D295 Task 2: Quest-Based Learning - Demonstrating Essential Concepts for K–12 E-Learning

John Mortensen

2025-10-23

# D295 Task 2: Quest-Based Learning - Demonstrating Essential Concepts for K–12 E-Learning

## Introduction

**In this task, I design an e-learning activity that introduces K–12 students to key concepts, skills, and tasks involved in JavaScript API integration. The activity serves as a demonstration of what students will learn and accomplish through a quest-based learning approach. It highlights how digital tools and interactive strategies can be used to illustrate clear learning outcomes and includes a description of the activity design, technologies employed, and methods of assessment.**

## A. Unit of Study Identification

### Topic of the Lesson

**JavaScript API Integration for Quest-Based Digital Citizenship Applications**

### Essential Concepts, Skills, and Tasks

Students will demonstrate mastery of the following essential concepts through their **Quest of Code** projects:

1. **JavaScript API Consumption** - Using promise-based fetch() with .then() chains and functional programming patterns
2. **AI Integration via APIs** - Designing purposeful Gemini AI prompts and formatting responses for quest UX
3. **Database API Usage** - Implementing CRUD operations with promise chaining for quest progression and rewards
4. **Digital Citizenship Implementation** - Applying ethical technology practices in real-world projects
5. **Quest UX Design** - Formatting API inputs/outputs into engaging, narrative-driven user experiences

## B. Demonstration Strategy and Implementation

### B1. Instructional Strategy: **Quest-Based Learning with API Integration**

**Primary Strategy**: **Instructor Introduction and Demonstration**

* **Show Examples** - Instructor demonstrates live API calls using browser console and development tools
* **Format Modeling** - Teacher shows how to transform raw API responses into quest-friendly formats
* **Error Handling Demonstration** - Live troubleshooting of common API integration issues and solutions
* **Reading Materials** - Comprehensive API documentation and integration guides provided for student reference
* **Step-by-Step Walkthroughs** - Instructor guides students through initial API discovery and integration process

**Secondary Strategy**: **Learn-by-Doing with Collaborative Problem-Solving**

* Students work on their actual **Quest of Code** capstone projects after demonstration
* API integration serves their specific quest narrative and functionality needs
* Team-based quest development with shared API resources and peer debugging
* Real-time problem-solving within the context of preparing for **Night at the Museum**
* Community-focused development preparing for public showcase

### B1a. Strategy Implementation

The lesson follows a **demonstration-first, quest-driven approach** where students:

1. **Observe** - Watch instructor demonstrate live API integration techniques
2. **Discover** - Explore existing Flask APIs through guided interactive testing
3. **Design** - Plan how APIs will enhance their quest user experience with instructor support
4. **Implement** - Integrate Gemini AI and database APIs into their projects using demonstrated patterns
5. **Test** - Validate API integration with guest login functionality
6. **Showcase** - Present working prototypes to community at Night at the Museum

**Demonstration Integration Process**:

* Instructor models how AI and data APIs serve quest narrative development
* Students follow demonstrated patterns with real-time instructor feedback
* Continuous scaffolding with demonstrated examples and troubleshooting techniques
* Progressive complexity building from demonstrated foundations toward independent implementation

### B2. Digital Tools for Quest Development Environment

**Primary Development Tools**:

* **VSCode Live Share** - Collaborative quest team development
* **GitHub Pages** - Hosting and deployment for quest prototypes
* **Browser Developer Tools** - API testing and debugging in real-time

**API Integration Tools**:

* **Postman Client** - Testing Flask API endpoints before JavaScript integration
* **Browser Console** - Real-time JavaScript debugging and API response inspection
* **Network Tab** - Monitoring API calls and troubleshooting connectivity issues

**Quest Development Platform**:

* **GitHub** - Quest project repositories and team collaboration
* **Jekyll/GitHub Pages** - Quest website hosting with integrated backends
* **Slack** - Real-time team communication during development sprints

**Community Showcase Tools**:

* **Guest Login Systems** - Simple authentication for Night at the Museum visitors
* **Data Visualization** - Charts and graphs to display quest interaction data
* **Mobile-Responsive Design** - Ensuring quest accessibility across devices

