

**«Санкт-Петербургский государственный электротехнический университет
«ЛЭТИ» им. В.И.Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)**

Направление	01.03.02 - Прикладная математика и информатика
Профиль	Математическое обеспечение программно-информационных систем
Факультет	КТИ
Кафедра	МО ЭВМ

К защите допустить

Зав. кафедрой

А.А. Лисс

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
БАКАЛАВРА**

Тема: РАЗРАБОТКА БИБЛИОТЕКИ ГЕНЕРАЦИИ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ ДЛЯ FRONTEND ТЕСТИРОВАНИЯ

Студент		<hr/>	Б.В. Прохоров
		<i>подпись</i>	
Руководитель	К.Т.Н., доцент	<hr/>	М.М. Заславский
	<i>(Уч. степень, уч. звание)</i>	<i>подпись</i>	
Консультанты	К.Т.Н., доцент	<hr/>	А.И. Маловский
	<i>(Уч. степень, уч. звание)</i>	<i>подпись</i>	

Санкт-Петербург

2024

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю

Зав. кафедрой МО ЭВМ

_____ А.А. Лисс

«___» _____ 20__ г.

Студент Прохоров Б.В.

Группа 0381

Тема работы: Разработка библиотеки генерации пользовательских данных для frontend тестирования

Место выполнения ВКР: ООО "ЯДРО ЦЕНТР ТЕХНОЛОГИЙ МОБИЛЬНОЙ СВЯЗИ"

Исходные данные (технические требования):

Требуется разработать библиотеку генерации пользовательских данных для frontend тестирования.

Библиотека иметь функциональность для генерации следующих типов пользовательских данных:

- UUID;
- Числа в заданном диапазоне;
- Случайные строки с заданным набором символов и длиной;
- Имена людей (имя, отчество, фамилия) с учетом пола и падежа (только для русского языка);
- Осмысленные строки на заданном языке (английский или русский).

Библиотека должна быть локализована на русский и английский языки.

Библиотека должна быть опубликована в реестре NPM, чтобы сторонние разработчики могли подключить её и использовать в своих проектах.

Содержание ВКР:

Введение, Обзор аналогов, Архитектура библиотеки, Реализация библиотеки, Сборка библиотеки, Регистрация пакета в NPM реестре, Внедрение пакета в проект, Оценка производительности библиотеки, Заключение.

Перечень отчетных материалов: пояснительная записка, иллюстративный материал, презентация

Дополнительные разделы: безопасность жизнедеятельности.

Дата выдачи задания

«01» _____ 2024 г.

Дата представления ВКР к защите

«__» _____ 2024 г.

Студент

Б.В. Прохоров

Руководитель к.т.н., доцент
(Уч. степень, уч. звание)

М.М. Заславский

Консультант к.т.н., доцент
(Уч. степень, уч. звание)

Б.Ю. Медошин

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю
Зав. кафедрой МО ЭВМ
_____ А.А. Лисс
« ____ » _____ 2024 г.

Студент Прохоров Б.В. Группа 0381
Тема работы: Разработка библиотеки генерации пользовательских данных
для frontend тестирования

№ п/п	Наименование работ	Срок вы- полнения
1	Обзор литературы по теме работы	01.04-10.04
2	Обзор аналогов	10.04-13.04
3	Архитектура библиотеки	14.04-16.04
4	Реализация библиотеки	27.04-25.04
5	Оценка производительности	26.04-30.04
6	Оформление пояснительной записки	30.04-06.05
7	Оформление иллюстративного материала	06.05-10.05
8	Оформление презентации	10.05-13.05
9	Предзащита	14.05

Студент _____ Б.В. Прохоров

Руководитель к.т.н., доцент _____ М.М. Заславский
(Уч. степень, уч. звание)

Консультант к.т.н., доцент _____ Б.Ю. Медошин
(Уч. степень, уч. звание)

РЕФЕРАТ

Пояснительная записка 68 стр., 20 рис., 3 табл., 46 ист.

ТЕСТИРОВАНИЕ, ГЕНЕРАЦИЯ ДАННЫХ, FRONTEND, WEB, NPM,
ПРЕФИКСНОЕ ДЕРЕВО.

Объектом исследования является тестирование пользовательского интерфейса.

Предметом исследования является генерация пользовательских данных для тестирования пользовательского интерфейса.

Цель работы: разработать библиотеку генерации пользовательских данных для тестирования пользовательского интерфейса.

В ходе выполнения этой работы были изучены решения в области генерации пользовательских данных для тестирования пользовательских интерфейсов. После этого были выполнены проектирование и разработка библиотеки, позволяющей генерировать пользовательские данные с помощью языка программирования TypeScript, сборщика Webpack, средой выполнения JavaScript-программ, с последующей регистрацией в NPM реестре. Было исследование времени выполнения работы реализованных функций.

ABSTRACT

Frontend testing plays a crucial role in ensuring the quality and functionality of web applications. It involves simulating user interactions and verifying the expected behavior of the application. A critical aspect of front-end testing is the use of user data to represent real-world scenarios. Creating user data manually can be time consuming and error prone. This research presents a novel library that addresses this challenge by automating the generation of user data for front-end testing.

СОДЕРЖАНИЕ

Введение	12
1 Обзор аналогов.....	14
1.1 Принципы отбора	14
1.1.1 ts-randomizer	14
1.1.2 zufall	15
1.1.3 @faker-js/faker	15
1.1.4 faux	15
1.1.5 factory.ts	16
1.2 Критерии сравнения	16
1.2.1 Уровень размера пакета (bundle size)	16
1.2.2 Уровень сложности внедрения.....	17
1.2.3 Уровень локализации	17
1.3 Сравнительный анализ.....	18
1.3.1 zufall	18
1.3.2 ts-randomizer	18
1.3.3 @faker-js/faker	19
1.3.4 faux	19
1.3.5 factory.ts	19
1.4 Выводы по итогам сравнения.....	20
2 Архитектура библиотечки	21
2.1 Назначение модулей.....	21
2.2 Взаимодействие модулей.....	22
3 Реализация библиотеки	26
3.1 generateUUID.....	26
3.1.1 Механизм работы	26
3.1.2 Оценка временной сложности.....	27
3.2 generateNumber	28
3.2.1 Механизм работы	28
3.2.2 Оценка временной сложности.....	29
3.3 generateString.....	30
3.3.1 Механизм работы	30
3.3.2 Оценка временной сложности.....	31
3.4 generatePerson.....	32
3.4.1 Механизм работы	32

3.4.2	Оценка временной сложности.....	33
3.5	Реализация префиксного дерева	34
3.5.1	Механизм работы метода getRandomFullString.....	35
3.5.2	Оценка временной сложности метода getRandomFullString	36
3.6	generateMeaningfulString	37
3.6.1	Механизм работы	37
3.6.2	Анализ временной сложности	38
4	Сборка библиотеки	39
4.1	Файл конфигурации TypeScript	39
4.2	Использование Webpack	39
4.2.1	Конфигурация для разработки	39
4.2.2	Конфигурация для релиза	40
5	Регистрация пакета в NPM реестре	41
5.1	Преимущества регистрации пакета	41
5.2	Процесс регистрации пакета	41
6	Внедрение пакета в проект	43
6.1	Установка и удаление пакета	43
6.2	Примеры использования.....	44
6.2.1	Генерация UUID	44
6.2.2	Генерация числа.....	44
6.2.3	Генерация строки.....	45
6.2.4	Генерация имён.....	46
6.2.5	Генерация осмысленной строки.....	46
7	Оценка производительности библиотеки.....	48
7.1	Принципы отбора	48
7.1.1	benchmark.....	48
7.1.2	pretty-hrtime	48
7.1.3	tinybench.....	49
7.1.4	perfy	49
7.1.5	micro-benchmark.....	51
7.2	Критерии сравнения	51
7.2.1	Еженедельное количество скачиваний (Popularity)	51
7.2.2	Количество звёзд у GitHub репозитория (Community Trust)	52
7.2.3	Частота обновления версии (Maintenance).....	53
7.3	Сравнительный анализ.....	53

7.3.1 benchmark.....	53
7.3.2 pretty-hrtime	53
7.3.3 tinybench.....	53
7.3.4 perfy	54
7.3.5 micro-benchmark.....	55
7.3 Выводы по итогам сравнения.....	55
7.4 Выбор метода решения	55
7.5 Описание метода решения.....	56
7.6 Проведение тестов производительности	57
7.7 Результаты тестов производительности.....	59
7.7.1 Генерация UUID	59
7.7.2 Генерация чисел.....	59
7.7.3 Генерация строк.....	60
7.7.4 Генерация имён.....	61
7.7.5 Генерация осмысленных строк	62
7.8 Выводы по результатам тестов производительности.....	63
Заключение	64
Список использованных источников.....	66

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ECMAScript (ES) - это спецификация, разработанная Ecma International для стандартизации языка программирования JavaScript.

JavaScript (JS) — мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией спецификации ES.

Движок JavaScript – интерпретатор JavaScript кода, написанный человеком, в машинный код, который может быть понят и выполнен компьютером.

Node.js – это программная платформа с открытым исходным кодом, построенная на движке V8 JavaScript от Google Chrome.

NPM (Node Package Manager) – это менеджер пакетов для JavaScript, входящий в состав среды выполнения Node.js.

Реестр NPM (NPM Registry) – это общедоступная база данных, где хранятся пакеты JavaScript, написанные с использованием NPM.

Библиотека — это набор программного обеспечения, предназначенный для повторного использования в различных проектах;

NPM пакет – это конкретная реализация библиотеки, опубликованная в реестре NPM. Он содержит код библиотеки, а также метаданные, такие как имя, описание, версия, зависимости, лицензия, под которой распространяется пакет;

Веб-приложение (web-приложение) – клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера.

Фронтенд (front-end, frontend) – пользовательский интерфейс веб-приложения.

Бандл (bundle) – это один сжатый файл или несколько файлов, который содержит все необходимые JavaScript-ресурсы (библиотеки, модули, утилиты и т.д.) для работы NPM-пакета.

CI/CD (continuous integration, continuous delivery или continuous deployment) — это комбинация непрерывной интеграции и непрерывного развертывания программного обеспечения в процессе разработки.

CI/CD пайплайн (CI/CD pipeline) — это процесс CI/CD, который объединяет разработку, тестирование и развёртывание приложений.

ВВЕДЕНИЕ

В современном развитии веб-приложений становится все более важным обеспечение их качества через тщательное тестирование [1]. Эффективное тестирование требует генерации разнообразных пользовательских данных для обеспечения наиболее полного охвата тестовых сценариев [2]. Написание некорректных тестов, может привести к катастрофическим последствиям, ущерб от которых может стоить много денег инвесторов.

С помощью библиотеки генерации пользовательских данных разработчики смогут генерировать различные виды псевдослучайных тестовых данных, таких как имена, фамилии, отчества пользователей, обычные и осмысленные строки, числа и UUID. Так, процесс создания данных для тестирования различных сценариев поведения веб-приложения будет автоматизирован.

Таким образом, разработка инструмента генерации пользовательских данных для проведения тестирования пользовательских интерфейсов представляет собой актуальную задачу.

Цель данной работы: разработать библиотеку генерации пользовательских данных для тестирования пользовательского интерфейса.

Задачи данной работы:

1. Выполнить обзор существующих решений в области генерации пользовательских данных.
2. Спроектировать и реализовать библиотеку для генерации пользовательских данных с помощью языка программирования TypeScript.
3. Реализовать сборку библиотеки с помощью бандлера Webpack.
4. Оформить библиотеку как NPM пакет и произвести его регистрацию в NPM реестре.
5. Исследовать разработанную библиотеку на предмет производительности и сравнить с аналогом.

Объектом исследования является тестирование пользовательского интерфейса.

Предметом исследования является генерация пользовательских данных для тестирования пользовательского интерфейса.

Практическая ценность работы: внедрение разработанного инструмента в проект позволит автоматизировать процесс создания данных для тестирования пользовательского интерфейса.

1 Обзор аналогов

1.1 Принципы отбора

В качестве аналогов библиотеки для генерации пользовательских данных были выбраны NPM-пакеты предоставляющие релевантные возможности – генерация случайных данных по заданному типу. NPM - это крупнейший в мире реестр программного обеспечения [3]. Разработчики открытого исходного кода со всех континентов используют NPM для обмена и заимствования пакетов, а многие организации используют NPM для управления частными разработками. Реестр NPM содержит пакеты, многие из которых также являются NPM-пакетами или содержат NPM-пакеты. Были ли отображены следующие NPM-пакеты: `ts-randomizer`, `zufall`, `@faker-js/faker`, `faux`, `factory.ts`.

Для поиска аналогов использовался сайт пакетного менеджера. Были использованы ключевые слова: `"randomizer"`, `"fake-data-generator"`, `"random"`, `"pseudo"`, `"test-data"`.

1.1.1 ts-randomizer

`ts-randomizer` – инструмент для создания случайных тестовых данных по заданным типам [4]. Он предназначен для минимизации фазы `arrange` (подготовки тестовых данных) в модульных тестах, чтобы максимизировать сопровождаемость, упрощая создание объектов, содержащих случайные тестовые данные. Таким образом, не нужно вручную создавать тестовые переменные в рамках фазы `arrange`. Случайность производимых данных обеспечивается за счёт использования функций библиотек `lodash` [5] и `uuid` [6]. К сожалению, пакет использует пользовательские трансформеры. Идея пользовательских трансформеров заключается в том, что TypeScript компилирует исходные файлы в JavaScript и вместе с этим формирует промежуточное представление (Abstract Syntax Tree) кода [7], которое можно изменять с помощью пользовательских плагинов. Сам TypeScript не предоставляет удобного способа использования пользовательских трансформаторов [8], что создаёт сложности при внедрении этого пакета в проект.

1.1.2 zufall

zufall – небольшая JavaScript/TypeScript библиотека [9] для генерации случайных тестовых данных по заданным типам, которая по сути является обёрткой над функцией `random()` из стандартной библиотеки JavaScript Math [10].

1.1.3 @faker-js/faker

@faker-js/faker – это библиотека [11], предоставляющий инструменты для генерации фиктивных данных в JavaScript-приложениях. Он включает в себя различные методы для создания случайных и реалистичных значений, таких как имена, адреса, электронные почты, числа и многое другое. Генерация случайных данных в пакете базируется на стандартных методах JavaScript, таких как `Math.random()`, но с использованием более высокоуровневого интерфейса для упрощения процесса, что обеспечивает удобство использования и более точный контроль над генерацией данных. Внедрение @faker-js/faker в проект довольно не является проблемой благодаря качественной документации. Пакет поддерживает различные языки и легко настраивается для соответствия конкретным потребностям приложения. Размер пакета варьируется в зависимости от выбора модулей и языковых настроек, однако полный пакет занимает 10.2 мб из-за хранения массивов фиктивных данных (имён, фамилий, адресов, номеров телефонов и т.д.), что может оказаться минусом при поиске леговесной библиотеки для генерации тестовых данных.

1.1.4 faux

faux – это NPM-пакет [12], предоставляющий инструменты для генерации случайных данных в JavaScript-приложениях. Он облегчает создание фиктивных данных для тестирования, прототипирования и разработки, предоставляя разнообразные функции для генерации случайных строк, чисел, дат и других типов данных. В основе генерации данных лежит алгоритмически контролируемая случайность, что обеспечивает воспроизводимость и предсказуемость результатов.

1.1.5 factory.ts

factory.ts - это NPM-пакет [13], предоставляющий инструменты для создания тестовых данных и объектов в TypeScript. Он позволяет легко определять синхронные или асинхронные фабрики объектов с предопределенными или случайными значениями для свойств. Пакет основан на концепции фабричного метода, что обеспечивает гибкость при создании разнообразных объектов для тестирования. Случайность генерируемых данных в factory.ts зависит от настроек и правил, определенных в фабриках. Пользователь может управлять уровнем случайности, предоставляя собственные генераторы данных или использовать встроенные средства для автоматической генерации значений.

