Backend

Файл Views.py

Модель, описывающая формулу в базе данных

```
class Formula(models.Model):
    formula = models.CharField(
        verbose_name='Формула',
    )
    source = models.CharField(
        verbose_name='Источник',
        max_length=1024,
        null=True,
        blank=True,
        default=None
    )
    time_create = models.DateTimeField(
        verbose_name='Время добавления',
        auto_now_add=True
    )
    time_update = models.DateTimeField(
        verbose_name='Время изменения',
        auto_now=True
     )
```

Включает в себя поля для: тела формулы, источник, откуда взята формула, время добавления и изменения формулы.

Обработка формулы

```
class FormulaAnalysisView(APIView):
    def post(self, request):
        serializer = FormulaSerializer(data=request.data)
```

```
        serializer.is_valid(raise_exception=True)
        match = find_coincidence(serializer.data['formula'])
        return Response(match)
```

Реализует получение формулы из фронтенда, сравнивает с имеющимися формулами в базе данных.

Обработка GET запросов

```
    def get(self, request):
        form = FileUploadForm()
        return render(request, self.template_name, {'form': form})
```

Обработка Post запросов

```
        def post(self, request):
            form = FileUploadForm(request.POST, request.FILES)
            if form.is_valid():
                uploaded_file = form.cleaned_data['file']

                extension = os.path.splitext(uploaded_file.name)[1]
```

При POST-запросе, когда пользователь пытается загрузить файл: - Метод post создает экземпляр FileUploadForm с данными запроса (request.POST) и самим файлом (request.FILES). - Проверяет, валидна ли форма с помощью метода is_valid(). Если форма валидна, извлекает загруженный файл из очищенных данных формы. - Определяет расширение файла и устанавливает временную директорию для его сохранения. - Генерирует уникальный идентификатор (UUID) для имени временного файла, чтобы избежать конфликтов имен файлов. - Сохраняет загруженный файл в временную директорию.

Обработка файлов и добавление в базу данных

```
 # Путь для сохранения файла
            temp_dir = os.path.join(settings.MEDIA_ROOT, 'temp')
            os.makedirs(temp_dir, exist_ok=True)  # Создаём папку,
если её нет

            unique_id = uuid.uuid4().hex

            # Полный путь сохранённого файла
            file_path = os.path.join(temp_dir,
f'{unique_id}{extension}')
            tex_path = os.path.join(temp_dir, f'{unique_id}.tex')

            # Сохранение файла
            with open(file_path, 'wb+') as destination:
                for chunk in uploaded_file.chunks():
                    destination.write(chunk)

            # Обработка файла
            try:
                if extension == '.docx':
                    convert_docx_to_tex(file_path, tex_path)
                    math_expressions =
extract_math_from_tex(tex_path)
                    # Создаём объекты Formula для каждой формулы
                    formula_objects = [Formula(formula=f.strip('\n
')) for f in math_expressions if f.strip('\n ')]
                else:
                    formula_objects = []
                    with open(file_path, 'rt') as txt_file:
                        formula_objects =
[Formula(formula=f.strip('\n ')[3:-3]) for f in txt_file if
f.strip('\n ')]

                # Добавляем все объекты в базу данных за один запрос
                Formula.objects.bulk_create(formula_objects)

            finally:
                # Удаляем файл после обработки
                if os.path.exists(file_path):
                    os.remove(file_path)
                if os.path.exists(tex_path):
                    os.remove(tex_path)

        return render(request, self.template_name, {'form': form})
```

После сохранения файла, код обрабатывает его в зависимости от расширения: - Если файл имеет расширение .docx, вызывается функция convert_docx_to_tex для

конвертации этого файла в формат LaTeX (.tex), потом извлекаются математические выражения с помощью extract_math_from_tex. На основе извлеченных выражений создаются объекты Formula, представляющие собой математические формулы. - Если файл не .docx, предполагается, что он уже в формате LaTeX или другом текстовом формате с математическими выражениями, и прямо из него читаются строки, соответствующие формулам. Каждая строка (формула) оборачивается в объект Formula.