### B3. Practice and Feedback Mechanisms

**Instructor-Led Demonstrations**:

* **Live API Integration Examples** - Teacher demonstrates real-time API calls and response handling
* **Error Simulation and Resolution** - Instructor shows common mistakes and debugging techniques
* **Format Transformation Modeling** - Step-by-step conversion of API data into quest elements

**Guided Practice Activities**:

* **API Discovery Exercises** - Students explore available endpoints with instructor guidance
* **Scaffolded Implementation** - Graduated complexity following demonstrated patterns
* **Peer Collaboration** - Students apply demonstrated techniques in team settings

**Feedback Delivery**:

* **Real-time Instructor Support** - Immediate guidance during API integration attempts
* **Demonstrated Best Practices** - Teacher shows effective coding patterns and debugging strategies
* **Progressive Assessment** - Building from demonstrated examples to independent implementation

### B4. Assessment Methods

**Demonstration-Based Assessment**:

* **Functional Quest Prototypes** - Working applications with API integration following demonstrated patterns
* **Digital Citizenship Portfolio** - Documentation of ethical technology decisions using provided frameworks
* **Night at the Museum Presentation** - Live demonstration to community audience

**Technical Implementation Assessment**:

* **Promise-Based API Integration** - Functional fetch() chains with .then() following demonstrated examples
* **Purposeful API Design** - Strategic use of APIs for quest lessons, rewards, and progression mechanics
* **UX-Focused Data Formatting** - Transforming API inputs/outputs using demonstrated techniques
* **Error Handling and Validation** - Robust promise chains with proper .catch() based on instructor examples

**Assessment Components**:

* **Quest Functionality** (40%) - APIs enhance the quest user experience using demonstrated patterns
* **Digital Citizenship** (25%) - Ethical implementation and user data protection following provided guidelines
* **Community Presentation** (20%) - Effective communication to authentic audience
* **Technical Quality** (15%) - Clean, maintainable JavaScript code following demonstrated best practices

**Authentic Assessment Context**:

* **Real Community Audience** - Night at the Museum provides genuine user feedback
* **Industry Professional Reviews** - CTE pathway mentors evaluate technical implementation
* **Peer Quest Testing** - Students validate each other’s work before public showcase

## C. Accessibility Considerations

### Universal Design for Learning (UDL) Implementation

**Visual Accessibility**:

* **High contrast code themes** for students with visual impairments
* **Scalable text** in all development environments
* **Screen reader compatibility** with descriptive code comments
* **Alternative text** for all diagrams and visual materials

**Motor Accessibility**:

* **Voice-to-code** integration for students with limited mobility
* **Customizable keyboard shortcuts** in development environments
* **Alternative input methods** (trackpad, stylus, eye-tracking)

**Cognitive Accessibility**:

* **Chunked learning modules** to prevent cognitive overload
* **Multiple representation formats** (visual, auditory, kinesthetic)
* **Scaffolded complexity** with clear progression indicators
* **Memory aids** through consistent naming conventions and patterns

**Language and Communication**:

* **Multilingual code comments** for ESL students
* **Clear, jargon-free explanations** with technical term glossary
* **Visual code flow diagrams** to supplement verbal explanations

## D. Differentiation Strategies

### Strategy 1: **Quest Complexity Tiers**

**Novice Quest Developers**:

* **Template-based integration** with pre-built API call examples from demonstrations
* **Simple AI interactions** using basic prompt-response patterns shown in class
* **Basic data display** from database APIs with provided styling templates
* **Guided quest narrative** with structured story templates and demonstrated examples

**Intermediate Quest Developers**:

* **Custom API integrations** designed around their unique quest themes using demonstrated patterns
* **Dynamic AI conversations** with context-aware prompt engineering following instructor examples
* **Interactive data visualizations** using charts and user-generated content with provided libraries
* **Original quest mechanics** with personalized user engagement features built on demonstrated foundations

**Advanced Quest Developers**:

* **Complex API orchestration** combining multiple services for rich experiences beyond basic demonstrations
* **AI-powered quest generation** where Gemini creates dynamic content using advanced prompt techniques
* **Advanced data analytics** tracking user behavior and quest completion rates with custom implementations
* **Innovative accessibility features** and multi-modal interaction design extending demonstrated concepts

