**PolicyCraft - Project Progress Report**

**Date**: June 2, 2025  
**Status**: Authentication System Complete - Ready for NLP Development  
**Student**: Jacek Robert Kszczot  
**Project**: MSc AI & Data Science - COM7016 - Leeds Trinity University

**🎯 Project Overview**

**Application Name**: PolicyCraft  
**Purpose**: AI Policy Analysis for Higher Education - Strategic and Ethical Integration of Generative AI  
**Tech Stack**: Python Flask, SQLAlchemy, spaCy/NLTK, MongoDB, Plotly/Matplotlib  
**Timeline**: 3 months development (May-August 2025)  
**Goal**: Distinction level MSc project

**✅ COMPLETED TODAY (June 2, 2025)**

**1. Project Structure**

* ✅ **Complete directory tree** created and organized
* ✅ **Minimal but complete** structure (no over-engineering)
* ✅ **Separated concerns**: NLP, web, auth, database, visualization modules
* ✅ **Clean architecture** ready for development

**2. Flask Application Foundation**

* ✅ **Main app.py** - complete Flask application with routing
* ✅ **Configuration system** (config.py) with dev/prod environments
* ✅ **Error handling** - 404/500 pages
* ✅ **Logging system** configured
* ✅ **Port 5001** (avoiding macOS AirPlay conflict)

**3. Complete Authentication System**

* ✅ **User registration/login** - working forms with validation
* ✅ **Session management** - Flask-Login integration
* ✅ **Password hashing** - Werkzeug security
* ✅ **User profiles** - first/last name, institution, role
* ✅ **Protected routes** - @login\_required decorators
* ✅ **Admin user** - auto-created (admin/admin123)

**4. Database Security**

* ✅ **SQLite database** for users (authentication)
* ✅ **Secure location** - databases outside application folder
* ✅ **Structure**: ../PolicyCraft-Databases/development/policycraft\_dev.db
* ✅ **Clean application folder** - no database files in src/
* ✅ **User data separation** - each user sees only their own data

**5. Web Interface**

* ✅ **Landing page** - public page for unauthenticated users
* ✅ **Login/Register pages** - complete forms with validation
* ✅ **Dashboard** - user-specific main page after login
* ✅ **Navigation** - different menus for auth/unauth users
* ✅ **Responsive design** - mobile-friendly CSS
* ✅ **Flash messages** - success/error notifications

**6. Security Features**

* ✅ **User isolation** - filename prefixing with user\_id
* ✅ **File access control** - users can only access their own files
* ✅ **Session security** - proper Flask-Login configuration
* ✅ **CSRF protection** disabled in dev, ready for prod
* ✅ **Secure file uploads** - filename sanitization

**7. Policy Document Collection**

* ✅ **6 university policies** collected and saved to data/policies/pdf\_originals/:
  + Stanford University (restrictive approach)
  + MIT (guidance-based approach)
  + Oxford University (educational focus)
  + Cambridge University (communication-focused)
  + Imperial College (comprehensive hub)
  + Columbia University (corporate-style)
* ✅ **Diverse approaches** for training NLP models
* ✅ **Ready for analysis** - good variety of policy styles

**📁 Current File Structure**

PolicyCraft/

├── app.py # ✅ Main Flask application

├── config.py # ✅ Secure configuration

├── requirements.txt # ✅ All dependencies

├── README.md # ✅ Project documentation

├── src/

│ ├── auth/ # ✅ Complete authentication system

│ │ ├── models.py # ✅ User model, database

│ │ ├── routes.py # ✅ Login/register/logout

│ │ └── forms.py # ✅ WTForms validation

│ ├── web/

│ │ ├── templates/ # ✅ All HTML templates

│ │ │ ├── base.html # ✅ Navigation, flash messages

│ │ │ ├── auth/ # ✅ Login/register templates

│ │ │ ├── public/ # ✅ Landing page, about

│ │ │ └── errors/ # ✅ 404/500 pages

│ │ └── static/style.css # ✅ Complete responsive CSS

│ ├── nlp/ # 🚧 PLACEHOLDER - ready for development

│ │ ├── text\_processor.py # 🚧 TODO: PDF extraction

│ │ ├── theme\_extractor.py # 🚧 TODO: spaCy/NLTK themes

│ │ └── policy\_classifier.py # 🚧 TODO: ML classification

│ ├── database/ # 🚧 PLACEHOLDER - MongoDB operations

│ │ └── operations.py # 🚧 TODO: Analysis data storage

│ ├── recommendation/ # 🚧 PLACEHOLDER - Policy suggestions

│ │ └── engine.py # 🚧 TODO: Rule-based recommendations

│ └── visualisation/ # 🚧 PLACEHOLDER - Charts/dashboard

│ └── charts.py # 🚧 TODO: Plotly/Matplotlib

├── data/

│ ├── policies/pdf\_originals/ # ✅ 6 university AI policies

│ ├── processed/ # 📁 Ready for processed data

│ └── results/ # 📁 Ready for analysis results

└── logs/ # ✅ Application logging

../PolicyCraft-Databases/ # ✅ SECURE - outside application

├── development/

│ └── policycraft\_dev.db # ✅ User authentication database

├── production/ # 📁 Ready for production

└── backups/ # 📁 Ready for backup strategy

**🔧 Technical Implementation Details**

**Dependencies Installed**

Flask==3.0.0, Flask-Login>=0.6.3, Flask-SQLAlchemy>=3.1.0

Flask-WTF>=1.2.0, WTForms>=3.1.0, Flask-Bcrypt>=1.0.1

spacy>=3.7.0, nltk>=3.8.1, PyPDF2>=3.0.1, pdfplumber>=0.10.0

pandas>=2.1.0, numpy>=1.24.0, matplotlib>=3.7.0, plotly>=5.17.0

pymongo>=4.5.0, python-docx>=1.1.0

**Current Routes**

* / - Landing page (redirects to dashboard if authenticated)
* /auth/login - User login
* /auth/register - User registration
* /auth/logout - User logout
* /auth/profile - User profile page
* /dashboard - User dashboard (protected)
* /upload - File upload (protected)
* /analyse/<filename> - Document analysis (protected)
* /about - Public about page

