|  |
| --- |
| **Functional Requirement Document** |
| Prepared for  **CGI Group Inc**.  Project  **Scoring Database**  Prepared by  Jiri Ulrich  Contributors  **Stefan Bertos**  **Michal Stepan** |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description |
| 0.1 | 2015-07-01 | Jiri Ulrich | Draft version |
| 0.2 | 2015-09-01 | Jiri Ulrich | Updated section |
| 0.3 | 2015-12-01 | Stefan Bertos | Updated requirements |
| 0.9 | 2016-02-13 | Stefan Bertos | Switch to CGI template |
| 0.10 | 2016-02-15 | Radim Kostka | Review |
| 0.11 | 2016-02-15 | Stefan Bertos | Added section with abbreviations |
| 0.12 | 2016-02-22 | Stefan Bertos | Updated planning details |
| 1.0 | 2016-03-29 | Stefan Bertos | Finalized version |
| 2.0 | 2017-04-28 | Michal Stepan | Update for 2017 |
| 2.1 | 2018-08-22 | Alena Marikova, Lukas Sykora, Zdenek Vyskocil | Update for 2018 |

Table of Contents

[1 Executive Summary 5](#_Toc443330038)

[2 Introduction 6](#_Toc443330039)

[3 Purpose 7](#_Toc443330040)

[3.1 Abbreviations 7](#_Toc443330041)

[4 Future State Business Process 9](#_Toc443330042)

[4.1 Process / Business Scenario - Collect data 10](#_Toc443330043)

[4.2 Process / Business Scenario – Calculate PD 12](#_Toc443330044)

[4.3 Process / Business Scenario – Distribute data to downstream systems 14](#_Toc443330045)

[5 General System-Wide Requirements 15](#_Toc443330046)

[5.1 Global Support 15](#_Toc443330047)

[5.1.1 Multi-Language 15](#_Toc443330048)

[5.1.2 Multi-Currency 15](#_Toc443330049)

[5.2 Data Management 15](#_Toc443330050)

[5.2.1 Data Import 15](#_Toc443330051)

[5.2.2 Data Export 15](#_Toc443330052)

[5.2.3 Data Cleansing 15](#_Toc443330053)

[5.3 Organization Structure 15](#_Toc443330054)

[5.3.1 Business Units 15](#_Toc443330055)

[5.3.2 Teams 15](#_Toc443330056)

[5.3.3 Users 15](#_Toc443330057)

[5.4 Security 15](#_Toc443330058)

[5.4.1 Roles 15](#_Toc443330059)

[5.4.2 Privileges 15](#_Toc443330060)

[5.4.3 Views and Forms 15](#_Toc443330061)

[6 Analytics 16](#_Toc443330062)

[6.1 P1 - Collect data 16](#_Toc443330063)

[6.1.1 FR1 - Database 16](#_Toc443330064)

[Data Relationship and Constraints 16](#_Toc443330065)

[Data Input 16](#_Toc443330066)

[Data Collection Requirements 16](#_Toc443330067)

[Database Structure 16](#_Toc443330068)

[6.1.2 FR2 - Input files interface 17](#_Toc443330069)

[6.1.3 FR3 - Loader application 17](#_Toc443330070)

[6.1.4 FR4 – Data clean-up 20](#_Toc443330071)

[6.1.5 FR5 – Logging (bonus) 20](#_Toc443330072)

[6.1.6 FR6 – Web application (bonus) 21](#_Toc443330073)

[6.1.7 NFR1 - Performance 22](#_Toc443330074)

[6.1.8 NFR2 - Documentation 22](#_Toc443330075)

[6.1.9 NFR3 – Test (bonus) 22](#_Toc443330076)

[6.1.10 NFR4 - Source code repository 22](#_Toc443330077)

[6.1.11 NFR5 - Binary release packages 22](#_Toc443330078)

[6.2 P2 – Calculate PD 23](#_Toc443330079)

[6.2.1 FR7 - Daily aggregation module 23](#_Toc443330080)

[6.2.2 FR8 - Monthly aggregation module 23](#_Toc443330081)

[6.2.3 FR9 - Pd calculation module 23](#_Toc443330082)

[6.2.4 NFR6 – Performance 23](#_Toc443330083)

[6.3 P3 - Distribute data to downstream systems 23](#_Toc443330084)

[6.3.1 FR10 – Integration options 23](#_Toc443330085)

[6.3.2 NFR7 – Availability 23](#_Toc443330086)

[7 References 24](#_Toc443330087)

[7.1 Business Processes and Requirements 24](#_Toc443330088)

[8 Planning 25](#_Toc443330089)

[8.1 P1 – Collect data 25](#_Toc443330090)

[26](#_Toc443330091)

[Appendix A – Interface specification for incoming files 1](#_Toc443330092)

[Appendix B – Technical recommendations 5](#_Toc443330093)

# Executive Summary

This document is intended to outline the need for the Scoring Database solution, identified by the customer coming from banking sector. The purpose of the project is to find a correlation between transactions and account balances and fraudulent behavior and the probability of default of the account owner. In the first phase, transaction data should be collected. Information about all transactions on all accounts of the given customer is supposed to be collected and stored to database. This phase is supposed to last 24 months. In the next phase, the collected data will be evaluated and scoring module will be developed. The last phase will provide the data to the downstream systems.

