VDJdb server Documentation

Release 2.3.2

Dmitry Bagaev, Mikhail Shugay

Feb 04, 2019

Contents

The VDJdb web-server application implements a browser-based GUI for the VDJdb, a database of T-cell receptor sequences with known antigen specificities.

This documentation describes the installation and setup of VDJdb-server at your local network, as well as the REST API for the VDJdb portal.

VDJdb web-server source code and binaries are located here.

CHAPTER 1

Table of Contents

1.1 Introduction

1.1.1 Overview

VDJdb is a curated database of T-cell receptor (TCR) sequences with known antigen specificities. The primary goal of VDJdb is to facilitate access to existing information on T-cell receptor antigen specificities, i.e. the ability to recognize certain epitopes in a certain MHC contexts.

Our mission is to both aggregate the scarce TCR specificity information available so far and to create a curated repository to store such data. In addition to routine database updates providing the most up-to-date information fetched from recently published studies utilizing TCR specificity assays, we make our best to ensure data consistency and correct irregularities in TCR specificity reporting with a complex database validation scheme:

- We take into account all available information on experimental setup used to identify antigen-specific TCR sequences and assign a single confidence score to highligh most reliable records at the database generation stage.
- Each database record is also automatically checked against a database of V/J segment germline sequences to ensure standardized and consistent reporting of V-J junctions and CDR3 sequences that define T-cell clones.

Note: This software is intended for advanced users. We recommend using either the VDJdb web portal or the standalone VDJdb application with command line interface that can be found here.

See the *Installing VDJdb server* section for more details.

1.2 Installing VDJdb server

First make sure that you have installed Java Runtime Environment (JRE) v1.8 by running `java -version`. Any recent Linux distribution will provide it via its package manager. If not, or if your system is running MacOSX or Windows, download the JRE from Oracle.

1.2.1 Installing binaries

The most straightforward way to install VDJdb as a local server is to download the latest release package.

After downloading unzip the package wherever you want, but please avoid long paths and spaces (Windows version is especially sensitive to it).

You can find the server executable in `bin/` directory. To set up the server:

- Run `vdjdb-server.bat` file (Windows)
- Run `./vdjdb-server` in your console (Linux/Mac OS)

Wait until the server is started, and go to `localhost:9000` URL in your browser to open VDJviz.

To stop application just press *Ctrl-C* at any time in console.

Note: Note that an exception will be thrown in case the `9000` port is busy: `org.jboss.netty.channel.ChannelException: Failed to bind to: /0.0.0.0:9000`. In order to fix it, either close the application that is using this port (in UNIX the `lsof -i:9000` will give the processes that are using the port) or pass the `-Dhttp.port=XXXX` (where `XXXX` is new port id) argument to `vdjdb-server` shell script (UNIX) / `vdjdb-server.bat` (Windows)

1.2.2 Compiling from sources

1.2.2.1 Back-end dependencies

- VDJtools
- VDJdb-standalone
- Scala Build Tools

Please check that the versions of VDJtools and VDJdb-standalone are matched to that in `build.sbt` file coming with VDJdb-server.

1.2.2.2 Front-end dependencies

- Node.js
- Yarn

1.2.2.3 Compiling

VDJdb could be compiled from source using Scala Build Tools (SBT). Compilation should be performed under JRE v1.8 by running the following commands:

```
$ git clone https://github.com/antigenomics/vdjdb-web.git
$ cd vdjdb-web/
$ sbt
[VDJdb-server] $ frontendInstallDependencies
[VDJdb-server] $ build
```

Binaries could then be found under the `target/universal/vdjdb-server-X.Y.Z.zip` package.

1.2.2.4 SBT console commands

SBT console supports some utility functions which can be executed in the following way.

```
$ sbt
$ [VDJdb-server] $ command1
$ [VDJdb-server] $ command2
```

Available commands:

Command	Description
build	Build VDJdb application
backendBuild	Build VDJdb backend only
backendTest	Test VDJdb backend
frontendBuild	Build VDJdb frontend bundle only
frontendCleanDependencies	Clean frontend dependencies
frontendCleanBuild	Clean frontend build
frontendInstallDependencies	Install frontend dependencies
frontendOutdated	Check frontend dependencies updates

Warning: Note that after `frontendCleanDependencies` command will be executed other commands with `frontend` prefix (excluding `frontendInstallDependencies`) will be unavailable.

