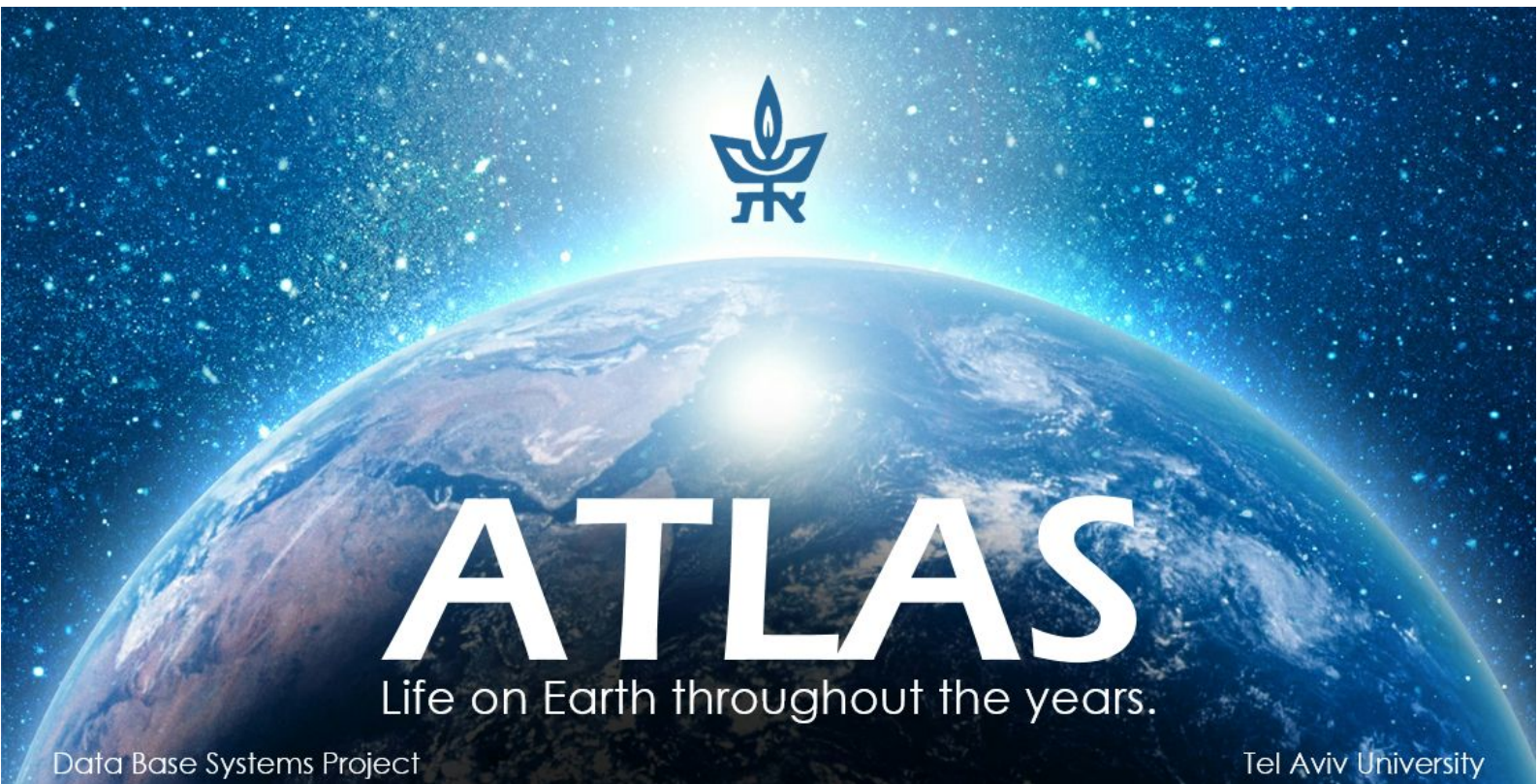


DB Project



Database Systems Course

Spring 2015

Submitted by:

Alon Grinshpoon	200968600
Etan Grundstein	301732285
Paz Dangur	200280857
Rony Jacobson Levi	302986997

Atlas

Table of contents:

[Application Overview](#)

[What is Atlas?](#)

[Application Features](#)

[Prerequisites](#)

[Technical Overview](#)

[DB structure](#)

[Queries](#)

