SE 3316A Web Technologies Lab #4: Due Sunday, Dec. 3, 11:55 pm

Deadlines:

1. Submission deadline: Sunday, Dec. 3, 11:55 pm. Submission closes on Wed. Dec 6, 11:55 pm (2 points/day late penalty).
2. Demonstration deadline: Wednesday, Dec 15, 5pm.

Evaluation

1. Marks are awarded for evidence of credible actions to finish by the due date. Such actions may be measured by functionality completed each week (as indicated by Git commits) as well as submission by due date.
2. Late submission penalty is 2 points per day.
3. Demonstrations must be completed before Wednesday Dec. 15th.
4. A test plan with detailed tests will be provided. Any test item that is checked must pass the test without any conditions. During the demonstration, you may request part marks by providing a written request that states what test items fail and the impact that will have on the functionality of that item. E.g. If test items 3.f.i, 3.f.ii and 3.f.iv passes, but 3.f.iii fails, what impact would that have on functionality of 3.f?

Revision History

1. Added Svelte to approved front-end frameworks, Nov 15, 5am.

Objectives:

A. Apply knowledge of server-side and client-side scripting and modern web application frameworks to create a complex web application.
B. Expose major functionality of the application via a ReSTfull web API.
C. Develop a client application using a front-end framework with the above API.
D. Implement an authentication protocol and provide different levels of functionality to authenticated vs. unauthenticated users.
E. Implement a client application that works on both mobile as well as a variety of desktop browsers and presents a responsive user interface.
F. Develop the application in a way that is resistant to malicious exploitation.
G. Create a security and privacy policy that is publicly accessible.
H. Create a DMCA notice & takedown policy that is publicly accessible.
I. Provide a DMCA takedown procedure and tools for the site administrator:
    a. Log requests, notices, and disputes.
    b. Send a takedown notice for any DMCA requests received and disable display of alleged copyright violations.

You must use a front-end framework (see the list of accepted frameworks at the end) for front-end and Node.js for back-end implementation. You may use other libraries and services to implement authentication. You must disclose all source code that you copy from any source if that code is contained in your submission on Owl. This disclosure must be in the form of comments above copied code and include a link to the source as well as a summary of what parts are copied and/or any prompts for AI tools. This disclosure does not apply to any libraries that you use (e.g. in "node_modules" directory or imported from an external source).

Submission Instructions:

Please carefully read the instructions and strictly follow them. Your grade depends on it. Submissions of items 3 and 4 on Owl (Git log and zip file) are the ones that determine submission time.

1. Ensure that your repository is named "se3316-xxx-lab4"(all lowercase) where "xxx" is your Western email ID (without @uwo.ca part).

2. Use a proper "`.gitignore`" file so that only the files that you edit are in your repository.
3. Copy the output of command "`git log`" and paste that onto the submission page (Assignments section) on Owl.
4. Download your repository as a zip file from Github and submit it on Owl as an attachment. Please do not zip the folder on your computer as it may contain a large amount of extraneous files. Ensure that libraries, data files, intermediate files or backup folders are not included in the zip file.
5. Submit a completed test plan on Owl in PDF format including the last commit ID, commit date/time and total points. If the completed and signed test plan was not submitted before the demonstration, you will be asked to reschedule the demo.
6. Submit your repository to Github Classroom (See detailed instructions).
7. Demonstrate your lab (on a public URL) before the demonstration deadline. You may set up the public URL after submission on Owl. As long as the changes only involve deployment issues, it is acceptable.

Penalties will apply for not following the naming convention or any of the submission steps

Description

Develop an enhanced version of the web application developed in lab 3. Additional features include the ability to create an account, save and edit hero lists for logged in users, ability to make these lists publicly accessible as well as the ability to add comments/ratings to individual heroes as well as to the list, admin functionality to manage accounts and copyright related tasks.

Use of Node.js and a front-end framework (see list below) is required. Other technologies may include Mongodb and Express or any alternatives that suit you better.

In case of any ambiguities, conflicting or unclear requirements, you are free to make a choice that is (a) sensible from a UI/UX point of view, (b) minimises your effort and (c) in keeping with the spirit of the application. It is strongly advised to discuss such assumptions on Discord if you are not sure. You are also required to document all such decisions using git comments.

This document may be revised to improve readability or to remove ambiguous, conflicting or unclear requirements. Please pay attention to the revision statement at the top of this document.

Design Tip

Keep the number of concepts you keep in your head simultaneously to about 5 so that you are not overwhelmed by the complexity. E.g. Keep the number of objects/concepts/modules that you have to grapple at one time to 5. At the highest level, you may want to imagine about 5 components that make up the full application. If these components as well as interactions among them work as intended, then your application should work as intended. You may call that the architecture of your application. Then think about the design of each component and apply the same principle.

