MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Design and implementation of a social network for making acquaintances

BACHELOR THESIS

Marek Bryša

Brno, spring 2012

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Marek Bryša

Advisor: doc. Ing. Michal Brandejs, CSc.

Acknowledgement

Thanks

abstract

Contents

Introduction

1 Design

1.1 Existing social networks for making acquaintances

1.1.1 PlentyofFish

PlentyofFish (`http://www.plentyoffish.com/`) was founded in 2003 in Canada. It generates most of it's revenue through advertising and some premium services. Unfortunately, it currently only serves users from Canada, UK, US, Australia, Ireland, New Zealand, Spain, France, Italy and Germany so the author could not sign up at all.

From the publicly available information, it allows users to create a profile, search for others, message and chat with others. A 'Chemisty test' and some other methods of finding a match are offered, but without exlaining precisely how they work.[7]

1.1.2 Match.com

Match.com (`http://www.match.com`) was launched in 1995 and is one of the oldest networks. It requires a paid subscription of ranging from 34.90EUR for one month to 77.40EUR for 6 months.

After signing up, the user is asked to upload a profile photo and fill in a detailed questionnaire about his or her character, interests, activities and relationships and preferences. Based on this information, the system tries to find the best matching partner. The user can then add the match to his or her favourites, follow their profile and message them. There is a special option to 'wink' at them, which can be used to quickly bring attention of the match and wait for their response to quickly assess their general interest without the need to send a message.[6]

1.1.3 OkCupid

OkCupid (`http://www.okcupid.com`) started in 2004. It claims to be the fastest growing site. TODO: use dating???
It is ad-supported and the essential features are free to use. A paid subscription called 'A-list' is also available for 14.95USD/month. It

removes the ads, allows for advaced search, changing of username etc.

Matches can be found through search using general criteria or by filling out questionnaires. A user can also create a his own questions, set their importance and expected answers. When another user fills them in, the system calculates a match percentage. This process is probably unique to OkCupid.[5]

1.1.4 eHarmony

eHarmony (`http://www.eharmony.com`) is a paid service that was launched in 2000. It claims to have more than 33 million members. Subscriptions cost from 59.95USD for a month to 239.4USD for 12 months. It is primarily focused on finding a partner for marriage.

The service uses personality tests, mathematical matching and expert advice to find the best match. There are separate subsites targeted for specific social groups such as Asians, Christians, Jews, gays and lesbians etc. A new user has to fill in a very detailed questionnaire about his current status, personality and preferences.[2]

1.2 User data protection

When using this kind of social networks, the user usually has to provide information about himself that is very sensitive and even intimate. Protection of this data is therefore a very serious concern.

The data is very valuable beyond it's original intent to find the best match. It can be used for instance to precisely target advertisments, give offers to buy new products and so on. Hence it is essential that the user is made clear how the information he enters on a website is used or if it is disclosed to third parties.

The user should also have the ability to choose what data is shared with other users. In the best case this control should be very fine, i.e. the user should not be forced to share information in blocks, should be able to deny concrete user from viewing his profile or parts of his profile etc. There should also be a simple tool to preview one's profile in the way others can see it.

If a user deletes any data on his profile, it should be physically

deleted from all the servers as well, unless it is expressly stated otherwise (e.g. for backup purposes).

Any changes to the privacy policy of a website should be only done with sufficent prior notice and preferabely be opt-in. The user must have the ability to close his account. In this context, it also very important for the user to able to simply download all his data in a package.

The language of the privacy policy should be as simple as possible, for every user to clearly understand it. Almost noone will read a lengthy legal text, which can lead to unfortunate misunderstandings later.

It goes almost without saying that the servers must be well protected from hacker attacks, especially when they contain this kind of sensitive data. A successful attack would not only harm the users, but probably mark the end for the website. Ideally there should be a regular security audit that the users can review.

1.3 The idea

TODO: [3] From the research of existing social networks for making acquaintances we can conclude the following points and issues:

- The target audience are single people from their late 20's to about 60 years old.
- Many require a paid subscription to access even the most basic functionality.
- All require new users to fill in a long, detailed and intimate questionnaire. This can discourage many users.
- Therefore all collect very sensitive user data that could be potentially misused.
- All offer a method to quickly find a matching partner, but then require an action from one of the users to make a first contact. Some users might have trouble finding courage to do so.