[Yago and GeoNames files](#)

[YagoParser](#)

[Code structure](#)

[External Packages](#)

[Connection Pool and Multi-Threading](#)

[Installations Guidelines](#)

[What's in the box](#)

[Creating the Schema](#)

[Connection Configuration File](#)

[log4j Configuration File](#)

[Runme Scripts](#)

[Application Screens Walkthrough](#)

[Sign In/Sign Up](#)

[First time on our app?](#)

[Ok, I Installed the app, what's next?](#)

[How do I register?](#)

[I'm a registered user, how do I login?](#)

[Initialize DB](#)

[Map Screen](#)

[Search for Person](#)

[Insert New Person](#)

[Edit Person Screen](#)

Application Overview

What is Atlas?

Atlas is a time travel application which lets you explore different people in different periods of time around the world.

Atlas is based on two open source databases: YAGO and Geonames. The Atlas DB filters and joins relevant data from those DBs.

Application Features

The main view is consisted of a map of the world and a timeline, you will be able to choose a specific time period and watch where important people were born and where they died during that time.

You will be presented with different categories that represent people e.g.: monarchists, poets, medalists, scientists, etc.

Each category will present the relevant people according to the time period and category that were selected.

Upon picking a specific person, a Wikipedia URL will be available for learning more information.

You will be able to mark your favorite persons and see them at all times.

A specified search lets search specific persons regardless of their category.

The DB is dynamic, and thus, can be updated by users who wish to add persons by themselves.

Prerequisites

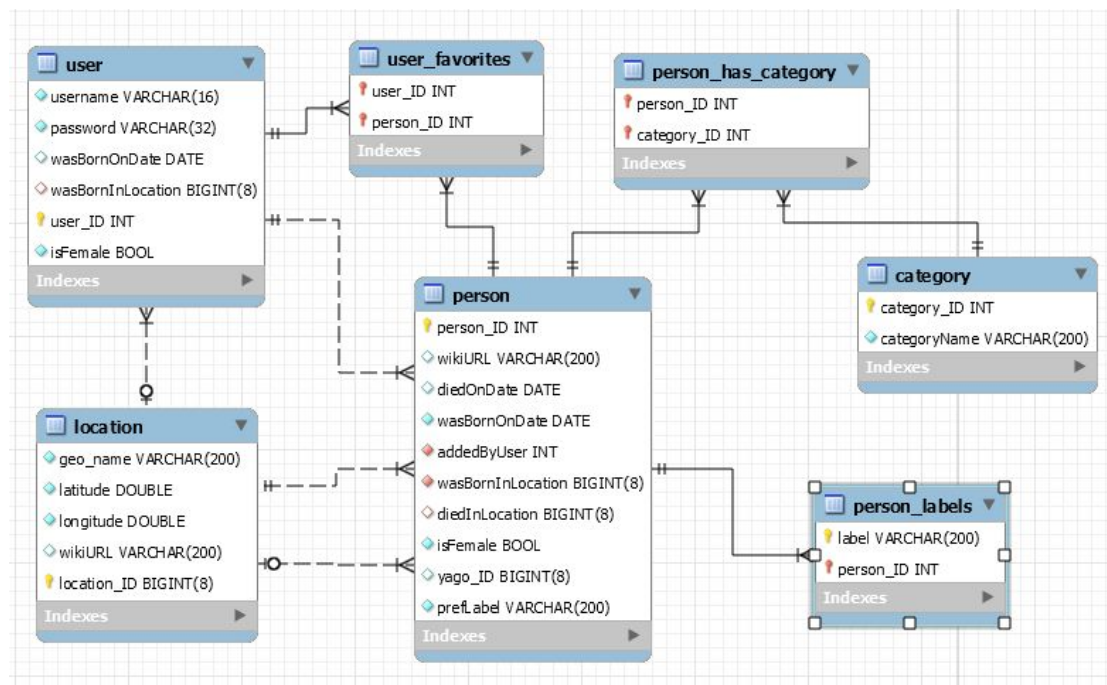
In order for atlas to work, you'll need to meet the following prerequisites:

- Have JRE 7.x (or higher) installed
- Have an internet connection (for loading all JavaScript libraries (including the map)
- Have 700 MB of free heap space and 15 GB of disk space (for the DB populating operation)

Technical Overview

DB structure

The following image represents our schema EER diagram:



Our database is mainly distributed into two major aspects:

1. The Entities database:

This part is a database used to store all the information that was extracted from Yago and Geonames.

