Documentation

## User Manual

### Application Purpose

Dione is a recommendation engine for movies, based on collaborative filtering. Dione allows users to describe a movie via tags, and rate tag relevance to movies. We also allow users to rate movies, and thus, establish the user's taste – which tags he prefers. We can therefore, recommend other movies, based on the crowd-sourced task of tagging movies. We encourage contribution, by means of social interaction – friendships, and notifications about friend activities.

### Installation guidelines

#### Pre installation

At first run you should copy the IMDB and YAGO files to perform data update.

#### Data files

The DataFiles Directory of the project, Should include YAGO and IMDB files, specified in the Running the import section

#### Jars

* swt – version, blabla
* jdbc – bla, bla**Application screens**

**Login Window** – Allows logging in as an existing user, or signing up as a new user.

Only usernames and passwords with 4-10 characters are allowed. Note: after the first login by any user, a rate movie window will be opened and will let the user rate movies.



**All Tabs Window –** The main window of Dione. This is a tab's oriented window, consists of five tabs:

* Note that in order to see updates of new data, like recommendation or recent activities the user should log out and log in again.

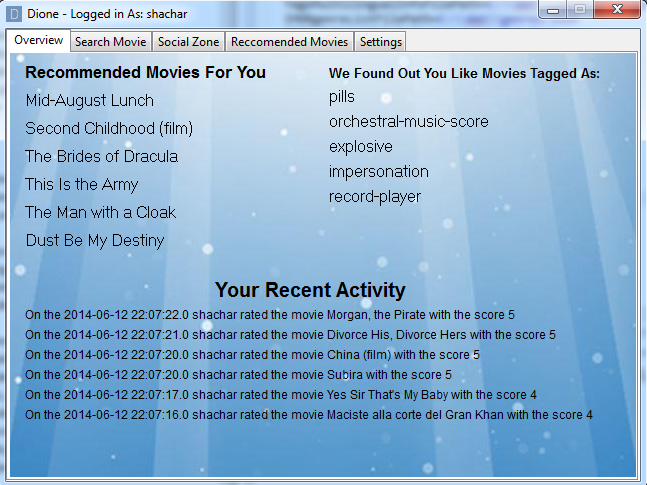
**Tab1: Overview tab –** The default tab, first shown after logging in. This tab consists of general information for the logged in user.

A short list of most recommended movies for the user is shown. A click on each of the movies will open it's Movie Details Window.

A short list of tags that describe movies that seems to be favorable by the user is also shown.

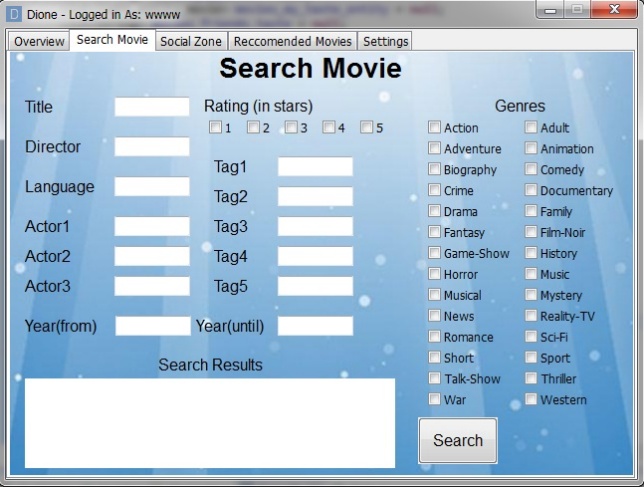
Note that the recommended movies and favorable tags will be shown after Dione will get the chance to learn the user's taste.

A short list of the user recent activity in Dione is also shown.



**Tab2: Search Movie Tab –** This tab allows the user to search for movies by various criterions: title, director, language, actors (up to three), years range, associated movie tags and movie genres.

The search results are shown in a scrollable list. A click on a movie will open it's Movie Details Window.



