

NovaData LMS: API-контракты LMS/checkers

Область: lms-novadata , SQLChecker , code_checker

Документ описывает API-контракты между LMS и внешними сервисами проверки. Основные интеграции:

- LMS -> SQLChecker для SQL-заданий;
- code_checker worker -> LMS validator API для programming-заданий;
- LMS/code_checker -> code_checker API для шаблонов и ручного запуска кода.

Связанные документы:

- README_DOCUMENTATION.md - индекс комплекта документации и рекомендуемый порядок чтения.
- PROJECT_DOCUMENTATION.md - общая техническая документация проекта.
- DEPLOYMENT_GUIDE.md - инструкция по развертыванию.
- ENVIRONMENT_VARIABLES.md - схема переменных окружения.
- OPERATIONS_RUNBOOK.md - регламент эксплуатации и восстановления после сбоев.
- PROGRAMMING_TASK_AUTHOR_GUIDE.md - руководство по созданию programming-заданий.
- SQL_TASK_AUTHOR_GUIDE.md - руководство по созданию SQL-заданий.

1. Общие правила

1.1. Формат данных

Все API используют JSON.

Рекомендуемые headers:

Content-Type: application/json

Accept: application/json

Для validator API LMS внешний checker должен передавать:

X-API-Key: <validator-token>

1.2. Базовые URL

Сервис	Пример base URL	Назначение
LMS	http://lms-host:8000/	Основное Django-приложение
SQLChecker	http://sql-checker-host:8000/	Проверка SQL
code_checker API	http://code-checker-host:5001/	API шаблонов и синхронных запусков кода

2. LMS validator API для code_checker

Эти endpoints реализованы в `lms-novadata/lms_project/apps/api/views_validator.py` и используются `code_checker/app/worker.py`.

Base path:

`/api/validator/`

Аутентификация:

X-API-Key: <LMS_API_TOKEN>

Токен должен совпадать с `VALIDATOR_PROGRAMMING_TOKEN` в `.env` LMS.

2.1. Получить список pending-заданий

GET `/api/validator/submissions/?status=pending`

Query parameters:

Параметр	Тип	Обязательный	Описание
status	string	Нет	Фильтр по статусу: pending , processing , checked , failed

Параметр	Тип	Обязательный	Описание
created_after	datetime	Нет	Фильтр по дате создания
created_before	datetime	Нет	Фильтр по дате создания
page_size	integer	Нет	Размер страницы, максимум 100
page	integer	Нет	Номер страницы

Успешный ответ:

```
{
  "count": 1,
  "page": 1,
  "page_size": 20,
  "total_pages": 1,
  "has_next": false,
  "submission_ids": [
    {
      "id": "9ef4e84a-5f7d-4a26-a7b1-7d6c4f0a9e11",
      "status": "pending",
      "created_at": "2026-05-26T08:00:00Z"
    }
  ]
}
```

Ошибки:

HTTP	Причина
403	Нет или неверный X-API-Key
400	Неверный status или некорректная пагинация

2.2. Получить payload задания

```
GET /api/validator/submissions/{submission_id}/
```

Path parameters:

Параметр	Тип	Описание
submission_id	UUID	ID записи в submission_queue

Успешный ответ:

```

{
  "id": "9ef4e84a-5f7d-4a26-a7b1-7d6c4f0a9e11",
  "user_id": 123,
  "task_id": 456,
  "payload": {
    "type": "programming",
    "step_id": 456,
    "user_id": 123,
    "mode": "run_tests",
    "student_code": "def solve():\n    return 1",
    "code": "def solve():\n    return 1",
    "language": "python",
    "environment": "base",
    "test_cases": [
      {
        "input": {
          "args": []
        },
        "output": 1
      }
    ],
    "validator": {
      "test_cases": [
        {
          "input": {
            "args": []
          },
          "output": 1
        }
      ]
    },
    "method_name": "solve",
    "timeout": 15,
    "memory_limit": 256,
    "points": 1
  }
}

```

Ошибки:

HTTP	Причина
403	Нет или неверный X-API-Key
404	Submission не найден или UUID некорректен

2.3. Отправить результат проверки

PATCH /api/validator/submissions/{submission_id}/

Также поддерживается:

POST /api/validator/submissions/{submission_id}/

Фактический request body, который отправляет code_checker worker:

```
{
  "status": "done",
  "report": {
    "passed": true,
    "reports": [],
    "output": "optional stdout"
  }
}
```

Если произошла ошибка:

```
{
  "status": "error",
  "report": {
    "error": "Compilation error"
  }
}
```

Поля:

Поле	Тип	Обязательное	Описание
status	string	Да	done или error по фактической логике worker

Поле	Тип	Обязательное	Описание
report	object	Да	Детальный отчет checker'a

Успешный ответ:

```
{
  "status": "Ок"
}
```

Побочный эффект:

- запись переводится в статус `checked` ;
- результат записывается в `submission_history` ;
- запись удаляется из `submission_queue` .

