# **Tasks for AI-Powered eLearning Platform User Stories**

## **User Story 1.1: User Registration**

**As a** new user, **I want** to register for an account with my email and password, **so that** I can access the eLearning platform and start learning.

### **Task 1.1.1: Develop Registration API Endpoint**

* **Role**: Developer
* **Description**: Create a RESTful API endpoint (POST /api/register) to handle user registration with email and password inputs. Use bcrypt for password hashing and validate email format.
* **Acceptance Criteria** (from User Story 1.1):
  + Accepts valid email and password (min 8 chars, 1 uppercase, 1 number).
  + Returns error for invalid email or weak password.
  + Stores user data securely in the database.
* **Estimated Effort**: 4 hours
* **SQA Note**: Testers will validate API responses and security.

### **Task 1.1.2: Implement Email Verification Logic**

* **Role**: Developer
* **Description**: Develop a system to send a verification email with a unique link after registration. Use an email service (e.g., SendGrid) and store verification tokens in the database.
* **Acceptance Criteria** (from User Story 1.1):
  + Sends verification email within 10 seconds of registration.
  + Verification link expires in 24 hours.
* **Estimated Effort**: 3 hours
* **SQA Note**: Testers will check email delivery and link functionality.

### **Task 1.1.3: Design Registration UI**

* **Role**: Developer
* **Description**: Build a responsive registration form UI (React/Vue) with fields for email, password, and confirm password. Ensure WCAG 2.1 compliance (e.g., ARIA labels).
* **Acceptance Criteria** (from User Story 1.1):
  + Form supports accessibility (WCAG 2.1).
  + Displays error messages for invalid inputs.
* **Estimated Effort**: 5 hours
* **SQA Note**: Testers will verify UI responsiveness and accessibility.

### **Task 1.1.4: Test Registration API**

* **Role**: Tester
* **Description**: Create and execute manual and automated test cases (using Postman) for the registration API endpoint. Test valid/invalid inputs, error handling, and security (e.g., SQL injection).
* **Acceptance Criteria** (from User Story 1.1):
  + API accepts valid inputs and rejects invalid ones.
  + No vulnerabilities (e.g., SQL injection, XSS).
* **Test Cases**:
  + TC1: Valid email/password → 200 OK, user created.
  + TC2: Invalid email → 400 Bad Request.
  + TC3: Weak password → 400 Bad Request.
* **Estimated Effort**: 3 hours

### **Task 1.1.5: Test Email Verification Flow**

* **Role**: Tester
* **Description**: Manually test the email verification process, including link generation, email receipt, and account activation. Use a test email service (e.g., Mailtrap).
* **Acceptance Criteria** (from User Story 1.1):
  + Verification email sent and received correctly.
  + Link activates account; expired links fail.
* **Test Cases**:
  + TC1: Click valid link → Account activated.
  + TC2: Click expired link → Error message.
* **Estimated Effort**: 2 hours

## **User Story 3.1: Course Recommendations**

**As a** student, **I want** to receive AI-generated course recommendations based on my profile and progress, **so that** I can find relevant courses to meet my learning goals.

### **Task 3.1.1: Develop AI Recommendation Algorithm**

* **Role**: Developer
* **Description**: Implement an AI model (e.g., Python with scikit-learn) to generate course recommendations based on user profile (preferences, progress). Integrate with backend API.
* **Acceptance Criteria** (from User Story 3.1):
  + Recommends at least 3 courses based on user data.
  + Updates recommendations within 5 seconds.
* **Estimated Effort**: 8 hours
* **SQA Note**: Testers will validate recommendation accuracy.

### **Task 3.1.2: Create Recommendation API Endpoint**

* **Role**: Developer
* **Description**: Build a RESTful API endpoint (GET /api/recommendations) to fetch AI-generated course recommendations for a user, including course details and relevance score.
* **Acceptance Criteria** (from User Story 3.1):
  + Returns course title, description, and relevance score.
  + Handles cases with no user preferences.
* **Estimated Effort**: 4 hours
* **SQA Note**: Testers will verify API responses and edge cases.