### Strategy 2: **Progressive Quest Development Phases**

**Phase 1: Design Thinking (Days 1-2)**:

* **Empathy mapping** for quest target audience and community showcase visitors
* **Problem definition** identifying how APIs can enhance quest narrative and engagement
* **Ideation sessions** brainstorming quest mechanics that leverage AI and data APIs
* **Storyboard creation** connecting user journey to technical implementation needs

**Phase 2: Skeletal Organization (Days 3-4)**:

* **Quest architecture planning** with wireframes and user flow diagrams
* **API integration points** identified within quest progression and reward systems
* **Technical feasibility validation** ensuring chosen APIs support quest vision
* **Team role distribution** for collaborative development and API specialization

**Phase 3: API Learning and Integration (Days 5-9)**:

* **Promise-based API consumption** with functional programming patterns following demonstrations
* **Purposeful AI prompt design** for quest characters, hints, and dynamic content using shown examples
* **Database API integration** for user progress, scoring, and personalization with instructor support
* **UX-focused data formatting** transforming API responses into quest experiences using demonstrated techniques

**Phase 4: Prototype Refinement (Days 10-15)**:

* **Cross-team testing** with focus on API responsiveness and user experience
* **Community showcase preparation** ensuring APIs enhance rather than distract from quest narrative
* **Performance optimization** of promise chains and API call efficiency using demonstrated best practices
* **Final polish** for Night at the Museum demonstration readiness

## E. Digital Citizenship Best Practices

### Practice 1: **Ethical AI Integration and Transparency**

**Implementation in Quest Development**:

* Students must **document AI assistance** in their quest development process
* **Attribution requirements** for AI-generated content, code, or quest narrative elements
* **Critical evaluation** of AI responses for bias, accuracy, and appropriateness
* **Responsible prompt design** that respects user privacy and cultural sensitivity

**Quest-Specific Applications**:

* Quest characters powered by AI must include **transparency disclaimers**
* Students evaluate **AI-generated quest content** for educational value and bias
* **User consent mechanisms** when quest interactions involve AI processing
* Discussion of **AI limitations** in quest design and user experience

### Practice 2: **Data Privacy and User Protection**

**Implementation in Quest Projects**:

* **Guest login systems** designed with minimal data collection
* **Clear privacy policies** explaining how quest interaction data is used
* **User consent** for data storage, especially for Night at the Museum visitors
* **Data minimization** principles - collect only data necessary for quest functionality

**Community Showcase Considerations**:

* **Anonymous interaction options** for public demonstrations
* **Parent/guardian consent** for any data collection from minor participants
* **Secure data handling** practices when community members interact with quests
* **Right to be forgotten** - easy data deletion for quest participants

### Practice 3: **Inclusive and Accessible Quest Design**

**Implementation in Community Presentation**:

* **Universal design principles** ensuring quest accessibility for diverse abilities
* **Multiple interaction methods** (keyboard, mouse, touch, voice) for quest navigation
* **Clear language and instructions** accommodating different literacy levels
* **Cultural sensitivity** in quest narratives and character design

**Community Engagement Focus**:

* **Diverse representation** in quest characters and storylines
* **Multilingual support** considerations for diverse community members
* **Accessibility testing** with community members having different abilities
* **Feedback mechanisms** that welcome diverse perspectives and experiences

## F. Quest Development Timeline and Activities

### Week 1: API Discovery and Quest Planning

* **Day 1-2**: Instructor demonstrates API exploration; students follow with guided testing
* **Day 3-4**: Design quest user stories incorporating demonstrated AI and data features
* **Day 5**: Create quest wireframes and technical implementation plan using provided templates

### Week 2: JavaScript API Integration

* **Day 1-2**: Implement Gemini AI integration following demonstrated patterns
* **Day 3-4**: Add database API calls for user progress using shown examples
* **Day 5**: Integrate guest login system with instructor-provided authentication templates

### Week 3: Quest Refinement and Community Preparation

* **Day 1-2**: User testing within quest teams following demonstrated testing protocols
* **Day 3-4**: Polish user experience using demonstrated UX enhancement techniques
* **Day 5**: **Night at the Museum** - Community showcase and authentic feedback

### Assessment Checkpoints

* **End of Week 1**: Quest concept with technical feasibility validation using provided rubric
* **End of Week 2**: Functional API integration demonstrating core features following class examples
* **End of Week 3**: Complete quest ready for community presentation meeting demonstrated standards

### Quest Showcase Framework

Students prepare their **Quest of Code** projects for authentic community engagement, demonstrating both technical skill and digital citizenship principles through interactive experiences designed for diverse audiences, following the frameworks and examples provided throughout instruction.

