Полное руководство по созданию роутеров в FastAPI с подробными комментариями

Результат в конце

1. Базовая структура роутера

```
# Импорт необходимых модулей и классов
from fastapi import APIRouter, HTTPException, Depends  # APIRouter - для создания роутеров, HTTPException - для обработки ошибок, Depends - для зависимостей
from anylogiccloudclient.client.cloud_client import CloudClient  # Клиент для работы с AnyLogic Cloud
import logging  # Модуль для логирования (записи информации о работе приложения)

# Импорт наших собственных модулей
from app.models import SimulationRequest, SimulationResponse, ErrorResponse  # Pydantic модели для данных
from app.dependencies import get_cloud_client  # Функция-зависимость для получения клиента AnyLogic

# Настройка логирования - запись информации о работе приложения
logging.basicConfig(level=logging.INFO)  # Устанавливаем уровень логирования на INFO (будут видны информационные сообщения и ошибки)
logger = logging.getLogger(__name__)  # Создаем объект логгера с именем текущего модуля

# Создаем роутер - объект, который будет содержать наши эндпоинты (маршруты)
# APIRouter похож на мини-приложение FastAPI
router = APIRouter()
```

2. Детальное объяснение POST эндпоинта

```
# Декоратор @router.post - указывает, что это POST запрос
# "/simulations/run" - URL путь эндпоинта
# response_model=SimulationResponse - указывает FastAPI, какую модель использовать для ответа
# responses={500: {"model": ErrorResponse}} - документация для возможных ошибок
@router.post(
    "/simulations/run",  # URL путь: http://ваш-сервер/api/v1/simulations/run
    response_model=SimulationResponse,  # Модель для успешного ответа
    responses={500: {"model": ErrorResponse}}  # Документирование ошибки 500
)
```

```
# async def - объявление асинхронной функции
# request: SimulationRequest - параметр запроса, автоматически валидируется по модели SimulationRequest
# client: CloudClient = Depends(get_cloud_client) - зависимость, которая предоставит клиент AnyLogic
async def run_simulation(
    request: SimulationRequest,  # Данные, которые приходят в теле POST запроса
    client: CloudClient = Depends(get_cloud_client)  # Зависимость: FastAPI вызовет get_cloud_client() и передаст результат
):
    """
    Запуск симуляции демо-модели Service System Demo

    Args:
        request (SimulationRequest): Параметры для запуска симуляции
        client (CloudClient): Клиент для работы с AnyLogic Cloud

    Returns:
        SimulationResponse: Результаты симуляции

    Raises:
        HTTPException: Если произошла ошибка при выполнении симуляции
    """
    try:
        # Логируем начало операции - записываем информационное сообщение
        logger.info(f"Запуск симуляции с параметрами: {request.dict()}")

        # ========== ШАГ 1: Получение информации о модели ==========
        # client.get_latest_model_version() - метод клиента для получения последней версии модели
        # request.model_name - обращение к полю model_name объекта request
        version = client.get_latest_model_version(request.model_name)
        # Логируем успешное получение версии модели
        logger.info(f"Найдена версия модели: {version.id}")

        # ========== ШАГ 2: Создание входных параметров ==========
        # client.create_inputs_from_experiment() - создает параметры на основе эксперимента
        # version - объект версии модели, полученный на предыдущем шаге
        # request.experiment_name - имя эксперимента из запроса
        inputs = client.create_inputs_from_experiment(version, request.experiment_name)

        # ========== ШАГ 3: Установка пользовательских параметров ==========
        # inputs.set_input() - устанавливает значение для конкретного параметра модели
        # "Server capacity" - имя параметра в AnyLogic модели
        # request.server_capacity - значение из запроса пользователя
        inputs.set_input("Server capacity", request.server_capacity)
```

```
        # ========== ШАГ 4: Создание и запуск симуляции ==========
        # client.create_simulation() - создает новую симуляцию с заданными параметрами
        simulation = client.create_simulation(inputs)
        # Логируем создание симуляции
        logger.info(f"Создана симуляция с ID: {simulation.id}")

        # ========== ШАГ 5: Получение результатов ==========
        # simulation.get_outputs_and_run_if_absent() - запускает симуляцию (если еще не
запущена) и получает результаты
        outputs = simulation.get_outputs_and_run_if_absent()
        logger.info("Симуляция завершена, получены результаты")

        # ========== ШАГ 6: Извлечение конкретных данных из результатов ==========
        # outputs.value() - получает значение конкретного показателя из результатов
        # "Mean queue size|Mean queue size" - путь к данным в результатах AnyLogic
        mean_queue_size = outputs.value("Mean queue size|Mean queue size")
        server_utilization = outputs.value("Utilization|Server utilization")

        # ========== ШАГ 7: Возврат ответа клиенту ==========
        # Создаем объект SimulationResponse с результатами
        # FastAPI автоматически преобразует его в JSON
        return SimulationResponse(
            simulation_id=simulation.id,  # ID симуляции
            server_capacity=request.server_capacity,  # Входной параметр
            mean_queue_size=mean_queue_size,  # Рассчитанный средний размер очереди
            server_utilization=server_utilization,  # Рассчитанная загрузка серверов
            raw_outputs=outputs.get_raw_outputs(),  # Все сырые данные результатов
            status="completed"  # Статус выполнения
        )

    except Exception as e:
        # Обработка любых ошибок, которые могут возникнуть в блоке try
        # Логируем ошибку с уровнем ERROR
        logger.error(f"Ошибка при выполнении симуляции: {str(e)}")
        # Выбрасываем HTTP исключение с кодом 500 (Internal Server Error)
        raise HTTPException(
            status_code=500,  # HTTP статус код
            detail=f"Ошибка моделирования: {str(e)}"  # Сообщение об ошибке для клиента
        )
```

