WD Documentation

Moldovan Mihai

June 2023

1 Introduction

For my Web Design project I have chosen the task of creating a website of a medical clinic that contains information about the doctors and the administrator can update information.

The website will consist of a main landing page that will mostly serve as a placeholder, a login/signup page, a logout page and a page that contains information about the doctors employed at the clinic. The page that displays the doctors will display a grid of doctors with their name, speciality and photo to the normal user, but admins designated through manually changing a value in the MySQL database have the ability to add, remove and edit doctors.

I plan for the website to be responsive, easy to deploy and work on various device sizes, I will also try to make the website loading as fast as possible.

2 Technical specifications

To implement the above concepts I will use PHP (Apache) and MySQL from the XAMPP Control Panel, to store the data I will use a database titled "medical_clinic" with 2 tables "doctors" that has 4 columns "doctor_id", "name", "speciality" and the image data stored as bytes(base64 might have been better); and another table named "users" that stores all users and has 4 columns "id", "username", "password" which stores the hashed password and the "admin" flag which is a TINYINT representation of a boolean.

The site contains these pages:

- **index.php:**

  The main/landing page which is just a placeholder at the moment of writing this documentation consists of PHP code that will be present on all other pages to start a session "session_start();", then is followed by a header which contains the stylesheet "styles.css" and the page title, what follows is a header that displays the username of the logged in user if it exists, and a navbar that contains links to the various pages and a button that changes between Login and Logout.

- **doctors.php:**

  Starts with the code to start a session, followed by some variable declarations for the database connection and a placeholder image that is retrieved from a link, what follows is the connection to the database and a check. This is followed by some function declarations for manipulating doctor elements, the first 2 functions for adding and deleting a doctor are quite similar, both preparing a query and then running it on the database, the add function inserts a doctor with a name and a speciality and the delete function deletes a doctor by their id, both functions then run the query and provide a success or error message that is echoed.

  The third function is for editing an existing doctor and uses the execute function for running a query because it might be faster for uploading large strings of binary data, the function checks if an image to be uploaded is set and then creates a query for that condition, else runs a query without the image parameter.

  What follows is the retrieval of the list of doctors which is stored in a "result" variable and the database connection is closed, after this comes the header and navbar code that is also present on the main page. Then if the logged-in user is an admin an input menu that consists of a text field, a dropdown and a submit button is displayed after the grid of doctors is populated by iterating over the retrieved list from the database, if the doctor has set a profile picture it is displayed else it is replaced with a placeholder image. The actual doctor element is echoed after and consists of the image which is encoded as base64 after retrieval from db, the name and the speciality of the doctor, if the logged-in user is an admin the edit menu is then echoed.

  The edit menu is made up of a username text field, a speciality text field (might change to dropdown) and a file chooser for picking a profile image and 3 buttons, two for saving and cancelling the edit and one for deleting the doctor from the database.

  The showing/hiding of the edit menu and saving the edits are handled by some JavaScript code that consists of some selectors for the edit button, cancel button, edit menu and save button, then for each edit button click event listeners are added that hide the edit button and show the edit menu; for each cancel button click listeners are added that hide the edit menu and display the edit button; for each save button click listeners are added that submit the form.

- **login_register.php:**

  Starts with the code to start a session then checks if the action wanted is login or register and stores the action as a variable, then text field data is read and if the user wants to register a password match check is done and an error display if it fails. Following that is the connection to the database and a check if the user wants to log in, it succeeds the data

for the selected username is retrieved, and the password hash is verified against the one in the database if it is correct the user's data is stored in the session and the user is redirected to the main page, otherwise, if the password is incorrect or the user does not exist an error message is displayed. If the user wants to register the provided username is checked to be unique, otherwise, an error message is displayed, if the username is unique the provided password is hashed and the user is inserted into the database.

What follows is the header element and some PHP code that displays the error message stored in the session (if it exists, haven't figured out how to make the message persist only when it needs to), then the login and register forms are defined in HTML code with their corresponding fields, but both contain an element that toggles the form from login to register and vice versa without redirecting. This is done by having a link that has an onclick that runs the function "toggleForm" which displays the hidden form and hides the displayed one.

- **logout.php:**

  The function page that is run when the user clicks the logout button starts a session, checks if a user is actually logged in, then unsets all session vars and destroys the session.

3 Replicating the site and challenges encountered

To deploy the website and the database I used XAMPP, in the project's source I provided a SQL file that can be imported (rename to "medical_clinic" after import in myphpadmin panel) for a database populated with 50 entries and an admin account already setup up, to test modifying the doctors grid displayed on the website, one can use the account with the username "admin" and the password "admin". To host the website on one's local machine the folder containing the website should be placed in the "htdocs" folder created by XAMPP at installation.

A few challenges I encountered in making the website were first to do with XAMPP which corrupted my database after closing it, and required reinstalls multiple times until I managed to fix it, other challenging aspects were learning to understand PHP and fixing errors that occurred from unproperly initialized variables, wrong conditions in if statements and various other sources.

The login page presented a few challenges too, the main one being figuring out how to make the switch from the login to the register form and back work which took quite a bit. The doctor display grid was hard to get to look right, and it still has an error where every grid item on a row gets resized in one has its editing menu shown, also for a while I couldn't figure out why the grid took so long to load, but when I had only a few doctors with user-defined images, I

noticed it was because I was fetching the default image for each doctor with no picture set.

Setting up the database was comparably easier than the other aspects of the project since the myphpadmin interface was quite easy to use since I knew a bit about database queries from high school.