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

COS 216 Homework Assignment

- Date Issued: **4 March 2024**
- Date Due: **20 May 2024** before **08:00**
- Submission Procedure: **Upload to ClickUP. An assignment upload will be made available.**
- Submission Format: **zip or tar + gzip archive**
- This assignment consists of **3 tasks** for a total of **130 marks**.

NB: Please read through the entire specification before asking questions on Discord/Email, as there is a high likelihood you question may be answered later in the specification.

1 Introduction

During this homework assignment you will be implementing Web Sockets. You will be creating an online property auction website, where bidding for properties happen in real-time. The general idea is users can list a property for auction or can join a current auction to bid on the listed property. More details will follow below.
On completion, your assignment must contain the following:

- A PHP API hosted off Wheatley that pulls from a MYSQL DB.
- A local NodeJS socket server polling your PHP API from Wheatley.
- A Web client that connects to your NodeJS socket server and will act at as the front end.

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. **This includes ChatGPT!**
4. All written code should contain comments including your name, surname and student number at the top of each file.
5. You assignment must be programmed in NodeJS, HTML, JS, CSS and PHP. (Bootstrap is allowed).
6. The API and Database should be hosted off Wheatley, the NodeJS server should be hosted locally.

3 Submission Instructions

You are required to upload all your source files and a ReadME in an archive, either zipped or tar gzipped, to **clickUP**. No late submissions will be accepted (See due date and time at the top of the document), so make sure you upload in good time.

4 Resources

NodeJS - https://www.w3schools.com/nodejs/
https://www.tutorialspoint.com/nodejs/index.htm

ExpressJS - https://www.tutorialspoint.com/expressjs/index.htm
https://expressjs.com/
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs

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

Base64 https://www.w3docs.com/snippets/html/how-to-display-base64-images-in-html.html
https://www.base64-image.de
https://www.base64decode.org/

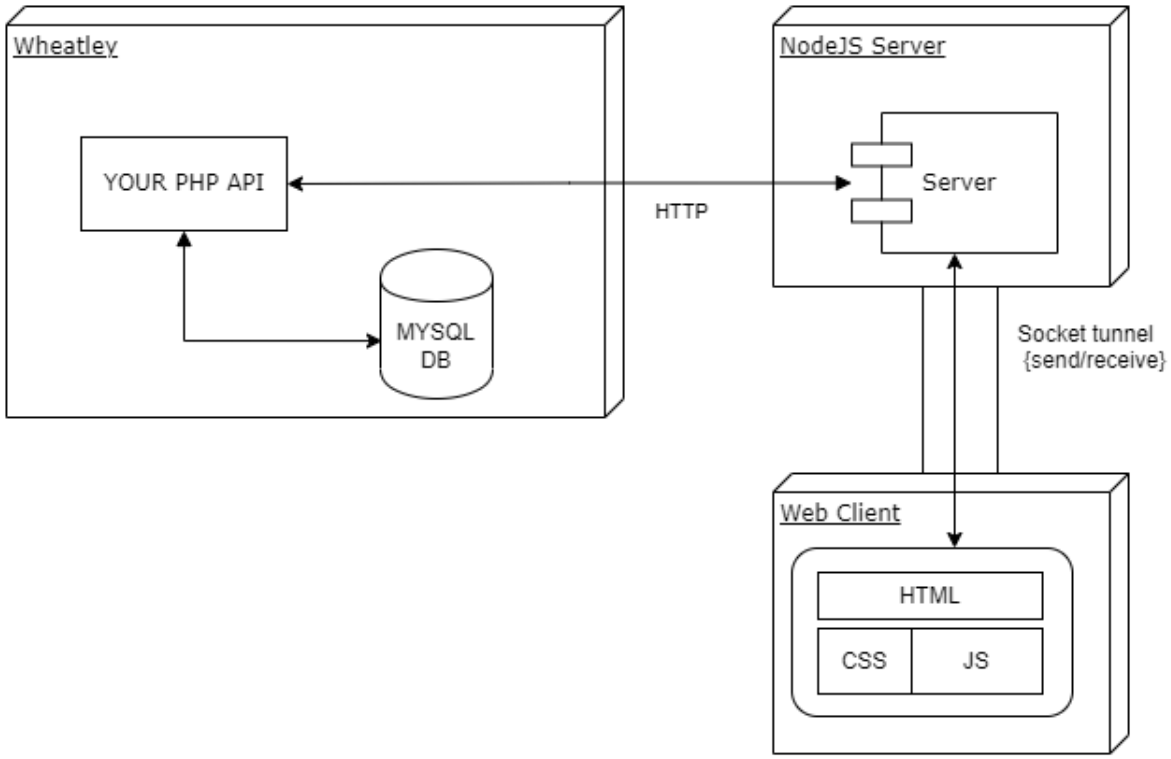
DOM Manipulation https://www.w3schools.com/js/js_htmldom.asp