To solve most of these issues, the author has come up with this idea for the new social network:

- The users will provide only general information: e-mail address, gender, year of birth, approximate location (county level), interest in men or women and a single profile photo.
- Based on simple search criteria such as age range, relative location to them (i.e. same county, neighbouring counties, etc.), they will browse profile pictures of other users one by one and mark the ones they like.
- Only once two users match their mark, both will be notified, added to their contact lists and be able to engage in real-time chat. Then they can get to know each other and possibly arrange a meeting.

This way only very little information is gathered in the database, which brings the user data privacy problems to minimum and it is not needed to fill in any lengthy questionnaires. User need not be shy when marking people they like, because until the mark is matched, the other person will not know about it.

However this also brings some new issues. Because the marking of others is essentially only based on their looks, the target audience is going to be reduced to users for whom it is an important criteria. That means mostly younger people seeking fun rather than a serious relationship.

2 Implementation

2.1 Technologies

It is very important to choose the right technology for the implementation of a project. We need to find the most suitable web application framework and a data store, if one is not hard-wired into the framework. The author has devised the following criteria for the evaluation of available technologies:

- **Availability for commercial use free of charge**
  Because of budgetary constrains, the technology must be free for commercial use. The project may later generate revenue through the use advertising.
- **General suitability for the project**
  It must facilitate creation of a website. It is expected that there will be a lot of HTTP requests that will make only little chages to the database, e.g. marking of photos a user likes. The data model will be quite simple. There must be an easy way of making HTTP push[1] communication to enable real-time chat.
- **Performance, scalability and stability**
  Again due to the low bugdet, the software must utilize the hardware as efficiently as possible. The user base could potentially grow very rapidly. It is therefore essential that all the system can match the growth cost efficiently. The framework should have a good track record of runtime stability.
- **Ease of development and developer community size**
  It should be easy to implement the project and good documentation is welcome. The framework should have a good community with which a developer can try to solve issues.
- **Codebase stability**
  The technologies should be past their rapid development phases

1. "HTTP server push (also known as HTTP streaming) is a mechanism for sending data from a web server to a web browser." `http://en.wikipedia.org/wiki/HTTP_push`, 2012-04-08

and the core APIs should be stable. This minimizes the effort needed to transition the project to a newer version of the framework.

- **Innovation factor**
  Younger technologies are preffered as their use can lead to innovation and discovery of new approaches to problems.

Because of the first criterion, our interest shall only be in open source frameworks.

2.1.1 General suitability for the project

The author is skilled in JavaScript, PHP, Python and Ruby, so we will further examine frameworks based on those languages. All have been used for HTTP server programming for a long time, except for JavaScript, which has emerged in recent years in the Node.js platform.

	Node.js	PHP	Python	Ruby
Simple	Express.js	plain PHP	CherryPy	Sinatra
Full MVC	Locomotive, Railway.js	Zend, CakePHP	Dajngo, web2py	Ruby on Rails

Table 2.1: Classification of web frameworks

Table 2.1 shows a basic classification of selected web frameworks by programming language and complexity of features they provide. Simple frameworks generally only provide a way to route HTTP requests to methods, parse HTTP headers and to send a response. Other features can be added on using plugins or modules. Full MVC[2] frameworks also have an ORM[3] engine for models and generate HTML views using a templating engine.

Because the project's uncomplicated data model would not utilize the complex feature set of full MVC frameworks and those could limit flexibility, we will further only focus on the simple ones, i.e. Express.js, plain PHP, CherryPy and Sinatra.

2. Model-View-Controller
3. Object-Relational Mapping

2.1.2 Performance, scalability and stability

Let us first compare performance of the languages and their virtual machines themselves. We can use results from *The Computer Language Benchmarks Game* [4]. It uses several algorithms written in different programming languages to measure their speed.

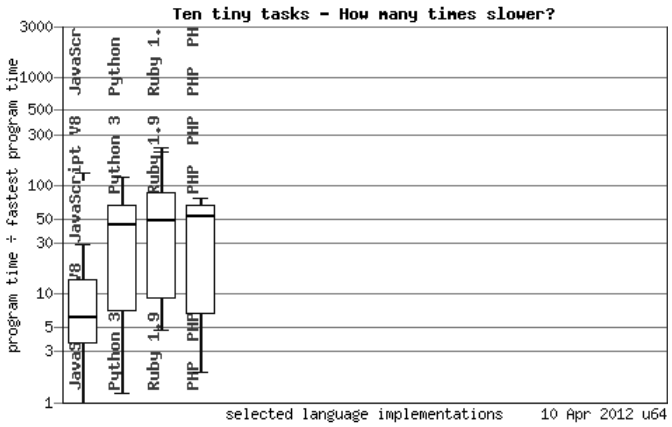

Figure 2.1: Language benchmark of V8 Engine, Python 3, Ruby 1.9 and PHP 5.4.0. Lower means faster. Source: [4]

A box plot of the benchmark is shown in figure 2.1. The vertical axis means how many times the language is slower than the fastest one (Intel Fortran 12.1). Out of the languages under consideration, Node.js on average 6 times faster than the rest[4].

