

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS**

Сравнительный анализ реализаций статической типизации в JavaScript

Обучающийся / Student Левусь Александр Владимирович

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники

Группа/Group P42081

Направление подготовки/ Subject area 09.04.04 Программная инженерия

Образовательная программа / Educational program Веб-технологии 2021


Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Магистр

Руководитель ВКР/ Thesis supervisor Авксентьева Елена Юрьевна, доцент, кандидат педагогических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Обучающийся/Student

Документ подписан	
Левусь Александр Владимирович	
29.05.2023	

(эл. подпись/ signature)

Левусь
Александр
Владимирович

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Авксентьева Елена Юрьевна	
26.05.2023	

(эл. подпись/ signature)

Авксентьева
Елена Юрьевна

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Левусь Александр Владимирович

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники

Группа/Group P42081

Направление подготовки/ Subject area 09.04.04 Программная инженерия

Образовательная программа / Educational program Веб-технологии 2021

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Магистр

Тема ВКР/ Thesis topic Сравнительный анализ реализаций статической типизации в JavaScript

Руководитель ВКР/ Thesis supervisor Авксентьева Елена Юрьевна, доцент, кандидат педагогических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Исследовать способы реализации статической типизации в JavaScript и провести их сравнительный анализ

Задачи, решаемые в ВКР / Research tasks

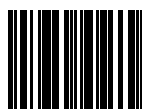
- провести анализ российских и зарубежных источников по теме исследования - провести анализ способов реализации статической типизации в JavaScript - разработать систему критериев для сравнения способов реализации статической типизации в JavaScript. - разработать веб-приложение с помощью разных инструментов для статической типизации - провести сравнительный анализ рассматриваемых инструментов - составить рекомендации по выбору инструмента для статической типизации.

Краткая характеристика полученных результатов / Short summary of results/findings

Исследованы способы реализации статической типизации в JavaScript и проведен их сравнительный анализ

Обучающийся/Student

Документ подписан
Левусь Александр Владимирович



Левусь
Александр

Руководитель ВКР/
Thesis supervisor

29.05.2023	
------------	--

(эл. подпись/ signature)

Владимирович

(Фамилия И.О./ name and surname)

Документ подписан	
Авксентьева Елена Юрьевна	
26.05.2023	

(эл. подпись/ signature)

Авксентьева
Елена Юрьевна

(Фамилия И.О./ name and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Левусь Александр Владимирович
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники
Группа/Group P42081
Направление подготовки/ Subject area 09.04.04 Программная инженерия
Образовательная программа / Educational program Веб-технологии 2021
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Магистр
Тема ВКР/ Thesis topic Сравнительный анализ реализаций статической типизации в JavaScript
Руководитель ВКР/ Thesis supervisor Авксентьева Елена Юрьевна, доцент, кандидат педагогических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Техническое задание:

- рассмотреть российские и зарубежные источники по теме исследования
- проанализировать способы типизации в JavaScript
- разработать систему критериев для сравнения разных подходов к статической типизации
- спроектировать и разработать веб-приложение с разными инструментами для статической типизации
- провести сравнительный анализ рассматриваемых инструментов
- сформулировать рекомендации по выбору инструмента для статической типизации

Цель работы:

Исследовать способы реализации статической типизации в JavaScript и провести их сравнительный анализ

Задачи работы:

- провести анализ российских и зарубежных источников по теме исследования
- провести анализ способов реализации статической типизации в JavaScript
- разработать систему критериев для сравнения способов реализации статической типизации в JavaScript.
- разработать веб-приложение с помощью разных инструментов для статической типизации
- провести сравнительный анализ рассматриваемых инструментов
- составить рекомендации по выбору инструмента для статической типизации.

Перечень подлежащих разработке вопросов:

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

ВВЕДЕНИЕ

1. ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ И АНАЛИЗ НАУЧНЫХ ИСТОЧНИКОВ

1.1 Статическая и динамическая типизации

1.2 Транспилиция

1.3 JavaScript

1.4 TypeScript

1.5 ReScript

1.6 Elm

1.7 PureScript

1.8 Flow

1.9 Сравнение инструментов для статической типизации

2. РАЗРАБОТКА СИСТЕМЫ КРИТЕРИЕВ ДЛЯ СРАВНЕНИЯ ИНСТРУМЕНТОВ

2.1. Разработка критериев

2.2. Выделенные критерии

3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРИЛОЖЕНИЙ

3.1 Проектирование приложения

3.2. Разработка приложений

3.3. Сбор данных о производительности

3.4 Итоговая таблица и рекомендации

ЗАКЛЮЧЕНИЕ

Рекомендуемые материалы и пособия:

1. Вандеркам Дэн Эффективный TypeScript: 62 способа улучшить код. — СПб.: Питер, 2020. — 288 с

2. Борис Черный Профессиональный TypeScript. Разработка масштабируемых JavaScript-приложений. — СПб.: Питер, 2021. — 352 с.

3. ReScript [Электронный ресурс]. Режим доступа: <https://rescript-lang.org/docs/latest>

4. PureScript [Электронный ресурс]. Режим доступа: <https://github.com/purescript/documentation/blob/master/guides/Getting-Started.md>

5. Elm [Электронный ресурс]. Режим доступа: <https://elm-lang.org/docs>

6. TypeScript [Электронный ресурс]. Режим доступа: <https://www.typescriptlang.org/docs/>

Форма представления материалов ВКР / Format(s) of thesis materials:

Презентация

Дата выдачи задания / Assignment issued on: 06.02.2023

Срок представления готовой ВКР / Deadline for final edition of the thesis 31.05.2023

Характеристика темы ВКР / Description of thesis subject (topic)

Название организации-партнера / Name of partner organization: ООО «ДТ АЙТИ РУС»

Тема в области фундаментальных исследований / Subject of fundamental research: нет / not

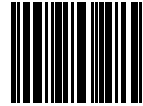
Тема в области прикладных исследований / Subject of applied research: нет / not

СОГЛАСОВАНО / AGREED:

--	--

Руководитель ВКР/
Thesis supervisor

Документ подписан
Авксентьева Елена Юрьевна
14.05.2023

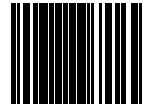


Авксентьева
Елена Юрьевна

(эл. подпись)

Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан
Левусь Александр Владимирович
18.05.2023

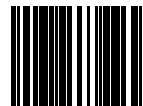


Левусь
Александр
Владимирович

(эл. подпись)

Руководитель ОП/ Head
of educational program

Документ подписан
Государев Илья Борисович
19.05.2023



Государев Илья
Борисович

(эл. подпись)

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ	4
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ	5
ВВЕДЕНИЕ	7
1 ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ И АНАЛИЗ НАУЧНЫХ ИСТОЧНИКОВ	10
1.1 Статическая и динамическая типизации	10
1.2 Транспилиция	11
1.3 JavaScript	12
1.4 TypeScript.....	14
1.5 ReScript	17
1.6 Elm	18
1.7 PureScript	20
1.8 Flow	21
1.9 Сравнение инструментов для статической типизации	23
2 РАЗРАБОТКА СИСТЕМЫ КРИТЕРИЕВ ДЛЯ СРАВНЕНИЯ ИНСТРУМЕНТОВ.....	26
2.1 Разработка критериев.....	26
2.1.1 Потребление ресурсов процессора	26
2.1.2 Потребление оперативной памяти.....	26
2.1.3 Время выполнения	27
2.1.4 Вес скомпилированного кода	28
2.1.5 Время компиляции.....	29
2.1.6 Постепенная типизация.....	30
2.1.7 Популярность.....	30
2.1.8 Инструменты и экосистема.....	31
2.1.9 Документация и материалы для обучения	33
2.2 Выделенные критерии.....	33
3 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРИЛОЖЕНИЙ	36
3.1 Проектирование приложения.....	36

3.2 Разработка приложений	37
3.2.1 TypeScript.....	37
3.2.2 ReScript	38
3.2.3 PureScript	40
3.2.4 Elm	43
3.3 Сбор данных о производительности.....	45
3.3.1 TypeScript.....	45
3.3.2 ReScript	46
3.3.3 PureScript	48
3.3.4 Elm	50
3.4 Итоговая таблица и рекомендации.....	52
ЗАКЛЮЧЕНИЕ	54
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	56

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

TS – TypeScript

JS – JavaScript

ООП – Объектно-ориентированное программирование

ФП – Функциональное программирование

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Язык программирования – формальный язык, предназначенный для записи компьютерных программ

Типизация – принцип, описывающий то, как выбранный язык будет распознавать типы данных в кодификации

Тип данных – класс данных, характеризуемый членами класса и операциями, которые могут быть к ним применены

