Automated Assessment of Students' Short Written Answers Using NLP and Large Language Models (LLMs)

Authors:
Arian Assadzadeh
Demiana Attia
Sanjana Ghanta
Tai Phan
Trey Walker

May 9, 2025

Course: CS4624 Multimedia/Hypertext
Virginia Tech
Instructor and Client: Dr. Mohamed Farag

Contents

List of Figures

1 Executive Summary / Abstract

This project proposes the design and implementation of an automated web-based system for automated formative assessment of student short written answer questions using Natural Language Processing (NLP) and Large Language Models (LLM). The proposed solution addresses the ever-growing challenge in modern education, where educators face increasing class sizes, different types of approach, and student abilities, and limited time to give personalized individual feedback for each student. The system utilizes Gemma3:4b LLM, hosted via Ollama, integrated with a React frontend and FastAPI backend to accomplish this task. Using recent advances in the area of machine learning, the objective of this project is to offer a scalable, efficient, and meaningful tool that can assist both students and instructors in the feedback process. The tool will feature user authentication, answer uploads, short answer grading, and grading results.

2 Introduction

The Automated Short Answers Assessment system is a web-based platform designed to grade students' short answer questions using advanced Natural Language Processing (NLP) techniques and open-source Large Language Models (LLMs). This project addresses the increasing need in modern education; it reduces the amount of time it takes to manually grade questions. Teachers no longer need to put in the cognitive effort as well when it comes to grading with the assist of the automated short answers assessment system.

Using NLP and LLMs, our system identifies and scores key ideas in student answers and evaluates how well they align with model answers or rubrics. This not only supports instructors in timely grading but also gives students immediate feedback to improve learning outcomes.

2.1 Client Introduction

The client for this project is Dr. Mohamed Farag, a faculty member at Virginia Tech. Dr. Farag specializes in computer science education and research, with a particular interest in scalable, data-driven educational tools. Dr. Farag has received is Ph.D. in computer science from Virginia Tech in 2016, and his M.Sc. degree in computer science from the Arab Academy for Science, Technology, and Maritime Transport (AAST) in Alexandria, Egypt in 2010. As a research associate at the Virginia Tech Transportation Institute, Dr. Farag has extensive experience leading in interdisciplinary projects that brings about applied computing and education impact. His role in this project ranges from setting requirements, providing feedback during our bi-monthly meetings, and guiding the team's progress and work with real-world instructional needs.

Contact Information: 540-231-0278, mfarag@vt.edu

3 Requirements

3.1 Functional Requirements

These requirements define the core features that the system must support to satisfy the project's objectives.

Here are the following features and their corresponding description:

- User Login and Authentication
  - Uses JWT-based authentication with NextAuth.js on the frontend and FastAPI on the backend.
- Student Answer Upload
  - Instructors can upload individual or a batch of student short answers through the website or by uploading a csv file.
- Model Answer Input
  - Instructors can specify idea/model answers per question for a comparison.
- Rubric Input
  - Users can define multi-level rubrics for a grading-by-criteria approach.
- Concept Mapping Input
  - Allows the educator, or rather user to list curriculum-relevant key ideas or phrases to check for in student answers.
- Three Grading Modes
  - Rubric: User provides a rubric for the LLM to grade against
  - Concepts: User provides several concepts for the LLM to grade against
  - Sample Answer: User provides a sample answer for the LLM to compare to

3.2 Non-functional Requirements

These capture the expected performance, usability, maintainability, and other quality factors of the system.

Here are the categories :

- Performance
- Scalability
- Security
- Portability

- Maintainability
- Usability
- Accessibility

3.3 Deliverables

Below is a table of deliverables for this semester that we would like to achieve.

Change to what we accomplished and didn't

Deliverable	**Status**	**Notes**
Working Web Application	Completed	Working login, collections, assessment logic partially in place)
Codebase (Frontend + Backend)	Completed	GitHub structure shown, work underway
Preprocessing and Evaluation Scripts	Completed	Model optimization
User Manual	Completed	Screens and flow exist (see slides), but not full write-up yet
Developer Manual	Completed	Consistent with internal work and repo design
Performance and Comparison Report	Completed	Can be generated within the admin panel
Final Documentation	Completed	Milestone 5 in timeline: "write and publish documentation
VTechWorks Archive Package	Completed	Will work according to timeline

Table 1: Project Deliverables Status

As the semester comes to an end, all the original deliverables have been met and accomplished by the team. There have been additional things added that were also accomplished, such as adding an admin panel.

4 Design

4.1 General Approach

The project is structured as a modular, full-stack web application that integrates front-end, back-end, database, and machine learning components into a unified pipeline. The end goal of the design is to ensure there is proper cohesiveness across components, enabling iterative development and easy future extension.

The architecture is built based on a microservice-inspired structure that uses Docker Compose for orchestration. Each part of the system (front-end, back-end, model engine) can run and be developed independently but communicates via APIs.

4.2 Backend

The backend is implemented using FastAPI and provides the following endpoints:

Endpoint	**Description**
POST /login	Authenticates user and returns JWT token
POST /collections	Creates a collection for the current user
POST /collections/<collection_id>/upload-questions	Accepts question uploads for a question or bat
POST /collections/<collection_id>/upload-answers	Accepts student answer uploads for a question
POST /student-answers/<student_answer_id>/grade	Invokes the grading engine for a given answer i
GET /results/{collection_id}	Retrieves a formatted table of graded results

Table 2: API Endpoints and Descriptions

The backend also handles:

- Schema validation with Pydantic
- Role-based permissions
- Input salinization
- Logging for traceability

4.3 Frontend

Built using React.js, the frontend is organized into:

Component	**Functionality**
LoginPage	Handles secure login flow via NextAuth.js
Upload Form	Lets instructors upload student answers and select grading mode

Table 3: Responsive design and accessibility are prioritized using semantic HTML

5 Implementation

5.1 Overview

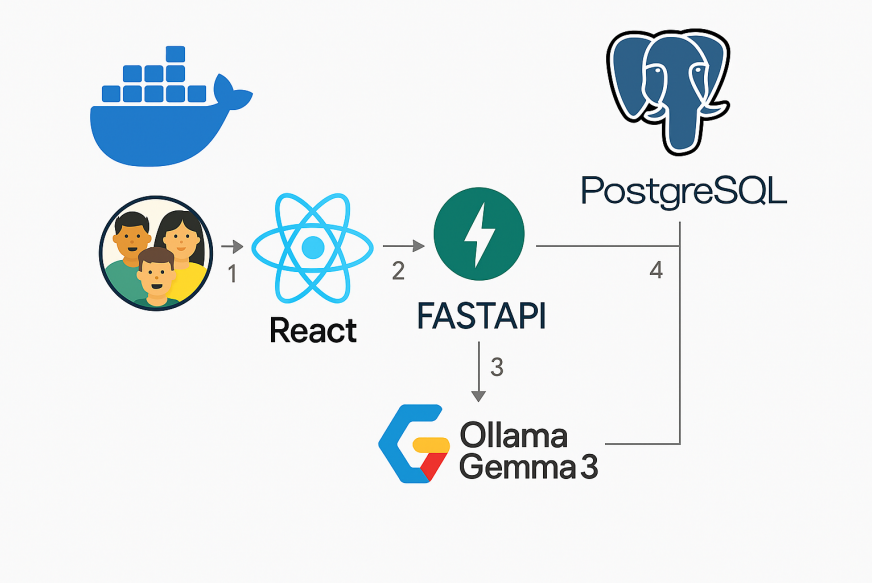

Figure 1: System Architecture

The core components: **Frontend** (React.js + NextAuth.js)

- Handles login, answer upload, and display of results
- Interacts with back-end via RESTful API calls
- Manages authentication using NextAuth.js and JWT tokens

Backend (FastAPI)

- Acts as the core controller and orchestrator of the system
- Manages authentication, routes, and validation
- Calls scoring engine and processes its results
- Interacts with the PostgreSQL database to store/retrieve users, answers, and collections

LLM Model: Gemma3:4b

- Lightweight and efficient open-source LLM optimized for reasoning and instruction-following tasks
- Hosted locally through Ollama, which handles model downloading
- Receives API calls from backend, evaluates student answers based on:

  - Semantic similarity to a model answer
  - Rubric-based alignment
  - Presence of key concepts
- Responds with structured outputs

Evaluation Metrics: Mean Absolute Error (MAE) is used to evaluate the performance of models in grading known questions and answers.

5.2 Backend

This project utilizes a FastAPI backend to create a RESTful application that contains several API endpoints to interact with the backend. At its core, the backend uses a REST API to perform CRUD operations on collections in the database and integrates LLM functionalities through HTTP calls. The FastAPI application tracks authentication using JWT tokens and employs password hashing for security in the database. Lastly, it uses SQLAlchemy to simplify database access.

A large portion of the backend codebase is dedicated to maintaining robust API endpoints so that the frontend and backend can communicate effectively using HTTP calls. To achieve this, a workflow was followed for each entity identified in the design phase of the project:

1. Create SQLAlchemy model in `models/` to represent the business logic object
2. Create a Pydantic schema for that model to standardize data validation for HTTP calls and responses
3. Create desired CRUD operations for that model in the `crud.py` entrypoint to the database
4. Create routes with error handling in the `api/` folder using the `crud.py` operations for that model

The backend is also the entry point into the automated grading functionality by accessing and managing large-language models through Ollama. To reduce dependencies on a specific model, the backend provides functionality to switch between different downloaded models, as well as allowing custom prompts to be used so that the application can function seamlessly if any of these components must be changed.

5.3 Frontend

The frontend is a React-based web application designed to manage and assess students' short answers. It provides an interface for teachers/instructors to create exam collections, add questions, manage students, and collect student answers. The application follows a component-based architecture with a clear separation of concerns.

5.3.1 Core Technologies

- **React:** The application is built using React (version 19.0.0)
- **React Router:** Used for navigation and routing (version 7.4.0)
- **Axios:** For making HTTP requests to the backend API

5.3.2 Application Structure

The frontend follows a standard React application structure:

5.3.3 Entry Points

- `index.js`: The application entry point
- `App.js`: Main component defining the routing structure

5.3.4 Pages

High-level components representing entire views:

- `Login.js`: Handles user authentication
- `Home.js`: Dashboard for viewing and managing collections
- `Students.js`: Interface for managing student data

5.3.5 Components

Reusable UI elements:

- `CollectionDetail.js`: Displays and manages collection information and questions
- `QuestionForm.js`: Form for creating and editing questions
- `StudentForm.js`: Form for adding and editing students
- `StudentDetail.js`: Displays student information and answers
- `StudentAnswerForm.js`: Interface for submitting student answers

5.3.6 Services

API communication layer:

- `api.js`: Contains all API calls to the backend

5.3.7 Authentication

- The application starts at the login page (`Login.js`)
- Users can login with username and password or register a new account
- Authentication uses JWT tokens stored in localStorage

5.3.8 Collection Management

- After login, users are directed to the home page (`Home.js`)
- Teachers can view their exam collections
- They can create new collections with a name and description
- Clicking on a collection opens the `CollectionDetail` component

5.3.9 Question Management

- Within `CollectionDetail`, users can:
  - View collection details
  - Add, edit, and delete questions
  - Each question has text content and an example answer

5.3.10 Student Management

- The Students page (`Students.js`) allows users to:
  - View all students
  - Add new students with name and school ID
  - Edit and delete existing students
  - View student details including submitted answers

5.3.11 Answer Submission System

- Users can submit answers for students through:
  - The "Submit Answers" tab in `CollectionDetail`
  - Select a student, answer questions, and submit
- `StudentAnswerForm` handles the submission of answers
- The system supports bulk submission of multiple answers at once

5.3.12 API Communication

- The frontend communicates with the backend through a REST API (hosted at `http://localhost:8001/api`)
- All API calls are organized in the `api.js` file, categorized by resource type:
  - Collection operations
  - Question operations
  - User operations
  - Student operations
  - Student answers operations

5.3.13 User Interface Design

- The application uses custom CSS styling for each component
- Modal dialogs are used for forms and detail views
- The interface includes a top taskbar for navigation between main sections
- Responsive design elements for forms and listings

5.3.14 Data Flow

1. User authenticates and receives a JWT token
2. Token is used for all subsequent API requests
3. UI components fetch data from the API when needed
4. User actions (create/edit/delete) trigger API calls and UI updates
5. Component state manages temporary data and UI rendering

This architecture enables a complete workflow for managing

6 Testing/Evaluation/Assessment

We conducted extensive model selection to determine the most suitable language model for grading. Eleven different models were evaluated using a set of 20 diverse questions, each associated with a range of acceptable answers scored from 0.0 to 1.0.

The evaluation prompt format was:

```
Question: {question}
Correct Answer: {model_answer}
Student's Answer: {student_answer}
Grade the student's answer based on the correct answer from (0.0 - 1.0)
```

We compared the grades produced by each model against reference (human) scores and used accuracy (1.0 - absolute error) as the main metric. Based on this, we selected **gemma3:4b**, which achieved an average accuracy of **89%**.

In addition to semantic and lexical similarity, we evaluated:

- **Grade Accuracy**: Calculated as $1.0 - |\text{model grade} - \text{reference grade}|$, which measures how close the model's output is to the human-labeled score (with 1.0 being a perfect match).
- **Grade Stability**: Standard deviation of grades across repeated evaluations, converted into a consistency score scaled from 0.0 (inconsistent) to 1.0 (perfect consistency).

7 Comparison with Previous System

To contextualize our evaluation results, we compared the performance of our selected model, **gemma3:4b**, against the prior system used by the previous team. Both systems were evaluated on the same set of 20 diverse questions with 100 total grading attempts. The performance metric is calculated with MAE by taking the absolute value of the extracted grade minus the model grade.

Metric	**Previous System**	**Gemma3:4b (Ours)**
Questions Evaluated	20	20
Grading Attempts	100	100
Overall Performance	0.669	**0.89**
Consistency Score	**1.00**	0.92
Average Response Time (s)	**1.84**	2.96

Table 4: Performance comparison between previous system and Gemma3:4b.

While the previous system achieved faster response times and perfect consistency, our model significantly outperformed it in grading performance improving from 66.9% to 89%, a relative gain of over 33%. This indicates that Gemma3:4b better approximates human graders, despite a modest trade-off in latency and slightly reduced stability. The previous system achieved a perfect consistency score of 1.00 because it produced identical outputs for repeated inputs. This is due to its rule-based grading mechanism, which relied solely on deterministic similarity thresholds and fixed phrase matching using the SentenceTransformer model, without any stochastic elements or sampling variability.

Ultimately, the enhanced semantic reasoning capabilities of Gemma3:4b make it the more reliable and effective grader for open-ended short-answer assessment tasks.

8 Feature Overview

This section outlines the core functionalities and capabilities of the system.

8.1 User Authentication and Access

- **User Login:** Registered users can log in using their unique username and password.
- **Account Creation:** New users can easily register for an account directly through the login interface.
- **JWT-Based Security:** Authentication is handled securely using JWT tokens, ensuring user sessions are protected.
- **Administrator Access:** Administrator accounts are supported, which have access to more features than normal end-users

8.2 Navigation and Interface

- **Intuitive Navigation Bar:** A persistent top navigation bar allows easy access to the main sections: Home (Collections), Students, and Admin Panel if the user is an Admin.
- **User Profile Access:** User-specific options are accessible via a user icon, located in the top-right corner.
- **Card-Based Displays:** Collections and students are presented using clear, informative cards for quick overview and selection.
- **Modal Dialogs:** Forms for creating/editing items (collections, questions, students) and viewing details are presented in modal windows.

8.3 Collection Management

- **Assessment Organization:** Users can create and manage 'Collections' to group related questions and student answers. A 'Collection' usually represents a specific exam or assignment
- **Collection Creation:** Create new collections by providing a name and an optional description.
- **Centralized Viewing:** The Home page serves as a dashboard displaying all created collections.

8.4 Question Management

- **Question Authoring:** Within a collection, users can add new short answer questions.
- **Model Answer Input:** Each question requires associated text and a corresponding model or example answer used for grading comparison.
- **CRUD Operations:** Full support for Creating, Reading, Updating, and Deleting questions within their respective collections.

8.5 Student Management

- **Student Roster:** Maintain a list of students, each identified by name and a unique ID.
- **Student Creation/Editing:** Add new students or modify existing student information (name, ID).
- **Student Deletion:** Remove student records as needed.
- **Centralized Student View:** A dedicated 'Students' section lists all registered students.
- **Individual Student Details:** View detailed information for each student, including their profile and a history of submitted answers across collections.

8.6 Answer Submission and Grading

- **Manual Answer Entry:** Provides an interface within a collection's detail view to manually enter answers for a selected student across all questions in that collection.
- **CSV Upload (Mentioned Elsewhere):** Provides support for bulk upload of questions, answers, and student answers.
- **Answer Viewing:** View previously submitted answers and graded scores.
- **Grading Trigger:** Grade a single student answer at a time or grade all student answers at once

8.7 Admin View

- **LLM Management:** Provides an interface to view and download LLMs supported by Ollama.
- **Prompt Management:** Provides support for the creations of custom prompts for the LLM's.
- **LLM-Prompt Pairs:** Gives the ability to pair specific LLM's with custom prompts

- **Robust Testing Wizard:** Testing Wizard allows testing of multiple LLM's and prompts with a specific question set to assess optimal combinations. The results of the test is displayed as the Mean Absolute Error of the LLM assessment on the specified question set.

9 User Guide for Short Answer Assessment System

This guide will walk you through the complete functionality of the Automated Student Short Answer Assessment system.

9.1 Getting Started: Login and Account Creation

1. Navigate to the application at **http://localhost:3000/**

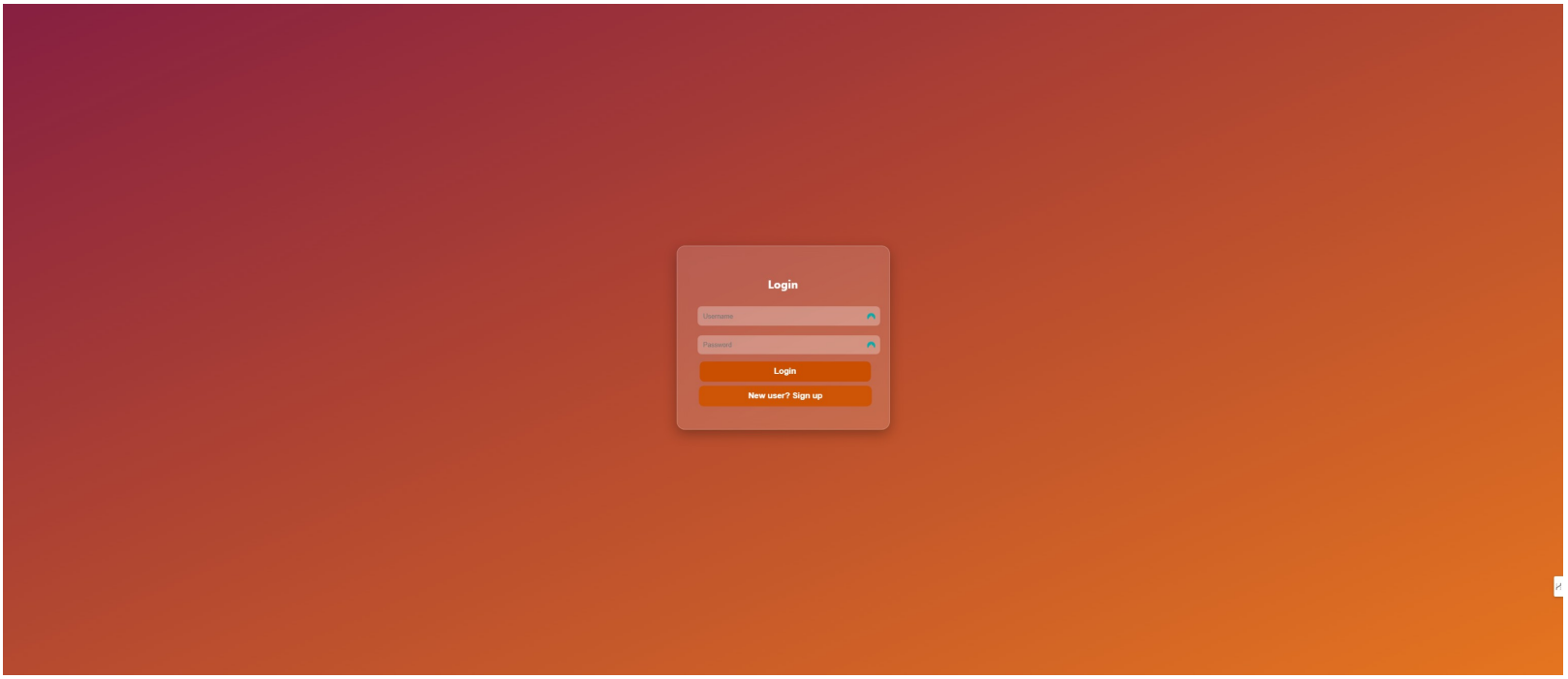

Figure 2: Initial login screen - The system welcomes you with a clean login interface

