Computer Science and Engineering Department, University of Nevada, Reno

Lecture Goggles

Team #22, Logan Long, Nathan Yocum, Zachary Johnson
Dr. Sergiu Dascalu, Devrin Lee
Dr. Shamik Sengupta
March 8, 2019

Table of Contents

Abstract

Lecture Goggles is a free, open-source, educational resource repository to help students gain a better understanding of school subjects. Lecture Goggles is meant to help resource sharing between students and professors. The implementation is focused on a web application design using a back-end web server and database to quickly manipulate and store uploaded resources and user accounts. Lecture Goggles utilizes a modular UI design to provide growth with little developer interaction, with permissions required to access certain web server routes.

Project Updates and Changes

Updates

Lecture Goggles is currently working towards having integration between the API server and the frontend. Account creation and authentication is working. The frontend is currently being written to store the JSON Web Token for authentication and to use React's Context API to maintain global state between all pages on the site. The abstract has been updated to better reflect the UI design approach.

Changes

Team 22, after some discussion, decided to switch from using Django for the API server of Lecture Goggles to using Flask. Since Team 22 decided to change technologies, they have made the following update to the previous document:

Engineering Standards and/or Technologies

~~1. Django (Technology)~~
   - ~~Django's REST framework is used for our web-based API. Zachary is creating a full API for use by the web server as well as 3rd party developers. It resides on a virtualized server in Azure, and is the only way edits can be made to the database.~~

1. Flask (Technology)
   - Flask RESTful API is use for our web based API. Zachary is creating a full API for use by the web server. It resides on a virtualized server in Azure, and is the only way edits can be made to the database. This adds a level of security and dynamic functionality to the website.

User Stories and Acceptance Criteria

Frontend

Scenario: As a new user of Lecture Goggles, I want to create a new account so that I can access features of the site restricted to authenticated users.

AC1. Landing page and navigation bar have hyperlink to account creation page.
AC2. Account creation page includes a blank form that can be filled out and submitted to create a new account
AC3. After submitting form, an email is sent to the email used on the form containing an email verification link.
AC4. After verification of email, user is authenticated and navigation bar and landing page update

Scenario: As a returning user of Lecture Goggles who is not signed in, I want to be able to sign in to my account so that I can access features of the site restricted to authenticated users.

AC1. Landing page and navigation bar have a hyperlink to sign in page
AC2. Account management and uploading pages redirect to sign in page
AC3. Sign In page includes a blank form with email and password that can be filled out and submitted.
AC4. After authentication, account management and uploading pages no longer redirect to sign in page. Resources can be voted on and reported.
AC5. After authentication, landing page and navigation bar update

Scenario: As an authenticated user of Lecture Goggles, I want to be able to upload a new resource belonging to a pre-existing subject and topic so that I can share a helpful resource I found.

AC1. Upload page includes a tab called ‘Resources’ with a form with inputs for URL, title, description, and select menu for subject and topic.
AC2. Subject and topic selects populate with pre-existing subjects and topics
AC3. Selecting a new subject and topic uploads the resource to the specified subject and topic
AC4. The uploaded resource is able to be viewed under the corresponding resource and topic.

Scenario: As an authenticated user of Lecture Goggles, I want to be able to add a new subject and topic so I can add new resources for a new topic and subject.

AC1. Upload page includes a tab called ‘Subjects’ with a form including inputs for subject name and description.

AC2. Upon form submission, subject appears in ‘Upload Resource’, ‘Upload Topic’, and ‘Resources’ page.

AC3. The subject appears in the subjects page. A report button and a button to navigate to the subject’s topics are present.

Scenario: As an authenticated user of Lecture Goggles, I want to be able to view resources which I can upvote, downvote, or report so I can help other users find helpful content.

AC1. While authenticated, resource page displays resource page with an upvote, downvote, and options button.

a. Pressing upvote cases an API call and renders the corresponding increased point total. Resource appears in upvoted state, with a different colored upvote button and point total.
b. Pressing downvote causes an API call and renders the corresponding decreased pont total. Resource appears in a downvoted state, with a different colored downvote button and point total.
c. Pressing option button causes a menu to appear with options to report and share. Share causes a modal with the resource URL to appear, and report opens the report dialogue.

