Assignment 3–Open Source Software: Analysis and Design

Thomas Orth

VM Name: csc415-server34.hpc.tcnj.edu

Full path to project on VM: /home/student1/Open-Learning-Platform

Full path to repository: https://github.com/TomOrth/Open-Learning-Platform

Statechart/State diagram: Please see separate PDF - "Assignment 3 - Statechart.pdf"

SSD (System Sequence Diagrams): Please see separate PDF - "Assignment 3 - SSD.pdf"

Design Class Diagram: Please see separate PDF - "Assignment 3 - Design Class.pdf"

NOTE: Some classes do not have any methods or parameters listed. This is because they were auto-generated by devise in order to handle conflicting routes. I did not override their

NOTE: Getters and setters are implicitly defined for ActiveRecord models. This means they are called under the hood when access properties. An example is:
puts lesson_plan.title -> this would call the getter implicitly
lesson_plan.title = "Title" -> this would call the setter implicitly

The design class uses the "getMethod()" and "setMethod(value)" notation in order to show that there are getters and setters in the class but the actual syntax in the code is so close to the variable name that I did not want to have confusion.

Detailed Use Case Descriptions

Use Case: Register
Primary Actor: Educator, Admin
Goal in context: To allow relevant users to login to the system
Precondition: The system is configured to allow users to register
Triggers: An educator or admin needs to be added to the system

Scenario:
1. User enters the site
2. User clicks on the relevant button on the home page to login
3. User clicks on the "register" button

4. User fills out the relevant credentials

Exceptions:
1. Users provide the same password as another; They need to provide different credentials
2. The page is not responding/not working; user cannot register

Use Case: Login
Primary Actor: Educator, Admin
Goal in context: To allow relevant users to login to the system
Precondition: The system is configured to allow users to login
Triggers: An educator or admin needs access to the system

Scenario:
1. User enters the site
2. User clicks on the relevant button on the home page to login
3. User logs in using their email and password

Exceptions:
1. Users provide the wrong password; They need to attempt to login again
2. Users are not registered in the system; They need to register then
3. The page is not responding/not working; user cannot login

Use Case: UploadVerificationPaperwork
Primary Actor: Educator
Goal in context: To allow educators to upload their paperwork for verification
Precondition: The system is configured to allow educators to upload the paperwork, and the educator is successfully logged in
Triggers: An educator needs to upload paperwork for verification

Scenario:
1. Educator goes to their profile
2. Educator clicks on the verification tab
3. Educator clicks to upload their paperwork
4. Paperwork is successfully uploaded

Exceptions:
1. The system is not configured properly; Educators cannot upload their paperwork
2. The server has no space; There is no room for the paperwork to be uploaded

Use Case: UpdateLessonPlan
Primary Actor: Educator
Goal in context: To allow educators to update lesson plans
Precondition: The system is configured to allow educators to update lesson plans and the educator is verified by an admin
Triggers: An educator needs to update a lesson plan

Scenario:

1. Educator goes to their profile
2. Clicks on the "Plans" tab
3. Clicks a lesson plan via the "Edit" link
4. On new page, updates the relevant information about the lesson plan
5. Click submit

Exceptions:

1. The system is not configured properly; The Educator cannot make lesson plans
2. The educator is not verified; They cannot do anything until they are verified

Use Case: DeleteLessonPlan
Primary Actor: Educator
Goal in context: To allow educators to delete lesson plans
Precondition: The system is configured to allow educators to delete lesson plans and the educator is verified by an admin
Triggers: An educator needs to update a lesson plan

Scenario:

1. Educator goes to their profile
2. Clicks on the "Plans" tab
3. Clicks a lesson plan via the "Edit" link
4. On new page, updates the relevant information about the lesson plan
5. Click submit

Exceptions:

1. The system is not configured properly; The Educator cannot delete lesson plans
2. The educator has no lesson plans; There are no lesson plans to delete
3. The educator is not verified; They cannot do anything until they are verified

Use Case: TagLessonPlans
Primary Actor: Educator

Goal in context: To allow educators to tag lesson plans with a topic
Precondition: The system is configured to allow educators to tag lesson plans and the educator is verified by an admin
Triggers: An educator needs to tag a lesson plan

Scenario:

1. Educator goes to their profile
2. Clicks on the “Plans” tab
3. Clicks a lesson plan via the “Edit” link
4. On new page, the educator chooses the relevant topic for the lesson plan
5. Click submit

Exceptions:

1. The system is not configured properly; The Educator cannot make tag lesson plans
2. The educator is not verified; They cannot do anything until they are verified

