University of Stuttgart
Germany

BioCatNet

a platform for the systematic analysis of enzyme sequence - structure - function relationships

A diploma thesis presented to the faculty in
partial fulfillment of the requirements
for the degree Diplom-Biologist (t.o.) in

Technische Biologie (Dipl.)

by

Waldemar Reusch

Examiners: Prof. Dr. Jürgen Pleiss
Prof. Dr. Bernhard Hauer
Supervisor: Dipl. Biol. (t.o.) Constantin Vogel

Institute of Technical Biochemistry
Prof. Dr. Bernhard Hauer

December 10, 2014

Abstract

Throughout the last decades, catalytically active proteins have greatly gained importance in the synthetic chemistry industry. Compared to traditional metallo- and organocatalysis, enzymes provide several key benefits: chemo- and stereoselectivity, low reaction temperatures and an immensely reduced need for toxic chemicals make biocatalysts an increasingly attractive alternative, both in the laboratory and on industrial scale. Key advances in protein sequencing, manipulation and expression are at the base of protein engineering, the tremendous process of tailoring proteins to fit them into new biosynthetic pathways ranging from the production of commodity chemicals to advanced pharmaceutical intermediates.

The increasing amount of research of course produces an increased amount of data, which in turn requires the involvement of bioinformatics to collect, store, process and redistribute the acquired information. A large number of tools are available online for this purpose, one type of tool being databases. In the last decade the Institute of Technical Biochemistry (ITB) has released several databases containing sequence and structure information, aiding in the improvement of established biocatalysts as well as the engineering of novel ones. These family-specific protein databases (FSPDs) were backed by the Data Warehouse system for protein Families (DWARF) system, developed by Markus Fischer at the ITB. To keep track with the steadily growing sequence space, expand the functionality and to update the user interface to current standards, the system recently has been replaced by a performance-optimized platform called BioCatNet.

The present thesis describes the authors contribution to the platform, prominently the creation of user and application interfaces to interact with the underlying databases and tools. The new platform has been build modular to allow easier extension and is already surpassing the previously used system in terms of functionality and usability. While sequence and structure information will be collected and processed automatically by tools, manual curation by experts is needed to ensure a high quality repository. Functional information will be inserted by bench scientists directly, using the user interface, avoiding the high effort and mediocre quality that comes with literature mining. A set of clearly structured and interactive forms guide collaborators through the process, ensuring high quality, consistent data, complying to, and partially exceeding, current standards.

We expect this new platform to be a solid, extensible and comfortable foundation for future FSPDs to provide scientists a high quality repository for information about sequence relationships, experimentally determined and computationally inferred structures, experimentally confirmed functions as well as critical parameters.

Table of Contents

List of Figures

List of Listings

Glossary

AJAX (asynchronous JavaScript and XML) is a set of techniques used in web applications to fetch and present additional data without reloading or leaving the current page 33

BLAST (Basic Local Alignment Search Tool) is an algorithm for comparing primary biological sequence information such as amino-acid and DNA sequences 4

CSS (Cascading Style Sheets) is a styling language used to describe the look and formatting of documents written in markup languages such as HTML 10

data mining is the computational process of discovering and extracting information from a data set and to transform it into an understandable structure for further use 25

DBMS (Database Management System) is a software designed to provide an interface for databases to interact with users and other programs. Moreover, it is critical in maintaining the integrity and consistency of the database 5

EC (Enzyme Commission) number is a numerical classification scheme for enzymes, based on the chemical reactions they catalyze 39

Entrez PubMed is a database of references and abstracts on life sciences and biomedical topics hosted by the NCBI 6

FASTA is a DNA and protein sequence alignment software as well its now ubiquitous plaintext file format for representing nucleotide and amino acid sequences 4

Firebird is a relational DBMS, and the DBMS of choice at the ITB 29

GenBank is a DNA sequence database provided by the NCBI in collaboration with the EMBL and DDBJ 6

Acronyms

API application programming interface 21

BRENDA Braunschweig Enzyme Database 6

CGI Common Gateway Interface 11

CRUD create, read, update, delete 13

DWARF Data Warehouse system for protein Families I

FSPD family-specific protein database I

GUI graphical user interface 21

HMM hidden Markov Model 30

HTML HyperText Markup Language 9

HTTP Hypertext Transfer Protocol 9

IRED Imine Reductase Engineering Database 37

ITB Institute of Technical Biochemistry I

KEGG Kyoto Encyclopedia of Genes and Genomes 58

MSA multiple sequence alignment 4

MVC model-view-controller 21

NCBI National Center for Biotechnology Information 6

NMR nuclear magnetic resonance 6

OOP object-oriented programming 14

PDB Protein Data Bank 5

1 Motivation

The development of novel biocatalysts and their application in large-scale processes is impeded on various levels nowadays. Bench scientists struggle with experiments failing to reproduce findings another group has discovered. Process Engineers struggle with set-up and scale-up of mass-production processes which have been shown to work flawlessly at the bench. One big problem underlying these issues is the lack of data. To be precise, the lack of high-quality and high-quantity data on the behavior of biocatalysts under certain conditions.

While the amount of effort and investment needed to investigate a protein sequence has dropped rapidly over the last years, the costs for the experimental characterization of protein function has stayed rather constant, leading to a widening gap between the numbers of known protein sequences and known protein functions. Additionally, publications on experimental characterization are often supported only by a few figures and tables to get their point across. Even if supplementary material is provided, only figures and tables directly contributing to the results are being attached. The larger part of collected raw data, whether in lab books or on hard disks, ends up in archives, never to be touched again.

This lack of uniform high-quality data also impedes comparability severely. Because individual research groups focus on different aspects of their research subject, similar experimental setups and findings might result in very different publications, making the comparison cumbersome. Even though a vast amount of research has already been conducted around the enzymatic activity of proteins, it is a Sisyphean task to find publications relevant to someone's special field of interest and extract data. Moreover, due to the often missing link between the processed data and the amino acid sequence of the applied biocatalysts, reproduction of experiments is hindered.

It is hard for scientists and engineers to formulate hypotheses based on scarce and divergent data. Much effort needs to be invested only to (re-)produce data, which may be lying buried in another labs archive. While providing extensive supplementary material with the publication – or better still, providing raw data online – may increase the amount of available data, the issue of comparability remains.

BioCatNet aims to alleviate this issue. By capturing and validating user-provided data it will hold uniform, comprehensive and high-quality data describing protein sources and relationships, kinetic and environmental parameters as well as substrate and product specificities. BioCatNet aims to be a platform for collecting, standardizing, analyzing and sharing biochemical information about catalytic proteins. It will include

information about catalytic activities, substrate specificities, product yields and distributions, environmental conditions and kinetics as well as information about protein structure and features and similarities of their amino acid sequences.

As manual entry of large amounts of primary data promises to be a cumbersome task, the key focus of the development will be the user. By providing an intuitive and easy to use interface, the time and effort needed to submit biochemical data to BioCatNet should be reduced to a minimum. Clearly structured and pleasant to look at, the user interface will facilitate repetitive tasks and provide a simple and short workflow to enter experimental data. To further motivate bench scientist to provide data, BioCatNet will present a means to store research data - publicly or confidential - and return it as well-formatted, standardized lab-reports.

2 Introduction

Biocatalysis is the application of enzymes and microbes in synthetic chemistry. Even though fermentation processes have been commonplace for millennia, the first accounts of selective applications of cell extracts on non-natural man-made organic compounds are only a century old. [11] Since then, enzymes have been gaining attention in synthetic chemistry because of their chemo-, regio- and enantioselectivity. The first challenge when working with biocatalysts is limited protein stability, nowadays primarily overcome by immobilization. [24] Since 1980, protein engineering techniques have been used to make biocatalysts work outside their original substrate range, often even on non-natural substrates. [11] To use protein engineering techniques more effectively, researchers in this field often use bioinformatics tools.

2.1 Bioinformatics

Bioinformatics is an interdisciplinary field that blends computer sciences and statistics with biomedical sciences. It emerged shortly after high-throughput DNA sequencing methods in the 1970s and gained importance ever since. One main task of this new field is the management, analysis and interpretation of biological experiments. Amongst other tools, scientists in this field make use of databases and apply statistical methods. [44]

2.1.1 Sequence alignment

To identify similarities between DNA, RNA, or amino acid sequences, bioinformaticians use sequence alignments. The aligned sequences are typically presented as rows within a matrix, with highlights on equal or similar sections. These alignments can then be used to infer functional, structural or evolutionary relationships.

For pairwise alignment - *i.e.* alignment of two sequences - one can choose between *global* alignments which attempt to align every residue and *local* alignments which focus to achieve high similarities in smaller fragments. Global alignments are best suited for sequences of similar length and dissimilar sequences tend to have better alignments with local alignment methods. Though pairwise alignments can only be applied between two sequences, they are efficient to calculate and find their use when searching for a sequence in a large sequence database, for example.

Multiple sequence alignment (MSA) are an extension of pairwise alignments and operates on more than two sequences at a time. MSA methods are often used to identify conserved regions of amino acids or nucleotides across a group of related sequences. Such motifs can be used to infer the position of active sites, or equally outstanding regions, in proteins. [45]

FASTA is a sequence alignment software package first designed for protein sequence similarity searches, but now supporting protein:protein, DNA:DNA and translated DNA:protein searches as well. Applying heuristic methods, it achieves considerable speed improvements compared to its stricter predecessors. [41]

BASIC LOCAL ALIGNMENT SEARCH TOOL (BLAST) is an algorithm for comparing amino acid and nucleotide sequences. It enables the user to search for sequences similar to the input sequence in a large library of sequences. Today, it is the most widely used tool for sequence searching and outperforms the older *FASTA* sequence search tools. This performance gain stems from the use of looser heuristic algorithms and comes at the cost of accuracy, so that it cannot "guarantee the optimal alignments of the query and database sequences". [1]

Standard numbering schemes for families of homologous proteins allow for the unambiguous identification of functionally and structurally relevant residues, to communicate results on mutations, and to systematically analyze sequence-function relationships in protein families. For this, a reference profile is created from a set of representative protein sequences. The subsequent pairwise alignment with a query sequence will yield *standard amino acid positions* for the query where key positions receive the same position number for every sufficiently closely related query. [70, 29]

2.1.2 Data formats

FASTA is a plaintext file format for representing nucleotide and amino acid sequences. The format originates from the FASTA sequence alignment software package but has since become a standard used widely across various bioinformatics software. One file can contain multiple distinct sequences, separated by sequence descriptions and comments, which are denoted by a line starting with the *greater-than* (>) symbol. Descriptions usually also contain a cross reference to an entry in a larger sequence database. An example is presented in Listing 2.1.

Listing 2.1: Example of an amino acid sequence in FASTA notation

```
>gi|31563518|ref|NP_852610.1| microtubule-associated proteins 1A/1B
MKMRFFSSPCGKAAVDPADRCKEVQQIRDQHPSKIPVIIERYKGEKQLPVLDKTKFLVPDHVNMSELV
KIIRRRLQLNPTQAFFLLVNQHSMVSVSTPIADIYEQEKDEDGFLYMVYASQETFGF
```

The PDB file format is representing the three dimensional structure of macromolecules in plaintext. Originally conceived at the Protein Data Bank (PDB), this file format is also used by various bioinformatics tools including homologous modelling and docking software. Though it has been developed to capture the structure of macromolecules such as proteins and DNA/RNA, it is also capable of storing structure information for small molecules. Indeed, protein structures saved in the PDB format are often interspersed with small molecules like water, ions and ligands. A PDB file consists of various sections of space-separated property and value tables, where the first column identifies the section type. An excerpt of an PDB file is presented in Listing 2.2.

Listing 2.2: Excerpt of an PDB file describing the structure of a synthetic collagen-like peptide, available under the accession code 1A3I. [39]

```
HEADER    EXTRACELLULAR MATRIX                      22-JAN-98   1A3I
TITLE     X-RAY CRYSTALLOGRAPHIC DETERMINATION OF A COLLAGEN-LIKE
TITLE     2 PEPTIDE WITH THE REPEATING SEQUENCE (PRO-PRO-GLY)
...
EXPDTA    X-RAY DIFFRACTION
AUTHOR    R.Z.KRAMER,L.VITAGLIANO,J.BELLA,R.BERISIO,L.MAZZARELLA,
AUTHOR   2 B.BRODSKY,A.ZAGARI,H.M.BERMAN
...
REMARK 350 BIOMOLECULE: 1
REMARK 350 APPLY THE FOLLOWING TO CHAINS: A, B, C
REMARK 350   BIOMT1   1  1.000000  0.000000  0.000000        0.00000
REMARK 350   BIOMT2   1  0.000000  1.000000  0.000000        0.00000
...
SEQRES   1 A    9  PRO PRO GLY PRO PRO GLY PRO PRO GLY
SEQRES   1 B    6  PRO PRO GLY PRO PRO GLY
SEQRES   1 C    6  PRO PRO GLY PRO PRO GLY
...
ATOM      1  N   PRO A   1       8.316  21.206  21.530  1.00 17.44 N
ATOM      2  CA  PRO A   1       7.608  20.729  20.336  1.00 17.44 C
ATOM      3  C   PRO A   1       8.487  20.707  19.092  1.00 17.44 C
ATOM      4  O   PRO A   1       9.466  21.457  19.005  1.00 17.44 O
ATOM      5  CB  PRO A   1       6.460  21.723  20.211  1.00 22.26 C
...
HETATM  130  C   ACY   401       3.682  22.541  11.236  1.00 21.19 C
HETATM  131  O   ACY   401       2.807  23.097  10.553  1.00 21.19 O
HETATM  132  OXT ACY   401       4.306  23.101  12.291  1.00 21.19 O
```

2.2 Databases

A database is a collection of information that is organized so that it can easily be accessed, managed and updated. In computing, databases are classified according to their organizational approach. The most prevalent approach is the relational database though object and graph databases have gained attention in recent years. [20, 58]

A *Database Management System (DBMS)* is software specially designed to interact with users, other applications and the database itself to capture and analyze data.

The database standards *Open Database Connectivity (ODBC)* and *Structured Query Language (SQL)* provide the means for the communication.

Today, complete and up-to-date databases are vital for biotechnical research. They range from diverse sequence repositories with little manual intervention to expertly curated specialized databases, and the amount of information they encompass has been growing exponentially in the last few decades. [75]

2.2.1 NCBI

The National Center for Biotechnology Information (NCBI) provides a comprehensive website for biologists including databases and tools centered around genomic and molecular data. As a division of the National Library of Medicine at the National Institutes of Health, the NCBI is conducting research on fundamental biomedical problems and developing standards for data deposition and biological nomenclature as well as developing, distributing and supporting a variety of databases and software for the scientific and medical communities. NCBIs most used services encompass *GenBank*, *Entrez PubMed* and *BLAST*. [49, 50]

2.2.2 BRENDA

The Braunschweig Enzyme Database (BRENDA) enzyme information system is the main collection of enzyme functional and property data, the majority of which has been extracted manually and by text mining from primary literature. Since 1987, its database covers structural and functional annotations as well as information about occurrence, preparation and application of enzymes and engineered variants. [60]

Though BRENDA has collected a more than 11,000 kinetic values for over 63,000 enzymes, [36] systematic analysis and comparison is hindered by the fact that BRENDA was not designed to provide an explicit connection between a protein (or a mutant variant) and kinetic data. Proteins are referenced by sometimes ambiguous names, and a precise amino acid sequence for the mentioned protein is hard to find, even in referenced primary literature.

2.2.3 PDB

The PDB is one of the largest archives of three-dimensional structural data of biological macromolecules. [6] Established in 1972, today it harbors more than 10,500 structures determined by X-ray crystallography, nuclear magnetic resonance (NMR) methods, cryoelectron microscopy and theoretical modelling. Every week approximately 50 structures are being added. The PDB file format, described in subsection 2.2.3 on page 6, is a widely used representation for macromolecular structures.

2.3 Database systems at the ITB - BioCatNet background

The basic idea behind BioCatNet is not a new one. For more than 10 years the bioinformatics group at the ITB has been publishing family-specific protein databases (FSPDs). Build atop the DWARF, these databases provide information about protein sequences and kinship.

2.3.1 DWARF

The Data Warehouse system for protein Families (DWARF) integrates data on sequence, structure and functional annotation for protein families. Tools for extracting and transforming data from public resources are provided to populate databases, though a lot of manual expert curation is also applied.

The basis for DWARF is a relational data-model encompassing data entities for protein sequence, family hierarchies, three-dimensional structure and sequence annotations. Amino-acid sequences and annotations are extracted from *GenBank*, structure information is extracted from PDB. Family hierarchies are subsequently established with clustering tools and by manual curation. The data is publicly accessible, ordered by available structures, source organisms or by family hierarchies. Numerous databases based on DWARF have been publishd, focusing on lipases, [76, 26, 53] cytochrome P450 oxidases, [63, 28] medium-chain dehydrogenase/reductases, [37] PHA depolymerases, [38] lactamases, [66] thiamine diphosphate dependent enzymes, [78] laccases, [64] triterpene cyclases [55] and metallo-β-lactamases. [77]

2.3.2 BioCatNet

The BioCatNet has been conceptualized by Constantin Vogel out of the need to overcome some of the shortcomings of DWARF. Though it is capable of storing functional sequence annotation, e.g. the position of active or cofactor-binding sites, DWARF is not designed to harbor functional parameters like substrate specificity or conversion rates. In order to enable systematic analyses of the relationships between sequences, structures, and functions of biocatalysts, a FSPDs has to encompass information on all three aspects and link them unambiguosly.

In collaboration with members of the FOR1296, Constantin Vogel revised the underlying data model to allow for the inclusion of detailed functional description using raw experiment data. This would allow the unambiguous linkage of a particular amino acid sequence to experiments, including experiment setups and results. Unfortunately, the DWARF could not be easily extended to hold the new data model, which led to the conception of an new software platform for the generation, maintenance and presentation of future FSPDs, the BioCatNet.

Accompanying the work on the user interface, application back end and data model described in the present thesis, bioinformaticians at the ITB are working on supporting modules.

DBParse is an automated pipeline for the initial population of FSPDs. [68] Given a set of *seed* sequences, the tool performs a search for similar proteins in publicly available protein databases and clusters the search results subsequently based on sequence similarity. Sequence annotations and structure information are extracted in the process, too. The resulting database must then be curated manually to establish higher-order hierarchies and weed out false-positives.

DBUpdate is developed to add proteins to established databases. [21] Every month, more than 250,000 sequences are being added to the *GenBank*, [31] making this update tool absolutely necessary, if the BioCatNet wants to keep up with current developments.

DBModel is a novel automated homologous modelling pipeline, so far applied exclusively to the Thiamine diphosphate-dependent Enzyme Engineering Database (TEED). Using the latest homologous modelling tools, information about sequence similarities and structure information provided by the PDB, several homologous structure models are build and evaluated for every sequence missing an experimentally determined structure.

2.4 Web-Application development

Code conventions are important to programmers for a number of reasons: 40%-80% of the lifetime cost of a piece of software goes to maintenance. [30] Hardly any software is maintained for its whole life by the original author. Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly. If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

2.4.1 Version control systems

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

Many people's version-control method of choice is to copy files into another directory, perhaps a time-stamped directory. This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.

To deal with this issue, programmers long ago developed local version control system (VCS) that had a simple database that kept all the changes to files under revision control. Soon after, Centralized VCS were developed to allow for collaboration. Nowadays, Distributed VCS gained much in popularity, *git* in particular. [16]

When publishing software, it is advised to set version numbers according to the *semantic versioning* convention, meaning that version numbers are in the format of `MAJOR.MINOR.PATCH`. `PATCH` number is increased for bug-fix releases, `MINOR` number is increased for releases including new features. Any release breaking backwards-compatibility must increase the `MAJOR` number. [54]

2.4.2 Code exchange and packages

When programming, one should always try to write DRY code. DRY stands for "Don't repeat yourself" and the principle is that there should only ever be *one copy* of any important piece of information. This improves efficiency as well as accuracy, as there is only one place to change the information and there is no copy which can be forgotten and get out of date. The way to achieve this is writing modular code: refactor often used routines into functions, collect functions often used together in modules.

Similarly, one should avoid trying to re-invent the wheel when programming. Most problems one encounters in everyday life have been tackled and most probably been solved before. The same is true for programming. For every programming language there are uncountable modules available on the internet, sometimes even well structured repositories and tools to ease the download, installation and use of modules. For *Perl* this is the *Central Perl Archive Network* (CPAN), for *PHP* the repository is called *Packagist* and *Composer* is the accompanying command-line tool. [52, 19, 18]

2.4.3 Scripting, styling, and markup languages

Build with current web-application development methods in mind, BioCatNet is using a variety of scripting, styling and markup languages.

HTML

Markup languages describe documents and data semantically. In the case of Extensible Markup Language (XML) and HyperText Markup Language (HTML) this is done using special language constructs, called *tags*. HTML is the standard language used to describe web documents today. [33] A web browser reads HTML content from a local file or an Hypertext Transfer Protocol (HTTP) server response and interprets it to construct an visual document. HTML allows images and tables to be embedded and structures documents semantically into headings, paragraphs, lists and many more distinct blocks.

Originally, HTML documents were static files read and served by a web server. Nowadays, many web documents are constructed dynamically from different static and dynamic content fragments on the server. An example of a simple HTML document is given in Listing 2.3.

Listing 2.3: Minimal example of an HTML document

```
1  <!DOCTYPE html>
2  <html>
3      <body>
4          <h1>Hello World</h1>
5          <p>Lorem ipsum ...</p>
6      </body>
7  </html>
```

CSS

Cascading Style Sheets (CSS) is used to describe the look and formatting of documents written in markup languages. It is primarily designed to enable the separation from document content and semantics from the document presentation, including elements such as layouts, colors and fonts. [73] It also allows for conditional formatting depending on the rendering method, such as screens or print-outs. An example of an simple *CSS* document is given in Listing 2.4.

Listing 2.4: Minimal example of an CSS style declaration

```
1  /* Style declarations consist of an element selector followed by
2   * the style description enclosed in curly braces. The description
3   *  is a set of property-value pairs, separated by a semicolon.
4   */
5  h1 {                        /* select all h1 elements (headings)*/
6    font-weight: bold;        /* Make every element's text bold */
7    color: blue;              /* Color every element's text blue */
8  }
9  p {                         /* select all p elements (paragraphs)*/
10   margin-top: 1em;          /* Add a margin above every element */
11 }
```

JavaScript

JavaScript is an interpreted dynamic programming language. Originally conceived to enable user interactions in web-browsers, in recent years it is gaining popularity as a general-purpose scripting language capable of driving HTTP servers and databases. JavaScript supports functional, object-oriented and imperative programming paradigms, though its prototypal approach to object inheritance deviates strongly from the approach purely object-oriented languages chose. [23] An example of an simple *JavaScript* program is given in Listing 2.5.

In the browser environment, JavaScript is used to manipulate the document, load additional information, interactive content like games, videos and audio elements, form validation and many more tasks.

Listing 2.5: Minimal example of an JavaScript function

```
/* define a function, which takes one argument */
function sayHelloTo(name) {
  /* Declare a new variable 'greeter' */
  var greeter;
  /* Create a new HTML text node with the contents
   * 'Hello ' followed by the argument. Assign this
   * node to the variable 'greeter'
   */
  greeter = document.createTextNode('Hello ' + name);
  /* append the new element to the HTML body element */
  document.body.appendChild(greeter);
}

sayHelloTo('World');
```

Perl

Perl is a family of interpreted general-purpose scripting languages including Perl5 and Perl6. The languages borrow features from other programming languages including *C*, *shell scripting*, *AWK* and *sed*, and provide powerful text processing facilities. It gained widespread popularity as a Common Gateway Interface (CGI) scripting language, processing HTTP requests and generating web content. Aditionally, Perl is used for graphics, administration, network programming and is popular in the bioinformatics community. [3, 5] Especially the *Bioperl* library has established itself as a standard tool for bioinformaticians.

PHP

Originally developed to build dynamic homepages and interpret form submissions, *PHP* has evolved to a sophisticated general-purpose language with included command-line capabilities. Like Perl, which *PHP* borrows heavily from, it is a scripting language, meaning it must be run by an interpreter, usually implemented as a web-server module. [72, 74]

Though *PHP* can be mixed with HTML, for large web-applications this practice is rather discouraged, as it often results in convoluted code with mixed concerns. Instead, various templating options exists to achive separation of presentation and business logic. An simple example of *PHP* code embedded in HTML is presented in Listing 2.6, more elaborate examples can be found in section 2.4.4 on page 16.

Listing 2.6: Example of PHP code embedded in HTML

```
<html>
    <body>
        <h1>Greetings</h1>
        <?php
          $name = 'Bob';
          echo "<p>Hello $name</p>";
        ?>
    </body>
</html>
```

Mustache

Mustache is a so-called *logic-less* templating language. Templating languages facilitate the separation of business and presentation logic of applications. Logic-less templating engines even more so than their counterparts, allowing only the simplest of conditionals and iterations. Though this may seem like a handicap, it actually enables easy and fast rendering engines to be implemented in every programming language. *Mustache* has been chosen because of its adjacency to HTML and the availability of *PHP* and JavaScript rendering engines.

Source code 2.7 gives an example of how *Mustache* is being used in *PHP*. *Mustache* encloses its templating logic in double curly braces (hence the name, Mustache).

Listing 2.7: Example of an Mustache template (greetings.html) and how it is rendered (index.php).

```
/* greetings.html */
<html>
    <body>
        <h1>Greetings</h1>
        <p>Hello {{name}}!</p>
        <ul>
             {{#favoriteColors}}
             <li>{{color}}</li>
             {{/favoriteColors}}
        </ul>
    </body>
</html>

/* index.php */

/* Load the template string. */
$template = file_get_contents('greetings.html');
/* Create a context to be renderd. */
```

```
$context = ['name'=> 'Bob',
            'favoriteColors' => [['color'=>'red'],
                                 ['color'=>'blue']]];
/* Use the Mustache rendering engine to create
   an HTML string */
$html = Mustache::render($template, $context);
print($html);

/* output */
<html>
    <body>
        <h1>Greetings</h1>
        <p>Hello Bob!</p>
        <ul>
             <li>red</li>
             <li>blue</li>
        </ul>
    </body>
</html>
```

SQL

SQL is a special-purpose programming language designed for managing data in relational database management systems. It is used for creating, reading, updating and deleting data (create, read, update, delete (CRUD)). Additionally, it is used for schema creation, schema modification and access control. A very simple example of an *SQL* query is presented in Listing 2.8.

Listing 2.8: Minimal example of an SQL expression

```
SELECT *                  /* select all columns */
  FROM Books              /* in the table `Books` */
  WHERE price > 100.00    /* where the value in the column */
                          /* `price` is larger than 100.00 */
  ORDER BY title;         /* and order the results by the */
                          /* values of column `title` */
```

Markdown

Markdown is a plain text formatting syntax, designed to be easily understood without prior experience of markup languages. It is popularly used in readme files and online discussion forums. [42] Its syntax is easily translated into HTML, making it an ideal format to create simple pages without dealing with HTML markup. An simple Markdown document is found in Listing 2.9.

Listing 2.9: Example of an Markdown Document

```
Heading
=======

Sub-Heading
-----------

Paragraphs are separated by a blank line. Text can be
formatted *italic*, **bold** and `monospace`
```

2.4.4 Object-oriented programming

Object-oriented programming (OOP) is a program design philosophy, which evolved as the logical extension of long established practices like structured programming. It is an approach to design modular, reusable software systems. Rather than structure programs as data and functions, object-oriented systems integrate the two using *objects* which carry a state - the data - and methods to act upon this state - the functions. These objects are used to interact with each other to design computer programs.

Some programming languages like JAVA and Objective-C only support OOP. Lisp and Haskell, on the other hand, allow only a functional programming style. The programming languages used to build the BioCatNet, *PHP*, Perl and JavaScript, all support object-oriented as well as functional programming styles. For the programming of website-back ends in *PHP*, the object-oriented model-view-controller pattern, outlined in subsection 2.4.5 on page 20, has established itself as the *de-facto* standard, though.

Encapsulation is a key feature of object oriented programming, used to section off responsibility of handling data to well-defined code modules. While an object may have an complex state (meaning it holds complex data), usually, only part of its state is exposed through its behaviors (methods).

Classes are blueprints or templates to build a specific type of object. The objects `your_car` and `my_car`, for example, would both be of `class Car`, sharing common methods like `function startEngine()` and `function stopEngine()` while having different states of `color` or `power`. A simple example of class declaration and usage in the programming language *PHP* is presented in Listing 2.10.

Class Inheritance allows for code reuse. Let's say `your_car` is of `class Truck` while `my_car` is of `class Van`. Instead of describing the behavior `function startEngine()` in both classes, we can define a new `class Vehicle`. Both descending classes `class Truck inherits Vehicle` and `class Van inherits Vehicle` will be able to use their parent's methods. Any inherited method or property can be overridden in the class definition.

SOLID is an mnemonic acronym that stands for the five basic principles of object-oriented programming and design:

Single responsibility principle says that any class should have only one responsibility.

Open/closed principle states that software entities as classes and objects should be open for extension, but closed for modification.

Liskov substitution principle declares that objects in a program should be replaceable with instances of their descendants without altering the correctness of that program.

Interface segregation principle says that many client-specific interfaces are better than one general-purpose interface.

Dependency inversion principle states that one should depend upon abstractions instead of concretions. In practice this means that functions and methods should be designed to only read/write to their containing object or passed parameters. In object-oriented languages this is achieved via the *dependency injection* pattern, where operands are explicitly passed to a function instead of depending on globally defined variables. An example of how such a pattern is realized and the accompanying anti-pattern are presented in Listing 2.10. In functional programming a similar paradigm exists, advising the use of *pure functions*, which only ever read/write to passed parameters.

Listing 2.10: Example of dependency injection in PHP.

```
class Person {
  public $name;
  public function __construct($name){ $this->name = $name; }
};

/* Example of an greeting function depending on an concrete
 * implementation, $bob.  This is an anti-pattern violating
 *  the dependency inversion principle
 */
$bob = new Person('Bob');
function sayHelloToBob(){
    print('Hello' . $bob->name);
}
sayHelloToBob(); // 'Hello Bob'

/* Example of an greeting function depending on an abstract,
 * where the concrete implementation is passed to the
 * function explicitly. This is the correct pattern for
 * dependency injection.
 */
function sayHelloToPerson(Person $person){
    print('Hello' . $person->name);
```

```
}
sayHelloTo($bob); // 'Hello Bob'
```

Object-oriented PHP

Having a *class-oriented* approach to objects, *PHP* has some advanced object-oriented programming features.

Visibility Which properties and methods of an object are exposed is described in the class definition by the use of visibility keywords. For *PHP*, these keywords are `public`, `private` and `protected`. If no visibility is specified, `public` is assumed by the Interpreter.

Interfaces are a means for unrelated objects to communicate with each other, a set of methods, arguments and return values which the objects agree upon in order to cooperate. The implementation is up to the class' definition.

Typehinting forces the parameters of a function or method to be of the given class or interface, an array or an function.

As a scripting language, *PHP* is dynamically typed. Thus, errors resulting from using wrong parameter types are only discovered at runtime. This in turn means that every function has to be properly tested. An elaborate example of how to work with classes in *PHP* is presented in Listing 2.11.