## G. API Integration Templates and Usage Documentation

### G1. Promise-Based API Call Templates

#### Template 1: Gemini AI Integration for Quest Characters

// Quest AI Character Interaction Template  
function questAIResponse(userInput, questContext) {  
 return fetch('/gemini', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({   
 message: userInput,   
 context: `Quest context: ${questContext}. Respond as a helpful quest guide.`   
 })  
 })  
 .then(response => response.json())  
 .then(data => formatAIForQuest(data.response))  
 .catch(error => handleQuestError(error));  
}  
  
// Check if Gemini API is available  
function checkGeminiHealth() {  
 return fetch('/gemini/health')  
 .then(response => response.json())  
 .then(status => status.healthy)  
 .catch(() => false);  
}

#### Template 2: Microblog API for Quest Interactions

// Create quest post or achievement share  
function createQuestPost(questTitle, achievement, topic) {  
 return fetch('/microblog', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({  
 title: questTitle,  
 content: achievement,  
 topic: topic  
 })  
 })  
 .then(response => response.json())  
 .then(post => displayQuestAchievement(post))  
 .catch(error => showPostError(error));  
}  
  
// Get quest-related posts for a specific page/topic  
function getQuestPosts(pageKey) {  
 return fetch(`/microblog/page/${pageKey}`)  
 .then(response => response.json())  
 .then(posts => formatQuestFeed(posts))  
 .catch(error => showFeedError(error));  
}  
  
// Add reaction to quest achievement  
function reactToQuestPost(postId, reactionType) {  
 return fetch('/microblog/reaction', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({  
 post\_id: postId,  
 reaction: reactionType  
 })  
 })  
 .then(response => response.json())  
 .then(reaction => updateReactionUI(reaction))  
 .catch(error => handleReactionError(error));  
}

### G2. Complete Quest Integration Examples

#### Example 1: AI-Powered Quest Hint System

// Quest hint system using Gemini API  
function getQuestHint(currentStep, playerProgress) {  
 const hintPrompt = `Player is stuck on ${currentStep}. Progress: ${playerProgress}.   
 Provide a helpful hint without giving away the answer.`;  
   
 return fetch('/gemini', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ message: hintPrompt })  
 })  
 .then(response => response.json())  
 .then(data => {  
 // Format AI response for quest UI  
 return {  
 hint: data.response,  
 timestamp: Date.now(),  
 step: currentStep  
 };  
 })  
 .then(formattedHint => displayHintModal(formattedHint))  
 .catch(error => showFallbackHint(currentStep));  
}

#### Example 2: Quest Achievement Sharing with Microblog

// Share quest completion with community  
function shareQuestAchievement(questName, completionTime, difficulty) {  
 // First create a post about the achievement  
 return fetch('/microblog', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({  
 title: `Quest Completed: ${questName}`,  
 content: `Finished in ${completionTime} on ${difficulty} difficulty!`,  
 topic: 'quest-achievements'  
 })  
 })  
 .then(response => response.json())  
 .then(post => {  
 // Then get recent achievements to display leaderboard  
 return fetch('/microblog/page/quest-achievements')  
 .then(response => response.json())  
 .then(achievements => updateLeaderboard(achievements, post));  
 })  
 .catch(error => handleSharingError(error));  
}

