Sprint 4 Retrospective Review Report

1. Links

- Pivotal Tracker
- GitHub Repository
- Heroku Deployment

2. Dates of the Sprint

- **Start Date:** 3/16/2025
- **End Date:** 3/28/2025

3. Team Contribution

Name	**Role**	**Points Completed**
Leo Gonzalez	Developer (Dev)	3
Owen Schultz	Developer (Dev)	5
Olivia Lee	Developer (Dev)	4
Casey Kung	Developer (Dev)	3
Sam Lightfoot	Scrum Master (SM)	4
Ethan Nguyen	Product Owner (PO)	4

4. Sprint Goal

Add relevant homepage content like questions completed for students and quick guides for instructors, have randomly generated numbers for the questions, implement advanced equation logic for complex questions including array operations, square root, exponentiation, etc, and implement the multiple choice questions.

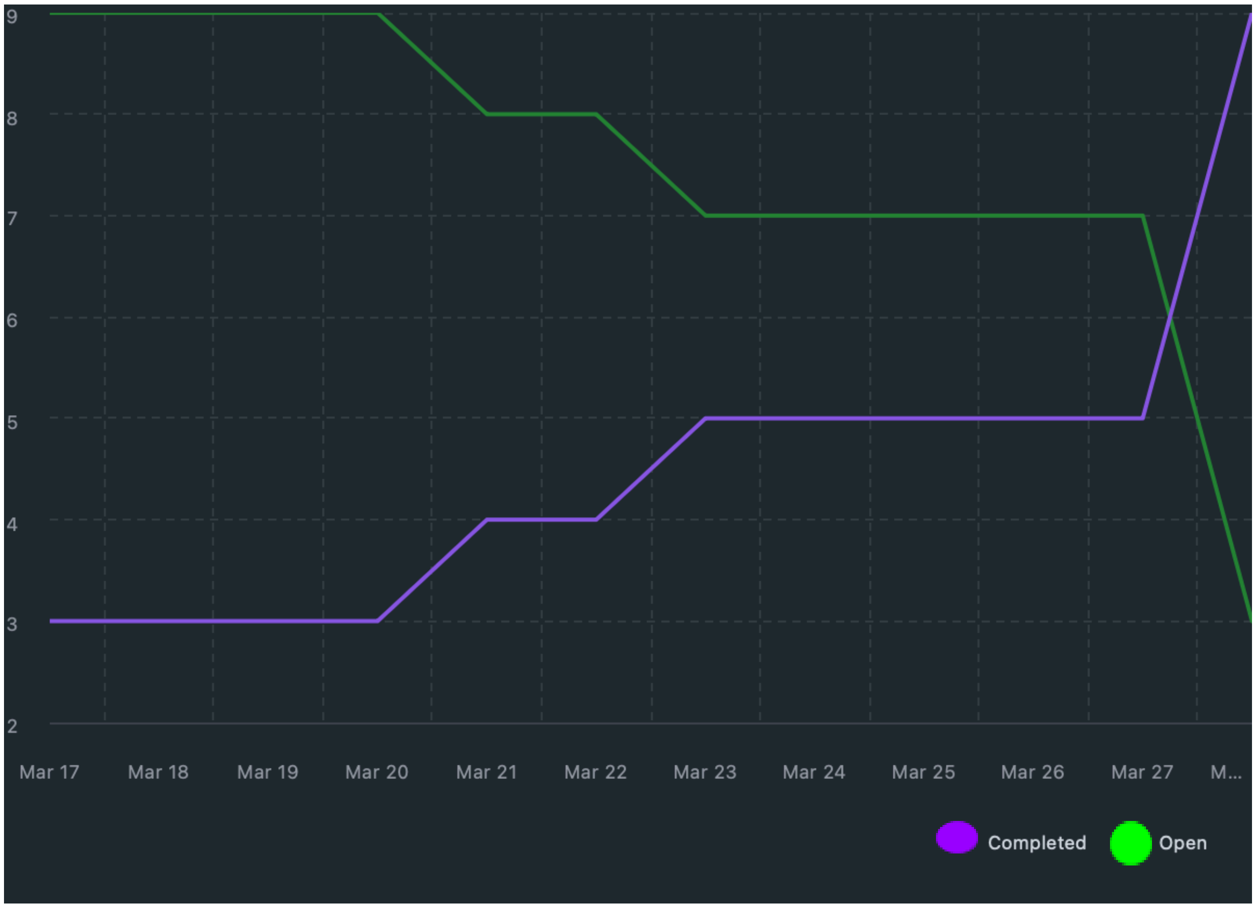
5. Sprint Achievements

In this sprint we were able to meet our sprint goal with functionality of all the user stories while maintaining consistent styling guidelines on all pages. Students can now see a dashboard that informs them of their most recent work on the practice questions. Instructors can now add multiple choice questions that can be seen by students, and we have the advanced equation logic that allows for instructors to add questions with array operations, square roots, exponents, and functions to the questions database.

6. Sprint Backlog Items and Status

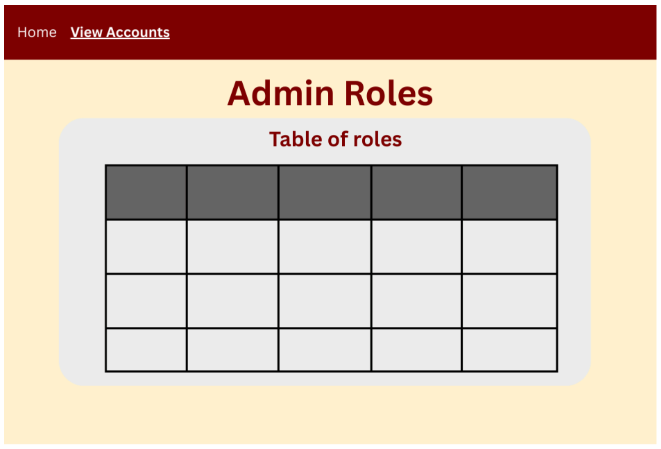
Backlog Item	**Status**
Admin Nav Bar	Completed
Home Page Dashboard	Completed
Rounded Answers	Completed
Randomly Generated Numbers	Completed
Multiple Choice Questions	Completed
Complex Equations	Completed