Слабая типизация – принцип, при котором язык программирования разрешает выполнение выражений с любыми типами и самостоятельно выполняет неявные преобразование

Сильная типизация – принцип, при котором язык программирования не позволяет смешивать в выражениях различные типы и не выполнять автоматические неявные преобразования

Динамическая типизация – принцип, при котором конечные типы переменных и функций устанавливаются во время выполнения программы

Статическая типизация – принцип, при котором конечные типы переменных и функций устанавливаются на этапе компиляции программы

Явная типизация – принцип, который предполагает, что указание принадлежности языковых элементов к конкретному типу возлагается на разработчика

Неявная типизация – принцип, в котором при объявлении языковых элементов не требует от разработчика указания принадлежности к конкретному типу данных и возлагает определение типов на компилятор или интерпретатор

Компиляция – трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду (абсолютный код, объектный модуль, иногда на язык ассемблера), выполняемая компилятором

Компилятор – программа, переводящая написанный на языке программирования текст в набор машинных кодов

Интерпретатор – программа, которая непосредственно выполняет

инструкции, написанные на языке программирования или сценариев, не требуя, чтобы они ранее были скомпилированы в программу на машинном языке

Интерпретация – построчный анализ, обработка и выполнение исходного кода программы или запроса

Транспилиция – преобразование программы, написанной на одном языке программирования, в эквивалентный код другой версии этого языка или в другой язык программирования того же уровня абстракции

Транспайлер – тип транслятора, который принимает исходный код программы, написанной на языке программирования, в качестве входных данных и выдает эквивалентный исходный код на том же или другом языке программирования

ВВЕДЕНИЕ

JavaScript – это один из самых популярных языков программирования в мире, который используется для создания динамических веб-приложений. Он был создан как язык скриптового программирования, который позволяет создавать интерактивные элементы на веб-страницах. Однако, с течением времени, его функциональность и возможности расширились, и сегодня JavaScript используется для создания более сложных и масштабных приложений.

Одним из ключевых аспектов разработки приложений на JavaScript является типизация. Типизация в программировании — это процесс определения типов данных, которые могут быть использованы в программе. Существуют два типа типизации: статическая и динамическая. Статическая типизация определяет типы данных на этапе компиляции, в то время как динамическая типизация определяет типы данных во время выполнения программы.

Отличия в типизации имеют большое значение, когда возникают ошибки, связанные с типом данных. В языках с статической типизацией ошибки обнаруживаются на этапе компиляции. Проверка типов гарантирует соответствие типа конструкции определенному инварианту. В языках с динамической типизацией, таких как JavaScript, ошибки могут быть обнаружены только во время выполнения программы. Согласно статистике, State of JS 2023 года, большинство опрошенных считает, что в JavaScript не хватает статической типизации.

Все выше сказанное подтверждает актуальность темы выпускной квалификационной работы «Сравнительный анализ реализаций статической типизации в JavaScript».

Целью данной работы является исследование и практическое выявление эффективных способов реализации статической типизации в JavaScript.

Объектом исследования является сравнительный анализ подходов к реализации статической типизации в JavaScript. Предметом исследования

являются подходы к реализации статической типизации в JavaScript.

Для достижения поставленной цели необходимо было решить следующие задачи:

- 1) провести анализ российских и зарубежных источников по теме исследования,
- 2) изучить способы реализации статической типизации в JavaScript,
- 3) разработать веб-приложения для проведения эксперимента,
- 4) провести эксперимент по сравнению производительности веб-приложений с разными подходами к статической типизации,
- 5) провести сравнительный анализ,
- 6) разработать рекомендации по применению подходов к реализации статической типизации в JavaScript.

Теоретической основой исследования выступили труды российских (А. Веселовский, Т. Гумеров, Б. Чёрный) и зарубежных (J. Goldberg, Д. Вандеркам, Н. Розенталс, W. Loder, J. Fairbank) ученых в области веб-разработки с использованием JavaScript.

Для решения перечисленных задач использовался комплекс теоретических и эмпирических взаимодополняющих методов исследования, среди которых ведущими были следующие методы: анализ, сравнение и обобщение, методы индукции и дедукции.

Основными информационными источниками для исследования являлись научные публикации по исследуемой тематике, найденные в научных электронных библиотеках, официальная документация по технологиям и ресурсы сети Интернет.

Практическая значимость проведенного исследования заключается в выделении достоинств и недостатков различных подходов к статической типизации в JavaScript, а также рекомендации по их применению.

Достоверность теоретических и практических результатов исследования обеспечиваются проведённым анализом области исследования, использованием комплекса методов для достижения цели и выполнения

поставленных задач исследования.

1 ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ И АНАЛИЗ НАУЧНЫХ ИСТОЧНИКОВ

1.1 Статическая и динамическая типизации

Статическая и динамическая типизация – это два основных подхода к определению типов данных в языках программирования. В статической типизации типы данных определяются на этапе компиляции программы, в то время как в динамической типизации типы данных определяются во время выполнения программы.

Статическая типизация обычно используется в языках программирования, таких как C++, Java, C# и других. При использовании статической типизации разработчик определяет типы данных для каждой переменной в программе. Компилятор проверяет типы данных на предмет соответствия перед выполнением программы. Это позволяет избежать ошибок типизации во время выполнения программы, таких как несоответствие типов данных или неверное использование функций.

Однако, статическая типизация может быть немного более сложной для начинающих программистов. Она требует более строгого контроля со стороны программиста, поскольку неправильное определение типов данных может привести к ошибкам компиляции. Кроме того, статическая типизация может ограничивать гибкость программиста, поскольку определение типов данных заставляет программиста учитывать конкретные типы данных, которые могут быть использованы в программе.

В динамической типизации типы данных определяются во время выполнения программы. Это означает, что программист может создавать переменные без предварительного определения типа данных. Типы данных могут изменяться во время выполнения программы, в зависимости от данных, которые программа получает. Динамическая типизация используется в языках программирования, таких как JavaScript, Python, Ruby и других.

Одним из главных преимуществ динамической типизации является гибкость. Программист может легко создавать переменные и изменять типы

данных, что может быть очень полезно в различных ситуациях. Кроме того, динамическая типизация может быть более простой для начинающих программистов, поскольку она не требует такой же строгой контроля, как статическая типизация.

Однако, динамическая типизация может приводить к ошибкам во время выполнения программы. Несоответствие типов данных или неверное использование функций может привести к ошибкам, которые будут обнаружены только во время выполнения программы. Это может усложнить процесс отладки программы и снизить ее производительность.

1.2 Транспилиция

Транспилиция (или компиляция в компиляторе) – это процесс преобразования исходного кода одного языка программирования в исходный код другого языка программирования. Она позволяет программистам использовать более удобный для них язык программирования, который будет затем транслирован в другой язык программирования, который будет работать в конечном счете на целевой платформе.

Одним из наиболее распространенных примеров транспилиции является использование языка программирования TypeScript. TypeScript является надмножеством языка JavaScript, который добавляет статическую типизацию и другие новые возможности в JavaScript. TypeScript-код может быть скомпилирован в JavaScript-код, который может быть запущен на любой платформе, поддерживающей JavaScript.

Главным преимуществом транспилиции является возможность использования более высокоуровневых конструкций языка программирования, которые могут быть не доступны в целевом языке программирования. Например, TypeScript позволяет использовать статическую типизацию и другие функции, которые не поддерживаются в JavaScript. Кроме того, транспилиция может упростить процесс разработки программного обеспечения, позволяя программистам использовать более простой и удобный для них язык программирования, который затем будет

преобразован в другой язык программирования.

Однако, транспилиция может также иметь некоторые недостатки. Код, скомпилированный из одного языка программирования в другой, может иметь более низкую производительность, чем нативный код, написанный в целевом языке программирования. Кроме того, при использовании транспилиции необходимо быть внимательным при выборе целевого языка программирования, поскольку некоторые функции языка программирования могут быть недоступны в другом языке программирования.

Существует множество инструментов и технологий, которые позволяют проводить транспилицию кода. Некоторые из них – это Babel, TypeScript, CoffeeScript, Dart и другие. Каждый из этих инструментов предназначен для определенных целей и имеет свои преимущества и недостатки, которые необходимо учитывать при выборе технологии для транспилиции.

1.3 JavaScript

JavaScript – это язык программирования, который используется веб-разработкой для создания динамических веб-страниц и приложений. Он был создан в 1995 году Бренданом Айком в Netscape Communications Corporation и изначально назывался Mocha, затем переименован в LiveScript и в конце концов в JavaScript.

