Разработка инструментов для обработки неструктурированных данных

Разрабатываемый продукт предназначен для автоматизации работы с **неструктурированными данными** (текст, документы, сообщения). Ключевые задачи:

- **Токенизация** — разбиение текста на смысловые единицы (слова, предложения).
- **Классификация** — определение категории текста (например, спам/не спам).
- **Извлечение сущностей** — идентификация именованных объектов (даты, имена, организации).
- **Хранение и экспорт результатов** — сохранение обработанных данных в структурированном формате (JSON, CSV).

Цель: Создание универсального инструмента для анализа текстовых данных с поддержкой настройки алгоритмов и интеграции с внешними системами.

Процесс генерации диаграмм

Диаграммы созданы в PlantUML для визуализации архитектуры и логики системы.

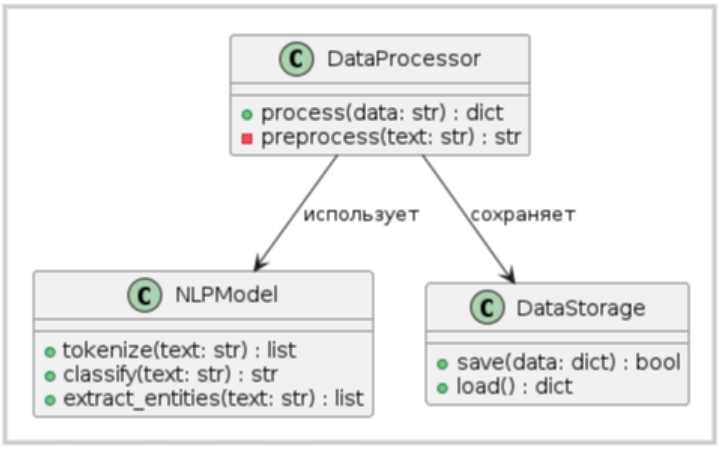
1. **Диаграмма вариантов использования (Use Case)**

Целью было отобразить взаимодействие пользователей с системой.

В процессе определены роли (User, Admin) и их цели (загрузка данных, обработка, настройка моделей).

- @startuml
- left to right direction
- actor User
- actor Admin
-
- rectangle "Обработка неструктурированных данных" {
- User --> (Загрузить данные)
- User --> (Обработать текст)
- (Обработать текст) --> (Токенизация)
- (Обработать текст) --> (Классификация)
- (Обработать текст) --> (Извлечение сущностей)
- User --> (Экспорт результатов)
-
- Admin --> (Настроить параметры)
- Admin --> (Обновить модели)
- }
- @enduml

2. **Диаграмма классов (Classes)**



Целью было показать структуру системы и связи между компонентами.

В процессе выделены ключевые классы:

DataProcessor — управляет обработкой данных.

NLPModel — реализует NLP-алгоритмы.

DataStorage — отвечает за сохранение результатов.

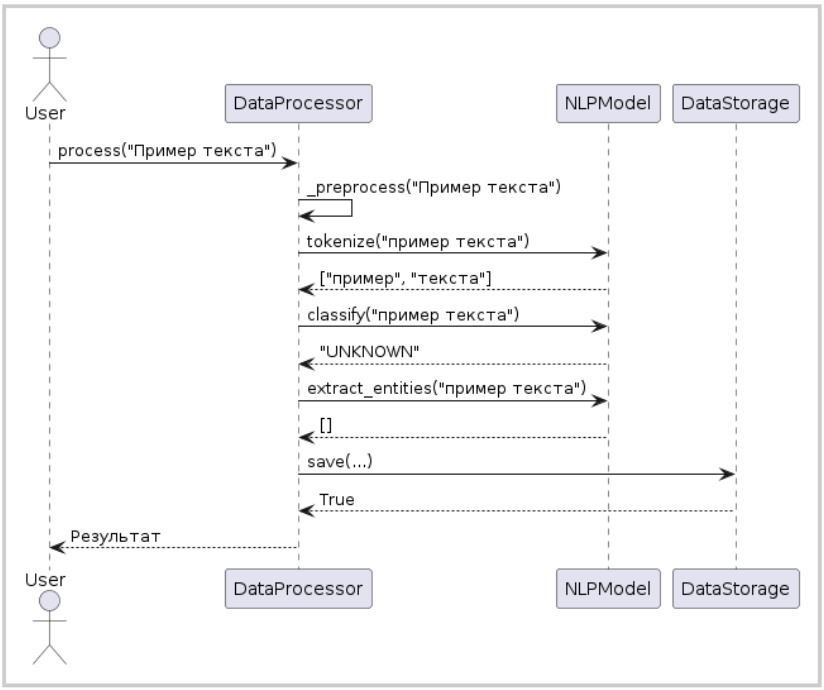
- @startuml
- class DataProcessor {
- +process(data: str) : dict
- -preprocess(text: str) : str
- }
-
- class NLPModel {
- +tokenize(text: str) : list
- +classify(text: str) : str
- +extract_entities(text: str) : list
- }
-
- class DataStorage {
- +save(data: dict) : bool
- +load() : dict
- }
-
- DataProcessor --> NLPModel : использует
- DataProcessor --> DataStorage : сохраняет
- @enduml