2. Click the **"New user? Sign up"** link at the bottom of the login form

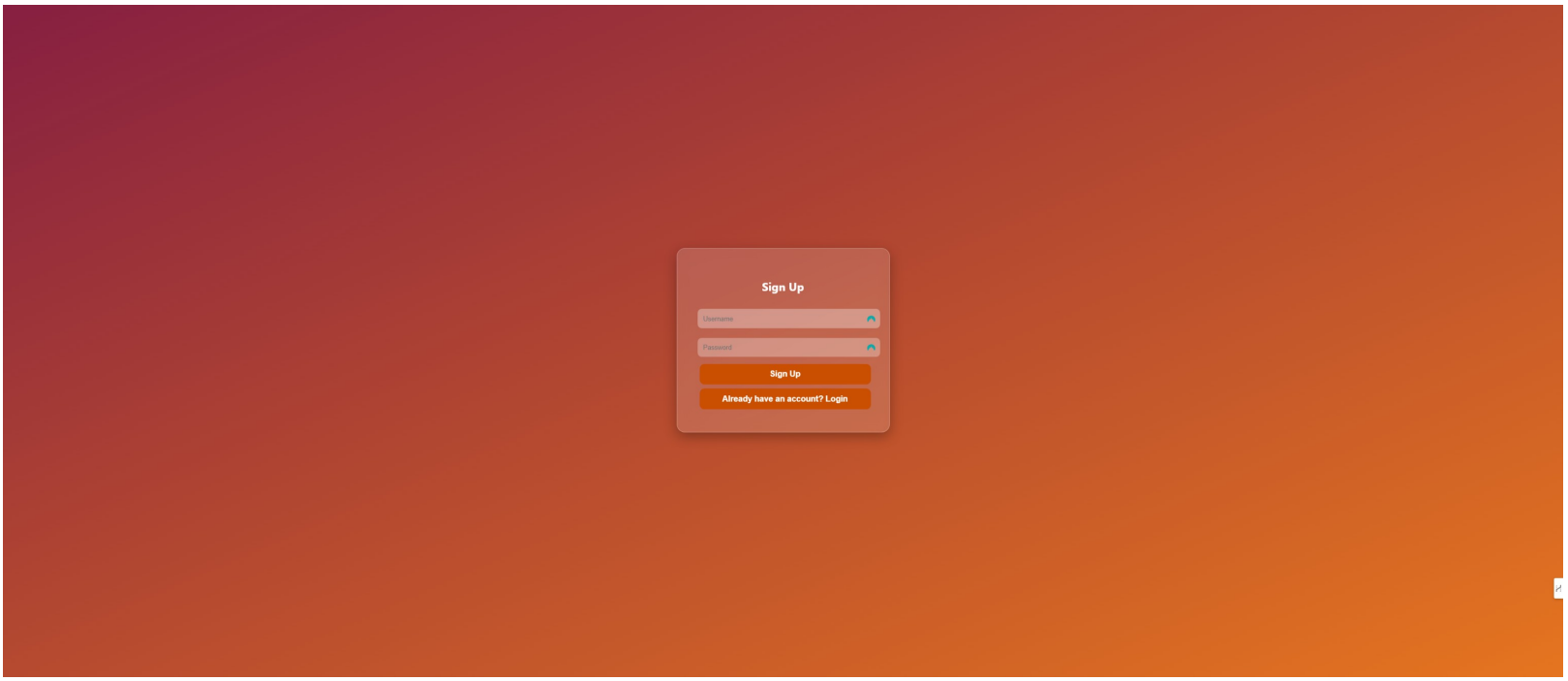

Figure 3: Sign-up option - Access the registration form for new users

3. Enter your desired username in the Username field

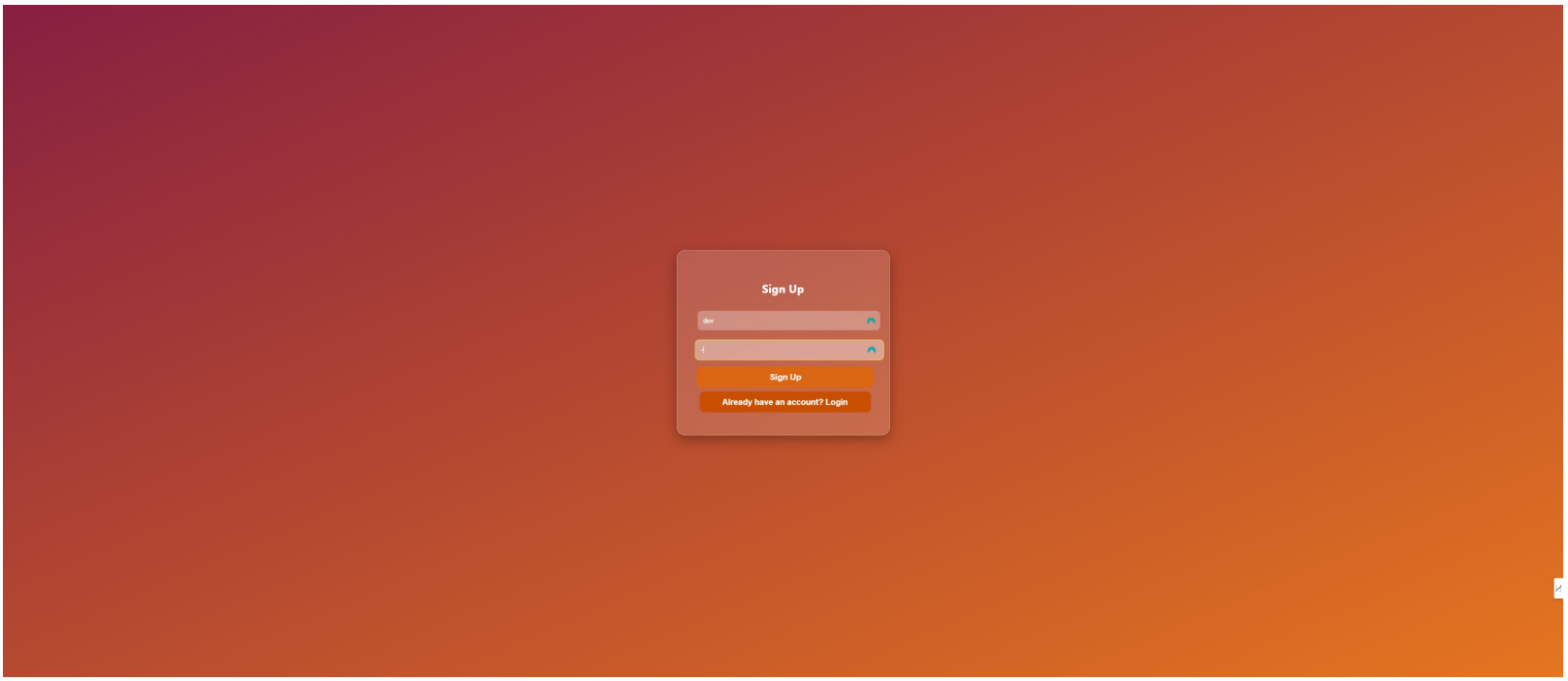

Figure 4: Registration form - Enter your credentials in the designated fields

4. For an administrator account, enter **"dev"** as your username and create a secure password

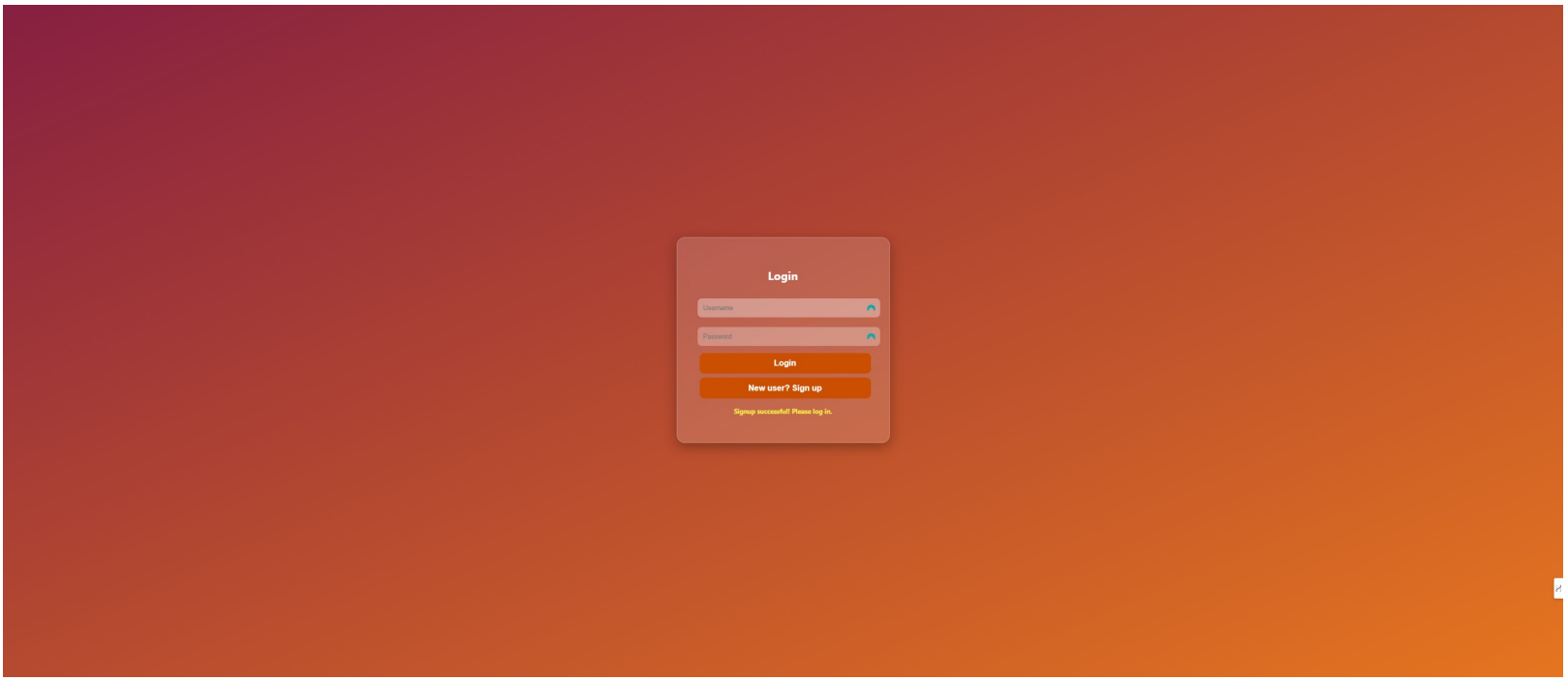

Figure 5: Successful registration - The system confirms your account creation

5. Click the **"Sign Up"** button to complete your registration
6. After successful registration, enter your username and password again to log in
7. Once logged in, you'll arrive at the home page where you can manage your collections

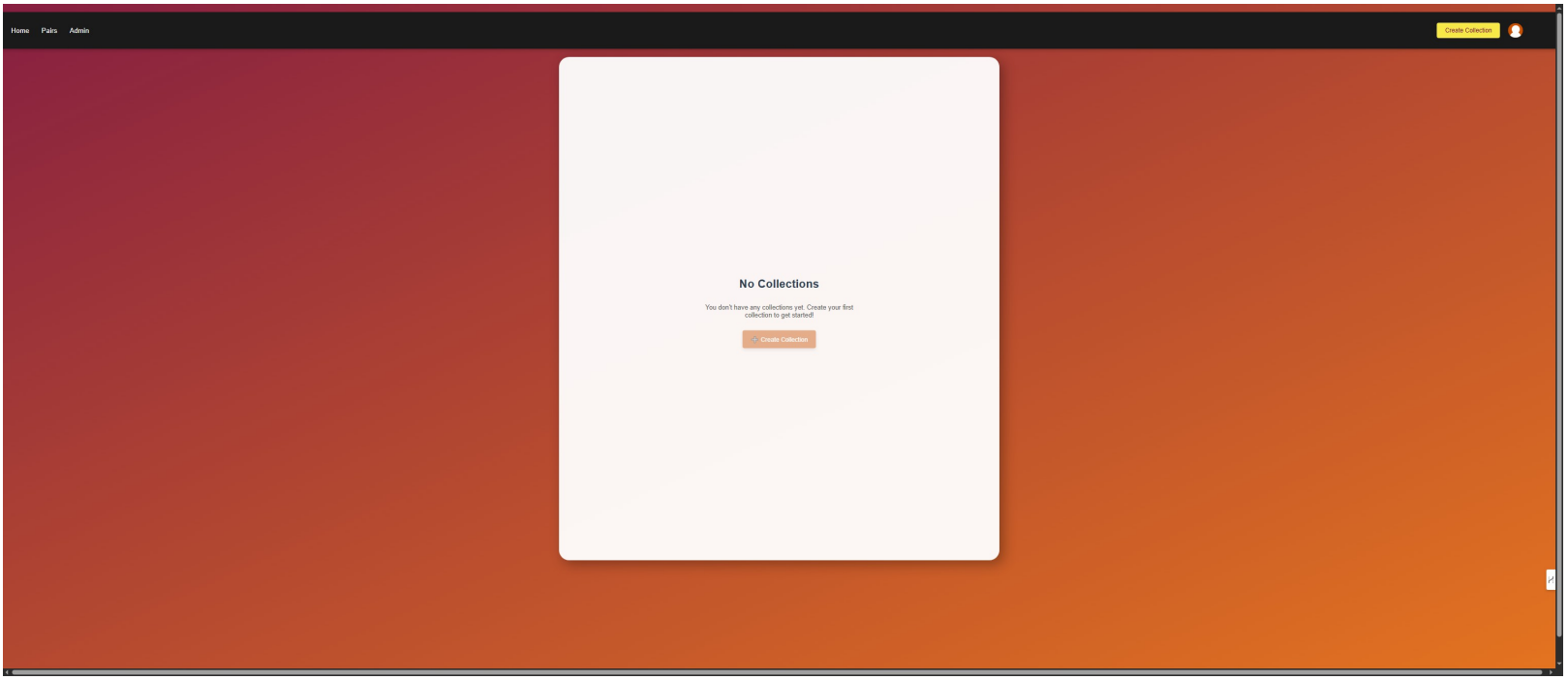

Figure 6: Home page - Central dashboard for accessing all system features

9.2 Creating and Managing Collections

Collections allow you to organize sets of questions and student answers for grading.

8. Click the **"+ Create Collection"** button on the home screen

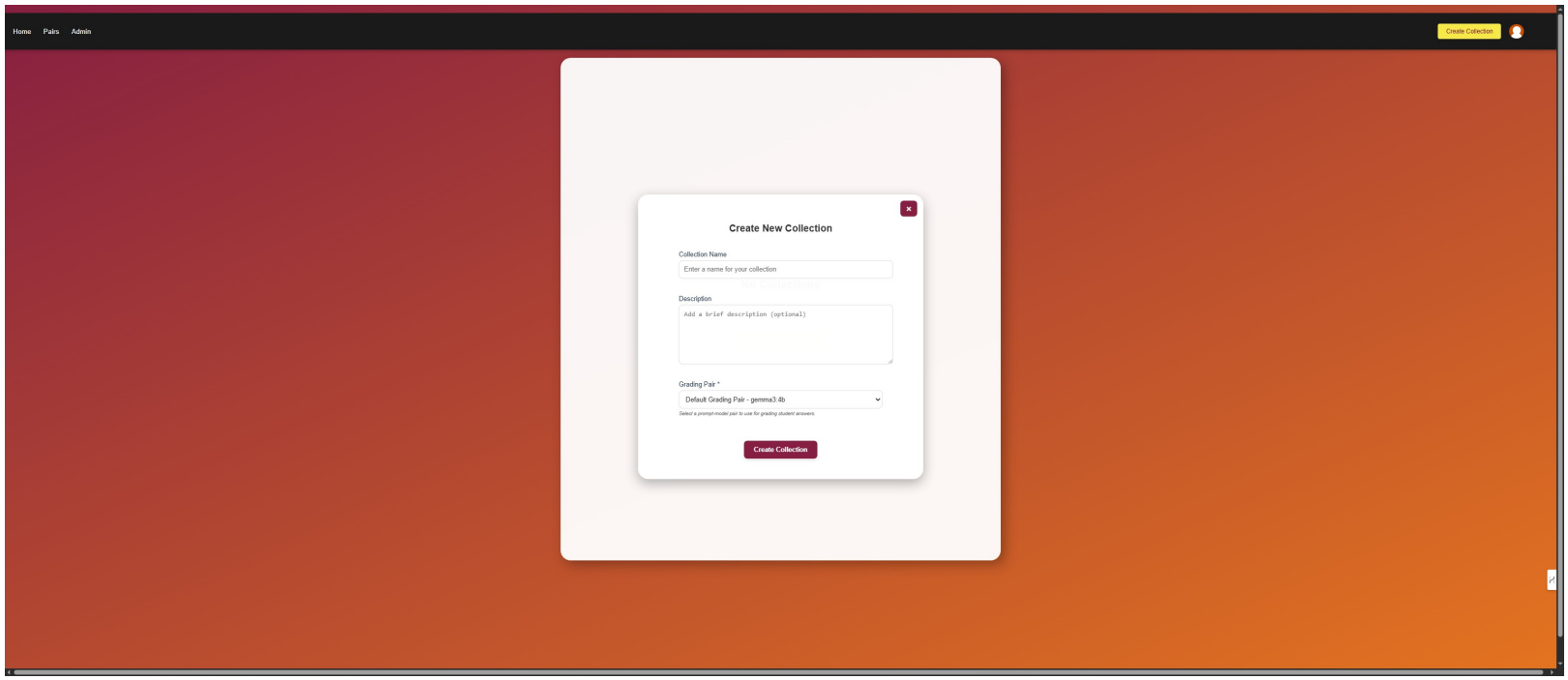

Figure 7: Collection creation - Initiate the process of creating a new assessment collection

9. Enter a name for your collection in the **"Collection Name"** field

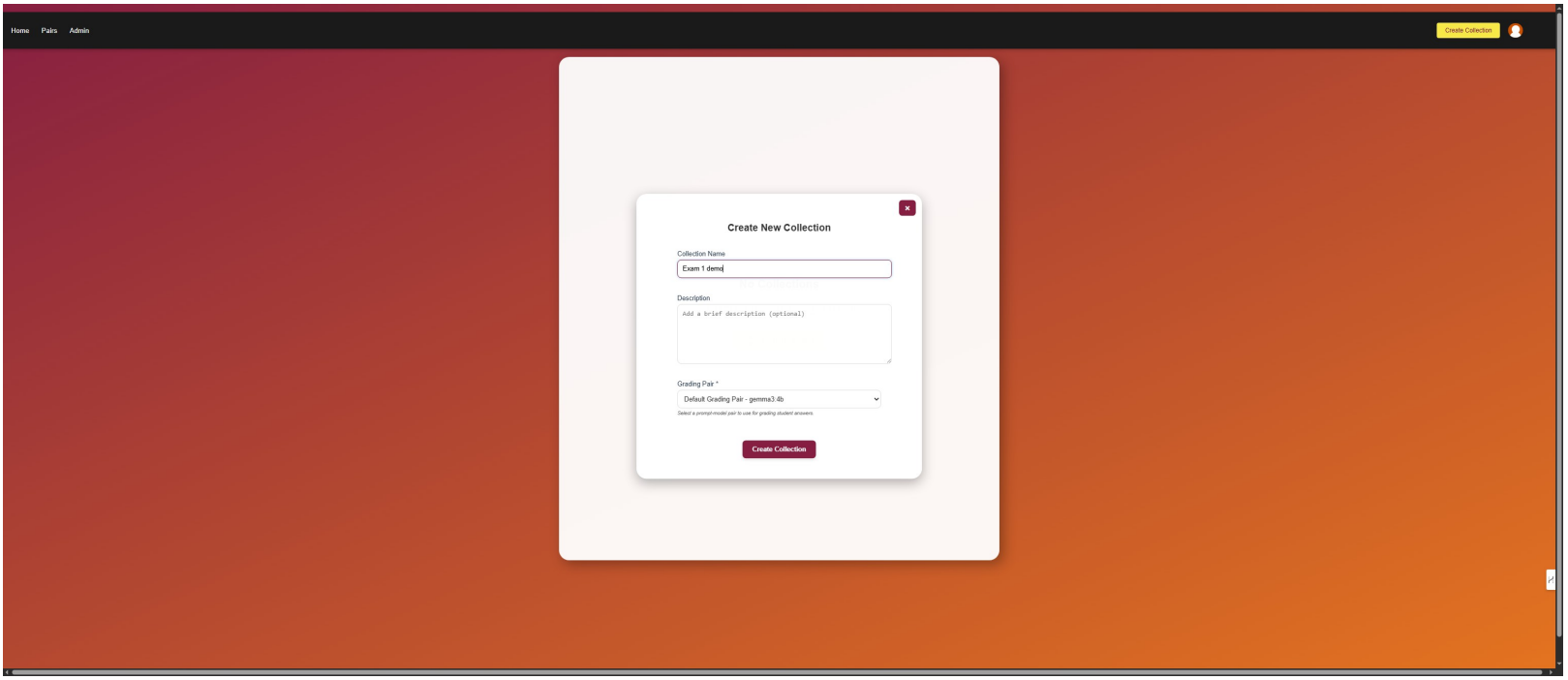

Figure 8: Collection naming - Provide a clear, descriptive name for your collection

10. For this guide, enter **"Exam 1 demo"** as the collection name
11. Add an optional description in the **"Description"** field

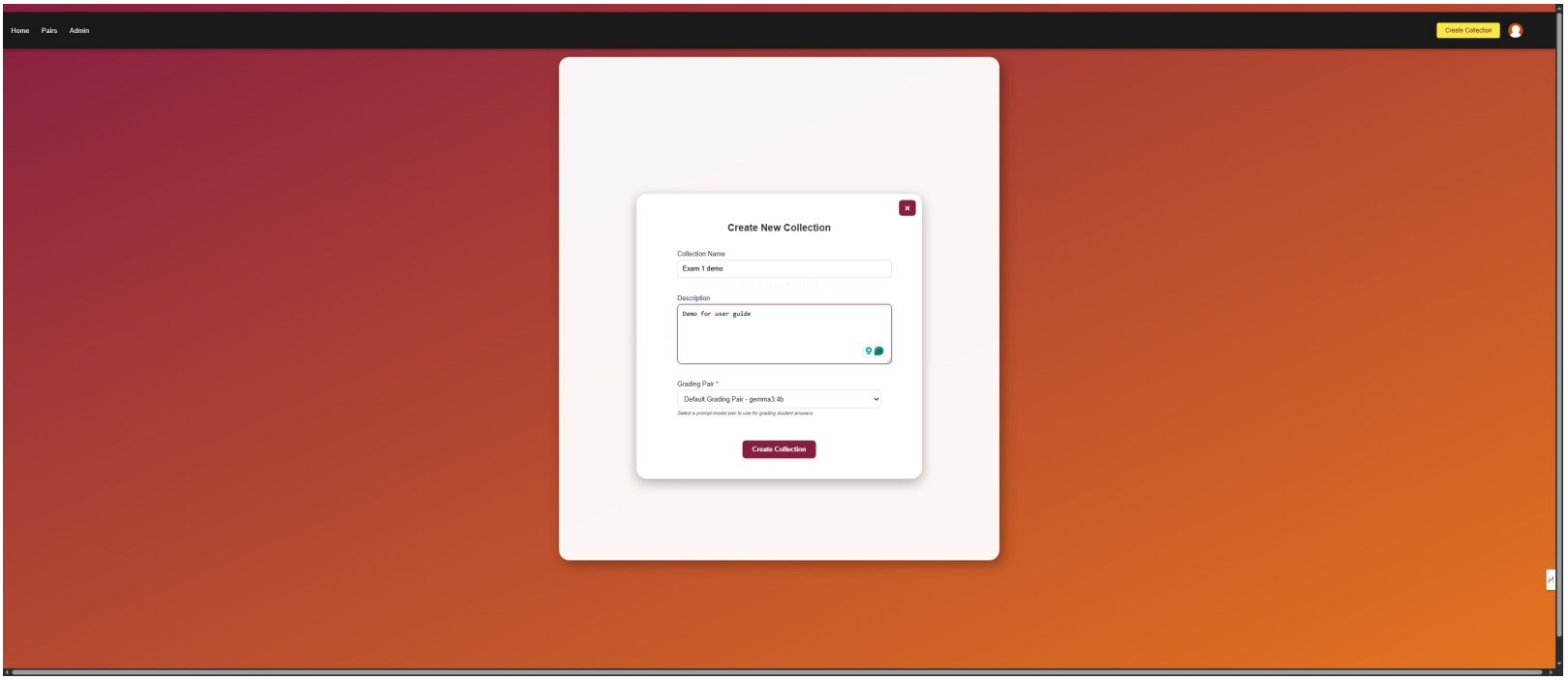

Figure 9: Adding description - Enter contextual information about the collection (optional)

12. Use the grading pair drop-down to select which model and prompt will assess student answers

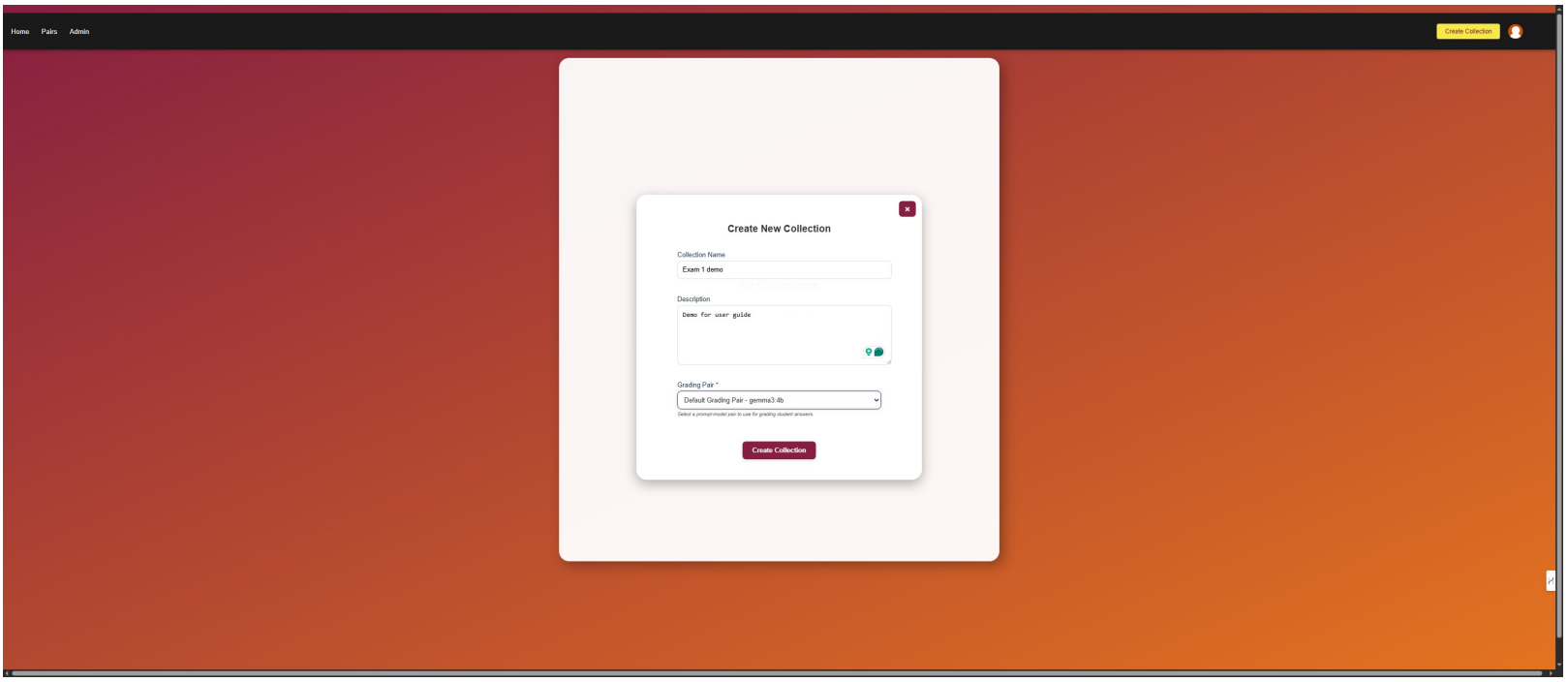

Figure 10: Grading pair selection - Choose the AI model and prompt combination for assessment

13. Click **"Create"** to finalize your collection

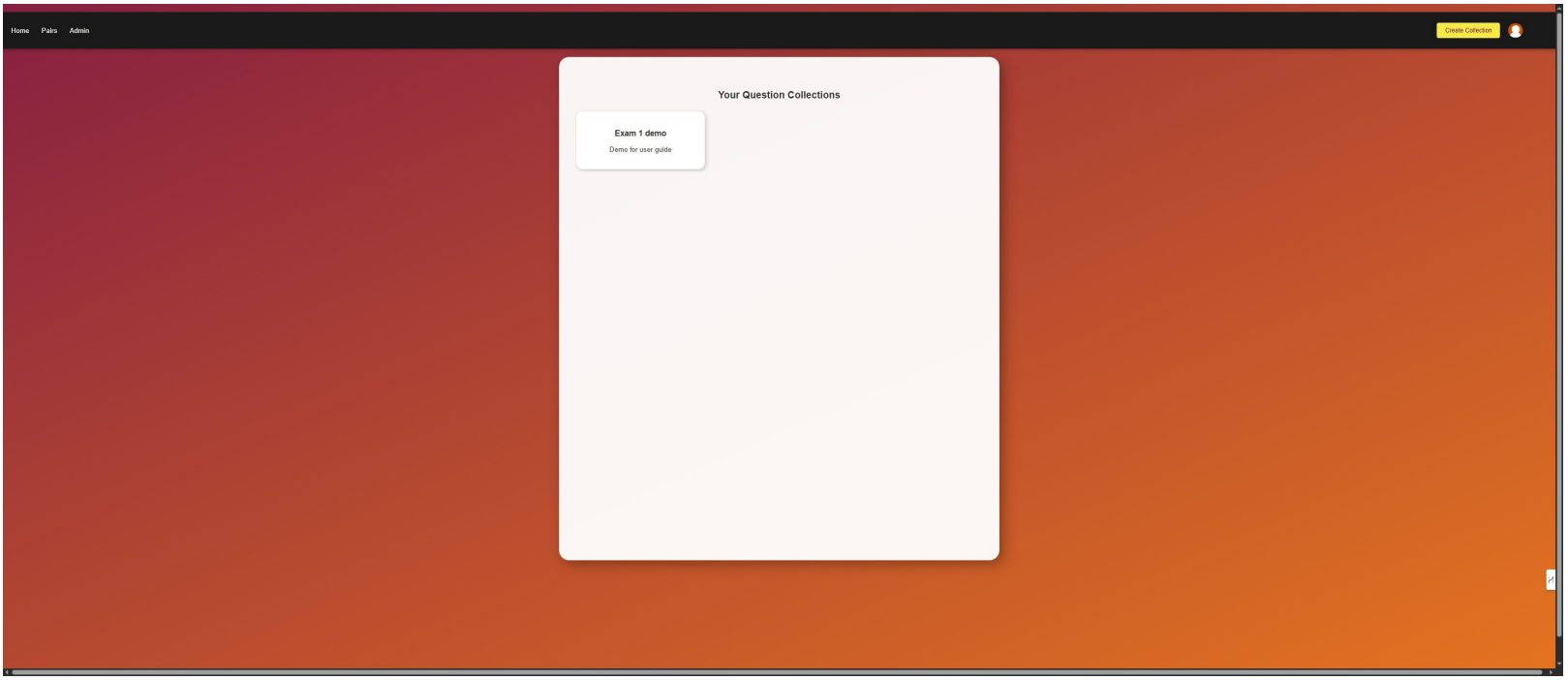

Figure 11: Collection created - Your new ”Exam 1 demo” collection is now ready for use