1.2 Критерии сравнения

1.2.1 Уровень размера пакета (bundle size)

Размер NPM-пакета играет важную роль в процессе разработки модульных тестов. Большие пакеты могут замедлять запуск тестов, особенно при использовании CI/CD пайплайнов, а также тестовые данные, генерируемые большим пакетом, могут увеличить время загрузки приложения. При написании тестов важно получать быструю обратную связь о том, проходят ли они успешно или нет. Медленные тесты могут замедлить цикл разработки, а быстрые тесты обеспечивают более эффективный процесс разработки.

- Низкий уровень – размер пакета меньше 50 кб. Пакеты такого размера идеально подходят для производительности веб-приложений, так как они оказывают минимальное влияние на время загрузки и объем потребляемых данных [14]. Такие пакеты могут быть особенно полезны для генерации тестовых данных, так как они позволяют поддерживать высокую скорость работы тестовых сценариев без значительного увеличения размера сборки.
- Средний уровень – размер пакета находится в пределах от 50 кб до 200 кб. Пакеты в этом диапазоне по-прежнему могут быть использованы без существенного влияния на производительность, но разра-

ботчикам следует учитывать общий размер зависимостей, чтобы избежать задержек при загрузке.

- Высокий уровень – размер пакета больше 200 кб. Пакеты такого размера могут существенно повлиять на время загрузки и производительность, особенно если используются в клиентской части веб-приложения. Их использование оправдано, если они предоставляют значительное улучшение функциональности или необходимы для специфических требований проекта.

За размер пакета берётся минифицированный размер пакета без gZip сжатия, взятый из характеристик пакета, описанных в реестре NPM. Замеры скорости скачивания были сняты посредством сайта BundleFobia [15].

1.2.2 Уровень сложности внедрения

Данный критерий важен, поскольку показывает возможность ли начать использование аналога без внесения существенных изменений в проект и качество документации.

- Низкий уровень – аналог легко интегрируется в проект, не требует значительных изменений в существующем коде, а также имеется качественная документация.
- Средний уровень – интеграция аналога может потребовать некоторой доработки проекта или документация описывает не все сценарии использования библиотеки, что не представляет существенных трудностей.
- Высокий уровень – интеграция аналога потребует значительных изменений в проекте или документация отсутствует.

1.2.3 Уровень локализации

Данный критерий подчёркивает, разнообразие генерируемых данных. Он важен для проектов, нацеленных не только на англоязычную аудиторию. Что релевантно для русскоязычных компаний.

- Низкий уровень – пакет локализован только для английского языка.

- Средний уровень – пакет локализован для других языков, но среди них нет русского.
- Высокий уровень – пакет локализован для русского и английского языков.

1.3 Сравнительный анализ

В таблице 1 приведён итог сравнения аналогов.

1.3.1 zufall

Размер пакета среди аналогов самый маленький – 2.6 КБ. Достигается это использованием проприетарных функций и небольшой функциональностью. Такой размер даёт преимущество в скорости скачивания пакета из NPM реестра – 25 миллисекунд при 3G стандарте (50 КБ/с) и 1 миллисекунда при 4G стандарте (875 КБ/с) сотовой связи.

Он имеет подробную документацию с примерами использования и не добавляет дополнительных зависимостей, что даёт преимущество при внедрении пакета в проект.

Поскольку, пакет локализован только для английского языка, использование в проектах, нацеленных не на англоязычный сегмент рынка будет нерациональным.

1.3.2 ts-randomizer

Размер пакета средний – 160.4 КБ. При таком размере скорости скачивание пакета из NPM реестра равняется 1.11 секунд при 3G стандарте (50 КБ/с) и 63 миллисекунд при 4G стандарте (875 КБ/с) сотовой связи.

Он имеет достаточную документацию с примерами использования, однако принуждает к добавлению в проект дополнительных зависимостей в виде плагина для пользовательских трансформеров и его настройке, что создаёт достаточно высокие сложности при внедрении.

Пакет локализован только для английского языка, потому внедрение в русскоязычный проект будет нецелесообразным.

1.3.3 @faker-js/faker

Среди всех аналогов пакет имеет наибольший размер – 2.8 МБ. Такой большой размер обуславливается локализацией для многих языков включая русский. Что делает его скорость скачивания из NPM реестра самой низкой – 18.91 секунд при 3G стандарте (50 КБ/с) и 1.08 секунд при 4G стандарте (875 КБ/с) сотовой связи.

Он имеет самую подробную среди всех аналогов документацию и не требует к проекту подключения дополнительных зависимостей, что делает внедрение в проект максимально комфортным.

Пакет локализован для многих языков, включая русский, соответственно, это делает его совместимым со многими сегментами не англоязычного рынка.

1.3.4 faux

Размер пакет небольшой, всего 14.1 КБ. При таком размере скорости скачивание пакета из NPM реестра равняется 119 миллисекунд при 3G стандарте (50 КБ/с) и 7 миллисекунд при 4G стандарте (875 КБ/с) сотовой связи.

Он не требует дополнительных зависимостей к проекту, однако имеет неполноценную документацию, в которой не описаны все примеры использования.

Пакет локализован для нескольких языков, среди которых нет русского, что делает внедрение в русскоязычный проект нерациональным.

1.3.5 factory.ts

Пакет имеет небольшой размер – 12 КБ. При таком размере скорости скачивание пакета из NPM реестра равняется 62 миллисекунды при 3G стандарте (50 КБ/с) и 4 миллисекунд при 4G стандарте (875 КБ/с) сотовой связи.

Он имеет исчерпывающую документацию с примерами использования и не принуждает к добавлению в проект дополнительных зависимостей, что делает его удобным во внедрении.

Пакет локализован только для английского языка, что ограничивает количество проектов, в которых он может быть использован.

Таблица 1 – Сравнение аналогов по выделенным критериям

Критерий	zufall	ts-randomizer	@faker-js/faker	faux	factory.ts
Уровень размера	Низкий	Средний	Высокий	Низкий	Низкий
Уровень сложности внедрения	Низкий	Высокий	Низкий	Средний	Низкий
Уровень локализации	Низкий	Низкий	Высокий	Средний	Низкий

1.4 Выводы по итогам сравнения

После проведения сравнительного анализа аналогов для генерации пользовательских данных ясно, что каждый из них обладает своими преимуществами и недостатками.

Фаворитом среди кандидатов вышел пакет @faker-js/faker в силу наличия качественной и подробной документации, а также наличия локализации для русского языка. Однако минифицированный размер пакета в 2.8 МБ вызывает ряд проблем:

- Раздутый пакет может привести к замедлению работы приложения, особенно при многократном использовании @faker-js/faker.
- Множество пользователей @faker-js/faker не используют все предлагаемые функции, что приводит к загрузке и хранению на устройстве ненужного кода.

Таким образом, для качественного решения задачи генерации пользовательских данных необходимо разработать пакет, удовлетворяющий поставленным всем критериям.

2 Архитектура библиотеки

Библиотека спроектирована таким образом, чтобы быть модульной, расширяемой и простой в использовании.

2.1 Назначение модулей

Структура модулей изображена на рис. 1. Библиотека состоит из следующих модулей:

- «src/index.ts» – точка входа в библиотеку. Экспортирует все функции генерации данных из src/lib/test-utils.ts;
- «src/lib/constants.ts» экспортирует константы, используемые в библиотеке, такие как алфавит английского языка;
- «src/lib/test-utils.ts» реализует и экспортирует функции для генерации данных: generateUUID(), generateNumber(), generateString(), generatePerson(), generateMeaningfulString().
- «src/lib/types.ts» определяет и экспортирует типы данных, используемых в библиотеке.
- «src/lib/utils.ts» экспортирует вспомогательные функции, используемые другими модулями библиотеки.
- «src/lib/init-classes.ts» экспортирует классы для инициализации и обработки переданных параметров генерации тестовых данных.
- «src/lib/trie/trie.ts» экспортирует структуру данных префиксного дерева для эффективного поиска осмысленных строк.
- «src/lib/locales/en/dictionary.ts» экспортирует массивы английских слов, префиксов и суффиксов, которые используются для генерации осмысленных строк на английском языке.
- «src/lib/locales/ru/dictionary.ts» экспортирует массивы русских слов, приставок и суффиксов, которые используются для генерации осмысленных строк на русском языке.
- «src/lib/persons/en/persons.ts» экспортирует объект, который содержит русские имена, фамилии и отчества для генерации имен людей.

- «src/lib/persons/ru/persons.ts» экспортирует объект, который содержит английских имена, фамилии и отчества для генерации имен людей.

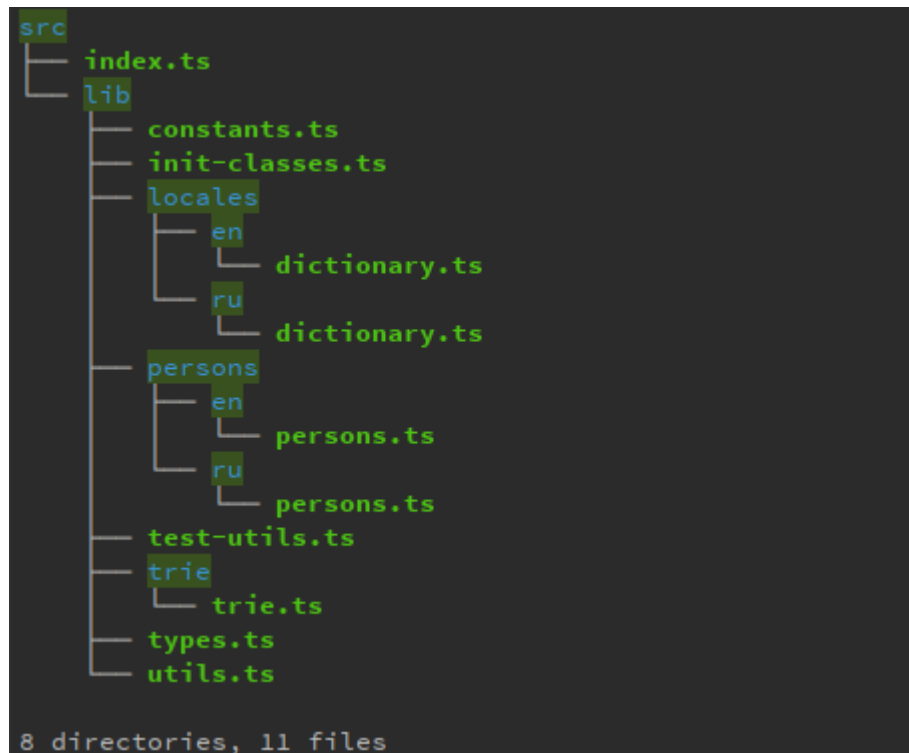


Рисунок 1 — Структура модулей

2.2 Взаимодействие модулей

Диаграммы последовательности взаимодействия и обзора взаимодействия модулей изображены на рис. 2. и на рис. 3. Эти диаграмма иллюстрирует, как различные модули взаимодействуют для генерации разнообразных тестовых данных, включая числа, строки, UUID, имена и осмысленные строки.

Участники взаимодействия:

- client (Клиент) – пользователь, который вызывает функции для генерации тестовых данных.
- index («src/index.ts») – точка входа в пакет, перенаправляет импорты в модуль test_utils.
- test_utils («src/lib/test-utils.ts») – основной модуль для генерации тестовых данных.

- constants («src/lib/constants.ts») – хранилище констант, таких как английский алфавит.
- init_classes («src/lib/init-classes.ts») – классы для инициализации и обработки переданных параметров генерации тестовых данных.
- utils («src/lib/utils.ts») – вспомогательные функции.
- trie («src/lib/trie/trie.ts») – модуль для работы с древовидной структурой данных (префиксным деревом) для генерации осмысленных строк.
- en_persons («src/lib/persons/en/persons.ts») – модуль, содержащий массивы имен, фамилий и отчеств на английском языке.
- ru_persons («src/lib/persons/ru/persons.ts») – модуль, содержащий массивы имен, фамилий и отчеств на русском языке.
- en_dictionary («src/lib/locales/en/dictionary.ts») – модуль, содержащий объект с префиксами, суффиксами и словами на английском языке.
- ru_dictionary («src/lib/locales/ru/dictionary.ts») – модуль, содержащий объект с префиксами, суффиксами и словами на русском языке.
- crypto (crypto) – криптографический Web API [16], предоставляемый, в зависимости от среды выполнения, либо браузером, либо Node.js [17], позволяющий генерировать случайные числа в доступном диапазоне с помощью getRandomValues(buffer).

Процессы взаимодействия:

1. Клиент импортирует нужную функцию генерации данных (generateNumber, generateString, generateUUID, generatePerson, generateMeaningfulString) из index.
2. index перенаправляет импорты в test_utils
3. Модуль test_utils обрабатывает запросы на генерацию данных:
 - а) generateNumber(opts) создает объект CNumberOpts в init_classes. В случае ошибки (max меньше min) - возвращает клиенту сообщение об ошибке. В случае корректных значений - делегиру-

ет генерацию случайного числа в `utils`. Возвращает клиенту сгенерированное число.

- b) `generateString(opts)` создает объект `CStringOpts` в `init_classes` с опциями генерации случайной строки. Импортирует `ENGLISH_ALPHABET` из `constants` (если не указано другое). Генерирует случайные значения через `crypto`. Выбирает символы из набора символов с помощью `init_classes`. Возвращает клиенту сгенерированную строку.
- c) `generateUUID()` выбирает метод генерации `UUID` в зависимости от переданного булева аргумента `strictRandom`. При значении истина - использует генератор `UUID` поставляемый пакетом `uuid`, иначе применяет собственную логику генерации `UUID` с помощью счётчика, и возвращает клиенту сгенерированный `UUID`.
- d) `generatePerson(opts)` создает объект `CPartNameOpts` в `init_classes` с опциями генерации имени. В зависимости от языка (`ru/en`) запрашивает объект, который содержит массивы имён, фамилий и отчеств из соответствующего модуля (`ru_persons` или `en_persons`). Возвращает клиенту сгенерированное имя.
- e) `generateMeaningfulString(opts)` создает объект класса `CMeaningfulStringOpts` с опциями генерации осмысленной строки. В зависимости от языка (`ru/en`) использует префиксы, слова и суффиксы из `en_dictionary` или `ru_dictionary` и запрашивает случайную осмысленную строку из модуля `trie.ts` посредством метода `getRandomFullString()`. Возвращает клиенту сгенерированную осмысленную строку.

