

## **Реализация переключателя языка интерфейса в веб-приложении на основе контекстной интернационализации Next.js**

*Андриенко Иван Сергеевич*

*Приамурский государственный университет имени Шолом-Алейхема*  
*Студент*

*Пащенко Екатерина Александровна*

*Приамурский государственный университет имени Шолом-Алейхема*  
*Студент*

### **Аннотация**

В статье рассматривается реализация механизма переключения языка пользовательского интерфейса в веб-приложении, разработанном на базе Next.js (React, TypeScript). Описан подход, основанный на контекстной модели интернационализации, при которой текущее значение локали хранится в состоянии провайдера `I18nProvider` и передаётся компонентам через контекст. Приведены принципы построения словарей переводов для русского и английского языков, способ формирования набора доступных локалей для элемента выбора языка, а также логика функции перевода `t`, поддерживающей обращение к строкам по пути и подстановку параметров.

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

## **Implementation of the interface language switch in a web application based on contextual internationalization Next.js**

*Andrienko Ivan Sergeevich*

*Sholom-Aleichem Priamursky State University*  
*Student*

*Paschenko Ekaterina Alexandrovna*

*Sholom-Aleichem Priamursky State University*  
*Student*

### **Abstract**

The article discusses the implementation of a mechanism for switching the user interface language in a web application developed on the basis of Next.js (React, TypeScript). An approach based on the contextual internationalization model is described, in which the current locale value is stored in the state of the `I18nProvider` provider and transmitted to components through the context. The principles of constructing translation dictionaries for Russian and English, the method of forming a set of available locales for the language selection element, as well as the logic of

the translation function `t`, which supports accessing strings along the path and substituting parameters, are presented.

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

## 1 Введение

### 1.1 Актуальность

Мультиязычность интерфейса является важным требованием для современных образовательных веб-сервисов, ориентированных на широкую аудиторию пользователей с различными языковыми предпочтениями. При разработке систем, поддерживающих генерацию и экспорт тестовых заданий, наличие нескольких языков интерфейса повышает доступность сервиса для преподавателей и обучающихся, упрощает освоение функционала и расширяет потенциальную географию применения. На практике ключевой проблемой становится обеспечение корректной и быстрой смены языка без перезагрузки страницы, сохранение выбранной локали между пользовательскими сессиями и поддержка единообразного доступа к переводам из любых компонентов приложения. Использование Next.js и компонентной архитектуры React создаёт предпосылки для реализации контекстной модели интернационализации, при которой локаль хранится централизованно, а компоненты получают переводы через общую функцию доступа. Такой подход позволяет унифицировать работу с текстовыми ресурсами и уменьшить связность интерфейсных модулей, что делает исследуемую задачу актуальной с позиции проектирования масштабируемых UI-систем.

### 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 механизма переключения языка интерфейса в веб-приложении AI Test Generator, обеспечивающего смену локали между русским и английским языками, сохранение пользовательского выбора в `localStorage` и динамическую перерисовку интерфейса с использованием контекста React. В рамках исследования рассматриваются структура `I18nProvider`, построение словарей переводов и списка доступных локалей, работа функции `t` с поддержкой пути к строке и `fallback`-логики, а также реализация переключателя языка в компоненте `Header` для различных режимов отображения.

## **2 Материалы и методы**

В качестве среды разработки использовалось клиентское приложение на основе Next.js, использующее React и TypeScript, а также модульную структуру компонентов и хуков. Механизм интернационализации реализован через контекстный провайдер `I18nProvider`, расположенный в модуле `frontend/context/i18n-context.tsx`, который оборачивает всё приложение в файле `providers.tsx`.

## **3 Результаты и обсуждения**

В результате выполненной разработки был реализован механизм переключения языка пользовательского интерфейса в веб-приложении AI Test Generator, обеспечивающий динамическую смену локали без перезагрузки страницы. Реализация основана на использовании контекста React и централизованного провайдера интернационализации, что позволило унифицировать доступ к переводам во всех компонентах приложения.

Ключевым элементом системы является контекст `I18nProvider`, отвечающий за хранение текущей локали и предоставление функций управления языком. Внутри провайдера используется состояние `locale`, инициализируемое значением по умолчанию, а также дополнительная логика загрузки сохранённой локали из `localStorage` при первом монтировании приложения. Такой подход обеспечивает восстановление пользовательских настроек между сессиями. Фрагмент реализации контекста и инициализации локали представлен на рисунке (рис. 1).

```

21 type Replacements = Record<string, string | number>;
22
23 type I18nContextValue = {
24   locale: Locale;
25   setLocale: (value: Locale) => void;
26   availableLocales: typeof localeOptions;
27   dictionary: Dictionary;
28   t: (path: string, replacements?: Replacements, fallback?: string) => string;
29 };
30
31 const STORAGE_KEY = "aitg_locale";
32
33 const I18nContext = createContext<I18nContextValue | undefined>(undefined);
34
35 const getValueFromPath = (obj: unknown, path: string): unknown => {
36   if (!obj) return undefined;
37   return path.split(".").reduce<unknown>((acc, segment) => {
38     if (acc && typeof acc === "object") {
39       return (acc as Record<string, unknown>)[segment];
40     }
41     return undefined;
42   }, obj);
43 };
44
45 const formatTemplate = (template: string, replacements?: Replacements): string => {
46   if (!replacements) return template;
47   return Object.entries(replacements).reduce((acc, [key, value]) => {
48     const safeValue = String(value);
49     return acc.replace(new RegExp(`{{\\s*${key}\\s*}}`, "g"), safeValue);
50   }, template);
51 };
52