# Introduction

The Functional Requirements Document (FRD) is the starting point of solution and system development and is a collaborative effort between all business and technology stakeholders.

The FRD defines “what” the user needs.

# Purpose

The purpose of the FRD is to communicate business needs in common terms to all project and technical team members to ensure that the end product meets the business objectives. It is the first phase of the Systems-Development Life Cycle.

## Abbreviations

|  |  |
| --- | --- |
| Abbreviation | Explanation |
| BC | Business customer |
| CSV | [Comma-separated values](https://en.wikipedia.org/wiki/Comma-separated_values) |
| CHAR | Character datatype |
| DDL | [Data definition language](https://en.wikipedia.org/wiki/Data_definition_language) |
| ER | [Entity relation model](https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model) |
| EUR | Euro currency |
| FR | Functional requirement |
| FRD | Functional Requirements Document |
| IBAN | [Internation Bank Account Number](https://en.wikipedia.org/wiki/International_Bank_Account_Number) |
| JAR | [Java file format](https://en.wikipedia.org/wiki/JAR_%28file_format%29) |
| JDBC | [Java database connectivity](https://en.wikipedia.org/wiki/Java_Database_Connectivity) |
| NFR | Non functional requirement |
| PC | Private customer |
| PD | [Probability of default](https://en.wikipedia.org/wiki/Probability_of_default) |
| POM | [Project object model](https://en.wikipedia.org/wiki/Apache_Maven#Project_Object_Model) |
| SDB | Scoring Database |
| SDLC | Software(Systems) Development Life Cycle |
| SQL | [Structured Query Language](https://en.wikipedia.org/wiki/SQL) |
| UML | [Unified Modeling Language](https://en.wikipedia.org/wiki/Unified_Modeling_Language) |
| XML | [Extensible Markup Language](https://en.wikipedia.org/wiki/XML) |

# Future State Business Process



Picture 1 - Future business process diagram

## Process / Business Scenario - Collect data

Transaction data should be collected. Information about all transactions on all accounts of the given customer is supposed to be collected and stored to database. This phase is supposed to last 24 months.



## Process / Business Scenario – Calculate PD

The collected data will be evaluated and scoring module will be developed to calculate PD.



## Process / Business Scenario – Distribute data to downstream systems

All the calculated data needs to be available to the downstream systems via multiple interfaces.



# General System-Wide Requirements

## Global Support

### Multi-Language

N/A

### Multi-Currency

N/A, only EUR currency is accepted

## Data Management

### Data Import

N/A

### Data Export

N/A

### Data Cleansing

N/A

## Organization Structure

### Business Units

[Wealth and investment management](https://en.wikipedia.org/wiki/Barclays_Wealth)

### Teams

Anti Money Laundering

### Users

Members of the Anti Money Laundering team

## Security

### Roles

### Privileges

### Views and Forms

# Analytics

## P1 - Collect data

### FR1 - Database

Database will be the storage for all input files.

### Data Relationship and Constraints

Customer is uniquely identified by its ID. One customer may have any number of accounts. It is also possible to have customer without accounts. Account is uniquely identified by the IBAN. Each account must belong to exactly one customer. It is not possible to have an account without customer. Transaction has to be always linked to exactly one account but account must not have any transactions. Account can have more than one transaction on given day and it can have none on the other day.

### Data Input

The information about customers, accounts and transactions is generated daily. The information about customers and transaction may be the same in subsequent days in case there was no change in customer or account information. The same customer or account may appear for example two days without any change except for the date. Transactions are always unique. Transaction with given order may appear only once. Next transaction regardless of date must have the order increased by one.

### Data Collection Requirements

One of the business requirements is that the system has to collect data on daily basis. Collected data will be later aggregated on daily and monthly bases. Output of this aggregation will be used for scoring and risk estimation.

### Database Structure

Your first task is to create database structure which will store all information needed - daily customer, account and transaction data.

Create all tables needed. Use Oracle database. Keep in mind that you have to select correct primary keys, create integrity constraints and reference integrity. The client is from financial industry and would like to have data as exact and as consistent as possible. This is conservative approach which is very relevant this industry.

If you are new to SQL, you can also use the tool (SQL Developer or similar) and create the table structure using this too. For more experienced SQL developers, we recommend creating all database structures manually by hand-written DDL script. You don’t have to care about partitioning or compression. It’s nice to put comments on columns. For easier orientation, you can also create ER model (not necessarily – but would be recommended for documentation purposes). In either case, the DDL (data definition, database setup) scripts are an important part of your solution.

For Customer, Account and Transaction tables, you are recommended to use column names as stated in Appendix A. Same names will occur also in headers of CSV file. Sticking strictly to this naming convention will make it easier for you (and us) to check for errors and problems.

