*Subti*Wiki 3.0: A relational database for the functional genome annotation of the model organism *Bacillus subtilis*

Dissertation

for the award of the degree

"Doctor rerum naturalium”

of Georg-August-Universität Göttingen

within the doctoral program Microbiology und Biochemistry

of the Georg-August University School of Science (GAUSS)

submitted by

Bingyao Zhu

from Jiangsu, P.R. China

Göttingen, 2017

I hereby declare that the doctoral thesis entitled, "*Subti*Wiki 3.0: A relational database for the functional genome annotation of the model organism *Bacillus subtilis*" has been written independently and with no other sources and aids than quoted.

City, date, name

Bingyao Zhu, Göttingen

Thesis Committee

Prof. Dr. Jörg Stülke, Dept. of General Microbiology, Institute for Microbiology and Genetics

PD. Dr. Fabian Commichau, Dept. of General Microbiology, Institute for Microbiology and Genetics

Prof. Dr. Burkhard Morgenstern, Dept. of Bioinformatics, Institute for Microbiology and Genetics

Members of the Examination board

Referee: Prof. Dr. Jörg Stülke

2nd Referee: PD. Dr. Fabian Commichau

Further members of the Examination Board

Prof. Dr. Burkhard Morgenstern, Dept. of Bioinformatics, Institute for Microbiology and Genetics

Prof. Dr. Stephan Klumpp, Institute for Nonlinear Dynamics

Prof. Dr. Ivo Feußner, Dept. of Plant Biochemistry, Albrecht von Haller Institute

Dr. Johannes Soeding, Computational Biology, Max Planck Institut for Biophysical Chemistry

Date of oral examination: 11.01.2018

Acknowledgements

When I arrived at Göttingen almost 5 years ago, I didn’t have a clear picture in my mind how the life would become. Now at the end of 2017, the end of my PhD study, I looked back and concluded that the past few years have been a unique and adventurous journey. It wasn’t always smooth and there were difficulties. But here I am, thanks to the wonderful people around me.

First, I would like to thank my advisor Jörg. I joined his lab during the iGEM project in 2013. After that he introduced me to the *Subti*Wiki project. It is a great project and I really appreciate the opportunity to work on it. I am also thankful for the freedom and the trust he has given me. It wasn’t very clear at the beginning of this project what the outcome could be like. I am also grateful to have Fabian and Burkhard as members of my thesis committee. Their suggestions are valuable to me and this project.

I want to say thank you to Rapha. He was my supervisor when I was doing the lab rotation and the master thesis. It was a great experience working with him. We had exciting and inspiring talks about new ideas and new technologies.

I want to thank all the former and current members of AG Stülke and HIF. I am grateful to Katrin Gunka for her guidance and supervision in the iGEM project and the practical course. The suggestions on *Subti*Pathways from Jonathan Rosenberg are very helpful to me. It is such a great experience to work with Daniel Reuss in the *Minibacillus* project. I am very grateful to David and Maaike for helping with the project. I would like to thank everyone for the nice atmosphere in this department.

I need to thank all my dear friends in Göttingen, Wanwan, Yehan, Yuanzi and Minhui. I am not having enough of those hot pot parties and hikes. It’s nice to have them as friends and let’s keep having fun together.

I would like to thank my parents for their love and support during the study. I would like to thank my boyfriend Chris. I am a lucky woman to have him as an indispensable part of life. His company helped me through the rough time in the past years. I am sincerely grateful.

Table of contents

List of publications

1. Zhu, B. & Stülke, J. (2017) *Subti*Wiki in 2018: from genes and proteins to functional network annotation of the model organism *Bacillus subtilis*. *Nucleic Acids Res.* doi:10.1093/nar/gkx908

2. Reuß, D. R., Altenbuchner, J., Mäder, U., Rath, H., Ischebeck, T., *et al.* (2017) Large-scale reduction of the *Bacillus subtilis* genome: Consequences for the transcriptional network, resource allocation, and metabolism. *Genome Res.* **27,** 289–299

3. Michna, R. H., Zhu, B., Mäder, U. & Stülke, J. (2016) *Subti*Wiki 2.0 - an integrated database for the model organism *Bacillus subtilis*. *Nucleic Acids Res.* **44,** 654–662

4. Reuß, D. R., Commichau, F. M., Gundlach, J., Zhu, B. & Stülke, J. (2016) The Blueprint of a Minimal Cell: *MiniBacillus*. *Microbiol. Mol. Biol. Rev.* **80,** 955–987

5. Morgenstern, B., Zhu, B., Horwege, S. & Leimeister, C. A. (2015) Estimating evolutionary distances between genomic sequences from spaced-word matches. *Algorithms Mol. Biol.* **10,** 5

6. Juhas, M., Reuß, D. R., Zhu, B. & Commichau, F. M. (2014) *Bacillus subtilis* and Escherichia coli essential genes and minimal cell factories after one decade of genome engineering. *Microbiology* **160,** 2341–2351

List of abbreviations

NIH	National Institute of Health
EMBL	European Molecular Biology Laboratory
NCBI	National Center of Biotechnology Information
DDBJ	DNA Data Bank of Japan
PDB	Protein Data Bank
LAMP	Linux, Apache, MySQL, PHP
PHP	Hypertext preprocessor
HTML	Hypertext markup language
DOM	Document object model
CGI	Common gateway interface
CSS	cascade style sheet
XHTML	Extensible Hypertext markup language
XML	Extensible markup language
JSON	JavaScript notation object
AJAX	Asynchronous JavaScript and XML
SQL	Structured query language
DNA	Deoxyribonucleic acid
RNA	Ribonucleic acid
SVG	Scalable vector graph

1 Summary

Biological databases emerged in the 1970s along with the rapid development of information science. Since then, they have greatly helped the research community in data management and information sharing, especially the model organism databases. Model organism databases focus on functional annotations of single well-studied model organisms, such as baker’s yeast, *Escherichia coli* and *Bacillus subtilis. B. subtilis* is a model organism for Gram-positive bacteria. It is of great importance in both labs and the industry and *Subti*Wiki is the model organism database dedicated to it. *Subti*Wiki is based on MediaWiki software and encourages the community to actively participate in the functional annotation of *B. subtilis*. With almost 9 years of constant updating, *Subti*Wiki has reached a state where the data it holds has outgrown the capacity of its engine. The limitations of the MediaWiki software have caused issues in data management, such as data duplication and inconsistency. Therefore, we have decided to migrate the *Subti*Wiki from MediaWiki to a relational database. The new database layout is structured, integrated and flexible as well. We included JavaScript Object Notation format to handle the challenges brought by the data complexity. In addition, batch operations of the data are now possible. Based on this new database layout, we built a content management system. With this system, the data of *Subti*Wiki can still be freely edited by the users and each edit is documented. With all data for protein-protein interaction and gene regulation cleaned from the wiki text, the visualization of large biological networks in *B. subtilis* is possible. Hence, we have included two more *Subti*-Apps, i.e. the interaction browser and the regulation browser. Those network browsers present biological networks at different levels. In addition, we introduced the genome browser for the access of DNA and protein sequences. The new implementation of *Subti*Wiki is user- and developer-friendly. Interactive data visualizations based on web technologies are created for efficient information communication. The modularized design makes *Subti*Wiki easily extendable. With the new database layout and *Subti*-Apps, *Subti*Wiki will continue serving the *Bacillus* research community by providing up-to-date and well-presented functional annotations of *B. subtilis.*

2 Introduction

Biological databases are computer-based information systems for reliable storage and fast access of biological information. They emerged along with the rapid development of information science in the 1970s. The very first computer-based biological database “Protein Data Bank (PDB)” was initiated in the year 1971 [1]. In the same year, E. F. Codd proposed the relation theory for relational databases [2]. In 1982, the United States National Institute of Health (NIH) initiated the GenBank [3] project and simultaneously the European Molecular Biology Laboratory (EMBL) started its own sequence collection. In the middle of the 1980s, NIH and EMBL started the collaboration on data sharing and synchronization. Together with the DNA Data Bank of Japan (DDBJ) [4], the International Nucleotide Sequence Database Collaboration (INSDC) was formed.

From the 1980s until now, there have been a great number of breakthroughs in molecular biology. The amount and variety of biological data grows exponentially. For example, the new sequencing technology made sequencing highly affordable. Therefore, large quantities of raw and processed sequence data have been accumulated. The requirements to store and share such data has led to a variety of different biological databases.

Based on the origin of the data, databases can be classified to primary and secondary databases. The primary databases collect data obtained directly from the experiments while the secondary databases use other databases as source and present combined or processed data views.

According to the type of information collected, databases can be classified into meta, expression, sequence, structure, function databases etc. Meta databases are databases of databases. They can merge data from different sources and present them in a suitable form. For example, the Entrez database from National Center for Biotechnology Information (NCBI) is a meta database. It provides combined search results in different NCBI databases like GenBank, PubMed etc. The expression databases store expression data and most of them are from micro arrays. The sequence databases collect protein and nucleic acid sequences while the structure databases focus on the structure of proteins and RNAs. Functional databases gather functional annotations of biological elements and their associations, such as regulation network or metabolic pathways.

2.1 Characteristics of biological data and databases

Biological data obtained from experiments or natural observation typically have high complexity. This complexity has posed great challenges for data modelling and database design. S.B. Navathe and U. Patil have concluded 9 characteristics of biological data and biological database application from their first-hand experience with MITOMAP, a database for human mitochondrial genome annotations [5]. Those characteristics can be summarized as follows.

Concerning the biological data:

1. **High complexity**. Biological data are highly complex in comparison to other applications such as data of shop inventory or human resources. This requires the biological data model to be able to present complicate schemas and relationships at different levels and apply a combination of structures, i.e. hierarchical, binary, tabular or graph data.
2. **High variety**. Biological information systems are required to be flexible in handling data types in case of outlier values.
3. **Fast evolution**. The schemas of biological database evolve fast. New discoveries in research might require change in data modelling and database design. For example, the discovery of mRNA degradation has added more complexity to the modelling of gene regulation. Hence, the information system should be extendable for the rapidly changing schemas.
4. **Multiple data interpretation and presentation**. The data presentation and terminology might not be consistent from biologist to biologist. The complex biological data can be interpreted in different ways and different data models can be developed. Hence, a mechanism is needed to perform the interchange between database schemas.
5. **Context is important**. Biological data are in organization highly associated. Isolated values do not provide a lot of meanings without context.

Concerning the biological database applications:

1. For a biological database, read-only access is adequate for most users. The search patterns of users are usually beyond the expectation of database developers. Most database

applications implement a user system which allows limited users to edit the content of the database.

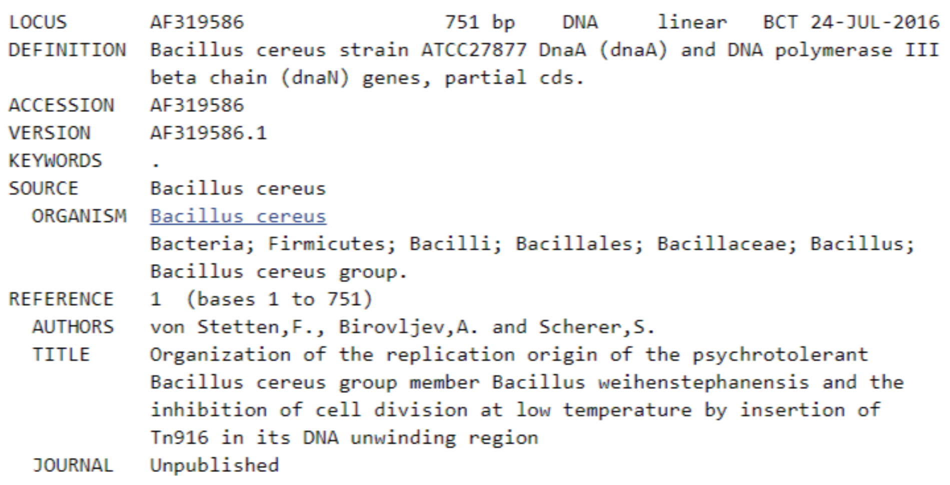
2. Most of the users of biological database do not have the knowledge of structure of databases. A clear instructive graphical interface is very important. The user interface and the user experience of the database application should present the information or work flow in the way applicable to user requests. A certain level of encapsulation would be recommended.
3. As context is important for biological data, complex queries are necessary for users to associate single values together and generate a combined view. The construction of such complex queries would be best done with tools that require no knowledge of detailed data structure.
4. For biological databases, version control is important. Old data should be properly archived for reviewing.

2.2 Implementation approaches of biological databases

A database is simply a collection of well-organized data for easy access and manipulation. In most cases a database management system (DBMS) is needed to keep the database secure, integrate and maintained. The database and the database management system can be in very different formats. The simplest case would be to use flat files to store information and use the file system as the DBSM, just like when we do taxes on our computers.

2.2.1 Flat file databases

For most of the sequence databases such as GenBank, EMBL and DDBJ, the data are stored in flat files with a specific syntax. For GenBank, the syntax is called Abstract Syntax Notation one (ASN.1). A flat file database is easy to initiate but efforts are needed to prevent data duplication and data corruption. In addition, parsers are needed to break the text into data segments and values. This could slow down large batch operations of data.



```
LOCUS       AF319586                 751 bp    DNA     linear   BCT 24-JUL-2016
DEFINITION  Bacillus cereus strain ATCC27877 DnaA (dnaA) and DNA polymerase III
            beta chain (dnaN) genes, partial cds.
ACCESSION   AF319586
VERSION     AF319586.1
KEYWORDS    .
SOURCE      Bacillus cereus
  ORGANISM  Bacillus cereus
            Bacteria; Firmicutes; Bacilli; Bacillales; Bacillaceae; Bacillus;
            Bacillus cereus group.
REFERENCE   1  (bases 1 to 751)
  AUTHORS   von Stetten,F., Birovljev,A. and Scherer,S.
  TITLE     Organization of the replication origin of the psychrotolerant
            Bacillus cereus group member Bacillus weihenstephanensis and the
            inhibition of cell division at low temperature by insertion of
            Tn916 in its DNA unwinding region
  JOURNAL   Unpublished
```

Figure 1. A partial GenBank flat file

Another approach for flat databases would be eXtensive Markup Language (XML). XML supports complicated nested data structures, which makes it very suitable for presentation of biological data.

2.2.2 Relational databases

Relational databases are databases based upon relation theory [2]. In relational database, data are organized in tables. Each row presents an instance while each column stores the value of the corresponding attribute. More details about the relational databases and the Entity-Relationship model will be introduced in the Chapter 3.2.

2.2.3 Object-oriented databases

In object-oriented databases, data are presented in the format of objects. The objects in the object-oriented databases are abstractions of concrete real-world entities, such as a car, a person or a gene. The objects can be described as a collection of attribute-value pairs and the values can also be the references to other objects.

Classes are groups of objects sharing the same properties. They can also be viewed as templates to create objects like the corresponding concept in object-oriented programming. A class can have sub classes. For example, the class “Animal” is of higher abstraction and it could have sub classes such as “dog”, “cat” or “mouse”.

The object-oriented databases have a lot of benefits. They are very well suited for data of high complexity. However, due to the lack of successful commercial implementation, there are no major biological databases using this implementation approach.

2.2.4 Biological Wikis

There are different ways to build a biological database application. Setting up a biological wiki is certainly one of the simplest ways. A "wiki" is a web site whose content can be freely modified by its users in a collaborative way. It usually runs on a wiki software like MediaWiki. Technically speaking, wiki software packages are mature and closed content management systems with their own database implementation and server-side applications.

Because it is very easy to set up a wiki without any knowledge of programming, there have been quite a few biological wikis providing platforms for sharing microarray data or functional annotation. *Subti*Wiki is one of them.

Name	**Description**
ArrayWiki	A community-maintained system for sharing public microarray data repositories and meta-analyses
BOWiki	An ontology-based wiki for annotation of data and integration of knowledge
EcoliWiki	A wiki-based community resource for *Escherichia coli*
ESND	A wiki-based English-to-Chinese scientific nomenclature dictionary
Gene Wiki	A wiki harnessing community intelligence to the annotation of human gene and protein function
GONUTS	A community-based browser and usage guide for Gene Ontology (GO) terms and a community system for general GO annotation of proteins
MetaBase	A community-curated database of commonly used biological databases
PDBWiki	A community annotated knowledge base of biological molecular structures

Proteopedia	A scientific wiki bridging the rift between three-dimensional structure and function of biomacromolecules
Rfam	A community-derived annotation of RNA families
RiceWiki	A wiki-based, publicly editable and open-content platform for community annotation of rice genes
SEQanswers Wiki	A wiki database of tools for high-throughput sequencing analysis
SNPedia	A wiki supporting personal genome annotation, interpretation and analysis
***Subti*Wiki**	A comprehensive community resource for the model organism *Bacillus subtilis*
Transdab Wiki	A wiki database of transglutaminase substrate proteins
WikiCell	A unified resource platform for human transcriptomics research
WikiGenes	A collaborative knowledge resource for the life sciences
WikiPathways	An open, public platform dedicated to the curation of biological pathways
WikiProteins	A wiki-based system for community annotation of proteins

Table 1. A list of biological wikis. Data source: http://bigd.big.ac.cn/sciencewikis/index.php/Biological_Wikis.

A biological wiki has many advantages in comparison to traditional database applications.

1. A biological wiki encourages the fellow researchers in the community to contribute to the database. Hence, it is not dependent on a single lab for maintenance. The wiki can be still updated even if the person or lab initiated the project is no longer participating.
2. Most wiki software provides the feature of version control, which means each edit of the content of a page is well documented. This allows the researchers to track the source of the information.
3. A wiki is easy to set up. There have been quite a few wiki software packages available. Only simple installation without any programming is required. This allows researchers without knowledge of programming to establish their own platform for information sharing.

4. The content of a wiki page does not require a fixed scheme. The users can freely edit the page structure and style. This prevent the possible data loss due to the limit of data model.

However, most implementations of wiki software are designed for narrative information rather than complicated biological data. The limitations of the software result several drawbacks:

1. Most wiki software packages are designed primarily for sharing text. They do not handle tabular or hierarchical data well. This makes it not very suited to store relationships between objects.
2. The database layout of most wiki software packages does not perform data consistency control. The pages in the wiki and associated with hyperlinks other than database references. For example, one can create a hyperlink to a non-existing page. This is usually not allowed in traditional databases

To resolve those issues, Brohée *et al.* has developed a plugin in 2009 for the popular wiki software MediaWiki [6], which supports batch data operations in wiki. However, this plugin has stopped updating and the download link is no longer accessible.

2.3 Model organism databases

Model organism databases (MODs) are databases which focus on functional annotations of well-studied model organisms [7]. These model organisms include *Bacillus subtilis, Escherichia coli, Saccharomyces cerevisiae, Drosophila melanogaster, Caenorhabditis elegans, Mus musculus,* and *Arabidopsis thaliana.* Most MODs are secondary databases hosting manually or automatically curated genomic and functional information. They sometimes also provide extra features such as data visualization and analysis in addition to data hosting.

In comparison to large cross-species sequence or structure databases like GenBank [8] or PDB [1], MODs focus on a specific domain of knowledge. This specification saves time and effort the users need to spend on finding and filtering data of their need. The MODs have proven to be very helpful in different stages of research [7].

Model organism	**Model organism database**
Saccharomyces cerevisiae	Saccharomyces Genome database
Schizosaccharomyces pombe	PomBase
Xenopus laevis	XenBase
Drosophila melanogaster	FlyBase
Mus musculus	Mouse Genome Informatics
Caenorhabditis elegans	WormBase
Rattus norvegicus	Rat Genome Database
Dictyostelium discoideum	dictyBase
Arabidopsis thaliana	The Arabidopsis Information Resource (TAIR)
Danio rerio	Zebrafish Information Network
Candida albicans	Candida Genome database
Escherichia coli	EcoCyc

Table 2. A list of model organism databases. Data source: https://en.wikipedia.org/wiki/Model_organism_databases

2.4 The model organism *Bacillus subtilis*

Bacillus subtilis is a rod-shaped soil bacterium. It is not pathogenic to humans and animals. It was among the first microorganisms people cultivated and studied. The first paper describing this bacterium dates to early 1900s. In the last 100 years, there have been 33250 research papers about *B. subtilis* available in PubMed and this number keeps growing (Figure 2).

Bacillus subtilis is a model organism to understand the biological processes such as spore formation, biofilm formation etc. It is also used as base organism in minimal genome projects [9]. Because of its excellent fermentation properties, it is of great value in industry as well. It is used as cell factories to produce enzymes, vitamins and other products [10].

The compete genome sequence of *B. subtilis subsp. subtilis 168* was published in 1997 [11]. *Bacillus subtilis* has over 6000 genes and RNA features. About 4200 of them are protein encoding genes. In *Subti*Wiki, there are currently 253 genes identified as essential.

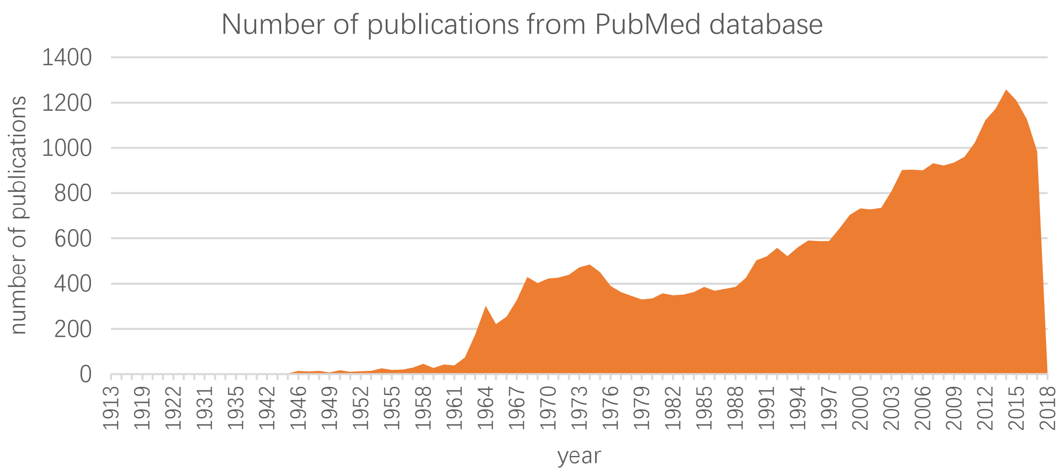


Figure 2. The number of publications about Bacillus subtilis every year from PubMed database [12].

2.5 Databases for *Bacillus subtilis*

As a well-studied model bacterium, *Bacillus subtilis* has several MODs dedicated to it. The first MOD "*Bacillus subtilis* ORF database (BSORF)" was established in 1995 in Japan [13]. This database project is initiated to release the sequencing results of *Bacillus subtilis* genome. This database was actively updated until 2006.

In the same year, another project "SubtiList" is initiated in Europe. SubtiList [14] is a relational database dedicated to functional genome annotation of *Bacillus subtilis*. This database stopped updating in 2004.

In 1999, the first version of DBTBS was release in Japan [15]. This database collects information about transcription factors and promoters. Later the focus was shifted to regulatory networks in *Bacillus subtilis.* The database has stopped updating in 2008.

BioCyc is a collection of pathway/Genome databases of different organisms. BsubCyc is one of the database collections dedicated to *Bacillus subtilis*. BsubCyc provides information about metabolic pathways, regulatory networks, and functional genome annotation. However, since 2017, BsubCyc has changed its access policy and its data are no longer open to public. The users are required to pay for a subscription to access the data.

2.6 *Subti*Wiki and *Subti*-Apps

As BSORF and SubtiList stopped updating around 2005, the *Bacillus* research community needed an up-to-date model organism database. Hence, the *Subti*Wiki project was initiated. It was first online in 2009 using the MediaWiki engine with the motivation to enable collective curation of annotations in the research community [16].

*Subti*Wiki was designed to be a collection of functional genome annotations. The information of genes is stored in separate "pages" and the names of genes are used as title (identifier). All pages are generated with a template for a uniform page structure. A table on the top provides brief information about the gene such as name, function, production, neighbors etc. More detailed information about the gene, the RNA, and the protein is displayed in sections [16].

Besides genes, a category system is developed and managed within the wiki. This category system classifies genes according to their functions. This system has a tree structure with 6 major categories and over 5 layers, offering a very detailed grouping according to functionality of genes [17].

As more and more information about *B. subtilis* became available, the focus of research shifted from single genes to association among the genes. Hence, two *Subti*-Apps are included in a later update, namely *Subti*Pathways and *Subt*Interact. *Subti*Pathways depicts the metabolic pathways as maps using system biology markup language while *Subt*Interact focuses on protein-protein interactions [17].

In 2012, a large-scale transcriptomic study was conducted and data were gathered under more than 100 experimental conditions [18]. Based on the results of this study and other proteomic data [19,20], *Subti*Express is introduced [21].

Those *Subti*-Apps supplemented *Subti*Wiki and makes *Subti*Wiki one of most complete knowledge base for a single organism.

2.7 Motivation of this project

*Subti*Wiki is a successful database and has served the *Bacillus* community in many ways. However, there are a few issues concerning the implementation.

*Subti*Wiki is not an integrated system. Annotations of genes are kept and managed in MediaWiki and each of *Subti*-Apps has its own separate database which is not updated in a synchronized manner. The resulting problem is that the name of genes in *Subti*-Apps are not synchronized and extra manual updates are required.

MediaWiki has its limitations. The content of pages in MediaWiki is mostly in text format with images or videos inserted, which brings two outcomes. First, the contents of pages are not structured enough for exports or batch operations. Second, the page is rather static. Dynamic contents and interactive parts are difficult to be integrated.

It is also difficult to store associations among genes under the framework of MediaWiki. Adding one protein-protein interaction always requires two edits on pages of each interaction partner. This introduced data duplication and data duplication introduced data inconsistency.

Those issues motivated us to develop a relational database and a content management system which are more adapted to our need. This system should:

1. be suited for biological information
2. be flexible and extendable
3. have good performance
4. improve the experience of data management
5. have a user system
6. have version control
7. introduce more interactivity

With this new system, *Subti*Wiki should become friendlier to both users and developers. For users, the interfaces should be simple and intuitive. For developers, it should be easy to extend the data scheme and add new functionalities.

3 Methods and tools

3.1 Web related

3.1.1 LAMP software bundle

LAMP is a software bundle for building dynamic web pages or web applications [22]. It is an acronym of the names of four software packages, namely Linux, Apache, MySQL and PHP. The *Subti*Wiki server is installed with LAMP bundle.

Linux is a Unix-like operating system. It commonly refers to a family of operating system distributions packed with a Linux kernel. It is a popular choice for web servers. Ubuntu and Debian, two Linux distributions, take up over 50% of the market [23].

Apache is a free and open-source web server software [24]. It is highly scalable, handling large numbers of requests at ease. It provides varies of feature as modules [25], which extend the core functionality of the software. Among all the modules, *Subti*Wiki installed two, namely `php5_module` for PHP support and `mod_rewrite` for URL rewriting.

MySQL [26] is the relational database management system in the bundle. It covers a broad subset of ANSI SQL 99 standard [27]. It provides multiple store engines such as InnoDB, MyISAM, Memory, CSV, etc. It also offers features like stored procedures, triggers and sub-selects. More details will be described in chapter 3.2.

PHP [28] is for hypertext preprocessor. It is scripting language primarily design for web developing. It has a syntax like C and the variable naming style like Perl. Its code can be embedded in HTML, which proved to be handy for generating dynamic web pages. More details will be introduced in the chapter 3.1.3.

As the image below illustrates, requests initiated by clients travel through the internet and arrived at the server. They are handled by the web server software, which is Apache in LAMP bundle. Server-side scripts are invoked to retrieve data from a persistent storage, database or file system. The raw data from persistent storage is processed and a response is generated upon them. The response is sent out by web server software and goes back the client.