Listing 2.11: Class inheritance in PHP

```
 /* a class definition starts with the keyword `class`
  * followed by the class name (capitalized by
  * convention) and the class definition in braces
  */
class Vehicle
{
    /* a method definition starts with a keyword stating the
     * visibility and the keyword `function`, followed by
     * the methods' arguments in parenthesis and the
     * function body in braces. Class properties also start
     * with a visibility keyword
     */
    public $num_wheels = 4;
    public function startEngine() {...}
    public function stopEngine() {...}
}

/* new instances are spawned using the `new` keyword.
 *  instance properties and methods are accessed
 *  with the arrow `->`.
```

```
 */

$my_car = new Vehicle();
print($my_car->num_wheels); // prints '4'
$my_car->startEngine();

/* PHP's inheritance is indicated by the `extend` keyword */

class Van extends Vehicle {}

/* The next class `Truck` will define an additional method
 * that is not known to `Vehicle` or `Van`
 */

class Truck extends Vehicle
{
    private $cargo = array();
    // this overrides the inherited value
    public $num_wheels = 6;
    public function lowerCargoBed() {...}
}

$my_car = new Van();
$your_car = new Truck();

$your_car->startEngine();      // works fine
$my_car->startEngine();        // works fine
$your_car->lowerCargoBed();    // works just as fine
$my_car->lowerCargoBed();      // will raise an error because
                               // the method is undefined
print($your_car->num_wheels); // prints '6'
print($your_car->cargo);       // will raise an error because
                               // `cargo` is private

/* The definition of an interface starts with the keyword
 * `interface` followed by the interface name. An interface
 * method defines the method name and the number, name and
 * type of arguments. No function body is given, because
 * the implementation is up to the class definition. */

interface Electric
{
    public function chargeAt(ChargingStation $station);
}

/* the implement keyword indicates the use of an interface */

class EBike extends Vehicle implements Electric
```

```
{
    public $num_wheels = 2;
    public function chargeAt(ChargingStation $station){...}
}

/* this definition will raise an error, because it is not
 *  fulfilling the contract defined by the interface
 */

class ECar extends Vehicle implements Electric
{
    public function chargeAt(Tree $station){...}
}

$my_bike     = new EBike();

/* the `instanceof` keyword checks if an object is of the
   given
 * class or implements the given interface.
 */
if ($my_bike instanceof Electric) {
    print('true');
}
// prints true
if ($my_bike instanceof Vehicle) {
    print('true');
}
// prints true

$home        = new ChargingStation();
$tree        = new Tree();

$my_bike->chargeAt($home); // works fine
$my_bike->chargeAt($tree); // will not work
```

Overloading provides means to dynamically create class properties and methods. Overloading methods are invoked when undefined or unaccessible properties or methods are interacted with. [51] An example of this pattern is presented in Listing 2.12.

Object-oriented JavaScript

Being a weakly and dynamically typed language, and following a prototypal approach to object inheritance, *JavaScript* object operations may seem very unusual to a programmer. *JavaScript* has no notion of *interfaces* or *visibility per se* and it does not provide *typehinting*. Some commonly used patterns exists to provide some version of these functionalities, but most commonly developers rely on proper documentation.

Listing 2.12: Example of overloading in PHP.

```
class Person(){};
$bob = new Person();
$bob->sayHi = function(){return "Hi";};

print('Bob says ' . $bob->sayHi());
/* This method call tries to access the method `sayHi`
 *  of the class `Person`, which is not defined and thus
 * throws the following error:
 * Fatal error: Call to undefined method Person::sayHi()
 */

class BetterPerson(){
  __call($method, $arguments) {
    return call_user_func($this->$method);
  }
};
$pete = new BetterPerson();
$pete->sayHi = function(){return "Hi";};

print('Pete says ' . $pete->sayHi());
/* This method call, too, tries to access the same method.
 * Because `sayHi` is not defined for the class `Person`,
 * the method `__call` is called instead with the name of
 * the inaccessible method and the passed arguments as
 * arguments: `$pete->__call('sayHi', []);`. This in turn
 * calls `call_user_func`, which is able to execute the
 * demanded method, resulting in the correct output:
 * Pete says Hi
 */
```

If a property or method is requested from an object, and it does not exist on this object, the interpreter walks up the objects *prototype* chain to find the requested property or method.

Listing 2.13: Object Oriented JavaScript

```
/* Object constructors in JavaScript are plain
 * functions invoced with the `new` keyword.
 * Public instance methods and properties are
 * attached to the `prototype` property of the
 * constructor, public class methods and properties
 * directly on the constructor. The `this` keyword
 * refers to the newly created instance itself. */

function Person(name) {
   this.name = name;
```

```
11     Person.numberOfPeople++;
12 }
13
14 /* properties and methods attached to Person
15  *  are public class methods/properties
16  */
17 Person.numberOfPeople = 0;
18
19 /* Every ‘Person‘ object inherits properties and
20  * methods of Person.prototype.
21  */
22 Person.prototype.greet = function(){
23     print("Hello, my name is " + this.name);
24 }
25
26 var bob = new Person('Bob');
27 bob.greet(); // Hello, my name is Bob
28 Person.numberOfPeople === 1; // true
```

2.4.5 Model-view-controller Architecture

Model-View-Controller is an architectural pattern applied in application development, facilitating the application of the five **SOLID** guidelines. It is a three-way factoring, whereby objects of different classes take over the operations related to the application, the display of the application's state and the user interaction with the model and the view. [56]

Software-internal representation of information is separated from the ways information is presented to or accepted from the user. Models read, hold and write data, controllers perform actions and calculations, views present data to the user and register user actions.

The model of an application is the implementation of the application's central structure. It can be as simple as an integer (as the model of an counter) or as complex as an object with various properties and methods.

Views deal with everything graphical (or *representational*, more generally). They request data from their model and forward user-actions to a controller. In some implementations, instead of requesting data from their model, the data is pushed to the view by the controller, breaking the coupling between the view and the model.

Controllers contain the interface between their associated models and views and the user input. In the case of applications with a graphical user interface (be it desktop- or web-applications) the user does not act on the controller itself; instead, the view forwards user-actions to the controller.

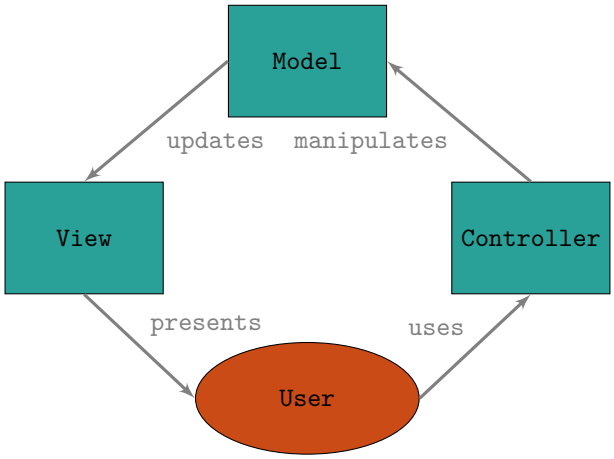


Figure 2.1: A typical collaboration of MVC components.

Routing is a technique which gained prominence with the rise of web-applications. Not classically a part of the model-view-controller (MVC)-pattern, this process translates URLs requested by the client into controller-actions on the server. It also helps to decouple controllers from one another, if there are any dependencies.

For example, take the action of displaying experiments a user has saved in his profile. In a classical MVC-pattern this would be achieved by an URL like the following: `$baseurl/experimentController.php?action=showExperiments&userID=1`. The full control of the HTTP request is passed directly to the `userController`. Any actions associated with the display of stored experiments are to be performed from this controller. This includes connecting to the database, authenticating the visitor, finding the correct user profile and accociated experiments. Even if the code is split in reusable modules, these need to be explicitly aware of one another, in other terms, they are *tightly coupled*.

By using a router as a dependency container, controllers do not depend on an specific implementation of another module, but are expecting the dependency container to provide a suitable module. In our example, this results in a *loose coupling* between the `userController` and the module responsible for database access.

Using a router, one can also use a more human-friendly URL for the action from our example: `$baseurl/user/1/experiments` or `$baseurl/experiments/byUser/1`. In fact, the router can be set up to provide the same result for both request URLs. This setup results in a slightly modified MVC-pattern, as is portrayed in Figure 2.2.

2.4.6 API

An application programming interface (API) specifies a software component in terms of its functions and expected input and output data types. Its main purpose is to define functionalities independent of their respective implementation. APIs provide means to access data, computer hardware, graphical user interfaces (GUIs) and connect

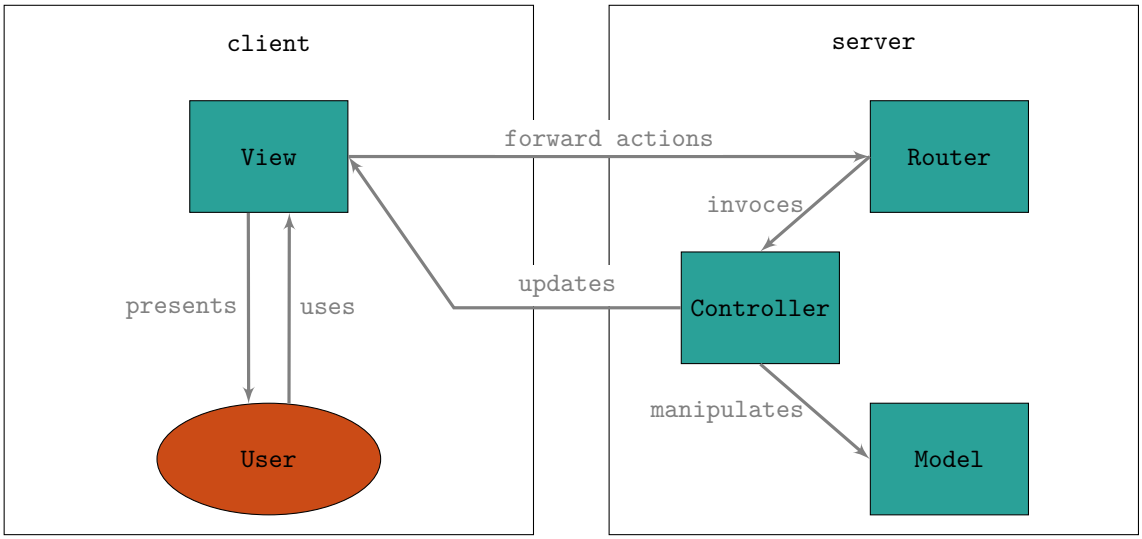


Figure 2.2: Modified MVC-pattern established in web-applications.

otherwise distinct applications. Additionally, APIs can provide access to services on remote machines.

For the communication between APIs, various message protocols and formats exists. API calls to local functions are usually performed directly. In large-scale business processes elaborate software architectures involving various proprietary message protocols and message types are used, while most web services favor HTTP and XML. In recent years, *JavaScript Object Notation (JSON)* has gained importance as a message type.

XML is a sibling to HTML, and various derivatives are used throughout computer programs from office-productivity tools like Microsoft Office to communication protocols and configuration files. It is based around *elements* and *attributes*, *Document Type Definitions* and *XML Schemas* provide powerful tools for data validation. An simple example of an record representing a person is provided in Listing 2.14.

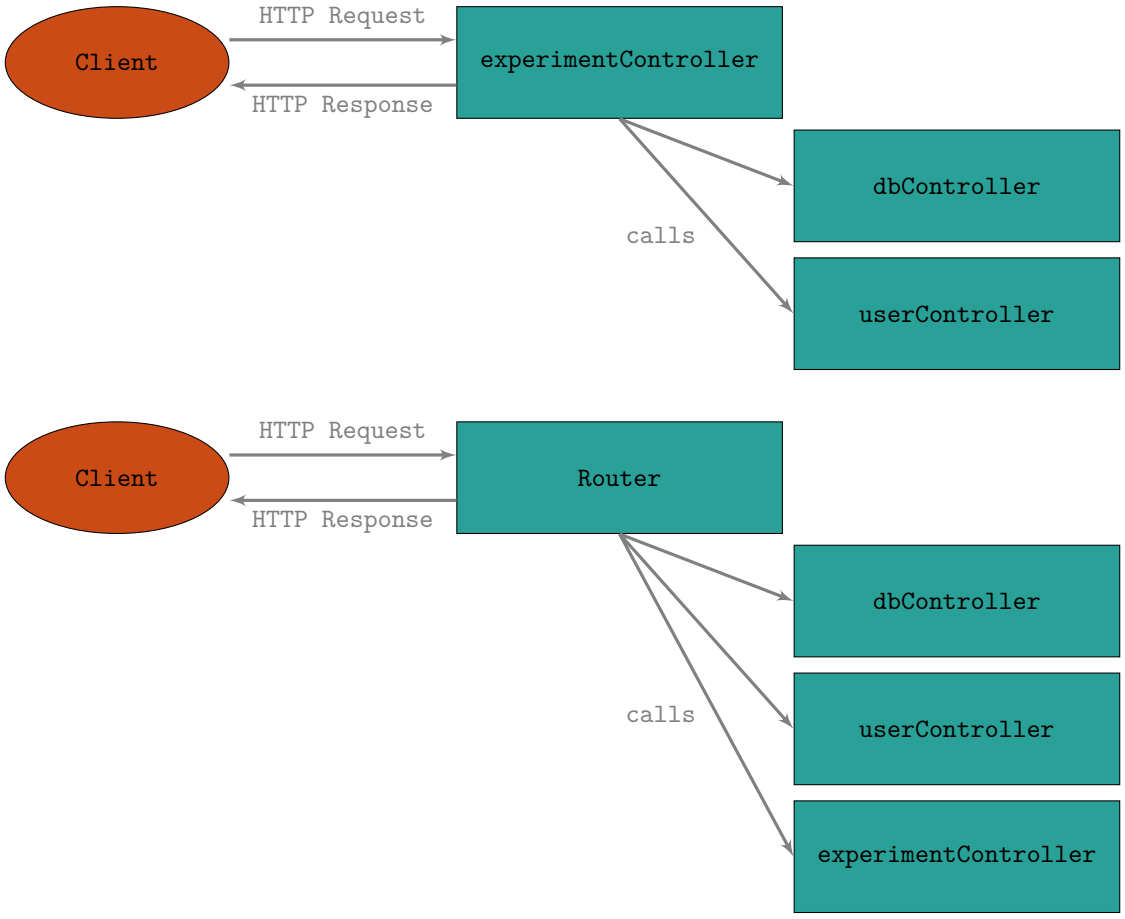


Figure 2.3: Control flow of an HTTP Request in the (a) classical and (b) routed mvc-pattern.

Listing 2.14: Example of an XML document

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber type="home">212 555-1234</phoneNumber>
    <phoneNumber type="fax">646 555-4567</phoneNumber>
  </phoneNumbers>
  <gender>male</gender>
</person>
```

JSON is promoted as a low-overhead alternative to XML. Using curly braces ({}) and brackets ([]) instead of repetitive opening and closing tags, *JSON* documents are often smaller than XML documents. Similar to *XML Schemas* and *XML Document Type Definitions*, *JSON Schemas* can be used for data validation. The example in Listing 2.15 depicts the same record which has been shown in the XML example in Listing 2.14.

Listing 2.15: Example of an JSON document

```
1  {
2    "type": "person",
3    "firstName": "John",
4    "lastName": "Smith",
5    "age": 25,
6    "address": {
7      "streetAddress": "21 2nd Street",
8      "city": "New York",
9      "state": "NY",
10     "postalCode": "10021"
11   },
12   "phoneNumbers: [
13     { "type": "home", "number": "212 555-1239" },
14     { "type": "fax", "number": "646 555-4567" }
15   ],
16   "gender": "male"
17 }
```

3 Aims

For more than a decade now, DWARF was the foundation of over a dozen FSPDs, created, curated and published at the ITB. [76, 26, 53, 63, 28, 37, 38, 66, 78, 64, 55, 77] These databases flexibly combine information about protein sequences kinship, functional annotation and structure information from a variety of sources, and custom tools to aid in the development of biocatalysts. A previously planned extension would have allowed the FSPDs to store not only functional sequence annotations acquired from other database sources, but to collect discrete information about functional parameters from bench scientists directly. For this to accomplish, the data model underlying the DWARF has been reevaluated and changed severely. Because the user interface provided by DWARF could not easily accommodate the extended data model and functionality that goes with it, it was decided that the whole platform would need a thorough refactoring and the result would be published under the name BioCatNet.

BioCatNet now aims to be an updated platform for comprehensive repositories of family-specific protein sequence, structure and functional information. In addition, the platform shall provide tools for the analysis of biochemical information as well as support collaboration and the exchange of knowledge about experimental setups, conditions and outcomes to support journal publications. Naturally all posted data must be handled confidential until the original author decides to publish it.

The present thesis describes the authors contribution to the development of the BioCatNet, namely the development of an application back end, including database access and user management, an web-based application front end, contributions to the shape of the data model and contributions to the set up and maintenance of the server and application environment.

3.1 Data acquisition

The acquisition and curation of sequence and structure information has been a well-established process at the ITB for many years now. The acquisition of data relevant to protein function, on the other hand, was a rather recent idea developed by Constantin Vogel in collaboration with members of the FOR1296 research group.

A prominent approach to acquire information about the biochemical properties of a catalytic protein is to use *data mining*, successfully demonstrated by the enzyme information system BRENDA. A major drawback of this approach is the immense effort

needed to create and maintain text scraping algorithms, which are powerful and tolerant at the same time. Often, information is hidden in tables and figures, impeding the extraction of information. Each scientific text emphasizes different key aspects of the presented enzyme, omitting information, which in turn might be emphasized in another writing. While the number of aspects gathered by *data mining* might be large, the number of aspects common to all scraped scientific writings is rather narrow. Simplified to a spreadsheet, this would mean a large number of rows and columns, but only few columns wholly filled.

The BioCatNet shall emphasize the quality of the data over the quantity of entries it contains. Therefore, a different approach was chosen to collect functional information. While sequence and structure information is acquired by automated processes, the heterogeneous biochemical information is collected directly from bench scientists. Through a set of online forms, selected and comprehensive biocatalytic information can be posted to the database. This of course requires a considerable amount of manual labor, compared to *data mining*, but it ensures a consistently large set of details in each database entry. Looking back at the spreadsheet comparison, this would mean a large number of columns filled completely, while the number of rows is lower.

Therefore, BioCatNet needs an easy to use and appealing user interface for data submission. The interface must be intelligent and adaptive, streamlined and flexible to encourage usage. Repetitive submission of duplicate data must be avoided, submission of similar data accelerated. It needs an stable and scalable back end, which will handle background processes as well as the validation and integration of posted data. This back end must encompass an API, to enforce a strict separation between business and data logic, and provide an easy programmatic access to the data for power users and custom tools.

3.2 Standardization

As mentioned before, *data mining* yields entries with a broad set of details, but only a small number of details is shared across all entries. Our approach to collect data enforces a standardization of the database contents, yielding a wide set of common details. At the same time, this approach enables us to store all information in an SI compliant manner, improving standardization further. As an additional feature, all data will be provided in an easy to use Application Programming Interface (API), as well as standardized and cleanly formatted reports, to be used in custom analyses and documentation.

3.3 Analyses

The large number of common traits makes an equally large number of interesting analyses possible. While some analysis tools will be available on the BioCatNet front end,

the user will also have the option to download data for custom processing or use an API to fetch and post information programmatically.

3.4 Sharing, collaboration, publishing

While the primary objective of the BioCatNet is the acquisition and analysis of biochemical data, we want to try to improve collaboration, too. We want to enable lab groups to collectively edit and review performed experiments, share analyses and comment on techniques. Posted experimental data can also be made public, so that other users of the BioCatNet can examine and comment on those findings. The main goal we are pursuing with this feature, is to give users an easier and streamlined insight into the current state of their respective field, so that they can better decide which experiments need to be performed to make a progress.

3.5 Family-specific protein databases

While the most prominent protein databases like PDB and BRENDA try to encompass all proteins under scientific investigation, the DWARF database system focused on smaller subsets of related protein families. Since 2000, the group of Prof. Dr. Jürgen Pleiss at the ITB, University of Stuttgart, has published multiple versions of FSPDs focusing on different protein families. All of these were build on the DWARF database system, and as a successor, BioCatNet aims to carry future versions of these databases. As section 5.7 on page 89 will describe in detail, several family-specific protein databases have been ported to the BioCatNet system already.

4 Methods

This Chapter will describe applied hardware and software and give an insight into the workflow and into some programming patterns used throughout the BioCatNet codebase. Additionally, third-party libraries used on BioCatNet will be introduced.

4.1 Machine and operating system

BioCatNet is being developed and published on two separate Linux machines running *Debian* operating systems.

The first machine, referred to as *private* subsequently, hosts the development and master branches of BioCatNet. Equipped with an 16-core AMD Opteron™6128 x64 processor and 64GB of RAM, this machine runs *Debian GNU/Linux 7.5 (wheezy)*. This machine is only reachable from within the institute network. The other machine, equipped with an 8-core *AMD Opteron™8214 x64* processor and 16GB of RAM, runs *Debian GNU/Linux 7.5 (wheezy)*, hosts the master branch of BioCatNet, is reachable from the internet and will be referred to as *public* subsequently.

4.2 Software

When deployed, BioCatNet depends only on two pieces of software, namely a *database server* and an *HTTP server*, though the latter requires some advanced configuration additional modules.

4.2.1 Database server

Because of its open source distribution and its longstanding and reliable presence within the institute, *Firebird* has been chosen to be the database system underlying the BioCatNet. Because of its superior performance on parallel queries, the *Firebird superserver* implementation in the latest version 2.5 is being used on both, the *private* and the *public* machine. *Firebird* is a *relational DBMS*, meaning it emphasizes consistent data types and relationships.

4.2.2 HTTP server

Because of its robustness and popularity, *Apache2* has been chosen to be the HTTP server driving the BioCatNet user and application interface. Both machines are running *Apache/2.2.22 x64 prefork*, which is handling most file requests while delegating more elaborate HTTP queries to *PHP* scripts which are described in section 5.2 on page 42 and section 5.6 on page 52.

Notable extensions required for BioCatNet to work properly are *mod-x-sendfile* and *mod-rewrite*. The former speeds up file downloads immensely, while the letter allows for the use of descriptive URLs instead of hard-to-read query strings (compare `biocatnet.de/sequence/1` and `biocatnet.de?page=sequence&sequenceId=1`). The most crucial extension though is *mod-php5* which passes the HTTP request to *PHP* scripts.

4.2.3 Bioinformatics tools

BLAST+ is a suit of tools provided by the NCBI. Improved algorithms and concurrent searches of sequence fragments provide an dramatically increased runtime compared to other local alignment search methods. [15] Despite its use of heuristic methods, the accuracy of the produced alignments is considered to be more than adequate. The BLAST+ suite exposes multiple command-line tools, two of which are being used within the BioCatNet: `blastp` searches for sequences similar to an input sequence in another set of sequences, which may be either a file containing multiple *FASTA* sequence entries, or an sequence database file optimized for the use with *BLAST*. These databases are generated with the command line tool `makeblastdb`. The BLAST+ suite is employed in the version `2.2.29+` on both BioCatNet machines.

Clustal-Omega is a general purpose MSA program for amino acid and nucleotide sequences. [62, 17] Due to its use of advanced algorithms for calculating guide trees, it can deal with many tens of thousands sequences in reasonable time with considerable accuracy. It can align sets of sequences against each other and produce hidden Markov Model (HMM) profiles for later alignment of novel sequences. Clustal-Omega is available as the command-line tool `clustalo` and its latest version `1.2.0` is being used on both BioCatNet machines.

The standard numbering generator is currently not published openly and only implemented for use within the ITB and BioCatNet as an *Perl* library. [68]

4.3 Workflow

As described previously, the *private* machine hosts the development branch of the BioCatNet. Using *SSH*, a command-line interface connection can be set up with the machine. Development was conducted then using the command-line text editor *GNU Emacs* and the version control tool *git*.

Development included mostly the following tasks:

- set-up, configuration and maintenance of the HTTP server
- set-up, configuration and maintenance of the Database server
- writing server-side code in *PHP* and *Perl*
- writing client-side code in HTML/*Mustache*, *CSS* and *JavaScript*
- integration of third-party-code on client- & server-side
- maintaining version control and publication to the public machine
- testing

4.3.1 Version control

To aid development and publication the open source version control tool *git* has been used (see subsection 2.4.1 on page 8). *Git* is distributed, has a tiny footprint and is being used in millions of projects including Android and Linux development.

The versioning workflow presented by Vincent Driessen has been adopted for the development of the BioCatNet.[22] Two main project branches are declared for the BioCatNet, *development* and *master*, with the latter being the stable release, published on the public machine and used inside the institute on the private machine. Minor *feature* and *hotfix* branches are created on-demand, worked on, merged and discarded.

4.3.2 Back end

The BioCatNet makes use of a couple of advanced *PHP* features and libraries. To understand the BioCatNet code base, one needs to understand these features and building blocks.

Namespaces are a means to further encapsulate code. Namespaces are designed to avoid name collisions between *PHP*-internal functions, third-party code and the code you create. That way, you could use the functions like `\Some\Library\sayHello` and `\My\Library\sayHello` side by side. For the BioCatNet, proprietary libraries are available under the namespace `ITB`, the application itself resides in the namespace `ITB\BCN`.

Autoloading abolishes the need for *include* statements in each and every file. While it is a common practice for small projects to include code from other files and third-party modules, it becomes a rather cumbersome task for projects consisting of hundreds of files. With *autoloading*, the *PHP* interpreter looks up class names based on their namespace at run-time. An instantiation of the class `cebe\Markdown\Parser` results in an automatic inclusion of the file `./cebe/Markdown/Parser.php`.

Method chaining is a simple technique to increase code readability and maybe reduce memory usage by a few bytes by avoiding the creation of new pointers. Instead of assigning the return value of an function to a variable, one can instantly call a method of the returned object. If instance methods return the instance itself, one can perform multiple operations on the instance in one expression. An example of this pattern can be found in Listing 4.1.

Listing 4.1: Method chaining in PHP.

```
/* Method chaining on separate objects: instead of this */
$db = new DB(...);
$transaction = $db->getTransaction();
$query = $transaction->getQuery(...);
$result = $query->execute();

/* with method chaining one can write this */
$result = New DB(...)
          ->getTransaction()
          ->getQuery(...)
          ->execute();

/* Method chaining on the same instance: instead of this */
$bob->setName('Bob');
$bob->setAge(26);
$bob->sayHello();

/* with method chaining one can write this */
$bob->setName('Bob')
    ->setAge(26)
    ->sayHello();
```

4.3.3 Front end

In contrast to regular PHP programs on a server, which are executed once and then exited, *JavaScript* web-applications in a browser environment are executed inside an *event loop*. This allows a decoupling of *calling* and *response-processing* code (which is referred to as *callback*) and gives the *JavaScript* runtime a chance to do other things while waiting for the answer.

Building on that premise, *event driven programming* has evolved to be the predominant programming paradigm in *JavaScript* web-applications, where the program flow is determined by events such as user actions, sensor outputs or messages from other parts of the program or even other programs. Such event interactions are described in Listing 4.2.

Listing 4.2: Event driven programming in Javascript

```
function sayhello(){ print('Hello World'); }
/* Every time ‘document‘ emits the event ‘click‘ the function
 * ‘sayhello‘ will be executed. In this case ‘sayhello‘ is the
 *  callback function.
 */
document.addEventListener('click', sayhello);

