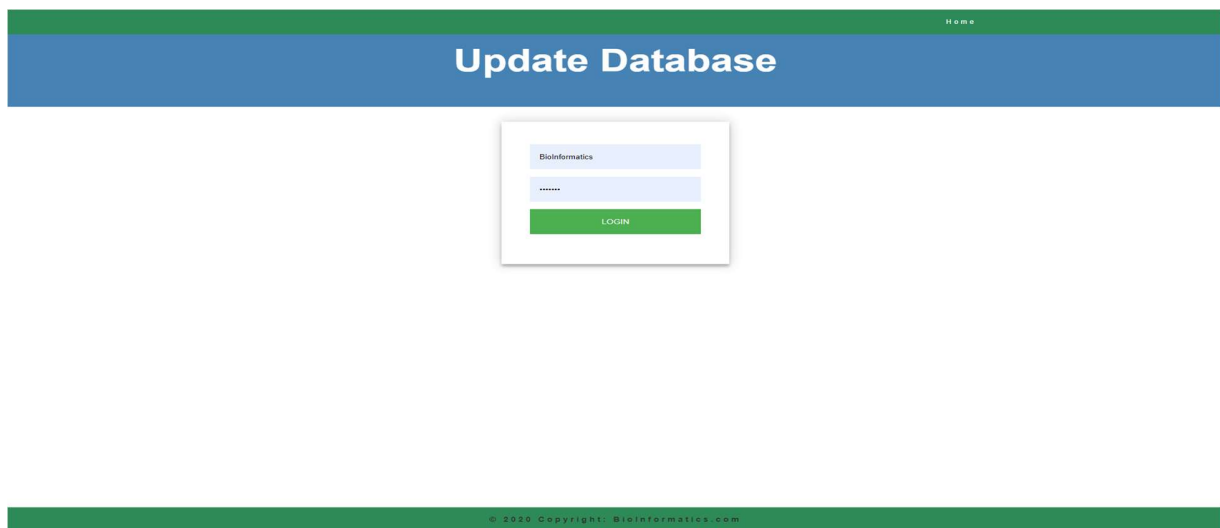


# Appendix

## Homepage



## AdminLoginPage



## Searching Page(Similar page for genes, reactions)

Metabolites		
<input type="text"/>		
<input type="button" value="Search"/>		
No Search Specified: Showing All Metabolites		
ID	Compound ID	Compound Name
10fthf_c	10fthf	10-Formyltetrahydrofolate
10fthf_e	10fthf	10-Formyltetrahydrofolate
10fthf_h	10fthf	10-Formyltetrahydrofolate
10fthf_l	10fthf	10-Formyltetrahydrofolate
10fthf_m	10fthf	10-Formyltetrahydrofolate
10fthf5glu_c	10fthf5glu	10-Formyltetrahydrofolate-[Glu](5)
10fthf5glu_e	10fthf5glu	10-Formyltetrahydrofolate-[Glu](5)
10fthf5glu_l	10fthf5glu	10-Formyltetrahydrofolate-[Glu](5)
10fthf5glu_m	10fthf5glu	10-Formyltetrahydrofolate-[Glu](5)
10fthf6glu_c	10fthf6glu	10-Formyltetrahydrofolate-[Glu](6)
10fthf6glu_e	10fthf6glu	10-Formyltetrahydrofolate-[Glu](6)
10fthf6glu_l	10fthf6glu	10-Formyltetrahydrofolate-[Glu](6)
10fthf6glu_m	10fthf6glu	10-Formyltetrahydrofolate-[Glu](6)
10fthf7glu_c	10fthf7glu	10-Formyltetrahydrofolate-[Glu](7)
10fthf7glu_e	10fthf7glu	10-Formyltetrahydrofolate-[Glu](7)
10fthf7glu_l	10fthf7glu	10-Formyltetrahydrofolate-[Glu](7)
10fthf7glu_m	10fthf7glu	10-Formyltetrahydrofolate-[Glu](7)
10m3ouACP_c	10m3ouACP	10-methyl-3-oxo-undecanoyl-ACP
10m3uACP_c	10m3uACP	10-methyl-3-hydroxy-undecanoyl-ACP
10mdACP_c	10mdACP	10-methyl-dodecanoyl-ACP

