Politecnico di Milano

A.A. 2017-2018

Software Engineering II project

Travlendar+
Implementation & **T**esting **D**ocument **V1**

Mirko Mantovani (893784), Matteo Marziali (893904)

December 23, 2017

Contents

1 Introduction

1.1 Purpose

This document is intended to provide information about how the implementation was actually accomplished, what we really implemented in term of functionalities and requirements, as well as how the testing phase was done in detail, the test cases we performed and their outcome. It also includes different installation instructions possibilities with the main solutions to struggles and problems you may encounter during this phase.

1.2 Definition and Acronyms

1.2.1 Definitions

- **Travel:** a travel is any suggested path that goes from the starting point to the meeting location.
- **Route:** this term is used as a synonym of travel.
- **Warning:** warning is the word used to define the conflict between two meetings.
- **Conflict:** a conflict between two or more meetings is what enables the creation of a warning, it means that the set of meetings in conflict are scheduled too close in time in order for the user to be able to attend them all in time.
- **Calendar:** the calendar contains the list of meetings and is grouped by day.
- **Meeting:** is an important keyword of the application, it includes all the informations of an appointment.

1.2.2 Acronyms

- **API**: Application Programming Interface
- **JEE**: Java Enterprise Edition
- **EJB**: Enterprise Java Bean
- **JPA**: Java Persistence API
- **JSP**: Java Server Page
- **JSTL**: JavaServer Pages Standard Tag Library
- **EL**: Expression Language

1.3 Revision

1.4 Document Structure

This document is structured in seven sections, here is an overview of the contents of each and every one:

- **Introduction**: This section provides a general introduction and overview of the document.
- **Implemented functionalities**: This section provides the implemented functionalities in detail, explaining what an user can and cannot do, what the application itself allows and takes into account while performing its tasks.
- **Adopted development frameworks**: This section states the programming languages, the frameworks, middlewares and APIs we used to implement the application.
- **Source Code structure**: In this section the adopted programming style will be explained, the subdivision in packages and the code structure will be part of it.
- **Testing**: This section will contain the various test cases we performed and their outcome.
- **Installation instructions**: Here you can find all the instructions in detail about the installation process of Travlendar+.
- **Appendix**: Here we provide information about the used software and the effort spent to redact this document.

2 Implemented functionalities

The set of functionalities we actually implemented with respect to the ones we initially defined are:

2.1 [F1] Signup and Login

Travlendar+ users must sign up the first time they intend to use the App, further usages of the app will require a login to access all its functionalities. Logout which will invalidate the current session and take the user to the login page is also present.

2.2 [F2] Meeting creation

This is the most important function of the app, it allows to generate an event related to an appointment. It requires the user to define all the details such as date, time, location.

2.3 [F3] Preferences set up

An important feature of Travlendar+ consists in allowing the user to filter out specific routes depending on some constraints about the travel, or to set break-dedicated time slots. In particular the preferences we implemented are:

- Minimize carbon footprint option, the result of flagging this option is that if possible (there exist at least a route and the travel means constraints are satisfied) only walking, cycling and public transportations routes will be taken into consideration.
- Avoid tolls option which will only affect driving mode, only routes without tolls will be considered.
- Avoid motorways option which will only affect driving mode, only routes without highways (motorways) will be considered.
- Setting up a maximum walking distance and taking into consideration only the routes which satisfy this constraint
- Setting up a maximum cycling distance and taking into consideration only the routes which satisfy this constraint
- Setting up a time from which the routes involving public transportations will be discarded, in particular, the User can select the time between the range 18:00 and 5:30, from that time till the morning any route involving public transportations will be avoided.
- Specifying the travel means the user intends to use for his travels, possible travel means are:

    - Owned car
    - Shared car
    - Owned bike
    - Shared bike
    - Walking
    - Public transports

2.4 [F4] Warnings management

In case a warning is generated by the system due to a possible overlap among two or more meetings, the user must solve the warning. In other words, the user has to decide whether he wants to ignore the warning notification or he intends to modify some meetings to be sure that he can reach and participate to all his appointments. In particular the warning are created if:

- There exists a break in the database and the user adds a meeting that makes the break impossible to actually be scheduled in the allowed time, this case happens if the overlapping meetings don't leave a free interval within the break range which is greater or equal to the break minimum duration.
- The minimum amount of time in order to get to the previous meeting location to the inserted one exceeds the interval there is between the ending time of the previous meeting and the starting time of the newly added meeting.
- The added meeting partially or totally overlaps in any possible combination with one or more meetings already present in the database. In this case a warning containing all the involved meeting is created

Notice that there should not be a meeting appearing in more than one warning, if that is the case a cleaner method is periodically invoked in order to mantain this consistency, there can also exist warning containing one or more breaks and one or more meetings, however, no warning consisting in only breaks without meetings.

2.5 [F5] Route generation

When requested, the app is able to find a route, if possible, from the specific location the user is at in the moment to the location of the considered meeting, the suggested travel will fit and satisfy the selected preferences.

2.6 [F8] Update/Delete meeting

This function is both basic and relevant, it allows the user to customize his meetings after their creation. In particular, everything except the name (since it's part of the identification of the meeting) can be modified.

2.7 [F9] Add/Delete break

This feature at first was specified as part of the preferences, however, since we considered that this is one of the main features, we named it alone as an independent functionality. The Flexible break is a slot of time in the day to be reserved for specific purposes (such as a lunch for example), the break has a starting time and an ending time, which are the extremes times in which the break must be contained, the break itself however will only last the time specified by the Duration attribute. The day to specify for the break is in the form of the day of week (monday, tuesday, ecc.), which means the first occurrence of that day of week from the creation time. The Break could also be recurrent, which means every week in that day of the week a break will be scheduled. The break is flexible in the sense that the actual break only lasts for a time specified by the duration and Travlendar will make sure that the user has at least that free time available inbetween his meetings, if not a Warning will be generated.