/* Callbacks can also be defined anonymously, i.e. without
 * prior function declaration and are often passed as
 * an parameter to asynchronous functions.
 */
require('http://www.biocatnet.de', function (response) {
    print('response received');
});
```

ASYNCHRONOUS JAVASCRIPT AND XML (AJAX) is the technique used to load additional content without refreshing the whole web page. The *JavaScript* `XMLHttpRequest` API, available in all browsers, is used to dispatch and react to HTTP requests. It enables loading of document fragments, images and data.

4.4 Third party libraries

4.4.1 Mustache

As described in section 2.4.3 on page 12, *Mustache* is a *logic-less* templating language, and the templating language of choice for the BioCatNet. Two implementations have been used within the BioCatNet: The server-side implementation in *PHP* (bobthecow/-mustache.php [32]) and the client-side implementation in *JavaScript* (janl/mustache.js [40]).

4.4.2 Ketcher

Ketcher is a front end tool for drawing chemical molecules with back end support for automatic layout and (de-)aromatization. The front end is written in pure *JavaScript* and utilizes Scalable Vector Graphics (SVG) for rendering, abolishing the need for Java or Flash plugins. The back end is a small *python* script running on the *apache* HTTP

server. *Ketcher* is free, open-source, and very easily integrated into existing websites. [65]

4.4.3 Back end

Indigo is a universal, open-source organic chemistry toolkit developed by GGA Software Services LLC. It contains first-class tools for end users and is free for non-commercial purposes. Supporting a wide variety of chemical file formats as well as automatic drawing of chemical compounds it is a valued toolkit among chemo- and bioinformaticians. The BioCatNet uses the indigo toolkit to generate depictions and canonical *SMILES* codes of chemical structures. The front end drawing tool *Ketcher* also depends on functions provided by the indigo library. [34]

Open Babel is a chemical toolbox to convert between different chemical file formats, rotate molecules, analyze conformers and many more tasks. On the BioCatNet, babel is being used to calculate the molecular weight of chemical compounds. [48, 46]

Jbroadway/analog is a small logging package for *PHP*. Despite its small size, the log format is customizable and it supports various logging handlers writing to local files, emails and log management servers if need be. An invaluable extension when debugging server-side code. [14]

Ircmaxell/password-compat is a library intended to provide forward compatibility with password functions planned for the next *PHP* version 5.5. Functions provided by this package are used to securely store obfuscated user passwords to the database and compare them during user log-in. [25]

Cebe/markdown is fast and highly extensible markdown parser for *PHP*. It is used to present the BioCatNet documentation, which itself is written in markdown format. [13]

4.4.4 Front end

RequireJS is a popular and powerful *JavaScript* module loader. With a growing number of front end modules it becomes difficult for the developer to manage module loading efficiently in regards of module dependencies and page loading time. RequireJS not only provides an simple to use library and a simple pattern to follow to alleviate these issues. Moreover, it provides command-line tools to further condense and optimize the front end code, to reduce page loading times even more. [57]

jQuery is a cross-platform *JavaScript* library designed to simplify the manipulation of HTML elements on websites. It provides an layer of abstraction to *JavaScript* functions which often seem convoluted originally or have different implementations on different browsers. Because of its high popularity and consequential influence in the developer community even influenced the development of the *JavaScript* language itself. [35]

Bootstrap is a popular HTML, *CSS* and *JavaScript* framework providing clearly structured and aesthetically pleasing standard website layouts. Though the framework has a focus on responsive websites - *i.e.* websites scaling well to the screen size - it is an excellent framework also for desktop-sized web-applications. [10]

The JalviewLite Applet is a free and easily embeddable MSA viewer for websites. While it does depend on *Java* being installed on the user machine, it provides superior functionality and very high performance. [71] The integration and usage will be described in more detail in section 5.6 on page 52.

PV is a *JavaScript* viewer to visualize protein structures directly in the browsers. It provides various visualization options and is performing extraordinary well on recent browsers supporting WebGL. It's main advantage the fact that it is written in *JavaScript* and thus requires no additional add-ons on the client machine. [8, 9]

5 Results

At the time of this writing, version 2.4.16 of the BioCatNet is published at `https://www.biocatnet.de`. The TEED and Imine Reductase Engineering Database (IRED) can be accessed directly at `http://teeds.biocatnet.de` and `https://ired.biocatnet.de`, respectively. A first try of an documentation can be found at `https://wiki.biocatnet.de`. Issues and feature requests can be posted at `https://bugs.biocatnet.de`.

Though BioCatNet is founded on DWARF, it has deviated significantly during development. The data model has grown not only in size but in complexity and the user interface has been recreated from scratch. Whereas the DWARF system provided separate user interfaces for administrators and users, the interface has been unified in the BioCatNet.

The BioCatNet is build on various standard web technologies. The server-side code is largely composed of *PHP*-scripts, arranged in an MVC-pattern (subsection 2.4.5 on page 20). Only a few scripts are written in Perl, handling long-running tasks.

The websites presented to the user are build from *Mustache* templates on the server- as well as client-side. The websites are styled using *CSS* and *JavaScript* provides client-side functionality.

5.1 BioCatNet data model

Being founded on DWARF, BioCatNet has inherited its understanding of protein kinship in terms of protein families, superfamilies, homologous families and proteins, with the amino acid sequence being the single dimension to define the degree of relatedness.

Protein sequences which are found to show more than 98% similarity are defined to belong to the same Protein. The protein *2-succinyl-5-enolpyruvyl-6-hydroxy-3-cyclohexene-1-carboxylate synthase* (protein#2649 from the latest TEED), for example, encompasses 4 distinct amino acid sequences which differ in only a dozen amino acids. Protein sequences which are more than 60% similar to one another, are grouped to form homologous families. Superfamilies are defined by experienced curators taking into account protein structure and functions. Extending this hierarchy, BioCatNet adds the notion of superfamily- and homologous family-groups, which are defined and assigned manually based on protein function, notable sequence motifs or structural characteristics.

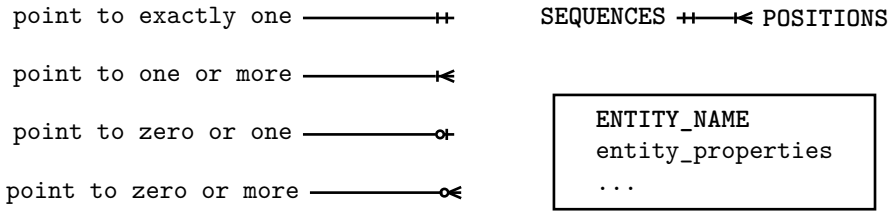


Figure 5.1: Legend and examples describing data model relationships. Any SEQUENCES entity is linked to at least one, but possibly more, POSITIONS entities, which themselves are linked to exactly one SEQUENCES entry.

At the time of this writing, the BioCatNet data model encompasses 57 user, protein, taxonomy and experiment-related entity types. Throughout the collaborative development with Constantin Vogel, the number has been growing to accommodate new entities and relations to arrive at a number more than twice as large than the DWARF's 23 entities.

The figures in this chapter describe the data model, *i.e.* entity types and their relationships. The type of relationship between connected entities is portrayed in their connector's arrow tips. (Figure 5.1).

5.1.1 Data model related to the protein sequence

Most of the entity types related to protein sequences as well as to their source organisms have been passed on from the original DWARF data model, though many relationships have changed.

One notable change applied in the context of sequence information, is the introduction of a SOURCES entity. Because the BioCatNet lays its focus on the amino acid sequence of an protein, it is desirable that this sequence is unique within the database. On the other hand, one particular protein sequence is not bound to any particular host organism. The SOURCES entity describes the unique connection between an amino acid sequence (SEQUENCES, Figure 5.3) and the organism it was found in (TAX_NODES, Figure 5.2 on page 39).

The NCBI databases, which are the source for sequence information do not make this distinction, neither does the PDB, which is why entities describing the connection between sequence information on the BioCatNet and foreign databases (PDB_ENTRIES, DB_ENTRIES, DB) refer to a particular SOURCES entry rather than a particular SEQUENCES entry. (Figure 5.2 on page 39)

To speed up database transactions and simplify user output, organism names have been split into preferred and secondary names (TAX_NAMES and TAX_SYNONYMS, respectively). Thus, each TAX_NODE entity is linked to exactly one TAX_NAMES entry and to zero or more TAX_SYNONYMS entries.

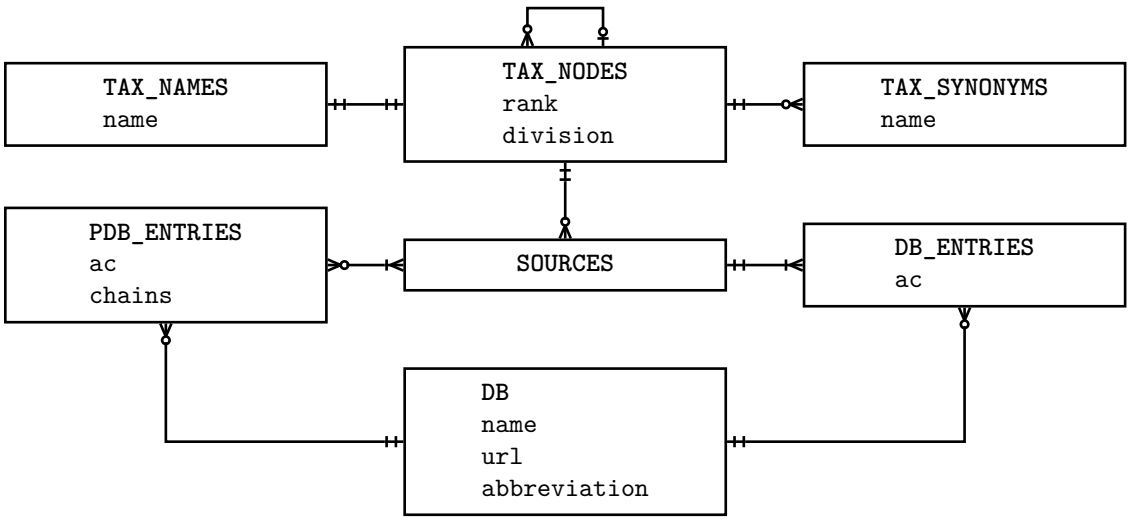


Figure 5.2: Objects and relations of the BioCatNet data model related to organisms and cross-database references.

The vertical protein sequence kinship hierarchy is realized with the `S_FAM`, `H_FAM`, `PROTEINS` and the `SEQUENCES` entities. Horizontal kinship between superfamilies and homologous families is established through `S_FAM_GROUPS` and `H_FAM_GROUPS` entities. Considering the discussion provided by Fischer *et al.* [27], the amino acid sequence is provided by distinct `POSITIONS` entries which point to a common `SEQUENCES` entry. In addition, this separation enables the BioCatNet to provide a standard amino acid position number with respect to standard numbering schemes.

Functional annotation is applied to proteins by the assignment of an *Enzyme Commission (EC)* number and the corresponding `EC` database entity, if it can be inferred from the sequence annotation. Sequence annotation is established by connecting an `ANNOTATIONS` entry with an `POSITIONS` entry using the `ANNOTATION_ENTRIES` entity, which points to the starting and ending position of an annotation. Because single amino acids carry distinct properties, too, the data model holds two additional entities describing amino acids and their properties (`AMINOACIDS` and `AA_PROPERTIES`, respectively).

5.1.2 Data model related to structural information

Structural information is in part covered by the entities `PDB_ENTRIES` and `DB`, described in the previous section. With `SOURCES` as an linking entity, multiple three-dimensional structures may be assigned to a single amino acid sequence. Because the PDB may store an X-ray structure of an multimer with heterogeneous chains, the `chains` property will point out which of these are linked to the respective amino acid sequence.

Additionally to X-ray crystallography structures, TEED has implemented an algorithm to predict three dimensional structures based on homology modelling. [68] In short, the

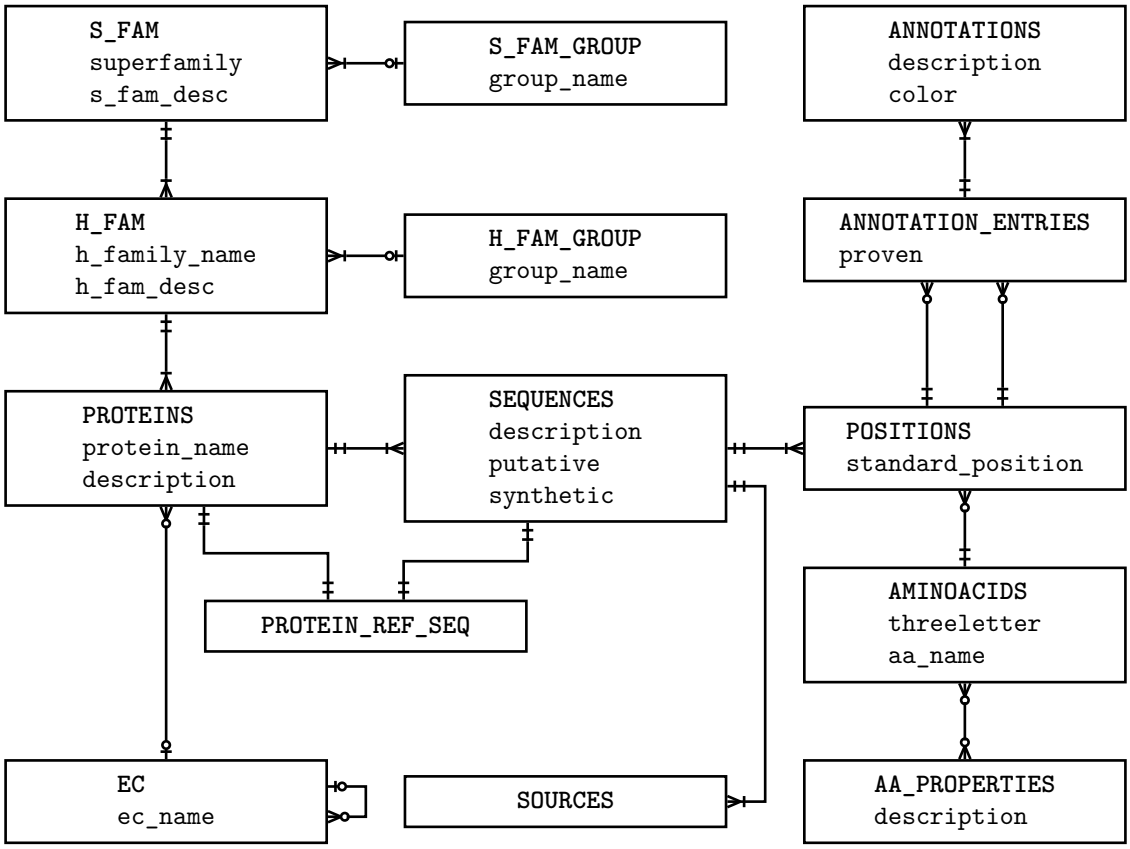


Figure 5.3: Objects and relations of the BioCatNet data model related to protein sequence.

algorithm finds and loads all PDB entries of the respective protein family database, performs an structural alignment, and subsequently tries to find suitable templates in this aligned structure library for sequences missing and experimentally determined structure. When successful, several homology models complete with analytics are computed. This additional information about homology models and templates is stored across the entities `MODEL_MONOMERS`, `MODEL_TEMPLATES`, potential protein-multimer information resides within `MODEL_MULTIMERS`

5.1.3 Data model related to biochemical function

The decision to refurbish the DWARF database system and re-launch it as BioCatNet was driven by the need to incorporate information about the catalytic activity of proteins. This refurbishment was accompanied by an extension of the data model to cover kinetic parameters of the enzyme, environmental conditions of the experiment and information about substrate and product specificities.

The central component of this extension is the `EXPERIMENTS` entity, pointing to experiment conditions as well as to enzyme, substrate, additive and product compositions over the course of the experiment.

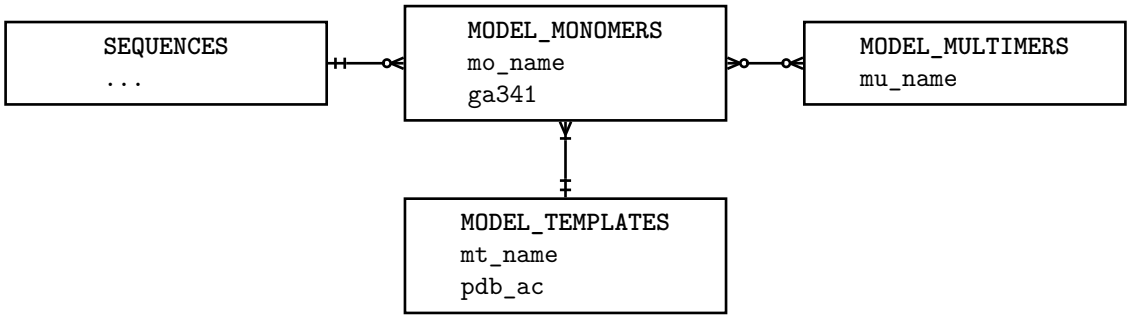


Figure 5.4: Objects and relation of the BioCatNet data model related to three-dimensional structure and homology models.

The CONDITIONS entity, directly linked to EXPERIMENTS, holds information about the temperature and pH-value of the reaction mixture as well as the pressure and shaking frequency thereof. This entity also refers to a single BUFFERS entity, to describe the solvent composition.

The entity type REACTIONS aims to generally describe reactions, i.e. what the reaction type is and what the expected substrates and products of this reaction are, and is therefore connected with one or more REACTION_TYPES and one or more COMPOUNDS. In theory and practice, a single experiment can encompass multiple reactions which may be concurrent or cascading. To accommodate this fact, EXPERIMENTS, too, is linked in a one to one-or-more relationship to REACTIONS.

One or more ENZYME_FEEDS entities point to a single experiment, describing the point in time and amount of enzyme added to the reaction as well as the solvent volume. This entity forms the bridge between the EXPERIMENT and SEQUENCES entities. The ENZYME_PREPARATION_METHODS entity holds detailed descriptions about the procedure used to prepare the enzyme. This information can be as simple as a vendor and an item identifier or as complicated as the engineering of expression strains and the purification process thereafter.

The entity types SUBSTRATE_FEEDS and ADDITIVE_FEEDS store information about the chemical compounds brought into the reaction. Like their counterpart, they describe the solvent volume as well as the point in time, amount, and methods used to prepare the respective chemical compounds. These are provided by the COMPOUNDS entity, while the vendor or preparation method are stored as SUBSTRATE_PREPARATION_METHODS entities.

The entity COMPOUND_MEASUREMENTS finally describes the concentrations of products measured in the course of the reaction. Pointing to the entity METHODS, a detailed description of the used measurement method is available, too.

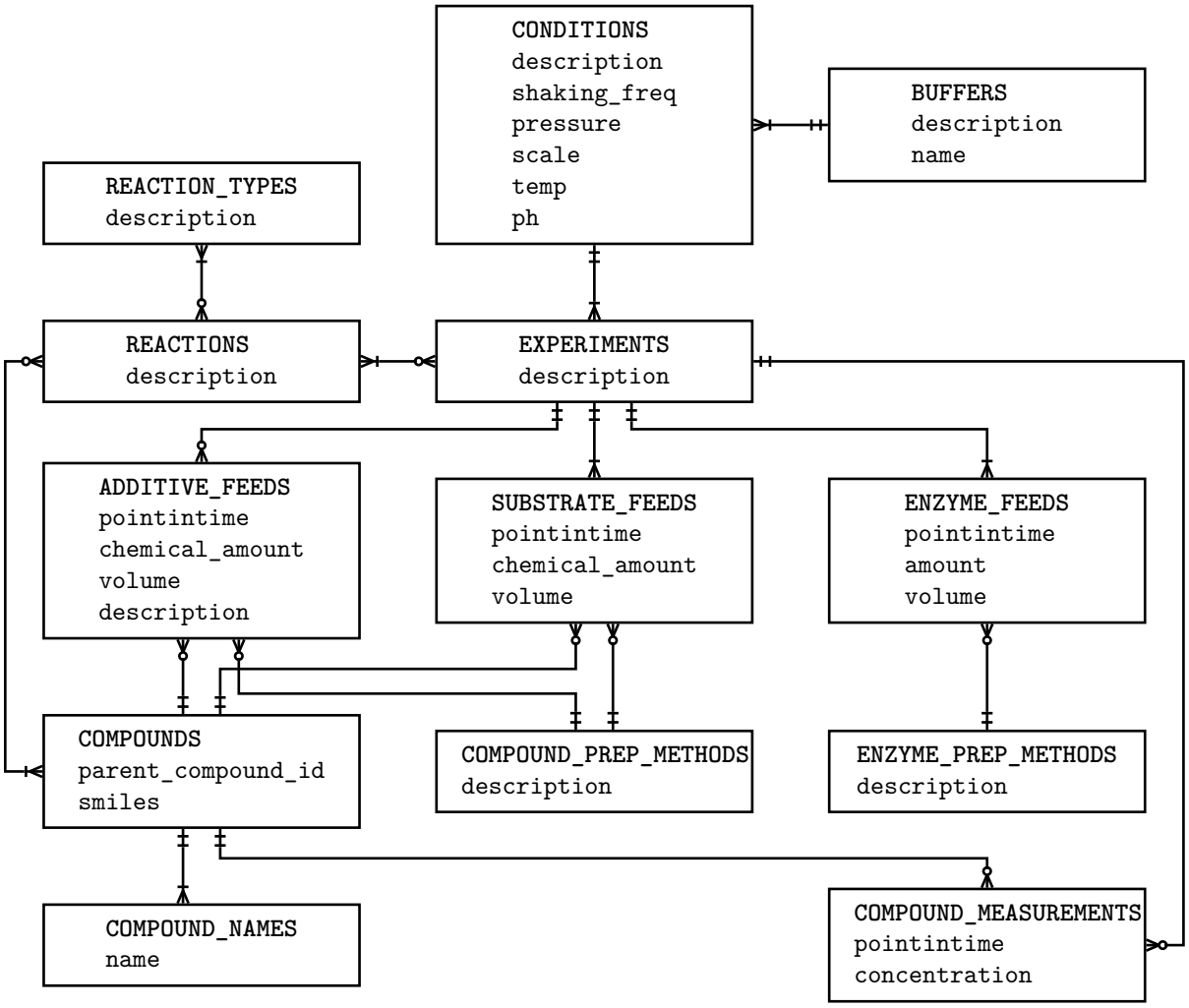


Figure 5.5: Objects and relations of the BioCatNet data model related to the experiment set-up.

5.2 BioCatNet back end libraries

Following the convention, in the following, object class names are capitalized, like `Application`, while object instances are lowercase variable names like `$application`. When talking about static class methods, the scope resolution operator *Paamayim Nekudotayim*, or simply *double colon*, is used, while instance methods of classes are symbolized with an arrow (`Application::init` and `$application->init`, respectively).

While there are several sophisticated *PHP* MVC-libraries available, it became clear early on, that they do not fit our requirements. As a result, an proprietary MVC-library has been created to host the BioCatNet. Similar is true for object-relational-mapping libraries. All proprietary libraries are found in the `lib` folder of the BioCatNet distribution and are available in the `ITB` namespace. Besides those two already named,

proprietary libraries have been created to handle custom Exceptions, *JSON* parsing, document types and support the creation of long-running tasks.

Within the BioCatNet, there are actually two libraries handling HTTP responses in cooperation. Available under the namespaces `ITB\MVC` and `ITB\Router`, the former supports the creation of objects fitting the MVC-pattern while the latter is responsible for routing, i.e. managing the connection between user request, executed controller actions and the response. The libraries `ITB\MIME`, `ITB\JSON` and `ITB\Traits` provide supportive functions, `ITB\Exceptions` provides custom error classes and `ITB\Worker` simplifies the interaction of *PHP* with *unix* tools.

5.2.1 ITB\Router

- `ITB\Router\Application`
- `ITB\Router\Response`
- `ITB\Router\Request`

Application Each HTTP Request is being handled by an object of the class `Application`, which holds the configuration as well as instances of the object classes `Request` and `Response`. In the BioCatNet, the `Application` class acts as the *Router* described in section 2.4.5 on page 21. In the following, *route* will refer to the specific sequence of controller actions associated with an user request.

Request class instances hold the HTTP request body, query string, and request headers. The class provides convenience methods to check for data types the user will accept and the freshness of the request. It will also carry custom data between controller actions, acting as a dependency container.

Response class instances hold response headers and provide methods to facilitate HTTP responses. Among others, methods for responding with *JSON* data, a file from hard-disk or an simple status code are provided. It is also responsible for rendering HTML templates and responding with HTML documents, using provided parameters. The engine used for this rendering process is defined in and fetched from the *Application* instance which created the response.

Objects of the classes `Request` and `Response` contain both a back-reference to the instance of class `Application` which is holding them.

Once the `$application` object is initialized, it routes the instances of `$request` and `$response` through applicable instances and methods of the object class `ITB\MVC\Controller`. Each controller method receives the `$request` and `$response` object and a continuation function `$next`, which must be called if the control is to be passed to the next controller. Any method that is not passing the control back to the

router by calling `$next()` must call `$response->end` to cleanly terminate the HTTP connection. Otherwise, the HTTP request is terminated abnormally, resulting in a *Request Timeout* for the end user.

5.2.2 ITB\MVC

- `ITB\MVC\ControllerAbstract`
- `ITB\MVC\ModelAbstract`
- `ITB\MVC\StorableAbstract`
- `ITB\MVC\Collection`
- `ITB\MVC\DatabaseStorage`
- `ITB\MVC\Model`
- `ITB\MVC\DatabaseWrapper`
- `ITB\MVC\StatementWrapper`

Controller objects have been mentioned already in the previous section. Controller objects accessed by `ITB\Router\Application` need to inherit from `ControllerAbstract`.

If present, the `Controller::init` method of each controller that has to be passed is called. If any particular controller action is configured in the route, the corresponding method is called next. Each method that is part of the current route receives the `$request` and `$response` objects as well as the continuation method `$next`, and it can modify these objects if need be. Database connections, for instance, are attached to the `$request` object, while output parameters are attached to the `$response` object. If `$next` is called, the control is passed back to the `Application` instance.

Model objects represent the data entities that an `Controller` instance can work with. The BioCatNet defines its models in the namespace `ITB\BCN\Models` and they inherit from `ITB\MVC\ModelAbstract`.

In the case of the BioCatNet, models have an extended functionality and serve as *Object Relation Mapping (ORM)* elements, building the bridge between persistence-layer entries and objects manipulated by `Controller` objects. To interact with a persistent storage, `Model` entities expect an object implementing `StorageInterface`. Using this *$storage* object, they create, retrieve, update and delete entries. Currently, the library only provides a `DatabaseStorage` class, specific to the *Firebird* relational database management system. Given the modular structure and the dependency on abstractions instead of concretions, one can easily extend the application to allow other forms of persistent storage.

Working in conjunction with objects of the classes `DatabaseWrapper` and `StatementWrapper`, objects of the class `DatabaseStorage` provide a layer of abstraction to database access, translating *PHP* statements into *SQL* queries. Listing 5.1 provides a simplified insight into the inner workings of these classes.

Listing 5.1: Simplified example of object-relation-mapping provided by the ITB\MVC library

```
$dbWrapper = new DatabaseWrapper($connectionOptions);
$storage = new DatabaseStorage($dbWrapper);

// find a ‘Sequence‘ object in ‘$storage‘ where the
// property ‘sequence_id‘ equals 1
$mySequence = Sequence::findOne($storage, ['sequence_id' =>
   1]);

// an alternative form for the same command is
$mySequence2 = $storage->findOne('Sequence', ['sequence_id'
   => 1]);

// DatabaseStorage::find translates the arguments into
// an SQL query:
//   SELECT FIRST 1 * FROM sequences WHERE sequence_id = 1;

// the table columns are mapped to object properties
echo $mySequence->id; // prints 1
echo $mySequence->name; // prints ‘pyruvate oxidase‘

// using ModelAbstract::findOne, only a single entry
// is returned, like in the cases above. Using
// ModelAbstract::find, a collection is returned

$mySequences = Sequence::find($storage, ['description' => '~
   pyruvate oxidase']);
echo count($mySequences); // 827
```

Views are no *PHP* objects in the BioCatNet by default, but rather templates written in the *Mustache* templating language (section 2.4.3 on page 12). Using templates, BioCatNet achieves increased separation of presentation logic and business logic. Additionally, *Mustache* templates can be shared between server- and client-side rendering implementations increasing code-reuse.

In this sense, views are no part of the BioCatNet MVC abstracts library, but concrete implementations within the BioCatNet application.

5.2.3 ITB\JSON

- `ITB\JSON\JSON`

The JSON class provides methods for the conversion of *PHP* objects to the interchangeable *JSON* format and vice versa. It abstracts *PHP*'s own *JSON* encoding and decoding methods and provides ways to read *JSON* files and parse *JSON* strings containing comments. This capability is not defined within the *JSON* specifications, but has proven to be an useful feature. Additionally, it provides a way to parse *JSON* data into class instances directly.

5.2.4 ITB\Mime

- `ITB\Mime\Mime`

The Mime class provides methods to recognize data types by file extensions, to recognize data types passed from user requests and check for data types the user might expect, a functionality missing in *PHP*.

5.2.5 ITB\Traits

- `ITB\Traits\URLTrait`
- `ITB\Traits\CallTrait`
- `ITB\Traits\CreateTrait`
- `ITB\Traits\ErrorLogTrait`
- `ITB\Traits\PermissionTrait`

This library contains traits which can be injected into *PHP* class definitions. Traits provide additional means for code reuse beside class inheritance.

URLTrait adds a method to construct a unique uniform resource locators (URLs) for the respective class instance. Additionally, it overrides the classes serialization method to always include the URL in the return value.

CallTrait appends the *magic method* `__call` to the receiving class. This feature enables instance methods to be called which have been defined at runtime.

CreateTrait appends the static class method `create` to the receiving class, providing an alternative to class instantiation using the `new` keyword. (Listing 5.2)

Listing 5.2: Example of an PHP class using ITB\Traits\CreateTrait.

```
class Person {
   use CreateTrait;
}

$bob = Person::create();
$pete = new Person();

