УДК 004.9

АНАЛИЗ И ВЫБОР СОВРЕМЕННЫХ ИНСТРУМЕНТОВ И ТЕХНОЛОГИЙ ДЛЯ РАЗРАБОТКИ СИСТЕМЫ УПРАВЛЕНИЯ ОБУЧЕНИЕМ

Смирнов Даниил Павлович[1], Дедович Татьяна Григорьевна[2]

[1]Аспирант;
Государственный университет «Дубна»;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: smirnov@lmsdubna.ru.

[2]Кандидат физико-математических наук, доцент;
Государственный университет «Дубна»;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: tdedovich@yandex.ru.

В статье рассмотрен вопрос актуальности разработки новой системы управления обучением студентов. Проведен анализ нефункциональных требований, с классификацией по основным категориям. Исследованы современные архитектурные решения, используемые на этапе разработки веб приложений и обоснован выбор решений для создаваемой системы. Рассмотрены современные технологии и инструменты разработки информационных систем. В результате выбраны технологии для создания клиентской и серверной части приложения, обеспечивающие необходимые требования к создаваемой системе.

Ключевые слова: электронное обучение, инструменты разработки, *web*-приложение, система управления обучением, *LMS*.

Для цитирования:
Смирнов Д. П., Дедович Т. Г. Анализ и выбор современных инструментов и технологий для разработки системы управления обучением // Системный анализ в науке и образовании: сетевое научное издание. 2022. № 1. С. 105–116. URL : http://sanse.ru/download/465.

ANALYSIS AND SELECTION OF MODERN TOOLS AND TECHNOLOGIES FOR THE DEVELOPMENT OF A LEARNING MANAGEMENT SYSTEM

Smirnov Daniil P.[1], Dedovich Tatyana G.[2]

[1]PhD student;
Dubna State University;
19 Universitetskaya Str., Dubna, Moscow region, 141980, Russia;
e-mail: smirnov@lmsdubna.ru.

[2]PhD in Physical and Mathematical Sciences, associate professor;
Dubna State University;
19 Universitetskaya Str., Dubna, Moscow region, 141980, Russia;
e-mail: tdedovich@yandex.ru.

The article deals with the topical issue of developing a new student learning management system. An analysis of non-functional safety was carried out, reduced to the main classification categories. The modern architectural solutions used in the development of web applications and based on decisions for making systems are investigated. Modern technologies and tools for developing information systems are considered. As a result, technologies were chosen for creating the client and server parts of applications that ensure the fulfillment of the requirements for starting the system.

Keywords: e-learning, development tools, web application, learning management system, LMS.

For citation:
Smirnov D. P., Dedovich T. G. Analysis and selection of modern tools and technologies for the development of a learning management system. System Analysis in Science and Education, 2022;(1):105–116(In Russ). Available from: http://sanse.ru/download/465.

Введение

В современном мире, наравне с очным образованием все большую роль начинает играть электронное обучение. По данным на 2021 год, в связи с *COVID-19*, в разных сферах образования доля электронного обучения колеблется от 30% до 40% [1]. Для контроля над электронным образованием в университетах и школах используются системы управления обучением (*LMS*). До недавнего времени такие системы справлялись со своими обязанностями, однако в связи с резким повсеместным переходом на электронное образование стало появляться все больше требований к таким системам. Добавление новых компонент в системы либо является нетривиальной задачей и требует много времени, либо невозможно.

Сложности перехода связаны с тем, что системы имеют упрощенную схему автоматизированного обучения. Как правило, они обладают ограниченными возможностями по выбору типов работ и представляют собой программы-тесты: вывод текста, контрольный вопрос, сравнения ответа с образцом. Наибольшие проблемы возникли при переходе на электронное обучение у математических дисциплин, так как для них не применим формат тестов.

Основными недостатками использования существующих систем в математических дисциплинах являются отсутствие:

- модернизации системы контроля и оценивания знаний;
- специализированных форм для проверки доказательств;
- модулей-тренажеров для изучения алгоритмов;
- учета очных занятий.

Поэтому для устранения выявленных недостатков целесообразно разработать новую систему электронного обучения, доступную из любого веб браузера [2]. Также разрабатываемая система позволит реализовать импортозамещение в сфере электронного образования.

При реализации новой системы необходимо правильно подобрать набор инструментов, который упростит ее разработку и поддержку, а также позволит выполнить поставленные нефункциональные требования.

Нефункциональные требования к разрабатываемой системе

Нефункциональные требования к СДО можно сгруппировать в четыре основных класса требований: расширяемость, производительность, безопасность и удобство. Перечислим каждые из них.

