Документация к действиям системы

Зачем нужна документация для действий?

Чтобы ИИ-агент мог эффективно использовать имплементированные действия, необходима четкая документация, содержащая информацию о типах данных, параметрах и возвращаемых значениях этих действий. Это позволяет агенту правильно интерпретировать и корректно применять действия в различных сценариях.

Из чего состоит документация?

Документация состоит из трех ключевых разделов: подсказки типов (TypeHints Definition), модели данных на основе Pydantic (Models Definition) и описание действий (API Calls Documentation). Эти компоненты помогают обеспечить целостное понимание работы действий.

Важно придерживаться следующего формата:

```
# TypeHints Definition
<здесь идут подсказки типов>

# Models Definition
<здесь идут модели данных>

# API Calls Documentation
<здесь идут описания действий>
```

Документация должна находиться в папке с системой (например **Todoist**) и иметь название `documentation.md`.

Подсказки типов (Type Hints)

Этот раздел определяет типы данных, используемые в действиях. Четкое описание типов позволяет ИИ-агенту точно понимать, какие значения

ожидаются в качестве аргументов и что возвращает действие.

Пример (работаем в `team_actions/src/actions/Todoist/documentation.md`):

```
Id = Annotated[str, Field(pattern="^[0-9]+$")]
TaskName = Annotated[str, Field(description="Task name")]
Datetime = Annotated[str, Field(pattern=r"^\d{4}-\d{2}-\d{2}T
\d{2}:\d{2}:\d{2}(\.\d+)?Z$")]
ShortDatetime = Annotated[str, Field(pattern=r"^\d{4}-\d{2}-
\d{2}$")]
DueString = Annotated[str, Field(description="Due date in Eng
lish", example="tomorrow")]
DueLang = Annotated[str, Field(pattern="^[a-z]{2}$", default
="en")]
DurationUnit = Literal['minute', 'day']
Priority = Annotated[int, Field(ge=1, le=4)]
```

Здесь каждый тип данных сопровождается описанием, что улучшает читаемость и понимание структуры данных.

Pydantic модели данных

В этом разделе описываются структуры данных, которые используются или возвращаются действиями. Мы используем Pydantic, так как эти модели широко известны и часто встречаются в тренировочных данных больших языковых моделей (LLM). Это позволяет ИИ-агенту лучше понимать структуру данных.

Пример (работаем в `team_actions/src/actions/Todoist/documentation.md`):

```
class Due(BaseModel):
    string: Optional[DueString]
    date: Optional[ShortDatetime]
    is_recurring: bool
    datetime: Optional[Datetime]
    timezone: Optional[str]

class Duration(BaseModel):
```

```
    amount: Optional[int]
    unit: Optional[DurationUnit]

class Task(BaseModel):
    id: Id
    assigner_id: Optional[Id]
    assignee_id: Optional[Id]
    project_id: ProjectId
    section_id: Optional[Id]
    parent_id: Optional[Id]
    order: int
    content: TaskName
    description: Optional[str]
    is_completed: bool
    labels: Optional[List[str]]
    priority: Priority
    comment_count: int
    creator_id: Id
    created_at: Datetime
    due: Optional[Due]
    url: HttpUrl
    duration: Optional[Duration]
```

Эти модели помогают ИИ-агенту точно интерпретировать данные, передаваемые и возвращаемые системой.

Описание действий (API Calls Documentation)

Это самый важный раздел, так как здесь детализируются действия: что именно делает каждое действие, какие параметры оно принимает и что возвращает.

Пример (работаем в `team_actions/src/actions/Todoist/documentation.md`):

```
## `create_task`

**Description**:
```

```
Creates a new task in the system with optional parameters lik
e descriptions, project and section IDs, and duration.

**Parameters**:
- content (TaskName): The name of the task.
- description (Optional[str]): A brief description of the tas
k.
- project_id (Optional[Id]): The task project ID. If not set,
task is put to user's Inbox.
- section_id (Optional[Id]): ID of section to put task into.
- labels (Optional[List[str]]): A list of names of objects or
events that might be associated with the task.
- priority (Optional[Priority]): Task priority from 1 (norma
l) to 4 (urgent).
- due_string (Optional[DueString]): Human-defined task due da
te (e.g., 'next Monday', 'Tomorrow').
- due_lang (Optional[DueLang]): 2-letter language code.
- due_date (Optional[ShortDatetime]): Specific date in YYYY-M
M-DD format relative to user’s timezone.
- due_datetime (Optional[Datetime]): Specific date and time i
n RFC3339 format, e.g. `2023-10-05T14:48:00-05:00`.
- duration (Optional[Duration]): A dictionary representing a
task duration.
- duration_unit (Optional[DurationUnit]): The unit of time th
at the duration field above represents.

**Returns**:
- created_task (Task)
```

Четкое описание входных параметров и возвращаемых значений упрощает взаимодействие с API и позволяет агенту работать с действиями без ошибок.

