🎓 BLOOMS BOT - Academic Question Paper Generator

Project Overview

BLOOMS BOT is a web-based application that automatically generates balanced examination question papers from uploaded syllabus or lecture notes (PDF). It uses **Bloom's Taxonomy** to classify questions by cognitive level and ensures fair distribution across units, marks, and difficulty levels.

Developed for: Final Year Computer Science Project
Tech Stack: Python, Flask, SQLite, HTML, CSS, JavaScript

✨ Key Features

1. PDF Processing

- Upload syllabus or lecture notes in PDF format
- Automatic text extraction using PyPDF2
- Intelligent unit/module detection
- Support for files up to 16MB

2. Automated Question Generation

- Generates 8-12 questions per unit automatically
- Questions cover different mark weights (2, 5, 10 marks)
- Diverse question types across all Bloom's levels
- Context-aware question creation from document content

3. Bloom's Taxonomy Classification

Questions are automatically classified into six cognitive levels:

- **Remember**: Recall facts and basic concepts
- **Understand**: Explain ideas or concepts
- **Apply**: Use information in new situations
- **Analyze**: Draw connections among ideas
- **Evaluate**: Justify a stand or decision
- **Create**: Produce new or original work

Each question is also tagged with difficulty:

- Easy, Medium, or Hard (based on Bloom's level and marks)

4. Intelligent Paper Generation

- Specify total marks, number of questions
- Set unit-wise mark distribution
- Configure Bloom's taxonomy distribution (percentages)
- Constraint-based selection algorithm
- Automatic fallback for impossible constraints

5. Review & Edit Interface

- Visual review of generated paper
- View distribution charts (Bloom's levels, difficulty)
- Replace questions with alternatives from same category
- Remove unwanted questions
- Real-time marks recalculation

6. Export Options

- **PDF Export**: Professional formatted question paper
- **Text Export**: Plain text with full details
- Includes metadata (date, marks, statistics)

🏗️ System Architecture

Backend Modules

1. `app.py` - Main Flask application
    - Handles routing and request processing
    - Manages file uploads and sessions
    - Coordinates between modules
2. `database.py` - SQLite operations
    - Two tables: `documents` and `questions`

- CRUD operations for questions and documents
- Filter-based query system

3. **pdf_processor.py** - PDF text extraction
    - Uses PyPDF2 for text extraction
    - Cleans extracted text (whitespace, special chars)
    - Identifies unit divisions in documents
    - Extracts key concepts and topics
4. **question_generator.py** - Question creation
    - Template-based question generation
    - Uses Bloom's taxonomy templates
    - Extracts key concepts from text
    - Ensures varied question distribution
5. **blooms_classifier.py** - Taxonomy classification
    - Keyword-based Bloom's level detection
    - Heuristic fallback for ambiguous questions
    - Difficulty calculation (cognitive level + marks)
    - Validation functions for distribution
6. **paper_generator.py** - Paper assembly
    - Constraint satisfaction algorithm
    - Fitness-based question selection
    - Relaxed fallback for strict constraints
    - PDF/Text export functionality

Frontend Components

- **base.html** - Base template with navbar, footer
- **index.html** - Upload interface
- **generate.html** - Paper configuration controls
- **review.html** - Review and edit interface
- **style.css** - Responsive styling
- **script.js** - Client-side interactions

🚀 How It Works

Input Processing Flow

1. **Upload PDF**

```
User uploads PDF → File saved to uploads/
```

2. **Text Extraction**

```
PDF → PyPDF2 → Extracted text → Clean text
```

3. **Unit Detection**

```
Extracted text → Regex patterns → Units identified
- Patterns: "UNIT 1", "MODULE 1", "Chapter 1"
- Fallback: Entire document = Unit 1
```

4. **Concept Extraction**

```
Unit text → Key concept extraction:
- Capitalized technical terms
- Terms after academic markers ("define", "concept of")
- Frequently occurring nouns
```