**Tab3: Social Tab –** This tab allows the user to manage his friendships with other users in Dione. The user can add a new friend or remove an existing friend (his friends are shown in a drop-down list). This tab also consists of the user recent social activity, and his friends recent (general) activity.



**Tab4: Recommendation Tab –** This tab allows the user to enjoy the smart engine of Dione.

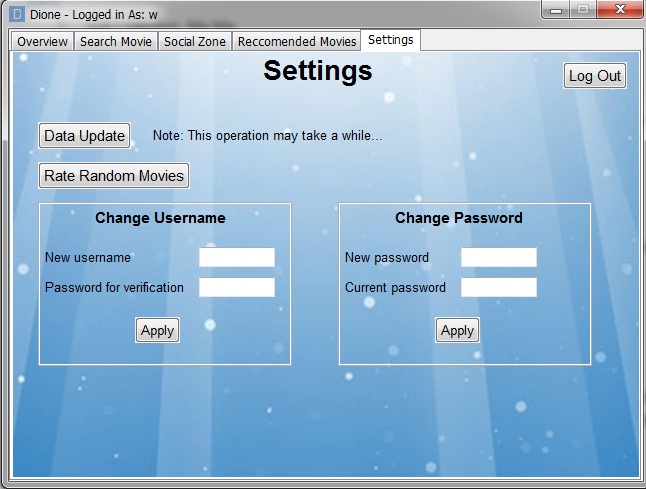
Dione recommends the user movies based on what it learned about him, and based on his friends taste. These are shown to the user in this tab, as well as the generally top rated movies in Dione. A click on a movie will open it's Movie Details Window.

**Tab5: Settings Tab –**This tab consists of account modification options, and some other options. In the settings tab the user can change his username and password (each of them requires the current password for verification).

This tab also allows the user to make a massive data update, a process which may take a lot of time. During the data update, there will shown a progress bar to indicate the update, which inclue an 'Abort' button to stop the update.

A button named 'Rate Random Movies' will open the Preference Window, which allows the user to rate random movies (which will help dione learn more about the user).

A log out button – will log the user out of Dione, and open the log in window.



**Movie Details Window –** This window consist of all the available data of the selected movie.

On the top of the window will be shown the movie title. The data that will be shown consists of genres, director, year, rating(1-5), star players, a clickable Wikipedia link, movie plot and tags associated with the movie.

The user can rate the movie, and can rate the relation of any shown tag to the movie (i.e how much he believes this tag is associated with the movie).

The user can also add a new tag to be linked with this movie.

The user can make a short comment about the movie (by clicking 'Make a Comment'), and can watch the recent comments about this movie (by clicking 'Recent Comments').

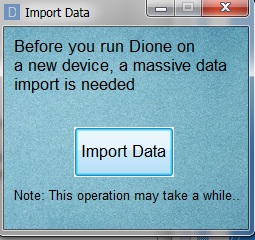
The last two windows will be closed if the main movie details window will be closed.



**Preferences Window –** This window will be shown either by user choice (optional in settings tab), or for any first log in of a new user. The user will be shown a random picked movie to be rated in a scale of 1-5 ('don’t know' option is also available). Every rate will get another random movie to be rated. This process will continue until the user will click the 'Stop Rating' button, which will close the window.



**Import Window –** This window will be shown to the user only if a first data import hasn't yet been made, and finished successfully. It consists of a single button which will run the data import.



## Running the YAGO import

The import must take place before Dione can run.

If an import hasn't yet been made, the user will be enforced to run data import before logging in.

At any point the user can run a data update via the settings tab.

Note that we allow for only one YAGO import to run, every 60 minutes. ????

