СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире важную роль играют обработка данных и математические расчеты. Матрицы широко используются в различных областях, таких как машинное обучение, физика, экономика и многие другие. Поэтому разработка приложения, позволяющего эффективно и быстро находить определитель исходной матрицы, является актуальной задачей. Интеграция приложения с API библиотеки "NumPy", позволяющей производить вычисления с массивами и матрицами в Python, значительно упрощает и ускоряет процесс работы с данными.

Объектом исследования данной работы является приложение для поиска определителя исходной матрицы. Предмет исследования заключается в особенностях проектирования, разработке и реализации архитектуры данного приложения, включая процессы интеграции и развертывания, а также функциональные особенности его компонентов.

Цель работы — разработать функциональное приложение и реализовать процесс его развертывания, а также обеспечить его интеграцию с заданным API.

Методы исследования, которые будут использоваться в данной работе:

- теоретический анализ;
- моделирование;
- эксперимент.

В ходе работы будут выполнены следующие задачи:

- проектирование архитектуры приложения;
- реализация функциональной части приложения;
- реализация серверной части приложения;
- развертывание серверной части приложения;
- реализация клиентской части приложения;
- развертывание клиентской части приложения;
- настройка конфигурации многоконтейнерного развертывания приложения.

Основными источниками информации будут документации библиотек "Django", "NumPy", а также книга по проектированию API.

В ходе работы будет осуществлена разработка структуры базы данных, проектирование архитектуры приложения, разработка набора функций для обработки данных, разработка веб-сайта и интеграция компонентов приложения между собой.

1 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ «ПОИСК ОПРЕДЕЛИТЕЛЯ ИСХОДНОЙ МАТРИЦЫ»

На Рисунке 1.1 представлена структурная схема приложения.

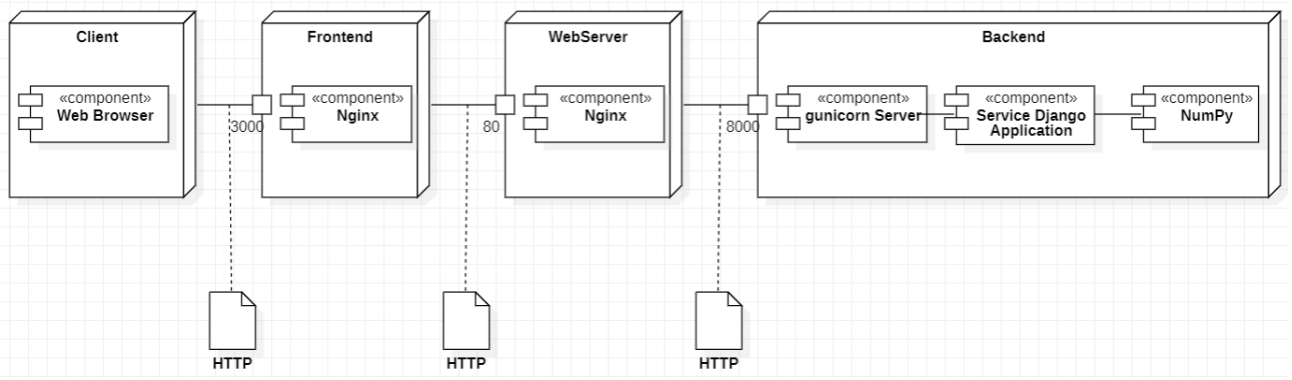

Рисунок 1.1 — Структурная схема приложения

В данной системе обмен данными между контейнерами и клиентской частью осуществляются по протоколу HTTP, который является универсальным решением для веб-коммуникаций за счет передачи данных в унифицированном формате [1].

Узел "Client" отвечает за устройство пользователя.

Узел "Frontend" — это Docker-контейнер, в котором находится компонент "Nginx", отвечающий за хостинг статических файлов приложения на стороне клиента [2, 3].

Узел "WebServer" — это Docker-контейнер, в котором находится прокси-сервер «nginx», который предоставляет возможность доступа к сервисной части с использованием gunicorn [4].

Узел "Backend" — это Docker-контейнер, в котором находится код сервисной части приложения. Компонент "Gunicorn" — это веб-сервер, на котором запускается Python-приложение "Service Django Application". Компонент "NumPy" — библиотека для языка программирования Python [5].

Внешние интерфейсы приложения включают в себя следующие компоненты:

1. Gunicorn: Gunicorn является uWSGI HTTP-сервером для приложений Python. Он предоставляет внешний интерфейс для приложения, позволяя ему получать HTTP-запросы и отправлять HTTP-ответы.
2. Nginx: Nginx является веб-сервером, который используется в качестве обратного прокси-сервера для приложения. Он предоставляет внешний интерфейс для приложения, позволяя ему обрабатывать HTTP-запросы и отправлять HTTP-ответы.
3. Docker: Docker предоставляет внешний интерфейс для приложения, позволяя развернуть его в контейнере.

Внутренние интерфейсы приложения включают в себя следующие компоненты:

1. Django: Django является веб-фреймворком Python, который предоставляет внутренний интерфейс для приложения. Он обрабатывает HTTP-запросы и ответы, а также предоставляет инструменты для работы с базой данных и шаблонами [6].
2. База данных: база данных является внутренним интерфейсом для приложения.
3. React: React является библиотекой JavaScript для создания пользовательских интерфейсов. Он предоставляет внутренний интерфейс для приложения, позволяя создавать динамические веб-страницы [7].

Взаимодействие между этими компонентами происходит следующим образом:

- пользовательский интерфейс создается с помощью React, браузер загружает JavaScript-файлы и рендерит страницу;
- приложение React может отправлять дополнительные HTTP-запросы в Django для получения данных или отправки их на сервер;

- Nginx перенаправляет запрос в Gunicorn, который запускает приложение Django;
- Django обрабатывает запрос и генерирует HTTP-ответ. Если необходимо, он может получить данные из базы данных или отправить их туда;
- Gunicorn отправляет HTTP-ответ обратно в Nginx;
- Nginx отправляет HTTP-ответ обратно в браузер пользователя.

2 РЕАЛИЗАЦИЯ ФУНКЦИОНАЛЬНОЙ ЧАСТИ ПРИЛОЖЕНИЯ «ПОИСК ОПРЕДЕЛИТЕЛЯ ИСХОДНОЙ МАТРИЦЫ»

2.1 Спецификация используемого API библиотеки

Библиотека "NumPy" (Numerical Python) — это основной инструмент для работы с многомерными массивами и матрицами в языке программирования Python. Она предоставляет широкие возможности для выполнения операций линейной алгебры, статистики, преобразования Фурье и других математических операций.

Основное предназначение "NumPy" — это ускорение вычислений и упрощение работы со сложными массивами данных. В библиотеке реализованы многие стандартные функции, такие как сложение, умножение, транспонирование, скалярное произведение и другие.

Основные методы "NumPy" включают:

1. Создание массивов: np.array(), np.arange(), np.zeros(), np.ones().
2. Математические операции: np.add(), np.subtract(), np.multiply(), np.divide().
3. Линейная алгебра: np.dot(), np.linalg.inv(), np.linalg.norm().
4. Статистика: np.mean(), np.std(), np.sum().
5. Индексация и срезы: array[index], array[start:stop:step].

Принципы использования библиотеки "NumPy" включают в себя:

1. Импортирование библиотеки.
2. Создание массивов.
3. Выполнение операций.
4. Использование функций и методов.

2.2 Описание алгоритма

Метод find_det предназначен для нахождения определителя матрицы, переданной в качестве аргумента. Он использует библиотеку "NumPy" для работы с матрицами. Метод сначала создает массив "NumPy" из входной матрицы, затем находит определитель этой матрицы с помощью функции np.linalg.det().

Затем метод сохраняет найденный определитель и переданную матрицу в словари self.determinants и self.matrices, используя уникальный идентификатор self.id для индексации. После этого увеличивает self.id на 1 для следующих операций.

В конце метод возвращает идентификатор матрицы self.id-1 и найденное значение определителя.

2.3 Описание реализации

Функциональная часть приложения, в которой используется библиотека "NumPy", представлена в Листинге 2.1.

Листинг 2.1 — Функциональная часть приложения

```
    def find_det(self, matrix: list[list[float]]) -> tuple[int, float]:
        arr = np.array(matrix)
        result = np.linalg.det(arr)
        self.determinants[self.id] = result
        self.matrices[self.id] = matrix
        self.id += 1
        return self.id-1, result
```

2.4 Описание тестирования

Для тестирования метода find_det воспользуемся программным обеспечением Postman [8]. Необходимо ввести квадратную матрицу, для которой требуется найти определитель. Результат тестирования представлен на Рисунке 2.1.

