**Practical Assignment — Web Service Development (Flask / FastAPI)**

In this assignment, you will design and implement a **small web service or API server** using **Python** with either **Flask** or **FastAPI**.  
The goal is to apply your Python programming skills to build a functional backend application with a clear structure, input validation, and simple persistent storage using only the **Python standard library** (e.g., json, csv, difflib, os).

You may work **individually** or in **groups of 2–3 students**.  
Working in a team is encouraged, as it allows for task distribution (e.g., backend logic, testing, documentation) and better project planning. If working in a group, make sure to **document team roles** in your README.md.

All data must be stored **without external databases**.  
Only **in-memory structures** or files (JSON/CSV/plain text) are allowed.  
No external Python packages or frameworks may be used beyond Flask or FastAPI.

At the end of the project, each team must provide:

* A working web service with a set of clearly defined endpoints;
* A short README.md with installation instructions, API description, and team member roles;

## 1) Mini Kanban API

**Goal:** Manage boards, lists, and cards.  
**Data:** In-memory structures; optional JSON dump to file on shutdown.

* Endpoints: create/list boards; add lists to a board; CRUD cards; move card between lists.
* Filters: list cards by status/assignee (query params).
* Auth: header X-API-Key.

## 2) Personal Expense API

**Goal:** Track expenses/income and simple summaries.  
**Data:** Append to a CSV file; read back for reports.

* Endpoints: add transaction; list by date range/category; monthly total; category totals.
* Validation: amount (float), date (ISO), category (string from a fixed set).

## 3) Simple Event Sign-Up

**Goal:** Register attendees for events with capacity limits.  
**Data:** JSON file per event (events/event\_id.json).

* Endpoints: create event (title, capacity, date); register attendee (name, email); list attendees; remaining seats.
* Duplicate check by email per event.

## 4) Recipe & Meal Plan API

**Goal:** Save recipes and generate a weekly shopping list.  
**Data:** Recipes/meal plans in JSON files.

* Endpoints: CRUD recipes (name, ingredients[], steps[]); create 7-day meal plan by recipe IDs; GET aggregated shopping list.

## 5) Micro-Learning Mini LMS

**Goal:** Lessons with multiple-choice quizzes and progress.  
**Data:** Lessons/quizzes in JSON; user progress in a separate JSON.

* Endpoints: list lessons; get quiz; submit answers → score; get progress summary.

## 6) Image Annotation (Metadata Only)

**Goal:** Label images with bounding boxes (no CV libs).  
**Data:** Store file paths; annotations as JSON {x,y,w,h,label}.

* Endpoints: upload image (save to /uploads); fetch next unlabeled; POST annotations; list/export annotations (JSON).

## 7) IoT Readings Collector

**Goal:** Accept time-series readings and query aggregates.  
**Data:** Append newline-delimited JSON (readings.ndjson).

* Endpoints: register device (returns simple token string you generate); POST readings (timestamp,value,device); GET min/max/avg over time range.

## 8) Book Listings & Swaps

**Goal:** List used books and request swaps.  
**Data:** JSON files for users, books, requests.

* Endpoints: add book (title, author, owner); search (title/author substring); create swap request; owner accepts/declines; list user’s requests.

## 9) Versioned Notes

**Goal:** Notes with automatic version history and diffs (line-based).  
**Data:** A folder per note with v1.txt, v2.txt, …

* Endpoints: create note; update note (new version saved); get latest; list versions; get simple line-diff (use difflib from stdlib).

## 10) Minimal Headless Shop

**Goal:** Products, carts, and orders (mock payment).  
**Data:** Products in JSON; carts in memory (expire after N minutes of inactivity).

* Endpoints: list products; add/remove cart item; checkout → create order (status paid/failed randomly or via a query flag); inventory decrement.