```

Рис. 1. Инициализация состояния локали и загрузка значения из localStorage в I18nProvider

При каждом изменении значения locale выполняется побочный эффект, обновляющий атрибут lang корневого HTML-элемента и сохраняющий новое значение языка в localStorage. Это решение повышает семантическую корректность документа, а также улучшает взаимодействие с браузерными механизмами доступности и поисковой индексации. Реализация данного эффекта показана на рисунке (рис. 2).

```

53 export function I18nProvider({ children }: { children: ReactNode }) {
54   const [locale, setLocaleState] = useState<Locale>(defaultLocale);
55
56   useEffect(() => {
57     if (typeof window === "undefined") return;
58     const stored = window.localStorage.getItem(STORAGE_KEY);
59     if (stored && isLocale(stored)) {
60       // Синхронизируем же состояние только при первом монтировании
61       setLocaleState(stored);
62     }
63   }, []);
64
65   useEffect(() => {
66     if (typeof document !== "undefined") {
67       document.documentElement.lang = locale;
68     }
69     if (typeof window !== "undefined") {
70       window.localStorage.setItem(STORAGE_KEY, locale);
71     }
72   }, [locale]);
73
74   const setLocale = useCallback((value: Locale) => {
75     setLocaleState(value);
76   }, []);
77

```

Рис. 2. Синхронизация локали с localStorage и атрибутом lang HTML-документа

Для получения текстовых ресурсов в зависимости от текущей локали была реализована универсальная функция перевода *t*. Она осуществляет поиск строки по иерархическому пути внутри словаря выбранного языка и поддерживает механизм *fallback*, при котором при отсутствии перевода используется значение из словаря языка по умолчанию. В случае отсутствия строки в обоих словарях возвращается сам путь, что упрощает отладку и выявление пропущенных переводов. Алгоритм поиска и обработки строк представлен на рисунке (рис. 3).

```
74   const setLocale = useCallback((value: Locale) => {
75     setLocaleState(value);
76   }, []);
77
78   const translate = useCallback(
79     (path: string, replacements?: Replacements, fallback?: string) => {
80       const localized = getValueFromPath(dictionaries[locale], path);
81       const fallbackValue =
82         localized ??
83         (locale !== defaultLocale ? getValueFromPath(dictionaries[defaultLocale], path) : undefined);
84       const rawValue =
85         typeof localized === "string"
86           ? localized
87           : typeof fallbackValue === "string"
88             ? fallbackValue
89             : fallback;
90       if (!rawValue) {
91         return path;
92       }
93       return formatTemplate(rawValue, replacements);
94     },
95     [locale],
96   );
97
98   const value = useMemo<I18nContextValue>(
99     () => ({
100       locale,
101       setLocale,
102       availableLocales: localeOptions,
103       dictionary: dictionaries[locale],
104       t: translate,
```

Рис. 3. Реализация функции перевода *t* с поддержкой *fallback*-логики

Сами словари переводов реализованы в виде статических объектов для русского и английского языков. Каждый словарь содержит структурированные разделы, соответствующие логическим блокам интерфейса, таким как заголовок, навигация и метаданные страниц. Для формирования списка доступных языков используется массив *localeOptions*, автоматически генерируемый на основе словарей и содержащий значения локали и отображаемые названия языков. Фрагмент описания словарей и генерации доступных локалей приведён на рисунке (рис. 4).

```
398      empty: "Операция еще не была.",
399      defaultReason: "Операция",
400      questionsLabel: "вопросов",
401      reasons: {
402        generate_test: "Генерация теста",
403        generate_file: "Генерация из файла",
404        refund_generate: "Возврат за генерацию",
405      },
406    },
407  },
408  oauth: {
409    loading: "Синхронизируем данные Google. Пожалуйста, подождите...",
410  },
411 },
412 en: {
413   languageName: "English",
414   meta: {
415     title: "AI Test Generator – create tests in minutes",
416     description:
417       "Help teachers and trainers generate question sets, edit drafts, and export them to Google F
418   },
419   header: {
420     brand: "AI Test Generator",
421     nav: {
422       home: "Home",
423       dashboard: "My tests",
424       profile: "Profile",
425     },
426     login: "Sign in"
```