## Individual Information Page(Similar page for genes, reactions)

Metabolite Details							
<a href="#">Go Back to Metabolite Search Page</a>							
<input type="button" value="Back"/>							
<p>Metabolite ID: 10fthf6glu_m</p> <p>Compound Name: 10-formyltetrahydrofolate-[Glu](6)</p> <p>Compound ID: 10fthf6glu</p> <p>Charge: NA</p> <p>Formula NA</p> <p>Inchi Key: NA</p> <p>Metanetri Chemical: MNXM3429</p>	<p>Reactions the metabolite can take part in and the model the reaction is associated with.</p> <table><thead><tr><th>Reaction</th><th>Model</th></tr></thead><tbody><tr><td>10FTHF6GLUtm</td><td>Recon3D</td></tr><tr><td>FPGS9m</td><td>Recon3D</td></tr></tbody></table>	Reaction	Model	10FTHF6GLUtm	Recon3D	FPGS9m	Recon3D
Reaction	Model						
10FTHF6GLUtm	Recon3D						
FPGS9m	Recon3D						
<p>See What People Say:</p> <table><thead><tr><th>User</th><th>Email</th><th>User's Thoughts</th></tr></thead><tbody></tbody></table> <p>FEEL FREE TO LEAVE A COMMENT!</p> <p>Name <input type="text"/></p> <p>Email <input type="text"/></p> <p>Description <input type="text"/></p> <p><input type="button" value="Send"/></p>		User	Email	User's Thoughts			
User	Email	User's Thoughts					

## PathSearching Fuctionality 1



Home

# Reaction Pathways

Search to see if there is a reaction pathway between two metabolites

Source

g6p\_c

target

pyr\_c


Knockout Reaction

Search

Source	Product	Pathway
g6p_c	pyr_c	3 steps pathway is : PGI -> F6PA -> DHAPT

© 2020 Copyright: Bioinformatics.com

## PathSearching Fuctionality 2



Home

# Reaction Pathways

Search to find the metabolites you can reach in certain amount of reactions and display the pathways

Source


Number of reaction steps

Search

44 results have been found

Product	Pathway
pep_c	1 steps pathway is : PPDK
dxyl5p_c	1 steps pathway is : DXPS
thdp_c	1 steps pathway is : PYAS
pydx_c	1 steps pathway is : PDYXPT_c
hpyr_c	1 steps pathway is : SPTc
2ahethmpp_c	1 steps pathway is : ACLSs
5g2oxpt_c	1 steps pathway is : ARUH
ptime_c	1 steps pathway is : X00002
alac_s_c	1 steps pathway is : ACLS
pyr_x	1 steps pathway is : PYR12p
actn_R_c	1 steps pathway is : PYRDC2
HC00591_c	1 steps pathway is : r0156
pyr_e	1 steps pathway is : PYRt
for_c	1 steps pathway is : PFL
CE5626_c	1 steps pathway is : RE1919C
4aheth_c	1 steps pathway is : DT0CT2?

## Modifying Pages(Similar page for metabolites, reactions)



HomeLogout

### Add Or Modify A Gene

[Genes](#)  
[Metabolites](#)  
[Reactions](#)

Genes

10FTHF7GLU8

GeneID:

10FTHF7GLU8

Name:

7-glutamyl-10FTHF transpo

NcbiId:

210031293

RefseqName:

Chdh

Sbo:

S

Model:

MM1415

© 2020 Copyright: Bioinformatics.com

## Project Management

Week	Goals	Status	Project Manager
6/21 - 6/27	Populate database with our data. Create a basic layout of our project. Complete Checkpoint 2	Finished	Hui
6/28 - 7/04	Implement on indexing approach(B-Tree). Add some functionality. Begin working on GUI.	Finished	Charles
7/05 - 7/11	Implement SQL Queries. Add more functionality(Inserting, modifying and deleting metabolites, reaction and genes). Continue working on GUI	Finished	Sam
7/11 - 7/18	Add more functionality (Finding shortest path between two metabolites,). Continue working on GUI	Finished	Hui
7/19 - 7/22	Clean up the GUI. Write end final report. Make video of application	Finished	Sam