### **Task 3.1.3: Design Recommendation Dashboard UI**

* **Role**: Developer
* **Description**: Develop Delevelop a dashboard UI to display course recommendations, with responsive design and accessibility features (WCAG 2.1).
* **Acceptance Criteria** (from User Story 3.1):
  + Dashboard displays recommendations clearly.
  + UI is responsive and accessible.
* **Estimated Effort**: 5 hours
* **SQA Note**: Testers will check UI functionality and accessibility.

### **Task 3.1.4: Test Recommendation API**

* **Role**: Tester
* **Description**: Test the recommendation API using Postman for valid and edge-case scenarios (e.g., no preferences, invalid user ID). Validate response time and data accuracy.
* **Acceptance Criteria** (from User Story 3.1):
  + API returns 3+ recommendations with correct details.
  + Response time < 5 seconds.
* **Test Cases**:
  + TC1: Valid user ID → Returns recommendations.
  + TC2: No preferences set → Returns default recommendations.
* **Estimated Effort**: 3 hours

### **Task 3.1.5: Test Recommendation Accuracy**

* **Role**: Tester
* **Description**: Manually verify that recommendations align with user preferences and progress. Use predefined user profiles to test AI model outputs.
* **Acceptance Criteria** (from User Story 3.1):
  + Recommendations match user preferences (e.g., topic, level).
  + Edge cases (e.g., new user) handled correctly.
* **Test Cases**:
  + TC1: User with programming preference → Programming courses recommended.
  + TC2: New user → General courses recommended.
* **Estimated Effort**: 2 hours

## **User Story 4.1: Interactive Quiz Interface**

**As a** student, **I want** to complete interactive quizzes with multimedia elements (e.g., images, videos), **so that** I can engage with course content dynamically.

### **Task 4.1.1: Develop Quiz Backend Logic**

* **Role**: Developer
* **Description**: Create a backend service to generate and serve quiz questions with text, image, and video content. Store user responses and calculate results.
* **Acceptance Criteria** (from User Story 4.1):
  + Supports text, image, and video-based questions.
  + Immediate feedback on submission.
* **Estimated Effort**: 6 hours
* **SQA Note**: Testers will validate question rendering and feedback.

### **Task 4.1.2: Build Quiz UI**

* **Role**: Developer
* **Description**: Develop a responsive quiz UI (React/Vue) with multimedia question support, answer submission, and feedback display. Ensure WCAG 2.1 compliance.
* **Acceptance Criteria** (from User Story 4.1):
  + UI supports multimedia and multiple-choice/text inputs.
  + Meets WCAG 2.1 accessibility standards.
* **Estimated Effort**: 6 hours
* **SQA Note**: Testers will verify UI responsiveness and accessibility.

### **Task 4.1.3: Test Quiz Functionality**

* **Role**: Tester
* **Description**: Manually and automatically (e.g., Selenium) test the quiz interface for functionality, multimedia rendering, and feedback accuracy across devices.
* **Acceptance Criteria** (from User Story 4.1):
  + Multimedia questions load correctly.
  + Feedback is immediate and accurate.
* **Test Cases**:
  + TC1: Image-based question → Displays correctly.
  + TC2: Submit answer → Immediate feedback shown.
* **Estimated Effort**: 3 hours

### **Task 4.1.4: Test Quiz Accessibility**

* **Role**: Tester
* **Description**: Use tools like WAVE to test quiz UI for WCAG 2.1 compliance (e.g., screen reader support, keyboard navigation).
* **Acceptance Criteria** (from User Story 4.1):
  + UI meets WCAG 2.1 AA standards.
  + Keyboard navigation works seamlessly.
* **Test Cases**:
  + TC1: Screen reader reads question text.
  + TC2: Tab navigation covers all interactive elements.
* **Estimated Effort**: 2 hours

**Notes**:

* Assign tasks to developers and testers in Jira with story point estimates (e.g., 3-5 points per task).
* Link tasks to their respective user stories in Jira for traceability.
* Testers should use tools like Postman, Selenium, and Mailtrap to validate APIs, UI, and email flows.
* Schedule tasks in sprints, prioritizing critical features like registration first.