В обоих случаях созданные объекты Formula добавляются в базу данных пакетной операцией bulk_create.

Файл parser_word.py

Конвертация .docx в .tex

```
def convert_docx_to_tex(docx_path, tex_path):
    try:
        subprocess.run(['convert/pandoc.exe', '-s', docx_path,
'-o', tex_path], check=True)
        print(f"Файл {docx_path} успешно конвертирован в
{tex_path}.")
    except subprocess.CalledProcessError as e:
        print(f"Ошибка при конвертации файла: {e}")
```

Функция convert_docx_to_tex использует pandoc (внешний инструмент командной строки, указанный как convert/pandoc.exe) для конвертации файла .docx в .tex. В случае успеха выводит сообщение об успешной конвертации, в случае ошибки выводит сообщение об ошибке.

Преобразование юникода в LaTeX

```
def unicode_to_latex(text):
    replacements = {
        'Σ': r'\sum', 'ψ': r'\psi', 'α': r'\alpha', 'β':
r'\beta', 'γ': r'\gamma',
        'δ': r'\delta', 'ε': r'\epsilon', 'φ': r'\phi', 'ϕ':
r'\varphi',
    }
    for unicode_symbol, latex_command in replacements.items():
        text = text.replace(unicode_symbol, latex_command)
    return text
```

unicode_to_latex принимает строку text и заменяет вхождения определенных юникод-символов (например, греческих букв) на соответствующие им

команды LaTeX. Это удобно для преобразования математических символов, полученных из различных источников, в формат, пригодный для LaTeX.

Преобразование текста в LaTeX

```
def convert_text_to_latex(text):
    replacements = {
        '\{': '{', '\}': '}'
    }
    for old, new in replacements.items():
        text = re.sub(re.escape(old), new, text)
    return text
```

Функция convert_text_to_latex преобразует конкретные последовательности символов, которые могут быть интерпретированы некорректно при компиляции LaTeX документа, в их "безопасные" эквиваленты. Примером такого преобразования является замена управляющих символов LaTeX на их текстовые представления.

Извлечение математических формул из файла

```
def extract_math_from_tex(tex_path):
    """Извлекает математические выражения из файла LaTeX и
преобразует их в LaTeX формат."""
    with open(tex_path, 'r', encoding='utf-8') as file:
        tex_content = file.read()

    math_expressions = []

    # Ищем выражения в стандартных математических блоках LaTeX
    math_expressions +=
re.findall(r'\\begin\{(equation|align|multline)\}.*?\\end\{\1\
}', tex_content, re.DOTALL)

    # Ищем inline-выражения LaTeX
    math_expressions += re.findall(r'\$.*?\$', tex_content,
re.DOTALL)

    # Ищем выражения в display-style
    math_expressions += re.findall(r'\\\[.*?\\\]',
tex_content, re.DOTALL)
```

```
    # Фильтрация для чистки выражений от нежелательных
разрывов строк
    processed_expressions = [expr.replace('\n', '') for expr
in math_expressions if
                             any(char in expr for char in
'\\${}')]

    return processed_expressions
```

extract_math_from_tex извлекает математические выражения из файла LaTeX, находя выражения, заключенные в специфические окружения equation, align, multline, а также инлайн ($...$) и дисплейные (\[...\]) форматы математических выражений. Возвращается список всех найденных выражений, предварительно очищенных от разрывов строк.