5 Rubric for marking

Task 1 - PHP API & DB	**30**
MySQL DB	5
Ability to store images in the DB	3
ReadMe explanation, Base64 vs Link	2
Login	5
CreateAuction endpoint	5
UpdateAuction endpoint	5
GetAuction endpoint	5
Task 2 - Multi-User server	**60**
Port can be chosen at run-time	3
Reserved Ports cannot be chosen	2
Server can accept multiple clients simultaneously	5
'LIST' command is implemented and fully functioning	4
'KILL' command is implemented and fully functioning	5
'QUIT' command is implemented and fully functioning	4
'AUCTIONS' command is implemented and fully functioning	5
If connection is closed on the client the server closes the socket	5
Server can detect when an auction should start/end and start/end the auction on the user side	5
Server can send through Auction Details to the client using sockets and the API	5
Server can generate a unique AuctionID	3
Server implements the flow of the Auction correctly	8
Server updates database	4
ReadMe explanation, Update Everytime vs Update Interval	2
Task 3 - Web Client	**35**
The client can connect to a server through a socket	3
The client allows the user to login	3
The client allows user to create an auction or join one	3
The client starts the Auction on the user side on cue from the server	5
Error Messages(Socket disconnected, Bid failed validation etc.)	6
The ability to implement the Auction correctly	15
Security	**5**
Upload	
Not uploaded to clickUP	**-140**
Bonus	**15**
Total	**130**

6 Overview

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



7 Auction details

The auction will be structured as a normal auction where users bid on a property and the highest bidder wins. The auction involves two types of users:

- Auctioneer: User who listed the property.
- Buyer: User who bids on the property listing.

