**EduPrompt Studio - Technical Documentation**

**System Overview**

EduPrompt Studio is a research-based AI prompt generator designed specifically for educators. The application combines modern web technologies with educational theory (TPACK, UDL, Bloom's Taxonomy) to help teachers create effective AI prompts for their teaching needs.

**Architecture Diagram**

Frontend (HTML/CSS/JS)

↓

Django Backend

↓

Gemini AI API

↓

Enhanced Prompts

**Technology Stack**

* **Backend**: Django 4.x (Python)
* **Frontend**: HTML5, CSS3, JavaScript (ES6)
* **Styling**: Tailwind CSS, DaisyUI
* **AI Integration**: Google Gemini 2.5 Flash API
* **HTTP Client**: Requests library
* **Static Files**: Django static files system

**Core Components**

1. **Form Interface**: User input collection with TPACK-aligned fields
2. **Enhancement Engine**: Backend logic for theoretical prompt improvement
3. **Gemini Integration**: AI prompt generation and refinement
4. **Improvement System**: User-driven prompt optimization with suggestions

**Installation & Setup**

**Prerequisites**

* Python 3.8+
* Django 4.0+
* Google Gemini API key
* Modern web browser

**Step-by-Step Installation**

1. **Clone the repository**
2. git clone <repository-url>
3. cd promptapp
4. **Create virtual environment**
5. python -m venv venv
6. source venv/bin/activate # On Windows: venv\Scripts\activate
7. **Install dependencies**
8. pip install django requests python-decouple
9. pip install google-generativeai
10. **Environment configuration**

Create .env file in project root:

GEMINI\_API\_KEY=your\_gemini\_api\_key\_here

SECRET\_KEY=your\_django\_secret\_key

DEBUG=True

1. **Database setup**
2. python manage.py migrate
3. **Static files setup**
4. python manage.py collectstatic
5. **Run development server**
6. python manage.py runserver

**Configuration Guide**

**Django Settings**

Key settings in settings.py:

# Static files

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [BASE\_DIR / 'static']

# API Configuration

GEMINI\_API\_KEY = config('GEMINI\_API\_KEY')

# CORS settings (if needed)

CORS\_ALLOWED\_ORIGINS = [

"http://localhost:8000",

"http://127.0.0.1:8000",

]

**Gemini API Setup**

1. Get API key from [Google AI Studio](https://makersuite.google.com/app/apikey)
2. Add to environment variables
3. Configure rate limits and safety settings in views.py

**API Documentation**

**Main Endpoint**

**POST /generate/**

Generates AI prompts based on user input and enhancement preferences.

**Request Format:**

{

"prompt": "User-constructed prompt string",

"enhancement": "enhanced|basic",

"role": "Selected teacher role",

"task": "Selected task type",

"context": "Learning context",

"methodology": "Teaching methodology",

"subject": "Subject area and objectives",

"tone": "Communication tone"

}

**Response Format:**

{

"response": "Generated AI prompt text"

}

**Error Response:**

{

"error": "Error description",

"response": "Fallback message"

}

**Enhancement Types**

**Enhanced Mode**

Applies theoretical frameworks:

* **TPACK Integration**: Technology-Pedagogy-Content alignment
* **UDL Principles**: Multiple means of representation, engagement, expression
* **Bloom's Taxonomy**: Cognitive level progression
* **Age-appropriate scaffolding**: Developmental considerations
* **Subject-specific enhancements**: Discipline-based optimizations

**Basic Mode**

Returns prompt exactly as specified without theoretical enhancements.

**Prompt Improvement Endpoint**

Same endpoint (/generate/) handles improvement requests when prompt contains "prompt engineering expert".

**Improvement Response:**

{

"prompt\_improvements": "1. Suggestion one...\n2. Suggestion two...\n3. Suggestion three..."

}

**Code Structure**

**File Organization**

promptapp/

├── manage.py

├── .env

├── static/

│ └── eduPromptStudio\_logo.png

├── promptapp/

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

└── generator/

├── views.py

├── urls.py

└── templates/

└── generator/

└── index.html

**Key Functions**

**add\_theoretical\_enhancement(prompt, form\_data)**

**Purpose**: Applies educational theory-based enhancements to prompts

**Parameters**:

* prompt: Original user prompt
* form\_data: Dictionary with user selections

**Enhancement Logic**:

* **Bloom's Taxonomy**: Applied to cognitive tasks
* **UDL**: Applied when learner diversity detected
* **TPACK**: Applied to technology-enhanced tasks
* **Constructivism**: Applied to discovery-based approaches
* **Age-specific scaffolding**: Based on context selection

**Returns**: Enhanced prompt string with theoretical requirements

**generate\_prompt(request)**

**Purpose**: Main view handling prompt generation requests

**Flow**:

1. Parse JSON request data
2. Detect request type (regular/improvement/theory)
3. Apply enhancements if requested
4. Call Gemini API
5. Process and return response

**Error Handling**:

* JSON parsing errors
* API timeout handling
* Rate limit management
* Fallback responses