Next we will benchmark the performance of the HTTP handling of the frameworks. All the tests will be run on a dual-core i686 Linux 3.2.11 PC with 4GB of RAM. We will use the Apache HTTP server benchmarking tool (`ab`). Source codes for Sinatra and Node.js are in listings 1 and 2, the other two for PHP and CherryPy are on the attached CD.

The `add` method has two parameters `a` and `b` and simply returns their sum. It will simulate simple `GET` parameter parsing and response.

The `sleep` method has one paramater `ms`. Execution is suspended for `ms` miliseconds and a simple response is sent. This is intended to

4. Node.js uses the V8 Engine internally

Listing 1 Benchmark for Sinatra in Ruby

```ruby
require 'sinatra'

get '/add' do
    (params[:a].to_i+params[:b].to_i).to_s
end

get '/sleep' do
    ms=params[:ms].to_i
    sleep(ms/1000.0)
    "Slept %s miliseconds." % ms
end
```

Listing 2 Benchmark for Node.js in JavaScript

```javascript
var app = require('express').createServer();
var util = require('util');

app.get('/add', function(req, res){
  var x=parseInt(req.param('a'))+parseInt(req.param('b'));
  res.send(x.toString());
});

app.get('/sleep', function(req, res){
  var ms=parseInt(req.param('ms'));
  setTimeout(function() {
    res.send(util.format('Slept %s miliseconds.', ms));
  }, ms);
});

app.listen(3000);
```

simulate a database query that takes given time. We will use a 20ms delay.

Node.js uses its internal HTTP server, Sinatra uses the Thin server, CherryPy uses its internal WSGI server and PHP is hosted through `mod_php` on the Apache server. Unlinke CherryPy and Apache which use a thread pool to serve requests, Node.js and Thin use the `libevent` that utilizes `epoll` on Linux and `kqueue` on FreeBDS theoretically allowing for better concurrency. Node.js is also strictly single-threaded.

Here is a list of versions and parameters used:

- Node.js 0.6.13
- PHP 5.3.10, Apache 2.2.22
- Python 3.2.2, CherryPy 3.2.2, 100 threads in the pool
- Ruby 1.9.3p125, Sinatra 1.2.7, Thin 1.3.3
- All logging including access is disabled.
- 5000 request per test
- Concurrency in set (1,10,30,50,100,200,300,500,700,1000)
- `ab` is run 10 times for each parameter set and a mean is calculated.
- `/proc/sys/net/ipv4/tcp_tw_reuse` set to 1. This allows for reuse of sockets in the `TIME_WAIT` state. This is a recommanded setting for high concurrency web servers.
- 2 seconds wait time between each run of `ab`
- In case one of the runs fails (i.e. any of the 5000 requests fails), a score of 0 request per second is awarded for the run.
- Source code of the Python script used to perform the benchmark can be found in attached file `bench.py`.

Graph 2.2 shows the results of the `add` benchmark. We can see that PHP keeps up with Node.js until the 100 concurrent requests mark, but then declines sharply. Node.js is able to serve about 4700 request per second regardless of concurrency.

Graph 2.3 shows the results of the `sleep` benchmark. Node.js is again the clear winner with about 4700 request per second regardless of concurrency. This is because Node.js's `setTimeout` is non-blocking. This applies also to any database query. Once the query is made, Node.js moves to serve other requests.

Node.js and Sinatra on Thin are the only framework to remain stable with increasing load. PHP and CherryPy have started dropping requests at concurrency levels of 300 and 30 respectively.