Расширяемость:

- платформонезависимость системы;
- возможность добавить новый функционал без изменения высокоуровневой архитектуры;
- масштабируемость системы для возможности простого увеличения количества пользователей;
- минимизация объема передаваемой информации между серверной и клиентской частями системы.

Производительность:

- высокая скорость работы системы;
- низкие аппаратные и программные требования к компьютеру студента.

Безопасность:

- наличие защиты от взломов и основных типов уязвимостей веб-приложений;
- высокий уровень защиты от сбоев с возможностью восстановления (система резервного копирования);
- отслеживание изменений, вносимых пользователями при работе в системе.

Удобство:

- наличие приятного внешнего вида и удобного интерфейса, необходимых для быстрого освоения и работы с приложением;
- интерактивность изменения данных;
- поддержка удаленного обучения (через Интернет) и корректная работа в современных веб-браузерах: *Firefox* 3.5+, *Opera* 10.5+, *Google Chrome* 6+, *Apple Safari* 3+.

Расширяемость является наиболее критичным классом нефункциональных требований, так как в ходе работы системы постоянно появляются отзывы и пожелания пользователей, которые должны быть реализованы оперативно.

Производительность требуется для поддержки работы в системе с большим количеством студентов одновременно.

Безопасность подразумевается прежде всего в смысле защищенности системы от неправомерных действий студентов, направленных на получение непредназначенных для них данных (например, правильных ответов или формул).

Дополнительным требованием является использование при разработке только программных средств, распространяемых бесплатно и находящихся в свободном доступе для уменьшения финансовых трудозатрат на этапе разработки.

Анализ и выбор архитектуры приложения

Рассмотрим существующие архитектурные решения, которые направлены на реализацию нефункциональных требований.

Расширяемость

Расширяемость разрабатываемой системы достигается за счет:

- использования типовых архитектурных решений, модулей;
- применение гибких масштабируемых технологий разработки;
- спецификация *API* (*Application Programming Interface*), которая включает в себя описание наборов классов, процедур, функций, структур и констант, с помощью которых одно приложение может взаимодействовать с другим;
- покрытия тестами;
- документации.

Ниже рассмотрим анализ возможных решений для обеспечения системы расширяемости.

Программные интерфейсы приложений (*API*) используются для взаимного доступа к информации и функциональности различных систем между собой.

Чтобы способствовать быстрой и масштабной интеграции приложений, *API* реализуются с использованием протоколов и спецификаций, определяющих семантику и синтаксис передаваемых сообщений. Эти спецификации составляют архитектуру *API.*

На рис. 1 показана история появления *API* спецификаций в веб среде.

Рис. 1. История появления API спецификаций для web приложений

Сегодня актуальными спецификациями для *API* являются *REST* [3] и *GraphQL* [4]. Проведем сравнительный анализ этих спецификаций:

- Архитектура *REST API* определяется несколькими конечными точками на сервере. Обмен данных в *GraphQL* происходит через одну конечную точку, которая может возвращать сложный граф данных, что приводит к уменьшению количества запросов по сети до одного показа.

- Набор данных возвращаемый в *REST API* состоит из необработанных объектов, найденных на сервере. Такие данные содержать избыточную информацию, например, если для представления требуется только одно свойство из ответа. В другом случае конечная точка списка может возвращать не все запрошенные свойства, тогда потребуется дополнительный запрос. При помощи декларативных запросов *GraphQL* предотвращает избыточное или неполное получение данных.

- Обработка ошибок в REST, использует различные коды *HTTP* [5] уровня 400 для ошибок клиента и коды уровня 200 для успешных ответов. *GraphQL* не обязательно обслуживается через HTTP, соответственно нет никаких требований об использовании кодов ответов *HTTP* для ошибок. Обычно все конечные точки *GraphQL* разрешаются ответом кода 200, а неудачные результаты будут включать свойство “*error*” вместе со свойством *data* в ответе.

- Для управления версиями конечных точек в шаблоне REST применяются дополнительные префиксы. *API*-интерфейсы *GraphQL* стремятся к обратной совместимости и избегают критических изменений, часто выдавая */v1* или */v2* в самом *URL*-адресе для обозначения версии.

В результате проведенного анализа в качестве способа организации обмена данными с сервером, выбран язык запросов *GraphQL*. Отметим ключевые свойства необходимые для обеспечения расширяемости системы:

- Запросы декларативны, что позволяет клиенту точно указать интересующие его поля, и ответ будет содержать только заданные свойства.

- Запросы иерархичны. Возвращаемые данные соответствуют форме запроса.

- Сервер с использованием схемы может узнавать, действителен ли запрос, прежде чем пытаться запросить данные. Проверка гарантирует, что запрос синтаксически верен, однозначен и не содержит ошибок.