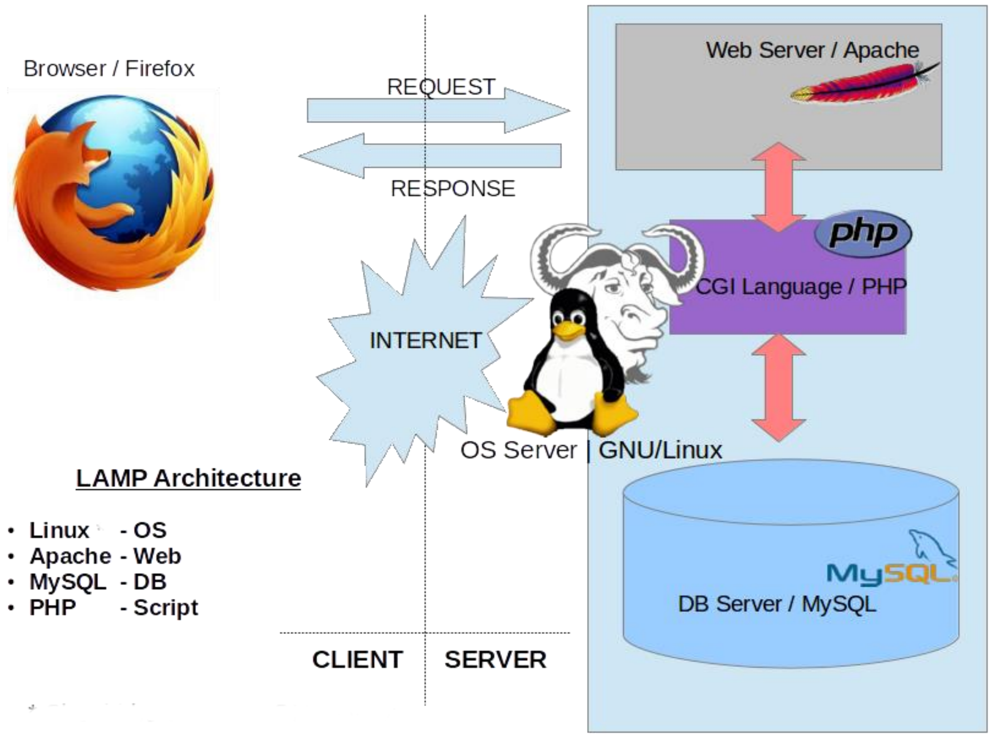


Figure 3. A over view of components in LAMP software bundle. Work flows marked in arrows. This image is created by K7.india at English Wikipedia, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=31270395.

3.1.2 URL rewrite and mod_rewrite

URL rewriting is the process to modify uniform resource locators (URLs). It is often performed to shorten the long URLs or make them more user/search-engine friendly. It is also a way of implementing routing in web application, which provides a layer of abstraction between the script generating a web page and the URL with which this page can be accessed.

The URL rewriting function is provided in Apache as a module [29]. The Apache module `mod_rewrite` uses a rule (regular expression) based engine. Those rules can be applied to different contexts depending on where they are placed [29]. Written in the global configuration file of Apache `httpd.conf`, the rules affect all virtual hosts on the server. When placed in the `<VirtualHost>` block, those rules will only apply to specific virtual host. A `.htaccess` file with such rules can be placed under a directory to make those rules effective only for this directory [29].

The URL rewriting module of Apache is extremely powerful and complicated. A typical `.htaccess` starts with the line “`RewriteEngine on`” to enable the rewrite engine. The `RewriteCond` keyword can be used to define a condition under which URL rewriting will take place. The `RewriteRule` keyword is followed by the actual rules in regular expression. Finally, the `RewriteOption` keyword defines the scope of the rewrite engine [29].

When an URL enters the rewrite engine and the rewrite engine is enabled, it is compared with the pattern provided in `RewriteCond` syntax. If it is a match, the rewrite rules will be executed, and the rewritten URL continues to be compared with the pattern in the next `RewriteCond` syntax. Inappropriately written rules could result an endless loop.

This code block is the actual `.htaccess` file used in the new implementation of *Subti*Wiki.

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} -f
RewriteCond %{REQUEST_FILENAME} /(var|app|res)/
RewriteRule /(var|app|res)/ html/404.php [PT]

RewriteCond %{REQUEST_FILENAME} -f
RewriteCond %{REQUEST_FILENAME} src
RewriteCond %{REQUEST_FILENAME} !src/init.php
RewriteRule ^(.+)$ html/404.php [PT]

RewriteCond %{REQUEST_FILENAME} -d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.+)$ html/404.php [PT]

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f [OR]
RewriteCond %{REQUEST_FILENAME} .php$
RewriteRule ^(.+)$ src/init.php [END,L]
```

Code block 1. The sample of a .htaccess file

The first snippet filters requests on existing files. If those files are under the directories `var/`, `app/`, or `res/`, request will be redirect to an error message page. The second code snippet blocks access to the files under `src/` except for `init.php`, which serves the sole entry point of the requests. The third code snippet rejects all access to directories. The last snippet redirects all requests except the allowed files to `src/init.php`. The flag at the end of line `[END]` indicates the rewriting stops here.

3.1.3 Server-side scripting and PHP

Server-side scripting is a technique used to create dynamic web pages according to the requests initiated by clients. It differs from the client-side scripting, which is embedded and evaluated at the client-side, mostly web browser. In earlier times, the server-side scripting was mostly done by a combination of C programs, Perl scripts and shell scripts using the Common Gateway Interface (CGI). Nowadays there is a big variety of server-side scripting language for this purpose, such as ASP, Java, server-side JavaScript, Lua, PHP, Python, R etc.

PHP is the server-side scripting language in the LAMP software bundle. It is primarily designed for web developing. The syntax of PHP is like C. All the variables begin with a dollar sign, which resembles Perl. PHP is not type strict. It has four scalar type, i.e. Boolean, integer, float, and string. For compound types it offers arrays, objects callables (functions) and iterables. A special type resource is included to present the references to external resource, mostly C pointers [30].

PHP provides two types of arrays, the simple array and the associative array. For arrays, the keys must be either integers or strings. The values in an array does not require to be the same the type[31]. The arrays can be converted into objects by simple casting [32]. However, the most common way to create an object is to define a class and instantiate the class [33]. This gives PHP the object orienting feature.

In PHP, the functions are of the type callables /callbacks [34], which means functions can be used as parameters or return values of other functions. This gives it features of function based programming.

3.1.4 HTML and document object model

Hypertext markup language (HTML) is a standard markup language used to build web pages[35]. HTML is one of the technologies along with cascade style sheet (CSS) and JavaScript used to generate contents on the internet. HTML describes the underlying structure of the web pages and provides different elements for different purposes.

HTML elements are the building blocks of web pages. Tags using angle brackets are used to determine them. Some of the HTML elements introduce content directly into the page, such as `<img />` or `<input />`. Some of the HTML elements consist of two tags, an opening tag and a closing one, like `<p></p>`. In this case, the content to be rendered is placed between the opening and closing tags.

Web browsers do not display HTML tags. They just display the rendered page using tags as guidelines. For most modern web browsers, document object model is a built-in feature. This model interprets an HTML or XHTML or XML document as a tree structure with HTML elements as nodes in the tree. Each HTML element is presented as an object and can be accessed and manipulated programmatically. The scripting language used to alter DOM elements is JavaScript.

3.1.5 JavaScript, JSON and AJAX

JavaScript is a high-level interpreted programming language [36]. Along with HTML and CSS, JavaScript is an important element for web developing. JavaScript code can be executed in the web browser and supports manipulation of the document object model (DOM)[35], making the web page more dynamic and interactive.

JavaScript is not type strict. It has three primary data types, i.e. number, string or Boolean. For composite data types, JavaScript offers objects and arrays. Objects in JavaScript are associative arrays. The keys are of string type and the values unlimited. To access the attributes in an object, a dot notation (`person.name`) or bracket notation (`person[“name”]`) can be used [36].

JavaScript is almost completely object-oriented. However, unlike object-oriented programming languages based on class, JavaScript uses prototypes. In JavaScript, functions double as object constructors. The new keyword is used to create an object from a prototype.

JavaScript also natively support many function-based features. In the modern implementation of JavaScript, functions are constructed as objects. A function can have properties and even methods, like bind() or call(). Functions can be taken as parameter or used as return value of other functions. Nested functions, which means a function defined inside another function is also allowed. Those features greatly enriched the functionality of JavaScript [36].

JavaScript Object Notation (JSON) is derived from JavaScript [37]. It is a light-weight data exchange format. It is easy to read and write for both humans and machines. JSON is a text format which is independent from JavaScript.

JSON is built on objects and arrays. Objects in JSON are an unordered collection of key value pairs. The keys are of string type while the value can be null, strings, numbers, Boolean values, objects or array. An array in JSON is an ordered list of values. In some implementation of JSON parser, values in an array do not require to be of the same type [37].

A JSON object is wrapped within a pair of curly brackets. Each key value pair in this object is separated by a comma. A colon is placed between the key and the value in each key value pair. A JSON array is wrapped within a pair of square brackets. Each value in the array is separated by a comma [37]. The example below shows the employee information in JSON format.

```json
{
    "name":"John Doe",
    "gender":"male",
    "employee_id": 314,
    "association": "Univeral heritage"
}
```

Code block 2. The sample of JSON text

To update the partial data without refreshing the whole page, asynchronous JavaScript and XML (Ajax) can be applied to reduce the data traffic. Ajax is a set of technologies which allow request sending and response receiving run in the background without interfering the rendered web page. This enabled much more activity of the web site and reduced the data traffic. With ajax, web applications can be built with a different model, which is illustrated in Figure 4.

3.2 Relational databases

Relational databases are based on the relational model of data. This model was proposed by E. F. Codd in 1970 [2]. The purpose is to provide a declarative way to specify data and queries. In this model, data are presented as tuples and grouped in relations. Here the word “relation” has a counter-intuitive mathematical meaning. Thus, it is commonly conceived as “table”.

In relational databases, data are presented in tables. A table is a collection of objects of the same type. It has columns and rows. The column headers are the name of attributes of objects while each row is an object [38]. Relational databases follow certain rules to ensure data accessibility and integrity via various keys and constraints [38]

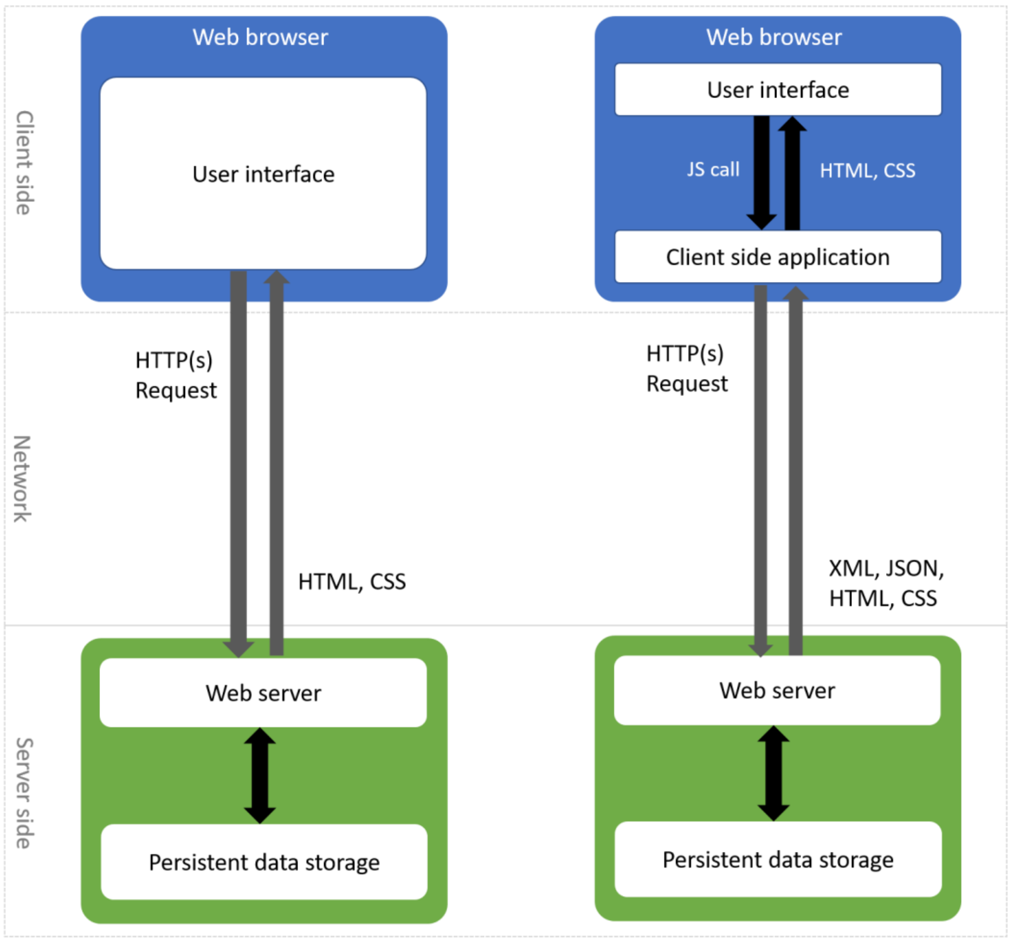


Figure 4. The conventional model for a Web Application versus an application using Ajax. A client-side application is added serve as a middle layer between the user interface and server side. This client-side application initiates HTTP(s) requests and handles the update of the user interface.

3.2.1 Primary key

A primary key is a column or a combination of columns which is used to identify a row or present a relation to another table. Its main purpose is identification. Thus, one table can only have one primary key and values of this primary key cannot be `NULL`.

3.2.2 Foreign key

A foreign key is a relationship between rows in different tables. The foreign key refers from a column in the parent table to a column in the child table, building a dependency between two tables. This not only enables fast queries on related data, but also prevents data being inserted, updated, or deleted when the dependency will break. One row in the child table cannot be inserted if the referred row in parent table does not exist. The update or deletion on a row in the table will be either prevented or cascaded to the row in parent table depending on the definition of this foreign key.

3.2.3 Stored procedures

A stored procedure is a database object which implements a database routine. It can be considered as functions in the context of SQL. Stored procedures can take parameters and deliver results. It can be used to encapsulate complicated business logic, perform data validation etc.

3.2.4 Triggers

A trigger is a database object which implements a routine and is associated with a table. The implemented logic is executed when an event occurred on the table, such as insert, update, and delete. Triggers can be used to validate data or maintain other tables. The trigger and the associated statement is grouped up as a transaction, which means if the trigger failed, the associated statement will not take effect.

3.2.5 Structured query language

Structure query language (SQL) is the first commercial implementation of the relational model [27]. It doesn’t not completely conform to the relation model, but this didn’t affect its success as the most widely used database language.

SQL uses a collection of imperative verbs for the process of modifying scheme or data. This makes it very intuitive and read like a nature language. It consists of a data definition language, a data manipulation language, and a data control language [27]. The data definition language defines/ alters the scheme of data, for example, the scheme of table, its keys, its indexes etc. The data manipulation language operates on the data themselves. The select/update/insert/delete statements are parts of it. The data control language defines or alters the permissions on certain data. It defines the user privileges on databases, tables, and columns.

3.2.5.1 `Insert` statement

An `insert` statement is used to add new rows to a table. It should specify the name of intended table, the header of columns and the values to be inserted. The strings in the syntax should be properly quoted. The insert statement is not successful when the data type of any given value does not confront to the defined table scheme. For example, the following query inserts a new row in the gene table.

```
INSERT INTO gene (id, gene_name)
VALUES      (12, "dnaa");
```

Code block 3. A sample of insert statement

3.2.5.2 `Select` statement

A `select` statement or a query is used to retrieve data from the database. It should specify one or a few column headers, one or more tables to select data from and as well as a `where` clause to specify the rows. For example, a query to find short genes from the “gene” table is as follows:

```
SELECT *
FROM   gene
WHERE  geneLength < 500.00
ORDER  BY locus;
```

Code block 4. A sample of select statement

This query finds all rows in “gene” table which the value of `geneLength` column is smaller than 500. The results are sorted in ascending order by the `locus` (locus tag) column. The wild card symbol “*” directly after `select` indicates all columns are included in the result data set.

3.2.5.3 `Update` statement

An `update` statement updates the existing data in an existing table. It should specify the name of the table to be updated, the columns to be updated, the new values, and a where clause to specify the rows.

```
UPDATE gene
SET    gene_name = "dnaB"
WHERE  id = 12;
```

Code block 5. A sample of update statement

The update statement above updates the row with the id “12”. The value of gene_name column of this row is updated to “dnaB”.

3.2.5.4 Delete statement

A delete statement deletes a row or rows from a table. It should specify the name of the table and a where clause to specify the rows to be delete. The following statement delete the row with id 12 from the gene table.

```
DELETE FROM gene
WHERE  gene_name = "dnaA";
```

Code block 6. A sample of delete statement

3.2.5.5 Where clause

The where clause in select/update/delete statement specifies the rows to operate on with predicates. A few comparison operators can be used in the predicates such as “=”, “>”, “<”, “is”, “like” etc. The comparison operator “like” is used to compare strings to given string or pattern while Comparison operator “is” is often used to determine if a value is a NULL. Predicates can be combined with logic operators.

```
...
WHERE gene_name LIKE "dna%"
AND geneLength > 1000;
```

Code block 7. A sample of where clause

The where clause above will specify the rows whose values in gene_name column start with “dna” and the values in geneLength column greater than 1000. The wild card symbol “%” in the provided pattern presents matches to one or more unspecified characters.

3.2.5.6 Join syntax

A join syntax is used when information from more than one table is retrieved. There must be one column appearing in both tables which can be used as criterium for joining. For example, we are interested in the names of interaction partners of protein DnaA. We now have a "protein" table with an "id" column and a "name" column. We also have an "interaction" table with columns "protein1" and "protein2" which store only ids from "protein" table. The select statement to fulfill our purpose would be the statement below.

```
SELECT protein_table_1.name, protein_table_2.name
FROM   interaction
       JOIN protein AS protein_table_1
         ON protein_table_1.id = interaction.protein1
       JOIN protein AS protein_table_2
         ON protein_table_2.id = interaction.protein2
WHERE  protein_table_1.name = "dnaA"
        OR protein_table_2.name = "dnaA";
```

Code block 8. A sample of select statement with join syntax

In this syntax, the "protein" table is joined with "interaction" table twice as two different instances and two aliases are given to distinguish them. As interaction is mutual, the disjunction of two predicates are included in the where clause.

3.2.6 Entity-relationship model

The Entity-Relationship model was first proposed by Chen in the year 1976[39]. It is based on set theory and relation theory and can be considered as a generalization of the network model, relational model, and entity-set model, which were the three major data models.

The Entity-relationship model presents data in an abstract level. It is often applied in the conceptual designing of a relational database. It concludes the domains of knowledge which should be part of the database and presents this knowledge with Entities and Relationships.

An entity, is by Chen's definition, *a thing that can be distinctly identified*. An entity could be a person, a car, a gene or a protein. A relationship, is the *associations among entities*. For example, "marriage" can be described as the relationship between two "person" entities.

The entities and relationships are objects of higher abstraction in the design process. To gradually implement those concepts into a physical database, entities and relationships need to be described in an information structure. The information concerning the entities are gathered and expressed as an attribute-value set.

The Entity-relationship model can be illustrated with a diagram, in which boxes present entities and diamonds relationships. Attributes are drawn as circles connected to entities or relationships. The type of the relationships, such as one-to-one or one-to-many or many-to-many, should be also marked on the line connecting entities and relationships.

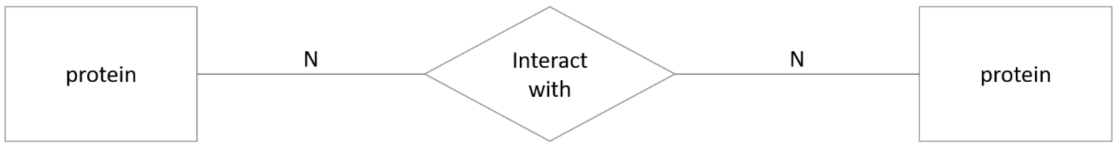


Figure 5. A demonstration of ER diagram, presenting the relationship "protein-protein interaction" between "protein" entities

Among those attributes, one or a combination of multiple attributes can be used to identify the entity as *entity primary key*. With the *entity primary key* defined, the relationship between two entities can be presented as the relationships between the primary keys.

3.3 Graph drawing

Graphs, as abstract mathematical objects, are commonly used to present the relationships among things. The formal definition of a graph is an ordered pair of the set of nodes (vertices) and the set of edges, which is the two-element subset of the set of nodes [40].

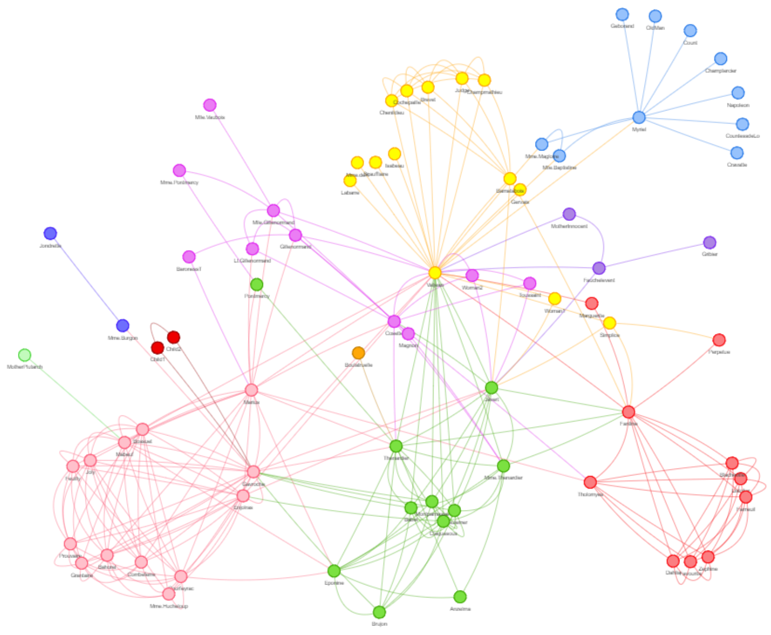


Figure 6. The node-link diagram of the graph depicting the relationship between characters of the novel "Les miserables". Image generated as Visjs show case [41].

Graphs are applicable to present a lot of things in the everyday life and scientific research: the bus and subway systems with all their lines and stops, the association among people at work or in social media, the interaction of binding elements inside the cell like DNA segments, RNAs, proteins and smaller molecules.

Graph drawing is a set of mathematical and computer science methods to generate visualizations of graphs [40]. This visualization often depicts the nodes and edges in the graph in node-link diagram (Figure 6) where nodes are presented with dots, circles, boxes, etc. and edges with lines [40]. Arrow heads are sometimes included for directed graphs to indicate the direction of edges. In addition to node-link diagrams, other presentations of graph are available such as circle packings [42], intersection representations [43]. In those methods, nodes are represented in areas and edges are presented as adjacency or intersection of those regions.

To evaluate the results of visualization, different measurements are defined. The crossing number of edges is the criterion universal to graph layout methods using node-link diagrams. Symmetry is also another aspect to consider as patterns are always easy to human eyes.

3.3.1 Circular layout

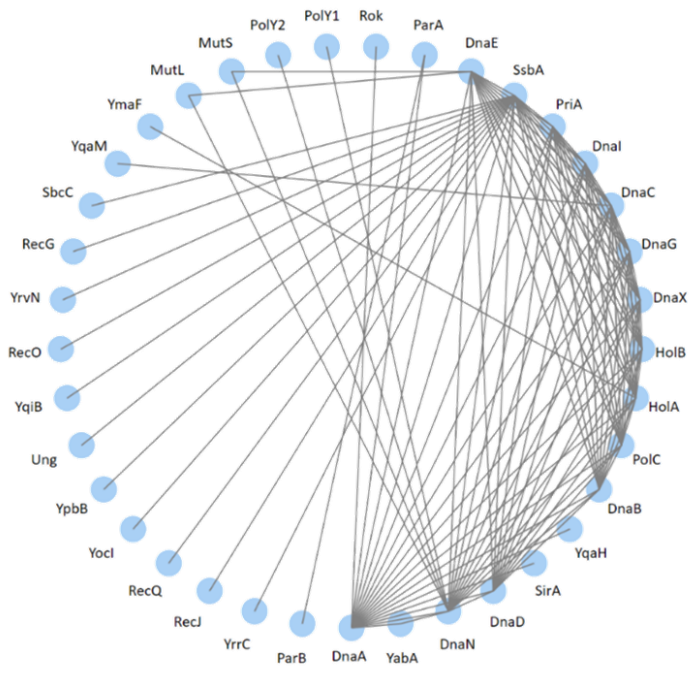


Figure 7. The protein-protein interaction diagram of DnaA in circular layout[44].

Circular layout is quite intuitive. In this layout, all nodes are placed on a circle. Edges are drawn as straight or curved lines connecting the nodes inside or outside the circle. The position of the nodes need to be adjusted to minimize edge crossing for a clear visualization.

3.3.2 Orthogonal layout

Orthogonal layout allows the edges of the graph to run horizontally or vertically as single lines or polylines. This layout is variously used in presenting flow charts, database diagrams, etc.

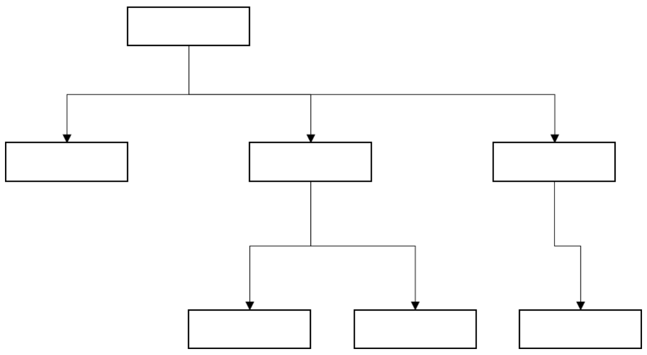


Figure 8. A random graph in orthogonal layout

3.3.3 Force-directed layout methods

Force-directed layout methods generate pleasing and aesthetic results. The visualizations tend to have uniform edge lengths, evenly distributed nodes, clear structure, and beautiful symmetry.

The force-directed layout methods, as the name suggests, calculate the positions of nodes by assigning forces to them: global repel force between all nodes but attractive forces between the adjacent ones. A configuration with the lowest energy and most force balance is considered as the best solution.

3.3.3.1 The algorithm of Eades

The algorithm of Eades was proposed in 1984 [45]. In this method, edges are modelled as springs with logarithmic strength, which means the force on the spring is not linear according to Hook's law, but rather logarithmic to its deformation. A global repel force between non-adjacent nodes exists under an inverse square law. For the layout, nodes are initially randomly. Nodes are moved according to the forces on them in each iteration.

$$f_{attraction} = c_1 * \log\left(\frac{d}{c_2}\right)$$

$$f_{repel} = \frac{c_3}{\sqrt{d}}$$

c_1, c_2, c_3 are both constant where c_2 is the resting length of the spring. d is the distance between nodes.

3.3.3.2 The algorithm of Fruchterman and Reingold

The later algorithm of Fruchterman and Reingold [46] improved the methods by including additional measurements. In this algorithm, it is an important how evenly nodes were distributed. Hence, they defined the repelling and attractive forces differently.

$$f_{attraction} = \frac{d^2}{k}$$

$$f_{repel} = -\frac{k^2}{d}$$

Here d is the distance between the nodes. Note that k represents the optimal distance between nodes and is not a constant. It is associated with the "density" of the nodes and defined as following.

$$k = C\sqrt{\frac{area}{number\ of\ nodes}}$$

Similar to the algorithm of Eades, the method computes the attractive and repel forces and moves nodes step by step trying to find a force balance state. Different from the algorithm of Eades, the concept "temperature" is introduced to scale down the movement of nodes in each iteration. The temperature should be set with an initial value proportional to the drawing area (frame) and reduced in iterations. This "cools" the system down as in later iterations, the movement of nodes are scaled down more.

3.3.3.3 The algorithm of Kamada and Kawai

The algorithm of Kamada and Kawai [47] takes the graph distances between nodes into consideration, which is defined as the length of shortest pathway from one node to another. The goal then becomes to make the geometrical distances match the graph distance. Therefore, the forces between the nodes are simplified. If the geometrical distance between two nodes is relatively smaller than the graph distance, the nodes repel each other and otherwise the nodes attract each other.

Therefore, graph theoretical terms begin to map with geometrical terms. The width or the length of the drawing area corresponds to the diameter of the graph. The desired geometrical distance between two nodes should be proportional to the graph distance, and the coefficient should be also linear to the ratio between the previous two values mentioned. This algorithm is computationally expensive as the pair-wise shortest paths need to be found first.

3.3.3.4 Multi-level approaches

For large graphs with over thousands of nodes, the drawing process can be extremely expensive. It is natural to think of an approach which generates a rough layer first and then gradually finalize the details of the graph. The concept of multi-level approaches was first raised by Hadany and Harel, who introduces the strategy below [48].

1. *Perform fine-scale relocations of vertices that yield a locally organized configuration*
2. *Perform coarse-scale relocations (through local relocations in the coarse representations, correcting global disorders not found in stage 1)*
3. *Perform fine-scale relocations that correct local disorders introduced by stage 2*

Since the concept was presented, there have been quite a few different multi-level graph drawing algorithms. The algorithm of Harel and Koren [48] uses the k-centers problem for an abstraction. The algorithm of Walshaw [49] extended the algorithm of Fruchterman and Reingold. The algorithm of Quigley and Eades [50] assigned gravity as repel force (with negative gravity constant) and relies on the Banes-Hut simulation [51] for n-body problems in physics to reduce the complexity of repel force computation.

3.4 Mobile development

3.4.1 SQLite

SQLite [52] is a light-weight relational database management system. It implements most of the SQL standards. Unlike client-server database management systems, SQLite usually comes as a library linked to the target program and becomes an active part of it [52].

SQLite is a compact library which can run with minimal memory. This makes it very suitable for platforms with memory constrains, such as embedded systems and smart phone applications.

3.4.2 Development of an Android application

Android is an open-source and free mobile operation system based on Linux kern [53]. It was first developed by Android Inc. which was later bought by Google. It has a wide support for different hardware and takes up 85% market share in the first quarter of 2017 [54].

Android applications are written with Java. Code and other resource files are compiled and packed into an Android package by Android SDK tools. Those packages are later used to install application on devices.

Each application runs its own secure sandbox, which means the code and files of each applications are isolated from each other. Each process has its own virtual machine and a unique user is created for each application. This ensures that the private files cannot be accessed by other apps.

The applications usually consist of a few components such as Activities, Fragments, Services, Broadcast receivers and content providers. Each of the components serves for a distinct purpose.

3.4.2.1 Activity

An Activity is an application component which provides an interactive interface [55]. Each Activity acquires a window on which the interface is drawn on. This window usually takes up the whole device screen or sometimes appears as a flowing window, like an alert message etc.

An Android application usually consists of a few loosely tied activities. One of those activities are assigned as “main” in the Android manifest file. This activity will first appear when the user launches the application. When a new activity starts, the old activity will pause, and its states will be kept in the return stack in case the user returns by pressing the back button.

Therefore, an activity has a life cycle from being created to being destroyed, depending on the operation of users. This life cycle is implemented with a few callback methods in the `Activity` class. To create an activity, a new subclass of the `Activity` class should be implemented. The life-cycle callbacks should be overridden with proper logic to respond to the changes of state. More details about the activity and its life cycle is available in the developer’s guild [53].

3.4.2.2 Fragment

The Fragment is introduced to Android 3.0+ [56]. It can be considered as an encapsulation of behavior or user interface inside the activity. It is primarily designed to enable more flexible layout for tablets which have larger screens. With Fragments, the application can adapt to different kind of screens in a modularized way and no extra effort is required to adjust the layout (Figure 9).

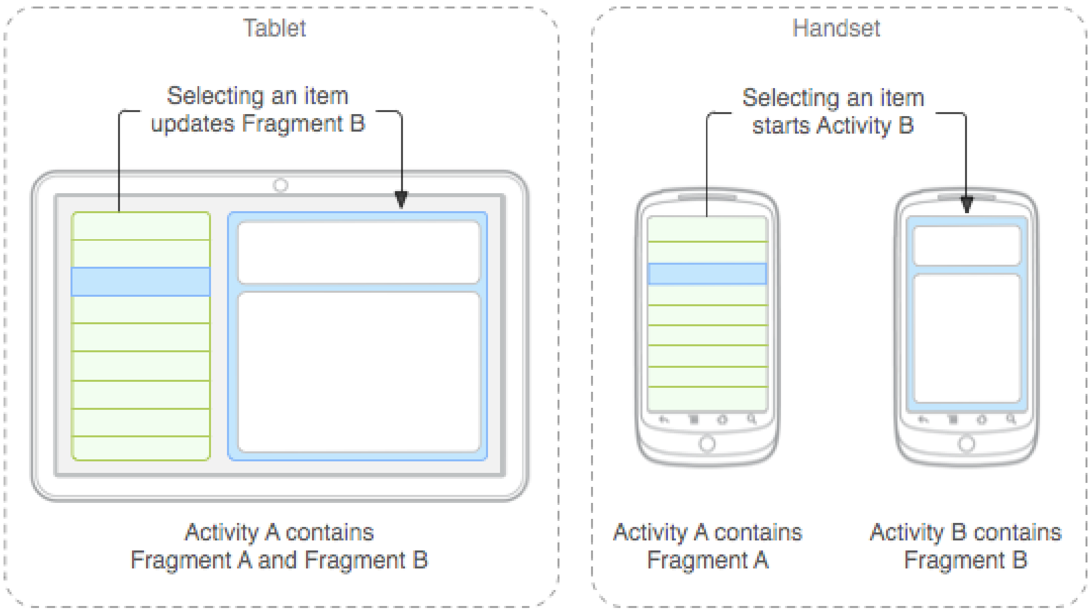

Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

Figure 9. A demonstration of using fragments for different screens.
Data source: https://developer.android.com/guide/components/fragments.html

To create a fragment, a subclass of Fragment class should be implemented. Like activity, the fragment also has a life cycle. The life-cycle callback methods needed to be overriden to handle the fragment state change. A fragment can be added to an activity either in runtime or throught the layout definition file.

A `FragmentManager` class is provided to manage the fragments within an activity. The fragments can be added, replaced or removed. The default action on back buttons also affects fragment transactions althought this action can be overriden.

3.4.2.3 Multi-threading in Android

Android uses the single threat model [57]. By default, all component of the same application run in the same process. When an application is launched, the system creates a thread for it, called “main”. This thread is sometimes called “User interface (UI) thread” as it is responsible for UI drawing and other events on user interface elements. When the UI thread is blocked, the user interface elements do not respond and lose interactivity [57].

Therefore, for time consuming tasks such as file operation or internet access, it is recommended to perform them on a separated thread (work thread). However, the UI elements cannot be updated from other threads. Thus, Android offers several different ways to access UI thread from other thread, such as

`Activity.runOnUiThread(Runnable)`,

`View.post(Runnable)`,

or `View.postDelayed(Runnable, long)`.

Another solution would be the `AsyncTask` class. It provides a few callback methods such as `onPreExecute`, `doInBackground`, `onProgressUpdate`, and `onPostExecute`. Of all four methods provided, `doInBackground` is executed on a separate thread, while the other methods are executed on the UI thread.

3.4.3 Development of an iOS application

iOS is the operating system running on mobile Apple devices such as iPod touch, iPhone and iPad. The operating system is developed and maintained by Apple Inc. The operation system manages the hardware of the devices, and provides the framework to develop applications. For application development, objective-C or Swift can be used.

3.4.3.1 The Swift programming language

Swift is a new programming language introduced for iOS, macOS, watchOS and tvOS. It is not a scripting language but has some extremely interesting features which makes it very flexible and handy.

Swift is a type-strict language. The type of a constant or variable can be implied from initial value or explicitly defined. Swift provides scalar types such Int, Double, Float, Bool and String. For collection types, swift offers Array, Set, and Dictionary. In swift, constants are declared by `let` keyword and variables `var` keyword.

Optionals are another new feature that Swift offers. An optional variable can be assigned with either nil or a value of designated type. Here nil is not like nil in C or NULL in Java. It presents the absence of a value and can only be assigned to optional variables.

An optional variable can be "unwrapped", which means this variable will be cast from optional type to a non-optional type. Before unwrapping an optional variable, it is always necessary to check if this variable does have a value, otherwise runtime error can be raised. This can be done in an `if`-statement by comparing the variable with nil. Optional binding can also be used to find out an optional variable has a value or not and directly assign the value to another constant to be used in the if block.

