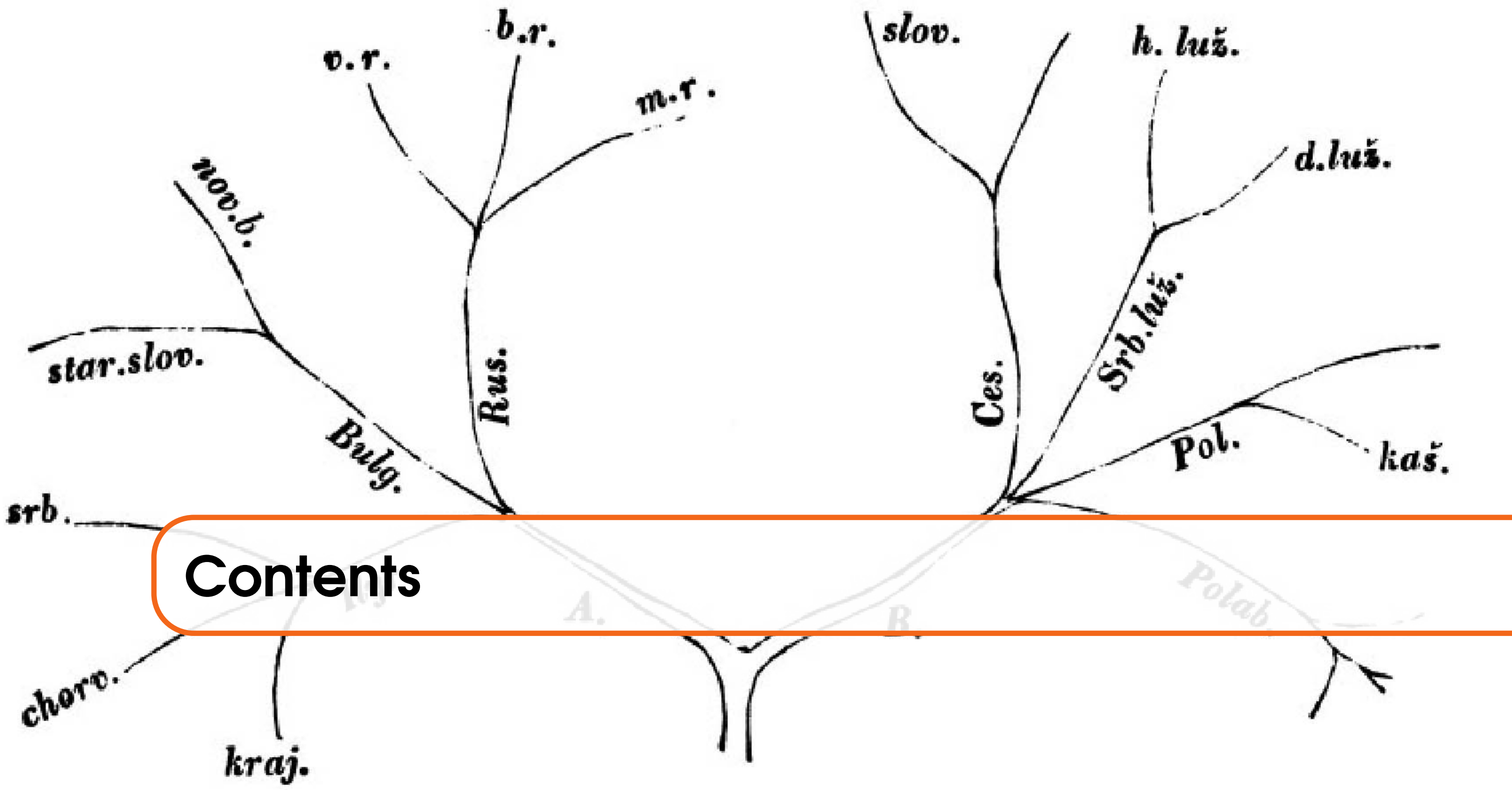
Stemmarest

Dokumentation des PSE2 Projekt

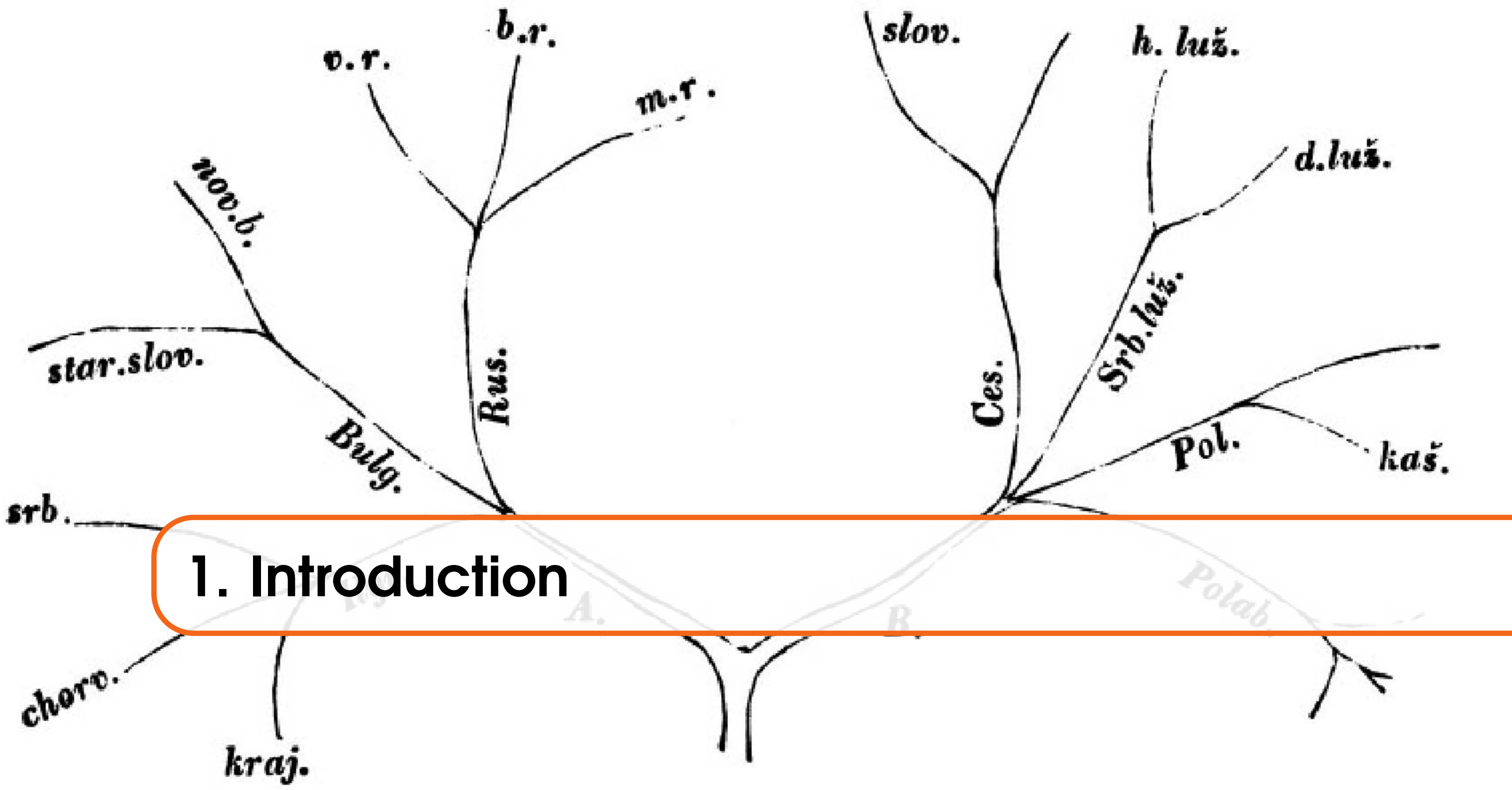
Jakob Schaerer, Severin Zumbrunn, Ido Gershoni, Joel Niklaus, Ramona Imhof

First printing, March 2013


b.r.
v.r.
m.r.
nov.b.
star.slov.
Bulg.
Rus.
srb.
chorv.
kraj.
slov.
h. luž.
d.luž.
Srb.luž.
Čes.
Pol.
kaš.

