Real-time chat service for WunderBarKids

Pradnyesh Sawant

January 22, 2016

1 Requirements

Realtime chat cross browser, device and platform with video sharing service (Any language)
Time : 1 Week

- You are required to create a service to support realtime chat from multiple devices (browser, native clients, mobile etc)
- User should be able to share videos
- Client connectivity is going to be choppy hence service need to support sync methods
- Design the platform (DB, Services, Interfaces) to support the above requirement
- Design a simple REST api to aggregate a weekly information on who , what and when used this API

You should be able to answer the assumptions that you make for performance, extendibility and scalability.

2 Introduction

Even though I was required to create just the service, I created both the service and a web-based frontend, because of the following reasons:

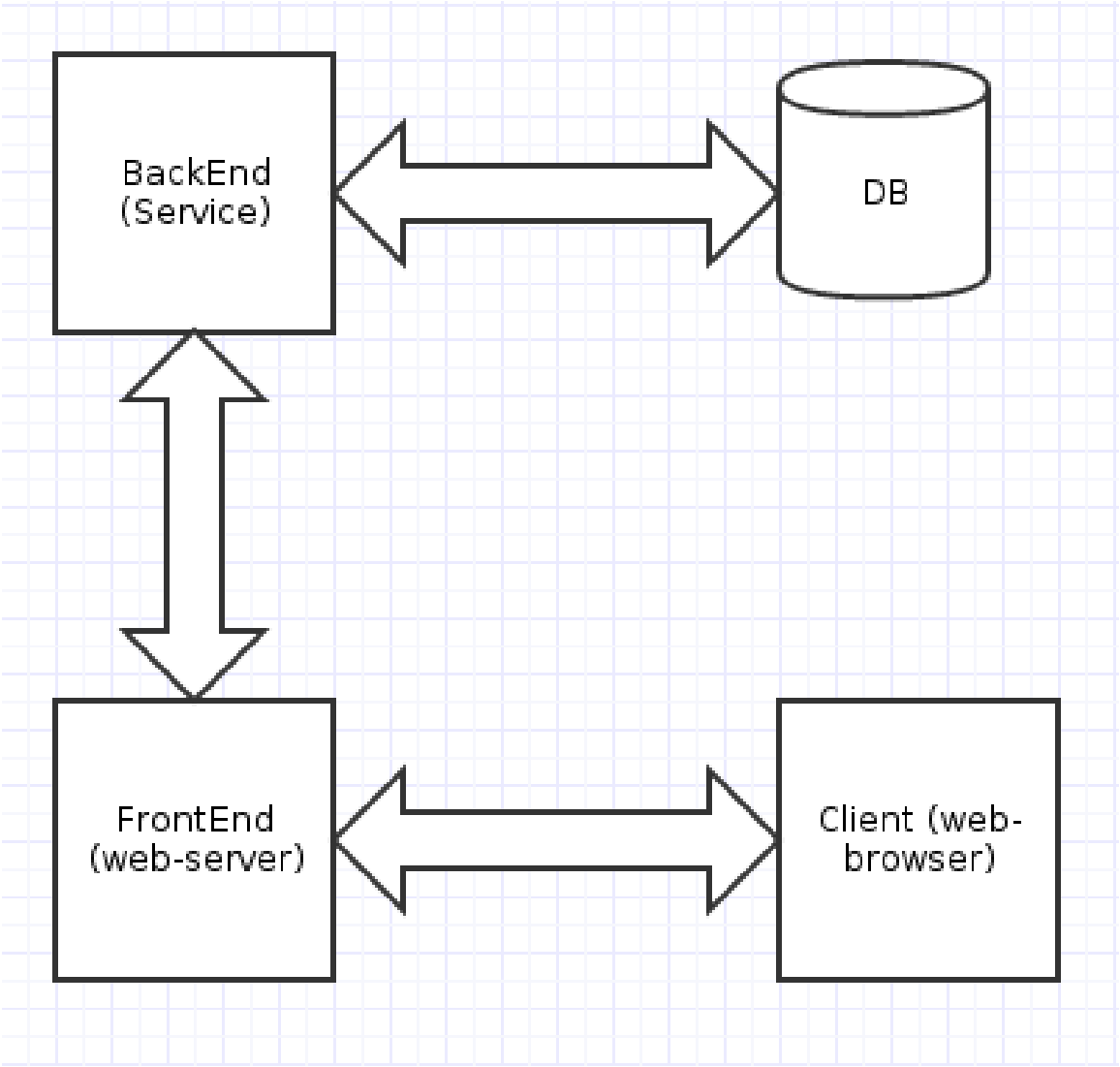
- Service would have been a "pull" model, and would have suffered from performance issues (see "Performance Considerations -> WebSockets" section below)
- I was able to demonstrate my knowledge about the whole stack