JavaScript является интерпретируемым языком программирования, что означает, что он выполняется непосредственно в браузере, без необходимости компиляции. Это позволяет разработчикам создавать динамические веб-страницы, которые могут обновляться без перезагрузки страницы. JavaScript используется для создания различных функций, включая анимацию, формы, валидацию, динамические меню и многие другие.

JavaScript поддерживает объектно-ориентированную и функциональную парадигмы программирования, что делает его мощным и гибким языком программирования. В отличие от статически типизированных языков программирования, таких как C++ или Java, JavaScript является динамически типизированным языком программирования, что означает, что

тип переменной может изменяться во время выполнения программы.

JavaScript имеет множество встроенных объектов, таких как Math, Date и Array, которые упрощают разработку веб-приложений. Он также поддерживает работу с API браузера, позволяющий взаимодействовать с веб-страницей, например, изменять содержимое веб-страницы, обрабатывать события мыши и клавиатуры и многое другое.

JavaScript также поддерживает множество библиотек и фреймворков, таких как React, Angular и Vue.js, которые значительно упрощают создание сложных веб-приложений. Они предоставляют готовые компоненты и модули, которые могут быть использованы в приложении, что значительно сокращает время разработки и упрощает поддержку приложения.

Однако, JavaScript также имеет некоторые ограничения. JavaScript-код может быть медленнее, чем нативный код, написанный на других языках программирования, таких как C++ или Java. Кроме того, JavaScript не поддерживает многопоточность, что может быть проблемой при разработке сложных веб-приложений, требующих большого количества обработки данных.

JavaScript также стал известен своей проблемой с обработкой ошибок. Из-за своей динамической типизации, JavaScript может быть труден для отладки, и ошибки могут быть трудно выявить и исправить.

Однако, JavaScript продолжает оставаться одним из самых популярных языков программирования, используемых в веб-разработке. Благодаря своей гибкости и удобству, он продолжает эволюционировать и улучшаться.

Недавние улучшения включают в себя новые возможности ECMAScript, такие как стрелочные функции, деструктурирование объектов, парамсы по умолчанию и многое другое. Эти возможности улучшают читаемость и поддерживаемость кода, а также позволяют разработчикам писать более компактный и эффективный код.

Некоторые разработчики также используют TypeScript, язык программирования, основанный на JavaScript, но с добавлением статической

типизации. TypeScript позволяет предотвращать ошибки типизации, что может значительно улучшить производительность и поддерживаемость кода.

JavaScript остается одним из самых популярных языков программирования в мире и продолжает эволюционировать, чтобы удовлетворить потребности разработчиков веб-приложений.

1.4 TypeScript

TypeScript – это язык программирования, разработанный Microsoft и основанный на JavaScript, который добавляет статическую типизацию, классы, интерфейсы и другие возможности, чтобы улучшить поддерживаемость и производительность при разработке веб-приложений. TypeScript является суперсетом JavaScript, что означает, что любой действительный код JavaScript также является действительным кодом TypeScript.

TypeScript имеет несколько преимуществ по сравнению с JavaScript, которые делают его более удобным и безопасным для разработки сложных проектов:

1) Статическая типизация: Одно из главных преимуществ TypeScript – это статическая типизация. Это позволяет обнаруживать ошибки на этапе разработки, а не во время выполнения программы. TypeScript обеспечивает поддержку строгой типизации, что означает, что типы данных могут быть определены для каждой переменной, свойства и функции.

2) Улучшенная поддержка IDE: TypeScript позволяет разработчикам использовать различные IDE и редакторы кода, такие как Visual Studio, Visual Studio Code, WebStorm, и другие. Эти инструменты могут предоставить функции автодополнения, предложить варианты исходя из типов данных и даже проверять правильность написания кода.

3) Улучшенная читаемость и поддерживаемость кода: TypeScript позволяет разработчикам создавать более читаемый и поддерживаемый код. Он предоставляет документацию и подсказки, которые упрощают работу над проектом другим разработчикам, а также способствуют легкой поддержке кода после его выпуска.

4) Более высокая скорость разработки: TypeScript предоставляет функции, которые помогают быстрее писать код. Например, TypeScript позволяет использовать конструкторы классов для определения типов данных и свойств. Это упрощает создание новых объектов и методов, что позволяет разработчикам быстрее создавать новый функционал.

5) Поддержка ES6 и новых функций языка: TypeScript поддерживает новые функции JavaScript, такие как Promise, async / await, и декораторы, которые не поддерживаются в более старых версиях JavaScript. Это означает, что TypeScript предоставляет возможность использовать новые функции языка, что помогает создавать более современный код.

6) Поддержка модульной архитектуры: TypeScript предоставляет поддержку модульной архитектуры, что означает, что разработчики могут использовать модули для организации и структурирования своего кода. Это упрощает совместную работу над проектом, а также позволяет избежать конфликтов в именах функций и переменных.

Несмотря на множество преимуществ, TypeScript имеет и некоторые недостатки. Ниже представлены некоторые из них:

1) Обучение. TypeScript является относительно новым языком программирования, и для его освоения может потребоваться время и усилия. Некоторые разработчики могут столкнуться с трудностями в освоении новых концепций и синтаксиса, которые отличаются от традиционного JavaScript.

2) Сложность проекта. TypeScript может добавить некоторую сложность в проект, особенно для менее опытных разработчиков. Некоторые функции и концепции TypeScript могут быть трудными для понимания и могут занимать больше времени на реализацию.

3) Размер файлов. TypeScript добавляет дополнительный слой абстракции, что может привести к увеличению размеров файлов. Это может замедлить время загрузки приложения и увеличить нагрузку на сервер.

4) Необходимость компиляции. TypeScript должен быть скомпилирован в JavaScript перед запуском на клиентской стороне. Это может занять некоторое время и создать дополнительный шаг в процессе разработки.

5) Несовместимость со старым кодом. Если нужно использовать TypeScript в существующем проекте, возможно, потребуется провести значительную работу по конвертации старого кода в TypeScript. Это может быть трудным и затратным процессом.

6) Не подходит для всех проектов. TypeScript не является универсальным решением и не подходит для всех проектов. Если ваш проект маленький или имеет простую архитектуру, то использование TypeScript может быть излишним и неоправданным.

7) Большой порог входа для новых разработчиков. Если нужно нанять новых разработчиков для работы над проектом, которые не имеют опыта работы с TypeScript, то им может потребоваться время, чтобы освоить язык. Это может замедлить развитие проекта и создать дополнительные расходы на обучение.

В языке TypeScript содержатся следующие типы, несвойственные JavaScript, представленные в таблице 1.

Таблица 1 – Типы TypeScript

Тип	Описание
1	2
Any	Неявный тип. По умолчанию язык не выводит ошибки типов для неявно объявленных значений типа any. Благодаря флагу noImplicitAny это можно настроить в tsconfig.json и избежать ошибок типа.

Продолжение таблицы 1

1	2
Unknown	Тип, который представляет значение, о котором ничего не известно. В отличие от типа <code>any</code> , <code>unknown</code> более строго типизирован и не позволяет совершать опасные операции без явного приведения типов или дополнительных проверок.
Tuple	Представляет собой массив фиксированной длины с различными типами элементов.
Enum	Тип данных, который используется для определения набора именованных констант.
Void	Тип данных, который используется для обозначения отсутствия возвращаемого значения функции.
Never	Указывает на то, что функция никогда ничего не вернет. Например, она содержит только исключение или бесконечный цикл.

1.5 ReScript

ReScript (ранее известный как ReasonML) – это язык программирования с открытым исходным кодом, который был разработан для упрощения разработки и поддержки кода JavaScript. Он построен на основе синтаксиса OCaml и может быть скомпилирован в JavaScript.

ReScript имеет ряд преимуществ перед JavaScript и TypeScript. Он предлагает более строгую и безопасную типизацию, что уменьшает количество ошибок в процессе разработки и повышает надежность кода. Кроме того, ReScript поддерживает функциональное программирование и имеет синтаксис, который более лаконичен и понятен, чем у JavaScript.

Одним из ключевых преимуществ ReScript является его типизация. Она строгая, что помогает предотвратить ошибки, связанные с типами данных. Кроме того, в отличие от TypeScript, который может использовать только статические типы, ReScript также поддерживает динамическую типизацию. Это позволяет разработчикам использовать динамические типы данных, когда это необходимо, улучшая гибкость кода.

ReScript также имеет некоторые особенности, которые делают его более удобным для работы с некоторыми типами проектов. Например, ReScript легко интегрируется с React, что делает его отличным выбором для разработки интерфейсов. Кроме того, ReScript поддерживает различные среды выполнения, включая браузеры, Node.js и серверную часть.