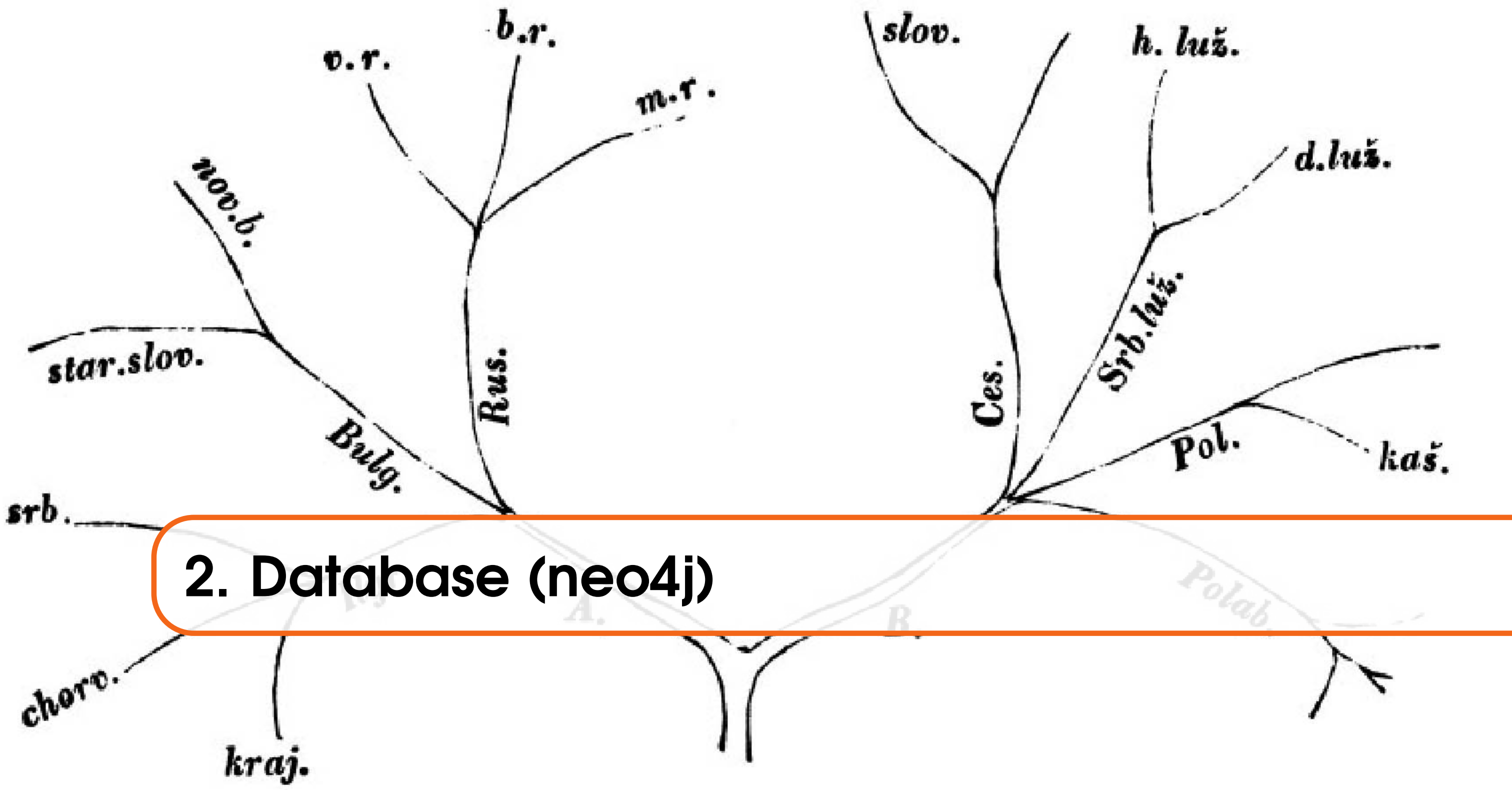
Contents

Project



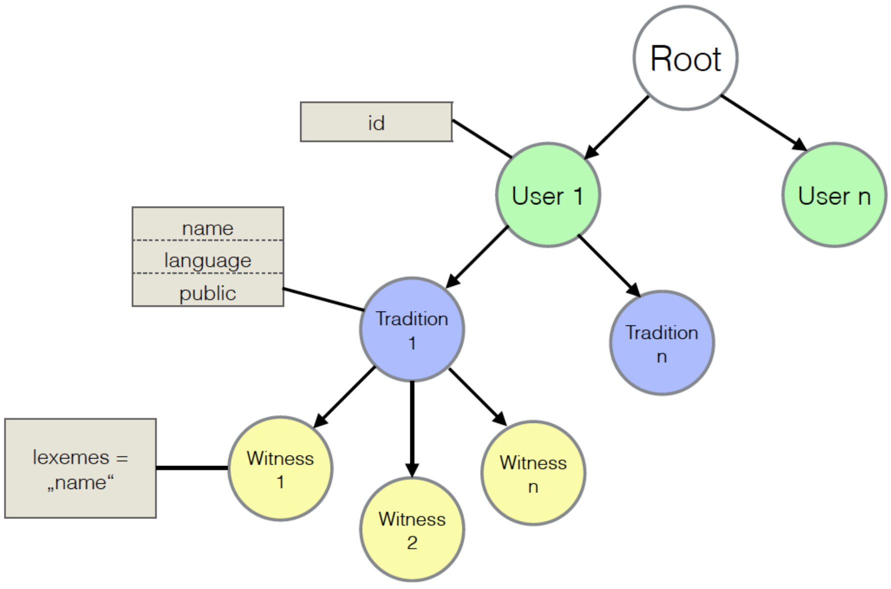
1. Introduction

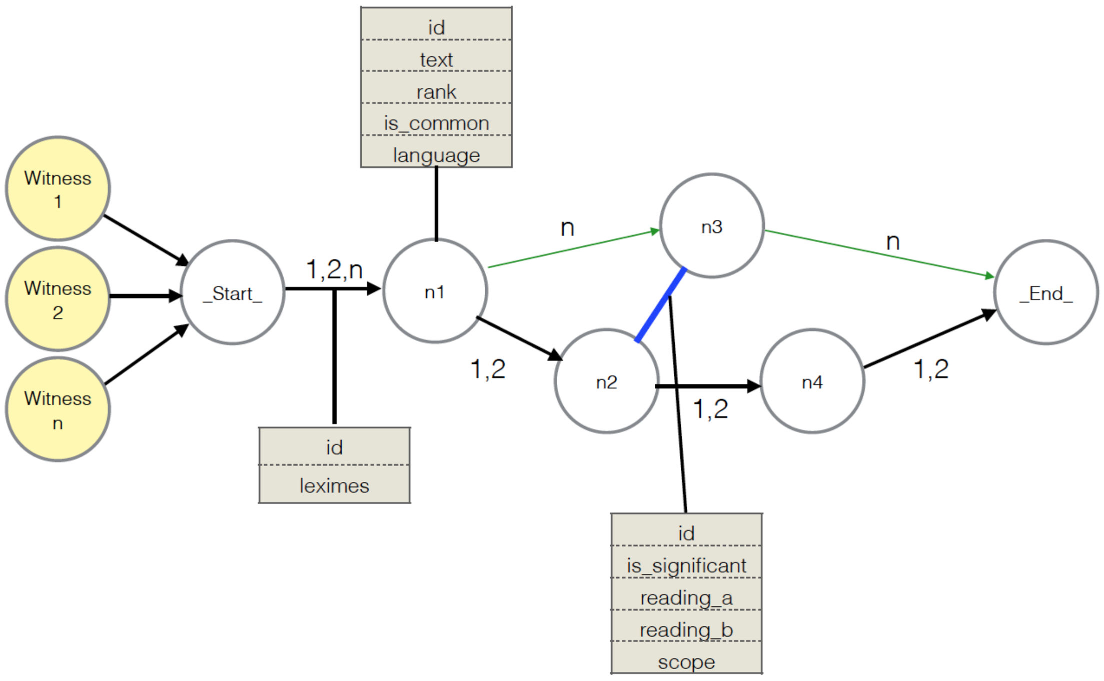
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

