COS 498

Final Project: Less Wild West Forum

100pts

Due: Dec 19th, 11:59pm

1 Assignment Description

For this final project, you will transform your midterm "Wild West Forum" into a secure, production-ready web application. This project builds directly on your midterm implementation, adding database persistence, security features, user profiles, password recovery, real-time chat functionality, and enhanced comment features.

The application must use SQLite3 for data persistence, implement proper security measures (password hashing, account lockout, HTTPS), provide user profile management, support password recovery via email, include a real-time chat system using Socket.io, and implement pagination for comment history.

2 Core Requirements

2.1 Database Migration

- Migrate all data from in-memory storage to SQLite3 database
- Create database schema for:
  - **Users**: username, password (hashed), email, display_name, profile_customization fields, account lockout fields
  - **Sessions**: session data linked to users
  - **Comments**: author (linked to user), text, timestamps, etc
  - **Login Attempts**: username, IP address, timestamp, success/failure status
- Include database connection handling and error management

2.2 HTTPS with Nginx Proxy Manager

- Configure HTTPS using Nginx Proxy Manager (or similar reverse proxy)
- Set up SSL/TLS certificates (Let's Encrypt)
- Ensure all traffic is routed through HTTPS
- Update Docker configuration to work with the proxy manager
- Document the proxy configuration in your README

2.3 User Authentication and Security

2.3.1 Password Security

- Hash all passwords using a secure hashing algorithm (argon2)
- Never store plaintext passwords
- Implement password strength requirements (minimum length, complexity)
- Validate password requirements on registration and password changes

2.3.2 Account Lockout

- Track login attempts in the database
- Implement account lockout after a specified number of failed attempts (e.g., 5 failed attempts)
- Lock accounts for a specified duration (e.g., 15 minutes)
- Display appropriate error messages when accounts are locked
- Log all login attempts (successful and failed) with IP addresses and timestamps

2.3.3 User Fields

- Add **email** field to user accounts (required, must be unique)
- Add **display_name** field to user accounts (required, needs to be different from username)
- Display names should be shown in comments and user profiles instead of usernames. If a user changes their display name, all comments made by that user should be updated to show the new display name.
- Usernames remain for login purposes and should be unique.
- Validate email format on registration and updates.

2.4 User Profile Page

Create a user profile page accessible to logged-in users with the following features:

2.4.1 Account Management

- **Change Password**:
  - Require current password verification
  - Validate new password meets strength requirements
  - Hash new password before storing
  - Invalidate all existing sessions after password change (force re-login)
- **Change Email**:
  - Require current password verification
  - Validate new email format and uniqueness
  - Send confirmation email to new address (optional but recommended)
- **Change Display Name**:
  - Allow users to update their display name
  - Validate display name (length, allowed characters)
  - Update display name across all existing comments (or maintain historical display names)

2.4.2 Profile Customization

- Allow users to customize their public profile appearance (pick one or more):
  - Profile name color (choose from predefined colors or hex codes)
  - Profile icon/avatar (upload image or select from predefined icons)
  - Other customization options (theme preference, bio, etc.)
- Display customizations in comments and user profiles
- Store customization data in the database
- Validate and sanitize user inputs for customization

2.5 Password Recovery System

- Implement "Forgot Password" functionality:
  - Add a "Forgot Password" link on the login page
  - User enters their email address
  - Generate a secure, time-limited password reset token
  - Store reset token in database with expiration timestamp
  - Send password reset email containing a link with the token

  - Email should include instructions and expiration time

- Create password reset page:
  - Accept reset token via URL parameter
  - Validate token exists and hasn't expired
  - Allow user to enter new password
  - Validate new password meets requirements
  - Hash and store new password
  - Invalidate the reset token after use
  - Redirect to login page with success message
- Email Configuration:
  - Configure email service (Gmail)
  - Document email configuration in README

2.6 Real-Time Chat System

- Implement an active chat area using Socket.io:
  - Create a dedicated chat page or section
  - Use Socket.io for real-time bidirectional communication
  - Display chat messages in real-time to all connected users
  - Show user display names and profile customizations in chat
  - Include timestamps for chat messages
- API Backend for Chat:
  - Create REST API endpoints for chat functionality
  - Endpoints should handle:
    * Sending messages
    * Retrieving chat history
  - Store chat messages in the database
  - Authenticate API requests (verify user is logged in)