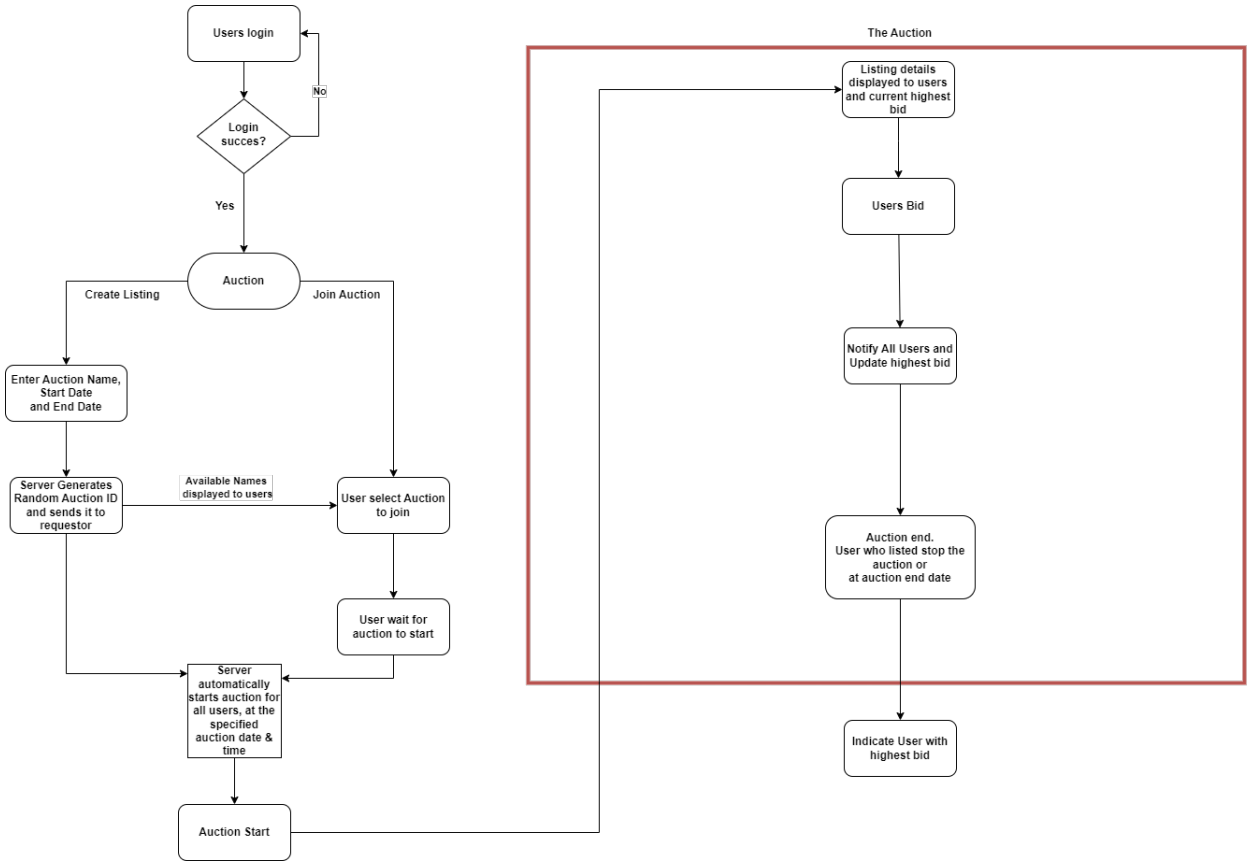
7.1 Auctioneer

The auctioneer will be the user that list the property (specify all property details) and that specify the auction details (Name, Start date and End date). The auctioneer will also have the power to stop the auction at any given time.

7.2 Buyer

The buyer will see the details of the property up for auction as well as the current highest bid. They should be able to enter their bid. Their bid should be validated:

- Client side validation: A number is entered and not any text.
- Server side validation: The bid is higher than the current highest bid.



8 Assignment Instructions

Task 1: PHP API + Database ..... (30 marks)

You need to create a PHP API that is hosted off Wheatley as well as a MYSQL DB along with it. Due to the nature of an auction a simple database will be needed and updated accordingly. The database will be populated as users create auctions, therefore you cannot populate it manually. Users should login before they can create an auction or bid, therefore your api should handle user login and you should also have a table in your database for the users. You will create login functionality in PA4 so it won't carry much weight in the HA.

API:

Your PHP API will need to handle user login and three other endpoints namely: `CreateAuction`, `UpdateAuction` and `GetAuction`.

- CreateAuction - Adds an auction to the database.
- UpdateAuction - Update the relevant fields in the database (Active, Highest Bid, Buyer).
- GetAuction - Returns the details of the auction. This endpoint need a way to return one auction or the names of all active auctions.

Database:

You will need a table for the users and a table with the following fields (You are welcome to add more):

- Auction Name
- Auction Start Date
- Auction End Date
- Listing details (Details specified in Task 3)
- Highest Bid
- State (Waiting, Ongoing, Done)
- Buyer (The state of the auction is not "Done", this should be null)

With regards to the listing image, you have two Options of how you can **transfer images**. *There is no incorrect answer here, but you need to back up your choice.*

- Base64 - you will store images in base64 and decode them client side.
- Link - you will use a link to an external image ***e.g.*** *https://helium.privateproperty.co.za/live-za-images/property/1826/38/7577826/images/property-7577826-14320949_e.jpg*.

You will need to write a short explanation of your choice and why you believe its the better option. Please write this is a ReadMe.txt that you should include with your clickup submission.

Task 2: Multi-User NodeJS server ..... (65 marks)

For this task you are required to create a **Multi-User** 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 and 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. This is also not possible to do due to UP's firewall.