ReScript имеет также некоторые недостатки, которые могут снижать его привлекательность для некоторых разработчиков. Например, ReScript является относительно новым языком программирования, что может создавать проблемы с недостатком ресурсов и сообщества. Кроме того, в настоящее время нет поддержки некоторых популярных библиотек и инструментов, что может ограничивать его использование в некоторых проектах.

Тем не менее, ReScript является мощным инструментом для разработки веб-приложений, который предлагает множество преимуществ по сравнению с JavaScript и TypeScript. Он обладает строгой и гибкой типизацией, лаконичным и понятным синтаксисом, а также интеграцией с различными средами выполнения. ReScript также активно развивается и постоянно улучшается благодаря активному сообществу разработчиков, которые работают над расширением его возможностей и улучшением производительности.

1.6 Elm

Elm – это функциональный язык программирования, созданный для разработки веб-приложений с использованием архитектуры Model-View-Update (MVU). Он был создан в 2012 году Эваном Чаплин и получил широкое распространение в сообществе функционального программирования.

Elm является статически типизированным языком, что означает, что типы данных проверяются на этапе компиляции. Это позволяет выявлять многие ошибки до запуска программы, уменьшая возможность ошибок в работающей системе. Кроме того, Elm имеет очень строгую систему типов, которая гарантирует, что программы, написанные на Elm, будут корректно работать в течение всего их жизненного цикла.

Elm также предлагает удобный и понятный синтаксис, который делает код читаемым и легко понятным для других разработчиков. Кроме того, Elm обеспечивает отличную поддержку функционального программирования, что позволяет программистам писать более чистый, более эффективный и более легко поддерживаемый код.

Одной из ключевых особенностей Elm является архитектура Model-View-Update (MVU), которая обеспечивает отделение логики от представления. В MVU все изменения состояния модели происходят внутри функции Update, которая принимает текущее состояние модели и действие, выполняемое пользователем, и возвращает новое состояние модели. После этого новое состояние модели используется для обновления представления в функции View. Такой подход делает Elm легко масштабируемым и поддерживаемым, что особенно важно при разработке крупных приложений.

Хотя Elm имеет множество преимуществ, которые делают его привлекательным для разработки веб-приложений, у него также есть некоторые недостатки. Рассмотрим некоторые из них:

- 1) Небольшое сообщество. Elm является относительно новым языком программирования, и у него нет такого большого сообщества разработчиков, как у JavaScript или TypeScript. Это означает, что может быть сложно найти ответы на вопросы или решения для проблем, с которыми столкнется разработчик в процессе разработки.

- 2) Ограниченный набор библиотек и плагинов. Elm не имеет такого большого экосистемы, как JavaScript или TypeScript, и у него есть ограниченный выбор библиотек и плагинов. Это означает, что разработчик

может столкнуться с проблемами, когда попытаетесь добавить какую-то функциональность, которая не поддерживается существующими библиотеками.

3) Ограниченная поддержка для некоторых функций языка. Elm активно развивается, но на данный момент некоторые функции языка все еще находятся в разработке или отсутствуют вообще. Например, Elm не поддерживает метапрограммирование, что может быть проблемой для некоторых типов приложений.

4) Ограничения на работу с DOM. Elm использует виртуальный DOM для манипуляции элементами на странице, что означает, что он не может работать напрямую с существующими библиотеками JavaScript для манипуляции DOM. Это может быть ограничением для тех, кто уже имеет опыт работы с такими библиотеками и не хочет переучиваться на новый подход.

В целом, Elm – это язык программирования с множеством преимуществ, который может быть хорошим выбором для разработки веб-приложений. Но, как и у любого языка программирования, у него есть свои недостатки, которые нужно учитывать при выборе инструмента для разработки веб-приложений.

1.7 PureScript

PureScript – это функциональный язык программирования, который, как и Elm и Reason, компилируется в JavaScript. Он предоставляет мощный набор инструментов для разработки приложений, основанных на функциональном подходе и имеет множество преимуществ по сравнению с JavaScript.

Одним из главных преимуществ PureScript является строгая статическая типизация, которая позволяет избежать многих ошибок во время выполнения программы. Типы данных в PureScript могут быть полностью инферированы, что упрощает процесс разработки и позволяет снизить количество ошибок. Кроме того, PureScript обладает очень сильной системой типов, которая позволяет более точно определить типы данных и функций, что в свою очередь повышает безопасность программы.

Другим преимуществом PureScript является его синтаксис, основанный на Haskell, который делает язык более читабельным и понятным для программистов, работавших с функциональными языками. Это также означает, что PureScript поддерживает полную функциональную парадигму, включая монады, каррирование, рекурсию и неизменяемые структуры данных.

PureScript также обладает открытым и активным сообществом разработчиков, которые работают над улучшением языка и разработкой новых инструментов. Среди этих инструментов можно выделить PureScript Halogen, которая представляет собой библиотеку для разработки пользовательских интерфейсов и PureScript WebSockets, позволяющую легко работать с WebSockets в PureScript.

У PureScript также есть некоторые недостатки:

1) PureScript является достаточно новым языком программирования, и его экосистема не так широка, как у некоторых других языков. Это может означать, что не всегда можно найти необходимые инструменты или библиотеки, которые могут быть легко доступны в других языках.

2) Для того чтобы использовать PureScript, необходимо иметь знания функционального программирования и основные принципы языка Haskell. Это может создавать некоторые сложности для разработчиков, не имеющих опыта работы с функциональными языками программирования.

В целом, PureScript – это мощный и прогрессивный язык программирования, который обладает рядом преимуществ перед другими языками, используемыми для веб-разработки. Он предлагает строгую и безопасную типизацию, полную функциональность и богатые функциональные конструкции, что делает его идеальным выбором для разработчиков, которые хотят создавать качественные приложения.

1.8 Flow

Flow – это статический типизатор для JavaScript, разработанный командой Facebook. Flow позволяет разработчикам использовать типы данных для обеспечения безопасности и предотвращения ошибок во время

выполнения программы. Он предназначен для использования в существующих проектах на JavaScript, которые могут быть постепенно переведены на более строгую типизацию.

Одним из главных преимуществ Flow является то, что он не требует переписывания всего проекта на другой язык программирования. Он работает с существующим кодом на JavaScript, добавляя типы данных. Flow использует аннотации типов, которые можно добавлять к существующему коду. Это позволяет программистам постепенно добавлять типы данных в проект и тем самым улучшать его качество.

Одним из основных преимуществ Flow является поддержка разных уровней строгости типизации. Например, если программист не знает тип данных, он может пометить его как `any`. Это означает, что Flow не будет выдавать ошибки для этого значения, но также не сможет предоставить полезную информацию о нем. Если тип данных известен, программист может указать его явно. Таким образом, можно контролировать уровень строгости типизации в проекте.

Flow также обладает высокой производительностью, что является важным фактором для больших проектов. Он выполняет проверку типов в фоновом режиме и предоставляет быструю обратную связь по ошибкам. Flow позволяет настроить глубину анализа, что позволяет контролировать время, затрачиваемое на проверку типов.

У Flow есть и некоторые недостатки. Один из главных недостатков – это то, что его аннотации типов могут быть громоздкими и трудными для понимания. Это может привести к тому, что код станет менее читабельным и трудным для сопровождения.

Кроме того, Flow может потребовать дополнительных усилий в процессе настройки и поддержки. В частности, необходимо убедиться, что все используемые библиотеки поддерживают Flow и что они правильно описывают свои типы данных.

В целом, Flow – это мощный инструмент для обеспечения безопасности

и стабильности JavaScript-кода. Он позволяет добавлять типы данных в существующие проекты, что может значительно уменьшить количество ошибок и улучшить читаемость кода.

1.9 Сравнение инструментов для статической типизации

По данным статистики Stack Overflow Developer Survey 2023 40.08% из 53,421 опрошенных профессиональных разработчиков используют язык TypeScript. Если рассмотреть диаграмму популярности языков, представленную GitHub по итогам 2023 года (рисунок 1), то с каждым годом популярность языка растет и на прошедший год язык TypeScript занимает 4 место.