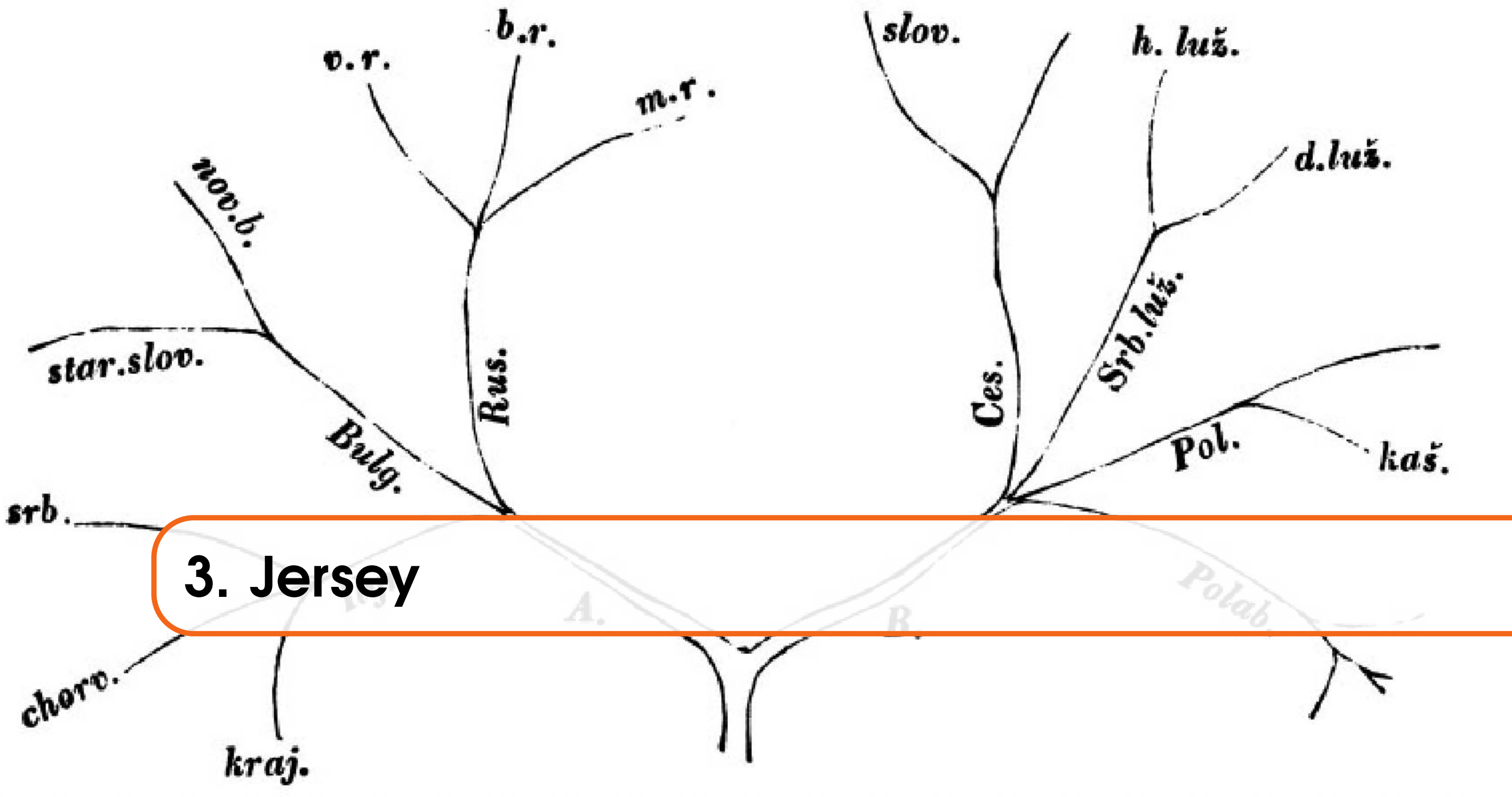


2. Database (neo4j)

2.1 Structure

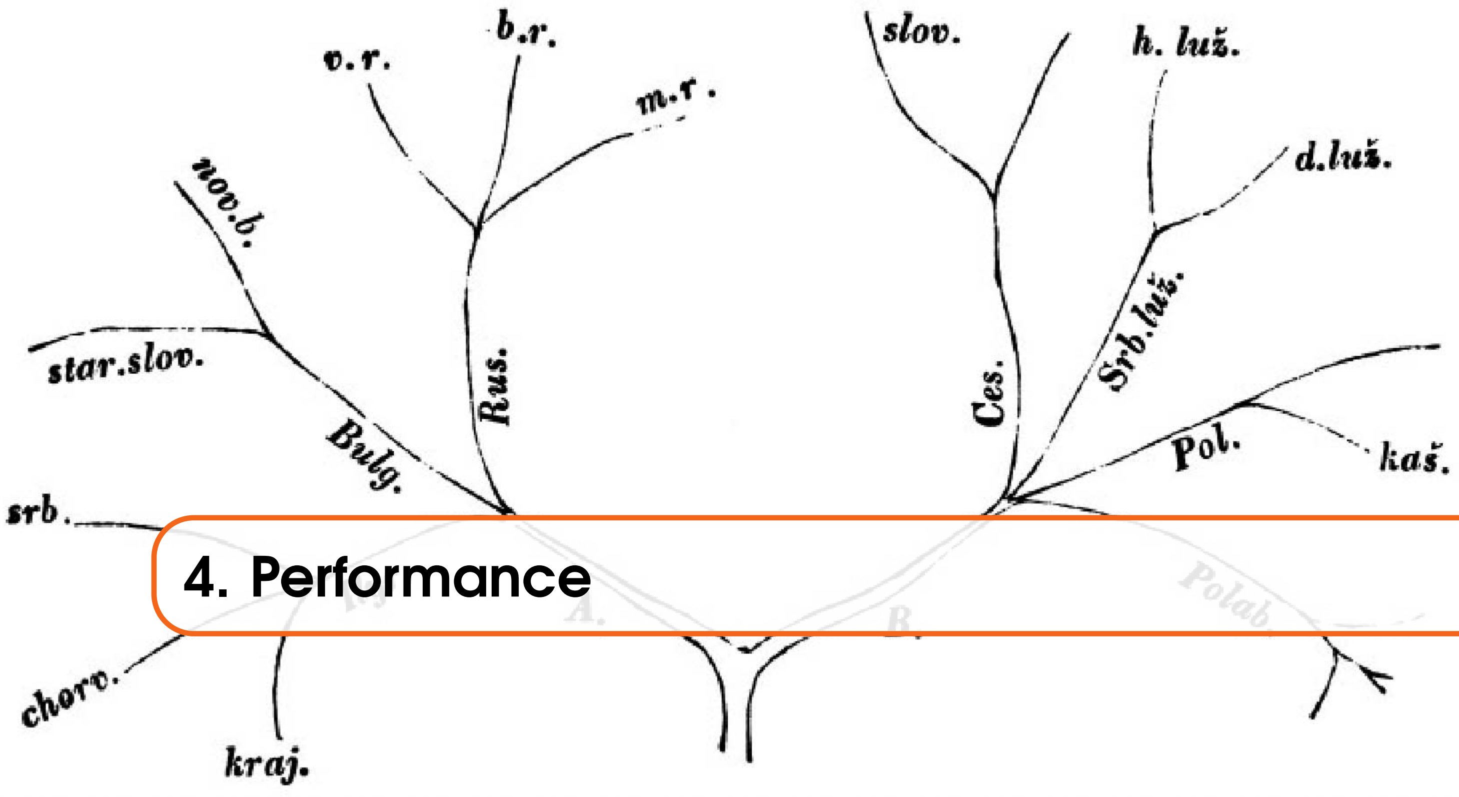



id
text
rank
is_common
language
Witness 1
Witness 2
Witness n
Start
1,2,n
n1
n
n3
n
End
1,2
n2
n4
1,2
1,2
id
leximes
id
is_significant
reading_a
reading_b
scope



3. Jersey

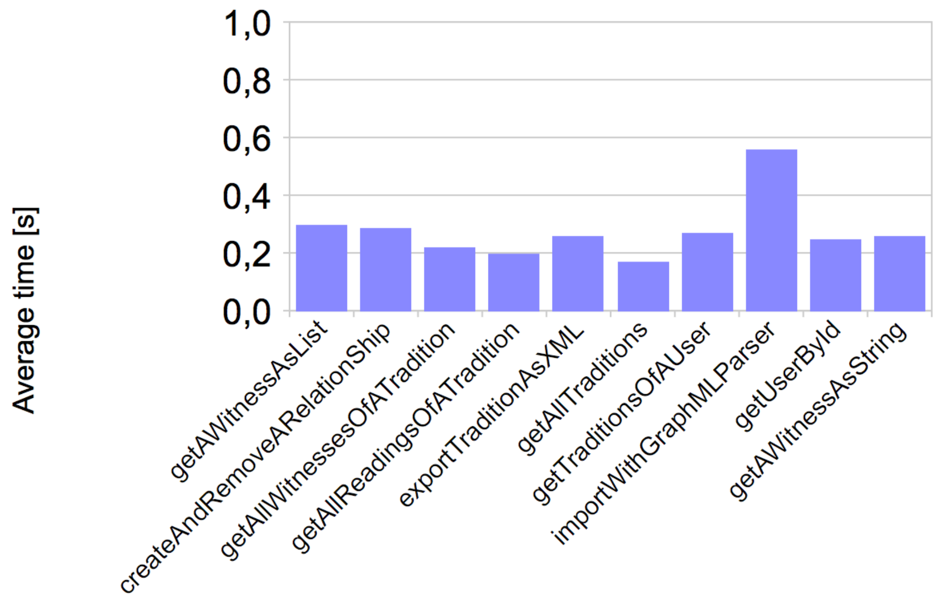
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.



4. Performance

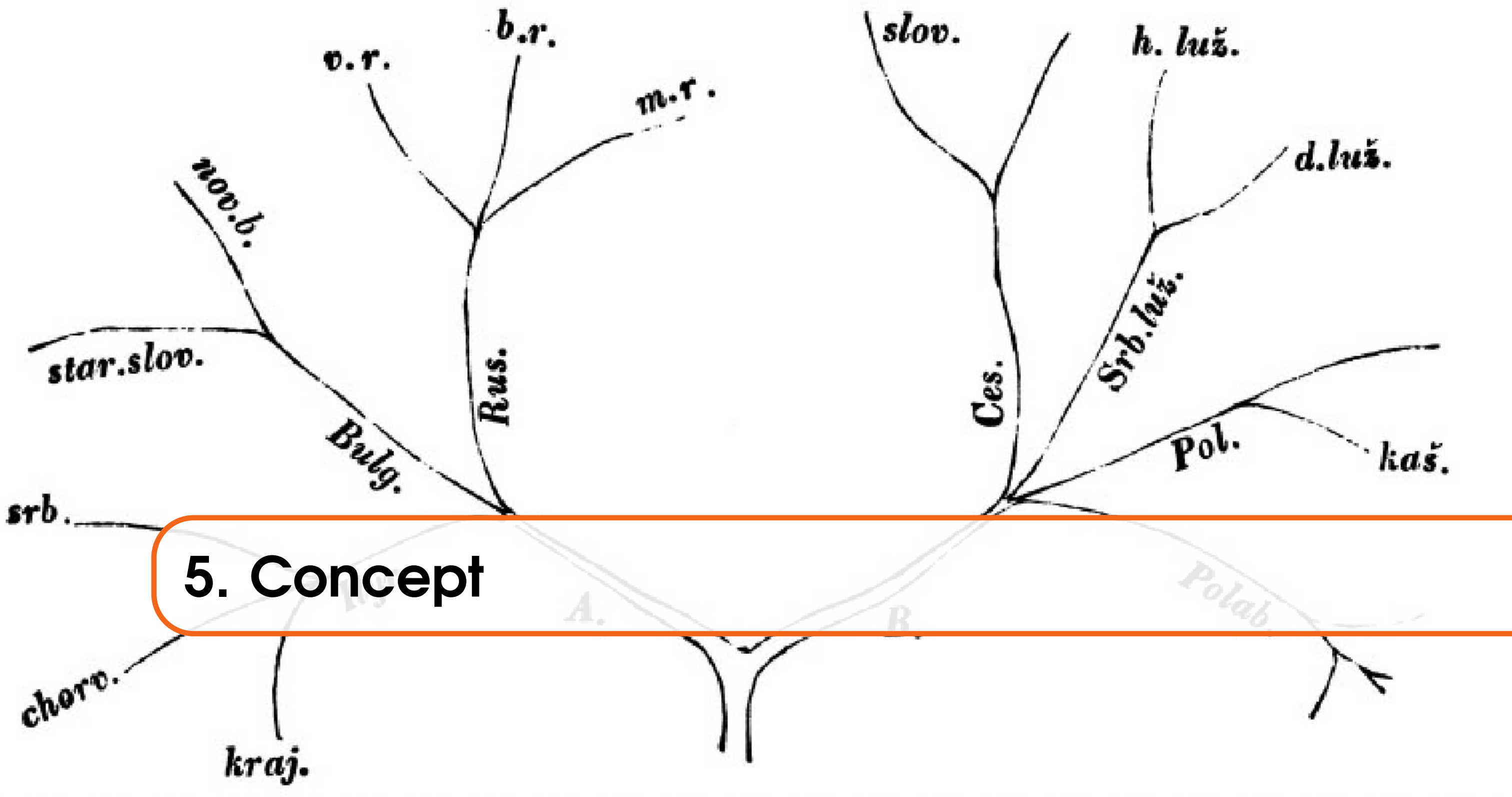
The performance is tested by benchmark tests. Please consult the chapter Benchmark Tests in the part Testing of this documentation for further technical information.