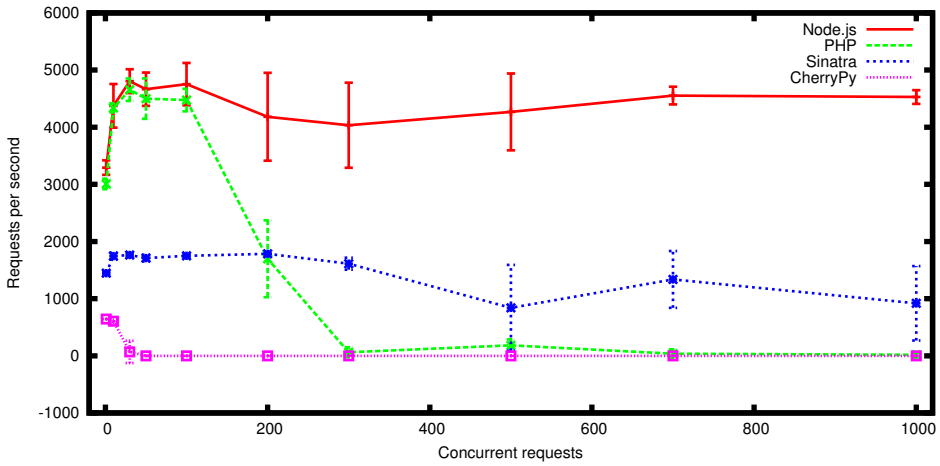

Figure 2.2: Benchmark of the `add` method. Error bars denote standard deviation.

In conclusion, Node.js performs the best as both a language and a HTTP server framework and remains stable under any load. Sinatra is also stable, however it is slower, which could be partially solved by the use of clustering. Under low loads PHP on Apache is just as fast as Node.js, but it's stability is hardly acceptable. CherryPy is eliminated.

2.1.3 Ease of development and developer community size

All the frameworks in the comparison provide very similar levels of functionality. From the author's experience, Ruby allows for shortest code at the slight expense of readability. Javascript on the other hand requires the longest code and can be a little tricky. See aganin listings 1 and 2.

PHP is arguably the most used framework for web programming, hence it has the biggest developer community. It is well documented in many languages. Countless modules, snippets and add-on libraries are available.

Node.js and Sinatra have a smaller but very active community. Thousands of modules and add-on are available through their pack-

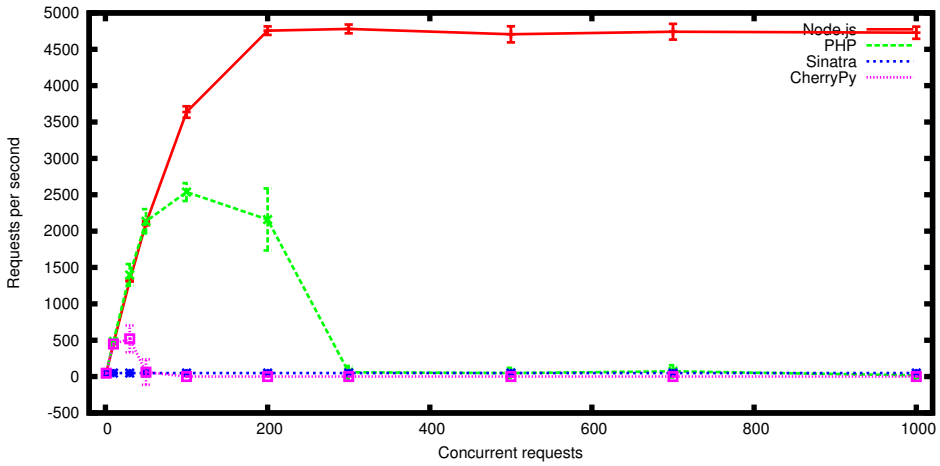

Figure 2.3: Benchmark of the `sleep` method. Error bars denote standard deviation.

age managers such as npm for Node.js and RubyGems for Sinatra. Both are adequately documented if not deeply.

2.1.4 Codebase stability

PHP's codebase is the most stable. It is the oldest framework in the comparison, laguage and API changes are almost non-existen and don't break backwards compatibility.

Sinatra and Node.js split the second place. There have still been some changes to the Ruby language from version 1.8 to 1.9 that could break backwards compatibility. Node.js is still under active development, however the API tends to only grow and not change.

2.1.5 Innovation factor

Node.js is the clear winner in this criterion. It is the most innovative in that it uses JavaScript on the server side, event-based polling and non-blocking API.

Sinatra comes in the second place. It utilizes Ruby's advanced language constructions allowing for tidy coding. PHP finishes last because of its age and lack of modern approaches.

2.1.6 The overall winner: Node.js

The author has chosen Node.js for the implementation of this project. It is just as suitable as other frameworks and clearly wins performance, scalability and stability tests. It is also the most innovative framework of recent years with growing developer community support.

There are two drawback to using Node.js. Firstly it is still relatively immature and bigger API chages could still happen. Secondly the use of JavaScript and non-blocking function calls can lead to poorly readable code. The author however believes that the advantages greatly outweigh this.