9.3 Uploading Questions and Student Answers

14. Click on your newly created collection to access its details

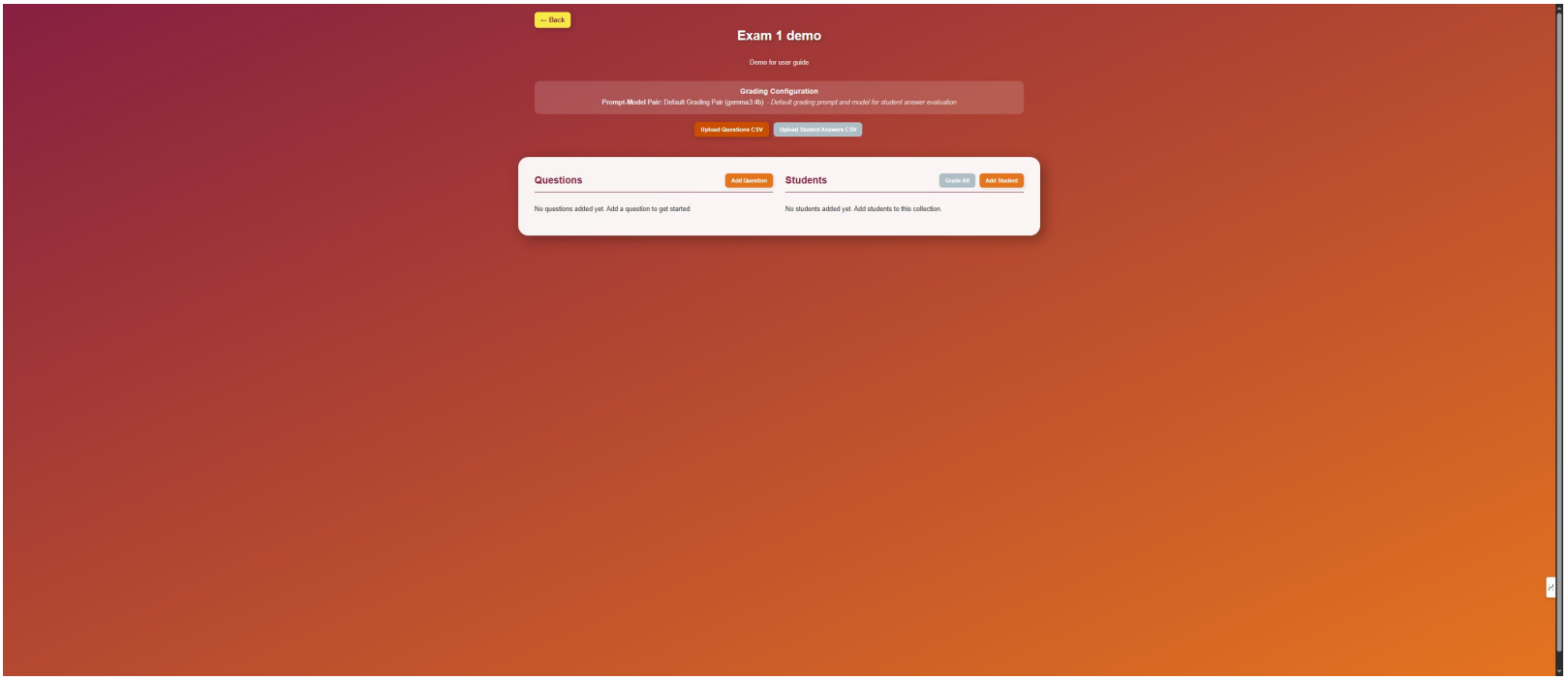

Figure 12: Collection view - Access the detailed view of your ”Exam 1 demo” collection

15. Click the **"Upload Question CSV"** button to import your assessment questions

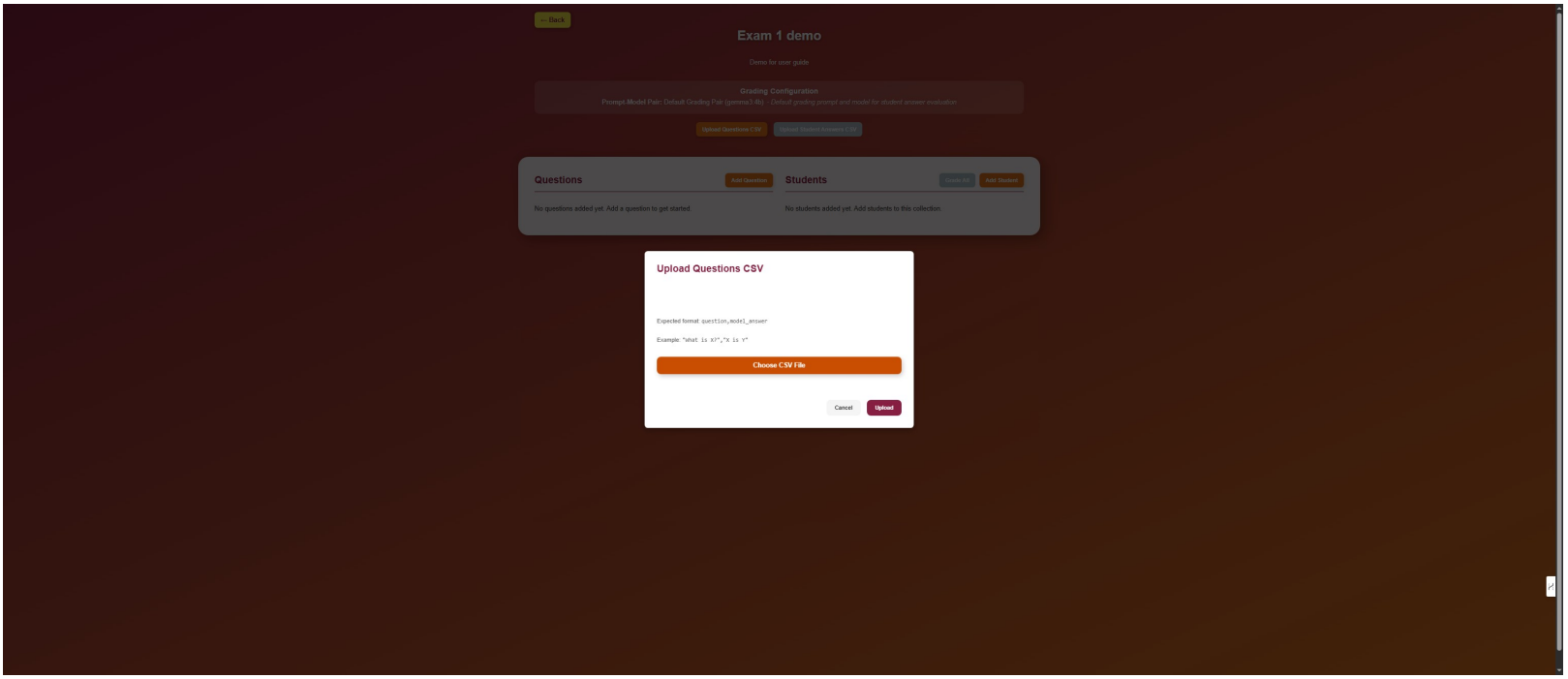

Figure 13: Question upload option - Begin the process of importing questions from a CSV file

16. Select your question CSV file from your local storage, ensuring it follows the required format

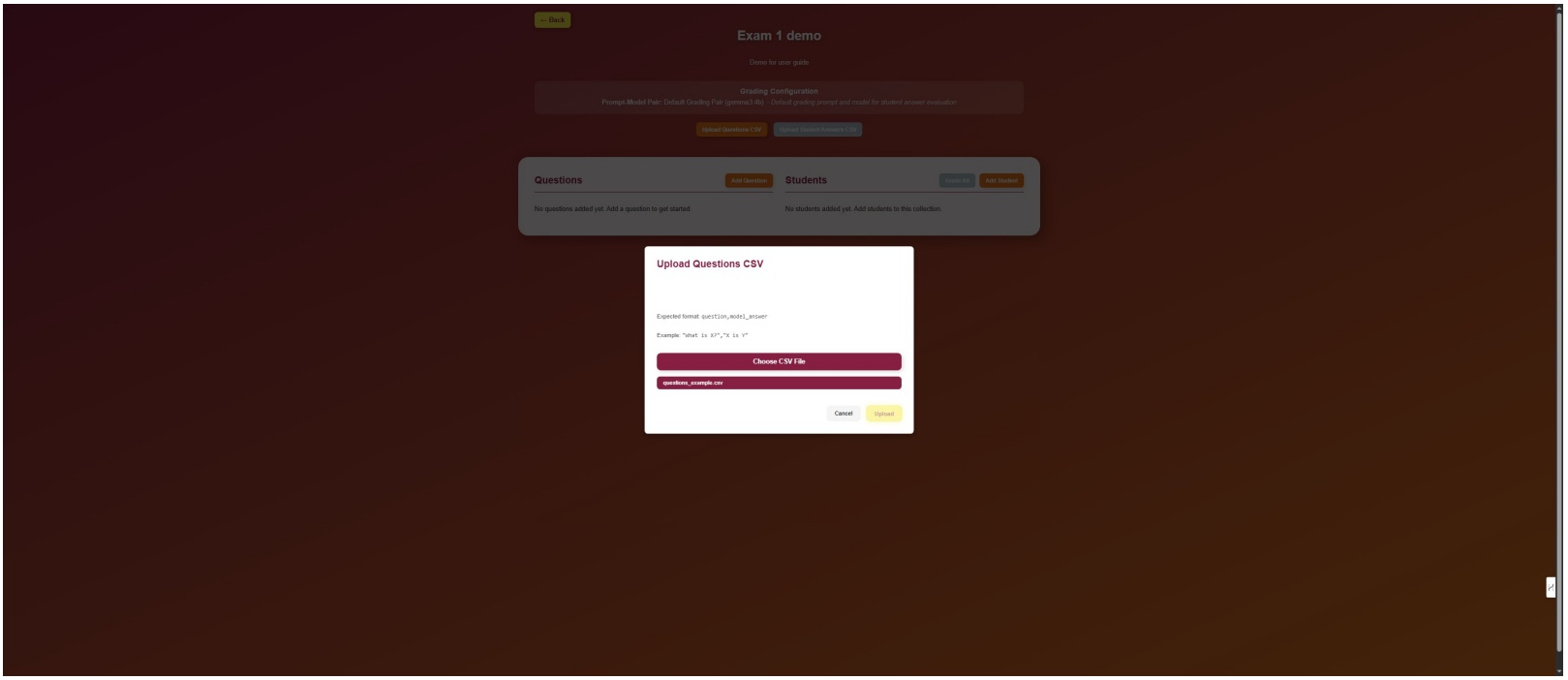

Figure 14: File selection - Choose your properly formatted question CSV file for upload

17. Click **"Upload"** to import the questions into your collection

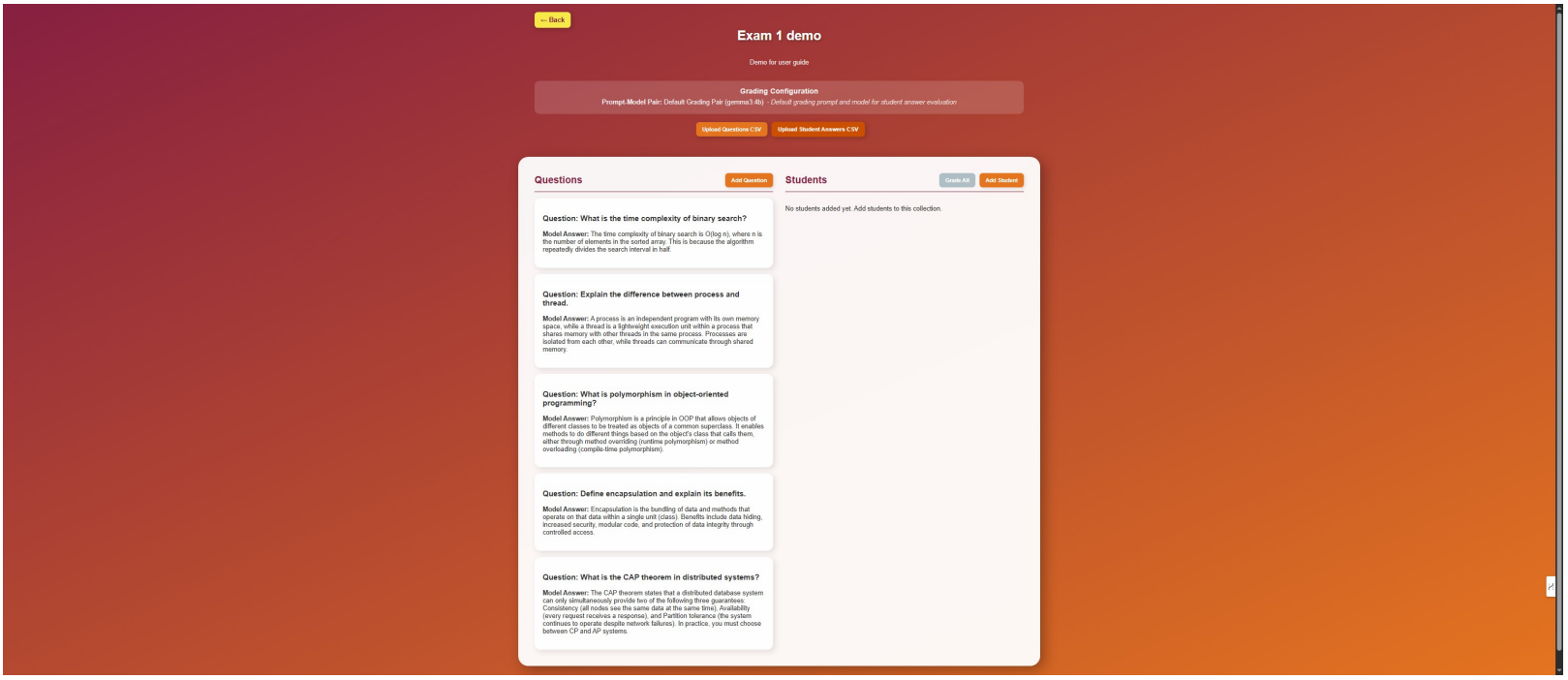

Figure 15: Questions loaded - Your assessment questions have been successfully imported

18. Now import student answers by clicking the **"Upload Student Answer CSV"** button

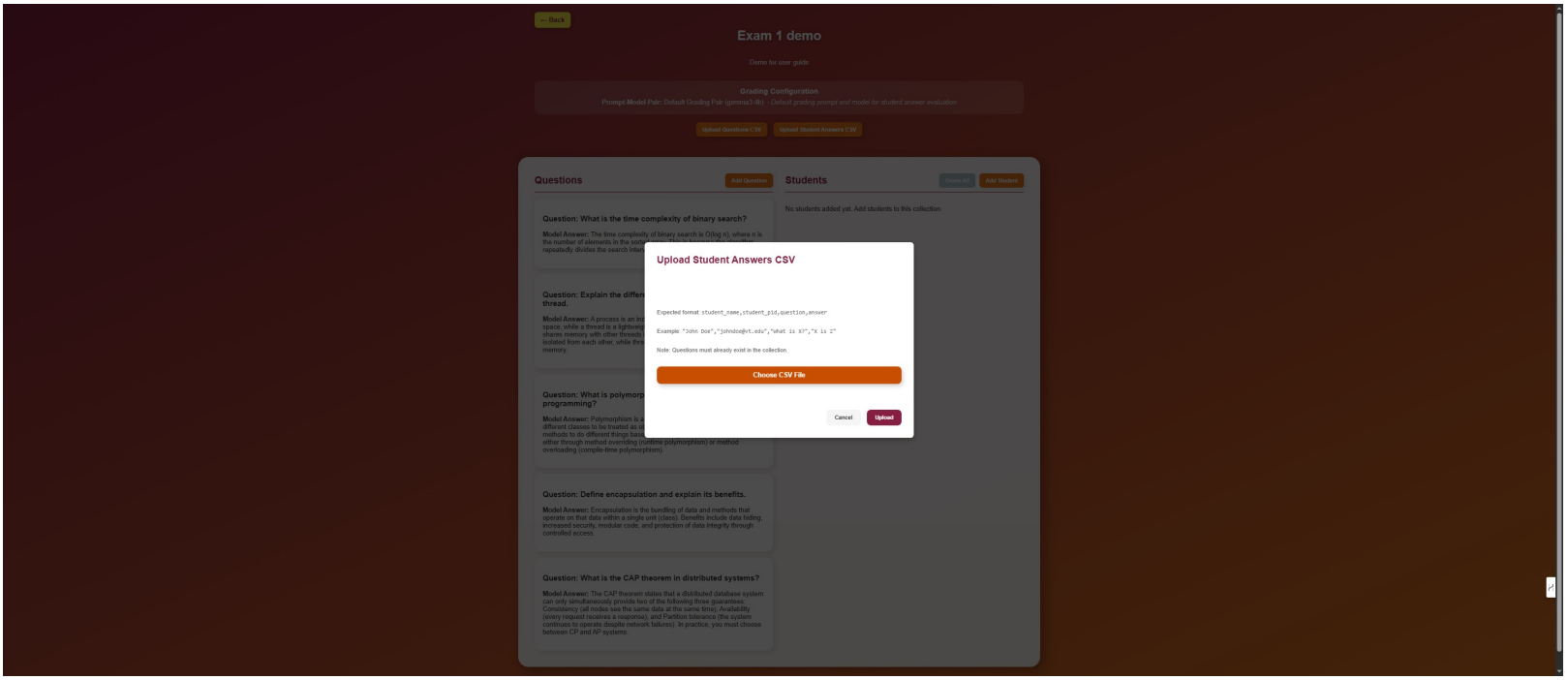

Figure 16: Student answer upload - Begin the process of importing student responses

19. Select your student answers CSV file and click **"Upload"**

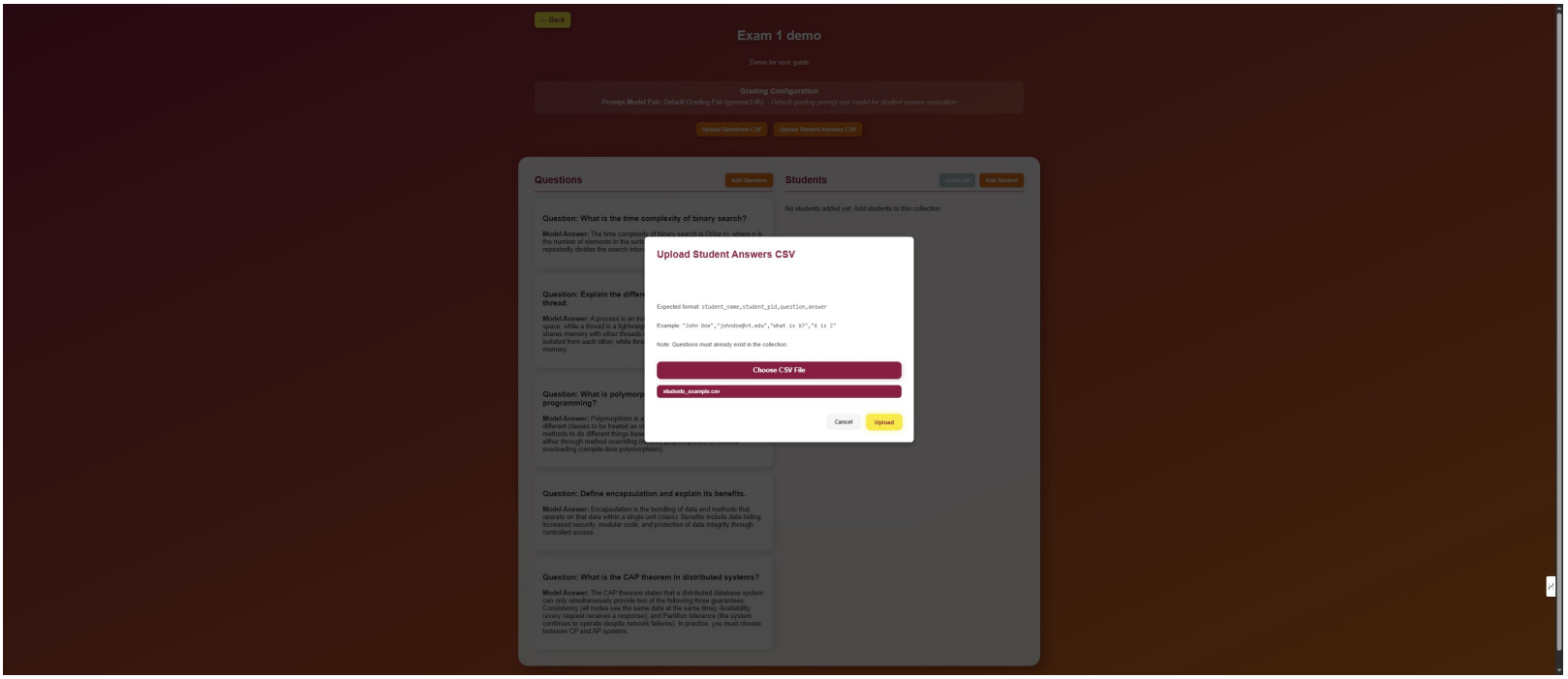

Figure 17: Student file selection - Choose your properly formatted student answers CSV file

20. Verify that student answers have been successfully imported

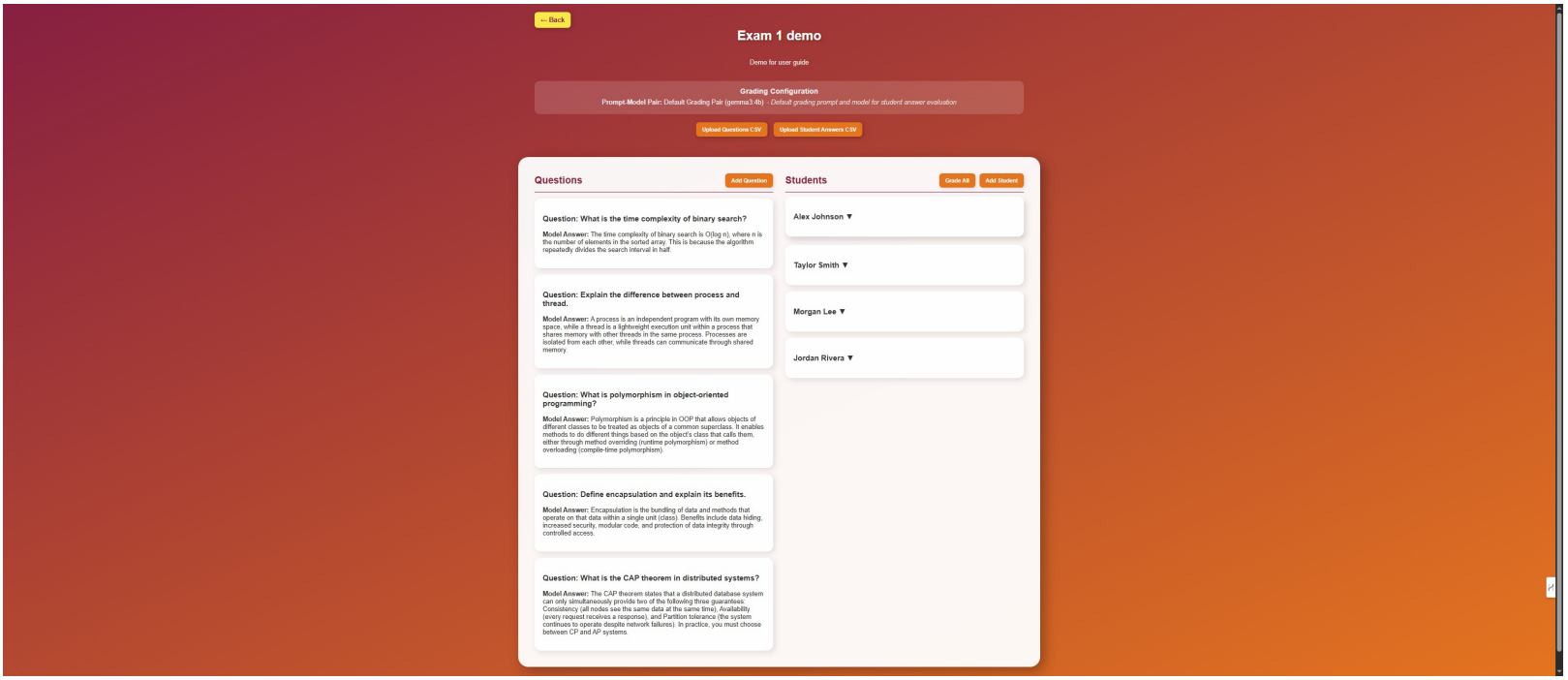

Figure 18: Student answers imported - All student responses are now ready for assessment

9.4 Grading Student Answers

21. To grade an individual answer, locate a student (for this guide, "Alex Johnson") and click the **"Grade"** button

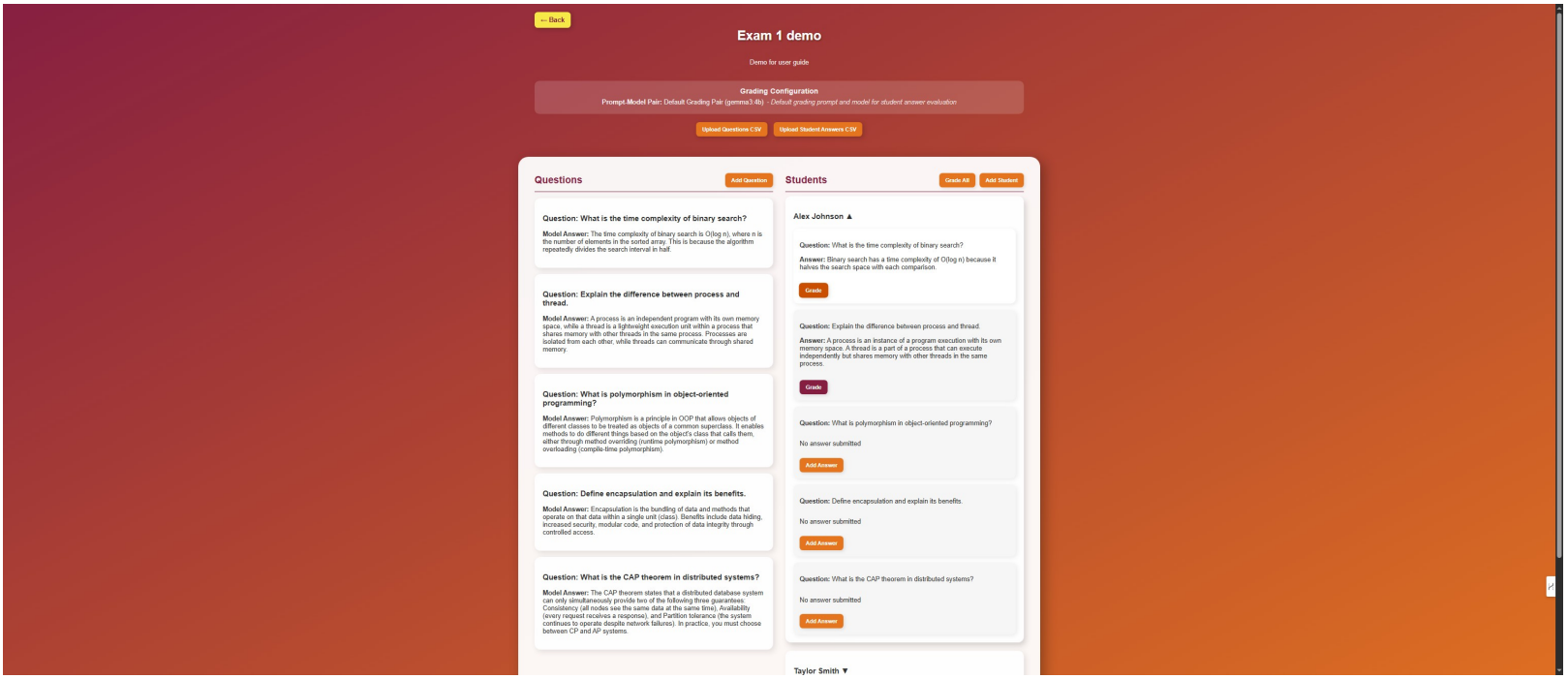

Figure 19: Grade button - Initiate the grading process for an individual student response

