

Интеграция языковой модели DeepSeek в веб-сервис на основе NestJS

Андрюенко Иван Сергеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Пащенко Екатерина Александровна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассматривается архитектурное и программное решение по интеграции языковой модели DeepSeek в серверную часть веб-сервиса генерации тестовых заданий. Описывается реализация уровня доступа к API нейросети в виде специализированного сервиса DeepLlmService в рамках фреймворка NestJS, а также его использование в модуле AiModule для построения промптов и обработки результатов генерации. Приводятся особенности конфигурации соединения с API DeepSeek, включая работу с переменными окружения, выбор моделей deepseek-reasoner и deepseek-chat под различные сценарии генерации, параметры max_tokens и тайм-аутов. Отдельное внимание уделяется построению промптов для генерации тестов по теме и по загруженному файлу, а также механизму постобработки ответа нейросети с извлечением JSON-массива вопросов. Показано, что предложенная структура изолирует работу с внешним LLM в отдельном слое, упрощает масштабирование и модификацию логики генерации и обеспечивает надёжное взаимодействие фронтенда, серверной логики и внешнего API.

Ключевые слова: языковая модель, DeepSeek, NestJS, веб-сервис, генерация тестов, API, промпт, JSON.

Integration of the DeepSeek language model into a NestJS-based web service

Andrienko Ivan Sergeevich

Sholom-Aleichem Priamursky State University

Student

Paschenko Ekaterina Alexandrovna

Sholom-Aleichem Priamursky State University

Student

Abstract

The article discusses an architectural and software solution for integrating the DeepSeek language model into the backend of a web service for generating test tasks. It describes the implementation of the neural network API access layer in the

form of a specialized DeepImservice service within the NestJS framework, as well as its use in the AiModule module for building promptings and processing generation results. The features of the DeepSeek API connection configuration are given, including working with environment variables, selecting deepseek-reasoner and deepseek-chat models for various generation scenarios, max_tokens and timeouts parameters. Special attention is paid to the construction of prompta for generating tests on the topic and on the uploaded file, as well as the post-processing mechanism of the neural network response with the extraction of a JSON array of questions. It is shown that the proposed structure isolates work with external LLM in a separate layer, simplifies scaling and modification of the generation logic, and ensures reliable interaction between the frontend, server logic, and external API.

Keywords: language model, DeepSeek, NestJS, web service, test generation, API, prompt, JSON.

1 Введение

1.1 Актуальность

Интеллектуальные системы генерации учебного контента на основе больших языковых моделей занимают всё более заметное место в современной образовательной среде. Использование нейросетевых моделей позволяет автоматизировать создание тестовых заданий, адаптировать сложность вопросов к уровню обучающихся и ускорять разработку проверочных материалов. Вместе с тем интеграция таких моделей в реальное веб-приложение требует решения ряда архитектурных задач: организации безопасного и отказоустойчивого доступа к внешнему API, инкапсуляции логики взаимодействия с моделью в отдельных сервисах, обработки ошибок и корректной постобработки ответов. В условиях микросервисной или модульной архитектуры на основе NestJS важной задачей становится построение слоя, отвечающего за работу с внешней языковой моделью, и его интеграция с доменной логикой генерации тестов. Применение DeepSeek API в качестве источника генеративных возможностей представляет дополнительный интерес ввиду различий в моделях, ограничений по токенам и особенностей формата ответов, что делает обсуждаемый подход актуальным для разработчиков образовательных платформ и систем автоматизации тестирования.

1.2 Обзор исследований

Разработка веб-приложений на основе React и TypeScript активно рассматривается в научной литературе. В работе М.М. Ковцура, В.Е. Филимонова и Е.К. Щёголева анализируются подходы к созданию многопользовательских систем на базе React, где особое внимание уделяется организации типизации и архитектуры клиентской части [1]. А.А. Дудак в своём исследовании проводит сравнительный анализ инструментов разработки для проектных систем, выделяя преимущества стека технологий React и TypeScript по производительности и удобству сопровождения кода [2]. В статье Д.А. Хайруллина и М.Р. Богданова рассматриваются особенности

применения статической типизации в React-приложениях, включая методики типизации компонентов, хуков и состояния, а также влияние строгой типизации на надёжность пользовательских интерфейсов [3]. К.Р. Удальцов исследует вопросы обеспечения качества кода в командной разработке, предлагая формализацию шаблонов безопасных pull request-ов для проектов на React и TypeScript в рамках Git Flow [4]. Особое внимание созданию библиотек компонентов на основе TypeScript уделяется в работе Т.М. Воронко, где описываются принципы модульности, повторного использования кода и проектирования архитектуры компонентов для масштабируемых React-решений [5].

