Hello,

Here it is - a small implementation of the core of a social network. It is built with PhP on the server side, uses Zend Framework for the back-end and implements the Model-View-Controller design pattern.

The following text includes considerations on:

1. Demonstration links and sources
2. Database structure
3. General overview
4. Points of entry (Data importer, Front-End and API), security and SEO
5. Re-usable components
6. Performance

Of course, I am available for any clarifications which may be needed.

1. Demonstration links and sources
==========

The sources for the applications are freely available on GitHub:
https://github.com/amorroxic/SocialNetwork

Front-end:
http://social.nouauzina.ro/

Demonstrative API links are as following:
Friends list:
http://social.nouauzina.ro/api/v1/friends/of/peter-mac (XML format)
http://social.nouauzina.ro/api/v1/friends/of/peter-mac?format=json (JSON)

Friends of friends list:
http://social.nouauzina.ro/api/v1/friends-of-friends/of/sandra-phelan (XML)
http://social.nouauzina.ro/api/v1/friends-of-friends/of/sandra-phelan?format=json (JSON)

Suggested friends:
http://social.nouauzina.ro/api/v1/suggested-friends/for/sarah-lane (XML)
http://social.nouauzina.ro/api/v1/suggested-friends/for/sarah-lane?format=json (JSON)

Recommended cities (city rankings are not rounded up to this point):
http://social.nouauzina.ro/api/v1/recommended-cities/for/lisa-daly (XML)
http://social.nouauzina.ro/api/v1/recommended-cities/for/lisa-daly?format=json (JSON)

Of course these are only random demo links, everything works for any of the users.

2. Database structure
==========

There are four tables. You can find the database dumps in two flavours:
- structure only
- structure plus data

You can see these files here: https://github.com/amorroxic/SocialNetwork/tree/master/documentation/database
Should you use only the structure, you need to run the importer process (social.nouauzina.ro/import)

The tables are as following:

```
CREATE TABLE `accounts` (
  `account_id` mediumint(9) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(50) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,
  `last_name` varchar(50) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,
  `age` varchar(10) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,
  `gender` tinyint(1) NOT NULL,
  `slug` varchar(200) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`account_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Holds the accounts. For each of the users, during the import process a unique slug is being generated. This is done in order to: a) Hide any database ID and b) Makes everything a bit more SEO friendly.
The slug generation process is automatic and the core of the routine can be seen here: https://github.com/amorroxic/SocialNetwork/blob/master/application/library/Helpers/Slug.php
Also, the gender and age fields are being checked and sanitized.

```
CREATE TABLE `accounts_connections` (
  `id` mediumint(9) NOT NULL AUTO_INCREMENT,
  `account_id` mediumint(9) NOT NULL,
  `friend_id` mediumint(9) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `account_id` (`account_id`),
  KEY `friend_id` (`friend_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_bin
AUTO_INCREMENT=1 ;
```

Holds the connections information. I resisted the temptation of normalizing to the extreme because that would have generated slower queries afterwards. A possible structure could have been
connection_id | person_id
connection_id | person_id
[..]

It's worth being noted that the connections are a two-way street in the case I implemented: if x is friend with y, y is also friend with x.

```
CREATE TABLE `cities` (
  `city_id` mediumint(9) NOT NULL AUTO_INCREMENT,
  `city_name` varchar(200) COLLATE utf8_bin NOT NULL,
  `rating` double NOT NULL,
  PRIMARY KEY (`city_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_bin
AUTO_INCREMENT=1 ;
```

The cities' information. City ranking is being computed automatically during the import process. For now, the rank is pretty goofy (only a classic average). A more correct rank would take into consideration each user's influence, the average of all notes in the system, per-user averages and many other factors. Details on that are here, for instance: http://www.marinbezhanov.com/web-development/18/calculating-average-rating-the-right-way-using-php-and-mysql/
If needed I can implement it.