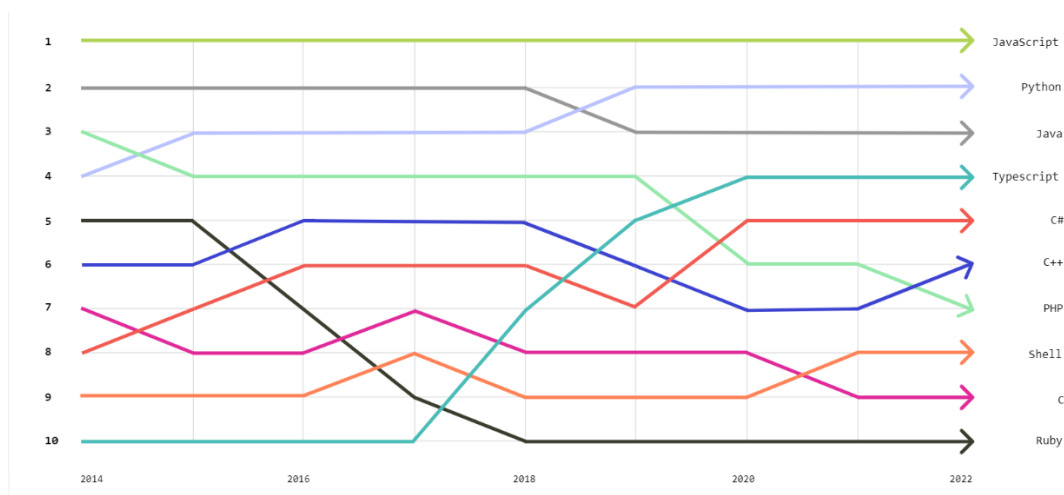


Рисунок 1 – Популярность языков

То, что язык с каждым годом используется все больше, подтверждает статистика NPM (менеджер пакетов, входящий в состав Node.JS). На основе данных NPM Trends, можно сделать выводы о развитии распространенности пакетов. Так, TypeScript однозначно набирает популярность на протяжении последних пяти лет (рисунок 2).

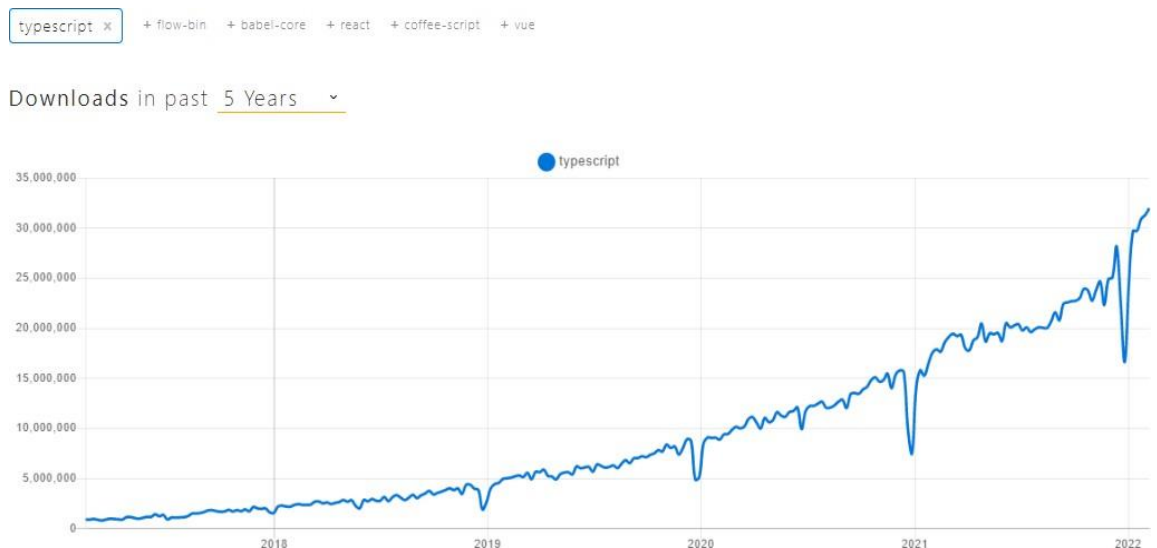


Рисунок 2 – Установка пакета TypeScript

Что нельзя сказать о Flow, распространение которого слабо меняется и число установок падает в последний год (рисунок 3).

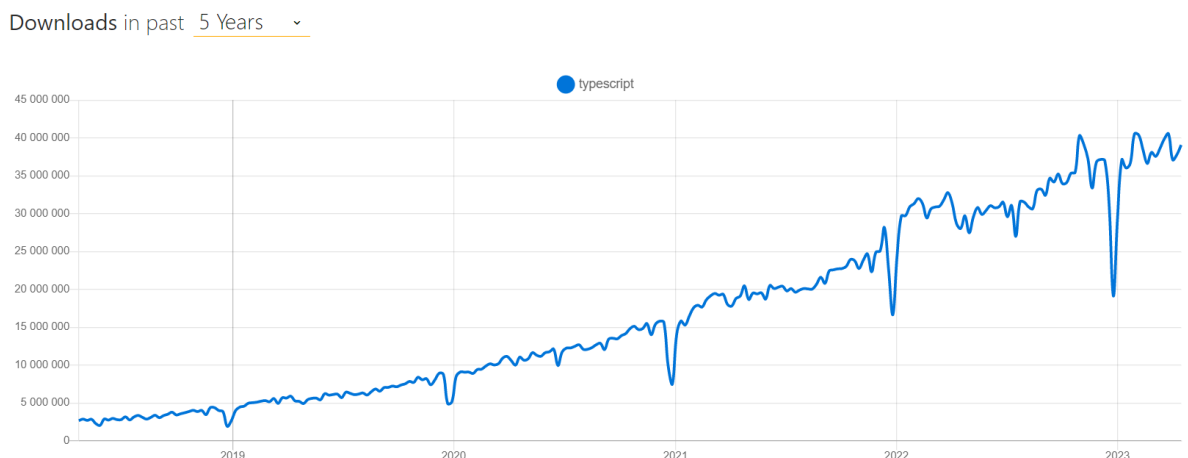


Рисунок 3 – Установка пакета Flow

В отчете State of Frontend 2022 представлены данные о том, что 77,2% респондентов уже используют TypeScript, и 53% из них предпочитают его JavaScript, который является самым популярным языком программирования в разных рейтингах в последние годы.

Согласно статистике, State of JS 2022, начиная с 2016 года и по 2022 год TypeScript занимает лидирующее место среди компилирующихся в JavaScript языков и процент его использования с каждым годом растет. Также 92,96% использующих TypeScript довольны им. Второе место занимает Elm, но разрыв в процентах между ним и TypeScript на 2022 год составил 72%. В пятерку лидеров также входят языки со статической типизацией ReScript, PureScript

(рисунок 4).

Downloads in past 2 Years ▾

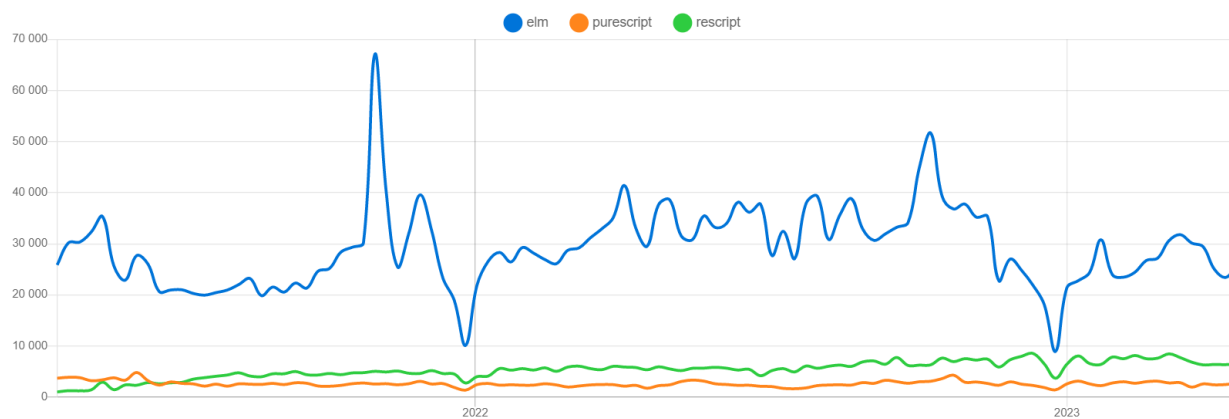


Рисунок 4 – Количество скачиваний Elm, PureScript и ReScript за 2 года

2 РАЗРАБОТКА СИСТЕМЫ КРИТЕРИЕВ ДЛЯ СРАВНЕНИЯ ИНСТРУМЕНТОВ

2.1 Разработка критериев

В результате проведенного обзора инструментов для статической типизации и анализа исследований российских и зарубежных ученых были определены слабые и сильные стороны каждого из рассмотренных инструментов. Каждый из данных инструментов решает проблему статической типизации, но также имеет ряд существенных недостатков. Чтобы эффективно использовать данные инструменты, необходимо понимать их различия и знать, когда и какой из них использовать. Для этого необходимо определить критерии сравнения и произвести сравнительный анализ данных инструментов для статической типизации.