1.3 Цель исследования

Целью данной работы является разработка и описание программного решения по интеграции языковой модели DeepSeek в серверную часть веб-сервиса генерации тестовых заданий, реализованного на базе NestJS. В рамках исследования рассматриваются организация отдельного модуля для работы с LLM, построение сервиса DeepLlmService для обращения к API, использование этого сервиса в AiService при формировании промптов и обработке ответов, а также разделение сценариев генерации тестов по теме и на основании содержимого загружаемого файла.

2 Материалы и методы

Серверная часть веб-сервиса разработана с использованием фреймворка NestJS. Данный сервис реализован как Injectable-класс и использует библиотеку axios для выполнения HTTP-запросов к REST API DeepSeek. API-ключ был взят на DeepSeek Platform [6], считываются из переменных окружения DEEPSEEK_BASE_URL и DEEPSEEK_API_KEY.

3 Результаты и обсуждения

Перед началом разработки был получен индивидуальный API-ключ на платформе DeepSeek Platform [6], обеспечивающий доступ к эндпоинтам генеративных моделей и позволяющий выполнять защищённые запросы к сервису. Ключ был добавлен в переменные окружения сервера, что обеспечило безопасную конфигурацию и исключило необходимость хранения секретных данных в исходном коде.

Реализация интеграции языковой модели DeepSeek в серверную часть веб-приложения была выполнена на основе модульной структуры NestJS, что позволило выделить работу с внешним API в изолированный сервис и обеспечить устойчивость системы к изменениям внешнего поставщика. Первым этапом стала разработка отдельного модуля DeepLlmModule, предоставляющего доступ к сервису DeepLlmService через механизм внедрения зависимостей. Структура модуля крайне компактна и содержит только регистрацию провайдера, что позволяет подключать его в другие модули проекта без дублирования логики. Соответствующий код модуля представлен на рисунке (рис. 1).

```
backend > src > deepllm > TS deepllm.module.ts > DeepLlmModule
1  import { Module } from '@nestjs/common';
2  import { DeepLlmService } from './deepllm.service';
3
4  @Module({
5    providers: [DeepLlmService],
6    exports: [DeepLlmService],
7  })
8  export class DeepLlmModule {}
9
```

Рис. 1. Модуль DeepLlmModule с экспортируемым сервисом для взаимодействия с API DeepSeek

Ключевым компонентом интеграции является DeepLlmService, в котором сосредоточены настройки API, чтение параметров окружения, конфигурация заголовков и обработка запросов к языковой модели. В конструкторе сервиса считываются значения DEEPSEEK_BASE_URL и DEEPSEEK_API_KEY, а методы defaultHeaders и generateFromPrompt формируют корректный HTTP-запрос к эндпоинту /chat/completions. Метод generateFromPrompt принимает текст промпта, название модели и максимальное количество токенов, после чего формирует JSON-тело в формате dialog-messages, применяемом DeepSeek. Внутри предусмотрена обработка ошибок, включая отдельный случай с недостатком баланса на стороне API. Полный фрагмент реализации сервиса приведён на рисунке (рис. 2).

```
backend > src > deepLlm > TS deepLlm.service.ts > DeepLlmService > generateFromPrompt > body
1  import { Injectable, InternalServerErrorException } from '@nestjs/common';
2  import axios from 'axios';
3  import * as dotenv from 'dotenv';
4  dotenv.config();
5
6  @Injectable()
7  export class DeepLlmService {
8    private baseUrl =
9      process.env.DEEPSEEK_BASE_URL || 'https://api.deepseek.com/v1';
10   private apiKey = process.env.DEEPSEEK_API_KEY;
11
12   private defaultHeaders() {
13     return {
14       'Content-Type': 'application/json',
15       Authorization: `Bearer ${this.apiKey}`,
16     };
17   }
18
19   /**
20    * Call DeepSeek chat/completions API.
21    * prompt – полный текст промпта.
22    */
23   async generateFromPrompt(
24     prompt: string,
25     model = 'deepseek-reasoner',
26     max_tokens = 40000,
27   ): Promise<string> {
28     try {
29       const url = `${this.baseUrl}/chat/completions`;
30       const body = {
31         model,
32         messages: [{ role: 'user', content: prompt }],
33         max_tokens,
34       };
35
36       const resp = await axios.post(url, body, {
37         headers: this.defaultHeaders(),
38         timeout: 60000,
39       });
40     } catch (error) {
41       throw new InternalServerErrorException('Error generating response');
42     }
43   }
44 }
```

Рис. 2. Реализация сервиса DeepLlmService для взаимодействия с API генеративной модели DeepSeek