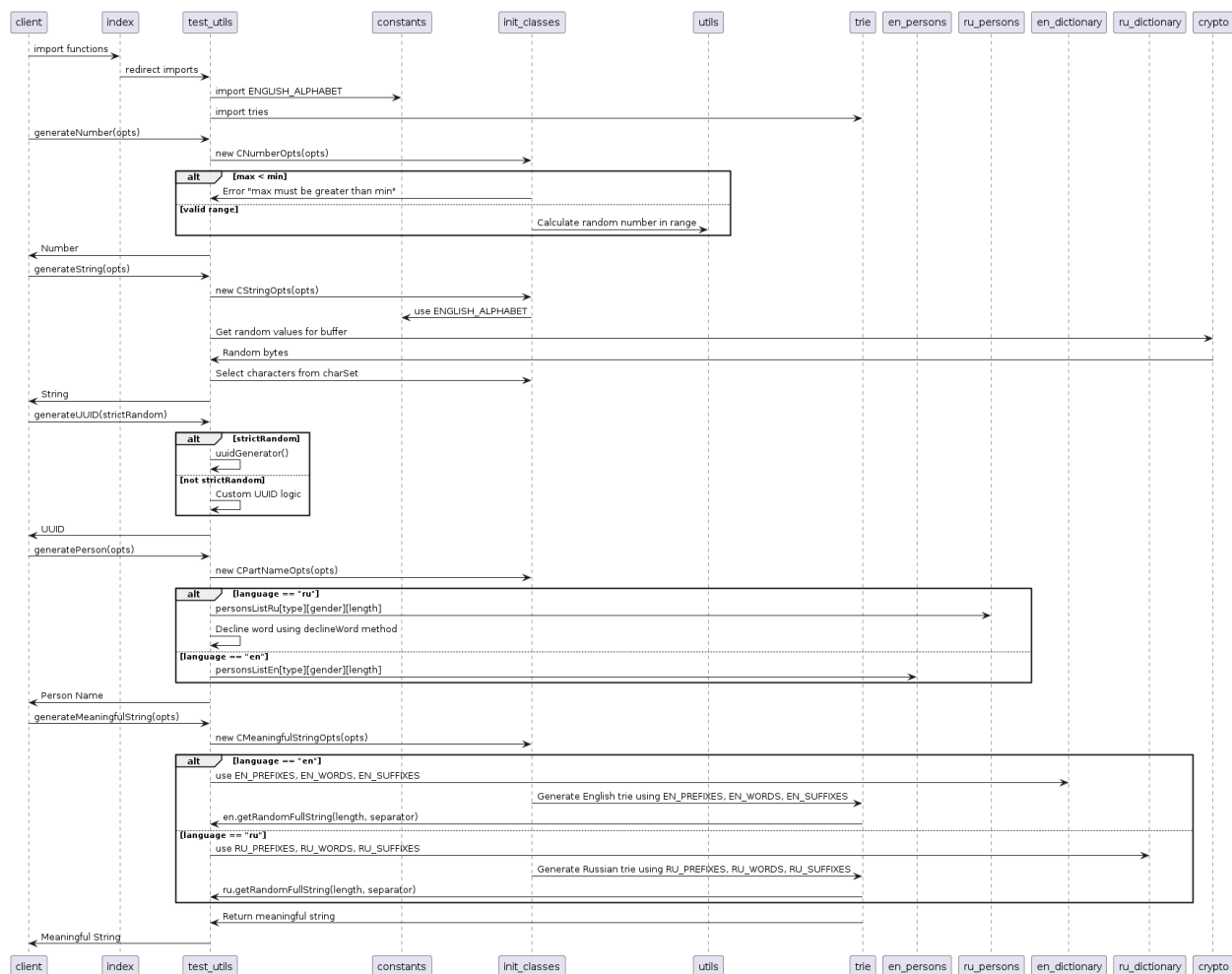


Рисунок 2 — Диаграмма последовательности взаимодействия модулей

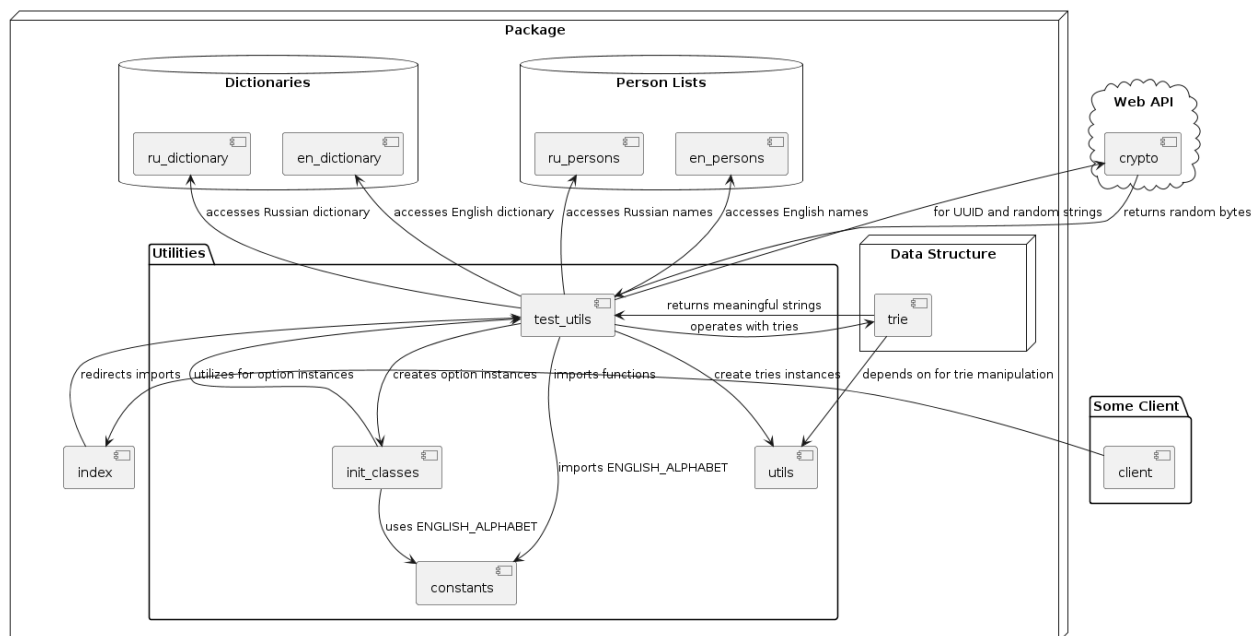


Рисунок 3 — Диаграмма обзора взаимодействия модулей

3 Реализация библиотеки

Библиотека предоставляет возможность импортировать пользователю следующие функции для генерации тестовых данных: `generateUUID()`, `generateNumber()`, `generateString()`, `generatePerson()`, `generateMeaningfulString()`.

3.1 generateUUID

Функция служит для генерации уникальных идентификаторов (UUID). В случае нестрогого рандома используется сквозной счетчик, преобразованный в строку формата шестнадцатеричного числа, который затем добавляется к базовому UUID. При достижении определенного числа счетчик сбрасывается.

3.1.1 Механизм работы

Диаграмма последовательности алгоритма работы функции изображена на рис. 4.

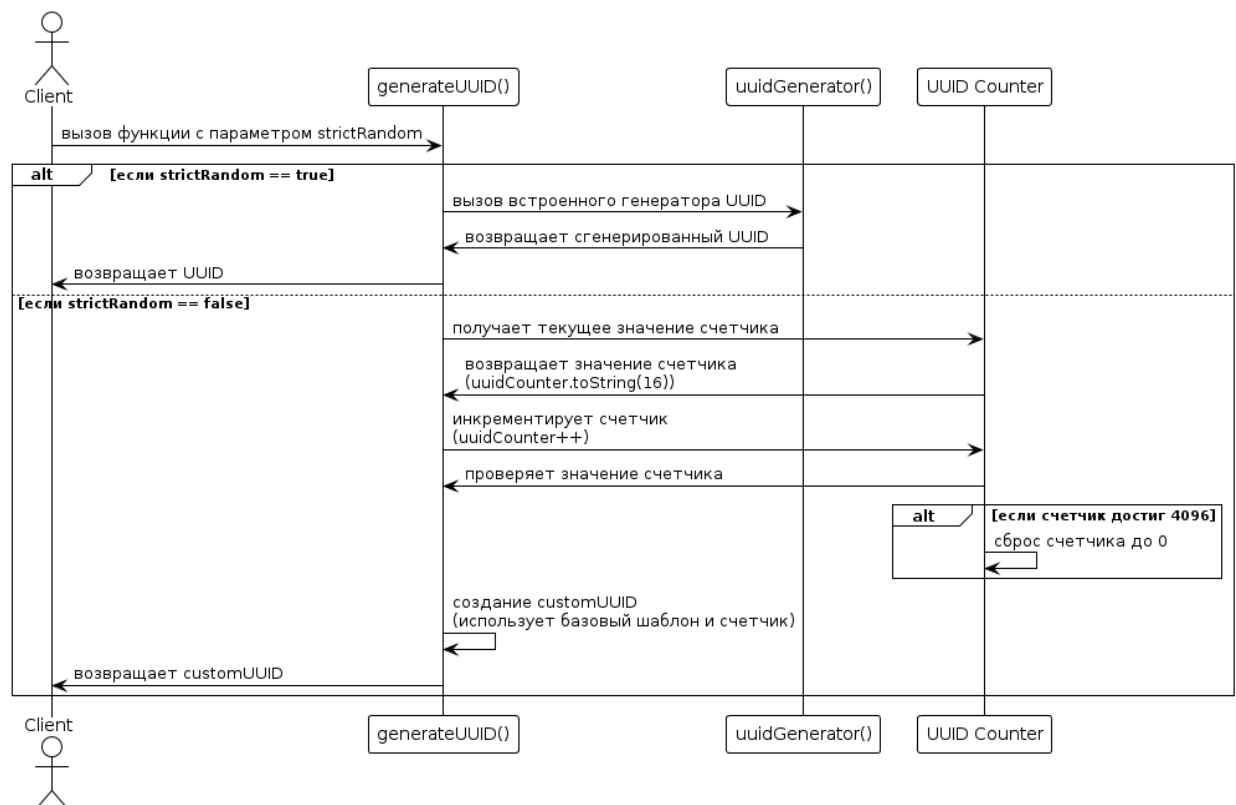


Рисунок 4 — Диаграмма последовательности алгоритма работы функции `generateUUID()`

Функция имеет два основных потока выполнения в зависимости от параметра строгости генерации.

Если параметр `strictRandom` установлен в `true`, функция `generateUUID` делегирует генерацию UUID стандартной библиотеке `uuid`.

Если параметр `strictRandom` установлен в `false`, функция `generateUUID` генерирует UUID путем комбинирования статической последовательности символов с текущим значением `uuidCounter`, преобразованного в шестнадцатеричную строку. Счетчик инкрементируется на каждый вызов. Если счетчик достигает 4096, он сбрасывается до 0.

3.1.2 Оценка временной сложности

Функция `generateUUID` представлена в двух вариантах выполнения в зависимости от значения параметра `strictRandom`.

Когда `strictRandom` установлен в `true` функция `generateUUID` напрямую вызывает функцию `randomUUID()` криптографического Web API, которая генерирует UUID с помощью криптографического генератора псевдослучайных чисел, что означает проход по каждой позиции генерации значения, так что время этой части $O(32)$, что все равно сводится к $O(1)$, так как это константное время не зависящее от входных данных. Следовательно, сложность по времени для `generateUUID` будет $O(1)$, что означает константное время выполнения независимо от входных данных.

Когда `strictRandom` установлен в `false` функция самостоятельно сгенерирует значение UUID, используя базовый шаблон и счетчик:

- Получение текущего значения счетчика $O(1)$
- Инкремент счетчика $O(1)$
- Проверка значения счетчика и сброс при необходимости $O(1)$
- Форматирование счетчика и создание пользовательской UUID строки $O(1)$

Так как все операции, которые выполняются в функции `generateUUID` при `strictRandom` равном `false`, имеют постоянную сложность, общая временная сложность этой функции также будет $O(1)$.

В результате, не зависимо от значения параметра `strictRandom`, временная сложность функции `generateUUID` будет константной, или $O(1)$.

3.2 generateNumber

Функция, возвращающая случайное число в заданном интервале между минимумом и максимумом. В случае если максимум меньше минимума, генерируется ошибка.

3.2.1 Механизм работы

Диаграмма последовательности алгоритма работы функции изображена на рис. 5.

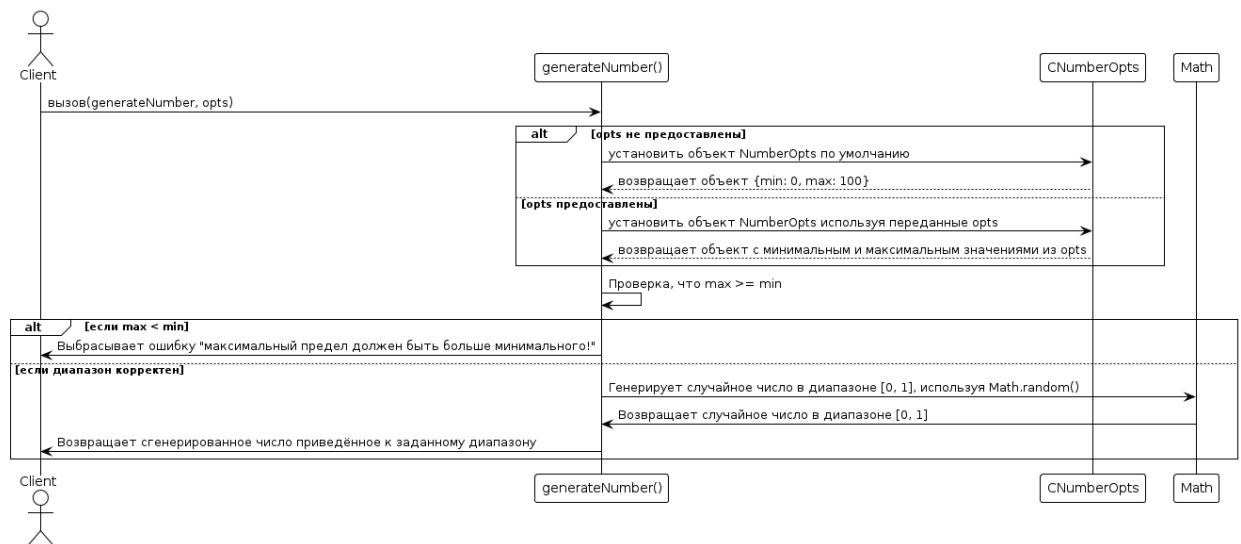


Рисунок 5 — Диаграмма последовательности алгоритма работы функции `generateNumber()`

Клиент вызывает функцию `generateNumber()` с параметром `opts` типа `NumberOpts`, которые определяют диапазон значений для генерируемого числа. Если параметр `opts` не предоставлен, функция `GenNumber` обращается к объекту `CNumberOpts` с запросом на установление объекта опций по умолчанию. Объект `CNumberOpts` затем возвращает стандартный объект опций `{min: 0, max: 100}`. Если параметр `opts` предоставлен, функция `GenNumber` обращается к объекту `CNumberOpts` с запросом на установление объекта опций, используя предоставленные `opts`. `CNumberOpts` затем возвращает объект опций, который включает минимальное и максимальное значения, указанные

в `opts`. Функция `generateNumber()` проверяет, что максимальное значение (`max`) больше или равно минимальному значению (`min`). Если максимальное значение меньше минимального, функция возвращает клиенту ошибку "the maximum limit must be greater than the minimum!". Если предыдущая проверка успешно пройдена (максимальное значение больше или равно минимальному), функция `GenNumber` обращается к объекту `Math` с запросом на генерацию случайного числа в диапазоне от 0 до 1, используя `Math.random()`. Полученное от `Math` случайное число в диапазоне от 0 до 1 затем преобразуется функцией `GenNumber` к заданному диапазону значений (исходя из минимального и максимального значений, определённых в объекте опций). После преобразования случайного числа к заданному диапазону, сгенерированное число возвращается клиенту.

3.2.2 Оценка временной сложности

Создание экземпляра `CNumberOpts` включает в себя копирование предоставленных опций в новый экземпляр `CNumberOpts`. В конструкторе объекта используется `Object.assign()`, который копирует каждое свойство в цикле, его сложность составит $O(n)$, где n - количество свойств для копирования. Поскольку количество свойств постоянно и невелико (два свойства: `min`, `max`), этот шаг имеет постоянную временную сложность $O(2)$, что сводится к $O(1)$.

Проверка того, что `max` больше `min` является сравнением с постоянным временем, поэтому его сложность равна $O(1)$. Вычисление диапазона - это операция вычитания и сложения, обе операции с постоянным временем и сложностью $O(1)$.

Генерация случайного числа с помощью `Math.random()` также считается операцией с постоянным временем независимо от размера диапазона, поэтому ее сложность равна $O(1)$.

Учитывая все вышеперечисленное, каждый шаг `generateNumber()` выполняется за постоянное время $O(1)$, следовательно, общая временная слож-

ность функции `generateNumber()` равна $O(1)$, то есть она генерирует случайное число за постоянное время независимо от размера входных данных.

3.3 generateString

Функция возвращает строку случайных символов заданной длины из набора символов, определенного пользователем, или из английского алфавита по умолчанию

3.3.1 Механизм работы

Диаграмма последовательности алгоритма работы функции изображена на рис. 6.