print($bob instanceof Person); // true
print($pete instanceof Person); // true
```

PermissionTrait adds methods and properties to the receiving class to determine whether an object is readable and/or writable by an specified user.

5.2.6 ITB\Workers

- `ITB\Worker\ShellWorker`

The ShellWorker augments application calls at the system level. It supports background tasks as well as the definition of output and error files.

5.3 BioCatNet front end libraries

During development of the BioCatNet front end, several often used functionalities have crystallized to small libraries, ready to be used modular and detached from the BioCatNet. They reside within the directory `pub/js/lib` in the BioCatNet distribution.

The libraries `obj.js, fn.js, string.js, arr.js` are the most generic libraries and provide some useful extensions in the context of *JavaScript* objects, functions, strings and arrays. The libraries `promise.js`, `eventEmitter.js` and `xhr.js` simplify the development of asynchronous functions in *JavaScript*. Functions from `cookie.js` simplify access to HTTP cookies on the client side and `param.js` helps with the construction and parsing of URLs.

The two libraries `api.js` and `pubchem.js` simplify the usage of the BioCatNet HTTP API and PubChems HTTP API, respectively.

The library `render.js` provides an wrapper around the client site *Mustache* templating engine, `remote.js` extends this functionality further and allows asynchronous fetching of templates and data for the rendering process.

5.4 BioCatNet application back end

The BioCatNet application itself resides in the directory `app` of the BioCatNet distribution and available *PHP* classes are available in the `ITB\BCN` namespace. The application is comprised of different MVC-pattern components as well as some helper modules which wrap around system calls and long-running tasks.

5.4.1 Models

The *PHP* classes defined in the `ITB\BCN\Models` namespace resigning in the `app/Models` directory are mostly small classes of descriptive nature, inheriting virtually all of their functionality from the library class `ITB\MVC\ModelAbstract`. Acting as *ORM* elements, they provide a means to manipulate the database in an object-oriented manner. As such, they correspond directly to the data entities defined in section 5.1 on page 37. The few classes which provide functionality beyond this scope, are described in the following.

ITB\BCN\Models\User has additional methods to encrypt user passwords before they are saved to the database as well as a method to check if a given password matches the password provided by the user.

ITB\BCN\Models\TaxSibling does not correspond to a database entity, but by adopting the common models interface, it provides a homogeneous method to find taxonomic nodes which share the same parent node.

ITB\BCN\Models\Sequence is the larges model class with the greatest number of additional functions. It provides various methods to access related data like associated structures and sources, construct a sequence representation in FASTA format, and enrich the sequence instance with functional annotations as well as family relations.

5.4.2 Views

As described in subsection 5.2.2 on page 44, *views* in the BioCatNet are no *PHP* classes as the original MVC pattern would suggest but rather HTML templates written in *Mustache*. While the templates used for the general page layout reside in the directory `app/Views/layout`, the rest is organized in page-specific subdirectories within `app/Views`.

5.4.3 Controllers

Most `Controller` classes in BioCatNet directly correspond to one or more pages of the graphical user interface. Those which do will be described together with the user interface they are presenting in section 5.6 on page 52. The API controller will be described in detail in section 5.5 on page 50. The others will be outlined briefly in this subsection.

ITB\BCN\Controllers\Cookie is a small utility class that is usually loaded before any other controller and handles, as the name implies, HTTP cookies. It attaches itself to the current instance of `ITB\Router\Request`, so that any following controller can make use of its functions.

ITB\BCN\Controllers\Session depends on the cookie controller described previously and establishes a way to preserve data across subsequent access. A session is limited to the same user/machine/browser combination and to a certain time-span. This controller wraps around the native session functionality of *PHP* to provide object-oriented methods for the manipulation of sessions. Like the cookie controller, it attaches itself to the current instance of `ITB\Router\Request`.

ITB\BCN\Controllers\Cache provides a way to preserve a larger amount of data in memory than the cookie and sessions controllers are capable to do. It wraps around the *apc* extension of *PHP*, and attaches itself, like the two controllers mentioned before, to the current instance of `ITB\Router\Request`.

All of the above mentioned controllers expose a simple, homogeneous, object-oriented syntax, with methods the methods `get(key)` and `set(key, value)` to manipulate underlying structures.

ITB\BCN\Controllers\Flash exposes developer-friendly methods to create user notifications to be displayed in either the current request or in subsequent ones. This controller depends on the session controller to store the user messages and is attached to the current instance of `ITB\Router\Response`, as it is closer related to user output.

ITB\BCN\Controllers\Database is responsible for the database connection and exposes, by initializing objects from the `ITB\MVC` library, primitive methods for database interaction.

ITB\BCN\Controllers\Workbench contains a group of controllers which control long running tasks, the input of experiment data and storage of intermediate input data. The respective components are described in detail in subsection 5.5.1 on page 51 and subsection 5.6.10 on page 70.

5.4.4 Worker

The directory `app/Workers` harbors the scripts used for long-running tasks like the generation of BLAST databases and sequence alignments. Those functionalities will be discussed in detail in subsection 5.5.1 on page 51 and subsection 5.6.10 on page 70.

5.5 BioCatNet API

Early in the BioCatNet development it was clear, that we want a strict separation between the business and presentation logic. Not only would this comply to the *Separation of Concerns* paradigm defined in subsection 2.4.4 on page 14, but also ease the access to BioCatNet data and function from clients other than the BioCatNet website.

The communication between business logic and presentation logic, or any other client, is enabled by an HTTP API. An abundance of HTTP libraries is available for every programming language, making the API easy to use with little programming experience.

To perform basic CRUD-Operations, a dedicated URL route is available at `$baseURL/API`. To fetch a list of sequences matching some naming pattern, for example, one can simply issue an HTTP GET request to the URL `$baseURL/API/sequences?name=$pattern`.

The API returns results in *JSON* format, described in subsection 2.4.6 on page 22, if not stated otherwise. In general, **READ** operations are performed issuing an HTTP GET request:

- `$baseURL/API/$collection/$id`
- `$baseURL/API/$collection[?$parameters]`

While the first pattern returns exactly one item, the latter API call will return a list of items. `$parameters` is a URL query string formatted according to RFC3986 section 3.4 [7]. If `$parameters` is present, the values present therein will limit the result set. Parameter keys applicable to all collections are the following:

- `offset` defines how many items will be skipped
- `limit` defines the number of items to be returned
- `include` defines which related objects will be fetched

If not explicitly defined, `offset` and `limit` are implied to equal 0 and 100, respectively. Any additional parameter key is assumed to be an item property that has to match the parameter value. With the parameter `include`, one is able to include related elements in the result set. With the URL `$baseURL/API/Sequences?include[]=Protein&include[]=ProteinRefSeq`, for example, one may retrieve a list of Sequences and the connected Protein and ProteinRefSeq entries.

The returned object will, if not declared otherwise, contain the properties

- `request` which reflects the query parameters passed to the API
- `response length` a numeric value reflecting the number of items returned in the result set.
- `response` the actual response set. This may be either an array of uniform items or an single item, depending on the used URL pattern.

5.5.1 Long running tasks

The invocation of long running tasks is handled through the controllers in the namespace `ITB\BCN\Controllers\Workbench`. These controllers make use of the `ITB\Worker\ShellWorker` class to start a task and send it to the background, giving the user an immediate response while the task is still running. While there are user-driven long running tasks, like the *BLAST* search and the application of a *Standard Numbering Scheme* on a user-defined protein sequence, the tasks described in this section are of administrative nature and thus only available for database curators through the workbench interface described in subsection 5.6.10 on page 70. A short description of the inner workings of these tasks is described in this section.

generate_blastdb.pl collects all amino acid sequences present in the FSPD into one large *FASTA* file and uses the command `makeblastdb` from the *BLAST* software suite provided by the NCBI. This creates a *BLAST* database which can then be used to find homologues to sequences provided by the user through the *workbench* GUI or the API, described in subsection 4.2.3 on page 30.

generate_alignments.pl automatically generates multi sequence alignments for homologous and superfamilies of protein sequences within the FSPD. For this, it uses the MSA tool `clustalo` which has been described in subsection 4.2.3 on page 30.

annotate_alignments.pl creates *feature annotation* files for the use within the *Jalview* MSA viewer. For this, it reads the previously generated MSA, fetches the respective sequence annotation information from the FSPD and creates a *Jalview*-specific feature annotation file. What the results will look like, will be presented in subsection 5.6.6 on page 58.

build_tax_tree.pl and build_seq_tree.pl are used to create *binary trees* for the taxonomy and sequence-related entities of the FSPD. A *binary tree* is a data structure which enables access to hierarchy information a magnitude faster than with the *relational* data model. The drawback of *binary trees* is the effort that needs to be spent to update the structure upon insertions or deletions, which is why in the BioCatNet, both data models are used cooperatively.

BCN_donumbering_thdp.pl is the script which applies a *Standard Numbering Scheme* to a user-provided amino acid sequence using a pre-build *BLAST* database and HMM profiles.

5.6 BioCatNet website

As mentioned before, BioCatNet is publicly available at `https://biocatnet.de`. The welcome page presents news about the BioCatNet project as well as links to the different family-specific protein database build on top of BioCatNet (Figure 5.6 on page 53). The user will also find links to the BioCatNet *wiki* and *bugtracker*.

5.6.1 Wiki

The BioCatNet documentation is available at `https://wiki.biocatnet.de`. This documentation shall provide users an overview of the capabilities as well as an guide to how to use the BioCatNet. (Figure 5.7 on page 54)

5.6.2 Issues and feature requests

The bugtracker for the BioCatNet can be viewed at `https://bugs.biocatnet.de`. It lists the known issues, their severity and status, and gives users the opportunity to post issues they encounter while using the BioCatNet. (Figure 5.8 on page 55).

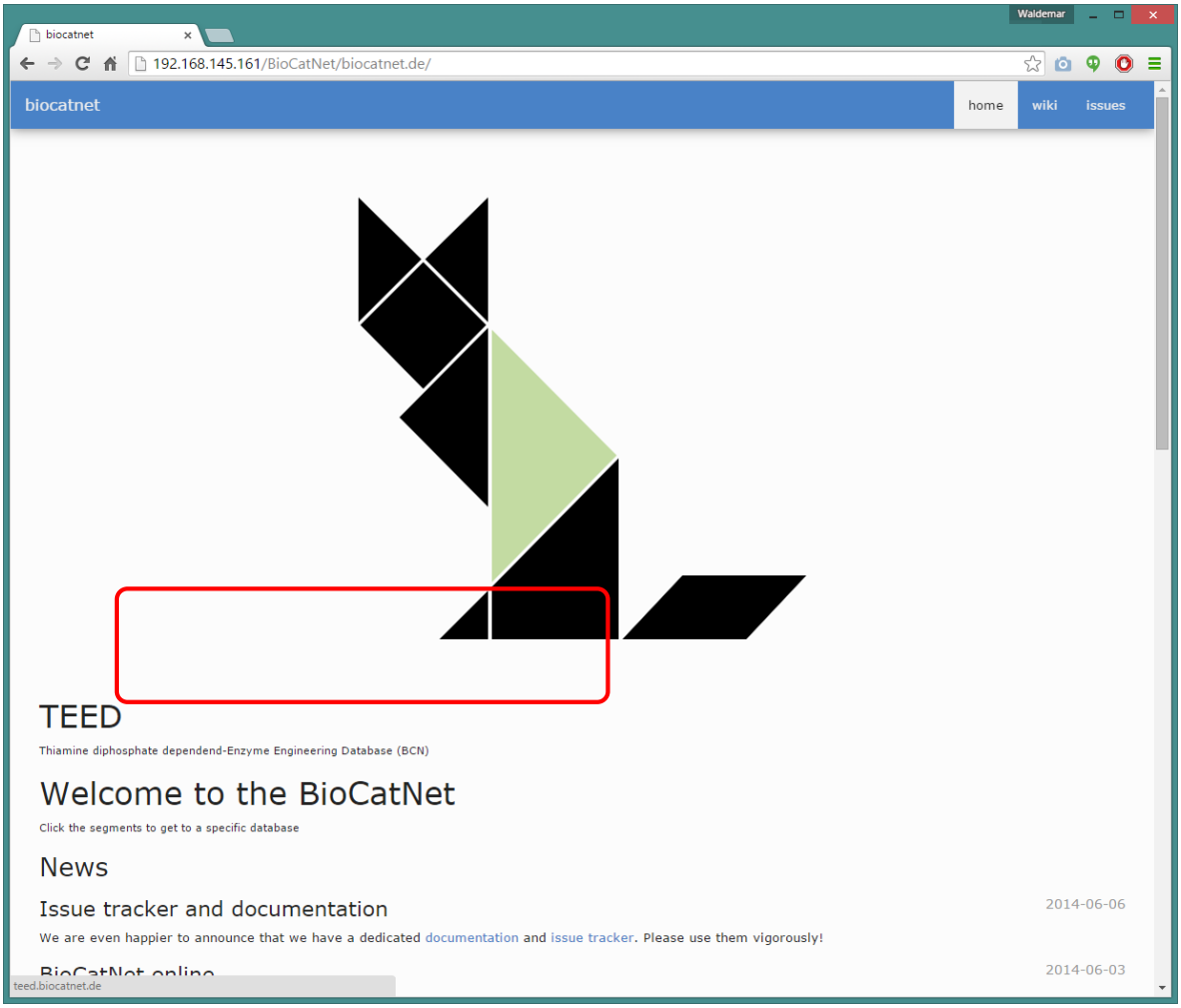


Figure 5.6: Screenshot of the BioCatNet *welcome page* as it can be seen on `http://biocatnet.de`. At the top, links to the documentation and the bug-tracker are provided. At the bottom, news from the BioCatNet project are presented. The segments of the large BioCatNet logo in the center link to the different FSPDs. Upon hovering over an fragment, the fragment changes its color and informs the user which FSPD is lying under that segment (red box), as it can be seen for the green segment, which is linking to the TEED.

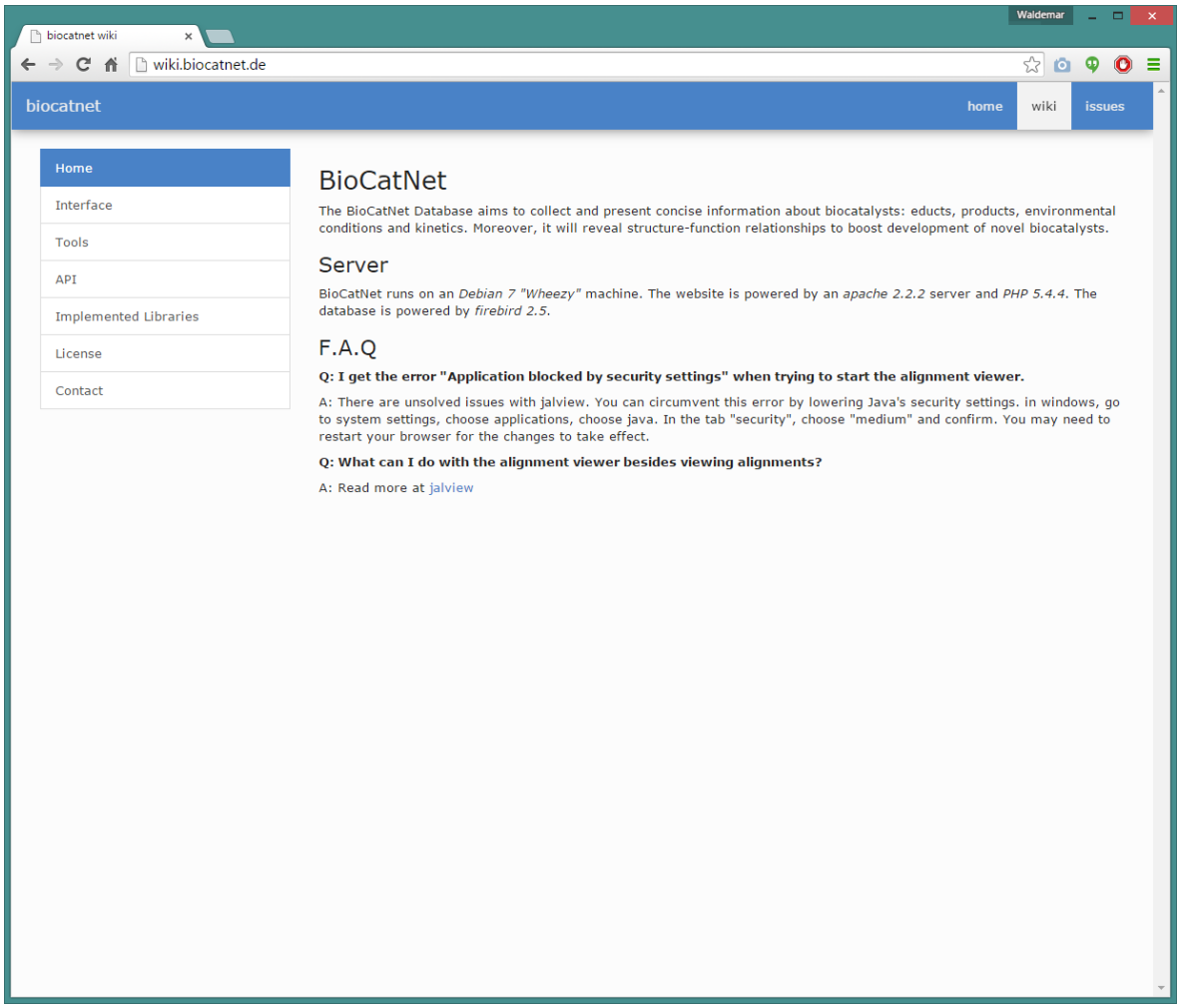


Figure 5.7: Screenshot of the BioCatNet *wiki* as it can be seen at `http://wiki.biocatnet.de`. At the top, links to the homepage and the bugtracker are provided. The menu on the left takes the user to the different documentation pages.

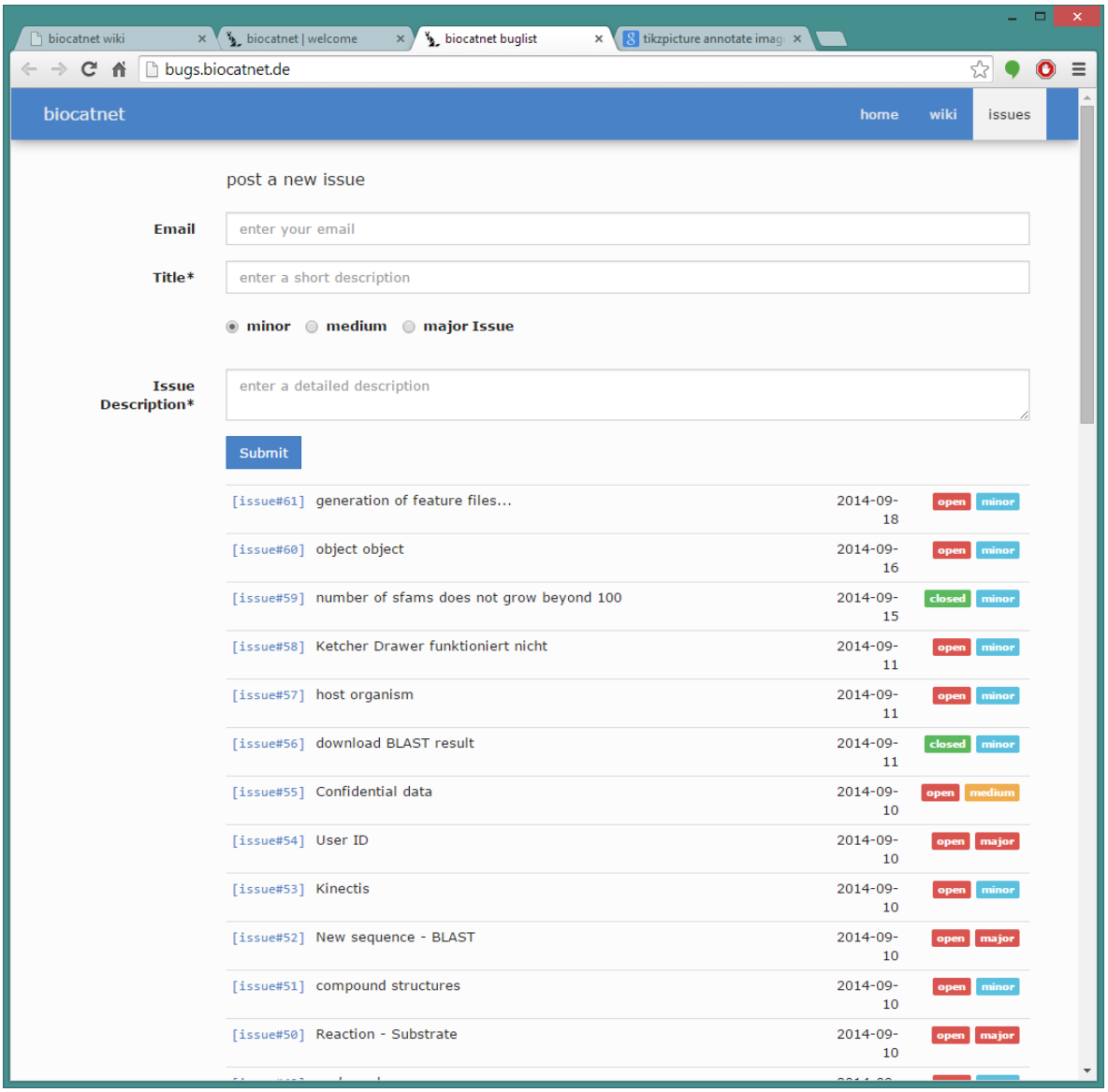


Figure 5.8: Screenshot of the BioCatNet *bugtracker* as it can be seen at `http://bugs.biocatnet.de`. At the top, links to the homepage and the documentation are provided. The from at the top gives users the opportunity to inform the developers about issues in the BioCatNet they might not be aware of. The list below the form shows a summary of known issues, their status and their severity. Each issue links to a page with the details of the issue.

5.6.3 Family-specific protein databases

As of the time of this writing, two FSPDs have been published on top of the BioCatNet platform. These are the TEED, [69] available at `https://teed.biocatnet.de` and the IRED [61], available at `http://ired.biocatnet.de`. In the following, the details of the BioCatNet web application will be described using the TEED as an example.

Upon reaching an FSPD, the user is presented with a short welcome screen specific to the FSPD (Figure 5.9 on page 57). Here, the user will find a short description of the database, the names of the curators, and a list of publications which should be cited when publishing results supported by the BioCatNet.

From here, the user can choose to browse the database by choosing a specific page, perform a search in the sequence and taxonomy database, or register as a contributer to the database.

5.6.4 Search view

When choosing the search icon from the top navigation, a search form slides down from under the navigation bar. Here, the user can choose to perform a *BLAST* search, jump quickly to a detail page, search for sequences or super/homologous families by different criteria or search for an organism. (Figure 5.10 on page 58 to Figure 5.14 on page 60).

5.6.5 Sequence browser

The *sequence browser* presents the sequences known to the FSPD in different views, representing the different hierarchical sequence kinship levels, as discussed in section 5.1 on page 37.

An overview of all known *superfamilies* is presented when the user chose the *Sequence* tab from the top menu. Below a short statistical overview, presented as doughnut diagrams, the superfamilies are listed along with some detailed parameters. From here, the user can choose to print the overview, perform a search, sort the *superfamily* list by different criteria, download the list, or display one superfamily in more detail. (Figure 5.9 on page 57)

The print view is available throughout every level of the *sequence* and *taxonomy browser* via the print icon (🖶). As the BioCatNet is optimized to printed output, one can simply use the usual *print* command from the browser, too.

The superfamily and homologous family views are opened when the user selects a specific superfamily or homologous family. Similarly to the FSPD overview page, the user will be presented with a short description of the subfamily, some statistics, and the opportunity to view or download the MSA for all the sequences of this subfamily. Below, the homologous families or proteins contained in this superfamily or homologous family are presented, respectively. (Figure 5.16 on page 62)

The superfamily and homologous family group views are opened when the user selects a family group from the overview or superfamily view pages. These pages provide

Figure 5.9: Screenshot of the BioCatNet *FSPD-specific welcome page*, as it can be seen on `https://teed.biocatnet.de` for example. (1) The top navigation allows the user to browse the database by different criteria, use BioCatNet tools on the workbench (🔧), register or log in as a contributor (👤), or perform a search (🔍). (2) Below, the database name and version are stated, (3) as well as an short description of the BioCatNet and the respective FSPD. (4) Further below, the curator of the FSPD and the related citations are presented. (5) At the far bottom, there are more links related to the BioCatNet.

no description of the family groups but only a list of superfamilies and homologous families contained in the respective group. (Figure 5.17 on page 63)

The protein view is opened when the user selects a protein from the homologous proteins view or search results. Similar to the previously described detail views, the user is presented with a short description, if available, some tasks, and a list of sequences belonging to this protein. (Figure 5.18 on page 64)

The sequence view provides extensive details about a specific protein sequence. Beside the actual amino acid sequence complete with sequence annotations extracted from the primary source database, the user is provided with links to the host organism, three dimensional structures, both documented and inferred, and documented functions. If an *EC* number is provided, documentation from the Kyoto Encyclopedia of Genes and Genomes (KEGG) is fetched and displayed as inferred functional information. (Figure 5.19 on page 65)

5.6.6 Alignment viewer - Jalview

Every super- and homologous family on the BioCatNet provides a set of files composed of an pre-calculated multiple sequence alignment, a distance tree and a *feature file* which can be used to visualize sequence annotations. These alignment files can be either downloaded or visualized directly on the BioCatNet website using the *Jalview light* MSA viewer applet. This applet needs the user to have an up-to-date Java installation enabled on his computer. In turn, it provides a simple and powerful way to visualize multiple sequence alignments together with phylogenetic trees and sequence annotations. On a modern workstation, alignments with more than 5000 sequences can

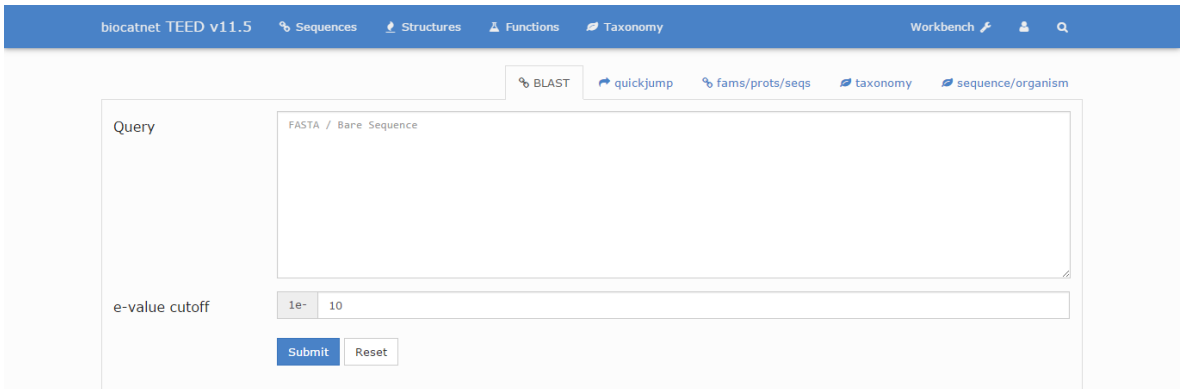


Figure 5.10: Screenshot of the BioCatNet *BLAST search* form as it can be seen when selecting the search icon (🔍) from the top navigation on `https://teed.biocatnet.de`. Providing a query sequence and a cutoff e-value, the user can perform a BLAST search. The view of the results is described in Figure Figure 5.31 on page 74.

be displayed without performance issues. (Figure 5.20 on page 66 and Figure 5.21 on page 66)

5.6.7 Structure browser

The structure browser provides an overview over all experimentally determined three dimensional structures known to the present FSPD. (Figure 5.22 on page 67) On this page the structure entries are ordered by the protein sequences they belong to and their superior hierarchies like proteins and homologous families.

The homology model view has not yet been fully implemented at the time of this writing. Though there is no summary available yet which would present all generated homology models, single models can be reached using links on the sequence detail views. The homology model presented in Figure 5.23 on page 68, for example, is a model for *sequence 147*, and can be reached from the sequence detail page `https://teed.biocatnet.de/sequence/147` using the links provided under the headline *inferred structures*. On the homology model view, the three dimensional structure is presented using the *pv.js* library. The user can choose from a number of visualization options for the homology model as well as for the template structure. Below the visualization, details of the target and template sequences are presented and various homology model analyses results are available for download.

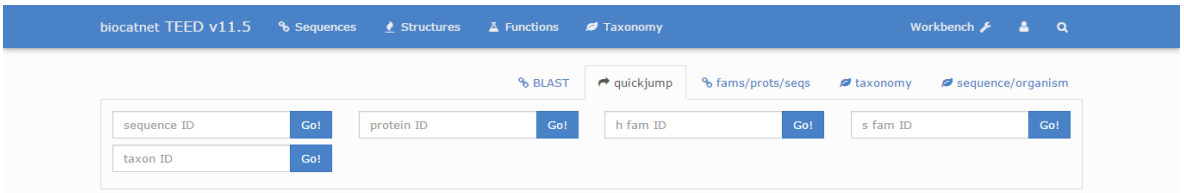


Figure 5.11: Screenshot of the BioCatNet *quickjump* form as it can be seen when selecting the search icon (🔍) from the top navigation on `https://teed.biocatnet.de` and choosing the *quickjump* link afterwards. Provided an id of a specific sequence, protein, organism, etc., the user can quickly jump to the respective details page.

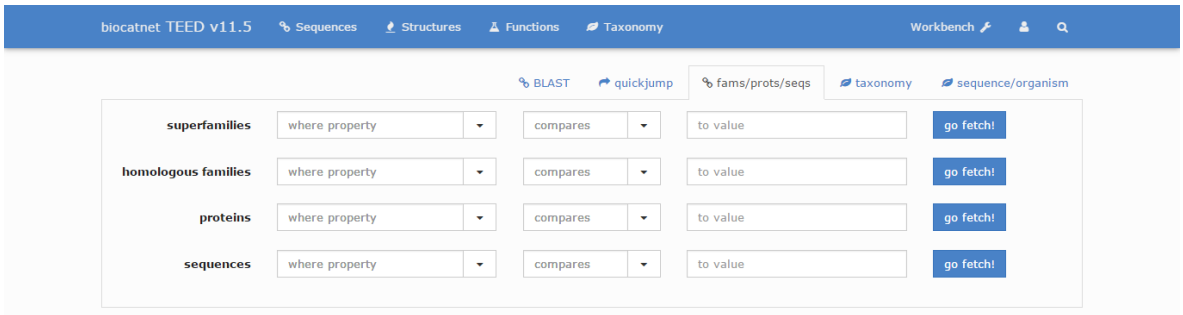


Figure 5.12: Screenshot of the BioCatNet *advanced search* form as it can be seen when selecting the search icon (🔍) from the top navigation on `https://teed.biocatnet.de` and choosing the *fams/prots/seqs* link afterwards. Here, the user can choose form a number of parameters to perform a refined search for a specific entity in the BioCatNet.

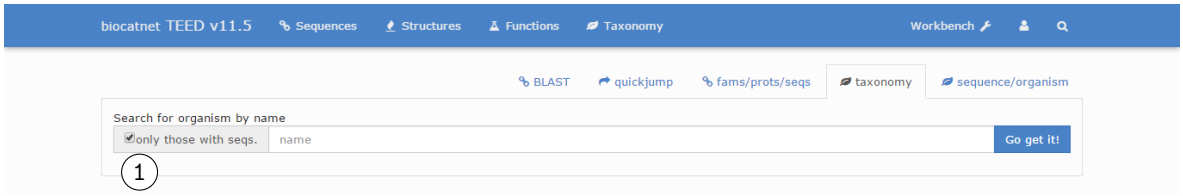


Figure 5.13: Screenshot of the BioCatNet *organism search* form as it can be seen when selecting the search icon (🔍) from the top navigation on `https://teed.biocatnet.de` and choosing the *taxonomy* link afterwards. Here, the user can search for an organism by name. Enabling the checkbox (1) filters the results to only show organisms which are known to express a protein present in this FSPD.

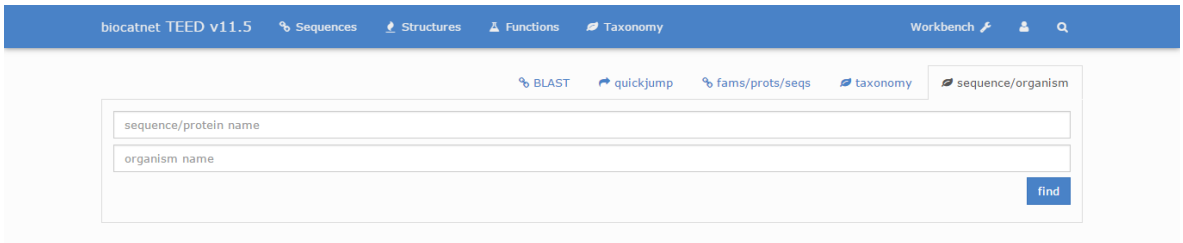


Figure 5.14: Screenshot of the BioCatNet *sequence-organism combination search* form as it can be seen when selecting the search icon (🔍) from the top navigation on `https://teed.biocatnet.de` and choosing the *sequence/organism* link afterwards. Here, the user can search for sequences that match the provided protein name as well as the provided organism name.

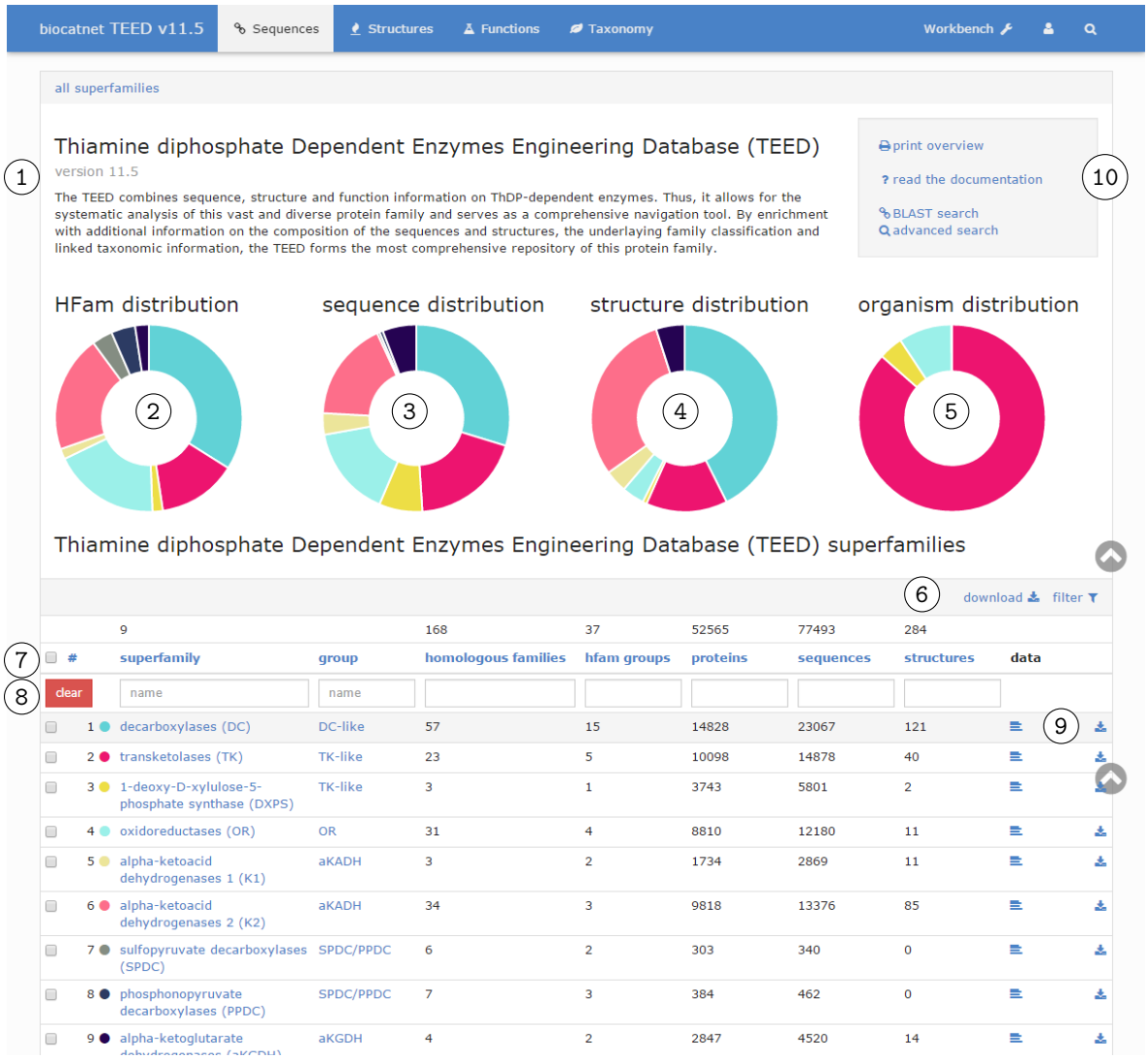


Figure 5.15: Screenshot of the protein family overview page as it can be seen on `https://teed.biocatnet.de/sequence-browser`. On this page, the user is presented (1) a description of the protein family and several simplified statistics: (2) the number of homologous families in each superfamily, (3) the number of sequences in each superfamily, (4) the number of experimentally determined structures in each superfamily and (5) the number of organisms in each kingdom of life expressing an protein from this family. (10) Some common tasks can be performed by clicking the respective links in the upper right corner of the overview. (6) The user can download the list of superfamilies as a table and (8) open a *filter row* to filter the list by different criteria. (7) A click on a column headline sorts the list by the specified criteria. (9) A click on the alignment icon (☰) opens the *Jalview* MSA viewer, a click on the download button (⬇) allows the user to download the MSA or other provided files.

biocatnet TEED v11.5 | Sequences | Structures | Functions | Taxonomy | Workbench

all superfamilies / [SFG#1] DC-like / [SF#1] decarboxylases (DC)

1 [SF#1] decarboxylases (DC)

are of the inter-monomer/PYR-PP type and form the most comprehensive and divers superfamily of ThDP-dependent enzymes. The cofactor ThDP is bound between two monomers in the homodimeric decarboxylases and each monomer consists of an N-terminal PYR domain, a subsequent TH3 domain and a C-terminal PP domain. The TH3 domain conveys FAD-binding to some homologous families of decarboxylases, but has in most homologous families lost its original nucleotid-binding function.

10 print overview
view alignment in JalView
get fasta
get alignment
get feature
get tree
get package

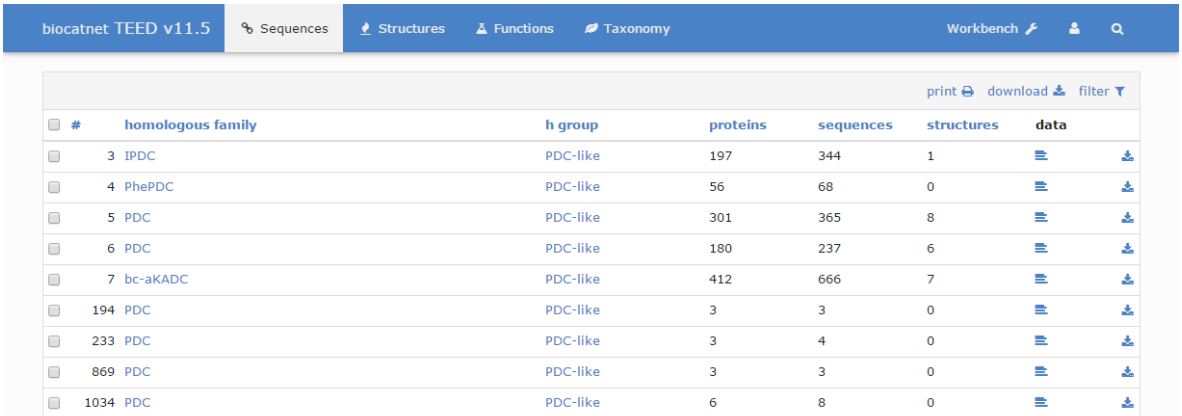
protein distribution (2) | sequence distribution (3) | structure distribution (4) | organism distribution (5)

Homologous Families of decarboxylases (DC)

6 download filter

	#	homologous family	h group	proteins	sequences	structures	data
		57	15	14828	23067	121	
8 clear		name	name				
	1	POX	POX-like	744	1338	15	9
	2	PDH cyt.	POX-like	676	1236	2	
	3	IPDC	PDC-like	197	344	1	
	4	PhePDC	PDC-like	56	68	0	
	5	PDC	PDC-like	301	365	8	
	6	PDC	PDC-like	180	237	6	
	7	bc-aKADC	PDC-like	412	666	7	
	194	PDC	PDC-like	3	3	0	
	233	PDC	PDC-like	3	4	0	
	869	PDC	PDC-like	3	3	0	
	1034	PDC	PDC-like	6	8	0	
	8	SHCHCS	SHCHC	377	796	5	
	9	SHCHCS	SHCHC	1144	1601	2	
	10	PHYLLO	SHCHC	23	27	0	

Figure 5.16: Screenshot of the superfamily details page as it can be seen on `https://teed.biocatnet.de/sFam/1`, for example. The details page for homologous families has the same structure. On these pages, the user is presented with (1) an description of the family and several simplified statistics: (2) the number of proteins in each homologous family, (3) the number of sequences in each homologous family, (4) the number of experimentally determined structures in each homologous family and (5) the number of organisms in each kingdom of life expressing an protein from this family. (10) Some common tasks can be performed by clicking the respective links in the upper right corner of the overview. (6) The user can download the list of superfamilies as a table and (8) open a *filter row* to filter the list by different criteria. (7) A click on a column headline sorts the list by the specified criteria. (9) A click on the alignment icon (☰) opens the *Jalview* MSA viewer, a click on the download button (⬇) allows the user to download the MSA or other provided files.



#	homologous family	h group	proteins	sequences	structures	data
3	IPDC	PDC-like	197	344	1	
4	PhePDC	PDC-like	56	68	0	
5	PDC	PDC-like	301	365	8	
6	PDC	PDC-like	180	237	6	
7	bc-aKADC	PDC-like	412	666	7	
194	PDC	PDC-like	3	3	0	
233	PDC	PDC-like	3	4	0	
869	PDC	PDC-like	3	3	0	
1034	PDC	PDC-like	6	8	0	

Figure 5.17: Screenshot of the homologous family groups details page as it can be seen on `https://teed.biocatnet.de/homologousfamilies/?h_fam_group_id=2`, for example. The structure basically corresponds to the list from the superfamily and homologous family views. The page for superfamily groups has the same structure.

biocatnet TEED v11.5 | Sequences | Structures | Functions | Taxonomy | Workbench

4 all superfamilies / [SFG#1] DC-like / [SF#1] decarboxylases (DC) / [HFG#1] POX-like / [HF#1] POX / [P#30] pyruvate oxidase

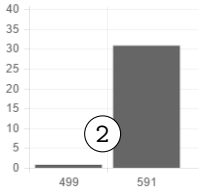
1 [P#30] pyruvate oxidase (EC:1.2.3.3 pyruvate oxidase)

no detailed description

3 print overview
get fasta file
BRENDA/1.2.3.3
biocatnet/1.2.3.3

sequence length distribution

2



pyruvate oxidase sequences

download filter

32 | 0

#	sequence	length	data	source	structures	accession
45	pyruvate oxidase	591	★	[T#28037] S. mitis (?) (ncbi)		gi|446113937
46	pyruvate oxidase	591		[T#28037] S. mitis (?) (ncbi)		gi|446113972
47	Pyruvate oxidase	591		[T#1301] Streptococcus (?) (ncbi)		gi|544714115
48	pyruvate oxidase	591		[T#1343] S. vestibularis (?) (ncbi)		gi|489183284
49	pyruvate oxidase	591		[T#1343] S. vestibularis (?) (ncbi)		gi|489187808
50	pyruvate oxidase	591		[T#68891] S. peroris (?) (ncbi)		gi|493116748
51	pyruvate oxidase	499		[T#45634] S. cristatus (?) (ncbi)		gi|491766633
52	pyruvate oxidase	591		[T#1054460] S. pseudopneumoniae (?) (ncbi)		gi|342163406
53	pyruvate oxidase	591		[T#28037] S. mitis (?) (ncbi)		gi|490372744
54	pyruvate oxidase	591		[T#28037] S. mitis (?) (ncbi)		gi|527097067
55	pyruvate oxidase	591		[T#1303] S. oralis (?) (ncbi)		gb|AAC69578.1 gi|3818594
56	pyruvate oxidase	591		[T#1095727] S. sp. (?) (ncbi)		gi|446113968
57	pyruvate oxidase	591		[T#365659] S. mitis (?) (ncbi)		gi|289168264
58	pyruvate oxidase	591		[T#1302863] S. oligofermentans (?) (ncbi)		gi|488650688
59	pyruvate oxidase, partial|pyruvate oxidase	591		[T#1313] S. pneumoniae (?) (ncbi)		gb|AAL75572.1 gi|18542397

5 6 7 8 9

Figure 5.18: Screenshot of the protein details page as it can be seen on `https://teed.biocatnet.de/protein/30`, for example. On this page, the user is presented with (1) an description of the protein, if available and (3) tasks the user can perform with this protein. If an *EC* number is defined for this protein, the user can choose to look up more information on BRENDA or get an list of all proteins with this *EC* known to the current FSPD. (2) A small statistic about the lengths of sequences belonging to this protein is displayed, too. (4) The breadcrumbs at the top describe the families this protein belongs to. The sequences belonging to the displayed protein are listed below. Here, the user can see (5) the sequence name, (6) length, (7) the source organism and (8) has the possibility to look up the primary information on the source database. (9) In the *data* column the user has direct access to the amino acid sequence in *FASTA* format and the reference sequence of this protein is marked with a star (★).

biocatnet TEED v11.5 | Sequences | Structures | Functions | Taxonomy | Workbench

all superfamilies / [SFG#1] DC-like / [SF#1] decarboxylases (DC) / [HFG#1] POX-like / [HF#1] POX / [P#30] pyruvate oxidase / [S#45] pyruvate oxidase

1 [S#45] pyruvate oxidase ([P#30] pyruvate oxidase)

★ this is the reference sequence for its parent protein

2 print overview
copy FASTA to clipboard
dowload FASTA
biocatnet/1.2.3.3
BRENDA/1.2.3.3
ExplorEnz/1.2.3.3
KEGG/1.2.3.3
gi|446113937
2013-11-21 00:00:00
standard positions

Sequence