```
if let actualNumber = Int(possibleNumber) {
    print("has a number")
} else {
    print("is nil")
}
```

Code block 9. Sample code of optional binding in Swift programming languages

Besides optional binding, Swift also provides optional chaining to handle a series of queries on attributes or subscripts which can be nil. By adding a "?" behind the variable or attribute of optional type, the chained expression will fail if the variable or attribute is nil. This saves a large amount of if statement in comparison to the code in Java expressing the same logic.

Beside optional variables, Swift has a lot of other interesting and exciting features. For example, subscript function can be custom defined, which makes implementing our own collection type easier. A defined class can be extended with more methods, which makes Swift as flexible and enjoyable as scripting languages.

In general, Swift is a new language introduced in recent years. It combines the nice features and designs from multiple languages. It is a very pleasant experience to write Swift.

3.4.3.2 Model-View-Controller pattern

The model-view-controller pattern is a software design pattern which divides the software into three components, namely data model, view, and view controllers. The data model defines the information or knowledge the software will handle. Views are the visual presentations of information on device screens. This could be in the format of text, images, diagrams or videos etc. The controller is responsible for process requests and updates views.

The framework for macOS and iOS development (Cocoa and Cocoa Touch) is built using this pattern. They also provide various helpful components to assist the application development. In practice, the user interface can be designed in the interface builder editor integrated in XCode. In the interface builder editor, the developer can drag and drop view controllers and view widgets. The storyboard is other technology to assist the design of user experience. With story board, the relationships between different view controllers are visually presented in the interface builder editor and separated from the code.

All view controllers should be implemented as a sub class of `UIViewController`. In the view controller classes, class attributes marked with “`@IBOutlet`” can be used to create a reference to an object in the views, which can be a widget, a constraint, or an attribute of the view. In Xcode this can be done by simply drag and drop.

The model part needs to be implemented completely by the developers. The Cocoa and Cocoa Touch framework provides a CoreData library to assist the development. The CoreData library handles the association of persistent data storage and in-memory object. With CoreData, the developer can focus more on the model itself. Another option would be to implement the model as own classes and use SQLite for persistent data storage.

The Model-View-Controller focus on the separation of the three components of software and achieve modularized development. With the Cocoa and Cocoa Touch framework and the assisting tools provided by Apple, the application development for iOS is simple, straight-forward and clear.

4 Results: Implementation of *Subti*Wiki v3.0

4.1 Database construction

A proper data model is vital to any software. For *Subti*Wiki, the task is to analyze and re-organize the collected annotations, build a proper data model and implement a relational database according to this data model.

4.1.1 Conceptual design

The conceptual design of the *Subti*Wiki data model is centered around the entity *gene*. Here, *gene* is an abstraction of a certain gene or RNA feature, its transcribed RNA (if any) and its translated protein (if any). Therefore, protein-protein interactions can be modelled as relationship between *genes*. This relationship is a many-to-many relationship as one protein can interact with multiple proteins.

We also included the entity *operon*. In genetics, an operon is functional unit of DNA which consists of one or more genes. Those genes are transcribed together into a single mRNA and share the regulatory mechanism in the transcriptional level. An *operon* can have one or more *gene(s)* and a *gene* can be in more than one *operon*. With the entity *operon* included, we can model the gene regulation on transcriptional level as the relationship between *regulator* and *operon*. The *regulator* here is an abstract entity. A *regulator* can be either a *gene* (a protein, in biological meaning) or a *riboswitch*.

Smaller molecules play vital roles in a living cell as solvent, reactant or ligand of biochemical reactions. Thus, we included *metabolite*, *reaction*, and *pathway* to properly present the biochemical reactions and pathways in an abstract level. A *pathway* is a collection of *reactions*. A *reaction* can be assigned to more than one *pathway*. A *reaction* is a collection of multiple *metabolites* and a *catalyst*. The *catalyst* here is an abstract entity, which can be of the type *gene* (protein in biological meaning) or *protein complex*, which is a group of *genes* (proteins in biological meaning). Both the relationship between *reaction* and *metabolite* and the relationship between *reaction* and *catalyst* are many-to-many relationships.

In *Subti*Wiki, genes/proteins are classified into different categories according to their functions. Thus, we included the entity *category*, which has no corresponding biological concept. The category system used in *Subti*Wiki is tree-structured. Thus, a one-to-many relationship is needed to describe the position of a certain *category* in the tree. The relationship between *gene* and *category* is many-to-many as a *gene* can be assigned to more than one *categories* and a *category* has multiple *genes*.

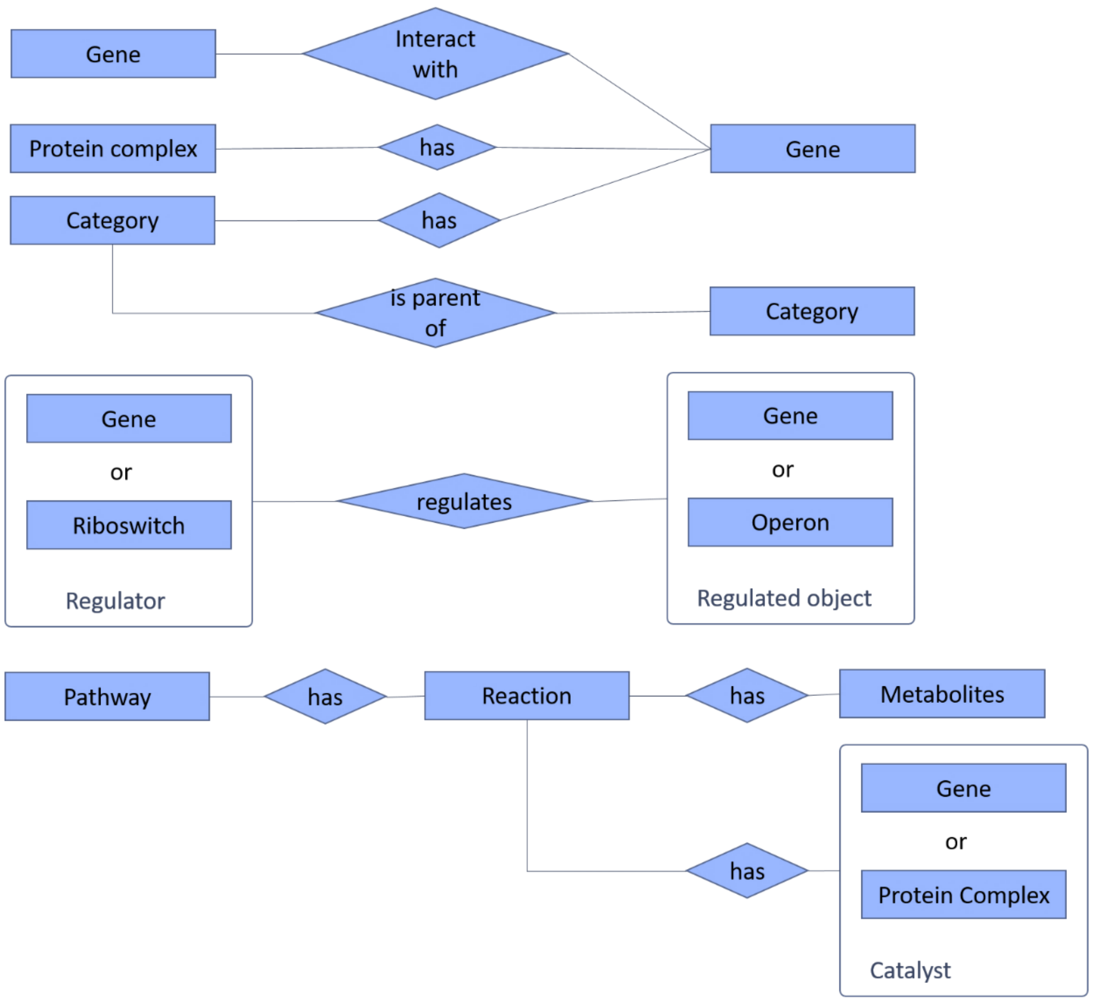


Figure 10. The ER-diagram showing the described conceptual schema of SubiWiki database.

For the references included in *Subti*Wiki, we included the entity *paper*. To fulfill the function of content management system, the data model of *Subti*Wiki should also include the entity *user* and the entity *history*, which is used to present the data of older versions. For the transcriptomic data and proteomic data, the entity *condition* is introduced, which presents the experimental condition of omics data.

4.1.2 Database implementation

We implemented a relational database based on the conceptual schema described in the previous chapter. The database management system we used is MySQL. Below are tables created in `subtiwiki_v3` database with short description. Detailed information about all tables including description of all column can be found in the appendix.

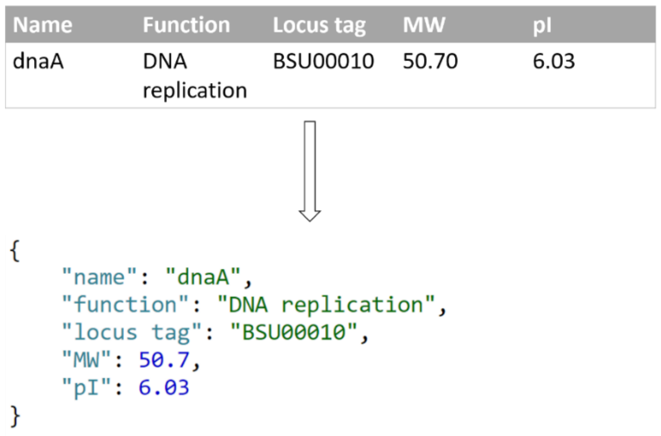
Name of table or view (case sensitive)	**Description**
Category	Stores information of categories in *Subti*Wiki
Chemical (beta)	Stores information of metabolites
Complex (beta)	Stores information of protein complexes
Condition	Stores information of the experimental condition of collected transcriptomic and proteomic data
Config	Stores system parameters about database changes
Gene	Stores annotations of genes
GeneCategory	Stores information of classification of genes into category
GenomicContext	Stores coordinates of genomic elements (genes, transcription upshift, transcription downshift) on the genome.
History	Stores older versions of all tables. This table collects only data from tables with a "id" field.
Interaction	Stores information of protein-protein interaction
MaterialViewGeneRegulation	This is a read only table maintained by triggers. No update or delete operation should be performed on this table. It functions like a view. Details will be discussed in the following chapter

Metabolite	Stores information of metabolites
Operon	Stores information of operons
Pathway	Stores information of pathways
PathwayGene	Stores information of proteins as catalyst in pathways
PathwayMetabolite	Stores information of metabolites in pathways
ProteinComplex (beta)	Stores information of enzymatic protein complexes
ProteomicData	Table of proteomic data
Pubmed	Stores information of papers
Reaction (beta)	Stores information of single biochemical reactions
Reaction_chemical (beta)	Stores information of metabolites in reaction
Reaction_enzyme (beta)	Stores information of enzymes in reaction
Regulation	Stores information of gene regulation, both transcriptional and translational
Regulon	Stores information about a specific regulon.
Sequence	Stores DNA and amino acid sequences
Statistics	Stores information about clicks on pages
Tracker	Stores information of every request, no ip address is recorded
TranscriptomicData	Table of transcriptomic data
ViewGeneOperon	View with the information of genes in operons
idmapping	Table which maps ids of genes (v3) to ids of genes (v2)
user	This is a copy of user table from MediaWiki database. This allows the v3 to use the same user account.

Table 3. A list of all tables in the `subtiwiki_v3` database with a short description. The tables marked with "beta" are not used in the production environment and do not have foreign key constraint with other tables.

4.1.2.1 The Gene table

During implementation, we were facing a big challenge raised by the complex structure of the entity *gene.* In previous *Subti*Wiki, a template is provided to keep each page in the wiki uniformed. This template is not compulsory during editing and no mechanism is applied so that each page confronts this template. In addition, it is highly valued to enable a flexible structure of data, at least for the *gene* entity. Thus, we didn't define a list of attributes for the entity *gene,* which should be implemented as columns in a table. Instead, we transform the annotations of genes into key-value pairs and serialize the data into text with JSON format. The JSON format is chosen because it is well supported by most of programming languages and is light-weighted.



Name	Function	Locus tag	MW	pI
dnaA	DNA replication	BSU00010	50.70	6.03

```
{
    "name": "dnaA",
    "function": "DNA replication",
    "locus tag": "BSU00010",
    "MW": 50.7,
    "pI": 6.03
}
```

Figure 11. From table to JSON.

Using the parsable text, the attributes of entity *gene* can be free defined for each instance. However, this trick lowers the performance of searching and indexing of the table, as searches on longer text is always more expensive than on shorter text or on other simpler data types. To compensate this disadvantage, we created index columns, which holds a copy of the value for a certain key path. In the implemented database, names of those columns start with an under score. The elements in key path are joined by "$$" to generate the unique column name. The *Gene* table has the schema shown in Table 4.

The "data" column of the *Gene* table has the "mediumtext" datatype. This is the column where the JSON formatted text is stored. In the implementation of server-side applications and client-side applications of *Subti*Wiki, we established a consistent code routine concerning this special column. More details will be discussed in chapter 4.2.1.3.

Column name	Column data type	Actual data type (if is text)	Description
id	char(40)	String	Unique id of the gene, a sha1 hash string
title	varchar(255)	String	Current name of the gene
data	mediumtext	JSON object	Text in JSON format
_locus	varchar(50)	String	Index column for key “locus”
_function	text	String	Index column for key “function”
_names	text	JSON array of strings	Index column for key “names”, which consists of all names of this gene
_mw	double	-	Molecular weight
_pI	double	-	Isoelectric point
_description	text	String	Description of this gene
_essential	varchar(10)	enum(“yes”,”no”)	Essentiality of this gene
_ec	varchar(30)	String	E.C. number of this gene
_geneLength	int(11)	-	Gene length
_proteinLength	int(11)	-	Protein length
bank_id	int(11)	-	ID in *Subti*Bank
count	int(11)	-	Access count of this record
lastUpdate	timestamp	-	The time when this record is last updated
lastAuthor	varchar(255)	String	The user who performed the last edit

Table 4. Schema of the Gene table. Columns whose names start with underscore serve as indexes.

In the new MySQL community server 5.8+, a new datatype named "JSON" is included. This new feature will provide even better support to our database layout.

4.1.2.2 The Regulation table and views

The expression of a gene can be affected by many different factors via different mechanisms in different stage of its life cycle. To simplify the model, we included two abstract entities in the conceptual design, i.e. *regulator* and *regulated object*. A *regulator* can be a *gene* or *riboswitch* and a *regulated object* can be a *gene* or an *operon*. In physical implementation, we created the `Regulation` table based on those two abstract entities. The schema of the table is described in the table below.

Column name	**Column data type**	**Description**
id	int(11)	The numeric id for version
regulator	varchar(255)	The id of the regulator, type indicated
regulated	varchar(255)	The id of the regulated object, type indicated
mode	varchar(255)	The mechanism of this regulation
lastAuthor	varchar(255)	Name of the user who last operated on this record
lastEdit	timestamp	The time when this record is last updated / created

Table 5. The schema of Regulation table in the database "subtiwiki_v3".

To indicate the type of record referred in the field regulator and regulated, a markup is used. This markup has two parts, a short string describing the record type and the referred record id. Figure 12 demonstrates the format of this markup.

This implementation is clear, simple and reduces data duplication. However, a rather complicated SQL syntax (Code block 10) with join clause must be used to retrieve the transcriptional regulatory information from Gene to Gene.

```
+----------------------------------------------------+---------------------------------------------------+
| regulator                                          | regulated                                         |
+----------------------------------------------------+---------------------------------------------------+
| {protein|00BCAAE16576DC5426A652580C69A570FC7A1C2C} | {operon|168BC8B91CBFE3EA050DA8A9187C8C67D6BBE4F9} |
+----------------------------------------------------+---------------------------------------------------+

+---------------------------------------------------+-------------------------------------------------+
| regulator                                         | regulated                                       |
+---------------------------------------------------+-------------------------------------------------+
| {protein|147AFEBA546FDBEEB08E9A8D9C7BCDC9B83CC90} | {gene|57B9E37F6232189CD47C1B41FDCF43FCB8016AEB} |
+---------------------------------------------------+-------------------------------------------------+
```

Figure 12. Records in the Regulation table in the production environment. The record above presents a transcriptional regulation. The record below presents a translational regulation.

```
SELECT regulator,
       gene,
         mode,
        description
FROM   Regulation
JOIN   ViewGeneOperon
ON     Regulation.regulated
LIKE Concat("{operon|", ViewGeneOperon.operon, "}")
WHERE  …
```

Code block 10. The SQL syntax to retrieve the transcriptional regulatory information from Gene to Gene without dependency on MaterialViewGeneRegulation.

The performance of this syntax has become the bottleneck for our regulation browser. It takes as long as 43.13 seconds to retrieve all rows (around 5800 rows). This is probably due to three reasons: First the syntax is dependent on the view `ViewGeneOperon` which is not very efficient (18.47 second to retrieve all records). Second the syntax has a string function "concat" which drags down the performance. Third, the column on which two table join has a string datatype. Comparisons on strings are always slower than numbers.

To solve this issue, we created a "material view" `MaterialViewGeneRegulation`, which is a table rather than a view in MySQL. This table ought to be read-only and maintained solely by triggers. Triggers are a stored process in MySQL which is executed when a given operation occurs on a given table. In our case, those triggers create / delete related records in the `MaterialViewGeneRegulation` table whenever a record is created or deleted in the Regulation table. The triggers are defined with the syntax in Code block 11.

Column name	Column datatype	Foreign key references	Description
gene	char(40)	Gene.id	The regulated gene
regulation	int(11)	Regulation.id	The related regulation

Table 6. Schema of MaterialViewGeneRegulation in database "subtiwiki_v3"

```
delimiter $$
CREATE definer = root@localhost trigger Regulation_AFTER_DELETE
  AFTER INSERT ON regulation FOR EACH row
BEGIN
  IF old.regulated LIKE "{operon%}"
     THEN DELETE FROM MaterialViewGeneRegulation
     WHERE regulation = old.id;
  END IF;
END $$
delimiter ;

delimiter $$
CREATE definer = root@localhost trigger Regulation_AFTER_INSERT
  AFTER INSERT ON regulation FOR EACH row
BEGIN
  IF NEW.regulated LIKE "{operon|%}" THEN
    INSERT INTO MaterialViewGeneRegulation
                (gene,
                 regulation)
    SELECT gene,
           NEW.id
    FROM   ViewGeneOperon
    WHERE  operon = Substr(NEW.regulated, 9, 40);
  END IF;
END $$
delimiter ;
```

Code block 11. The SQL syntax used to define triggers to maintain the table MaterialViewGeneRegulation.

To minimize data duplication and inconsistency caused by hard copy, the table `MaterialViewGeneRegulation` only has two columns, namely gene and regulation, which refer to the id columns in Gene and Regulation table respectively. A simpler and more efficient SQL syntax (Code block 12) can then be used to retrieve the transcriptional regulatory information from Gene to Gene.