Следующим этапом стала интеграция DeepLlmService в общий модуль генерации тестов AiModule. Такой подход позволяет отделить задачу построения промптов и обработки результатов от задач сетевого взаимодействия. AiModule импортирует DeepLlmModule, передаёт DeepLlmService в AiService и регистрирует контроллер AiController. Архитектура обеспечивает чёткое разделение ответственности: модуль отвечает за генерацию тестов, а DeepLlmService – за работу с нейросетью. Соответствующий код модуля показан на рисунке (рис. 3).

```
backend > src > ai > TS ai.module.ts > ...
1  import { Module } from '@nestjs/common';
2  import { AiService } from './ai.service';
3  import { AiController } from './ai.controller';
4  import { DeepLlmModule } from '../deepllm/deepllm.module';
5  import { FileExtractionService } from './services/file-extraction.service';
6
7  @Module({
8    imports: [DeepLlmModule],
9    controllers: [AiController],
10   providers: [AiService, FileExtractionService],
11 })
12 export class AiModule {}
13
```

Рис. 3. Подключение DeepLlmService к модулю AiModule через механизм импортов NestJS

Основная логика генерации тестов сосредоточена в AiService, где DeepLlmService используется для обращения к нейросети при различных сценариях – генерации по теме и генерации на основе загруженного файла. В методе requestBatch выполняется формирование промпта на основе темы, количества вопросов, сложности, типов заданий и предыдущего контекста. Данный промпт передаётся в generateFromPrompt, после чего результат обрабатывается методом extractJsonArray, выделяющим JSON-массив с вопросами. Этот фрагмент показан на рисунке (рис. 4).

```

backend > src > ai > TS ai.service.ts > ...
 9  export class AiService {
23
24      private buildPrompt(
25          topic: string,
26          count: number,
27          difficulty: string,
28          types: string[],
29          existingQuestions: string[] = [],
30          language: 'ru' | 'en' = 'ru',
31          additionalInstructions = '',
32      ) {
33          const isRu = language === 'ru';
34          const difficultyDesc = isRu
35              ? difficulty === 'easy'
36                  ? 'простые факты'
37                  : difficulty === 'hard'
38                      ? 'сложный анализ и выводы'
39                      : 'умеренная сложность'
40              : difficulty === 'easy'
41                  ? 'basic facts'
42                  : difficulty === 'hard'
43                      ? 'complex analysis and reasoning'
44                      : 'moderate complexity';
45
46          const typesList = types.length ? types.join(', ') : 'mcq';
47
48          let dedupeBlock = '';
49          if (existingQuestions.length) {
50              const limited = existingQuestions.slice(-this.CONTEXT_LIMIT);
51              const lines = limited.map((q, index) => `${index + 1}. ${q}`).join('\n');
52              dedupeBlock = isRu
53                  ? `\\n\u2192 этого уже были вопросы (избегай повторов):\n${lines}\n`
54                  : `\\nPrevious questions (avoid duplicates):\n${lines}\n`;
55          }
56
57          let extraLine = '';
58          if (additionalInstructions.trim()) {
59              extraLine = isRu
60                  ? `\\n\u2192 дополнительные требования: ${additionalInstructions.trim()}`
61                  : `\\nAdditional requirements: ${additionalInstructions.trim()}`;
62          }
63
64          const blocks = {
65              ru: `Ты — помощник-преподаватель. Сгенерируй ${count} вопросов по теме "${topic}".
66 Сложность: ${difficulty} (${difficultyDesc}).
67 Типы вопросов: ${typesList}.${dedupeBlock}
68 Формат JSON-массива:
69 {

```

Рис. 4. Использование DeepLlmService внутри метода requestBatch для генерации набора вопросов

Отдельный путь реализации выделен для генерации тестов из файла. В методе generateFromFile выполняется обработка загруженного документа, извлечение текста с помощью FileExtractionService, нормализация параметров и построение специализированного промпта. Важно, что для файлов используется модель deepseek-chat с ограничением max_tokens=8192, соответствующим техническим лимитам API. Такой подход позволяет снизить нагрузку и обеспечить корректность работы генерации даже при больших входных данных. Схематичный фрагмент логики метода приведён на рисунке (рис. 5).

```

147 private buildFilePrompt(
148     fileName: string,
149     mime: string,
150     text: string,
151     count: number,
152     difficulty: string,
153     types: string[],
154     truncated: boolean,
155     language: 'ru' | 'en' = 'ru',
156     additionalInstructions = '',
157 ) {
158     const typesList = types.length ? types.join(', ') : 'mcq';
159     const difficultyDesc = this.describeDifficulty(difficulty);
160     const note = truncated
161       ? language === 'ru'
162         ? '\nВНИМАНИЕ: текст был обрезан по длине, избегай вопросов про обрезанную часть.'
163         : '\nNOTE: text was truncated for length; avoid questions about omitted parts.'
164       : '';
165
166     let extraLine = '';
167     if (additionalInstructions.trim()) {
168       extraLine = language === 'ru'
169         ? '\nДополнительные требования: ${additionalInstructions.trim()}'
170         : '\nAdditional requirements: ${additionalInstructions.trim()}';
171     }
172
173     const blocks = {
174       ru: `Ты – помощник-преподаватель. Ниже текст, извлечённый из файла "${fileName}" (${m
175       Все вопросы должны быть сосредоточены на фактах, причинах, следствиях, датах, участниках и
176
177     Формат JSON-ответа (массив объектов):
178     {
179       "id": number,
180       "type": "mcq" | "tf" | "open" | "open_auto",
181       "question": "строка",
182       "options": [...],
183       "correct": number | "True" | "False" | null,
184       "correctText": "строка",
185       "explanation": "строка",
186       "answerPatterns": [],
187       "errorMessage": "строка"
188     }