Complexity of an application is not just the number of components, but also the number of interactions between these components. A system with 5 components where each component interacts with every other component is more complex than a system with 10 components where each component only interacts with two other components although they both have the same number of interactions.

Requirements and Rubric

Items marked with 🌈 indicates features that are relatively easier.

1. Show evidence of planning and credible actions to finish by the due date. Such actions are measured by functionality completed each week (as indicated by Git commits) as well as submission by due date. {**Up to 10 of points at the discretion of the instructor**}
2. Authentication method: {**total 10 points**}
   a. Provide a local authentication mechanism (create an account with an email, a password and a nickname, update password). {**6 points**}
   b. Input validation for email (proper format). {**1 points**} 🌈

    c. Verification of email (see References). {**2 points**}
    d. If the account is flagged as disabled, show a message asking to contact the site administrator and not allow logging in. {**1 point**}

3. Limited functionality for unauthenticated users. {**total 25 points**}
    a. Start page showing application name, a short “about” blurb that says what the site offers, and a login mechanism. {**2 points**} 🌈
    b. Interface for searching heroes matching any combination of name, race, power or publisher. For multiple patterns, results must match all fields. Empty fields must match all values and each search string must match the beginning of the given field. Search results must show the name and publisher at a glance. {**8 points**}
    c. Expand each search result to view all additional information of each hero. {**2 points**} 🌈
    d. A “Search on DDG” button for each result which launches a DuckDuckGo search on a new browser tab for the selected hero. {**2 points**} 🌈
    e. Search terms are soft-matched (e.g ignore white-space, minor spelling variations or mistakes). {**4 points**}
    f. List of public hero lists (up to 10) ordered by last modified date and showing name, creator’s nickname, number of heroes in the list and average rating. {**3 points**} 🌈
    g. Ability to expand each hero list to show the description and the list of heroes each showing the name, power and publisher. {**1 point**} 🌈
    h. Ability to display additional information for each hero in a public list. {**3 points**}
4. Additional functionality for authenticated users: {**total 24 points**}
    a. Create and show up to 20 named lists of heroes. Each list must have a unique name (required), a description (optional), a collection of heroes (required) and a visibility flag of public or private with the default set to private. {**10 points**}
    b. Clicking on a list shows the description of the list and name and publisher of all the heroes in the list. {**2 points**}
    c. Edit all aspects of an existing list and record the last edited time. {**4 points**} 🌈
    d. Delete a list. {**2 points**} 🌈
    e. Add a review with a rating (required) and a comment (optional) to any public list. {**6 points**}
5. Administrator functionality related to site maintenance: {**total 10 points**}
    a. Special user with administrator access. {**4 points**}
    b. Ability to grant site manager privilege to one or more existing users: {**2 points**} 🌈
    c. Ability to mark a review as hidden or clear the “hidden” flag if set: {**2 points**} 🌈
    d. Ability to mark a user as “disabled” or clear the “disabled” flag if set: {**2 points**} 🌈
6. Web service API: {**total 10 points**}
    a. Revise the API developed for lab 3 as necessary to provide required functionality. {**6 points**}
    b. Build your application using this API. {**4 points**} 🌈
7. Administrator functionality related to copyright enforcement: {**total 11 points**}
    a. Create a security and privacy policy that is publicly accessible. {**2 points**} 🌈
    b. Create an “acceptable use policy” (AUP) that is publicly accessible. {**2 points**} 🌈
    c. Create a DMCA notice & takedown policy that is publicly accessible. {**2 points**} 🌈
    d. Provide a DMCA takedown procedure and tools for the administrator: {**total 5 points**}
        i. Document to describe the workflow and usage of tools. {**1 point**} 🌈
        ii. For each review, a mechanism to create a DMCA entry to log requests, notices, and disputes. E.g. Set-up a log entry in a database with a unique “request ID” for entering “date request received” (date), “date notice sent” (date), “date dispute received” (date), “notes” (text) and “Status” (Active/Processed) for each review and provide an interface to set these properties. {**2 points**}
        iii. Tools to hide reviews with alleged copyright or AUP violations. {**1 point**}
        iv. Tools to restore displaying of any contested reviews. {**1 point**}
8. Usability, code quality and other non-functional requirements:

a. Insufficient input sanitization including not limiting range (where applicable), length and content (where applicable) as well as interpreting any user input as HTML, CSS or JavaScript, to safeguard against injection attacks. {**up to -10 points**}
b. Not using JWT for any authorization task. {**up to -10 points**}
c. Not able to handle any user input in any language. {**up to -5 points**}
d. Usability issues of the application on multiple browsers and form factors. {**up to -5 points**}
e. Lack of modular code that is easily extensible and maintainable. {**up to -15 points**}
   i. E.g. Changes to operational parameters such as server names, port numbers etc should not cause changes in code.
f. Presence of code duplication. {**up to -5 points**}
   i. E.g. Full URLs that are duplicated in calls to ReST api, copy/paste of code blocks that are mostly similar etc.
g. Presence of hard-coded literals in code. {**up to -5 points**}
   i. E.g. Hard-coded port numbers, URLs
h. Lack of sufficient and meaningful comments in code. {**up to -5 points**}
i. Not exercising proper precautions in saving user information. {**up to -10 points**}
   i. Use of sufficiently strong hash algorithms such as BCrypt or Argon2, use of a salt.

