# SCOPE DOCUMENT

TWSSNA

The World's Simplest Social Networking App!

**BY**

Jason Stoyles

Deer Park, NY 11729

**DATE**

1/15/2018

## Project Purpose

The purpose of this project is to develop a very simple web application that mimics other social networking applications like Twitter, in order to showcase some of the basic functional requirements, security best practices, and scalability options.

## High Level Requirements

This project will be built using a Linux, Apache, MySQL and PHP (LAMP) stack.
It will require at minimum Apache 2.4, MySQL 5.7, and PHP 7, as well as memcached to be installed.

## Frameworks Used

The HTML Purifier library will be used. See: http://htmlpurifier.org/

Development will be done using Navicat for MySQL and Sublime Text for coding.

## Database Schema

A complete database schema will be included with the project (see file db_schema.sql), but high level requirements include:

2 database tables - a users table for collecting and storing user information, and a comments table for collecting and storing user comments that will link back to user records in the users table.

In addition, store procedures will be used to interact with the database.

## Internet Traffic Concerns

This project will be developed under the assumption that there will be high levels of internet traffic to the publicly hosted application. In order to mitigate any long periods of down time, the application will be hosted on a scalable VPS like DigitalOcean or AWS. This will allow for near instant scalability when needed and the ability to scale down when not needed, for added cost savings. In addition, memcached will be utilized to cache data for faster load times and to limit the need to query the database for read-only actions. Stored procedures will also be used as a way to increase query speed for locally pre-defined query execution paths.

## Security Concerns

The OWASP Top 10 will be used as a baseline for addressing know security threats.
The latest top 10 can be found at: https://www.owasp.org/index.php/Top_10-2017_Top_10

Here's how these threats will be mitigated:

1) Injection
   a) Stored procedures will be used to force parametrization of all database queries.
   b) PDO prepared statements will be used as a top-level safeguard to help sanitize SQL before it reaches the server.
   c) Avoidance of concatenation will be used for all queries to further prevent injection attacks.
2) Broken Authentication - This will not be addressed based on the limited use case for this project. However, if it were to be addressed we would incorporate the following:
   a) Dual authentication for access to sensitive information or change user data.
   b) Force session timeouts after a period of time (E.g. 5 minutes).
   c) Prevent passing record ID's via URL's – This has been implemented through the use of obfuscated GUID's that still require a session in order to be used when making database changes.
3) Sensitive Data Exposure
   a) No sensitive data aside form password will ever be stored.
   b) HTTPS will be used for every HTTP request
   c) All password will be double salted using SHA512 encryption
   d) Since user generated content is allowed, it is possible to accept and store sensitive information unknowingly. To prevent this we can implement a verification system to validate all submissions before allowing them to be served. This will not be implemented for this version of the application based on the limited use case.
4) XML External Entities (XXE)
   a) The use of XML will not be part of this application, so this item is not a high concern.
5) Broken Access Control
   a) Every user will have the same level of access, so this item is not a high concern.
   b) However, users should only be allowed to delete comments they have themselves submitted. In order to enforce this, every delete action must include the users session object userID (which will never be exposed), and that userID must match the userID of the comment being deleted via a stored procedure call.
6) Security Misconfiguration
   a) Based on the limited use case for this application, only the following will be addressed:
      1. Browser error reporting will be disabled to prevent errors possibly showing sensitive information.
      2. The latest versions of software should be used.
7) Cross-Site Scripting (XSS)
   a) The HTML Purifier library will be used as a trusted framework to prevent data that can be stored as part of a possible XSS attack.
8) Insecure Deserialization
   a) Serialization will not be used as part of this project.
   b) Cookies are also not used, so there is no sensitive information that can be read on

the client side as part of any type of object data.

9) Using Components with Known Vulnerabilities
   a) The application is very limited in scope and so no know software and/or software components containing know vulnerabilities have been used.
10) Insufficient Logging & Monitoring
   a) Based on the limited use case for this application, only basic out of the box logging will be used (e.g. php, mysql, and web server logs).
   b) In a more robust scenario we would implement higher levels of monitoring and alerting based on OWASP recommendations.

## Additional Notes

A working copy of this application can be found at https://twssna.jstoyles.com/

This hosted version includes:

1) A TLS certificate generated provided by letsencrypt.org
2) Memcached for caching latest comments (expires and refreshes every 10 seconds)

It is hosted on a digitalocean.com VPS server running:
Linux (Ubuntu 16.04), Apache 2.4.18, PHP 7, and MySQL 5.7.2

## Things I Might Do Differently Given More Time/Resources

1) Host the php config file outside of the application root directory.
2) Incorporate a templating system like smarty for better layer separation.
3) Incorporate a middle tier layer that controls database calls and business logic (if no framework is ever used).
4) Separate the database server from the web server for better scalability.
5) Run more robust tests against the hosted application using SQL Map for injection testing (http://sqlmap.org/), and a load testing tool like Loader.io (https://loader.io/).
6) Use the Laravel PHP framework. This was not used due to my limited knowledge. The learning curve would require at least a few weeks and I wanted to get this done sooner.