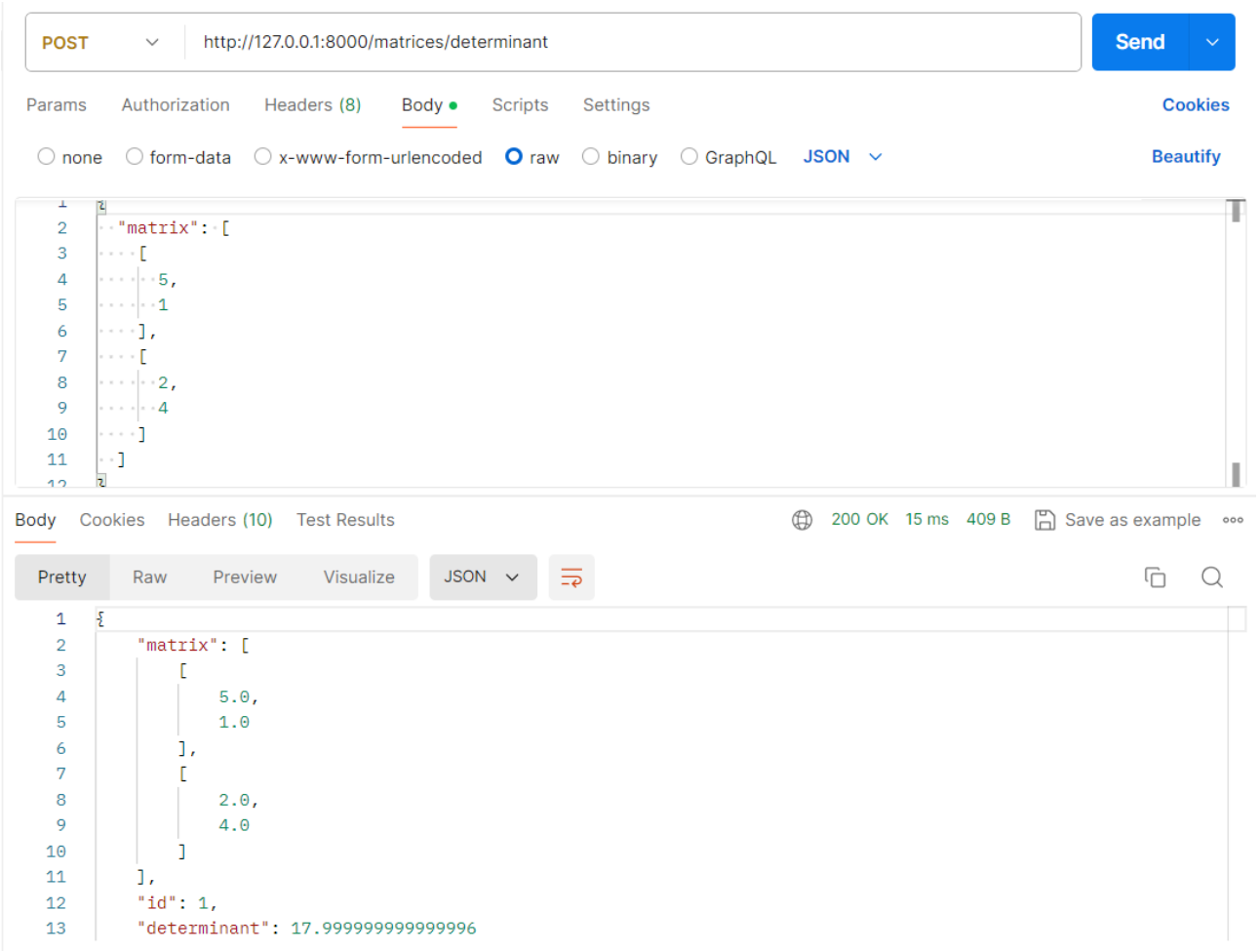

Рисунок 2.1 — Результат тестирования

3 РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ «ПОИСК ОПРЕДЕЛИТЕЛЯ ИСХОДНОЙ МАТРИЦЫ»

3.1 Проектирование внешнего API

Для разработки сервисной части приложения была построена таблица целей API (Таблица 3.1) [9]. Для роли «Пользователь» предусмотрен сценарий «Управление матрицами». В результате было выделено четыре цели.

Таблица 3.1 — Цели API

Кто (Пользователь)	**Что (Действие)**	**Как (Этапы)**	**Входные данные (Источник)**	**Выходные данные (Использование)**	**Цели**
Пользователь	Управление матрицами	Получение определителя матрицы	Информация для расчётов (Пользователь)	Вычисления (Поиск матрицы и её определителя, Удаление матрицы и её определителя, Получение списка всех матриц и их определителей)	Получение определителя матрицы
		Поиск матрицы и её определителя	Информация для поиска (Получение определителя матрицы)	Матрица и её определитель (-)	Поиск матрицы и её определителя
		Удаление матрицы и её определителя	Информация для удаления (Получение определителя матрицы)	-	Удаление матрицы и её определителя
		Получение списка всех матриц и их определителей	Информация для получения списка (Получение определителя матрицы)	Список всех матриц и их определителей (-)	Получение списка всех матриц и их определителей

В рамках проектирования был определен набор схем входных и выходных данных, представленных в Таблицах 3.2-3.3.

Таблица 3.2 — Схема данных "Matrix"

Имя	Тип	Обязательно	Описание
matrix	Двумерный массив из чисел с плавающей точкой	Да	Матрица

Таблица 3.3 — Схема данных "ResponseDeterminant"

Имя	Тип	Обязательно	Описание
matrix	Двумерный массив из чисел с плавающей точкой	Да	Матрица
id	Целое число	Да	Идентификатор матрицы и определителя
determinant	Число с плавающей точкой	Да	Значение определителя

3.2 Спецификация реализованного внешнего API

Спецификация реализованного внешнего API представлена в Листинге 3.1.

Листинг 3.1 — Спецификация реализованного API

```
openapi: 3.0.3
info:
  title: ''
  version: 0.0.0
paths:
  /api/schema/:
    get:
      operationId: api_schema_retrieve
      description: |-
        OpenApi3 schema for this API. Format can be selected via content nego-
tiation.

        - YAML: application/vnd.oai.openapi
        - JSON: application/vnd.oai.openapi+json
      parameters:
      - in: query
        name: format
        schema:
          type: string
          enum:
          - json
          - yaml
      - in: query
        name: lang
        schema:
```

Продолжение Листинга 3.1

```
          type: string
      tags:
      - api
      security:
      - cookieAuth: []
      - basicAuth: []
      - {}
      responses:
        '200':
          content:
            application/vnd.oai.openapi:
              schema:
                type: object
                additionalProperties: {}
            application/yaml:
              schema:
                type: object
                additionalProperties: {}
            application/vnd.oai.openapi+json:
              schema:
                type: object
                additionalProperties: {}
            application/json:
              schema:
                type: object
                additionalProperties: {}
          description: ''
  /matrices/determinant:
    get:
      operationId: matrices_determinant_retrieve
      description: get all matrices and their determinants
      tags:
      - matrices
      responses:
        '422':
          description: No response body
        '200':
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Matrix'
          description: ''
        '404':
          description: No response body
    post:
      operationId: matrices_determinant_create
      description: find the determinant of the entered matrix
      tags:
      - matrices
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Matrix'
          application/x-www-form-urlencoded:
            schema:
              $ref: '#/components/schemas/Matrix'
          multipart/form-data:
            schema:
              $ref: '#/components/schemas/Matrix'
        required: true
```

Продолжение Листинга 3.1

```
      responses:
        '422':
          description: No response body
        '200':
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ResponseDeterminant'
          description: ''
        '404':
          description: No response body
  /matrices/determinant/{id}:
    get:
      operationId: matrices_determinant_retrieve_2
      description: get the matrix and its determinant by id
      parameters:
      - in: path
        name: id
        schema:
          type: integer
        required: true
      tags:
      - matrices
      responses:
        '422':
          description: No response body
        '200':
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ResponseDeterminant'
          description: ''
        '404':
          description: No response body
    delete:
      operationId: matrices_determinant_destroy
      description: delete a matrix and its determinant by id
      parameters:
      - in: path
        name: id
        schema:
          type: integer
        required: true
      tags:
      - matrices
      responses:
        '422':
          description: No response body
        '200':
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ResponseDeterminant'
          description: ''
        '404':
          description: No response body
components:
  schemas:
    Matrix:
      type: object
      properties:
        matrix:
```

Продолжение Листинга 3.1

```
          type: array
          items:
            type: array
            items:
              type: number
              format: double
            minItems: 2
          minItems: 2
      required:
      - matrix
    ResponseDeterminant:
      type: object
      properties:
        matrix:
          type: array
          items:
            type: array
            items:
              type: number
              format: double
            minItems: 2
          minItems: 2
        id:
          type: integer
        determinant:
          type: number
          format: double
      required:
      - determinant
      - id
      - matrix
  securitySchemes:
    basicAuth:
      type: http
      scheme: basic
    cookieAuth:
      type: apiKey
      in: cookie
      name: sessionid
```

Спецификация в иллюстрированном виде частично представлена на Рисунке 3.1.

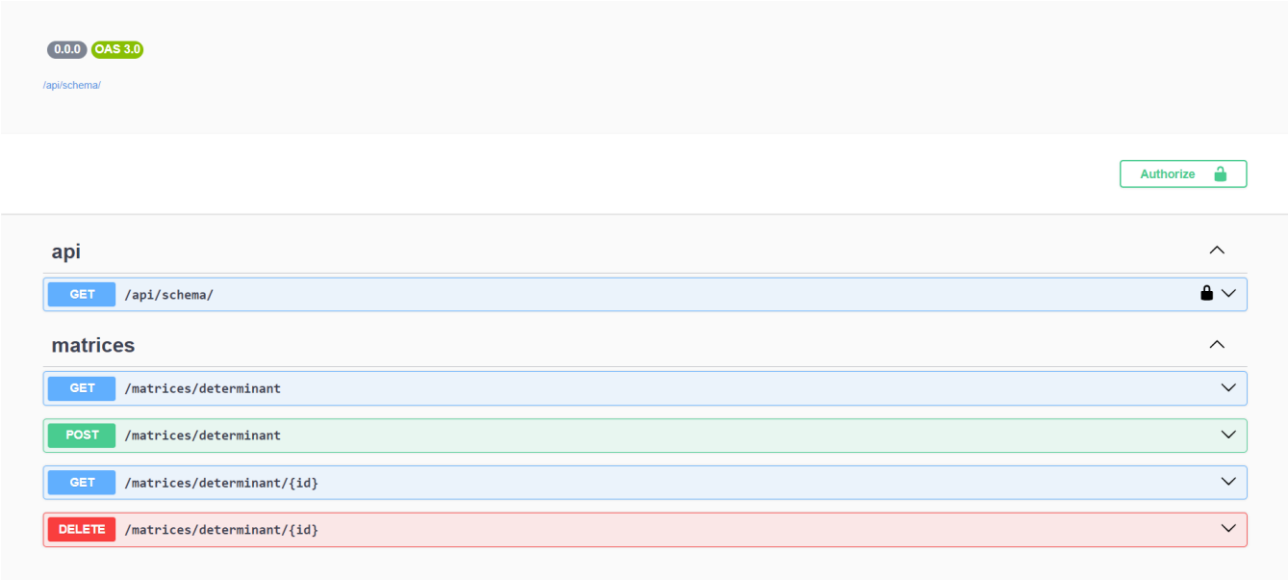

Рисунок 3.1 — Графическое представление спецификации API

Описание спецификации API (Листинг 3.1) имеет следующие элементы: версию используемой спецификации, список путей, набор компонентов.

Список путей включает в себя индивидуальный путь, соответствующий каждой операции. Для каждого пути определен набор действий. Каждое действие содержит в себе описание данного действия, информацию о теле запроса, которое включает в себя описание тела запроса, тип передаваемого контента в теле запроса. Далее следует описание ответов. Описание ответов включает описание обязательного «положительного» ответа, который описывается, как код ответа 200 протокола HTTP. Описание ответа включает в себя краткую характеристику, а также формат тела ответа. Далее следует описание ошибок, которые характеризуются различными кодами ответов протокола HTTP из группы ошибок клиента. Такие коды начинаются с цифры 4.

Описание набора компонентов в спецификации примера включает в себя описание определенных в рамках планирования схем данных. Описание каждой схемы данных включает в себя определение того, что данный тип данных является объектом, который имеет набор обязательных свойств. При описании свойств упомянуты их имена и типы.

3.3 Описание процесса разработки серверной части приложения

3.3.1 Технологический стек

Для разработки сервисной части приложения использовались следующие библиотеки:

1. Django — свободный фреймворк для веб-приложений на языке Python.
2. Django Rest Framework — это мощный набор инструментов для создания веб-сервисов и API на основе фреймворка Django. DRF предоставляет готовые компоненты, такие как сериализаторы, представления, маршрутизация и система аутентификации, что упрощает создание API и позволяет сосредоточиться на бизнес-логике.
3. Django Cors Headers — библиотека, необходимая для добавления заголовков CORS к ответам сервера.

3.3.2 Артефакты процесса разработки

В первую очередь была реализована модель матрицы (Листинг 3.2).