2.1.1 Потребление ресурсов процессора

Критерий (рисунок 5) показывает процентное использование ресурсов процессора за определенный период при выполнении веб-приложения. Чем меньше процент использования ресурсов процессора, тем более эффективно оптимизировано приложение. Для улучшения этого показателя можно использовать оптимизацию обработки больших объемов данных, например, массивов, а также применять современные методы и технологии оптимизации производительности. Некоторые языки программирования, такие как ReScript, имеют встроенную оптимизацию производительности и могут быть более эффективными в использовании ресурсов процессора, чем другие языки.

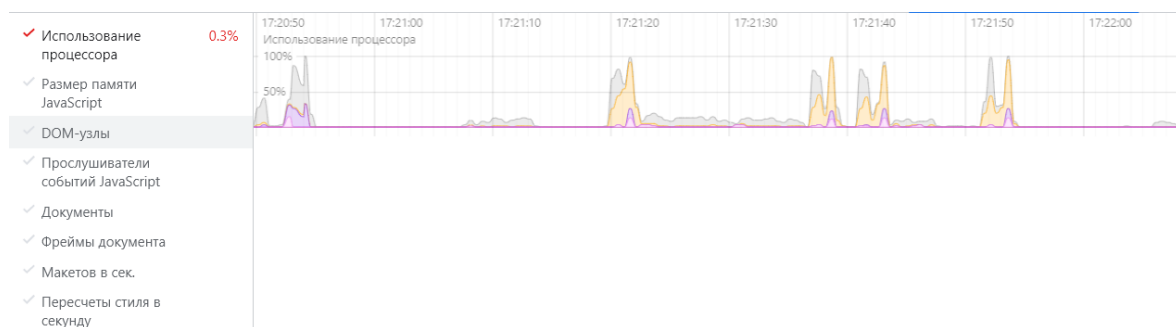


Рисунок 5 – График использования процессора

2.1.2 Потребление оперативной памяти

Данный критерий (рисунок 6) оценивает количество оперативной

памяти, которое затрачивает приложение во время работы. Избыточное потребление оперативной памяти может привести к сбоям приложения и ограничить его производительность. Путем сравнения можно выявить, насколько эффективно приложение использует оперативную память и какие структуры данных занимают больше места в памяти приложений, созданных с использованием разных языков программирования.

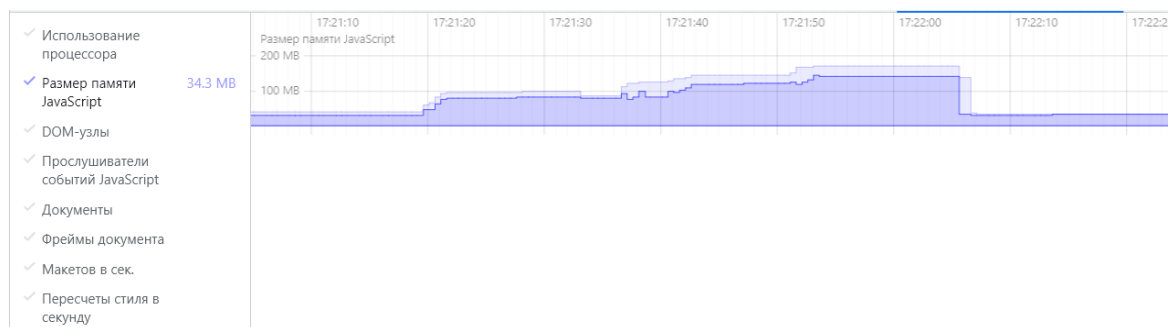


Рисунок 6 – График использования оперативной памяти

2.1.3 Время выполнения

Время выполнения программы может зависеть от многих факторов, включая эффективность алгоритмов и структур данных, оптимизации компилятора или интерпретатора, а также характеристики самого языка программирования.

Например, язык программирования C++ известен своей скоростью выполнения благодаря компиляции в машинный код, а Python обычно работает медленнее из-за своей интерпретации. Однако, существует множество фреймворков и библиотек, которые могут улучшить производительность на определенном языке.

При сравнении TypeScript, ReScript, PureScript и Elm по этому критерию, необходимо учитывать, что все эти языки компилируются в JavaScript, что может замедлить работу программы. Тем не менее, каждый язык имеет свои собственные оптимизации и инструменты, которые могут повысить производительность. Скорость работы языков зависит от нескольких факторов.

Во-первых, это производительность скомпилированного JavaScript-кода, сгенерированного компилятором каждого языка. TypeScript, ReScript,

PureScript и Elm компилируются в JavaScript, и скорость работы зависит от производительности этого скомпилированного кода. При этом можно отметить, что все эти языки генерируют достаточно эффективный код, который позволяет приложениям работать быстро.

Во-вторых, скорость работы может зависеть от оптимизаций и улучшений, внедряемых в компиляторы языков. Например, ReScript использует под капотом собственную систему сборки Ninja, которая позволяет повысить производительность скомпилированного кода и ускорить его работу. Также, ReScript имеет множество оптимизаций, таких как эффективный сбор мусора и оптимизированную работу с циклами, что дополнительно повышает скорость работы программы. PureScript также предоставляет различные оптимизации, такие как оптимизация размера стека и эффективное управление памятью, чтобы обеспечить быструю работу приложений.

В-третьих, скорость работы может зависеть от специфических особенностей языков. Например, Elm имеет собственную систему виртуального DOM, что позволяет минимизировать количество изменений в реальном DOM и ускорить работу приложения.

Таким образом, при оценке скорости работы языков TypeScript, ReScript, PureScript и Elm, следует учитывать несколько факторов, включая производительность скомпилированного JavaScript-кода, оптимизации и улучшения компиляторов, а также специфические особенности каждого языка. В целом, все эти языки обеспечивают высокую скорость работы приложений.

2.1.4 Вес скомпилированного кода

Критерий оценивает размер скомпилированного кода для каждого языка программирования. Это важный критерий для фронтенд-разработки, где размер скомпилированного кода может иметь прямое влияние на производительность и скорость загрузки веб-страницы.

Для языков TypeScript, ReScript, PureScript и Elm размер

скомпилированного кода может варьироваться в зависимости от многих факторов, таких как использование сторонних библиотек, наличие неиспользуемого кода и оптимизации компилятора.

Все эти языки компилируются в JavaScript, который выполняется в браузере или на сервере. При этом у каждого языка есть свой набор оптимизаций, который позволяет уменьшить количество скомпилированного кода.

2.1.5 Время компиляции

Этот критерий определяет, сколько времени занимает процесс компиляции программного кода на целевой машине. Чем меньше время компиляции, тем быстрее можно получить отклик от программы и быстрее можно выполнять изменения в коде.

TypeScript, ReScript, PureScript и Elm—это все языки программирования, которые компилируются в JavaScript, и, следовательно, время компиляции зависит от сложности и размера проекта, а также от используемого компилятора.

TypeScript—компилируется с помощью TypeScript Compiler (tsc). Он может быть запущен из командной строки или настроен в качестве плагина для сред разработки. Время компиляции TypeScript зависит от количества файлов, размера проекта и настроек компилятора. В целом, TypeScript обеспечивает быстрое время компиляции, особенно при использовании инкрементальной компиляции.

ReScript – компилируется с помощью ReScript Compiler (rescript). Он использует более эффективный алгоритм сравнения файлов для определения того, какие файлы нуждаются в перекомпиляции. Это позволяет ReScript обеспечивать очень быстрое время компиляции, даже для больших проектов.

PureScript—компилируется с помощью PureScript Compiler (psc). Это компилятор на основе Haskell, который может быть запущен из командной строки или настроен в качестве плагина для среды разработки. Время компиляции PureScript зависит от сложности проекта и используемых

библиотек. PureScript может компилировать большие проекты за несколько секунд.

Elm – компилируется с помощью Elm Compiler (elm-make). Он может быть запущен из командной строки или настроен в качестве плагина для среды разработки. Elm Compiler обеспечивает быстрое время компиляции, но процесс может затянуться для больших проектов с большим количеством зависимостей.

Таким образом, все эти языки обеспечивают быструю и эффективную компиляцию, но время компиляции может варьироваться в зависимости от сложности проекта и используемых инструментов. ReScript и PureScript обеспечивают особенно быструю компиляцию, благодаря своим современным алгоритмам компиляции, а также меньшему количеству зависимостей.

2.1.6 Постепенная типизация

Критерий относится к языкам программирования, которые позволяют добавлять типы данных постепенно, по мере необходимости.

В языках программирования с постепенной типизацией разработчики могут выбирать, когда и как использовать типы данных, вместо того чтобы жестко требовать их использование в каждой части программы. Такая возможность позволяет постепенно типизировать старую кодовую базу без типов.