22. If the required AI model isn't already downloaded, the system will automatically download it

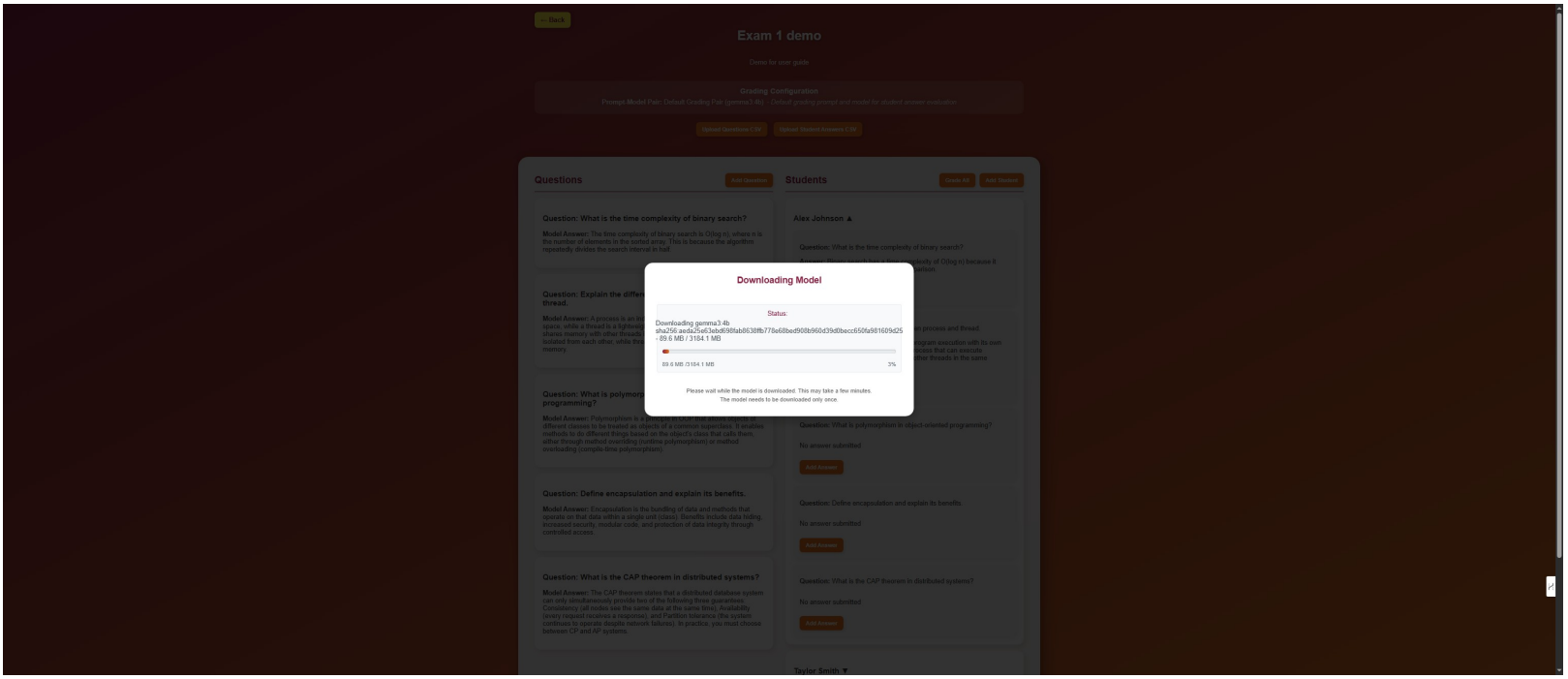

Figure 20: Model downloading - Progress indicator shows the AI model being prepared

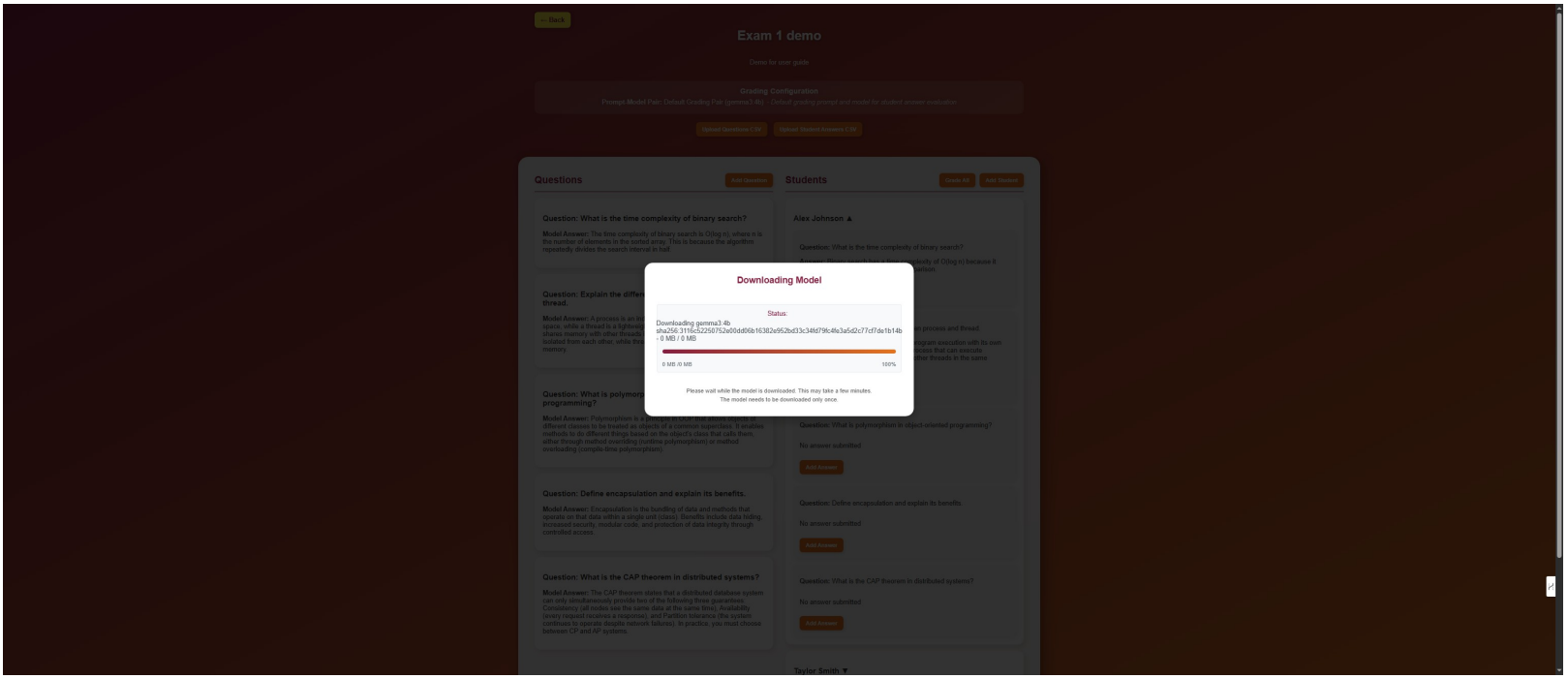

Figure 21: Download complete - The required AI model is now ready for assessment

24. The system will process and display the student's grade (refresh page if not shown)

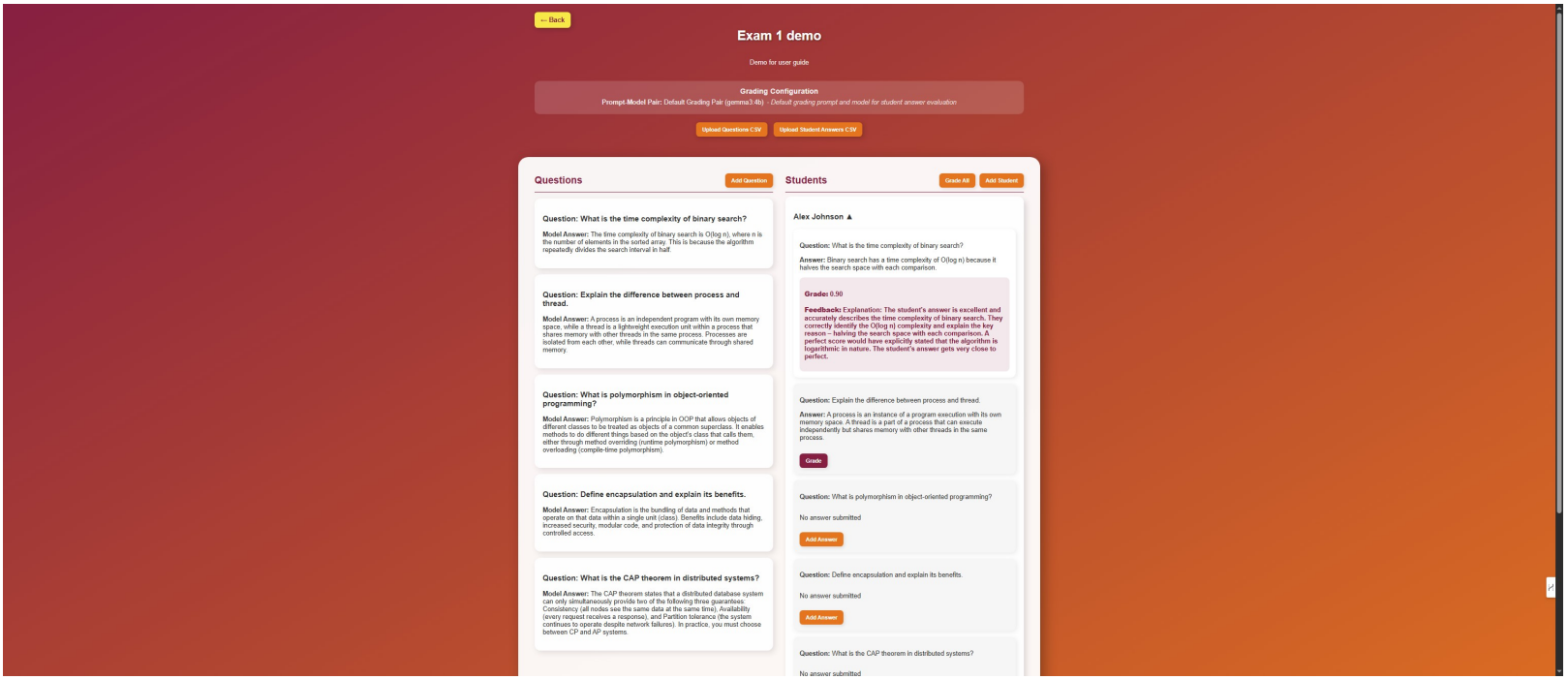

Figure 22: Grading result - Automated assessment displayed below the student's answer

25. To grade all student answers simultaneously, use the **"Grade All"** button at the top of the page

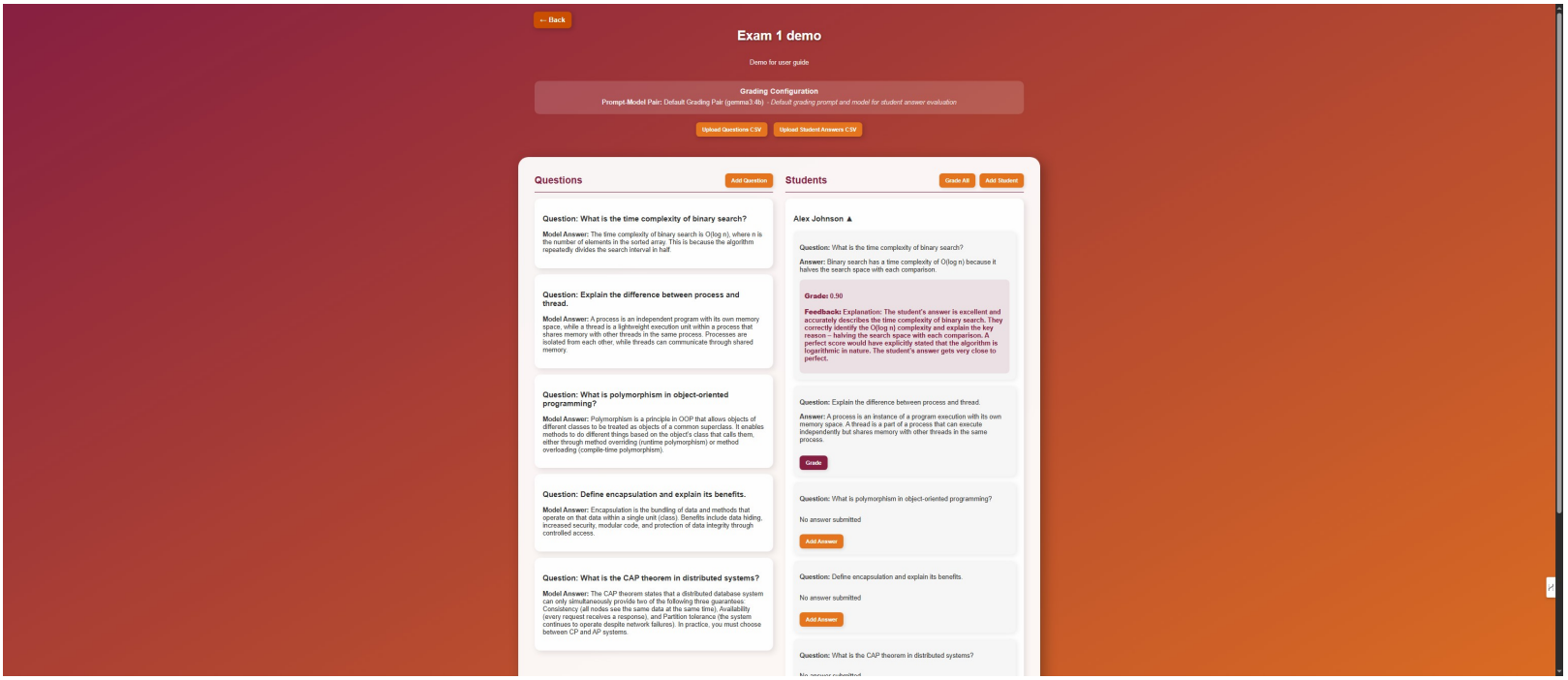

Figure 23: Grade All option - Process all student responses with a single click

26. Return by clicking the **"← back"** button or **"Home"** in the navigation bar

9.5 Admin Panel: Managing Models

27. From the home page, access administrative functions by clicking the **"Admin"** button

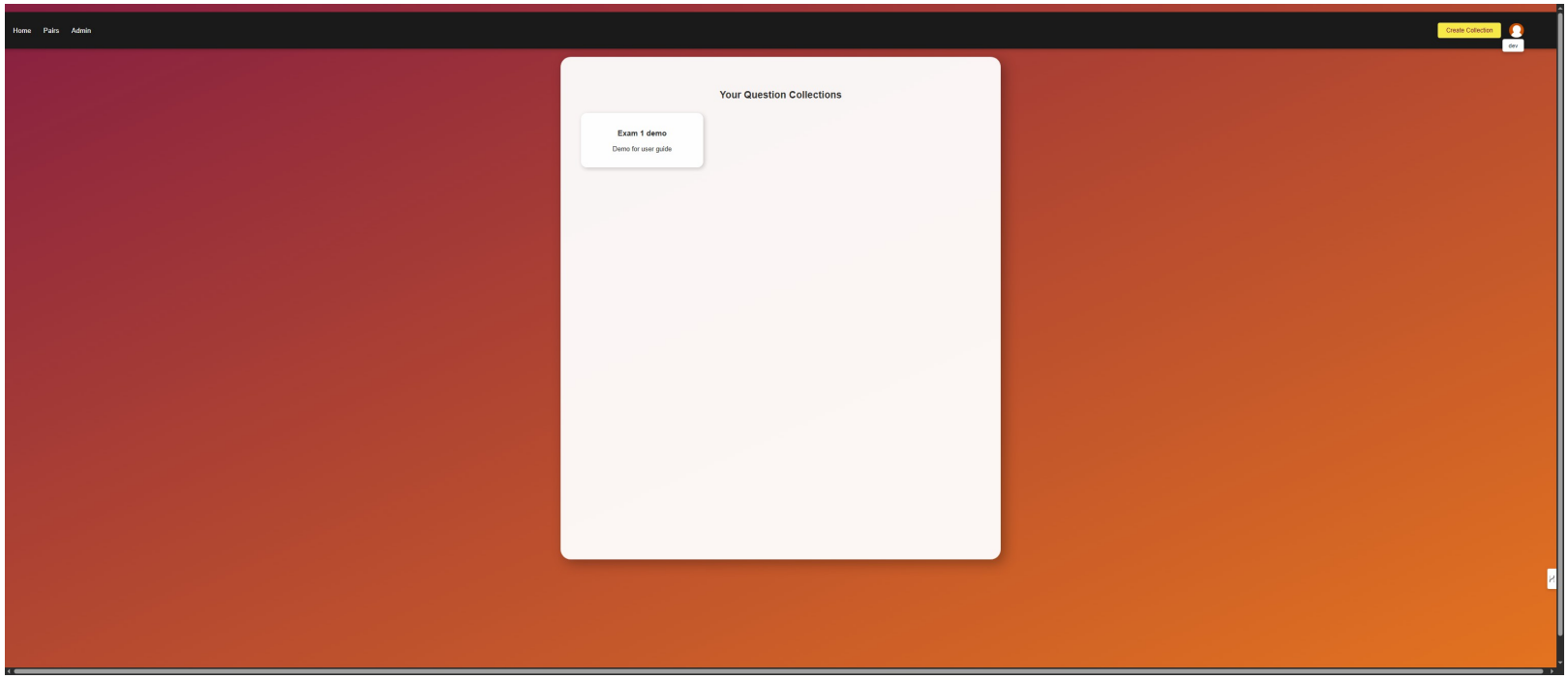

Figure 24: Home page navigation - Locate the Admin button in the navigation bar

28. Click the **"Admin"** button in the navigation bar to open the administrator panel

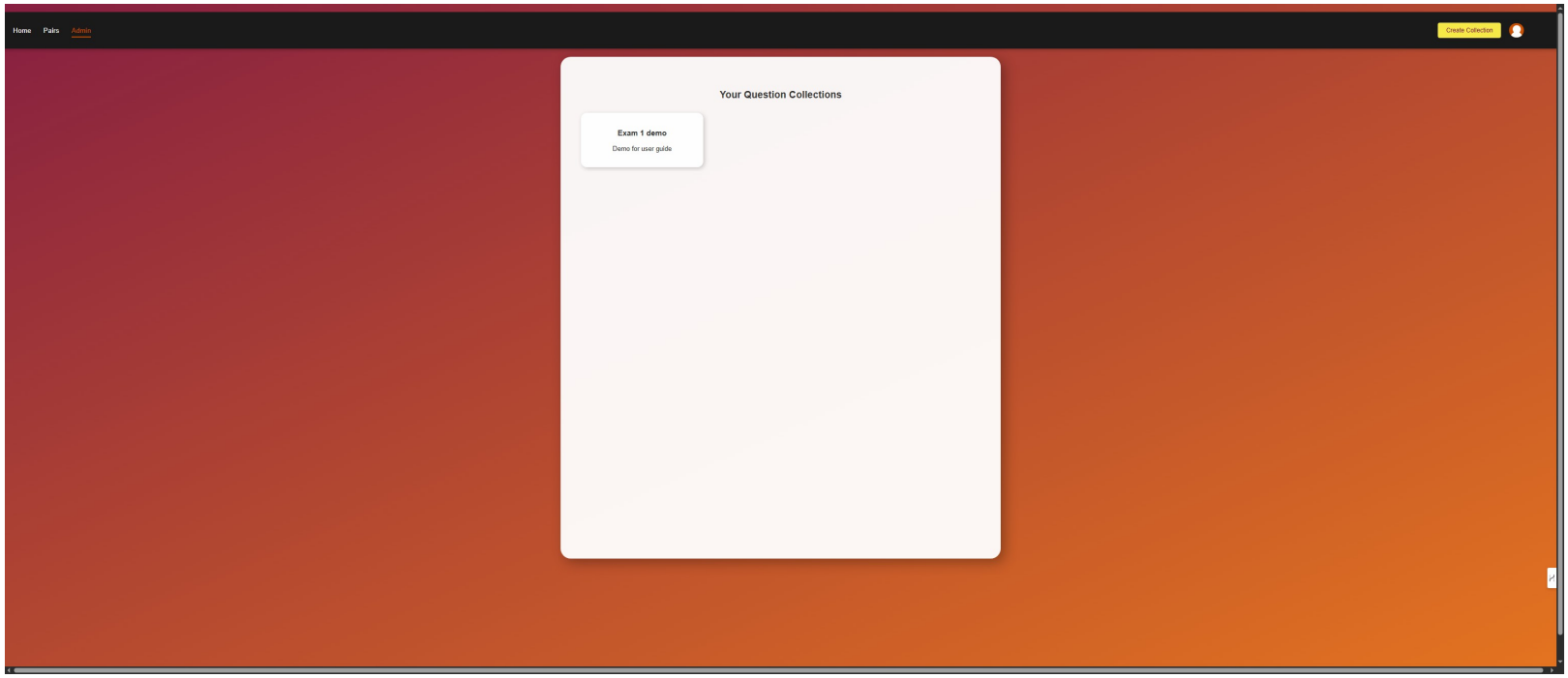

Figure 25: Admin button - Access advanced system configuration options

Figure 26: Admin Panel - Central interface for system configuration and management

30. To add a new AI model, enter its name in the model field (model names are available at https://ollama.com/library)

Figure 27: Model download - Enter ”tinyllama” to install the TinyLlama AI model

31. Click the **”Pull model”** button to begin downloading the selected model

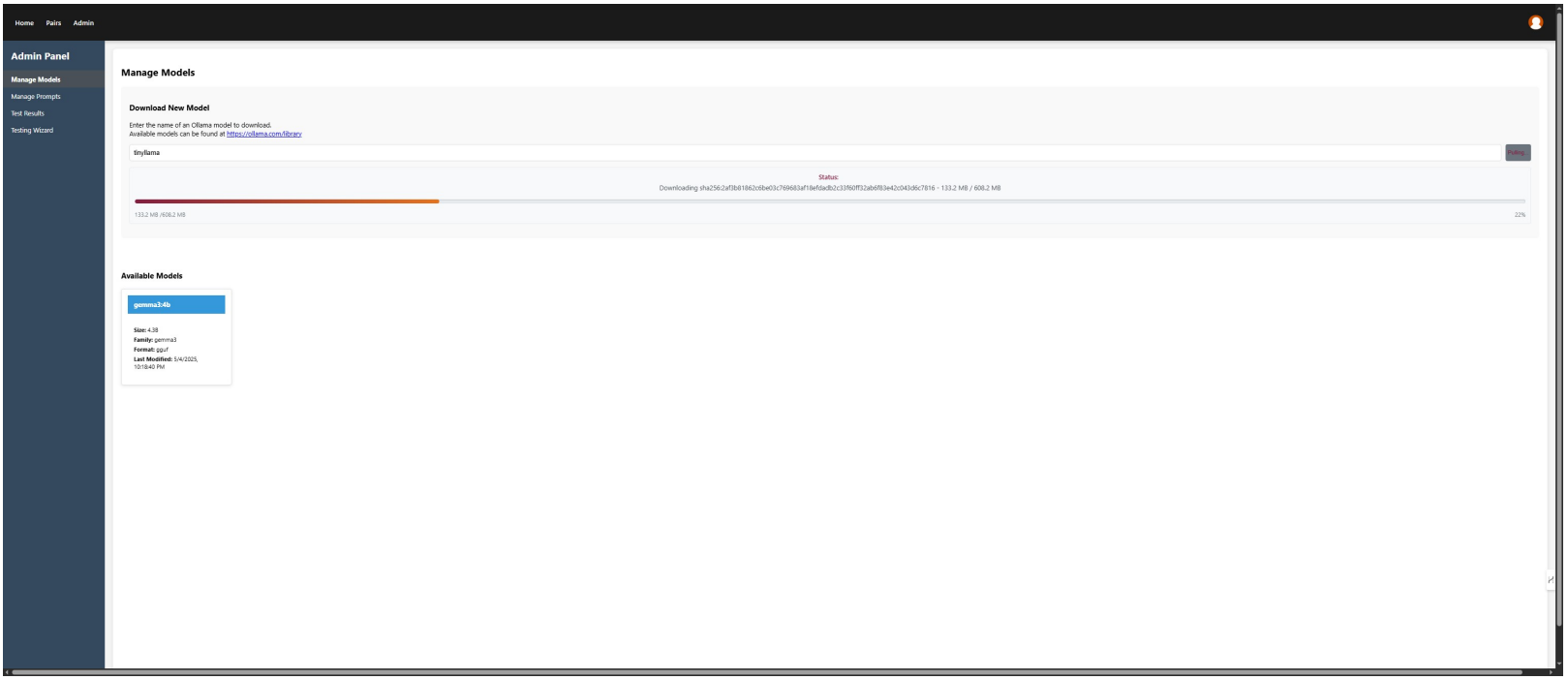

Figure 28: Download progress - System shows the download and installation status

32. Wait for the download and installation process to complete

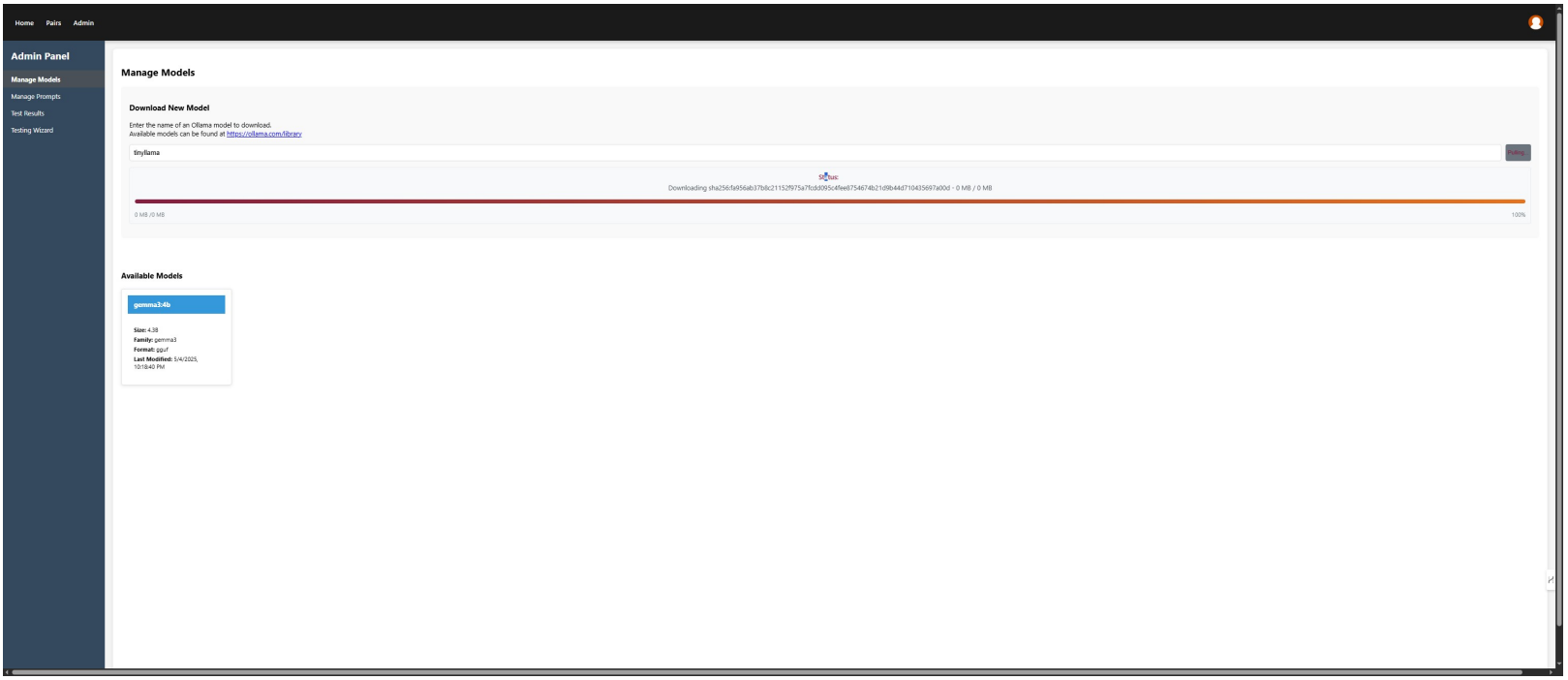

Figure 29: Installation complete - The TinyLlama model is now ready for use

33. View all installed models in the **"Available Models"** section of the Admin Panel

Figure 30: Available models - List of all AI models currently installed (tinyllama and gemma3:4b)