Листинг 3.2 — Содержимое файла models.py

```
class Matrix(models.Model):
    id = models.IntegerField(primary_key=True, editable=False)
    rows = models.JSONField(null=False)


class MatrixDTO:
    id: UUID
    matrix: list[list[float]]

    def __init__(self, matrix: list[list[float]]):
        self.id = uuid4()
        self.matrix = matrix
```

Так как данные передаются сервису в формате JSON, а также ответ отправляется в том же формате, был предусмотрен процесс сериализации входных данных в объект для работы с ним и результата в JSON (Листинг 3.3).

Листинг 3.3 — Содержимое файла serializers.py

```
class MatrixSerializer(serializers.Serializer):
    matrix = serializers.ListField(
        min_length=2,
        child=serializers.ListField(
            min_length=2,
            child=serializers.FloatField(),
        ),
    )

    def validate_matrix(self, value):
        try:
            matrix = np.array(value)
            if matrix.ndim != 2:
                raise serializers.ValidationError(
                    "Matrix must be two-dimensional array"
                )
        except ValueError:
            raise serializers.ValidationError("Matrix rows must be the same
length")
        return value


class UUIDSerializer(serializers.Serializer):
    id = serializers.IntegerField(required=True)


class ResponseDeterminantSerializer(MatrixSerializer):
    id = serializers.IntegerField(required=True)
    determinant = serializers.FloatField(required=True)


class ResponseListMatricesSerializer(serializers.Serializer):
    id = serializers.CharField(required=True)
    matrix = serializers.ListField(
        min_length=2,
        child=serializers.ListField(
            min_length=2,
            child=serializers.FloatField(),
        ),
    )
```

Далее был реализован сервис матриц (Листинг 3.4), отвечающий за бизнес-логику.

Листинг 3.4 — Код сервиса матриц

```
class MatrixService:
    matrices: dict[int: MatrixDTO]
    determinants: dict[int: float]
    id = 1
    def __init__(self):
```

Продолжение Листинга 3.4

```
        self.matrices = {}
        self.determinants = {}

    def find_det(self, matrix: list[list[float]]) -> tuple[int, float]:
        arr = np.array(matrix)
        result = np.linalg.det(arr)
        self.determinants[self.id] = result
        self.matrices[self.id] = matrix
        self.id += 1
        return self.id-1, result

    def get_matrices(self):
        result = []
        for key, value in self.matrices.items():
            result.append({"id": str(key), "matrix": value, "determinant":
self.determinants[key]})
        return result

    def find_matrix_and_determinant(self, id: int) -> dict | None:
        try:
            found = self.matrices[id]
        except KeyError:
            return None
        return {"id": str(id), "matrix": found, "determinant": self.determi-
nants[id]}

    def delete_matrix_and_determinant(self, id: int) -> bool:
        try:
            self.matrices.pop(id)
        except KeyError:
            return False
        self.determinants.pop(id)
        return True
```

Для обработки входящих запросов был разработан контроллер запросов, связанных с матрицами (Листинг 3.5).

Листинг 3.5 — Код контроллера матриц

```
class MatrixViewSet(ViewSet):
    m_service = MatrixService()

    @extend_schema(
        description="find the determinant of the entered matrix",
        request=MatrixSerializer,
        responses={
            status.HTTP_422_UNPROCESSABLE_ENTITY: None,
            status.HTTP_200_OK: ResponseDeterminantSerializer,
            status.HTTP_404_NOT_FOUND: None,
        },
        auth=False,
    )
    @action(detail=False, methods=["POST"])
    def post_matrix(self, request):
        serializer = MatrixSerializer(data=request.data)
        if not serializer.is_valid():
            return Response(
                status=status.HTTP_422_UNPROCESSABLE_ENTITY,      data=serial-
izer.errors
```

Продолжение Листинга 3.5

```
            )
        id,     determinant     =     self.m_service.find_det(serializer.vali-
dated_data["matrix"])
        print(id)
        return Response(
            status=status.HTTP_200_OK,
            data=ResponseDeterminantSerializer(
                {"id": id,
                 "determinant": determinant,
                 "matrix": serializer.validated_data["matrix"]}).data,
        )

    @extend_schema(
        description="get all matrices and their determinants",
        parameters=[],
        responses={
            status.HTTP_422_UNPROCESSABLE_ENTITY: None,
            status.HTTP_200_OK: MatrixSerializer,
            status.HTTP_404_NOT_FOUND: None,
        },
        auth=False,
    )
    @action(detail=False, methods=["GET"])
    def get_matrix_list(self, _):
        response = self.m_service.get_matrices()
        return Response(
            status=status.HTTP_200_OK,
            data=ResponseDeterminantSerializer(response, many=True).data,
        )

    @extend_schema(
        description="get the matrix and its determinant by id",
        responses={
            status.HTTP_422_UNPROCESSABLE_ENTITY: None,
            status.HTTP_200_OK: ResponseDeterminantSerializer,
            status.HTTP_404_NOT_FOUND: None,
        },
        auth=False,
    )
    @action(detail=False, methods=["GET"])
    def get_matrix(self, _, id: int):
        serializer = UUIDSerializer(data={"id": id})
        if not serializer.is_valid():
            return Response(
                status=status.HTTP_422_UNPROCESSABLE_ENTITY,      data=serial-
izer.errors
            )
        result = self.m_service.find_matrix_and_determinant(
            serializer.validated_data["id"],
        )
        if result is None:
            return Response(status=status.HTTP_404_NOT_FOUND)
        return Response(
            status=status.HTTP_200_OK,
            data=ResponseDeterminantSerializer(result).data,
        )

    @extend_schema(
        description="delete a matrix and its determinant by id",
        request=UUIDSerializer,
        responses={
```

Продолжение Листинга 3.5

```
            status.HTTP_422_UNPROCESSABLE_ENTITY: None,
            status.HTTP_200_OK: ResponseDeterminantSerializer,
            status.HTTP_404_NOT_FOUND: None,
        },
        auth=False,
    )
    @action(detail=False, methods=["POST"])
    def delete_matrix(self, _, id):
        serializer = UUIDSerializer(data={"id": id})
        if not serializer.is_valid():
            return Response(
                status=status.HTTP_422_UNPROCESSABLE_ENTITY,      data=serial-
izer.errors
            )
        result = self.m_service.delete_matrix_and_determinant(
            serializer.validated_data["id"],
        )
        if not result:
            return Response(status=status.HTTP_404_NOT_FOUND)
        return Response(
            status=status.HTTP_200_OK,
        )
```

Для того чтобы Django мог правильно адресовать запросы, были добавлены пути в файл urls.py (Листинг 3.6).

Листинг 3.6 — Содержимое файла urls.py

```
urlpatterns = [
    path('matrices/determinant',     MatrixViewSet.as_view({'get':     'get_ma-
trix_list', 'post': 'post_matrix'}),
         name='matrix_transpose'),
    path('matrices/determinant/<int:id>',        MatrixViewSet.as_view({'get':
'get_matrix', 'delete': 'delete_matrix'}), name='matrix_operations'),
    path('api/schema/', SpectacularAPIView.as_view(), name='schema'),
    path('api/docs/',       SpectacularSwaggerView.as_view(url_name='schema'),
name='docs'),
]
```

3.4 Тестирование сервисной части приложения

Для тестирования сервисной части использовался встроенный в фреймворк веб-сервер и программное обеспечение Postman. В Таблицах 3.4-3.10 представлен набор тест-кейсов. На Рисунках 3.2-3.9 представлены результаты тестирования.

Таблица 3.4 — Тест-кейс успешного нахождения определителя

Номер	1	
Название	Нахождение определителя	
Входные данные	Ожидаемый результат	Результат

Матрица:
{
"matrix": [
[
2,
0
],
[
0,
2
]
]
}
	200 — Определитель успешно найден	200 — Пройден

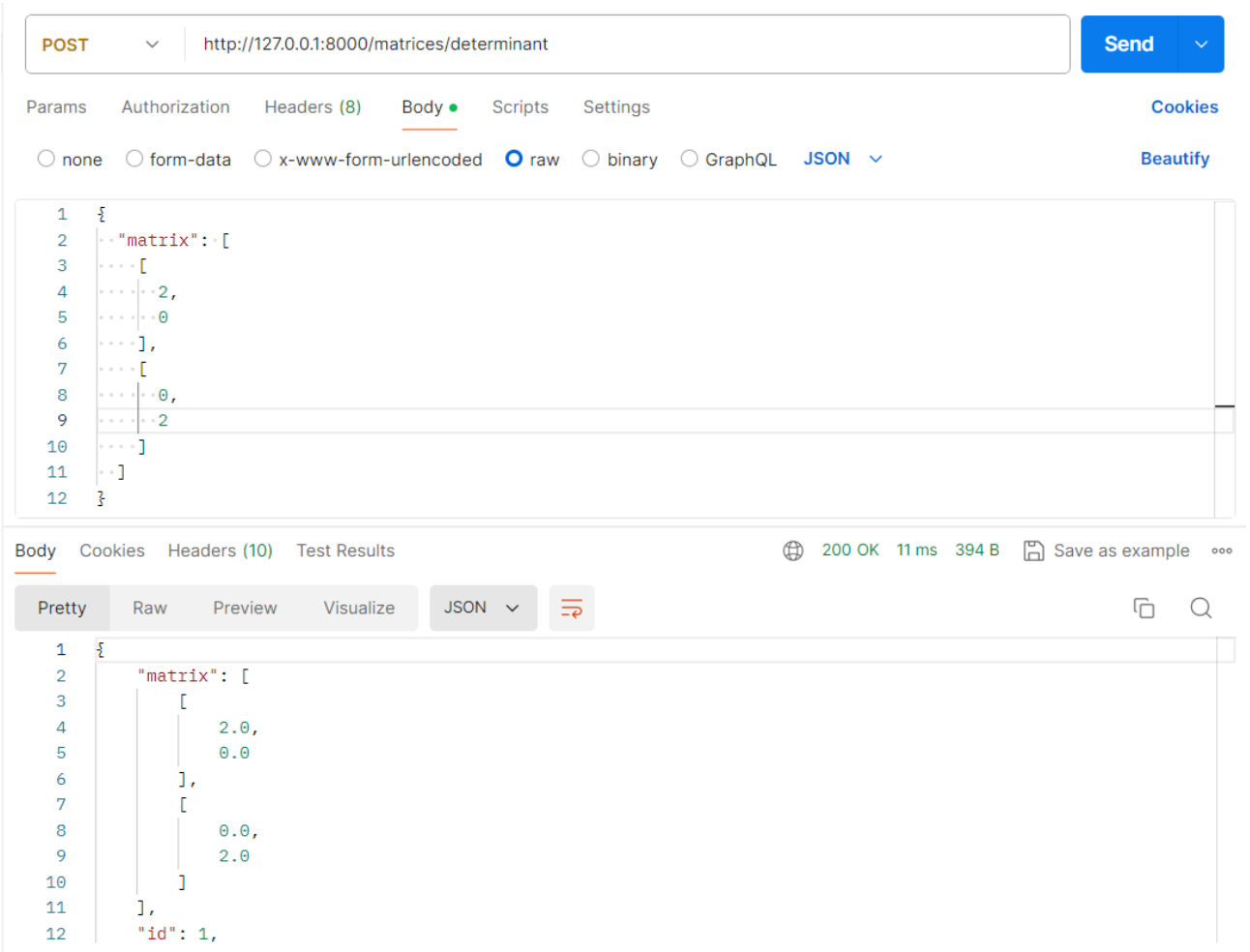

Рисунок 3.2 — Результат тест-кейса №1

Таблица 3.5 — Тест-кейс успешного получения списка всех матриц и определителей

Номер	2	
Название	Получение списка всех матриц и определителей	
Входные данные	Ожидаемый результат	Результат