```
SELECT regulator,
       gene,
       mode,
       description
FROM   MaterialViewGeneRegulation
JOIN   Regulation
ON     MaterialViewGeneRegulation.regulation = regulation.id
WHERE  ...
```

Code block 12. The more efficient syntax to retrieve the transcriptional regulatory information from Gene to Gene with the table MaterialViewGeneRegulation.

With this query syntax, the latency is reduced to 0.03 second to retrieve all data (around 5800 rows). This is a significant improvement of performance and greatly increased the user experience of our regulation browser, which always retrieves the whole regulatory network at once and processes this network with graph algorithms.

4.1.2.3 The Reaction tables

Three tables in subtiwiki_v3 database, namely `Reaction`, `Reaction_chemical`, `Reaction_enzyme`, are designed to store information of biochemical reactions in the cell of *Bacillus subtilis.* Due to limited time and man power, those tables are only partially finished and are not yet integrated to the rest of the database.

Column	**Data type**	**Extra**
id	int(11)	primary key, auto increment
pathway	int(11)	
reversible	int(1)	default yes

Table 7. The table scheme of table Reaction

Column	Data type	Extra
id	int(11)	primary key, auto_increment
title	varchar(255)	unique
synonym	varchar(255)	
pubchem	int(11)	

Table 8. Scheme of table Chemical. It is named "Chemical" to differentiate it from the "Metabolite" table. The "pubchem" column stores id of the chemical in PubChem database.

Column	Data Type	Extra
id	char(40)	primary key
title	varchar(255)	unique

Table 9. The schema of table Complex. It is named "Complex" to differentiate it from the "ProteinComplex" table.

Column	Data type	Extra
id	int(11)	primary key, auto increment
reaction	int(11)	
chemical	int(11)	
side	enum('L', 'R')	
isMain	int(1)	

Table 10. The scheme of table Reaction_chemical

Column	Data type	Extra
id	int(11)	primary key, auto increment, not null
reaction	int(11)	not null
enzyme	varchar(255)	not null
modification	varchar(45)	
position	varchar(255)	

Table 11. The scheme of table Reaction_enzyme

The biochemical reactions in a living cell are mostly reversible reactions. Therefore, we do not differentiate between reactants and products. In the table `Reaction_chemical`, the column "side" indicates where the metabolite should locate in the reaction diagram and "reversible" column indicates whether this reaction is reversible or not.

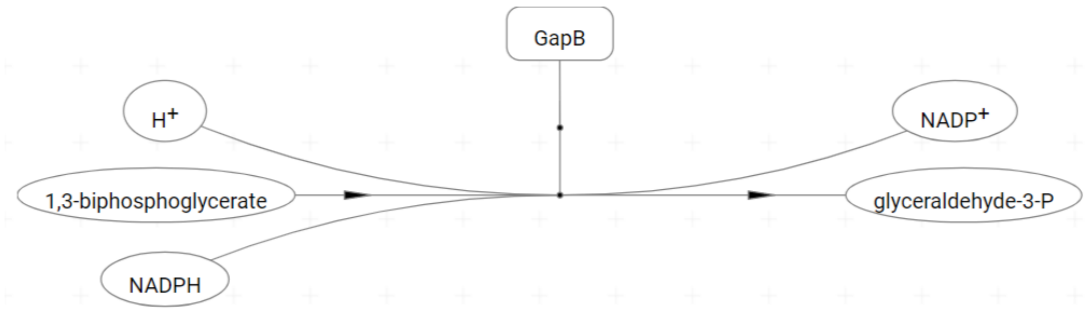


Figure 13. The diagram of a biochemical reaction catalyzed by GapB

For display reasons, we divided the metabolites involved in the reaction into two groups. The main metabolites will be on the main axis of the diagram while the other metabolites will be placed around them. The main metabolites in the reaction are also seen as the connecting point to another reaction.

The `enzyme` column in the table `Reaction_enzyme` uses the same markup described in the previous chapter, as an *enzyme* here can be either a *gene* or a *protein complex.*

4.1.2.4 The Category table

The category system used in *Subti*Wiki has a tree-like structure. Storing such hierarchical data in a relational database has long been discussed and quite a few options are applicable for MySQL, such as adjacency list, nested set, materialized path, bridge table, etc. In the conceptual design, we included the relationship "is the parent of" between *categories* to present the tree structure, which should be implemented as adjacency list. However, we decided to use materialized path approach instead due to a few reasons: The category system is well developed and relative stable. Queries on the level of a certain node occur often at the server backend and frontend.

In the materialized path approach, each node stores its full path to the root node. In *Subti*Wiki, we created a unique id based on its path for each category. All ids start with "SW" and contains a series of numbers separated by a dot ".". The id of a child category is composed of the id of its parent and its position among its siblings. For example, the category with id "SW 1.1.2" is the second child of the category with id "SW 1.1" and the category with id "SW 1.1" is the first child of "SW 1".

Title	ID1	ID2
ABC transporters for the uptake of iron/ siderophores	SW 1.3.3.1	SW 2.6.5.4
Acquisition of iron / Other	SW 1.3.3.3	SW 2.6.5.1
Acquisition of iron/ based on similarity	SW 1.3.4	SW 2.6.5.2
Biosynthesis of antibacterial compounds	SW 2.6.6.1	SW 4.3.15
Biosynthesis of lipoteichoic acid	SW 1.1.1.3	SW 2.6.1.2
Biosynthesis of peptidoglycan	SW 1.1.1.1	SW 2.6.1.1
Biosynthesis of teichoic acid	SW 1.1.1.4	SW 2.6.1.3
Biosynthesis of teichuronic acid	SW 1.1.1.5	SW 2.6.1.4
Biosynthesis of the carrier lipid undecaprenylphosphate	SW 1.1.1.8	SW 2.6.1.5
Elemental iron transport system	SW 1.3.3.2	SW 2.6.5.5
Genetic competence	SW 3.1.7	SW 4.1.3
Other protein controlling the activity of the phosphorelay	SW 3.4.7.6	SW 4.2.2.6
Phosphatases controlling the phosphorelay	SW 3.4.7.5	SW 4.2.2.5
phosphorelay	SW 3.4.7	SW 4.2.2
Proteins controlling the activity of the kinases	SW 3.4.7.2	SW 4.2.2.2
Sigma factors	SW 3.2.1.2	SW 3.4.1.1
The kinases	SW 3.4.7.1	SW 4.2.2.1
The phosphotransferases	SW 3.4.7.3	SW 4.2.2.3
The ultimate target	SW 3.4.7.4	SW 4.2.2.4
Utilization of amino sugars	SW 2.2.2.17	SW 2.3.3.3

Table 12. A list of categories sharing the same name and genes.

Using this approach, we can calculate the parent's id from the id of a child category. The level of a certain node corresponds to how many numbers there are in the id. And a regular syntax can be used to retrieve children of different levels.

During the transfer of data from old wiki to the new database, we discovered that several categories share the same names and have the same genes assigned to them. Those categories have different paths but should be treated as same category. Therefore, in practice we use “title” as identifier for update. A list of such categories are shown in Table 12.

4.2 Construction of server-side applications

Server-side applications serve as middle layer between the clients and database. They analyze the requests from users, perform database operations, receive data from database, and create dynamic web pages. For *Subti*Wiki, the complexity of data adds difficulty to the development process. Thus, we have created a small framework to modularize the code structure and to reduce coupling.

4.2.1 The framework

The framework we created consists of a few loosely connected modules. A core module sorts the requests according to URLs, invokes designated handlers, manages database connection, and keeps logs. A database model implements two data access models and provides a set of pre-defined secure methods for database operations. The controller manager and the model manager organize the handlers for view controlling and data accessing. Finally, a template module helps to generate dynamic web pages in a uniform style.

4.2.1.1 Request process routine

We implemented a routine for request processing in the core module (Figure 14). All requests coming from the clients are first redirected to `src/init.php` by the web server. Each request is sorted by its URL and forwarded to its designated handler. The handler consumes the request and returns a value indicating whether this procedure is successful. If not successful, an error message is delivered.

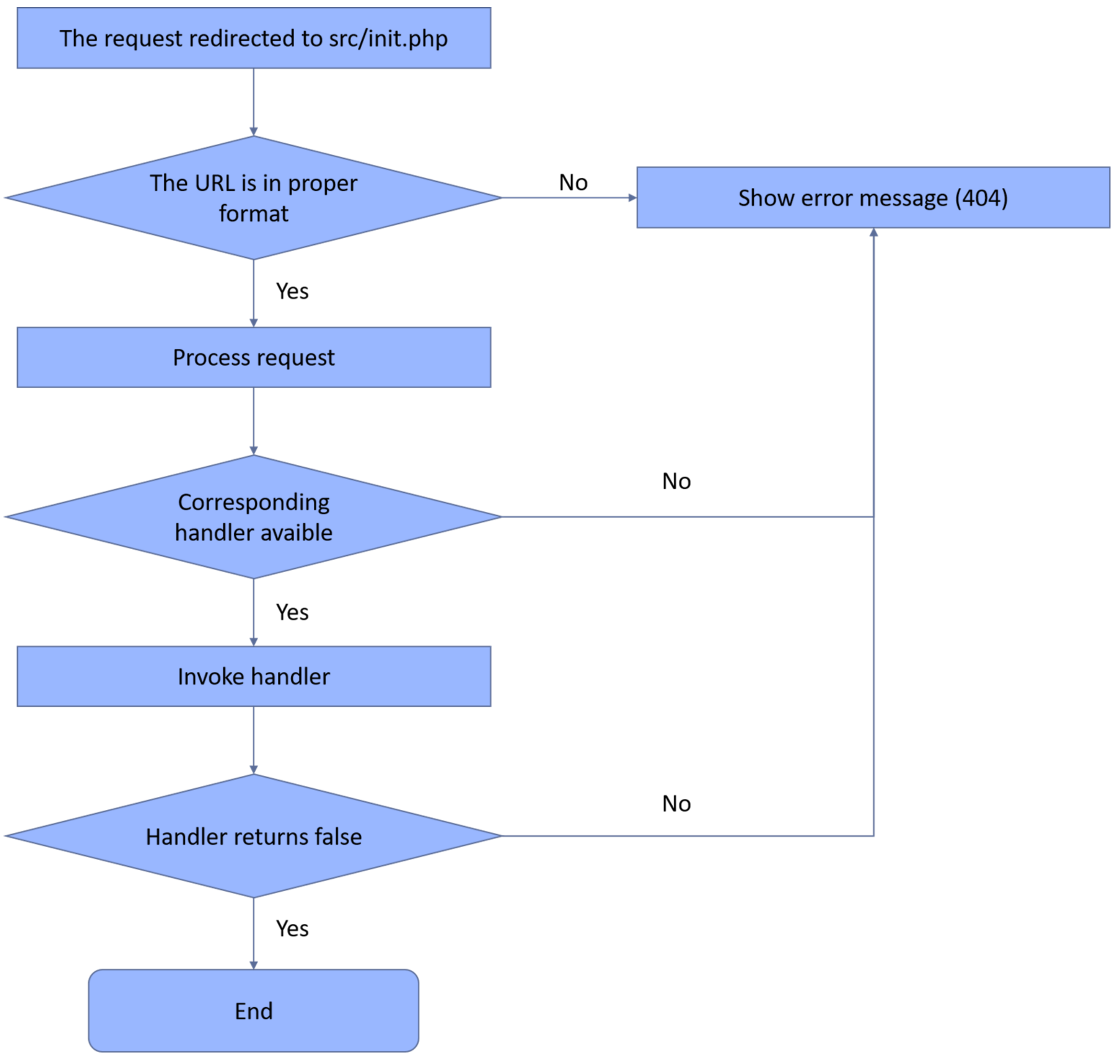

The request redirected to src/init.php
The URL is in proper format
No
Show error message (404)
Yes
Process request
Corresponding handler avible
No
Yes
Invoke handler
Handler returns false
No
Yes
End

Figure 14. Flow chart describing the routine for request processing implemented in the core module.

The handlers are the only concern for the developers under this framework. They implement the business logic for each application. They are associated directly with URLs of certain format and organized by the `ControllerManager` class. With the static `ControllerManager::register` method, a function or closure can be assigned as the handler for a certain action of a certain data model. For example, the code block below assigns an anonymous closure as the handler for “view” action of “gene”. The output of this closure can be accessed with the URL “http://domain /path/gene/view/12". In this URL, the path component “12” is passed as the first argument to the closure. More components are allowed and passed to the closure respectively.

```
ControllerManager::register("gene", "view", function($id) {
    echo "information of gene with the id $id will show";
});
```

Code block 13. The sample code snippet for registering a handler with ControllerManager::register method.

4.2.1.2 Common logic for handlers

The server of *Subti*Wiki focuses more on data presentation and interactivity than data processing or heavy computing. Thus, the logic for most handlers can be broken down to a few major steps (Figure 15). First the user input should not be trusted for security reasons. Database operations such as create, read, update, or delete are than expected. Based on the results returned from the database, a proper response is required. If the database operation is successful, data presentation ought to be prepared. If not, an error message should be delivered.

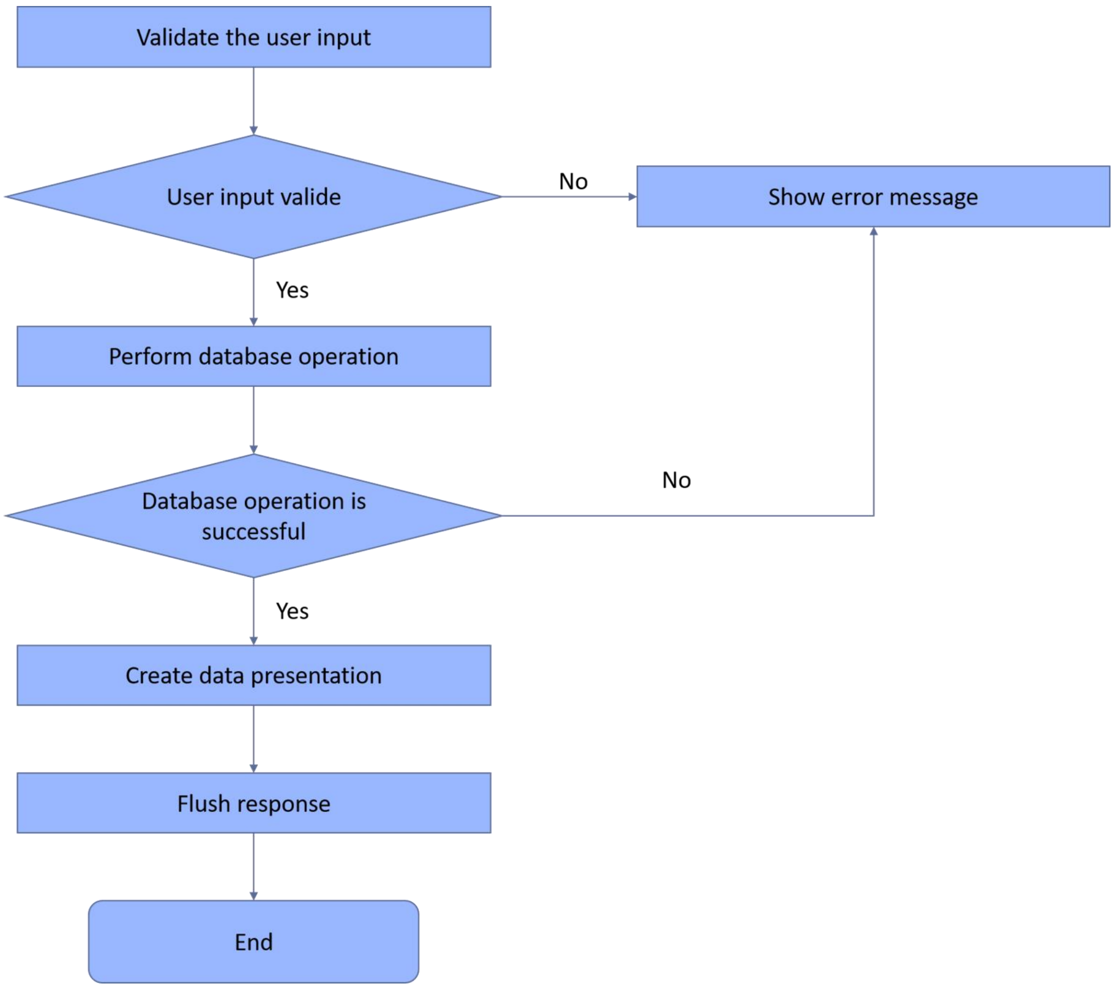


Figure 15. Flow chart of common logic implemented in handlers in SubtiWiki.

It is recommended to encapsulate the database operations in the data access handlers organized by the `ModelManager` class. They provide an extra layer of abstraction and modularization when the requested data are in complex format. The template module can help to generate dynamic web pages which share the same static part. Using the template module can keep the code clean, reduce the trouble of escaping special characters in string literals and allow better control of universal structure and style of the whole website.

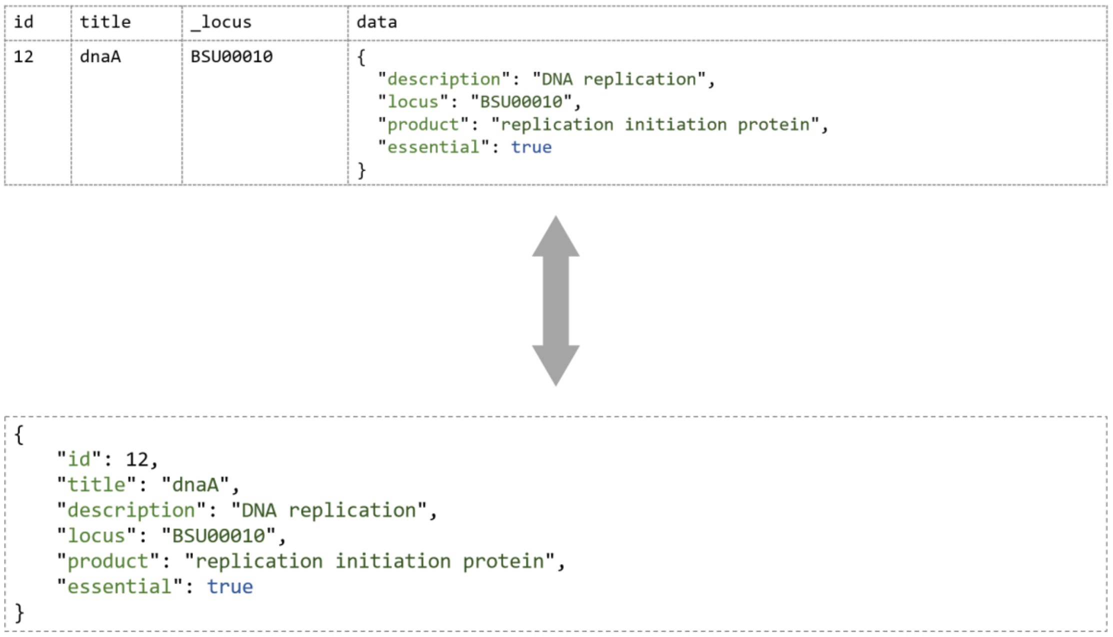
More information is available in the document provided within the installation package of this framework. This framework is freely available under the URL “http://subtiwiki.uni-goettingen.de/Monkey.zip".

4.2.1.3 Adapted data access objects

In the previous chapter we have described the technique when implementing the Gene table. By storing JSON text in a MySQL table, we created a mixed database which is relational and document-oriented at the same time. This implementation brings extra steps for routine database operations such as create, read, and update. For example, when reading a record from Gene table, the JSON text should be parsed into native PHP data structures. When updating or creating a record in Gene table, the JSON text should be validated and corresponding index columns should also be updated at the same time. Those extra steps are not the concerns of server-side applications as they should focus on business logic.

Therefore, a layer of abstraction is needed to hide the detailed database structure from the applications and database module is implemented to serve as data access objects. It implements active record pattern and an adapted data access pattern specific for our database layout.

In our data access pattern, the “data” (column name can be assigned differently) column of a table is considered a virtual container of the data. From the aspect of applications which initiate database operations, this column does not exist. It has a text datatype and stores the JSON text. When a record of such table is retrieved, an object, let’s name it *row,* is mapped from the record using active record pattern: column names to keys and fields to values. The JSON text which is under the key “data” is then parsed into an object and key-value pairs of this object will be copied to the *row* object and the key “data” is deleted. An example is shown in Figure 16 how a record is mapped with an object.



id	title	_locus	data
12	dnaA	BSU00010	{ "description": "DNA replication", "locus": "BSU00010", "product": "replication initiation protein", "essential": true }

```
{
    "id": 12,
    "title": "dnaA",
    "description": "DNA replication",
    "locus": "BSU00010",
    "product": "replication initiation protein",
    "essential": true
}
```

Figure 16. The pattern maps a record to an object.

When a record needss to be updated, an object is mapped to a record. In this case, the other columns are updated first, and corresponding keys are consumed according to whether this is an index column or not. The object, with unconsumed keys, is encoded to JSON and stored in the "data" column.

In this way, we included features of document oriented database into a relational database.

4.2.2 The applications

Currently *Subti*Wiki has 22 applications corresponding to all data models. The applications are listed in the table below.

Applications	**Description**
bank	resolves compatibility issue with the previous *Subti*Wiki versions, providing redirections
category	provides viewing and editing functions for categories
complex (beta)	provides viewing, editing functions for protein complexes in biochemical pathways
expression	provides access to transcriptomics and proteomics data

gene	provide viewing and editing functions for gene pages
geneCategory	enables adding and removing gene to/ from category
geneTranslationalRegulation	provides editing function to translational regulations of genes
genome	provides access to DNA sequences
genomicContext	provides access to coordinates of genes
history	enables version control
interaction	provides viewing, editing, and removing functions of protein-protein interactions
operon	provides viewing, editing, and removing functions of operons
pathway	provides access of data of biochemical pathways
proteinComplex (beta)	provides viewing, editing, and removing functions of large protein complexes such as ribosomes etc.
pubmed	enables access to references in *Subti*Wiki
reaction	provides viewing, editing, and removing functions of biochemical reactions
regulation	provides viewing, editing, and removing functions of transcriptional regulations
regulon	provides viewing and editing functions of additional information about the regulators
statistics	provides viewing of access data of *Subti*Wiki
subtiwiki	provides redirects to previous *Subti*Wiki versions
user	handles user information

Table 13. The applications in SubtiWiki

4.2.2.1 The Application “Gene”

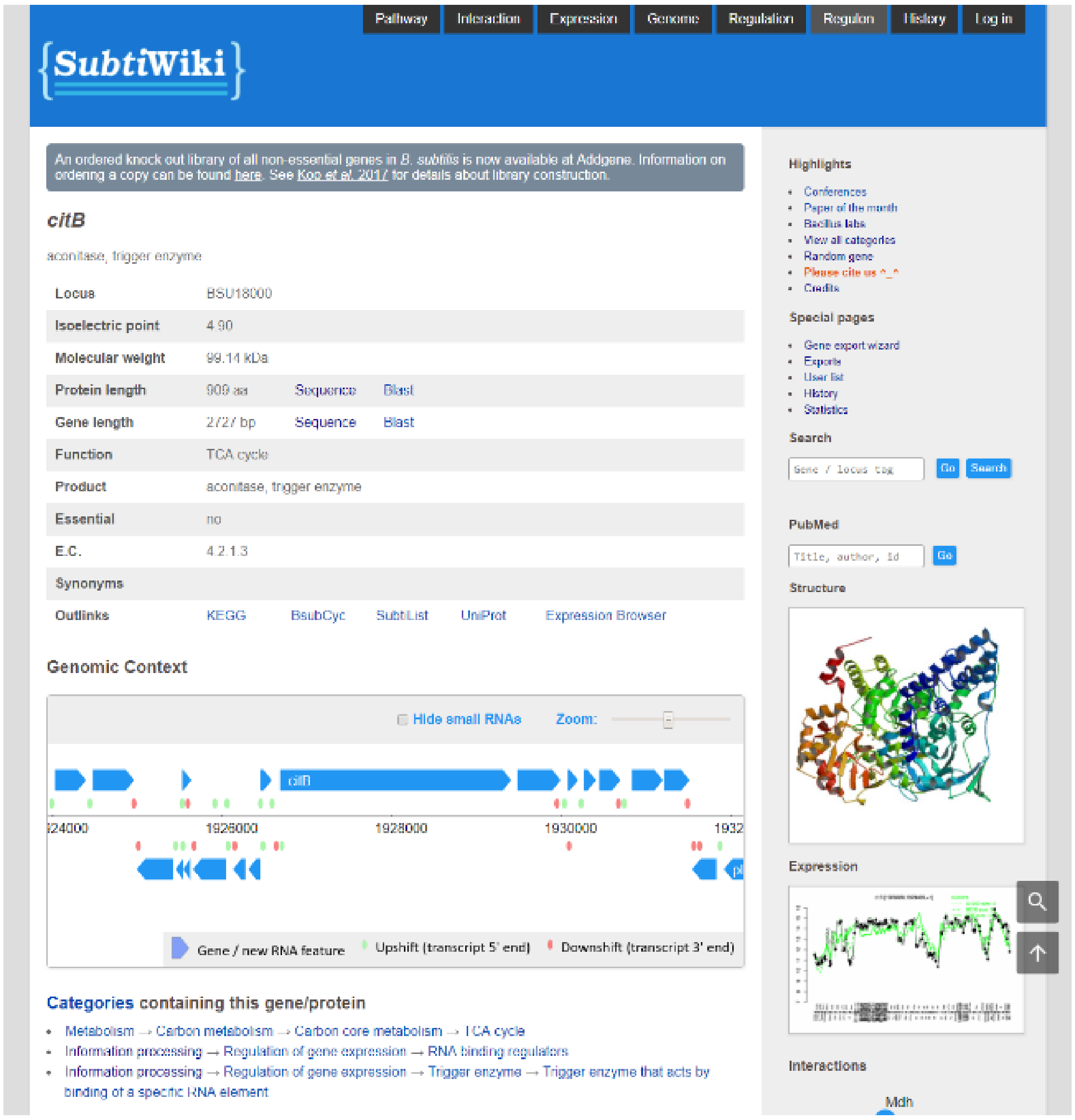
The application “gene” is the center of *Subti*Wiki. It generates pages for genes/proteins and offers the editing interface for updating the information. To be consistent with the previous versions, only minimal changes are made to web site layout. We have updated the genomic context browser with more interactivity and included diagrams of interaction and regulation on the side panel. Some minor adjustments are also made to offer a better reading experience. A partial screenshot of a gene page is shown in Figure 17.

The updated genomic context browser allows the user to move up or down the genome with mouse wheel. The JavaScript code for the genomic library is extended and rewritten into a library which can be applicable for other biological or bioinformatic websites. This library can display both linear and circular genomes. In circular mode, this library will show the end of genome overlapping with the starting point.

As described in the previous chapter, we have implemented the Gene table in the way that attributes of entity *gene* can be freely defined for each instance. Therefore, the way to present our data should be data-driven as well. To achieve that, we simply defined a template adapter (details see inline document) which prints out the JSON object to HTML recursively. This method will save us trouble because no extra code is needed in the future when the structure of data is updated. It is also consistent with our editing interface: It shows exactly whatever the user actually edits.

However, this “print” technique brings a problem. The designed web pages of genes contain information which is not stored in the Gene table. That information needs to be integrated and presented at a specific position on the web page. To resolve this issue, we used a “[[this]]” markup to indicate the location and displayed title of this information. For example, the key-value pair `{genomicContext: “[[this]]”}` will be displayed as the genomic context browser.

For editing, we introduced a markup system. This markup system not only indicates the style of the page, but also the structure the page. In this markup system, all section titles follow one or multiple asterisk “*” depending on the level of the title and a space is required between the asterisk and the actual title.



An ordered knock out library of all non-essential genes in *B. subtilis* is now available at Addgene. Information on ordering a copy can be found here. See Koo et al. 2017 for details about library construction.

citB

aconitase, trigger enzyme

Locus	BSU18000		
Isoelectric point	4.90		
Molecular weight	99.14 kDa		
Protein length	909 aa	Sequence	Blast
Gene length	2727 bp	Sequence	Blast
Function	TCA cycle		
Product	aconitase, trigger enzyme		
Essential	no		
E.C.	4.2.1.3		
Synonyms			
Outlinks	KEGG BsubCyc SubtiList UniProt Expression Browser		

Genomic Context

Categories containing this gene/protein

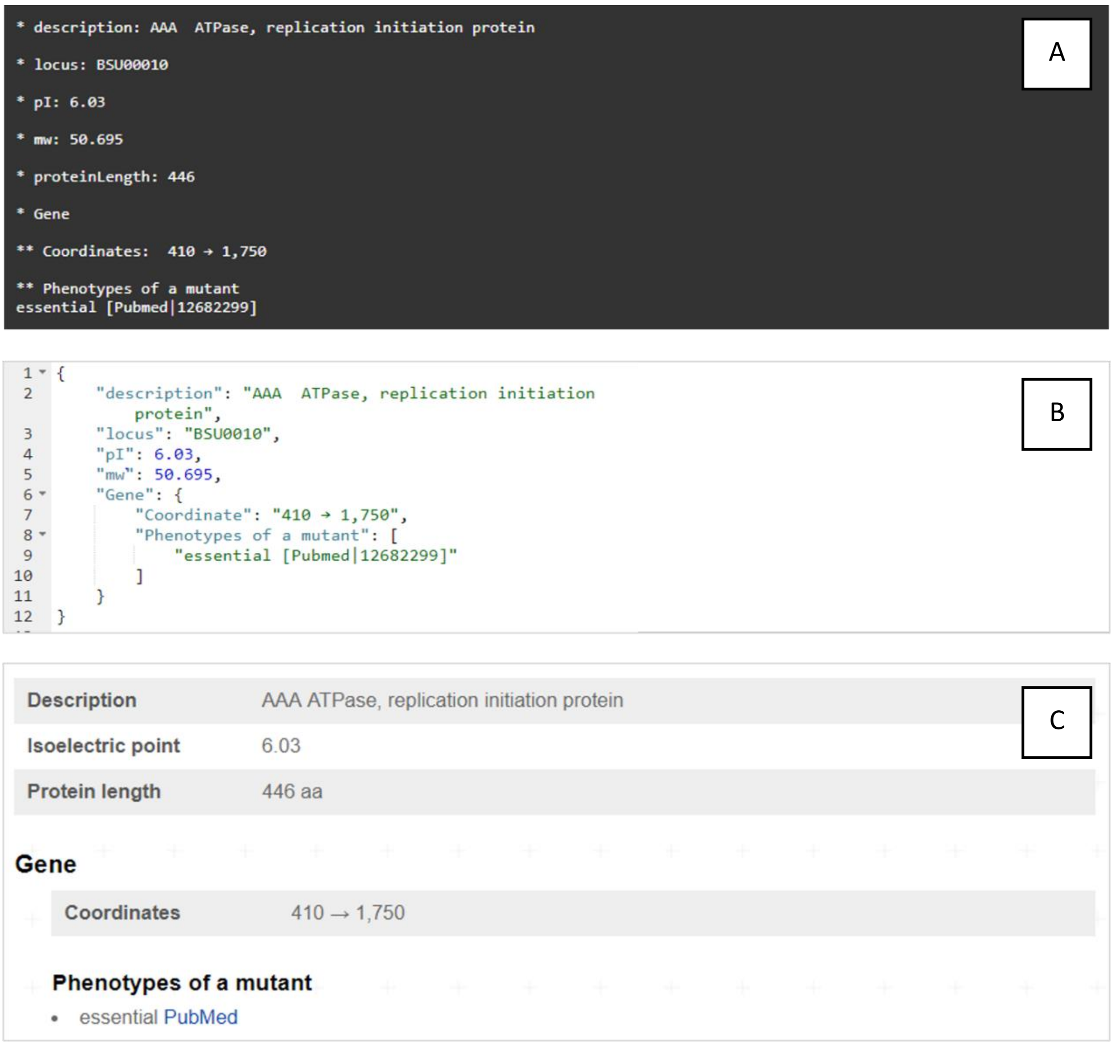
- Metabolism → Carbon metabolism → Carbon core metabolism → TCA cycle
- Information processing → Regulation of gene expression → RNA binding regulators
- Information processing → Regulation of gene expression → Trigger enzyme → Trigger enzyme that acts by binding of a specific RNA element

Figure 17. The partial screenshot of the web page for the gene citB.

Contents of a section can be placed either right behind the title, following a ":" and a space, or under the title in a new line. A new line in this edit box will be a new bullet point on the parse web page and multiple new empty lines are ignored. For gene's page, a universal template is provided. Information can be added to the intended section.

In the MediaWiki engine, a markup such as "[[citB]]" generates a link to the page with the title "citB". In our new markup, such feature is kept and adjusted. To create a link to the page of *citB*, a markup in the format "[[gene|citB]]" is required. The text "gene" before the vertical bar indicates the type of the link, in other words, name of associated data model.

More details about the new markup is available at our FAQ page and a quick reference table is accessible in the editing interface. Quick tool buttons are also provided with hot key combinations.