### FR2 - Input files interface

Data has to be validated against the interface description which is added as Appendix A

Customer, account and transaction details will arrive daily into dedicated folder on the file system. The files will be visible and editable for your application. The data will arrive in three CSV files – one file with customers, one file with accounts and one file with transactions. In case of technical problem, it can happen that data for more days from the past will be sent at once.

First line of the file contains headers. It is a list of names (not quoted) separated by comma. There is no comma behind the last column name.

Data start on the second row. CHAR data columns are always quoted. Timestamps, integer number and decimal numbers are never quoted. Decimal places are separated by dot. Integer and decimal numbers are always in decimal format.

Simple example for the customer table:

C\_DAY,C\_ID,C\_TYPE,C\_ORIGIN\_DATE,C\_REL\_DATE,C\_LEGAL\_ID,C\_NAME

2015-08-10,123456,”P”,1990-11-22,2010-01-02,”CZ12345678”,”MyCompany s.r.o”

Complete data sample with all three files is attached.

YYYYMMDD identifies the date to which the data is related. It has to be equal to the C\_DAY or A\_DAY or T\_DAY columns inside the CSV file. If other cases, the program has to log error and stop processing of the file.

### FR3 - Loader application

We need a loader application, written in Java which we can run from command line. And a way where we could provide configuration for directory paths for input csv files.

In the second step, you have to create Java program which will be started from the command line. The program shall be compiled into JAR file with all referenced libraries. It means that it must be possible to copy the JAR to any computer with Java installed and it will run. (user should be able to use default configuration with relative path in ideal case sample data for one day).

Provided example loader application is already pre-configured with some features, which you might want to focus on. Feel free to navigate through classes of the project and read comments. Visit the [official](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/) documentation for more informations.

#### Database Configuration File

The program will read one configuration file – application.properties. This property file is placed on classpath and contains information about connection to database. Example of the configuration property file:

database.url=jdbc:oracle:thin:@//10.29.8.156:1521/xe

database.username=user1

database.password=password1

Example configuration file is already attached in the sample project.

DO NOT RENAME OR MOVE THIS FILE UNLESS YOU KNOW, WHAT ARE YOU DOING. USE YOUR PROVIDED LOGIN TO CONFIGURE THE DATABASE.

#### Path Configuration File

The program has to accept one command line argument: path and file name of XML file which will contain path to input directory and path to directory for processed files.

* INPUT PATH – Input path where incoming data can be found. The program is supposed to scan this directory for new CSV files.
* PROCESSED PATH – Processed path to which all processed (read) CSV files have to be shifted after their processing finished. This directory is meant to avoid duplicity and has a function of raw data archive.
* BAD PATH – Directory for bad files (damaged files, incorrect data)

Example configuration:

<?xml version="1.0"?>

<sdb>

<config>

<directories>

<directory type=”input”><path>/data/tcos/input</path></directory>

<directory type=”processed”><path>/data/tcos/output</path></directory>

<directory type=”bad”><path>/data/tcos/output</path></directory>

</directories>

</config>

</sdb>

Example XML file is also attached.

IMPLEMENT VALIDATION AND ERROR HANDLING FOR THIS FILE.

#### Program Startup

After start, the program shall check for the command line argument and for provided configuration file. If configuration file is missing or invalid report error must be issued.

Configration file have to be opened and configuration parameters have to be loaded. All parameters have to be verified. It must be possible to establish database connection and all paths specified in configuration file must exist.

You shall also create TCOS.BAT or TCOS.CMD file which will setup the environment and start your program just by running this BAT or CMD without any arguments.

#### Reading Data Files

In the next step, the program shall scan input directory for CSV files with customers, accounts and transactions. Just stick to these files and ignore other files.

The program has to check each timestamps in the CSV file names and verify if the date already exist in database. If so, the input file is invalid and manual action is needed. In this case, the program shall issue warning, move the file to the “bad” directory and continue with next available file.

As stated above, first line of each CSV contains headers. The program must parse the line and check column names against specification. In case of mismatch, warning must be issued, the file has to be moved to the “bad” directory and program has to proceed with next file.

During processing of the data rows, program has to check the data format If the program detects error during processing of the CSV file, the processing of the whole file has to be stopped. Data from the beginning of the file (before the error) which were correctly processed and already stored to the database can be kept in the database. Bonus task is to delete (rollback) these data.

Remember the constraint in all “DAY” columns (first column) in all three input files. Value entered here must contain the same day as is the name in the name of the file.

After the CSV file is processed, it shall be moved to the processed directory. If the file already exists in the processed directory, it shall be overwritten without any notice. The same “overwrite” rule applies for the directory with bad files. Bonus task is to avoid overwriting and versioning the files (naming convention, special suffix or similar).

### FR4 – Data clean-up

We need data clean-up procedure to clean the data which is older that the given day.

Our support engineer will use it from time-to-time to clean some data from database so we don’t run out of the space.

It shall accept one input parameter – date or timestamp. All data customer, account and transaction older than this date shall be deleted.