TypeScript–язык программирования, который обеспечивает лучшую поддержку постепенной типизации по сравнению с другими языками, такими как ReScript, PureScript и Elm, в которых каждое значение требует строгого определения типов.

2.1.7 Популярность

Данный критерий является важным при сравнении языков программирования, так как он показывает, насколько широко язык используется в различных проектах и сообществах разработчиков.

Существуют различные способы измерения популярности языка программирования. Один из таких способов–это индексы популярности,

которые компании и сайты составляют на основе статистических данных, таких как количество запросов в поисковых системах, количество проектов на GitHub, количество вакансий на сайтах поиска работы и т.д. Некоторые из таких индексов популярности включают Tiobe Index, PYPL Index, GitHub Stars, Stack Overflow Developer Survey и др.

В контексте сравнения TypeScript, ReScript, PureScript и Elm, можно провести сравнение их популярности на основе таких индексов. Например, Tiobe Index за апрель 2023 года показывает, что TypeScript занимает 20-е место в списке самых популярных языков программирования, а PureScript и Elm не входят в топ-50. ReScript не учитывается в этом индексе. GitHub Stars также могут дать представление о популярности проектов на GitHub, на таблице 2 представлены результаты сравнения.

Таблица 2 – Популярность языков программирования

Язык программирования	Количество звезд
TypeScript	90 657
PureScript	8102
Elm	7163
ReScript	6118

Следует помнить, что популярность не является главным критерием для выбора языка программирования. Она может быть важна при выборе из нескольких альтернативных языков, но не является определяющей при выборе подходящего языка для конкретной задачи.

2.1.8 Инструменты и экосистема

Инструменты и экосистема – это критерий, который оценивает наличие и качество инструментов разработки и поддержки, а также наличие и разнообразие библиотек и фреймворков в экосистеме языка программирования.

TypeScript – один из наиболее популярных языков программирования,

имеет широкую экосистему, поддерживаемую крупными компаниями, такими как Microsoft. Он имеет богатый набор инструментов разработки, включая плагины для IDE, такие как Visual Studio Code, и инструменты сборки, такие как Webpack. TypeScript также имеет большое сообщество разработчиков, которые активно создают и поддерживают библиотеки и фреймворки, такие как Angular и Nest.js.

ReScript – новый язык программирования, который активно развивается и имеет небольшую экосистему, но имеет поддержку от компании Bloomberg. У него есть собственная система сборки и компиляции, а также интеграция с существующими библиотеками и фреймворками JavaScript. ReScript также имеет растущее сообщество разработчиков и постоянно добавляет новые инструменты и библиотеки.

PureScript – функциональный язык программирования, имеет небольшую, но активно развивающуюся экосистему. PureScript имеет свою систему сборки и компиляции, а также интеграцию с библиотеками и фреймворками JavaScript. Он также имеет поддержку от сообщества разработчиков, которые создают и поддерживают библиотеки и фреймворки, такие как Halogen и PureScript React.

Elm – функциональный язык программирования, разработанный для создания веб-приложений. Он имеет собственную систему сборки и компиляции, а также библиотеки и фреймворки для создания веб-приложений, такие как Elm UI и Elm Bootstrap. Elm имеет небольшую, но активную экосистему, которая постоянно растет благодаря сообществу разработчиков.

Таким образом, все языки программирования – TypeScript, ReScript, PureScript и Elm, имеют собственную экосистему и поддержку от сообщества разработчиков. TypeScript имеет наиболее широкую экосистему и множество инструментов разработки, таких как Visual Studio Code, WebStorm и другие IDE. Он также имеет обширную документацию и большое количество библиотек и плагинов, что делает его очень привлекательным для использования в больших проектах.

2.1.9 Документация и материалы для обучения

Критерий оценивает наличие и качество документации и обучающих материалов, которые помогают новым и опытным разработчикам понимать и использовать языки программирования.

TypeScript имеет обширную документацию, которая содержит подробные описания языка, стандартной библиотеки и различных инструментов. Также имеются курсы обучения и множество статей и видеоуроков, доступных онлайн. Благодаря широкой популярности TypeScript, в интернете можно найти большое количество обучающих материалов на разных языках.

ReScript, PureScript и Elm также имеют документацию и обучающие материалы, но они более ограничены в сравнении с TypeScript. Однако, каждый язык имеет свои уникальные особенности и преимущества, которые можно изучить в документации и обучающих материалах.

Кроме того, все эти языки имеют активные сообщества пользователей, которые создают и обновляют материалы для обучения, а также помогают другим разработчикам решать проблемы и находить ответы на вопросы.

2.2 Выделенные критерии

В таблице 3 приводится обзор каждого критерия с обоснованием его важности для включения в анализ.

Таблица 3 – Итоговая таблица критериев

Критерий	Описание
1	2
Потребление ресурсов процессора	Процент использования ресурсов процессора программой. Чем ниже значение – тем эффективнее работа
Потребление оперативной памяти	Количество оперативной памяти, используемое приложением. Чем ниже значение – тем эффективнее работа

Продолжение таблицы 3

1	2
Время выполнения	Критерий оценивает, насколько быстро может выполняться код на определенном языке
Размер скомпилированного кода	Критерий оценивает размер скомпилированного кода для языка программирования
Время компиляции	Критерий определяет, сколько времени занимает процесс компиляции программного кода. Чем меньше время компиляции, тем быстрее можно получить отклик от программы и выполнить изменения в коде
Популярность	Критерий показывает, насколько широко язык используется в различных проектах и сообществах разработчиков
Инструменты и экосистема	Возможные вспомогательные инструменты, используемые в разработке приложения
Документация и материалы для обучения	Критерий оценивает наличие и качество документации и обучающих материалов, которые помогают новым и опытным разработчикам понимать и использовать язык программирования

Продолжение таблицы 3

1	2
Постепенная типизация	Критерий относится к языкам программирования, которые позволяют добавлять типы данных постепенно, по мере необходимости

3 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРИЛОЖЕНИЙ

3.1 Проектирование приложения

Для проведения эксперимента были выбраны языки TypeScript, PureScript, ReScript и Elm. Все языки поддерживают статическую типизацию и компилируют свой код в JavaScript. В рамках эксперимента будет создано 4 идентичных веб-приложений, написанных на разных языках программирования, но с использованием одного инструмента для создания клиентской части (React). Для измерения скорости работы кода профайлер Chrome Dev Tools. Веб-приложение будет представлять собой обыкновенный счетчик (рисунок 7).



Рисунок 7 – Веб-приложение

Функциональные требования приложения будут следующими:

- инкрементировать значение счетчика,
- декрементировать значение счетчика.

3.2 Разработка приложений

3.2.1 TypeScript

TypeScript – самый популярный инструмент для статической типизации JavaScript кода. Его основное преимущество - это то, что он представляет собой надмножество, которое компилируется в JavaScript — хотя по ощущениям TypeScript похож на новый язык со статической типизацией сам по себе. Также этот инструмент не очень сложен в изучении, что несомненно является плюсом.

Структура проекта представлена на рисунке 8.

