Описание проекта

Что здесь написать

– описание структуры проекта

– описание функционала api / streamlit-приложения

– инструкция по использованию, для пользователя в минимальном варианте (на следующих чекпоинтах надо будет подробнее написать)

TS

Описание API

- Статус данных

GET /api/data-status

Описание: метод проверяет наличие локальных данных

Тело ответа:

- **status (bool)**
- **message (str)**

Пример ответа:

```
{
  "status": true,
  "message": "success"
}
```

- Список очередей

GET /api/queues

Описание: метод возвращает данные по всем доступным очередям

Тело ответа:

- **id (int):** id очереди

- **name (str):** имя очереди
- **load (int):** количество тикетов в очереди
- **level (int):** уровень очереди

Пример ответа:


```
{
  "queues_by_level": {
    "1": [
      {
        "id": 10,
        "name": "Queue 10",
        "load": 2463146,
        "level": 1
      }
```


- Нагрузка в определенной очереди за выбранный период

GET /api/historical

Описание: метод возвращает исторические данные по определенной очереди за выбранный период

Параметры:

- **queue_id* (int):** номер очереди
- **granularity (str):** гранулярность временного ряда
- **start_date (str):** начальная дата временного периода
- **end_date (str):** конечная дата временного периода

Пример ответа объекта TimeSeriesData:


```
{
  "queue_id": 10,
  "values": [
    195,
    2067,
```

```
    5805
  ],
  "timestamps": [
    "2017-01-01",
    "2017-01-08",
    "2017-01-15"
  ],
  "granularity": "daily"
}
```


- Средние значения нагрузки по дням недели в определенной очереди за выбранный период

GET /api/daily_average
Описание: метод возвращает средние значения по дням недели в определенной очереди за выбранный период
Параметры:

- **queue_id* (int):** номер очереди
- **start_date* (str):** начальная дата временного периода
- **end_date* (str):** конечная дата временного периода

Пример ответа объекта AverageLoadWeekdays:


```
{
  "queue_id": 10,
  "weekdays": [
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday",
    "Sunday"
```

```
 ],
 "average_load": [
  1323.3333333333333,
  1223.6666666666667,
  1254.1666666666667,
  1247.4,
  1178.2,
  685.6666666666666,
  653.3333333333334
 ]
}
```


- Средние значения нагрузки по дням недели в определенной очереди за выбранный период

GET /api/weekly_average/{queue_id}
Описание: метод возвращает значения нагрузки по неделям в определенной очереди за выбранный период
Параметры:

- **queue_id* (int):** номер очереди
- **start_date (str):** начальная дата временного периода
- **end_date (str):** конечная дата временного периода

Пример ответа объекта AverageLoadWeekly:


```
{
 "queue_id": 10,
 "week": [
  "week 1",
  "week 2",
  "week 3"
 ],
```

```
 "average_load": [
  622.5,
  1082.857142857143,
  1075.7142857142858
 ]
}
```


- Статистика по выбранной очереди за определенный период

GET /api/queue_stats
Описание: метод возвращает статистику по выбранной очереди за определенный период
Параметры:

- **queue_id* (int):** номер очереди
- **start_date* (str):** начальная дата временного периода
- **end_date* (str):** конечная дата временного периода
- **granularity (str):** гранулярность временного ряда

Пример ответа объекта QueueStats:


```
{
 "queue_id": 10,
 "start_date": "2017-09-09T00:00:00",
 "end_date": "2017-10-10T00:00:00",
 "granularity": "daily",
 "structure": {
  "level": 1,
  "direct_children": 10,
  "all_descendants": 38,
  "leaf_nodes": 28,
  "depth": 3,
  "parent_id": null
 },
```

```
 "load": {
  "total_tickets": 33537,
  "avg_tickets": 1048.03125,
  "peak_load": 1402,
  "min_load": 401,
  "median_load": 1177,
  "std_dev": 288.68092491761615,
  "percentile_90": 1308.9,
  "percentile_95": 1346.5
 },
 "time": {
  "busiest_day": "2017-09-18T00:00:00",
  "quietest_day": "2017-10-08T00:00:00",
  "weekend_avg": 651.2,
  "weekday_avg": 1228.409090909091
 }
}
```


- Получить пример данных для загрузки

GET /api/sample_data
Описание: метод возвращает пример данных для загрузки в сервис
Параметры:

- **df_type* (string):** тип датасета

Пример ответа объекта DataFrameResponse:


```
{
 "columns": [
  "queueId",
  "date",
```

```
  "new_tickets"
 ],
 "data": [
  {
   "queueId": 1,
   "date": "2017-01-01T00:00:00",
   "new_tickets": 15
  },
  ],
 "shape": [
  418327,
  1
 ],
 "df_type": "tickets"
}
```


- Загрузить локальные данные

POST /api/reload_local_data
Описание: метод перезагружает данные
Тело ответа:

- **status (bool)**
- **message (str)**

Пример ответа объекта BaseResponse:


```
{
 "status": "data reloaded",
 "message": "success"
}
```


- Предсказание нагрузки

POST /api/forecast
Описание: метод выполняет прогнозирования ряда
Тело запроса:

- **queue_id* (int):** номер очереди
- **forecast_horizon (int):** сколько временных отрезков нужно спрогнозировать
- **granularity (str):** гранулярность временного ряда
- **include_confidence_intervals (bool)**

Пример ответа объекта ForecastResult:


```
{
  "queue_id": 10,
  "values": [
    2849.0922719402993,
    2553.1075123747573
  ],
  "timestamps": [
    "2020-10-01",
    "2020-10-02"
  ],
  "granularity": "daily"
}
```


- Загрузка данных

POST /api/upload_data
Описание: метод позволяет загрузить данные
Параметры:

- **file* (string):** CSV файл
- **df_type* (string):** тип датасета

Пример ответа объекта BaseResponse:


```
{
```

"status": "success",

"message": "Data updated successfully"

}

Описание Streamlit - приложения (TS)

Приложение предназначено для анализа и прогнозирования нагрузки очередей в рамках системы технической поддержки. Оно предоставляет возможность визуализации исторических данных, анализа временных паттернов и прогнозирования будущей нагрузки.

Основные функции:

1. **Загрузка и управление данными**:
    - Использование предустановленных данных или загрузка пользовательских наборов (CSV-файлов) для анализа нагрузки и структуры очередей.
    - Функция восстановления данных по умолчанию.
2. **Анализ очередей**:
    - Выбор очередей через боковую панель с учетом иерархической структуры.
    - Визуализация структуры выбранной очереди (уровень, наследники, глубина).
3. **Анализ временных данных**:
    - Отображение временных рядов с трендами и средними значениями.
    - Средняя нагрузка по дням недели.
    - Распределение нагрузки по неделям.
4. **Прогнозирование нагрузки**:
    - Генерация прогноза на заданный временной горизонт (до 300 дней).
    - Визуализация прогноза с наложением исторических данных.

Технические особенности:

- **Интеграция с API**:
    - Запросы на получение данных о статистике очередей, исторической нагрузке и прогнозах.

    - Управление данными через API (загрузка, проверка доступности, перезагрузка).
- **Кэширование данных**:
    - Использование @st.cache_data для уменьшения числа запросов к серверу.
- **Интерактивные графики**:
    - Построение графиков с помощью Plotly (временные ряды, распределение по дням недели и неделям, структура очередей).
- **Обработка ошибок**:
    - Обработка исключений при взаимодействии с API и отображение сообщений пользователям.

Пользовательский интерфейс:

- Главная страница с описанием приложения и функциональности.
- Боковая панель для выбора очередей, временных интервалов и параметров прогноза.
- Секция загрузки данных с примерами форматов файлов.
- Визуализация данных в виде интерактивных графиков и метрик.

NLP

На 30.12.2024 реализованы API-методы:

- Обучение NLP-модели

`POST /api/fit_nlp`

Описание:
Метод запускает обучение NLP-модели на основе предоставленных параметров.

Тело запроса:

- **model** (строка): Тип модели, которая будет обучена (Logistic | SVM)
- **config** (объект): Конфигурация обучения, включая гиперпараметры и настройки.

Пример тела запроса (JSON):

```
{
```

```
  "model": "logistic",
  "config": {
    "C": 10,
    "solver": “lbfgs”
  }
}
```

Возвращает идентификатор обученной модели (`model_id`) в виде строки.

Пример ответа: `"model_12345"`

- Получение статистики модели

`GET /api/statistics/{id}`

Описание:
Метод возвращает отчёт о классификации для модели с указанным идентификатором.