The clean-up is supposed to keep data consistent on daily level. If deleting data from one day, all data must be deleted or all data must stay in database. Status when all transaction data for one day is deleted while account or customer data is still available is invalid.

Procedure must be accessible via webservice. Example of webservice call is in sample project.

### FR5 – Logging (bonus)

We need to have technical logging and audit logging for our staff.

This information shall make it possible to identify:

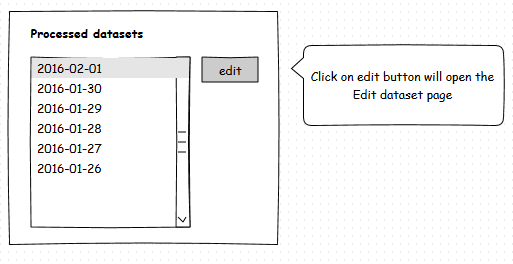
* which files were loaded
* from which folder were these files loaded to the database
* how many rows were successfully loaded
* what was the result of the file load (if the file was processed completely without error or if the load failed)
* when the load started and finished

For example, you can create log file or log table in Oracle database.

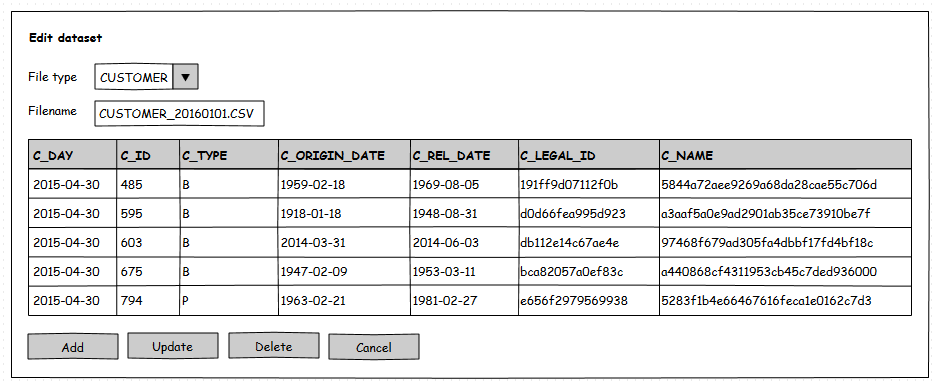
### FR6 – Web application (bonus)

We would like to have a UI where we could see the list of day which were processed, we could edit the data stored in the database and execute the cleanup procedure.

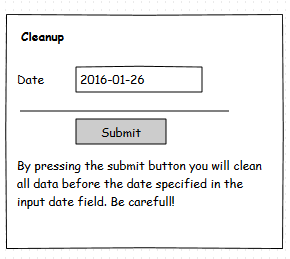
See wireframes, how the screen should look like:



Picture 2 - Available datasets for edit



Picture 3 - Edit dataset table



Picture 4 - Data cleanup screen

### NFR1 - Performance

We expect one dataset daily and the worst time for the load has to be below 1hour.

Estimated max file size is 10MB. Average file size would be 500kB.

### NFR2 - Documentation

We need all the documention to be collected. Design document, JavaDoc, User manual……

### NFR3 – Test (bonus)

We need that the solution is fully tested, we expect Java JUnit test for each single Java class you write.

We will measure the code coverage for JUnit test with [Cobertura](http://www.mkyong.com/qa/maven-cobertura-code-coverage-example/). We want that the coverage is not below 80%.

Our staff will want to see all of the test artifacts. We will not accept and move to the production without having a proper testing evidence.

### NFR4 - Source code repository

We need to have all the source code stored in GIT repository, along with the configuration files. Git is running on TBD and there is repository created for each of you.

### NFR5 - Binary release packages

We need that the binary release package is runnable and testable.

## P2 – Calculate PD

### FR7 - Daily aggregation module

TBD

### FR8 - Monthly aggregation module

TBD

### FR9 - Pd calculation module

TBD

### NFR6 – Performance

Scoring & pd calculation for the whole month has to be performed under 24 h so the downstream systems can use the data as soon as possible.

## 

# P3 - Distribute data to downstream systems

### FR10 – Integration options

Prepare multiple integration options for downstream systems, which they can use to source our data. We would offer database link, webservice interface or messaging system. The solution is up to Architecture design and solution design.

### NFR7 – Availability

Calculated data has to be available 24x7 for the downstream systems.

