

Экспорт тестовых заданий в Google Forms на основе Google Forms API с автоматической настройкой режимов викторины

Андриенко Иван Сергеевич

Приамурский государственный университет имени Шолом-Алейхема
Студент

Пащенко Екатерина Александровна

Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация

В статье рассматривается практическая реализация экспорта тестовых заданий в Google Forms как завершающего этапа работы веб-сервиса формирования тестов. Показан подход, при котором после редактирования набора вопросов на клиентской стороне выполняется серверная сборка Google-формы через Google Forms API с включением режима викторины, назначением баллов, фиксацией правильных ответов и публикацией формы. Особое внимание уделяется обработке нескольких типов вопросов, нормализации и санитарной обработке текстовых полей, а также обеспечению устойчивости интеграции за счёт контроля срока действия OAuth-токена и повторных попыток при ошибках внешнего API. Результатом является унифицированный экспортный конвейер, который создаёт форму, наполняет её вопросами и возвращает пользователю готовые ссылки на просмотр и редактирование.

Ключевые слова: Google Forms API, экспорт тестов, веб-интеграция, OAuth 2.0, автоматизация оценивания, формирование викторины, NestJS, TypeScript.

Exporting test tasks to Google Forms based on the Google Forms API with automatic quiz mode settings

Andrienko Ivan Sergeevich

Sholom-Aleichem Priamursky State University
Student

Paschenko Ekaterina Alexandrovna

Sholom-Aleichem Priamursky State University
Student

Abstract

The article discusses the practical implementation of exporting test tasks to Google Forms as the final stage of the test generation web service. An approach is shown in which, after editing a set of questions on the client side, a server build of the Google form is performed via the Google Forms API, enabling quiz mode, assigning points,

fixing correct answers and publishing the form. Special attention is paid to handling several types of issues, normalization and sanitization of text fields, as well as ensuring the stability of integration by monitoring the validity period of the OAuth token and retries in case of external API errors. The result is a unified export pipeline that creates a form, fills it with questions, and returns ready-made links for viewing and editing to the user.

Keywords: Google Forms API, test export, web integration, OAuth 2.0, assessment automation, quiz generation, NestJS, TypeScript.

1 Введение

1.1 Актуальность

На практике преподавателям и методистам требуется быстро переводить подготовленные тестовые задания в удобный формат, пригодный для распространения и автоматической проверки. Google Forms является распространённым инструментом для проведения опросов и тестирования, однако ручное создание формы и настройка оценивания для большого количества вопросов занимает значительное время и повышает риск ошибок, связанных с некорректными баллами, неверно заданными правильными ответами и несогласованностью формулировок. Автоматизированный экспорт устраняет рутинные операции, формирует единый стандарт представления вопросов и обеспечивает повторяемость результата при многократном создании форм на основе разных тестов.

1.2 Обзор исследований

Реализация авторизации пользователей с использованием протокола Google OAuth активно рассматривается в научной и прикладной литературе. В работе Р.И. Круглика анализируются практические аспекты внедрения Google OAuth API в веб-приложениях, включая особенности настройки внешней авторизации и упрощения процесса входа для пользователей без хранения паролей на стороне сервиса [1]. А.С. Мосягин в своём исследовании подробно рассматривает принципы функционирования системы авторизации Google, описывая этапы OAuth-аутентификации, получение токенов доступа и обработку пользовательских данных на примерах реальных веб-сценариев [2]. В статье И.С. Красильникова исследуются вопросы разработки функций авторизации в сервисах компании Google, где подчёркивается универсальность OAuth-подхода и его применимость для интеграции сторонних веб-систем с экосистемой Google [3]. 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 [4]. 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 [5].

1.3 Цель исследования

Целью исследования является разработка и описание программного решения, обеспечивающего автоматическое создание Google-формы в режиме викторины на основе структурированного набора тестовых вопросов, включая поддержку нескольких типов заданий, назначение баллов, настройку параметров формы и выдачу пользователю ссылок для дальнейшего использования и редактирования.

2 Материалы и методы

Серверная часть реализована на TypeScript с применением NestJS и официального пакета `googleapis` для обращения к Google Forms API; клиентская часть формирует запрос экспорта и передаёт данные теста и параметры формы, используя OAuth-access token пользователя.

3 Результаты и обсуждения

В результате выполненной разработки реализован механизм экспорта тестовых заданий в сервис Google Forms, обеспечивающий автоматическое создание форм в режиме викторины на основе предварительно сформированной структуры теста. Реализованное решение позволяет исключить ручное создание форм, сократить время подготовки контрольных материалов и обеспечить единообразие настроек оценивания и структуры вопросов.