7. Burndown Chart

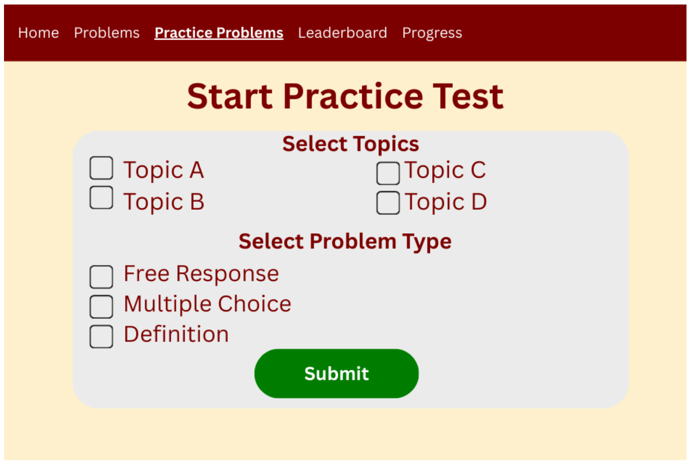


- **Notes:** In this sprint all tasks were finished. It was a particularly busy sprint with midterms but most tasks were completed earlier then merged close to the end for more testing

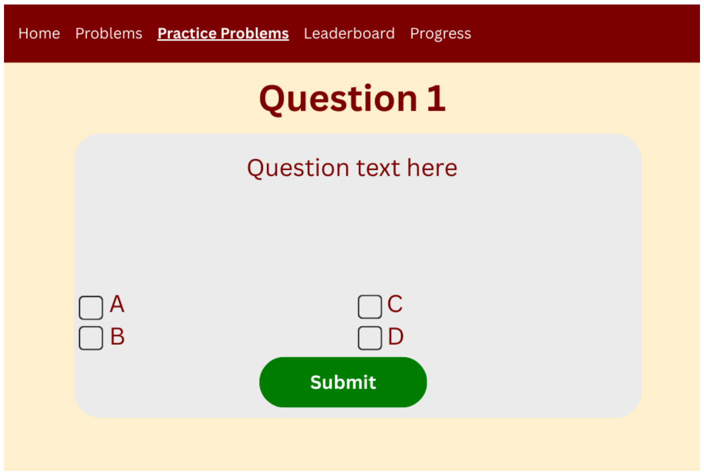
8. Design Diagrams



Example of Admin Nav Bar Diagram



Example of Student Nav Bar Diagram



Example of Multiple Choice Questions

9. Documentation of Changes

- **Changes to Design:**
  - The dashboard for instructors was changed to include more relevant content rather than summarizing info on other pages

    - The problem templates were changed to accommodate dataset questions (such as finding the mode) and only including the relevant fields for each kind of question (definition, equation, and dataset)
- **Changes to Cucumber Tests:**
    - Many features were changed to better fit the ruby code that was written and increase coverage
    - The following scenarios were added to cover the new question kinds:

```
Scenario: Displaying dataset problem with generated dataset values
        Given a predefined dataset question exists
        Given I am logged in with a valid tamu email
        And I visit the practice problems page
        And I select topic "Statistics"
        And I select question type "Free Response"
        And I press "Submit"
        Then I should be on the problem generation page
        When the problem is displayed
        Then I should see a list of numbers representing the dataset


 Scenario: Displaying median problem with generated dataset values
        Given a predefined median question exists
        Given I am logged in with a valid tamu email
        And I visit the practice problems page
        And I select topic "Statistics"
        And I select question type "Free Response"
        And I press "Submit"
        Then I should be on the problem generation page
        When the problem is displayed
        Then I should see a list of numbers representing the dataset


 Scenario: Displaying mode problem with generated dataset values
        Given a predefined mode question exists
        Given I am logged in with a valid tamu email
        And I visit the practice problems page
        And I select topic "Statistics"
        And I select question type "Free Response"
```

And I press "Submit"
Then I should be on the problem generation page
When the problem is displayed
Then I should see a list of numbers representing the dataset

Scenario: Displaying definition problem
Given a predefined definition question exists
Given I am logged in with a valid tamu email
And I visit the practice problems page
And I select topic "Velocity"
And I select question type "Free Response"
And I press "Submit"
When I should be on the problem generation page
When the problem is displayed
Then I should see the definition question

- And the following scenarios were added to accommodate the new templates:

Background:
Given I am logged in as an instructor

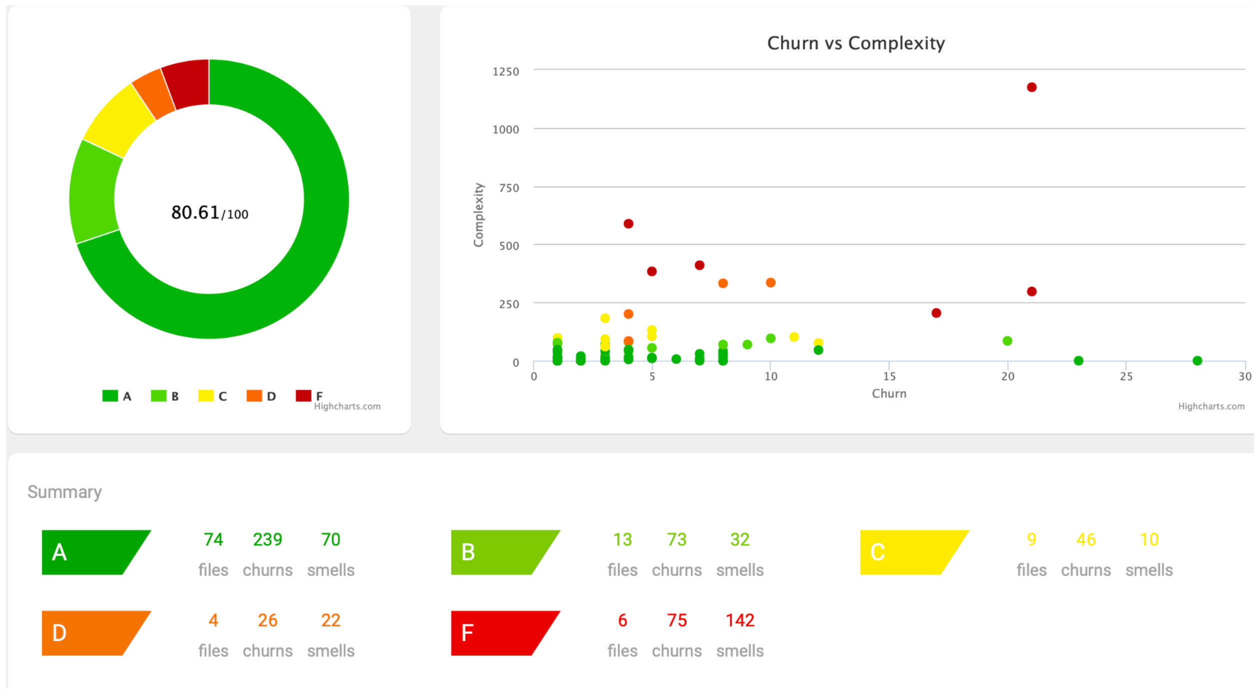
Scenario: Instructor selects equation template
When I visit the template selector page
And I select "Equation" from the question type dropdown
And I press "Continue"
Then I should be on the equation template form