The server must have the following functionality:

- Be able to specify a port that the server will listen on at run-time (create the socket), using input arguments or prompts. It is good practice to block the user from using reserved ports (Ports that have been reserved for other task). However there is still a chance you can choose a port in use, for the sake of

this assignment if this happens, just try a different port. The ports you are allowed to use are from 1024 - 49151, make sure the user can only choose a port in this range. *If you are curious about why this is the case, you can read up about it at* *https://www.techtarget.com/searchnetworking/definition/port-number*.

- Block the user from attempting to open a Reserved Port (Explained above).
- Accept multiple client connections and interact with the clients concurrently through sockets.
- **The server must utilize functionality of the PHP API you developed in Task 1.** Therefore you must call your API endpoints to create, update and get an auction.
- The server should have a `KILL` command to close a connection based on their username. This also means that you need to keep track of which socket ID corresponds to which Username since you need the SocketID to close a socket connection.
- The server needs to account for lost sockets (when a socket was created and the client [HTML web page] has closed/disconnected). These lost sockets should be closed in order to allow for new connections. If that user was in an auction they will simply have to rejoin if they want. This mean users can come and go as they please.
- The server should have a `LIST` command to list all the connections it currently has, be sure to list the usernames and their respective SocketIDs.
- 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.
- The server should have a `AUCTIONS` command that shows all auctions in session or created, The AuctionID, AuctionName and the users currently bidding usernames.
- The server should be able to generate a Unique AuctionID. This should be a random 10 Character Alpha-Numeric String.
- The server should be able to start an auction automatically at the time the auction is set to start and end the auction at time it is set end. It is recommended that you get all active auctions when you start the server so that the server have the times of all auctions.
- Once the auction has started take note of the following from the server side
  - The server should notify the user when the auction is about to start.
  - The server should call your API to update the auction.
  - The server should handle incoming bids.
  - The server should be able to implement what is shown in the Flowchart under Auction Details.

  You have two options of how you can update the database. *There is no incorrect answer here, but you need to motivate your choices.*
  - Every time there is an update - This option mean that you will poll your API `UpdateAuction` every time there is a new bid or a change.
  - In an interval - This option will only poll your API `UpdateAuction` in a certain interval, this mean that you will have to keep some local history of the the bids and bidders.

  You will need to write a short explanation of your choice and why you believe its the better option. Please write this is a ReadMe.txt that you should include with your ClickUp submission.
- 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.
NB: Ensure that you do not submit your code with your passwords included in your clickUP submission.

To test the functionality of your server you may use `https://www.piesocket.com/websocket-tester` as the client before proceeding to Task 3.

Task 3: Web Client ..... (35 marks)

For this task you are required to develop a web page / web site that will interact with your server (that runs on your local machine [localhost]) you wrote for Task 2. The web client must be implemented in HTML, CSS and JS using Web Sockets. **The client should also be on localhost**.
Note: You may use any client side library of your choice (e.g. WebSockets, Socket.io, etc.). To make the changes you can/should make use of DOM Manipulation with javascript, that way you do not need to redirect to a new page everytime which might disconnect the socket

The client must have the following functionality:

- The user should be able to enter login (Enter username and password). User registration is not needed.
- After the user has logged in they have 2 options:
  - Create an auction. The user should be able to do the following:
    - Enter auction name.
    - Set auction start/end date and time.
    - Enter the following property details: title, price, location, bedrooms, bathrooms, parking spaces, amenities,short description and an image.
    - Control to stop the auction.
  - Join an auction.
    - See available auctions to join.
- The Auction:
  - Once the server starts the auction the screen should update to show the details of the auction.
  - The highest bid and bidder should be displayed.
  - The user should be able to bid.
  - Users should be notified when there is a new highest bid.
- The users should be notified when the auction end who the highest bidder was.

Note: Bonus marks may be given to students who add extra functionality to them. A maximum of 15 bonus may be awarded (up to 7 for standard extra functionality but to achieve 15 you must add something extra-ordinary!) Bonus marks exceeding full marks will be capped at 15

Some ideas for bonus marks:

- If there are two bidders that bid against each other they enter a speed round, where there are set bid increments and these increments are quite high.
- A single user can have multiple ongoing auction and the ability to manage them.
- Send a email to the user who where the highest bidder at the end of the auction.