2.1.7 Data store

The decision to use simple web framework gives us a free hand in choosing a separate data store. The traditional choice is an SQL database, e.g. MySQL and PostgreSQL. Lately NoSQL databases (MongoDB, CouchDB etc.) and advanced key-value storages such as Redis have come into focus. In a recent paper *Social-data storage-systems* [8] that compares all of above, no clear winner is given.

The author has chosen Redis. "Redis is an open source, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets."[1] Redis keeps the entire database in memory with optional regular persistent storage snapshots, which allows for very fast read/write access and good reliability. Benchmarks such as [9] show that Redis is about eight times af fast as MySQL when it comes to simple operations.

One of the drawbacks is that Redis mostly has simple commands so even moderately complex operations are difficult to program.

The innovation factor is high because Redis is usually not used as a single storage for all the data. As a bonus, Redis contains a simple publisher-subscriber functionality, which will come handy when implementing the real-time chat.

2.1.8 Modules and libraries used

The following libraries and modules will be used to simplify implementation of the project:

Underscore.js "A utility-belt library for JavaScript that provides a lot of the functional programming support that you would expect in Prototype.js (or Ruby), but without extending any of the built-in JavaScript objects." [5] It will be used in both client and server JavaScript code.

Express.js A simple web framework on top of Node.js. Allows for easy routing, GET and POST method parameter parsing and response sending. Uses the common *Connect* architecture, therefore is also a base for other modules. `http://expressjs.com/`

node-jade A library for the Jade HTML templating engine. `http://jade-lang.com/`

redis A Node.js Redis client. Performance can be enhanced using the *hiredis* native backend. `https://github.com/visionmedia/connect-redis`

connect-redis A *Connect* module for saving of user session data to the Redis data store. Uses a signed cookie for client identification. `https://github.com/visionmedia/connect-redis`

node-sechash A library for calculation of cryptographically secure hashes to be used to store passwords. Automatically adds salt. `https://github.com/kbjr/node-sechash`

formaline An advanced HTTP POST request parser. Especially useful to handle file uploads. `https://github.com/rootslab/formaline`

node-gm A Node.js GraphicsMagick library. Facilitates image manipulation such as resizing, cropping and format coversion. `http://aheckmann.github.com/gm/`

---

5. `http://documentcloud.github.com/underscore/` 2012-04-18

i18n-node A simple internationalization library. `https://github.com/mashpie/i18n-node`

async A library to simplify asynchronous function calls on arrays of data, typically in series or parallely. `https://github.com/caolan/async`

RedBack "A fast, high-level Redis library for Node.JS that exposes an accessible and extensible interface to the Redis data types." [6] Its *RateLimit* class will be used for spam protection.

node-recaptcha A Node.js reCaptcha service client. Used for spam prevention. `https://github.com/mirhampt/node-recaptcha`

2.2 Basic functionality

2.2.1 User registration

2.2.2 Profile photo upload

2.2.3 Acquaintance selection

2.2.4 Notifications

2.2.5 Chat

2.3 Implementation in detail

2.3.1 Security

2.3.2 I18n

2.3.3 Geolocation

2.3.4 Graphical design

6. `http://redbackjs.com/` 2012-04-18

3 Conclusion

Bibliography

[1] Citrusbyte. Redis. http://redis.io/, 2012. [Online; accessed 10-April-2012].

[2] eHarmony, Inc. Why eHarmony? http://www.eharmony.com/why/, 2012. [Online; accessed 31-March-2012].

[3] Eli J. Finkel, Paul W. Eastwick, Benjamin R. Karney, Harry T. Reis, and Susan Sprecher. Online dating. *Psychological Science in the Public Interest*, 13(1):3–66, 2012.

[4] Brent Fulgham. The Computer Language Benchmarks Game. http://shootout.alioth.debian.org/, 2012. [Online; accessed 10-April-2012].

[5] Humor Rainbow, Inc. Okcupid. http://www.okcupid.com, 2012. [Online; accessed 27-March-2012].

[6] Match.com, L.L.C. Match.com. http://www.match.com, 2012. [Online; accessed 27-March-2012].

[7] Plentyoffish Media, Inc. Plenty of FAQ. http://www.pof.com/faq.aspx, 2012. [Online; accessed 27-March-2012].

[8] Nicolas Ruflin, Helmar Burkhart, and Sven Rizzotti. Social-data storage-systems. In *Databases and Social Networks*, DBSocial '11, pages 7–12, New York, NY, USA, 2011. ACM.

[9] Ruturaj. Redis, Memcached, Tokyo Tyrant and MySQL comparision. http://www.ruturaj.net/redis-memcached-tokyo-tyrant-and-mysql-comparision/, 2009. [Online; accessed 10-April-2012].