2.8 [F10] Search meeting

An added functionality with respect to the ones we initially defined is the seach meeting, it allows the user to search for a meeting by typing part of the name, the result will be the page of the best match meeting page, if any.

3 Adopted development frameworks

3.1 Adopted programming languages

To implement Travlendar+ we mainly used Java. The front-end was created using HTML, css and javascript while the back-end is plain Java and for it we adopted some of the functionalities that JEE provides.

3.2 Adopted middlewares and libraries

We could say that the main middleware we used is Glassfish Web server, it provided us with the functionality of using the Java Persistence API framework in an easy way, allowing us to define and create the entities in the database and mapping them to Java classes. Other adopted frameworks are bootstrap and jquery, in order to simplify the creation of an enhanced UI/UX. The only library we used is json.simple, a library to simply parse the JSON files arrived as a response by the Google Maps APIs in order to retrieve specific and useful information.

3.3 Used APIs

The main APIs we interfaced with are Google Maps APIs, such as:

- Google Maps Distance Matrix API to: Estimate travel time and distance from origin to destination.
- Google Maps Directions API to: Calculate directions between origin to destination specifying the means of transport.
- Google Maps Place Autocomplete to: Input real addresses for meetings avoiding mispelling of addresses

4 Source Code structure

The following two subsections will be needed for any external person in order to understand the code structure and possibly modify or extend the Application.

4.1 Back end code structure

The back end code of the application is logically subdivided into four Source Packages:

- **Entities**: This package contains all the Java entity classes which, due to the JPA mapping to the database entity tables, allow us to directly manipulating persistent data by simply setting fields of the relative instances. Each of these classes has a few named queries to easily retrieve data sets from the database, fields with specifications defining the SQL type and constraints on the attributes of the database tables, and getters/setters.
- **Servlets**: This package contains all the servlets which are needed by the server in order to handle http get/post requests and responses, most of the used and significant endpoints are managed in these servlets. In our case servlets can either process data in the requests and interact with Java beans in order to make some changes in the persistent memory, or they are used as access points to JSPs and their purpose in this case is mainly that of customizing the JSP based on the User Session.
- **Session Beans**: This package contains all the EJB needed both for interacting with the database and to run concurrently some weighty and onerous algorithms such as conflict checking between meetings. The common abstract class AbstractFacade exposes methods to interact with the database by mapping the methods to the corresponding method call on the JPA EntityManager interface. One facade per entity extend the abstract one and contain a few additional methods to handle queries if needed.
- **Utils**: This package mainly contains plain Java classes with static services methods to be used by servlets and Beans.

4.2 Front end code structure

The front end of the application is primarily composed by JSPs, in which we decided not to put any scriplets in order not to have the possibility for any runtime exception to occur. We thought it would be better to have semantical incorrectness than exceptions, that is why we decided to use JSTL/EL to give dynamicity to the plain html code of the JSPs.
The JSPs html code is enriched with css and javascipt to make it more appealing and responsive. We also used and included in the source code jquery framework and bootstrap.

5 Testing

In order to evaluate the consistency and the reliability of our application we have conducted some unit tests.
The tests regard the business logic tier, the core of the application in terms of complexity and relevance of computation.
Considering that the business logic classes interacts directly with data in the database, we had to adopt a specific framework, named "Mockito" to Mock the method calls which retrieve data from the database and to specify an action to be performed in case of calls during testing execution.
It is relevant to say that we adopted this mocking framework assuming that the operations related to reading and writing data in the database are executed correctly, indeed they are not tested with a proper testing component. However, we are quite confident that this assumption is consistent because we checked it informally .
As far as the unit test are concerned, we have checked them using jUnit libraries.
Finally, we have tested also that the calls to the Google Maps APIs,in particular the Distance Matrix API. We checked that it was executed correctly and that the retrieved information was consistent.
Further details about the test cases and the related outcomes have been included in the document.

5.1 JUnit and Mockito

5.1.1 Meeting vs. Meeting Conflict

This test case has the purpose to evaluate the method to check whether two meeting are in conflict. Considering that this kind of evaluation involves two meetings at a time, we have built two meetings, we have assigned them overlapping schedules and we have tested the execution of the method 'CheckMeetingOverlaps(Meeting m)' of the class ConflictCheckerBean.java . The result of the test was consistent with the expected outcome.

5.1.2 Meeting vs. Break Conflict

This test case was made to check whether the conflict among meetings and breaks was computed correctly. Due to the fact that the comparison involves one meeting and one break at a time, we have built a break and a meeting that should have been overlapping and we tested the execution of the method 'BreakConflictChecker(Meeting m)' of the class ConflictCheckerBean.java . As we expectd, the outcome certified that the conflict was computed.

5.1.3 Flexible break rescheduling

In order to verify the flexible break rescheduling, one of the main functions of Travlendar+, we made an apposite test case. This analysis simply consists in calling the method checkReschedule(String uid) of the class ConflictChecker-Bean.java and checking that the outcome, in case the flexible rescheduling is infeasible, reports that effectively the meeting and the break are in conflict.