1.3 Usage

1.3.1 Command line usage

General way to run VDJdb server would be the following:

```
$ ./vdjdb-server -Darg1=value1 -Darg2=value2 ...etc
```

Note: If you want to change JRE arguments use `-J-` prefix. For example, to change memory heap size for Java Virtual Machine run the following command in your console:

```
$ ./vdjdb-server -J-Xmx16G
```

If insufficient amount memory is allocated, the Java Virtual Machine could drop with a *Java Heap Space Out of Memory* error.

1.3.2 Advanced configuration

VDJdb server configuration can be performed by passing additional arguments to `vdjdb-server` script with `-D` prefix in console or by manually editing application.conf file in the `conf/` directory.

List of all available arguments can be found in `conf/application.conf` file.

Play framework uses the HOCON syntax for configuration.

More about HOCON syntax, data types and other features can be found here.

You can also specify another local configuration file not packaged into the application artifacts:

```
$ ./vdjdb-server -Dconfig.file=/path/to/application.conf
```

1.3.2.1 VDJdb Database configuration

Argument	Type	Default	Description
application.database.useLocal	Boolean	true	Specify if use local vdjdb-database or try to download latest release from github repository
application.database.path	String	database/	Specify path to local vdjdb-database relative to `vdjdb-server` binary executable

Example:

```
// For local database
$ ./vdjdb-server -Dapplication.database.useLocal=true -Dapplication.database.
↪path=path/to/local/database/

// Automatic download latest release from github repository
$ ./vdjdb-server -Dapplication.database.useLocal=false
```

1.3.2.2 Temporary files configuration

VDJdb server can create some temporary files during execution.

Argument	Type	Default	Description
application.temporary.path	String	/tmp/vdjdb-temporary	Path to store temporary files
application.temporary.keep	Duration	12 hours	Specify the time during which the file is guaranteed to exist
application.temporary.interval	Duration	12 hours	Specify an interval for deleting expired temporary files

Example:

```
$ ./vdjdb-server -Dapplication.temporary.path=/tmp -Dapplication.temporary.
↪interval=30minutes -Dapplication.temporary.keep=1hour
```

1.3.2.3 Autorization configuration

Common configuration:

Argument	Type	Default	Description
application.auth.common.uploadLocation	String	/tmp/vdjdb/	Is verification required or not
application.auth.common.createUsers	List[User]	`[{ "login": "test", "email": "test@mail.com", "password": "123456", "permissionsID": "1", }]`	Create default users with application startup **Note:** This field can't be passed as command line argument. Use `application.conf` file instead.
application.auth.common.enableDefaultUsers	Boolean	true	Enable default users feature
application.auth.common.clearDefaultUsers	Boolean	false	Clear default users if exists

Demo user configuration:

Argument	Type	Default	Description
application.auth.demo.enabled	Boolean	true	Is demo user enabled or not
application.auth.demo.filesLocation	String	/tmp/vdjdb/demo	Demo account sample files location
application.auth.demo.login	String	vdjdb@vdjdb-demo.com	Demo account login and email
application.auth.demo.password	String	demo	Demo account password

List of available permissions ID:

- **0** - No-limits
- **1** - Standart, max files count is equal to 10, max file size is equal to 16MB
- **2** - Demo account, uploading is not allowed

Verification token configuration:

Argument	Type	Default	Description
application.auth.verification.required	Boolean	true	Is verification required or not
application.auth.verification.method	String	console	Verification method (console or email)
application.auth.verification.server	String	localhost	Verification server location
application.auth.verification.keep	Duration	24 hours	Specify the time during which the token is guaranteed to be valid
application.auth.verification.interval	Duration	24 hours	Specify an interval for deleting expired tokens