Важное замечание: в `serializers.py` есть модель, ожидающая поле `result` , но фактический `view` принимает `status` и `report` на верхнем уровне. Для внешней интеграции актуальным считается фактический контракт `status + report` .

3. LMS API статуса programming-задания

Endpoint используется фронтендом LMS для проверки результата текущего пользователя.

```
GET /api/validator/status/?step_id={step_id}
```

Аутентификация: пользовательская сессия Django. Пользователь должен быть авторизован.

Query parameters:

Параметр	Тип	Обязательный	Описание
step_id	integer	Да	ID programming-шага

Ответ, если задание еще в очереди:

```
{
  "status": "in_progress",
  "submission_id": "9ef4e84a-5f7d-4a26-a7b1-7d6c4f0a9e11",
  "queue_status": "pending",
  "created_at": "2026-05-26T08:00:00Z"
}
```

Ответ, если проверка завершена:

```
{
  "status": "done",
  "queue_id": "9ef4e84a-5f7d-4a26-a7b1-7d6c4f0a9e11",
  "completed_at": "2026-05-26T08:01:00Z",
  "execution_time_ms": null,
  "result": {
    "status": "done",
    "report": {
      "passed": true,
      "reports": []
    }
  }
}
```

Ответ, если данных нет:

```
{
  "status": "not_found"
}
```

4. Формат payload programming-задания

LMS формирует payload в момент отправки решения студентом.

Общий формат:

```

{
  "type": "programming",
  "step_id": 456,
  "user_id": 123,
  "mode": "run_tests",
  "student_code": "...",
  "code": "...",
  "language": "python",
  "environment": "base",
  "test_cases": [],
  "validator": {
    "test_cases": []
  },
  "method_name": "solve",
  "timeout": 15,
  "memory_limit": 256,
  "points": 1
}

```

Поля:

Поле	Тип	Обязательное	Описание
type	string	Да	Для code_checker используется programming
step_id	integer	Да	ID шага LMS
user_id	integer	Да	ID пользователя LMS
mode	string	Да	run_tests , stdin_stdout , run_stdin_stdout , run_custom_input
student_code	string	Да	Код студента
code	string	Нет	Дублирует код для совместимости
language	string	Да	Язык: python , java , scala , cpp , csharp , go , haskell , r , rust
environment	string	Да	Окружение: base , spark , pyspark , sklearn , tensorflow , pytorch и т.п.
test_cases	array	Да	Список тест-кейсов

Поле	Тип	Обязательное	Описание
validator.test_cases	array	Да	Основной список тест-кейсов для checker'a
method_name	string/null	Для run_tests да	Имя функции/метода
timeout	integer	Нет	Таймаут из настройки шага
memory_limit	integer	Нет	Лимит памяти из настройки шага
points	number	Нет	Баллы за шаг

4.1. run_tests

Используется для проверки функции/метода.

Пример:

```
{
  "mode": "run_tests",
  "language": "python",
  "environment": "base",
  "method_name": "sum_numbers",
  "student_code": "def sum_numbers(a, b):\n    return a + b",
  "validator": {
    "test_cases": [
      {
        "input": {
          "args": [2, 3],
          "kwargs": {}
        },
        "output": 5
      }
    ]
  }
}
```

4.2. stdin_stdout

Используется для задач, где программа читает stdin и пишет stdout.

Пример:

```
{
  "mode": "stdin_stdout",
  "language": "python",
  "environment": "base",
  "student_code": "a, b = map(int, input().split())\nprint(a + b)",
  "validator": {
    "test_cases": [
      {
        "input": "2 3\n",
        "output": "5\n"
      }
    ]
  }
}
```

В этом режиме `method_name` не требуется.

5. LMS SQL endpoints

Эти endpoints вызываются фронтендом LMS и требуют авторизованного пользователя.

5.1. Выполнить SQL-запрос

POST `/api/sql/execute/`

Request body:

```
{
  "step_id": 456,
  "query": "SELECT * FROM users LIMIT 10"
}
```

Поля:

Поле	Тип	Обязательное	Описание
step_id	integer	Да	ID SQL-шага

Поле	Тип	Обязательное	Описание
query	string	Да	SQL-запрос студента

Успешный ответ:

```
{
  "success": true,
  "columns": ["id", "name"],
  "rows": [
    {
      "id": 1,
      "name": "Alice"
    }
  ],
  "row_count": 1,
  "execution_time": 12.5,
  "limited": true,
  "messages": ["Запрос выполнен успешно"],
  "ast_analysis": {},
  "syntax_valid": true,
  "preview": []
}
```