YAGO files can be downloaded from:

<http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/downloads/>, and the files are:

* yagoFacts.tsv
* yagoLiteralFacts.tsv
* yagoMultilingualInstanceLabels.tsv
* yagoSimpleTypes.tsv
* yagoWikipediaInfo.tsv

IMDB files can be downloaded from:

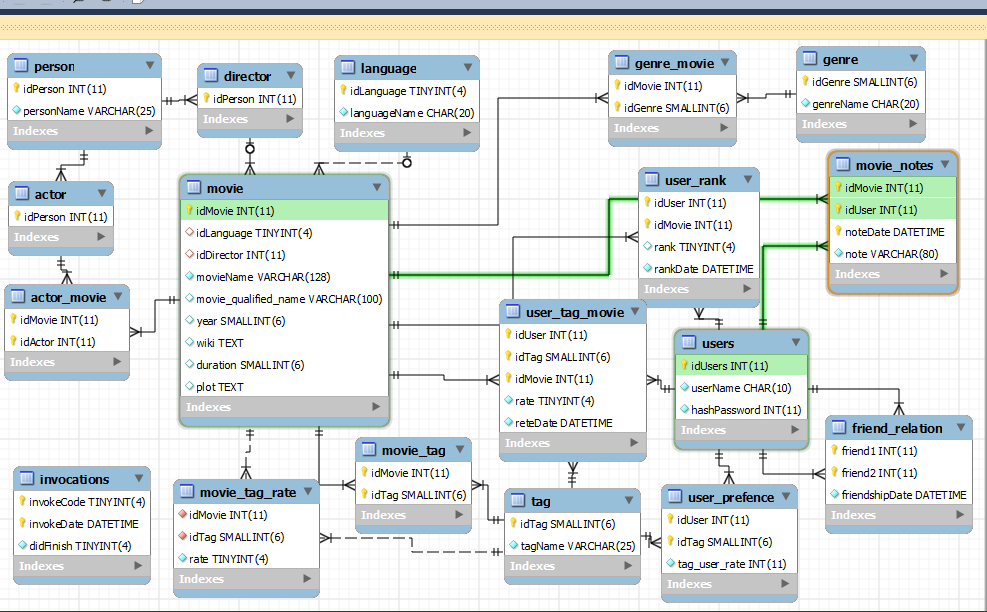
<ftp://ftp.fu-berlin.de/pub/misc/movies/database/>, and the files are:

* directors.list
* genres.list
* keywords.list
* language.list
* plot.list

**guy: what about updating the config file paths?**

## DB structure

### EER

****

### Things that need to be explained:

* A tag is something that characterizes a movie (for example: 1950's, blood exc.)
* A user can rank a movie to say whether he liked it and which movie has what tag, this is how we tell witch movies he would like in the future (what movies he said he liked and what tags do they have - The preferences of users are saved in user\_prefence).
* Things that would be presented in the site as activities are saved in the data base with a date.

### Considerations in creating the DB

* **InnoDB vs MyISAM**  
  MyISAM- used on tables with no foreign keys, when more SELECTs (then INSERTs and UPDATEs) are performed on the table.  
  InnoDB- used otherwise
* **INT and CHAR sizes**we had checked what the biggest possible value is, and determined by that parameter the smallest range possible to save DB space.
* **HAR vs VARCHAR vs NVARCHAR**NVARCHAR- no Unicode was used, so no nvarchar was used.  
  CHAR - where words are not very long (<= 24) we used char to get better query speed (with low space cost)   
  VARCHAR- otherwise, to save space.
* **DATE and TIME**  
  DATETIME - the only date type used, seemingly the simplest type for usage with java
* **Redundancy issues**  
  **actors**- some persons are actors but not all of them. To get an easy way to enforce constraints their person IDs are duplicated to the 'actor' table  
  **directors**- same issue as with actors   
  **tags and genres** - some but not all of the genres are also tags but they have different duties and come from different sources

### SQL injection defense

All of our queries are executed using prepared statements, which is considered an SQL injection free mechanism

## Code Structure

***GENERAL***we need to refactor stuff – appname.ui/core/db/parsing, and move the parser entities to the BL classes, I think.