Our main entities are persons: Each person has a unique person_ID which is auto-generated at the time of insertion, which serves as the primary key for the Person table, where person records are stored. Once the DB is populated, this key is used to bind other tables to the main table in order to fetch concrete details about a person such as the location of his birth (and possibly death), categories he is associated with, names he goes by (labels) and whether he's a favorite of some user.

* In order to support full updates from YAGO, we had to choose another method of identification for entities originating from the YAGO db. This is done by storing a hashed version of the yago_ids for such persons (unlike user-created ones) to be able to detect changes in YAGO.

A person **MUST** have a birth date and a birth location, and can have either **BOTH** the date and location of its death, or **NEITHER**. Also, a person may have an additional wikipedia link to allow users to learn more about him/her.

Another entity in our database is a location: These originate either from YAGO or from the GeoNames database. Every location has a unique location_ID which is used to identify it and acts as a foreign key for a person's birth/death places. Each location of course has a name and a set of coordinates to be able to place it on the map.

Each person entity can come back as 2 results- one birth result and one death result. Each result will have a date and a location provided from the Location table. Each result can have one or more categories.

If the search was category oriented the result will have its category.

If the search was general- by name, by dates or only favorites, the result will hold all the person's categories (to show a summary of a result).

In a search by name we also use person labels table, one person can have multiple labels in our DB.

Because our searches are by dates, years and names mostly we chose this fields as Indexes.

2. The user's environment: this part is consisting mainly with the relations used to store the user's information; the registered user's details and user's favorites persons picks.

As shown in the diagram, our DB contains many foreign key constraints, such as a person must have born/died in location value which exists in the location table. A person's label must be linked to an actual person

Queries

We created a 'DBConstants' class for all table names and columns names. Most of the queries use these constant to avoid typo's etc. Some queries where very unclear when written like that and where re-written hard-coded.

We have a 'Queries Interface' that allows anyone who wishes to perform any query to call a simple method. This way we could easily test and work separately. We have two implementations to this interface:

- "MockQueries": An implementation that is mocking all queries and was written while the GUI was developed in order to test it and one.
- "DBQueries": An implementation that actually calls the queries that connect with the DB and used in our program.

We have a class called "BaseDBCommand" which handles connections and thread pooling for performing queries. All other queries inherit these features from it.

General queries:

1. GetUserQuery - Fetching User by username.
2. CheckConnectivityQuery - Check if we have basic connection to the DB.
3. AddPersonQuery - Transaction for adding a new person to the DB (SQL procedure).
4. GetCategoriesQuery – Get all categories available.
5. GetGeoLocationsQuery- Get all locations name and ID's from the

6. GetUserQuery- get a registered user and his data from the DB.
7. ParseFilesCommand – Filling up the DB.
8. UpdatePerson – for editing an existing person
9. UpdateFavorties – for adding and removing favorites

GetResultsQueries (for searches):

GetResultsGeneralQuery is a base class that creates the basic logic of fetching result sets from the DB. All of our searches will eventually need to create the same table but with different rows, so the basic logic of building it is there. Each other getResults class inherits from this one and extends it to be valid to the specific search criteria:

- GetGoResults – implements search by category and years from the main UI
- GetSearchResults – implements search by name or search by dates from the search screen.
- GetNewFavoritesResult – implement search by the user's favorites.

We actually have a package of oldCommands which are deprecated. The design shown above was built after lessons learned of this old queries to avoid code duplication.

Yago and GeoNames files

We used the following files:

yagoDateFacts.tsv

To extract birth and death dates.

yagoFacts.tsv

To extract birth and death locations, as well as person's gender

yagoTransitiveType.tsv

To extract which categories are associated with each person

yagoLabels.tsv

To extract the names of persons and locations originated in YAGO

yagoLiteralFacts.tsv

To extract coordinates of locations originated in YAGO

cities1000.txt

To extract coordinates of locations originated in GeoNames DB

yagoGeonamesEntityIds.tsv

To translate GeoNames IDs to YAGO IDs for said locations

yagoWikipediaInfo.tsv

To extract wikipedia URLs of persons and locations

YagoParser

To handle the raw data from YAGO and GeoNames, which is provided as large text files (15 GB in total) we've created a parser that scans the files one by one and extracts meaningful information into objects in java.