Scenario: Instructor selects dataset template
When I visit the template selector page
And I select "Dataset" from the question type dropdown
And I press "Continue"
Then I should be on the dataset template form

Scenario: Instructor selects definition template
When I visit the template selector page
And I select "Definition" from the question type dropdown
And I press "Continue"
Then I should be on the definition template form

10. Evaluations of Code and Test Quality

Smells



Summary

A	74 files	239 churns	70 smells
B	13 files	73 churns	32 smells
C	9 files	46 churns	10 smells
D	4 files	26 churns	22 smells
F	6 files	75 churns	142 smells

Based on the rubycritic output, most of our code is sitting very well but there are multiple files that need to be addressed. Some of the more common smells were feature envy which occurs when a class uses another class's methods too much. Another smell that came up alot was Too Many Instance Variables, this occurs when you should be breaking up code into smaller methods. Both of these are easily addressed by refactoring and dont require logic changes.

Style Offenses

- Style issues were addressed with rubocop, there were no issues that could not be fixed either by "rubocop -a" or reading the offenses and manually fixing them.

Coverage

- Cucumber
  - 53 scenarios (15 undefined, 38 passed)
  - 342 steps (3 skipped, 27 undefined, 312 passed)

  - Line Coverage: 98.65% (478 / 485)
- Rspec

    - 171 examples, 0 failures
    - Line Coverage: 96.7% (469 /484)
- **Details:** With our efforts to write effective tests we were able to get above 90% test coverage for both Cucumber and RSpec testing.

11. Client Meeting

Meeting Date: 3/21/25

- **Meeting Time:** 4:45
- **Meeting Place:** Zoom
- **Summary:** Showed the client all of the new features that were added in sprint 3. Client was happy with results and styling, asked for a nav bar for the admin page and front page content for all users

Meeting Date: 3/28/25

- **Meeting Time:** 2:45
- **Meeting Place:** Zoom
- **Summary:** Provided the client with a detailed overview of the progress made during the last sprint. The client expressed satisfaction with the current state of the project, and we discussed plans for releasing the MVP to student users during the following week following database seeding and feedback google form creation.

12. BDD & TDD

Cucumber Features:

```
Feature: Practice Multiple Choice Questions

  As a user
  So that I can practice multiple choice questions
  I want to select multiple choice questions and solve them

  Background:
    Given I am logged in with a valid tamu email
    And the following multiple choice questions exist:
      | Topic            | Question       | Choices                    | Correct Choice |
      | Basic Arithmetic | What is 2 + 2? | 3,4,5,6                    | 4              |


  Scenario: Display and select answer choices for MCQ
    Given I have selected Multiple choice and Basic Arithmetic
    Then I should see four answer choices
    And I should be able to select one
  Scenario: Submit answer and see correct solution
    Given I have selected Multiple choice and Basic Arithmetic
    And I select the correct answer choice
    And I press submit
    Then I should see feedback indicating the answer was correct
    And I should see the correct answer explanation

Feature: Instructor creates a custom template
  As an instructor
  I want to create reusable templates
  So that I can quickly build assignments or modules

  Background:
    Given I am logged in as an instructor

  Scenario: Successfully create a custom template
    Given I am on the instructor home page
    When I click on "Add Question"
    And I select "Velocity" from "Select Topic"
    And I select "Definition" from "Select Type"
    And I fill in "Question Template Text" with "My Custom Template"
    And I fill in "Equation" with "F / m"
    And I fill in "Variables (comma separated)" with "F, m"
    And I fill in "Answer Format" with "F / m"
    And I fill in "Round Decimals" with "2"
    And I fill in "Explanation" with "Explanation"
    And I press on the button: "Create"
    Then I should see the string "Question template created successfully!"

Feature: Rounding Answers Display
    As a student
        I want the correct answer to be displayed with the proper number of decimal places
        So that I know the answer is rounded as required

    Background:
        Given I am logged in as a student
        And a predefined question exists
        And I navigate to the practice problems page
        And I select topic "Velocity"
        And I select question type "Free Response"
        And I press "Submit"

    Scenario: Displaying rounded solution with trailing zeros
```

```
        Then I should see the instruction "Round your answer to 3 decimal places"

    Scenario:
        And I input the correct solution
        When I click "Submit"
        And I should see the correct answer displayed in fixed decimal format

Feature: Navigation Bar

  Scenario: Display navigation bar
    Given I am logged in with a valid tamu email
    When I am on any page of the application
    Then I should see a navigation bar at the top of the screen

  Scenario: Navigation bar contains links for student view
    Given I am logged in as a student
    And I am on any page of the application
    Then the student navigation bar should have links to "Home", "Profile", "Logout", "Problems",
"Practice Tests", "Leaderboard", and "Progress"

  Scenario: Navigation bar contains links for instructor view
    Given I am logged in as an instructor
    And I am on any page of the application
    Then the instructor navigation bar should have links to "Home", "Profile", "Logout", "Add
Question", and "Student Summary"

  Scenario: Navigation bar contains links for admin view
    Given I am logged in as an admin
    And I am on any page of the application
    Then the admin navigation bar should have links to "Home", "Profile", "Logout", "View
Accounts", "Problems", "Practice Tests", "Leaderboard", "Progress", "Add Question", and
"Student Summary"

  Scenario: Clicking on a navigation link routes to the correct page
    Given I am logged in with a valid tamu email
    When I am on any page of the application
    When I click on the "Home" link in the navigation bar
    Then I should be redirected to the "Home" page

  Scenario: Active page is highlighted in the navigation bar
    Given I am logged in with a valid tamu email
    When I am on the "Home" page
    Then the "Home" link on the navigation bar should be bold and underlined
Feature: Instructor creates question templates
```

```
Background:
  Given I am logged in as an instructor
  And the following topics exist:
    | topic_id | topic_name        |
    | 1        | Motion            |
  And the following types exist:
    | type_id | type_name         |
    | 1       | Free Response     |
    | 2       | Multiple Choice   |

Scenario: Instructor successfully creates an equation-based template
  When I visit the equation template form
  And I fill in valid equation data
  And I press "Create Equation"
  Then I should be redirected to the instructor home page
  And I should see "Equation-based question template created!"
  And a new question with kind "equation" should exist

Scenario: Instructor submits invalid equation
  When I visit the equation template form
  And I fill in an invalid equation
  And I press "Create Equation"
  Then I should see "Invalid equation"

Scenario: Instructor creates a dataset-based question
  When I visit the dataset template form
  And I fill in valid dataset data
  And I press "Create Dataset Template"
  Then I should be redirected to the instructor home page
  And I should see "Dataset-based question template created!"
  And a new question with kind "dataset" should exist

Scenario: Instructor creates a definition-based question
  When I visit the definition template form
  And I fill in valid definition data
  And I press "Create Definition"
  Then I should be redirected to the instructor home page
  And I should see "Definition-based question template created!"
  And a new question with kind "definition" should exist

Scenario: Instructor submits definition without required fields
  When I visit the definition template form
  And I press "Create Definition"
```

```
    Then I should see "Both definition and term are required."
Feature: Instructor selects a question template type

  Background:
    Given I am logged in as an instructor

  Scenario: Instructor selects equation template
    When I visit the template selector page
    And I select "Equation" from the question type dropdown
    And I press "Continue"
    Then I should be on the equation template form

  Scenario: Instructor selects dataset template
    When I visit the template selector page
    And I select "Dataset" from the question type dropdown
    And I press "Continue"
    Then I should be on the dataset template form

  Scenario: Instructor selects definition template
    When I visit the template selector page
    And I select "Definition" from the question type dropdown
    And I press "Continue"
    Then I should be on the definition template form
```

