COMP1230 Assignment 2: Voting Application

Create and Participate in Voting

Course: COMP 1230
Assignment Weight: 7% of the final mark
Estimated Time: 20 hours
Due Date: October 28, 2025 @11:59

Objective:

In this assignment, you will build a **PHP-based voting application** that allows users to **register, log in, create topics, and vote**. This application will use **file handling**, **sessions**, and **cookies** to store and retrieve data dynamically. You will break down the application into **smaller, manageable tasks** using **functions** to implement each core feature. By the end of the assignment, you will have a complete voting system where users can engage by creating topics and participating in votes. 🗳️

Features:

The assignment is divided into multiple stages to guide you through building the application step-by-step. Read each part carefully and implement them in the given order**.**

Part 1: User Registration and Login (15 points)

Implement a **registration and login system** that allows users to create an account and log in. Users should be able to:

1. **Register**: Create an account with a unique username and password.
    - Store user data in a file named users.txt.
    - Use the following format: username:password (one user per line).
    - Send user to login page once regisertion is completed.
2. **Log In**: Authenticate registered users.
    - If the credentials match, start a **session** to keep track of the logged-in user.
    - Redirect to the **Dashboard** page after a successful login.
1. **Log Out**: Create a logout link that destroys the session and redirects to the login page.

Requirements:

- Use **files** to handle user data storage.
- **Do not hard-code** any usernames or passwords in the PHP code.
- Create a separate functions.php file to store helper functions like registerUser($username, $password) and authenticateUser($username, $password).

Hints:

- Make sure each username is unique by checking the users.txt file before registering.
- Implement proper **error messages** for failed login attempts.

Part 2: Topic Creation (20 points)

Once a user is logged in, they should be able to **create a new voting topic**. Implement a form on the **Dashboard** page where users can add a new topic with the following fields:

- **Topic Title**: The main title of the topic.
- **Description**: A short description explaining the topic.

Requirements:

1. Store each topic in a **file** named topics.txt with the following format:
    - topicID|creator|title|description
    - Each new topic should have a **unique ID** starting from 1and incrementing for each new topic. (check appendix for code sample)
2. Create a function createTopic($creator, $title, $description) to handle topic creation and save it in the topics.txt file.
3. After a topic is created, **redirect** the user to a **Topic List Page** where they can see all the topics and vote on them.

Hints:

- Use a **unique ID** for each topic to keep track of topics in the topics.txt file.
- Make use of necessary file functions to handle file operations.

Part 3: Voting on Topics (30 points)

Allow users to **vote up** or **down on any topic** on the **Topic List Page**. Each user can vote **only once** per topic. Implement this functionality using the following guidelines:

1. Store the votes in a **file** named votes.txt in the following format:
    - username|topicID|voteType
    - voteType should be either "up" or "down".

2. Create functions *vote($username, $topicID, $voteType)* and *hasVoted($username, $topicID)* to handle the voting process.
    - Check if the user has already voted on a topic using *hasVoted()*.
    - If the user hasn't voted, add the vote to votes.txt using the *vote()* function.
3. Display the **current vote count** for each topic (e.g., 5 Upvotes, 3 Downvotes) on the **Topic List Page**.
4. **Optional:** Show a **Leaderboard Page** that lists the top-voted topics based on upvotes.

Hints:

- Implement *getVoteResults($topicID)* to count a specific topic's total upvotes and downvotes.
- **Optional:** Handle file locking using flock() to prevent race conditions when multiple users are voting simultaneously.

Part 4: Viewing and Managing Voting History (20 points)

Each user should have a **Profile Page** where they can view their **voting history**. Implement a separate page that:

1. Lists all topics the user has voted on, along with their vote type (Up or Down).
2. Include a summary that shows:
    - Total number of topics they have created.
    - Total number of votes they have cast.

Requirements:

1. Create a function *getUserVotingHistory($username)* that reads from votes.txt and returns an array of topics the user has voted on.
2. Display the voting history on the **Profile Page**.

Hints:

- Implement getTotalTopicsCreated($username) and getTotalVotesCast($username) to retrieve total number of topics create by the user and total number of votes casted by the user.

Part 5: Using Cookies to Remember User Preferences (15 points)

Enhance the application by using **cookies** to remember user preferences:

1. Implement a **Theme Selector** where users can choose between **Light** and **Dark** themes.
2. Store the selected theme in a **cookie** and automatically apply the theme when the user returns.