Ключевым элементом серверной части является сервис `GoogleFormService`, отвечающий за взаимодействие с Google Forms API. В рамках его работы осуществляется создание OAuth-клиента на основе access-токена пользователя, полученного в процессе авторизации, после чего инициируется запрос на создание новой формы с заданным заголовком. Данный этап формирует основу будущего теста и подготавливает идентификатор формы для последующих операций. Фрагмент создания OAuth-клиента и инициализации формы приведён на рисунке (рис. 1).

```
26 @Injectable()
27 export class GoogleFormService {
28   private CLIENT_ID = process.env.GOOGLE_CLIENT_ID!;
29   private CLIENT_SECRET = process.env.GOOGLE_CLIENT_SECRET!;
30
31   private createOAuthClient(access_token: string) {
32     const oauth2Client = new google.auth.OAuth2(
33       this.CLIENT_ID,
34       this.CLIENT_SECRET,
35     );
36     oauth2Client.setCredentials({ access_token: access_token });
37     return oauth2Client;
38   }
39
40   private sanitizeText(input: any, maxLength = 1000): string {
41     if (input === null || input === undefined) return '';
42     let s = String(input);
43     s = s.replace(/\r\n/g, ' ');
44     s = s.replace(/\s+/g, ' ').trim();
45     if (s.length > maxLength) s = s.slice(0, maxLength).trim();
46     return s;
47   }
48
49   private containsCyrillic(text: string | undefined | null) {
50     if (!text) return false;
51     return /[А-Яа-яЁё]/.test(text);
52   }
53
54   private mergeFormSettings(input?: Partial<FormSettings>): FormSettings {
55     const source = input ?? {};
56     return {
57       collectEmails: source.collectEmails ?? defaultFormSettings.collectEmail,
58       showMissedQuestions:
59         source.showMissedQuestions ?? defaultFormSettings.showMissedQuestions,
60       showCorrectAnswers:
61         source.showCorrectAnswers ?? defaultFormSettings.showCorrectAnswers,
62       showQuestionScores:
63         source.showQuestionScores ?? defaultFormSettings.showQuestionScores,
64       showTotalScore:
65         source.showTotalScore ?? defaultFormSettings.showTotalScore,
66       requireFullName:
67         source.requireFullName ?? defaultFormSettings.requireFullName,
68       requireGroupInfo:
69         source.requireGroupInfo ?? defaultFormSettings.requireGroupInfo,
70     };
71   }
72 }
```

Рис. 1. Создание OAuth-клиента и инициализация Google-формы

После успешного создания формы формируется пакет batch-запросов, используемый для последовательного добавления элементов формы и изменения её параметров. В первую очередь включается режим викторины и настраивается сбор адресов электронной почты респондентов в зависимости от выбранных параметров. Использование batch-обновления позволяет сократить количество обращений к API и обеспечить атомарность применения настроек. Реализация формирования batch-запросов показана на рисунке (рис. 2).

```
73  ✓ async createFormWithQuestions(  
74      accessToken: string,  
75      title: string,  
76      questions: Question[],  
77      formSettings?: Partial<FormSettings>,  
78      attempt = 0,  
79  ) {  
80  ✓  try {  
81      const auth = this.createOAuthClient(accessToken);  
82      const forms = google.forms({ version: 'v1', auth });  
83  
84      const sanitizedTitle = this.sanitizeText(title || 'Новый тест', 200);  
85      const settings = this.mergeFormSettings(formSettings);  
86  
87  ✓      const createRes = await forms.forms.create({  
88  ✓          requestBody: {  
89  ✓              info: {  
90                  title: sanitizedTitle,  
91              },  
92          },  
93      });  
94  
95      const createData: any = createRes.data;  
96      let formId: string | undefined = createData?.formId;  
97  ✓      if (!formId && createData?.name) {  
98          const parts = (createData.name as string).split('/');  
99          formId = parts[parts.length - 1];  
100      }  
101  
102  ✓      if (!formId) {  
103          throw new Error(  
104              'Cannot determine formId from create response: ' +  
105              JSON.stringify(createData),  
106          );  
107      }  
108  
109  ✓      const requests: any[] = [  
110  ✓          {  
111  ✓              updateSettings: {  
112  ✓                  settings: {  
113                      quizSettings: { isQuiz: true },  
114                      emailCollectionType: settings.collectEmails  
115                          ? 'RESPONDER_INPUT'  
116                          : 'DO_NOT_COLLECT',
```