3 Service APIs

URI	Type	Parameters	Functionality	Remarks
/register	POST	email, passwd, passwd2, first-name, last-name	Register user	
/login	POST	email, passwd	Login	
/logout	POST	token	Logout	
/send-msg	POST	token, to, msg	Send messages	Deleted, in favor of websockets (see below)
/share	POST	token, to, vid	Share videos	
/sync	GET	token, msg-id	Sync messages	pass last-see-message-id
/sync-vids	GET	token, msg-id	Sync videos	pass last-see-message-id
/reports	GET	–	Weekly reports	NOTE: only till the last week, nothing for current week

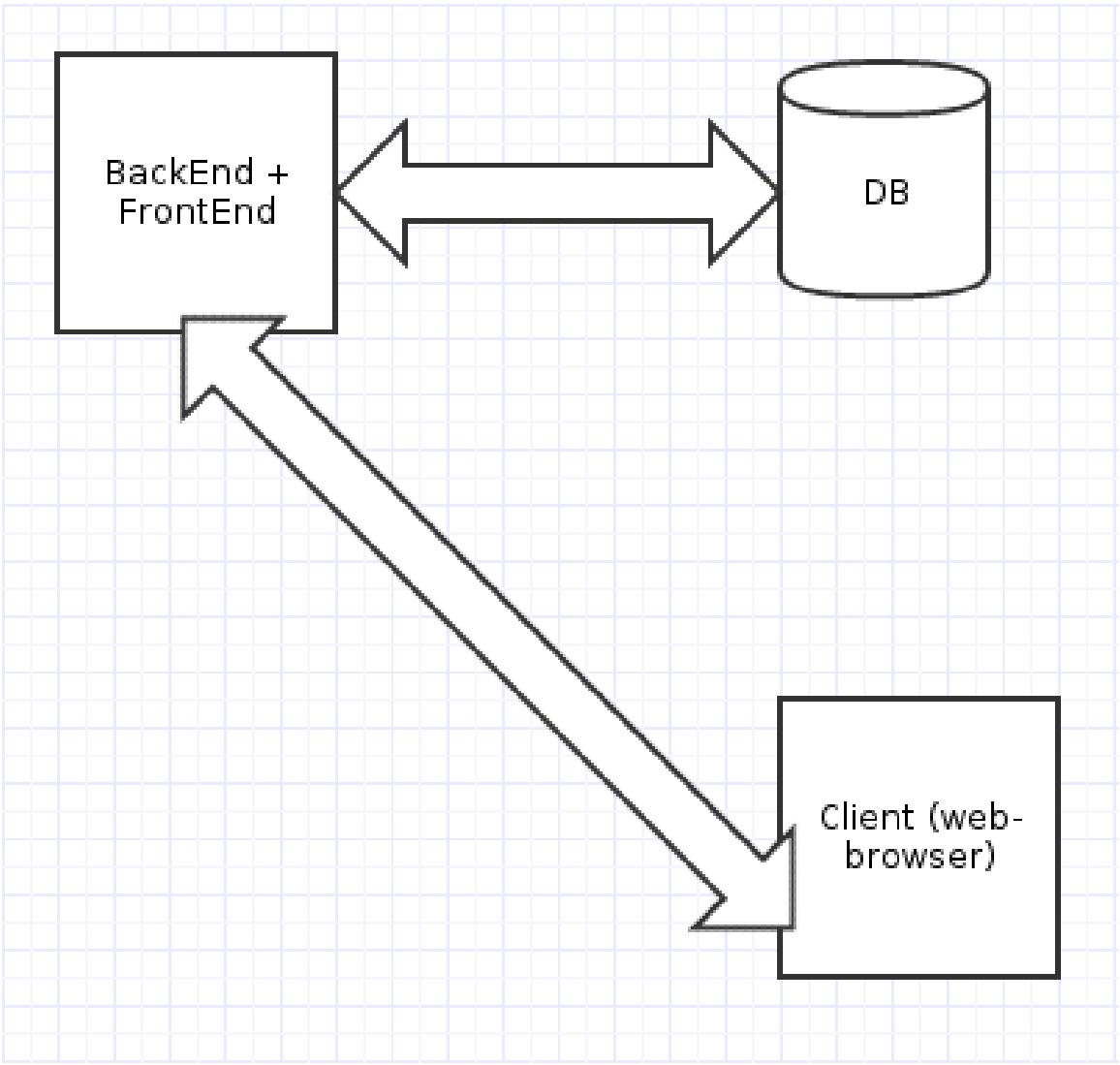
4 System Design

In the real world scenario, the system design should look as shown below:



However, in our case, it looks like as shown below. This is because of the following reasons:

- Due to security policies, JS can only talk to it's originating server => backend and frontend need to be separate servers
- It would have been an overkill (and too much effort) to create and maintain 2 servers (backend and frontend)
- So I ended up clubbing the 2 servers, but keeping the actual code modular such that it can be easily separated in the near future if need be



I have used the following stack:

- DB: postgresql
- web-server: http-kit (http://www.http-kit.org/)
- programming language: clojure+clojurescript
- (notable) libraries used:
  - luminus: initial project template
  - selmer: html templating
  - ring: server-side middleware
  - bouncer: encryption
  - buddy: validation
  - cronj: cron-jobs
  - bootstrap: css
  - figwheel: rapid clojurescript development
  - reagent: clojurescript wrapper to FaceBook react.js
  - cljs-ajax: for AJAX

5 Performance Considerations

5.1 WebSockets

In the initial design, I had a "/send-msg" POST API. However, while creating the frontend I realized that this is a extremely non-performant way of doing things, because it was necessary to call the "/sync" API multiple times every second to achieve the "real-time" effect. This would have, obviously, created exponetially more requests as the number of clients grew. And, even though the response body was pretty small (actually, empty, most of the time), the sheer number of requests to the server would have been too much.

The obvious solution was to use "websockets" and get rid of the "send-msg" API. The advantages of using this approach are:

- The load on the servers (frontend -> backend -> database) reduces to less than 1% of it's previous value

The disadvantage of using this method are:

- File sharing cannot be done over websockets. So we need to create another pair of APIs ("/share" and "/sync-vids"). Also, to keep server load low (and have a non-realtime effect), we set the interval for calling the "/sync-vids" API to every 10 seconds

5.2 Cron Jobs

Cron jobs are needed for the following 2 tasks

- Weekly report generation (not really a performance thing)
- Delete old (seen-by-user) messages and shared-files: this will reduce the DB size and disk usage and will thus improve performance

Both the jobs run on a weekly schedule

6 Miscellaneous

6.1 State

I have used "atom" (a clojure data-structure) to maintain in-memory state, like

- last-seen-msgs for every user
- websocket channels
- etc...

These states need not be stored in DB; the worst thing that will happen if this state is lost is that the user will find himself logged-out from the system => a bad user-experience, but not a critical one for this assignment. It can be easily replaced with memcached or redis in a real-world application.

6.2 Demo Videos

You will also find the following demo videos, that demonstrate the running application, in the current folder

- user-registration.mp4
- login.mp4
- real-time-chat.mp4
- sync-feature.mp4
- video-sharing.mp4

6.3 Running The App

To deploy and run the app on a local machine, please follow these steps (steps prepended with a $ need to be executed at the terminal):

- Install java (jdk)
- Install postgresql (and create database with following credentials)
  - DB name: wbk_dev
  - DB username: wbk
  - DB passwd: wbk
- Install leiningen (http://leiningen.org/#install)
- $ git clone https://github.com/spradnyesh/wbk-chat-be
- $ cd wbk-chat-be
- $ lein migratus migrate
- Now in 2 different terminals run the following commands:
  - $ lein run
  - $ lein figwheel
- visit location “http://localhost:3000” in the browser

6.4 Open-Source Contribution

During implemetation of this assignment, I was able to fix the issue regarding http://www.luminusweb.net/docs/websockets.md#adding_the_routes_to_the_handler (see “***important***” part). I will submit a pull-request for the same shortly.