Файл utils.py

Поиск совпадающих блоков

```
    def get_matching_indexes(str1, str2):

        matcher = SequenceMatcher(None, str1, str2)

        return matcher.find_longest_match(0, len(str1), 0,
    len(str2))
```

Определяет функцию get_matching_indexes, которая использует SequenceMatcher для нахождения самого длинного совпадающего блока между двумя строками (str1 и str2). Возвращает объект Match, содержащий индексы начала и длину совпадающего участка.

```
def find_coincidence(formula):
    db_formulas = Formula.objects.values_list('formula',
flat=True)
    best_match = process.extractOne(formula, db_formulas,
scorer=fuzz.partial_ratio)

    match = get_matching_indexes(best_match[0], formula)

    if match.size == 0:
```

```
        return {'formula': formula, 'coincidence': 0}

    formula = best_match[0]

    matched_formula = (
            formula[:match.a] +
            '\\textcolor{red}{' +
            formula[match.a:match.a + match.size] +
            '}' +
            formula[match.a + match.size:]
    )

    return {'formula': matched_formula, 'coincidence':
best_match[1]}
```

Функция find_coincidence осуществляет поиск заданной математической формулы среди формул, хранящихся в базе данных, с целью выявления наиболее похожей. Она извлекает все формулы в базу и применяет инструмент fuzzywuzzy для нахождения частичного совпадения, используя partial_ratio для оценки. Затем с помощью difflib.SequenceMatcher ищет наибольший общий участок между найденной формулой и исходной, выделяя его красным цветом в LaTeX для визуализации совпадения. Функция возвращает формулу с выделенным совпадением и процентом совпадения. Если совпадений нет, возвращает исходную формулу с нулевым процентом.

Frontend

Файл FormulaEditor.js

Отображение LaTeX формул

```
      <MathJaxContext>

            <div className="h-full w-full">

              <h2>Редактор формул</h2>

              <textarea

                 className="w-full h-full p-4 bg-transparent
      border border-gray-300 rounded resize-none focus:outline-
      none"

                 value={localLatex}
```

```
                ref = {textAreaRef}
                onChange={handleInputChange}
                placeholder="Введите формулу LaTeX"
                style={{
                  fontFamily: "monospace",
                }}
              />
            </div>
          </MathJaxContext>
```

Используется для оборачивания JSX кода компонента, что позволяет поддерживать отображение и редактирование LaTeX формул внутри компонента.

Изменение текста в редакторе.

```
    const handleInputChange = (event) => {
      const newLatex = event.target.value;
      setLocalLatex(newLatex);
      onFormulaChange(newLatex);
    };
```

Эта функция вызывается каждый раз, когда значение в текстовом поле изменяется. Она устанавливает новое значение в состояние "localLatex" и вызывает функцию обратного вызова "onFormulaChange", чтобы сообщить родительскому компоненту об этом изменении.

Вставка текста

```
      const insertAtCursor = (insertText) => {
        const textarea = textAreaRef.current;
        if (!textarea)  return;

        const cursorStart = textarea.selectionStart;
        const cursorEnd = textarea.selectionEnd;
```

```
        const currentValue = textarea.value;


        const beforeCursor = currentValue.substring(0,
    cursorStart);
        const afterCursor =
    currentValue.substring(cursorEnd);


        const updatedValue
    =`${beforeCursor}${insertText}${afterCursor}`;


        setLocalLatex(updatedValue);
        onFormulaChange(updatedValue);
        textarea.focus();
        textarea.setSelectionRange(
            cursorStart + insertText.length,
            cursorEnd + insertText.length
        );
      };
```

"insertAtCursor" - это функция, которая вставляет текст в текущей позиции курсора в текстовом поле, используя ссылку на DOM-элемент

Файл formulaRedner.js

Преобразование LaTeX в JSON

```
    const handleSubmit = () => {


        const payload = JSON.stringify({ formula: latex });
        console.log('это отправляем на сервер:', payload);
        fetch('/api/analysis/', {
         method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: payload,
```

```
        })
        .then(response => response.json())
        .then(data => {
          console.log(data);
        })
        .catch(error => {
          console.error('Error:', error);
        });
      };
```

Эта функция handleSubmit сериализует LaTeX строку в JSON объект и отправляет его на серверный эндпоинт /api/analysis/ с использованием HTTP метода POST. В процессе обработки ответа сервера она выводит результат или ошибку в консоль.

