Reading Generator Tool - Technical Report

Executive Summary

The Reading Generator Tool is an AI-powered content creation feature that generates comprehensive, well-structured reading materials on any given topic. It leverages OpenAI's GPT-4o-mini model to produce engaging articles with rich formatting, including markdown elements, code snippets, tables, and callouts.

---

1. Overview

Purpose

The Reading Generator Tool enables users to quickly create high-quality educational reading materials tailored to specific audiences, tones, and length requirements. It's designed to produce "awesome" content that goes beyond simple text walls by incorporating visual structure and diverse formatting elements.

Key Features

- **AI-Powered Content Generation**: Uses OpenAI GPT-4o-mini for intelligent content creation
- **Customizable Parameters**: Topic, tone, length, and target audience
- **Rich Formatting**: Automatic inclusion of markdown elements (headings, tables, code blocks, blockquotes)
- **On-Page Display**: Results shown immediately without page redirect
- **Database Integration**: Automatically saves generated content as a course in the system
- **Professional Styling**: Custom CSS for PDF-like reading experience

---

2. Technical Architecture

2.1 Backend Implementation

File: `/routes/tools.py`

Endpoint: `GET /tools/readings`

- **Purpose**: Serves the reading generator interface
- **Authentication**: Requires user session (`get_session_user`)
- **Template**: `tools/reading_generator.html`

Endpoint: `POST /tools/generate/reading`

- **Purpose**: Processes reading generation requests
- **Parameters**:
  - `topic` (required): Subject matter for the article
  - `tone` (default: "conversational"): Writing style
  - `length` (default: "medium"): Article length
  - `audience` (default: "general audience"): Target readers

- **Authentication**: Requires user session
- **Database**: Uses AsyncSession for async database operations

Generation Process

```
1. Receive form parameters (topic, tone, length, audience)
2. Construct AI prompt with specific formatting requirements
3. Call OpenAI API with JSON response format
4. Parse JSON response containing:
   - title: Article title
   - description: Summary
   - content_markdown: Full markdown content
5. Create database records:
   - Course (type: "reading")
   - CourseModule
   - Lesson (with markdown content)
6. Return JSON response with success status and data
```

AI Prompt Engineering

The prompt is carefully crafted to ensure high-quality output:

```
- Clear hierarchy with H2/H3 headings
- Bold for key concepts, italics for emphasis
- Code snippets when relevant
- Data tables for comparisons/statistics
- Blockquotes for key takeaways
- Frequent use of lists
```

Length Mapping:

- Short: ~400 words
- Medium: ~1000 words
- Long: 2000+ words

2.2 Frontend Implementation

File: `/course_material_service/templates/tools/reading_generator.html`

User Interface Components

1. **Header Section**
    - Green gradient background with texture overlay
    - Clear title and description
    - Visually distinct from other tools
2. **Input Form**

- Topic input (required)
- Tone selector (academic, conversational, simplified, professional)
- Length selector (short, medium, long)
- Audience input (optional)
- Submit button with icon

3. **Loading Overlay**
   - Animated spinner
   - Status messages
   - Backdrop blur effect

4. **Results Section**
   - Title and description display
   - Markdown-rendered content with custom styling
   - "View in Library" button

JavaScript Functionality

```
Key Functions:
- Form submission handler (async)
- Fetch API call to backend
- Error handling and user feedback
- displayReading(data): Renders markdown content
- Syntax highlighting with highlight.js
- Smooth scrolling to results
```

Custom CSS Styling

The tool includes extensive custom CSS for professional reading experience:

- **Typography**: Hierarchical heading sizes (2.25rem → 1.35rem)
- **Spacing**: Generous margins and padding
- **Tables**: Styled with green headers, alternating row colors, shadows
- **Blockquotes**: Green left border, light green background
- **Code Blocks**: Dark background (One Dark theme), syntax highlighting
- **Links**: Green color with underline offset

---

3. Data Flow

```
User Input → Frontend Form
    ↓
JavaScript Event Handler
    ↓
POST /tools/generate/reading
    ↓
```

```
OpenAI API Call (GPT-4o-mini)
    ↓
JSON Response Parsing
    ↓
Database Storage (Course/Module/Lesson)
    ↓
JSON Response to Frontend
    ↓
Markdown Rendering (marked.js)
    ↓
Syntax Highlighting (highlight.js)
    ↓
Display Results
```

---

4. Database Schema

Course Record

```
{
    title: str,              # From AI response
    description: str,        # From AI response
    user_id: int,           # Current user
    learning_outcomes: [],   # Empty array
    course_type: "reading", # Fixed type
    is_published: True      # Auto-published
}
```

Module Record

```
{
    course_id: int,         # Foreign key
    title: "Reading Material",
    order_index: 1
}
```