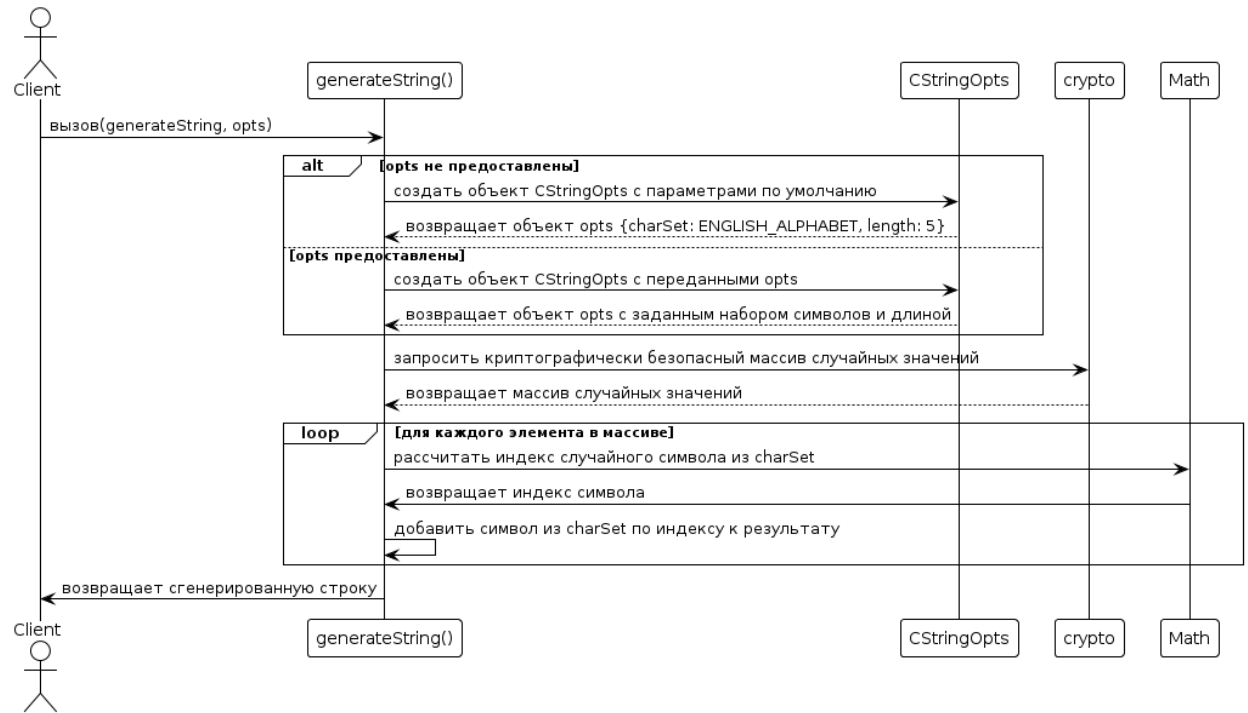


Рисунок 6 — Диаграмма последовательности алгоритма работы функции `generateString()`

Клиент вызывает функцию `generateString()` с опциональным параметром `opts`. Если параметры `opts` не предоставлены, создаётся объект `CStringOpts` с значениями по умолчанию (набор символов - `ENGLISH_ALPHABET`, длина строки - 5). В случае предоставления параметров `opts`, создаётся объект `CStringOpts` с указанными в `opts` параметрами. Функция `generateString()` запрашивает у Web API криптографически безопасный массив случайных значений, длина которого равна запрошенной длине строки. Для каждого эле-

мента полученного массива случайных значений функция рассчитывает индекс случайного символа из заданного набора символов (`charSet`) с использованием `Math.floor()`. Полученный индекс используется для выбора символа из `charSet`, который затем добавляется к результирующей строке. Возвращается сгенерированная строка заданной длины клиенту.

3.3.2 Оценка временной сложности

Создание экземпляра `CStringOpts` включает в себя копирование предоставленных опций в новый экземпляр `CStringOpts`. В конструкторе объекта используется `Object.assign()`, который копирует каждое свойство в цикле, его сложность составит $O(n)$, где n - количество свойств для копирования. Поскольку количество свойств постоянно и невелико (два свойства: `charSet`, `length`), этот шаг имеет постоянную временную сложность $O(2)$, что сводится к $O(1)$.

Создание криптографически безопасного массива случайных чисел через Web API не зависит напрямую от данных, присутствующих в `charSet`, но зависит от длины строки `length`. Однако, поскольку операция не обрабатывает каждый символ набора отдельно и выполняется внутренне оптимизированно, следовательно, можно считать этот шаг выполняющимся за константное время $O(1)$.

На стадии генерации строки выполняется основная работа - для каждого элемента в массиве случайных чисел выбирается символ из `charSet` и добавляется в итоговую строку. Поскольку количество итераций цикла прямо зависит от требуемой длины строки `length`, временная сложность этой части алгоритма будет $O(n)$, где n - это желаемая длина строки.

Итак, основываясь на анализе приведенного кода, можно заключить, что общая временная сложность функции `generateString()` составляет $O(n)$, где n - это длина генерируемой строки. Это означает, что временная сложность алгоритма линейно зависит от размера заданной длины строки.

3.4 generatePerson

Функция генерирует случайные имя, фамилию или отчество в зависимости от предоставленных параметров, таких как длина, пол и желаемый падеж (для русского языка).

3.4.1 Механизм работы

Диаграмма последовательности алгоритма работы функции изображена на рис. 7.

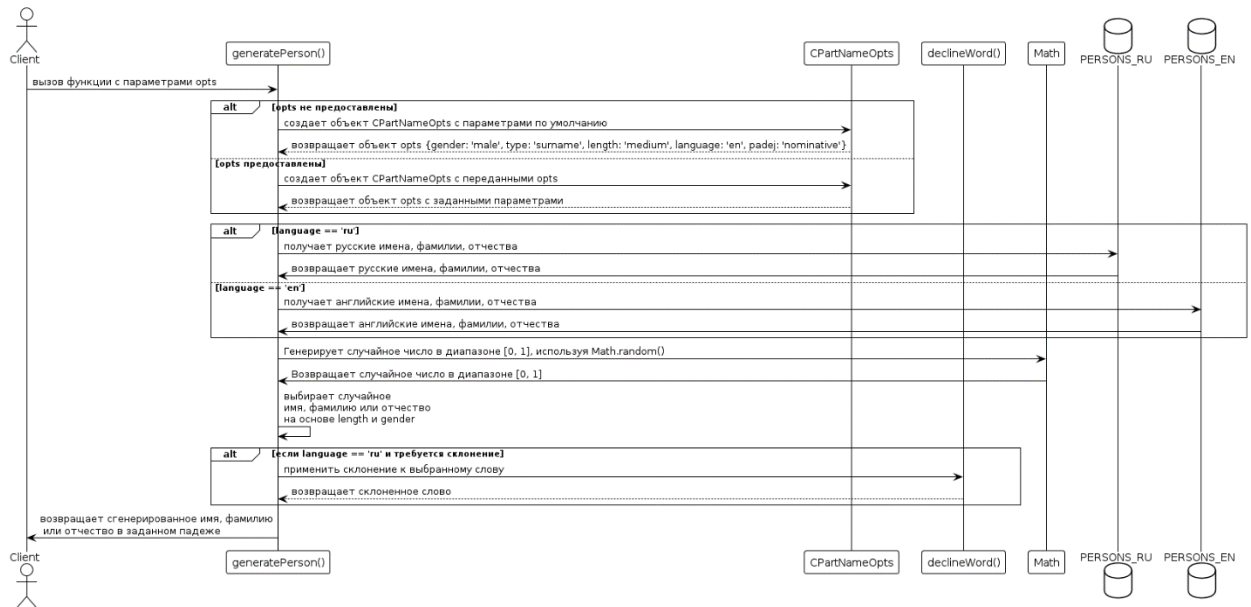


Рисунок 7 — Диаграмма последовательности алгоритма работы функции generatePerson()

Функция generatePerson() принимает объект opts в качестве аргумента, который является объектом типа PartNameOpts. Этот объект может содержать пользовательские настройки для выбора пола (gender), типа имени (type), длины имени (length), языка (language) и падежа (padej), если необходимо.

Внутри функции generatePerson(), создается новый объект CPartNameOpts с использованием предоставленных параметров opts. Если opts не предоставлены, используются значения параметров по умолчанию.

Затем, в зависимости от выбранного языка (ru или en), берётся соответствующий объект, который содержит массивы имён, фамилий и отчеств в за-

зависимости от типа (name, surname, patronymic), пола (male, female) и размера (small, medium, large, extra_large).

Из полученного списка данных случайным образом с помощью `Math.random()` выбирается слово, которое затем возвращается как результат работы функции.

Для русского языка (ru) может быть вызвана дополнительная функция `declineWord()`, которая склоняет выбранное слово в соответствии с заданным падежом (padej).

В конце, функция `generatePerson()` возвращает итоговую строку — это выбранное имя, фамилия или отчество на выбранном языке, уже склоненное, если это было требуется.

3.4.2 Оценка временной сложности

Создание экземпляра `CPartNameOpts` включает в себя копирование предоставленных опций в новый экземпляр `CPartNameOpts`. В конструкторе объекта используется `Object.assign()`, который копирует каждое свойство в цикле, его сложность составит $O(n)$, где n - количество свойств для копирования. Поскольку количество свойств постоянно и невелико (пять свойств: gender, type, length, language, padej), этот шаг имеет постоянную временную сложность $O(5)$, что сводится к $O(1)$.

Выбор объекта, содержащего массивы имен, фамилий и отчеств также является операцией с постоянным временем выполнения, так как число свойств в этом объекте не зависит от введенных данных и остается неизменным.

Выбор случайного элемента из массива — это операция с постоянным временем выполнения.

Если требуется склонение слова (только для русской версии), то функция `declineWord()` также имеет постоянное время выполнения, поскольку по сути она изменяет поданную строку, в зависимости от заданного падежа.

3.5 Реализация префиксного дерева

Префиксное дерево (Trie) состоит из узлов (TrieNode), каждый из которых соответствует одному символу строки [18]. Корень дерева не содержит символа и является начальной точкой для всех строк, хранящихся в Trie. Путь от корня до листа (или внутреннего узла, который помечен как конец слова) соответствует строке, хранящейся в Trie.

В реализации Trie используются два основных класса: Trie и TrieNode.

TrieNode инкапсулирует узел Trie, храня в себе карту дочерних узлов и флаг, обозначающий конец слова.

Диаграмма отношения классов Trie и TrieNode изображена на рис. 8.

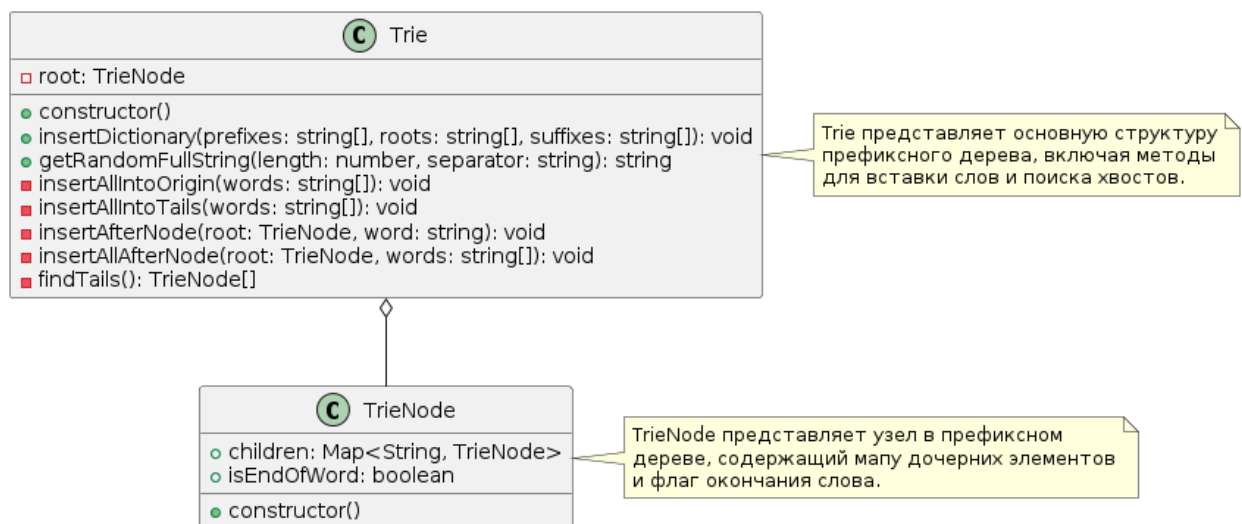


Рисунок 8 — Диаграмма отношения классов Trie и TrieNode

Класс Trie содержит следующие публичные методы:

`insertDictionary(prefixes, roots, suffixes)` – метод, который принимает массивы префиксов, корневых частей и суффиксов, после чего добавляет все слова в дерево, используя приватный метод `insertAllAfterNode`.

`getRandomFullString(length, separator)` – метод, который генерирует случайную осмысленную строку заданной длины из элементов дерева, используя алгоритм обхода в глубину (DFS) на основе стека [19]. Метод позволяет разделять слова заданным разделителем.

Класс Trie содержит следующие приватные методы:

`insertAllIntoOrigin(words)` – метод добавляет строки в Trie, начиная с корневого узла.

`insertAllIntoTails(words)` – метод добавляет строки в Trie, начиная с "хвостовых" узлов (конечных узлов слов).

`insertAfterNode(root, word)` – метод добавляет слово, начиная с указанного узла root, перебирая символы слова и добавляя узлы в Trie.

`findTails()` – метод находит все "хвосты" дерева (узлы) с помощью стека, которые помечены как конец слова.

3.5.1 Механизм работы метода `getRandomFullString`

Метод `getRandomFullString()` использует стек для итеративного обхода Trie в стиле поиска в глубину. При этом обеспечивается возможность случайного выбора путей для генерации строки, благодаря перемешиванию дочерних узлов.

Создается пустой стек и пустой массив слов. Добавляется в стек корневой узел и пустая строка. Пока стек не пуст длина строки с разделителем не соответствует требуемой, извлекается последний элемент из стека. Если узел является концом слова, оно добавляется в массив слов. Получаются все дочерние узлы узла, и они перемешиваются с помощью функции `shuffleArray()` реализующий алгоритм Фишера-Йетса [20]. Добавляются все дочерние узлы в стек. Объединяются все слова в одну строку с использованием разделителя и обрезается до заданной длины. Возвращается сгенерированная строка.

Диаграмма алгоритма работы метода `getRandomFullString()` изображена на рис. 9.



Рисунок 9 — Диаграмма алгоритма работы метода getRandomFullString()

3.5.2 Оценка временной сложности метода getRandomFullString

Поскольку метод использует поиск в глубину, в худшем случае он посетит каждый узел в дереве, что приведет к временной сложности $O(n)$, где n – общее количество узлов, исходя из структуры префиксного дерева, n также является максимальной длиной слова в дереве.

Вызов функции `shuffleArray` для перемешивания массива детей каждого узла требует $O(k)$, где k – среднее количество детей у узла, исходя из

структуры префиксного дерева, число k равняется $n-1$. Поскольку перемешивание происходит для каждого узла, общая временная сложность на этом этапе будет $O(n^{n-1})$.

Поскольку в конце алгоритма требуется объединить строки с использованием разделителя используется строковый метод `join()`, который у которого линейная временная сложность, что даёт $O(l)$, где l – требуемая длина строки.

Исходя из этих рассуждений, общая временная сложность метода `getRandomFullString()` составит $O(l + n^{n-1})$, где n – максимальная длина слова в дереве, а l – требуемая длина строки.

3.6 generateMeaningfulString

Основываясь на предварительно подготовленных префиксных деревьях (tries) для английского и русского языков, функция генерирует случайную осмысленную строку заданной длины, которая может включать в себя слова или их комбинации с разделителями.

3.6.1 Механизм работы

Функция `generateMeaningfulString(opts)` принимает на вход объект `opts`, который может содержать параметры `length` (длина строки), `separator` (разделитель) и `language` (язык).

Создается новый экземпляр класса `CMeaningfulStringOpts`, используя переданные параметры. Значения по умолчанию для длины – 5, для разделителя – пустая строка, для языка – английский (`en`).

В зависимости от указанного языка (`language`), функция определяет, какое дерево использовать. Для каждого языка предварительно подготовлено префиксное дерево с соответствующими словами, префиксами и суффиксами.