```
>sid|45|pid|30|hfid|1|sfid|1|gi|446113937|taxid|28037|
MTQGKITASA AMLNVLKTWG VDTIYGIPSG TLSSLMDALA EDKDIRFLQV RHEETGALAA   60
VMQAKFGGSI GVAVGSGGPG ATHLINGVYD AAMDNTPFLA ILGSRPVNEL NMDAFQELNQ  120
NPMYHGVAVY NKRVAYAEQL PKVIDEACRA AVSKKGPAVV EIPVNFGFQE IDENSYYGSG  180
SYERSFIAPA LNEVEINKAV EILNNAERPV IYAGYGGVKA GEVITELSRK IKAPIITTGK  240
NFEAFEWNYE GLTGSAYRVG WKPANEVVFE ADTVLFLGSN FPFAEVYEAF KNTEKFIQVD  300
IDPYKLGKRH ALDASILGDA GQAAKAILDK VNPVESTPWW RANVKNNQNW RDYMNKLEGK  360
TEGELQLYQV YNAINKHADQ DAIYSIDVGD TTQTSTRILH MTPKNMWRTS PLFATMGIAL  420
PGGIAAKKDM PDRQVWNIMG DGAFNMCYPD VITNVQYDLP VINVVFSNGK YAFIKDKYED  480
TNKHLFGCDF PNADYAKIAE AQGAVGFTVD RIEDIDAVVA EAVKLNKEGK TVVIDARISQ  540
HRPLPVEVLE LDPKLHSEEA IKAFKEKYEA EELVPFRLFL EEEGLQSRAI K
```

3 [anno#141] PYR
[anno#142] PP

Sources

4 [T#28037] Streptococcus mitis (?) (ncbi) 5 gi|446113937

Documented Structures

There are no experimentally determined three-dimensional structures for this sequence yet.

Inferred Structures (by homology modelling)

6

model	template	GA341
[HM#8878]	pdb|2DJI (chains: A) [S#1017] Chain A, Crystal Structure Of Pyruvate Oxidase Containing Fad, From Aerococcus Viridans|Chain A, Crystal Structure Of Pyruvate Oxidase Complexed With Fad And Tpp, From Aerococcus Viridans [T#1377] Aerococcus viridans (?) (ncbi)	1.000000
[HM#8879]	pdb|2DJI (chains: A) [S#1017] Chain A, Crystal Structure Of Pyruvate Oxidase Containing Fad, From Aerococcus Viridans|Chain A, Crystal Structure Of Pyruvate Oxidase Complexed With Fad And Tpp, From Aerococcus Viridans [T#1377] Aerococcus viridans (?) (ncbi)	1.000000

Documented Functions

No experiments with this protein sequence have been posted yet.

Inferred Functions

Source	Educt	Product	Pathways
KEGG rn:R00207	Pyruvate + Orthophosphate + Oxygen	Acetyl phosphate + Hydrogen peroxide + CO2	Pyruvate metabolism Metabolic pathways

Figure 5.19: Screenshot of the sequence details page as it can be seen on `https://teed.biocatnet.de/sequence/45`, for example. (1) As with the other details pages, the user is presented with a short summary. A star (★) marks a sequence as being the primary reference sequence for the protein. (2) To the right, the user can choose to copy the amino acid sequence to the clipboard or download the corresponding *FASTA* file. If an *EC* number is given for the protein, links to other enzyme databases with more information on the respective *EC* number are provided. Also, links to the primary source database entries are provided. (3) The amino acid sequence is displayed, colored by available annotation data. Hovering over an amino acid of the sequence will present a small popup with the position and, if available, standard position of the respective amino acid, as well as potential annotation information. (4) The source organisms are displayed below the amino acid sequence together with (5) links to the primary database entry. (6) Below, documented and inferred structures and functions are presented, with links to pages containing more information.

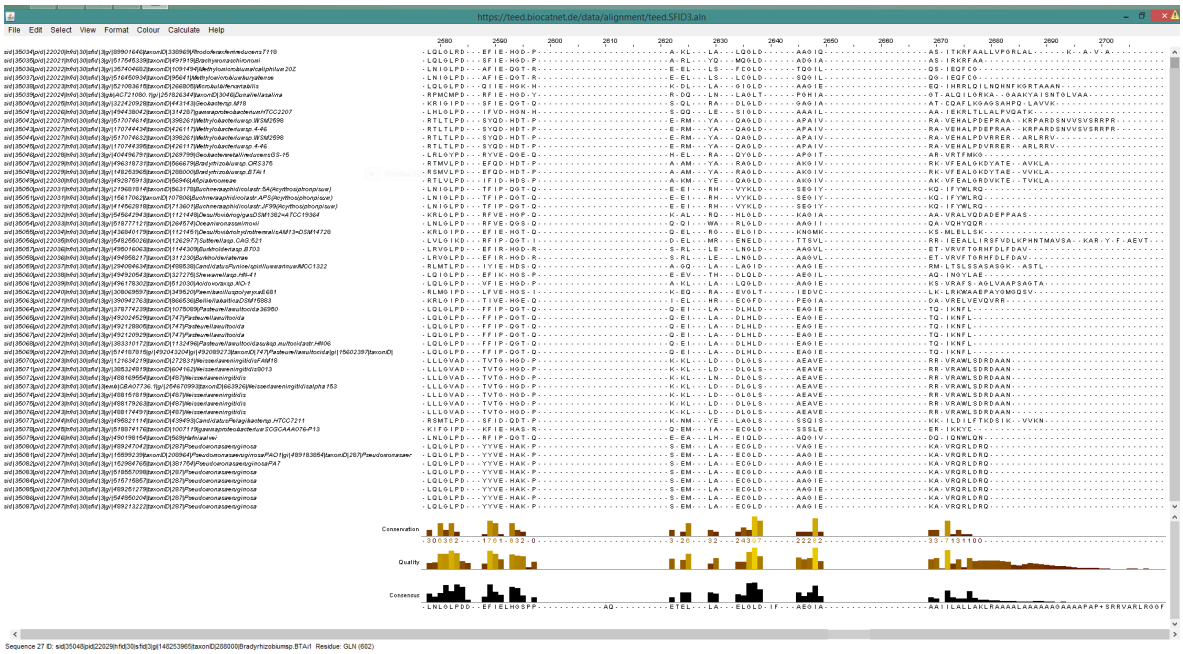


Figure 5.20: Example of an MSA visualization with Jalview. The displayed alignment is an MSA for `superfamily 3` of the TEED FSPD.

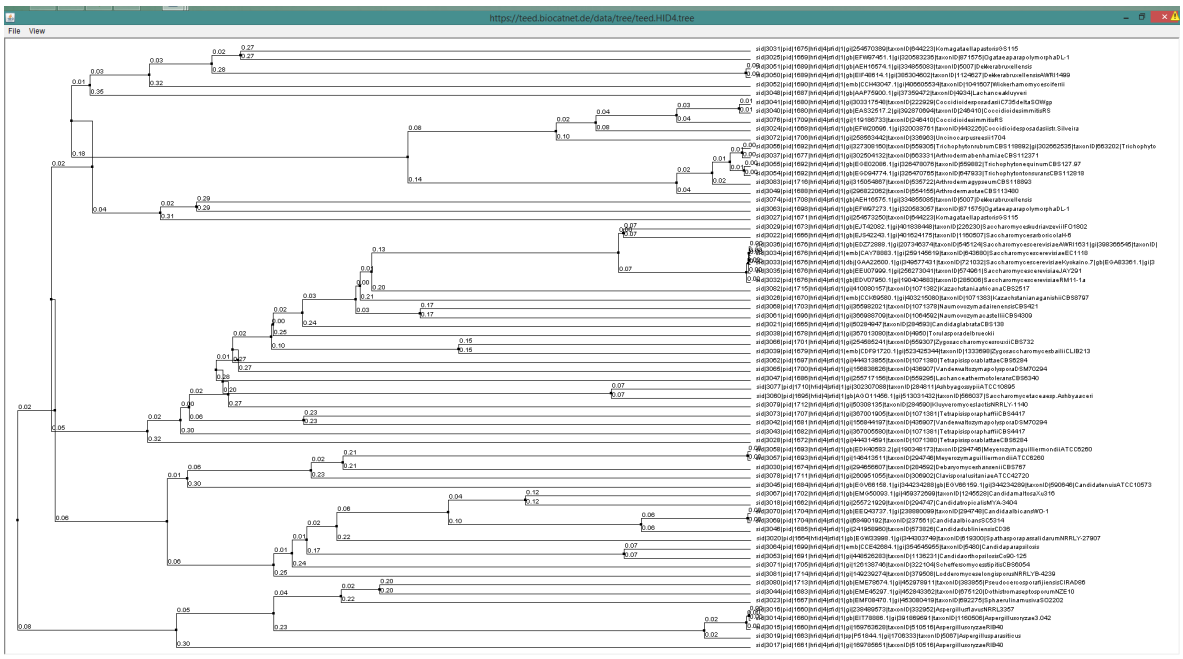


Figure 5.21: Example of an MSA phylogenetic tree visualization with Jalview. The tree displayed portrays the sequence distances between members of `homologous family 4` of the TEED FSPD.

biocatnet TEED v11.5 | Sequences | Structures | Functions | Taxonomy | Workbench | Waldemar Reusch

structures — print download

SFam	HFam	Protein	Sequence	Source	PDBEntries
[SF#1] decarboxylases (DC)	[HF#1] POX	[P#41] pyruvate oxidase	[S#148] Chain A, The Refined Structures Of A Stabilized Mutant And Of Wild-Type Pyruvate Oxidase From Lactobacillus Plantarum	[T#1590] L. plantarum (?) (ncbi)	1POW (A, B)
			[S#149] Chain A, The Refined Structures Of A Stabilized Mutant And Of Wild-Type Pyruvate Oxidase From Lactobacillus Plantarum	[T#1590] L. plantarum (?) (ncbi)	1POX (A, B)
			[S#153] pyruvate oxidase	[T#644042] L. plantarum (?) (ncbi)	4FEE (A, B)
					4FEG (A, B)
					4KGD (A, B)
			[S#154] Chain A, Pyruvate Oxidase Variant V265a From Lactobacillus Plantarum	[T#1590] L. plantarum (?) (ncbi)	1Y9D (A, B, C, D)
			[S#155] Chain A, Pyruvate Oxidase Variant F479w	[T#1590] L. plantarum (?) (ncbi)	2EZ4 (A, B)
					2EZ8 (A, B)
					2EZ9 (A, B)
					2EZT (A, B)
					2EZU (A, B)
		[P#570] pyruvate oxidase	[S#1017] Chain A, Crystal Structure Of Pyruvate Oxidase Containing Fad, From Aerococcus Viridans|Chain A, Crystal Structure Of Pyruvate Oxidase Complexed With Fad And Tpp, From Aerococcus Viridans	[T#1377] A. viridans (?) (ncbi)	1V5F (A)
					1V5E (A)
					2DJI (A)
			[S#90795] Chain A, Crystal Structure Of The Reaction Intermediate Between Pyruvate Oxidase Containing Fad And Tpp, And Substrate Pyruvate	[T#1377] A. viridans (?) (ncbi)	1V5G (A)
	[HF#2] PDH cyt.	[P#897] pyruvate dehydrogenase	[S#1715] Chain A, Structural Basis For Membrane Binding And Catalytic Activation Of The Peripheral Membrane Enzyme Pyruvate Oxidase From Escherichia Coli|pyruvate dehydrogenase (pyruvate oxidase), thiamin-dependent, FAD-binding|pyruvate dehydrogenase	[T#562] Escherichia coli (?) (ncbi)	3EY9 (A, B)
					3EYA (A, B, C, D, E, F, G, H, I, J, K, L)
	[HF#3] IPDC	[P#1494] indolepyruvate decarboxylase	[S#2715] RecName: Full=Indole-3-pyruvate decarboxylase; Short=Indolepyruvate decarboxylase	[T#550] E. cloacae (?) (ncbi)	1OVM (A, B, C, D)

Figure 5.22: Screenshot of the structure browser as it can be see on `https://teed.biocatnet.de/structure-browser`. The structure entries are ordered by superfamily, homologous family, protein, sequence and host organism.

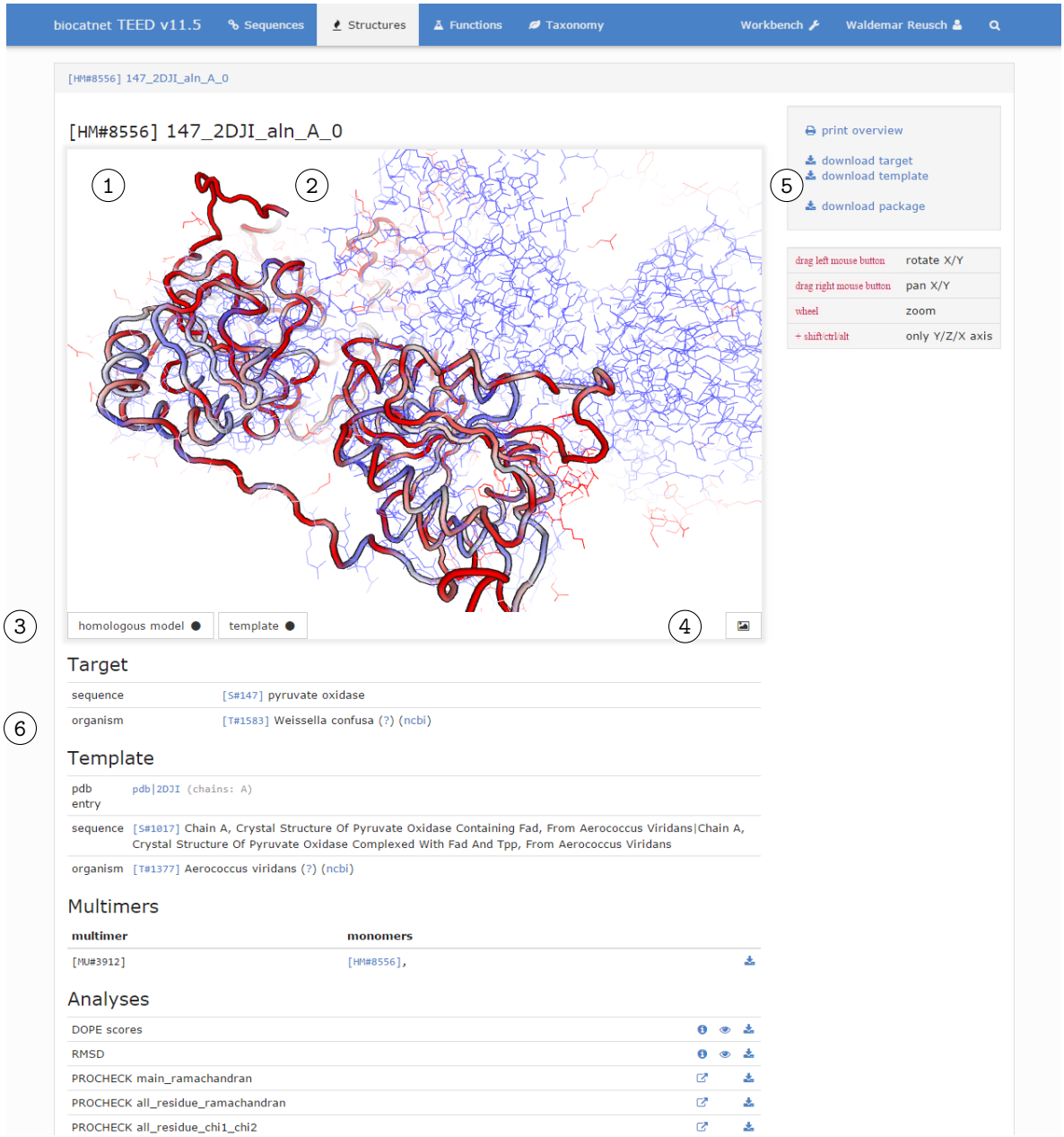


Figure 5.23: Screenshot of the homology model viewer as it can be seen on `https://teed.biocatnet.de/modelMonomer/8556`. (1) At the very top, the homology model is visualized. (2) By default, the model template is not shown, but it can be visualized using (3) the switches at the bottom of the visualization. There, the user can also choose from a number of visualization and coloring options. In this example, the template is visualized using sticks and the model is represented by tubes. (4) A click on the picture icon (🖼) at the lower right allows the user to save a snapshot of the current view. (5) Further to the right, the user can choose to download the structure files of the homology model, its template or an package containing all structure files and analyses related to this homology model. (6) Below the visualization, links related to the target and template sequences are presented as well as information and results on the performed model analyses.

5.6.8 Functions browser

The *functions browser* shall presents protein functions connected to unambiguous amino acid sequence entries, the relations of which have been discussed in detail in section 5.1 on page 37.

An overview of all reactions which have been posted to the BioCatNet is displayed when the user chooses the *Functions* tab from the topmost menu. Reactions will be listed here with their name and with their reaction formula. (Figure 5.24 on page 69)

The reaction details view will additionally depict the reaction, using the *Indigo* library, and list any experiments known to the present FSPD where this reaction has been examined. (Figure 5.25 on page 69)

The chemical compound view finally displays details about a chemical compound found in a reaction. It lists alternative names, shows a computed molecular mass and provides the possibility to download the structure of the compound. (Figure 5.26 on page 70)

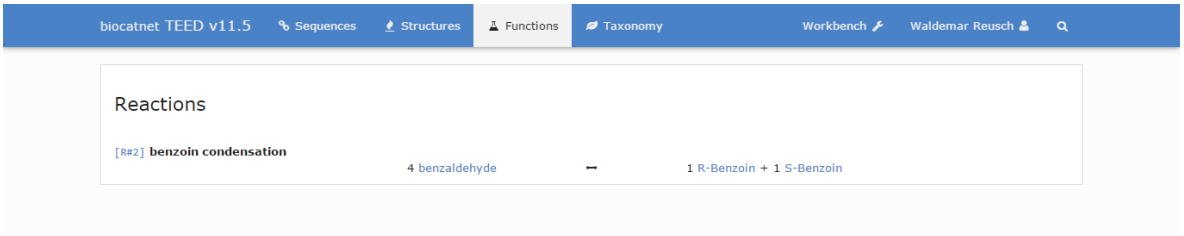


Figure 5.24: Screenshot of the functions browser as it can be seen on `https://teed.biocatnet.de/function-browser`. The view lists all reactions that have been posted to the BioCatNet with their names and reaction formulas.

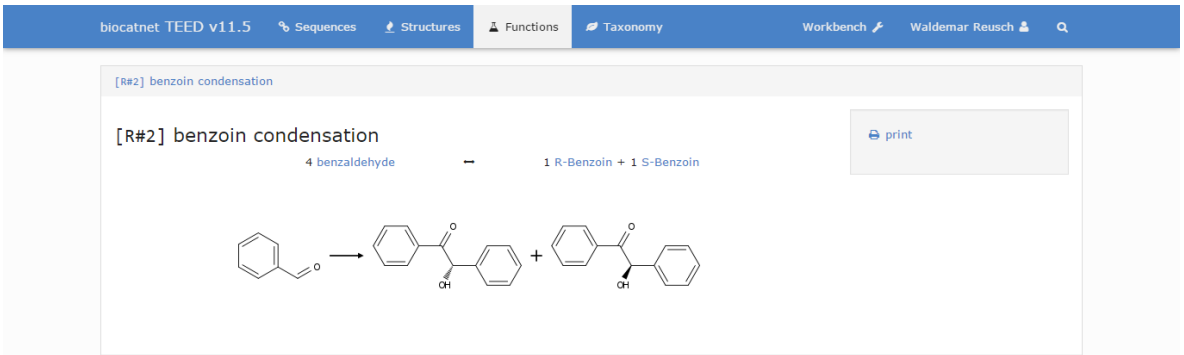


Figure 5.25: Screenshot of a reaction details view as it can be seen on `https://teed.biocatnet.de/reaction/2`.

5.6.9 Taxonomy browser

When the user clicks on a linked organism name on any BioCatNet page, he will be presented with the taxonomy details view for the clicked organism. The *taxonomy* tab in the uppermost navigation, takes the user to the taxonomy details view for the taxonomic root node. This detail view displays the requested organism, its lineage and its sibling and child nodes in a tree view. Furthermore, it lists all sequences which are known to originate from this taxonomic node, and hints to the number of sequences each of the sibling, child and parent nodes contribute to the FSPD.

5.6.10 Workbench

As the name implies, the *Workbench* is the place BioCatNet users can make use of the tools they are provided with and contribute to the database by posting experiment data. The workbench provides different tools depending whether the user is a first-time visitor, a registered collaborator or even an database curator. (Figure 5.28 on page 72)

The workbench stash provides a way for users to collect frequently used entities, and store them like *bookmarks* or *favorites* on various other software and websites.

The workbench cache on the other hand automatically stores unfinished user processes. When, for example, a user looses his internet connection or accidentally closes

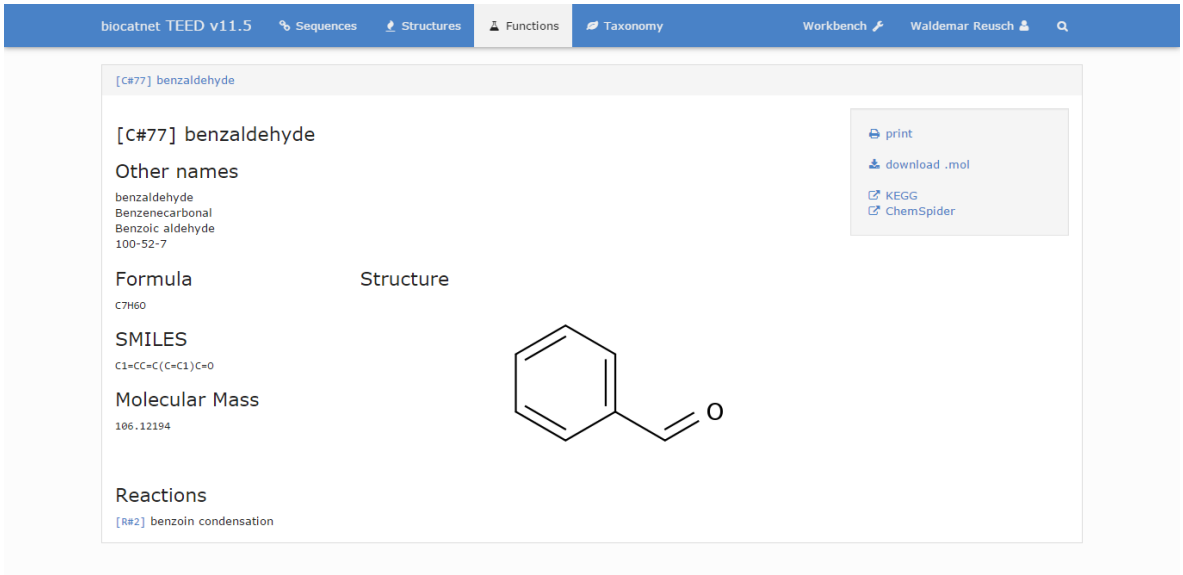


Figure 5.26: Screenshot of a compound details view as it can be seen on `https://teed.biocatnet.de/compound/77`. The view presents alternative names as well as the computed molar mass for the current compound and depicts the chemical structure as well as the *SMILES* code.

biocatnet TEED v11.5 | Sequences | Structures | Functions | Taxonomy | Workbench | Waldemar Reusch

[T#1239] phylum Firmicutes

1 [T#1239] phylum Firmicutes

Bacillus/Clostridium group, Clostridium group firmicutes, Firmacutes, Firmicutes, Firmicutes corrig. Gibbons and Murray 1978, Gram positive bacteria, Gram-positive bacteria, Low G+C firmicutes, clostridial firmicutes, firmicutes, low G+C Gram-positive bacteria, low GC Gram+

print overview
NCBI/1239
show own sequences
download own sequences
show all sequences
download all sequences

Lineage

5

2 \ [T#1] root (?) (NCBI) (0 > 79085)
\ [T#131567] no rank cellular organisms (?) (NCBI) (0 > 79017)
\ [T#2] superkingdom Bacteria (?) (NCBI) (4 > 68411)
\
3 | [T#1117] phylum Cyanobacteria (?) (NCBI) (0 > 1488)
| [T#1224] phylum Proteobacteria (?) (NCBI) (1 > 30775)
| [T#1297] phylum Deinococcus-Thermus (?) (NCBI) (0 > 306)
| [T#2323] unclassified Bacteria (?) (NCBI) (0 > 258)
| [T#32066] phylum Fusobacteria (?) (NCBI) (0 > 311)
| [T#40117] phylum Nitrospirae (?) (NCBI) (0 > 85)
| [T#48479] environmental samples (?) (NCBI) (0 > 249)
| [T#51290] superphylum Chlamydiae/Verrucomicrobia group (?) (NCBI) (0 > 437)
| [T#67814] phylum Caldiserica (?) (NCBI) (0 > 15)
| [T#67819] phylum Armatimonadetes (?) (NCBI) (0 > 12)
| [T#68297] phylum Dictyoglomi (?) (NCBI) (0 > 18)
| [T#68336] superphylum Bacteroidetes/Chlorobi group (?) (NCBI) (0 > 4367)
| [T#74152] phylum Elusimicrobia (?) (NCBI) (0 > 12)
| [T#131550] superphylum Fibrobacteres/Acidobacteria group (?) (NCBI) (0 > 180)
| [T#142182] phylum Gemmatimonadetes (?) (NCBI) (0 > 9)
| [T#200783] phylum Aquificae (?) (NCBI) (0 > 130)
| [T#200795] phylum Chloroflexi (?) (NCBI) (0 > 269)
| [T#200918] phylum Thermotogae (?) (NCBI) (0 > 181)
| [T#200930] phylum Deferribacteres (?) (NCBI) (0 > 43)
| [T#200938] phylum Chrysiogenetes (?) (NCBI) (0 > 7)
| [T#200940] phylum Thermodesulfobacteria (?) (NCBI) (0 > 14)
| [T#201174] phylum Actinobacteria (?) (NCBI) (0 > 11406)
| [T#203682] phylum Planctomycetes (?) (NCBI) (0 > 233)
| [T#203691] phylum Spirochaetes (?) (NCBI) (0 > 974)
| [T#508458] phylum Synergistetes (?) (NCBI) (0 > 185)
| [T#544448] phylum Tenericutes (?) (NCBI) (0 > 265)
| [T#1293497] phylum Nitrospinae (?) (NCBI) (0 > 38)
4 | [T#1239] **phylum Firmicutes (?) (NCBI) (21 > 16122)**
\
| [T#33974] unclassified Firmicutes sensu stricto (?) (NCBI) (0 > 9)
| [T#47928] environmental samples (?) (NCBI) (0 > 281)
| [T#91061] class Bacilli (?) (NCBI) (0 > 9687)
| [T#186801] class Clostridia (?) (NCBI) (0 > 5263)
| [T#455491] class Thermolithobacteria (?) (NCBI) (0 > 0)
| [T#526524] class Erysipelotrichia (?) (NCBI) (0 > 157)
| [T#909932] class Negativicutes (?) (NCBI) (0 > 725)

Sequences from this Organism

[S#98] pyruvate oxidase
[S#83343] transketolase
[S#83496] transketolase
[S#83964] transketolase
[S#84086] transketolase
[S#55251] acetoin dehydrogenase E1 component beta subunit
[S#36491] 1-deoxy-D-xylulose-5-phosphate synthase
[S#43140] uncharacterized protein

Legend
experiments present
structures present

Figure 5.27: Screenshot of a taxonomy detail view of the phylum *Firmicutes*, as it can be seen on `https://teed.biocatnet.de/taxNode/1239`. (1) Alternative names are shown dimmed below the primary and most widely recognizable name of the shown taxonomic node. Below that, the lineage is listed, (2) always starting at the *root* node. Each backslash (\) symbolizes a step down the hierarchy from *cellular organisms* to the *superkingdom Bacteria*, to the *phyli* of the latter. (3) Each pipe (|) symbolizes that the hierarchy stays on the same level, and thus marks taxonomic sibling nodes. In this case, the whole list starting from *Cyanobacteria* and ending with *Firmicutes* are sibling nodes, with the currently viewed taxonomic node *Firmicutes* marked bold. (4) The next backslash (\) marks another step down the hierarchy and thus the child nodes of the *phylum Firmicutes*. (5) Each Node is followed by three links: the question mark (?) opens a small dialog showing synonyms, the next link takes the user to the NCBI taxonomy page, and lastly, the two numbers denote the number of sequences that the corresponding node and all its child nodes contribute to the FSPD.

the browser while creating a new experiment data set, he can pick up where he left after reopening BioCatNet and choosing the unfinished process from the workbench cache.

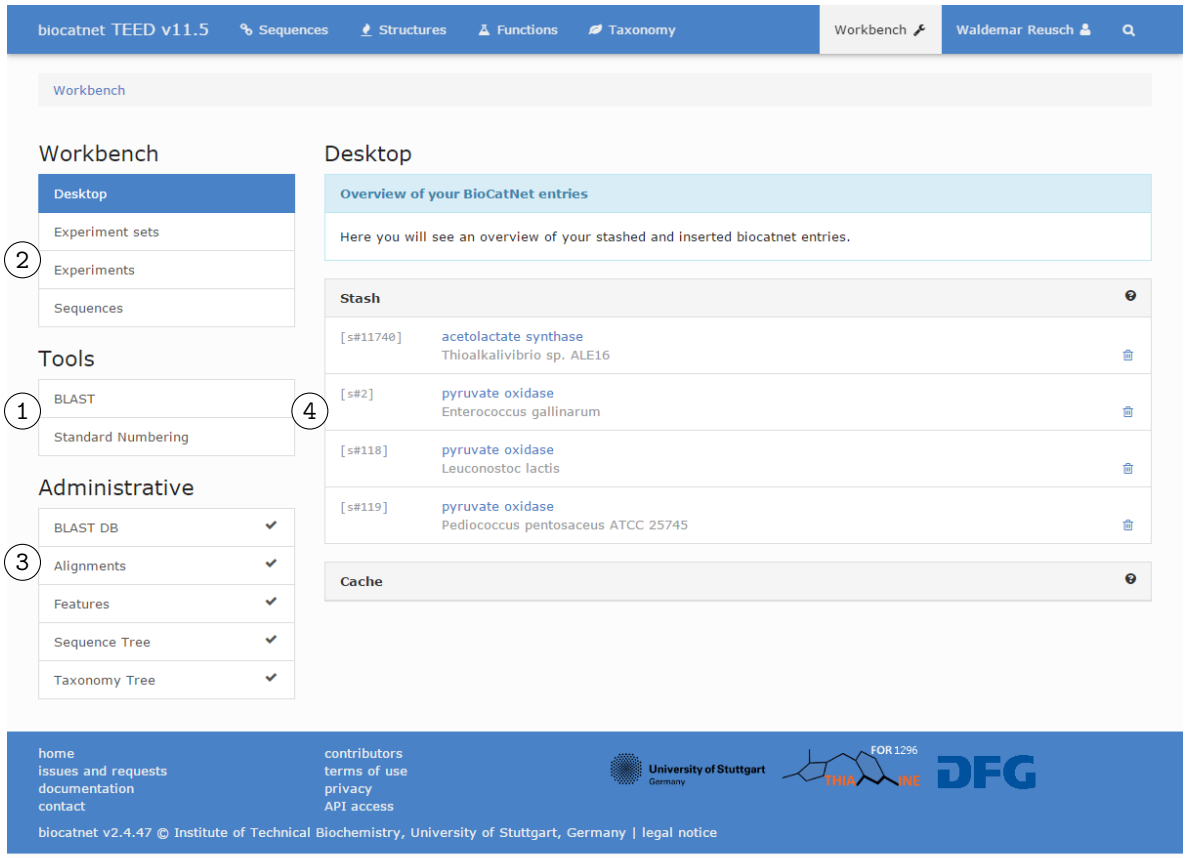


Figure 5.28: Screenshot of the BioCatNet workbench. This figure shows the workbench desktop screen for an administrator with the highest privileges. First-time visitors are presented with a short note encouraging them to register, registered users will see a desktop page much like the one shown above. (1) Unregistered users can only choose the tools BLAST and *Standard Numbering*. (2) Registered users additionally can post experiment data and sequence information. (3) Database curators and administrators are presented with a number of advanced housekeeping tools. (4) The workbench *desktop* harbors the *stash* and the *cache*.

Tools available for all users