- На основе схем запросов можно сгенерировать документацию при помощи специализированных инструментов.

- Спецификация имеет простую и неизбыточную семантику.

Также для корректной разработки использовались базовые принципы:

- *SOLID* [6] (*Single responsibility, Open-closed, Liskov substitution, Interface segregation* и *Dependency inversion*) – аббревиатура пяти основных принципов проектирования в объектно-ориентированном программировании.

- *DRY* [7] (*Don't repeat yourself*) – принцип нацеленности на снижение повторения информации различного рода.

- *KISS* [8] (*Keep it short and simple*) – принцип, при котором простота системы декларируется в качестве основной цели и/или ценности.

- *YAGNI* [9] (*You Ain't Gonna Need It*) – принцип, при котором в качестве основной цели и/или ценности декларируется отказ от добавления избыточной функциональности.

Применение всех вышеперечисленных решений и принципов гарантирует то, что созданное приложение будет простым как в поддержке, так и в тестировании, а также производительным и оптимизированным под любой нагрузкой.

Для проверки работоспособности системы и предотвращения дефектов, было решено проводить тестирование следующих видов: *Unit*, *Integration*, *E2E* [10].

Unit (Модульные) тесты – это тип тестирования программного обеспечения, при котором тестируются отдельные модули или компоненты программного обеспечения

Integration (Интеграционные) тесты используются для проверки бизнес-логики без подключения пользовательского интерфейса.

E2E (Сквозные) тесты – это тип тестирования программного обеспечения, который проверяет программную систему вместе с ее интеграцией с внешними интерфейсами.

На рис. 2 изображен классический вариант пирамиды тестов, описывающий различные уровни и объемы тестирования на каждом слое.

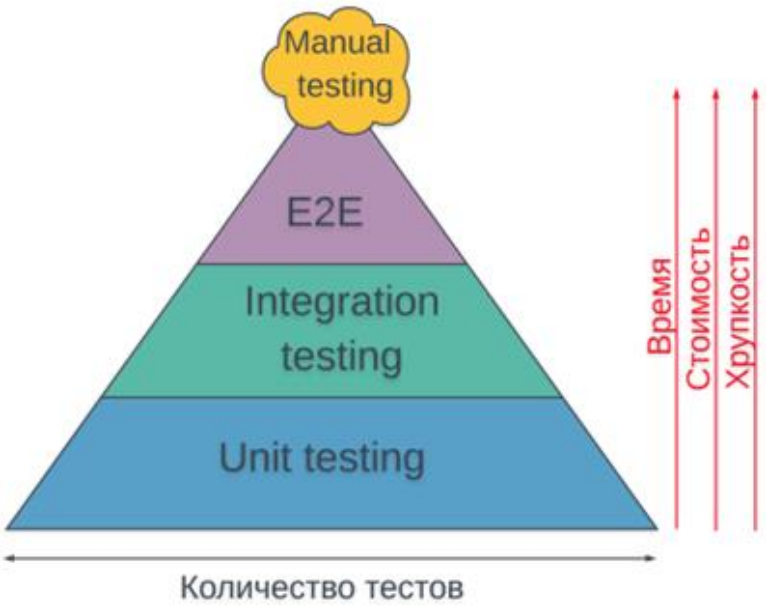

Рис. 2. Пирамида тестов

Производительность

Производительность интернет-приложений повышается за счет оптимизированного формата данных, использования современных и мощных инструментов, минимизации объема пересылаемых данных между сервером и клиентом.

Выбранная ранее спецификация для *API GraphQL* ускоряет передачу данных между клиентом и сервером, что повышает производительность.

Проведем анализ наиболее часто используемых форматов данных. Рассмотрим *JSON*, *XML*, *YAML* [11] по следующим критериям:

- Удобство чтения – предполагает простую и удобную разметку передаваемых данных. При этом язык должен иметь незначительное количество символов-разделителей (скобки, кавычки и т.д.).

- Простота сериализации – преобразования данных в поток байтов для дальнейшего хранения или передачи по каналу связи, в память или файл.

- Простота десериализации – преобразования потока байтов в объект данных.

- Проверка формата входных данных – наличие в формате обмена данными внутреннего языка описания структуры документа, необходимого для осуществления предварительной проверки на соответствие приходящих данных, например, со стороны клиента.

- Сжатие данных – скорость выполнения алгоритма компрессии и коэффициент сжатия.

- Распространенность – наличие большого количества разработчиков, использующих тот или иной формат обмена данными.

- Динамика развития - характеристика, связанная со скоростью популяризации и развития.

Для количественной оценки соответствия тому или иному критерию используется пятибалльная шкала как одна из самых простых и распространенных в системах оценивания.