```
CREATE TABLE `cities_ratings` (
  `id` mediumint(9) NOT NULL AUTO_INCREMENT,
  `account_id` mediumint(9) NOT NULL,
  `city_id` mediumint(9) NOT NULL,
  `rating` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_bin
AUTO_INCREMENT=1 ;
```

The ratings - each rating is submitted by an account_id (belongs to a user) and is related to a city_id (it's for a specific city).

The main SQL queries (and their OOP implementation) are here:
https://github.com/amorroxic/SocialNetwork/blob/master/application/modules/default/models/Accounts/Manager.php
https://github.com/amorroxic/SocialNetwork/blob/master/application/modules/default/models/Cities/Manager.php

3. General overview
==========
The application implements the MVC design pattern. It has two modules - the front-end and the API.
Most of the classes reside outside the webroot for added security and a data import class is present (which takes the socialGraph file, analyzes it and populates the database).

The front-end: https://github.com/amorroxic/SocialNetwork/tree/master/application/modules/default
The API: https://github.com/amorroxic/SocialNetwork/tree/master/application/modules/api
The source data file: https://github.com/amorroxic/SocialNetwork/blob/master/configuration/socialGraph.php
Importer: https://github.com/amorroxic/SocialNetwork/blob/master/application/modules/default/models/Import/Manager.php

All controllers extend base controllers which handle all the low-level model instantiations and various other common tasks.
The base controllers are found here: https://github.com/amorroxic/SocialNetwork/tree/master/application/library/Site

4. Points of entry (Data importer, Front-End and API), security and SEO
==========

There are three points of entry.

a) The front-end (which is also a presentation layer) - displays information on all accounts, performs AJAX queries in order to receive relevant information on the account selected and receives API links (xml/json formats) towards that selected account.
The database id for any of the accounts is at no moment displayed. Besides the obvious security benefits, it also exhibits a SEO friendly URI. Front-end makes use of the jQuery library, the Chosen jQuery plugin (which handles the <select> element) and I've developed a jQuery plugin which connects everything together:
https://github.com/amorroxic/SocialNetwork/blob/master/httpdocs/javascript/logic.js

b) API
The API has a version layer. This is done with the purpose of allowing the API to grow while maintaining compatibility with clients supporting older versions of the API. Let's be developer friendly here :)

The api links are in the following format
http://social.nouauzina.ro/api/v1/friends/of/[user-slug]
http://social.nouauzina.ro/api/v1/friends/of/[user-slug]?format=json
http://social.nouauzina.ro/api/v1/friends-of-friends/of/[user-slug]
http://social.nouauzina.ro/api/v1/friends/of-friends/of/[user-slug]?format=json
http://social.nouauzina.ro/api/v1/suggested-friends/for/[user-slug]
http://social.nouauzina.ro/api/v1/suggested-friends/for/[user-slug]?format=json
http://social.nouauzina.ro/api/v1/recommended-cities/for/[user-slug]
http://social.nouauzina.ro/api/v1/recommended-cities/for/[user-slug]?format=json

The user-slug is being cured (regexp) allowing only a-z, A-Z, 0-9 and '-' characters in order to prevent SQL injection or XSS.
It also only answers to AJAX (XML HTTP) POST requests.

c) The importer
The importer process receives the socialGraph.php file, regenerates the account id's, sanitizes the gender, age and people's name, computes rankings for cities and inserts everything into the database. It is not behing a password-protected area as the system is only a demonstration for now.

Most of the links are or can be easily implemented to be SEO friendly. The front-end makes heavy use of AJAX calls (which ain't the best SEO route, but the purpose here was to demonstrate stuff, highlighting different ways of accessing the data).

The URI is never trusted as it is: it's being checked and sanitized. Nor is the source datafile trusted fully.

5. Re-usable components
==========

Wherever possible, a strong adhesion towards the OOP principles is being implemented. Classes inherit each other and there are lots of components freely available to be reused:

- the data models:

https://github.com/amorroxic/SocialNetwork/tree/master/application/modules/default/models

- controller plugins (cache, profiler, gzipped output):
https://github.com/amorroxic/SocialNetwork/tree/master/application/library/Plugins

- helpers (slug generation / handling of data)
https://github.com/amorroxic/SocialNetwork/tree/master/application/library/Helpers

The API can be extended to support other formats (AMF for instance).

6. Performance
==========

There are components which monitor the system (memory usage/execution times) which are plugged into the architecture. Also, there is a cache system in place which I did not have time to link directly with the data models.

Basically, the cache (which for now is only a file-level cache, could be extended to support other backends such as memcached) would recognize a specific url, load a precached output and send it towards the user, avoiding querying the database.