Используя метод `getRandomFullString(length, separator)` выбранного дерева, функция генерирует строку, которая имеет заданную длину (`length`), состоит из слов, префиксов и/или суффиксов, рандомизированных и объединенных в соответствии с логикой префиксного дерева, использует заданный разделитель (`separator`) между элементами строки.

Сгенерированная строка возвращается в качестве результата функции.

Диаграмма последовательности алгоритма работы функции изображена на рис. 10.

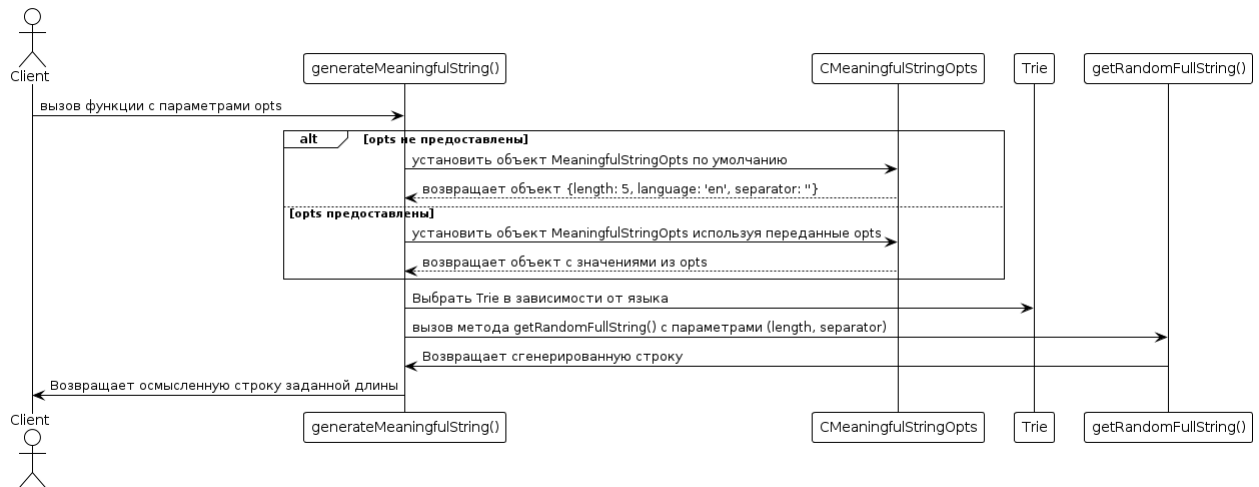


Рисунок 10 — Диаграмма последовательности алгоритма работы функции `generateMeaningfulString()`

3.6.2 Анализ временной сложности

Создание экземпляра `CMeaningfulStringOpts` включает в себя копирование предоставленных опций в новый экземпляр `CMeaningfulStringOpts`. В конструкторе объекта используется `Object.assign()`, который копирует каждое свойство в цикле, его сложность составит $O(n)$, где n - количество свойств для копирования. Поскольку количество свойств постоянно и невелико (три свойства: `length`, `language`, `separator`), этот шаг имеет постоянную временную сложность $O(3)$, что сводится к $O(1)$.

Выбор дерева в зависимости от языка также является операцией с постоянным временем выполнения, поскольку реализован с помощью конструкции «switch» [21].

Исходя из этих рассуждений, общая временная сложность функции `generateMeaningfulString()` зависит от временной сложности метода `getRandomFullString()`, следовательно, составляет $O(l + n^{n-1})$, где n – максимальная длина слова в дереве, а l – требуемая длина строки.

4 Сборка библиотеки

Проект имеет четко настроенный процесс сборки, обеспечивающий удобную разработку и подготовку кода к распространению через NPM.

4.1 Файл конфигурации TypeScript

Настройки TypeScript определены в файле «tsconfig.json». Конфигурация указывает на:

1. Компиляцию кода TypeScript в JavaScript с целевым стандартом ECMAScript 5 ("target": "es5").
2. Генерацию деклараций типов («.d.ts» файлы) ("declaration": true).
3. Использование CommonJS модулей ("module": "commonjs"), что делает библиотеку совместимой с Node.js и другими средами, которые поддерживают этот формат модулей.
4. Включение дополнительных проверок строгости и безопасности типов как «strictNullChecks».

4.2 Использование Webpack

Проект использует Webpack [22] для упаковки TypeScript-кода в бандлы. Для режимов разработки и релиза реализованы две конфигурации.

В обеих конфигурациях:

1. Указываются расширения файлов для разрешения импорта «.ts» и «.js».
2. Прописываются правила для обработки TypeScript файлов через ts-loader, что позволяет Webpack компилировать TypeScript напрямую в JavaScript, минуя необходимость предварительной компиляции.
3. Файл «./src/index.ts» в качестве точки входа.
4. Настройки модулей для обработки файлов .ts с использованием ts-loader.

4.2.1 Конфигурация для разработки

Настройки Webpack определены в «webpack.config.dev.js». Настройки для режима разработки включают:

1. Режим `development`, который не минифицирует выходной код и сохраняет исходные карты (`devtool: 'inline-source-map'`) для удобства отладки [23].
2. Вывод в папку «`dist`» с именем файла, указанным в файле «`projectInfo.js`», и экспорт в глобальную переменную, также указанную в «`projectInfo.js`».

4.2.2 Конфигурация для релиза

Настройки Webpack определены в файле «`webpack.config.npm.js`».

Настройки для сборки релизной версии:

1. Режим `production` автоматически минифицирует код [24].
2. Выходной файл помещается в папку «`lib`», что соответствует тому, как структурированы пакеты в NPM реестре.

5 Регистрация пакета в NPM реестре

5.1 Преимущества регистрации пакета

Регистрация пакета в NPM реестре делает его доступным для любого разработчика в мире. Разработчики могут просто использовать команду `npm install`, чтобы установить пакет в свои проекты.

NPM реестр предоставляет инструменты для управления версиями пакета, он даёт возможность публиковать обновления, исправления и новые версии пакета, сохраняя при этом предыдущие версии.

NPM реестр един для всех пакетных менеджеров, поэтому вне зависимости от используемого пакетного менеджера (NPM, YARN, PNPM, BUN) пакет можно будет внедрить в проект [25].

NPM автоматически управляет зависимостями пакетов, упрощая разработчикам процесс включения и обновления пакетов в их проектах.

Публикация пакета повышает вашу профессиональную видимость как разработчика и может привести к обратной связи и вкладам в проект со стороны сообщества.

NPM проводит базовые проверки безопасности пакетов, что помогает обеспечить безопасность экосистемы [26].

5.2 Процесс регистрации пакета

С помощью команды `npm init` для проекта был создан файл «`package.json`», содержащий описание разрабатываемого проекта и информацию об используемых зависимостях. В нём были указаны свойства, характеризующие пакет: название, версия, описание, точка входа, путь до декларации типов, скрипты для разработки, ссылка на GitHub репозиторий, ключевые слова для поиска, имя автора, лицензия распространения, ссылка на отправку найденных багов, ссылка на домашнюю страницу, зависимости при разработке.

Для публикации созданного пакета разработчик должен быть авторизован в системе NPM, для была выполнена команду `npm login`.

Публикация пакета была осуществлена с помощью команды командой `npm publish`.

6 Внедрение пакета в проект

6.1 Установка и удаление пакета

Установка пакета осуществляется посредством выполнения команды `npm install test-data-utils` в корне проекта.

Пакет устанавливается в директорию «`node_modules`».

При необходимости автоматически создать запись в «`package.json`» в раздел зависимостей добавляется флаг «`--save`»: `npm install --save test-data-utils`.

При необходимости автоматически создать запись в «`package.json`» в раздел зависимостей на этапе разработки (а не поставки заказчику) добавляется флаг «`--save-dev`»: `npm install --save-dev test-data-utils`.

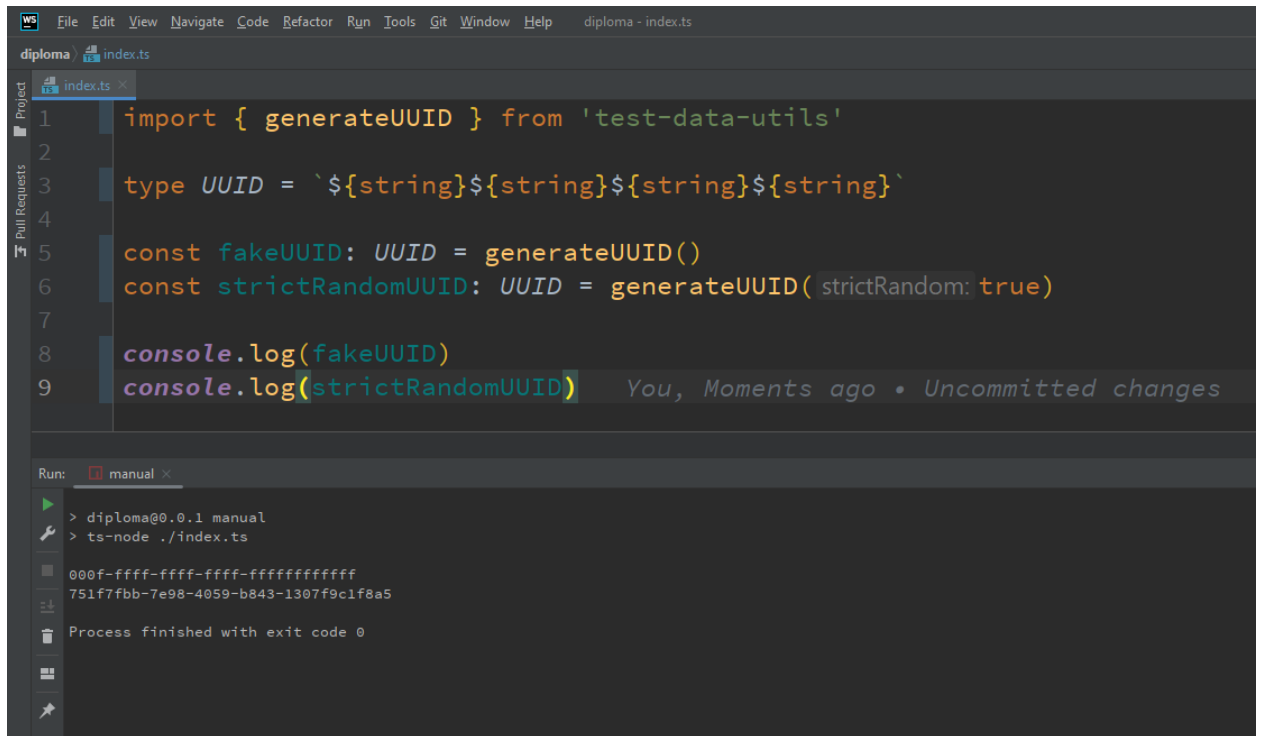
При необходимости обновления пакета можно воспользоваться командой `npm update test-data-utils`.

Удаление пакета осуществляется посредством выполнения команды `npm remove test-data-utils`.

6.2 Примеры использования

6.2.1 Генерация UUID

Для генерации уникального идентификатора нужно импортировать из пакета `test-data-utils` и использовать функцию `generateUUID()`, передав ей опциональный флаг, обозначающий строго рандомизированную генерацию при необходимости. На рис. 11 продемонстрирован пример использования функции.



```
1 import { generateUUID } from 'test-data-utils'
2
3 type UUID = `${string}${string}${string}${string}`
4
5 const fakeUUID: UUID = generateUUID()
6 const strictRandomUUID: UUID = generateUUID( strictRandom: true)
7
8 console.log(fakeUUID)
9 console.log(strictRandomUUID)
```

Run: manual ×

```
> diploma@0.0.1 manual
> ts-node ./index.ts
000f-ffff-ffff-ffff-ffffffffffff
751f7fbb-7e98-4059-b843-1307f9c1f8a5
Process finished with exit code 0
```

Рисунок 11 — Демонстрация использования функции `generateUUID()`

6.2.2 Генерация числа

Для генерации случайного числа нужно импортировать из пакета `test-data-utils` и использовать функцию `generateNumber()`, передав ей требуемые характеристики генерируемого объекта. На рис. 12 продемонстрирован пример использования функции.

The screenshot shows a VS Code editor with a file named `index.ts`. The code defines two constants: `fakeNumber` and `fakeNumberWithRange`, both using the `generateNumber` function from the `test-data-utils` package. The `fakeNumberWithRange` call includes options for a minimum value of -999 and a maximum value of 999. Both values are logged to the console. The Run and Debug console shows the output of the script, displaying the generated numbers 1 and 966. The status bar indicates 'Uncommitted changes'.

```
1 import { generateNumber } from 'test-data-utils'
2
3 const fakeNumber: number = generateNumber()
4
5 const fakeNumberWithRange: number = generateNumber( opts: {
6   min: -999,
7   max: 999,
8 })
9
10 console.log(fakeNumber)
11 console.log(fakeNumberWithRange)
```

Run: manual x

```
> diploma@0.0.1 manual
> ts-node ./index.ts
```

1
966

Process finished with exit code 0

Рисунок 12 — Демонстрация использования функции `generateNumber()`

6.2.3 Генерация строки

Для генерации случайной строки нужно импортировать из пакета `test-data-utils` и использовать функцию `generateString()`, передав ей требуемые характеристики генерируемого объекта. На рис. 13 продемонстрирован пример использования функции.

The screenshot shows a VS Code editor with a file named `index.ts`. The code imports the `generateString` function from the `test-data-utils` package and uses it to generate a string `stubString` with a length of 36 and a character set of 'abc'. The string and its length are logged to the console. The Run and Debug console shows the output of the script, displaying a 36-character string and the number 36. The status bar indicates 'Uncommitted changes'.

```
1 import { generateString } from 'test-data-utils'
2
3 const stubString: string = generateString( opts: {
4   length: 36,
5   charSet: 'abc'
6 })
7
8 console.log(stubString)
9 console.log(stubString.length)
```

Run: manual x

```
> diploma@0.0.1 manual
> ts-node ./index.ts
```

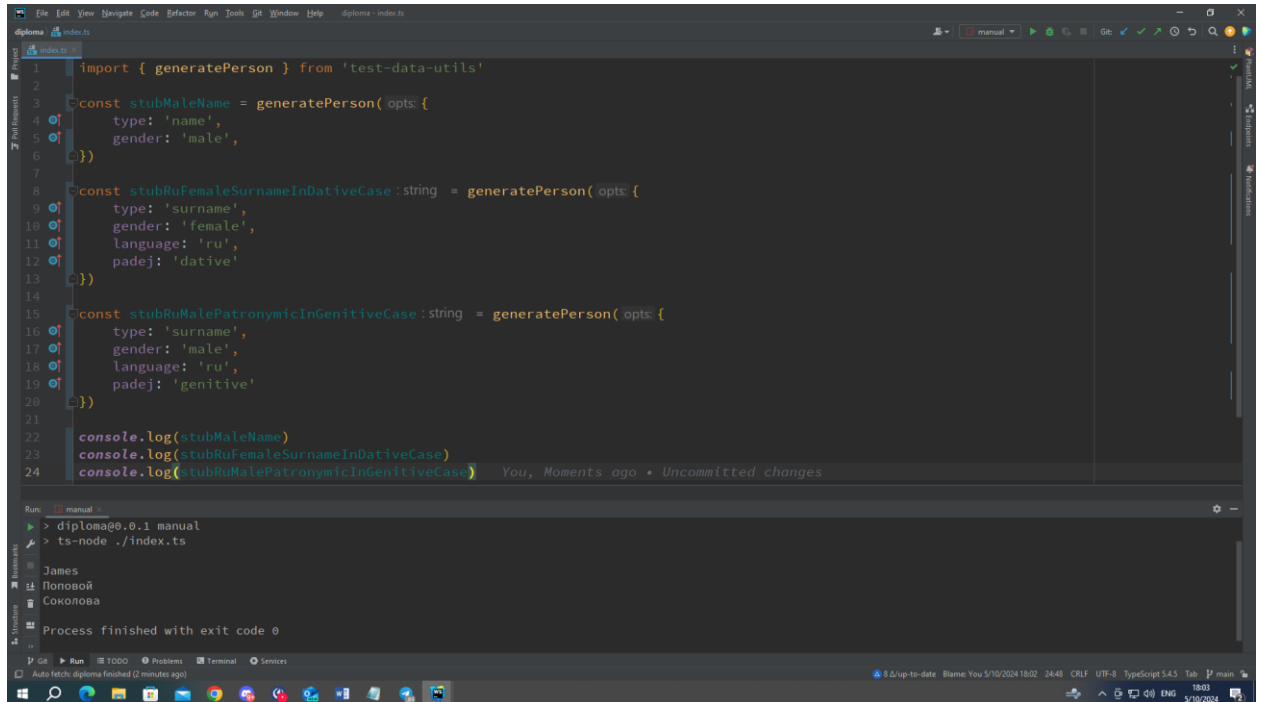
caabbaaccbcbacccaacbabcaaacacbbcaab
36

Process finished with exit code 0

Рисунок 13 — Демонстрация использования функции `generateString()`

6.2.4 Генерация имён

Для генерации случайного имени, фамилии или отчества нужно импортировать из пакета `test-data-utils` и использовать функцию `generatePerson()`, передав ей требуемые характеристики генерируемого объекта. На рис. 14 продемонстрирован пример использования функции.



```
1 import { generatePerson } from 'test-data-utils'
2
3 const stubMaleName = generatePerson( opts: {
4   type: 'name',
5   gender: 'male',
6 })
7
8 const stubRuFemaleSurnameInDativeCase: string = generatePerson( opts: {
9   type: 'surname',
10  gender: 'female',
11  language: 'ru',
12  padelj: 'dative'
13 })
14
15 const stubRuMalePatronymicInGenitiveCase: string = generatePerson( opts: {
16   type: 'surname',
17   gender: 'male',
18   language: 'ru',
19   padelj: 'genitive'
20 })
21
22 console.log(stubMaleName)
23 console.log(stubRuFemaleSurnameInDativeCase)
24 console.log(stubRuMalePatronymicInGenitiveCase)
```