Рендеринг формулы

```
    return (
      <MathJaxContext>
        <div>
          <h3>Рендеринг формулы:</h3>
          <MathJax className="border rounded min-h-
    [40%]">{`${latex}`}</MathJax>
        </div>
        <div>
          <button className=""
    onClick={handleSubmit}>Сохранить формулу</button>
        </div>
      </MathJaxContext>
    );
```

Возвращает JSX разметку, которая содержит контекст MathJaxContext для обеспечения функциональности рендеринга LaTeX. Внутри контекста: - Отображается заголовок, - Используется компонент MathJax для рендеринга переданной формулы LaTeX, - Кнопка, при нажатии на которую вызывается функция handleSubmit для отправки формулы на сервер.

Файл formulaTollBox.js

Набор элементов

```
const categories = [
  {
      label: "\\text{Математические знаки}",
      elements: [
          // Массив объектов
      ],
  },
  {
      label: "\\text{Математическая логика}",
      elements: [
          // Массив объектов
      ],
  },
  {
      label: "\\text{Знаки сравнения}",
      elements: [
          // Массив объектов
      ],
  },
];
```

Задает набор категорий и элементов в них, где каждый элемент имеет метку (label) для отображения и код (code), который можно вставить в LaTeX редактор. Это статические данные для представления различных математических символов и операций, которые пользователи могут вставлять.

Файл layout.js

Локальные шрифты

```
const geistSans = localFont({
```

```
  src: "./fonts/GeistVF.woff",

  variable: "--font-geist-sans",

  weight: "100 900",

});

const geistMono = localFont({

  src: "./fonts/GeistMonoVF.woff",

  variable: "--font-geist-mono",

  weight: "100 900",

});
```

Определяет локальный шрифт Geist Sans и geistMono с переменной CSS для легкой интеграции в стили компонента. Шрифт подгружается из локального файла, указан следующий диапазон веса шрифта: от 100 до 900.

Корневой контейнер

```
export default function RootLayout({ children }) {

  return (

    <html lang="en">

      <body                    className={`${geistSans.variable}
${geistMono.variable} antialiased`}>



          {children}

      </body>

    </html>

  );

}
```

Это React компонент RootLayout, который служит корневым контейнером для всего содержимого приложения. Он принимает дочерние компоненты в качестве children и рендерит их внутри контейнера <body>. Классы, применяемые к <body>, включают переменные для шрифтов geistSans и geistMono, а также добавляют антиалиасинг (antialiased) для улучшения отображения текста.

Файл page.js

Импорт хуков

```
"use client";

import { useState, useRef } from "react";

import FormulaEditor from './formulaEditor';

import FormulaRenderer from './formulaRenderer';

import FormulaToolBox from "./formulaToolBox";

import Image from 'next/image';
```

Этот блок импортирует необходимые хуки из React (useState, useRef), компоненты для редактирования (FormulaEditor), отображения (FormulaRenderer) LaTeX формул и инструментарий (FormulaToolBox). Также используется компонент Image из Next.js для оптимальной работы с изображениями.

Компонент home

```
export default function Home() {

  const [latex, setLatex] = useState("");

  const editorRef = useRef(null);
```

Определение компонента Home. Здесь объявляется состояние latex для хранения текущей LaTeX строки. Также создается реф editorRef для доступа к DOM элементу редактора формул.