Рис. 4. Структура словарей переводов и формирование списка доступных локалей

Переключатель языка интерфейса реализован в компоненте Header и представлен элементом select, связанным со значением текущей локали. При выборе нового языка вызывается функция setLocale, что приводит к обновлению состояния в I18nProvider и автоматической перерисовке всех компонентов, использующих функцию t. Аналогичная логика применяется как для десктопной версии заголовка, так и для мобильного меню. Фрагмент реализации переключателя языка в заголовке представлен на рисунке (рис. 6).

```

54   return (
55     <header className="sticky top-0 z-40 border-b border-divider bg-surface/90 backdrop-blur">
56       <div className="mx-auto flex max-w-6xl items-center justify-between gap-4 px-4 py-4 sm:px-6 lg:px-8">
57         <Link href="/" className="text-lg font-semibold text-ink">
58           {t("header.brand")}
59         </Link>
60
61         <nav className="hidden items-center gap-4 md:flex">{renderNavLinks()}</nav>
62
63         <div className="flex items-center gap-3">
64           <label className="hidden text-sm text-muted lg:block">
65             {t("header.languageLabel")}
66             <select
67               value={locale}
68               onChange={event => setLocale(event.target.value as Locale)}
69               className="ml-2 rounded-full border border-divider bg-surface px-3 py-1 text-sm text-ink focus:outline focus:outline-2 focus:outline-ink"
70             >
71               {availableLocales.map(option => (
72                 <option key={option.value} value={option.value}>
73                   {option.label}
74                 </option>
75               ))}
76             </select>
77           </label>
78           {!user && (
79             <Button onClick={login} size="sm">
80               {t("header.login")}
81             </Button>
82           )}

```

Рис. 5. Реализация переключателя языка в компоненте Header

Смена языка происходит мгновенно и корректно отражается во всех элементах пользовательского интерфейса, включая навигацию, кнопки и текстовые блоки. Выбранный язык сохраняется между сессиями пользователя, а архитектура решения позволяет без существенных изменений расширить количество поддерживаемых языков (рис. 6).

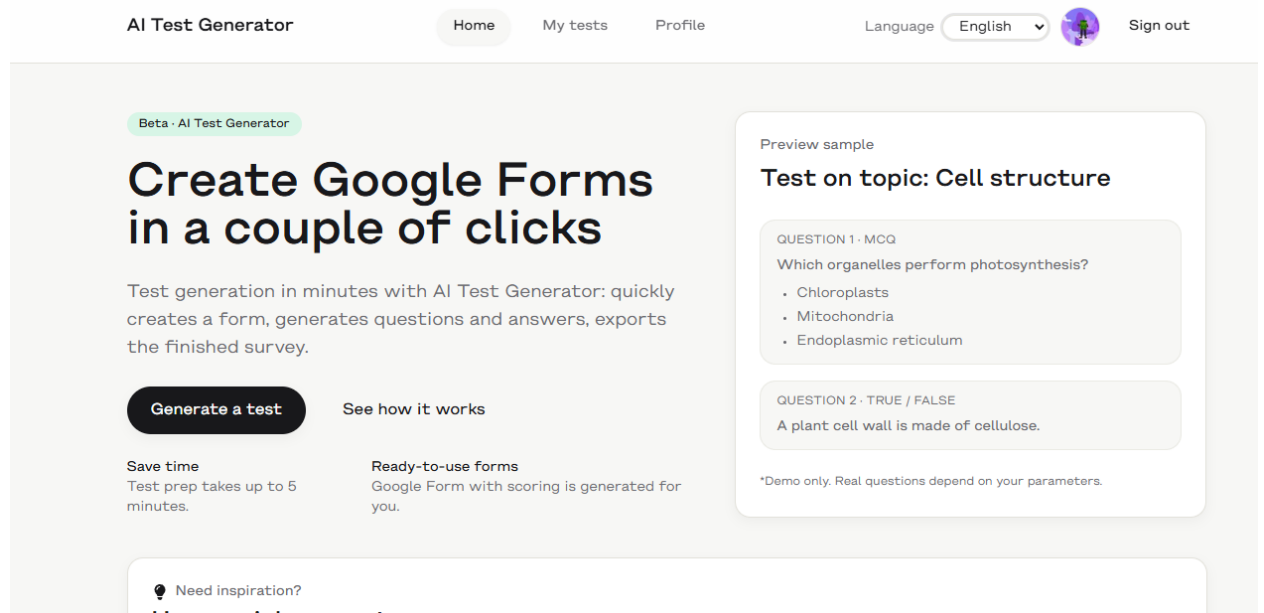


Рис. 6. Тестирование переключателя

#### **4 Выводы**

В ходе работы был реализован механизм переключения языка интерфейса в веб-приложении, основанный на контекстной модели интернационализации React. Разработанное решение обеспечивает динамическую смену локали без перезагрузки страницы, сохранение пользовательских настроек в localStorage и корректное обновление атрибута lang HTML-документа. Использование статических словарей переводов и централизованного провайдера позволило упростить доступ к текстовым ресурсам и обеспечить масштабируемость системы при добавлении новых языков, что подтверждает эффективность выбранного подхода для мультязычных веб-приложений.

#### **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.