#### GUI

* Main.java – Consists of the global main function.
* abstract\_window – an abstract dione window, extends Shell, defines a window with Dioneicons for tasksbar
* gui\_utils – several functions used all over the gui package. Consists of the launcher of the.   
  Application, some GridData and FormDatafactories, and some more utilities.
* all\_tabs\_window, log\_in\_window, movie\_details\_window, import\_window, import\_progress\_window, preferences\_window, comment\_window, movie\_comments\_window – all these classes extends  
  abstract\_window, each consists of the relevant data.
* overview tab, search\_movie\_tab, social\_tab, recommendation\_tab, settings\_tab, are all extend Composite. Each of these classes represents a tab in the All Tabs Window, which is the main Dione window.
* Layouts - The windows and tabs are mainly use the Grid and Form layouts for it's widgets.  
  A factory for the relevant GridData and FormData is available in the gui\_utils class.
* OS Resources – as a policy, any font, color or image created in the gui classes, is disposed in the DisposalListener of the widget it is associated with. For example if a label uses font, a call to font.dispose() will take place while the label is disposing.

Threads: guy is it the right place?

Every access to the DB is taking place by a different thread, created just for this task. It allows us make the GUI be responding while a massive query is taking place, and avoiding a possible failure n the DB which may cause the application become unresponsive.

An executor, with thread pool of size 10, is created at the beginning of the program and each thread is sent to the executor. We have used asyncExec and ssyncExec in order to “communicate “ between the “DB” queries Threads and the GUI thread, depending whether we must wait until the GUI execute the request or not.

In addition, a cron thread is created when the program starts, and will call functions for calculating the recommended movies every 15 minutes.

Communication between the GUI and the DB module: guy is it the right place?

This communication is achieved by having a BL module which has classes for movies, persons and user. For each instance of these classes, various functions for getting desired information from database are present. (We will discuss about this module in the next page).

#### DB

This package is responsible for data base connection handling and pooling, generic statement creation and implementation of queries. basically everything related to DB.

The package assumes data already exists in tables, though it will not crash if it doesn't.

**Jdbc\_connection\_pooling**this is a singleton class that gets connections from the driver. Saves them in a vector and on a call to getConnection, returns a SQL connection.

Has two vectors one for currently available connections and the other for free ones.

**db\_operations**An abstract class inherited by all the db\_queries files, it implements the generic SQL functions (select, delete exc.) and has some cool generic functions that saves us from code duplications. Also implements the functions relevant for invocations table

**db\_queries\_**\*  
these classes contain the actual methods that query the data base. All three pretty much have the same purpose and were split because of their size.

***BL***

This package is used to partition between the GUI layer and the DB layer and perform whatever changes the graphic unit expects to be made on the data after it was pulled from the data base.

**\*\_logic**  
as in the db\_queries files these classes have the same purpose and have been separated from size matters.

They are the core of the business logic level, they should do the big logical actions when actually most of.

Default values (such as limits) are pulled from the config.

The user\_logic class holds the current-user-id and uses it as the default id for some functions, this is the reason it is not abstract the others are.