Рис. 2. Формирование batch-запросов и настройка параметров формы

В ходе обработки теста сервис последовательно добавляет вопросы различных типов. Для заданий с множественным выбором формируется набор вариантов ответа, назначается правильный вариант и количество баллов, после чего создаётся элемент `choiceQuestion` с типом `RADIO`. Аналогичным образом обрабатываются задания формата «верно/неверно» с учётом языка вопроса и вариантов ответа. Логика формирования вопросов закрытого типа представлена на рисунке (рис. 3).

```

214 } else if (q.type === 'tf') {
215   const hasRuQuestion = this.containsCyrillic(q.question);
216   const providedRaw = Array.isArray(q.options) ? q.options : [];
217   const provided = providedRaw
218     .map(opt => this.sanitizeText(opt, 200))
219     .filter(opt => opt.length > 0);
220   const hasRuOptions = provided.some(opt =>
221     this.containsCyrillic(opt),
222   );
223
224   let options: string[];
225   if (hasRuQuestion || hasRuOptions) {
226     options = ['Верно', 'Неверно'];
227   } else if (provided.length >= 2) {
228     options = provided.slice(0, 2);
229   } else {
230     options = ['True', 'False'];
231   }
232
233   const defaultTrue = this.containsCyrillic(options[0])
234     ? 'Верно'
235     : 'True';
236   const defaultFalse =
237     options.length > 1 && this.containsCyrillic(options[1])
238     ? 'Неверно'
239     : 'False';
240
241   let correctIndex = 0;
242   if (typeof q.correct === 'number' && q.correct >= 0) {
243     correctIndex =
244       q.correct < options.length ? q.correct : options.length - 1;
245   } else if (typeof q.correct === 'string') {
246     const norm = q.correct.toString().toLowerCase();
247     const directMatch = options.findIndex(
248       (opt) => opt.toLowerCase() === norm,
249     );
250     if (directMatch >= 0) {
251       correctIndex = directMatch;
252     } else if (
253       norm === 'false' ||
254       norm === 'ложь' ||

```

Рис. 3. Добавление вопросов с множественным выбором и формата «верно/неверно»

Для открытых вопросов реализована поддержка как ручной проверки, так и автоматической оценки на основе шаблонов правильных ответов. В случае автоматической проверки формируется набор допустимых ответов и, при необходимости, текст обратной связи, отображаемый пользователю при ошибке. Это позволяет использовать Google Forms не только для тестов с выбором ответа, но и для более сложных заданий с текстовым вводом. Реализация обработки открытых вопросов приведена на рисунке (рис. 4).

```
289 } else if (q.type === 'open_auto') {
290   const normalizedPatterns = (q.answerPatterns ?? [])
291     .map(value => this.sanitizeText(value, 200))
292     .filter(value => value.length > 0);
293
294   if (!normalizedPatterns.length) {
295     normalizedPatterns.push('Правильный ответ');
296   }
297
298   const feedbackText = q.errorMessage
299     ? this.sanitizeText(q.errorMessage, 500)
300     : q.explanation
301       ? this.sanitizeText(q.explanation, 700)
302       : undefined;
303
304   requests.push({
305     createItem: {
306       item: {
307         title: qTitle,
308         questionItem: {
309           question: {
310             required: true,
311             grading: {
312               pointValue,
313               correctAnswers: {
314                 answers: normalizedPatterns.map((value) => ({
315                   value,
316                 })),
317             },
318             generalFeedback: feedbackText
319               ? {
320                 text: feedbackText,
321               }
322               : undefined,
323           },
324           textQuestion: {},
325         },
326       },
327     },
328   });
329 }
```