Продолжение Таблицы 3.5

Отсутствуют	200 — Список успешно получен	200 — Пройден

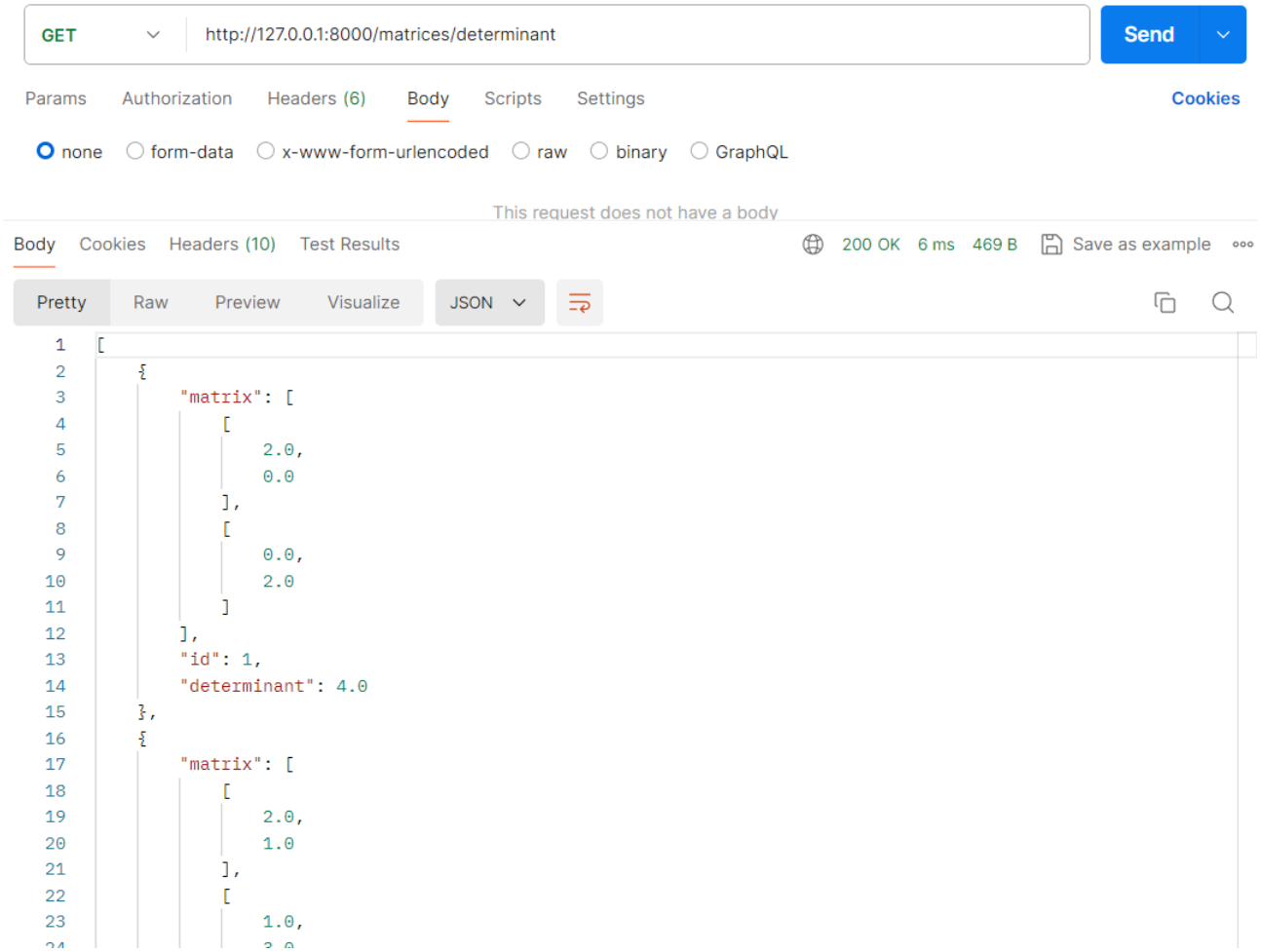

Рисунок 3.3 — Результат тест-кейса №2

Таблица 3.6 — Тест-кейс успешного получения матрицы и её найденного определителя по ID

Номер	3	
Название	Получение матрицы и её найденного определителя по ID	
Входные данные	Ожидаемый результат	Результат
1	200 — Матрица и её найденный определитель получены	200 — Пройден

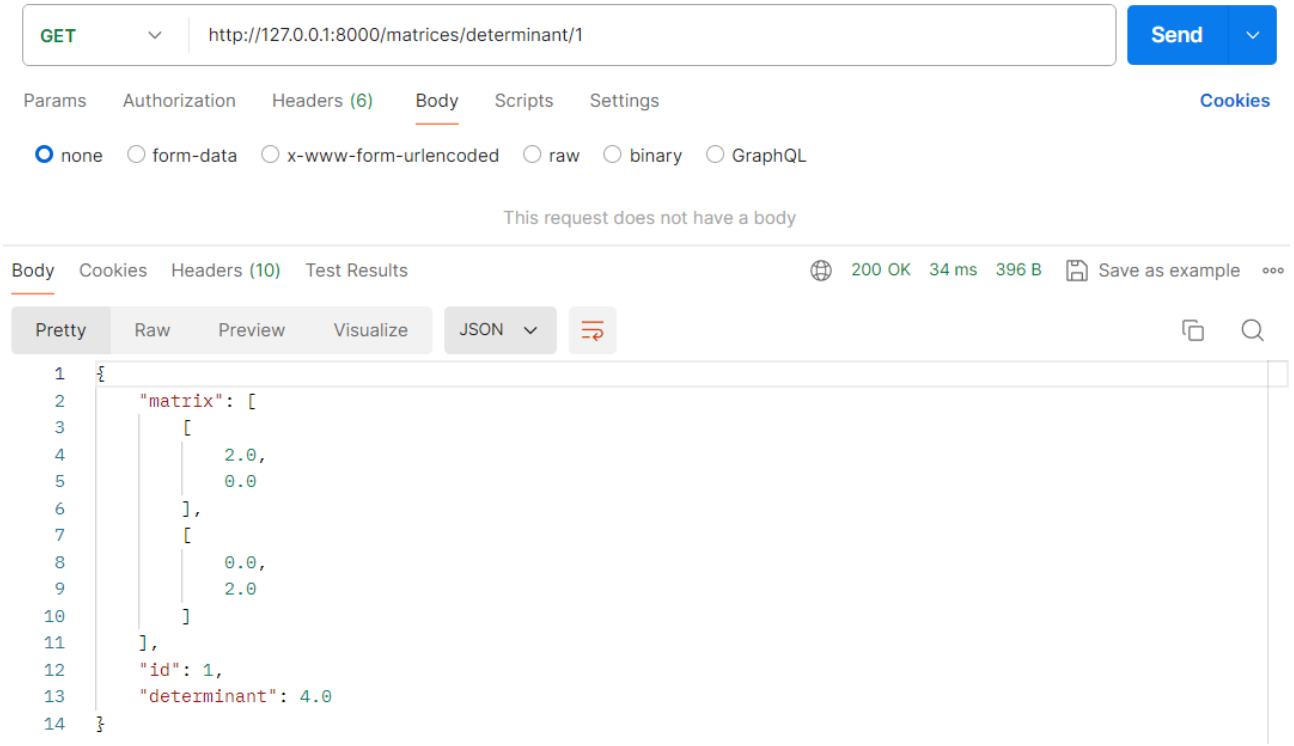

Рисунок 3.4 — Результат тест-кейса №3

Таблица 3.7 — Тест-кейс успешного удаления матрицы и её найденного определителя

Номер	4	
Название	Удаление матрицы и её найденного определителя	
Входные данные	Ожидаемый результат	Результат
1	200 — Матрица и её найденный определитель успешно удалены	200 — Пройден