**Database Schema**

**Users Table** (SQLite):

* id, username, email, password\_hash
* first\_name, last\_name, institution, role
* is\_active, is\_verified, created\_at, last\_login

**🚧 NEXT IMMEDIATE TASKS (Priority Order)**

**1. NLP Text Processor (CRITICAL - Next Session)**

**File**: src/nlp/text\_processor.py **Goal**: Extract text from uploaded PDF policy documents

**Implementation needed**:

def extract\_text\_from\_file(self, file\_path):

# Use PyPDF2 or pdfplumber to extract text from PDF

# Handle different file formats (PDF, DOCX, TXT)

# Return clean text string

def clean\_text(self, text):

# Remove extra whitespace, special characters

# Normalize text for NLP processing

# Return cleaned text

**Test with**: Stanford University policy PDF

**2. Theme Extraction (HIGH Priority)**

**File**: src/nlp/theme\_extractor.py **Goal**: Identify key themes in policy documents using NLP

**Implementation needed**:

def extract\_themes(self, text):

# Use spaCy for named entity recognition

# TF-IDF for key term extraction

# Return list of themes: ["academic integrity", "transparency", etc.]

**Expected themes**: ethics, plagiarism, transparency, academic integrity, assessment, etc.

**3. Policy Classification (HIGH Priority)**

**File**: src/nlp/policy\_classifier.py **Goal**: Classify policies as restrictive/permissive/balanced

**Implementation needed**:

def classify\_policy(self, text):

# Analyze policy language

# Rule-based or ML classification

# Return: "Restrictive", "Permissive", or "Balanced"

**4. Database Operations (MEDIUM Priority)**

**File**: src/database/operations.py **Goal**: Store and retrieve analysis results per user

**Implementation needed**:

def store\_user\_analysis\_results(self, user\_id, \*\*kwargs):

# Store analysis in MongoDB or SQLite

# Associate with user\_id for data separation

def get\_user\_analyses(self, user\_id):

# Retrieve all analyses for specific user

**🎯 End-to-End Flow TO IMPLEMENT**

1. **User uploads PDF** → Upload form saves to data/policies/pdf\_originals/
2. **Text extraction** → TextProcessor.extract\_text\_from\_file()
3. **Text cleaning** → TextProcessor.clean\_text()
4. **Theme extraction** → ThemeExtractor.extract\_themes()
5. **Policy classification** → PolicyClassifier.classify\_policy()
6. **Store results** → DatabaseOperations.store\_user\_analysis\_results()
7. **Display results** → templates/results.html with charts
8. **Generate recommendations** → RecommendationEngine.generate\_recommendations()

**🔍 Testing Strategy**

**Phase 1: Basic NLP Testing**

* Upload Stanford policy PDF
* Verify text extraction works
* Check theme extraction output
* Validate classification result

**Phase 2: Multi-document Testing**

* Test all 6 university policies
* Compare classification results
* Verify theme consistency
* Check user data separation

**Phase 3: Full System Testing**

* End-to-end user workflows
* Dashboard visualization
* Recommendation generation
* Performance testing

**📊 Success Metrics for Next Session**

1. ✅ **Text extraction working** - can read PDF and extract text
2. ✅ **Basic themes identified** - extract meaningful policy themes
3. ✅ **Classification working** - categorize policy approach
4. ✅ **Results displayed** - show analysis on results page
5. ✅ **User data separation** - each user sees only their analyses

**🔑 Key Commands for Development**

**Start application**:

cd PolicyCraft

python app.py

# Access: http://localhost:5001

# Login: admin/admin123

**Dependencies installation**:

pip install Flask Flask-Login Flask-SQLAlchemy Flask-WTF PyPDF2 spacy

python -m spacy download en\_core\_web\_sm

**Database info**:

python config.py # Shows secure database locations

**💡 Known Issues & Solutions**

1. **Port 5000 conflict** → ✅ SOLVED: Using port 5001
2. **current\_user undefined** → ✅ SOLVED: Context processor added
3. **Database location** → ✅ SOLVED: Secure external location
4. **File upload security** → ✅ SOLVED: User-specific filenames

**🎓 Academic Context**

* **Programme**: MSc Artificial Intelligence and Data Science
* **Module**: COM7016 - Project
* **University**: Leeds Trinity University
* **Supervisors**: Dr Xin Lu (Primary), Dr Yashar Baradaranshokouhi (Secondary)
* **Assessment**: 40% Artefact + 40% Report + 10% Plan + 10% Oral Exam
* **Target**: Distinction level (70%+)

**STATUS**: ✅ **READY FOR NLP DEVELOPMENT**  
**NEXT SESSION FOCUS**: Build real text processing and theme extraction  
**CURRENT BLOCKERS**: None - clear path forward  
**CONFIDENCE LEVEL**: High - solid foundation established