5.1.4 Warning existence

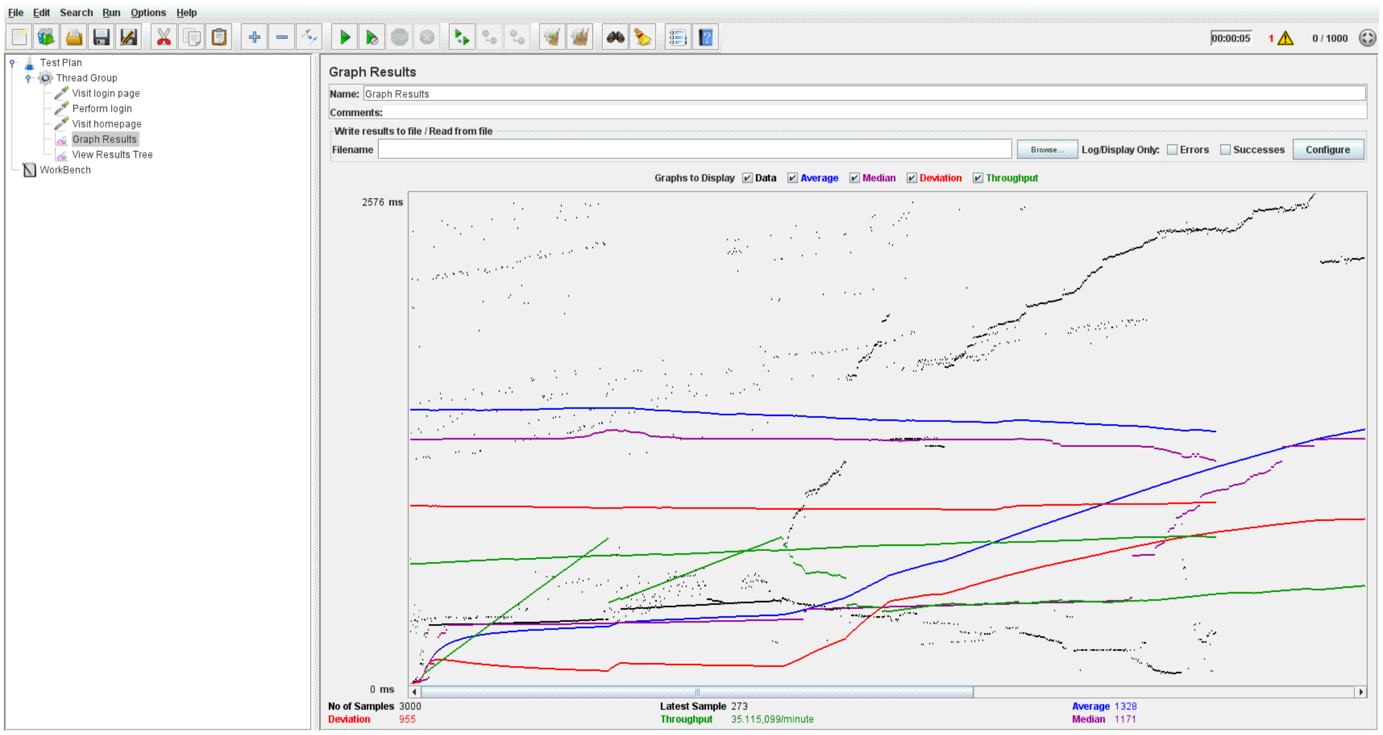
this test case regards this specific situation: a new Meeting is inserted in the db and according to its conflicts, it should be added in an already existent warning in the database. Hence, this analysis tests whether the already existent warning to be updated is retrieved correctly. To perform the test we have simulated the insertion of a warning in the database and we have created a meeting that had to be added to the existent warning. Then, we have called the method 'checkWarningExistence()' of the class ConflictCheckerBean.java and we verified that the outcome was consistent. Finally, for completeness, we replayed the test to check also a situation in which a new warning has to be created.

5.1.5 Retrieve duration

The current test case has the purpose to verify the interaction between our application and the Google Maps APIs. In particular we checked that the requested information is retrieved correctly by testing if the duration of a specific travel is correct, doing this we also tested the connection with Google servers and the correctness of our queries. In other words we called the method retrieveDuration(String origin,String destination, String uid, Date date) of the class RouteCalculatorBean().java and we passed them an origin and a destination parameter for which we already know the travel duration and we expected the same outcome.

5.2 JMeter

We also made a really trivial test to evaluate the non-functional requirements and in particular the performances given an high number of clients requesting a resource. The test was done using JMeter and the simple test consisted having a thousand users trying to access the application by performing a login and visiting the homepage, the requested are done almost all at the same time (ramp-up period equal to zero which means no delay between requests).
As you can see from the below graph the latest visited the homepage after less than 3 seconds, which suggests than the performances of the server are already a bit stressed and if the same amount of users requested a more sophisticated resource that would require computationally onerous algorithms by the server, the amount of time requires would probably not be reasonable for a user experience point of view. Notice however that, in such an application, users would mostly navigate in a burstly way, which implies that in order to have a thousand requests at the same exact time, the number of users would have to be much larger than a thousand.



Graph Results

Name: Graph Results

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Graphs to Display ☑ Data ☑ Average ☑ Median ☑ Deviation ☑ Throughput

No of Samples 3000
Deviation 955
Latest Sample 273
Throughput 35.115,099/minute
Average 1328
Median 1171

Figure 1: Graph results of 1000 users that simultaneously try to login

6 Installation Instructions

6.1 System Requirements

6.1.1 JDK

In case you don't have any version of Java Development Kit you should download its latest version from:
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

6.1.2 Glassfish Web Server

Being Travlendar+ a web application, it is essential to have a web server installed, here we provide information about the installation and deployment on Glassfish 4.1.1. You can download it from
https://javaee.github.io/glassfish/download
or alternatively, you can follow the quick installation instructions for Windows, which includes a Glassfish installation.

6.1.3 DBMS

For simplicity, we decided to use the basic DBMS provided by glassfish, Apache Derby RDBMS, thus you do not need anything else if you have installed Glassfish Server.

6.1.4 Browser

For the development and testing we always used Google Chrome as browser, we recommend installing the latest version of Chrome and we do not guarantee the absence of bugs and poor performances on other Browsers, even if they would probably work just as fine.