9.6 Admin Panel: Managing Prompts

34. Access prompt management by clicking the **"Manage Prompts"** button in the Admin Panel

Figure 31: Manage Prompts option - Access tools for creating and editing assessment prompts

35. The Manage Prompts page allows you to create or modify prompts for testing and assessment

Figure 32: Prompts management - Overview of existing prompts and creation options

36. Click the **"Create New Prompt"** button to begin creating a custom assessment prompt

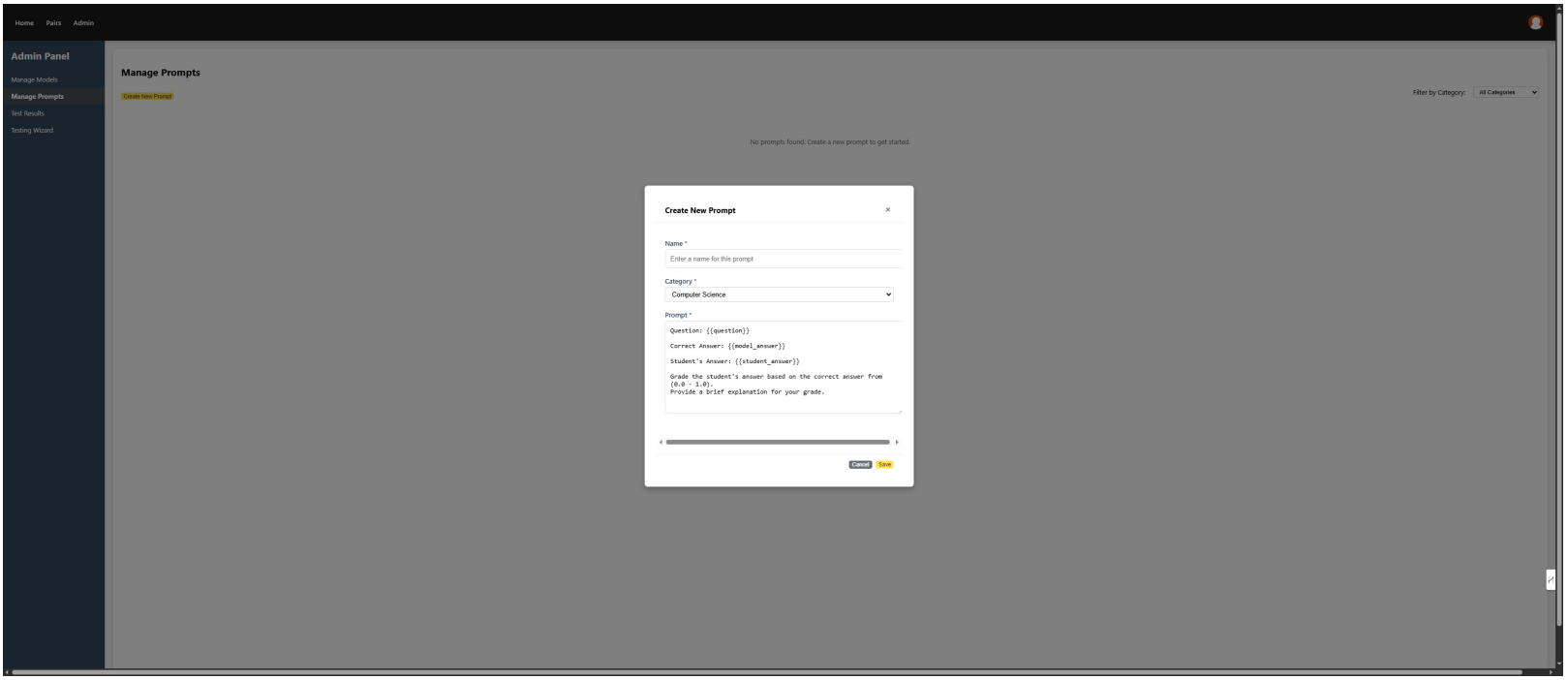

Figure 33: Prompt creation - Interface for designing a new assessment prompt

37. Enter a descriptive name for your prompt (for this guide: **"Demo User Guide"**)

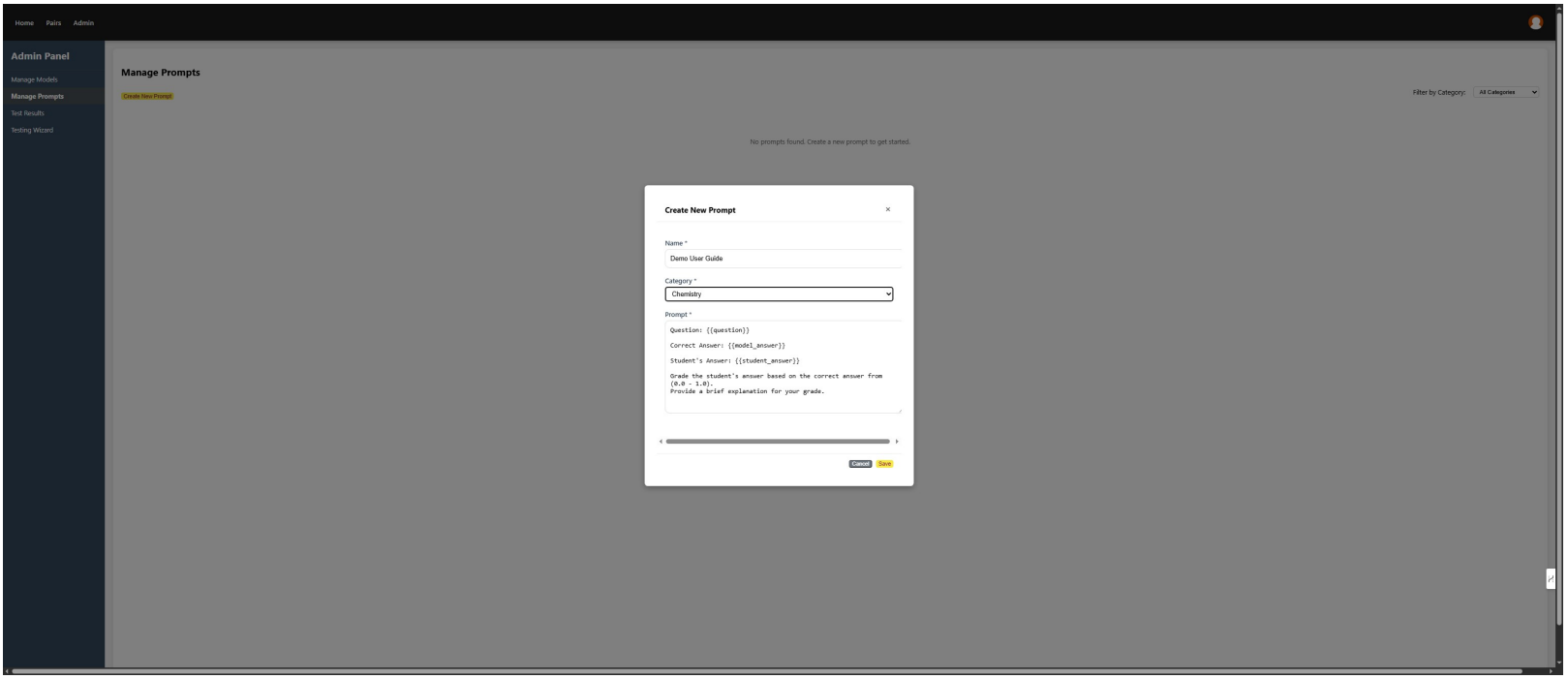

Figure 34: Prompt details - Enter the required information for your new prompt

38. Select an appropriate category for your prompt (for this guide: **"Chemistry"**)

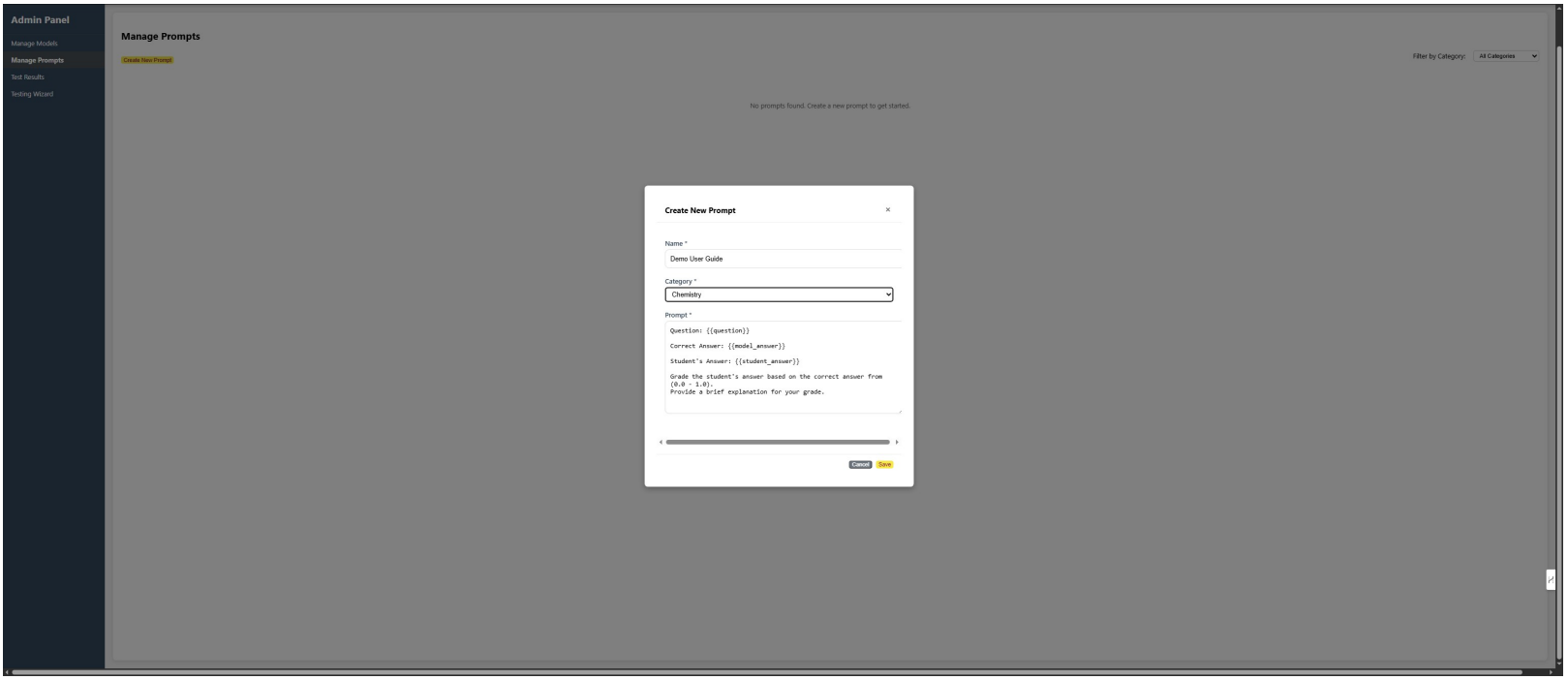

Figure 35: Category selection - Choose "Chemistry" for chemistry-related assessment questions

39. Enter the prompt text that will guide the AI in assessing student answers

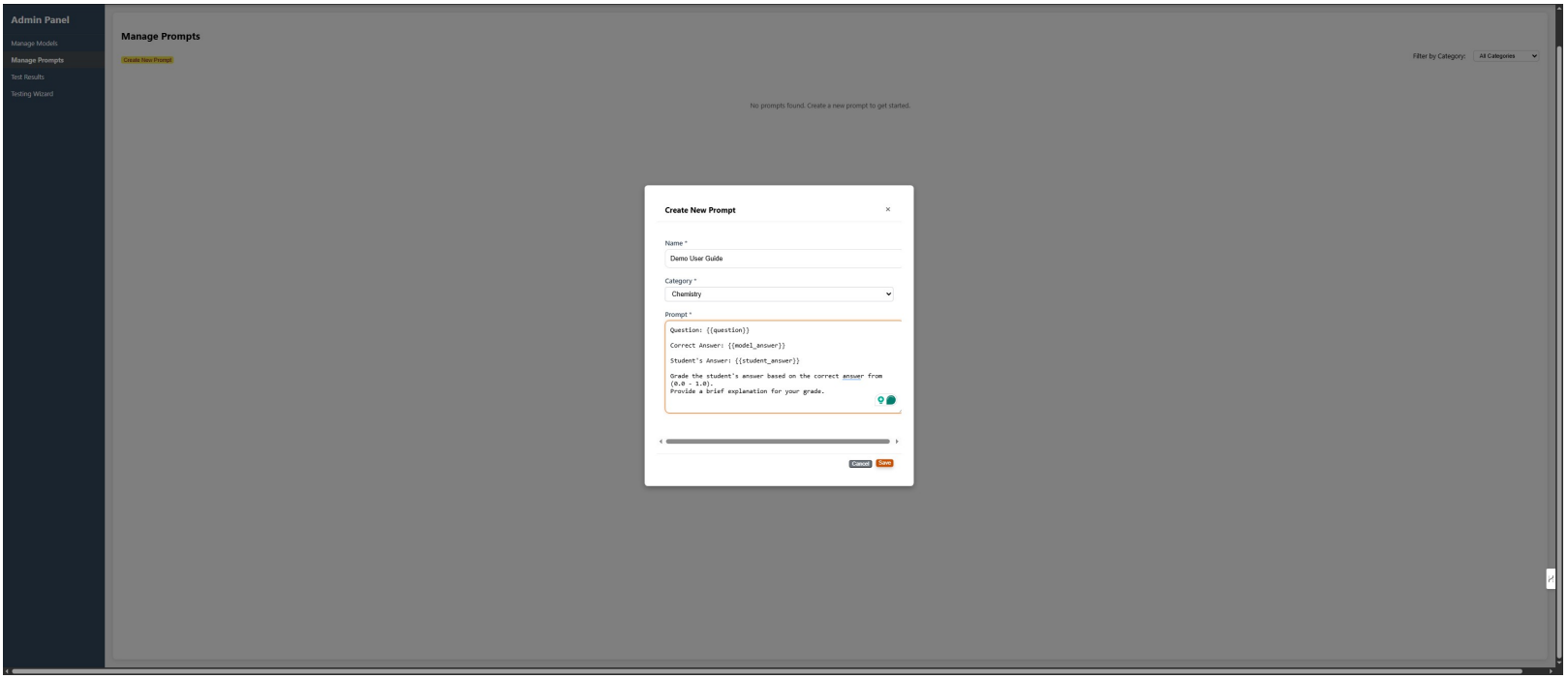

Figure 36: Prompt configuration - Define the instructions for the AI assessment process

40. Ensure your prompt maintains the required format for referencing questions, correct answers, and student responses

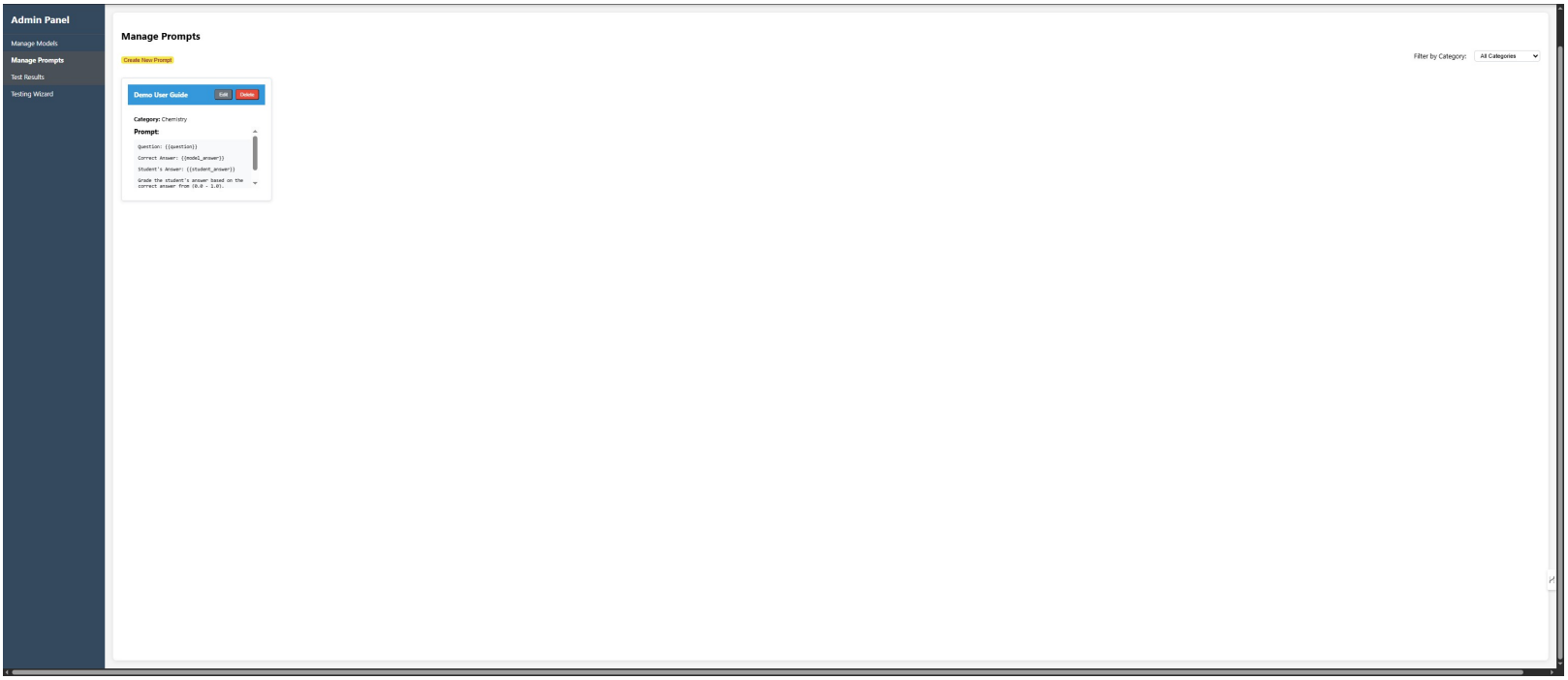

Figure 37: Prompt created - Your new prompt is now available for testing and use

9.7 Admin Panel: Testing Prompts and Models

41. Navigate to the Test Results page and click **"Create New Test"** to evaluate your prompt-model combination

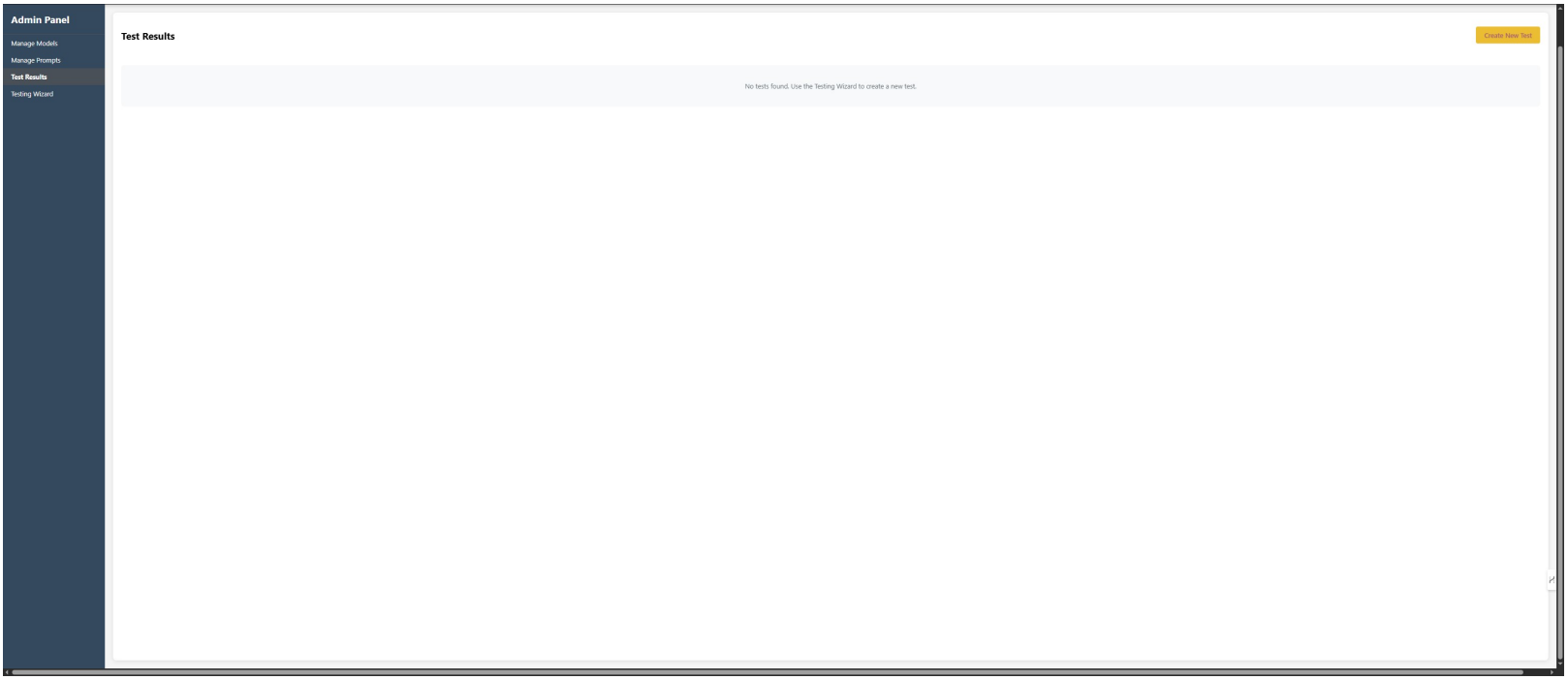

Figure 38: Test Results page - Starting point for creating and viewing assessment tests

42. The Testing Wizard will guide you through setting up an assessment test

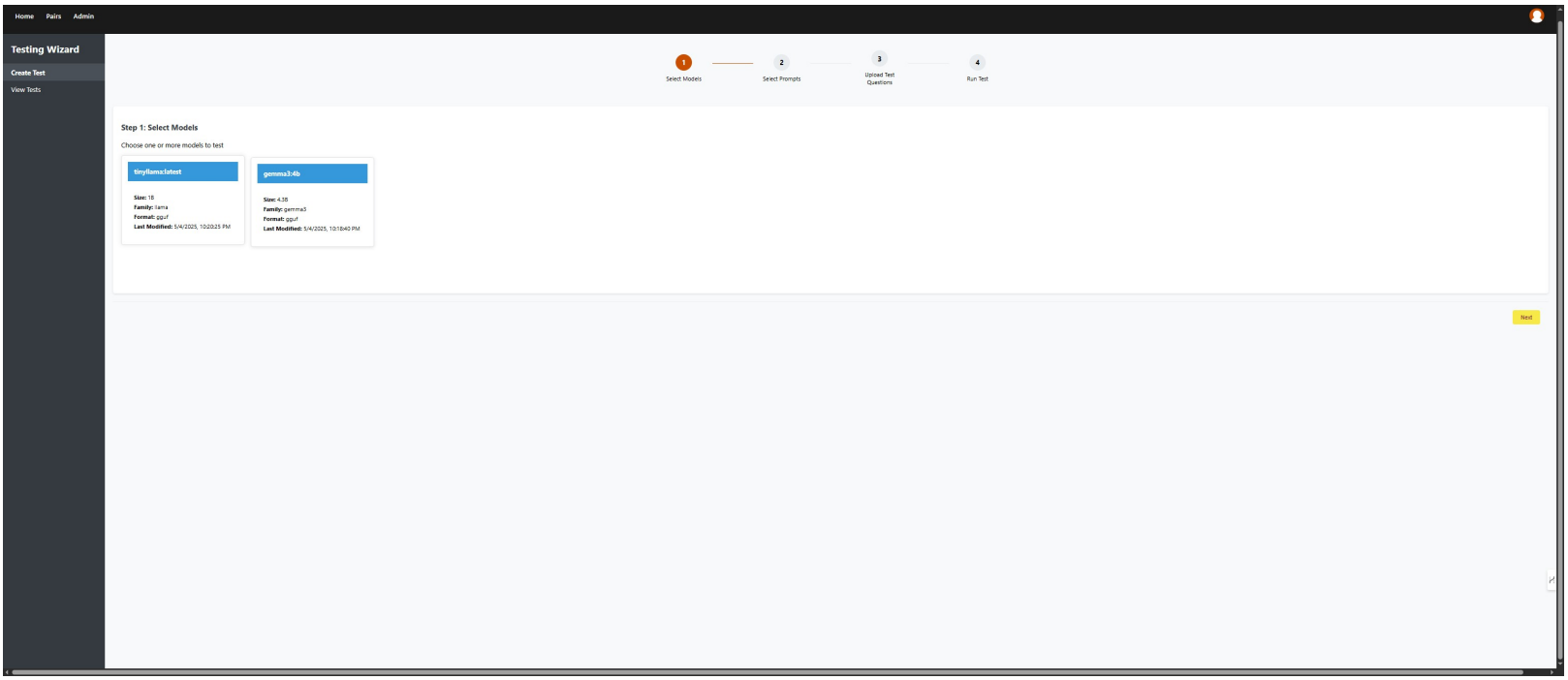

Figure 39: Testing Wizard - Step-by-step interface for configuring assessment tests

