UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

COS 216 Homework Assignment

- Date Issued: **17 February 2020**
- Date Due: **10 May 2020** before **10:00**
- Submission Procedure: **Upload to the CS web. An assignment upload will be made available.**
- Submission Format: **zip or tar + gzip archive**
- This assignment consists of **3 tasks** for a total of **105 marks**.

1 Introduction

During this homework assignment you will be implementing Web Sockets, this assignment is **based on all the practicals for this module**.

On completion, your assignment must provide the following functionality:

- A NodeJS server polling your PHP API from Wheatley.
- An HTML web page / web site where you would connect to your NodeJS server via a socket to get the progress of the music track / album a specific user has listened to. The progress may be in form of audio or video clip of the track / album.
- You may use your own multimedia player (or any of these: MediaElement.js, jPlayer, JWPlayer, etc).
- The ability to have multiple connections to the NodeJS server using sockets, keeping track of the progress of a specific music track / album by a specific user.

2 Constraints

1. You must complete this assignment individually.
2. You may ask the Teaching Assistants for help but they will not be able to give you the solutions.
3. You must produce all of the source files yourself; you may not use any tool to generate source files or fragments thereof automatically.
4. You assignment must be programmed in NodeJS, HTML, JS, CSS, PHP.

3 Submission Instructions

You are required to upload all your source files in an archive, either zipped or tar gzipped, to the CS website. The assignment submission system on the CS website will close the normal upload form automatically on the deadline. No late submissions will be accepted, so make sure you upload in good time. **Note: Wheatley is not available off-campus be sure to upload and test your work before the weekend it is due! FTP access to Wheatley will not be available during marking weeks (work on localhost).**

4 Resources

Web Sockets - `https://developer.mozilla.org/en-US/docs/Web/API/WebSocket`
`https://en.wikipedia.org/wiki/WebSocket`
`https://github.com/websockets/ws`
`https://socket.io/`
`https://www.npmjs.com/package/websocket`

Trakt - `https://trakt.tv`

Scrobbler - `https://www.last.fm/api/scrobbling`

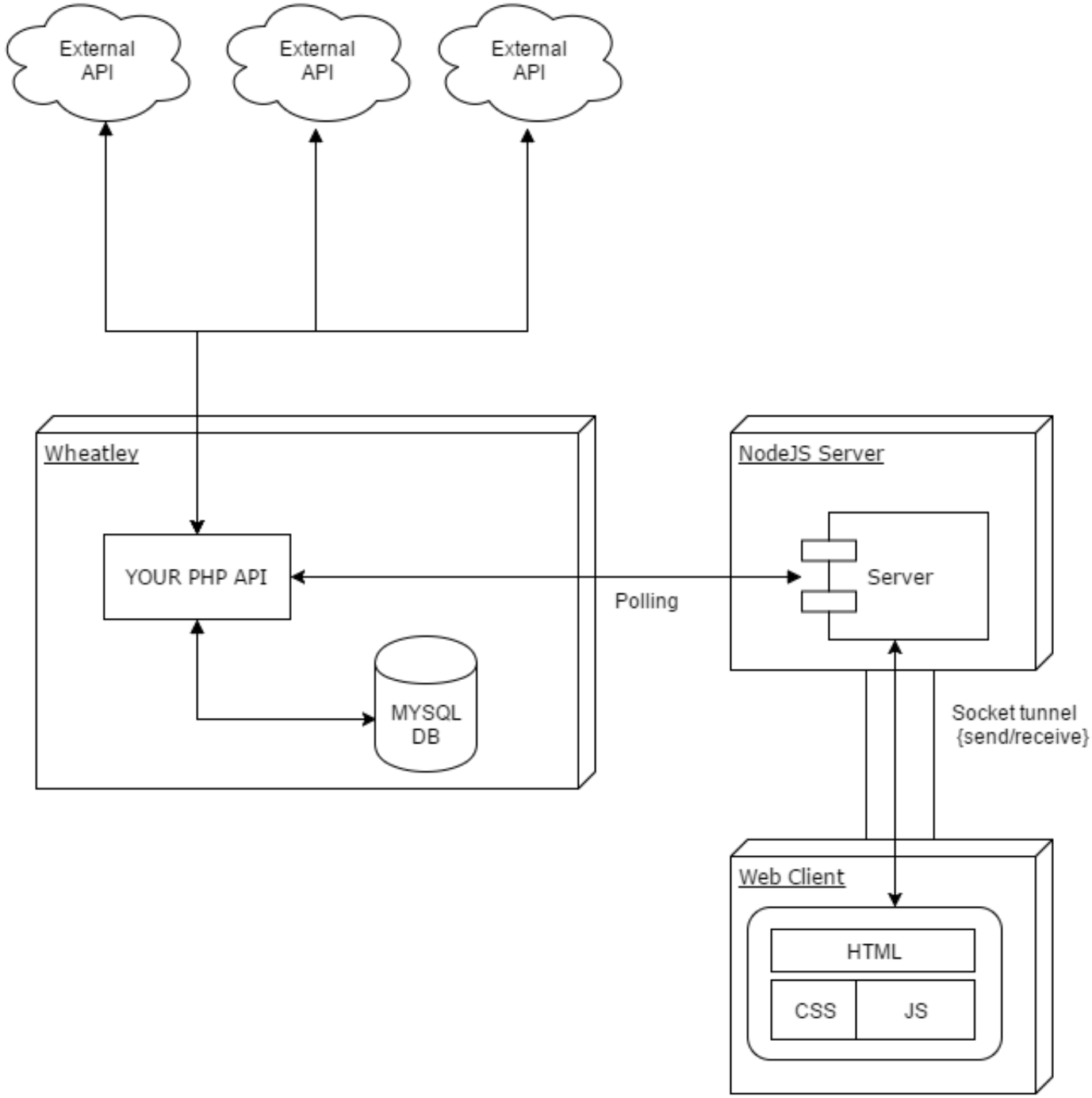
Since NodeJS is **not** available on the Web Server (Wheatley), you will be allowed to demo off your laptops or lab PC's.