5.2. Проверить SQL-решение

POST /api/sql/validate/

Request body:

```
{
  "step_id": 456,
  "query": "SELECT id, name FROM users"
}
```

Успешный ответ:

```

{
  "is_correct": true,
  "points_earned": 1,
  "message": "Задание выполнено правильно!",
  "validation_details": {
    "syntax_valid": true,
    "is_correct": true,
    "execution_matches": true,
    "keywords_check": true,
    "strict_result_match": true,
    "columns_match": true,
    "row_count_match": true,
    "elements_match": true,
    "execution_time": 12.5,
    "missing_functions": [],
    "ast_analysis": {},
    "suggestions": [],
    "student_columns": ["id", "name"],
    "reference_columns": ["id", "name"]
  },
  "student_preview": [],
  "reference_preview": [],
  "attempt_details": {
    "step_id": 456,
    "max_points": 1,
    "database_type": "postgres"
  },
  "attempts_used": 1,
  "remaining_attempts": 2,
  "progress_status": "completed"
}

```

Ошибки:

HTTP	Причина
400	Нет step_id , пустой query , шаг не SQL, превышены попытки
404	Шаг или SQL-контент не найден
500	Ошибка SQLChecker или внутренняя ошибка LMS

6. SQLChecker API

Base URL задается в LMS через SQL_CHECKER_URL .

6.1. Синхронная проверка SQL

POST /check

Request body:

```
{
  "external_id": "step_456_validate_abcd1234_1710000000",
  "db_type": "postgres",
  "init_script": "CREATE TABLE users(id int, name text); INSERT INTO users VALUES (1, 'Alice');",
  "reference_query": "SELECT id, name FROM users",
  "student_query": "SELECT id, name FROM users",
  "check_performance": false,
  "database_template": "",
  "is_ddl": false,
  "is_dml": false,
  "metadata": {
    "step_id": 456,
    "step_title": "SQL task",
    "mode": "validate",
    "max_rows": 100,
    "timeout_seconds": 30,
    "ignore_order": true,
    "check_column_names": false
  },
  "required_functions": []
}
```

Поля:

Поле	Тип	Обязательное	Описание
external_id	string/null	Нет	Внешний ID для трассировки
db_type	string	Нет	postgres или clickhouse , по умолчанию postgres

Поле	Тип	Обязательное	Описание
init_script	string	Нет	SQL для подготовки временной БД
reference_query	string	Да	Эталонный запрос
student_query	string	Да	Запрос студента
check_performance	boolean	Нет	Сравнение производительности
database_template	string/null	Нет	Имя шаблонной БД
is_ddl	boolean	Нет	Явно считать запрос DDL
is_dml	boolean	Нет	Явно считать запрос DML
metadata	object/null	Нет	Дополнительные параметры
required_functions	array	Нет	Обязательные SQL-функции/ ключевые слова

Response body:

```
{
  "task_id": null,
  "external_id": "step_456_validate_abcd1234_1710000000",
  "is_correct": true,
  "execution_matches": true,
  "error_message": null,
  "ast_analysis": {
    "syntax_valid": true,
    "query_type": "SELECT"
  },
  "performance_comparison": null,
  "student_result_preview": [
    {
      "id": 1,
      "name": "Alice"
    }
  ],
  "reference_result_preview": [
    {
      "id": 1,
      "name": "Alice"
    }
  ],
  "execution_time_ms": 12.5,
  "timestamp": "2026-05-26T08:00:00",
  "missing_functions": []
}
```

6.2. Пакетная проверка

POST /check/batch

Request body:

```
{
  "checks": [
    {
      "external_id": "task-1",
      "db_type": "postgres",
      "init_script": "",
      "reference_query": "SELECT 1",
      "student_query": "SELECT 1"
    }
  ],
  "max_concurrent": 5
}
```

Response body: массив объектов формата `/check` .

6.3. Асинхронная проверка SQLChecker

POST `/check/async`

Request body: такой же, как `/check` .

Response body:

```
{
  "task_id": "4d5df4aa-1a10-4d13-8d18-438e3a401a3d",
  "external_id": "task-1",
  "status": "pending",
  "check_status_url": "/tasks/4d5df4aa-1a10-4d13-8d18-438e3a401a3d",
  "details_url": "/tasks/4d5df4aa-1a10-4d13-8d18-438e3a401a3d/details"
}
```

6.4. Статус задач SQLChecker

GET `/tasks?status=pending&limit=100&offset=0`

GET `/tasks/{task_id}`

Статусы:

pending
running
completed
failed

6.5. Мониторинг SQLChecker

GET /health
GET /stats
GET /

Пример /health :

```
{  
  "status": "healthy",  
  "database_connected": true,  
  "active_tasks": 0,  
  "uptime_seconds": 0.0  
}
```

6.6. Статический анализ SQL

GET /analyze?query=SELECT%201

Response body:

```
{  
  "query": "SELECT 1",  
  "ast_analysis": {  
    "syntax_valid": true,  
    "query_type": "SELECT"  
  }  
}
```

7. code_checker API

Этот API запускается на порту 5001 . Он полезен для получения шаблонов кода и ручных синхронных запусков. Основной production-поток programming-проверки использует worker и LMS validator API.