# References

## Business Processes and Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process ID | Process Name | Requirement ID | Requirement Description | Priority |
| P1 | Collect data | FR1 | Database | 1 |
| FR2 | Input files interface | 1 |
| FR3 | Loader application | 1 |
| FR4 | Data clean-up | 1 |
| FR5 | Logging (bonus) | 2 |
| FR6 | Web application (bonus) | 3 |
| NFR1 | Performance | 1 |
| NFR2 | Documentation | 1 |
| NFR3 | Test (bonus) | 2 |
| NFR4 | Source code repository | 1 |
| NFR5 | Binary release packages | 2 |
| P2 | Calculate PD | FR7 | Daily aggregation module | 4 |
| FR8 | Monthly aggregation module | 4 |
| FR9 | Pd calculation module | 4 |
| NFR6 | Performance | 4 |
| P3 | Distribute data to downstream systems | FR10 | Integration options | 5 |
| NFR7 | Availability | 5 |

# Planning

## P1 – Collect data

FR1

FR2

FR3

FR4

FR5

NFR1

NFR2

NFR3

NFR4

NFR5

FR6

Development environment setup

Dates: 1.10.2018 – 14.10.2018

In case you cannot follow the original plan, and you feel you cannot make the deadline, please contact the assigned coach.

# cover.jpg

APPENDICES

1. Appendix A – Interface specification for incoming files

Incoming files will be stored on file system in directory structure which will be configurable. Files will follow naming convention.

|  |  |
| --- | --- |
| Details | Comment |
| Frequency | One a day |
| File size | N/A |
| Has header | Y |
| File format | Flat file – CSV |

**Filename convention:**

|  |  |  |
| --- | --- | --- |
| File type | Name convention | Example |
| Customer | CUTOMER\_YYYYMMDD.CSV | CUSTOMER\_20160101.CSV |
| Account | ACCOUNT\_YYYYMMDD.CSV | ACCOUNT\_20160101.CSV |
| Transaction | TRANSACTION\_YYYYMMDD.CSV | TRANSACTION\_20160101.CSV |

YYYY – year (1900 - 2999)

MM – month (01 - 12)

DD – day (01 - 31)

**Customer file:**

|  |  |  |
| --- | --- | --- |
| Column | Data type | Description |
| C\_DAY | TIMESTAMP | To which date the record is related.  Format: YYYY-MM-DD.  Field must be filled. |
| C\_ID | INTEGER | ID of the customer inside the bank.  Whole positive number, non-zero, up to 10 decimal digits.  Field must be filled. |
| C\_TYPE | CHAR[1] | “P” for Private Customer (PC)  “B” for Business Customer (BC)  Only these two values are allowed. |
| C\_ORIGIN\_DATE | TIMESTAMP | For PC, date of birth.  For BC, date of foundation.  Format: YYYY-MM-DD.  This field must be filled. |
| C\_REL\_DATE | TIMESTAMP | Relationship with the Bank since this date or  Since when is the Customer with the Bank.  Format: YYYY-MM-DD.  This field may not be known for some customers. |
| C\_LEGAL\_ID | CHAR[24] | For BC, it is tax identification number (IČO) or equivalent identification number (e.g. for companies from foreign countries).  For PC, it is birth number or equivalent. (e.g. for foreigners).  This field may not be available. |
| C\_NAME | CHAR[64] | For BC, name of the company.  For PC, name and surname.  The field is generated from various sources and there is no fixed format. For some customers it can even be blank. |

**Account file:**

|  |  |  |
| --- | --- | --- |
| Column | Data type | Description |
| A\_DAY | TIMESTAMP | To which date the record is related.  Format: YYYY-MM-DD.  Field must be filled. |
| A\_C\_ID | INTEGER | ID of the customer to which the account belongs.  Same format as CUSTOMER.C\_ID |
| A\_IBAN | CHAR[34] | IBAN of the account.  Up to 35 alphanumeric characters. |
| A\_CURRENCY | CHAR[3] | “EUR” Only Euro accounts are considered. |
| A\_F\_ID | INTEGER | ID of the home filial of the Bank which is taking care about this account  Whole positive number, non zero, up to 10 decimal digits.  This field has to be filled. |
| A\_TYPE | CHAR[2] | “CH” for checking account  “S” for savings account  “CD” for certificate deposit  “MM” for money market account  “IR” for individual retirement |

**Transaction file:**

|  |  |  |
| --- | --- | --- |
| Column | Data type | Description |
| T\_DAY | TIMESTAMP | To which date the record is related.  Format: YYYY-MM-DD.  Field must be filled. |
| T\_A\_IBAN | CHAR[34] | IBAN of the account to which this transaction is related |
| T\_NUMB | INTEGER | This is the order of the transaction on this account.  Transactions belonging to one account must be ordered and the order must be stored.  First transaction on the account even has order one. The order is increased with each transaction regardless of type or date. |
| T\_DATE | TIMESTAMP | Timestamp of the transaction.  Format: YYYY-MM-DD HH:MM.SS.mmm |
| T\_INTERNAL | CHAR[1] | “I” for internal transaction (transaction between two accounts of the same customer)  “” blank otherwise |
| T\_DIRECTION | CHAR[1] | Direction.  One character identification of the transaction direction:  “C” for credit booking (incoming transaction)  “D” for debit booking (outgoing transaction) |
| T\_TYPE | CHAR[3] | Type of the payment.  “MAM” Monthly account maintenance  “ASE” Account statement electronic form  “ASP” Account statement paper form  “IP” Incoming payment  “OO” Outgoing online payment  “OP” Outgoing payment other  “UP” Urgent payment  “UC” Uncategorized credit booking  “UD” Uncategorized debit booking  “SO” Standing order  “SIP” SIPO  “SEP” SEPA payment  “INT” International payment  “W” Withdrawal  “DB” Direct Debit  “CP” Credit Card Payment  “CC” Credit Card Other  “U” Unknown |
| T\_AMOUNT | NUMBER | Transaction amount.  Decimal positive number with accuracy up to 6 decimal places |