Рис. 4. Реализация открытых вопросов и автоматической проверки ответов

Для повышения надёжности экспорта в сервисе реализована санитарная обработка текстовых данных, включающая удаление лишних переносов строк, нормализацию пробелов и ограничение длины строк. Данный механизм позволяет избежать ошибок Google Forms API, связанных с некорректным форматированием или превышением допустимых размеров текстовых полей. Реализация функций очистки текста показана на рисунке (рис. 5).

```
12 type QuestionType = 'mcq' | 'tf' | 'open' | 'open_auto';
13
14 type Question = {
15   id: number;
16   type: QuestionType;
17   question: string;
18   options?: string[];
19   correct: number | 'True' | 'False' | null;
20   points?: number;
21   explanation?: string;
22   answerPatterns?: string[];
23   errorMessage?: string;
24 };
25
26 @Injectable()
27 export class GoogleFormService {
28   private CLIENT_ID = process.env.GOOGLE_CLIENT_ID!;
29   private CLIENT_SECRET = process.env.GOOGLE_CLIENT_SECRET!;
30
31   private createOAuthClient(access_token: string) {
32     const oauth2Client = new google.auth.OAuth2(
33       this.CLIENT_ID,
34       this.CLIENT_SECRET,
35     );
36     oauth2Client.setCredentials({ access_token: access_token });
37     return oauth2Client;
38   }
39
40   private sanitizeText(input: any, maxlen = 1000): string {
41     if (input === null || input === undefined) return '';
42     let s = String(input);
43     s = s.replace(/\r\n/g, ' ');
44     s = s.replace(/\s+/g, ' ').trim();
45     if (s.length > maxlen) s = s.slice(0, maxlen).trim();
46     return s;
47   }
48
49   private containsCyrillic(text: string | undefined | null) {
50     if (!text) return false;
51     return /[А-Яа-яЁё]/.test(text);
52   }
53 }
```

Рис. 5. Санитарная обработка текстовых данных перед экспортом

После добавления всех элементов формы выполняется этап публикации, в рамках которого форма переводится в состояние, доступное для прохождения респондентами. По завершении операции сервер возвращает ссылки на просмотр и редактирование формы, которые используются клиентской частью приложения для отображения результата экспорта пользователю. Логика публикации формы и получения итоговых ссылок представлена на рисунке (рис. 6).

```
869     let batchData: any = null;
870     if (requests.length > 0) {
871         const batchRes = await forms.forms.batchUpdate({
872             formId,
873             requestBody: { requests },
874         });
875         batchData = batchRes.data;
876     }
877
878     try {
879         await forms.forms.setPublishSettings({
880             formId,
881             requestBody: {
882                 publishSettings: {
883                     publishState: {
884                         isPublished: true,
885                         isAcceptingResponses: true,
886                     },
887                 },
888             },
889         });
890
891         const getRes = await forms.forms.get({ formId });
892         const formData = getRes.data as any;
893         const responderUri = formData?.responderUri ?? null;
894         const editUrl = `https://docs.google.com/forms/d/${formId}/edit`;
895         const viewUrl =
896             responderUri ?? `https://docs.google.com/forms/d/${formId}/viewform`
897     }
```

Рис. 6. Публикация формы и получение ссылок на просмотр и редактирование

Проведённые испытания подтвердили корректность работы механизма экспорта для различных типов заданий, устойчивость к временным ошибкам внешнего API и корректную обработку истечения срока действия access-токена. В целом реализованное решение продемонстрировало практическую применимость и готовность к использованию в образовательных веб-сервисах, ориентированных на автоматизированное создание тестовых материалов.

4 Выводы

В ходе выполненной работы реализован механизм экспорта тестовых заданий в сервис Google Forms, обеспечивающий автоматизированное создание форм с поддержкой различных типов вопросов, системы оценивания и публикации результатов. Использование серверного сервиса для взаимодействия с Google Forms API позволило централизовать логику формирования форм, повысить надёжность обработки данных и обеспечить корректную работу с OAuth-аутентификацией. Реализованные механизмы санитарной обработки входных данных, обработки ошибок и повторных запросов подтвердили устойчивость решения при практическом использовании. Итоговая архитектура показала свою пригодность для

применения в образовательных веб-системах и возможность дальнейшего расширения функциональности экспорта тестов.

Библиографический список

1. Круглик Р.И. Авторизация на сайте через Google OAuth API // Постулат. 2020. № 1 (51). С. 98.
2. Мосягин А.С. Рассмотрение принципов авторизации в системе Google с примерами // Аллея науки. 2018. Т. 3. № 7 (23). С. 119–124.
3. Красильников И.С. Разработка функций для авторизации пользователя в сервисах компании Google // В сборнике: XX Всероссийская студенческая научно-практическая конференция Нижневартковского государственного университета. Нижневартовск, 2018. С. 188–190.
4. Дудак А.А. Сравнительный анализ инструментов разработки для систем управления проектами: преимущества стек-технологий TypeScript и React // Новая наука: от идеи к результату. 2024. № 9. С. 32–40.
5. Хайруллин Д.А., Богданов М.Р. Изучение использования TypeScript в React-приложениях, преимущества статической типизации, подходы к типизации компонентов, хуков и состояния // Международный журнал гуманитарных и естественных наук. 2024. № 12-3 (99). С. 233–238.
6. Google Cloud console. URL: <https://console.cloud.google.com> (дата обращения: 28.11.2025)