AC2. Resources can be sorted by upload date and point total.

AC3. Resources can be filtered by subject and topic.

a. Before filter is applied, only subject is enabled. After selecting subject, topic is enabled. Changing subject automatically removes any topic selected.
b. Filter causes only resources for a certain topic to be rendered.

AC4. Resources can be searched by title, url, and author.

API

Scenario: As a new user of Lecture Goggles, I want to be able to create a account so that I can access features of the site restricted to authenticated users.

AC1. On submit of data, email is unique, password is of sufficient length and complexity, and user data is added to database.

AC2. On model added to database correctly, return json success message to front-end user.

Scenario: As a returning user of Lecture Goggles, I want to be able to sign in to my account so that I can access features of the site restricted to authenticated users.

AC1. On submit of registered user email and password, API responds with success 200 to front-end user.

AC2. With verification of browser submitted data with database, return json success message with user access-token to front-end user.

Scenario: As a logged in user of Lecture Goggles, I want to create a new subject on the website.

AC1. On submit of subject creation, check subject name to determine if duplicate subject already exists in database and submitted user access-token for logged in user.

AC2. On model added to database correctly, return json success message to front-end user.

Database

Scenario: As a logged in user of Lecture Goggles, I want to be able to see what posts I've made

AC1. On request of data, return all posts where user is the original author.

AC2. On successful retrieval is posts, send info to user.

AC3. On successful send, return json success message.

Scenario: As a returning user of Lecture Goggles, I want to be able to delete a post.

AC1. On request of deletion, verify the user can delete the resource.

AC2. Upon successful verification, remove the resource.

AC3. Upon removing the resource, return json success message

Scenario: As a logged in user of Lecture Goggles, I want to change my password.

AC1. On request of change, verify the user may change their password.

AC2. On verification, verify the old password hash.

AC3. If the old password hash verifies, replace the old hash with the new hash.

AC4. Upon replacing the hash, return json success message.

Testing Workflow

Happy Path Workflows

Upload Resource Path

1. Sign in to account and navigate to '/upload'.
2. Fill out form, adding a valid URL, title, and description. Select appropriate subject and topic.
3. Submit form. Confirmation message is displayed once API call is finished.
4. Resource appears on user's account with options to delete resource and edit description but not title.
5. Resource appears on resources page. The post will be upvoted by the uploader by default, and points will update on page refresh.

Validated by a combination of unit tests, integration tests, end to end tests (performed by hand for now), and Postman API tests. Frontend test number 3 tests this path (and the corresponding unhappy path).

Browse and Filter Resource Path

1. View resources without a filter applied, showing new resources of subscribed topics first, then new resources of unsubscribed topics after.
2. Posts can be filtered to contain only resources from a certain subject.
3. Posts can be filtered to contain only resources from a certain topic once a subject has been selected.
4. Changing the subject causes the topic to be cleared.
5. Resources can be filtered based on title, author, and url.

Validated by a combination of unit tests, integration tests, end to end tests (performed by hand for now), and Postman API tests. Frontend test number 5 outlines this test path.

Creating an Account

1. Navigate to Lecture Goggles website

2. Select create account in upper right hand side of screen
3. Fill out form which contains first name, last name, username, email, password, confirmation password, and attending university.
4. Select the submit button below the signup form.
5. Receive verification email to complete process of creating account

Validated by a combination of unit tests, integration tests, end to end tests (performed by hand for now), and Postman API tests. Frontend test number 1 and API test 1 outline this path.

Unhappy Path Workflow

Unsuccessful Account Creation Path

1. Form is not dirty.
    a. Submit button is disabled. Editing the HTML to un-disable the button to submit yields an internal server error for which an error message is rendered in the form error area of the page.
2. Missing fields are marked as required.
    a. Any missing fields disable the submit button and an internal server error occurs if the form is submitted with errors.
3. Invalid inputs (such as invalid name characters and invalid URLs) are marked as invalid.
    a. Any invalid inputs cause an error to render with a message of what is wrong and disables the button.
    b. Submitting the form by enabling the button causes an internal server error, which is indicated in an error message. The server parses the input in a way that does not allow XSS, SQL Injection, and other issues highlighted in the OWASP Top 10.
4. Email and confirm email inputs do not match or password and confirm password inputs do not match
    a. An error is rendered and the submit button is disabled.
    b. Submitting the form anyways causes an internal server error.