Run manual ...
diploma@0.0.1 manual
ts-node ./index.ts
James
Поповой
Соколова
Process finished with exit code 0

Рисунок 14 — Демонстрация использования функции `generatePerson()`

6.2.5 Генерация осмысленной строки

Для генерации случайной осмысленной строки нужно импортировать из пакета `test-data-utils` и использовать функцию `generateMeaningfulString()`, передав ей требуемые характеристики генерируемого объекта. На рис. 15 продемонстрирован пример использования функции.

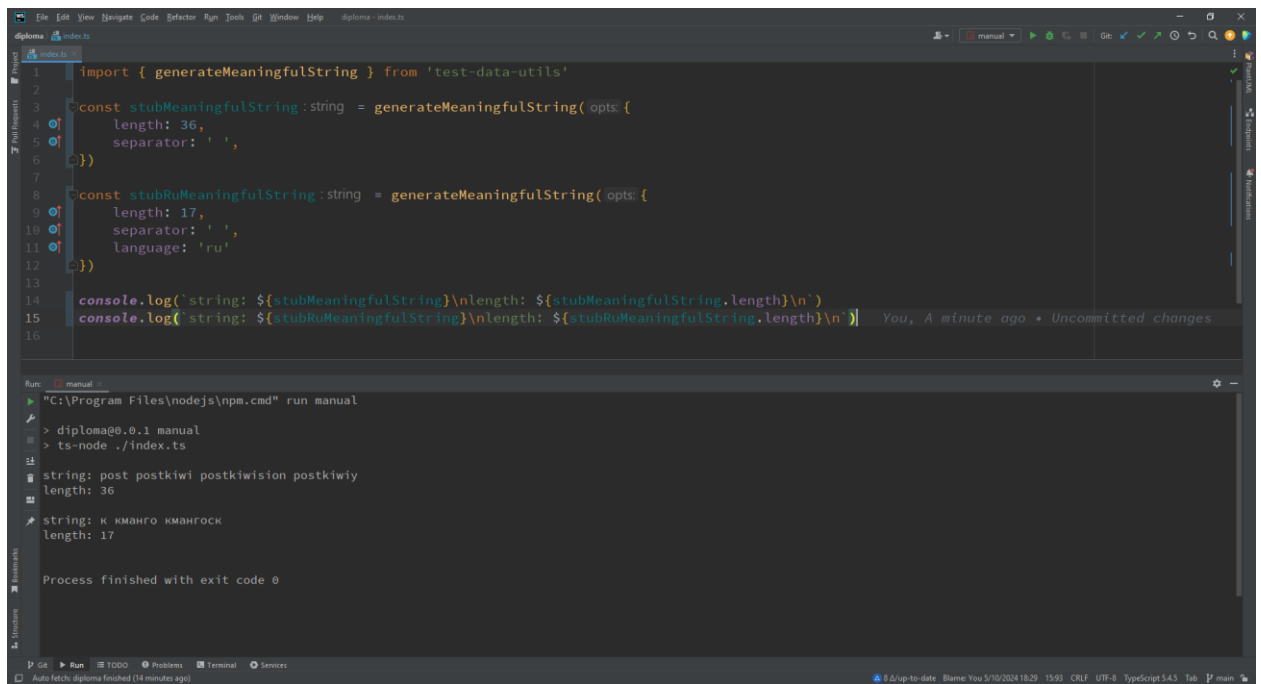


Рисунок 15 — Демонстрация использования функции `generateMeaningfulString()`

7 Оценка производительности библиотеки

7.1 Принципы отбора

В качестве аналогов бенчмарка для библиотек генерации пользовательских данных для frontend тестирования были выбраны NPM-пакеты предоставляющие релевантные возможности по тестированию производительности функций. Были ли отобраны следующие NPM-пакеты: benchmark, pretty-hrtime, tinybench, perfy, micro-benchmark.

Для поиска аналогов использовался сайт пакетного менеджера. Были использованы ключевые слова: "benchmark", "perfomance", "benchmarking".

7.1.1 benchmark

benchmark – это библиотека [27] для создания и выполнения бенчмарков (измерения производительности) в JavaScript/TypeScript. Она предоставляет удобный интерфейс для измерения времени выполнения кода, создания сравнительных тестов и анализа производительности различных реализаций. Он позволяет создавать тестовые наборы, которые могут содержать несколько тестов для сравнения разных реализаций. Набор тестов обычно создается с использованием конструктора Suite. После создания набора тестов можно добавлять индивидуальные тесты, используя метод add(). Можно задавать различные опции для тестов, такие как количество итераций, максимальное время выполнения и другие параметры. Библиотека предоставляет события, которые могут быть использованы для отслеживания различных этапов выполнения тестов, таких как начало, завершение и другие с помощью метода on(). После добавления всех тестов можно запустить набор тестов с использованием метода run(). После выполнения тестов библиотека предоставляет результаты посредством использования полей экземпляра suite в текстовом представлении, которые могут быть использованы для сравнения производительности различных реализаций.

7.1.2 pretty-hrtime

pretty-hrtime – небольшая JavaScript/TypeScript библиотека [28], которая используется для работы с высокочастотными таймерами process.hrtime

[29] и предоставляет функциональность по форматированию времени в удобочитаемую строку (human-readable time). Функция `prettyHrtime()` импортируется из модуля `pretty-hrtime`. В начале измеряемого участка кода программы у объекта `process` вызывается метод `hrtime()`, возвращаемое значение которого кладётся в переменную `start` – переменную отвечающую за начало отсчёта. В конце измеряемого участка кода у объекта `process` вызывается метод `hrtime(start)`, возвращаемое значение которого кладётся в переменную `end`. Далее вызывается функция `prettyHrtime(end)` и кладётся в переменную `word`, в которой будет храниться результат измерения в виде строки с количеством затраченных миллисекунд. Функция `prettyHrtime()` может принимать дополнительные параметры, такие как `verbose` и `precise` – булевы значения отвечающие за отдельный вывод миллисекунд, микросекунд и полноту точности оценки времени выполнения кода. Это может быть полезно при измерении производительности кода, оптимизации алгоритмов или отладке времени выполнения определенных участков программы.

7.1.3 **tinybench**

`tinybench` – это библиотека [30], предоставляющий большой ассортимент инструментов для оценки времени выполнения как синхронных, так и асинхронных JavaScript/TypeScript функций [31]. Библиотека полностью основана на использовании Web API [32], для измерения времени выполнения кода используются `process.hrtime` и `process.now` [33]. Для того чтобы её использовать нужно создать экземпляр класса `Bench` и добавить в него задачи (тестируемые функции) с помощью метода `add()`. После добавления всех задач можно запустить процесс оценки с помощью метода `run()`. После завершения оценки можно вывести её результаты в табличном или текстовом представлении, хранящиеся в полях экземпляра `bench`.

7.1.4 **perfy**

`perfy` – это небольшой NPM-пакет [34], предоставляющий инструменты для оценки времени выполнения определённых участков кода программы. Оценка происходит с помощью вызова методов (меток) экземпляра объекта

perfy – start('mark') и end('mark'). Также есть возможность оценки асинхронных функций с помощью метода exes() и передачи функции обратного вызова done(), которая должна быть вызвана по окончании сценария оценки. Библиотека в сущности является обёрткой для метода now() [35] объекта Date из стандартной библиотеки JavaScript. Результат оценки храниться в текстовом виде в поле time переменной, которую назначит пользователь либо с помощью метода .end(), либо с помощью функции обратного вызова done().

7.1.5 micro-benchmark

`micro-benchmark` - это небольшая JavaScript библиотека [36], позволяющая создавать сравнительные тестовые сценарии на основе заданных параметров – названий операций, тестируемых функций, максимального числа операций и ограничения на время выполнения сценария. Параметры задаются с помощью вызова метода `suite()` у объекта `microBenchmark`, импортируемого из соответствующего модуля, и передачи в него значений полей `duration`, `maxOperations` и `specs` – массива, содержащего тестовые сценарии. Массив сценариев состоит из объектов с полями `name` и `fn` – название сценария и тестируемая функция соответственно. Тестовые сценарии для асинхронных функций генерируются аналогичным образом с помощью метода `suiteAsync({ props }, callback)`, где `props` – аналогичные параметры, передаваемые в метод `.suite()`, а `callback` – функция обратного вызова, вызываемая по окончании тестового сценария. Отчёты об оценке предоставляются в виде ASCII-таблиц при вызове метода `report()` у объекта `microBenchmark`, принимающего в качестве необязательного параметра ширину символов `charWidth`. Отчёт состоит из информации о названии операции, количества выполненных итераций в секунду и среднего время выполнения операции.

7.2 Критерии сравнения

7.2.1 Еженедельное количество скачиваний (Popularity)

Низкий уровень – менее 10,000 еженедельных загрузок. Пакет не пользуется широкой популярностью и, вероятно, не является основным выбором в своей области. Возможно, он новый или малоизвестный.

Средний уровень – от 10,000 до 100,000 еженедельных загрузок. Пакет имеет средний уровень популярности. Он используется в определенном объеме, но не является доминирующим. Поддержка и обновления обычно поддерживаются.

Высокий уровень – более 100,000 еженедельных загрузок. Пакет очень популярен в сообществе разработчиков. Высокое количество скачиваний

указывает на широкое распространение и доверие к пакету. Обычно такие пакеты активно поддерживаются и часто обновляются.

7.2.2 Количество звёзд у GitHub репозитория (Community Trust)

Низкий уровень – менее 100 звёзд. Репозиторий имеет ограниченное внимание со стороны разработчиков. Это может свидетельствовать о том, что пакет не получил широкого признания или не привлек много внимания сообщества.

Средний уровень – от 100 до 1,000 звёзд. Репозиторий имеет умеренное количество звёзд, что указывает на определенный уровень интереса и поддержки. Пакет, вероятно, представляет собой приемлемый выбор среди альтернатив.

Высокий уровень – более 1,000 звёзд. Репозиторий очень популярен среди разработчиков. Многочисленные звёзды свидетельствуют о том, что пакет пользуется широким признанием и может считаться одним из лидеров в своей области.

7.2.3 Частота обновления версии (Maintenance)

Низкий уровень – редкие обновления, менее одного раза в квартал. Пакет может быть устаревшим или неактивно поддерживаемым. Это может повлиять на безопасность и совместимость с последними версиями зависимостей.

Средний уровень – обновления примерно раз в месяц. Пакет регулярно обновляется, что говорит о том, что разработчики следят за изменениями и вносят исправления или новые функции с относительно стабильной периодичностью.

Высокий уровень – частые обновления, более одного раза в неделю. Пакет часто обновляется, что может свидетельствовать об активной разработке, устранении ошибок и внедрении новых возможностей. Такие пакеты обычно поддерживаются в актуальном состоянии.

7.3 Сравнительный анализ

В таблице 2 приведён итог сравнения аналогов.

7.3.1 benchmark

Количество еженедельных скачиваний стабильно варьируется от 300,000 до 400,000. GitHub репозиторий проекта имеет 5,491 звезду [37]. Пакет не обновлялся уже 7 лет.

7.3.2 pretty-hrtime

Количество еженедельных скачиваний достигает 6,3 млн. GitHub репозиторий проекта имеет 66 звёзд [38]. В последний раз пакет обновлялся 8 лет назад.

7.3.3 tinybench

Количество еженедельных скачиваний достигает 3,5 млн. GitHub репозиторий проекта имеет 1,475 звёзд [39]. Пакет стабильно обновляется раз в месяц.

7.3.4 perfy

Количество еженедельных скачиваний варьируется от 13,000 до 36,000. GitHub репозиторий проекта имеет 6 звёзд [40]. Пакет не обновлялся уже 6 лет.

7.3.5 micro-benchmark

Количество еженедельных скачиваний стабильно варьируется от 0 до 29. GitHub репозиторий проекта имеет 3 звезды [41]. Пакет не обновлялся уже 9 лет.

Таблица 2 – Сравнение аналогов по выделенным критериям

	benchmark	pretty-hrtime	tinybench	perfy	micro-benchmark
Popularity	Высокий	Высокий	Высокий	Средний	Низкий
Community Trust	Высокий	Низкий	Средний	Низкий	Низкий
Maintenance	Низкий	Низкий	Средний	Низкий	Низкий

7.3 Выводы по итогам сравнения

После проведения сравнительного анализа аналогов бенчмарка для библиотек генерации пользовательских данных, предназначенных для frontend тестирования, можно сделать вывод, что в силу выдающихся показателей по выбранным критериям наиболее перспективными кандидатами являются библиотеки benchmark и tinybench. Они предоставляют широкий набор инструментов для оценки времени выполнения как синхронных, так и асинхронных JavaScript/TypeScript функций. Однако у каждого аналога есть ограничения по выводу результатов тестирования производительности. Все они предоставляют только текстовую информацию, и в лучшем случае – табличное представление. Таким образом, для качественного выбора подходящей библиотеки необходимо разработать бенчмарк, который бы не только определял среднюю скорость выполнения кода, но также предоставлял графическое представление для большей наглядности [42].

7.4 Выбор метода решения

Необходимо разработать бенчмарк для библиотек генерации пользовательских данных. Бенчмарк должен представлять собой NPM-пакет с доку-

ментацией, который будет храниться в реестре NPM. Бенчмарк должен позволять определять производительность библиотеки.

Бенчмарк должен обладать следующей функциональностью:

1. Принимать функцию генерации пользовательских данных и выводить её среднее время выполнения.
2. Принимать функцию генерации пользовательских данных и выводить её время выполнения в виде графика в зависимости от аргументов.

7.5 Описание метода решения

Был разработан бенчмарк для библиотек генерации пользовательских данных представляющий собой NPM-пакет с документацией [43], который хранится в реестре NPM. Для вычисления времени выполнения функций для измерения времени выполнения кода была использована функция `process.now()` из библиотеки `perf_hooks`. Построение графиков происходит с помощью библиотеки `Chart.js` [44]. Работа с файлами происходит с помощью модуля файловой системы `Node.js` [45].