Requirements:

- Implement *setTheme($theme)* and *getTheme()* functions to handle theme settings.
- Use the selected theme to apply different CSS styles dynamically.

OBJ

Required Functions

Required Functions

The following functions must be defined in functions.php and implemented exactly as described to ensure they pass the unit tests:

User Management Functions

1. **registerUser($username, $password)**:
    - **Description**:
        - Registers a new user by storing the username and password in the users.txt file.
    - **Parameters**:
        - **$username**: The username to register.
        - **$password**: The password for the user.
    - **Returns**: true if registration is successful, false if the username already exists.

2. **authenticateUser($username, $password)**:
    - **Description**: Authenticates a user by checking the users.txt file for matching credentials.
    - **Parameters**:
        - **$username:** The username to authenticate.
        - **$password**: The password for the user.
    - **Returns**: true if the username and password match, false otherwise.

Topic Management Functions

3. **createTopic($username, $title, $description)**:
    - **Description**: Creates a new topic and stores it in topics.txt.
    - **Parameters**:
        - **$username**: The creator of the topic.

        - **$title:** The title of the topic.
        - **$description**: A brief description of the topic.
    - **Returns**: true if the topic is created successfully, false otherwise.

4. **getTopics()**:
    - **Description**: Retrieves all topics stored in topics.txt.
    - **Parameters**: None.
    - **Returns**: An **array** of topics, where each topic is an associative array with keys: **topicID, creator, title, and description**.

Voting Functions

5. **vote($username, $topicID, $voteType)**:
    - **Description**: Casts a vote (up or down) for a topic.
    - **Parameters**:
        - $username: The username of the voter.
        - $topicID: The ID of the topic being voted on.
        - $voteType: The type of vote ("up" or "down").
    - **Returns**: true if the vote is successfully recorded, false if the user has already voted on this topic.

6. **hasVoted($username, $topicID)**:
    - **Description**: Checks if a user has already voted on a given topic.
    - **Parameters**:
        - $username: The username of the voter.
        - $topicID: The ID of the topic to check.
    - **Returns**: true if the user has already voted, false otherwise.

7. **getVoteResults($topicID)**:
    - **Description**: Retrieves the total number of upvotes and downvotes for a topic.
    - **Parameters**:
        - $topicID: The ID of the topic.
    - **Returns**: An **associative array** with two keys: up (number of upvotes) and down (number of downvotes).

Session and Cookie Management Functions

8. **setSession($key, $value)**:
    - **Description**: Sets a session variable.
    - **Parameters**:
        - $key: The session key.
        - $value: The value to store in the session.
    - **Returns**: true if the session is set successfully.

9. **getSession($key)**:
    - **Description**: Retrieves a session variable.
    - **Parameters**:
        - $key: The session key.
    - **Returns**: The value stored in the session, or null if the key does not exist.

10. **setCookie($key, $value)**:
    - **Description**: Sets a cookie with a given key and value.
    - **Parameters**:
        - $key: The cookie key.
        - $value: The value to store in the cookie.
    - **Returns**: true if the cookie is set successfully.

11. **getCookie($key)**:
    - **Description**: Retrieves a cookie value.
    - **Parameters**:
        - $key: The cookie key.
    - **Returns**: The value stored in the cookie, or null if the key does not exist.

Built-in Function You Must Use

12. Use show_source(): At the end of your script, use show_source(__FILE__) to display the source code.

Reflection:

- Your reflection, which includes your explanation of how you approached the problem, challenges you faced, and how you solved them.
- Should be added to the **AI Usage Declaration** (aud.pdf)
- **Important: This reflection should be in **your own words** and should accurately represent your learning process and thought approach. Avoid copying generic explanations. Points may be deducted if the reflection is unclear or incomplete.

Hints and Tips

1. **Use Functions for Each Feature**:
    - Implement separate functions for registration, authentication, topic creation, and voting.
2. **Test Your Code Frequently**:
    - Test each function separately using sample input files before integrating.
3. **Keep File Handling Simple**:

- Use built-in PHP functions like file, fopen, fwrite, and fgetcsv to read and write files.

4. **Comments**:
   - Comment your code to explain the purpose of each function.

Grading Breakdown

Component	Percent
User Registration and Login and Logout	15 %
Topic Creation	20 %
Voting Mechanism and Display	30 %
Voting History and Summary	20 %
Theme Selector with Cookies	15 %
Total	**100 %**

Submission Instructions:

1. The final grading will include running the entire application to test the integrated functionality. Ensure that each function works seamlessly with others.