6.2 Quick Installation for Windows

The quick installation consists in downloading Glassfish 4.1.1 with the script to install Travlendar and the war file already in it.

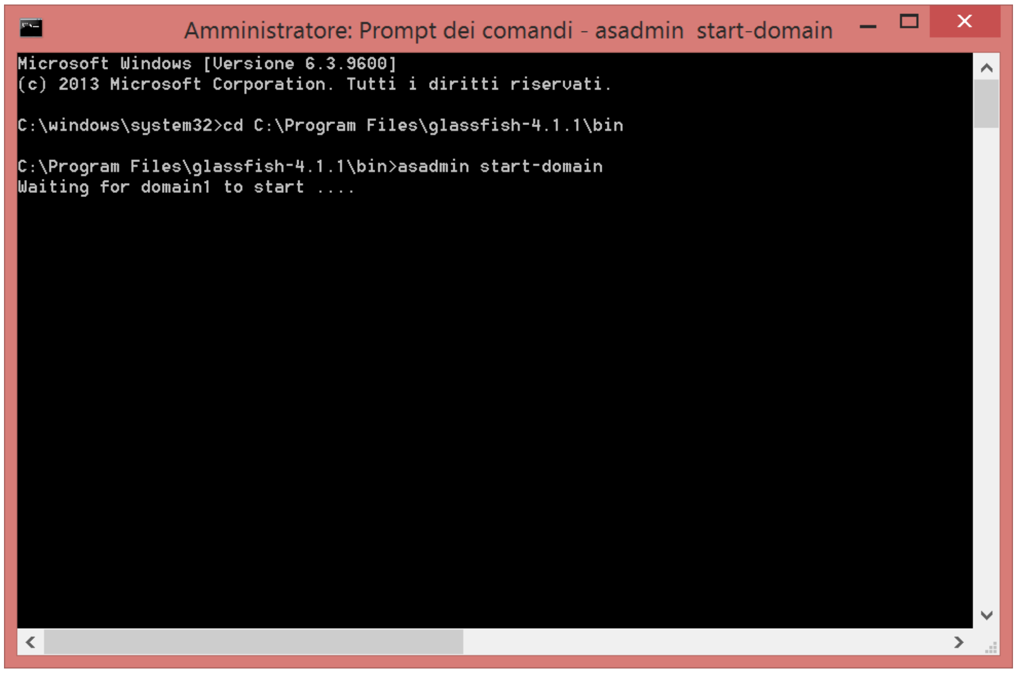
- Download the JDK if you don't have it already from http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html, usually it is stored in C:\Program Files\Java, put it there or wherever you prefer, keep in mind or save the path where you put it.
- Download the zip at: https://, unzip the folder and put the folder glassfish-4.1.1 it contains in C:\Program Files.
- Open the glassfish-4.1.1 folder and double click on installtravlendar file.
- Click yes to the popup that asks if you want to run it as administrator.

- Write down in the Command Prompt the JDK path when asked for it, for example write: C:\Program Files\Java\jdk1.8.0_121
- Press enter.
- If everything went fine the default browser should have been launched and you should see the login page of the app, signup and enjoy the app.

6.3 Manual installation - Environment Setup

6.3.1 Starting up Glassfish Server

- On Windows, open command prompt as administrator (right click on cmd.exe and click Run as administrator), on Linux or MacOS open Terminal.
- browse to Glassfish installation path and open bin folder, usual installation path on Windows: *C:\Program Files\glassfish-4.1.1\bin*, execute command *cd C:\Program Files\glassfish-4.1.1\bin*
- execute command *asadmin start-domain*



```
Microsoft Windows [Versione 6.3.9600]
(c) 2013 Microsoft Corporation. Tutti i diritti riservati.

C:\windows\system32>cd C:\Program Files\glassfish-4.1.1\bin

C:\Program Files\glassfish-4.1.1\bin>asadmin start-domain
Waiting for domain1 to start ....
```

Figure 2: Commands to execute in order to start Glassfish Server

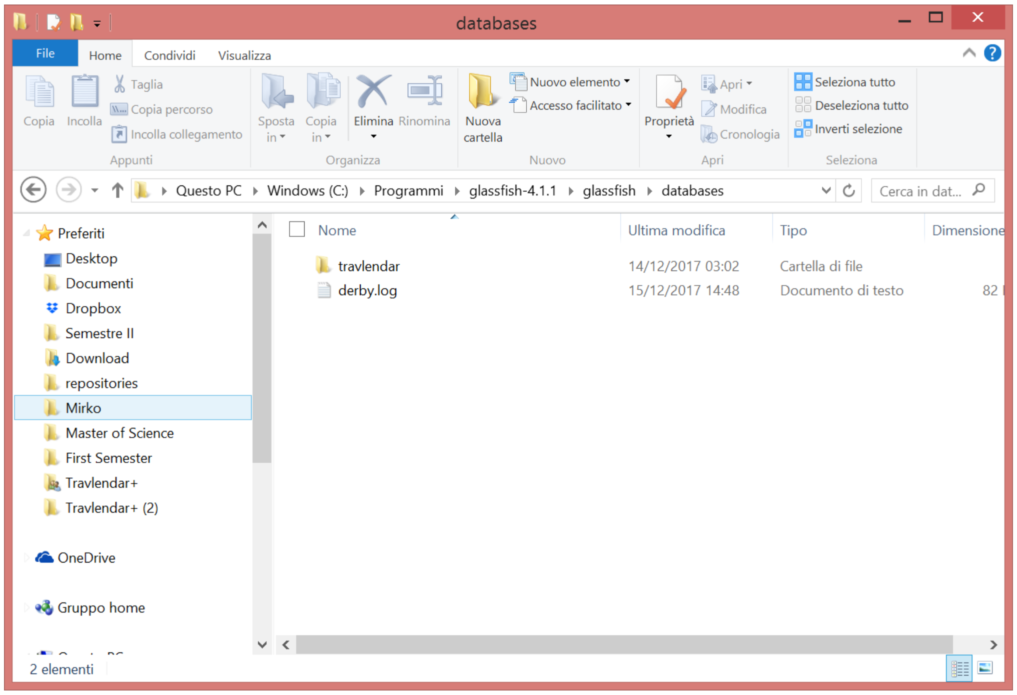
After the server has started you will be able to access the Admin Console at https://localhost:4848 and the web server at
https://localhost:8080
(If you need to stop Glassfish server just execute command *asadmin stop-domain*