43. Select the AI model you wish to test (for this guide: **"tinyllama"**)

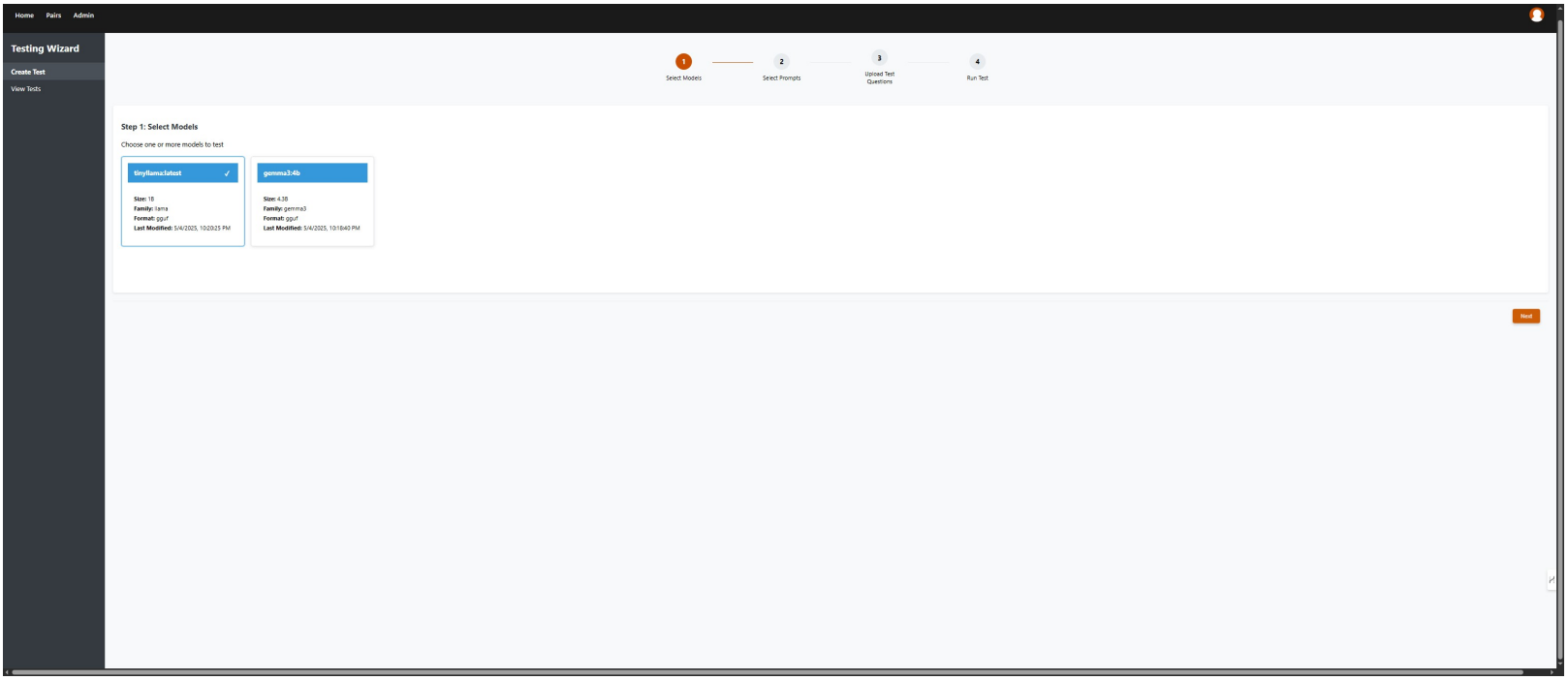

Figure 40: Model selection - Choose the AI model to evaluate in this test

44. Select the prompt you created earlier for testing with the chosen model

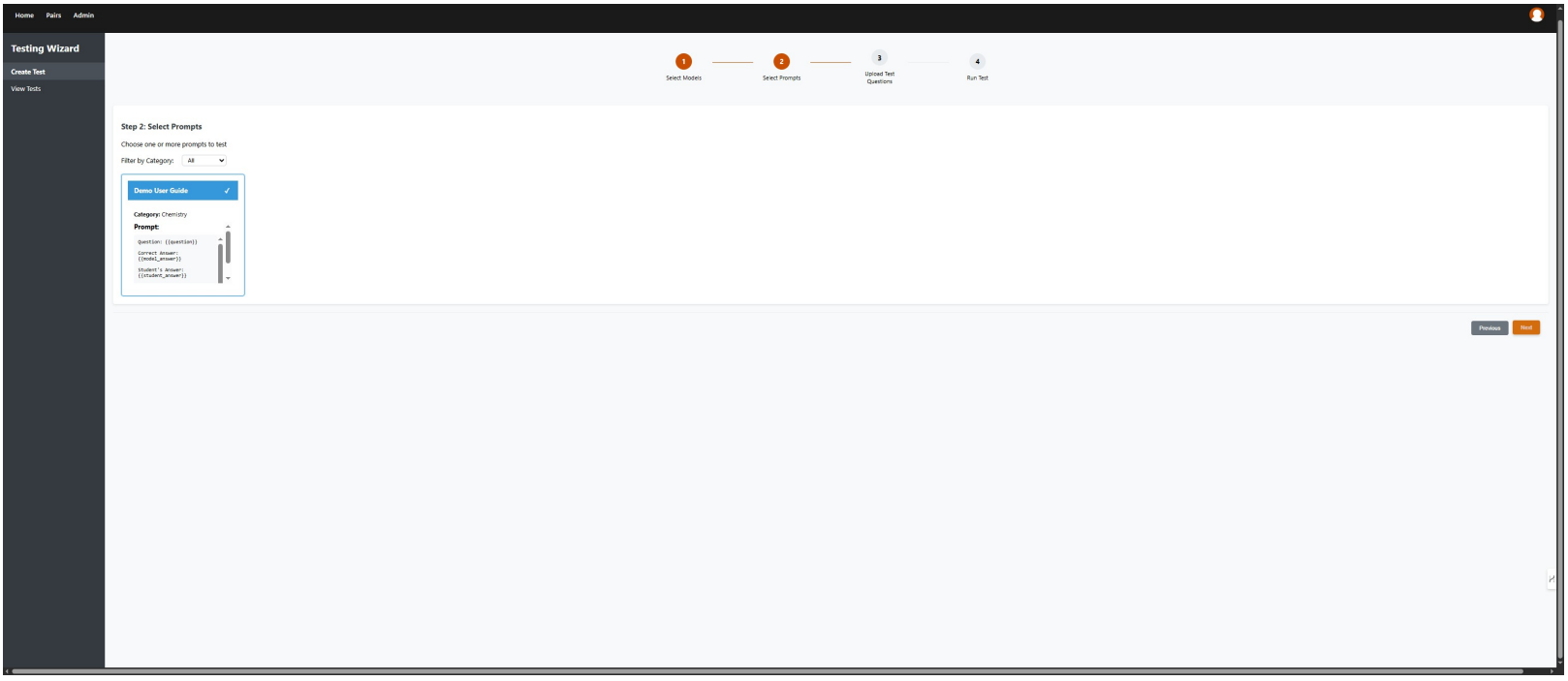

Figure 41: Prompt selection - Choose which assessment prompt to evaluate

45. Enter a name for your test and prepare to upload test questions

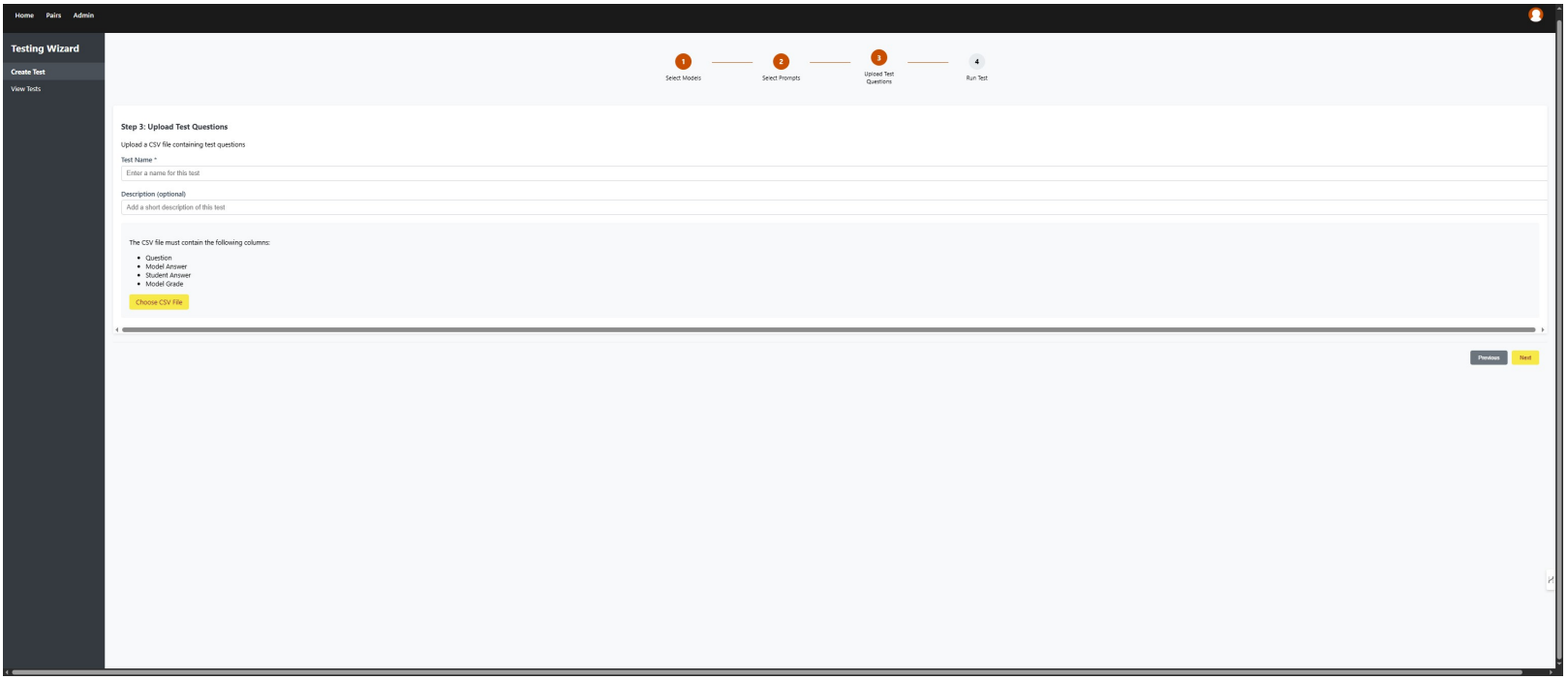

Figure 42: Test configuration - Set up basic parameters for your assessment test

46. Name your test (for this guide: **"UserGuide Demo"**) and add an optional description

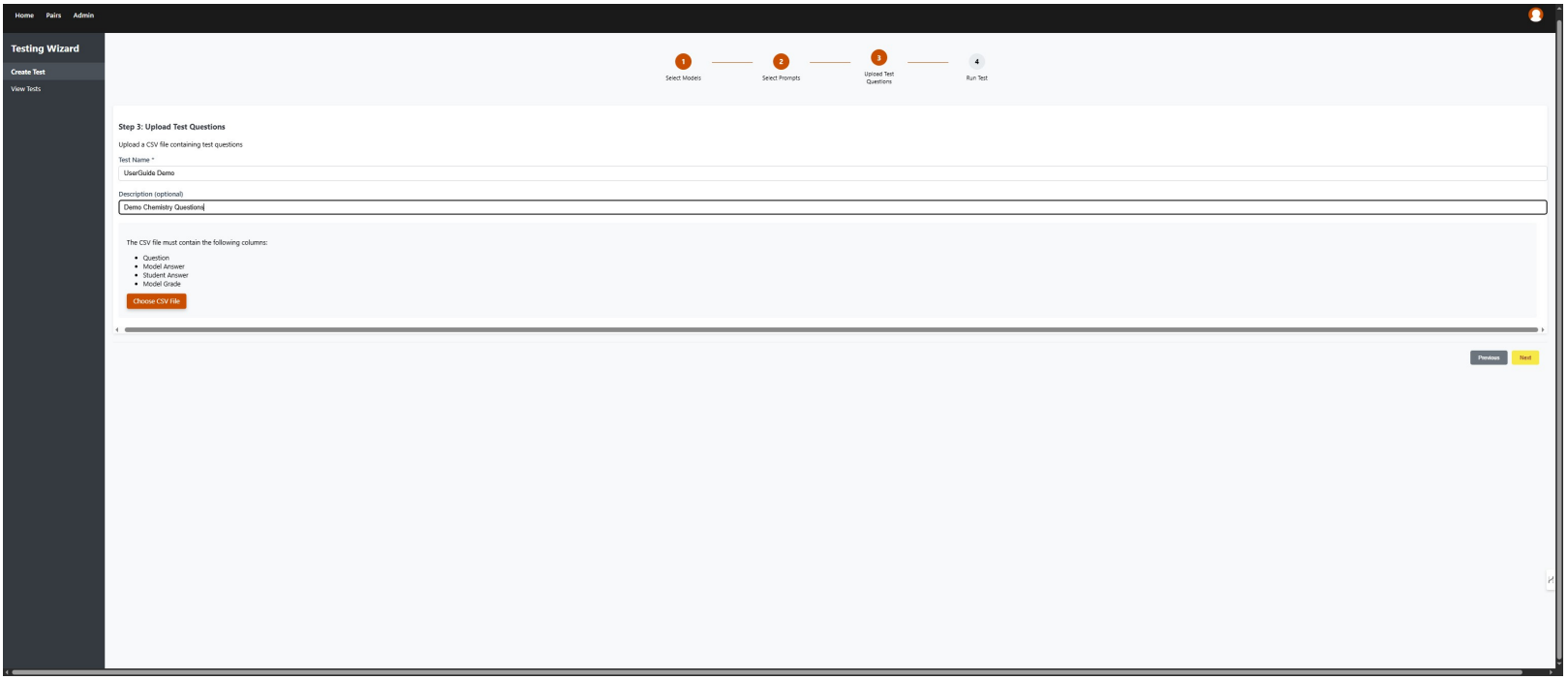

Figure 43: Test naming - Provide a clear identifier for your test configuration

47. Upload a CSV file containing questions for testing, following the required format

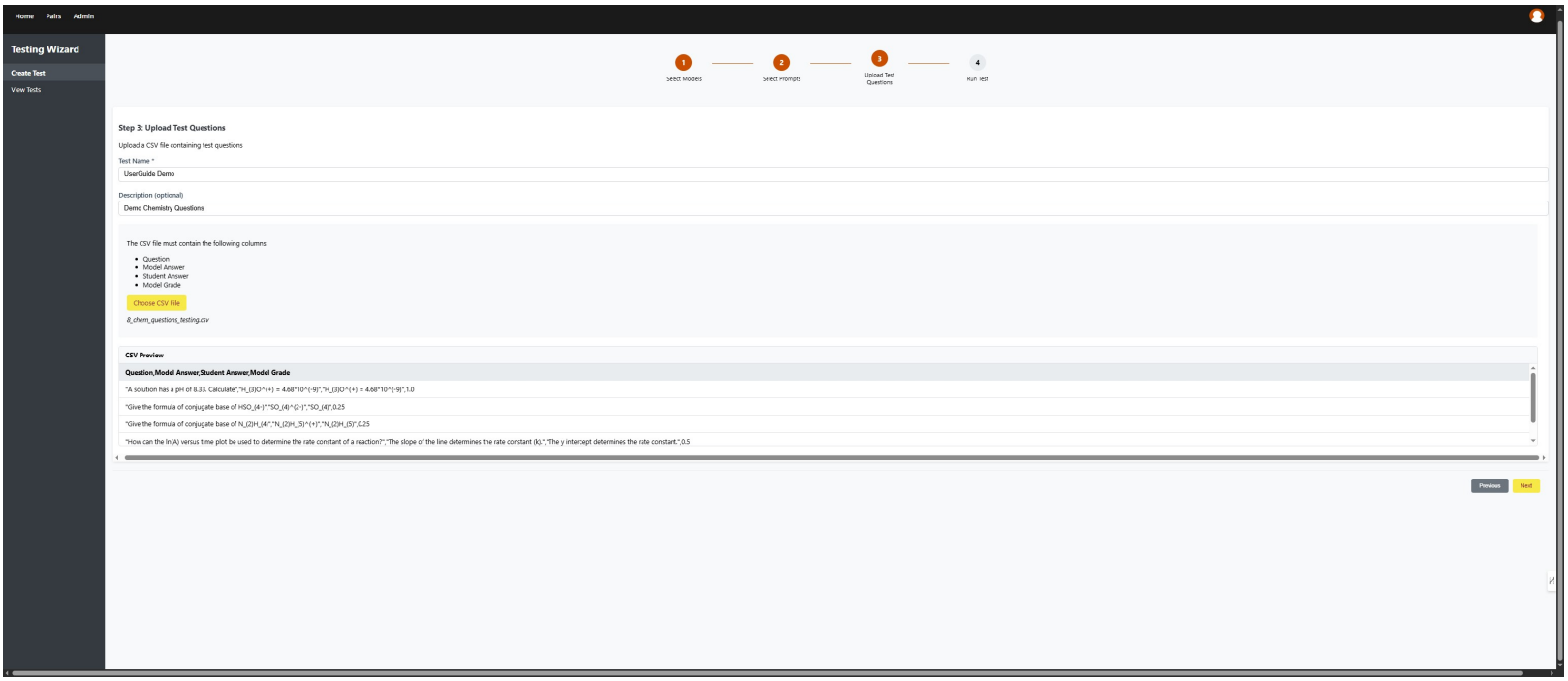

Figure 44: CSV upload - Select a properly formatted question file for testing

48. Ensure your CSV follows the specified format with quotes around string values (see example files for reference)

49. After uploading your CSV, click **"Next"** and then click the **"Run"** button to execute the test

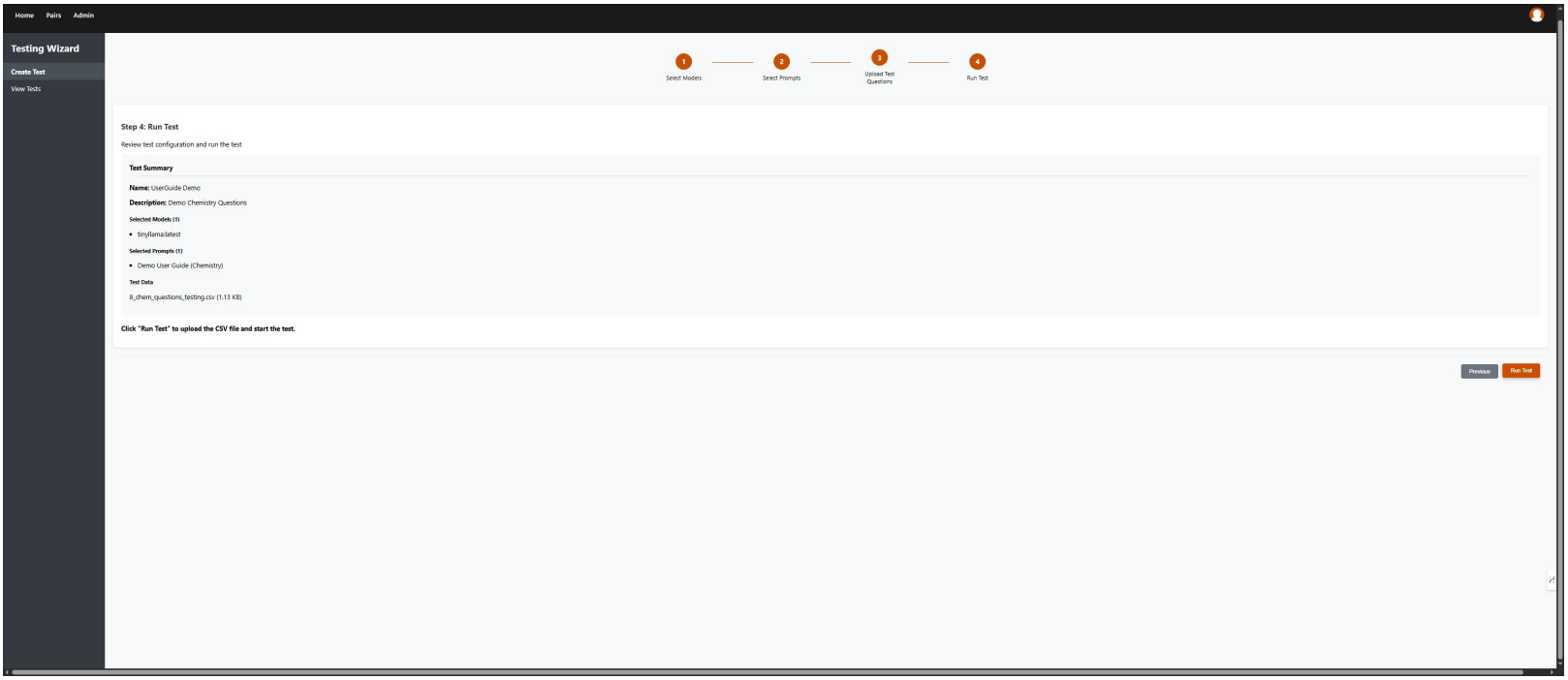

Figure 45: Test execution - Initiate the assessment test with the Run button

50. Monitor the test progress as the system evaluates the model-prompt combination

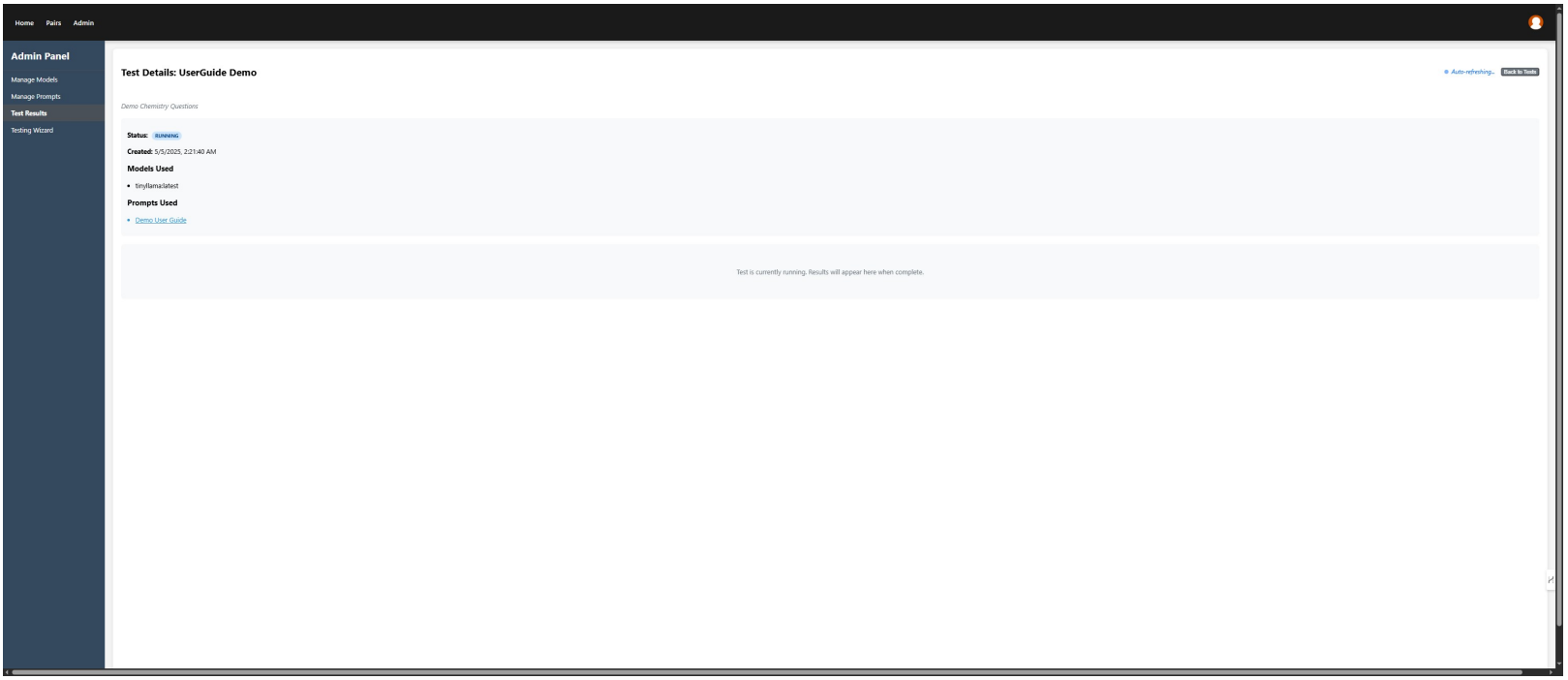

Figure 46: Test in progress - System shows status as the test executes

51. Review the detailed test results when the assessment is complete

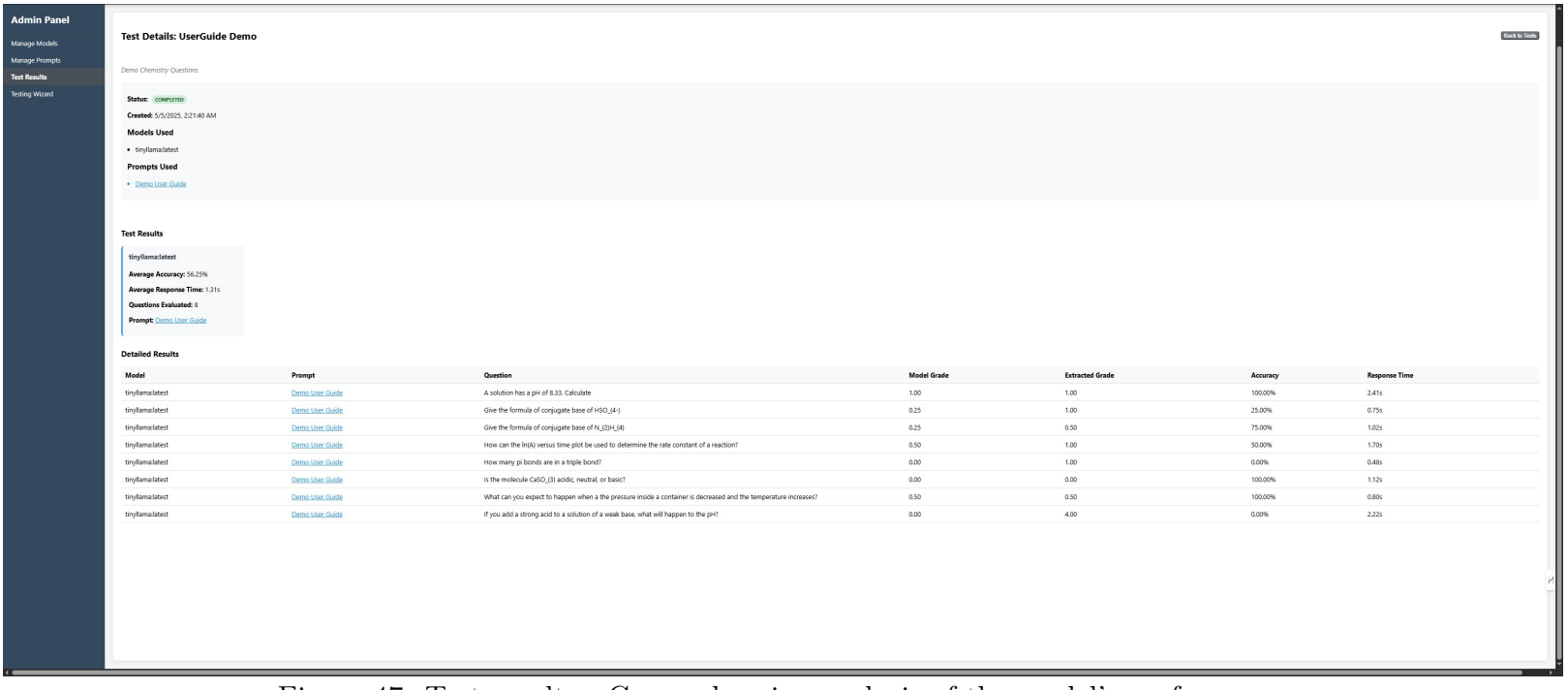

Figure 47: Test results - Comprehensive analysis of the model's performance

52. The test results are saved and can be accessed later from the Test Results page

Figure 48: Saved test results - Your test configuration and outcomes are permanently stored

53. Return to the home page by clicking the **"Home"** button in the navigation bar

9.8 Creating and Managing Pairs

Pairs are combinations of prompts and models used for grading student answers.