Validated by Postman tests, penetration/security tests, unit tests, and end-to-end tests.

Unsuccessful Subject Creation

1. Subject already exists.
    a. On submission of new subject, the database is checked to determine if a subject has already been submitted. If a subject is already submitted user will be redirected to said subject.
2. Invalid inputs
    a. If the user filling out the form incorrectly, this would be missing input document fields, the user will be unable to create the subject, yielding an internal server error.
3. Invalid or missing token submission
    a. If the user is not signed in, the user will be redirected to sign in page. Form will not be rendered.

Validated by Postman tests, penetration tests, unit tests, and end-to-end tests.

Unsuccessful Password Change

1. Form is not dirty
    a. “Change Password” button is disabled. Editing the HTML to un-disable the button to submit yields an internal server error which is rendered in the form error area of the page.
2. Old and new password hashes match
    a. An internal error is rendered and the user is informed that their password cannot be the same as their old one.
3. Minimum requirements for password complexity are not met
    a. A message indicating an internal server error is rendered and the user is informed that their password does not meet the complexity requirements and was not changed.
4. User’s old password is entered incorrectly.
    a. A message indicating an internal error is rendered and the user is informed that their old password was entered incorrectly.
    b. The user is informed their password was not changed.

Validated by Postman tests, penetration tests, unit tests, and end-to-end tests.

Testing Strategy

Frontend

The frontend is being developed by Nathan following TDD, so automated unit tests, written in Jest, are written by Nathan prior to developing a new feature. It will be the responsibility of Zachary and Logan to ensure the automated tests make sense and that they pass. The automated tests are written during development and will be integrated into a continuous integration pipeline when the tests are written and verified. The continuous integration will ensure that no breaking features can be pushed to the project without a documented change.

The frontend is written to mostly use integration tests with some unit tests where they make sense. Integration tests are primarily being used because integration tests “strike a great balance on the trade-offs between confidence and speed/expense.” (Kent C Dodds, https://kentcdodds.com/blog/write-tests). Team 22 has static testing set up in the form of ESLint/Prettier following Airbnb’s Javascript and React style guide, and React’s PropTypes library to perform type checking props used on the frontend. Team 22 may write end-to-end tests using either Cypress or a wrapper for Selenium to implement acceptance tests automatically, but that will be a stretch goal, and the end-to-end tests will likely be performed by hand. The acceptance tests will be carried out by Zachary and Logan.

Link to spreadsheet for better readability:
https://docs.google.com/spreadsheets/d/1LoyrUbfnS0itow8sCMrHO4RlFN6Prjse3cek_WNRBNQ/edit?usp=sharing

Frontend & Frontend Integration Testing Plan

Test No.	Test Type	Test Name	Purpose of Test	Test Data	Expected Result	Actual Result	Outcome and Actions Required
1	end-to-end unit integration	Account Creation	It allows me to create an account	Date: March 6th, 2019 env: Chrome 72.0, jest os: macOS	- /newAccount renders form - form renders errors when incorrect - Form submission sends an email to registered email - After email verification users can sign in to their account	- as expected - as expected - no email sent or required - user can sign in without email verification	Email verification needs to be implemented
2	end-to-end unit integration	Sign In	It allows me to sign in to an account	Date: March 6th, 2019 env: Chrome 72.0, jest os: macOS	- /signIn renders form - form renders error when invalid form submitted - form renders error when invalid email used - submitting correct form causes landing page, upload page, and account page to change	- as expected - no error rendered on submit - as expected - as expected	Error on submission needs to be implemented
3	end-to-end integration unit	Upload Resource	It allows me to upload a resource	Date: March 11th, 2019 env: Chrome 72.0, jest os: macOS	- /upload includes a tab called resource. when active, upload resource form rendered - errors rendered prior to submit - errors rendered after submission - after valid submission, resource exists in resource page and account page	TBD	Implement upload feature functionality