RSpec tests:

```
    context 'when a question has round_decimals set' do
      let!(:rounding_question) do
        Question.create!(
          topic_id: 1,
          type_id: 1,
          template_text: "What is the value of e?",
          equation: "2.71828",
          variables: [],
          explanation: "Value of e",
          round_decimals: 2
        )
      end

      before do
        allow_any_instance_of(PracticeTestsController).to
receive(:evaluate_equation).and_return(2.71828)
        session[:selected_topic_ids] = [rounding_question.topic_id.to_s]
        session[:selected_type_ids] = [rounding_question.type_id.to_s]
        get :practice_test_generation
      end

      it 'rounds the solution according to round_decimals' do
        exam_questions = assigns(:exam_questions)
        expect(exam_questions.first[:solution]).to eq(2.72)
      end
    end

 describe 'GET #problem_generation' do
    context 'when session[:question_id].present?' do
      let!(:question) do
        Question.create!(
          topic_id: 1,
          type_id: 1,
          template_text: "What is velocity given position, acceleration, and time?",
          equation: "x + a * t",
          variables: [ "x", "a", "t" ],
          explanation: "Velocity is the sum of position and acceleration multiplied by time.",
          round_decimals: 2
        )
      end
require "test_helper"

class AnswerChoiceTest < ActiveSupport::TestCase
```

```
  # test "the truth" do
  #   assert true
  # end
end
# spec/controllers/problems_controller_spec.rb
require 'rails_helper'

RSpec.describe ProblemsController, type: :controller do
  let!(:topic) { create(:topic, topic_id: 1, topic_name: "Motion") }
  let!(:type)  { create(:type, type_id: 1, type_name: "Free Response") }
  let!(:user)  { create(:user, role: :student) }

  before do
    allow(controller).to receive(:current_user).and_return(user)
  end

  describe 'GET #problem_form' do
    it 'clears session keys related to problem state' do
      session[:submitted_answer] = "test"
      session[:solution] = "test"
      session[:question_text] = "test"
      session[:question_img] = "test"
      session[:question_id] = 1
      session[:try_another_problem] = true
      session[:is_correct] = false
      session[:explanation] = "explain"

      get :problem_form

      expect(session[:submitted_answer]).to be_nil
      expect(session[:solution]).to be_nil
      expect(session[:question_text]).to be_nil
      expect(session[:question_img]).to be_nil
      expect(session[:question_id]).to be_nil
      expect(session[:try_another_problem]).to be_nil
      expect(session[:is_correct]).to be_nil
      expect(session[:explanation]).to be_nil
    end
  end

  describe 'GET #problem_generation' do
    context 'when reusing question from session' do
      let!(:question) { create(:question, topic_id: topic.topic_id, type_id: type.type_id,
question_kind: 'definition', template_text: 'Define something', answer: 'answer') }
```

```
  before do
    session[:question_id] = question.id
    session[:question_text] = 'some question text'
    session[:solution] = 'some solution'
    session[:question_img] = 'image.png'
    session[:submitted_answer] = 'answer'
    session[:is_correct] = true
    session[:explanation] = 'explanation'
    session[:round_decimals] = 2
    get :problem_generation
  end

  it 'uses existing session question' do
    expect(assigns(:question)).to eq(question)
    expect(assigns(:question_text)).to eq('some question text')
    expect(assigns(:solution)).to eq('some solution')
  end
end

context "when a matching equation question exists" do
  let!(:equation_question) do
    Question.create!(
      topic_id: 1,
      type_id: 1,
      question_kind: "equation",
      template_text: "What is the final velocity given \\(x\\), \\(a\\), and \\(t\\)?",
      equation: "x + a * t",
      variables: ["x", "a", "t"],
      variable_ranges: [[1, 1], [2, 2], [3, 3]],
      variable_decimals: [0, 0, 0],
      round_decimals: 2,
      explanation: "Use v = x + a*t"
    )
  end

  before do
    session[:selected_topic_ids] = [equation_question.topic_id.to_s]
    session[:selected_type_ids] = [equation_question.type_id.to_s]
    get :problem_generation
  end

  it "assigns a question" do
    expect(assigns(:question)).to eq(equation_question)
```

```
      end

      it "stores the question data in session" do
        expect(session[:question_id]).to eq(equation_question.id)
        expect(session[:question_kind]).to eq("equation")
        expect(session[:solution]).to eq(7)
      end
    end

    context 'when no questions match' do
      before do
        session[:selected_topic_ids] = ["999"]
        session[:selected_type_ids] = ["999"]
        get :problem_generation
      end

      it 'sets a flash alert' do
        expect(flash[:alert]).to eq("No questions found with the selected topics and types. Please
try again.")
      end
    end

    context "when a dataset question is selected" do
      let!(:dataset_question) do
        Question.create!(
          topic_id: 1,
          type_id: 1,
          question_kind: "dataset",
          template_text: "Find the mode of dataset: \\( D \\)",
          dataset_generator: "10-20, size=5",
          answer_strategy: "mode",
          explanation: "Pick the most frequent number."
        )
      end

      before do
        session[:selected_topic_ids] = [dataset_question.topic_id.to_s]
        session[:selected_type_ids] = [dataset_question.type_id.to_s]
        allow_any_instance_of(ProblemsController).to receive(:generate_dataset).and_return([10,
12, 12, 14, 15])
        get :problem_generation
      end

      it "uses dataset logic and sets dataset-based solution" do
```

```ruby
      expect(assigns(:question)).to eq(dataset_question)
      expect(session[:question_kind]).to eq("dataset")
      expect(session[:solution]).to eq(12)
      expect(session[:question_text]).to include("10, 12, 12, 14, 15")
    end
  end

  context "when a definition question is selected" do
    let!(:definition_question) do
      Question.create!(
        topic_id: 1,
        type_id: 1,
        question_kind: "definition",
        template_text: "The force that resists motion between surfaces.",
        answer: "friction",
        explanation: "Friction is a contact force that opposes motion."
      )
    end

    before do
      session[:selected_topic_ids] = [definition_question.topic_id.to_s]
      session[:selected_type_ids] = [definition_question.type_id.to_s]