Figure 4.1: An example to show what will come here.



II

Testing



5. Concept

This chapter describes the test-concept of the Digital Humanities PSE2 Project. The testing is used to assure the quality of the project and for test driven development. All tests are written in a manner they don't have any impact on the architecture of the project. To achieve this object mocking and injection by mockito is used.

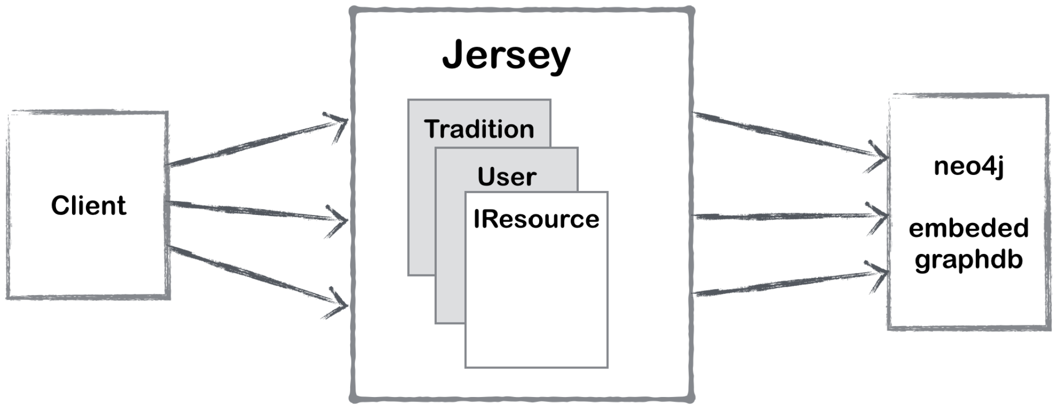
Integration Tests

Every user story is tested by an integration Test. The integration tests assures the quality of the project. The technique of integration tests is described in the Integration test chapter.

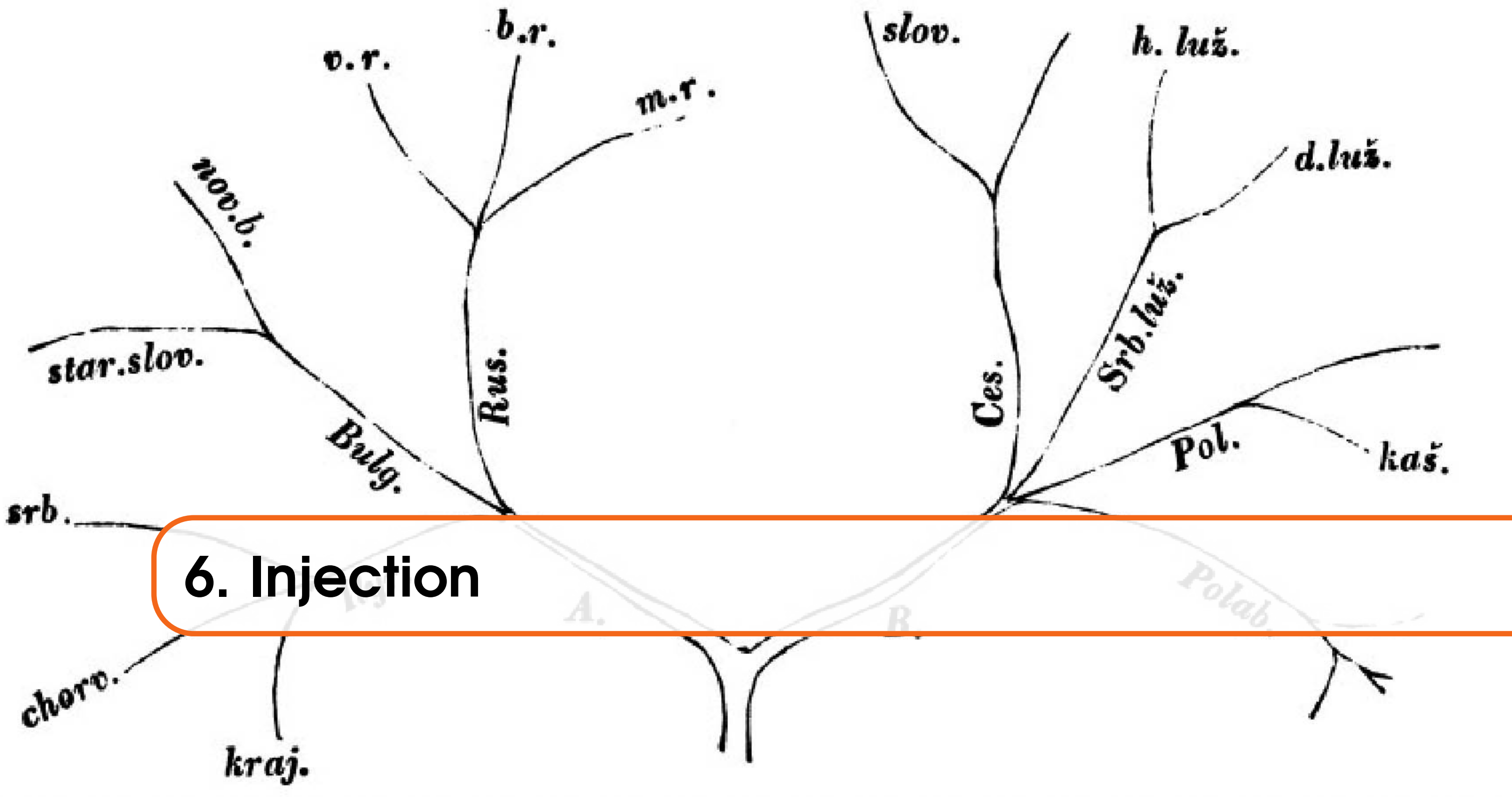
Unit Tests

Unit tests are used for test driven development and are defined by the developer. They are only for the development process and are not referenced in quality audits.

Jersey Overview



In the productive system for every REST call jersey is instantiating the requested resource and providing the service. Each resource object has its own database service, which is closed after each call. In production a embedded neo4j GraphDatabase is used. To achieve a minimal invasive test system the productive database needs to be replaced with a test database. To change the database without test related code in the project, object injection is used.



6. Injection

Mockito is used to mock, spy and inject objects in the tests. To use Mockito with junit the following annotation has to be done directly above the class header.

```
@RunWith(MockitoJUnitRunner.class)
```

With the MockitoJUnitRunner the annotations @Mock, @Spy and @InjectMocks can be used. To inject an GraphDatabaseService, the GraphDatabaseFactory is mocked and the newEmbedded-Database method overwritten that it returns an impermanent database.

```
@Mock
protected GraphDatabaseFactory mockDbFactory =
    new GraphDatabaseFactory();
```

The Mock annotation creates a Mock object. This is needed to overwrite the newEmbeddedDatabase method.

