**PROJECT 1 REPORT**

SY306

1151

14APR19

BRANDON SHIELDS

JUAN SANTANA

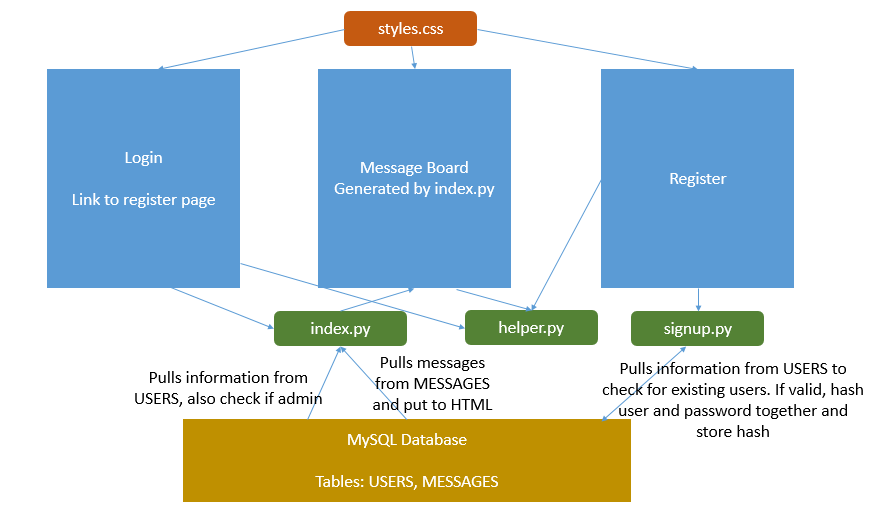
BEN VALDES

YIXIN YE

**PROJECT DESCRIPTION**

The team met to discuss the initial design of the website. Github was employed to aid in collaboration as we anticipated difficulties in sharing and monitoring progress. MIDN Ye’s directory will be used to host the message board, accessible through <http://midn.cs.usna.edu/~m207026/project01/>.

The approach to the message board was altered along the way as we discover new requirements from more careful reading. Figure 1 shows our the topology of our website and how different files interact with one another.



*Figure 1. Overall Topology*

For testing purposes, the below are a few accounts that could be used.

USERS

U: root P: yeyeyeye1 (Admin)

U: Juan P: password1

U: Ben P: Password123

U: Test P: passwordhash

To divide up the workload, MIDN Ye and MIDN Shields would work on the front-end of the message board, dealing with HTML, CSS and CGI. Additionally, MIDN Ye would work on figuring out the MySQL database. MIDN Santana and MIDN Valdes would work on the python scripts which would interact with the MySQL database. Although this distribution was not followed through in the end, it was useful in helping us get started with the project.

Before we begin discussing the mechanisms behind our website, Table 1 shows a list of all files required to run the website, sorted by the way they were used.

|  |  |  |  |
| --- | --- | --- | --- |
| **Scripts** | **Pages** | **Aesthetics** | **Database** |
| index.py  logout.py  helper.py | login.html  signup.html | styles.css  bubble.ico | config.py  tableCreate.sql |

*Table 1. List of files*

Index.py formed the backbone of the website, serving as the landing page for all users, whether they are new users or returning ones. If index.py does not detect a valid cookie, the user will be shown login.html, where there is the option to sign up for a new account in a separate page or to log in to an existing one.

SIGN UP

signup.html

1. Performs checks on username and password inputs using Javascript
2. Password requirement is at least a number and at least 6 characters.
3. Uses POST method to submit registration information to signup.py

signup.py

1. Catches information sent by POST method from signup.html
2. Performs server-side input validation in case Javascript was bypassed.
3. Check for duplicate usernames, print out unsuccessful sign-up if any tests failed.
4. If nothing is wrong. Username and Password hash of sha256(password+user) will be stored into SQL database. UserID will be given automatically by mySQL database through AUTO\_INCREMENT.
5. Parameterized query was used to protect against SQL injection.
6. No admin users can be registered through this method
7. Scripts from helper.py were used to generate the HTML page.