      get :problem_generation
    end

    it "uses definition logic and sets the answer directly" do
      expect(assigns(:question)).to eq(definition_question)
      expect(session[:question_kind]).to eq("definition")
      expect(session[:solution]).to eq("friction")
      expect(session[:question_text]).to eq(definition_question.template_text)
    end

    context 'when question is multiple choice' do
      let!(:mc_type) { create(:type, type_id: 2, type_name: "Multiple choice") }

      let!(:mc_question) do
        q = Question.create!(
          topic_id: topic.topic_id,
          type_id: mc_type.type_id,
          question_kind: "definition",
          template_text: "What is 2 + 2?",
          explanation: "2 + 2 = 4"
        )
        AnswerChoice.create!(question: q, choice_text: "3", correct: false)
        AnswerChoice.create!(question: q, choice_text: "4", correct: true)
        q
      end

      let!(:mc_choices) { mc_question.answer_choices.to_a }
      before do
        session[:selected_topic_ids] = [topic.topic_id.to_s]
        session[:selected_type_ids] = [mc_type.type_id.to_s]
        session[:question_id] = mc_question.id
        session[:question_text] = mc_question.template_text
        session[:solution] = mc_question.answer
      end

      it 'records correct answer when correct choice is submitted' do
        correct_choice = mc_choices.find(&:correct)

        expect {
          post :submit_answer, params: { answer_choice_id: correct_choice.id }
        }.to change { Submission.count }.by(1)

        expect(Submission.last.correct).to eq(true)
      end


      it 'records incorrect answer when incorrect choice is submitted' do
        incorrect_choice = mc_choices.find { |c| !c.correct }
        raise "No incorrect choice found!" unless incorrect_choice # guard

        expect {
          post :submit_answer, params: { answer_choice_id: incorrect_choice.id }
        }.to change { Submission.count }.by(1)

        expect(Submission.last.correct).to eq(false)
      end
    end
  end
end


describe 'POST #submit_answer' do
  context 'equation question logic' do
    let!(:question) { create(:question, topic_id: topic.topic_id, type_id: type.type_id,
question_kind: 'equation', equation: '2 + 2', template_text: 'Template text', variables: ['x'],
variable_ranges: [[1, 1]], variable_decimals: [0], round_decimals: 2)}
```

```
      before do
        session[:question_id] = question.id
        session[:question_kind] = 'equation'
        session[:solution] = '4'
      end

      it 'accepts correct numeric answer' do
        post :submit_answer, params: { answer: '4' }
        expect(session[:is_correct]).to eq(true)
      end

      it 'rejects incorrect answer' do
        post :submit_answer, params: { answer: '5' }
        expect(session[:is_correct]).to eq(false)
      end
    end

    context 'dataset question logic' do
      let!(:question) {
        create(:question, topic_id: topic.topic_id, type_id: type.type_id, question_kind: 'dataset',
template_text: 'Data: \( D \)', dataset_generator: '5-5, size=5', answer_strategy: 'mode')
      }

      before do
        session[:question_id] = question.id
        session[:question_kind] = 'dataset'
        session[:solution] = '5'
      end

      it 'accepts correct dataset-derived value' do
        post :submit_answer, params: { answer: '5' }
        expect(session[:is_correct]).to eq(true)
      end

      it 'rejects incorrect dataset value' do
        post :submit_answer, params: { answer: '99' }
        expect(session[:is_correct]).to eq(false)
      end
    end

    context 'definition question logic' do
      let!(:question) {
        create(:question, topic_id: topic.topic_id, type_id: type.type_id, question_kind: 'definition',
template_text: 'Define x', answer: 'truth')
      }

      before do
        session[:question_id] = question.id
        session[:question_kind] = 'definition'
        session[:solution] = 'truth'
      end

      it 'is case insensitive' do
        post :submit_answer, params: { answer: 'Truth' }
        expect(session[:is_correct]).to eq(true)
      end

      it 'rejects incorrect answer' do
        post :submit_answer, params: { answer: 'lies' }
        expect(session[:is_correct]).to eq(false)
      end
    end

    context 'unknown question_kind' do
      let!(:question) {
        create(:question, topic_id: topic.topic_id, type_id: type.type_id, template_text: 'Template
text', question_kind: 'unknown')
      }

      before do
        session[:question_id] = question.id
        session[:question_kind] = 'unknown'
        session[:solution] = 'anything'
      end

      it 'gracefully marks answer incorrect' do
        post :submit_answer, params: { answer: 'anything' }
        expect(session[:is_correct]).to eq(false)
      end
    end
  end

  describe 'GET #try_another_problem' do
    it 'sets session flag and redirects' do
      get :try_another_problem
      expect(session[:try_another_problem]).to eq(true)
      expect(response).to redirect_to(problem_generation_path)
    end
  end

  describe 'POST #create' do
    it 'stores topic and type ids in session' do
      post :create, params: { topic_ids: ["1"], type_ids: ["1"] }
      expect(session[:selected_topic_ids]).to eq(["1"])
      expect(session[:selected_type_ids]).to eq(["1"])
    end
  end

  describe "#generate_random_values" do
    it "generates correct values with ranges and decimals" do
      variables = ["x", "y"]
      ranges = [[1, 1], [2, 2]]
      decimals = [0, 1]

      controller = ProblemsController.new
      values = controller.send(:generate_random_values, variables, ranges, decimals)

      expect(values[:x]).to eq(1)
      expect(values[:y]).to eq(2.0)
    end

    it "generates default values when no range provided" do
      variables = ["a"]
      controller = ProblemsController.new
      values = controller.send(:generate_random_values, variables)
      expect(values).to have_key(:a)
      expect(values[:a]).to be_between(1, 10)
    end
  end

  describe "#generate_dataset" do
    it "returns correct dataset from generator string" do
      controller = ProblemsController.new
      dataset = controller.send(:generate_dataset, "1-1, size=5")
      expect(dataset).to eq([1, 1, 1, 1, 1])
    end

    it "returns empty array for blank generator" do
      controller = ProblemsController.new
      expect(controller.send(:generate_dataset, nil)).to eq([])
    end
  end

  describe "#compute_dataset_answer" do