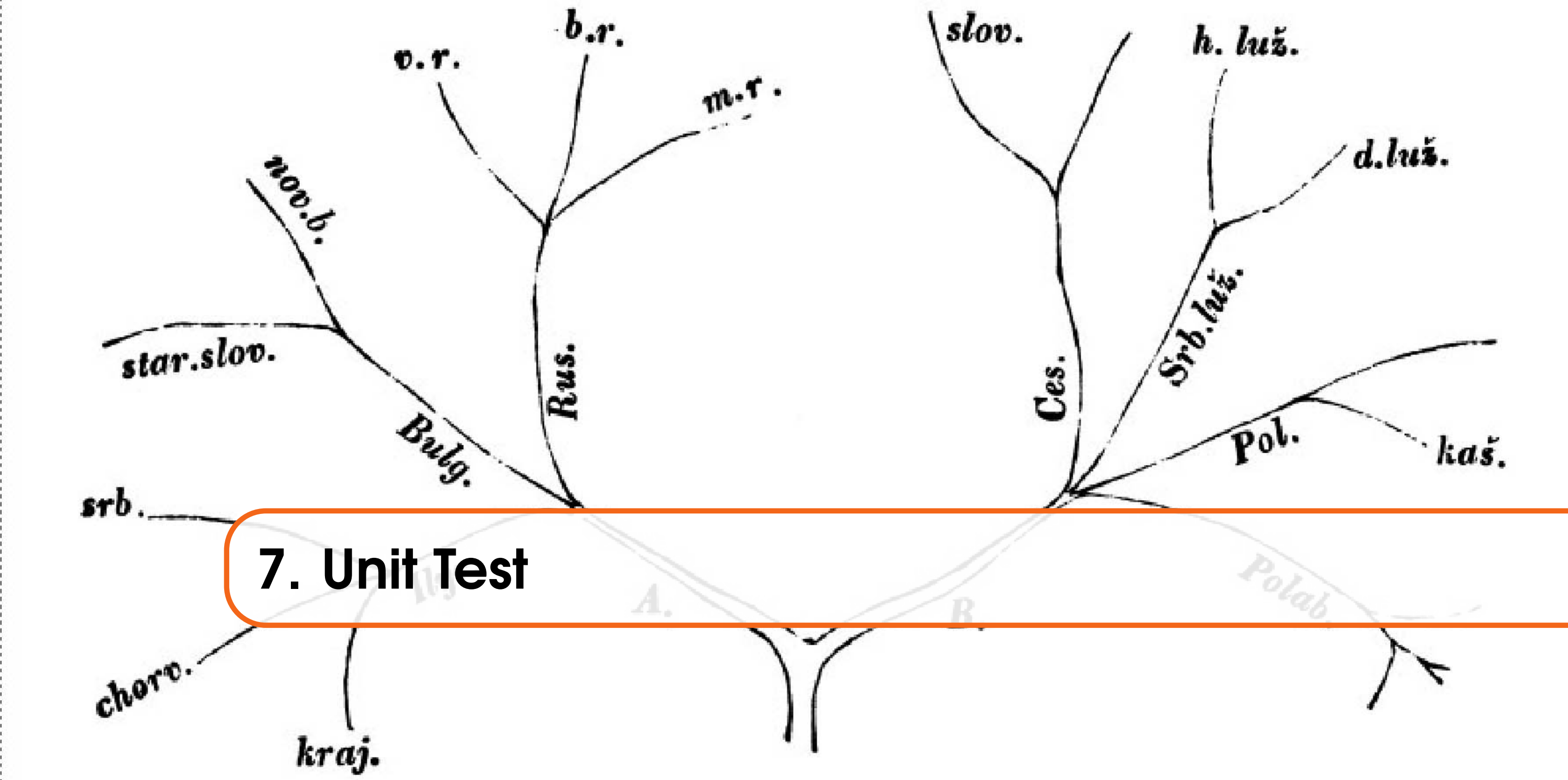
```
Mockito.when(mockDbFactory.newEmbeddedDatabase(Matchers.anyString()))
          .thenReturn(mockDbService);
```

Where mockDbService is a spy object. This is needed to suppress each shutdown call of this object. This is necessary because the resource is not new created for every restcall and so the database should not be closed. See more in the Integrationtest chapter.

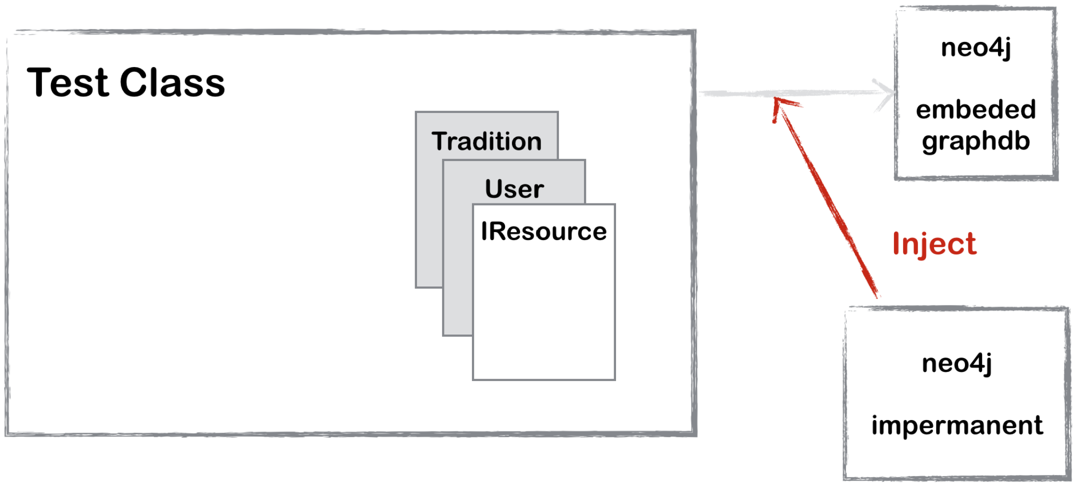
```
@Spy
protected GraphDatabaseService mockDbService =
   new TestGraphDatabaseFactory().newImpermanentDatabase();

Mockito.doNothing().when(mockDbService).shutdown();
```

And the injection of the Mockobjects into a resource

```
@InjectMocks
private User userResource;
```



7. Unit Test

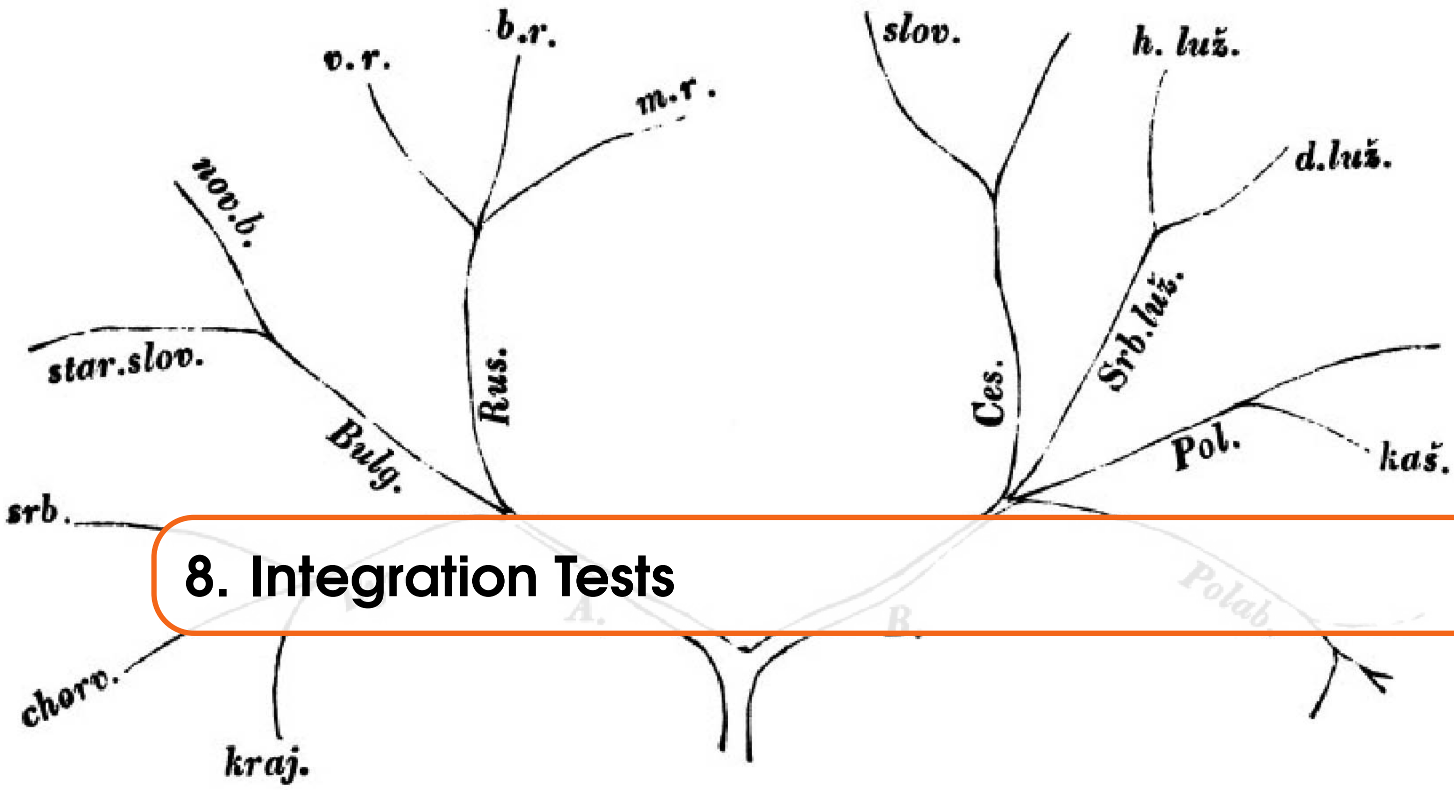


For Unit Tests the methods of the resource are called directly.