Example

```
//Disable verfication
$ ./vdjdb-server -Dapplication.auth.verification.required=false
```

Session token configuration:

Argument	Type	Default	Description
application.auth.session.keep	Duration	30 days	Specify the time during which the token is guaranteed to be valid
application.auth.session.interval	Duration	1 day	Specify an interval for deleting expired tokens

Reset token configuration:

Argument	Type	Default	Description
application.auth.reset.keep	Duration	24 hours	Specify the time during which the token is guaranteed to be valid
application.auth.reset.interval	Duration	24 hours	Specify an interval for deleting expired tokens

1.3.2.4 Annotations page configuration

Annotations upload page configuration:

Argument	Type	Default	Description
application.annotations.upload.maxFileSize	Size	64MiB	Max sample files size

1.3.2.5 Limits configuration

Argument	Type	Default	Description
application.limits.maxRequestsCount	Number	10 000	Max number of requests available to the user
application.limits.countClearInterval	Duration	1 hour	An interval for resetting number of request for user
application.limits.maxRequestsTime	Number	900 000	Total request time available to the user (in milliseconds)
application.limits.timeClearInterval	Duration	30 minutes	An interval for resetting total request time for user

1.3.2.6 SQL Database configuration

VDJdb server uses Slick API for accessing and storing data in SQL database.

Important: Standalone version uses H2 Database for handling metadata by default, if you want to change H2 to another DBMS please see the corresponding Play documentation section. You can also use this database to manually modify user limits.

It is safe to change default database location with `slick.dbs.default.db.url` argument.

Example:

```
// Change '/path/to/sql/database'
$ ./vdjdb-server -Dslick.dbs.default.db.url=jdbc:h2:file:/path/to/sql/database;DB_
↪CLOSE_DELAY=-1
```

The remaining `slick.dbs.default.*` arguments are best kept to the default settings.

More information about Slick API configuration arguments can be found here.

1.4 VDJdb server API

1.4.1 Accessing metadata

General information on database metadata (such as column IDs, names, etc..) can be obtained by performing a GET request to `/api/database/meta`.

HTTP Request

`GET /api/database/meta`

Example

```
$ curl -X GET https://vdjdb.cdr3.net/api/database/meta
```

The above command returns JSON structured like this:

```
{
    "metadata": {
        "numberOfRecords": Number,
        "numberOfColumns": Number,
        "columns":         Array[ColumnInfo]
    }
}
```

where `ColumnInfo` structured like this:

```
{
    "name":          String,        // Column's name
    "columnType":    String,        // 'txt' or 'seq'
    "visible":       Boolean,       // Should be always true
    "dataType":      String,        // Internal data type
    "title":         String,        // Formatted name
    "comment":       String,        // Column's description
    "values":        Array[String] // An array of all possible values (if available)
}
```

1.4.2 Accessing specific column info

HTTP Request

`GET /api/database/meta/columns/:column`

Note: You must replace `:column` with the correct column's name.

Example

```
$ curl -X GET https://vdjdb.cdr3.net/api/database/meta/columns/gene
```

The above command returns the following JSON structure:

```
{
    "column": {
        "name":       "gene",
        "columnType": "txt",
        "visible":    true,
        "dataType":   "factor",
        "title":      "Gene",
        "comment":    "TCR chain: alpha or beta.",
        "values":     [ "TRA", "TRB" ]
    }
}
```

1.4.3 Querying the database

You can query the database by sending a POST request with a specific JSON content

HTTP Request

`POST /api/database/search`

Request parameters

Request JSON structure:

```
{
    "filters":   Array[Filter],
    "page":      Optional[Number],      // Optional: page, if not specified server␣
↪will return all filtered rows from database
    "pageSize":  Optional[Number],      // Optional: page size, used only if ``page``␣
↪is specified. Default: 25
    "paired":    Optional[Boolean],     // Optional: specifies whether to include the␣
↪paired records. Default: false
    "sort":      Optional[String],      // Optional: sort rule, it has the following␣
↪structure: "<columnName>:<sortType>".
                                        //           Available sort types: 'asc' -␣
↪asceding order, 'desc' - descending order
                                        //           Example: "gene:asc"
}
```

