

WYTYCZNE REALIZACJI PROJEKTU

„Inteligentne systemy internetowe”

1. Architektura ogólna

Niezależnie od wybranego tematu, każdy projekt powinien być zbudowany modułowo, z podziałem na trzy spójne warstwy:

Warstwa	Zakres odpowiedzialności
Dane	wczytywanie (CSV/API), czyszczenie, preprocess, walidacja
Model/Logika	algorytmy ML, obliczenia statystyczne, scraping, rekomendacje
Prezentacja	interfejs użytkownika, wizualizacje, formularze, raporty

Takie rozdzielenie ułatwia testowanie, utrzymanie i dalszy rozwój projektu oraz umożliwia wymianę poszczególnych komponentów bez ingerencji w resztę.

2. Uniwersalny stos technologiczny (open source)

2.1 Język i środowisko pracy

Element	Rekomendacja
Język	Python 3.11+ (CPython)
Środowisko IDE	VS Code / PyCharm Community / JupyterLab
Zarządzanie pakietami	pip + venv, Poetry, conda (Miniconda)
Kontrola wersji	Git + GitHub/GitLab/Gitea
Konteneryzacja (opcjonalna)	Docker, docker-compose

2.2 Rdzeń danych i uczenia maszynowego

- **pandas + numpy** – przetwarzanie i analiza danych
- **scikit-learn** – klastrowanie, klasyfikacja, regresja, preprocessing
- **SciPy** – statystyka i funkcje naukowe (np. pearsonr, spearmanr, kendalltau)
- **joblib / pickle** – zapis i wczytywanie modeli

2.3 Wizualizacja

- **matplotlib + seaborn** – wykresy statyczne, heatmapy, clustermapy
- **plotly** – wykresy interaktywne (2D/3D)
- Pokrycie typów wykresów: scatter, box, histogram, line, kde, pairplot, bar, violin, macierze korelacji

2.4 Interfejs użytkownika (UI)

- **Shiny for Python** (+ *shinywidgets*) – szybkie aplikacje analityczne
- **Streamlit** lub **Plotly Dash** – alternatywy o podobnym przeznaczeniu
- **React / Vue.js** – jeśli projekt wymaga pełnoprawnego frontendu webowego
- **Bootstrap / Tailwind** – stylizacja i responsywność

2.5 Backend i API

- **Flask / FastAPI** – lekki backend REST
- **Gunicorn / Uvicorn** – serwer produkcyjny
- **JWT, flask-login, authlib** – autoryzacja i sesje użytkowników

2.6 Bazy danych

- **SQLite** – wbudowana, zero konfiguracji, idealna na start
- **PostgreSQL** – pełna baza relacyjna
- **MongoDB** – dane nienadzorowane (logi aktywności) – NoSQL
- **SQLAlchemy / Pony ORM** – mapowanie obiektowo-relacyjne

2.7 Interpretacja modeli

- **SHAP** – summary plot, force plot
- **LIME** – lokalne wyjaśnianie predykcji
- Zalecane szczególnie tam, gdzie wynik ma być uzasadniony użytkownikowi

2.8 Pozyskiwanie danych ze źródeł zewnętrznych

- **requests + BeautifulSoup4** – proste scrapowanie HTML
- **Scrapy** – rozbudowane, wydajne crawle
- **feedparser** – odczyt kanałów RSS
- **APScheduler / Celery** (+ Redis) – harmonogramowanie pobierania treści

2.9 Jakość kodu i testy

- **pytest** – testy jednostkowe i integracyjne
- **black, ruff, mypy** – formatowanie, linting, kontrola typów
- **pre-commit** – automatyczna kontrola przed commitem

- `requirements.txt` / `poetry.lock` – powtarzalność środowiska
-

3. Wspólne funkcjonalności wszystkich projektów

Niezależnie od specyfiki tematu, każdy projekt powinien zapewniać:

1. **Obsługę formatu CSV** – wczytanie zbioru i zapis wyników (pandas).
 2. **Interaktywne parametry** – hiperparametry/metody udostępniane w UI (slidery, pola liczbowe, menu wyboru) zamiast sztywnych wartości w kodzie.
 3. **Przypisywanie nowych przypadków** – dwa tryby:
 - generowany formularz (ręczne wpisanie cech),
 - wczytanie pliku CSV z nowymi danymi,
 - dla nowych obserwacji: `predict()` (KMeans) lub `NearestNeighbors` (DBSCAN).
 4. **Walidację i ocenę modeli** – train/test split, cross-validation, metryki dostosowane do zadania (accuracy, precision, recall, F1, AUC, `silhouette_score`, `calinski_harabasz_score`, `davies_bouldin_score`).
 5. **Wizualizacje wyników** – każda operacja kończy się czytelną grafiką (wykresy, heatmapy, drzewa, dashboardy).
 6. **Persystencję** – możliwość zapisu modeli, wyników i podzbiorów danych.
-

4. Rekomendacje dla poszczególnych tematów

Temat 1 – Klastrowanie danych (k-means, DBSCAN...)