Пример написания документации

Теперь рассмотрим, как правильно документировать действия для двух случаев: добавление документации к существующей платформе и создание

документации для новой платформы.

Случай 1: добавление документации для существующих систем

Когда добавляется новое действие для уже интегрированной системы (например, TeamFlame, Todoist или GitFlame), достаточно дополнить существующую документацию новыми API действиями. Вам будут предоставлены первые два раздела — **Подсказки типов** и **Pydantic модели данных**. Вы можете добавлять новые подсказки типов и модели данных, если это необходимо. В части с описанием действий просто добавьте описание своих действий. После интеграции система объединит их с уже существующими описаниями.

Рассмотрим **еще раз** пример выше с системой **Todoist** (работаем в `team_actions/src/actions/Todoist/documentation.md`):

```
# TypeHints Definition
Id = Annotated[str, Field(pattern="^[0-9]+$")]
TaskName = Annotated[str, Field(description="Task name")]
Datetime = Annotated[str, Field(pattern=r"^\d{4}-\d{2}-\d{2}T
\d{2}:\d{2}:\d{2}(\.\d+)?Z$")]
ShortDatetime = Annotated[str, Field(pattern=r"^\d{4}-\d{2}-
\d{2}$")]
DueString = Annotated[str, Field(description="Due date in Eng
lish", example="tomorrow")]
DueLang = Annotated[str, Field(pattern="^[a-z]{2}$", default
="en")]
DurationUnit = Literal['minute', 'day']
Priority = Annotated[int, Field(ge=1, le=4)]

# Models Definition
class Due(BaseModel):
    string: Optional[DueString]
    date: Optional[ShortDatetime]
    is_recurring: bool
    datetime: Optional[Datetime]
```

```
    timezone: Optional[str]

class Duration(BaseModel):
    amount: Optional[int]
    unit: Optional[DurationUnit]

class Task(BaseModel):
    id: Id
    assigner_id: Optional[Id]
    assignee_id: Optional[Id]
    project_id: ProjectId
    section_id: Optional[Id]
    parent_id: Optional[Id]
    order: int
    content: TaskName
    description: Optional[str]
    is_completed: bool
    labels: Optional[List[str]]
    priority: Priority
    comment_count: int
    creator_id: Id
    created_at: Datetime
    due: Optional[Due]
    url: HttpUrl
    duration: Optional[Duration]


# API Calls Documentation
## `create_task`

**Description**:
Creates a new task in the system with optional parameters lik
e descriptions, project and section IDs, and duration.

**Parameters**:
- content (TaskName): The name of the task.
```

```
- description (Optional[str]): A brief description of the tas
k.
- project_id (Optional[Id]): The task project ID. If not set,
task is put to user's Inbox.
- section_id (Optional[Id]): ID of section to put task into.
- labels (Optional[List[str]]): A list of names of objects or
events that might be associated with the task.
- priority (Optional[Priority]): Task priority from 1 (norma
l) to 4 (urgent).
- due_string (Optional[DueString]): Human-defined task due da
te (e.g., 'next Monday', 'Tomorrow').
- due_lang (Optional[DueLang]): 2-letter language code.
- due_date (Optional[ShortDatetime]): Specific date in YYYY-M
M-DD format relative to user’s timezone.
- due_datetime (Optional[Datetime]): Specific date and time i
n RFC3339 format, e.g. `2023-10-05T14:48:00-05:00`.
- duration (Optional[Duration]): A dictionary representing a
task duration.
- duration_unit (Optional[DurationUnit]): The unit of time th
at the duration field above represents.

**Returns**:
- created_task (Task)
```

Случай 2: добавление документации для новых систем

Если интегрируется новая платформа (например, **YandexEda**), всю документацию нужно создавать с нуля, включая подсказки типов, модели данных и описание действий.

Пример для новой платформы (работаем в `team_actions/src/actions/YandexEda/documentation.md`, отсутствует в репозитории AgniaChallenge):

```
# TypeHints Definition
MealName = Annotated[str, Field(description="Well-crafted mea
```

```
l name")]
Address = Annotated[str, Field(description="Delivery addres
s")]


# Models Definition
class DeliveryResult(BaseModel):
    meal: MealName
    address: Address
    status: Literal["ordered", "delivered", "canceled"]


# API Calls Documentation
## `order_delivery`

**Description**:
Orders a meal at the specified address

**Parameters**:
- meal (MealName)
- address (Address)

**Returns**:
- result (DeliveryResult)
```