2. **Upload Your PHP File**:
   - Upload your assignment PHP files to **GBLearn** under comp1230/assignments/assignment2.

3. Submit the **functions.php** to **my.gblearn.com** under comp1230, Assignment 2.

4. **Submit AI Usage Declaration**:
   - Download the **AI Usage Declaration** from D2L, fill it out with details on any AI tools you used, and include your reflection in the declaration.
   - Convert the completed declaration to a PDF (aud.pdf).
   - Upload **only** the aud.pdf to Assignment 1 on **D2L**. **Do not upload your PHP file** to D2L.

All core functionalities should be implemented using the following **standardized function names**, which will be used for automated testing. Do not create your own function names. Make sure function names match exactly to this document.

Tips to Avoid Plagiarism

- Break the assignment into multiple files and functions as described.
- Use meaningful variable names and comment your code thoroughly.
- Do not copy-paste code directly from online sources—focus on understanding and implementing the logic yourself.

Important Notes:

1. **AI Usage Declaration**: The declaration form requires you to detail any AI tools used, their purpose (e.g., coding assistance, idea generation), and how you refined the AI-generated content. This document must accompany your submission on D2L and include your reflection. **Failure to submit it will result in a zero for the assignment.**

2. **Similarity Check**: An automated tool will check All code submissions for similarity. If any identical code is found across submissions, it will result in a grade of zero for all involved students. Ensure that your work is original and reflects your effort.

Good luck, and if you have any questions, don't hesitate to ask!

Appendix:

Starter code:

The function below generates a unique id for one record. The unique id allows you to easily identify a record to modify or delete or create a relationship between records in different files.

```php
<?php
  function getID()
  {
    $file_name = 'ids';
    if(!file_exists($file_name))
    {
        touch($file_name);
        $handle = fopen($file_name,'r+');
        $id = 0;
    }
    else
    {
        $handle = fopen($file_name,'r+');
        $id = fread($handle,filesize($file_name));
        settype ($id,"integer");
    }
    rewind ($handle);
    fwrite($handle,++$id);

    fclose($handle);
    return $id;
  }
```

Unit Test Prepartion:

File Structure and Setup

Your project should follow the structure below. This structure will ensure that your code is compatible with the provided unit tests and will help organize your project.

/voting_app/ ├── **index.php** # Home/Login page

├── **register.php** # Registration logic

├── **login.php** # Authentication logic

├── **create_topic.php** # Topic creation page

├── **vote.php** # Voting page

├── **profile.php** # User profile and history page

├── **functions.php** # Core functions (to be submitted)

├── **users.txt** # User data

├── **topics.txt** # Topic data

├── **votes.txt** # Vote data

├── **/tests/** # Unit tests directory

    ├── **VotingAppTest.php** # PHPUnit test file

Suggested Content:

- user.txt
    - maziar:password
    - jane_smith:&*Adfds
- topics.txt
    - 1|John|test one|test one description
    - 2|maziar|PHP Programming|this is just some content about php programming
- votes.txt
    - 3|maziar|up
    - 4|albert|down

Integration Testing Guidelines

Testing your application ensures that your functions are correct individually and work together cohesively. This integration testing will help identify any missing links or errors in your workflow.

1. Register a New User
    a. Call the `registerUser()` function to create a new user, e.g., `registerUser('new_user', 'password123');`.

    b. Check the `users.txt` file to confirm that the new user is added.

2. Authenticate the User
    a. Use `authenticateUser('new_user', 'password123');` to log in.
    b. If successful, store the username in a session using `setSession('username', 'new_user');`.

3. Create a New Topic
    a. Call the `createTopic('new_user', 'Best Programming Language', 'Vote for the best programming language!');`.
    b. Check the `topics.txt` file to verify that the new topic is stored correctly.

4. Vote on the Topic
    a. Use the `vote('new_user', 0, 'up');` function to cast an upvote on the first topic (ID: 0).
    b. Check the `votes.txt` file to confirm that the vote is recorded.

5. View the Vote Results
    a. Retrieve the vote results using `getVoteResults(0);`.
    b. Verify that the function correctly returns the total number of upvotes and downvotes for the topic.

6. Logout and Login as a Different User
    a. Create a new user (`registerUser('other_user', 'pass456');`), log in as this user, and repeat the above steps to create a new topic and vote.

Application Demo

https://www.loom.com/share/c73c5eda8fc1449dbcf1228cfa7b441f?sid=04f82df0-09c6-4791-9ae2-8bb1c05d52bb