#### Importing

* Importer.java - This class serves as a main runnable logic, that handles all of the import steps. These include Parsing YAGO and IMDB files, and loading data to tables. The importer keeps track of import progress, within each step, and notifies it's clients accordingly. The thread can be terminated in a safe fashion, terminating whatever parse or batch being done, and shutting down.
* Abstract\_yago\_parser.java – an abstract class, that handles reading of a yago files (handlingio, and tag-level parsing of the lines), that relies on concrete classes, that handle each specific file – react differently to tags, expect different line structures, populate different entities, etc. these include: list. The Yago Parsers are responsible for our main entities – movies and persons (actors, directors), as well as relationships between these entities, and other movie details. Listeners can register to progress events, and termination of import, is handled at the main loop.
* Abstract\_imdb\_parser.java – an abstract class, that handles reading of a IMDB files (handling io and file scans). Concrete classes, handle the parsing of each line: extracting entities such as languages, tags and their frequency, plots, and more. To identify a YAGO title with an IMDB title, we rely on a combination of movie name, movie director, and year. The abstract parser manages the mapping of imdb names, to yago names, taking in account the possibility of missing details, and multilingual issues (foreign titles, usually appear under their foreign names, in IMDB, by English names, in YAGO). There is an extending class, responsible for parsing each file, and these include: list Listeners can register to progress events, and termination of import, is handled at the main loop.
* Abstract\_loader – an abstract class responsible for iterating over a collection of entities (persons, movies, languages, etc…), creating a batch, executing it, and handling batch exceptions. Extending classes, have some entity-specific logics – choosing whether to insert or update, creating specific prepared statements, specific bach execution, etc. there is a loader for each entity, and these are: list. Listeners can register to progress events, and termination of import, is handled at the main loop.
* Application entities: some entitites are simple, and are represented in a ID,Name map. Complex entities include movies, persons, and user-activities. "light movies", are a softer version of movies, passed to the UI, that is ignorant of some parsing logic. Entities are usually just member wrappers, with getters and setters. Should we implement equals, hash, etc?!, and movies have some "qualified name" creation logic – returning all possible name-director-year identifiers of this movie.

#### Miscellaneous

#### Data used

* YAGO data:
  1. yagoSimpleTypes.tsv - Movies, Actors, Directors
  2. yagoFacts.tsv - Actors that played in Movie, Directors of Movies
  3. yagoLiteralFacts.tsv – Lengths and Years of movies
  4. yagoMultilingualInstanceLabels.tsv – foreign names of Movies and Directors
  5. yagoWikipediaInfo.tsv - movie Wikipedia urls
* IMD data:
  1. directors.list – IMDB directors of movies
  2. genres.list – genres, and genres of movies
  3. keywords.list – tags, tag frequency, default tags
  4. language.list – languages, languages of movies
  5. plot.list – movie plots

#### External Packages

None

#### Invocations

A mechanism provided to synchronize performance of special actions between several instances of the application working with the same DB

#### Recommendations process

#### To predict which films you will like we use two mechanisms related to tagging:

#### movie\_tag\_rate (MTR) – which is the average of how well all of the users thought the tag is describing the movie

#### user\_prefence (a commemorated mistake for preference) – using the movies you rated and their MTR we try to find which tag you like the most.

#### At all times, in every instance of the application, every 15 minutes there is a check for an invocation of the filling of the table MTG (it will only happened every 15 minutes - once per DB, not per application instance)

#### In user\_prefence the rows related to each user are deleted and re-entered every time the user ranks a movie, taking to considerations the new MTR

#### General Flow of the Application

* As long as the db is empty (data import didn't happen yet), the only window that will be shown is the Import Window, which consists of only one button – "Import Data".If some user is currently making the first data import, any other user that will try to run the app will be asked to close is and try again later (in order to avoid to parallel updated). After the first successful import, the next time the application will be started, it will navigate directly to the log in window. However, the user can still update data via the settings tab.
* A successful run should open the log in window. When logged in for the first time, the user is asked to rate some movies in order to help Dione to learn about his movies preferences.
* When logged in, the user's main window is the All Tabs Window.
* Any movie name shown in the overview and recommended movies tabs is clickable, and a click on it will open it's movie window.
* On a movie window, a user can open only one comment at a time, and only one recent comments window at a time.
* Settings Tab Scenarios:
* If user is clicking Log out while updating data, he will fail and will be asked to first terminate the update.
* If user is clicking the X button while updating data, the update will terminate and the application will be closed.
* Logging out from user account will immediately close all the opened windows for this user (preference window, movie details window).
* If a preference window is open, or a data update is taking place, and the user asks for it again, he will get a Failed message box.