Step 1: locations, locations, locations...

We maintain a map of locations, which are comprised of location id, name, latitude and longitude. We populate the map with locations from YAGO as well as locations from GeoNames and then filter out locations which have incomplete records (missing some property).

Step 2: we are the people of YAGO

We maintain a map of persons, where the keys are a hashed version of the person's YAGO ID and filter out persons without a birth person or a birth location (as those would not be shown on the map).

Step 3: remove unused locations

Because YAGO consists of such vast amounts of data, it has lots and lots of locations as well, and some of it is utterly irrelevant for us (such as Israel's discount bank for example). for this reason we've decided to remove locations which no person in our database was neither born nor died in.

Step 4:

We extract categories, labels and wikipedia links to enrich our database.

Code structure

Our code is divided to 5 main packages:

il.ac.tau.cs.databases.atlas

Main class

il.ac.tau.cs.databases.atlas.core

Buisness logic.

il.ac.tau.cs.databases.atlas.core.exceptio

new exceptions introduces by Atlas

il.ac.tau.cs.databases.atlas.core.modal

Our Model of every object we need throughout the program: Result, Location and User.

il.ac.tau.cs.databases.atlas.core.progress

Package for handling progress-bar of installation process

il.ac.tau.cs.databases.atlas.db

DB interactions, management, installation and update.

il.ac.tau.cs.databases.atlas.db.command

All of our general and search commands, all inherit from base command

il.ac.tau.cs.databases.atlas.db.command.base

Base command for getting a connection from the thread pool and returning it for all commands.

il.ac.tau.cs.databases.atlas.db.command.oldCommands

Old and deprecated commands that are no longer in use in our code.

il.ac.tau.cs.databases.atlas.db.connection

Creating and managing the connection pool for multi-threading.

il.ac.tau.cs.databases.atlas.db.queries

The queries interface, it's implementations and tests.

il.ac.tau.cs.databases.atlas.parsing

Parsing Yago and Geonames files.

il.ac.tau.cs.databases.atlas.ui

GUI and all related to it.

il.ac.tau.cs.databases.atlas.ui.listeners

il.ac.tau.cs.databases.atlas.ui.map

The map object itself (A browser)

il.ac.tau.cs.databases.atlas.ui.screens

All of the different screens of the program.

il.ac.tau.cs.databases.atlas.ui.utils

Utils class for UI,

il.ac.tau.cs.databases.atlas.utils

General Utils class.

External Packages

- Log4j – class for logging the program flow.
- SWING, SWT, AWT- for GUI design.
- Toedter- for calendar dates handling.
- Junit- for testing queries without needing GUI (See DBQueriesTest class).
- org.apache.commons- For handling state of application