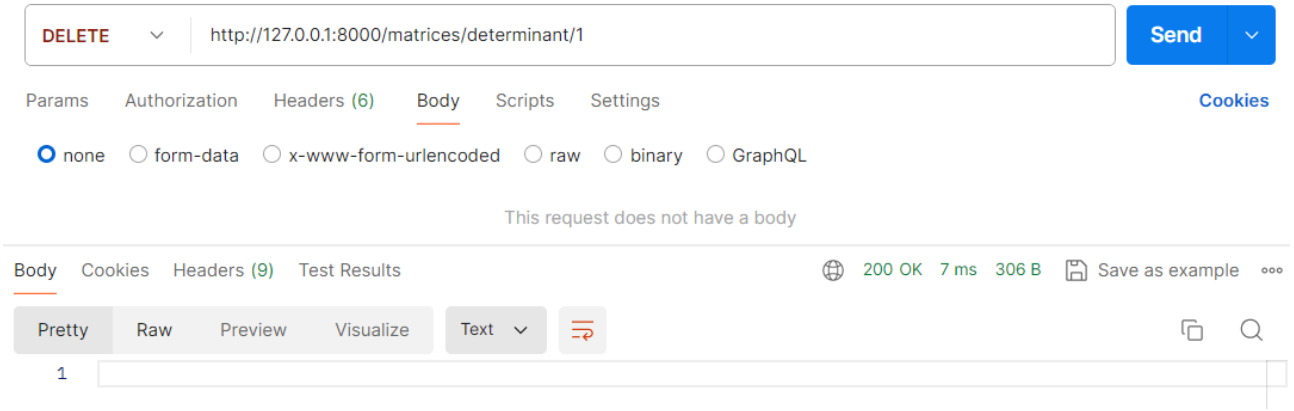

Рисунок 3.5 — Результат тест-кейса №4

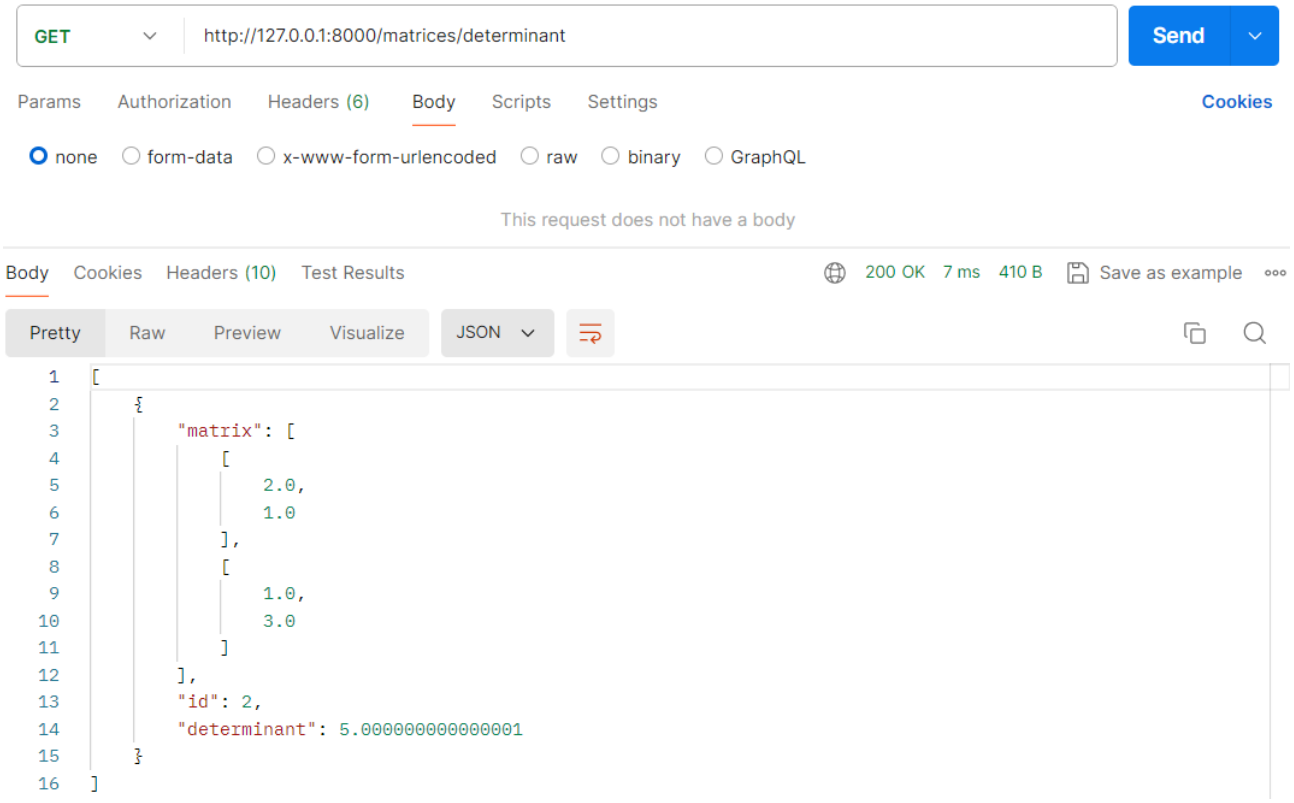

Рисунок 3.6 — Результат тест-кейса №4

Таблица 3.8 — Тест-кейс поиска несуществующей матрицы

Номер	4	
Название	Поиск несуществующей матрицы и её определителя	
Входные данные	Ожидаемый результат	Результат
5	404 — Матрица и определитель не найдены	404 — Пройден

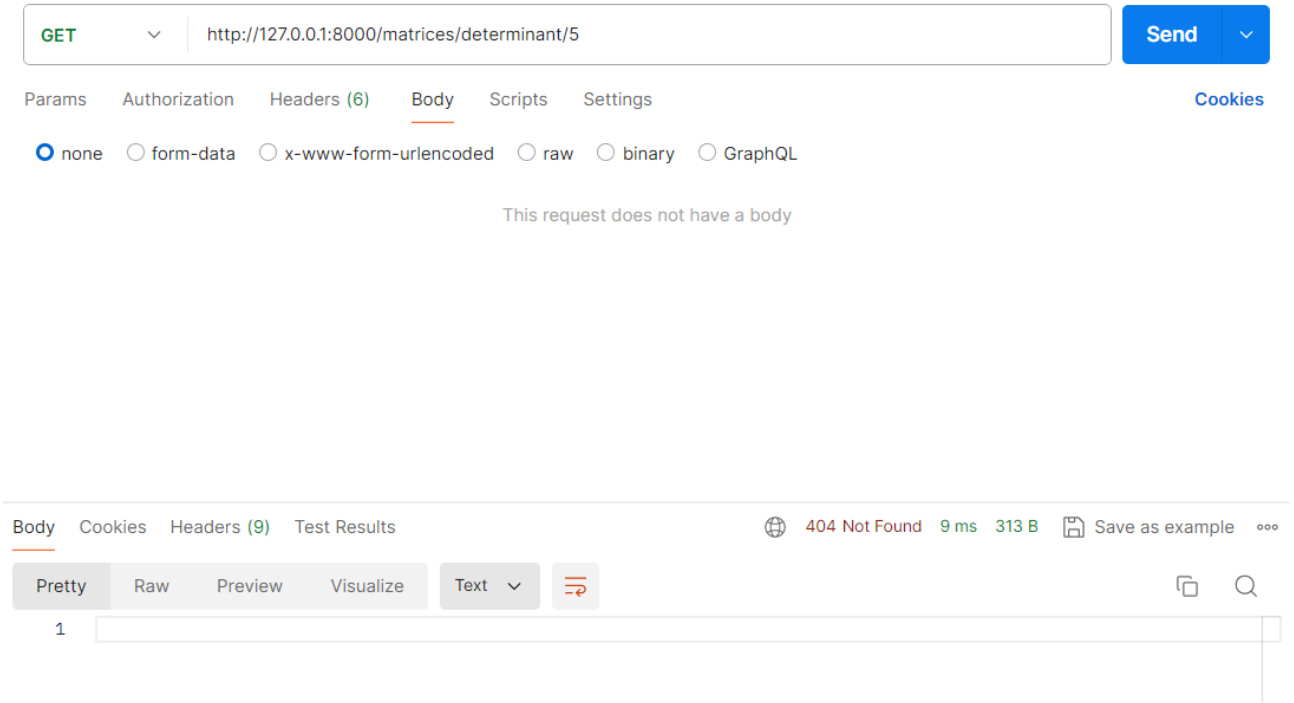

Рисунок 3.7 — Результат тест-кейса №5

Таблица 3.9 — Тест-кейс удаления несуществующей матрицы

Номер	4	
Название	Удаление несуществующей матрицы и её определителя	
Входные данные	Ожидаемый результат	Результат
5	404 — Матрица и определитель не найдены	404 — Пройден

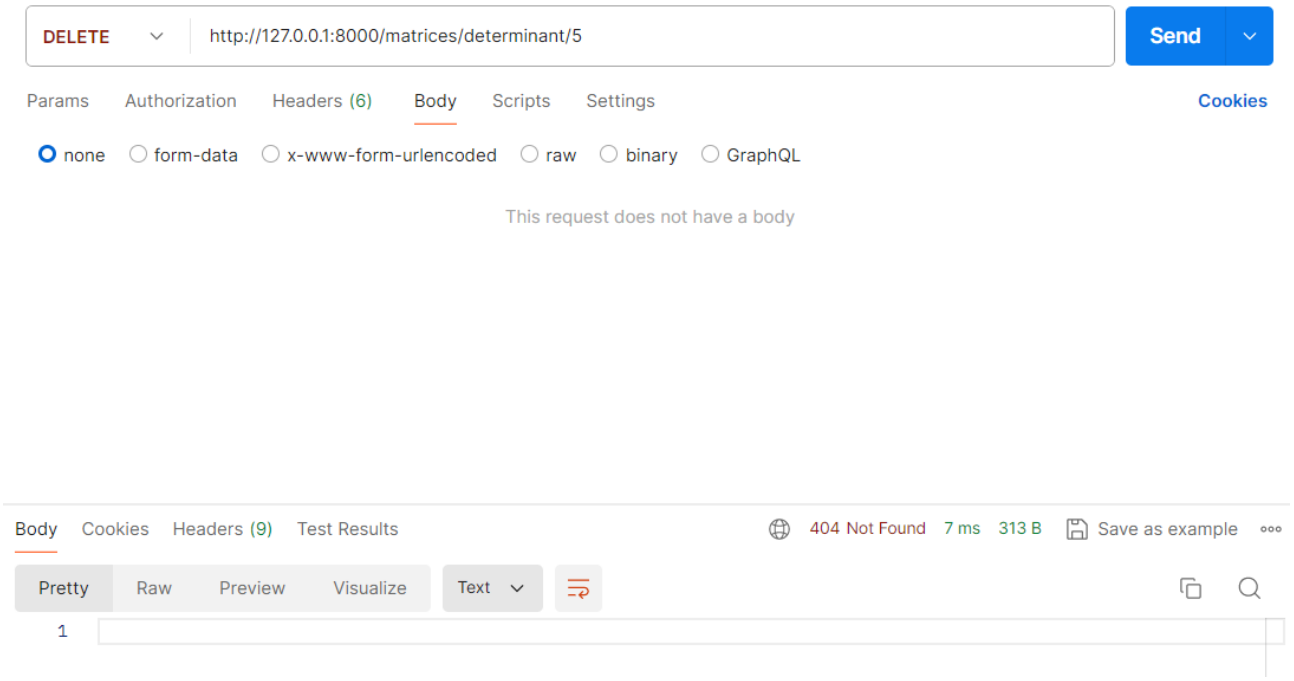

Рисунок 3.8 — Результат тест-кейса №6

Таблица 3.10 — Тест-кейс ввода невалидной матрицы

Номер	4	
Название	Ввод невалидной матрицы	
Входные данные	Ожидаемый результат	Результат

Матрица:
{
"matrix": [
[
5,
1,
9,
0
],
[
2,
4
]
]
}
	422 — Необрабатываемая сущность	422 — Пройден