Результаты сравнительного анализа форматов обмена данными для приложений с клиент-серверной архитектурой – приведены в таблице 1.

Табл. 1. Сравнительный анализ форматов обмена данных

	JSON	XML	YAML
Удобство чтения	5	4	4
Простота сериализации	5	5	5
Простота десериализации	5	5	5
Проверка входных данных	4	4	3
Сжатие данных	5	3	1
Распространенность	5	4	1

На основе проведенного сравнительного анализа в качестве формата данных для обмена между клиентом и сервером был выбран *JSON*. Определяющими требованиями для данного выбора было минимизация объема передаваемой информации между серверной и клиентской частями системы.

Безопасность

Реализация мер безопасности сводится к анализу потенциальных угроз для веб-приложения и принятия мер к их предотвращению. При исследовании популярных угроз для веб-приложения использовался *OWASP* [12]. В нем перечислены основные угрозы веб-приложений, защиту от которых стоит предусмотреть уже в процессе проектирования. Также к проблемам в безопасности можно отнести:

- Ошибки, допущенные во время разработки или развертывания веб-приложения;

- Отказ от полноценного тестирования веб-приложения на наличие уязвимостей или недостаток опыта при его проведении;

- Пренебрежение средствами защиты и мониторинга для своевременного реагирования на инциденты информационной безопасности.

Удобство

Для обеспечения требования интерактивности изменения данных, а также получения изменений в реальном времени, применяется технология веб-сокетов [13]. Веб-сокеты определяются как двусторонняя связь между серверами и клиентами, что означает, что обе стороны могут обмениваться данными одновременно. Решение очевидно, так как альтернативы веб сокетов, которые не являются современными и оптимизированными (*Short polling, Long polling* [14]), или не обладающими полноценным функционалом (*SSE* [15]).

Таким образом, для выбора основных архитектурных решений были рассмотрены технологии для передачи данных между клиентом и сервером, на основе чего выбрана спецификация *GraphQL*. Анализ существующих форматов данных привел к выбору технологии *JSON*. Выделены инструменты, обеспечивающие интерактивность данных, в результате чего будет применяться технология веб-сокеты. Дополнительно был составлен список базовых принципов, используемых при разработке, список видов тестирование и чек лист по основным проблемам в безопасности.

Исследование инструментов разработки

Рассмотрим инструменты, позволяющие реализовать выбранные архитектурные решения. К ним относятся разнообразные редакторы кода, выбранные технологии программирования, фреймворки (инструментарий, содержащий функциональный каркас), и т.д.

Если проанализировать существующие системы управления образовательным процессом (*Moodle* [16], Ё-стади [17], *iSpring Learn* [18]), то при их реализации в основном использовались следующие программные средства:

- *PHP* [19];
- JavaScript [20];
- *MySQL* [21].

Но данный стек технологий на сегодняшний день является устаревшим, и не может полностью удовлетворить все нефункциональные требования, поставленные для системы. В связи с этим был проведен анализ современных технологий для поиска оптимальных и актуальных решений.

Клиентская часть

При выборе языка написания клиентской части рассматривался стандартный *Javascript*, используемый в большинстве веб продуктов, и *Typescript* [22] расширяющий возможности *Javascript* и добавляющий статическую типизацию.

Разница между Javascript и Typescript:

- *Javascript* – это легкий для изучения язык, в то время как *Typescript* требует большого времени на изучение и предварительных знаний в области написания сценариев;
- *Javascript* – это язык сценариев, а *Typescript* – это объектно-ориентированный язык программирования;
- *Typescript* поддерживает интерфейсы, классы, подклассы;
- *Typescript* поддерживает модули, а *Javascript* – нет;
- *Typescript* поддерживает статическую типизацию, которая позволяет проверять правильность типа во время компиляции, тогда как *JavaScript* не поддерживает ее;
- Код *Typescript* должен быть скомпилирован, но нет необходимости компилировать *Javascript*;
- *Typescript* предотвращает часть потенциальных *runtime* ошибок при компиляции;
- *Typescript* является просто надстройкой над *Javascrip*t, поэтому он обладает всем функционалом последнего.

На основе проведенного анализа был сделан вывод о значительном преимуществе *Typescript* над *Javascript*. В итоге код на *Typescript* будет легко анализировать, поддерживать и развивать, что решает нефункциональные требования по масштабируемости.

За последнее время инструменты и технологии для создания интерфейса клиентской части значительно развились, и если раньше не было библиотек сложней *jQuery* [23], то сейчас появилось огромное количество актуальных и производительных замен. И за несколько лет появились несомненные лидеры в лице: *React* [24], *Vue.js* [25] и *Angular* [26].