Параметры пути:

`id (строка):` `Уникальный идентификатор модели.`

Возвращает объект ClassificationReport с метриками классификации.
Пример структуры ответа:

```
{
  "roc_auc": 0.95,
  "accuracy": 0.90,
  "macro_avg": {
    "precision": 0.88,
    "recall": 0.89,
    "f1_score": 0.88
  },
  "weighted_avg": {
    "precision": 0.91,
    "recall": 0.90,
```

```
    "f1_score": 0.91
  },
  "classes": {
    "class_1": {
      "precision": 0.85,
      "recall": 0.87,
      "f1_score": 0.86
    },
    "class_2": {
      "precision": 0.92,
      "recall": 0.93,
      "f1_score": 0.92
    }
  }
}
```


- Удаление NLP-модели

DELETE /api/remove_nlp/{id}

Описание:
Метод удаляет модель с указанным идентификатором.

Параметры пути:

- id (строка): Уникальный идентификатор модели.

Ответ:

- **200 OK**: Успешное удаление. Возвращает объект SuccessResponse.
  Пример ответа:


```
{
  "status": "success",
  "message": "model model_12345 was successfully removed"
```

```
}
```

- **404 Not Found**: Если модель не найдена. Возвращает объект ErrorResponse.
  Пример ответа:

```
{
  "status": "error",
  "message": "Model not found"
}
```

Методы в разработке

- Предсказание кластера по тексту

POST /api/predict_nlp

Описание:
Метод принимает описание тикета и возвращает номер кластера.

Пример тела запроса (JSON):

```
{
  "id": "model_id",
  "text: “string”
}
```

Возвращает идентификатор кластера в виде числа.

Пример ответа: "7"

- Предсказание кластеров для файла csv с описанием кластеров

POST /api/predict_nlp_csv

Описание:
Метод принимает csv файл и возвращает csv файл с предсказания модели для каждого примера.

Пример тела запроса (JSON):

```
{
  "id": "model_id",
  "file: “example.csv”
}
```

Возвращает идентификатор кластера в виде числа.

Пример ответа: `"attachment; filename=data.csv"`

Описание Streamlit - приложения (NLP)

1. **Структура приложения**:
    - Содержит три вкладки:

      Project info: информация о проекте и используемом датасете.

      Train: выбор, настройка, обучение и удаление моделей NLP.

      Prediction: визуализация предсказаний модели.

2. **Основные функции**:
    - **Обзор проекта**:

      Подробное описание целей и задач NLP-компонента проекта.

      Информация о датасете, включающем ~10,000 тикетов пользователей.

      Список участников команды.

    - **Обучение моделей**:

      Поддержка двух моделей: логистической регрессии и SVM.

      Настройка параметров моделей через боковую панель.

      Отправка конфигурации на сервер для обучения.

      Вывод ROC AUC и визуализация метрик (Precision, Recall, F1-Score) с использованием Plotly.

    - **Удаление моделей**:

Удаление обученных моделей через API с удалением связанных данных.

- **Визуализация**:

  Линейный график предсказаний модели.

  Визуализация метрик в формате интерактивного графика.

3. **API интеграция**:
   - Для обучения, анализа статистики и удаления моделей используются эндпоинты сервера, зависящие от среды исполнения (Docker/локальная).
4. **Обработка ошибок**:
   - Встроенная обработка ошибок для некорректного ввода параметров и отсутствия соединения с сервером.

Приложение предоставляет интуитивный интерфейс для управления процессом создания и анализа NLP-моделей.

Структура проекта

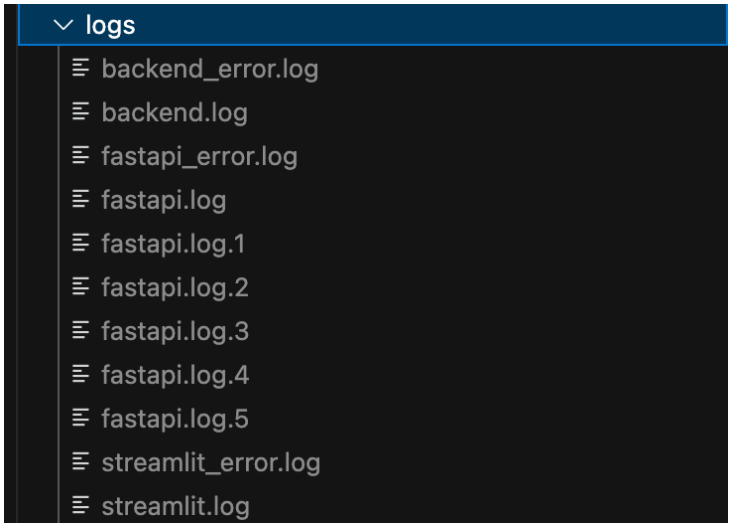
- data/ - данные проекта, подгружаются отдельной командой
- notebooks/ - ноутбуки с исследованием данных, качества моделей (TS + NLP)
- scripts/ - полезные скрипты
- tasks_support_system_ai/ - пакет со всем функционалом, который используется для работы сервиса. Мы его устанавливаем, чтобы не мучаться с импортами. Код между TS и NLP поделен почти во всех подпапках.
- tests/ - тесты пакета
- nginx/ - конфиг для nginx для miniO и самого сервиса
- Dockerfile, compose.yaml для запуска сервиса в контейнерах
- .github/ с конфигами для GitHub Actions.