    it "computes mean correctly" do
      result = controller.send(:compute_dataset_answer, [1, 2, 3], "mean")
      expect(result).to eq(2.0)
    end

    it "computes median correctly (odd)" do
      result = controller.send(:compute_dataset_answer, [3, 1, 2], "median")
      expect(result).to eq(2)
    end

    it "computes median correctly (even)" do
      result = controller.send(:compute_dataset_answer, [1, 2, 3, 4], "median")
      expect(result).to eq(2.5)
    end

    it "computes mode correctly" do
      result = controller.send(:compute_dataset_answer, [1, 2, 2, 3], "mode")
      expect(result).to eq(2)
    end

    it "returns nil for unknown strategy" do
      result = controller.send(:compute_dataset_answer, [1, 2], "unknown")
      expect(result).to be_nil
    end
  end

  describe "#format_template_text" do
    it "formats text using variable values with decimals" do
      text = "The value is \\( x \\)"
      values = { x: 3.14159 }
      decimals = [2]
      result = controller.send(:format_template_text, text, values, decimals, ["x"])
      expect(result).to eq("The value is 3.14")
    end

    it "returns original if no variables found" do
      result = controller.send(:format_template_text, "Plain text", {})
      expect(result).to eq("Plain text")
    end
  end
```

```
  describe "#evaluate_equation" do
    it "correctly evaluates expression" do
      eq = "x + y * z"
      vals = { x: 1, y: 2, z: 3 }
      result = controller.send(:evaluate_equation, eq, vals)
      expect(result).to eq(7.0)
    end

    it "returns nil on bad equation" do
      result = controller.send(:evaluate_equation, "x +", { x: 2 })
      expect(result).to be_nil
    end

    it "returns nil if values are empty" do
      result = controller.send(:evaluate_equation, "x + y", {})
      expect(result).to be_nil
    end
  end
end
# spec/controllers/problems_controller_spec.rb
require 'rails_helper'

RSpec.describe ProblemsController, type: :controller do
  let!(:topic) { create(:topic, topic_id: 1, topic_name: "Motion") }
  let!(:type)  { create(:type, type_id: 1, type_name: "Free Response") }
  let!(:user)  { create(:user, role: :student) }

  before do
    allow(controller).to receive(:current_user).and_return(user)
  end

  describe 'GET #problem_form' do
    it 'clears session keys related to problem state' do
      session[:submitted_answer] = "test"
      session[:solution] = "test"
      session[:question_text] = "test"
      session[:question_img] = "test"
      session[:question_id] = 1
      session[:try_another_problem] = true
      session[:is_correct] = false
      session[:explanation] = "explain"

      get :problem_form
      expect(session[:submitted_answer]).to be_nil
      expect(session[:solution]).to be_nil
      expect(session[:question_text]).to be_nil
      expect(session[:question_img]).to be_nil
      expect(session[:question_id]).to be_nil
      expect(session[:try_another_problem]).to be_nil
      expect(session[:is_correct]).to be_nil
      expect(session[:explanation]).to be_nil
    end
  end

  describe 'GET #problem_generation' do
    context 'when reusing question from session' do
      let!(:question) { create(:question, topic_id: topic.topic_id, type_id: type.type_id,
question_kind: 'definition', template_text: 'Define something', answer: 'answer') }

      before do
        session[:question_id] = question.id
        session[:question_text] = 'some question text'
        session[:solution] = 'some solution'
        session[:question_img] = 'image.png'
        session[:submitted_answer] = 'answer'
        session[:is_correct] = true
        session[:explanation] = 'explanation'
        session[:round_decimals] = 2
        get :problem_generation
      end

      it 'uses existing session question' do
        expect(assigns(:question)).to eq(question)
        expect(assigns(:question_text)).to eq('some question text')
        expect(assigns(:solution)).to eq('some solution')
      end
    end

    context "when a matching equation question exists" do
      let!(:equation_question) do
        Question.create!(
          topic_id: 1,
          type_id: 1,
          question_kind: "equation",
          template_text: "What is the final velocity given \\(x\\), \\(a\\), and \\(t\\)?",
          equation: "x + a * t",
          variables: ["x", "a", "t"],
          variable_ranges: [[1, 1], [2, 2], [3, 3]],
          variable_decimals: [0, 0, 0],
          round_decimals: 2,
          explanation: "Use v = x + a*t"
        )
      end

      before do
        session[:selected_topic_ids] = [equation_question.topic_id.to_s]
        session[:selected_type_ids] = [equation_question.type_id.to_s]
        get :problem_generation
      end

      it "assigns a question" do
        expect(assigns(:question)).to eq(equation_question)
      end

      it "stores the question data in session" do
        expect(session[:question_id]).to eq(equation_question.id)
        expect(session[:question_kind]).to eq("equation")
        expect(session[:solution]).to eq(7)
      end
    end

    context 'when no questions match' do
      before do
        session[:selected_topic_ids] = ["999"]
        session[:selected_type_ids] = ["999"]
        get :problem_generation
      end

      it 'sets a flash alert' do
        expect(flash[:alert]).to eq("No questions found with the selected topics and types. Please
try again.")
      end
    end

    context "when a dataset question is selected" do
      let!(:dataset_question) do
        Question.create!(
          topic_id: 1,
          type_id: 1,
          question_kind: "dataset",
          template_text: "Find the mode of dataset: \\( D \\)",
          dataset_generator: "10-20, size=5",
          answer_strategy: "mode",
          explanation: "Pick the most frequent number."
        )
      end

      before do
        session[:selected_topic_ids] = [dataset_question.topic_id.to_s]
        session[:selected_type_ids] = [dataset_question.type_id.to_s]
        allow_any_instance_of(ProblemsController).to receive(:generate_dataset).and_return([10,
12, 12, 14, 15])
        get :problem_generation
      end

      it "uses dataset logic and sets dataset-based solution" do
        expect(assigns(:question)).to eq(dataset_question)
        expect(session[:question_kind]).to eq("dataset")
        expect(session[:solution]).to eq(12)
        expect(session[:question_text]).to include("10, 12, 12, 14, 15")
      end
    end

    context "when a definition question is selected" do