`Filter` structure:

```
{
    "column":     String,               // Column's name
    "filterType": FilterType,           // Filter type (list of all available types␣
↪will be described below)
    "negative":   Boolean,              // Invert the filter: return only results␣
↪that do **NOT** match the filter
    "value":      String                // Filter's value
}
```

`FilterType` available options:

```
"exact"          // Exact match
"exact:set"      // Comma separated values, exact match for at least one value.
"substring:set" // Comma separated values, substring match for at least one value.
"pattern"        // Pattern match
"level"          // Number, greater or equal than **value**
"range"          // Range for numbers, it has the following structure "<min>:<max>".␣
↪Example: "5:10"
"sequence"       // Fuzzy match filter, it has the following structure "<query>:
↪<substitutions>:<insertions>:<deletions>"
                 // Note that sequence filter can be only applied to column with 'seq'␣
↪columnType.
                 // Example: "CASSFGVNSDYTF:1:1:1"
```

Example 1

Fetching information about `CAAAASGGSYIPTF`.

```
$ curl https://vdjdb.cdr3.net/api/database/search    \
   -H "Content-Type: application/json"               \
   -X POST                                           \
   -d '{ "filters" : [{ "column" : "cdr3", "value" : "CAAAASGGSYIPTF", "filterType" :
↪"exact", "negative" : false }] }'
```

The above command will response in:

```
{
    "page":         -1,
    "pageSize":     -1,
    "pageCount":     1,
    "recordsFound": 1,
    "rows":          [{
        "entries":  [ "TRA", "CAAAASGGSYIPTF", "TRAV1-2*01", "TRAJ6*01", "HomoSapiens
↪", ...etc ],
        "metadata": {
            "pairedID":     "3236",
            "cdr3vEnd":          2,
            "cdr3jStart":        4
        }
    }]
}
```

The above response has the following JSON structure:

```
{
    "page":         Number,          // Current page, equals -1 if 'page' argument␣
↪was not specified in request
    "pageSize":     Number,          // Page size, equals -1 if 'page' argument was␣
↪not specified in request
    "pageCount":    Number,          // Pages count, equals -1 if 'page' argument was␣
↪not specified in request
    "recordsFound": Number,          // Filtered records count (it is not include␣
↪paired rows)
    "rows":         Array[SearchRow] // Filtered database entries
}
```

`SearchRow` has the following structure:

```
{
    "entries":      Array[String],   // Array of entries, the order of the elements␣
↪matches the order of columns
                                     // See `Accessing metadata` section.
    "metadata":     {
        "pairedID":     String,      // Specifies paired record stringified ID number.
↪ If record isn't paired the value will be equal to "0".
        "cdr3vEnd":     Number,      // V region end index in cdr3 sequence.
        "cdr3jStart":   Number       // J region start index in cdr3 sequence.
    }
}
```

Example 2

Fetching information about `CAAAASGGSYIPTF` and his paired record.

```
$ curl https://vdjdb.cdr3.net/api/database/search    \
   -H "Content-Type: application/json"               \
   -X POST                                           \
   -d '{ "filters" : [{ "column" : "cdr3", "value" : "CAAAASGGSYIPTF", "filterType" :
↪"exact", "negative" : false }], "paired": true }'
```

The above command will response in:

```
{
    "page":         -1,
    "pageSize":     -1,
    "pageCount":    -1,
    "recordsFound":  1,
    "rows":         [
        { "entries": [ "TRA", "CAAAASGGSYIPTF", ...etc ], "metadata": { "pairedID":
↪"3236", "cdr3vEnd": 2, "cdr3jStart": 4 } },
        { "entries": [ "TRB", "CASGTGDSNQPQHF", ...etc ], "metadata": { "pairedID":
↪"3236", "cdr3vEnd": 4, "cdr3jStart": 7 } }
    ]
}
```