Все эти три фреймворка были бы отличным выбором для разработки клиентской части, хоть и у каждого из них есть как свои плюсы, так и минусы. Но при анализе других существующих инструментов по производительности обращает внимание на себя *Solid js* [27]. *Solid js* – это *Javascript* библиотека, аналогичная *React, Vue* или *Angular*, которая спроектирована для эффективной отрисовки интерфейса веб приложений. Она обеспечивает все современные наборы возможностей, включая поддержку *Typescript, Granular Reactivity, Async Concurrency* и *JSX* размеченных шаблонных литералов. Кроме того, *Solid js* уже три года является наиболее производительной библиотекой отрисовки в браузере и занимает верхние строчки тестов производительности. На рисунке 3 изображен график анализа производительности тестового приложения при использовании различных фреймворков. Коэффициент, характеризующую каждую библиотеку, рассчитывается на основе нормировки к самой быстрой реализации.

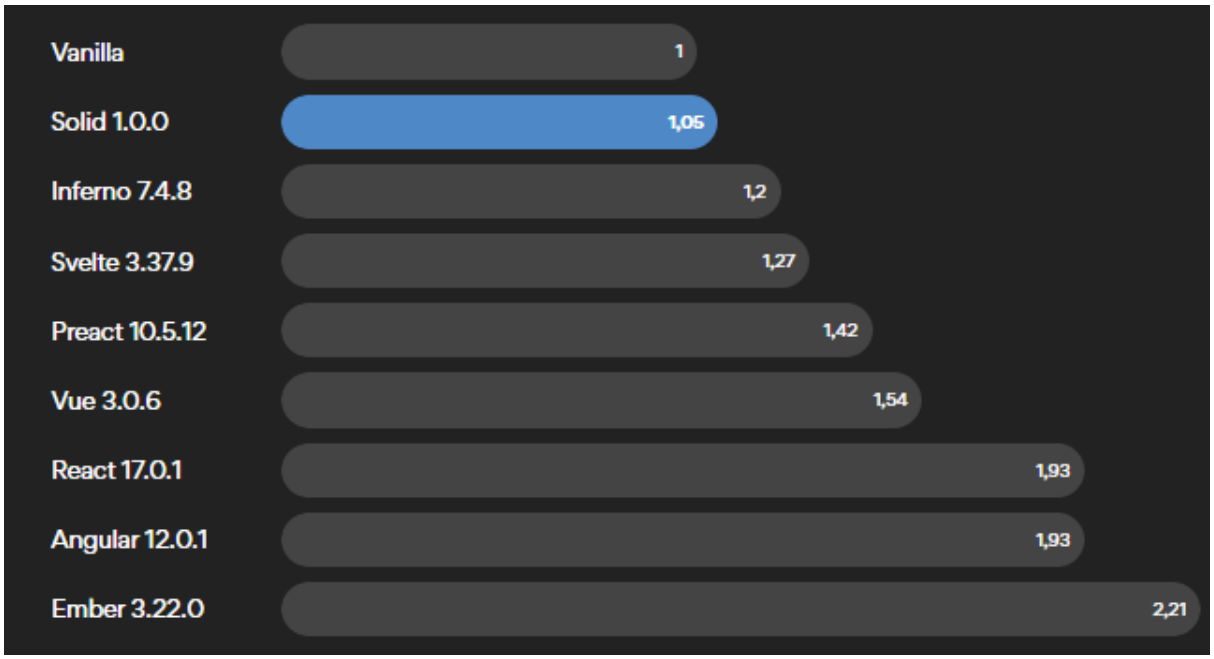

Рис. 3. Характеристика фреймворков для рендера веб приложений

Так как одним из главных нефункциональных требований была производительность, то был выбран *Solid js*. Также использование данного инструмента позволяет разработать удобный интерфейс, необходимый для быстрого освоения при работе с системой.

Серверная часть

Рассмотрим инструменты разработки серверной части веб приложения.

Для сокращения время изучения новых языков и реализации сервера был рассмотрен вариант *Node js* [28]. *Node.js* – платформа для использования *Javascript* на стороне сервера. Эта платформа позволяет писать серверный код для динамических веб-страниц и веб-приложений, а также для программ командной строки. Если говорить о языке, то как для клиентской, так и для серверной части используется один и тот же *Javascript*. Разница только в наборе используемых спецификаций *API*.

Выбор *Node js* обеспечивает проекту ряд преимуществ:

- Рост эффективности разработки благодаря использованию одного языка для клиентской и серверной части, с помощью возможности переиспользования кода;

- возможность использования пакетного менеджера, содержащего большое количество библиотек *npm* [29];

- более простой по сравнению с другими стеками поиска исполнителей, так как *Javascript* входит в число самых популярных языков программирования;

- *Node.js* хорошо подходит для разработки веб-приложений, реагирующих на действия пользователя в режиме реального времени.