- **UI:** Shiny for Python + shinywidgets
- **Algorytmy:** `sklearn.cluster` (KMeans, DBSCAN, `AgglomerativeClustering`, `GaussianMixture`, `OPTICS`)
- **Metryki:** `silhouette_score`, `calinski_harabasz_score`, `davies_bouldin_score`
- **Wizualizacja:** `plotly` (2D/3D), PCA do redukcji wymiarów
- **Format danych:** `pandas.read_csv`, `st.file_uploader`

Temat 2 – Korelacja danych różnymi miarami

- **Biblioteka:** `scipy.stats` (`pearsonr`, `spearmanr`, `kendalltau`, `pointbiserialr`)
- **Wizualizacja:** `seaborn/matplotlib` (heatmapa korelacji, `clustermap`)
- **Selekcja zmiennych:** analiza macierzy korelacji + zapis podzbioru do CSV (`pandas.to_csv`)

Temat 3 – Drzewa decyzji

- **Modele:** DecisionTreeClassifier/Regressor, RandomForest, GradientBoosting
- **Wizualizacja drzew:** plot_tree, dtreeviz, graphviz
- **Hiperparametry:** max_depth, min_samples_split, criterion, ccp_alpha (pruning)

Temat 4 – Sieci neuronowe

- **Podstawowe:** sklearn.neural_network.MLPClassifier/MLPRegressor
- **Rozszerzenie (opcjonalnie):** PyTorch lub TensorFlow/Keras (licencje BSD/Apache)
- **Optymalizacja:** GridSearchCV / RandomizedSearchCV, validation curves

Temat 5 – System predykcji zdrowia

- **Frontend:** React (create-react-app/Vite) lub Bootstrap
- **Backend:** Flask lub FastAPI
- **Modele:** LogisticRegression, RandomForest; porównanie metryk (accuracy, precision, recall, F1, AUC)
- **Interpretacja:** SHAP, LIME
- **Baza danych:** SQLite (wbudowana) lub PostgreSQL + SQLAlchemy
- **Zbiór przykładowy:** Pima Indians Diabetes Database (diabetes.csv)

Temat 6 – System rekomendacji artykułów

- **Frontend:** React lub Vue.js (Nuxt)
- **Backend:** Flask / FastAPI
- **Baza danych:** MongoDB (logi aktywności) lub PostgreSQL
- **Metody ML:** collaborative filtering (cosine similarity, matrix factorization), PCA/TruncatedSVD, NearestNeighbors
- **Dane testowe:** MovieLens lub własna macierz użytkownik–artykuł

Temat 7 – Wizualizacja danych

- **Biblioteki:** plotly express, matplotlib, seaborn
- **Typy wykresów:** scatter, box, histogram, bar, line, kde, pairplot, violin, correlation heatmap
- **Eksport:** PNG/SVG/HTML

Temat 8 – Inteligentny agregator informacji

- **Web scraping:** Scrapy, BeautifulSoup4, feedparser (RSS), requests
- **Klasyfikacja tekstu:** TfidfVectorizer + LogisticRegression/RandomForest/NB
- **Wykrywanie duplikatów:** simhash / hashowanie wektorów TF-IDF, cosine similarity
- **Dashboard trendów:** Flask + Chart.js / Plotly Dash / Streamlit
- **Harmonogram:** APScheduler / Celery (+Redis)

5. Sugerowany stos uniwersalny (do szybkiego startu)

Python 3.11 + Shiny for Python (tematy 1–4, 7)

lub

React / Vue + Flask / FastAPI + scikit-learn + SQLite (tematy 5, 6, 8)

Zakres	Narzędzia
UI interaktywne (analizy)	Shiny for Python, Streamlit, Plotly Dash
API backend	Flask, FastAPI, Django REST Framework
Bazodanowe	SQLite, PostgreSQL, MongoDB
Modelowanie	scikit-learn, SciPy
Wizualizacja	plotly, seaborn, matplotlib
Interpretacja	SHAP, LIME
Scraping	Scrapy, BeautifulSoup, feedparser
Serwer/autoryzacja	Gunicorn, JWT, flask-login/authlib

6. Uniwersalne dobre praktyki

- Stosować **modularną strukturę repozytorium** (np. data/, models/, app/, tests/, docs/).
 - Publikować dokumentację w **Markdown** (README) z opisem architektury i instrukcją uruchomienia.
 - Ustalać stałe **SEED** dla powtarzalności wyników eksperymentów.
 - Logować kluczowe kroki działania aplikacji (biblioteka logging).
 - Pokrywać najważniejsze ścieżki testami **pytest**.
 - Przechowywać dane wejściowe i wyniki zgodnie z zasadami bezpieczeństwa (brak danych osobowych w repozytorium, hasła w zmiennych środowiskowych).
 - Generować requirements.txt (pipreqs) lub używać poetry.lock dla zapewnienia powtarzalności środowiska.
-

7. Schemat wdrożenia projektu

7. Analiza wymagań i wybór konkretnej funkcjonalności głównej.
8. Przygotowanie środowiska (Python + venv/Poetry + Git).
9. Obróbka danych wejściowych (czyszczenie, preprocess, EDA).
10. Budowa prototypu modelu/algorytmu (notebook Jupyter na start).
11. Implementacja warstwy prezentacji i połączenie z modelem.

12. Testy, optymalizacja parametrów, interpretacja wyników.
 13. Dokumentacja i przygotowanie instrukcji uruchomienia.
-

8. Podsumowanie

Wszystkie wskazane technologie są **open source** (licencje MIT, BSD, Apache 2.0) i nie generują kosztów licencyjnych, co czyni je odpowiednimi do projektów studenckich. Całościowe podejście zakłada jeden spójny ekosystem oparty na języku Python z bibliotekami **scikit-learn**, **pandas**, **numpy** (warstwa analityczna), **matplotlib/seaborn/plotly** (wizualizacje), interfejsem **Shiny for Python / Streamlit** lub **React/Vue + Flask/FastAPI**, bazą **SQLite/PostgreSQL/MongoDB** oraz modułami **SHAP/LIME** i **Scrapy/BeautifulSoup** do wyspecjalizowanych zadań. Takie ujednolicone narzędzia zmniejszają koszt nauki, upraszczają organizację pracy zespołu i zapewniają spójność projektu niezależnie od wybranego zagadnienia.