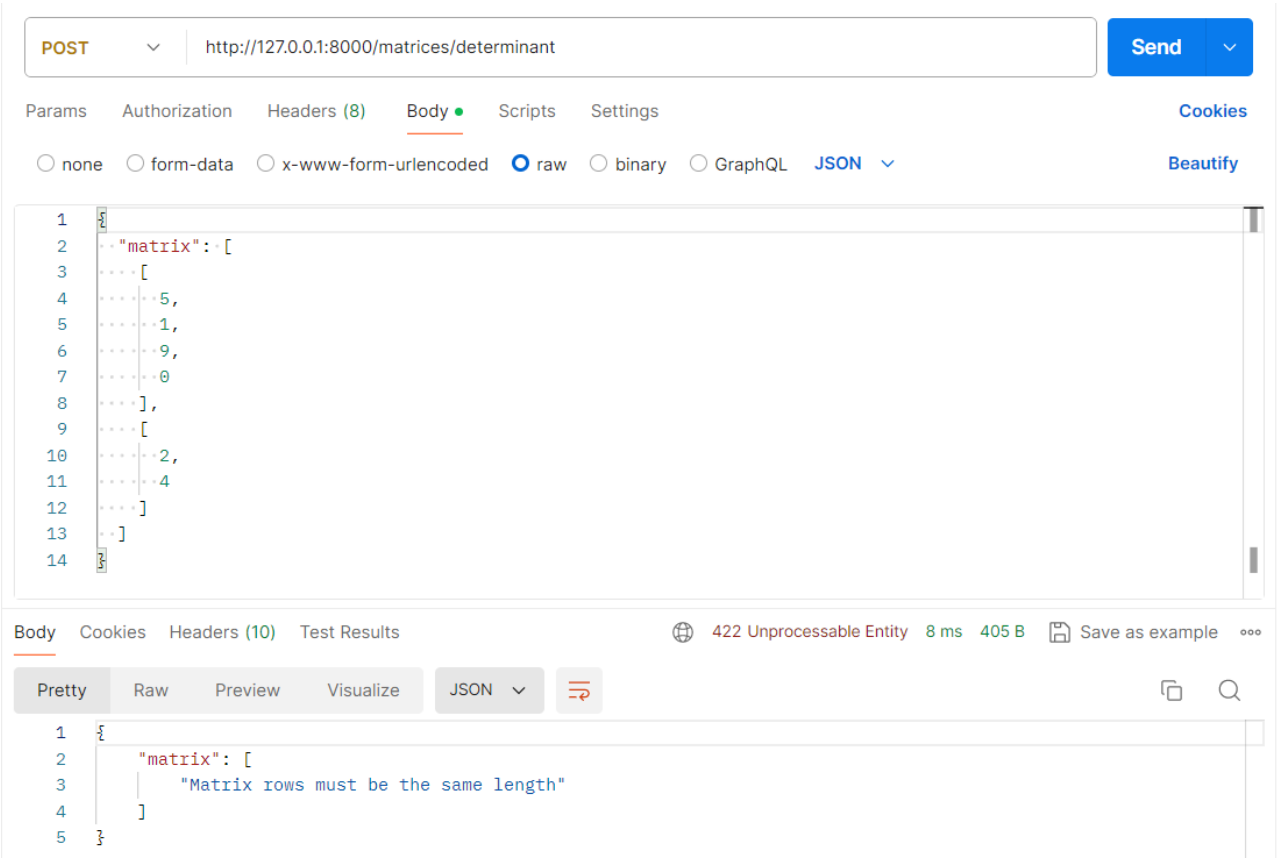

Рисунок 3.9 — Результат тест-кейса №7

4 РАЗВЕРТЫВАНИЕ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ «ПОИСК ОПРЕДЕЛИТЕЛЯ ИСХОДНОЙ МАТРИЦЫ»

4.1 Описание процесса развертывания

Для развертывания серверной части приложения необходимо два Docker-контейнера. Первый будет отвечать за прокси-сервер nginx, второй за серверную часть приложения.

Для работы прокси-сервера была написана конфигурация (Листинг 4.1). Сервер слушает «80» порт, все входящие запросы перенаправляются в Django приложение.

Листинг 4.1 — Конфигурация прокси-сервера nginx

```
resolver 127.0.0.11 ipv6=off valid=15s;
resolver_timeout 5s;

server {
    listen 80;

    location / {
        set $upstream_backend 'backend:8000';
        proxy_pass http://$upstream_backend;
    }

    location /media {
        alias @media;
    }
}
```

Далее была описана спецификация запуска прокси-сервера в виде Docker-контейнера (Листинг 4.2).

Листинг 4.2 — Спецификация запуска прокси-сервера в виде Docker-контейнера

```
  proxy:
    container_name: proxy
    image: nginx:1.25.5-alpine
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - backend
```

Продолжение Листинга 4.2

```
    ports:
      - "80:80"
```

Для запуска Django приложения используется веб-сервер gunicorn. Команда с параметрами для его запуска представлена в Листинге 4.3. Флаг "--workers" используется для указания количества рабочих процессов, флаг "-b" — для указания сокета сервера для привязки.

Листинг 4.3 — Команда запуска веб-сервера gunicorn

```
gunicorn --workers=1 -b=0.0.0.0:8000 kr_determinant.wsgi:application
```

Также была описана спецификация сборки серверной части приложения в виде Docker-контейнера (Листинг 4.4). Образ контейнера определяется на основе "python:3.11-alpine3.19". Затем устанавливаются переменные окружения, устанавливается рабочий каталог /app и копируется файл requirements.txt внутрь контейнера. Далее выполняются команды установки зависимостей, таких как библиотеки из requirements.txt, а также устанавливается gunicorn. Затем происходит копирование всех файлов из текущего каталога внутрь контейнера. Последующие команды запускают миграции базы данных Django (makemigrations и migrate) и запускают приложение с помощью gunicorn, чтобы обеспечить его работу на порту 8000 с 1 рабочим процессом.

Листинг 4.4 — Спецификация сборки серверной части приложения в виде Docker-контейнера

```
FROM python:3.11-alpine3.19

ENV PYTHONUNBUFFERED 1

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt
RUN pip install gunicorn

COPY . .
RUN python ./manage.py makemigrations
RUN python ./manage.py migrate
CMD gunicorn --workers=1 -b=0.0.0.0:8000 kr_determinant.wsgi:application
```

4.2 Тестирование развертывания

На Рисунке 4.1 представлен результат сборки образа серверной части приложения.

```
(venv) D:\KR_Petrova\KR_Petrova\kr_determinant>docker build -t back .

[+] Building 96.9s (14/14) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 359B
 => [internal] load metadata for docker.io/library/python:3.11-alpine3.19
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 167.35kB
 => [1/8] FROM docker.io/library/python:3.11-alpine3.19@sha256:0b5ed25d3cc27cd35c7b0352bac8ef2ebc8dd3da72a0c03caaf4eb15d9ec827a
 => => resolve docker.io/library/python:3.11-alpine3.19@sha256:0b5ed25d3cc27cd35c7b0352bac8ef2ebc8dd3da72a0c03caaf4eb15d9ec827a
 => => sha256:3912f7fe31112ee0f747848328e1a2b225a3aad18d0800bac6e13042642fd202 1.37kB / 1.37kB
 => => sha256:76351c33299b900aa86b33176eac198fc861d4a7978046db4ae8b319e148088a 3.13MB / 3.13MB
 => => extracting sha256:c3cdf40b8bda8e4ca4be0f5fa7f1d128907271efcbc72cbfc7c8b0f939ec25ea
 => => extracting sha256:ac499ccf2147611bc4388058b362c0bcc1ca63ec1a320a2f4ed5c0a9a76d08ea
 => => extracting sha256:416bfceb623eb12bf1c373489e0dba32f00fd4fef037b369538d190c615d49cd
 => => extracting sha256:76351c33299b900aa86b33176eac198fc861d4a7978046db4ae8b319e148088a
 => [2/8] WORKDIR /app
 => [3/8] COPY requirements.txt .
 => [4/8] RUN pip install -r requirements.txt
 => [5/8] RUN pip install gunicorn
 => [6/8] COPY . .
 => [7/8] RUN python ./manage.py makemigrations
 => [8/8] RUN python ./manage.py migrate
 => exporting to image
 => => exporting layers
 => => writing image sha256:d8451142899ee9fd3526d2a5e6ef700738ddec774672eb7ac149d2f822616a4a
 => => naming to docker.io/library/back
```

Рисунок 4.1 — Результат сборки образа серверной части

Для запуска контейнера необходимо воспользоваться командой "docker run". В результате получен идентификатор запущенного контейнера (Рисунок 4.2).

```
(venv) D:\KR_Petrova\KR_Petrova\kr_determinant>docker run --name app -d back
dc1cde73baa90e5a029ab2482aec98d09355b7f4498f574767de6c54eaf69b4b
```

Рисунок 4.2 — Результат запуска контейнера

Для проверки статуса контейнера необходимо воспользоваться командой "docker ps" с фильтрацией по названию контейнера (Рисунок 4.3).

```
(venv) D:\KR_Petrova\KR_Petrova\kr_determinant>docker ps -f "name=app"
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
dc1cde73baa9   back      "/bin/sh -c 'gunicor…"   14 minutes ago   Up 14 minutes             app
```

Рисунок 4.3 — Проверка статуса контейнера

На Рисунке 4.4 представлен результат сборки образа прокси-сервера.

```
proxy       | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
proxy       | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
proxy       | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
proxy       | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
proxy       | 10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from the packaged version
proxy       | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
proxy       | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
proxy       | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
proxy       | /docker-entrypoint.sh: Configuration complete; ready for start up
proxy       | 2024/05/24 00:21:51 [notice] 1#1: using the "epoll" event method
proxy       | 2024/05/24 00:21:51 [notice] 1#1: nginx/1.25.5
proxy       | 2024/05/24 00:21:51 [notice] 1#1: built by gcc 13.2.1 20231014 (Alpine 13.2.1_git20231014)
proxy       | 2024/05/24 00:21:51 [notice] 1#1: OS: Linux 5.15.146.1-microsoft-standard-WSL2
proxy       | 2024/05/24 00:21:51 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
proxy       | 2024/05/24 00:21:51 [notice] 1#1: start worker processes
proxy       | 2024/05/24 00:21:51 [notice] 1#1: start worker process 29
proxy       | 2024/05/24 00:21:51 [notice] 1#1: start worker process 30
proxy       | 2024/05/24 00:21:51 [notice] 1#1: start worker process 31
proxy       | 2024/05/24 00:21:51 [notice] 1#1: start worker process 32
```

Рисунок 4.4 — Результат сборки образа прокси-сервера

5 РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ «ПОИСК ОПРЕДЕЛИТЕЛЯ ИСХОДНОЙ МАТРИЦЫ»

5.1 Технологический стек

Для реализации клиентской части приложения были использованы следующие библиотеки:

1. React. Основные принципы React включают виртуальный DOM (Document Object Model) для оптимизации производительности, однонаправленный поток данных и принцип "одного источника правды" (Single Source of Truth).
2. Axios. Библиотека предоставляет простой и удобный способ работать с HTTP запросами на основе промисов. Axios поддерживает все основные методы запросов, такие как GET, POST, PUT, DELETE, а также обработку запросов с использованием различных параметров, заголовков и т.д. Библиотека Axios также обладает встроенной возможностью для преобразования данных в различные форматы, такие как JSON, формы, строки и другие [10].

5.2 Описание реализации

Клиентская часть представляет собой Single Page Application. Она позволяет задать размерность квадратной матрицы, заполнить её поэлементно, найти определитель матрицы, получить список всех введённых матриц и найденных определителей, найти матрицу и определитель по ID и удалить матрицу и определитель по ID. Шаблоном выступает файл index.html (Листинг 5.1).

Листинг 5.1 — Содержимое файла index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

После сборки проекта при помощи webpack к документу добавляется скрипт, который отрисовывает все элементы, используя React.