Question Generation Process

1. **Template Selection**

```
For each unit:
- Extract 10-15 key concepts
- Select marks distribution [2, 2, 5, 5, 5, 10, 10, 10]
- Choose Bloom's level based on marks:
  * 2 marks → Remember/Understand
  * 5 marks → Understand/Apply
  * 10 marks → Apply/Analyze/Evaluate/Create
```

2. **Question Creation**

```
Template: "Explain {topic} in your own words."
Concept: "Machine Learning"
Result: "Explain Machine Learning in your own words."
```

3. **Classification**

```
Question text → Keyword matching → Bloom's level
Bloom's level + Marks → Difficulty calculation
```

Paper Generation Algorithm

```
python

1. Parse user constraints:
  - Total marks
  - Number of questions
  - Unit distribution {Unit 1: 30, Unit 2: 40, ...}
  - Bloom distribution {Remember: 15%, Understand: 25%, ...}

2. Calculate targets:
  - Bloom targets: {Remember: 15 marks, Understand: 25 marks, ...}

3. Iterative selection:
  For each position in paper:
    - Find all eligible questions (not already selected)
    - Calculate fitness score for each:
      score = unit_need_score + bloom_need_score + mark_fit_score
    - Select highest scoring question
    - Update remaining targets

4. Validation:
  - Check if ≥70% of target questions achieved
  - Check if total marks within ±20% of target

5. Fallback (if strict matching fails):
  - Use greedy algorithm with randomization
  - Allow 10% mark overage
```

Fitness Scoring

```python
fitness_score(question):
  score = 0

  # Unit need (0-10 points)
  if unit needs marks:
    score += min(10, unit_need / question_marks)

  # Bloom need (0-10 points)
  if bloom_level needs coverage:
    score += min(10, bloom_need / question_marks)

  # Mark fit (0-2 points)
  mark_fit = 1 - |remaining_marks - question_marks| / remaining_marks
  score += mark_fit * 2

  return score
```

💻 Installation & Setup

Prerequisites

- Python 3.8 or higher
- pip package manager

Steps

1. **Clone/Download the project**

```bash
  cd blooms-bot/
```

2. **Install dependencies**

```bash
```

```
pip install -r requirements.txt
```

3. **Run the application**

```
bash

python app.py
```

4. **Open in browser**

```
http://localhost:5000
```

First-Time Setup

- The application automatically creates required folders: `uploads/`, `exports/`
- SQLite database `blooms.db` is created automatically
- No manual database setup needed

📖 Usage Guide

Step 1: Upload Document

1. Go to home page
2. Select document type (Syllabus or Lecture Notes)
3. Choose PDF file (max 16MB)
4. Click "Process Document & Generate Questions"
5. Wait for processing (10-30 seconds depending on file size)

Step 2: Generate Paper

1. Navigate to "Generate" page
2. Configure basic settings:
    - Total marks (e.g., 100)
    - Number of questions (e.g., 10)
3. Set unit-wise mark distribution:

- Allocate marks to each detected unit
- Leave 0 to skip a unit

4. Configure Bloom's distribution:
    - Use sliders to set percentages
    - Must total 100%
5. Click "Generate Question Paper"

Step 3: Review & Edit

1. Review generated paper
2. Check distribution charts
3. Replace questions if needed (same criteria)
4. Remove unwanted questions
5. Verify total marks

Step 4: Export

1. Choose export format:
    - **PDF**: Professional formatted paper
    - **Text**: Plain text with statistics
2. Download file
3. Use for examination

🎯 Example Use Cases

Case 1: Mid-Term Exam (50 marks)

```
Configuration:
- Total marks: 50
- Questions: 6
- Units: Unit 1 (25 marks), Unit 2 (25 marks)
- Bloom's: Remember 20%, Understand 30%, Apply 30%, Analyze 20%

Result: Balanced 50-mark paper with 6 questions
```

Case 2: Final Exam (100 marks)

```
Configuration:
- Total marks: 100
- Questions: 10
- Units: Equal distribution across 4 units
- Bloom's: Standard distribution (15%, 25%, 25%, 20%, 10%, 5%)

Result: Comprehensive 100-mark paper covering all units
```