3. Детальное объяснение GET эндпоинта

```
# Декоратор для GET запроса
# "/models" - URL путь
@router.get("/models")  # URL: http://ваш-сервер/api/v1/models
async def get_models(
    client: CloudClient = Depends(get_cloud_client)  # Та же зависимость для клиента
AnyLogic
```

```
):
    """
    Получение списка доступных моделей в AnyLogic Cloud

    Args:
        client (CloudClient): Клиент для работы с AnyLogic Cloud

    Returns:
        dict: Словарь с ключом "models" и списком моделей

    Raises:
        HTTPException: Если произошла ошибка при получении списка моделей
    """
    try:
        # ========== ШАГ 1: Получение списка моделей ==========
        # client.get_models() - метод клиента для получения всех доступных моделей
        models = client.get_models()

        # ========== ШАГ 2: Преобразование данных в удобный формат ==========
        # Создаем пустой список для результатов
        models_list = []

        # Проходим по всем моделям в цикле
        for model in models:
            # Для каждой модели создаем словарь с нужными полями
            models_list.append({
                "id": model.id,  # Уникальный идентификатор модели
                "name": model.name,  # Название модели
                # Условное выражение: если есть latest_version, берем ее id, иначе None
                "latest_version_id": model.latest_version.id if model.latest_version else None
            })

        # ========== ШАГ 3: Возврат результата ==========
        # Возвращаем словарь с ключом "models" и списком моделей
        # FastAPI автоматически преобразует в JSON
        return {"models": models_list}

    except Exception as e:
        # Обработка ошибок
        raise HTTPException(
            status_code=500,  # HTTP статус 500 - внутренняя ошибка сервера
            detail=f"Ошибка получения списка моделей: {str(e)}"  # Сообщение об ошибке
        )
```