Use Case: SearchLessonPlans
Primary Actor: Educator
Goal in context: To allow educators to search for lesson plans they ned
Precondition: The system is configured to allow educators to search and the educator is verified by an admin
Triggers: An educator needs to find a lesson plan

Scenario:

1. Educator goes to the home page
2. Clicks on the search bar
3. Educator enters a search query for a title, and can limit based on the category
4. Clicks Submit/Search
5. Relevant lesson plans now appear

Exceptions:

1. The system is not configured properly; The Educator cannot search
2. The educator is not verified; They cannot do anything until they are verified

Use Case: IssueDMCATakedown
Primary Actor: Educator
Goal in context: To allow educators to issue takedowns of copyrighted material
Precondition: The system is configured to allow educators to issue takedowns and the educator is verified by an admin

Triggers: An educator needs a lesson plan removed

Scenario:

1. Educator views a lesson plan
2. Clicks “Issue takedown”
3. A DMCA takedown has now been submitted

Exceptions:

1. The system is not configured properly; The Educator cannot issue takedowns
2. There are no lesson plans yet in the system; A takedown cannot be executed
3. The educator is not verified; They cannot do anything until they are verified

Use Case: VerifyEducator
Primary Actor: Admin
Goal in context: To allow admins to verify educators
Precondition: The system is configured to allow admins to verify educators
Triggers: An admin needs to verify an educator

Scenario:

1. Admin goes to profile
2. Clicks on “Verification” tab
3. Can download the paperwork of the educators
4. Clicks “Verify” next to an educator to mark them as verified

Exceptions:

1. The system is not configured properly; The admin cannot verify educators

Use Case: Approve DMCA takedowns
Primary Actor: Admin
Goal in context: To allow admins to approve DMCA takedowns
Precondition: The system is configured to allow admins to approve takedowns
Triggers: An admin wants to approve a takedown

Scenario:

1. Admin goes to profile
2. Clicks “DMCA” tab
3. Views the relevant information for a takedown
4. Clicks “Approve” to approve a takedown
5. The lesson plan is deleted and the relevant educators are emailed

Exceptions:

1. The system is not configured properly; The admin cannot approve takedowns

UI Mockups: Please see separate PDF - "Assignment 3 - UI Mockups.pdf"

Description on how 8 principles of good UI design are met:

1. Strive for consistency

   Throughout the application, a consistent style is provided through the use of bootstrap, a front end design framework. Another aspect that shows consistency is the use of partials. These are pieces of UI elements that can be reused. This is shown with the navbar that is displayed on every page.
2. Enable frequent users to use shortcuts

   The navbar at the top of the page provides shortcuts to relevant pages on the web page. On each page where a lesson plan is, there are shortcut links for common actions such as "Show", "Edit", and "Delete" where appropriate
3. Offer informative Feedback

   Flash messages appear when actions occur, such as updating the database. Alongside this, there is error handling for the login and register page
4. Design dialogs to yield closure

   These flash messages that appear in the web page clearly show the results of the action in order to give the user closure for any action they partake in
5. Offer simple error handling

   Simple error handling for this application includes the form handling to ensure that the correct data is inputted, as well as simple permission restriction to ensure that users can only access pages they are allowed to.
6. Permit easy reversal of actions

   On pages that ask for updates and creation such as creating lesson plans, provide an option to cancel this action and return to the previous page
7. Support internal locus of control.

   Users have free range to perform actions. There are very little barriers to the actions in this application
8. Reduce short-term memory load

   Relevant information is displayed on each page to ensure the user is provided the proper info and not overloaded on each page change.

Test case design

The approach I will take to unit and integration tests will be to perform tests via test driven development. This would involve writing test cases to run pieces of code individually. Unit tests would test individual portions of the code (i.e. a model) while the integration tests written would run multiple module interaction (i.e. login) within the rails application. For system testing, I would use an automated testing software that would run against the running server in order to test the whole system.