The BLAST tool is available to every user and works the same way as the *BLAST* search available in the search forms (Figure 5.29 on page 73). Once the BLAST request is submitted, a *job status* page is presented, indicating whether the request has been submitted successfully or if it has been rejected (Figure 5.30 on page 74). The user can safely navigate away from the status page and return later, the page will automatically track the job status and display the results as soon as they are available, without the

need to refresh the status page (Figure 5.31 on page 74). The results will be available under the same address for an yet to be defined amount of time.

The Standard Numbering Tool provides users with the possibility to apply a standard numbering scheme on a query sequence, as it has been described in section 2.3 on page 7. (Figure 5.32 on page 75) Similar to the process described for the *BLAST* tool, the user will be presented a status page after the query has been submitted, where the results will be displayed as soon as they are available. (Figure 5.33 on page 76)

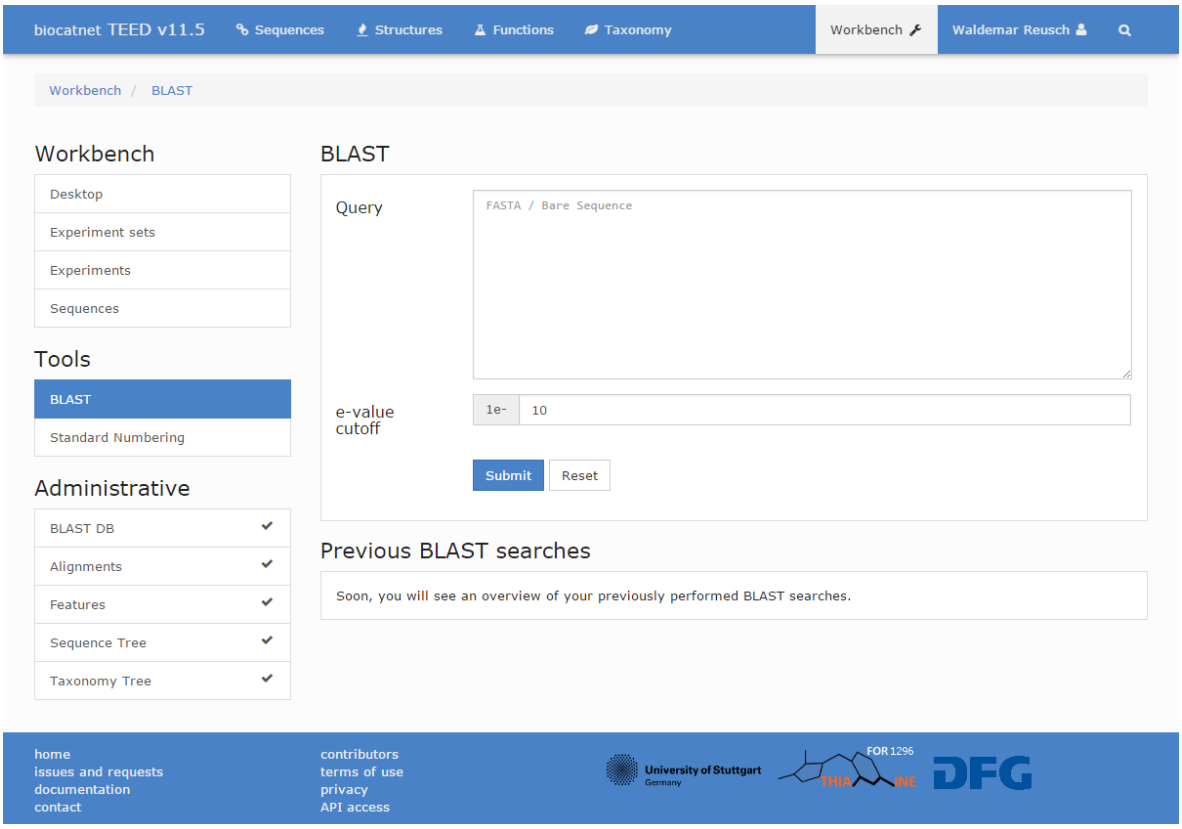


Figure 5.29: Screenshot of the BioCatNet Workbench BLAST tool.

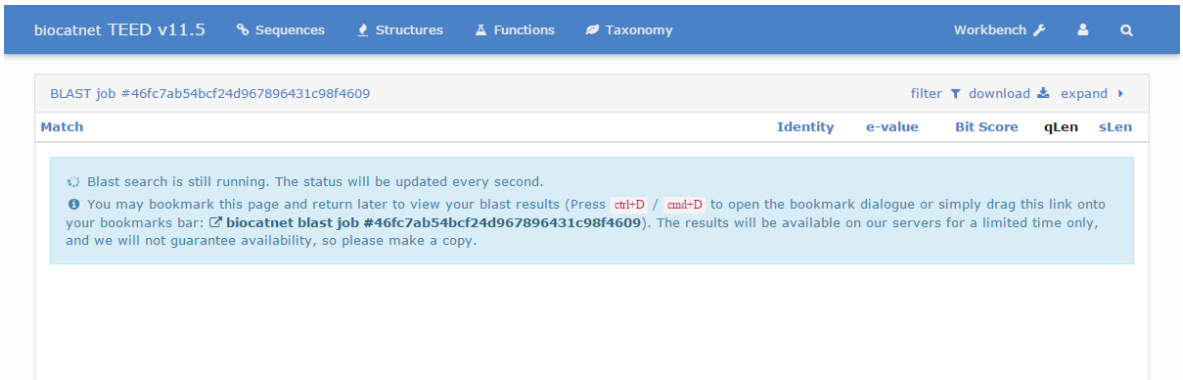
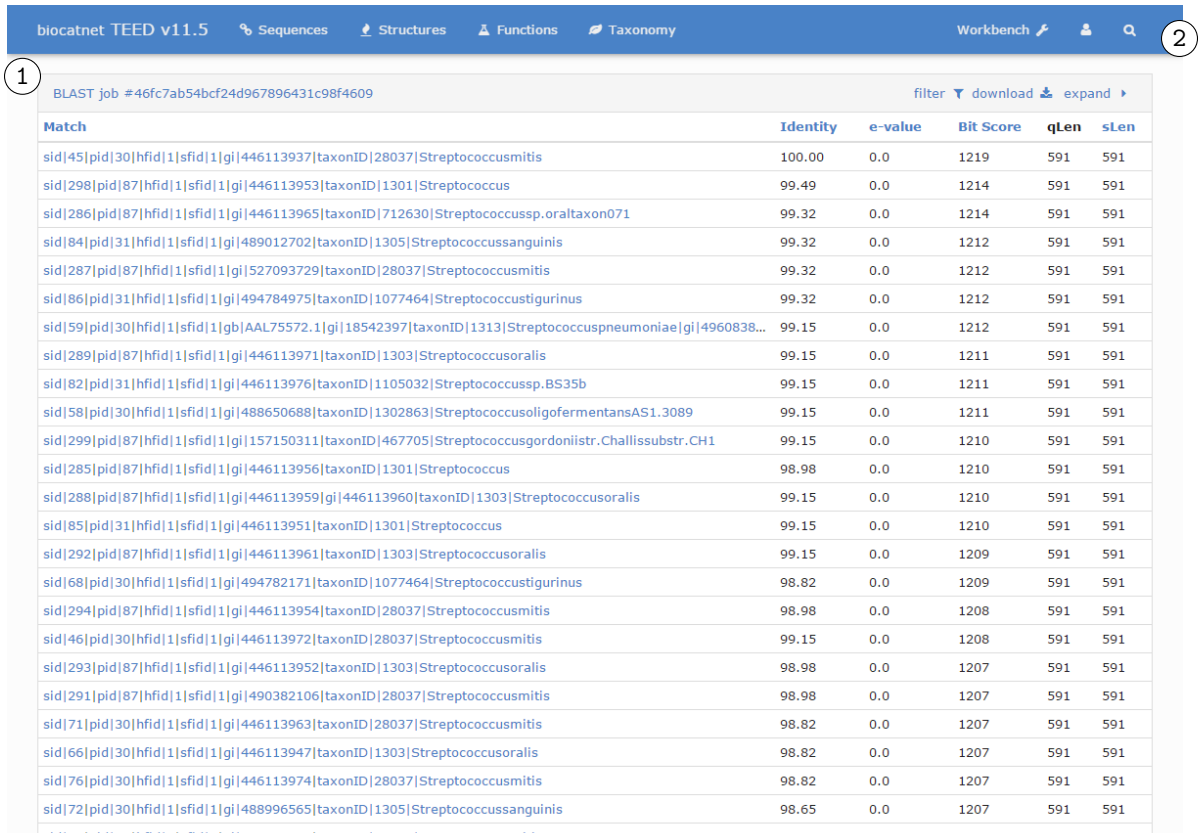


Figure 5.30: Screenshot of the BioCatNet BLAST status page while waiting for results.



Match	Identity	e-value	Bit Score	qLen	sLen
sid|45|pid|30|hfid|1|sfid|1|gi|446113937|taxonID|28037|Streptococcusmitis	100.00	0.0	1219	591	591
sid|298|pid|87|hfid|1|sfid|1|gi|446113953|taxonID|1301|Streptococcus	99.49	0.0	1214	591	591
sid|286|pid|87|hfid|1|sfid|1|gi|446113965|taxonID|712630|Streptococcussp.oraltaxon071	99.32	0.0	1214	591	591
sid|84|pid|31|hfid|1|sfid|1|gi|489012702|taxonID|1305|Streptococcussanguinis	99.32	0.0	1212	591	591
sid|287|pid|87|hfid|1|sfid|1|gi|527093729|taxonID|28037|Streptococcusmitis	99.32	0.0	1212	591	591
sid|86|pid|31|hfid|1|sfid|1|gi|494784975|taxonID|1077464|Streptococcustigurinus	99.32	0.0	1212	591	591
sid|59|pid|30|hfid|1|sfid|1|gb|AAL75572.1|gi|18542397|taxonID|1313|Streptococcuspneumoniae|gi|4960838...	99.15	0.0	1212	591	591
sid|289|pid|87|hfid|1|sfid|1|gi|446113971|taxonID|1303|Streptococcusoralis	99.15	0.0	1211	591	591
sid|82|pid|31|hfid|1|sfid|1|gi|446113976|taxonID|1105032|Streptococcussp.BS35b	99.15	0.0	1211	591	591
sid|58|pid|30|hfid|1|sfid|1|gi|488650688|taxonID|1302863|StreptococcusoligofermentansAS1.3089	99.15	0.0	1211	591	591
sid|299|pid|87|hfid|1|sfid|1|gi|157150311|taxonID|467705|Streptococcusgordoniistr.Challissubstr.CH1	99.15	0.0	1210	591	591
sid|285|pid|87|hfid|1|sfid|1|gi|446113956|taxonID|1301|Streptococcus	98.98	0.0	1210	591	591
sid|288|pid|87|hfid|1|sfid|1|gi|446113959|gi|446113960|taxonID|1303|Streptococcusoralis	99.15	0.0	1210	591	591
sid|85|pid|31|hfid|1|sfid|1|gi|446113951|taxonID|1301|Streptococcus	99.15	0.0	1210	591	591
sid|292|pid|87|hfid|1|sfid|1|gi|446113961|taxonID|1303|Streptococcusoralis	99.15	0.0	1209	591	591
sid|68|pid|30|hfid|1|sfid|1|gi|494782171|taxonID|1077464|Streptococcustigurinus	98.82	0.0	1209	591	591
sid|294|pid|87|hfid|1|sfid|1|gi|446113954|taxonID|28037|Streptococcusmitis	98.98	0.0	1208	591	591
sid|46|pid|30|hfid|1|sfid|1|gi|446113972|taxonID|28037|Streptococcusmitis	99.15	0.0	1208	591	591
sid|293|pid|87|hfid|1|sfid|1|gi|446113952|taxonID|1303|Streptococcusoralis	98.98	0.0	1207	591	591
sid|291|pid|87|hfid|1|sfid|1|gi|490382106|taxonID|28037|Streptococcusmitis	98.98	0.0	1207	591	591
sid|71|pid|30|hfid|1|sfid|1|gi|446113963|taxonID|28037|Streptococcusmitis	98.82	0.0	1207	591	591
sid|66|pid|30|hfid|1|sfid|1|gi|446113947|taxonID|1303|Streptococcusoralis	98.82	0.0	1207	591	591
sid|76|pid|30|hfid|1|sfid|1|gi|446113974|taxonID|28037|Streptococcusmitis	98.82	0.0	1207	591	591
sid|72|pid|30|hfid|1|sfid|1|gi|488996565|taxonID|1305|Streptococcussanguinis	98.65	0.0	1207	591	591

Figure 5.31: Screenshot of the BioCatNet BLAST status page with results. (1) A click on a header in the result table will sort the results by that respective column. (2) The user has also the ability to filter and download the results, and to expand the table to display all columns returned by *BLAST*.

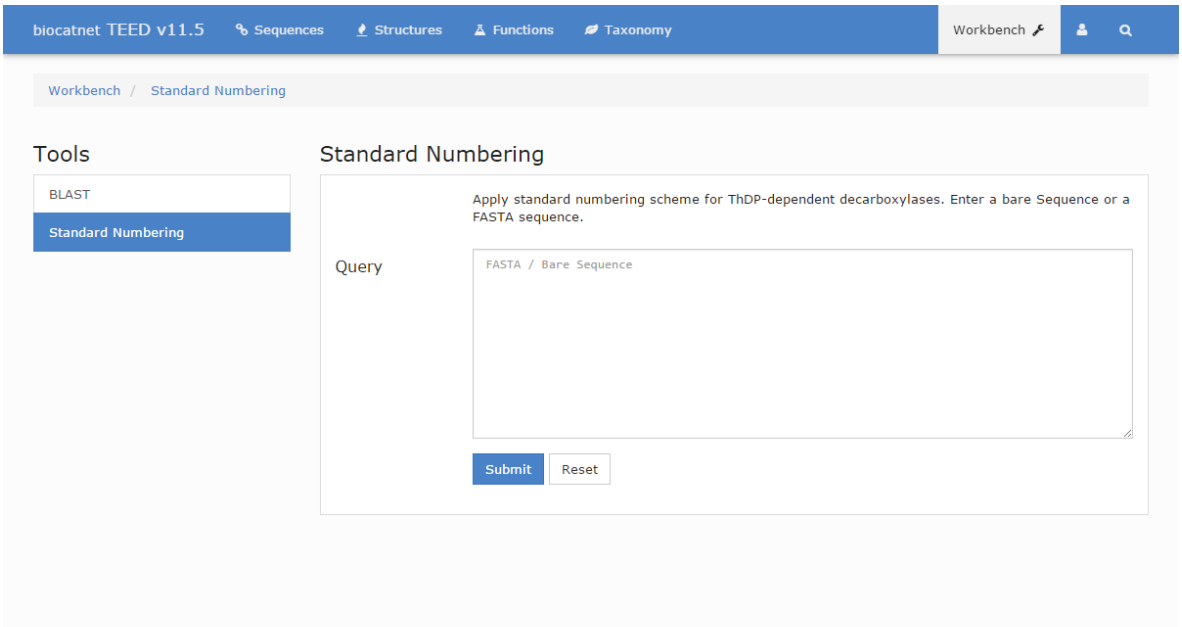


Figure 5.32: Screenshot of the BioCatNet *standard numbering* tool, where the user posts a query sequence to be aligned with the reference sequence.

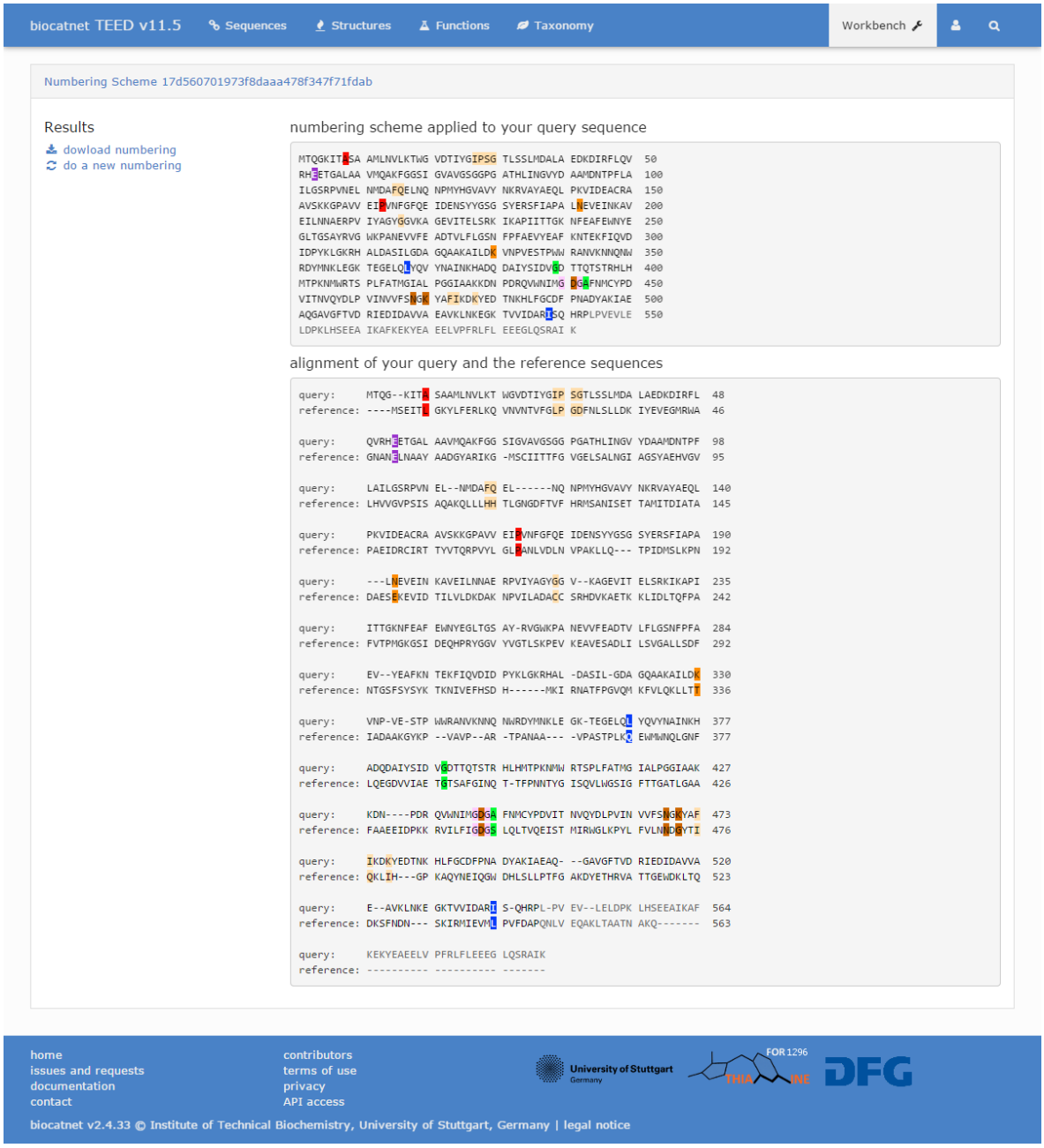


Figure 5.33: Screenshot of the BioCatNet *standard numbering* result page. Functionally and structurally relevant annotations are highlighted with colors. Hovering the cursor over an amino acid reveals its native position number and the *standard position* number it has been assigned by the *standard numbering scheme*.

Tools for registered users

Registered users have additional views to choose from when visiting the workbench, one for each entity they will be able to post to the BioCatNet. At the moment these are limited to `EXPERIMENT_SETS`, `EXPERIMENTS` and `SEQUENCES`. These pages are similarly structured, giving an overview of entities the user has created, and the ones that are being shared with him by other contributors, and the opportunity to create and post novel entities. The *experiments* view additionally lists uncompleted entries from the *cache*. (Figure 5.34 on page 77)

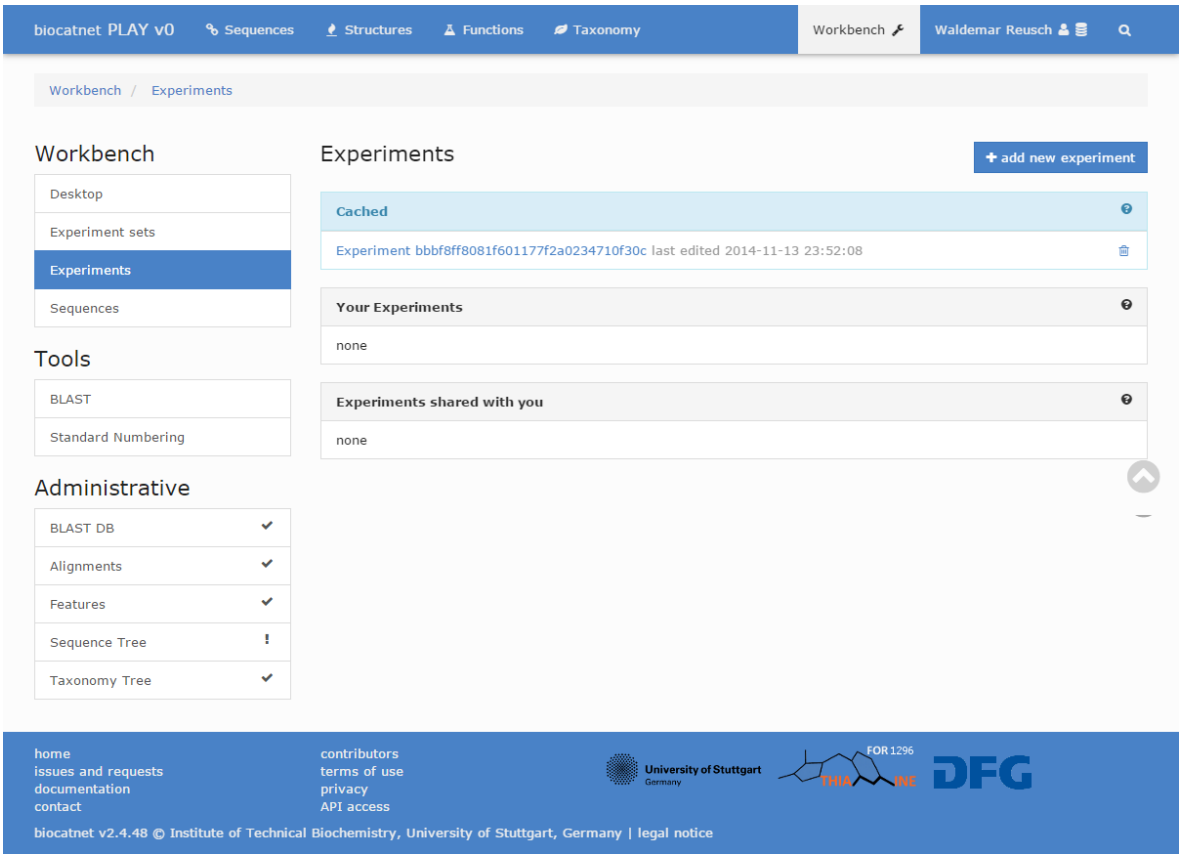


Figure 5.34: Screenshot of the BioCatNet experiments overview page. At the top right, the user has the possibility to create a new experiment. Below that, `EXPERIMENT` entities are shown which are unfinished, owned by the user and shared with the user.

The creation of an experiment set is a simple process, only requiring an descriptive name for the set. (Figure 5.35 on page 78)

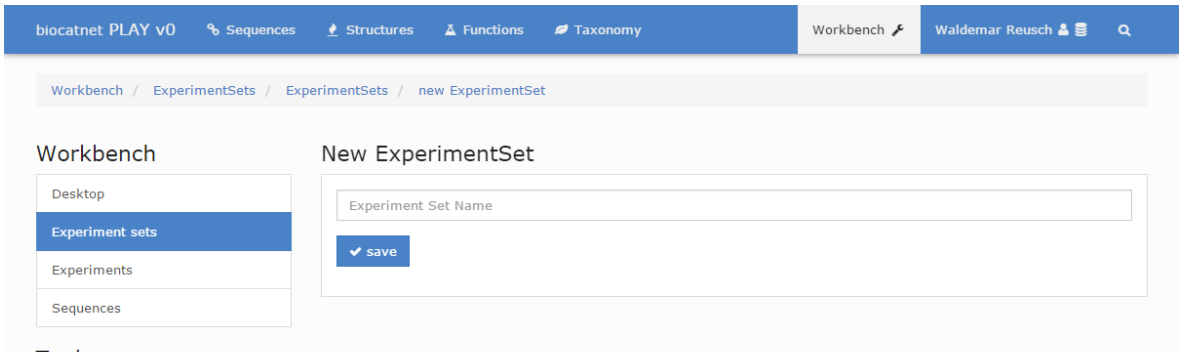


Figure 5.35: Screenshot of the BioCatNet workbench experiment set creation form. To create a new experiment set the user only needs to provide a descriptive name.

To insert a novel sequence into the FSPD, the user starts with providing the one unambiguous protein description: its amino acid sequence (Figure 5.36 on page 78). The sequence is then compared against the present sequence database using *BLAST* to find the proper protein, homologous family or superfamily to assign the new sequence to. The user then proceeds to complete missing categorization information as well as the source organism(s) (Figure 5.37 on page 79). If an identical sequence has been found in the database, on the other hand, the user can only choose to add the sequence to his stash. (section 5.6.10 on page 77)

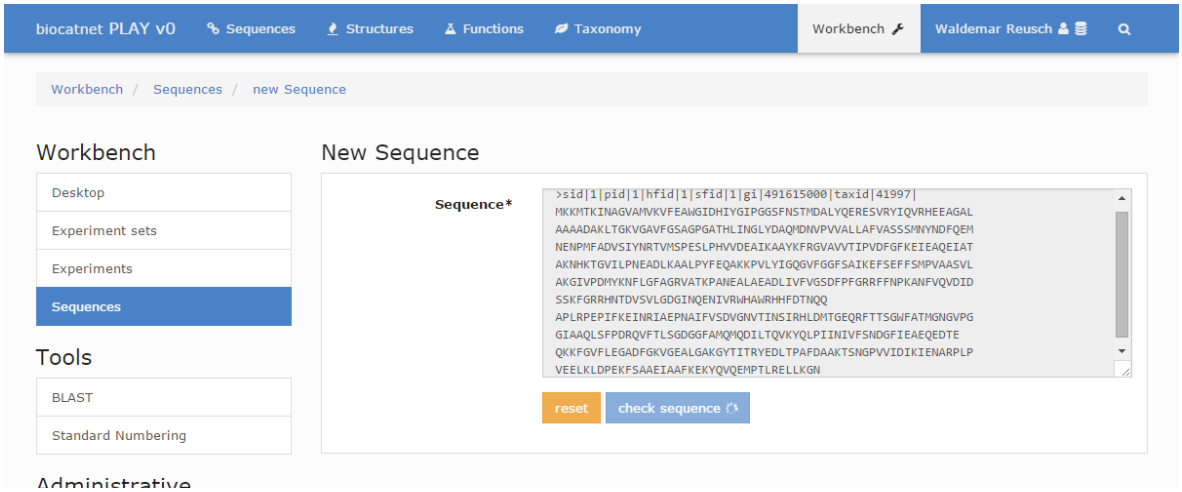


Figure 5.36: Screenshot of the first step of the sequence creation form. Here, the user starts by providing the unambiguous amino acid sequence. On submission, a BLAST search is performed against the sequence database to find related proteins that may already be present in the FSPD.

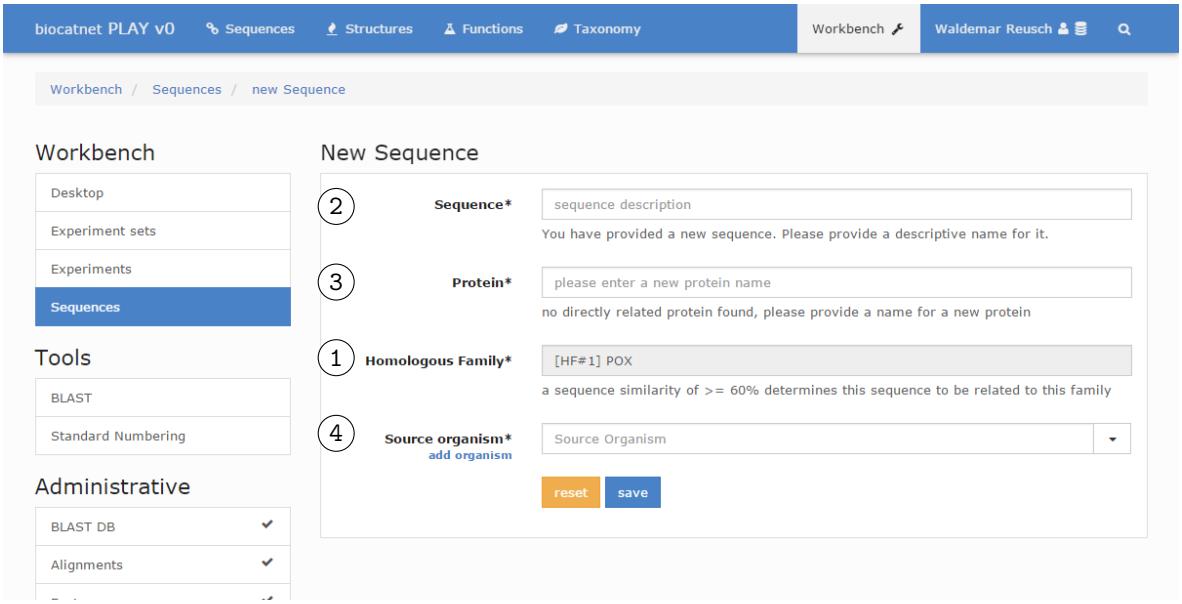


Figure 5.37: Screenshot of the second step of the sequence creation form. Here, the user is presented with the result of the sequence check. In this example, the provided sequence could not be assigned to a known protein, (1) but was found to be part of the `homologous family 1 POX`. The user now needs to provide (2) a descriptive sequence and (3) protein name, (4) as well as one or multiple source organisms.

Creating experiments is a slightly more elaborate process guided by multiple forms, and starts with the selection of the `EXPERIMENT_SET` it will belong to and an descriptive name (Figure 5.38 on page 80). The user can choose one of his previously defined experiment sets, or create a new one by simply choosing a unique name. Then he proceeds to define the reaction type and conditions, the amount of enzyme, substrates and additives he used, and the products he observed (Figure 5.39 on page 81 to Figure 5.43 on page 85). Often the user will encounter drop-down menus next to the form fields, indicated by downward facing carets (▼). Other fields will have buttons carrying a plus sign (✚), indicating that the user is supposed to create new entities if he cannot find what he is looking for in the drop-down menu. A click on these buttons will unveil small forms, where the user can define a new reaction type, buffer or compound, for example (Figure 5.45 on page 87 to Figure 5.46 on page 87).

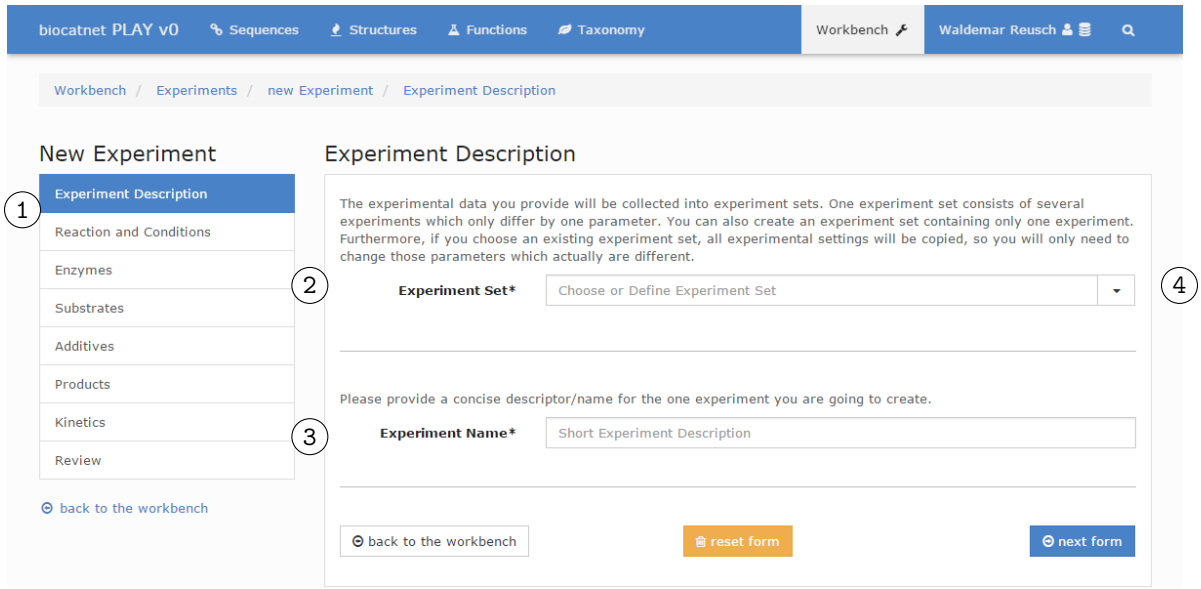


Figure 5.38: Screenshot of the first step in the creation of a new experiment entity. (1) To the left, an overview of the forms involved in the creation of the experiment are presented as links and the user can jump between the forms at any given time, though some forms need the previous form to be complete to be visible. In the first form, the user needs to choose (2) an experiment set and (3) an experiment name. (4) Using the drop-down toggle (▼), the user can choose from a list of his previously defined experiment sets, or simply fill in a new name to create a new `EXPERIMENT_SET`.

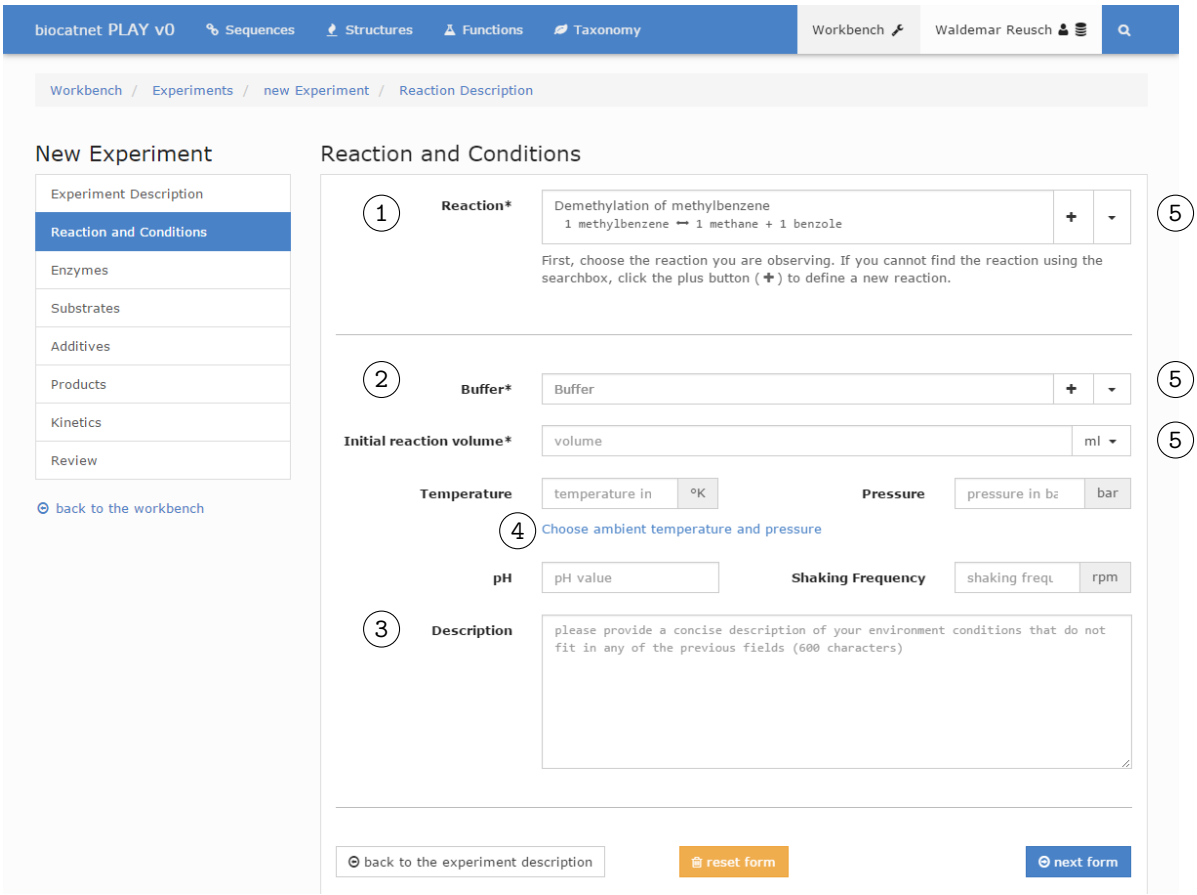


Figure 5.39: Screenshot of the second step second step in the experiment creation form. Here, the user must choose (1) the reaction he is observing, in terms of supposed substrates and products, and (2) various reaction conditions. (3) Additional information about the reaction set up can be provided using the text form field, (4) ambient temperature and pressure can be filled in using a shortcut. (5) Using the drop-down toggles (▼), he can choose the reaction type and used buffer from a list of already defined entities. Similarly, he can choose a volume unit from a drop-down list. In case the needed reaction or buffer are not already defined, the user can open smaller forms using the 'add' buttons next to the form fields (✚). (Figure 5.45 on page 87 and Figure 5.46 on page 87)

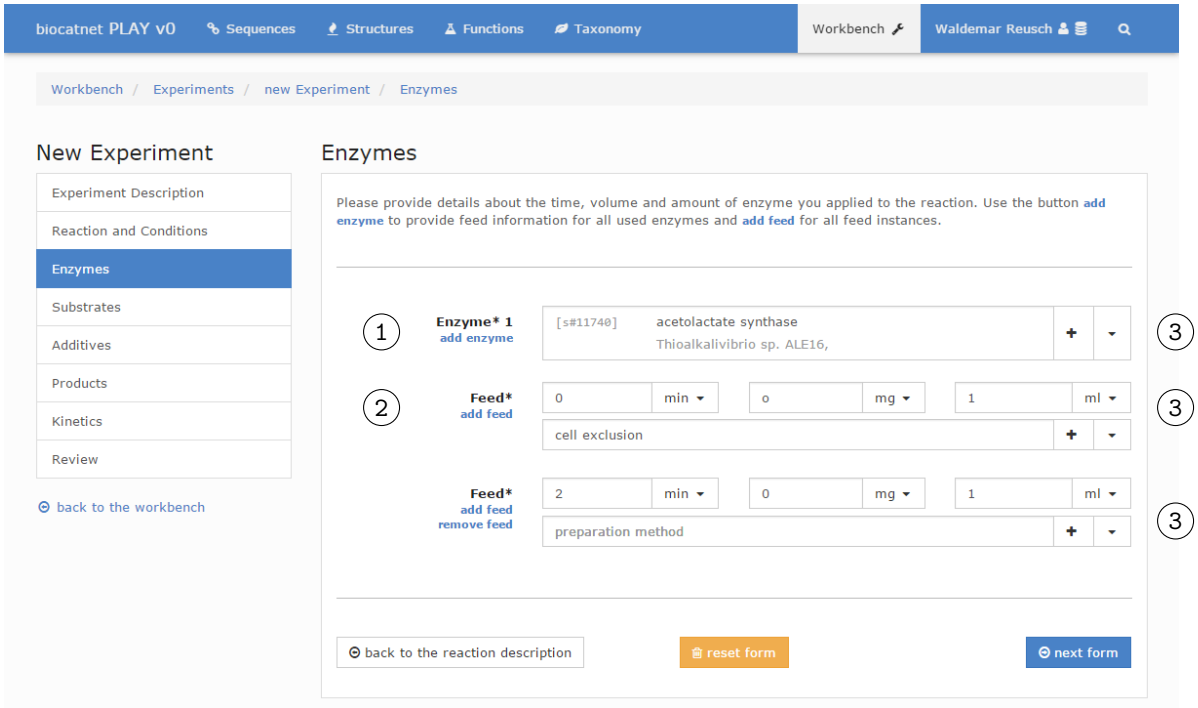


Figure 5.40: Screenshot of the third step in the experiment creation form. Here, the user must choose (1) at least one enzyme he used in the reaction as well as (2) at least one feed instance, i.e. combination of feed time, feed amount, the volume of added solvent, if any, and the method used to prepare the enzyme. Using the shortcuts `add/remove enzyme` and `add/remove feed`, he can control the number of form elements and thus the number of enzymes and feed instances. (3) Using the drop-down toggles (▼), the user can choose from a list of enzymes, preparation methods and time/amount/volume units. While the list of preparation methods and units is common to all users, the list of sequences is particular to the user and reflects the users stash of sequences (section 5.6.10 on page 77). Using the 'add' buttons (✚), the user can add new enzymes to his stash and create novel buffer entries, which he than can use to fill the form.

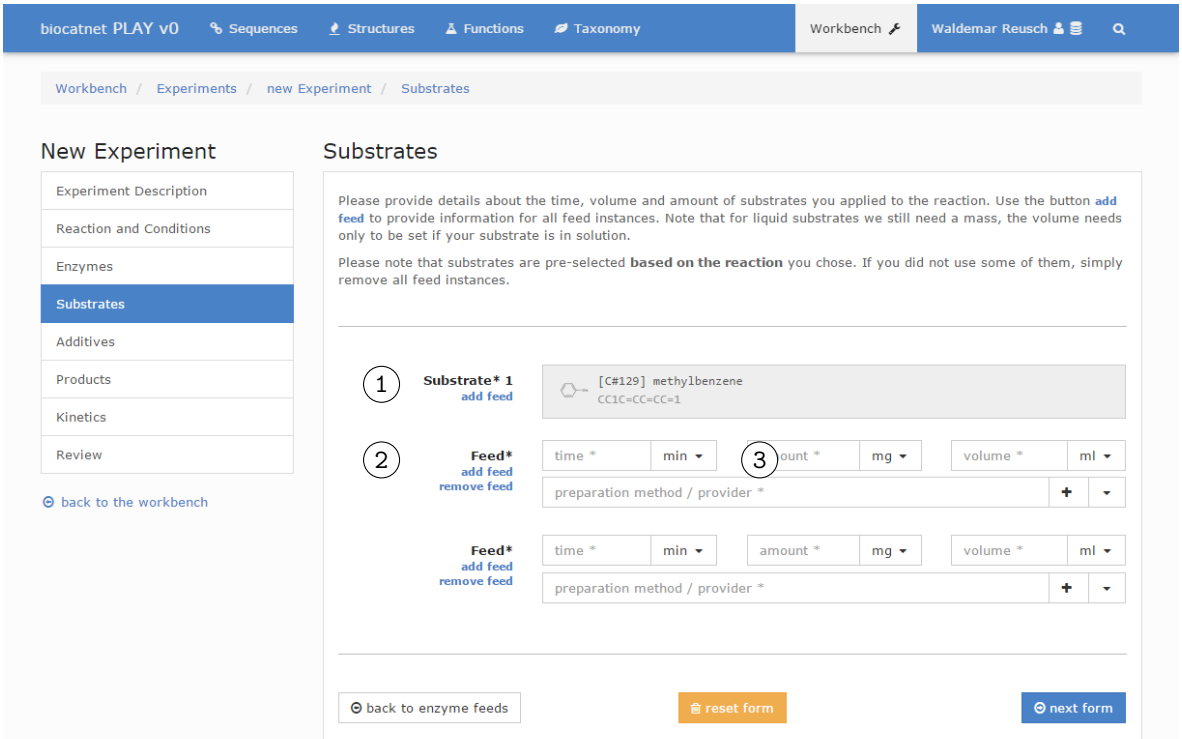


Figure 5.41: Screenshot of the fourth step in the experiment creation form. Here, the user must declare which substrates were added to the reaction. (1) The substrates are pre-selected based on the reaction the user chose on the second form 'Reaction and Conditions'. (2) Using the hyperlinks 'add feed' and 'remove feed', he can control the number of feed instances. (3) Each feed instance consists of a feed time, feed amount and feed volume as well as a description of the method – or supplier – by which the substrate was acquired. In the case that a user has not used a substrate suggested by the reaction, he can simply remove all feed instances and thus mark the substrate as not used.

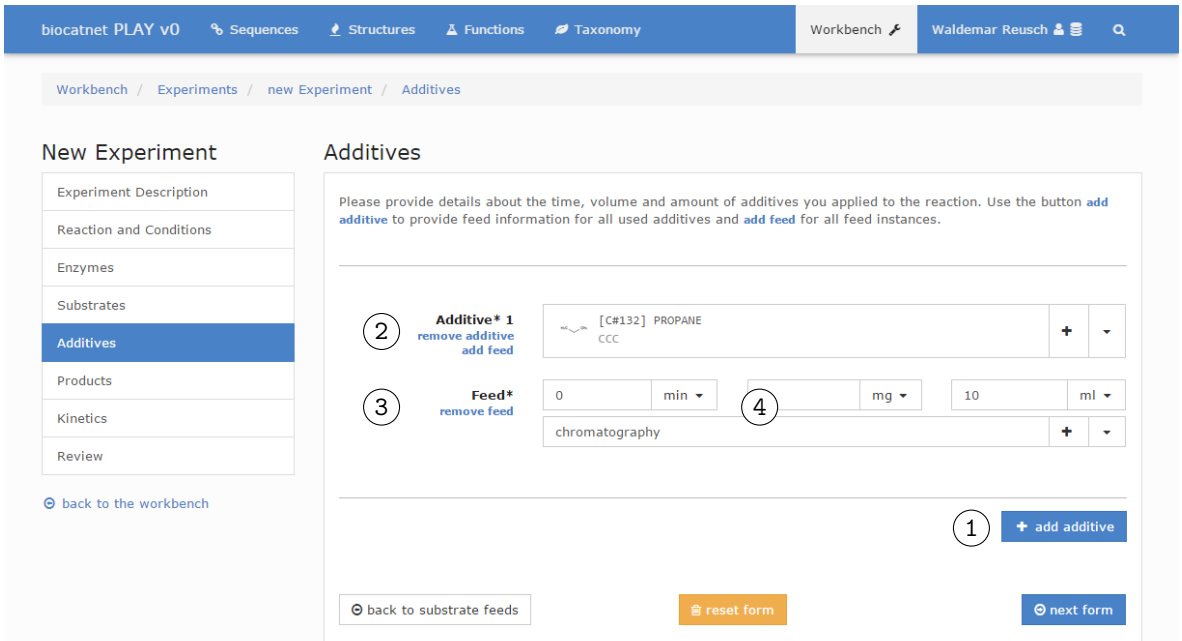


Figure 5.42: Screenshot of the fifth step in the experiment creation form. Here, the user has the chance to define additives he used in the experiment, i.e. compounds which do not primarily partake in the described reaction, but may be affecting the reaction in other ways. Initially, there are no form fields. (1) The user must use the button 'add additive' to spawn form fields where he can choose (2) a chemical compound and (3) define one or more feed instances using the buttons 'add feed' and 'remove feed'. (4) Each feed instance is defined by a feed time, amount and volume.

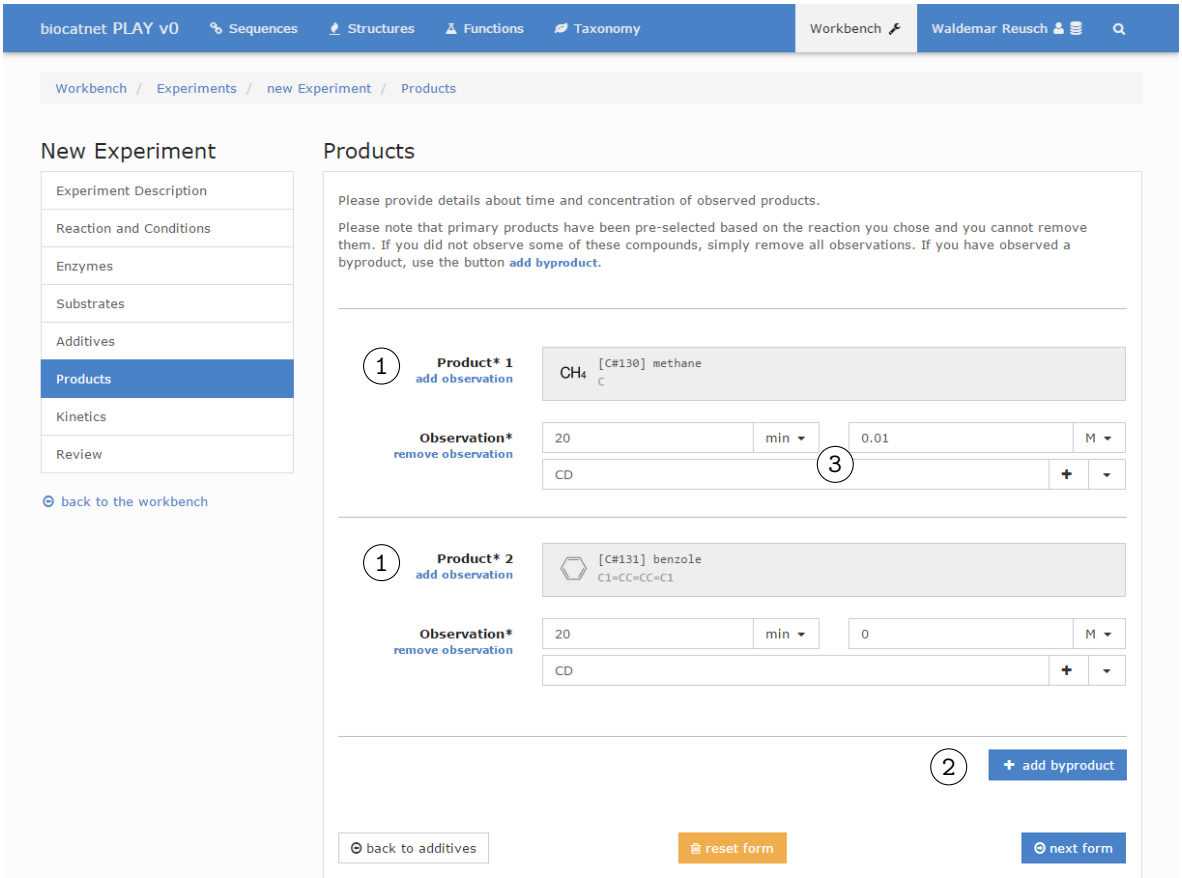


Figure 5.43: Screenshot of the sixth step in the experiment creation form. Here, the user can define which products have been measured. (1) The products defined by the observed reaction are pre-selected, (2) additionally observed byproducts can be defined using the 'add byproduct' button. (3) Each observation consists of an observation time, concentration and the observation method.

biocatnet PLAY v0 | Sequences | Structures | Functions | Taxonomy | Workbench | Waldemar Reusch

Workbench / Experiments / new Experiment / Review

New Experiment

1

- Experiment Description
- Reaction and Conditions
- Enzymes
- Substrates
- Additives
- Products
- Kinetics
- Review

back to the workbench

Review

2

ExperimentSet

Another Experiment Set

Experiment

Some meaningful experiment name

Reaction

[R#26] Demethylation of methylbenzene

1 [C#129] methylbenzene → 1 [C#130] methane + 1 [C#131] benzole

Conditions

No further description provided. Would you like to change that?

Initial reaction volume	0.01	ml
Shaking Frequency		min^{-1}
Temperature	273.15	°K
pressure	1	bar
pH	6	

Buffer

[B#12] Testbuffer

Water, Na+, O2

Enzyme Feeds

	Feed Time	Feed Amount	Feed Volume
[S#55] pyruvate oxidase from [T#1303] Streptococcus oralis acquired by [EM#3] ionic exchange chromatography expressed in [T#43] Cystobacter fuscus	0	5 mg	0

Substrate Feeds

	Feed Time	Feed Amount	Feed Volume
[C#129] methylbenzene acquired by [CM#4] chromatography	0 min	10 mg	0 ml
[C#129] methylbenzene acquired by [CM#4] chromatography	0 min	10 mg	0 ml

Additive Feeds

	Feed Time	Feed Amount	Feed Volume
[C#132] PROPANE	0 min	2 mg	10 ml

Products

	Time	Concentration
[C#130] methane measured by [MM#4] CD	20 min	0.01 M
[C#131] benzole	20 min	0 M

Figure 5.44: Screenshot of the last step in the experiment creation form. Here, the user has a chance to review his entries. (1) Using the navigation on the left or the (2) pencil icons (✎) next to the headlines, the user can jump to any of the previous forms to make some changes. After confirming this review, the experiment entry will finally be written to the database, and the user will be redirected to the experiment details view of the result.

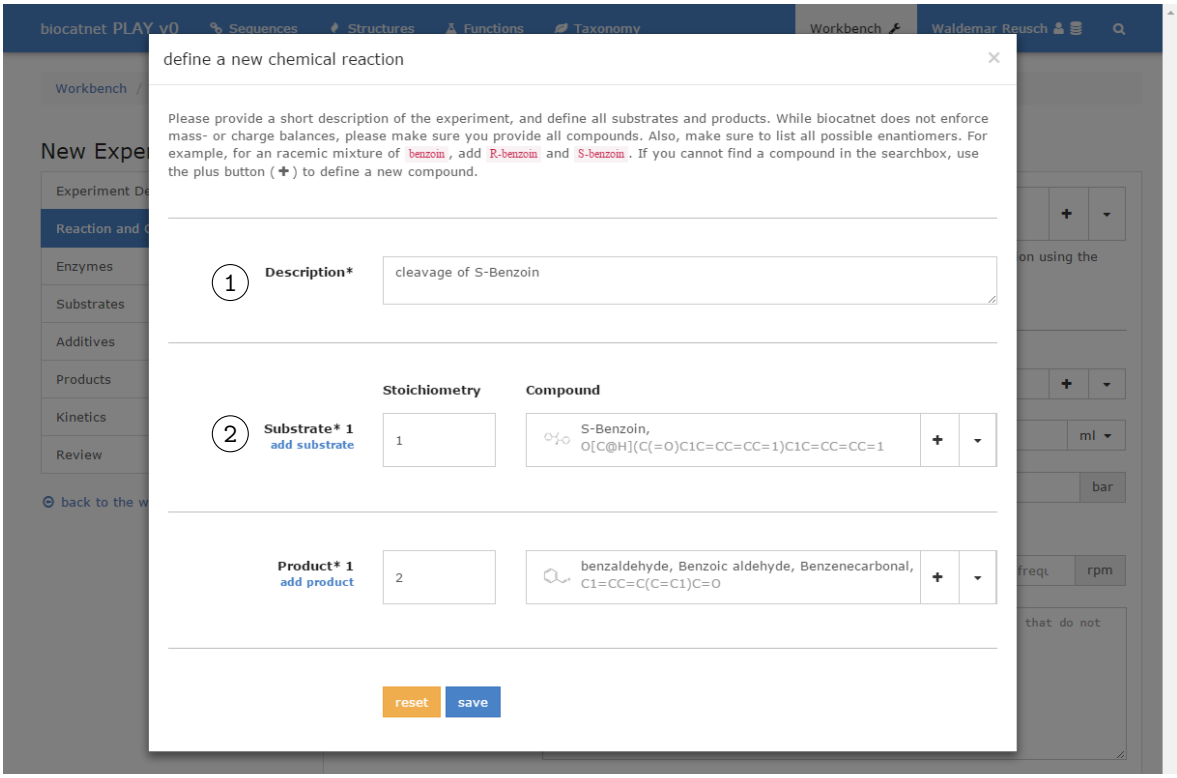


Figure 5.45: Screenshot of the hovering reaction creation form, triggered by using the 'add reaction' button in the second step of the experiment creation form. (Figure 5.39 on page 81) Here, the user can define a new reaction that he has observed. For this, he must provide (1) a short description, and (2) one or more substrates and products complete with stoichiometric coefficients.

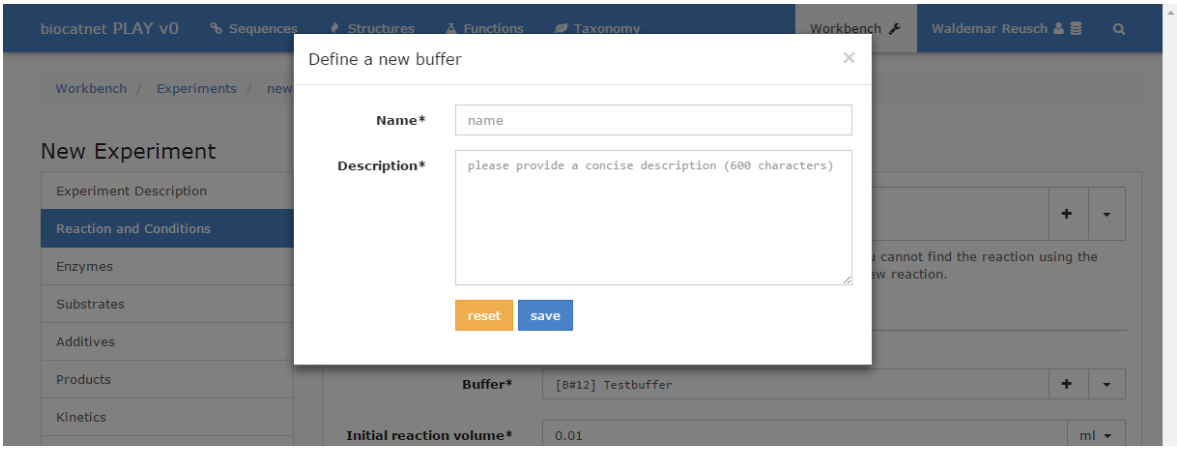


Figure 5.46: Screenshot of the hovering buffer creation form. A short name and detailed description suffice in the creation of new buffer entities.