4	end-to-end integration unit	Upload Subject Upload Topic	It allows me to upload a new subject or topic	Date: March 12th, 2019 env: Chrome 72.0, jest os: macOS	- /upload includes a tab called subject. when active, upload subject form rendered - errors rendered prior to submit - errors rendered after submission - after valid submission, subject exists in resource page - /upload includes a tab called topic. when active, upload topic form rendered - errors rendered prior to submit - errors rendered after submission - after valid submission, topic exists in resource page	TBD	Implement submit feature functionality
5	end-to-end integration unit	View, filter, sort, and search resources	It allows me to view different resources	Initial Test Date: March 13th, 2019 Test Date: March 28th, 2019 env: Chrome 72.0, jest os: macOS	- Resources button when authenticated renders upvote, downvote, and option button - Not rendered when not authenticated - Pressing upvote, downvote, and options cause the corresponding actions and API calls - Resources can be sorted by pont total and by upload date - Resources can be filtered by subject and topic - Topic not enabled	TBD	Implement voting and sorting functionality

					until subejct chosen - Changing subject clears topic		

Table 1: Frontend and frontend integration testing plan.

API Server

The API server will be tested using Postman to test and validate the API. Tests will be written using Postman's BDD tools, which are very similar to how Jest and Mocha both format their tests. Since Zachary has not been following TDD as closely during the development of the API, Nathan and Logan will be writing the tests for Postman as they test the features related to the API server.

API Testing Plan

Test No.	**Test Type**	**Target API Request**	**Test Name**	**Purpose of Test**	**Test Data**	**Expected Result**	**Actual Result**	**Outcome and Actions Required**
1	Functional Testing	/users/signup	User account creation	Test that user creation detects duplicate user creation and returns success/fail 200/400 response.	First Name, Last Name, Username, Email, Password, School	Json response with success or fail error code	(JSON) 'Success' : True, 200 (JSON) 'Success' : False, 'Description': Duplicate account, 400	On success: User account is created and saved into database. On failure: New API required after user changes data.

2	Functional Testing	/users/login	User account login	Test that accounts that have been registered can be logged into and returns success/fail 200/400 response.	Email, Password	Json response with success or fail error code	(JSON) 'Success': True, 'Access-Token': (result), 200 (JSON) 'Success': False, 'Description': Invalid account credentials, 400	On success: user Access-Token is returned for browser side to save. This token is used to access user only features. On failure: New API required after user changes data or creates account.
3	Functional Testing	/subject/create	New subject creation	Test to see if new subjects can be created with json return of success/fail 200/400.	Subject, Description	Json response with success or fail error code	(JSON) 'Success': True, 200 (JSON) 'Success': False, 'Description': Duplicate subject, 'Redirect': Directory to subject, 400	On success: New subject is added to database for website to display. On failure: Redirect user to duplicate subject.

Table 2: Test plan for the API server.

Database Server

The database server will be indirectly tested through the testing of the front end and API servers. The database itself is using PostgreSQL, which is extensively tested on its own. We will know that the database is storing the correct data and the correct

commands are being issued when the API and Front End respond properly. It is neither practical nor realistic to test the database completely by itself.

Contribution of Team Members

Zachary Johnson did the API User Story, API Test Driver, assisted in unhappy/happy workflow, and assisted in Test Driver. Zachary worked on the project for approximately 5 hours.

Logan Long did the Database User Story, and assisted with the Testing strategies, and happy/unhappy workflow. Logan worked on the project for approximately 4 hours.

Nathan Yocum worked on the Project Updates and Changes, updated the abstract, worked on the Frontend User Stories and Acceptance Criteria, the Upload Resource and Browse and Filter Resource Happy Paths, Unsuccessful Account Creation Unhappy Path, writing unit tests and integration tests for the frontend, the frontend testing plan criteria, and performed tests on the API server using Postman. Nathan worked on the project for around 13 hours, excluding writing the integration and unit tests for the frontend, which he has been doing since the start of the semester.

All three members were involved with checking each other's work and removing errors.