Technical Specification: AI Module for Automating Public Space Usage Requests (ZUVP)

1. Overview & Project Objective

The goal is to develop a **Flask application** to fully automate the processing of applications for "Special Use of Public Space" (e.g., placing containers, excavation work). The system must handle unstructured inputs, extract key data using AI, automatically generate official documents (Consent/Permit), and prepare the necessary payment instructions.

2. Process Architecture (Pipeline)

The application processing is divided into four main stages, forming the automated workflow.

1. **Ingestion: Receiving input data (local folder, file uploads).**
2. **Extraction & Analysis (AI Core):** Optical Character Recognition (OCR) and entity extraction using an LLM.
3. **Generation:** Creating official PDF documents based on templates.
4. **Action:** Draft creation and preparation for distribution.

---

3. Detailed Functional Requirements

A. Ingestion Module

Requirement	**Description**
Supported Formats	PDF, JPG, PNG, DOCX, TXT.
Input Sources	1. Local folder: Monitoring Zadosti/ folder for new files. 2. REST API Endpoint: For direct file uploads**.**
Key Requirement (OCR)	The system must effectively handle **handwritten forms** (scanned images/photos).

B. Extraction & Analysis (AI Core)

This phase is the core AI engine that transforms unstructured input data into structured output.

- **Entity Extraction:** Applicant name, Company ID, Contact details, Purpose of use, Specific **Location** (address/plot number), **Duration** (Start-End date)

C. Document Engine (sample form is here)

1. Permit/Consent Generation

The system populates predefined templates (DOCX/HTML) with the data extracted in the AI phase:

- Populating **Applicant Details**.
- Inserting the **Duration and Purpose** of use.
- Appending standard **legal conditions/boilerplate text**.

2. Fee Calculation

The calculation of the total fee is governed by the following logic:

$$\text{Fee} = \text{Area}_{\text{sqm}} \times \text{Duration}_{\text{days}} \times \text{Rate}_{\text{per sqm/day}}$$

- **Rate:** Must be configurable (e.g., 10 CZK/m^2/day).

3. Payment Instructions

- Generation of a **Variable Symbol (VS)** linked to the $request\_id$.
- Formatting bank account details, amount, and VS into a structured format.

D. Output & Notification

- **Draft Mode (Recommended):** The AI system **does not send** emails directly to the public. Instead, it creates an internal **"Draft"** entry (or sends an email to the clerk) containing all generated files. The clerk reviews the draft and confirms the sending by clicking "Approve/Send".
- **Email Content Construction:**
  - **Body:** "Please find the Consent attached..." + "Payment Instructions: Account X, VS Y, Amount Z".
  - **Attachments:** Generated Consent (PDF) + Payment Info (PDF).

---

4. Recommended Tech Stack

Area	**Recommended Technology**	**Reason / Note**
Programming Language	**Python**	Standard for AI, Data Science, and automation.
AI Model (LLM/Vision)	**Nebius AI (Qwen/Qwen3-235B-A22B-Thinking-2507 for LLM, Qwen/Qwen2.5-VL-72B-Instruct for Vision)**	Highly recommended for its excellent ability to read handwriting, analyze image attachments (maps), and reliable single-pass extraction.
Orchestration	**requests (direct API calls)**	Direct Nebius API calls (OpenAI compatible endpoint).
Document Handling	**pdf2image**	Converting PDF inputs for Vision models.
Output Generation	**python-docx**	Generating documents from DOCX templates.
Database	**File-based cache**	Caching Vision API results and extracted text (vision_cache/, extracted_text_cache/).
Deployment	**Docker + docker-compose**	Application containerization for easy deployment.

5. Evaluation Criteria

- **Simplicity** (lines of code — the fewer, the better)
- **Maintainability** (basic principles, folder organization, naming conventions, clean functions, DRY)

- **Performance and observability** (a robust yet simple logging system)
- **Big bonus:** known system metrics, failure modes, pitfalls, and limitations, possible improvements, parallelization