Connection Pool and Multi-Threading

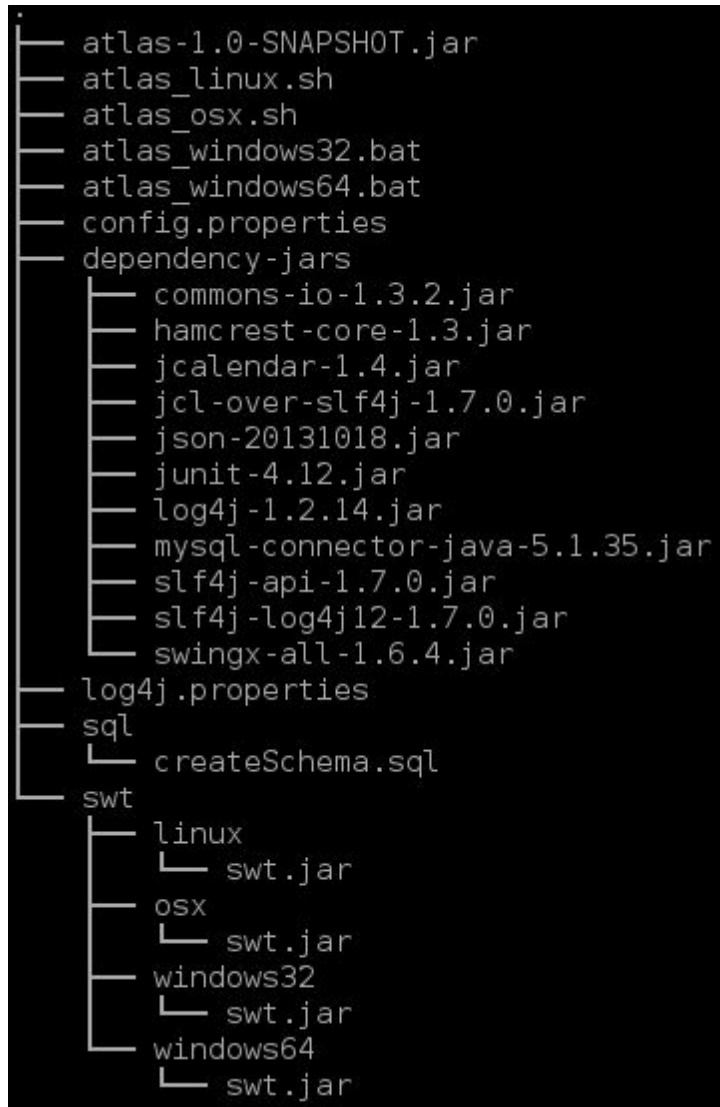
The Program is multithreaded for all processes that run from different windows than the Map screen. The reason for that is that we don't want the user to abuse the getter queries, and create race-conditions. The full upload, edit and add can run simultaneously with getter queries.

In order to allow several queries at once, we maintain a connection pool. We have implemented two types:

1. Dynamic Connection Pool - When a connection is to be checked out, the pool check whether it contains a connection which is not expired. If there is such connection, the pool returns it. if not, the pool creates a new one. Once a connection is returned, it's expiration time is prolonged.
2. Fixed Connection Pool - contains fixed number of connections (10). When initialized, this pool starts 10 connections in advance and when a connection is requested, if the pool currently contains a connection it will return it, otherwise it will block.

Installations Guidelines

What's in the box?



The main JAR is the atlas.jar.

All the JARs with the exception of SWT, are platform independent and reside in the 'dependency-jars' directory. SWT is supplied separately for each platform and is handled in the appropriate runme script for the platform.

Creating the Schema

Before you can use the app, you must create the schema by executing the supplied 'createSchema.sql' file.

Connection Configuration File

The following settings are set in the 'config.properties' file:

- user (string - up to 16 chars) - username for connecting to the MYSQL server
- password (string - up to 32 chars) - password for said user
- schemaName (string) - the name of the schema
- ip (string) - the ip address / hostname of the MYSQL server
- port (integer) - which port to use
- connectionType ('dynamic' or 'fixed') - connection pool type to use

log4j Configuration File

By default, the logger would output to stdout and to an 'atlas.log' file in the base working directory. You can change the logger file by altering the 'log4j.appender.file.File' property in the log4j.properties' file.

Runme Scripts

The following runme scripts are provided:

- *Windows x64*
execute the 'atlas_windows64.bat'.
- *Windows x32*
execute the 'atlas_windows32.bat'.
- *Linux*
execute 'sh atlas_linux.sh' from the terminal.
- *Mac OS X*
execute 'sh atlas_osx.sh' from the terminal.

Application Screens Walkthrough

Sign In/Sign Up



First time on our app?

If it is your first time using Atlas, press the "Install DB" button, to open "Initialize DB". More on that is explained in the "Full Update Screen". So

Ok, I Installed the app, what's next?

First off all, please notice you have to click on the fields before writing to them (and not use the tab keyboard button), so that we will know you entered data. So if you've entered your credentials and you get the following message: "Please enter login credentials." click on the "Username" textbox, and fill the credentials again.

How do I register?

In order to register you need to choose a username and enter it (an original one), a password (be creative!) and press the 'Glimpse into the past!' button. The app will prompt you with a message to register, if you choose to accept it, you will need to fill where and when you were born and your gender. press the button again. and of the username was not taken you will be passed to the "Map" screen, Have fun!



I'm a registered user, how do I login?

In order to log in, you need to insert your username and password, which will be validated against the DB. Press the "Glimpse into the past!" button. If a match has been found, you will be passed to the "Map" screen.

Initialize DB

In order to be able to populate and initialize the DB, it needs to have a schema set on it. For schema installation, please refer to the Installation section.