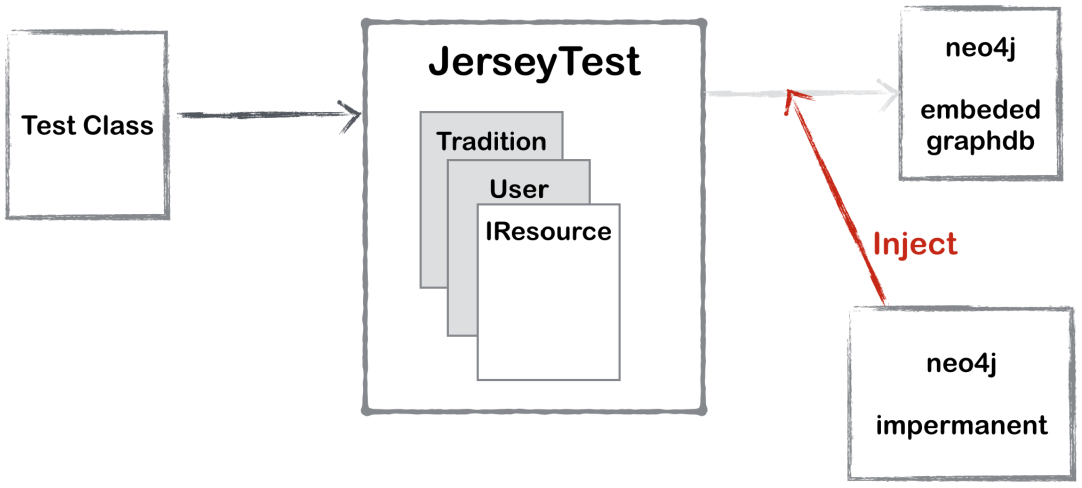
```
@Test
public void SimpleTest(){
  String actualResponse = userResource.getIt();
  assertEquals(actualResponse, "User!");
}
```

Example

https://github.com/tohotforice/PSE2_DH/blob/e364fcb0c164981281c5799a6bf9f9f9ea5eb503/stemmarest/src/test/java/net/stemmaweb/stemmaserver/UserUnitTest.java



8. Integration Tests



To inject objects into a resource it is mandatory that the resource is created statically at the place the injection is done. This is not possible when the resources are instantiated when a REST call occurs. To solve this JerseyTestServerFactory creates a server where already instantiated resources can be registered. To start a JerseyTestServer a global JerseyTest has to be created.

```
private JerseyTest jerseyTest;
```

The JerseyTestServerFactory creates a JerseyTest with already instantiated resources. This is necessary to inject the mock objects. Multiple resources can be added by chaining .addResource(..).addResource()

```
jerseyTest = JerseyTestServerFactory.newJerseyTestServer()
        .addResource(userResource).create();
jerseyTest.setUp();
```

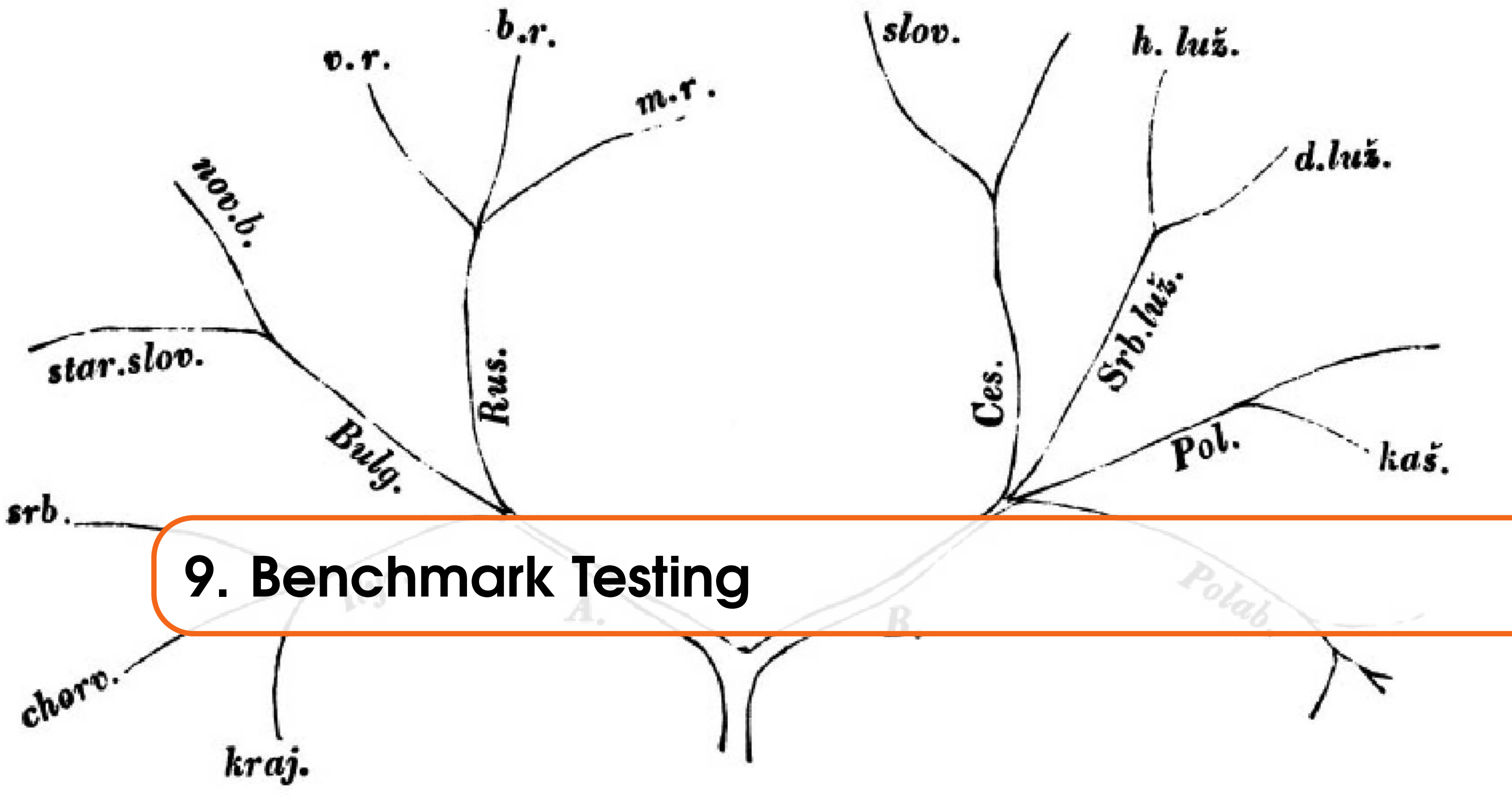
The test is done by calling a webresource of jerseyTest

```
@Test
```

```
public void SimpleTest(){
  String actualResponse = jerseyTest.resource()
      .path("/user").get(String.class);
  assertEquals(actualResponse, "User!");
}
```

Example

https://github.com/tohotforice/PSE2_DH/blob/e364fcb0c164981281c5799a6bf9f9f9ea5eb503/stemmarest/src/test/java/net/stemmaweb/stemmaserver/UserTest.java



9. Benchmark Testing

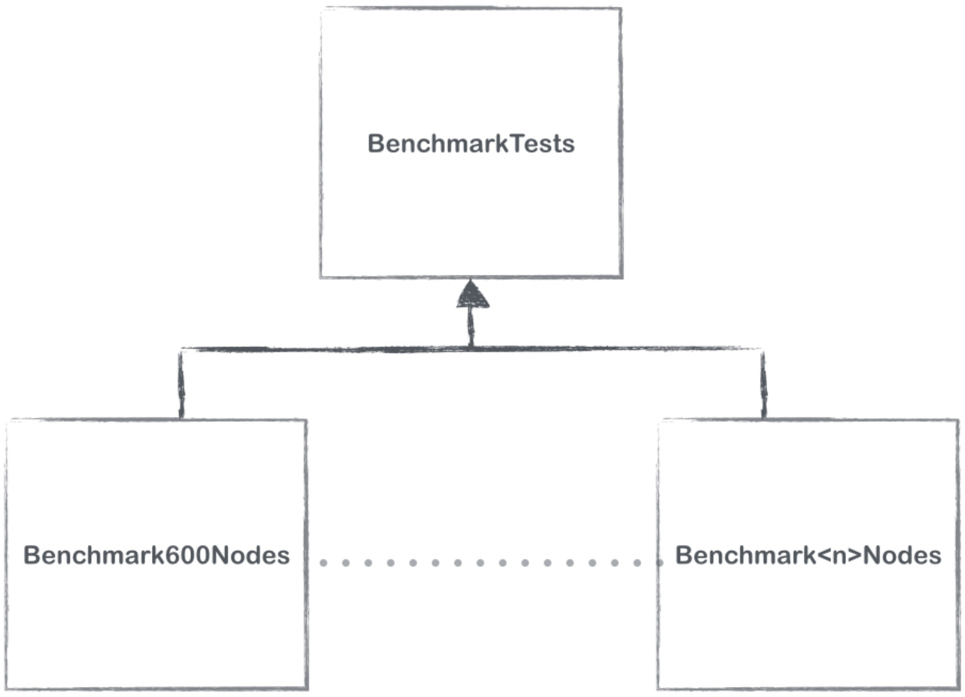
A main goal of the PSE2 stemmarest project is a good performance compared to the previous RESTful service. To measure the performance benchmark testing is needed. A benchmark test basically calls the RESTful service multiple times and measure the response time. To achieve this *com.carrotsearch.junitbenchmarks* a handy JUnit benchmark test suite is used. This JUnitbenchmarks measure the time which is used to execute a test and can generate visual representations of the measurement.
For the benchmark testing it is of interest to have a variety of different databases. Those databases should differ in their size from small to very huge. This allows to measure the algorithms in extreme situations. To generate valid graphs only limited by diskspace the class *RandomGraphGenerator* can be used. By calling the static method role a graph is generated according to the parameters.