LOG IN

login.html

1. Provides link to signup.html
2. Takes user inputs through HTML forms and submits information to index.py using the POST method

index.py

1. Takes user input from login.html and authenticates with the MYSQL database
2. Parameterized query was used to protect against SQL injection.
3. The MYSQL database is accessed using credentials stored in config.py
4. If not authenticated, user will be sent back to the login page with a generic error message (so as not to compromise security)
5. Index.py determines UserID and if User is an admin
6. Index.py generates table of messages stored in the database
7. Using UserID, it determines which messages could be deleted by the user
8. Index.py uses scripts from helper.py to do hashing and printing out HTML

POSTING/DELETION

index.py

1. Index.py uses a HTML forms to handle posting and deletion of messages
2. Index.py serves as the action file in these forms. It is written to catch these requests to post messages or to delete them
3. UserID is provided in both posting and deletion so that index.py can provide user information when posting and also determine if the message can be deleted by the user.

**WAY AHEAD**

Our focus in Project 1 was primarily on the functionality of the message board. Minimal measures were taken to secure the website. Basic hashing and salting was done to provide better authentication and security given that we are dealing with cookies for authentication. Parameterized queries were implemented to defend against SQL injection. An input class was also utilize to escape characters which do not sit well with HTML.

What can be done to improve security is to make use of privilege management available in mySQL. Instead of using m207026, the owner of the database, to perform every function, we can use another account with limited privileges given by GRANT. We could also implement Content Security Policy so that only white-listed scripts could be run on the message board.

**TIMELINE**

29Mar19

Created github repository <https://github.com/privat33r/project01.git>

Created index.html, styles.css

Accessible at <http://midn.cs.usna.edu/~m207026/project01/>

5Apr19

Made sure every member could access Github repository

Database created with example entries put in

Started work on login.py, message.py, signup.py

12Apr19

Identified need for post.py

13Apr19

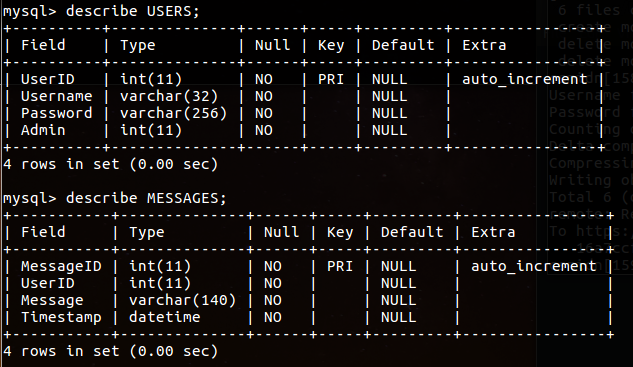
Decided to use index.py as landing page as well as page for generating message board, posting messages and other functionalities.

Most functionalities completed

14Apr19

Testing and report writing.

**MYSQL DATABASE**



SQL Database creds:

User: m207026

Password: mysqlpassword

Command to access database: mysql -h csmidn -u m207026 -p

SQL script:

CREATE TABLE USERS1 (

UserID int NOT NULL AUTO\_INCREMENT PRIMARY KEY,

Username varchar(32) NOT NULL,

Password varchar(256) NOT NULL,

Admin int DEFAULT 0 NOT NULL

);

CREATE TABLE MESSAGES1 (

MessageID int NOT NULL AUTO\_INCREMENT PRIMARY KEY,

UserID int NOT NULL,

Message varchar(140) NOT NULL,

Timestamp DATETIME NOT NULL

);

INSERT INTO USERS1 (Username,Password,Admin) VALUES("root","9c03c5265606e5da39ca2dcc5982d1be88ec2749904860cf37948bb6ee3fe501",1);