6.3.2 Setting up JDK path

Sometimes an error is displayed if Glassfish does not find the JDK automatically, you can execute the command
asadmin set "server.java-config.java-home=C:\Program Files\Java\jdk1.8.0_121"
Just substitute *C:\Program Files\Java\jdk1.8.0_121* with your path to the JDK you previously downloaded.

6.3.3 Database configuration

In order to make it simpler to create the database you can download the already configured Derby database from
https://linkdropbox
Unzip it and put it in the folder: *.\glassfish\databases*
in the glassfish installation path (on Windows usually
cd C:\Program Files\glassfish-4.1.1



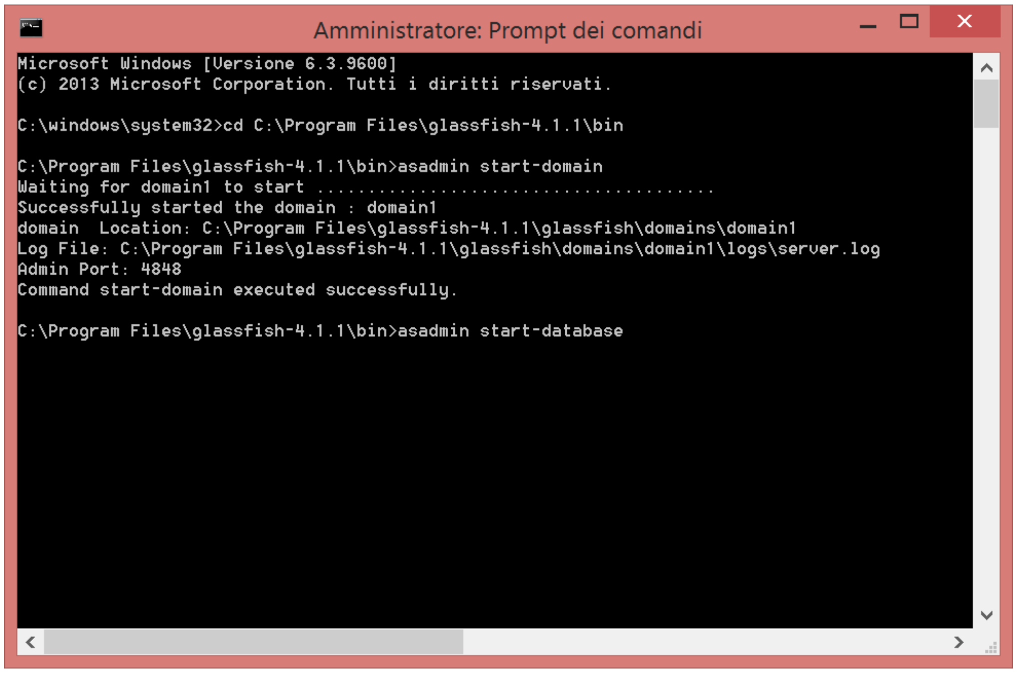
Questo PC › Windows (C:) › Programmi › glassfish-4.1.1 › glassfish › databases

Nome	Ultima modifica	Tipo	Dimensione
travlendar	14/12/2017 03:02	Cartella di file	
derby.log	15/12/2017 14:48	Documento di testo	82

Figure 3: Location in which the database should be stored

6.3.4 Starting Apache Derby DBMS

Execute command
asadmin start-database
in order to start the database.



```
Microsoft Windows [Versione 6.3.9600]
(c) 2013 Microsoft Corporation. Tutti i diritti riservati.

C:\windows\system32>cd C:\Program Files\glassfish-4.1.1\bin

C:\Program Files\glassfish-4.1.1\bin>asadmin start-domain
Waiting for domain1 to start .....................................
Successfully started the domain : domain1
domain  Location: C:\Program Files\glassfish-4.1.1\glassfish\domains\domain1
Log File: C:\Program Files\glassfish-4.1.1\glassfish\domains\domain1\logs\server.log
Admin Port: 4848
Command start-domain executed successfully.

C:\Program Files\glassfish-4.1.1\bin>asadmin start-database
```

