DBHost

a resource management system

by **Tibor Péter Szabó** for **DLBCSPJWD01** in **2023. Sept. 06**

The aim of the project was to create a webserver and webpage combination, that can provide access to data in a database from a browser window.

The project was written using Java, SQL, HTML, CSS, JavaScript, and Batch (for launching the webserver and the documentation quickly).

The software uses Java 8 and SQLite3. The SQLite3 is already provided with the files, but Java 8 must be downloaded for the application to function. To get started, just launch the Start.bat file from the main directory of the application and wait till the command line displays the message “Waiting for connections...”. You can access the webserver by typing “localhost” into the browsers search bar. From this you can start using the application with the provided example accounts in the Accounts.txt file.

The webserver was written purely in Java for simplicity’s sake as it can be run on any device with Java 8 installed. The server uses a server socket API from the standard java net library. It uses multi-threading to avoid bottlenecks when handling multiple users at the same time, to further reduce the load on the server, the communication system is based on REST architecture, to ensure that the server will not story any information about the users’ current state. To handle the SQL part of the project I used SQLite3 as it’s lightweight, and portable, suitable for the design.

The SQL database stores mainly data that is displayable on the webpage, but it also stored the data of the user accounts. The password is stored in a SHA256 encrypted format, to comply with the safety standards of account security and privacy. The users are also assigned a session ID that is changed every time they login. This reduces the amounts of data that is needed to be processed by the server as a form of authentication. An invention I’m proud of is using SQL database to store commands that can be ran on the same database. This provides a lightweight solution for adding, removing commands from the application. This can also server a way to revoke access to a command for multiple users at the same time, as the whole system is based on a role system, where commands can only be ran as certain roles.

On the front end, instead of using a CSS (Cascading Style Sheet) library like bootstrap.css or tailwind.css I opted for using pure CSS. JavaScript was extended with the CryptoJS library, so SHA256 encoding is available for authentication. HTML was not extended with any DOM manipulating libraries, but HTML injection was done using pure JavaScript.

Communication was done using HTTP (HyperText Transfer Protocol), as it is the standard of the internet communication. For additional information requests JavaScript XMLRequest was used. The HTTP header contained custom cookies for authentication during login, a session token was used after the user was logged in, so the user only has to login once per hour (to lower the risk of security break from unlocked computers with logged in users) and a special command token was used in the cookies to signal the current command for the server. This means that all the user information is stored in the headers, so the server can stay REST-like.

An example of how the system handles data parsing and responses given a login, a data query, a sorting, and a logout. After the application is running on the host machine, it can be connected to by using the IP of the computer, or if you would like to access the server from the host machine, you can use localhost.

1. Given a request coming to the root URL of the server (“/”) the server redirects the user to the login page (“/login”).
2. After a request was made to the login page (“/login”) the server reads the login.html file from the OS file system and sends it to the browser using a 200 OK response code.
3. The user can now attempt a login, which sends the username, and a encrypted version of the password to the server for authentication, to which the server responses with a personalised session cookie for future authentications and a URL for the next page. Upon the response, the browser requests the URL given in the previous response (“/”).
4. Now with a session cookie in the request HTTP header, the server can check if the SQL database contains a session at one of the users. If it does, the server sends a response with the main.html landing page. Before sending the page, a parser goes through the texts in the code, and replaces certain notations in the documents, like $name$ for the user’s full name.
5. To load the page, the browser gets the commands from the server (“/commands”). This makes the server to query all the commands and select the commands, to which the user should get access to given their access level. After this a list of commands are sent to the browser then formatted and inserted into the page.
6. On this page the user can now select a command that they would like to use. Upon selecting a command, the browser requests the table.html page from the server machine on the same (“/”) URL, and it’s served to the user. After this the browser sends a request to the command URL (“/command”) with the command id delivered trough a cookie in the header. This makes the server get the command from the SQL commands list, and it is executed on the database. The result is then formatted and served to the browser to be displayed in a table. The commands are queried once more in the same way, that was described in point 5.
7. The user now can read or print (using CTRL+P) the requested data. The user can also sort alphabetically in every column of the table. To reduce the amount of unwanted data, the user has the option to filter it. Using the magnifier glass icon, you can open the filter page. Here you can filter to every data in every column (or reset it using the provided button). The filtering is done completely on the front-end for two reasons: It limits the strain on the server, and nowadays all computers can filter bigger databases quickly. (Tested on a I3-3100K CPU with 4GB of RAM, 1000 entries took ~300ms to filter).
8. At this point the user can either continue browsing data using the commands menu (like point 6), or to log out after done using the program. Using the accounts menu, the user can easily find the big red logout button. After pressing the button, the browser calls the logout URL (/logout), which sends a reroute to the browser with the target being the login page (back to point 3.). On the back end, the server finds the user based on the provided session cookie and creates an expired session cookie and stores it as the current session in the database. This way it cannot be faked to attempt an unauthorised login.

Learned skills from the course:

I personally feel a huge improvement in both CSS and HTML structuring compared to the previous projects. I discovered most of HTTP, as I didn’t used it before this. I learned how to dispatch SQL queries from Java to a SQL engine. Because of the security measures I had to get comfortable with Base64 encrypting and decrypting and SHA256 encrypting.