На основе диаграммы написан Python-код, реализующий логику классов:

- class DataProcessor:
- def __init__(self, model, storage):
- self.model = model
- self.storage = storage
-
- def process(self, data):
- processed_text = self._preprocess(data)
- tokens = self.model.tokenize(processed_text)
- category = self.model.classify(processed_text)
- entities = self.model.extract_entities(processed_text)
- result = {"tokens": tokens, "category": category, "entities": entities}
- self.storage.save(result)
- return result
-
- def _preprocess(self, text):
- return text.strip().lower()
-
- class NLPModel:
- def tokenize(self, text):
- return text.split()
-
- def classify(self, text):

- return "UNKNOWN"
-
- def extract_entities(self, text):
- return []
-
- class DataStorage:
- def save(self, data):
- print(f"Сохранено: {data}")
- return True
-
- def load(self):
- return {}

3. **Диаграмма последовательности (Sequence)**



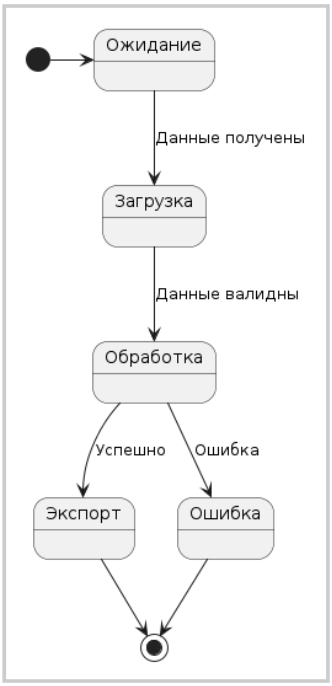
Целью было визуализировать пошаговый процесс обработки данных.

В процессе описаны этапы: препроцессинг, токенизация, классификация, извлечение сущностей, сохранение.

- @startuml
- actor User
- participant DataProcessor
- participant NLPModel
- participant DataStorage
-
- User -> DataProcessor: process("Пример текста")

- DataProcessor -> DataProcessor: _preprocess("Пример текста")
- DataProcessor -> NLPModel: tokenize("пример текста")
- NLPModel --> DataProcessor: ["пример", "текста"]
- DataProcessor -> NLPModel: classify("пример текста")
- NLPModel --> DataProcessor: "UNKNOWN"
- DataProcessor -> NLPModel: extract_entities("пример текста")
- NLPModel --> DataProcessor: []
- DataProcessor -> DataStorage: save(...)
- DataStorage --> DataProcessor: True
- DataProcessor --> User: Результат
- @enduml

4. **Диаграмма состояний (State)**



Целью было показать жизненный цикл данных в системе.

В процессе определены состояния: ожидание, загрузка, обработка, экспорт, ошибка.

- @startuml
- [*] -> Ожидание
- Ожидание --> Загрузка : Данные получены
- Загрузка --> Обработка : Данные валидны
- Обработка --> Экспорт : Успешно
- Обработка --> Ошибка : Ошибка
- Ошибка --> [*]
- Экспорт --> [*]
- @enduml

5. **Диаграмма деятельности (Activity)**

Целью было детализировать алгоритм обработки текста.

В процессе описана последовательность действий: загрузка данных → валидация → NLP-обработка → экспорт.

- @startuml
- start
- :Загрузить данные;
- if (Данные валидны?) then (Да)
- :Токенизация текста;
- :Классификация текста;
- :Извлечение сущностей;
- :Экспорт результатов;
- stop
- else (Нет)
- :Вывести ошибку;
- stop
- @enduml