4. Полный пример с дополнительными возможностями

```
from fastapi import APIRouter, HTTPException, Depends, Query, Path
from typing import List, Optional
from anylogiccloudclient.client.cloud_client import CloudClient
import logging
from datetime import datetime

from app.models import SimulationRequest, SimulationResponse, ErrorResponse,
ModelInfo
from app.dependencies import get_cloud_client

# Настройка логирования
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

# Создание роутера с префиксом и тегами для документации
router = APIRouter(
    prefix="/simulations",  # Автоматический префикс для всех путей в этом роутере
    tags=["simulations"],   # Группировка в документации Swagger
    responses={404: {"description": "Not found"}}  # Общие ответы для всех эндпоинтов
)

@router.post(
    "/run",
    response_model=SimulationResponse,
    summary="Запуск симуляции",  # Краткое описание для документации
    description="""
Запускает симуляцию демо-модели Service System Demo с заданными параметрами.

Параметры:
- server_capacity: Количество серверов в системе
- model_name: Название модели для запуска
- experiment_name: Название эксперимента
""",
    response_description="Результаты выполненной симуляции",
    status_code=200  # Статус код при успешном выполнении
)
async def run_simulation(
    request: SimulationRequest,
    client: CloudClient = Depends(get_cloud_client)
) -> SimulationResponse:  # Указание типа возвращаемого значения (опционально)
    """
    Основной эндпоинт для запуска симуляций в AnyLogic Cloud.

    Процесс:
    1. Получение информации о модели
```

```
    2. Настройка параметров симуляции
    3. Запуск симуляции
    4. Получение и обработка результатов
    5. Возврат результатов клиенту
    """
    # Логируем начало операции с дополнительной информацией
    logger.info(f"Запуск симуляции для модели '{request.model_name}' с
{request.server_capacity} серверами")

    try:
        # Валидация входных данных (дополнительная проверка)
        if request.server_capacity <= 0:
            raise HTTPException(
                status_code=400,  # 400 - Bad Request
                detail="Количество серверов должно быть положительным числом"
            )

        # Основная логика (как в предыдущем примере)
        version = client.get_latest_model_version(request.model_name)
        inputs = client.create_inputs_from_experiment(version, request.experiment_name)
        inputs.set_input("Server capacity", request.server_capacity)
        simulation = client.create_simulation(inputs)
        outputs = simulation.get_outputs_and_run_if_absent()

        # Извлечение результатов
        mean_queue_size = outputs.value("Mean queue size|Mean queue size")
        server_utilization = outputs.value("Utilization|Server utilization")

        # Логируем успешное завершение
        logger.info(f"Симуляция {simulation.id} завершена успешно")

        return SimulationResponse(
            simulation_id=simulation.id,
            server_capacity=request.server_capacity,
            mean_queue_size=mean_queue_size,
            server_utilization=server_utilization,
            raw_outputs=outputs.get_raw_outputs(),
            status="completed",
            timestamp=datetime.now()  # Добавляем временную метку
        )

    except HTTPException:
        # Перевыбрасываем HTTP исключения без изменений
        raise
    except Exception as e:
        # Логируем полную информацию об ошибке
        logger.error(f"Ошибка симуляции: {str(e)}", exc_info=True)
        raise HTTPException(
```

```
            status_code=500,
            detail=f"Внутренняя ошибка при выполнении симуляции: {str(e)}"
        )

@router.get(
    "/models",
    response_model=List[ModelInfo],  # Теперь возвращаем список моделей
    summary="Получить список моделей",
    description="Возвращает список всех доступных моделей в AnyLogic Cloud"
)
async def get_models(
    client: CloudClient = Depends(get_cloud_client),
    include_versions: bool = Query(False, description="Включать информацию о версиях")
# Query параметр
) -> List[ModelInfo]:
    """
    Получение списка моделей с возможностью фильтрации.

    Query параметры:
    - include_versions: Если True, включает подробную информацию о версиях
    """
    try:
        models = client.get_models()
        result = []

        for model in models:
            model_info = {
                "id": model.id,
                "name": model.name,
                "latest_version_id": model.latest_version.id if model.latest_version else None
            }

            # Добавляем дополнительную информацию если запрошено
            if include_versions and model.latest_version:
                model_info["version_name"] = model.latest_version.name
                model_info["version_number"] = model.latest_version.number

            result.append(model_info)

        logger.info(f"Возвращено {len(result)} моделей")
        return result

    except Exception as e:
        logger.error(f"Ошибка получения моделей: {str(e)}")
        raise HTTPException(
            status_code=500,
            detail="Не удалось получить список моделей"
        )
```

```
# Дополнительный эндпоинт для получения информации о конкретной модели
@router.get(
    "/models/{model_id}",  # Path параметр - часть URL
    response_model=ModelInfo,
    summary="Получить информацию о модели"
)
async def get_model_by_id(
    model_id: str = Path(..., description="ID модели"),  # Path параметр (обязательный)
    client: CloudClient = Depends(get_cloud_client)
) -> ModelInfo:
    """
    Получение подробной информации о конкретной модели по ID.

    Path параметры:
    - model_id: Уникальный идентификатор модели
    """
    try:
        models = client.get_models()

        # Ищем модель по ID
        for model in models:
            if model.id == model_id:
                return {
                    "id": model.id,
                    "name": model.name,
                    "latest_version_id": model.latest_version.id if model.latest_version else None
                }

        # Если модель не найдена
        raise HTTPException(
            status_code=404,
            detail=f"Модель с ID {model_id} не найдена"
        )

    except HTTPException:
        raise
    except Exception as e:
        logger.error(f"Ошибка поиска модели {model_id}: {str(e)}")
        raise HTTPException(
            status_code=500,
            detail="Ошибка при поиске модели"
        )
```

Результат:

simulations

POST `/api/v1/simulations/run` Запуск симуляции

Запускает симуляцию демо-модели Service System Demo с заданными параметрами.

Параметры:

- server_capacity: Количество серверов в системе
- model_name: Название модели для запуска
- experiment_name: Название эксперимента

Parameters | Try it out

No parameters

Request body required | application/json

Example Value | Schema

```
{
  "server_capacity": 8,
  "model_name": "Service System Demo",
  "experiment_name": "Baseline"
}
```

Responses

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8001/api/v1/simulations/models?include_versions=false' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8001/api/v1/simulations/models?include_versions=false
```

Server response

Code	Details
200	Response body


```
[
  {
    "id": "0e585c0f-3ab9-47d2-8a4f-f444d73bf429",
    "name": "File IO API Demo",
    "latest_version_id": null,
    "version_name": null,
    "version_number": null
  },
  {
    "id": "d87ab423-605c-4e93-a5b8-902ed5d42cb1",
    "name": "Transporters Moving in Free Space",
    "latest_version_id": null,
    "version_name": null,
    "version_number": null
  },
  {
    "id": "7d49a08e-2641-42a9-bf0a-11b2dffe1408",
    "name": "Bass Diffusion Demo 8.5.0",
    "latest_version_id": null,
    "version_name": null,
    "version_number": null
  },
  {
    "id": "1ba2f2f6-7c7f-4067-885a-441bb0bd5d03",
    "name": "Service System Demo",
    "latest_version_id": null,
    "version_name": null,
```


Download

Response headers

```
content-length: 728
content-type: application/json
date: Sat,25 Oct 2025 08:41:10 GMT
server: uvicorn
```