      let!(:definition_question) do
        Question.create!(
          topic_id: 1,
          type_id: 1,
          question_kind: "definition",
          template_text: "The force that resists motion between surfaces.",
          answer: "friction",
          explanation: "Friction is a contact force that opposes motion."
        )
      end

      before do
        session[:selected_topic_ids] = [definition_question.topic_id.to_s]
        session[:selected_type_ids] = [definition_question.type_id.to_s]
        get :problem_generation
      end

      it "uses definition logic and sets the answer directly" do
        expect(assigns(:question)).to eq(definition_question)
        expect(session[:question_kind]).to eq("definition")
        expect(session[:solution]).to eq("friction")
        expect(session[:question_text]).to eq(definition_question.template_text)
      end

      context 'when question is multiple choice' do
        let!(:mc_type) { create(:type, type_id: 2, type_name: "Multiple choice") }

        let!(:mc_question) do
          q = Question.create!(
            topic_id: topic.topic_id,
            type_id: mc_type.type_id,
            question_kind: "definition",
            template_text: "What is 2 + 2?",
            explanation: "2 + 2 = 4"
          )
          AnswerChoice.create!(question: q, choice_text: "3", correct: false)
          AnswerChoice.create!(question: q, choice_text: "4", correct: true)
          q
        end

        let!(:mc_choices) { mc_question.answer_choices.to_a }
        before do
          session[:selected_topic_ids] = [topic.topic_id.to_s]
          session[:selected_type_ids] = [mc_type.type_id.to_s]
          session[:question_id] = mc_question.id
          session[:question_text] = mc_question.template_text
          session[:solution] = mc_question.answer
        end

        it 'records correct answer when correct choice is submitted' do
          correct_choice = mc_choices.find(&:correct)

          expect {
            post :submit_answer, params: { answer_choice_id: correct_choice.id }
          }.to change { Submission.count }.by(1)

          expect(Submission.last.correct).to eq(true)
        end


        it 'records incorrect answer when incorrect choice is submitted' do
          incorrect_choice = mc_choices.find { |c| !c.correct }
          raise "No incorrect choice found!" unless incorrect_choice # guard
```

```
        expect {
          post :submit_answer, params: { answer_choice_id: incorrect_choice.id }
        }.to change { Submission.count }.by(1)

        expect(Submission.last.correct).to eq(false)
      end
    end
  end
end


describe 'POST #submit_answer' do
  context 'equation question logic' do
    let!(:question) { create(:question, topic_id: topic.topic_id, type_id: type.type_id,
question_kind: 'equation', equation: '2 + 2', template_text: 'Template text', variables: ['x'],
variable_ranges: [[1, 1]], variable_decimals: [0], round_decimals: 2)}

    before do
      session[:question_id] = question.id
      session[:question_kind] = 'equation'
      session[:solution] = '4'
    end

    it 'accepts correct numeric answer' do
      post :submit_answer, params: { answer: '4' }
      expect(session[:is_correct]).to eq(true)
    end

    it 'rejects incorrect answer' do
      post :submit_answer, params: { answer: '5' }
      expect(session[:is_correct]).to eq(false)
    end
  end

  context 'dataset question logic' do
    let!(:question) {
      create(:question, topic_id: topic.topic_id, type_id: type.type_id, question_kind: 'dataset',
template_text: 'Data: \( D \)', dataset_generator: '5-5, size=5', answer_strategy: 'mode')
    }

    before do
      session[:question_id] = question.id
      session[:question_kind] = 'dataset'
      session[:solution] = '5'
    end

    it 'accepts correct dataset-derived value' do
      post :submit_answer, params: { answer: '5' }
      expect(session[:is_correct]).to eq(true)
    end

    it 'rejects incorrect dataset value' do
      post :submit_answer, params: { answer: '99' }
      expect(session[:is_correct]).to eq(false)
    end
  end

  context 'definition question logic' do
    let!(:question) {
      create(:question, topic_id: topic.topic_id, type_id: type.type_id, question_kind: 'definition',
template_text: 'Define x', answer: 'truth')
    }

    before do
      session[:question_id] = question.id
      session[:question_kind] = 'definition'
      session[:solution] = 'truth'
    end

    it 'is case insensitive' do
      post :submit_answer, params: { answer: 'Truth' }
      expect(session[:is_correct]).to eq(true)
    end

    it 'rejects incorrect answer' do
      post :submit_answer, params: { answer: 'lies' }
      expect(session[:is_correct]).to eq(false)
    end
  end

  context 'unknown question_kind' do
    let!(:question) {
      create(:question, topic_id: topic.topic_id, type_id: type.type_id, template_text: 'Template
text', question_kind: 'unknown')
    }

    before do
      session[:question_id] = question.id
      session[:question_kind] = 'unknown'
      session[:solution] = 'anything'
    end

    it 'gracefully marks answer incorrect' do
      post :submit_answer, params: { answer: 'anything' }
      expect(session[:is_correct]).to eq(false)
    end
  end
end

describe 'GET #try_another_problem' do
  it 'sets session flag and redirects' do
    get :try_another_problem
    expect(session[:try_another_problem]).to eq(true)
    expect(response).to redirect_to(problem_generation_path)
  end
end

describe 'POST #create' do
  it 'stores topic and type ids in session' do
    post :create, params: { topic_ids: ["1"], type_ids: ["1"] }
    expect(session[:selected_topic_ids]).to eq(["1"])
    expect(session[:selected_type_ids]).to eq(["1"])
  end
end

describe "#generate_random_values" do
  it "generates correct values with ranges and decimals" do
    variables = ["x", "y"]
    ranges = [[1, 1], [2, 2]]
    decimals = [0, 1]

    controller = ProblemsController.new
    values = controller.send(:generate_random_values, variables, ranges, decimals)

    expect(values[:x]).to eq(1)
    expect(values[:y]).to eq(2.0)
  end

  it "generates default values when no range provided" do
    variables = ["a"]
    controller = ProblemsController.new
    values = controller.send(:generate_random_values, variables)
    expect(values).to have_key(:a)
    expect(values[:a]).to be_between(1, 10)
  end
end

describe "#generate_dataset" do
  it "returns correct dataset from generator string" do