5 Rubric for marking

Task 1 - 'track' API type	**20**
Ability to set progress	10
Ability to retrieve progress	10
Task 2 - Multi-threaded server	**55**
Port can be chosen at run-time	3
Server uses multi-threading to accept multiple clients simultaneously	10
'LIST' command is implemented and fully functioning	5
'KILL' command is implemented and fully functioning	10
'QUIT' command is implemented and fully functioning	10
If connection is closed on the client the server closes the socket	5
Server syncs after X seconds	2
Server only pushes changes for consequent interactions on the socket	10
Task 3 - Web Client	**30**
The client displays notifications if socket disconnects	2
The client tries to re-establish a broken/closed connection	3
The client clearly shows change in feed/data	5
The client shows all the data from the PHP API	20
Upload	
Not uploaded to CS web	**-70**
Bonus	**5**
Total	**105**

6 Overview

To understand the flow of how this all works please consult the diagram below.



7 Assignment Instructions

Task 1: 'track' API type ..... (20 marks)

In Practical 3, you will have to build your own PHP API which will provide certain functionality types, one such type is 'track' which simply tracks the progress of media files. For this task you will need to implement the "track" type for the API (which is similar to scrobbler and trakt). This feature simply tracks the progress of a music track / album / audio clip that a user has listened / watched so that it can resume at a later stage on a different device. You should be able to update and retrieve the progress, so make sure you structure your requests and modify your database accordingly. You may just use one file (music track / album / audio clip) to demonstrate this, although having a generic solution may earn you bonus marks. You may use the provided file for this or use your own, but keep the filesize and its length short. Only registered users can track file progress. The remainder of this practical revolves around this functionality.

Task 2: Multi-threaded NodeJS server ..... (55 marks)

For this task you are required to create a **multi-threaded** socket server **(on localhost)**. The server must be coded in NodeJS locally and not off wheatley. You are allowed to use libraries for this.
Note: Wheatley is a web server that follows the LAMP stack and installing other applications on it brings security issues, you must therefore use localhost for this since the NodeJS server needs to run on an open port it would be difficult to allow the server to be run on Wheatley as the chances of students using the same port would be high. Also, this is done to avoid some students from intentionally and unintentionally performing port blocking.

The server must have the following functionality:

- Be able to specify a port that the server will listen on (create the socket), using input arguments or prompt.
- Accept multiple client connections and interact with the clients concurrently through sockets.
- The server must utilize all the functionality of the PHP API you developed/will develop from the practicals.
- Since Wheatley is password protected you will need to include your login details in the URL as follows:

  username:password@wheatley.cs.up.ac.za/uXXXXXXXX/path_to_api

  It is your responsibility to keep your login details as safe as possible, it is recommended that you store your username and password in a global variable and use that variable throughout. Alternatively, you can secure your details through other means. **Bonus marks may be given depending on how secure your solution is**.
- The PHP API needs to make use of an API key which the NodeJS server would need to authenticate itself to the API (you will implement this in Practical 3). What this entails is that your PHP API would need to generate a random key (e.g 8 alpha-numeric random characters) in which you would add to the NodeJS server to each request you make to the PHP API authenticating the request. More details will be given in Practical 3.
- In order to get the latest data the server should sync to the PHP API you created after every X seconds.
- The server should be smart enough to only push data that has changed and not all the data every time the PHP API is synced. **Hint:** Make use of timestamps, your server would send the timestamp of the last request to your PHP API and depending if anything is new a response will be sent.
- The server needs to account for lost sockets (when a socket was created and the client[HTML web page] has closed/disconnected) and close them, to allow for new connections.
- The server should have a `LIST` command to list all the connections it currently has.
- The server should have a `KILL` command to close a connection based on some ID.
- The server should have a `QUIT` command that sends a broadcast to all connections notifying that the server will now go off-line, thereafter closing all connections.

To test the functionality of your server you may use `https://www.websocket.org/echo.html` as the client before proceeding to Task 3.

Task 3: Web Client ..... (30 marks)

For this task you are required to develop a web page that will interact with your server (that runs on your local machine or lab PC [localhost]) you wrote for Task 2. The web client must be implemented in HTML and JS using Web Sockets. **The client should also be on wheatley, but it can also be on localhost**.
Note: You may use any client side library of your choice (e.g. WebSockets, Socket.io, etc.)
The client must have the following functionality:

- The client should have login functionality, no registration needed.
- The client should make use of an HTML 5 audio or video tag to stream the music track / album.
- The client should notify (via text/icon/pop-up) the user when the server has disconnected.
- The client should notify (via text/icon/pop-up) the user when a socket closes and attempt to re-establish the connection.
- The client should clearly indicate/show when new progress has been received to display or indicate (this can be done by simply adding a notification and then advancing the track or video clip to the tracked progress).
- If two tabs are opened and the same music track / clip is being listened to / watched when the progress of a media in one tab advances, the media in the other tab also needs to advance accordingly. Hence, you need to sync the progress using sockets and occasionally send the progress to the API for more permanent progress.