**Data Flow**

1. **User Input**: Form submission with educational parameters
2. **Request Processing**: Django view processes and validates data
3. **Enhancement Application**: Theoretical frameworks applied if requested
4. **AI Generation**: Gemini API call with enhanced prompt
5. **Response Processing**: JSON parsing and error handling
6. **Frontend Update**: Dynamic UI update with generated content

**Frontend JavaScript Structure**

**Core Functions**

* generateWithGemini(): Main prompt generation
* showPromptImprovements(): Improvement modal display
* applySelectedImprovements(): Apply user-selected improvements
* add\_theoretical\_enhancement(): Backend enhancement logic

**Template System**

const templates = {

template\_id: {

role: "Teacher role",

task: "Learning task",

context: "Learning context",

methodology: "Teaching approach",

tone: "Communication style",

include: "Required elements",

exclude: "Elements to avoid"

}

}

**Deployment Guide**

**Production Setup**

1. **Environment Configuration**
2. DEBUG=False
3. ALLOWED\_HOSTS=yourdomain.com,www.yourdomain.com
4. GEMINI\_API\_KEY=production\_api\_key
5. SECRET\_KEY=production\_secret\_key
6. **Static Files**
7. python manage.py collectstatic --noinput
8. **Database**
9. python manage.py migrate

**Security Considerations**

**Environment Variables**

* Store sensitive data in environment variables
* Use strong, unique SECRET\_KEY
* Secure API key storage

**Django Security Settings**

# Production settings

DEBUG = False

SECURE\_SSL\_REDIRECT = True

SECURE\_BROWSER\_XSS\_FILTER = True

SECURE\_CONTENT\_TYPE\_NOSNIFF = True

X\_FRAME\_OPTIONS = 'DENY'

**API Security**

* Rate limiting on Gemini API calls
* Input validation and sanitization
* CSRF protection enabled
* Timeout handling for external API calls

**Server Requirements**

* **Python**: 3.8+
* **Memory**: Minimum 512MB RAM
* **Storage**: 1GB minimum
* **Network**: Stable internet for API calls

**Monitoring**

**Logging Configuration**

LOGGING = {

'version': 1,

'disable\_existing\_loggers': False,

'handlers': {

'file': {

'level': 'INFO',

'class': 'logging.FileHandler',

'filename': 'django.log',

},

},

'loggers': {

'generator': {

'handlers': ['file'],

'level': 'INFO',

'propagate': True,

},

},

}

**Key Metrics to Monitor**

* API response times
* Error rates
* Enhancement adoption rates
* User session duration

**Troubleshooting**

**Common Issues**

**Gemini API Errors**

* **403 Forbidden**: Check API key validity
* **429 Rate Limited**: Implement backoff strategy
* **500 Server Error**: Check API service status

**Static Files Not Loading**

* Verify STATIC\_URL and STATICFILES\_DIRS
* Run collectstatic command
* Check file permissions

**Enhancement Not Working**

* Verify form data transmission
* Check enhancement parameter in request
* Review enhancement function logic

**Debug Mode**

Enable detailed logging:

LOGGING = {

'version': 1,

'loggers': {

'generator': {

'level': 'DEBUG',

},

},

}

**Contributing**

**Development Setup**

1. Follow installation guide
2. Enable DEBUG mode
3. Use development database
4. Run tests before committing

**Code Standards**

* Follow PEP 8 for Python
* Use meaningful variable names
* Add docstrings to functions
* Implement error handling

**Testing**

Run comprehensive tests before deployment:

* Template functionality
* Enhancement modes
* API integration
* UI responsiveness

**License**

[Add appropriate license information]

**Support**

For technical issues or questions:

* Check troubleshooting section
* Review Django and Gemini API documentation
* Ensure all dependencies are correctly installed

**EduPrompt Studio v1.0** - Professional AI Prompt Generator for Educators

ΕΓΙΝΑΝ ΑΛΛΑΓΕΣ. ΟΡΊΣΤΕ:

**EduPrompt Studio - Technical Documentation Updates**

**New Sections to Add**

**Dual-Model Architecture (Add after "Technology Stack")**

**Advanced AI Integration**

EduPrompt Studio implements a sophisticated dual-model architecture for optimal performance:

* **Primary Model**: Gemini 2.5 Flash for main prompt generation
* **Secondary Model**: Gemini 2.0 Flash for improvement analysis

**Model Selection Logic**

# Automatic model selection based on request type

if is\_improvement\_request:

url = "https://generativelanguage.googleapis.com/v1/models/gemini-2.0-flash:generateContent?key=" + api\_key

else:

url = "https://generativelanguage.googleapis.com/v1/models/gemini-2.5-flash:generateContent?key=" + api\_key

**Performance Benefits**

* **Speed Optimization**: Gemini 2.0 Flash provides faster improvement analysis
* **Reliability**: Reduced intermittent failures in improvement requests
* **Cost Efficiency**: Lower token costs for analysis tasks
* **Quality Maintenance**: Full theory enhancements preserved in main generation