```
* description: AAA  ATPase, replication initiation protein

* locus: BSU00010

* pI: 6.03

* mw: 50.695

* proteinLength: 446

* Gene

** Coordinates:  410 → 1,750

** Phenotypes of a mutant
essential [Pubmed|12682299]
```

```
{
    "description": "AAA  ATPase, replication initiation
        protein",
    "locus": "BSU0010",
    "pI": 6.03,
    "mw": 50.695,
    "Gene": {
        "Coordinate": "410 → 1,750",
        "Phenotypes of a mutant": [
            "essential [Pubmed|12682299]"
        ]
    }
}
```

Description	AAA ATPase, replication initiation protein
Isoelectric point	6.03
Protein length	446 aa

Gene

Coordinates	410 → 1,750

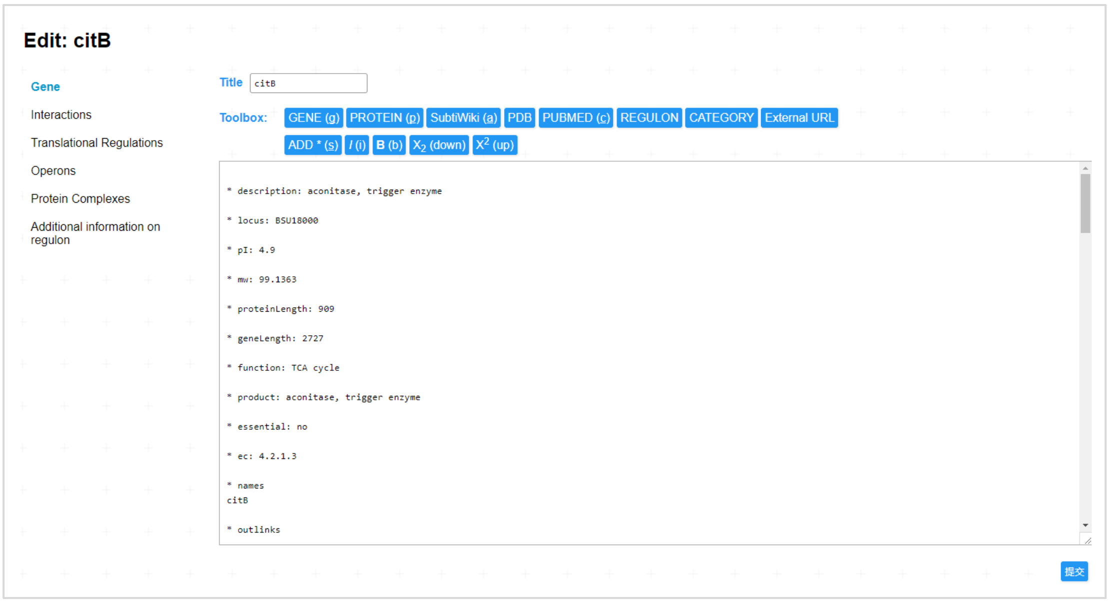
Phenotypes of a mutant

- essential PubMed

Figure 18. A: The new markup system indicates the structure of the page. B: The source text in the new markup is parsed to a JSON object in the front end. C: The JSON object is printed to a HTML page.

In the editing process, when the user hits the submit button, a grammar check is first performed to ensure correct structure of the data. The source text is parsed to JSON and sent to the server-side, where a second validation is performed. The editing can be rejected when the submitted data is too small in comparison to the older version, indicating vandalism.

We created this markup and avoided using JSON directly in the editing interface because of a few reasons. This markup is closer to natural language than JSON. This markup is more tolerant with grammatic errors than JSON. This markup does not require escaping strings as string is the only primitive data type here.



citB

```
* description: aconitase, trigger enzyme

* locus: BSU18000

* pI: 4.9

* mw: 99.1363

* proteinLength: 909

* geneLength: 2727

* function: TCA cycle

* product: aconitase, trigger enzyme

* essential: no

* ec: 4.2.1.3

* names
citB

* outlinks
```

Figure 19 The editing interface of gene citB.

In the gene's editing interface, we included portals of all editing interfaces concerning this gene/protein, such as protein-protein interaction, regulation, operons, etc. Those portals are listed on the left panel of the editing interface. Table 14 describes the intention of each section.

Section name	**Description**
Gene	General information about this gene, transcribed RNA, and translated protein (if any).
Interactions	Information on protein-protein interactions.
Translational regulations	regulation which happens at translational level, transcriptional regulation please go to "Operons" section
Operon	all operons this gene is a part of, information of transcriptional regulations which affecting the whole data stored here.
Protein Complexes	functional protein complexes, still under construction, but you are welcome to add in data already.
Additional information on regulon	If this protein is a regulator, some information about this protein as regulation can be added here, such as the reference papers or etc.

Table 14. Description of different sections in the editing interface.

As we cleaned and structured all text, exports of data by specific key chain is made possible. On the “gene export wizard” page, the user can select desired key chains and a csv file is generated according to selection.

More information about the application “gene”, its provided actions, and public APIs is included in the appendix.

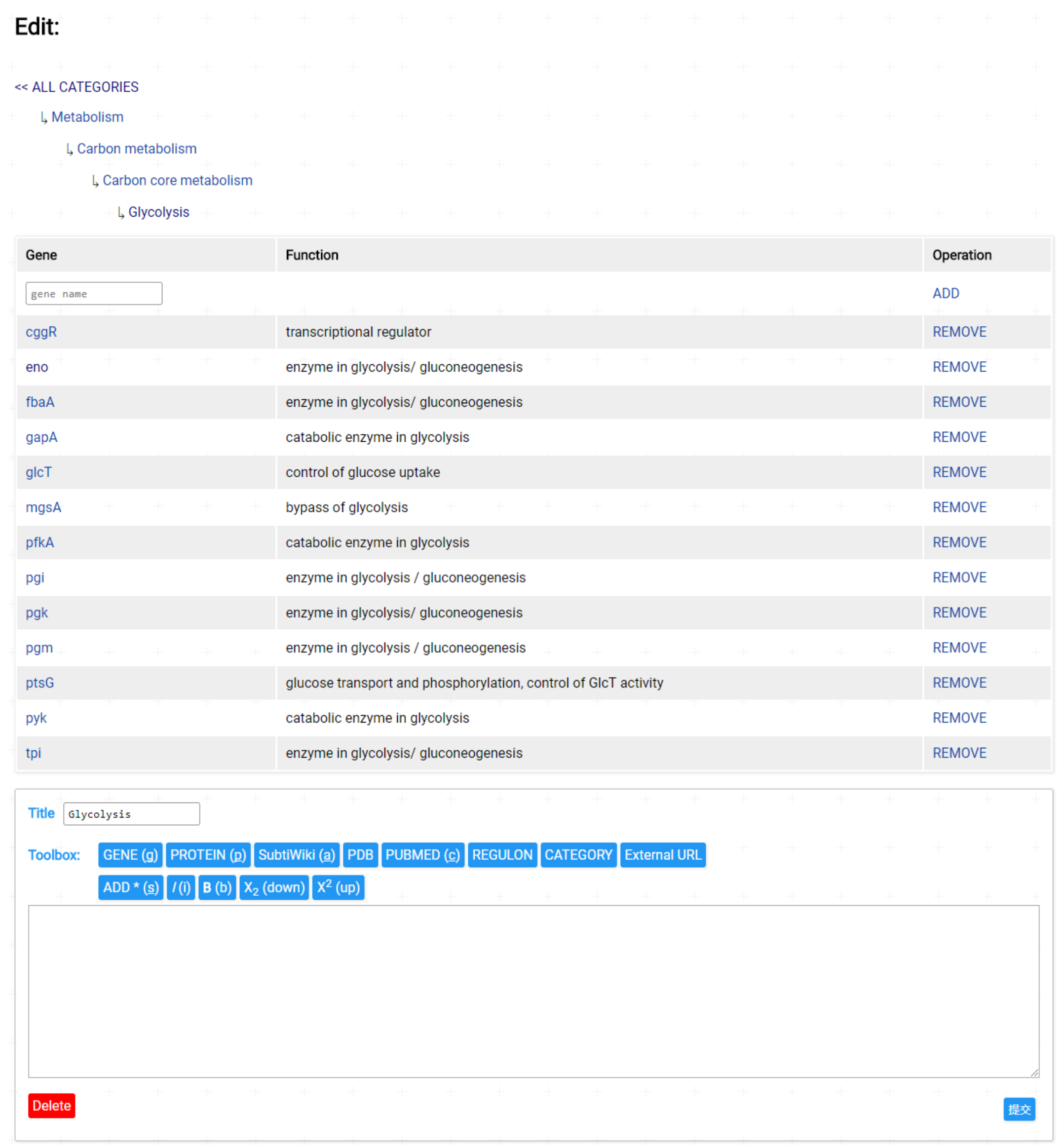
4.2.2.2 The application “Category”

In the previous version of *Subti*Wiki, the category system was maintained by hand. It requires two edits to assign a gene to a category: add link to the gene page on the category page and add link to the category page on the gene page. This doubled the amount of work and resulted in data inconsistency. Therefore, we have developed the application “category” for better data management.

At the application index page (http://subtiwiki.uni-goettingen.de/category), all categories are listed in a tree view. Subcategories can be extended or collapsed. This provides a better overview of the whole category system.

The page of the individual category includes more information. It shows the path from the root to the current category, all its child categories (if any), genes assigned to this category, descriptions and references of this category, and the sibling categories.

In the editing interface of each category, the user can rename this category, add a description and references. All genes assigned to this category will also be shown in a table format. The user can delete genes from this category or add new genes to it. The administrator of *Subti*Wiki can also add a new category in the editing interface of the designated parent category. Deletion of a category can only be performed by administrators of *Subti*Wiki when the category doesn’t have any gene assigned to it or any subcategories. Deletion of a category will result re-labeling of all its sibling categories.



Edit:

<< ALL CATEGORIES
↳ Metabolism
↳ Carbon metabolism
↳ Carbon core metabolism
↳ Glycolysis

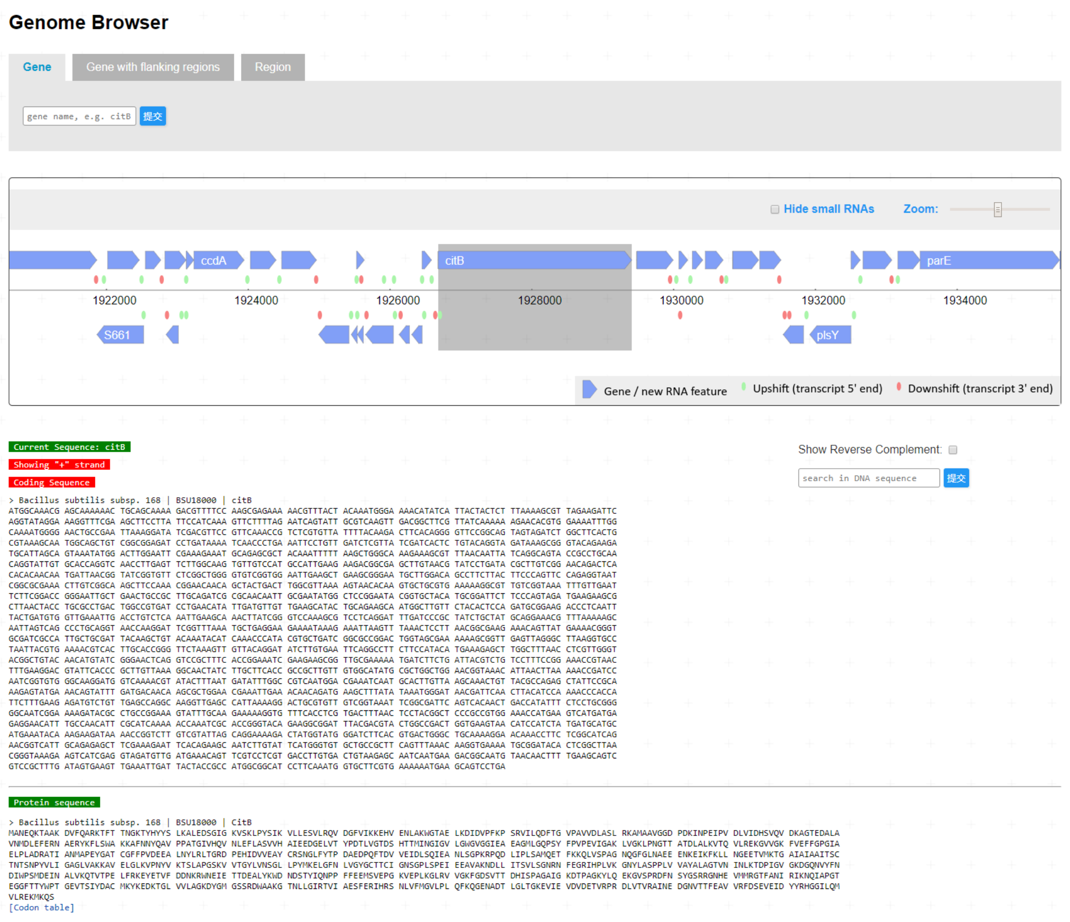
Gene	Function	Operation
gene name		ADD
cggR	transcriptional regulator	REMOVE
eno	enzyme in glycolysis/ gluconeogenesis	REMOVE
fbaA	enzyme in glycolysis/ gluconeogenesis	REMOVE
gapA	catabolic enzyme in glycolysis	REMOVE
glcT	control of glucose uptake	REMOVE
mgsA	bypass of glycolysis	REMOVE
pfkA	catabolic enzyme in glycolysis	REMOVE
pgi	enzyme in glycolysis / gluconeogenesis	REMOVE
pgk	enzyme in glycolysis/ gluconeogenesis	REMOVE
pgm	enzyme in glycolysis / gluconeogenesis	REMOVE
ptsG	glucose transport and phosphorylation, control of GlcT activity	REMOVE
pyk	catabolic enzyme in glycolysis	REMOVE
tpi	enzyme in glycolysis/ gluconeogenesis	REMOVE

Title: Glycolysis

*Figure 20. The editing interface of category "Glycolysis". On the top the path to this category is listed. A table of all genes assigned to this category is displayed and operations such as add and delete can be performed. With the form at the bottom of the page, name of this category can be changed, and description of this category can be added. Only administrators of Subti*Wiki *can delete categories.*

4.2.2.3 The genome browser

The major motivation to develop a genome browser for *Subti*Wiki is to provide our users a combined view of genomic elements and corresponding sequences. In the genome browser, the user can focus on a specific gene or a region. As the genome browser only handles static data, no editing interface is provided.



Current Sequence: citB

Showing "+" strand

Coding Sequence

```
> Bacillus subtilis subsp. 168 | BSU18000 | citB
ATGGCAAACG AGCAAAAAAC TGCAGCAAAA GACGTTTTCC AAGCGAGAAA AACGTTTACT ACAAATGGGA AAACATATCA TTACTACTCT TTAAAAGCGT TAGAAGATTC
AGGTATAGGA AAGGTTTCGA AGCTTCCTTA TTCCATCAAA GTTCTTTTAG AATCAGTATT GCGTCAAGTT GACGGCTTCG TTATCAAAAA AGAACACGTG GAAAATTTGG
CAAAATGGGG AACTGCCGAA TTAAAGGATA TCGACGTTCC GTTCAAACCG TCTCGTGTTA TTTTACAAGA CTTCACAGGG GTTCCGGCAG TAGTAGATCT GGCTTCACTG
CGTAAAGCAA TGGCAGCTGT CGGCGGAGAT CCTGATAAAA TCAACCCTGA AATTCCTGTT GATCTCGTTA TCGATCACTC TGTACAGGTA GATAAAGCGG GTACAGAAGA
TGCATTAGCA GTAAATATGG ACTTGGAATT CGAAAGAAAT GCAGAGCGCT ACAAATTTTT AAGCTGGGCA AAGAAAGCGT TTAACAATTA TCAGGCAGTA CCGCCTGCAA
CAGGTATTGT GCACCAGGTC AACCTTGAGT TCTTGGCAAG TGTTGTCCAT GCCATTGAAG AAGACGGCGA GCTTGTAACG TATCCTGATA CGCTTGTCGG AACAGACTCA
CACACAACAA TGATTAACGG TATCGGTGTT CTCGGCTGGG GTGTCGGTGG AATTGAAGCT GAAGCGGGAA TGCTTGGACA GCCTTCTTAC TTCCCAGTTC CAGAGGTAAT
CGGCGCGAAA CTTGTCGGCA AGCTTCCAAA CGGAACAACA GCTACTGACT TGGCGTTAAA AGTAACACAA GTGCTGCGTG AAAAAGGCGT TGTCGGTAAA TTTGTTGAAT
TCTTCGGACC GGGAATTGCT GAACTGCCGC TTGCAGATCG CGCAACAATT GCGAATATGG CTCCGGAATA CGGTGCTACA TGCGGATTCT TCCCAGTAGA TGAAGAAGCG
CTTAACTACC TGCGCCTGAC TGGCCGTGAT CCTGAACATA TTGATGTTGT TGAAGCATAC TGCAGAAGCA ATGGCTTGTT CTACACTCCA GATGCGGAAG ACCCTCAATT
TACTGATGTG GTTGAAATTG ACCTGTCTCA AATTGAAGCA AACTTATCGG GTCCAAAGCG TCCTCAGGAT TTGATCCCGC TATCTGCTAT GCAGGAAACG TTTAAAAAGC
AATTAGTCAG CCCTGCAGGT AACCAAGGAT TCGGTTTAAA TGCTGAGGAA GAAAATAAAG AAATTAAGTT TAAACTCCTT AACGGCGAAG AAACAGTTAT GAAAACGGGT
GCGATCGCCA TTGCTGCGAT TACAAGCTGT ACAAATACAT CAAACCCATA CGTGCTGATC GGCGCCGGAC TGGTAGCGAA AAAAGCGGTT GAGTTAGGGC TTAAGGTGCC
TAATTACGTG AAAACGTCAC TTGCACCGGG TTCTAAAGTT GTTACAGGAT ATCTTGTGAA TTCAGGCCTT CTTCCATACA TGAAAGAGCT TGGCTTTAAC CTCGTTGGGT
ACGGCTGTAC AACATGTATC GGGAACTCAG GTCCGCTTTC ACCGGAAATC GAAGAAGCGG TTGCGAAAAA TGATCTTCTG ATTACGTCTG TCCTTTCCGG AAACCGTAAC
TTTGAAGGAC GTATTCACCC GCTTGTTAAA GGCAACTATC TTGCTTCACC GCCGCTTGTT GTGGCATATG CGCTGGCTGG AACGGTAAAC ATTAACTTAA AAACCGATCC
AATCGGTGTG GGCAAGGATG GTCAAAACGT ATACTTTAAT GATATTTGGC CGTCAATGGA CGAAATCAAT GCACTTGTTA AGCAAACTGT TACGCCAGAG CTATTCCGCA
AAGAGTATGA AACAGTATTT GATGACAACA AGCGCTGGAA CGAAATTGAA ACAACAGATG AAGCTTTATA TAAATGGGAT AACGATTCAA CTTACATCCA AAACCCACCA
TTCTTTGAAG AGATGTCTGT TGAGCCAGGC AAGGTTGAGC CATTAAAAGG ACTGCGTGTT GTCGGTAAAT TCGGCGATTC AGTCACAACT GACCATATTT CTCCTGCGGG
GGCAATCGGA AAAGATACGC CTGCCGGAAA GTATTTGCAA GAAAAAGGTG TTTCACCTCG TGACTTTAAC TCCTACGGCT CCCGCCGTGG AAACCATGAA GTCATGATGA
GAGGAACATT TGCCAACATT CGCATCAAAA ACCAAATCGC ACCGGGTACA GAAGGCGGAT TTACGACGTA CTGGCCGACT GGTGAAGTAA CATCCATCTA TGATGCATGC
ATGAAATACA AAGAAGATAA AACCGGTCTT GTCGTATTAG CAGGAAAAGA CTATGGTATG GGATCTTCAC GTGACTGGGC TGCAAAAGGA ACAAACCTTC TCGGCATCAG
AACGGTCATT GCAGAGAGCT TCGAAAGAAT TCACAGAAGC AATCTTGTAT TCATGGGTGT GCTGCCGCTT CAGTTTAAAC AAGGTGAAAA TGCGGATACA CTCGGCTTAA
CGGGTAAAGA AGTCATCGAG GTAGATGTTG ATGAAACAGT TCGTCCTCGT GACCTTGTGA CTGTAAGAGC AATCAATGAA GACGGCAATG TAACAACTTT TGAAGCAGTC
GTCCGCTTTG ATAGTGAAGT TGAAATTGAT TACTACCGCC ATGGCGGCAT CCTTCAAATG GTGCTTCGTG AAAAAATGAA GCAGTCCTGA
```

Protein sequence