2.7 Enhanced Comment System

2.7.1 Pagination

- Implement page-based comment history:
  - Display a limited number of comments per page (e.g., 20-50 comments)
  - Use GET parameters to specify which page to display (e.g., `/comments?page=2`)
  - Include pagination controls (Previous, Next, page numbers)
  - Show total number of comments and current page information
  - Handle edge cases (invalid page numbers, empty pages)
- Efficient Database Queries:
  - Use LIMIT and OFFSET in SQL queries
  - Optimize queries to avoid loading all comments into memory
  - Display comments in reverse chronological order (newest first)

2.7.2 User-Friendly and Security Features

Implement at least **one** of the following features:

1. **Comment Compression/Truncation**:
   - Handle comments that are too large
   - Implement truncation with "Read More" functionality
   - Or implement client-side compression before sending
   - Set reasonable maximum comment length
2. **Markdown Support**:
   - Allow markdown formatting in comments
   - Parse and render markdown safely (prevent XSS)
   - Support common markdown features (bold, italic, links, code blocks)
   - Sanitize markdown output to prevent security vulnerabilities
3. **User Comment History**:
   - Allow clicking on a user's name/avatar to view their profile
   - Display all comments made by that user
   - Implement pagination for user comment history
   - Show user's profile customization on their profile page
4. **Edit and Delete Own Comments**:

    - Allow users to edit their own comments
    - Show edit history or "edited" indicator
    - Validate that users can only edit their own comments
    - Update comment timestamp or show edit timestamp

5. **Comment Reactions/Voting**:
    - Allow users to upvote/downvote comments
    - Store reactions in database
    - Display reaction counts
    - Prevent users from voting multiple times on same comment

6. **Comment Threading/Replies**:
    - Allow users to reply to specific comments
    - Display replies in a threaded format
    - Store parent-child relationships in database
    - Implement nested comment display

3 Testing Requirements

You must demonstrate that your system works correctly with:

- User registration with email and display name
- Login with account lockout after failed attempts
- Password hashing and verification
- Profile page functionality (change password, email, display name, customization)
- Password recovery via email
- Real-time chat functionality
- Comment pagination
- At least three additional comment features
- HTTPS connection
- Database persistence (data survives container restarts)

4 Code Quality Requirements

- Include comprehensive comments explaining security measures
- Organize code into logical modules/files
- Handle edge cases appropriately (expired tokens, locked accounts, etc.)
- Provide clear error messages for users
- Ensure code is readable and well-structured
- Use consistent coding style
- Implement proper error handling and logging

5 Documentation Requirements

You must provide:

- A comprehensive `README.md` file that includes:
  - Instructions on how to set up and run the application
  - Database schema documentation
  - Environment variables and configuration requirements
  - Nginx Proxy Manager setup instructions
  - Email service configuration
  - Security features implemented
  - API endpoint documentation (for chat API)
  - Known limitations or issues
- Inline code comments explaining complex security implementations
- Documentation of design decisions and trade-offs

6 Point Distribution

Component	**Points**
Database migration (SQLite3, all tables)	10pts
HTTPS with Nginx Proxy Manager	5pts
Password hashing and security	5pts
Account lockout system	5pts
User fields (email, display name)	5pts
User profile page (all features)	10pts
Password recovery system	10pts
Real-time chat (Socket.io + API)	15pts
Comment pagination	5pts
Additional feature (chosen by you)	15pts
Code quality and comments	5pts
Documentation (README.md)	10pts
Total	**100pts**

7 Submission Requirements

You must submit the following:

- Link to your public GitHub repository (must be the same repository from midterm, with new commits)
- Link to your hosted website (with HTTPS)
- A link to a video demonstrating:
  - User registration with email and display name
  - Login with account lockout demonstration
  - Profile page features (changing password, email, display name, customization)
  - Password recovery flow
  - Real-time chat functionality
  - Comment pagination
  - Your chosen feature
  - HTTPS connection verification
- Updated README.md with all required documentation

8 Additional Notes

- This project builds on your midterm, but make it it's own repository (create a branch off your midterm repository, and then merge that branch into your main branch when you're done).
- Start early! This is a substantial project requiring multiple complex features.
- Test security features thoroughly (try to break your own system).
- Consider user experience when implementing features.
- Make sure your database schema is well-designed before implementing features.
- Document your security decisions and trade-offs in your README.

9 Recommended Development Order

1. Set up HTTPS with Nginx Proxy Manager
2. Set up SQLite3 database
3. Implement password hashing and update authentication
4. Add email and display name fields
5. Implement account lockout system
6. Create user profile page with account management
7. Implement password recovery system
8. Implement real-time chat with Socket.io
9. Add comment pagination
10. Implement your chosen comment features
11. Security audit and testing
12. Documentation and final testing