5.3 Описание процесса взаимодействия конечного пользователя с клиентом

На Рисунке 5.1 представлен интерфейс клиентской части приложения. Имеется возможность найти определитель введенной матрицы (Рисунок 5.2), получить список всех матриц и их определителей (Рисунок 5.3), получить матрицу и ее определитель по ID (Рисунок 5.4), удалить матрицу и ее определитель по ID (Рисунок 5.5).

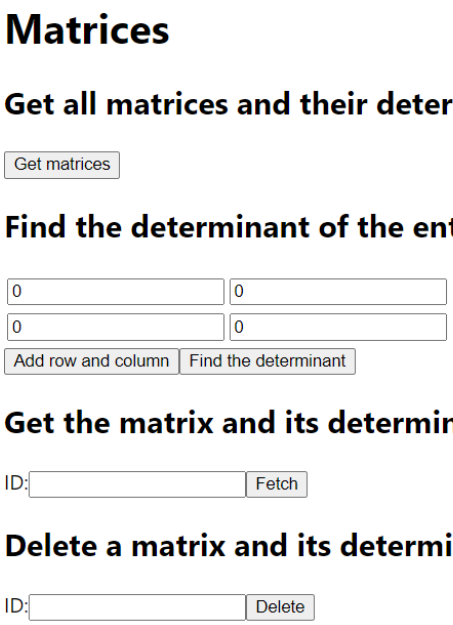

Рисунок 5.1 — Интерфейс для роли «Пользователь»

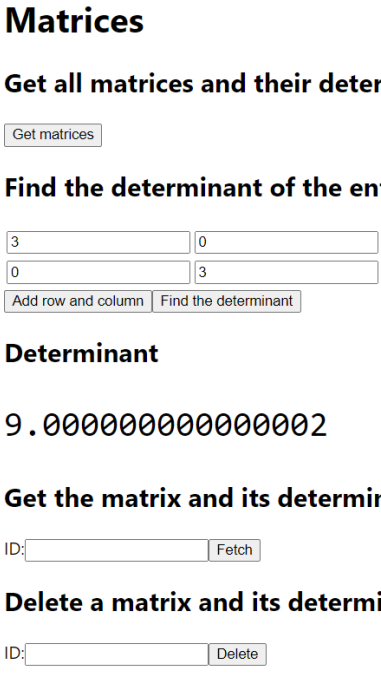

Рисунок 5.2 — Использование интерфейса для поиска определителя введенной матрицы

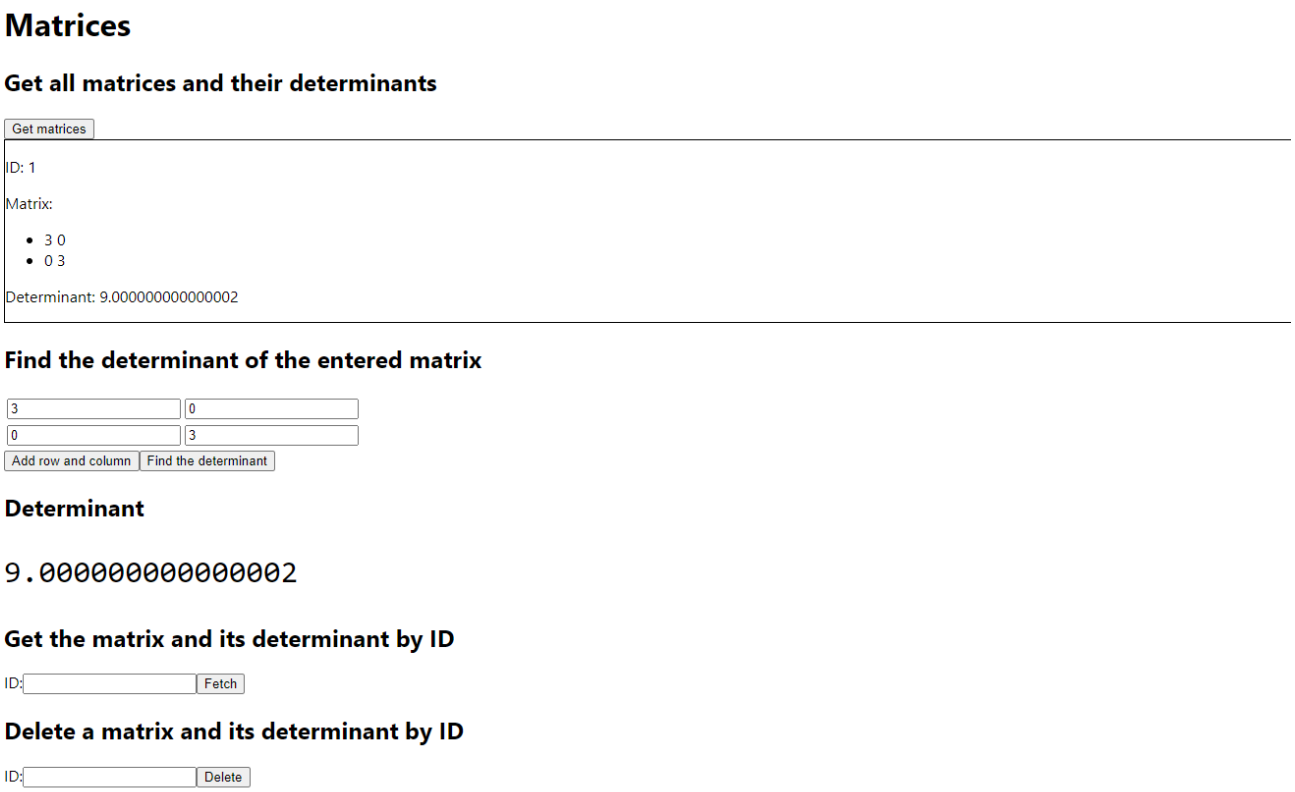

Рисунок 5.3 — Использование интерфейса для получения списка всех матриц и их определителей

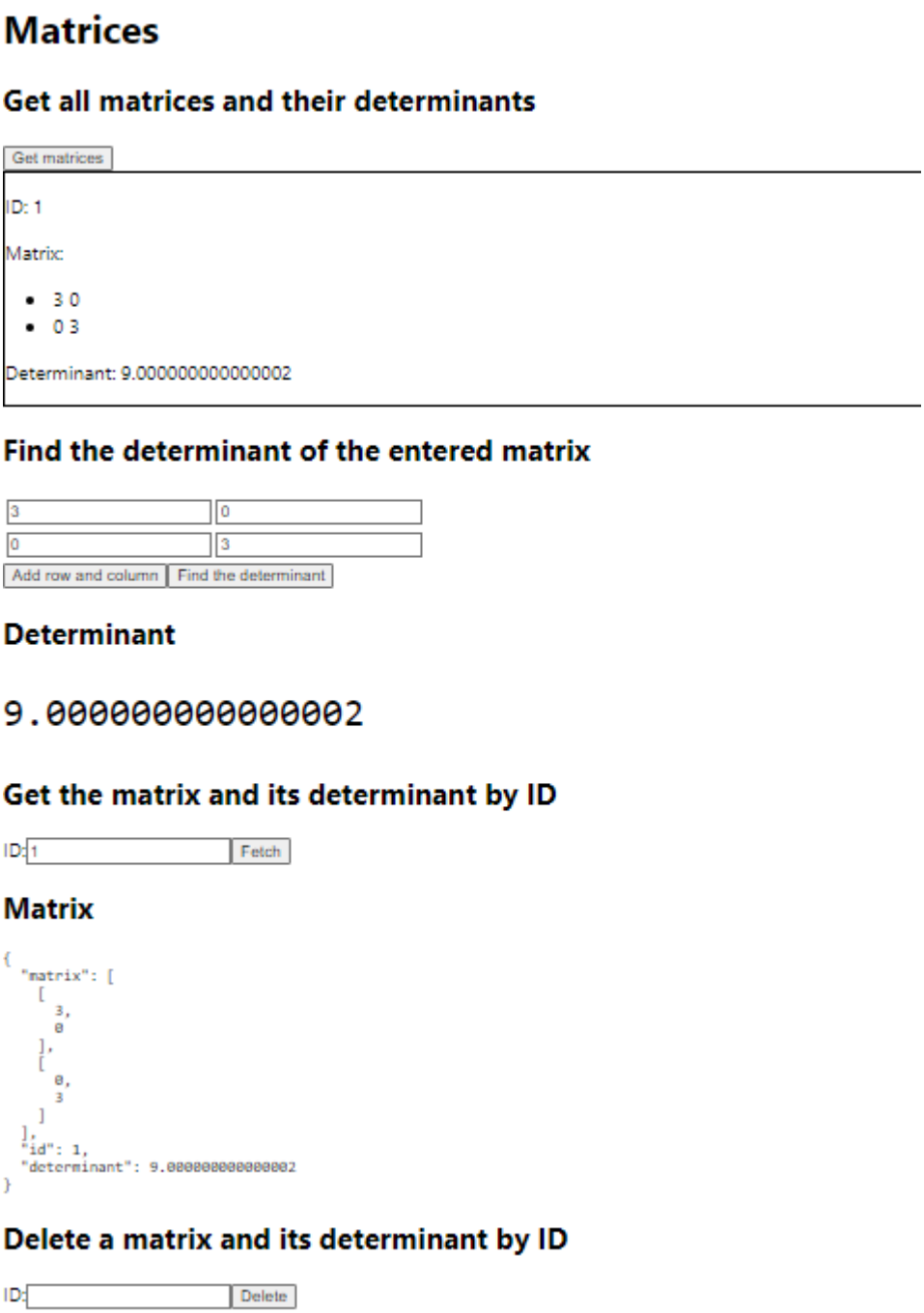

Рисунок 5.4 — Использование интерфейса для получения матрицы и ее определителя по ID

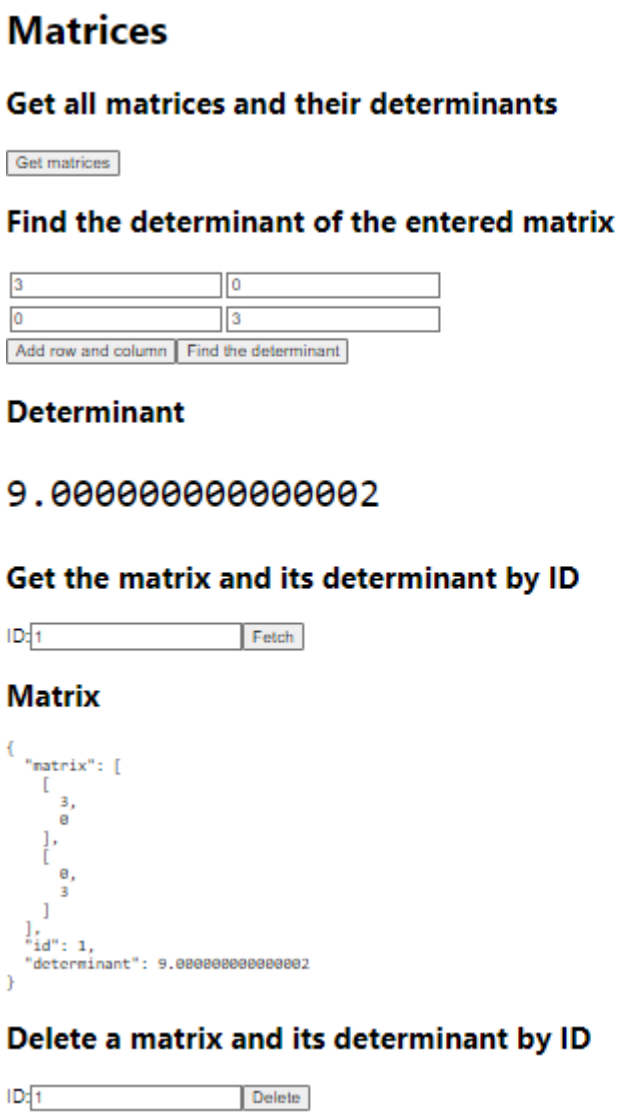

Рисунок 5.5 — Использование интерфейса для удаления матрицы и ее определителя по ID

User Story (Таблица 5.1), или пользовательская история, помогает увидеть функции продукта глазами конечного пользователя.