Далее структура может быть уже неактуальной, так как код развивается.

- - `tasks_support_system_ai/main.py` - приложение fastapi
- - `tasks_support_system_ai/web/app.py` - приложение streamlit
- - `tasks_support_system_ai/services/` - бизнес логика приложения
- - `tasks_support_system_ai/api/` - эндпоинты и модели для приложения fastapi
- - `tasks_support_system_ai/data/` - всякое вспомогательное
- - `tasks_support_system_ai/utils/` - всякое вспомогательное

Инфраструктура

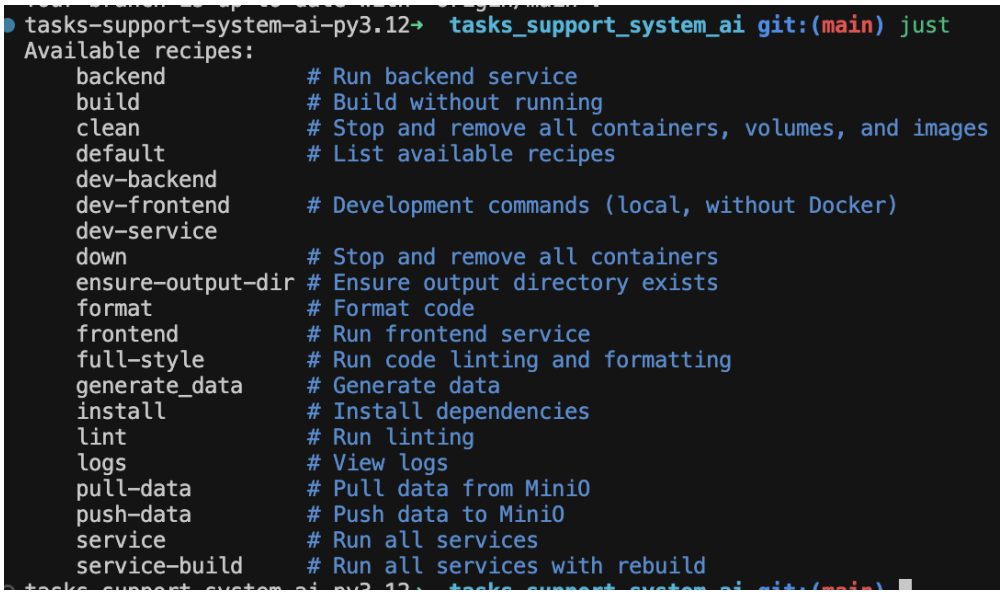
- Для контроля версий зависимостей используется poetry
- Настроено логгирование с ротацией по достижении размера 10 MB, логи имеют формат JSON для более легкой автоматической аналитики. FastAPI логгируется через добавление Middleware Starlette, таким образом мы видим все запросы и ответы.



- Для проверки качества кода в GitHub Actions добавлены линтер и форматтер ruff, линтер Dockerfile (hadolint)
- В GA также проверяются тесты pytest (но тестов почти нет)
- В ветку main можно коммитить только через PR, история линейная.
- В GA добавлен скрипт для деплоя приложения на удаленный сервер. Скрипт копирует код репозитория, удаляет текущие контейнеры, поднимает новые контейнеры, скачивает данные с MinIO (как бы сам с себя, но скачивает честно через интернет), поднимает приложение полностью.

- Настроен обратный прокси nginx для докер сервисов MinIO, фронтенд сервиса и бэкенд сервиса. Сертификаты на субдомены выданы CA Let's Encrypt, на VPS настроено автопродление сертификатов. Конфиги nginx находится в репозитории.