Figure 4: Commands to execute in order to start Apache Derby DBMS

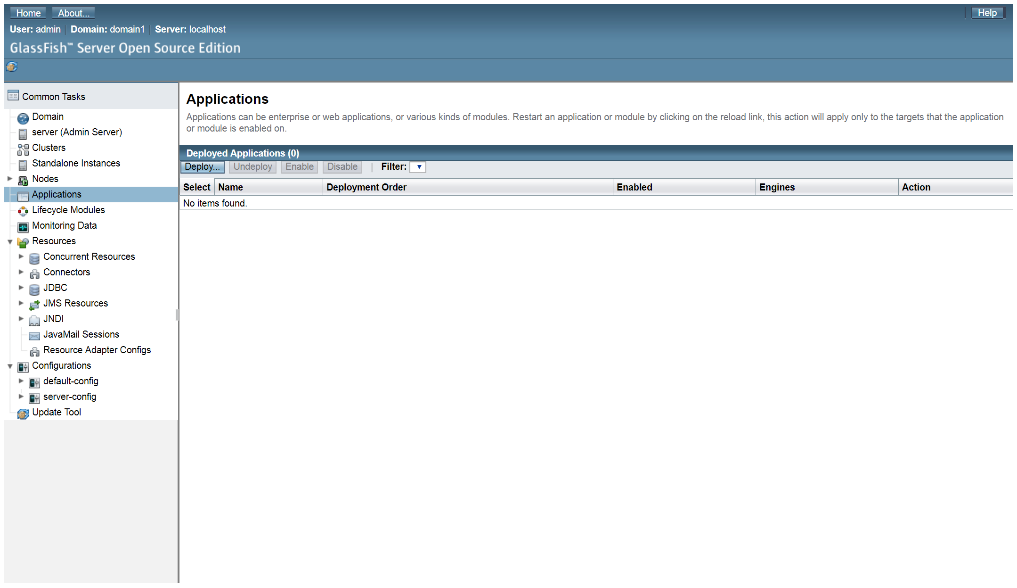
Be sure the port is the default one (1527).
If at any moment you want to stop the database just execute command
asadmin stop-database

6.4 Application Deployment

After having configured and set up the environment, download the `.war` file `Travlendar.war` at
link release in github.
You can now follow one of these ways in order to deploy the Application on Glassfish.

6.4.1 Manual deployment from Admin Console

After the server has started you will be able to access the Admin Console at https://localhost:4848
Click on `Applications` tab on the left. Then click `Deploy...` button.



Applications

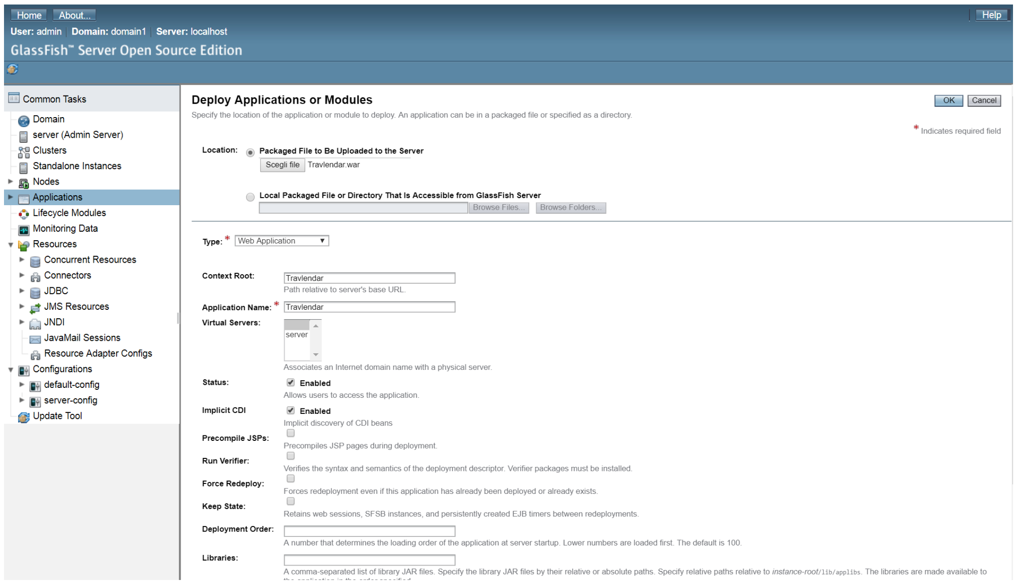
Applications can be enterprise or web applications, or various kinds of modules. Restart an application or module by clicking on the reload link, this action will apply only to the targets that the application or module is enabled on.

Deployed Applications (0)

Select	Name	Deployment Order	Enabled	Engines	Action
No items found.					

Figure 5: Glassfish admin console

Click on `Choose file` and select the `Travlendar.war` release file previously downloaded, then click ok.



Deploy Applications or Modules

Specify the location of the application or module to deploy. An application can be in a packaged file or specified as a directory.

* Indicates required field

Location: [● **Packaged File to Be Uploaded to the Server**] Scegli file Travlendar.war

○ **Local Packaged File or Directory That Is Accessible from GlassFish Server** Browse Files... Browse Folders...

Type: * Web Application

Context Root: Travlendar
Path relative to server's base URL.

Application Name: * Travlendar

Virtual Servers: server
Associates an Internet domain name with a physical server.

Status: [✓] **Enabled**
Allows users to access the application.

Implicit CDI [✓] **Enabled**
Implicit discovery of CDI beans

Precompile JSPs: [ ]
Precompiles JSP pages during deployment.

Run Verifier: [ ]
Verifies the syntax and semantics of the deployment descriptor. Verifier packages must be installed.

Force Redeploy: [ ]
Forces redeployment even if this application has already been deployed or already exists.