На основе вышесказанного было решено в качестве языка разработки серверного приложения использовать *Node js* с поддержкой *Typescript*.

При анализе фреймворка для серверного приложения рассматривалась статистика, приведенная в "*The state of JS 2021*" [30]. В этой системе был проведен опрос JavaScript разработчиков для составления рейтингов сторонних библиотек и фреймворков. На рисунке 4 изображены рейтинги различных фреймворков начиная с 2017 и заканчивая 2021 годом. Стоит учитывать, что в рейтинге помимо полноценных серверных фреймворков (*Fastify, Express, Nest, Strapi, Blitz, Redwood*) участвуют фреймворки для серверного рендеринга клиентской части (*SvelteKit, Astro, Next.js, Remix, Eleventy, Nuxt, Gatsby*). В данном представлении нас интересуют полноценные серверные фреймворки.

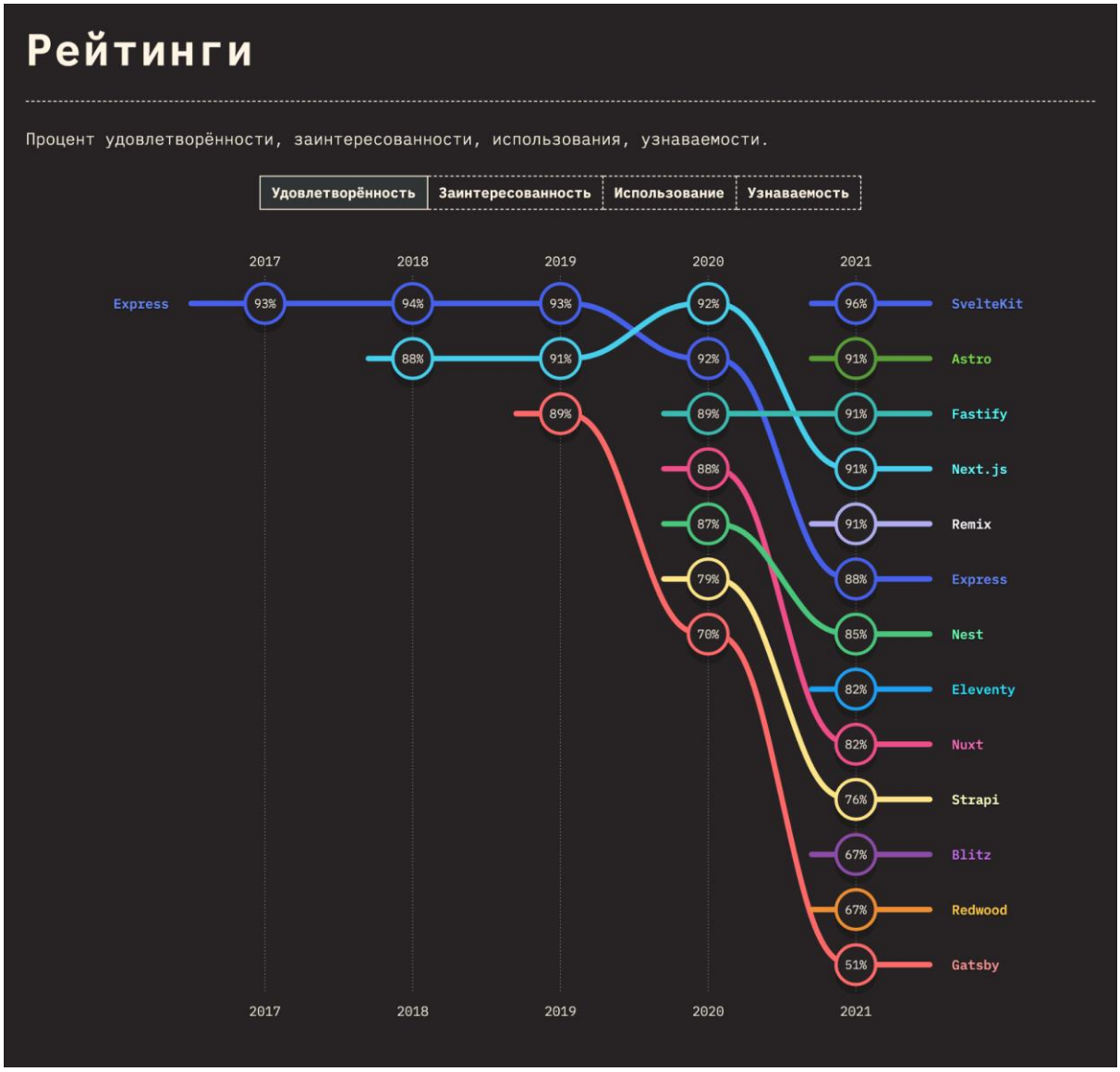

Рис. 4. Рейтинг Node js фреймворков за 2017-2021 годы

На основе рейтинга (рис. 4) были выявлены два популярных *Node js* фреймворка для сервера: *Express* [31] и *Nest* [32]. При тщательном анализе установлено, что *Express* популярней, но не предоставляет полнофункциональный режим работы с сервером. *Nest* является мощным инструментом,