**Examples:**

********

**More examples can be found on git repository.**

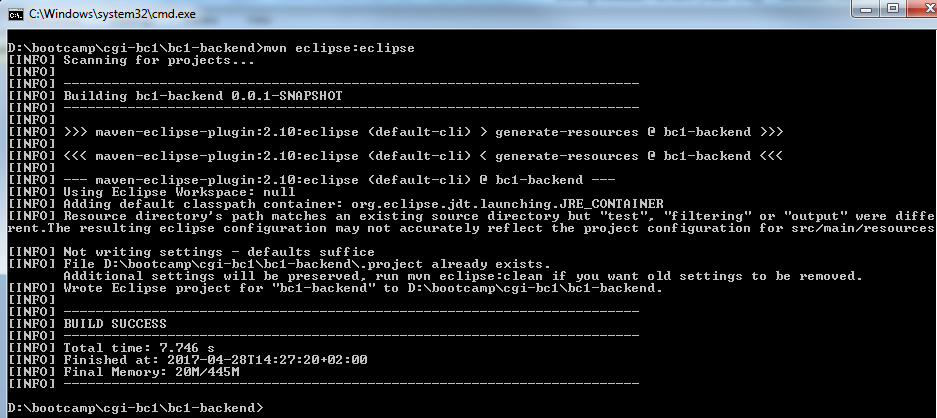
1. Appendix B – Technical recommendations

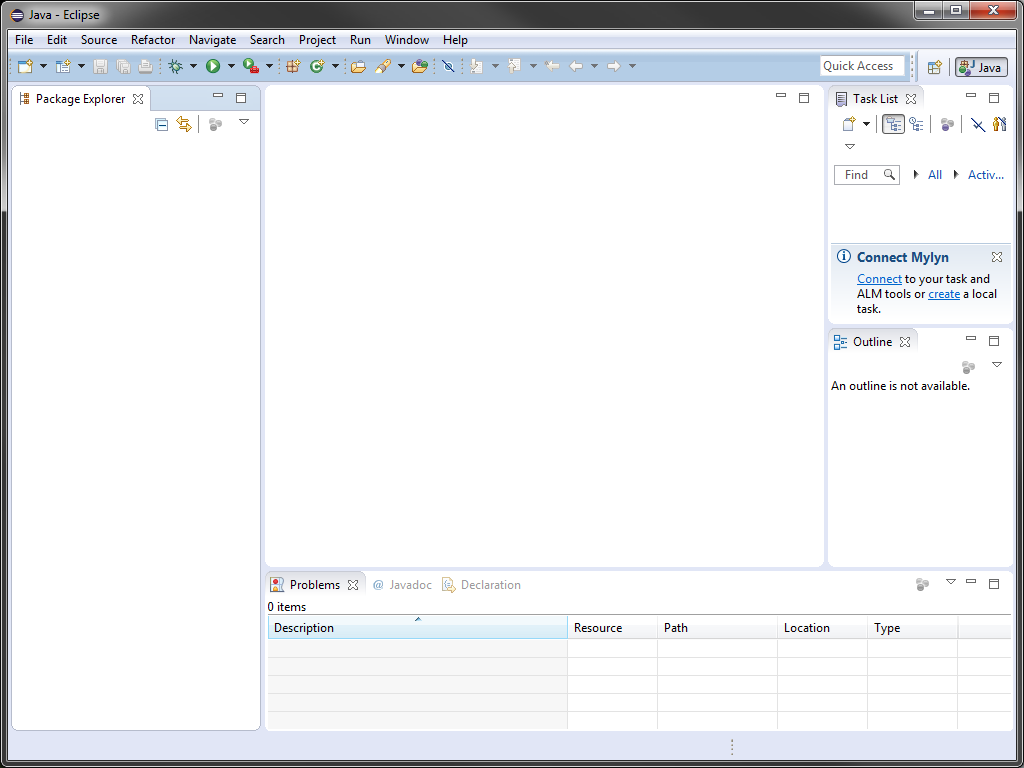
This is normally not part of this document (it’s for the Java Boot camp only)

How to start with sample maven project

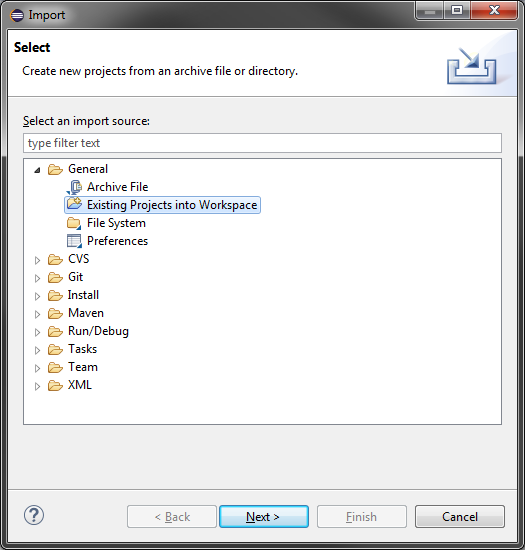
Sample project structure and POM file for building is available for you on git server. You can use pom target eclipse:eclipse to generate and import Eclipse project.