53. From the home page, access pair management features

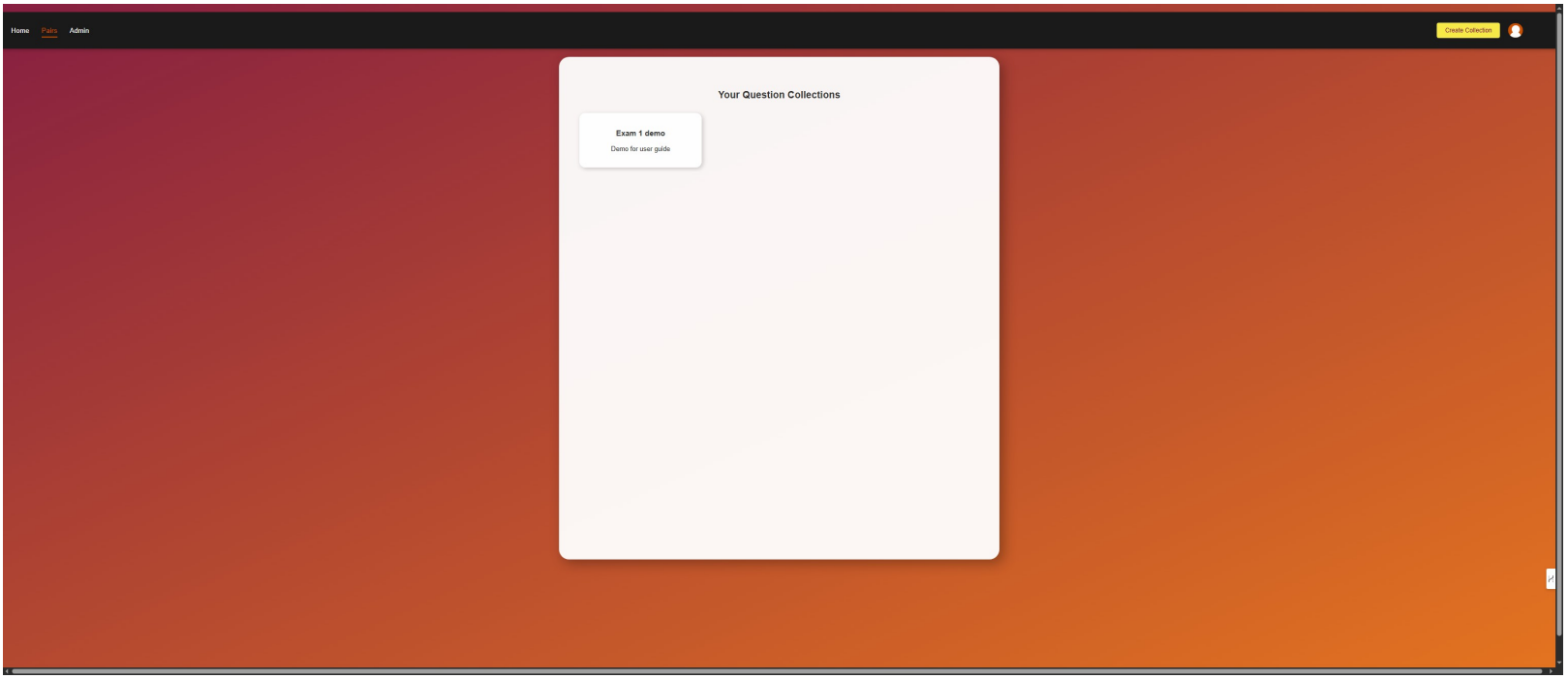

Figure 49: Home page navigation - Access the system's main interface

54. Click **"Pairs"** in the navigation bar to open the Pairs management section

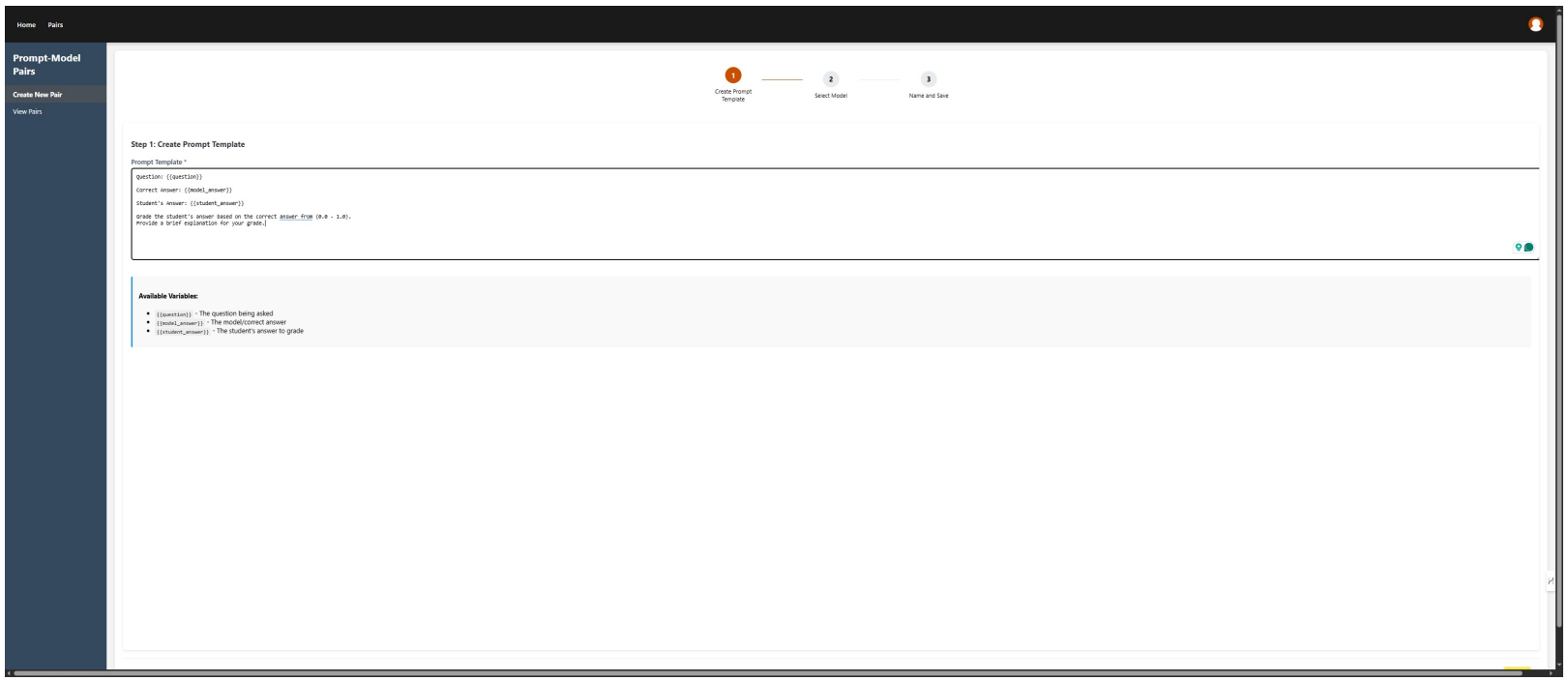

Figure 50: Pairs section - Interface for creating model-prompt combinations

55. Modify the prompt template as needed (e.g., adjusting the scoring scale from 0-1 to 0-100)

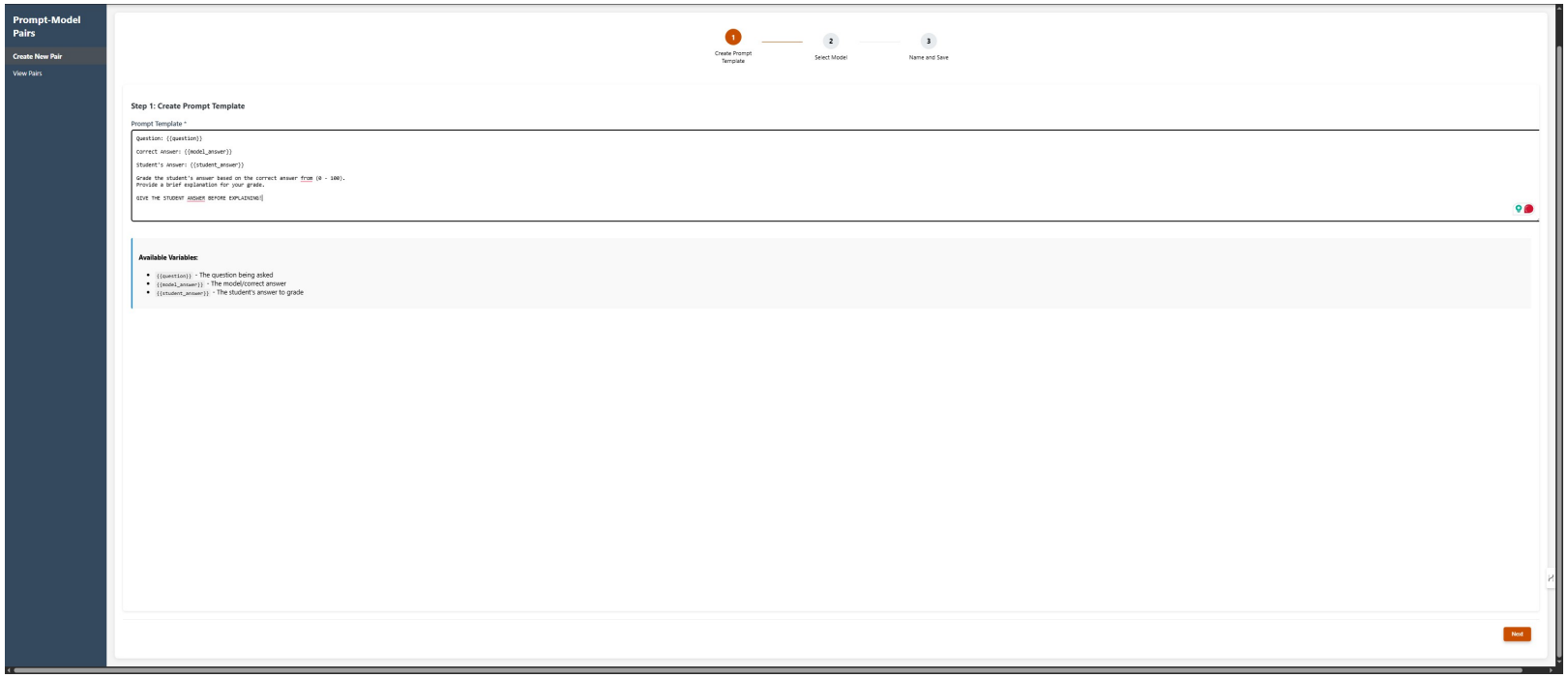

Figure 51: Template editing - Customize the prompt for specific assessment needs

56. Select your desired AI model from the dropdown menu (for this guide: **"tinyllama:latest"**)

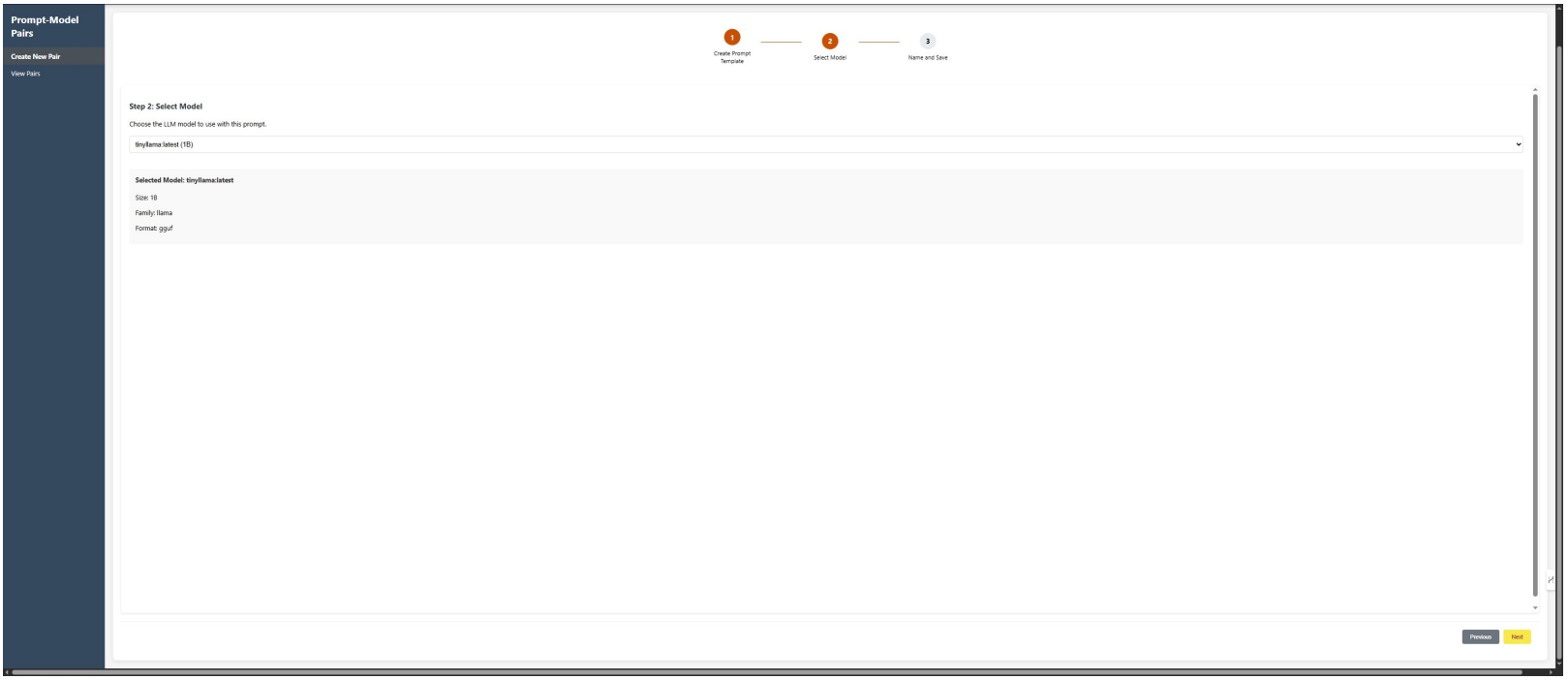

Figure 52: Model selection - Choose which AI model to pair with your prompt

57. Click **"Next"** and provide a name and optional description for your model-prompt pair

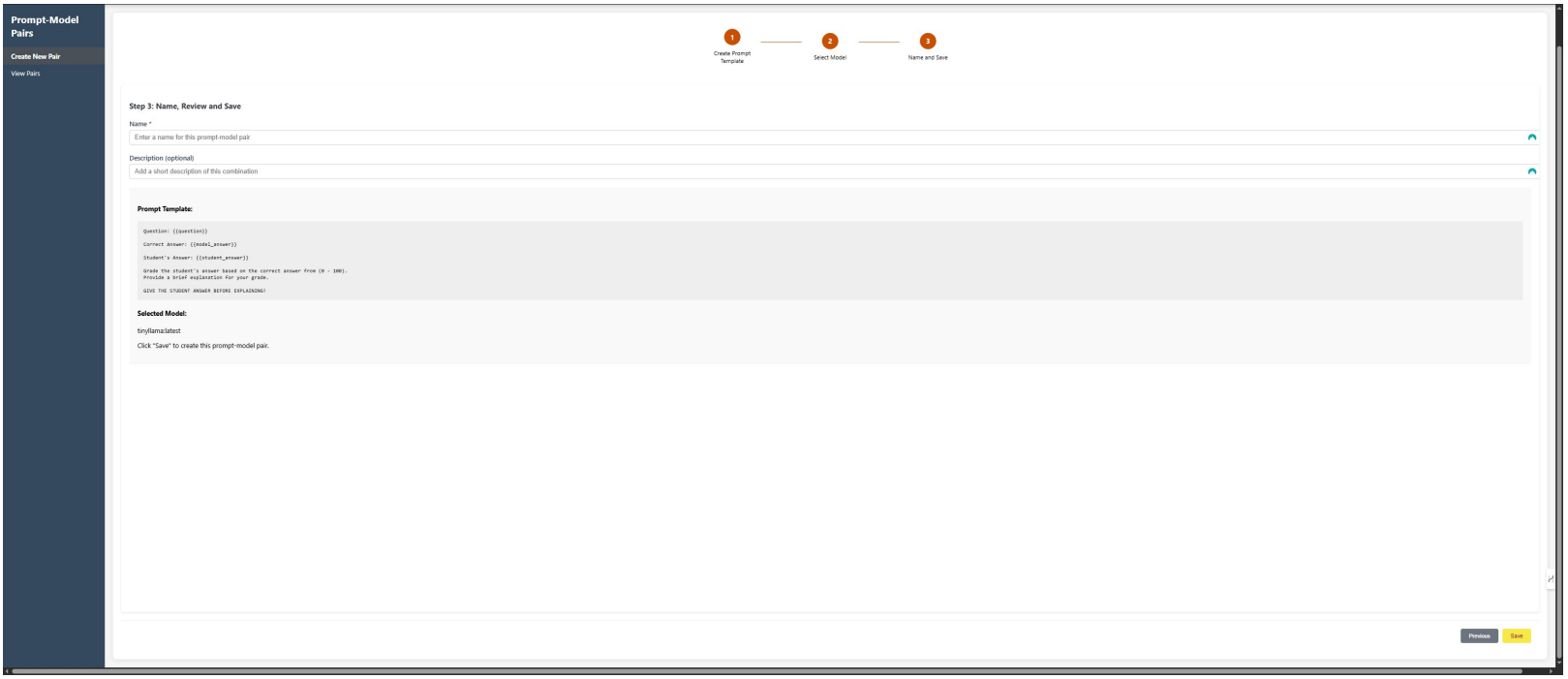

Figure 53: Pair configuration - Navigate through the setup process

58. Review the summary and click **"Save"** to create your prompt-model combination

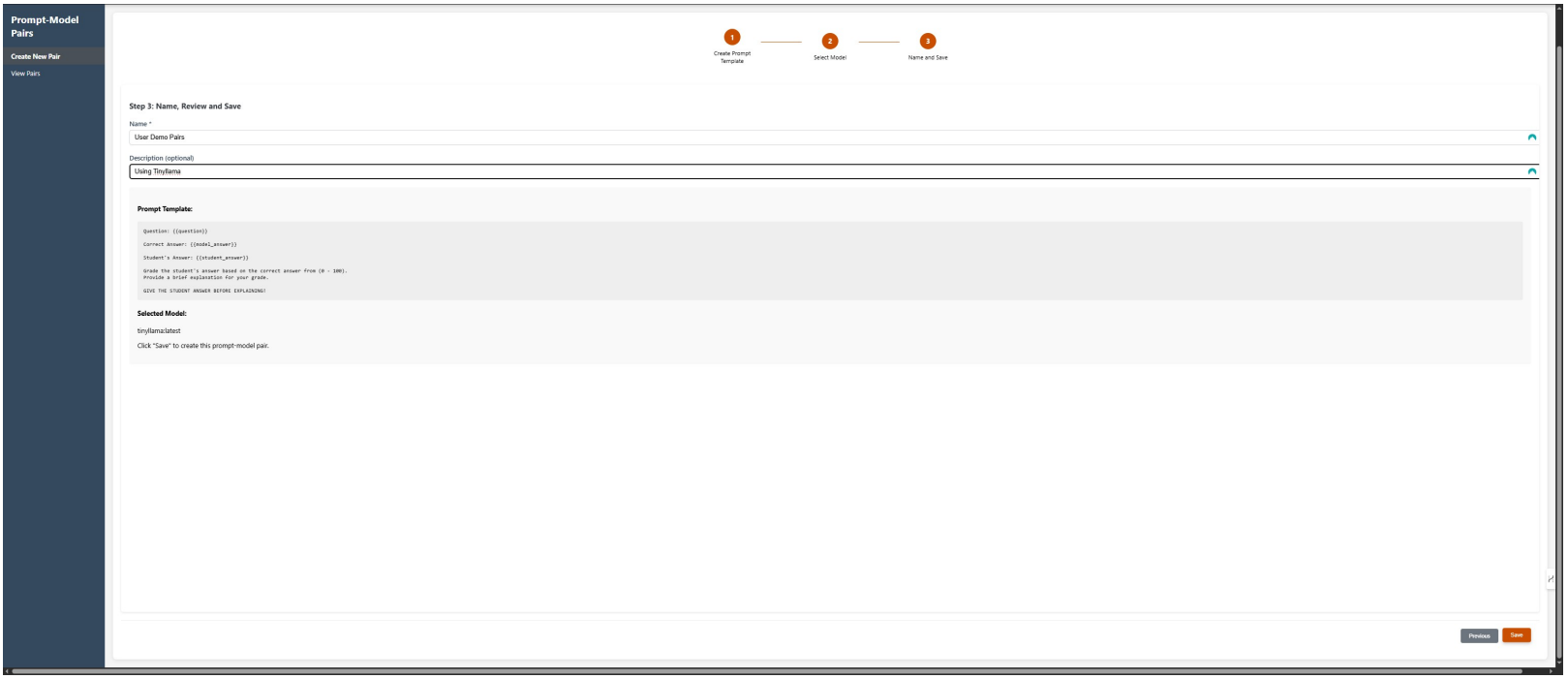

Figure 54: Navigation options - Return to the home page after configuration

59. Verify your newly created pair is now available in the Pairs section

Figure 55: Saved pairs - List of all available prompt-model combinations

60. Return to the home page to create a new collection using your custom pair

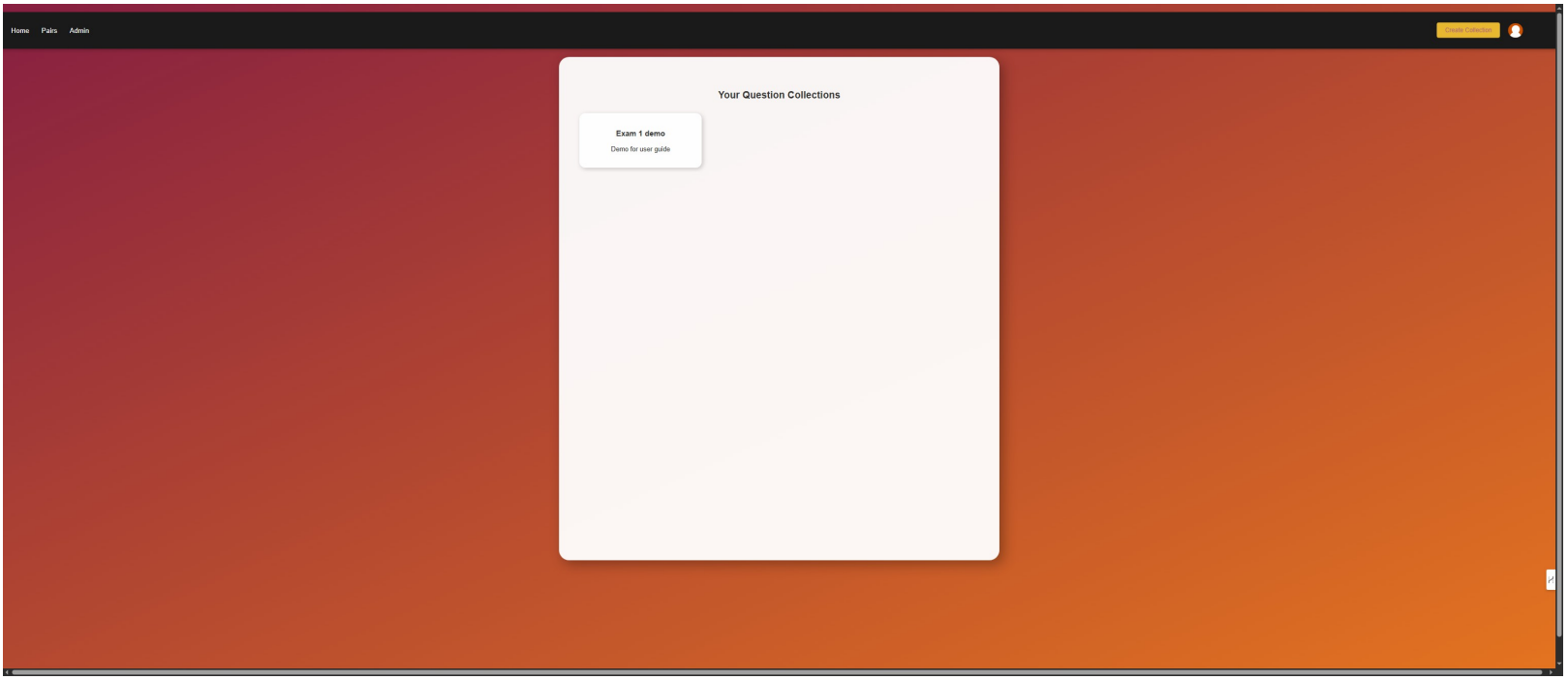

Figure 56: Home page - Return to the main dashboard to create a new collection

61. When creating a new collection, select your custom pair from the **"Grading Pair"** dropdown menu

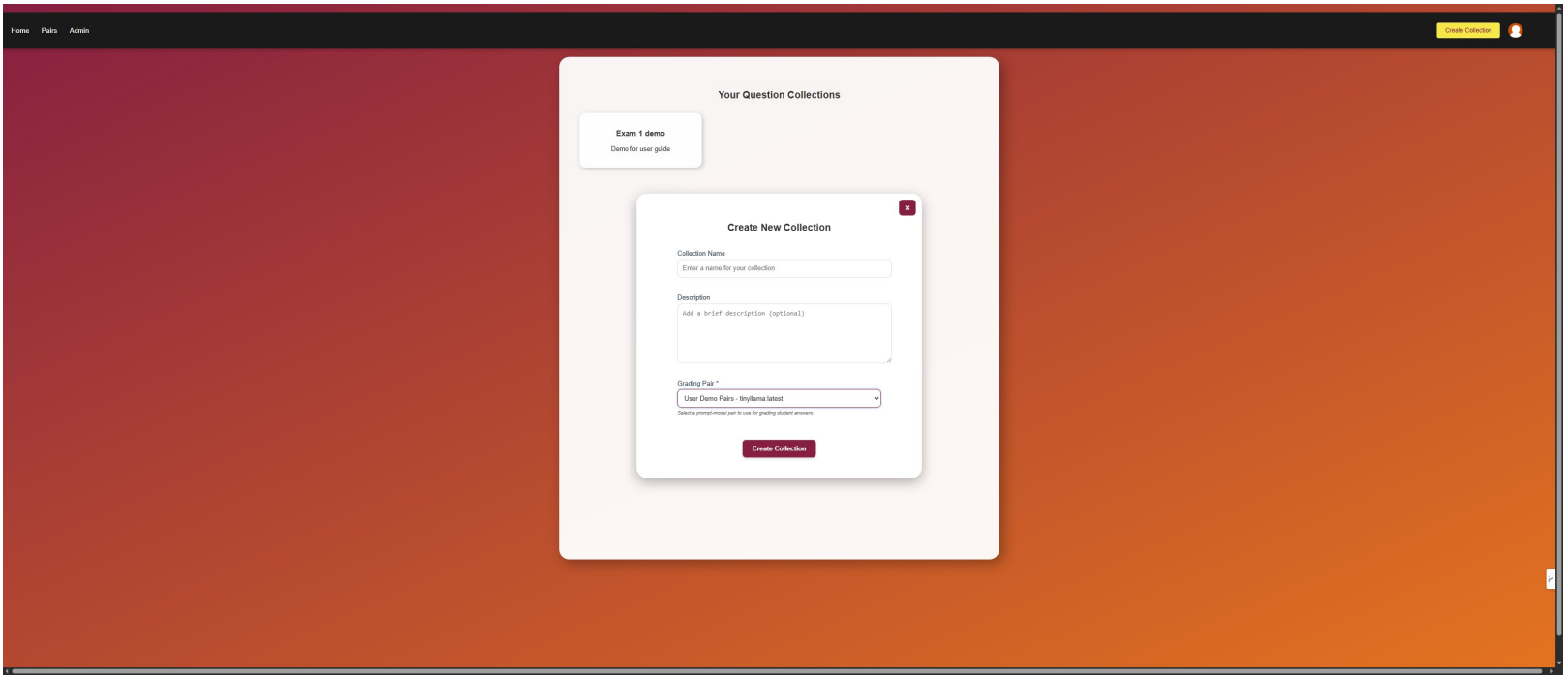

Figure 57: Pair selection - Choose your custom model-prompt combination for assessment

10 Developer Manual

You may choose to install and run the system locally, through terminal, or through Docker (with 1 or 4 images).