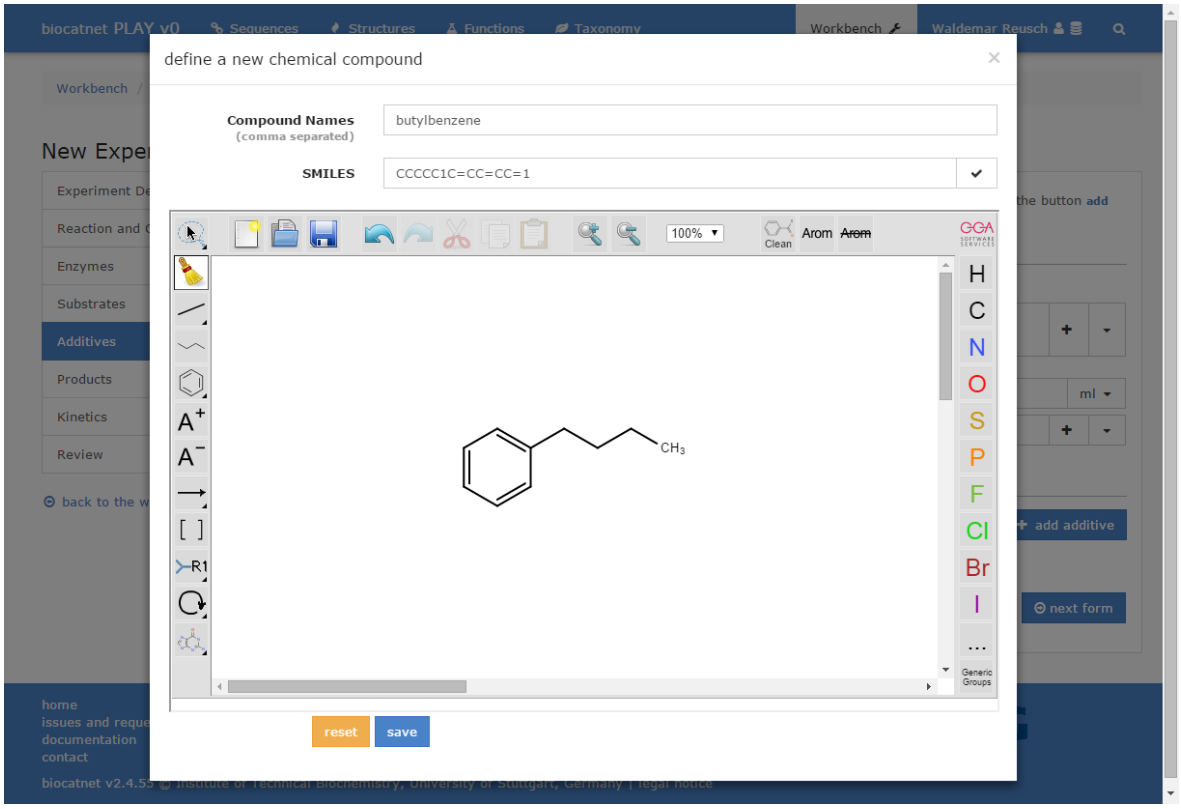


Figure 5.47: Screenshot of the hovering compound creation form. When creating a new compound, the user must provide at least one name and a *SMILES* code. Next to the *SMILES* form field, a pencil button (✎) triggers the extension of the form to show an canvas for chemical drawings (provided with *Ketcher*, [65]). Changes to the drawing will be immediately reflected in the *SMILES* form field and vice versa.

5.7 Use cases

Several FSPDs are already up and running atop the BioCatNet, two of which have been published recently. Other FSPDs, centering around Cytrochrome P450, lipases, laccases and multicopper oxidases, transaldolases, hydratases, short-chain dehydrogenases/reductases and various other protein families, are currently being updated to use the new platform.

5.7.1 Analysis of thiamine diphosphate-dependent enzymes

Thiamine diphosphate (ThDP)-dependent enzymes form a diverse protein family present in all kingdoms of life. Their ability to catalyze a broad range of reactions makes them promising candidate for biocatalysis. To provide a comprehensive database for a systematic sequence and structure analysis, the Thiamine diphosphate-dependent Enzyme Engineering Database (TEED) was updated using the DBParse toolbox [68] and the FSPD platform BioCatNet. Based on 51 seed sequences, representatives taken from previous versions of the TEED, 77,493 sequences of 52,565 proteins and 240 crystal structures were found and fed into the updated TEED. The proteins were grouped into 168 homologous families, and the 168 homologous families were classified into 9 superfamilies. [69] The updated version of the TEED is available at `http://teed.biocatnet.de`.

5.7.2 Analysis of imine reductases

Research revolving around imine-reducing enzymes is still in its infancy, but because of their exquisite selectivities, they are a promising reagent for the generation of chiral compounds for the fine-chemical industry. Using two imine reductases described only recently as seed sequences, the DBParse toolbox [68] found just under 449 sequences of 398 proteins which were then classified into 13 homologous families and 9 superfamilies. Combining this database and current knowledge on imine reductases, three novel enzymes were identified which exhibited comparable to higher catalytic efficiencies as compared to previously described enzymes. [61] The IRED is available at `http://ired.biocatnet.de`.

6 Discussion

More than ten years of experience with family-specific protein databases (FSPDs) such as the Lipase Engineering Database (LED) or Thiamine diphosphate-dependent Enzyme Engineering Database (TEED) have shown that a wide variety of questions can be answered when protein-related data from different sources can be flexibly combined. FSPDs center around multiple sequence alignments (MSAs) containing all available sequences from an entire protein family. The platform containing the protein database and the software to collect and process data to populate it, developed at the Institute of Technical Biochemistry (ITB), has since been known as the Data Warehouse system for protein Families (DWARF).

With the BioCatNet, we have successfully overhauled the DWARF. Taking into account recent developments and best practices in web-application architecture, we have come up with an improved FSPD system which is better performing, easier to use and easier to extend. At the same time, BioCatNet already surpasses DWARF's capabilities, the major point being the ability to store details of biochemical functionality and link it unambiguously to one specific amino-acid sequence. Thus, BioCatNet now effectively brings together protein sequence, structure and functional information in one database.

Many choices made during the development of the BioCatNet reflect the developers' objective to keep the system modular and extensible. Fundamental application development principles like *encapsulation*, *dependency injection* and the MVC-pattern have been applied throughout the BioCatNet back end code base to facilitate future development and expansion of the back end system (see section 5.2 on page 42). The core of BioCatNet now consists of an API providing access to the underlying database (see section 5.5 on page 50). This decision allowed to build modular, loosely-coupled services and the GUI. Moreover, the API provides skilled users with a direct access to the BioCatNet data, ready to be consumed by services and tools written by themselves or third-party software. The choice to use popular and well-documented front end libraries like *jQuery* [35] and *Twitter Bootstrap* [10] as well as the creation of specialized libraries (see section 5.3 on page 47) will ease further development of the GUI. Especially the front end framework *Twitter Bootstrap* allowed to emphasize on a truly intuitive and fluid user interface (see section 4.4.4 on page 35). The resulting layout adapts to the screen size and output medium, resulting clean print-outs as well as in an pleasant experience even on handheld devices.

The main objective of the BioCatNet platform is to enable the systematic analysis of biochemical properties and functions of proteins and therefore to aid in the discovery and development of novel biocatalysts as well as the engineering and enhancement of

established ones. BioCatNet is by far not the first approach undertaken to try and allow an systematic analysis of enzymes. Protein databases like *BRENDA/KENDA*, [60, 36] *PANTHER* [43] and *UniProtKb* [12] are only a few of a large number of services that provide the scientific community with information about protein sequence, structure and function while initiatives like *biosharing* [59], *Standards for Reporting Enzymology Data (STRENDA)* [2, 67] and *bioDBcore* [4] provide standards and best practices to acquire, store and share this information.

In contrast to holistic protein databases like *BRENDA* [60] or the *PDB* [6], BioCatNet pursuits an approach focused on protein families. Inherited from the preceding DWARF, this approach allows the BioCatNet developers to focus on a smaller set of questions and problems, as the size of the target group is much smaller. On the other hand, throughout the scientific community different standards are conceived and worked upon in smaller groups with similar interests first, and they are only presented to the broader community once they have matured. Aiming to serve smaller, focused scientific groups, BioCatNet will be part of the standardization process early on and will help proliferation and discussion as a central repository for biochemical information of this group. In time, BioCatNet will be adapted to accommodate more different protein families and with each adaption the number of functions and capabilities of BioCatNet will grow. At the same time, it will help improve on standards and cross-introduce them between the different focus groups. BioCatNet thereby also offers a central repository for information revolving around a specific protein family, partly extracting data from other databases (like *NCBI* or *PDB*) and partly linking them to our records (like *KEGG*, [47]). In future, the number of source databases for the BioCatNet will grow as its functionality will be extended to include more protein families as well as information about, for example, the success of different expression systems or genetic data.

Despite the wealth of information offered by countless established and mature protein databases, scientists struggle with the simple question *"What happens to protein* X*'s function if I change the amino acid* A *on position* Y *to* B*?"*. There are cases in which other laboratories have asked the exact same question, conducted experiments and published the results. Even in these cases, finding an definite answer can turn out to be quite difficult. After the initial struggle of finding the appropriate literature, comparability of the data is an enormous issue to deal with. Though the questions might sound the same at first, every laboratory focuses on a different aspect of the experiment. Therefore, often careful study of experimental setups and conditions is needed to assess the relevance of each publication. Expression systems, buffer composition and temperature are only a few parameters which can influence the outcome of a biochemical assay immensely. Differing experiment parameters and functional data is not even the real problem, but rather the effort one has to spend to extract this information from numerous publications revolving around the question at hand.

To ease this task, BioCatNet stores raw experiment data. The BioCatNet database is designed to hold information about experimental setups, conditions, substrates, products and additives and, most importantly, an unambiguously defined amino acid sequence of the protein in question. The database scheme is laid out in a fashion that

ensures consistent and comparable datasets and largely complies to the guidelines suggested by the *STRENDA* initiative. [67] With consistent and conforming data, comparability becomes much easier and researchers can focus on single parameters. Conclusions about the effect of the temperature on a biochemical assay can be assessed with much more confidence when all other parameters are asserted to be equal. Thus, scientist will be able to construct detailed models of biochemical experiments with much more confidence. Better models in turn will yield more accurate hypotheses, accelerating the cycle of the scientific method - model, predict, experiment, observe and model again.

While most protein databases listed above solely rely on *text-mining* and expert curation, BioCatNet has chosen a more user-centered approach to the acquisition of experimental data. While it has been proven to be quite successful in extracting relevant data from scientific publications, *text-mining* involves an extraordinary effort and is still far from being the perfect solution. The BRENDA is an exemplary database with biochemical information collected by *text-mining* and groomed by experts. Still, miscategorizations and entries of poor quality are found quite often. Therefore, from its conception, BioCatNet was designed to rely on raw experiment data provided by bench scientists.

This design goal has been achieved with the creation of the BioCatNet *Workbench*, a set of web pages revolving around user-contributed data (see subsection 5.6.10 on page 70). Here, contributors can use the provided forms to create and post new entries concerning protein sequences and experiment parameters and results. Minimal, clearly structured forms provide a simple to use, yet powerful interface. Many form elements provide predictive capabilities, known from various search engines, while other form elements are filled based on user's previous choices or on the context he is working in. We know that nobody really likes the exhaustive repetitive task of filling out forms, which is why we focused on simplifying this task as much as possible without sacrificing data accuracy and integrity or imposing impractical standards on our contributors.

Here, it's worth to point out that *contributing* to the BioCatNet is not automatically *publishing*. The user will be in full control of his data at all times. BioCatNet provides an user and group management system which allows all contributors to specify who can read and who can edit their data, based on single users or whole groups. This does not only apply to experimental setups and results, but to every entry they contribute. That way, scientists can make use of the intricate data model and the growing number of tools the BioCatNet will be providing to conduct their research 'in private', releasing the data only after the publication of a scientific paper, for example.

More than ten papers (with more than 200 citations) and ten databases published in the last decade by the ITB alone prove that family-specific protein databases are beneficial to the scientific community and are indeed accelerating research. And this is true for the FSPDs build atop DWARF. With the BioCatNet as the DWARF's successor, presenting a platform that is more user-friendly, is more robust, has more functions and is overall more mature, we predict that the contribution of FSPDs build atop the BioCatNet will boost scientific endeavors even more and contribute to the improvement of established and the discovery of novel biocatalysts.

7 Outlook

What started out as an upgrade to the DWARF to allow the inclusion of functional information, turned out to be an major overhaul and rejuvenation, complete with a new name: BioCatNet. Though much has been done as of this writing, there is still much to be drawn on the potential which BioCatNet presents.

For one, many more services and functionalities can be implemented using the established modular system of the BioCatNet. On the other hand, FSPDs established on the DWARF need to be ported to the new system and information for other protein families needs to be gathered to create novel FSPDs. For this to succeed, the number of core developers must increase, to split the burden of development and maintenance.

To improve the user experience, BioCatNet needs to exchange its multiple sequence alignment (MSA) viewer in favor of an implementation which does not depend on *Java* browser plugins. Also, BioCatNet may choose to display three-dimensional protein structures directly, instead of referring the user to the Protein Data Bank. That this is perfectly feasible, is shown with the display of three-dimensional homology models. Ideas have been developed for the BioCatNet to support commenting and collaboration on experimental setups in the future. In general, BioCatNet can implement more ideas revolving around functional parameters and data, i.e. experimental setups and results, though it is hard to predict what features will be feasible and useful.

Therefore, the most crucial aspect of BioCatNet's future is collaboration. The data model is well defined and the user interface has underwent first tests. Now we need to populate the database with experimental data, collect and process feedback, exchange ideas and see what conclusions can be drawn, what services are missing and which direction BioCatNet shall take in the future.

Acknowledgements

First I want to thank Prof. Dr. Jürgen Pleiss and Prof. Dr. Bernhard Hauer for giving me the opportunity to work on such an interesting topic, especially since the development of software and web applications is not yet a core interest at the ITB.

Special gratitude I have to express to my supervisor Constantin Vogel. Through many discussions he has helped me understand the science community, educated me in database design, led the way during development and has kept me from going astray with my work many times. Especially I have to thank him for his patience, which gave me the opportunity to explore novel methods and experiment with new features.