1. Run Maven: mvn eclipse:eclipse
2. Import project to Eclipse

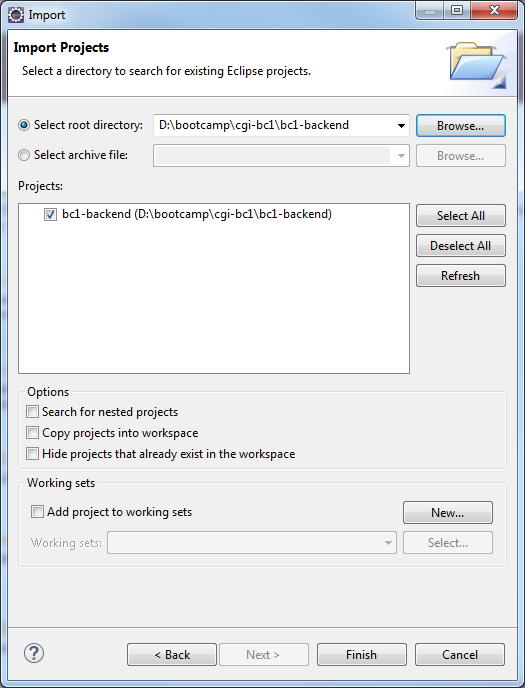




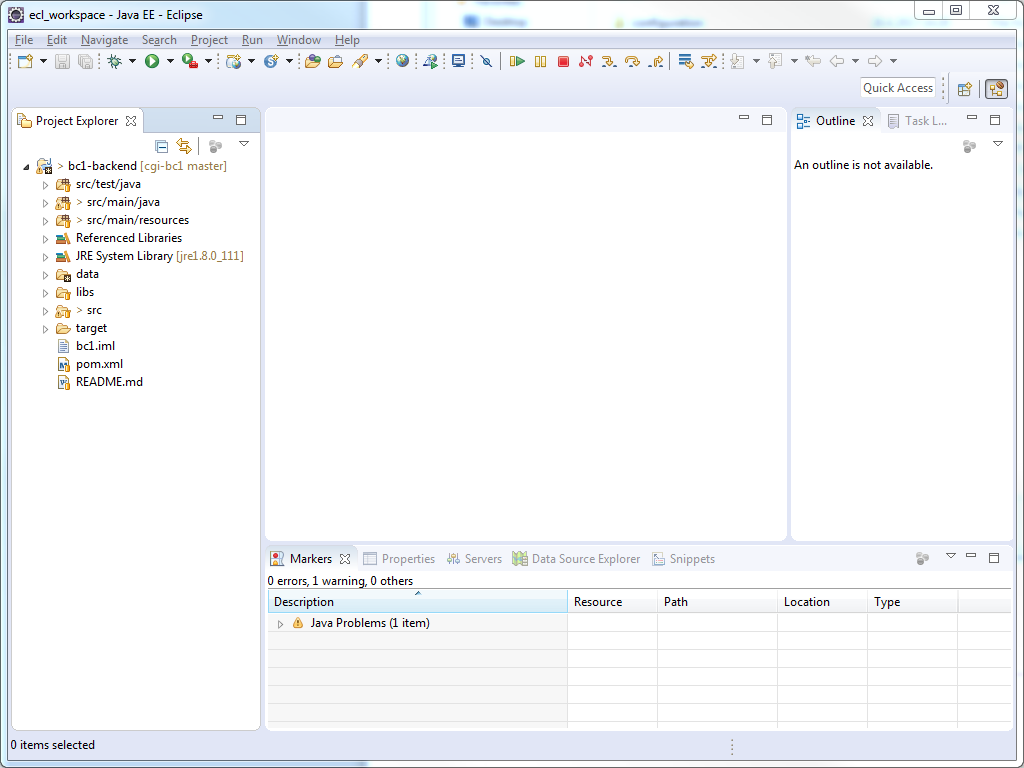
File-Import-General-Existing Projects into Workspace



Select your project and Finish:



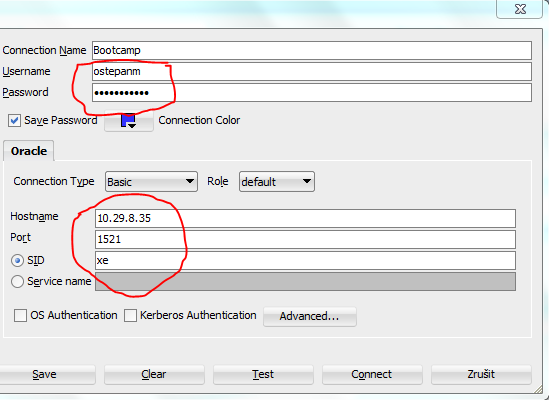
Now, you project has been imported to Eclipse:

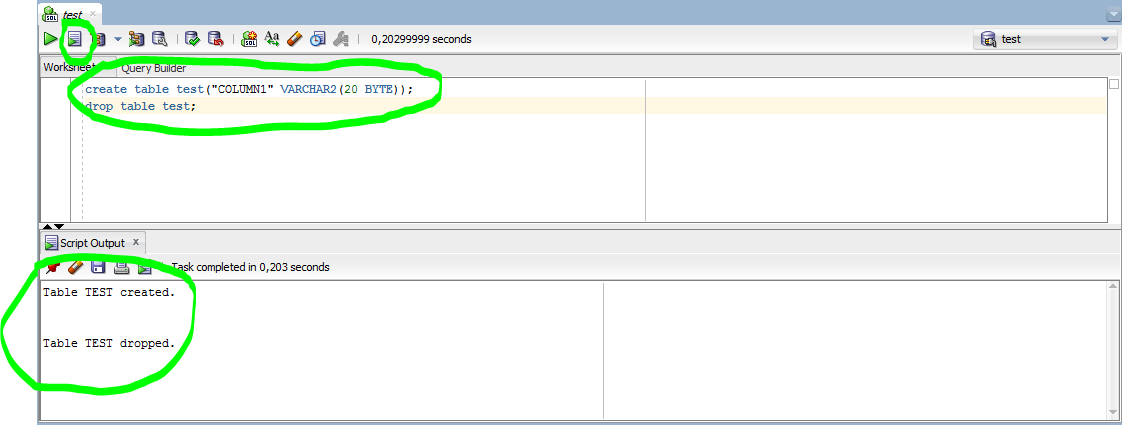


Be sure that project is packaged via maven before running the main method.

How to setup Oracle SQL Developer

1. Download the latest version of [Oracle SQL Developer](http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html)
2. Create a new connection to shared Oracle database (use your user and password)

User name a Heslo v emailu od Evzena Zukovskeho  


1. Test if you can create and drop table (create table test; drop table test;)  
   

How to add Oracle JDBC driver to maven project

Oracle JDBC driver is not available from the Maven Central repository and you have to download it from the [Oracle site](http://download.oracle.com/otn/utilities_drivers/jdbc/121020/ojdbc7.jar) or take version located in project directory in libs folder.

Steps to add the driver to local maven repository:

1. Install file  
   **mvn install:install-file -Dfile=ojdbc7.jar -DgroupId=com.oracle -DartifactId=ojdbc7 -Dversion=12.1.0.2 -Dpackaging=jar**
2. Add dependency to the project  
    **<dependency>**

**<groupId>com.oracle</groupId>**

**<artifactId>ojdbc7</artifactId>**

**<version>12.1.0.2</version>**

**</dependency>**

1. Refresh your project

Unless you setup the dependency and connection successfully, your application won’t start.

If you haven’t worked with JDBC before there is a nice [Java Tutorial](https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html) on Oracle doc site.

Logging

Error reporting can be done by easy output to the console. We recommend more advanced error reporting using Log4J/SLF4J or other 3rd party library. Since the example project is build with Spring Boot Starter module, loggers are accessible by default.

In Java Code:

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

Logger LOGGER = LoggerFactory.getLogger(App.class.getName());

More about logging can be found here: <http://www.slf4j.org/>.

XML

There are many way to parse the XML file. For example, you can also use any available SAX or DOM parser or maybe even JAXB (configuration might be tricky also for small simple projects). We would be happy to see XPATH query. We advise you not to parse XML completely manually.

<https://docs.oracle.com/javase/7/docs/api/javax/xml/xpath/package-summary.html>

<http://viralpatel.net/blogs/java-xml-xpath-tutorial-parse-xml/>

Release package

Deliver one release file – compressed ZIP which would include the whole application, with java code, binaries, libraries, sample configuration files, documentation, tests , readme file…. to your assigned mentor.

Technical documentation

We require you to write some documentation. We would like to have the standard JavaDoc + technical design documentation (template to use).

The technical design document could include:

|  |  |
| --- | --- |
| * Chapter name | * Description |
| * About | * Brief description about the application |
| * Architecture | * Overview of the architecture + if there is something specific or interesting to mention, could contain layers description, components description, scenarios |
| * Database model | * Ideally a nice diagram |
| * Usage | * How the user is expected to use the application .. |
| * Other UML models | * Interaction, class, deployments ….. what is relevant |



Testing (bonus)

Please try to cover the whole application with [unit](http://junit.org/) or integrations tests. Measure the coverage with [Cobertura](http://cobertura.github.io/cobertura/) plugin.

The dependency for testing is already set. Please follow [these](https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-testing.html) instruction to create valid tests or see attached test in example project.