который ускоряет и упрощает разработку масштабируемых серверных приложений на основе программной платформы *Node.js*. Он использует прогрессивный JavaScript и полностью поддерживает *Typescript*. В связи с этим финальным выбором был *Nest js*.

Для обеспечения аудита пользователей и мониторинга попыток взлома было решено использовать библиотеку для *Node js – Winston* [33]. Также для создания и управления резервными копиями использовалась комплексная система – *New Relic* [34].

Проведено исследование современных технологий и инструментов для разработки системы заданным требованиям. На основе анализа были выбраны следующие клиентские технологии: *Typescript* как язык программирования и *Solid js* как фреймворк для разработки пользовательского интерфейса. Для серверной части был выбрана платформа *Node js* с *Typescript*, и фреймворк *Nest js*.

Заключение

В статье проведен анализ нефункциональных требования для новой системы контроля над обучением. Данные требования были классифицированы по четырем категориям, и выделены наиболее критичные.

Для выбора основных архитектурных решений были рассмотрены технологии для передачи данных между клиентом и сервером, на основе чего выбрана спецификация *GraphQL*. Анализ существующих форматов данных привел к выбору технологии *JSON*. Выделены инструменты, обеспечивающие интерактивность данных, в результате чего будет применяться технология Веб-сокеты. Дополнительно был составлен список базовых принципов, используемых при разработке, список видов тестирование и чек лист по основным проблемам в безопасности.

Проведено исследование современных технологий и инструментов для разработки системы заданным требованиям. На основе анализа были выбраны следующие клиентские технологии: *Typescript* как язык программирования и *Solid js* как фреймворк для разработки пользовательского интерфейса. Для серверной части был выбрана платформа *Node js* с *Typescript*, и фреймворк *Nest js*.

Список источников

1. Зафиер А. Статистика и тенденции онлайн-электронного обучения на 2022 год // Website Rating : [веб-сайт]. – Search Ventures Pty Ltd, 2022. – URL: https://www.websiterating.com/ru/research/online-learning-statistics/ (дата обращения: 20.04.2022).
2. Кочешков А. Д., Смирнов Д. П., Дедович Т. Г. Разработка информационной системы контроля и оценки знаний студентов по математическим дисциплинам в университете «Дубна» // Системный анализ в науке и образовании: сетевое научное издание. 2021. № 2. С. 140–150. URL : http://sanse.ru/download/442.
3. Архитектура REST // Хабр : [сайт]. Habr, 2006–2022. Дата публикации: 20.09.2008. URL: https://habrahabr.ru/post/38730/ (дата обращения: 19.04.2022)
4. Бэнкс А. GraphQL. Язык запросов для современных веб-приложений / А. Бэнкс, Е. Порселло. СПб. : Питер, 2019. 240 с.
5. Кришнамурти, Б. Web-протоколы. Теория и практика. HTTP/1.1, взаимодействие протоколов, кэширование, измерение трафика / Б. Кришнамурти, Д. Рексфорд. – М. : БИНОМ, 2002. – 597 с.
6. Кузьмин М. SOLID // Хабр : [сайт]. Habr, 2006–2022. Дата публикации: 05.02.2018. URL: https://habr.com/ru/post/348286/ (дата обращения: 19.04.2022)
7. Ошибочное понимание принципа DRY // Хабр : [сайт]. Habr, 2006–2022. Дата публикации: 27.02.2018. URL: https://habr.com/ru/company/vk/blog/349978/ (дата обращения: 20.04.2022)
8. Принцип программирования KISS — делайте вещи проще // Web-creator. Сложные IT-проекты. Автоматизация бизнеса : [сайт]. ООО «Веб Креатор»; ТМ «Web Creator»; Depix, 2004-2022. URL: https://web-creator.ru/articles/kiss (дата обращения: 20.04.2022)

9. Принцип программирования YAGNI — «Вам это не понадобится» // Web-creator . Сложные IT-проекты. Автоматизация бизнеса : [сайт]. ООО «Веб Креатор»; ТМ «Web Creator»; Depix, 2004-2022. URL: https://web-creator.ru/articles/yagni (дата обращения: 20.04.2022)

10. Питтет С. Различные виды тестирования ПО // Atlassian : [сайт]. Atlassian, 2022 URL: https://www.atlassian.com/ru/continuous-delivery/software-testing/types-of-software-testing (дата обращения: 20.04.2022)

11. Погодин Г. В., Фиго Д. М., Васильев Э. Н. Сериализация структур данных для хранения и передачи в информационных системах. Методы и средства // Молодежь в науке : сборник докладов XVI научно-технической конференции. Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2018. Т. 2. С. 231-236.