Lesson Record

```
{
    module_id: int,         # Foreign key
    title: str,             # Article title
    content: str,           # Full markdown content
    order_index: 1
}
```

---

5. User Experience Flow

1. **Access Tool**: User navigates to `/tools/readings`
2. **Input Parameters**:
    - Enter topic (e.g., "The impact of AI on Healthcare")
    - Select tone (conversational)
    - Choose length (medium)
    - Specify audience (optional)
3. **Submit**: Click "Generate Reading" button
4. **Loading State**: Animated overlay with status message
5. **View Results**: Scroll to rendered article with rich formatting
6. **Save to Library**: Content automatically saved, accessible via library

---

6. Strengths

Content Quality

- ✅ Rich, structured content with multiple formatting elements
- ✅ Professional appearance resembling published articles
- ✅ Consistent quality through prompt engineering

User Experience

- ✅ Immediate on-page results (no redirect)
- ✅ Smooth animations and transitions
- ✅ Clear loading states
- ✅ Responsive design

Technical Implementation

- ✅ Async/await for non-blocking operations
- ✅ Proper error handling
- ✅ JSON response format for reliability
- ✅ Database integration for persistence

---

7. Potential Improvements

Content Enhancement

1. **Export Options**: Add PDF/DOCX export functionality
2. **Image Generation**: Integrate AI image generation for article illustrations
3. **Citations**: Add option to include references/bibliography
4. **Multi-language**: Support content generation in multiple languages

User Experience

1. **Preview Mode**: Show estimated reading time
2. **Edit Capability**: Allow in-place editing before saving

3. **Templates**: Provide content structure templates
4. **Regeneration**: Option to regenerate specific sections

Technical Enhancements

1. **Caching**: Cache similar requests to reduce API calls
2. **Streaming**: Stream content as it's generated
3. **Version Control**: Save multiple versions of generated content
4. **Analytics**: Track popular topics and user engagement

---

8. Dependencies

Backend

- `fastapi`: Web framework
- `openai`: OpenAI API client
- `sqlalchemy`: Database ORM
- `jinja2`: Template engine

Frontend

- `marked.js`: Markdown parsing
- `highlight.js`: Syntax highlighting
- `TailwindCSS`: Utility-first CSS framework

Environment Variables

- `OPENAI_API_KEY`: Required for content generation

---

9. Error Handling

Backend Errors

- Missing API key → HTTP 500 with error message
- API failure → Exception caught, logged, HTTP 500 returned
- Database errors → Transaction rollback (implicit)

Frontend Errors

- Network failure → Alert with error message
- Invalid response → Console error + user alert
- Loading timeout → User can see loading state continues

---

10. Performance Considerations

Response Time

- **API Call**: 5-15 seconds (depends on length)

- **Database Write**: <1 second
- **Rendering**: <1 second

Optimization Opportunities

1. Implement request queuing for high traffic
2. Add rate limiting per user
3. Cache frequently requested topics
4. Optimize database queries with indexes

---

11. Security Considerations

Current Implementation

- ✅ User authentication required
- ✅ Form validation (required fields)
- ✅ SQL injection protection (ORM)
- ✅ XSS protection (template escaping)

Recommendations

1. Add input sanitization for topic/audience fields
2. Implement rate limiting to prevent abuse
3. Add content moderation for generated text
4. Validate content length before database storage

---

12. Usage Statistics & Metrics

Trackable Metrics

- Number of readings generated per user
- Most popular topics
- Average generation time
- Tone/length preferences
- User retention after generation

Current Limitations

- No analytics implementation
- No usage tracking
- No performance monitoring

---

13. Conclusion

The Reading Generator Tool is a well-implemented feature that successfully combines AI capabilities with user-friendly design. It produces high-quality, visually appealing reading materials that are automatically

integrated into the course management system. The tool demonstrates good software engineering practices with proper separation of concerns, error handling, and async operations.

Overall Rating: ⭐⭐⭐⭐ (4/5)

Strengths: Quality output, professional UI, solid technical foundation **Areas for Improvement**: Export options, editing capabilities, performance optimization

---

Appendix A: API Response Format

```
{
  "status": "success",
  "course_id": 123,
  "redirect_url": "/library",
  "data": {
    "title": "Article Title",
    "description": "Brief summary of the article",
    "content_markdown": "# Title\n\n## Section\n\nContent..."
  }
}
```

Appendix B: Supported Tone Options

Tone	Description	Use Case
Academic	Formal, scholarly language	Research papers, academic courses
Conversational	Friendly, engaging style	General learning, blog posts
Simplified	Easy-to-understand language	Beginners, young learners
Professional	Business-appropriate tone	Corporate training, professional development

---

Report Generated: December 24, 2025 Version: 1.0