🔧 Technical Details

Database Schema

documents table:

```
sql

id: INTEGER PRIMARY KEY
filename: TEXT
document_type: TEXT (syllabus/lecture_notes)
extracted_text: TEXT
upload_date: TIMESTAMP
```

questions table:

```
sql

id: INTEGER PRIMARY KEY
document_id: INTEGER (Foreign Key)
unit: TEXT
question: TEXT
marks: INTEGER
bloom_level: TEXT
difficulty: TEXT
created_date: TIMESTAMP
```

Question Template Examples

Remember Level:

- "Define {topic}."
- "What is {topic}?"

- "List the main features of {topic}."

Analyze Level:

- "Analyze the structure of {topic}."
- "Compare and contrast different aspects of {topic}."
- "Examine the relationship between {topic} and related concepts."

Bloom's Level Detection

Keyword Matching:

```python
BLOOM_KEYWORDS = {
    'Remember': ['define', 'list', 'state', 'identify', 'recall'],
    'Understand': ['explain', 'describe', 'summarize', 'discuss'],
    'Apply': ['apply', 'demonstrate', 'use', 'implement', 'solve'],
    'Analyze': ['analyze', 'compare', 'contrast', 'examine'],
    'Evaluate': ['evaluate', 'assess', 'judge', 'critique'],
    'Create': ['create', 'design', 'develop', 'construct']
}
```

🚨 Limitations & Considerations

1. **PDF Quality**: Text extraction works best with text-based PDFs (not scanned images)
2. **Question Quality**: Generated questions follow templates; manual review recommended
3. **Constraint Satisfaction**: Very strict constraints may not be satisfiable
4. **Language**: Optimized for English academic content
5. **Storage**: All data stored locally; no cloud sync

🔮 Future Enhancements

- [ ] Support for scanned PDFs (OCR)
- [ ] Custom question templates
- [ ] Multi-language support
- [ ] Question bank export/import

- [ ] Duplicate question detection
- [ ] Answer key generation
- [ ] User accounts and authentication
- [ ] Question difficulty prediction using ML
- [ ] Collaborative paper creation

📝 For Viva/Demo

Key Points to Explain

1. **Bloom's Taxonomy**: Six cognitive levels from simple recall to creation
2. **Constraint Satisfaction**: How the algorithm balances multiple requirements
3. **Fitness Scoring**: How questions are selected optimally
4. **Fallback Strategy**: Relaxed constraints when strict matching fails
5. **PDF Processing**: PyPDF2 extraction and text cleaning

Demo Flow

1. Show homepage and explain workflow
2. Upload sample PDF (prepare 2-3 page syllabus)
3. Show question generation and classification
4. Configure paper with specific constraints
5. Demonstrate review interface
6. Export and show final PDF
7. Explain database structure
8. Show code highlights (blooms_classifier.py, paper_generator.py)

Common Questions

Q: How does Bloom's classification work?
A: Keyword matching against predefined lists + heuristic fallback for ambiguous cases

Q: What if constraints are impossible to satisfy?
A: System tries strict matching first, then relaxes constraints with greedy approach

Q: Can it handle scanned PDFs?
A: No, requires text-based PDFs. OCR support is future enhancement.

Q: How are questions generated?
A: Template-based approach using predefined question patterns for each Bloom's level

---

👨‍💻 Developer Information

Project Type: Final Year Computer Science Project
Domain: Educational Technology
Complexity: Medium-High
Lines of Code: ~2,500
Development Time: 4-6 weeks (estimated)

📜 License

This project is developed for academic purposes.

🤝 Acknowledgments

- Bloom's Taxonomy framework by Benjamin Bloom (1956)
- Flask web framework
- PyPDF2 for PDF processing
- FPDF for PDF generation

For any issues or questions, refer to code comments or contact the developer.

Good luck with your project submission! 🎓