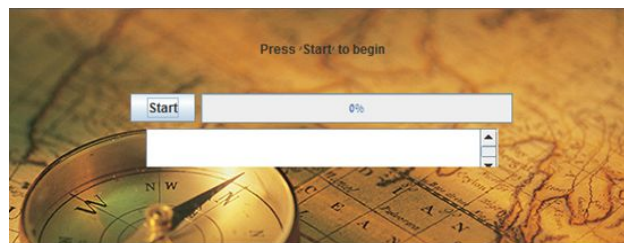
There are two ways to get to the DB initialization screen:

- Pressing the  button from the Map screen
- Pressing the  button from the Login screen

After pressing those buttons, a directory selection prompt will pop up, and you will have to select a directory that includes all of the following files **unpacked**:

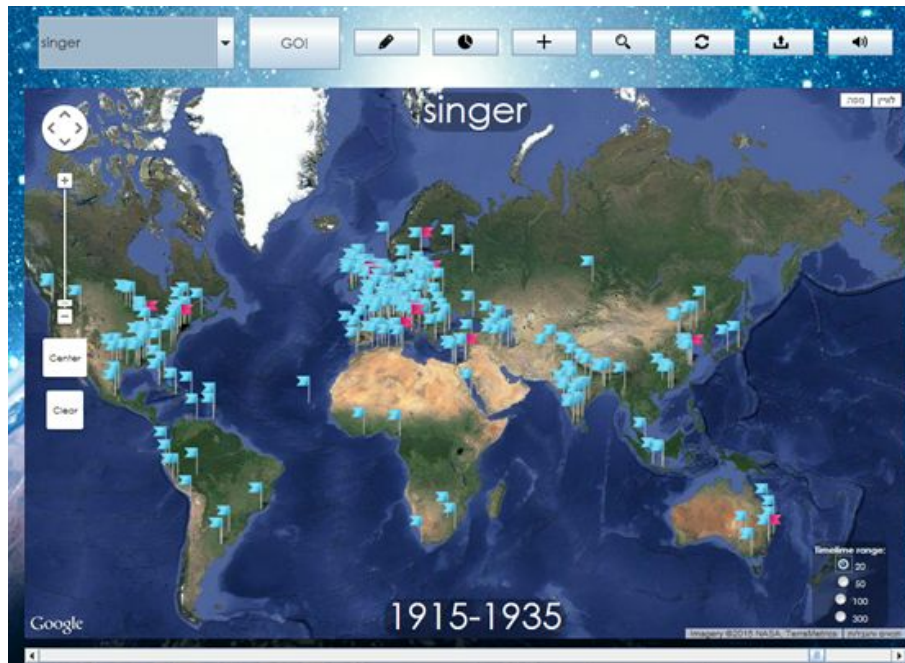
- [yagoLiteralFacts.tsv](#) [from [YAGO](#)]
- [cities1000.txt](#) [from [GeoNames](#)]
- [yagoDateFacts.tsv](#) [from [YAGO](#)]
- [yagoWikipediaInfo.tsv](#) [from [YAGO](#)]
- [yagoFacts.tsv](#) [from [YAGO](#)]
- [yagoTransitiveType.tsv](#) [from [YAGO](#)]
- [yagoGeonamesEntityIds.tsv](#) [from [YAGO](#)]
- [yagoLabels.tsv](#) [from [YAGO](#)]

If you are missing a file, an error message will inform you what you are missing. If you choose a valid directory, you will be redirected to a the Update screen:



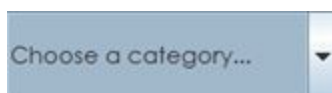
When you'll press the start button, the files will be parsed and used to populate the DB. If you got here in purpose of updating the DB, don't worry, you should still be able to work on the DB as usual. Just mind that if you add a new person which should be updated be the updater it will be overridden.

Map Screen



Sometimes it takes some time for the map screen to initialize (depending on your internet connection), so be patient! The Map screen contains the following components:

1. Categories bar:



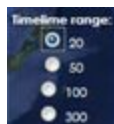
A drop down menu where you select the category of persons you want to display.

2. Timeline slider:



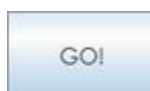
You can move this slider in order to set the years' range for the search by category.