```

Рис. 5. Генерация вопросов на основе содержимого загруженного файла с использованием модели deepseek-chat

Проведённое тестирование показало, что интеграция DeepSeek выполняется корректно во всех сценариях. API стабильно возвращает результаты в ожидаемом формате, а построение промптов и разбор ответов позволяют получать структурированные наборы тестовых вопросов (рис. 6).

```
query: START TRANSACTION
query: INSERT INTO "tests"("id", "title", "questions", "maxScore", "status", "exportedUrl", "editUrl", "formSettings", "createdAt", "updatedAt", "userId") VALUES (DEFAULT, $1, $2, $3, $4, DEFAULT, DEFAULT, $5, DEFAULT, DEFAULT, $6) RETURNING "id", "maxScore", "status", "formSettings", "createdAt", "updatedAt" -- PARAMETERS: [{"литература 3 класс", [{"id":1,"type":"mcq","question": "басня И.А. Крылова «Стрекоза и Муравей» Муравей отказал Стрекозе в помощи. Какая основная мысль (мораль) этого произведения? \", \"options\": [\"Нужно всегда помогать друзьям, даже если они ленились.\", \"Лето красное пропела, оглянуться не успела, как зима ка глаза.\", \"Попрыгунья Стрекоза лето красное пропела.\", \"Помни, что делом нужно заниматься вовремя, а не бездельничать.\", \"ct\":3,\"explanation\": \"Мораль басни заключается в том, что нужно трудиться и готовиться к трудным временам заранее, а не ве ззаботный образ жизни.\", \"points\":1}]}],1,\"draft\", \"{\\\"collectEmails\\\":true,\\\"showMissedQuestions\\\":true,\\\"showCorrectAnswers\\ e,\\\"showQuestionScores\\\":true,\\\"showTotalScore\\\":true,\\\"requireFullName\\\":true,\\\"requireGroupInfo\\\":false}\", \"8d88b098-6954-463 4-24111670d8cf\"]
query: COMMIT
query: SELECT "TestEntity"."id" AS "TestEntity_id", "TestEntity"."title" AS "TestEntity_title", "TestEntity"."questions" AS "T tity_questions", "TestEntity"."maxScore" AS "TestEntity_maxScore", "TestEntity"."status" AS "TestEntity_status", "TestEntity"
```

Рис. 5. Результат запроса на backend

4 Выводы

Реализация интеграции языковой модели DeepSeek в серверную архитектуру веб-сервиса показала, что использование отдельного модуля и специализированного сервиса позволяет надёжно и структурированно организовать взаимодействие с внешним API. Построенная система промптов, разбор ответов и распределение функций между DeepLlmService и AiService обеспечили стабильную генерацию тестовых заданий как по текстовой теме, так и по содержимому загружаемых файлов. Проведённые испытания подтвердили корректность работы механизмов обработки ошибок, адаптацию под разные модели DeepSeek и устойчивость архитектуры к расширению. Итоговая структура продемонстрировала высокую гибкость и пригодность для дальнейшего развития сервиса автоматизированной генерации учебных материалов.

Библиографический список

1. Маршалл М. DeepSeek R1 и OpenAI Deep Research дали новое направление развития ИИ // БИТ. Бизнес & Информационные технологии. 2025. № 1 (144). С. 16–19.
2. Мурунов С. С. Использование педагогом нейросети DeepSeek при подготовке к уроку по иностранному языку // Иностранные языки в школе. 2025. № 2. С. 91–96.
3. Емельянов В., Емельянова Н. OpenAI vs DeepSeek. Сравнительный анализ популярных нейросетей // Системный администратор. 2025. № 3 (268). С. 68–71.
4. Свищёва И. В., Посысаев Г. С. Next.js, Nuxt.js и Nest.js: концептуальные и практические различия фреймворков // Инновации. Наука. Образование. 2021. № 46. С. 508–512.
5. Пензяков П. Д. Возможности фреймворка Nest JS для реализации серверной части веб-систем // Научный альманах Центрального Черноземья. 2022. № 1–4. С. 50–53.
6. DeepSeek Platform.URL: <https://platform.deepseek.com/> (дата обращения: 28.11.2025)