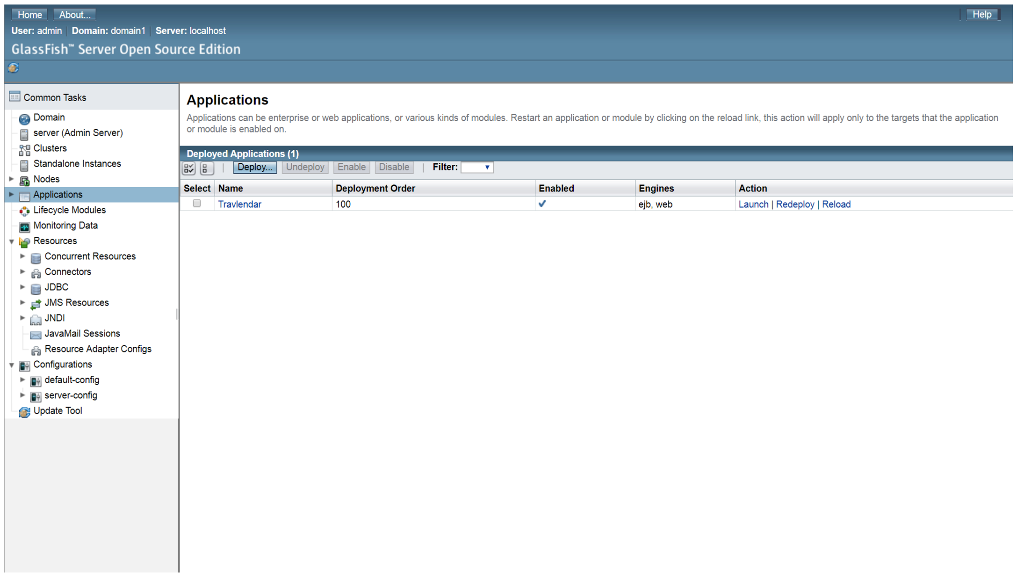
Keep State: [ ]
Retains web sessions, SFSB instances, and persistently created EJB timers between redeployments.

Deployment Order:
A number that determines the loading order of the application at server startup. Lower numbers are loaded first. The default is 100.

Libraries:
A comma-separated list of library JAR files. Specify the library JAR files by their relative or absolute paths. Specify relative paths relative to *instance-root*/`lib/applibs`. The libraries are made available to

Figure 6: Glassfish admin console

If everuthing went fine you should see something similar to this.



Applications

Applications can be enterprise or web applications, or various kinds of modules. Restart an application or module by clicking on the reload link, this action will apply only to the targets that the application or module is enabled on.

Deployed Applications (1)

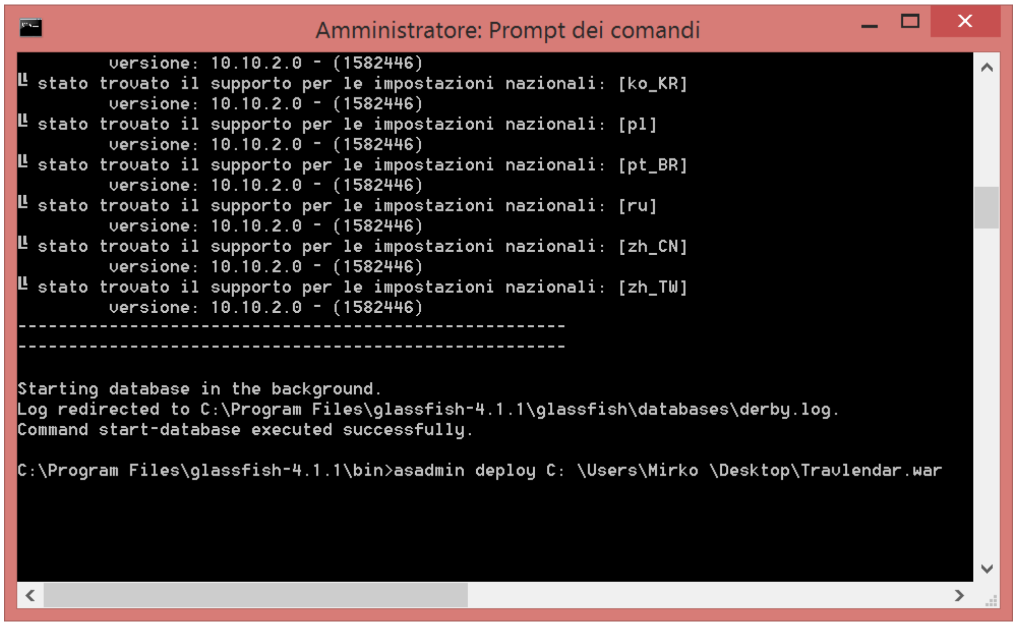
Select	Name	Deployment Order	Enabled	Engines	Action
☐	Travlendar	100	✓	ejb, web	Launch | Redeploy | Reload

Figure 7: Glassfish admin console

Just click on `Launch` to start the application.
You can access the application opening the URL:
https://localhost:8080/Travlendar

6.4.2 Manual deployment from Command Line

Supposing you already started the server and the database, and you are in the bin folder in the Glassfish installation path, just execute
asadmin deploy C: \Users\Mirko \Desktop\Travlendar.war
substituting *C: \Users\Mirko \Desktop* with your path to `Travlendar.war` release file.



```
        versione: 10.10.2.0 - (1582446)
╚ stato trovato il supporto per le impostazioni nazionali: [ko_KR]
        versione: 10.10.2.0 - (1582446)
╚ stato trovato il supporto per le impostazioni nazionali: [pl]
        versione: 10.10.2.0 - (1582446)
╚ stato trovato il supporto per le impostazioni nazionali: [pt_BR]
        versione: 10.10.2.0 - (1582446)
╚ stato trovato il supporto per le impostazioni nazionali: [ru]
        versione: 10.10.2.0 - (1582446)
╚ stato trovato il supporto per le impostazioni nazionali: [zh_CN]
        versione: 10.10.2.0 - (1582446)
╚ stato trovato il supporto per le impostazioni nazionali: [zh_TW]
        versione: 10.10.2.0 - (1582446)
------------------------------------------------------
------------------------------------------------------

Starting database in the background.
Log redirected to C:\Program Files\glassfish-4.1.1\glassfish\databases\derby.log.
Command start-database executed successfully.

C:\Program Files\glassfish-4.1.1\bin>asadmin deploy C: \Users\Mirko \Desktop\Travlendar.war
```

Figure 8: Commands to execute in order to Deploy the application on Glassfish Server

You can now access the application by executing
start http://localhost:8080/Travlendar/ on Windows, *open http://localhost:8080/Travlendar/* on MacOS X, or *xdg-open http://localhost:8080/Travlendar/* on Linux, or simply open the browser and type *localhost:8080/Travlendar* in the URL bar and press Enter.