7.1. Получить доступные языки и окружения

GET /info

Response body:

```
{
  "python": {
    "base": {
      "versions": ["3.14"],
      "default": "3.14"
    }
  },
  "java": {
    "base": {
      "versions": ["21"],
      "default": "17"
    }
  }
}
```

Важное замечание: для java_base в текущем config есть риск рассинхронизации default 17 и доступного image 21 .

7.2. Получить шаблон кода

POST /get_template

Request body:

```
{
  "language": "python",
  "language_version": "3.14",
  "environment": "base",
  "method_name": "solve",
  "validator": {
    "test_cases": [],
    "required_constructs": {}
  }
}
```

Response body:

```
{
  "template": "def solve():\n    pass"
}
```

7.3. Запустить полную проверку кода

POST /code_test

Request body:

```

{
  "id": "submission-1",
  "payload": {
    "language": "python",
    "language_version": "3.14",
    "environment": "base",
    "method_name": "solve",
    "student_code": "def solve():\n    return 1",
    "validator": {
      "test_cases": [
        {
          "input": {
            "args": []
          },
          "output": 1
        }
      ],
      "required_constructs": {}
    }
  }
}

```

Response body:

```

{
  "id": "submission-1",
  "passed": true,
  "reports": []
}

```

7.4. Запустить код с custom input

POST /code_run

Request body:

```
{
  "id": "submission-1",
  "payload": {
    "language": "python",
    "language_version": "3.14",
    "environment": "base",
    "method_name": "sum_numbers",
    "student_code": "def sum_numbers(a, b):\n    return a + b",
    "validator": {
      "test_cases": [
        {
          "input": {
            "args": [1, 2]
          },
          "output": 3
        }
      ]
    }
  },
  "custom_input": [10, 20]
}
```

Response body:

```
{
  "id": "submission-1",
  "output": "30"
}
```

8. Коды ошибок

8.1. LMS validator API

HTTP	Причина
400	Неверный query parameter, отсутствует <code>step_id</code> , некорректный запрос
403	Неверный validator token
404	Submission или шаг не найден

HTTP	Причина
500	Ошибка финализации задания или внутренняя ошибка

8.2. SQLChecker

HTTP	Причина
422	Ошибка валидации Pydantic
500	Ошибка выполнения проверки, БД или внутренней логики

8.3. code_checker API

HTTP	Причина
400	Нет <code>student_code</code> для запуска/проверки
404	Не удалось сгенерировать <code>template</code> для указанного профиля
422	Ошибка валидации Pydantic
500	Внутренняя ошибка checker'a

9. Контрольные сценарии интеграции

9.1. Проверка programming через worker

1. LMS создает запись `submission_queue` со статусом `pending` .
2. `code_checker` делает `GET /api/validator/submissions/?status=pending` .
3. `code_checker` получает ID задания.
4. `code_checker` делает `GET /api/validator/submissions/{id}/` .
5. `code_checker` запускает `run_check(payload)` .
6. `code_checker` отправляет `PATCH /api/validator/submissions/{id}/` с `status` и `report` .
7. LMS переносит результат в `submission_history` .

9.2. Проверка SQL

1. Фронтенд LMS отправляет `POST /api/sql/validate/` .
2. LMS получает `StepContentsSQL` .

3. LMS формирует `SQLCheckRequest` .
4. LMS отправляет `POST {SQL_CHECKER_URL}/check` .
5. `SQLChecker` возвращает `CheckResultResponse` .
6. LMS преобразует ответ в формат фронтенда и обновляет `StepProgress` .

10. Известные расхождения и риски контрактов

- `SubmissionResultSerializer` описывает поле `result` , но фактический `ValidatorSubmissionView` принимает `status` и `report` на верхнем уровне.
- `code_checker` worker не вызывает отдельный endpoint для перевода `submission` в `processing` ; при нескольких worker'ах возможна конкуренция за `pending`-задания.
- `code_checker` API `TaskPayload.method_name` в Pydantic-модели обязательный, но основной worker-поток для `stdin_stdout` работает напрямую через `run_check(payload)` и может обходиться без `method_name` .
- `SQL_CHECKER_URL` сейчас задан в `settings.py` жестко, поэтому фактический URL `SQLChecker` должен быть проверен при развертывании.
- В `code_checker` есть риск рассинхронизации `Java default version` и доступного `Docker image`.