#### Example 3: Dynamic Quest Topic Creation

// Auto-create topic for new quest and populate with intro post  
function initializeQuestTopic(questId, questTitle, questDescription) {  
 return fetch('/microblog/topics/auto-create', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({  
 topic\_name: `quest-${questId}`,  
 description: `Discussion and progress for ${questTitle}`  
 })  
 })  
 .then(response => response.json())  
 .then(topic => {  
 // Create welcome post for the quest  
 return fetch('/microblog', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({  
 title: `Welcome to ${questTitle}!`,  
 content: questDescription,  
 topic: topic.name  
 })  
 });  
 })  
 .then(response => response.json())  
 .then(welcomePost => setupQuestEnvironment(welcomePost))  
 .catch(error => handleQuestSetupError(error));  
}

### G3. UX-Focused Data Formatting Examples

**Gemini AI Response Formatting for Quest Narrative**:

* **Quest Character Dialogue**: Transform Gemini responses into NPC speech bubbles with character avatars
* **Interactive Hints**: Format AI suggestions as mysterious scroll reveals or glowing text overlays
* **Dynamic Story Content**: Use Gemini to generate quest flavor text that adapts to player choices
* **Smart Help System**: Convert AI responses into contextual tooltips and progressive disclosure

**Microblog API Integration for Community Features**:

* **Achievement Gallery**: Display quest completions as visual cards with reaction counts
* **Progress Feed**: Format microblog posts into timeline showing community quest progress
* **Collaborative Hints**: Transform microblog replies into peer-to-peer help system
* **Quest Leaderboards**: Use topic-filtered posts to create dynamic ranking displays

**Promise Chain UX Patterns**:

// Smooth loading states with promise chains  
function loadQuestWithFeedback(questId) {  
 showLoadingSpinner("Preparing your quest...");  
   
 return checkGeminiHealth()  
 .then(healthy => {  
 if (!healthy) showOfflineMode();  
 updateLoadingText("Loading quest data...");  
 return fetch(`/microblog/page/quest-${questId}`);  
 })  
 .then(response => response.json())  
 .then(questData => {  
 updateLoadingText("Initializing quest environment...");  
 return formatQuestInterface(questData);  
 })  
 .then(questUI => {  
 hideLoadingSpinner();  
 return displayQuest(questUI);  
 })  
 .catch(error => {  
 hideLoadingSpinner();  
 showQuestLoadError(error);  
 });  
}

### G4. API Documentation Structure for Students

**Available API Endpoints Reference**:

**Gemini AI Endpoints**:

* POST /gemini - Send messages to AI for quest interactions
* GET /gemini/health - Check if AI service is available
* GET /gemini/debug - Development endpoint for troubleshooting

**Microblog Data Endpoints**:

* POST /microblog - Create quest posts and achievements
* POST /microblog/reply - Reply to quest discussions
* POST /microblog/reaction - React to quest posts (likes, etc.)
* GET/POST /microblog/topics - Manage quest topic categories
* GET /microblog/page/<page\_key> - Get posts for specific quest pages
* POST /microblog/topics/auto-create - Auto-generate topics for new quests

**Required Documentation Components for Students**:

1. **Promise Chain Examples** - Functional programming patterns for each endpoint
2. **Error Handling Patterns** - .catch() implementations for robust user experience
3. **UX Integration Guides** - Formatting API responses for quest engagement
4. **Testing Templates** - Browser console scripts for API validation
5. **Quest Integration Strategies** - When and how to use each API in quest design

**Student-Created Documentation**:

* **Quest API Usage Journal** - Personal documentation of API integration decisions
* **Promise Pattern Library** - Reusable code snippets for team sharing
* **UX Transformation Examples** - Before/after showing raw API data vs. quest presentation
* **Community Integration Showcase** - Examples of microblog features enhancing quest experience

## H. Resources and References

### Technical Documentation

* Flask Documentation. (2025). *Flask Web Development Framework*. Retrieved from <https://flask.palletsprojects.com/>
* Google AI. (2025). *Gemini API Documentation*. Retrieved from <https://ai.google.dev/docs>
* SQLAlchemy Documentation. (2025). *Python SQL Toolkit*. Retrieved from <https://sqlalchemy.org/>

### Code Repository

* Open Coding Society. (2025). *GitHub Pages: Unify the frontend for full-stack apps* [Source code]. GitHub. <https://github.com/Open-Coding-Society/pages>
* Open Coding Society. (2025). *Flask Portfolio Starter* [Source code]. GitHub. <https://github.com/Open-Coding-Society/flask>