Компоненты редактирования и отображения формул

```
<main className="flex-1 p-4 grid grid-cols-2 gap-4">

  <section className="p-4 rounded font-[family-name:var(--font-
geist-mono)]">

    <FormulaEditor

    onFormulaChange={handleFormulaChange}

    ref = {editorRef}

    latex={latex}/>

  </section>
```

```
  <section className=" p-4 font-[family-name:var(--font-geist-mono)] col-span-1">

    Анализ Формул

    <FormulaRenderer latex={latex} />

  </section>

</main>
```

Эта часть JSX включает FormulaEditor и FormulaRenderer; первый позволяет редактировать LaTeX код, второй отображает результат в формате LaTeX. Оба компонента связаны через переменную состояния latex, так что изменения в FormulaEditor отображаются в FormulaRenderer.

Web сервер и прокси

Настройка nginx

```
server {

    listen 80;

    server_name _;


    location / {

        proxy_pass http://frontend:3000;

        proxy_http_version 1.1;

        proxy_set_header Upgrade $http_upgrade;

        proxy_set_header Connection 'upgrade';

        proxy_set_header Host $host;

        proxy_cache_bypass $http_upgrade;

    }


    location /api/ {

        proxy_pass http://backend:8000;

        proxy_http_version 1.1;
```

```
        proxy_set_header X-Real-IP $remote_addr;

        proxy_set_header                            X-Forwarded-For
$proxy_add_x_forwarded_for;

        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_set_header Host $http_host;

    }

}
```

Этот файл конфигурации Nginx определяет перенаправление запросов к фронтенду и бэкенду. Все HTTP запросы на порт 80 рассматриваются, где запросы к корню (/) перенаправляются на фронтенд приложение, работающее на http://frontend:3000, и запросы к /api/ направляются на бэкенд API, работающее на http://backend:8000. Конфигурация также включает настройку прокси для поддержки веб-сокетов и передачи информации о клиенте, такой как IP-адрес и протокол запроса, что критически важно для адекватного логирования и обслуживания запросов на бэкенд.

Docker

Dockerfile для backend

```
FROM python:3.10-slim



WORKDIR /app/backend



COPY ./requirements.txt ./requirements.txt

RUN apt-get update \

    && apt-get -y install libpq-dev gcc \

    && pip install psycopg2



RUN pip install --no-cache-dir -r requirements.txt



COPY . .
```

```
EXPOSE 8000



CMD ["python", "manage.py", "runserver", "backend:8000"]
```

Этот Dockerfile описывает процесс развертывания backend-приложения на Python в контейнере. Он использует обрезанную версию официального образа Python 3.10 (python:3.10-slim) в качестве базового. Рабочая директория внутри контейнера устанавливается в /app/backend. Сначала файл зависимостей requirements.txt копируется в контейнер, а затем обновляются пакеты через apt-get, и устанавливаются необходимые системные библиотеки libpq-dev и gcc для компиляции расширений, таких как psycopg2 для работы с PostgreSQL. Все Python-зависимости из requirements.txt устанавливаются с опцией --no-cache-dir, чтобы не сохранять временные файлы установки. Затем в контейнер копируются все файлы проекта, и устанавливается порт 8000 для внешнего взаимодействия. Финальная команда запускает Django сервер разработки, используя команду manage.py runserver, указывая хост backend и порт 8000.

Dockerfile для frontend

```
FROM node:23.1-alpine



WORKDIR /app/frontend



COPY package*.json ./



RUN npm ci --only=development --force



COPY . .



RUN npm run build



EXPOSE 3000



CMD ["npm",  "run",  "dev"]
```

Этот Dockerfile предназначен для создания контейнера для фронтенд-приложения на Node.js. Он основывается на легковесной версии официального образа Node.js 23.1 (node:23.1-alpine). В контейнере устанавливается рабочая директория /app/frontend. Файлы зависимостей package.json и при наличии package-lock.json копируются в контейнер. Затем выполняется установка зависимостей, необходимых для разработки, с помощью npm ci с флагом --only=development для установки только разработческих зависимостей и --force для игнорирования предупреждений о несовместимых версиях. После установки зависимостей копируются все файлы проекта в контейнер, и запускается команда сборки npm run build. Контейнер настраивается для прослушивания порта 3000, и в конечном счете запускается dev-сервер с помощью команды npm run dev, что позволяет вести разработку и тестирование приложения внутри контейнера.