```
> Bacillus subtilis subsp. 168 | BSU18000 | CitB
MANEQKTAAK DVFQARKTFT TNGKTYHYYS LKALEDSGIG KVSKLPYSIK VLLESVLRQV DGFVIKKEHV ENLAKWGTAE LKDIDVPFKP SRVILQDFTG VPAVVDLASL RKAMAAVGGD PDKINPEIPV DLVIDHSVQV DKAGTEDALA
VNMDLEFERN AERYKFLSWA KKAFNNYQAV PPATGIVHQV NLEFLASVVH AIEEDGELVT YPDTLVGTDS HTTMINGIGV LGWGVGGIEA EAGMLGQPSY FPVPEVIGAK LVGKLPNGTT ATDLALKVTQ VLREKGVVGK FVEFFGPGIA
ELPLADRATI ANMAPEYGAT CGFFPVDEEA LNYLRLTGRD PEHIDVVEAY CRSNGLFYTP DAEDPQFTDV VEIDLSQIEA NLSGPKRPQD LIPLSAMQET FKKQLVSPAG NQGFGLNAEE ENKEIKFKLL NGEETVMKTG AIAIAAITSC
TNTSNPYVLI GAGLVAKKAV ELGLKVPNYV KTSLAPGSKV VTGYLVNSGL LPYMKELGFN LVGYGCTTCI GNSGPLSPEI EEAVAKNDLL ITSVLSGNRN FEGRIHPLVK GNYLASPPLV VAYALAGTVN INLKTDPIGV GKDGQNVYFN
DIWPSMDEIN ALVKQTVTPE LFRKEYETVF DDNKRWNEIE TTDEALYKWD NDSTYIQNPP FFEEMSVEPG KVEPLKGLRV VGKFGDSVTT DHISPAGAIG KDTPAGKYLQ EKGVSPRDFN SYGSRRGNHE VMMRGTFANI RIKNQIAPGT
EGGFTTYWPT GEVTSIYDAC MKYKEDKTGL VVLAGKDYGM GSSRDWAAKG TNLLGIRTVI AESFERIHRS NLVFMGVLPL QFKQGENADT LGLTGKEVIE VDVDETVRPR DLVTVRAINE DGNVTTFEAV VRFDSEVEID YYRHGGILQM
VLREKMKQS
[Codon table]
```

Figure 21. The screenshot of the genome browser displaying genomic context of citB, its DNA sequence, and protein sequence.

In the *Subti*Wiki database, coordinates of genomic elements are stored. These coordinates are used to extract DNA sequences of desired genes or RNA features. The protein sequences are translated from the DNA sequence using the common codon table for prokaryotes. The link to the codon table is included at the end of protein sequence.

We also enabled a very simple sequence search function for the DNA sequences. This can be helpful for finding restriction enzyme sites in the displayed DNA sequence.

4.2.2.4 The network browsers and NetVis

After all the data about gene regulation have been cleaned from text and put into the relational database, it become extremely interesting for us to visualize the whole network. In the previous *Subti*Wiki version (*Subti*Bank), a JavaScript library was implemented to display the protein-protein interaction network. In this library, a circular layout is used with all nodes (proteins) put on a large circle and straight lines representing the edges. However, this implementation is not suitable for large and inter-connected networks such as the regulatory network.

Therefore, we switched to force directed drawing algorithms introduced in chapter 3.3.3. We used a JavaScript library (Visjs[41]) which implements Kawaii algorithm[47] and Barnes-Hut simulation[51] to draw the network in the front end. This provides us more interactivity and lowers the computational burden of our server. The back end of the regulation browser is mainly responsible for retrieving the subnetworks to be displayed.

It is unrealistic to display the whole regulatory network at once. The simulation is extremely long, and the generated visualization is overwhelmed with large overlapping clusters. Therefore, we offer to display a subnetwork of a certain gene. This subnetwork is defined by two factors, a gene and a radius. The regulation network is treated as an undirected network when the subnetwork is calculated. This subnetwork consists of all nodes whose distances to the selected node (gene) are no greater than the radius. The subnetworks are calculated with an algorithm adapted from the Dijkstra algorithm [58].

The SigA regulon is by default excluded from the subnetwork of genes other than *sigA.* As the house-keeping sigma factor, SigA controls a lot of genes in *B. subtilis* and sharing SigA as sigma factor doesn't provide a lot of information.

For example, the subnetwork of gene *citB* consists of all genes directly regulated by CitB and all proteins which directly regulate *citB* when the radius is 1. When radius is 2, the subnetwork includes all neighboring genes of previous subnetwork.

With higher radius, more and more genes are included in the subnetwork. Figure 22 shows the regulatory network around *citB* with the radius of two. In this presentation, we clearly observe regulatory clusters and the association between them.

The most important parameter for the gravity model is the global gravity constant. The global gravity constant defines how hard nodes repel each other. With a smaller gravity constant (be aware this value is negative), the nodes tend to push each other further away, lengthen the edges, and result a clearer presentation. In our regulation browser, the gravity constant is defined exponential to the nodes count for better presentations.

The edges are color labelled according the regulatory mechanism each edge represents. To avoid too much color, all regulator mechanisms are sorted into three categories, namely positive (green color), negative (red color), and other (gray color). Those colors can be adjusted individually in the settings panel. A highlight function is also available to search for a series of gene in large networks. When highlighted, the genes stay opaque while the rest of the network fades away. The users can highlight multiple genes at once and remove the highlight simply by clicking on the blank area.

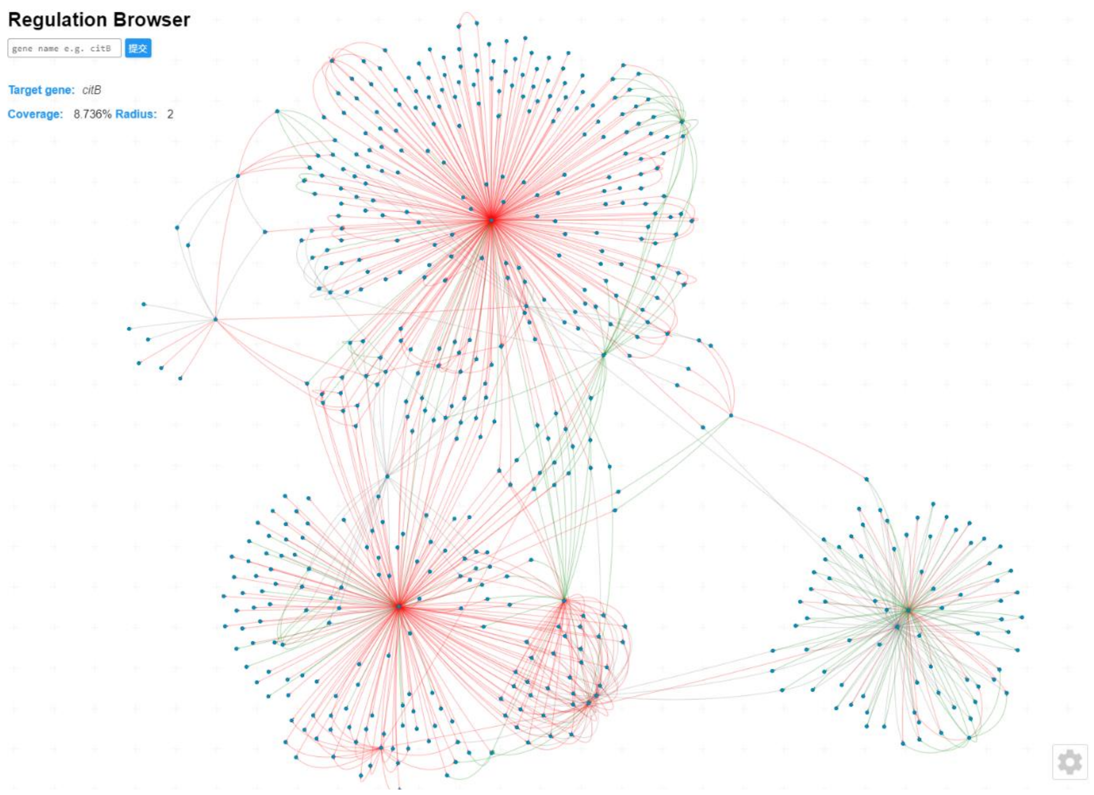


Figure 22. The subnetwork of citB with the radius of 2.

The users can also overlay transcriptomic and proteomic data with the regulatory network. Data from a certain experiment can be selected from the control panel. The nodes are then colored according to the expression level with a color legend at the lower left corner.

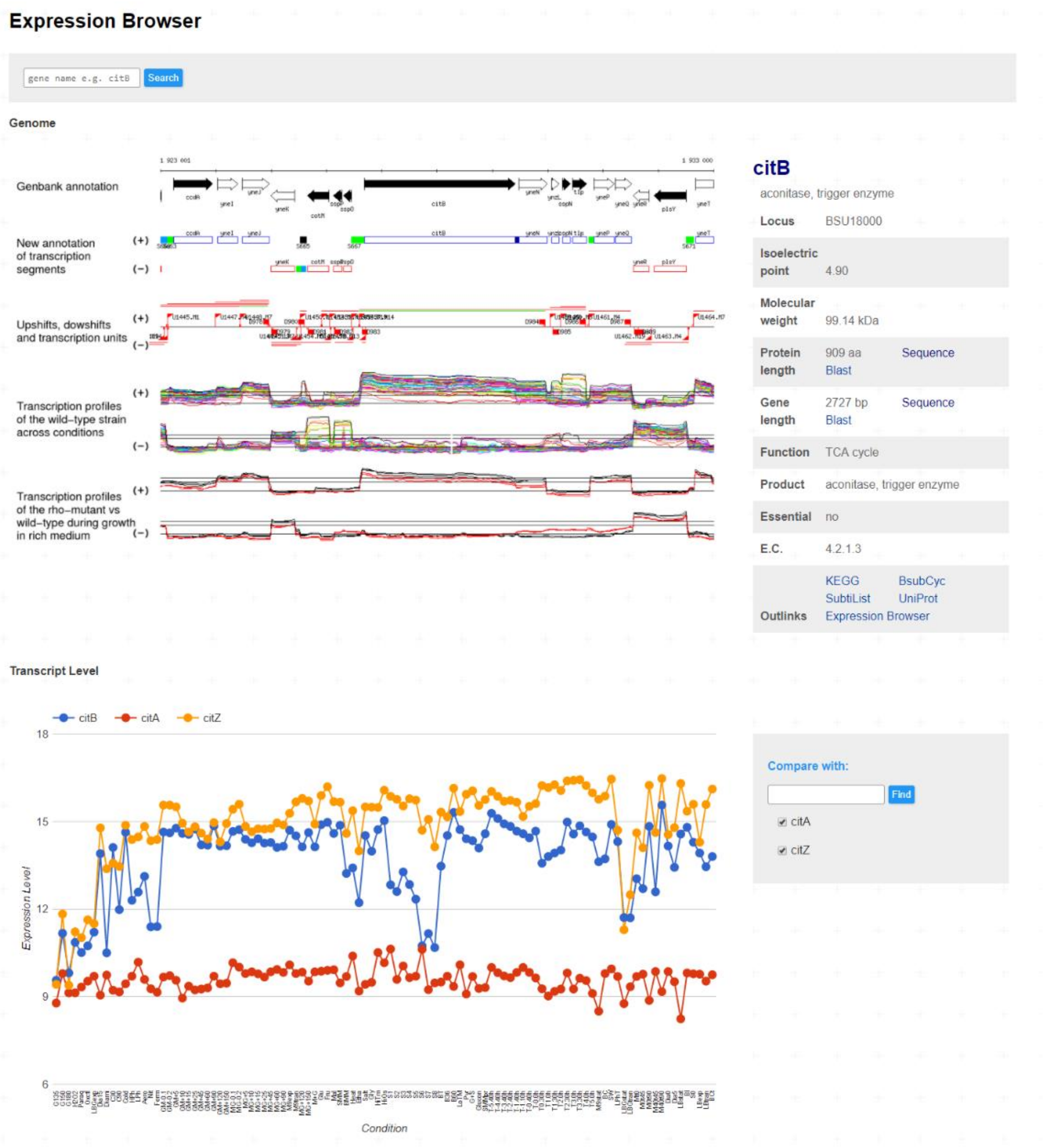
The export function is enabled for the regulation browser, too. Right click on the blank area brings the menu for different formats. Data can be exported in csv format, which is text based and can be processed in Excel. The current viewport can be exported as an image file (png format). The “nvis” format is intended for our network visualization tool NetVis. This file consists of coordinates of nodes and no additional time is needed for simulation.

The data for whole regulatory network can be exported in the data export page of *Subti*Wiki. The data is in csv format which is good for both human reading and processing with scripts.

The interaction browser is implemented with the same logic as regulation browser.

In addition to our online tool which integrates data collected by *Subti*Wiki, we offer a cross platform network visualization tool NetVis. With NetVis, users can visualize their own data and adjust the style of the visualization. Simple network manipulations such as adding or deleting nodes or edges are enabled. The users can also save the results of a simulation to avoid waiting time.

4.2.2.5 The expression browser


Expression Browser
gene name e.g. citB
Search
Genome
Genbank annotation
New annotation of transcription segments
Upshifts, dowshifts and transcription units
Transcription profiles of the wild-type strain across conditions
Transcription profiles of the rho-mutant vs wild-type during growth in rich medium
Transcript Level
citB
citA
citZ
Expression Level
Condition
18
15
12
9
6

citB

aconitase, trigger enzyme

Locus	BSU18000	
Isoelectric point	4.90	
Molecular weight	99.14 kDa	
Protein length	909 aa Blast	Sequence
Gene length	2727 bp Blast	Sequence
Function	TCA cycle	
Product	aconitase, trigger enzyme	
Essential	no	
E.C.	4.2.1.3	
Outlinks	KEGG SubtiList Expression Browser	BsubCyc UniProt

Compare with:

Find

☑ citA

☑ citZ

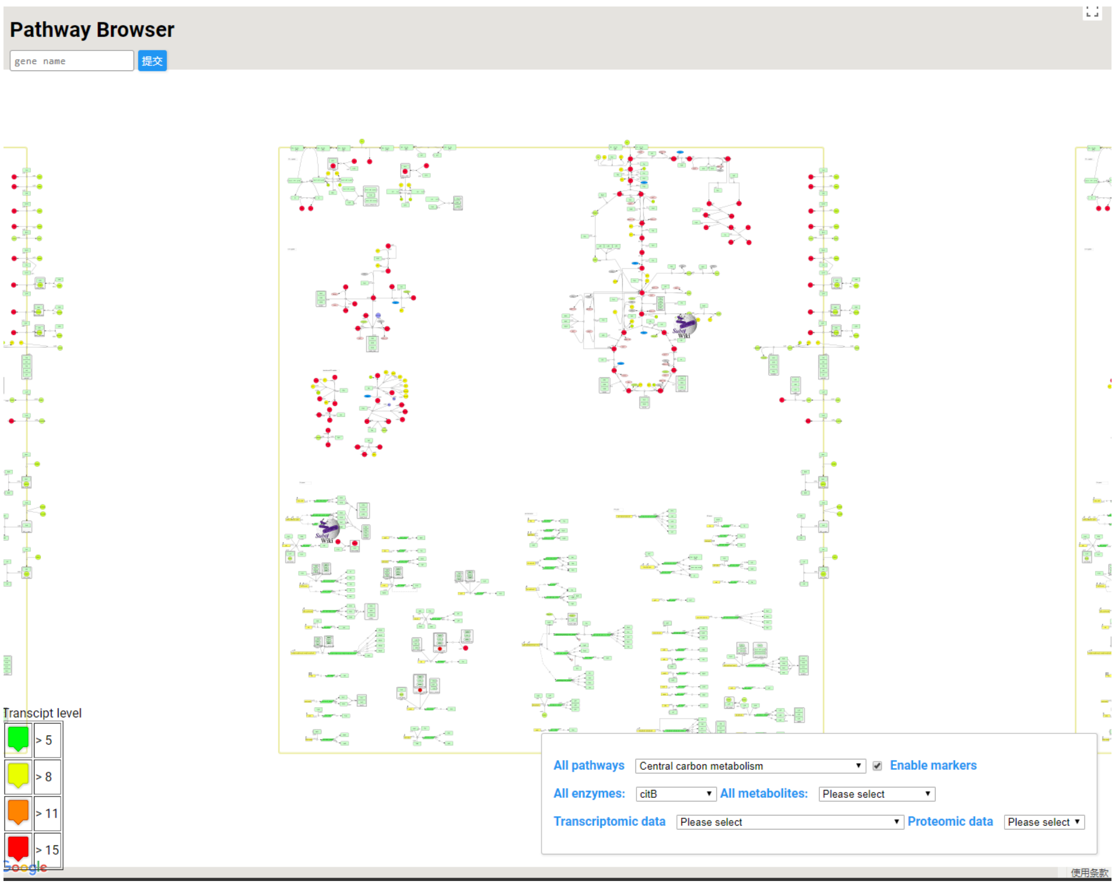
Figure 23. The expression data of citB in comparison with citA and citZ.

We updated the expression browser for better visualization and a uniform web site design throughout the *Subti*-Apps. In the updated expression browser, google charts library is used to draw the interactive diagrams. The comparison function is extended, now allowing comparison of multiple genes.

4.2.2.6 The pathway browser

We updated the pathway browser for better interactivity sand a uniform web site design throughout the *Subti*-Apps.

In the updated pathway browser, the markers on the pathway map can be turned off for a clearer view. Selecting an enzyme from a drop menu in the control panel will highlight all occurrences of the selected enzyme with markers. The same logic applies to metabolites as well.

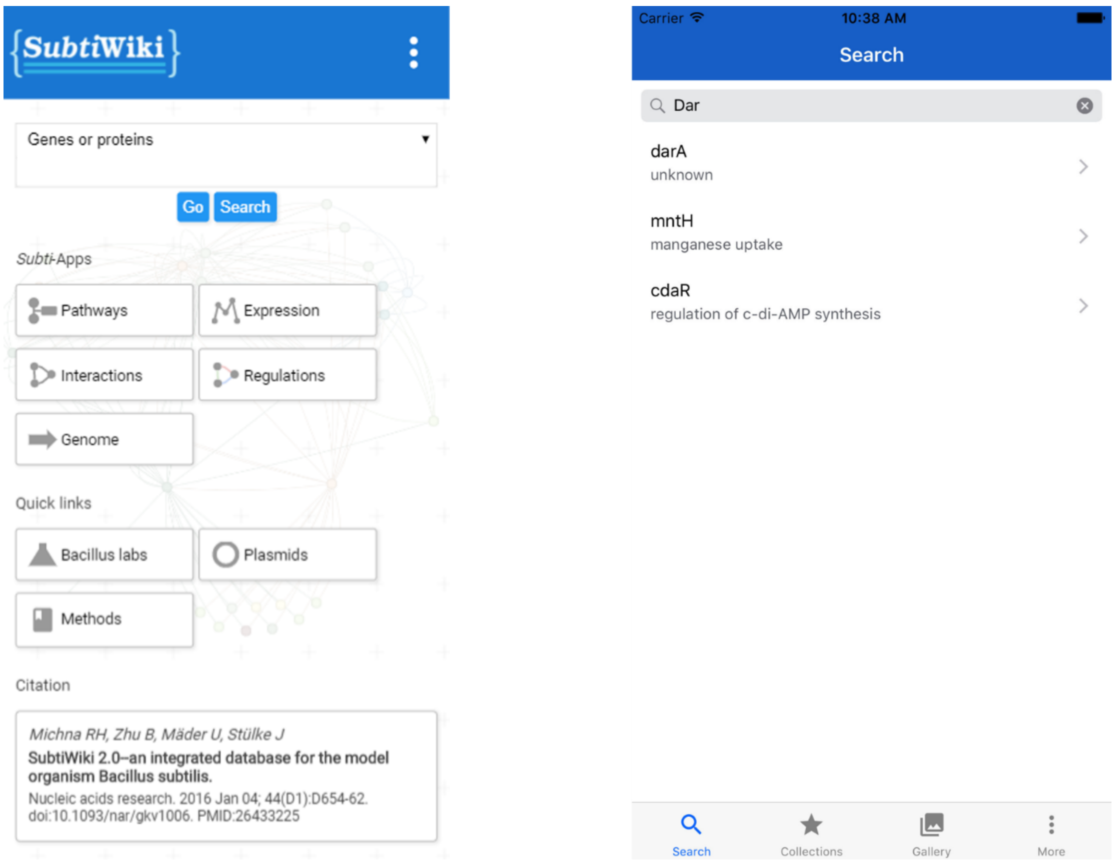


All pathways: Central carbon metabolism ✓ Enable markers
All enzymes: citB
All metabolites: Please select
Transcriptomic data: Please select
Proteomic data: Please select

Figure 24. Pathway map for central carbon metabolism, with occurrences of citB highlighted

4.3 The Mobile Apps

With the popularity of smart phones and tablets, the habit of internet users has undergone significant changes. Mobile internet and mobile apps has become an important aspect of life. Therefore, we took effort to ensure better mobile access to our web site. First, we have adjusted the layout and style of the web pages to better fit the smaller screens (Figure 25). We also developed mobile apps for both Android and iOS.



Michna RH, Zhu B, Mäder U, Stülke J
SubtiWiki 2.0–an integrated database for the model organism Bacillus subtilis.
Nucleic acids research. 2016 Jan 04; 44(D1):D654-62. doi:10.1093/nar/gkv1006. PMID:26433225

darA
unknown

mntH
manganese uptake

cdaR
regulation of c-di-AMP synthesis

Figure 25. The index page of SubtiWiki on mobile device and the search tab of the SubiWiki iOS App.

We designed the mobile apps as quick dictionary for gene annotations and focus more on the extended offline functions. The user can search for a gene using name or locus tag. Detailed information about a selected gene shows on a separated screen. The user can add genes, categories, and regulons to their favorite. This saves the added items locally for accessibility when device is offline. Notes in text format or images can be added to the app and associated with certain genes. For iOS users, local data, including all notes, images, and favorite items are saved in iCloud and synced among devices.

4.3.1 Local and remote data storage

We used the SQLite database on both Android and iOS apps for local data storage. Since the mobile app doesn't provide the function of editing, the data model and corresponding database layout is rather simple. We have entities like *Gene, Category, Regulon* which are consistent with the database on our server. For offline functions, we introduced *Image* and *Collection.* Foreign keys are enabled to keep the data consistent.

Table name	**description**	**Foreign key references**
Gene	table to store gene added to favorites	None
Category	table to store category added to favorites	None
Regulon	table to store regulon added to favorites	None
Image	table to store information about imported images, including a md5 hash value of image and imported timestamp	None
Collection	table to store information about user created collection of genes	None
ImageGene	table to store association between genes and images	image -> Image.id gene -> Gene.id
CollectionGene	table to store association between genes and user created collections	collection -> Collection.id gene -> Gene.id

Table 15. The database layout for mobile apps

For iOS users, user created data can be synchronised among devices with the help of iCloud. Under the CloudKit framework, the development frame work for iCloud provided by Apple, a private remote database can be established for each user. The layout of this remote database is very similar to the one of the local database.

4.3.2 WebView to present gene pages

Both Android and iOS provide a system component called WebView which can load HTML pages and even run JavaScript. We took advantage of this component to present gene pages.

As described in previous chapters, the attributes of *gene* can be freely defined, which means all information should be presented in a data driven way. Thus, we applied the same idea on the mobile end too. The Apps receive data in JSON format. The JSON text is parsed, processed and printed into a HTML page, which is presented in the WebView. The Apps also load styling sheets from the *Subti*Wiki server, which allows us to change the style of gene pages on mobile Apps without updating the Apps themselves.

With the WebView, we can present content from other websites seamlessly integrated into our App. WebViews from both Android and iOS provide delegate methods allowing us to control links and redirects within the WebView. Thus, we used a specific URI format with the scheme name “subtiwiki” to indicate in-house content and of course URI with “http” as external source.

5 Discussion and outlook

5.1 Usage report of *Subti*Wiki

*Subti*Wiki is one the most dedicated online knowledge base for *Bacillus subtilis.* It has a great coverage of information about over 6000 genes, proteins and RNA features. Until now (27-09-2017), *Subti*Wiki has 1791 annotated operons, 2468 protein-protein interactions, 5802 gene regulations, and over 6838 included references. *Subti*Wiki is also frequently updated. Since the *Subti*Wiki v3 is online in June 2017, there has been 15482 edits.

*Subti*Wiki is popular among fellow researchers. Our internal traffic tracker logged 1189175 requests in a period of 96 days, from which 5284 requests come from mobile app access. The *Subti*Wiki App has 46 installations from Android users and 2551 downloads from App store.

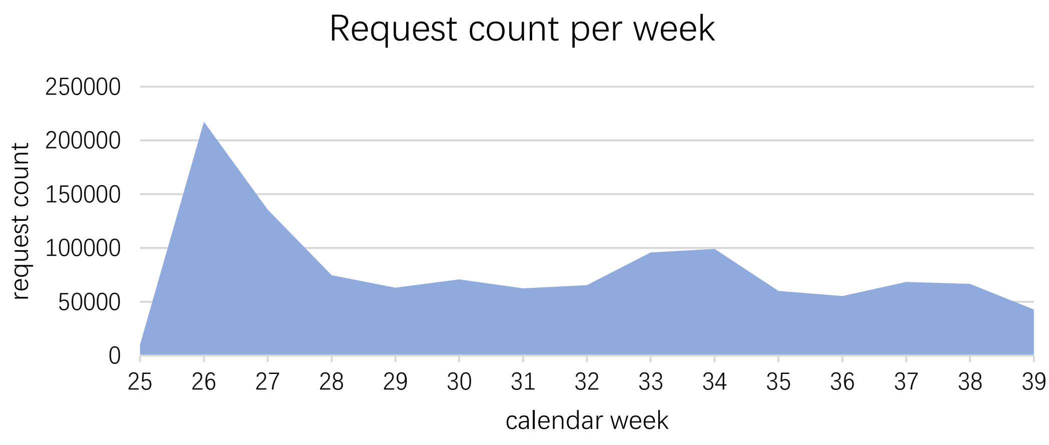


Figure 26. The requests of SubtiWiki per week. Data collected between 25th and 39th calendar week of year 2017

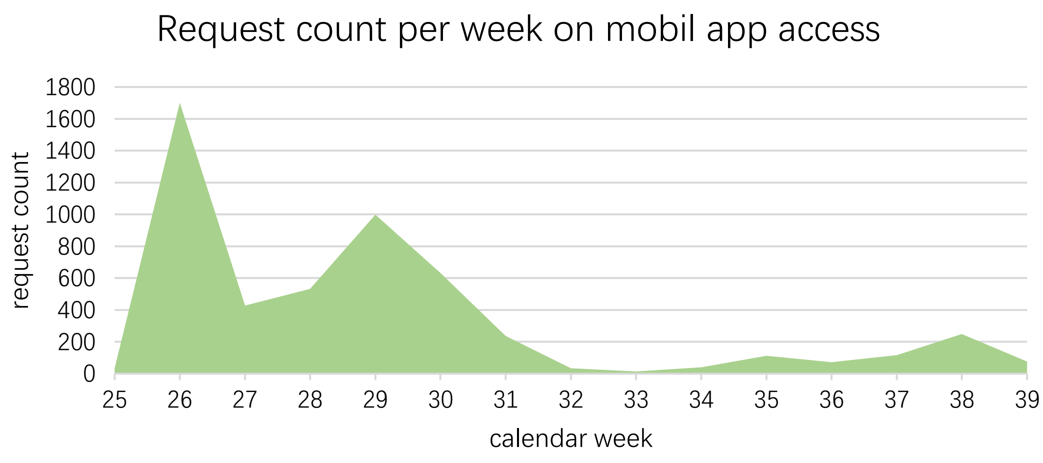


Figure 27. The requests of SubtiWiki originated from mobile apps per week. Data collected between 25th and 39th calendar week of year 2017.

5.2 Assessment of the new implementation

The construction of biological database applications is more challenging compared to applications in other fields. This is caused by the nature of the biological data, which is described and summarized in chapter 2.1. As a model organism database for *B. subtilis,* we are facing the same challenges. In this section, we will review all those characteristics and show how the new implementation addressed the requirements.

Concerning the biological database:

1. High complexity

The complexity of biological data requires the database to be able to present complex schemes at different levels. In the new implementation, we use JSON, a semi-structured data format, to tackle this problem. Semi-structured data are naturally powerful to present complicated nested structures which are difficult to store in relational databases. By using JSON, we also separated the platform and content. The task to define the scheme is shifted from database developers to content curators, who have a better understanding of the data.

2. High variety

The high variety of biological data requires the databases to be flexible with data types because there are always outlier values. The new implementation of *Subti*Wiki solves this issue simply by using only one data type, which is string. All values are treated as string unless they can be casted to numbers or Boolean values.

3. Fast evolution

The schemes of biological data change fast and the databases should be able to adapt to this fast evolution. The new implementation of *Subti*Wiki applies the semi-structured data, which do not have a fixed scheme. The change of the scheme of a single object does not require the change of the table scheme. This makes *Subti*Wiki easily extendable.

4. Multiple data interpretation and presentation

This is not applicable to *Subti*Wiki because this project is about migration of *Subti*Wiki from wiki to relational database.

5. Context is important

Biological data are in organization highly associated and information with context has more value. Hence, we focus on data integration in *Subti*Wiki. In the *Subti*-Apps, we enabled the function to integrate omics data to networks or pathways. We could further integrate all three types of biological networks, i.e. protein-protein interaction, gene regulation, and metabolic pathways, to present a full “life cycle” of elements of life.

Concerning biological database applications:

1. User accessibility

*Subti*Wiki always provides open access to all its data. A user account is needed only when a researcher would like to contribute.

2. Instructive interfaces

*Subti*Wiki is search centered and the search function is designed to be simple. For individual gene pages, the new implementation of *Subti*Wiki did not change the page layout. All information of a gene or RNA feature is presented on a single web page. The editing interface provides tool bars and full templates to help users with the new markup system.

3. Tools to build queries

*Subti*Wiki does not support complex queries. We compensate that with data export functions. The users can export the information from *Subti*Wiki in csv format and use Excel or other office software to filter out the information they need.

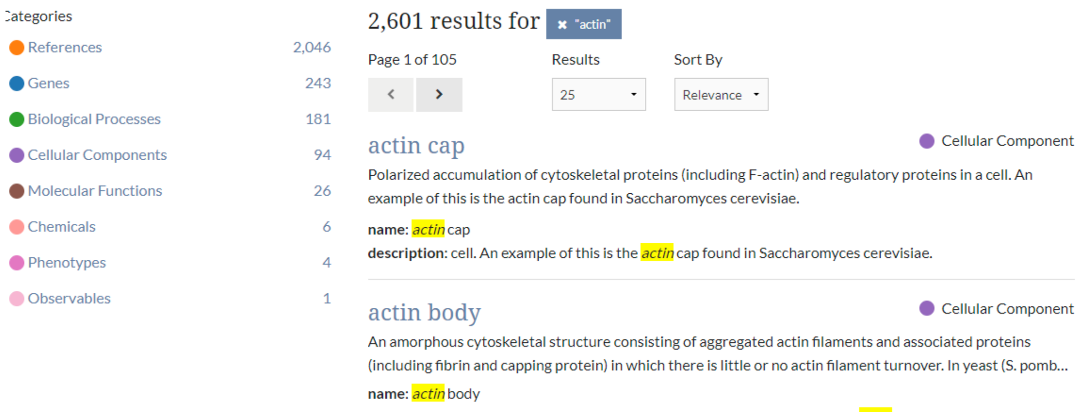
4. Version control

The new implementation also has a version control system. All historic versions are kept so each edit is traceable. The user can also select two versions to compare and differences are displayed according the key paths.

5.3 *Subti*Wiki compared with other databases

In this chapter we will look at other biological databases which also focus on a single organism and compare them with *Subti*Wiki. In the first part of this chapter, we will discuss a few databases of other microorganisms, i.e. yeast, *E. coli, Mycoplasma pneumoniae* and *Listeria monocytogenes*. In the second part of this chapter, we will have a closer look at all the databases which dedicate to *Bacillus subtilis.*

The Saccharomyces Genome Database (SGD) is the model organism database for baker's yeast [59]. It covers a wide spectrum of data, including manually curated annotations from published literature as well as high-throughput data. In SGD, experimental results gathered from various of sources are presented in a controlled language to ensure a consistent interpretation of information. In general, SGD is quite similar to *Subti*Wiki as both databases gather manually curated annotations. In *Subti*Wiki, we have a category system, which is equivalent to the gene ontologies. However, the search function of SGD is much more powerful than *Subti*Wiki. In *Subti*Wiki, the search function is gene-orientated, which means the users use gene names as keywords and get gene pages as results. In SGD, it appears to be ontology-oriented, which means the users can use functional descriptions as keywords, such as actin or kinase. The search results of the keyword "actin" covers different domains, such as genes, biological processes (ontology term), chemicals, phenotypes etc.


ategories

● References	2,046
● Genes	243
● Biological Processes	181
● Cellular Components	94
● Molecular Functions	26
● Chemicals	6
● Phenotypes	4
● Observables	1

2,601 results for "actin"

Page 1 of 105 · Results 25 · Sort By Relevance

actin cap — ● Cellular Component
Polarized accumulation of cytoskeletal proteins (including F-actin) and regulatory proteins in a cell. An example of this is the actin cap found in Saccharomyces cerevisiae.
name: *actin* cap
description: cell. An example of this is the *actin* cap found in Saccharomyces cerevisiae.

actin body — ● Cellular Component
An amorphous cytoskeletal structure consisting of aggregated actin filaments and associated proteins (including fibrin and capping protein) in which there is little or no actin filament turnover. In yeast (S. pomb...
name: *actin* body

Figure 28. Seach results of keyword "actin" in SGD. The search results cover different domains of information

The current database layout of *Subti*Wiki already supports such multi-domain searches. Enhancement of the search function can be a topic of future *Subti*Wiki development.

SGD also applies a modern web design and very clear data presentation. The contents of the web page are well compartmentalized, and examples and hints are given for search boxes or other user interface elements. We can see that a lot of efforts have been invested in design work, which is really rare for scientific websites.

