EXGAMES

**Members:**

Zhi Wei He Zheng Victor Gong

Suman Dutta Seung Hoon Lee

**Getting Started:**

Users are required to register an account in order to browse other pages. Once a user logs in, the user can then view other posts on the main page or search for specific posts. If buyers are interested in a posting, they can send a message to seller or write a comment under the post. After seller receive the message, they can arrange trade detail.

[insert “heroku stuff if needed to setup our app” ]

**Design Decisions:**

We followed the MVC designs strictly while implementing our application, and we chose Express as our main framework to best generate the basic MVC structure.

Model: We chose mongoDb with Mongoose modeling because mongoose provides simple data schemas with built-in type casting and validation which greatly simplifies form submission between pages.

Views: We chose the EJS template as our frontend because it generates HTML content with simple embedded JavaScript code. With EJS we could display any data and information using code within the views themselves.

Controller: We used Express as our core framework , and all major function such as register, create post , login authentication, etc. are located here.

**Missing features:**

Admin login: While we do have an admin page with the ability to view all postings, all users, and modify user account information (username and password), we were unable to make sure that only admins could log in. Admins are part of the user database and as such users are able to log into the admin page as well. We set up the site so that the admin login page can only be accessed by typing in the url, not with a link on the main page. Ideally the way this works is that only admins can log into the admin page, and any user who tries will be redirected to the user login page instead.

**Security:**

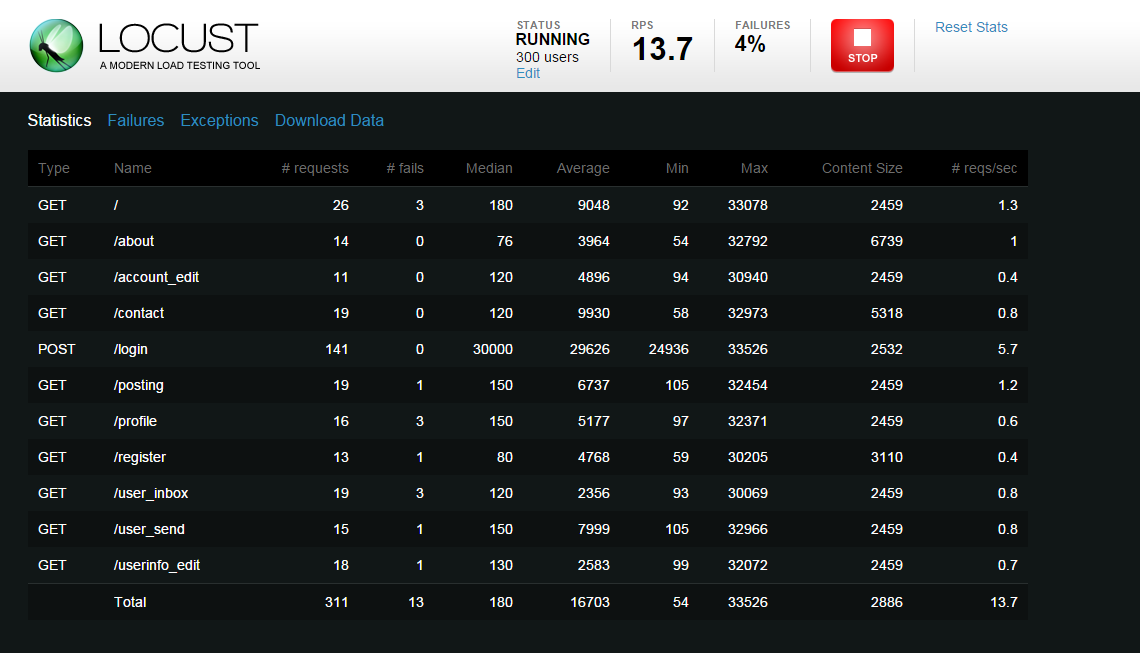
We used three measures to enhance the security of the app. First, passwords are hashed and then saved in the database.

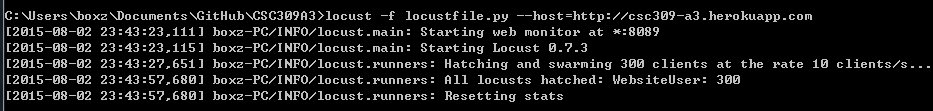
Second, we used express-validator to sanitize user input in forms for protection against XSS.

Third, we used express-csurf to guard against XSRF attacks. Csurf puts a token in the response when we make a request. The server verifies the existence of this token in the next request and gives the permission to take actions such as submitting a form. In an attempt to test the security during the mocha testing, pages with the csruf protection could not be accessed without passing the csruf token. We had to manually get the token from ‘get’ and pass it to ‘post’ to get around the security protection.

**Performance:**

Locust.io was used to test our web performances with a standard of 300users with spawn rate at 10user/second. Ideally, we will implement indices in the mongoose database to increase performance. This will prevent mongoose from scanning every document in a collection on user query requests, if indices are implemented properly, it can limit the number of document it must inspect and thus increase efficiency.





**Testing:**

We used various testing frameworks such as mocha, supertest, and chai. Testing is divided into two groups, primarily differentiated by whether or not the module has csruf protection or not. For pages without it, we checked the status code of the response from the server. For pages with csruf protection, a post request was nested inside a get request. We extracted the csruf token in the cookie from the response for get request and put it in the post request and checked whether the redirection was performed after form submission. The testing is done under test/test.js.

Video Link: <https://www.youtube.com/watch?v=td37zdEqifg> (it’s unlisted)

Github link: <https://github.com/g3hezhi/CSC309A3>

Heroku link: https://csc309-a3.herokuapp.com/login