j. Code management with Git {**up to -10 points**}
   i. Less than 20 commits that are meaningful, no meaningful commit messages
   ii. Not using a proper `.gitignore` file to ignore external dependencies,
   iii. Not using Git pull to deploy code to the server.

k. Logistics {**up to -20 points**}
   i. Repository name not in required format,
   ii. No zip file (not graded)
   iii. Code is not attached as a zip file or it contains content that is not in the Git repository or is missing content that is in the repository
   iv. Test plan is inaccurate or not in PDF format (not graded).
   v. No show for a signed up demo: -20

References

1. Authentication library: http://www.passportjs.org/ or https://auth0.com/
2. Email verification (see FAQ): https://www.npmjs.com/package/email-verification
3. Salted Password Hashing - Doing it Right: https://crackstation.net/hashing-security.htm
4. Responsive design using Angular - https://material.angular.io/
5. DMCA Demystified: http://www.sfwa.org/2013/03/the-dmca-takedown-notice-demystified/
6. GitHub DMCA policy: https://help.github.com/articles/dmca-takedown-policy/
7. Angular Security; https://angular.io/guide/security

Resources

1. Allowed front-end frameworks are: Angular, React, Vue, Svelte.
2. Firebase is a good option which provides authentication and database: https://firebase.google.com. Firebase UI, SDK or equivalent library/framework is acceptable for implementing a login mechanism. Note: Firebase should only be used to provide data storage and/or authentication services. All other back-end functionality must be implemented on your own server.
3. JWT is the required method for implementing authorization and protected routes:
   a. Main site: https://jwt.io
   b. Good tutorial: https://github.com/dwyl/learn-json-web-tokens
4. Copyright enforcement functionality: See slides 16-19 of "social issues" unit.

FAQ

1. Questions about item 6 (copyright enforcement): Please look at the test plan. You may devise and implement any mechanism to satisfy the tests. As long as it is part of the application and satisfies the

test, it will be accepted. No automation is needed. E.g. Assume that the site administrator may receive copyright/abuse claims via email. The administrator only needs to be able to log such requests via the application.

2. Email verification (2.c) requires the ability to send emails out, which is getting harder due to stricter anti-spam controls implemented by major service providers. Because of this, you may implement a mock-up of the verification step as follows:
    a. User enters an email address to create a new account.
    b. An email is crafted that includes a unique link (eg. generated from a hash of user information) that the user is required to click in order to verify the email address.
    c. Instead of sending this email to the address that the user provided, it is shown on the client.
    d. When a user visits the unique link (click or copy+paste to another browser window), the address is verified.
3. Soft-matching strings using the Dice coefficient: https://www.npmjs.com/package/string-similarity
4. You may use any database.
5. You may redesign the back-end API as necessary.
6. A mechanism to register users is required. See 2.a in requirements.

Suggested Workflow

Following is a suggested workflow to plan the implementation of this lab. Some design suggestions are provided only for information and you are not required to follow them.

1. Read the Requirements section several times until you develop a mental picture of what the application does.
2. Clone your repository for lab 3 with the new name for lab 4 (se3316-xxx-lab4). Create the project for the front-end in the “client” folder.
3. Design a Database structure for the selected database.
4. Revise the API designed in lab 3 to provide basic functionality. It is a good idea to designate separate API prefixes for non-authenticated, regular user and admin categories. E.g. All paths in “/api/secure/” require authentication as a regular user. Paths in “/api/admin/” require admin privileges. Paths in “/api/open/” do not require authentication.
5. Review steps 1-3 and re-examine each design decision to verify that it provides the foundation for your current understanding of the requirements.
6. Implement access control logic first for /api/secure and /api/admin.
    a. Implement all the routes first. Don’t need to implement actual functionality. E.g. log message on console to verify that it receives the request.
    b. Implement access control for one route in /api/secure. Each request must present a JWT and this token must be verified by the back-end functionality. See slides for “router.use()” for implementing such common functionality.
    c. You may start with a pre-generated JWT and use a ReST client (E.g. Insomnia) to test the functionality.
    d. Implement access control in all remaining routes and ensure that code for access control is not duplicated.
7. Implement back-end functionality for a particular feature or a group of connected features and test it with a ReST client.
8. Implement front-end Angular components that use the back-end functionality developed above.
9. Keep dependencies between modules to a simple tree-like structure.