Dockerfile для nginx

```
FROM nginx:alpine



RUN rm /etc/nginx/conf.d/default.conf

COPY nginx.conf /etc/nginx/conf.d



EXPOSE 80



CMD ["nginx", "-g", "daemon off;"]
```

Этот Dockerfile создает образ, основанный на легковесном образе nginx:alpine, для веб-сервера Nginx, работающего на Alpine Linux. Первая инструкция удаляет конфигурационный файл по умолчанию Nginx (default.conf), чтобы предотвратить его автоматическую загрузку. После удаления стандартной конфигурации, пользовательская конфигурация Nginx копируется из локальной директории в контейнер (COPY nginx.conf /etc/nginx/conf.d). Это позволяет настроить веб-сервер согласно специфическим требованиям проекта, определяя, например, поведение сервера, маршрутизацию запросов и настройки безопасности.

Дальше, образ настроен для прослушивания стандартного HTTP порта 80 (EXPOSE 80), что делает его подходящим для обработки веб-трафика. Финальная инструкция CMD ["nginx", "-g", "daemon off;"] запускает Nginx в режиме фореграунда (foreground), так как по умолчанию Nginx запускается в режиме daemon, который автоматически отправляет его в фоновый режим. Запуск в режиме фореграунда важен для Docker-контейнеров, чтобы процесс

Nginx оставался основным процессом контейнера и его можно было правильно управлять через Docker.

Docker-compose.yml

```
services:
  db:
    image: postgres:13.3
    restart: always
    hostname: db
    environment:
      POSTGRES_DB: "db"
      POSTGRES_USER: "user"
      POSTGRES_PASSWORD: "admin"
    ports:
      - "5432:5432"
    volumes:
      - ./db-data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U user -d db"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 10s
    networks:
      - nett
  frontend:
    hostname: frontend
```

```
    build:
      context: ./latex/
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    networks:
      - nett
    volumes:
      - ./latex:/app/frontend
      - /app/frontend/node_modules
  backend:
    hostname: backend
    build:
      context: ./latexhahaton/
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    volumes:
      - ./latexhahaton:/app/backend
    networks:
      - nett
    depends_on:
      db:
        condition: service_healthy
  webserver:
    hostname: webserver
```

```
    build:
      context: ./nginx
      dockerfile: Dockerfile
    networks:
      - nett
    ports:
      - "8080:80"
    depends_on:
      - backend
      - frontend
networks:
  nett:
    driver: bridge
```

Этот файл описывает четыре сервиса:

1. **db**: Контейнер с изображением PostgreSQL 13.3, который автоматически перезапускается. Имеет переменные окружения для настройки базы данных, пользователя и пароля, пробрасывает порт 5432 и монтирует локальный каталог `./db-data` для хранения данных. Проведен тест на готовность сервиса с помощью `pg_isready`.

2. **frontend**: Контейнер строится из локальной папки `./latex/` с использованием `Dockerfile`. Пробрасывает порт 3000 для приложения и монтирует локальный каталог `./latex` и системный путь для node_modules, участвует в сети `nett`.

3. **backend**: Контейнер строится из `./latexhahaton/`, наследуя конфигурацию от `Dockerfile`, монтирует код из локальной директории `./latexhahaton`, пробрасывает порт 8000. Он зависит от базы данных, которая должна быть в состоянии 'healthy' перед запуском.

4. **webserver**: Использует Nginx для маршрутизации, строится из `./nginx`, пробрасывает порт 80 на хосте как 8080, и зависит от `backend` и `frontend` контейнеров, участвуя в той же сети.

Все сервисы подключены к сети `nett` с драйвером `bridge`, что позволяет контейнерам взаимодействовать друг с другом по именам хостов, указанным в конфигурации.