EcoliWiki is subsystem of PortEco [60]. It is the community-based platform of functional annotation of *E. coli* K12 strain. It has a lot of similarities with *Subti*Wiki as both are community-based platforms and both are dedicated to a bacterial model organism. However, unlike *Subti*Wiki, which has always focused on the 168 strain of *B. subtilis*, EcoliWiki extended its functions and now covers other *E. coli* strains. Like other biological wikis, EcoliWiki is limited by its engine. Hence, the information presentation in EcoliWiki is limited to text and static images and Interactivity barely exists.

EcoCyc is another model organism database for *E. coli*. It is a part of BioCyc database collection like BsubCyc. All the BioCyc databases are established in the exact same way. Hence, we will only discuss about BsubCyc in the later part of this chapter.

MyMPN is the model organism database for *Mycoplasma pneumoniae*. It is established in 2014 and it is a relational database. It hosts very detailed information about the protein features as well as omics data. On each gene’s page, information about the gene, the protein product, and the transcriptional level is displayed in different sections. Detailed information about the homologous proteins from different organisms is also included. MyMPN also offers a pathway browser which is based on CellDesigner (Figure 29). The same software is used to generate pathways in *Subti*Wiki. Hence, both pathway browsers look extremely similar. However, in MyMPN all pathways are depicted in the same map. This is because the metabolic network in *M. pneumoniae* is much simpler than that in *B. subtilis*.

From the website, we can see that the database is quite well structured. However, this database has already stopped updating. It is a real pity as the database layout is still usable. This addresses the necessity for database developers to separate the content from the platform.

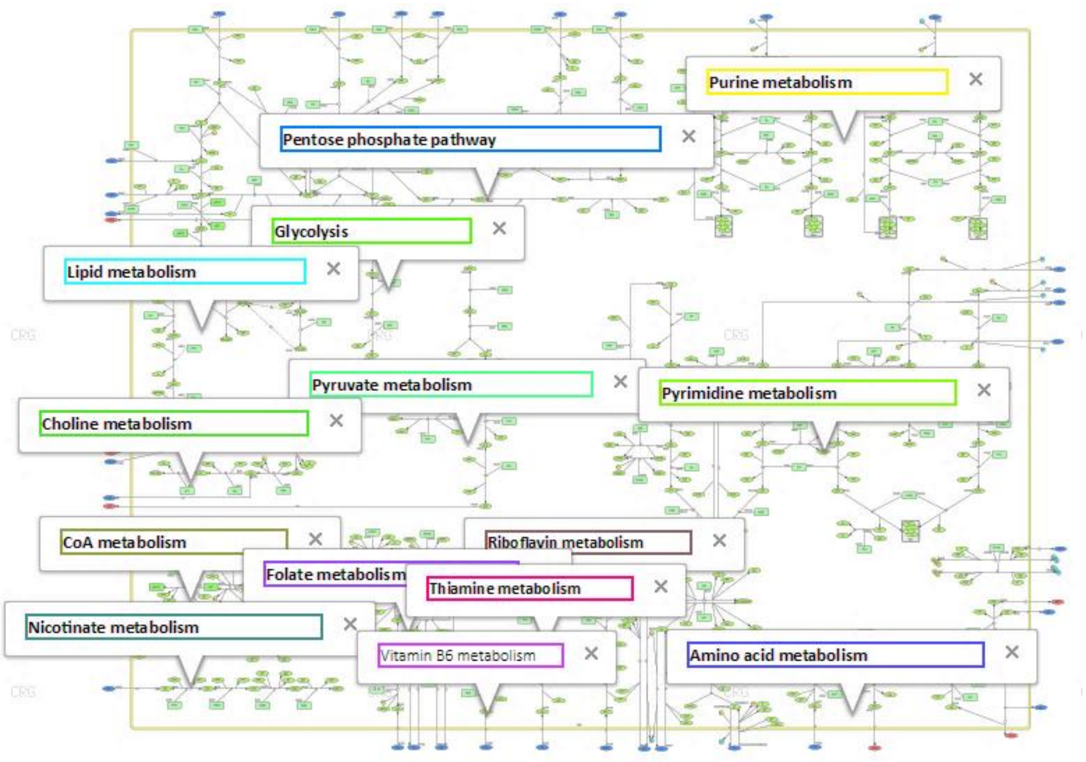
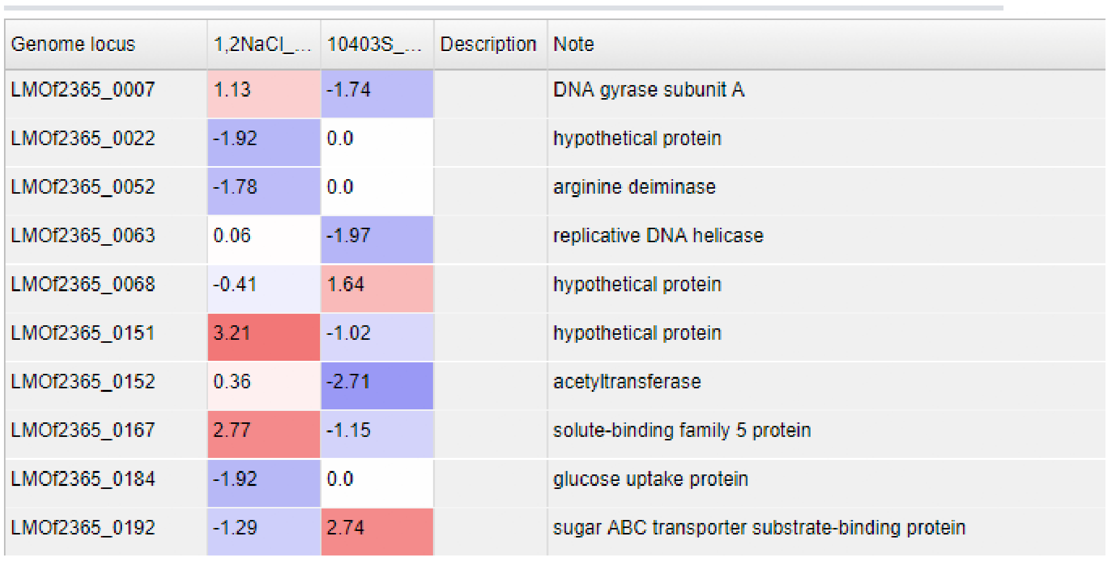


Figure 29. The overview of metabolic pathways in M. pneumoniae provided by MyMPN. This application is based on CellDesigner and Google Maps.

For researchers working on *Listeria monocytogenes*, there is a database named ListiList[14] dedicated to genome annotations of *L. monocytogenes*, which is a part of GenoList project like SubtiList. This database project was initiated to share genome sequencing data and has stopped updating since 1998.

There is also a model organism database named ListeriaBase [61]. This database offers detailed annotations of *L. monocytogenes* genome as well as the tools for comparative data analysis. This database was established in 2015. However, currently (13.11.2017), the database cannot be accessed.

Listeriomics [62] is a database focusing on the system biology of *L. monocytogenes.* It hosts basic genome annotations of *Listeria* species and allows users to export the data in a tabular format. It also includes transcriptomics and proteomics data. The users can select different datasets and view those data in a heat map. In the expression browser of *Subti*Wiki, we presented the expression level of gene or proteins in charts.



Genome locus	1,2NaCl_...	10403S_...	Description	Note
LMOf2365_0007	1.13	-1.74		DNA gyrase subunit A
LMOf2365_0022	-1.92	0.0		hypothetical protein
LMOf2365_0052	-1.78	0.0		arginine deiminase
LMOf2365_0063	0.06	-1.97		replicative DNA helicase
LMOf2365_0068	-0.41	1.64		hypothetical protein
LMOf2365_0151	3.21	-1.02		hypothetical protein
LMOf2365_0152	0.36	-2.71		acetyltransferase
LMOf2365_0167	2.77	-1.15		solute-binding family 5 protein
LMOf2365_0184	-1.92	0.0		glucose uptake protein
LMOf2365_0192	-1.29	2.74		sugar ABC transporter substrate-binding protein

Figure 30. The heap map view of two different transcriptomic datasets provided by Listeriomics.

In conclusion, compared with other databases which focus also on a single organism, *Subti*Wiki has quite a few advantages. It has a broad data coverage and focuses on the integration of data as well. It has also a modern web design and applies modern methods for data visualization.

*Subti*Wiki is not the only model organism database for *B. subtilis*. As mentioned in the introduction part (chapter 2.5), there are other databases such as BSORF, DBTBS, SubtiList and BsubCyc. In this chapter, we will also compare different MODs of *Bacillus subtilis* with *Subti*Wiki in three aspects, i.e. data coverage, data organization and data presentation.

1. Data coverage

BSORF was the earliest database for *Bacillus subtilis*. The initial motivation was to share the sequencing data from the genome sequencing project. It provides basic information on the gene and protein features. Information on protein-protein interactions and gene regulations is not included.

DBTBS, on the other hand, focuses on the information concerning the gene regulation, such as transcription factors, regulated operons, motifs etc. It does not cover detailed functional annotations of each gene.

SubtiList is very similar to BSORF. It also provides basic information on the gene and protein features. Information on the protein-protein interactions and the gene regulations is also not included.

*Subti*Wiki covers a much wider spectrum of information. It offers information on the gene, the RNA and the protein features. It also collects information on biological networks, such as the protein-protein interactions, the gene regulation, and metabolic pathways. A category system is included in *Subti*Wiki which is equivalent to the gene ontology in other databases. The *Subti*Wiki database also collects information on biological materials such as mutants, antibodies etc.

BsubCyc is the part of BioCyc database collections which focuses on *B. subtilis*. It also collects a great variety of data including information on the gene and the protein as well as gene regulations and metabolic pathways. In comparison to *Subti*Wiki, information on protein-protein interactions is missing there.

2. Data organization

For databases like BSORF, DBTBS, and SubtiList, the data they collect do not have a complicated structure. Hence, modelling and data organization are not big issues. However, for *Subti*Wiki and BsubCyc, the challenge of data complexity is real. In this chapter we will compare *Subti*Wiki and BsubCyc in the aspect of data organization.

In *Subti*Wiki, a gene, its transcribed RNA and its translated protein are abstracted as one entity *gene* based on the information flow which is depicted by central dogma. For BsubCyc, the gene, the RNA (transcription units), and the protein are modelled separately, based on the molecules in the cell. Hence, the BsubCyc database has a more complicated database structure which is shown in Figure 31. The model for *Subti*Wiki is more abstract and focused on the information flow from DNA to RNA to proteins. This model is suitable for our data and is not applicable to eukaryotic organisms. Because of alternate splicing, the relationships between DNA and the transcribed RNA and the translated proteins are not one-to-one but rather one-to-many.

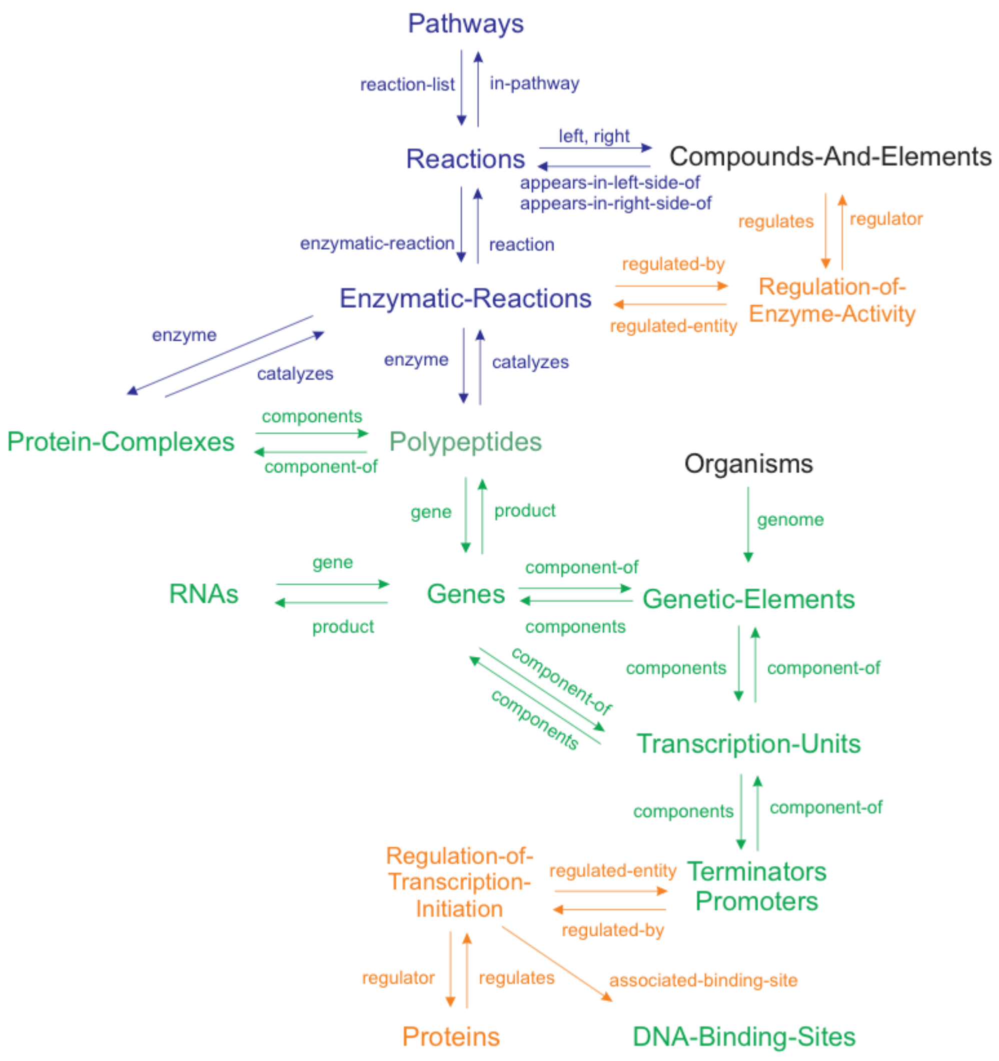


Figure 31. Pathway tool schema (version 15.0, Jan 2011). Image source: https://biocyc.org/schema.shtml. This schema is much more complicated than the schema of SubtiWiki.

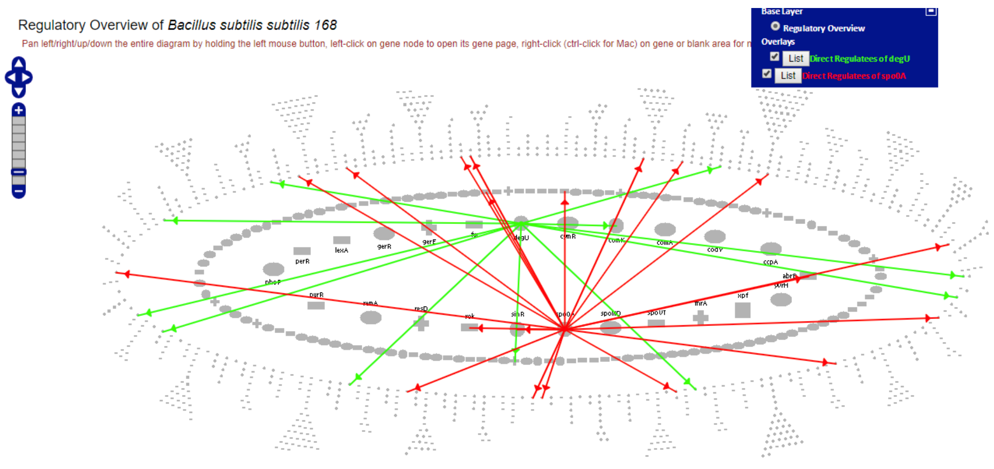
3. Data presentation

BSORF, DBTBS, and SubtiList were established in earlier times when there were only limited methods to present the data. Pages in these three databsases all have a tabular format. Information is presented in narrative text and static images are used to present diagrams.

*Subt*Wiki and BsubCyc were both established in the late 2000s. Both databases present the information of the gene, the RNA and the protein features on the same page. Both database embed diagrams in gene pages to visualize the connections between the genes. In addition, both databases provide different kinds of browsers to present overviews of biological networks.

a) Regulation browser

*Subti*Wiki offers a regulation browser which presents the regulatory networks around a gene at different levels. The regulation browser in *Subti*Wiki uses a force-directed graph drawing algorithm which present regulatory clusters well. Details of this regulation browser is stated in the results part (chapter 4.2.2.4).


Regulatory Overview of *Bacillus subtilis subtilis 168*

Pan left/right/up/down the entire diagram by holding the left mouse button, left-click on gene node to open its gene page, right-click (ctrl-click for Mac) on gene or blank area for

Base Layer
Regulatory Overview
Overlays
List Direct Regulatees of degU
List Direct Regulatees of spo0A

Figure 32. The screenshot of the regulation browser provided by BsubCyc. The regulations of two regulators are highlighted.

BsubCyc also provides a portal to view the regulatory connections between the genes with a completely different design. In this application, regulatory networks in *B. subtilis* is presented in a hierarchical structure. Genes and proteins are arranged in circles with top level regulators in the inner circle and secondary regulators in the outer circle.

The two applications are also built in different ways. The regulation browser in BsubCyc is based on static images and is probably curated manually. The regulation browser in *Subti*Wiki uses an automatic graph layout algorithm, which makes update easier.

b) Genome browser

The genome browser in BsubCyc is in general similar to the genome browser in *Subti*Wiki. The genome browser in BsubCyc is implemented with static images. Hence it provides less interactivity than our genome browser. On the other hand, more detailed information on operons, promoters and terminators is collected in BsubCyc and that information is depicted in its genome browser.

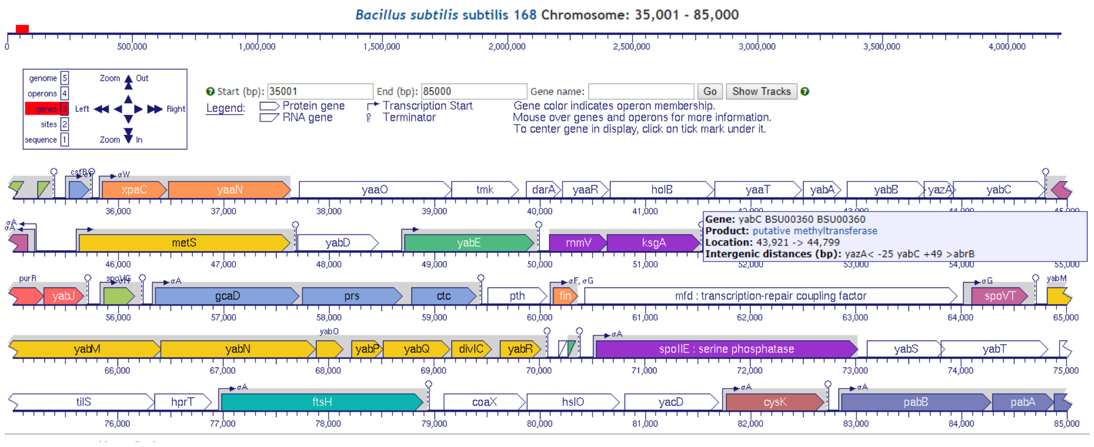

Bacillus subtilis subtilis 168 Chromosome: 35,001 - 85,000
Start (bp): 35001
End (bp): 85000
Gene name:
Go
Show Tracks
Legend: Protein gene, RNA gene, Transcription Start, Terminator
Gene color indicates operon membership.
Mouse over genes and operons for more information.
To center gene in display, click on tick mark under it.
Gene: yabC BSU00360 BSU00360
Product: putative methyltransferase
Location: 43,921 -> 44,799
Intergenic distances (bp): yazA< -25 yabC +49 >abrB

Figure 33. The genome browser of BsubCyc

c) Pathway browser

In *Subti*Wiki, an application is provided to present the metabolic pathways. BsubCyc provides an application to view the metabolic pathways in *B. subtilis* as well. In the pathway browser of BsubCyc, a large map of metabolic pathways is offered. The users can use the mouse wheel to zoom in and out for details or overview. However, it is not so well implemented. It is very lagging to zoom or move with the mouse and is sometimes not responding. In chapter 5.4, we will discuss more about the pathway browser in *Subti*Wiki and compare it with the pathway browser from KEGG[63].

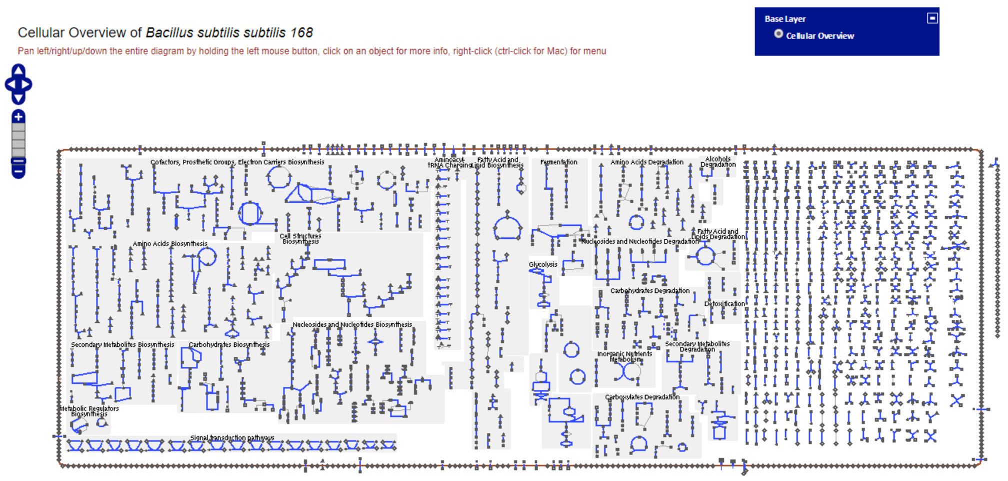

Cellular Overview of Bacillus subtilis subtilis 168
Pan left/right/up/down the entire diagram by holding the left mouse button, click on an object for more info, right-click (ctrl-click for Mac) for menu
Base Layer
Cellular Overview

Figure 34. The pathway browser provided by BsubCycs

In conclusion, both BsubCyc and *Subti*Wiki exist in the same niche and have quite a few functional overlaps, which make them direct competitors with each other. BsubCyc is a part of the BioCyc database collections, which allow the users to compare data in *B. subtilis* with other organisms. However, *Subti*Wiki, on the other hand, has a wider data coverage. In addition, the data visualization tools *Subti*Wiki offers are more interactive and have better performance. In early 2017, BsubCyc has changed its access policy. The users need a paid subscription to access the data in BsubCyc, which means that *Subti*Wiki is now the only free, up-to-date information source for *Bacillus* research community.

5.4 Presentation of metabolic pathways

In this project, we have migrated *Subti*Wiki to a relational database. This new database layout has integrated the category system, *Subti*Express and *Subt*Interact. However, due to limited time, the integration and update of *Subti*Pathways remains unfinished. It was established in 2010 and the implementation was not updated since then. There are three major issues with the current version:

First, pathways concerning different aspects of life are displayed separately in *Subti*Pathways, although it is an integrated system in the cell. The database KEGG [63] present the biochemical pathways in various levels. An overview is provided with links to more detailed pathway diagrams. Interconnections between pathways are also labelled as entry points to other pathway maps. It would be good to integrate all data from different pathways and create a multi-level presentation.

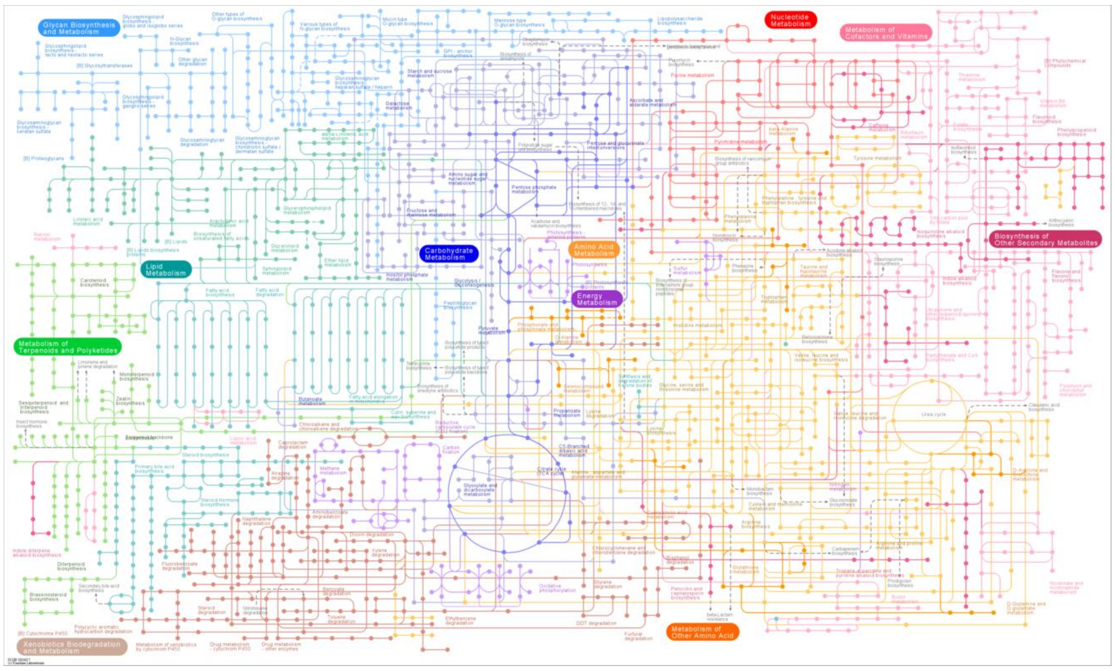

Glycan Biosynthesis and Metabolism
Nucleotide Metabolism
Metabolism of Cofactors and Vitamins
Carbohydrate Metabolism
Lipid Metabolism
Amino Acid Metabolism
Energy Metabolism
Biosynthesis of Other Secondary Metabolites
Metabolism of Terpenoids and Polyketides
Xenobiotics Biodegradation and Metabolism
Metabolism of Other Amino Acid

Figure 35. The pathway overview provided by KEGG.

Second, the process to update data in *Subti*Pathways is complicated. The maps are created with CellPublisher[64], then uploaded to *Subti*Wiki server. It would be much more convenient if a pathway editor is included in the current system and the user can see the results directly.

Third, the data for *Subti*Pathways are stored in XML files, which are not synchronized with the *Subti*Wiki database. Therefore, data should be imported into the current database so that *Subti*Pathways becomes a truly integrated part of *Subti*Wiki family.

As described in the results section, the tables to store pathway data have been created. A temporary JavaScript library has been developed to present single biochemical reactions in diagrams. What remains to be done is a JavaScript library to connect biochemical reactions to pathways and present the pathways in a proper format.

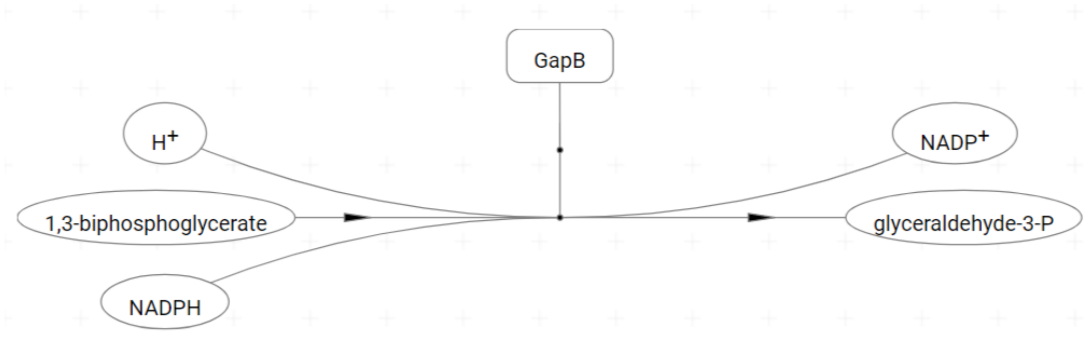


Figure 36. The diagram of a biochemical reaction generated by the JavaScript library based on SVG.

Consider the reactions as nodes and common metabolites between the reactions as edges, the full pathway can be modelled as a graph. Therefore, the presentation of pathway information in a living cell is another graph drawing problem. To present the whole pathway, we could use force-directed layout or orthogonal layout. The KEGG pathway databases applies the orthogonal layout. The pathway maps from KEGG are generated manually, which is a big work load. For *Subti*Wiki, it would a better solution to use an automatic layout algorithm to generate a rough presentation and manually adjust the unsatisfying part.

5.5 Multi-mode database as a possible solution

Like any software, *Subti*Wiki is affected by many factors which are not technological. As a non-profit scientific database, *Subti*Wiki needs a solution which is stable, mature and does not require an expensive license. This is one of the major reasons we decided to construct a relational database using MySQL. To handle the complexity of biological data, we applied the JSON format in our database. This gives the database features like document databases.

Because of the nature of biological data, it is challenging to build a database with a single approach mentioned in the introduction part. From the experience we have obtained migrating *Subti*Wiki to a relational database, we have concluded that a multi-mode database might be most suitable for biological data.

The data collected in *Subti*Wiki over the years can be used as a good example as they cover almost every aspect of the functional annotations. The entity gene in *Subti*Wiki has complex nested structure, which makes document model suitable. The associations among biological elements in the cell, like protein-protein interactions and gene regulations, can be modelled as graphs and should be stored in a graph database. The omics data are mostly already in a tabular format, which makes relational databases the best solutions.

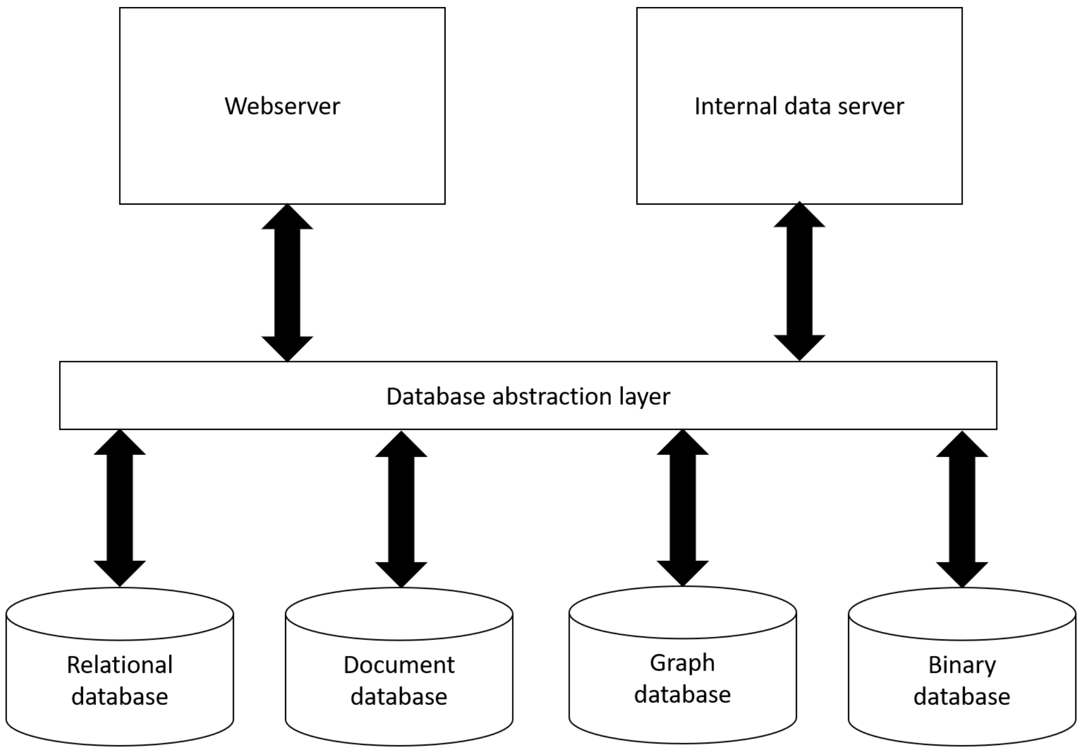


Figure 37. Implementation of multi-mode database with the help of database abstraction layer. This database abstraction layer will be responsible to coordinate database operation in different database management systems. This layer will isolate the applications from the detailed database layouts.

The multi-mode database can be implemented with two different approaches. One is to combine different database management systems with the help of a data abstraction layer to enable cross-database operations (Figure 37). This data abstraction layer is responsible for cross-database operations and keeping data integrate and consistent. This structure is illustrated in Figure 37. The current version of *Subti*Wiki is implemented in this way. The data access handlers serve as the database abstraction layer and they access data from the relational database and a flat file (genome sequence file).

Another approach is the application of a multi-mode DBMS. Since 2010, there have been quite a few multi-model DBMS developed, such as ArangoDB, CosmosDB, etc. Those DBMS software packages support document model, relational model, graph model and key-value pairs.

Name	**Initial release**	**Supported models**	**Special feature**	**Availability**
ArangoDB[65]	2011	document, graph, key-value		free open-source
CosmosDB[66]	2017	document, tabular, relational, key-value	cloud storage, distributive system	free for basic functions
CouchBase[67]	2010	document, key-value	distributive system	free open-source
CrateDB[68]	c.a. 2014	relational, document	distributive system	free open-source
Oracle Database[69]	1977 (incremental software update)	relational, document, graph, key-value, geospatial, binary	object-oriented relational database	free for basic functions

Table 16. Comparison of several multi-model DBMSs.

From the table above, we can see that the multi-mode DBMS is still quite fresh out of oven and needs to be tested by the market. For maintainability reasons, we did not apply such DBMSs. However, it could be an option in the future when the current structure can no longer meet the requirements.

5.6 Web-based biological data visualization

Data visualization is the process to present the data in a graphical form by encoding it as visual objects, such as dots, lines, bars, etc. The goal of data visualization is clear and effective communication of information through graphical means. The old saying, "a picture is worth a thousand words", states how efficient and communicative visual presentations are in comparison with plain text.

The beginning of this century witnessed the rapid development of the internet. It is no longer a luxury for a small fraction of people but rather an infrastructure for the modern society. More and more web technologies are introduced, such as scalable vector graph (SVG) and html canvas. Those technologies provide us new means for data visualization.

Before SVG and canvas is available, the data visualization on a web page is commonly done using static images or flash. For example, the DNA sequence database GenBank provides a graphical view of the DNA sequences with the annotations as shown in Figure 38.

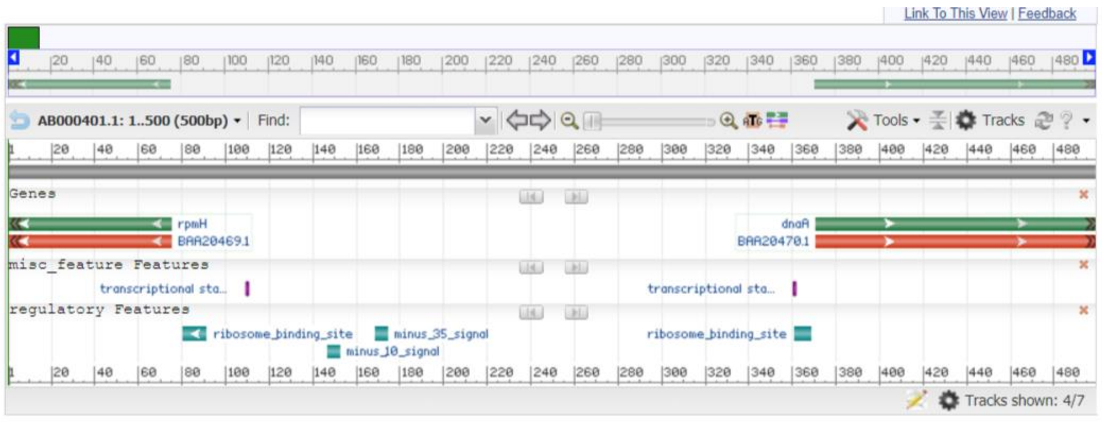
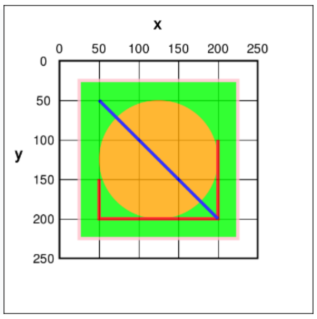


Figure 38. The graphical view of DNA sequences provided by GenBank

If we take a closer look at the source code of this web page, we will realize that the ruler and the bars of the diagram are static images. If we zoom in the web page, the images will blur. This is the disadvantage of static images. Scalable vector graphs, as the name suggests, are scalable. Those graphs can be scaled without losing resolution. They are also much smaller than the static images because they are basically text file of instructions of how to draw the images.