10.0.1 Local Installation

1. **Install prerequisites**

   (a) Git: `https://git-scm.com/downloads/win` (select the 64-bit installer and accept all default options).

   (b) Python: `https://www.python.org/downloads/`

   (c) Ollama: `https://ollama.com/download/windows` (run the installer).

   (d) Node.js (if `npm` is unavailable): `https://nodejs.org/en/download` (choose the "Windows Installer" and accept defaults).

   (e) PostgreSQL 17.4 (x86-64): `https://www.postgresql.org/download/windows/`. During setup choose a memorable password.

2. **Clone the repository**
   Open a Windows `cmd` window, navigate to the directory where you want the project, and run:

   ```
   git clone https://github.com/f74dragon/Automated-Students-short-answers-assessment
   ```

3. **Configure the project**

   (a) In the project root open `.env` and set `DOCKERIZED=false`.

   (b) Create a Python virtual environment and activate it:

   ```
   cd Automated-Students-short-answers-assessment
   python -m venv venv
   ```

   (c) Run the Postgres configuration script and enter the password you created during installation twice:

   ```
   configPostWin.bat
   ```

4. **Build and launch the front-end**

   ```
   cd frontend
   npm install
   npm run build
   npm start
   ```

   If prompted by Windows Firewall, click *Allow*.

5. **Launch the back-end** (in a new command prompt)

```
venv\Scripts\activate
cd backend
pip install -r requirements.txt
uvicorn app.main:app --reload --host 127.0.0.1 --port 8000
```

10.0.2 Local Installation (Terminal-Only Workflow)

The following steps assume you will perform *all* actions from a terminal (PowerShell on Windows, Terminal on macOS, or any shell on Linux). Replace the commands with the variant appropriate for your platform as shown.

1. **Install prerequisites from the command line**

   Windows (PowerShell `winget`)

   ```
   winget install --id Git.Git -e
   winget install --id Python.Python.3.12 -e
   winget install --id OpenJS.NodeJS.LTS -e
   winget install --id PostgreSQL.PostgreSQL -e
   winget install --id Ollama.Ollama -e
   # Re-open PowerShell so new PATH entries are active
   ```

   macOS (`brew`)

   ```
   brew update
   brew install git python@3.12 node postgresql ollama
   ```

   Ubuntu / Debian

   ```
   sudo apt-get update

   sudo apt-get install -y git python3 python3-venv nodejs npm \
       postgresql postgresql-contrib

   curl https://ollama.ai/install.sh | sh
   ```

   Tip: After installation, ensure each tool is visible on your `PATH` by running `git --version`, `python --version`, `node --version`, `psql --version`, and `ollama --help`.

   Run Ollama : Make sure Ollama is running. Run the following command:

   ```
   Run Ollama in the background
   Linux/MacOs:
           ollama serve &
   Windows:
           start "" ollama serve
   Note: you can just use "ollama serve" but it will occupy the terminal.
   A new terminal must be opened to continue.
   ```

2. **Clone the repository**

```
git clone https://github.com/f74dragon/Automated-Students-short-answers-assessment
cd Automated-Students-short-answers-assessment
```

3. **Configure environment variables**
   Copy the template and edit as needed (e.g. to match your local Postgres or Ollama ports):

```
cp .env.local.example .env.local   # Linux/macOS
copy .env.local.example .env.local # Windows
# Then open .env.local in a text editor (e.g. code .env.local) and set:
DOCKERIZED=false
```

4. **Initialise PostgreSQL (first run only)**

   (a) *Windows:* execute the helper script from the project root:

```
configPostWin.bat
```

   (b) *macOS / Linux:* create the database and user manually:

```
# Switch to the 'postgres' system user and enter the psql shell
sudo -u postgres psql

# Then run the following SQL commands inside psql:
CREATE ROLE "user" LOGIN PASSWORD 'mypassword';
CREATE DATABASE mydatabase OWNER "user";

# Exit the psql shell
\q
```

   (c) **Create and activate a Python virtual environment**

```
python -m venv venv
# Windows
venv\Scripts\activate
# macOS / Linux
source venv/bin/activate
```

   (d) **Install Python dependencies and launch the back-end**

```
cd backend
pip install -r requirements.txt
uvicorn app.main:app --reload --host 127.0.0.1 --port 8000
```

   (e) **Build and serve the front-end (in a second terminal)**

```
cd frontend
npm install
npm run build          # generates static assets
npm start              # serves at http://localhost:3000
```

If the terminal warns about blocked ports or permissions, resolve them, then re-run the command.

Once both terminals show their respective "`Listening on …`"
messages, open `http://localhost:3000` in a browser to use the application.

10.0.3 All-in-One Docker Image

This option allows you to run the entire project stack (frontend, backend, database, and Ollama) from a **single custom Docker image**, ideal for Linux users seeking a streamlined setup. To install Docker, go to Section 10.0.4 below.

(a) Clone the repository:

```
git clone https://github.com/f74dragon/Automated-Students-short-answers-assessment
cd ./Automated-Students-short-answers-assessment/LinuxDocker
```

(b) Ensure Docker is installed and running. To install Docker, go to Section 10.0.4.

(c) Build and run the all-in-one Docker container:

```
For Linux/Mac:
./rebuild.sh

For Windows
./rebuild.bat
```

(d) After setup completes, access the system in your browser at:

`http://localhost:3000`

This method automatically installs dependencies, configures PostgreSQL, builds the frontend, installs the backend, and launches all services.

Note: Make sure that the setup is complete. In the terminal, the last thing will be **uvicorn** running. Should see something like this:

```
INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
```

10.0.4 Docker Installation 4 Images (Recommended)

This project uses Docker Desktop version 4.40.0 and Docker Engine 28.0.4. Any later versions are not currently tested and may not work with this version of the project. Refer to the section 10.4 for more information on using Docker, running the compose file, and Docker commands.

Windows Installation

(a) Download Docker Desktop for Windows from the official Docker website: `https://www.docker.com/products/docker-desktop`

(b) Double-click the installer file `Docker Desktop Installer.exe`

(c) Follow the installation wizard instructions, keeping the default settings

(d) When prompted, ensure the options to install required Windows components for WSL 2 are selected

(e) Click `Finish` to complete the installation

(f) Restart your computer when prompted

(g) **Enable Virtualization in BIOS (if not already enabled):**

i. Reboot your computer and enter the BIOS/UEFI settings. This is usually done by pressing a key like `Del`, `F2`, `F10`, or `Esc` during startup (varies by manufacturer).

ii. Navigate to the **Advanced**, **CPU Configuration**, or **Security** tab—location varies by BIOS.

iii. Look for an option named `Intel Virtualization Technology` or `SVM Mode` (for AMD processors).

iv. Set this option to `Enabled`.

v. Save your changes and exit the BIOS (usually by pressing `F10`).

vi. **Tip:** If you cannot find the virtualization setting, consult your motherboard and CPU documentation or search online for instructions specific to your hardware model.

macOS Installation

(a) Download Docker Desktop for Mac from the official Docker website: `https://www.docker.com/products/docker-desktop`

(b) Open the downloaded `.dmg` file

(c) Drag the Docker icon to the Applications folder

(d) Open the Applications folder and double-click Docker to start it

(e) When prompted, enter your password to allow Docker to make system changes

(f) Wait for the Docker initialization process to complete

Linux Installation (Ubuntu) In a terminal, enter the following commands sequentially:

```
# Update package information
sudo apt-get update

# Install required dependencies
```

```
sudo apt-get install apt-transport-https ca-certificates \
curl software-properties-common

# Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

# Add Docker repository
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com \
/linux/ubuntu$(lsb_release -cs) stable"

# Update package information again
sudo apt-get update

# Install Docker
sudo apt-get install docker-ce docker-ce-cli containerd.io

# Add your user to the docker group to run Docker without sudo
sudo usermod -aG docker $USER
```

Note: For other Linux distributions, refer to the official Docker documentation.

10.0.5 Verifying Docker Installation

After installation, verify that Docker is working correctly:

```
docker --version
docker run hello-world

docker compose up --build
```

10.0.6 Running the Project with Docker

Clone the repo first:

```
git clone https://github.com/f74dragon/Automated-Students-short-answers-assessment
```

After configuring Docker and cloning the repository, navigate to the project folder using the terminal:

```
cd Automated-Students-short-answers-assessment
```

If you are running the project on a computer equipped with an **NVIDIA GPU**, no further changes are needed. However, if your system does **NOT** have an **NVIDIA GPU**, you must do the following:

(a) Rename the default `docker-compose.yml` file to some other name.

(b) Rename the file `docker-compose-no-nvidia.yml` to `docker-compose.yml`.

This ensures the project uses the correct Docker configuration based on your hardware.

Once this is set up, run the following command to build and start all the services defined in the Docker configuration:

```
docker compose up --build
```

This command builds the Docker images (if it is not already built) and starts the containers.

After it completes, the frontend server should be running and accessible at `http://localhost:3000`.

10.1 Technologies Used

Here is a list of the technologies and frameworks used in the current iteration of this project:

- FastAPI
- React.js
- Docker
- SQLAlchemy
- PostgreSQL

10.2 Project Structure

The project follows a modular architecture to enhance decoupling, ensuring that individual components can be seamlessly replaced or upgraded with alternative implementations as needed:

```
Automated-Students-short-answers-assessment/
|-- backend/
|-- examples/
|-- frontend/
|-- .dockerignore
|-- .env
|-- .env.local
|-- .gitignore
|-- README.md
|-- configPostWin.bat
|-- docker-compose.yml
|-- docker-compose-no-nvidia.yml
```

The backend/, frontend/, with llm/ subdirectories contain all source code for each respective component's functionalities. The examples/contain questions for testing and the data pertaining to the collections. The Dockerfile that specifies how to containerize these components for deployment. A .env file is used to specify environment variables for local development, but environmental variables should be specified in the docker-compose.yml file for deployment. The configPostWin.bat/ helps set up the backend environment in a more feasible manner. Lastly, the compose files specify how to connect the different images built from the Dockerfiles. docker-compose.yml assumes a NVIDIA GPU is installed, however renaming docker-compose-no-nivida.yml to docker-compose.yml configures the project without a NVIDIA GPU.

10.3 Backend

The backend contains a standard FastAPI project structure, with most of the functionality contained in the app/ subdirectory. The Dockerfile contains information on how to containerize the backend image. tests/ contains the folders with .gitkeep.

```
Automated-Students-short-answers-assessment/backend/
|-- ..
|-- app/
 -- tests/
 -- Dockerfile
```

10.3.1 app/

The app/ subdirectory contains all the source code to handle API requests, database functions, and LLM requests.

```
Automated-Students-short-answers-assessment/backend/app/
|-- ..
|-- api/
|-- auth/
|-- database/
|-- models/
|-- schemas/
|-- main.py
```

10.3.2 main.py

The main.py file serves as the entry point for the FastAPI backend application. It defines and initializes the main FastAPI app instance, configures Cross-Origin Resource Sharing (CORS) to allow frontend requests (e.g., from http://localhost:3000), and includes all API routers such as users, login, questions, and others.

Upon application startup, the `startup_event()` function is triggered. This function performs two key tasks:

(a) Initializes the database connection by calling `init_db()`.

(b) Checks whether any combinations exist in the `Combination` table. If none are found, it creates a default grading combination with a predefined prompt and model name (`gemma3:4b`).

This logic ensures the system always has a fallback grading configuration upon first deployment.

```
    # Default prompt template for grading
    DEFAULT_PROMPT = """Question: {{question}}

Correct Answer: {{model_answer}}

Student's Answer: {{student_answer}}

Grade the student's answer based on the correct answer from (0.0 - 1.0).
Provide a brief explanation for your grade."""
```

Figure 58: Default grading prompt template inserted into the database if no combinations exist. This prompt is used to evaluate student answers by comparing them to a reference answer using an gemma3:4b.

10.3.3 `api/`

The `api/` subdirectory contains the API router endpoints for different backend features. Each route is defined in a modular fashion to keep the application maintainable and scalable.
Here is the `collections.py` API route that handles collection management:

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from app.database.connection import get_db
from app.database import crud
from app.schemas.collection_schema import CollectionCreate, CollectionResponse,
```

Figure 59: `collections.py` API Imports

The imports at the top are necessary imports from FastAPI and SQLAlchemy. The rest of the imports come from other packages within the `app/` directory.

```
router = APIRouter(prefix="/collections", tags=["collections"])
```

Figure 60: `collections.py` base router

This line defines the base router for the collection endpoints as well as the prefix used to distinguish it. The base router will be imported into `main.py` to define the routes for the collection endpoints.

```python
# Creates a collection and inserts into db
@router.post("/", response_model=CollectionResponse)
def create_collection(collection: CollectionCreate, db: Session = Depends(get_db)):
    try:
        return crud.create_collection(db=db, collection=collection)
    except ValueError as e:
        raise HTTPException(status_code=404, detail=str(e))

# Get a list of all collections in the db
@router.get("/", response_model=CollectionListResponse)
def get_all_collections(db: Session = Depends(get_db)):
    return crud.get_all_collections(db=db)
```

Figure 61: `collections.py` REST API Endpoints

These are two sample `GET` and `POST` HTTP endpoints for use in the frontend. The `response_model` is a Pydantic model for data validation, `crud` is a semantic way to access the database for `CRUD` operations, and `get_db` is a function that utilizes dependency injection to obtain the current database session to mitigate database access errors.

10.3.4 `auth/`

This directory contains scripts for authentication in the backend. It only contains one file `auth.py`, which has various methods for hashing passwords, comparing hashed passwords, creating JWT tokens, and verifying JWT tokens.

10.3.5 `database/`

There are two files in the `database/` subdirectory, `connection.py` and `crud.py`. `connection.py` is responsible for the initial database connection and database session management, which is used every time the database is accessed. It uses a dependency injection function `get_db` to manage database access.

The `crud.py` module implements the CRUD operations for database access, providing an abstraction layer that simplifies interaction with the database. This allows other components of the backend to handle entities within the context of business logic more efficiently.

Each entity is given CRUD access through SQLAlchemy queries, which are structured similarly to SQL queries:

```python
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Figure 62: `get_db/` Dependency Injection Function

```python
def get_all_collections(db: Session) -> CollectionListResponse:
    collections = db.query(Collection).all()
    return CollectionListResponse(collections=[
        CollectionResponse(
            id=col.id,
            name=col.name,
            description=col.description,
            user_id=col.user_id,
            combination_id=col.combination_id
        ) for col in collections
    ])
```

Figure 63: Example Database Query for Collections

In the future, it may be helpful to split up the crud file based on entities as the application grows in size to improve scalability, since the crud.py file is very large.

10.3.6 `models/`

This directory contains the database models used in the application. This is not to be confused with the Large Language Models used in this application. These models were made using the SQLAlchemy module to allow for easier back-end and database integration. Here is an example of the `Collections` model:

```python
class Collection(Base):
    __tablename__ = "collections"
    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String, nullable=False)
    description = Column(String)
    user_id = Column(Integer, ForeignKey("users.id", ondelete="CASCADE"), nullable=False)

    # Define relationships
    owner = relationship("User", back_populates="collections")
    students = relationship("Student", back_populates="collection", cascade="all, delete")
    questions = relationship("Question", back_populates="collection", cascade="all, delete")
```

Figure 64: Example Collections Model

SQLAlchemy allows table name declarations, column definitions, and foreign key con-

straints to reference other tables. `Collections` requires relationships with other entities such as `User`, `Student`, and `Question`, so foreign key constraints are declared at the bottom of the class.

10.4 Introduction to Docker

Docker is a platform that uses containerization technology to package applications and their dependencies into a standardized unit for software development and deployment. This section provides instructions for installing and running Docker on different operating systems.

10.4.1 System Requirements

- **Windows:** Windows 10 64-bit: Pro, Enterprise, or Education (Build 17134 or later) with Hyper-V enabled, or Windows 10 64-bit Home (Build 19018 or later) with WSL 2 enabled
- **macOS:** macOS version 10.15 or newer (Catalina, Big Sur, Monterey, or newer)
- **Linux:** Kernel version 3.10 or higher, 64-bit installation
- **RAM:** At least 4GB of system RAM, 8GB recommended
- **CPU:** 64-bit processor with hardware virtualization support
- **GPU:** 8GB VRAM recommended

The first command shows the installed Docker version, and the second downloads and runs a test container to verify the installation.

10.5 Running the Application with Docker Compose

10.5.1 Prerequisites

Docker Compose is included with Docker Desktop for Windows and Mac. For Linux, you might need to install it separately:

```
sudo curl -L "https://github.com/docker/compose/releases/download/ \
v2.18.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

10.5.2 Starting the Application

1. Clone the application repository or extract the application files to a local directory
2. Open a terminal or command prompt
3. Navigate to the application directory containing the `docker-compose.yml` file
4. Run the following command to start all services:

```
docker-compose up -d
```

This command builds, creates, and starts the containers defined in the `docker-compose.yml` file. The `-d` flag runs containers in the background.

10.5.3 Accessing the Application

Once the containers are running, access the application through a web browser:

- Frontend: `http://localhost:3000`
- Backend API: `http://localhost:8001`

10.5.4 Stopping the Application

To stop all running containers:

```
docker-compose down
```

To stop and remove all containers, networks, and volumes:

```
docker-compose down -v
```

10.5.5 Common Docker Commands

Command	Description
`docker ps`	List running containers
`docker ps -a`	List all containers (running and stopped)
`docker images`	List all images
`docker logs [container-id]`	View logs for a specific container
`docker exec -it [container-id] bash`	Open a shell inside a running container
`docker-compose logs`	View logs from all containers defined in docker-compose.yml
`docker-compose restart`	Restart all services defined in docker-compose.yml

10.5.6 Common Issues

- **Port conflicts:** If ports 3000 or 8001 are already in use, modify the port mapping in the `docker-compose.yml` file
- **Permission issues on Linux:** Ensure your user is added to the docker group or run commands with `sudo`
- **WSL 2 not enabled on Windows:** Follow the prompts to enable WSL 2 during installation
- **Insufficient memory:** Increase the memory allocated to Docker in Docker Desktop settings

10.5.7 Resetting Docker

If you encounter persistent issues, you can reset Docker to its default state:

1. Open Docker Desktop
2. Go to Settings/Preferences
3. Click on "Troubleshoot" or "Reset"
4. Select "Reset to factory defaults"

10.5.8 Updating Docker

To update Docker:

- **Windows and macOS:** Docker Desktop will notify you when updates are available
- **Linux:** Use your package manager to update (e.g., `sudo apt-get update && sudo apt-get upgrade`)

10.5.9 Additional Resources

- Official Docker Documentation: `https://docs.docker.com`
- Docker Hub: `https://hub.docker.com`
- Docker Community Forums: `https://forums.docker.com`

11 Timeline

Milestone	Target Date	Status
Research & Prototyping	Feb 24	Done
Basic UI & Database	Mar 17	Done
Backend & AI Finalization	Mar 31	Done
Deployment & E2E Testing	Apr 14	Done
Final Documentation	Apr 28	Done

- Feb 24 - Research & Prototyping
  - Completed. Technologies selected, initial API routes planned, and LLM models compared.
- Mar 17 - Basic UI & Database Setup
  - Completed. Login/authentication implemented, collections page functional, PostgreSQL integrated.
- March 31 - Backend & AI integration
  - Completed. There's a default LLM chosen, however admins have the option to save and use different models from the Ollama library.
- Apr 14 - Deployment & Testing
  - Completed. Full system testing with Docker Compose, final integration of frontend/backend/LLM
- Apr 28 - Final Documentation & VTechWorks Submission
  - Completed. Final report, source code, manuals, and presentation materials archived and submitted.

12 Problems and Solutions

12.1 Technical Lessons

Integration between LLMs and scoring models is non-trivial

Early in the project, we anticipated that there wouldn't be an extremely big emphasis on what LLM to use. However, we quickly understood that wasn't the case. Model performance varied drastically in terms of output format, latency, and reliability. This led to a more comparative evaluation of multiple models like (Mistral 7b vs Gemma 3:4b for its consistency and performance in local environments.

Creating flexible grading strategies required a deeper understanding of educational rubrics and LLM prompts

Translating a grading rubric or concept list into logic for an LLM is not straightforward. We learn that prompt design is something that is constantly evolving and is an iterative process. As a result, we decided on allowing admins to modify and customize prompts, since different prompts will significantly change the assessment given by the LLM. Rubrics do not translate cleanly into a structured prompting logic, signifying the need for iterative prompt design.

Early decisions on API design helped streamline parallel development

Our strategy involved concurrent development of the frontend and backend components with a defined API contract. However, we encountered setbacks during the integration phase whenever discrepancies arose in the API contract-specifically concerning expected data structures in calls or responses. This experience served as a critical lesson in documenting meaningful response formats and status codes when working with API's. Once this contract was refined and adhered to, the parallel development workflow proceeded effectively, allowing for the smooth completion of remaining project tasks with minimal integration issues.

12.2 Project Process Lessons

Cross-Functional Communication is Critical

Since roles were divided (LLM integration, frontend, backend, testing), there was a need for effective communication. As a team, regular sync meetings were scheduled to ensure that all members were updated on expectations, progress, and roadblocks preventing the team from making progress.

Documentation Should be Continuous

While writing the interim report, it was immediately evident that it would be easier to document the project and progress while in development rather than after. This showed the need of constant and consistent documentation during development.

Agile Iteration

Our team truly benefited from using a lightweight agile methodology when it came to weekly check-ins and we followed our milestones pretty well. This helped us adapt quickly when we were approached by technical blocks or other requests made by the client.

13 Plans

13.1 Potential Future Work

Advanced Analytics Dashboard

One thing future teams could do is build an advanced analytics dashboard. This would offer admins with visual insights into how their class is performing over time. It would include trends of concept mastery, invidual student progress, and overall areas of difficulty. With these capabilities, instructors could examine performance in more specific instances by filtering by a certain question or rubric criteria.

LMS Integration

Another avenue for change is the integration of Learning Management Systems like Canvas and Blackboard. Building plugins or APIs that would allow for automatic syncing of student data, submissions, grades, and feedback would create a smoother experience for instructors and students. It would be rather helpful if combined with Single Sign-On support.

Multilingual Support

This improvement is essential and could prove very useful. The system could be expanded to support student answers and feedback in multiple languages, which is definitely a top priority in diverse and international classrooms. To ensure that this could be met with full usability, the interface should comply with accessibility standards and offer support for screen readers, keyboard navigation, and other technologies that assist users.

Optimizing Prompts and Grading Strategies

More future work could also focus on optimizing prompts and grading strategies. This would involve testing different prompts and LLMs to identify which combination provides the more accurate and fair grading. Developers could essentially explore a more rubric-based scoring, comparisons against ideally/partially correct answers, and concept based grading to find the best-performing approach.

Customization and Instructor Controls

This would allow for the system to be more flexible and adaptable, allowing instructors to tailor grading behavior, feedback tone, rubric emphases based on their specific teaching style. This would all be made easy to a configurable interface.

14 Acknowledgments

We thank our client, Dr. Mohamed Farag, for his guidance. We also acknowledge Virginia Tech and CS4624 - Multimedia/Hypertext for providing us with the opportunity and support to explore impactful applications of NLP in education.

15 References

1. Singh, Purushartha, et al. *Automated Support to Scaffold Students' Written Explanations in Science.* SpringerLink.
2. Client: Mohamed Farag, mmagdy@vt.edu
3. Mistral 7b Documentation
4. BERTScore: https://github.com/Tiiiger/bert_score
5. FastAPI: https://fastapi.tiangolo.com/
6. React.js: https://reactjs.org/