    controller = ProblemsController.new
    dataset = controller.send(:generate_dataset, "1-1, size=5")
    expect(dataset).to eq([1, 1, 1, 1, 1])
  end

  it "returns empty array for blank generator" do
    controller = ProblemsController.new
    expect(controller.send(:generate_dataset, nil)).to eq([])
  end
end

describe "#compute_dataset_answer" do
  it "computes mean correctly" do
    result = controller.send(:compute_dataset_answer, [1, 2, 3], "mean")
    expect(result).to eq(2.0)
  end

  it "computes median correctly (odd)" do
    result = controller.send(:compute_dataset_answer, [3, 1, 2], "median")
    expect(result).to eq(2)
  end

  it "computes median correctly (even)" do
    result = controller.send(:compute_dataset_answer, [1, 2, 3, 4], "median")
    expect(result).to eq(2.5)
  end

  it "computes mode correctly" do
    result = controller.send(:compute_dataset_answer, [1, 2, 2, 3], "mode")
    expect(result).to eq(2)
  end

  it "returns nil for unknown strategy" do
    result = controller.send(:compute_dataset_answer, [1, 2], "unknown")
    expect(result).to be_nil
  end
end
```

```
  describe "#format_template_text" do
    it "formats text using variable values with decimals" do
      text = "The value is \\( x \\)"
      values = { x: 3.14159 }
      decimals = [2]
      result = controller.send(:format_template_text, text, values, decimals, ["x"])
      expect(result).to eq("The value is 3.14")
    end

    it "returns original if no variables found" do
      result = controller.send(:format_template_text, "Plain text", {})
      expect(result).to eq("Plain text")
    end
  end

  describe "#evaluate_equation" do
    it "correctly evaluates expression" do
      eq = "x + y * z"
      vals = { x: 1, y: 2, z: 3 }
      result = controller.send(:evaluate_equation, eq, vals)
      expect(result).to eq(7.0)
    end

    it "returns nil on bad equation" do
      result = controller.send(:evaluate_equation, "x +", { x: 2 })
      expect(result).to be_nil
    end

    it "returns nil if values are empty" do
      result = controller.send(:evaluate_equation, "x + y", {})
      expect(result).to be_nil
    end
  end
end
require 'rails_helper'

RSpec.describe TemplatesController, type: :controller do
  let(:instructor) { User.create!(first_name: "Inst", last_name: "Ructor", email:
"inst@example.com", role: :instructor) }
  let!(:topic) { Topic.create!(topic_id: 1, topic_name: "Physics") }
  let!(:type)  { Type.create!(type_id: 1, type_name: "Free Response") }

  before do
```

```
    allow(controller).to receive(:current_user).and_return(instructor)
  end

  describe "GET template forms" do
    it "renders new_equation template" do
      get :new_equation
      expect(response).to have_http_status(:success)
      expect(response).to render_template(:new_equation)
    end

    it "renders new_dataset template" do
      get :new_dataset
      expect(response).to have_http_status(:success)
      expect(response).to render_template(:new_dataset)
    end

    it "renders new_definition template" do
      get :new_definition
      expect(response).to have_http_status(:success)
      expect(response).to render_template(:new_definition)
    end
  end

  describe "POST #create_equation" do
    context "with valid input" do
      it "creates an equation question and redirects" do
        Question.delete_all
        post :create_equation, params: {
          topic_id: topic.id,
          type_id: type.id,
          template_text: "Calculate final velocity: \\(x\\), \\(a\\), \\(t\\)",
          equation: "x + a^(2)",
          variables: "x, a",
          variable_ranges: "1-10, 2-5",
          variable_decimals: "0, 0",
          answer: "x + a^2",
          round_decimals: 2,
          explanation: "v = x + a²"
        }

        expect(Question.last.question_kind).to eq("equation")
        expect(flash[:notice]).to eq("Equation-based question template created!")
        expect(response).to redirect_to(instructor_home_path)
      end
  end

  context "with invalid equation" do
    it "redirects back with error" do
      post :create_equation, params: {
        topic_id: topic.id,
        type_id: type.id,
        template_text: "Bad equation",
        equation: "x + (", # malformed
        variables: "x",
        variable_ranges: "1-10",
        variable_decimals: "0",
        answer: "error",
        round_decimals: 2,
        explanation: "fail"
      }

      expect(flash[:alert]).to match(/Invalid equation/)
      expect(response).to redirect_to(custom_template_equation_path)
    end
  end
end

describe "POST #create_dataset" do
  context "with missing fields" do
    it "redirects back with error" do
      post :create_dataset, params: {
        topic_id: topic.id,
        type_id: type.id,
        template_text: "Find the mode of \\( D \\)",
        dataset_generator: "",
        answer_strategy: ""
      }

      expect(flash[:alert]).to eq("Dataset generator and answer type are required.")
      expect(response).to redirect_to(custom_template_dataset_path)
    end
  end

  context "with valid input" do
    it "creates dataset question and redirects" do
      post :create_dataset, params: {
        topic_id: topic.id,
        type_id: type.id,
          template_text: "Find the mode of \\( D \\)",
          dataset_generator: "1-10, size=5",
          answer_strategy: "mode",
          explanation: "Find most common"
        }

        expect(Question.last.question_kind).to eq("dataset")
        expect(response).to redirect_to(instructor_home_path)
      end
    end
  end

  describe "POST #create_definition" do
    context "with missing fields" do
      it "redirects back with error" do
        post :create_definition, params: {
          topic_id: topic.id,
          type_id: type.id,
          template_text: "",
          answer: ""
        }

        expect(flash[:alert]).to eq("Both definition and term are required.")
        expect(response).to redirect_to(custom_template_definition_path)
      end
    end

    context "with valid input" do
      it "creates a definition question and redirects" do
        post :create_definition, params: {
          topic_id: topic.id,
          type_id: type.id,
          template_text: "The force that opposes motion between surfaces.",
          answer: "friction",
          explanation: "Friction is the term"
        }

        expect(Question.last.question_kind).to eq("definition")
        expect(response).to redirect_to(instructor_home_path)
      end
    end
  end
end
```