R Please note that the response time highly depends on the hardware the tests are running on and the actual state of Javas virtual machine.

To reduce the influence of the virtual machine before the measurements 5 warm-up calls are done to bring the virtual machine to live. The hardware which was used for testing is represented in the report.

Setup

All the classes related to the Benchmark Tests can be found in the package *net.stemmaweb.stemmaserver.benachmarktests*. The class BenchmarkTests contains all Tests. The classes Benchmark<n>Nodes contain the database generation. Here is configured how many nodes the database has. This are also the classes which are run with JUnit test. BenchmarkTests cant be run as a JUnit test as it is a abstract class. s



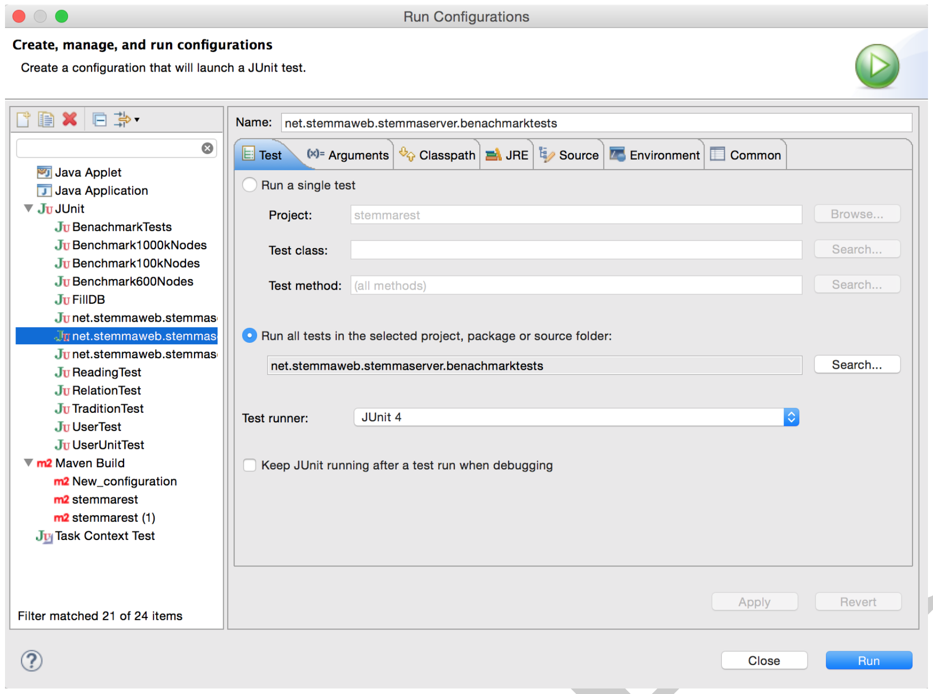
Tests are simply implemented in the BenchmarkTests class with the @Test annotation. It is best practice only to implement the restcall in this method and only test if Response.Status is OK. This assures that as less as possible overhead time is measured. And the integration- and JUnit tests should be done on a other place.

R JUnitBenchmarks measures the time to execute (@Before, @Test, @After). Heavy operations which should not be measured can be done in @BeforeClass and @AfterClass.

To create a new database test-environment copy the class Benchmark600Nodes and rename it to the count of Nodes that should be inserted. In the class itself only two small adjustments need to be done. First change the name of the report file *@BenchmarkMethodChart(filePrefix = "benchmark/benchmark-600Nodes")*. Second adjust the properties of the database which should be generated *rgg.role(db, 2, 1, 3, 100);*. role(databaseService, cardinalityOfUsers, cardinalityOfTraditionsPerUser, cardinalityOfWitnessesPerTradition, degreeOfTheTraditionGraphs)

Run Benchmarktests

The Benchmarktests can be run as every JUnit test. But to generate the report an argument needs to be passed by. Create a JUnit Test as follows:


Run Configurations

Create, manage, and run configurations

Create a configuration that will launch a JUnit test.

- Java Applet
- Java Application
- JUnit
  - BenachmarkTests
  - Benchmark1000kNodes
  - Benchmark100kNodes
  - Benchmark600Nodes
  - FillDB
  - net.stemmaweb.stemmas
  - net.stemmaweb.stemmas
  - net.stemmaweb.stemmas
  - ReadingTest
  - RelationTest
  - TraditionTest
  - UserTest
  - UserUnitTest
- Maven Build
  - New_configuration
  - stemmarest
  - stemmarest (1)
- Task Context Test

Filter matched 21 of 24 items

Name: net.stemmaweb.stemmaserver.benachmarktests

Test | Arguments | Classpath | JRE | Source | Environment | Common

Run a single test

Project: stemmarest

Test class:

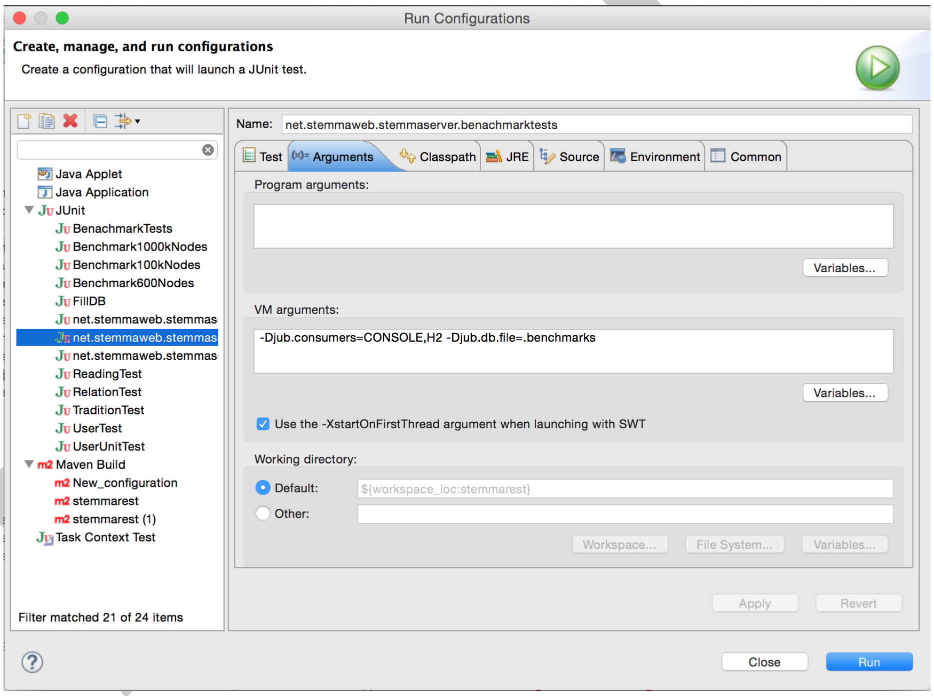
Test method: (all methods)

Run all tests in the selected project, package or source folder:

net.stemmaweb.stemmaserver.benachmarktests

Test runner: JUnit 4

Keep JUnit running after a test run when debugging


Run Configurations

Create, manage, and run configurations

Create a configuration that will launch a JUnit test.

Name: net.stemmaweb.stemmaserver.benachmarktests

Test | Arguments | Classpath | JRE | Source | Environment | Common

Program arguments:

VM arguments:

```
-Djub.consumers=CONSOLE,H2 -Djub.db.file=.benchmarks
```

[✓] Use the -XstartOnFirstThread argument when launching with SWT

Working directory:

Default: ${workspace_loc:stemmarest}

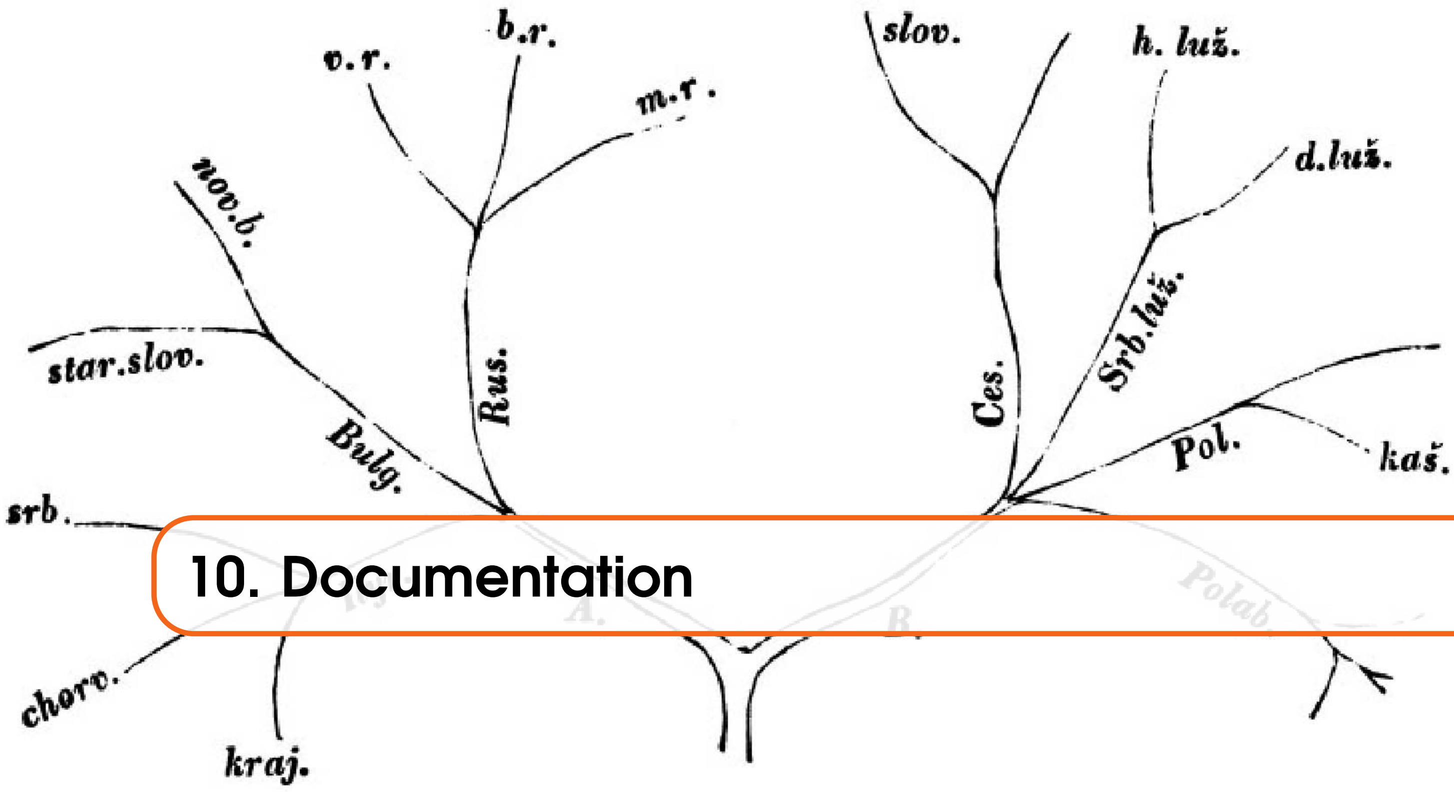
Other:

On the tab Arguments *-Djub.consumers=CONSOLE,H2 -Djub.db.file=.benchmarks* has to be inserted into the VM Arguments input. After the test can be run as usual. After the test execution the reports are stored under *benchmark/*.

R The execution of the tests will take some time because of the generation of huge graphs. Its recommended not to use the computer during this tests.

III

RESTful API



10. Documentation

10.1 /user

GET /user

Summary

Returns a welcome message

Parameter

Return — SUCCESS. text/plain
"User!"

10.2 /user/create

POST /user/create

Summary

Creates a user.

Parameter application/json
{ "userId":<userId>, "isPublic":<isAdmin> }

Return — CREATED. application/json
{ "userId":<userId>, "isPublic":<isAdmin> }

Return — CONFLICT. application/json
Error: A user with this id already exists

10.3 /user/{id}

GET /user/{id}

Summary

Returns the user as JSON Object

Parameter URL
Id: the user id

Return — OK. application/json
{ 'userId': <userId>, 'isAdmin': <isAdmin> }
The information about the user

Return — NOT_FOUND. application/json
The information about the user

10.4 /user/traditions/{userId}

GET /user/traditions/{userId}

Summary

List all Traditions of a user

Parameter URL
userId: the id of the user

Return — OK. application/json
{"traditions":[{"name":<traditionName> }] }

Return — NOT_FOUND. application/json
Error: A user with this id does not exist!

10.5 /textinfo/{textId}

POST /textinfo/{textId}

Summary

Update the textInfo of a tradition.

Parameter URL
textId: the id of the tradition

Parameter application/json
{ 'name': <new_name>, 'language': <new_language>, 'isPublic': <is_public>, 'ownerId': <new_ownerId> }

Return — SUCCESS. application/json
{ 'name': <new_name>, 'language': <new_language>, 'isPublic': <is_public>, 'ownerId': <new_ownerId> }
The new information of the tradition.

Return — CONFLICT. application/json
"Error: A user with this id does not exist"
If the user does not exist.

Return — NOT_FOUND. application/json
If the tradition was not found.

10.6 /tradition/witness/{tradId}

GET /tradition/witness/{tradId}

Summary

List all Witness of a tradition

Parameter URL
tradId: the id of the tradition

Return — OK. application/json

Return — NOT_FOUND. application/json
Error: A tradition with this id does not exist!

10.7 /tradition/new

POST /tradition/new

Summary

Create a new tradition.

Parameter text/plain
name: The name of the tradition

Parameter multipart/form-data
language: The language of the tradition
public: 0 if the tradition is not public 1 if the tradition is public
name: The name of the tradition
file: multipart file input stream

Return — CONFLICT. application/json
"Error: No user with this id exists"

Return — INTERNAL_SERVER_ERROR. application/json
"Error: Tradition could not be imported!" *If the server was not able to parse the input file*

Return — OK. application/json
"Tradition imported successfully"

10.8 /tradition/get/{tradId}

GET /tradition/get/{tradId}

Summary

Get a graphml of a tradition

Parameter URL
tradId: the id of the tradition

Return — OK. application/xml

Return — NOT_FOUND. application/json
Error: A tradition with this id does not exist!

10.9 /tradition/reading/{tradId}/{readId}

GET /tradition/reading/{tradId}/{readId}

Summary

Get a specific reading of a tradition

Parameter URL
tradId: the id of the tradition

Parameter URL
readId: the id of the reading

Return — OK. application/json The reading in json

Return — NOT_FOUND. application/json
Error: A tradition with this id does not exist!

Return — NOT_FOUND. application/json
Error: A reading with this id does not exist!

10.10 /tradition/duplicate/{tradId}

POST /tradition/duplicate/{tradId}

Summary

Duplicates readings in a tradition. Is the opposite method of merge.

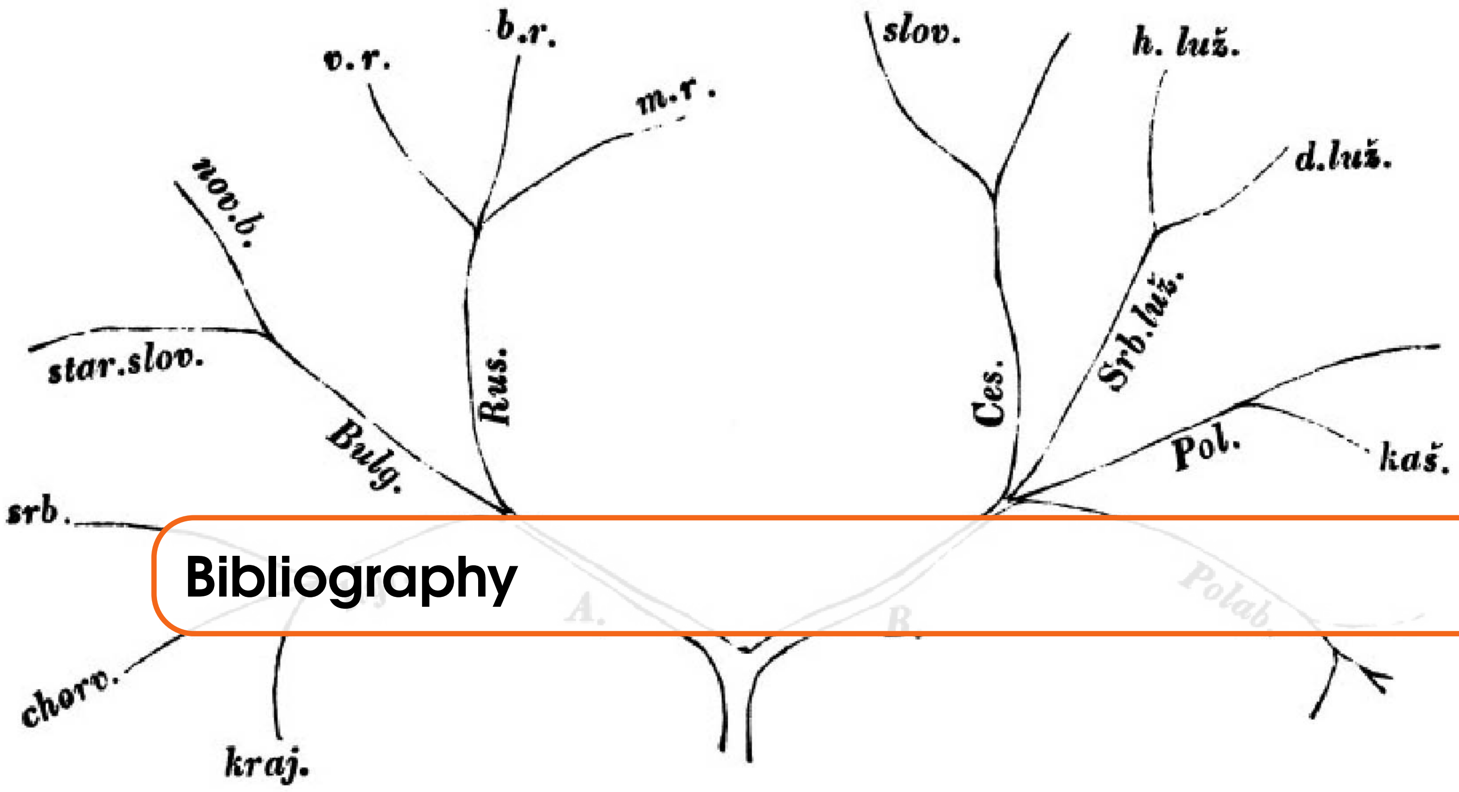
Parameter URL
tradId: the id of the tradition

Parameter application/json
{ ’readings’: [readId1, readId2], ’witnesses’: ["wit1", "wit2"]}

Return — OK. application/json
Successfully duplicated readings

Return — NOT_FOUND. application/json
If the tradition was not found.

Return — NOT_FOUND. application/json
If one of the readings was not found.



Bibliography