```
<?xml version="1.0" encoding="UTF-8" ?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect x="25" y="25" width="200" height="200" fill="lime" stroke-
width="4" stroke="pink" />
  <circle cx="125" cy="125" r="75" fill="orange" />
  <polyline points="50,150 50,200 200,200 200,100" stroke="red"
stroke-width="4" fill="none" />
  <line x1="50" y1="50" x2="200" y2="200" stroke="blue" stroke-
width="4" />
</svg>
```



Code block 14. A sample of a SVG file and the rendered image. Image and code from Wikipedia user Offnfopt

SVG is XML-based and can be integrated directly in the HTML file. It can be manipulated with JavaScript, providing interactivity to the presentation. In *Subti*Wiki, the genomic context viewer, protein-protein interaction diagram and the gene regulation diagram, which are embedded in each gene page is developed with SVG.

HTML canvas is another technology commonly used for data visualization. Canvas is a HTML element which allows graphs to be drawn inside. The drawing is done purely by JavaScript. HTML canvas is faster than SVG. Hence, it is more suitable for the situation when large numbers of visual objects need to be drawn. In *Subti*Wiki, the network browsers use canvas instead of SVG to draw the networks because of the large size of the networks.

Presenting data with SVG and HTML canvas is certainly a bit more complicated than with static images because it requires programming. However, those technologies are faster, which means the visualization application loads faster and runs smoother. They also provide more interactivity. Each object in the visualization can respond to user actions. The visualizations based on those technologies are data-driven, which means no extra effort is needed when some data change. The market itself has already proven how powerful those technologies are. They have completely replaced flash, a previous technology to present interactive interfaces.

5.7 A database implementation for other MODs

Since the release of the new implementation, we have been frequently asked whether the same system we developed can be applied to other organisms. In principle, the new database layout is specifically designed and optimized for our data in *Subti*Wiki. This system certainly can be applied to other organisms, which has similar data structure to *Bacillus subtilis.* For example, the whole system can be applied to *Mycoplasma* species without major adjustment because the data scheme is similar. However, for eukaryotic organisms, like *Arabidopsis thaliana,* more biological entities such as *chromosome, intron and extron* need to be added into the database layout. A secondary development is required to adjust the system to the data to be collected.

There is a general model organism database(GMOD) project started in 2000, which is a collection of useful tools to set up model organism databases. In this project, a core relational database implementation, named “Chado”, is provided. This database implementation was a generalization of FlyBase[70] database (MOD for fruit fly). It covers a broad variety of interesting biological concepts for a model organism, such as sequences, sequence comparisons, phenotypes, genotypes, ontologies, phylogeny etc.

The GMOD project has generated many fruitful results. Many biological databases participated in the project and some MODs are established with the software and tools from GMOD project. However, the Chado database implementation is not suitable for the data we have gathered. In *Subti*Wiki, data presents a complex nested structure which is difficult for pure relational database. A lot of effort would be requirement to extend the Chado database to fit our data. In addition, the Chado database has a complicated structure, which makes maintenance more difficult.

As described in chapter 2.1, the schema of biological database evolves fast. This is the same with database and web technologies. It might not be possible to develop a general solution once for all. Instead of creating a general model, it might be better to create individual models which are interchangeable with each other. It would also be better to keep a certain level of abstraction in those models. Database developers love precise models but in the real world there are always ambiguity. In addition, it is important to separate the platform from the content, like the wikis. This would make maintenance much easier. it will be also possible to hand over the database from one lab to another one so that the database which takes a lot of effort to build, can live longer.

6 References

1. Parasuraman, S. (2012) Protein data bank. *J. Pharmacol. Pharmacother.* **3,** 351

2. Codd, E. F. (1983) A relational model of data for large shared data banks. *Commun. ACM* **26,** 64–69

3. Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J. & Wheeler, D. L. (2005) GenBank. *Nucleic Acids Res.* **33,** D34–D38

4. Mashima, J., Kodama, Y., Kosuge, T., Fujisawa, T., Katayama, T., *et al.* (2016) DNA data bank of Japan (DDBJ) progress report. *Nucleic Acids Res.* **44,** D51–D57

5. Navathe, S. B. & Patil, U. (Springer Berlin Heidelberg, 2004) Genomic and proteomic databases and applications: a challenge for database technology. in *Database Systems for Advanced Applications: 9th International Conference, DASFAA 2004, Jeju Island, Korea, March 17-19, 2003. Proceedings,* (eds. Lee, Y., Li, J., Whang, K.-Y. & Lee, D.) 1–24 doi:10.1007/978-3-540-24571-1_1

6. Brohée, S., Barriot, R. & Moreau, Y. (2010) Biological knowledge bases using Wikis: combining the flexibility of Wikis with the structure of databases. *Bioinformatics* **26,** 2210–2211

7. Oliver, S. G., Lock, A., Harris, M. A., Nurse, P. & Wood, V. (2016) Model organism databases: essential resources that need the support of both funders and users. *BMC Biol.* **14,** 49

8. NCBI Resource Coordinators. (2017) Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.* **45,** D12–D17

9. Reuß, D. R., Commichau, F. M., Gundlach, J., Zhu, B. & Stülke, J. (2016) The Blueprint of a Minimal Cell: *MiniBacillus*. *Microbiol. Mol. Biol. Rev.* **80,** 955–987

10. Zweers, J. C., Barák, I., Becher, D., Driessen, A. J. M., Hecker, M., *et al.* (2008) Towards the development of *Bacillus subtilis* as a cell factory for membrane proteins and protein complexes. *Microb Cell Fact* **7,** 10

11. Kunst, F., Ogasawara, N., Moszer, I., Albertini, A. M., Alloni, G., *et al.* (1997) The complete genome sequence of the gram-positive bacterium *Bacillus subtilis*. *Nature* **390,** 249–256

12. Search on Bacillus subtilis. Available at: https://www.ncbi.nlm.nih.gov/pubmed/?term=bacillus+subtilis.

13. Ogiwara, A., Ogasawara, N., Watanabe, M. & Takagi, T. (1996) Construction of the *Bacillus subtilis* ORF database (BSORF DB). *Genome Informatics* **7,** 228–229

14. Moszer, I., Glaser, P. & Danchin, A. (1995) Subtilist: a relational database for the *Bacillus subtilis* genome. *Microbiology* **141,** 261–268

15. Ishii, T., Yoshida, K., Terai, G., Fujita, Y. & Nakai, K. (2001) DBTBS: a database of *Bacillus subtilis* promoters and transcription factors. *Nucleic Acids Res.* **29,** 278–280

16. Flórez, L. A., Roppel, S. F., Schmeisky, A. G., Lammers, C. R. & Stülke, J. (2009) A community-curated consensual annotation that is continuously updated: The Bacillus subtilis centred wiki SubtiWiki. *Database* **2009,** bap012

17. Lammers, C. R., Flórez, L. A., Schmeisky, A. G., Roppel, S. F., Mäder, U., *et al.* (2010) Connecting parts with processes: *Subti*Wiki and *Subti*Pathways integrate gene and pathway annotation for *Bacillus subtilis*. *Microbiology* **156,** 849–859

18. Nicolas, P., Mäder, U., Dervyn, E., Rochat, T., Leduc, A., *et al.* (2012) Condition-dependent transcriptome reveals high-level regulatory architecture in Bacillus subtilis. *Science* **335,** 1103–1106

19. Maaβ, S., Wachlin, G., Bernhardt, J., Eymann, C., Fromion, V., *et al.* (2014) Highly precise quantification of protein molecules per cell during stress and starvation responses in Bacillus subtilis. *Mol. Cell. Proteomics* **13,** 2260–2276

20. Maass, S., Sievers, S., Zühlke, D., Kuzinski, J., Sappa, P. K., *et al.* (2011) Efficient, global-scale quantification of absolute protein amounts by integration of targeted mass spectrometry and two-dimensional gel-based proteomics. *Anal. Chem.* **83,** 2677–2684

21. Michna, R. H., Commichau, F. M., Tödter, D., Zschiedrich, C. P. & Stülke, J. (2014) *Subti*Wiki-a database for the model organism *Bacillus subtilis* that links pathway, interaction and expression information. *Nucleic Acids Res.* **42,** D692–D698

22. LAMP (software bundle). Available at: https://en.wikipedia.org/wiki/LAMP_(software_bundle).

23. Debian/Ubuntu extend the dominance in the Linux web server market at the expense of Red Hat/CentOS. Available at: https://w3techs.com/blog/entry/debian_ubuntu_extend_the_dominance_in_the_linux_web_server_market_at_the_expense_of_red_hat_centos. (Accessed: 11th October 2017)

24. Welcome! - The Apache HTTP Server Project. Available at: https://httpd.apache.org/. (Accessed: 11th October 2017)

25. Module Index - Apache HTTP Server Version 2.4. Available at: https://httpd.apache.org/docs/2.4/en/mod/. (Accessed: 11th October 2017)

26. MySQL. Available at: https://www.mysql.com/. (Accessed: 11th October 2017)

27. Gulutzan, P. & Pelzer, T. (1999) *SQL-99 Complete, Really*.

28. PHP: Hypertext Preprocessor. Available at: http://php.net/. (Accessed: 11th October 2017)

29. mod_rewrite - Apache HTTP Server Version 2.4. Available at: http://httpd.apache.org/docs/current/mod/mod_rewrite.html. (Accessed: 11th October 2017)

30. PHP: Introduction - Manual. Available at: http://php.net/manual/en/language.types.intro.php. (Accessed: 11th October 2017)

31. PHP: Arrays - Manual. Available at: http://php.net/manual/en/language.types.array.php. (Accessed: 11th October 2017)

32. PHP: Objects - Manual. Available at: http://php.net/manual/en/language.types.object.php. (Accessed: 11th October 2017)

33. PHP: Classes and Objects - Manual. Available at: http://php.net/language.oop5. (Accessed: 11th October 2017)

34. PHP: Callbacks / Callables - Manual. Available at: http://php.net/manual/en/language.types.callable.php. (Accessed: 11th October 2017)

35. HTML 5.1 2nd Edition. Available at: https://www.w3.org/TR/html/. (Accessed: 11th October 2017)

36. ECMAScript® 2017 Language Specification (ECMA-262, 8th edition, June 2017). Available at: http://www.ecma-international.org/ecma-262/8.0/index.html.

37. JSON. Available at: http://www.json.org/index.html. (Accessed: 11th October 2017)

38. A Relational Database Overview (The Java^{TM} Tutorials JDBC(TM) Database Access JDBC Introduction). Available at: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html. (Accessed: 11th October 2017)

39. Chen, P. P.-S. (1976) The entity-relationship model---toward a unified view of data. *ACM Trans. Database Syst.* **1,** 9–36

40. Tarawneh, R. M., Keller, P. & Ebert, A. (2011) A general introduction to graph visualization techniques. *Proc. IRTG 1131 - Vis. Large Unstructured Data Sets Work.* 151–164 doi:10.4230/OASIcs.VLUDS.2011.151

41. Network | Les miserables. Available at: http://visjs.org/examples/network/exampleApplications/lesMiserables.html. (Accessed: 11th October 2017)

42. Brightwell, G. R. & Scheinerman, E. R. (1993) Representations of Planar Graphs. *SIAM J. Discret. Math.* **6,** 214–229

43. Erdős, P., Goodman, A. W. & Pósa, L. (1966) The representation of a graph by set intersections. *Can. J. Math.* **18,** 106–112

44. Michna, R. H., Zhu, B., Mäder, U. & Stülke, J. (2016) *Subti*Wiki 2.0 - an integrated database for the model organism *Bacillus subtilis*. *Nucleic Acids Res.* **44,** 654–662

45. Eades, P. (1984) A Heuristic for Graph Drawing. *Congr. Numer.* **42,** 149–160

46. Fruchterman, T. M. J. & Reingold, E. M. (1991) Graph drawing by force-directed placement. *Softw. Pract. Exp.* **21,** 1129–1164

47. Kamada, T. & Kawai, S. (1989) An algorithm for drawing general undirected graphs. *Inf. Process. Lett.* **31,** 7–15

48. Hadany, R. & Harel, D. (2001) A multi-scale algorithm for drawing graphs nicely. *Discret. Appl. Math.* **113,** 3–21

49. Walshaw, C. (2003) A Multilevel Algorithm for Force-Directed Graph-Drawing. *J. Graph Algorithms Appl.* **7,** 253–285

50. Quigley, A. & Eades, P. (Springer Berlin Heidelberg, 2001) FADE: Graph Drawing, Clustering, and Visual Abstraction. in *Graph Drawing: 8th International Symposium, GD 2000 Colonial Williamsburg, VA, USA, September 20--23, 2000 Proceedings* (ed. Marks, J.) 197–210 doi:10.1007/3-540-44541-2_19

51. Barnes, J. & Hut, P. (1986) A hierarchical O(N log N) force-calculation algorithm. *Nature* **324,** 446–449

52. SQLite Home Page. Available at: https://www.sqlite.org/. (Accessed: 11th October 2017)

53. Application Fundamentals | Android Developers. Available at: https://developer.android.com/guide/components/fundamentals.html. (Accessed: 11th October 2017)

54. IDC: Smartphone OS Market Share. Available at: https://www.idc.com/promo/smartphone-market-share/os. (Accessed: 11th October 2017)

55. Activities | Android Developers. Available at: https://developer.android.com/guide/components/activities/index.html. (Accessed: 11th October 2017)

56. Fragments | Android Developers. Available at: https://developer.android.com/guide/components/fragments.html. (Accessed: 11th October 2017)

57. Processes and Threads | Android Developers. Available at: https://developer.android.com/guide/components/processes-and-threads.html. (Accessed: 11th October 2017)

58. Dijkstra, E. W. (1959) A Note on Two Problems in Connexion with Graphs. *Numer. Math.* **1,** 269–271

59. Cherry, J. M., Hong, E. L., Amundsen, C., Balakrishnan, R., Binkley, G., *et al.* (2012) Saccharomyces Genome Database: the genomics resource of budding yeast. *Nucleic Acids Res.* **40,** D700-5

60. Hu, J. C., Sherlock, G., Siegele, D. A., Aleksander, S. A., Ball, C. A., *et al.* (2014) PortEco: a resource for exploring bacterial biology through high-throughput data and analysis tools. *Nucleic Acids Res.* **42,** D677–D684

61. Tan, M. F., Siow, C. C., Dutta, A., Mutha, N. V. R., Wee, W. Y., *et al.* (2015) Development of ListeriaBase and comparative analysis of *Listeria monocytogenes*. *BMC Genomics* **16,** 755

62. Bécavin, C., Koutero, M., Tchitchek, N., Cerutti, F., Lechat, P., *et al.* (2017) Listeriomics: an Interactive Web Platform for Systems Biology of *Listeria*. *mSystems* **2,**

63. Ogata, H., Goto, S., Sato, K., Fujibuchi, W., Bono, H., *et al.* (1999) KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Research* **27,** 29–34

64. Flórez, L. A., Lammers, C. R., Michna, R. & Stülke, J. (2010) Cellpublisher: A web platform for the intuitive visualization and sharing of metabolic, signalling and regulatory pathways. *Bioinformatics* **26,** 2997–2999

65. ArangoDB - highly available multi-model NoSQL database. Available at: https://arangodb.com/. (Accessed: 3rd November 2017)

66. Azure Cosmos DB – Globally Distributed Database Service | Microsoft Azure. Available at: https://azure.microsoft.com/en-us/services/cosmos-db/. (Accessed: 3rd November 2017)

67. NoSQL Engagement Database | Couchbase. Available at: https://www.couchbase.com/. (Accessed: 3rd November 2017)

68. CrateDB - Put machine data to work. Scalable, open source SQL database. Available at: https://crate.io/. (Accessed: 3rd November 2017)

69. Database 12c | Oracle. Available at: https://www.oracle.com/database/index.html. (Accessed: 3rd November 2017)

70. Attrill, H., Falls, K., Goodman, J. L., Millburn, G. H., Antonazzo, G., *et al.* (2016) FlyBase: establishing a Gene Group resource for *Drosophila melanogaster*. *Nucleic Acids Res* **44,** D786–D792

7 Appendix

7.1 Table schemes

Table Category

Column	Data type	Extra
id	varchar(255)	primary key
title	varchar(255)	
data	text	
lastUpdate	timestamp	on update current timestamp
lastAuthor	varchar(255)	by default “ghost”
count	int	

Table Chemical

Column	Data type	Extra
id	int(11)	primary key, auto_increment
title	varchar(255)	unique
synonym	varchar(255)	
pubchem	int(11)	

Table Complex

Column	Data Type	Extra
id	char(40)	primary key
title	varchar(255)	unique

Table Condition

Column	Data type	Extra
id	int(6)	not null

title	varchar(50)	not null, primary key
description	text	
type	enum('T', 'P')	
short	varchar(255)	

Table Config

Column	**Data type**	**Extra**
name	varchar(255)	primary key
value	varchar(255)	

Table Gene

Column	**Data type**	**Extra**
id	char(40)	primar key
title	varchar(255)	
data	mediumtext	
_locus	varchar(50)	index
_function	text	
_names	text	
_mw	double	index
_pI	double	
_description	text	
_essential	varchar(10)	
_ec	varchar(30)	
_geneLength	int(11)	
_proteinLength	int(11)	
bank_id	int(11)	index
count	int(11)	not null

lastUpdate	timestamp	on update current timestamp
lastAuthor	varchar(255)	default “ghost”

Table GeneCategory

Column	**Data type**	**Extra**
id	int(11)	primary key, auto increment
gene	char(40)	foreign key to Gene.id unique(gene,category)
category	varchar(255)	foreign key to Category.id unique(gene,category)
lastAuthor	varchar(255)	
lastUpdate	timestamp	current time stamp

Table GenomicContext

Column	**Data type**	**Extra**
start	int(11)	
stop	int(11)	
object	varchar(255)	
strand	int(1)	

Table History

Column	**Data type**	**Extra**
version	char(16)	primary key
id	varchar(255)	index
data	longtext	
user	varchar(255)	
lastOperation	varchar(255)	

time	timestamp	current timestamp

Table Interaction

Column	**Data type**	**Extra**
id	int(11)	primary key, auto increment
prot1	varchar(255)	unique(prot1, prot2)
prot1	varchar(255)	unique(prot1, prot2)
data	text	
lastUpdate	timestamp	on update current timestamp
lastAuthor	varchar(255)	

Table MaterialViewGeneRegulation

Column	**Data type**	**Extra**
gene	char(40)	
regulation	int(11)	

Table Metabolite

Column	**Data type**	**Extra**
name	varchar(255)	
id	int(11)	
syn	varchar(255)	

Table Operon

Column	**Data type**	**Extra**
id	char(40)	primary key
title	text	
data	mediumtext	

_genes	text	
lastUpdate	timestamp	on update current timestamp
lastAuthor	varchar(255)	
count	int(11)	

Table Pathway

Column	Data type	Extra
id	int(3)	primary key, auto increment
title	varchar(255)	
markers	blob	

Table PathwayGene

Column	Data type	Extra
pathway	int(11)	
gene	char(40)	

Table PathwayMetabolite

Column	Data type	Extra
pathway	int(11)	
metabolite	int(11)	

Table ProteinComplex

Column	Data type	Extra
id	char(40)	primary key
title	varchar(255)	
_categories	varchar(255)	
_Complex_members	text	

lastUpdate	timestamp	on update current timestamp
lastAuthor	varchar(255)	

Table proteomicData

Column	**Data type**	**Extra**
id	char(40)	unique
bsu	varchar(100)	
con1 … con16	int(11)	

Table Pubmed

Column	**Data type**	**Extra**
id	int(11)	primary key
report	text	

Table Reaction

Column	**Data type**	**Extra**
id	int(11)	primary key, auto increment
pathway	int(11)	
reversible	int(1)	default yes

Table Reaction_chemical

Column	**Data type**	**Extra**
id	int(11)	primary key, auto increment
reaction	int(11)	
chemical	int(11)	
side	enum(‘L’, ‘R’)	
isMain	int(1)	

Table Reaction_enzyme

Column	Data type	Extra
id	int(11)	primary key, auto increment, not null
reaction	int(11)	not null
enzyme	varchar(255)	not null
modification	varchar(45)	
position	varchar(255)	

Table Regulation

Column	Data type	Extra
id	int(11)	not null, primary key, auto increment
regulator	varchar(255)	unique(regulator, regulated)
regulated	varchar(255)	unique(regulator, regulated)
mode	varchar(255)	not null
description	text	
lastAuthor	varchar(255)	default “ghost”
lastUpdate	timestamp	current timestamp

Table Regulon

Column	Data type	Extra
id	varchar(255)	primary key, not null
title	varchar(255)	
data	text	
lastUpdate	timestamp	current timestamp
lastAuthor	varchar(255)	default “ghost”
count	int(11)	

Table Sequence

Column	Data type	Extra
gene	char(40)	primary key
dna	mediumtext	
aminos	mediumtext	

Table Statistics

Column	Data type	Extra
id	varchar(255)	
count	int(11)	

Table TranscriptomicData

Column	Data type	Extra
id	char(40)	unique
con1…con105	decimal(5,3)	

View ViewGeneOperon

Column	Data type	Extra
operon	char(40)	
gene	char(40)	

The table user is a copy of the MediaWiki user table. Details please see MediaWiki document here: https://www.mediawiki.org/wiki/Manual:User_table .

7.2 Directory structure

Directory	Description
/app	directory of all applications
/css	directory of all CSS files
/html	directory of all HTML templates
/imgs	directory of all images
/js	directory of all JavaScript files
/res	directory of resource files, such as genome sequence file.
/src	directory of source files of the framework
.htaccess	.htaccess file, contains URL redirect rules, might need to be updated when apache is updated
favicon.ico	icon file for the website
index.php	the entry page of *Subti*Wiki.
index_mobile.php	this file is currently not in use
Config.php	a configuration of the whole website, contains information for database connections

7.3 Default data access methods

In this section we will describe the default data access handlers defined for *Subti*Wiki. Those methods are defined in /app/default.php and overridden the methods defined in /src/default.php.

1. **Default::updateCount($id)**

Updates the count column of a table row by 1. The id of the given row need to be given

2. **Default::update($id, $data)**

Updates a table row with $data, which is identified by $id. Here $id should be an array or an object. This update method generates a history record. If update fails, this method returns false and an error message is written in debug log.

3. **Default::remove($id)**

Removes a table row, which is identified by $id. Here $id should be an array or an object. This remove method generates a history record. If deletion fails, this method returns false and an error message is written in debug log

4. **Default::insert($data)**

Inserts a row to the table with the given data. This method generates a history record. If insertion fails, this method returns false and an error message is logged.

5. **Default::list($page, $pagesize)**

Lists the records of the table in pages. The table must have id, title, lastUpdate, lastAuthor columns. This method is currently used.

6. **Default::saveToHistory($id, $operation)**

Generates and inserts a history record. The operation here can be add, update, or remove.

7. **Default::naturalJoin($resultSet)**

Process raw result sets from the database. Any object markup in the format of {object type|object id} is replaced with the object itself.

Bingyao Zhu

Ernst-Fahlbusch-Str. 30

37077, Göttingen, Germany

lucil_zby@hotmail.com

PERSONAL	**Date of Birth**: August 26th, 1991
	Place of Birth: Jiangsu, China
	Citizenship: Chinese

EDUCATION

PhD study in GGNB program Microbiology and chemisty

November 2015 – November 2017

Thesis: *Subti*Wiki v3.0: A relational database for functional genome annotation of the model organism *Bacillus subtilis*

Master study in the program Microbiology and chemisty at University of Göttingen

October 2012 – November 2014

Thesis: A document oriented approach to organize genes' annotation in model organism *Bacillus subtilis.*

Bachelor study in the honor program (Life science) at China Agriculture University

September 2008 – July 2012

Thesis: A study of *in vitro* phosphorlational regulation of MAP18

College Entrance Examination in Jiangsu

June 2008

SKILLS

Foreign language - Fluent in English, intermediate German

Computer Skills – Programming skills in C and Java. Experience of front-end and back-end development.