Для его использования необходимо импортировать класс `Benchmark`.

Класс `Benchmark` является инструментом для измерения производительности функций. Он позволяет производить тестирование производительности функций, а также визуализировать и сохранять результаты в виде графиков зависимости количества операций в секунду от номера итерации.

Функциональность класса реализуется с помощью следующих статических методов, доступных пользователю:

1. `pushCandidate()` – метод для добавляет одну или несколько функций-кандидатов в очередь для тестирования. Принимает в качестве аргументов функции, которые требуется протестировать.
2. `removeCandidate()` – метод для удаления одной или нескольких функций из списка тестируемых кандидатов.
3. `clearCandidates()` – метод для полного очищения списка добавленных функций-кандидатов.

4. `plotAndSaveMeasurementTimesCharts()` – метод для создания и сохранения графика зависимости количества операций в секунду от номера итерации. Если аргумент `pathToMeasurementsPng` не предоставлен, используется стандартный путь «./measurements».
5. `printAvgGenerationTimes()` – метод для вывода в консоль среднего время выполнения для каждой функции-кандидата. Принимает необязательный числовой аргумент `precision`, который указывает количество знаков после запятой.

Для обеспечения функциональности класса `Benchmark` были реализованы следующие приватные статические методы, функции и тип:

1. `getFunctionTimeSeriesAndIterationIndices(fn: Function, iterationCount = 100): ComparedResult` – метод для проведения серий измерений времени выполнения функции. Возвращает объект `ComparedResult`, содержащий массивы времени выполнения и индексов итераций для дальнейшего анализа функции.
2. `generateGraph()` – метод для создания графика времени выполнения, используя индексы итераций и массивы времени выполнения. Визуализация осуществляется с помощью библиотеки `Chart.js`.
3. `getFunctionName(): string` – функция, которая возвращает имя переданной функции.
4. `calculateOpsPerSecond()` – функция, которая вычисляет и возвращает количество операций в секунду на основе переданного времени выполнения.
5. `ComparedResult` – тип, который описывает результаты измерений времени выполнения функции, содержит массив времени выполнения (`timeToGeneration`), индексы итераций (`iterationIndices`) и хэш измерений (`measurementHash`).

7.6 Проведение тестов производительности

Для оценки производительности разработанной библиотеки были проведены тесты сравнения скорости выполнения реализованных функций с

аналогичными функциями из популярной библиотеки `@faker-js/faker`. Тестируемые функциональности включали генерацию UUID, чисел, строк, имён людей, а также генерацию осмысленных строк.

Каждая функциональность тестировалась путём выполнения соответствующих функций многократно с использованием заготовленных входных данных, а затем фиксировалось среднее время выполнения одной итерации, которое приведено в таблице 3. Помимо этого, для визуализации результатов выполнялась генерация графиков.

Таблица 3 – Среднее время выполнения тестируемых функций

Тестируемая функция	Среднее время выполнения (мкс)
<code>generateUUID()</code>	0.618999
<code>faker.string.uuid()</code>	7.959
<code>generateNumber()</code>	0.801995
<code>faker.number.int()</code>	0.697999
<code>generateString()</code>	13.111
<code>faker.string.fromCharacters()</code>	200.62
<code>generatePerson()</code>	0.435987
<code>faker.person.firstName()</code>	0.355999
<code>generateMeaningfulString()</code>	208.73
<code>faker.lorem.slug()</code>	237.54

7.7 Результаты тестов производительности

7.7.1 Генерация UUID

Функция `generateUUID()` показала существенно более высокую производительность по сравнению с функцией `uuid()` из `@faker-js/faker`, среднее время выполнения аналога было больше в 11.85 раз по сравнению с реализованной функцией. График зависимости количества итераций функции за одну секунду от номера итерации измерения представлен на рис. 16.

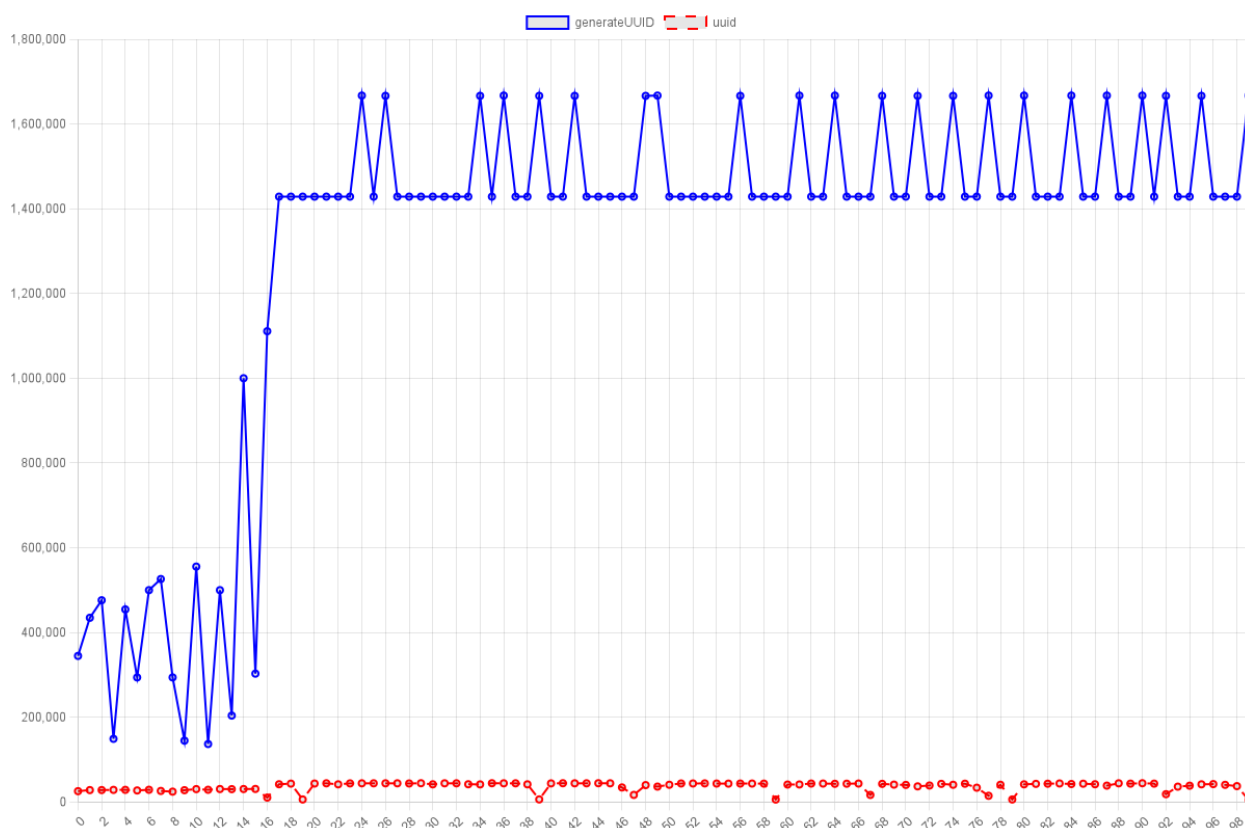


Рисунок 16 — График зависимости количества итераций функции за одну секунду от номера итерации измерения. Генерация UUID

7.7.2 Генерация чисел

Функция `generateNumber()` из реализованной библиотеки и функция `int()` из `@faker-js/faker` показали очень близкую производительность, с небольшим преимуществом библиотеки `faker`. Среднее время реализованной функции было больше на 14.9%. График зависимости количества итераций функции за одну секунду от номера итерации измерения представлен на рис. 17.

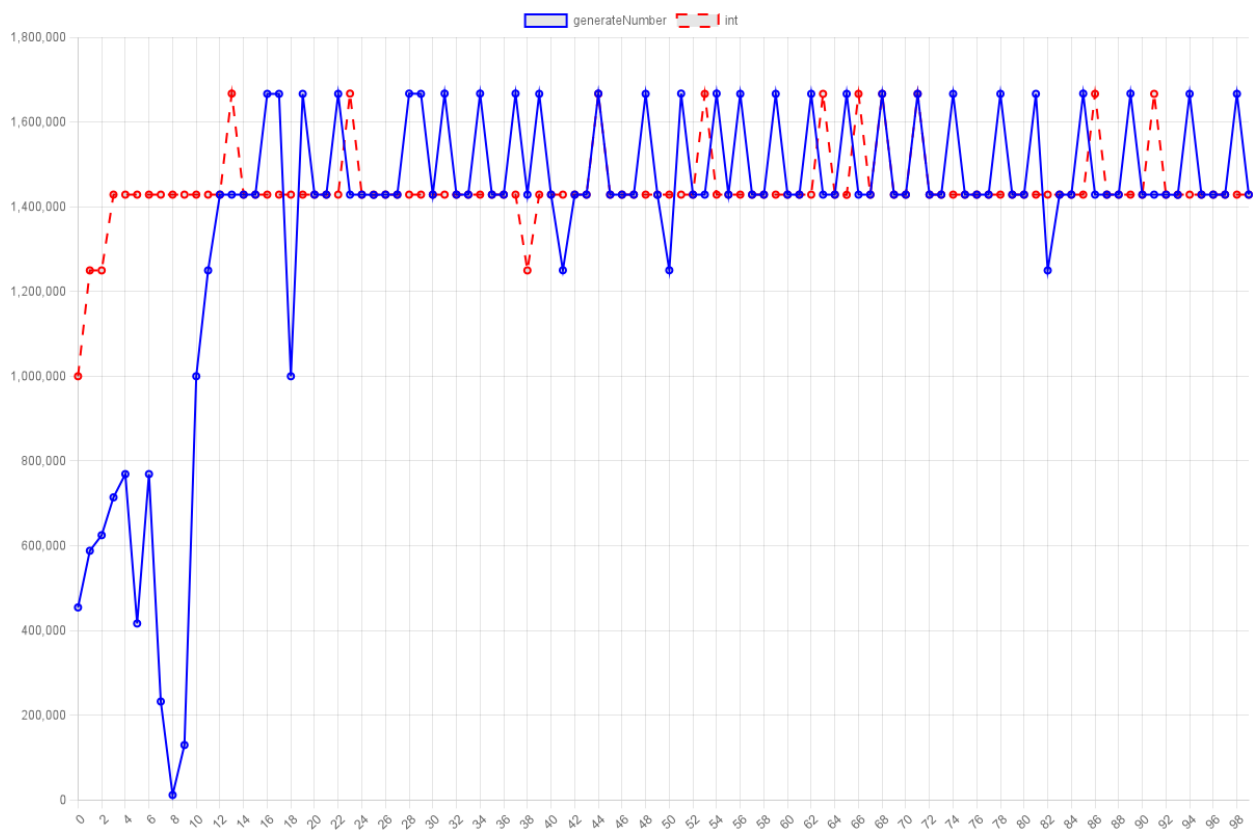


Рисунок 17 — График зависимости количества итераций функции за одну секунду от номера итерации измерения. Генерация чисел

7.7.3 Генерация строк

Функция `generateString()` из разработанной библиотеки по сравнению с аналогом `fromCharacters()` из `@faker-js/faker` показала существенно более производительный результат, среднее время выполнения аналога было больше в 15.3 раз по сравнению с реализованной функцией. График зависимости количества итераций функции за одну секунду от номера итерации измерения представлен на рис. 18.

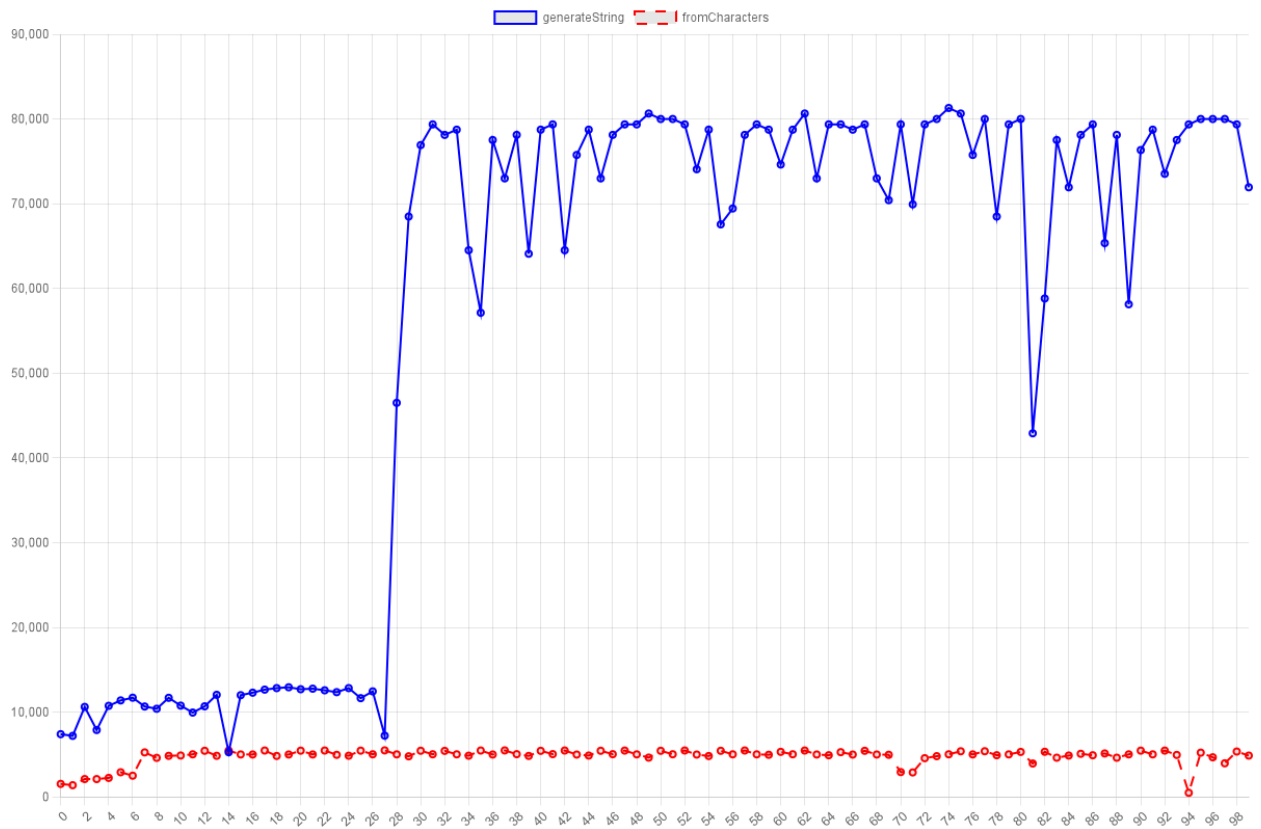


Рисунок 18 — График зависимости количества итераций функции за одну секунду от номера итерации измерения. Генерация строк

7.7.4 Генерация имён

Обе тестируемые функции (`generatePerson()` и `firstName()`) показали схожие результаты по времени выполнения, что указывает на их оптимальность для данной задачи. Среднее время реализованной функции было больше на 22.5%. График зависимости количества итераций функции за одну секунду от номера итерации измерения представлен на рис. 19.