The tool I would use for unit testing and integration testing would be minitest, a library designed for testing in ruby (https://github.com/blowmage/minitest-rails, https://github.com/seattlerb/minitest). I chose this tool because it is a tool I have used before and I really like it. For system testing, I would use Selenium in order to perform automated system testing and run through the different pages quickly. Also, this would allow for stress testing by being able to run automated testers quickly.
Test cases based off of the functionality implemented in Assignment 3:

Functionality Tested	**Inputs**	**Expected Output**	**Actual Output**
Educator Registration	Email: test@test.com Password: testing123 First name: John Last Name: Doe	Success message displayed, redirection to profile	
Admin Registration	Email: admin@admin.com Password: testing456 First name: Jane Last Name: Doe	Success message displayed, redirection to profile	
Password error handling for registration of educator (same for admin, just use proper input info)	Email: real@test.com Password: testing123 First name: John Last Name: Doe	Error message, saying that the password is already used	
Educator Login (be sure to be logged out)	Email: test@test.com Password: testing123	Success message displayed, redirection to profile	
Admin Login (be sure to be logged out)	Email: admin@admin.com Password: testing456	Success message displayed, redirection to profile	

Invalid credentials for login of educator (same for admin, juse use proper input info)	Email: test@test.com Password: wrong123	Error message, saying invalid email or password	
Upload verification paperwork for educator	A PDF file	Success message that paperwork was submitted	
Prevent educator actions until verified	Click on “plans” tab on educator profile	A display should be shown, saying that they cannot do anything until the educator is verified	
Admin verifies Educator	“Verification” tab of the admin profile, click verify next to the educator with the name “John Doe”	A success message that the educator is now verified should be shown	
Educator is can now do actions	Under the educator profile, go to the “plans” tab	There should now be a “New lesson plan” button	
Educator creates a lesson plan	Click the “New lesson plan button” Title: Test plan Description: I am meant to help educate Content: 2 pdfs	A success message displayed, redirect to profile, a new lesson plan should be displayed with the name “Test Plan”	
Educator updates a lesson plan	Click on “Edit” next to the lesson plan. Change the title to “New Test Plan”, the description to “I am now a different plan” and two different files	A success message should be displayed, redirection to profile, and now there should be a lesson plan in the list “New Test Plan”	
Educator can tag lesson plan	Click “Edit” next to the lesson plan on educator profile. Choose “Science” for	Now the lesson plan as the “Science” category associated with it	

	the topic		
Educator deletes a lesson plan	Click “Destroy” or “Delete” next to the lesson plan “New Test Plan”	Success message displayed, the lesson plan should no longer show up	
Educator searches for lesson plans	Please re-run the “Educator creates lesson plan” test case Home page Input for search: “New”, topic: “All”, click submit/search	On that page should be a list containing the lesson plan “New Test Plan”	
Educator search returns no lesson plans	Home page Input for search: “Z” Click submit/search	On that page, a message in the blank area should say “No lesson plans found”	
Educator can issue DMCA takedown	Follow test cases to create a new educator and a new lesson plan associated with that educator. Use different credentials. Log back into the John Doe account. Use the search bar to find the lesson plan of educator 2. Click to view it. Click “Issue takedown”	A success message should appear, saying the request was submitted.	
Admin can approve DMCA takedown	Admin goes to the profile, clicks the “DMCA” tab. They see the recently submitted takedown. Clicks “approve”	Success message displayed, both educators receive an email.	

How requirements were met for software engineering concepts described below:

- **Modularity and encapsulation to facilitate information hiding and reuse**

  In order to achieve modularity and encapsulation, the different components of the MVC architecture are represented by classes. There are controller classes for the different pages, models for the different data points. Views do not have classes but are modularized through the concept of partial components (UI components that can be reused). This will allow for reusability of code throughout the application. For information hiding, ruby has the ability for private methods to be created for the controllers. So helper methods can be made along with private data points for the controller internally. The routes, represented by the public methods in the controller, will represent the data needed for use in the other modules of the application. An example of information hiding with private methods can be seen in LessonPlanController. For models, ActiveRecord creates getters and setters by default. This allows for ease in information hiding.
- **Elegance and efficiency of algorithms**

  For all algorithms, only the necessary lesson plans are considered. An example to show what I mean by this is the plan tab of the educator. On this page we would only want to show lesson plans that are related to the educator. This is handled with a where clause. This limits the amount of data that is held in memory, making my algorithms more efficient. Another aspect of efficiency is the search algorithm on the home page. We do not want to keep all lesson plans in memory in a list then filter, that would be crazy inefficient. So a like clause is used on the title of the lesson plan for search. This only considers the lesson plans that are relevant to the search.
- **Appropriateness of data structures used for the problem**

  For the data structures used, there are two I used. The first are the postgreSQL tables. These tables encapsulate the correct properties needed for each data point in the application. The advantage of these tables is that relations can be defined. This allows for lesson plans to be associated with the correct educator in order to ensure only proper educators can edit these plans. The second one is a list (aka the built-in array structure in ruby). The list is appropriate to use because after applying the SQL query to retrieve relevant entries, I need to access all elements in the list. This works because since the list does not contain all lesson plans in the database, it is only iterating over a subset of the total database, allowing for more scalability.