I want to thank Sivlia Racolta for testing the BioCatNet since its very early stages and providing helpful feedback as well as bearing with me when I broke the system for a day. I thank Silvia, Constantin, Lukasz Griczman and Sven Benson for providing me with an insight into the scientific community and for their support in questions concerning proteins and computational biology alike.

I also acknowledge contributions of the FOR1296 research group, which helped to shape the concept behind BioCatNet. Especially Martina Pohl and Dörte Rother helped a great deal formulating the minimal set of information needed to describe an biochemical experiment. Further, I thank Anna Baier, Saskia Bock, Robert Westphal and Martina Pohl for their contributions and ideas to the shape of the user interface.

I am grateful to the whole Institute of Technical Biochemistry for having me and providing a professional and supportive atmosphere.

References

[1] S.F. Altschul et al. "Basic Local Alignment Search Tool". In: *J. Mol. Biol.* (1990), pp. 403–410. ISSN: 0022-2836. DOI: 10.1016/S0022-2836(05)80360-2.

[2] Rolf Apweiler et al. *A large-scale protein-function database.* 2010. DOI: 10.1038/nchembio.460.

[3] Elaine Ashton. *Perl Timeline.* 2001. URL: http://history.perl.org/PerlTimeline.html (visited on 09/18/2014).

[4] Alex Bateman. "Curators of the world unite: The International Society of Biocuration". In: *Bioinformatics* 26 (2010), p. 991. ISSN: 13674803. DOI: 10.1093/bioinformatics/btq101.

[5] *Beginner's Introduction to Perl.* Oct. 2000. URL: http://www.perl.com/pub/a/2000/10/begperl1.html (visited on 09/18/2014).

[6] H M Berman et al. "The Protein Data Bank." In: *Nucleic acids research* 28 (2000), pp. 235–242. ISSN: 0305-1048. DOI: 10.1093/nar/28.1.235.

[7] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax.* 2005. URL: http://tools.ietf.org/html/rfc3986 (visited on 08/31/2014).

[8] Marco Biasini. *PV - JavaScript Protein Viewer.* 2014. URL: http://biasmv.github.io/pv/ (visited on 11/12/2014).

[9] Marco Biasini. "PV - WebGL-based protein viewer". In: (Nov. 2014). DOI: {10.5281/zenodo.12620}. URL: %7Bhttp://dx.doi.org/10.5281/zenodo.12620%7D.

[10] *Bootstrap.* 2014. URL: http://getbootstrap.com (visited on 10/21/2014).

[11] U T Bornscheuer et al. "Engineering the third wave of biocatalysis." In: *Nature* 485.7397 (May 2012), pp. 185–94. ISSN: 1476-4687. DOI: 10.1038/nature11117. URL: http://www.readcube.com/articles/10.1038/nature11117?utm%5C_campaign=readcube%5C_access%5C&utm%5C_source=nature.com%5C&utm%5C_medium=purchase%5C_option%5C&utm%5C_content=thumb%5C_version.

[12] Emmanuel Boutet et al. "UniProtKB/Swiss-Prot." In: *Methods in molecular biology (Clifton, N.J.)* 406 (2007), pp. 89–112. ISSN: 1064-3745.

[13] Carsten Brandt. *Markdown php renderer.* 2014. URL: https://github.com/cebe/markdown (visited on 09/18/2014).

[14] Johnny Broadway. *analog.* 2014. URL: https://github.com/jbroadway/analog (visited on 09/18/2014).

[15] Christiam Camacho et al. "BLAST+: architecture and applications." In: *BMC bioinformatics* 10 (2009), p. 421. ISSN: 1471-2105. DOI: 10.1186/1471-2105-10-421.

[16] Scott Chacon. *Pro Git*. Berkeley, CA: Apress, 2009, pp. 1–210. ISBN: 978-1-4302-1833-3. DOI: 10.1007/978-1-4302-1834-0. URL: http://gitbookio.gitbooks.io/progit/content/en/index.html.

[17] *CLUSTAL-OMEGA*. 2014. URL: http://www.clustal.org/omega/README (visited on 10/28/2014).

[18] *Composer Documentation*. 2014. URL: OBhttps://getcomposer.org (visited on 09/15/2014).

[19] *Comprehensive Perl Archive Network*. 2014. URL: http://www.cpan.org (visited on 09/15/2014).

[20] *DB-Engines Ranking*. 2014. URL: http://db-engines.com/en/ranking (visited on 10/01/2014).

[21] Hannah Dienhart. "Bachelors Thesis: DBUpdate - a tool to update family specific protein databases implemented in the BioCatNet system". University of Stuttgart, 2014.

[22] Vincent Driessen. *A successful git branching model*. 2010. URL: http://nvie.com/posts/a-successful-git-branching-model/ (visited on 09/16/2014).

[23] *ECMAScript Language Specification*. 2011. URL: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf (visited on 09/18/2014).

[24] K Engelmark Cassimjee et al. "A general protein purification and immobilization method on controlled porosity glass: biocatalytic applications." en. In: *Chemical communications (Cambridge, England)* 50.65 (Aug. 2014), pp. 9134–7. ISSN: 1364-548X. DOI: 10.1039/c4cc02605e. URL: http://pubs.rsc.org/en/content/articlehtml/2014/cc/c4cc02605e.

[25] Anthony Farrara. *password-compat*. 2014. URL: https://github.com/ircmaxell/password_compat (visited on 09/18/2014).

[26] Markus Fischer and Jürgen Pleiss. "The Lipase Engineering Database: a navigation and analysis tool for protein families." In: *Nucleic acids research* 31.1 (Jan. 2003), pp. 319–21. ISSN: 1362-4962. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=165462%5C&tool=pmcentrez%5C&rendertype=abstract.

[27] Markus Fischer et al. "DWARF–a data warehouse system for analyzing protein families". In: *BMC bioinformatics* 7.1 (2006), p. 495.

[28] Markus Fischer et al. "The cytochrome P450 engineering database: A navigation and prediction tool for the cytochrome P450 protein family". In: *Bioinformatics* 23 (2007), pp. 2015–2017. ISSN: 13674803. DOI: 10.1093/bioinformatics/btm268.

[29] M Galleni et al. "Standard numbering scheme for class B beta-lactamases." In: *Antimicrobial agents and chemotherapy* 45.3 (Mar. 2001), pp. 660–3. ISSN: 0066-4804. DOI: 10.1128/AAC.45.3.660-663.2001. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=90352%5C&tool=pmcentrez%5C&rendertype=abstract.

[30] R.L. Glass. *Facts and Fallacies of Software Engineering*. Agile Software Development. Pearson Education, 2002. ISBN: 9780321630094. URL: http://books.google.de/books?id=lvTrvhZa1rEC.

[31] *Growth of GenBank and WGS*. 2014. URL: http://www.ncbi.nlm.nih.gov/genbank/statistics (visited on 10/15/2014).

[32] Justin Hileman. *Mustache.php*. 2014. URL: https://github.com/bobthecow/mustache.php (visited on 09/18/2014).

[33] *HTML 4 - Conformance: requrements and recommendations*. URL: http://www.w3.org/TR/html401/conform.html#deprecated (visited on 09/18/2014).

[34] *Indigo Toolkit*. 2014. URL: http://www.ggasoftware.com/opensource/indigo (visited on 10/21/2014).

[35] *jQuery*. 2014. URL: http://jquery.com/ (visited on 10/21/2014).

[36] *KENDA (Kinetic Enzyme Data)*. 2014. URL: http://www.brenda-enzymes.info/search_result.php?a=55 (visited on 10/15/2014).

[37] Michael Knoll and Jürgen Pleiss. "The Medium-Chain Dehydrogenase/reductase Engineering Database: a systematic analysis of a diverse protein family to understand sequence-structure-function relationship." In: *Protein science : a publication of the Protein Society* 17 (2008), pp. 1689–1697. ISSN: 1469-896X. DOI: 10.1110/ps.035428.108.

[38] Michael Knoll et al. "The PHA Depolymerase Engineering Database: A systematic analysis tool for the diverse family of polyhydroxyalkanoate (PHA) depolymerases." In: *BMC bioinformatics* 10 (2009), p. 89. ISSN: 1471-2105. DOI: 10.1186/1471-2105-10-89.

[39] R Z Kramer et al. "X-ray crystallographic determination of a collagen-like peptide with the repeating sequence (Pro-Pro-Gly)." In: *Journal of molecular biology* 280.4 (July 1998), pp. 623–38. ISSN: 0022-2836. DOI: 10.1006/jmbi.1998.1881. URL: http://www.ncbi.nlm.nih.gov/pubmed/9677293.

[40] Jan Lehnart. *Mustache.js*. 2014. URL: https://github.com/janl/mustache.js/ (visited on 09/18/2014).

[41] D J Lipman and W R Pearson. "Rapid and sensitive protein similarity searches." In: *Science (New York, N.Y.)* 227 (1985), pp. 1435–1441. ISSN: 0036-8075. DOI: 10.1126/science.2983426.

[42] The Daring Fireball Company LLC. *Markdown specification*. 2014. URL: http://daringfireball.net/projects/markdown/ (visited on 09/18/2014).

[43] Huaiyu Mi et al. "The PANTHER database of protein families, subfamilies, functions and pathways". In: *Nucleic Acids Research* 33 (2005). ISSN: 03051048. DOI: 10.1093/nar/gki078.

[44] Jason H. Moore. "Bioinformatics". In: *Journal of Cellular Physiology* 213.June (2007), pp. 365–369. ISSN: 00219541. DOI: 10.1002/jcp.21218.

[45] David W. Mount. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, 2004. ISBN: 9780879697129. URL: http://books.google.de/books?id=bvY21DGa1OwC.

[46] Noel M. O'Boyle et al. "Open Babel: An Open chemical toolbox". In: *Journal of Cheminformatics* 3 (2011). ISSN: 17582946. DOI: 10.1186/1758-2946-3-33.

[47] Hiroyuki Ogata et al. *KEGG: Kyoto encyclopedia of genes and genomes*. 1999. DOI: 10.1093/nar/27.1.29.

[48] *Open Babel: The Open Source Chemistry Toolbox*. 2014. URL: http://openbabel.org/wiki/Main_Page (visited on 10/21/2014).

[49] *OpenStax CNX - Introduction to NCBI*. URL: http://cnx.org/contents/388e2c74-93a0-4e1b-8dab-fda7f94ca04c@2/Introduction%5C_to%5C_NCBI (visited on 10/12/2014).

[50] *Our Mission*. URL: http://www.ncbi.nlm.nih.gov/About/glance/ourmission.html (visited on 10/12/2014).

[51] *Overloading*. 2014. URL: http://php.net/manual/en/language.oop5.overloading.php (visited on 10/05/2014).

[52] *Packagist - The PHP Package Archivist*. 2014. URL: http://packagist.org (visited on 09/15/2014).

[53] J. Pleiss et al. "Lipase engineering database - Understanding and exploiting sequence-structure-function relationships". In: *Journal of Molecular Catalysis B: Enzymatic* 10.5 (2000-10-02T00:00:00), pp. 491–508. DOI: doi:10.1016/S1381-1177(00)00092-8. URL: http://www.ingentaconnect.com/content/els/13811177/2000/00000010/00000005/art00092.

[54] Tom Preston-Werner. *Semantic Versioning 2.0.0*. 2014. URL: http://semver.org/ (visited on 08/15/2014).

[55] Silvia Racolta et al. "The triterpene cyclase protein family: A systematic analysis". In: *Proteins: Structure, Function and Bioinformatics* 80 (2012), pp. 2009–2019. ISSN: 08873585. DOI: 10.1002/prot.24089.

[56] Trygve Reenskaug and James O. Coplien. *The DCI Architecture: A New Vision of Object-Oriented Programming*. 2009. URL: http://www.artima.com/articles/dci_vision.html (visited on 08/16/2014).

[57] *RequireJS*. 2014. URL: http://requirejs.org/ (visited on 10/21/2014).

[58] Margaret Rouse. *SearchSQLServer - Database*. 2014. URL: http://searchsqlserver.techtarget.com/definition/database (visited on 10/01/2014).

[59] Susanna Sansone et al. "BioSharing Catalogue". In: *University of Oxford* (2011). URL: http://otter.oerc.ox.ac.uk/biosharing/?q=standards.

[60] Maurice Scheer et al. "BRENDA, the enzyme information system in 2011". In: *Nucleic Acids Research* 39 (2011). ISSN: 03051048. DOI: 10.1093/nar/gkq1089.

[61] Philipp N. Scheller et al. "Enzyme Toolbox: Novel Enantiocomplementary Imine Reductases". In: *ChemBioChem* 15.15 (Aug. 2014), n/a–n/a. ISSN: 14394227. DOI: 10.1002/cbic.201402213. URL: http://www.ncbi.nlm.nih.gov/pubmed/25163890.

[62] Fabian Sievers and Desmond G. Higgins. "Clustal omega, accurate alignment of very large numbers of sequences". In: *Methods in Molecular Biology* 1079 (2014), pp. 105–116. ISSN: 10643745. DOI: 10.1007/978-1-62703-646-7-6.

[63] Demet Sirim et al. "The cytochrome P450 engineering database: Integration of biochemical properties." In: *BMC biochemistry* 10 (2009), p. 27. ISSN: 1471-2091. DOI: 10.1186/1471-2091-10-27.

[64] Demet Sirim et al. "The Laccase Engineering Database: A classification and analysis system for laccases and related multicopper oxidases". In: *Database* 2011 (2011). ISSN: 17580463. DOI: 10.1093/database/bar006.

[65] GGA Software. *Ketcher*. 2014. URL: http://ggasoftware.com/opensource/ketcher (visited on 09/18/2014).

[66] Quan K Thai and Juergen Pleiss. "SHV Lactamase Engineering Database: a reconciliation tool for SHV β-lactamases in public databases." In: *BMC genomics* 11 (2010), p. 563. ISSN: 1471-2164. DOI: 10.1186/1471-2164-11-563.

[67] Keith F Tipton et al. "Standards for Reporting Enzyme Data: The STRENDA Consortium: What it aims to do and why it should be helpful". In: *Perspectives in Science* 1 (2014), pp. 131–137. ISSN: 22130209. DOI: 10.1016/j.pisc.2014.02.012.

[68] Constantin Vogel. "PhD Thesis: Systematic Analysis of the Sequence-Structure-Function Relationships of Thiamine Diphosphate-dependent Enzymes". University of Stuttgart, 2014.

[69] Constantin Vogel and Jürgen Pleiss. "The modular structure of ThDP-dependent enzymes." In: *Proteins* 82.10 (Oct. 2014), pp. 2523–37. ISSN: 1097-0134. DOI: 10.1002/prot.24615. URL: http://www.ncbi.nlm.nih.gov/pubmed/24888727.

[70] Constantin Vogel et al. "A standard numbering scheme for thiamine diphosphate-dependent decarboxylases." In: *BMC biochemistry* 13 (2012), p. 24. ISSN: 1471-2091. DOI: 10.1186/1471-2091-13-24. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3534367%5C&tool=pmcentrez%5C&rendertype=abstract.

[71] Andrew M. Waterhouse et al. "Jalview Version 2-A multiple sequence alignment editor and analysis workbench". In: *Bioinformatics* 25 (2009), pp. 1189–1191. ISSN: 13674803. DOI: 10.1093/bioinformatics/btp033.

[72] *What can PHP do?* 2014. URL: http://php.net/manual/en/intro-whatcando.php (visited on 09/18/2014).

[73] *What is CSS?* URL: http://www.w3.org/standards/webdesign/htmlcss#whatcss (visited on 09/18/2014).

[74] *What is PHP?* 2014. URL: http://php.net/manual/en/intro-whatis.php (visited on 09/18/2014).

[75] Eleanor J Whitfield, Manuela Pruess, and Rolf Apweiler. "Bioinformatics database infrastructure for biotechnology research." In: *Journal of biotechnology* 124.4 (Aug. 2006), pp. 629–39. ISSN: 0168-1656. DOI: 10.1016/j.jbiotec.2006.04.006. URL: http://www.sciencedirect.com/science/article/pii/S016816560600321X.

[76] Michael Widmann, P Benjamin Juhl, and Jürgen Pleiss. "Structural classification by the Lipase Engineering Database: a case study of Candida antarctica lipase A." In: *BMC genomics* 11 (Jan. 2010), p. 123. ISSN: 1471-2164. DOI: 10.1186/1471-2164-11-123. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2841678%5C&tool=pmcentrez%5C&rendertype=abstract.

[77] Michael Widmann, Jürgen Pleiss, and Peter Oelschlaeger. "Systematic analysis of metallo-β-lactamases using an automated database". In: *Antimicrobial Agents and Chemotherapy* 56 (2012), pp. 3481–3491. ISSN: 00664804. DOI: 10.1128/AAC.00255-12.

[78] Michael Widmann, Robert Radloff, and Jürgen Pleiss. "The Thiamine diphosphate dependent Enzyme Engineering Database: a tool for the systematic analysis of sequence and structure relations." In: *BMC biochemistry* 11 (2010), p. 9. ISSN: 1471-2091. DOI: 10.1186/1471-2091-11-9.

Declaration of Authorship

I hereby declare that the thesis submitted is my own work. All direct or indirect sources used are acknowledged as references.

Stuttgart, December 10, 2014

Waldemar Reusch