3. Timeline range modifier:



A set of radio buttons to set the range of the years. Move after you select the range to refresh.

4. GO! Button:



This button initiates a search of all persons that fall into the category and the range of years from the years displayed. The person's will then be displayed on the map as flags. Blue – represent a birth and red- a death.

5. Edit Button:



This button will open the "Table View Screen" (See section), where you choose a person to edit.

6. Statistics button:



Shows a summary of the results presented on the map.

7. Add Button:



Opens the "Add Person Screen"

8. Search Button:



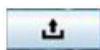
Opens the "Search for Persons Screen"

9. Sync Button:



After you mark/unmark a person as a favorite, you need to sync these changes into the DB. This is what this button serves.

10. Full DB Update Button:



This button opens up the Full update/ DB Init Screen mentioned in the previous section

11. Sound Button:



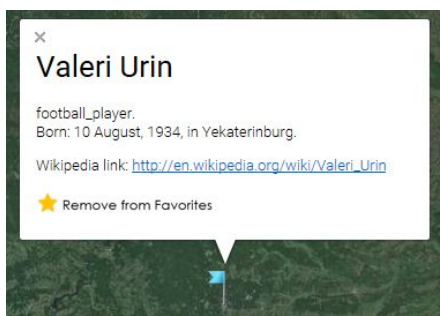
Mutes/plays the background theme music

15. Person's Flag:



The blue flags represent a person's birth, while the red one represents a death. These icons are clickable and once clicked, they open some more information about that person.

15. Person's Data:



If you click on a flag, a bubble will open containing data of the person. You can choose to visit this person's wiki link and learn more about him or you can choose to add/remove him from your favorites. Don't forget to sync your favorites after changing this property!

Search for Person



This screen presents two searching options for persons:

- The upper one is meant for searching all of the persons with labels matching the name. For example, if you search for "Ronaldo", you will see "Cristiano Ronaldo", "Ronaldinho", etc.
- The lower one is meant for searching all of the persons (regardless of their category), that were born or died in between the dates requested.

Insert New Person



This window is intended for adding a new person to the DB. A URL must be supplied in the "Link to wikipedia" textbox

Edit Person Screen

This flow is comprised of two windows. When there are results on the map, you can edit them by pressing the edit button. The following window will be opened:

Edit selected person						
Name	Sex	Born in	Date of Birth	Died in	Date of Death	Wiki Link
Naldo (footballer born 1982)	Male	Londrina	10 September, 1982			http://en.wikipedia.org/wiki/Naldo_%28footballer_born_1982%29
Ronaldo Guiaro	Male	Piracicaba	18 February, 1974			http://en.wikipedia.org/wiki/Ronaldo_Guiaro
Ronaldinho	Male	Porto Alegre	21 March, 1980			http://en.wikipedia.org/wiki/Ronaldinho
Roberto Brown	Male	Panama City	15 July, 1977			http://en.wikipedia.org/wiki/Roberto_Brown
Ronaldo Gonçalves Drummond	Male	Belo Horizonte	1 January, 1946			
Theo Robinson	Male	Birmingham	22 January, 1989			http://en.wikipedia.org/wiki/Theo_Robinson
Naldo	Male	Londrina	1 January, 1982			http://en.wikipedia.org/wiki/Naldo
Tiago Ronaldo	Male	Loures	1 January, 1988			http://en.wikipedia.org/wiki/Tiago_Ronaldo
Ronaldo	Male	Rio de Janeiro	18 September, 1976			http://en.wikipedia.org/wiki/Ronaldo
Ronaldo Tres	Male	Rodeio Bonito	13 February, 1987			http://en.wikipedia.org/wiki/Ronaldo_Tres

Select a person from the table to edit and press the "Edit selected person" button.

The following window will appear:

The screenshot shows a form titled "Edit this person:" with a background image of a starry night sky. Below the title are radio buttons for "Male" (selected) and "Female". There is a text input field containing "Ronaldo". Below this are two date input fields: "Birth date:" with the value "18/09/1976" and "Death date (Optional):" which is empty. Below the dates is a dropdown menu showing "Rio de Janeiro". Below the dropdown is a text input field containing the URL "http://en.wikipedia.org/wiki/Ronaldo". At the bottom of the form is a button labeled "Update Record".

Edit all of the details you wish and press "Update Record".