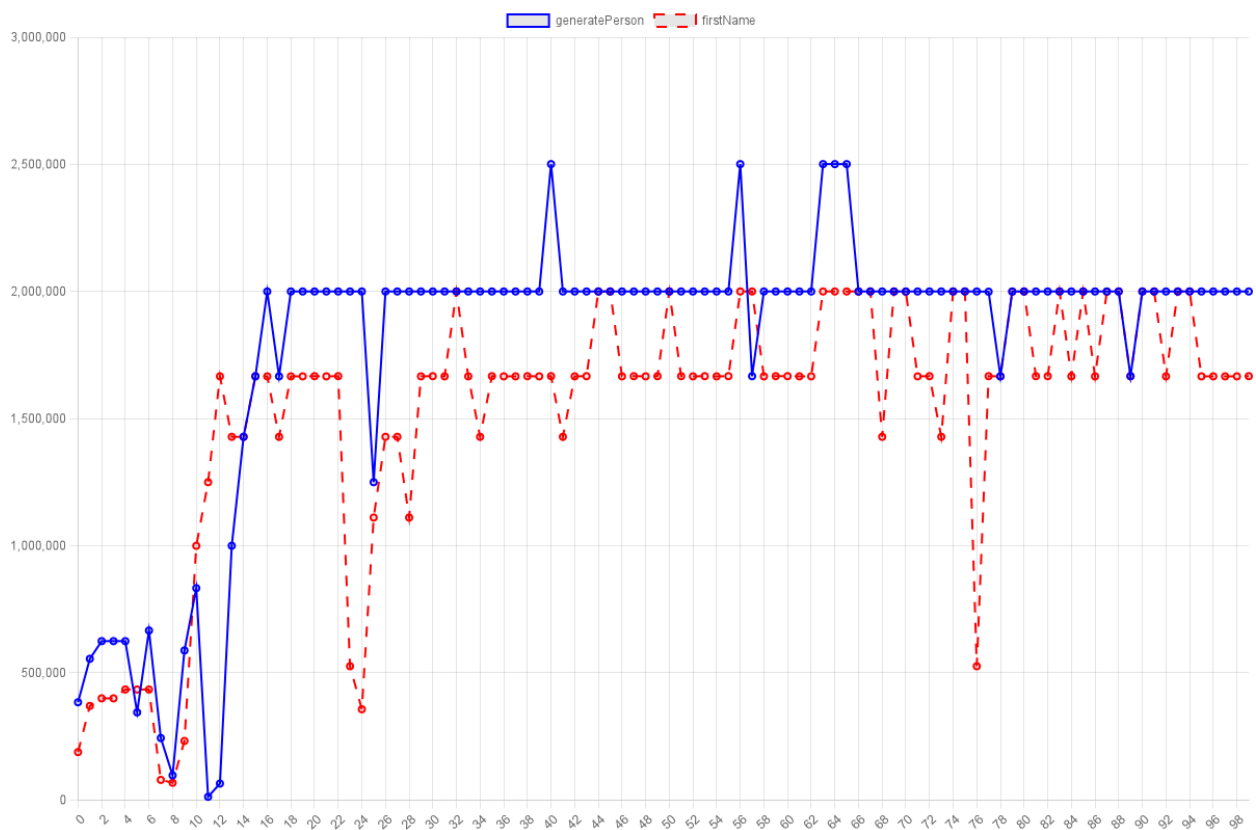


Рисунок 19 — График зависимости количества итераций функции за одну секунду от номера итерации измерения. Генерация имён

7.7.5 Генерация осмысленных строк

Функция `generateMeaningfulString()` из реализованной библиотеки имела чуть более высокую производительность в сравнении с аналогом `slug()` из `@faker-js/faker`, но разница не существенна. Среднее время выполнения реализованной функции на 12.13% меньше, чем у аналога. График зависимости количества итераций функции за одну секунду от номера итерации измерения представлен на рис. 20.

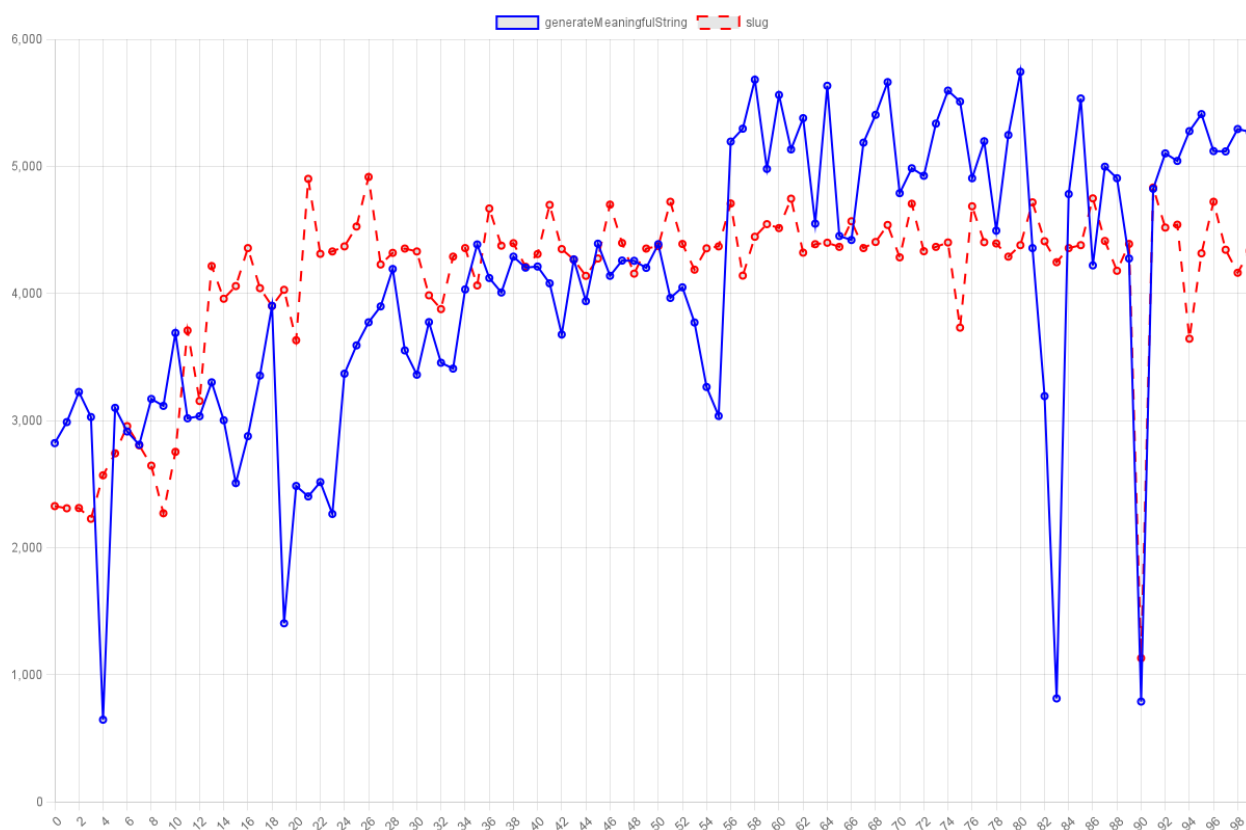


Рисунок 20 — График зависимости количества итераций функции за одну секунду от номера итерации измерения. Генерация осмысленных строк

7.8 Выводы по результатам тестов производительности

Тестирование производительности показывает, что разработанная библиотека обеспечивает либо сопоставимую, либо более высокую производительность по сравнению с `@faker-js/faker`. Особенно выделяются результаты для генерации UUID и строк, где реализованная библиотека демонстрирует значительно лучшие временные характеристики. Эти результаты подтверждают эффективность использования разработанной библиотеки для задач, требующих генерации различных типов данных с высокими требованиями к производительности.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была достигнута поставленная цель – был успешно разработан инструмент в виде NPM-пакета генерации пользовательских данных для тестирования пользовательских интерфейсов.

Для этого в процессе работы были выполнены следующие задачи:

1. Выполнить обзор существующих решений в области генерации пользовательских данных.
2. Спроектировать и реализовать библиотеку для генерации пользовательских данных с помощью языка программирования TypeScript.
3. Реализовать сборку библиотеки с помощью бандлера Webpack.
4. Оформить библиотеку как NPM пакет и произвести его регистрацию в NPM реестре.
5. Исследовать разработанную библиотеку на предмет производительности и сравнить с аналогом.

Размер разработанного пакета небольшой, всего 13.6 КБ. При таком размере скорости скачивание пакета из NPM реестра равняется 92 миллисекунды при 3G стандарте (50 КБ/с) и 5 миллисекунд при 4G стандарте (875 КБ/с) сотовой связи.

Он не требует дополнительных зависимостей к проекту и имеет документацию, в которой описаны все примеры использования [46].

Пакет локализован для нескольких языков, среди которых есть русский, что делает внедрение в русскоязычный проект рациональным.

Как показал сравнительный анализ, разработанная библиотека обеспечивает либо сопоставимую, либо более высокую производительность по сравнению с её аналогом `@faker-js/faker`, особенно выделяются функции `generateUUID()` и `generateString()` своей высокой производительностью. Функции `generateUUID()` и `generateString()` оказались быстрее.

Функция `generateUUID()` продемонстрировала значительно лучшую производительность, чем функция `uuid()` из `@faker-js/faker`, опережая последнюю по скорости в 11.85 раза. В то время как функция `generateNumber()` и

функция `int()` из `@faker-js/faker` показали сопоставимую производительность, `generateNumber()` оказалась на 14.9% медленнее. Функция `generateString()` существенно превзошла функцию `fromCharacters()` по скорости – последняя оказалась на 15.3 раза медленнее. Функции `generatePerson()` и `firstName()` из `@faker-js/faker` продемонстрировали схожие временные показатели, с небольшим отставанием `generatePerson()` на 22.5%. Функция `generateMeaningfulString()` оказалась на 12.13% быстрее по сравнению с функцией `slug()` из `@faker-js/faker`.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Agha, Dua & Sohail, Rashida & Meghji, Areej & Qaboolio, Ramsha & Bhatti, Sania. (2023). Test Driven Development and Its Impact on Program Design and Software Quality: A Systematic Literature Review. VAWKUM Transactions on Computer Sciences. 11. 268-280. 10.21015/vtcs.v11i1.1494.
2. Khanam, Zeba & Ahsan, M.N.. (2017). Evaluating the effectiveness of test driven development: Advantages and pitfalls. International Journal of Applied Engineering Research. 12. 7705-7716.
3. NPM [Электронный ресурс]. URL: <https://www.npmjs.com/> (дата обращения: 03.12.2023).
4. ts-randomizer [Электронный ресурс]. URL: <https://www.npmjs.com/package/ts-randomizer> (дата обращения: 04.12.2023).
5. Lodash [Электронный ресурс]. URL: <https://lodash.com/> (дата обращения: 04.12.2023).
6. uuid [Электронный ресурс]. URL: <https://www.npmjs.com/package/uuid> (дата обращения: 04.12.2023).
7. Kim, Jaehyun & Lee, Yangsun. (2018). A Study on Abstract Syntax Tree for Development of a JavaScript Compiler. International Journal of Grid and Distributed Computing. 11. 37-48. 10.14257/ijgdc.2018.11.6.04.
8. Plugin Support for Custom Transformers [Электронный ресурс]. URL: <https://github.com/microsoft/TypeScript/issues/14419> (дата обращения: 04.12.2023).
9. zufall [Электронный ресурс]. URL: <https://www.npmjs.com/package/zufall> (дата обращения: 04.12.2023).
10. Math.random() [Электронный ресурс]. URL: <https://developer.mozilla.org/en->

- US/docs/Web/JavaScript/Reference/Global_Objects/Math/random (дата обращения: 04.12.2023).
11. @faker-js/faker [Электронный ресурс]. URL: <https://fakerjs.dev/guide/> (дата обращения: 04.12.2023).
 12. faux [Электронный ресурс]. URL: <https://www.npmjs.com/package/faux> (дата обращения: 04.12.2023).
 13. factory.ts [Электронный ресурс]. URL: <https://www.npmjs.com/package/factory.ts> (дата обращения: 04.12.2023).
 14. Optimizing Content Efficiency [Электронный ресурс]. URL: <https://web.dev/articles/performance-optimizing-content-efficiency> (дата обращения: 24.04.2024).
 15. BUNDLEPHOBIA [Электронный ресурс]. URL: <https://bundlephobia.com/> (дата обращения: 25.04.2024).
 16. Web Crypto API [Электронный ресурс]. URL: https://developer.mozilla.org/ru/docs/Web/API/Web_Crypto_API (дата обращения: 26.04.2024).
 17. Node.js [Электронный ресурс]. URL: <https://nodejs.org/en> (дата обращения: 23.11.2023).
 18. Karimov, Elshad. (2020). Trie Data Structure. 10.1007/978-1-4842-5769-2_9.
 19. Putri, Sheila Eka & Tulus, Tulus & Napitupulu, Normalina. (2011). Implementation and Analysis of Depth-First Search (DFS) Algorithm for Finding The Longest Path. 10.13140/2.1.2878.2721.
 20. Febriani, Indah & Ekawati, Risma & Supriadi, Untung & Abdullah, Muhammad Irsyad. (2021). Fisher-Yates shuffle algorithm for randomization math exam on computer based-test. AIP Conference Proceedings. 2331. 060015. 10.1063/5.0042534.

- 21.switch [Электронный ресурс]. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch> (дата обращения: 05.01.2024).
- 22.Webpack [Электронный ресурс]. URL: <https://webpack.js.org/> (дата обращения: 06.05.2024).
- 23.source-map-loader [Электронный ресурс]. URL: <https://webpack.js.org/loaders/source-map-loader/> (дата обращения: 06.05.2024).
- 24.Production [Электронный ресурс]. URL: <https://webpack.js.org/guides/production/> (дата обращения: 06.05.2024).
- 25.yarn publish [Электронный ресурс]. URL: <https://classic.yarnpkg.com/en/docs/cli/publish#search> (дата обращения: 28.11.2023).
- 26.Auditing package dependencies for security vulnerabilities [Электронный ресурс]. URL: <https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities> (дата обращения: 08.05.2024).
- 27.benchmark [Электронный ресурс]. URL: <https://www.npmjs.com/package/benchmark> (дата обращения: 04.12.2023).
- 28.pretty-hrtime [Электронный ресурс]. URL: <https://www.npmjs.com/package/pretty-hrtime> (дата обращения: 04.12.2023).
- 29.process.hrtime [Электронный ресурс]. URL: <https://nodejs.org/api/process.html#processhrtime> (дата обращения: 04.12.2023).
- 30.tinybench [Электронный ресурс]. URL: <https://www.npmjs.com/package/tinybench> (дата обращения: 04.12.2023).

- 31.async function [Электронный ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function (дата обращения: 04.12.2023).
- 32.Performance APIs [Электронный ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/API/Performance_API (дата обращения: 04.12.2023).
- 33.Performance: now() method [Электронный ресурс]. URL: <https://developer.mozilla.org/en-US/docs/Web/API/Performance/now> (дата обращения: 04.12.2023).
- 34.perfy [Электронный ресурс]. URL: <https://www.npmjs.com/package/perfy> (дата обращения: 04.12.2023).
- 35.Date.now() [Электронный ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/now (дата обращения: 04.12.2023).
- 36.micro-benchmark [Электронный ресурс]. URL: <https://www.npmjs.com/package/micro-benchmark> (дата обращения: 04.12.2023).
- 37.GitHub: benchmark.js [Электронный ресурс]. URL: <https://github.com/bestiejs/benchmark.js> (дата обращения: 01.05.2024).
- 38.GitHub: pretty-hrtime [Электронный ресурс]. URL: <https://github.com/robrich/pretty-hrtime> (дата обращения: 01.05.2024).
- 39.GitHub: tinybench [Электронный ресурс]. URL: <https://github.com/tinylibs/tinybench> (дата обращения: 01.05.2024).
- 40.GitHub: perfy [Электронный ресурс]. URL: <https://github.com/onury/perfy> (дата обращения: 01.05.2024).
- 41.GitHub: micro-benchmark [Электронный ресурс]. URL: <https://github.com/f-xyz/micro-benchmark> (дата обращения: 01.05.2024).

42. Prasad, G.V.R.J.S. & Ojha, Amitash. (2012). Text, Table and Graph -- Which is Faster and More Accurate to Understand?. Proceedings - 2012 IEEE 4th International Conference on Technology for Education, T4E 2012. 126-131. 10.1109/T4E.2012.18.
43. data-generators-benchmark [Электронный ресурс]. URL: <https://www.npmjs.com/package/data-generators-benchmark> (дата обращения: 30.04.2024).
44. Chart.js [Электронный ресурс]. URL: <https://www.chartjs.org/> (дата обращения: 06.02.2024).
45. File system | Node.js Documentation [Электронный ресурс]. URL: <https://nodejs.org/api/fs.html> (дата обращения: 11.11.2023).
46. test-data-utils [Электронный ресурс]. URL: <https://www.npmjs.com/package/test-data-utils> (дата обращения: 11.05.2024).