12. OWASP. Open Source Foundation for Application Security : [сайт] . Inc, 2022. URL: https://owasp.org/ (дата обращения: 22.04.2022)

13. WebSockets / MDN Web Docs // MDN Web : [сайт]. Mozilla.org, 1998–2022. Дата изменения : 20.10.2021. URL: https://developer.mozilla.org/ru/docs/Web/API/WebSockets_API (дата обращения: 20.04.2022)

14. Long polling // Learn Javascript : [сайт]. Ilya Kantor, 2007—2022. Дата публикации: 12.12.2021. URL: https://javascript.info/long-polling (дата обращения: 20.04.2022)

15. Server Sent Events // Learn Javascript: [сайт]. Ilya Kantor, 2007—2022. Дата публикации: 11.05.2020. URL: https://learn.javascript.ru/server-sent-events (дата обращения: 20.04.2022)

16. MoodleDocs. Documentation // Moodle – Open-source learning platform. Дата изменения : 14.07.2021. URL: https://docs.moodle.org/311/en/Main_page. Дата обращения: 08.04.2022.

17. Сервис (сайт) для дистанционного обучения Ё-Стади. ООО "СЕДЬМОЕ НЕБО", [2022]. URL: https://your-study.ru/. Дата обращения: 08.04.2022.

18. iSpring Learn // iSpring Help Docs : [сайт]. URL: https://www.ispringsolutions.com/docs/display/ispringlearn/iSpring+Learn. Дата обращения: 13.03.2021.

19. Дари, К. AJAX и PHP. Разработка динамических веб-приложений / К.Дари, Б. Бринзаре. СПб. : Символ–Плюс, 2009. 336 с.

20. Крокфорд Д. JavaScript. Сильные стороны. М.: Питер, 2016. 262 с.

21. SQL против NoSQL на примере MySQL и MongoDB // Tproger – всё о программировании : [сайт]. Дата публикации: 24.09.2018. URL: https://tproger.ru/translations/sql-vs-nosql/.

22. Typescript : JavaScript with syntax for types : [сайт]. Microsoft, 2012-2022. URL: https://www.typescriptlang.org/. Дата обращения: 13.04.2022.

23. Бер Б. jQuery. Подробное руководство по продвинутому JavaScript/ Д. Бер, И. Кац. СПб: Символ-плюс, 2017. 624 с.

24. Бэнкс А. React и Redux. Функциональная веб-разработка / А. Бэнкс , Е. Порселло. СПб.: Питер, 2018. 458 с.

25. Vue.js – The Progressive JavaScript Framework : [сайт]. Evan You, 2014-2022. URL: https://vuejs.org/. Дата обращения: 13.04.2022.

26. AngularJS : API Reference // AngularJS : [сайт]. Google, 2010-2021. URL: https://angularjs.org/api/. Дата обращения: 09.04.2022.

27. SolidJS : реактивная JavaScript библиотека. Solid, [2022]. URL: https://www.solidjs.com/. Дата обращения: 09.04.2022.

28. Пауэрс Ш. Изучаем Node.js. СПб.: Питер, 2015. 400 с.

29. Журавлев И. Шпаргалка по пакетному менеджеру NPM // Хабр : [сайт]. Habr, 2006–2022. Дата публикации: 25.11.2011. URL: https://habr.com/ru/post/133363. Дата обращения: 19.04.2022.

30. State of JS : [сайт]. URL: https://2021.stateofjs.com/ru-RU/. Дата обращения: 19.04.2022.

31. Браун И. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript / И. Браун. Санкт-Петербург: Питер, 2017. 336 с.

32. NestJS- A progressive Node.js framework : [сайт]. Kamil Mysliwiec, 2017-2022. URL: https://nestjs.com. Дата обращения: 13.06.2021.

33. Robbins C. Winston // npm : [сайт]. URL: https://www.npmjs.com/package/winston. Дата обращения: 19.04.2022.

34. New Relic — полный мониторинг вашего RoR приложения // Хабр : [сайт]. Habr, 2006–2022. Дата публикации: 29.06.2009. URL: https://habr.com/ru/post/63072. Дата обращения: 19.04.2022.