**Enhanced Analytics Integration (Add after "Core Components")**

**5. Analytics Engine**: Comprehensive educational data analysis and classification

**Analytics Components**

* **PromptAnalyzer Class**: Automated content and pedagogical analysis
* **Educational Classification**: Subject, age group, methodology categorization
* **Quality Assessment**: Theory integration and innovation scoring
* **Behavioral Tracking**: User interaction and professional development patterns

**Automatic Data Collection**

Every prompt generation triggers comprehensive analytics:

# Auto-analysis integration in views.py

subject\_category = PromptAnalyzer.categorize\_subject(data.get("subject", ""))

age\_group\_category = PromptAnalyzer.categorize\_age\_group(data.get("context", ""))

methodology\_category = PromptAnalyzer.categorize\_methodology(data.get("methodology", ""))

complexity\_level = PromptAnalyzer.assess\_complexity(text\_response, data.get("task", ""), data.get("methodology", ""))

# Content analysis

content\_analysis = PromptAnalyzer.analyze\_content(text\_response)

# Behavioral analysis

innovation\_level = PromptAnalyzer.assess\_innovation\_level(data.get("methodology", ""), enhancement\_type, template\_used)

student\_centeredness\_score = PromptAnalyzer.calculate\_student\_centeredness(data.get("methodology", ""), data.get("task", ""), data.get("context", ""))

**Research Data Schema**

47+ variables tracked per interaction:

* Educational classifications (7 categories)
* Quality scores (6 metrics)
* Content analysis (8 measures)
* User behavior (12 indicators)
* Professional development tracking (14+ variables)

**Updated File Organization (Replace existing)**

promptapp/

├── manage.py

├── .env

├── static/

│ └── eduPromptStudio\_logo.png

├── promptapp/

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

└── generator/

├── views.py

├── urls.py

├── models.py # ← Extended analytics models

├── analytics.py # ← New: PromptAnalyzer class

├── admin.py # ← Enhanced visual interface

└── templates/

└── generator/

└── index.html

**Updated Troubleshooting (Replace "Troubleshooting" section)**

**Common Issues**

**Improvement Feature Problems**

* **Empty Improvements**: Fixed via dual-model architecture with Gemini 2.0 Flash
* **Truncated Content**: Resolved by removing 300-character limits in fallback responses
* **JSON Parsing Errors**: Automatic markdown cleaning for ```json blocks

**Analytics Issues**

* **Missing Data**: Ensure PromptAnalyzer runs in views.py after successful generation
* **Classification Errors**: Check pattern matching in analytics.py
* **Admin Display Issues**: Verify model fields and admin.py configuration

**Gemini API Errors**

* **403 Forbidden**: Check API key validity
* **429 Rate Limited**: Implement backoff strategy, consider model switching
* **500 Server Error**: Check API service status
* **Model Not Found**: Verify correct model names (gemini-2.5-flash, gemini-2.0-flash)

**Performance Issues**

* **Slow Improvements**: Normal with dual-model architecture (2.0 Flash is optimized for speed)
* **Memory Usage**: Analytics processing adds minimal overhead (~50ms per request)

**Enhanced API Documentation (Update existing "API Documentation")**

**Improvement Detection Logic**

# Request type detection

is\_improvement\_request = 'prompt engineering expert' in prompt.lower()

**Model-Specific Responses**

**Gemini 2.5 Flash Response** (Main Generation):

{

"response": "Enhanced educational prompt with theory integration..."

}

**Gemini 2.0 Flash Response** (Improvements):

{

"response": "```json\n{\n \"prompt\_improvements\": \"1. Suggestion...\n2. Suggestion...\n3. Suggestion...\"\n}\n```"

}

**Analytics Data Endpoint**

**GET /admin/generator/promptgeneration/**

Access comprehensive analytics through Django admin interface with:

* Color-coded educational classifications
* Progress bars for quality scores
* Interactive filtering by 15+ dimensions
* Export capabilities for research data

**Monitoring Updates (Update "Key Metrics to Monitor")**

**Enhanced Metrics**

* **Dual-Model Performance**: Track response times for each model
* **Analytics Processing**: Monitor classification accuracy and performance
* **Educational Quality**: Theory integration scores, innovation levels
* **User Engagement**: Improvement request rates, copy-to-clipboard success
* **Research Metrics**: Data completeness, export frequency

**Analytics Dashboard Access**

Monitor real-time educational analytics via:

* Django Admin: /admin/generator/promptgeneration/
* Color-coded classifications and quality indicators
* Interactive filtering and search capabilities
* Export functionality for research analysis

**Additional Dependencies (Update "Install dependencies")**

pip install django requests python-decouple google-generativeai textstat

**New Dependency**: textstat for readability analysis in PromptAnalyzer.

**Version History (Add at end)**

**v1.1 - Analytics & Dual-Model Release**

* Implemented dual-model architecture (Gemini 2.5 Flash + 2.0 Flash)
* Added comprehensive analytics system (47+ variables)
* Enhanced admin interface with visual analytics
* Fixed improvement feature reliability issues
* Integrated automated educational classification

**v1.0 - Initial Release**

* Basic prompt generation with theory enhancements
* Single-model Gemini 2.5 Flash integration
* Template system and basic UI