- Так как все сервисы поднимаются через docker compose, то мы не выставляем наружу порты бэкенда или фронтенда, и nginx использует общую докер сеть для доступа к backend, frontend контейнерам. Наружу выставлены порты 80, 443 nginx, которые он перенаправляет на нужный сервис, 80 порт перенаправляется на 443, чтобы использовалось безопасное соединение.
- https://data.pyrogn.ru/ - это инстанс MinIO (в браузере добавляется ui/, через клиент надо использовать обычный url)
- https://support.pyrogn.ru/ сервис с последнего коммита в main ветке
- https://support.pyrogn.ru/api/ бэкенд fastapi
- Для быстрого запуска частых команд используется just (похож на make, но удобней для нашего случая). Все команды можно посмотреть, запустив just.



- Авторизации на публичный сервис пока нет.

Для запуска и деплоя сервиса на удаленном сервере

1. Установить docker
2. Установить certbot-nginx, выдать сертификаты на домены
3. Создать юзера для GA, дать root доступ и добавить его в группу docker (команды useradd, usermod).
4. Сгенерировать пару ключей для GA, приватный оставить как secrets в гитхабе, публичный оставить в .ssh/authorized_keys

5. Изменить DNS настройки для всего докера, чтобы достучаться до всех адресов с данного сервера

```
pavel@3934693-zi92271:~$ cat /etc/docker/daemon.json
{ "dns": ["8.8.8.8", "8.8.4.4"] }
```

6. IP адрес оставить в секретах гитхаб, чтобы знать, куда копировать репозиторий через scp.
7. Далее бонусы: закрыть доступ по логину, паролю (sudo vim /etc/ssh/sshd_config).
8. Установить fail2ban.

```
pavel@3934693-zi92271:~$ sudo fail2ban-client status sshd
[sudo] password for pavel:
Status for the jail: sshd
|- Filter
|  |- Currently failed: 3
|  |- Total failed:     8261
|  `- Journal matches:  _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
   |- Currently banned: 2
   |- Total banned:     1313
   `- Banned IP list:   92.255.85.253 111.173.89.134
```

9. Настроить ufw, открыть только необходимые порты

```
pavel@3934693-zi92271:~$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), deny (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22/tcp (OpenSSH)           ALLOW IN    Anywhere
8501                       ALLOW IN    Anywhere
80,443/tcp (Nginx Full)    ALLOW IN    Anywhere
8501/tcp                   ALLOW IN    Anywhere
9000                       ALLOW IN    Anywhere
9001                       ALLOW IN    Anywhere
10050/tcp                  ALLOW IN    Anywhere
22/tcp (OpenSSH (v6))      ALLOW IN    Anywhere (v6)
8501 (v6)                  ALLOW IN    Anywhere (v6)
80,443/tcp (Nginx Full (v6)) ALLOW IN    Anywhere (v6)
8501/tcp (v6)              ALLOW IN    Anywhere (v6)
9000 (v6)                  ALLOW IN    Anywhere (v6)
9001 (v6)                  ALLOW IN    Anywhere (v6)
10050/tcp (v6)             ALLOW IN    Anywhere (v6)
```

Пользовательская инструкция

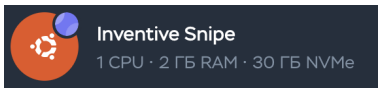
Предпосылки:

- Рекомендуется запускать, используя Docker, так как тогда больше гарантий, что проект запустится, как мы ожидаем.
- Данные синхронизируются между командой посредством загрузки и скачивания данных в бакет MinIO. Инстанс MinIO запускается на VPS из конфигурации docker compose из нашего репозитория. Необходимо создать .env файл с логином, паролем для инстанса.
- Для скачивания данных надо либо установить проект через poetry install, или запустить сервис backend через docker compose.
  Команда для скачивания данных: poetry run minio-sync pull или docker compose exec backend poetry run minio-sync pull

Для запуска достаточно выполнить docker compose up backend frontend, и открыть http://localhost:8501/, где будет доступ к streamlit сервиса

Узкие места в инфраструктуре

- Публичный сервис без авторизации
- Маломощный сервер (1 ядро, а значит обучение в отдельном процессе будет только медленней)



- Если подгрузить свои данные, то они подменят данные в бэкенде, а значит это повлияет на всех пользователей. Read only операции будут работать независимо, но write операции надо бы либо разграничивать между пользователями. Сейчас надежда на то, что таких конфликтов просто не будет. Легкого пути здесь нет.
- Хрупкий деплой, для дебага желательно иметь доступ серверу.