6.4.3 Autodeployment

Just put the `Travlendar.war` file in *¡GlassFish-Installation-Path¿/domains/domain1/autodeploy* and restart the server.

6.5 Running the app

Once the deployment is finished you can access the Application at *localhost:8080/Travlendar* on a browser in your local machine, at *192.168.1.3:8080/Travlendar* on a device connected to the same LAN (just replace *192.168.1.3* with the private IP address of the machine the server is running on.
You can even access the application from a remote device, you just need to open the port 8080 of the router of your LAN by creating a virtual server and NATting the external access onto the private IP address of the machine your server is running on (private IP should be configured as static). Then you can access from wherever you want just by going to *xx.xx.xx.xx:8080/Travlendar*, replace xx.xx.xx.xx with the public IP address of your router.

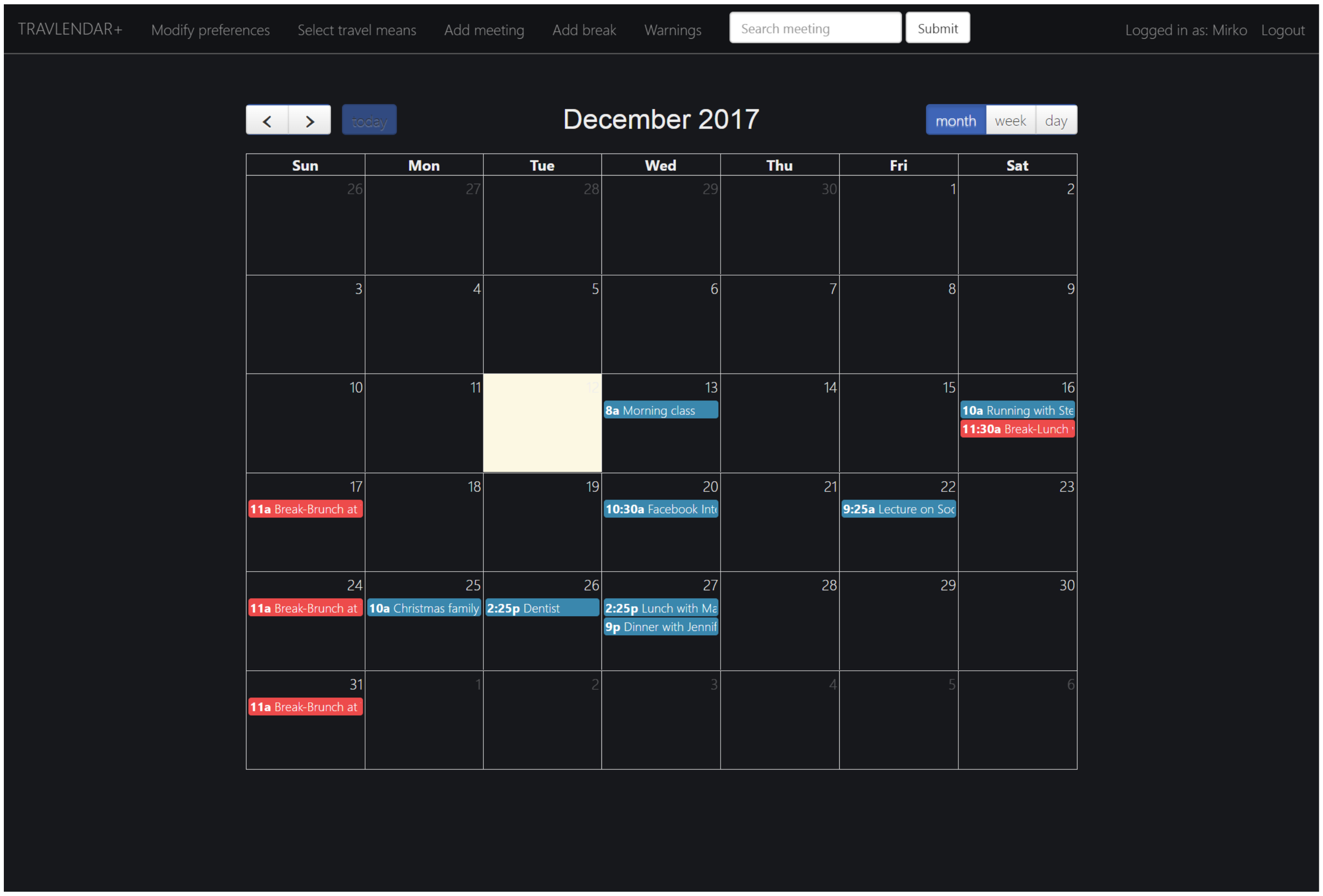


Figure 9: Travlendar Homepage

6.6 Possible issues and solutions

If you encounter any problem in the deployment regarding a database connectivity problem you could download Netbeans IDE (which includes Glassfish and Derby installation) and replace the Environment setup steps by creating a database with databasename = travlendar, name = mirko, password = mirko. Start glassfish server from there and then pass to the deployment phase as explained in these instructions.
If also this does not let you deploy the app as last chance you could clone the travlendar repository, import it as a project in netbeans build and clean the project, and run it.

7 Appendix

7.1 Used software

Task	Software
Edit and compile LaTeX code	TeXmaker, TeXstudio
Development IDE	Netbeans
Application server	Glassfish 4.1.1
DBMS	Java DB (Derby)
Performance Testing	JMeter
Unit Testing	JUnit
Testing	Mockito

7.2 Effort spent

- Matteo Marziali working hours: $\approxeq$ 100 hours
- Mirko Mantovani working hours: $\approxeq$ 100 hours