Таблица 5.1 — User Story

Кто?	Что хочет?	С какой целью?
Пользователь	Получить определитель матрицы	Просмотреть определитель введённой матрицы
Пользователь	Найти матрицу и её определитель по ID	Просмотреть конкретную матрицу и её определитель
Пользователь	Удалить матрицу и её определитель по ID	Удалить конкретную матрицу и её определитель
Пользователь	Получить список всех матриц и их определителей	Просмотреть все введённые матрицы и их найденные определители

6 РАЗВЕРТЫВАНИЕ КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ «ПОИСК ОПРЕДЕЛИТЕЛЯ ИСХОДНОЙ МАТРИЦЫ»

6.1 Описание процесса развертывания

Для развертывания клиентской части приложения была описана спецификация сборки клиентской части приложения в виде Docker-контейнера (Листинг 6.1).

Листинг 6.1 — Спецификация сборки клиентской части приложения в виде Docker-контейнера

```
FROM node:18-alpine AS BUILD
WORKDIR /usr/src/client
COPY package-lock.json ./
COPY package.json ./
RUN npm ci
COPY . ./
RUN npm run build

FROM nginx:stable-alpine-slim
COPY --from=BUILD /usr/src/client/build /usr/share/nginx/html
CMD [ "nginx", "-g", "daemon off;"]
```

Сборка клиентской части приложения состоит из двух этапов:

1. Сборка бандла.
2. Настройка nginx-сервера для хостинга.

На первом этапе в качестве базового образа используется "node:18-alpine". Сначала устанавливаются зависимости приложения, затем при помощи скрипта "build" собирается и минимизируется исходный код приложения.

На втором этапе в качестве базового образа используется "nginx:stable-alpine-slim". Собранный на первом этапе бандл копируется в папку со статическими файлами веб-сервера nginx и затем он запускается.

6.2 Тестирование развертывания

Результат сборки образа клиентской части с написанной ранее конфигурацией представлен на Рисунке 6.1.

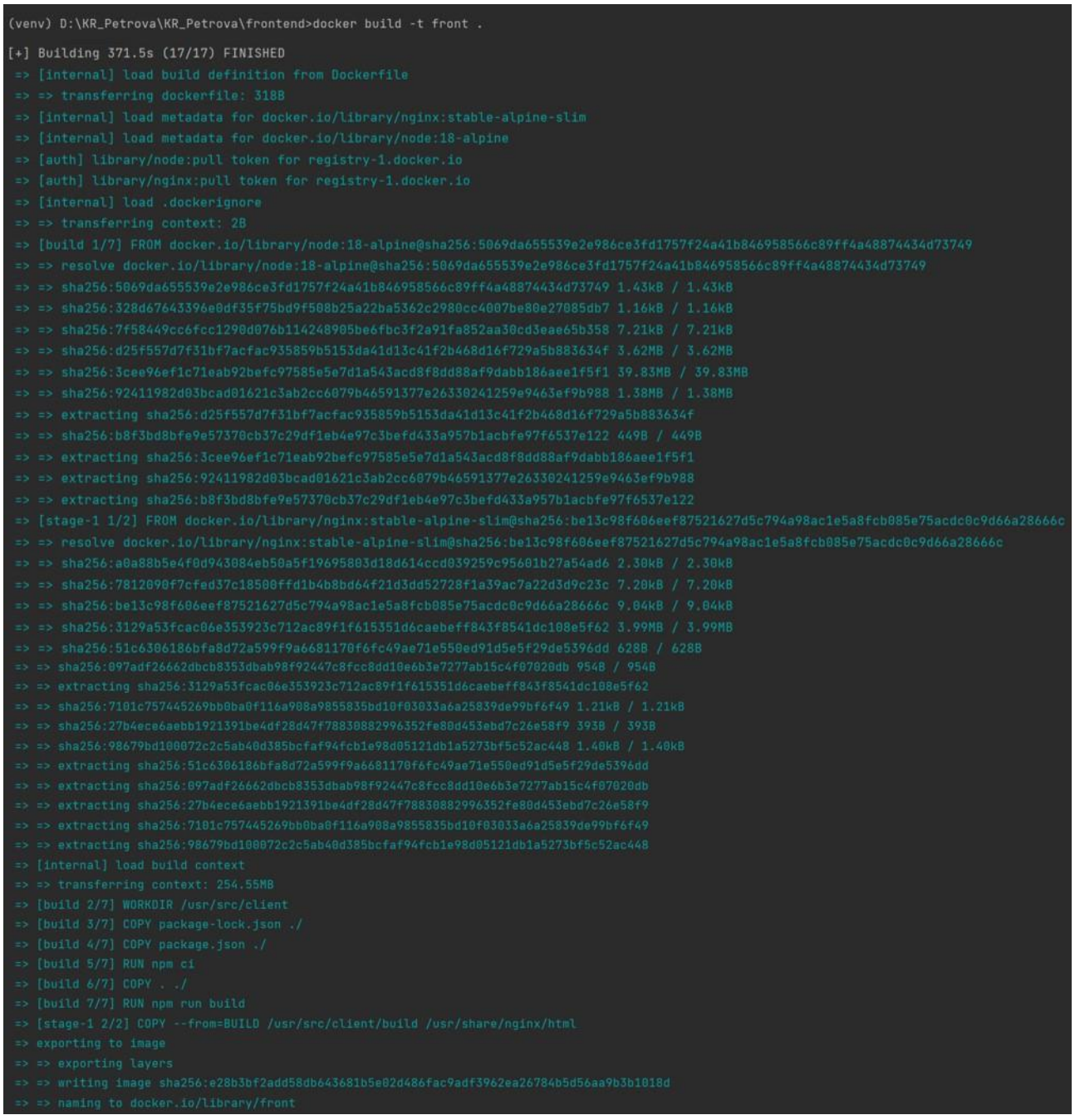

Рисунок 6.1 — Результат сборки образа клиентской части

Для запуска контейнера необходимо воспользоваться командой "docker run". В результате получен идентификатор запущенного контейнера (Рисунок 6.2).

```
(venv) D:\KR_Petrova\KR_Petrova\frontend>docker run --name client -d front
438cc7234753672026a614c76e56dc8d52b82bd3ea454ee76f71faa99ebabe95
```

Рисунок 6.2 — Результат запуска контейнера

Для проверки статуса контейнера необходимо воспользоваться командой "docker ps" с фильтрацией по названию контейнера (Рисунок 6.3).

```
(venv) D:\KR_Petrova\KR_Petrova\frontend>docker ps -f "name=client"
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
438cc7234753   front     "/docker-entrypoint.…"   2 minutes ago   Up 2 minutes   80/tcp    client
```

Рисунок 6.3 — Проверка статуса контейнера

7 НАСТРОЙКА КОНФИГУРАЦИИ МНОГОКОНТЕЙНЕРНОГО РАЗВЕРТЫВАНИЯ ПРИЛОЖЕНИЯ «ПОИСК ОПРЕДЕЛИТЕЛЯ ИСХОДНОЙ МАТРИЦЫ»

Для запуска многоконтейнерного приложения используется технология Docker Compose. В Листинге 7.1 представлена разработанная спецификация.

Листинг 7.1 — Спецификация запуска многоконтейнерного приложения

```
version: "3.9"

services:
  backend:
    container_name: django_app
    build: ./kr_determinant
  proxy:
    container_name: proxy
    image: nginx:1.25.5-alpine
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - backend
    ports:
      - "80:80"
  front:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "3000:80"
    depends_on:
      - backend
      - proxy
```

Сервис "backend" отвечает за gunicorn сервер и Django приложение. Контейнер запускается с именем "django_app", Контекстом является директория "./kr_determinant", где находится Dockerfile для сборки образа.

Сервис "proxy" отвечает за прокси-сервер nginx, который перенаправляет запросы в Django приложение. Контейнер запускается с названием "proxy". Базовый образ nginx и его версия — "nginx:1.25.5-alpine". Файл nginx.conf монтируется из текущей директории в контейнер NGINX для настройки конфигурации. Этот контейнер зависит от сервиса "backend". Внутренний порт 80 сопоставляется с 80 портом хоста.

Сервис "front" отвечает за веб-сервер nginx, который раздает статические файлы клиентам. Контекстом является директория "./frontend", в которой находится код клиентской части приложения. Также build задает путь к Dockerfile для клиентской части приложения. Внутренний порт 80 сопоставляется с 3000 портом хоста. Этот сервис зависит от сервисов "backend" и "proxy".

ЗАКЛЮЧЕНИЕ

Выполнение данной курсовой работы позволило углубить знания в области программирования на языке Python и использования библиотеки "NumPy". Применение современных технологических решений и подходов способствовало созданию эффективного и масштабируемого приложения, что подтверждает актуальность выбранной темы.

В заключение, хочется отметить, что в ходе выполнения курсовой работы были успешно реализованы все поставленные задачи. С использованием Django и Django REST Framework было разработано полноценное приложение, способное эффективно выполнять поиск определителя исходной матрицы. Была также обеспечена поддержка принципов междоменного обмена ресурсами, что позволяет API обслуживать запросы с различных источников.

Использование API "NumPy" позволило значительно упростить процесс работы с матрицами, осуществить вычисление определителя исходной матрицы с помощью математических функций, предоставляемых библиотекой.

Процесс развёртывания приложения был успешно реализован с использованием современных инструментов и технологий, таких как Docker и Docker-Compose. Разрабатываемое API было выполнено в соответствии со всеми техническими требованиями.

Следовательно, можно заключить, что выбор Django для разработки API оказался обоснованным и результативным, что подтверждается успешным выполнением цели, поставленной в рамках курсовой работы.