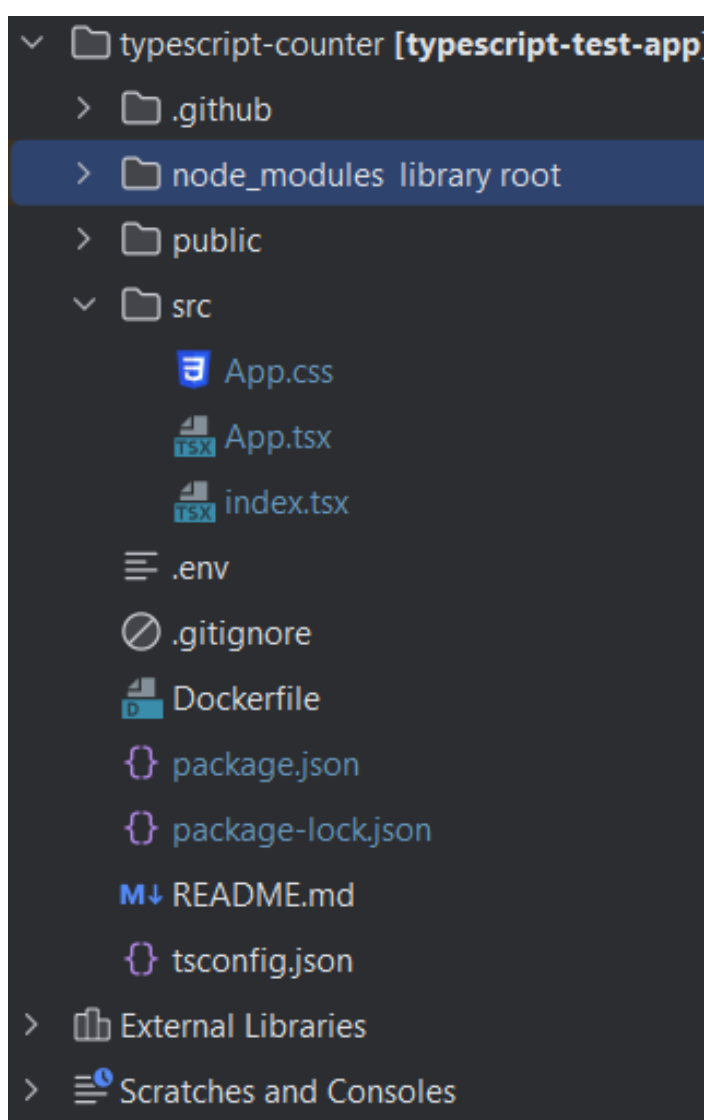


Рисунок 8 – Структура проекта

Код счетчика представлен на рисунке 9.

```

1 import React, { useState } from 'react';
2 import './App.css';
3
4 function App() : JSX.Element {
5   const [counter : number , setCounter : React.Dispatch<React.SetStateAction<number>> ] = useState( initialState: 0 );
6
7   const onIncrement = () => setCounter( value: counter + 1 )
8   const onDecrement = () => setCounter( value: counter - 1 )
9
10  return (
11    <div className="app">
12      <button onClick={onDecrement}>-</button>
13      <p>{counter}</p>
14      <button onClick={onIncrement}>+</button>
15    </div>
16  );
17 }
18
19 export default App;

```

Рисунок 9 – Код счетчика на TypeScript

3.2.2 ReScript

ReScript – это не новый язык, а новый синтаксис и набор инструментов для проверенного временем языка OCaml. Reason предоставляет синтаксис, ориентированный на JavaScript-программистов, и использует уже известный всем способ распространения через NPM/Yarn.

Процесс создания приложения:

- 1) установить компилятор ReScript и настроить окружение разработки,
- 2) создать новый проект с помощью команды `rescript init` и выбрать опцию "Empty" для создания пустого проекта,
- 3) установить необходимые зависимости для работы с веб-интерфейсами, например, `rescript-react`,
- 4) создать файлы для компонентов счетчика и связанных с ними функций,

5) реализовать функциональность компонентов: отображение текущего значения счетчика, кнопки для увеличения и уменьшения значения, обработчики событий для этих кнопок,

6) собрать приложение с помощью команды `rescript build` и запустить на локальном сервере.

Структура проекта представлена на рисунке 10.

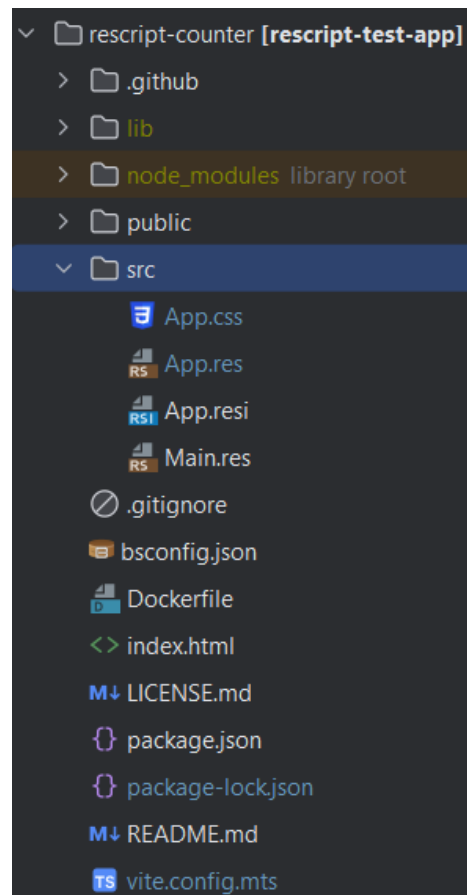


Рисунок 10 – Структура проекта

Код счетчика представлен на рисунке 11.

```
1  %%raw(`import './App.css'`)
2
3  @react.component
4  let make = () => {
5    let (count, setCount) = React.useState(() => 0)
6
7    <div className="app">
8      <button onClick={_e => setCount(count => count - 1)}> {'-'->React.string} </button>
9      <p> {'${count->Belt.Int.toString}'->React.string} </p>
10     <button onClick={_e => setCount(count => count + 1)}> {'+'->React.string} </button>
11   </div>
12 }
13
```

Рисунок 11 – Веб-приложение

3.2.3 PureScript

PureScript – строго типизированный язык программирования с выразительными типами, который компилируется в JavaScript. Он вдохновлен языком Haskell и на нем же написан.

Установка необходимых инструментов: PureScript компилируется в JavaScript, поэтому понадобится Node.js и npm. Также необходимо установить компилятор PureScript и пакетный менеджер для PureScript – spago.

Процесс создания приложения:

1) создание проекта: новый проект можно создать с помощью команды `spago init`, которая создаст структуру проекта и установит необходимые зависимости,

2) определение модели: модель представляет собой состояние нашего приложения. Модель будет состоять из одного целочисленного значения – счетчика,

3) определение действий: действия описывают возможные изменения состояния нашего приложения. Будет определено два действия: увеличение счетчика и уменьшение счетчика,

4) определение обработчиков событий: обработчики событий отвечают за реагирование на пользовательский ввод и вызов соответствующих действий. Будет определено два обработчика: один для увеличения счетчика и один для уменьшения счетчика,

5) определение отображения: отображение определяет, как состояние нашего приложения будет отображаться на экране. В разрабатываемом приложении будет выводиться текущее значение счетчика на экран,

6) компилирование кода с помощью `spago` и запуск полученного JavaScript-кода в браузере.

Структура проекта представлена на рисунке 12.

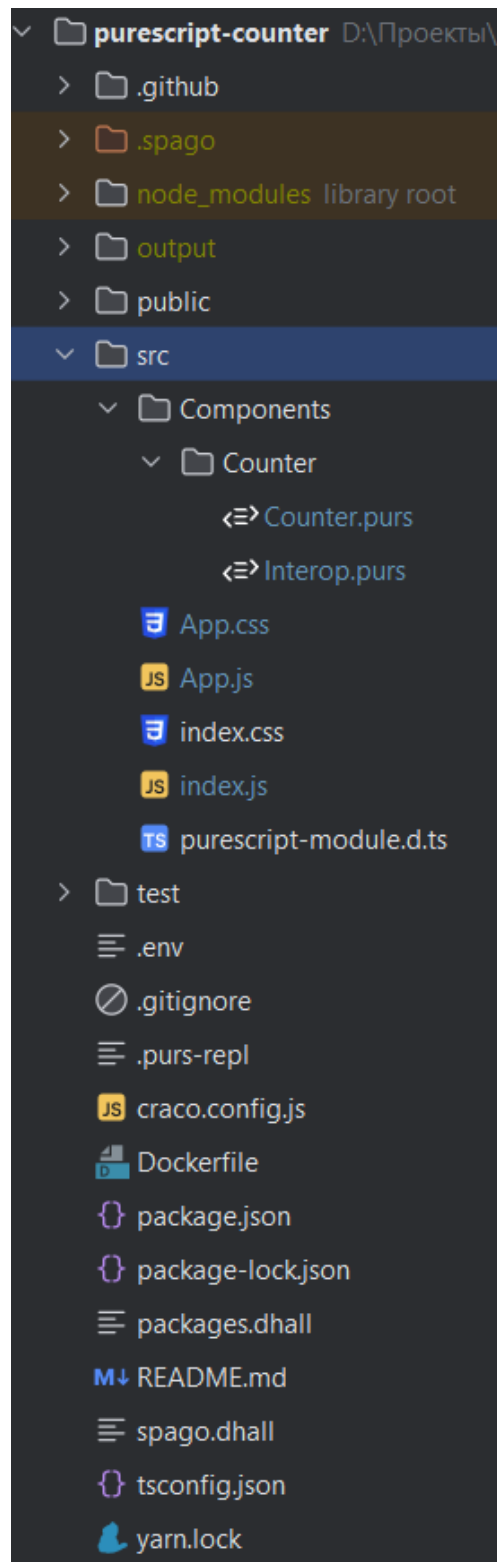


Рисунок 12 – Структура проекта

Код счетчика представлен на рисунках 13 и 14.

```

1 module Components.Counter where
2
3 import Prelude
4
5 import Data.Interpolate (i)
6 import Data.Maybe (Maybe(..))
7 import Data.Tuple.Nested ((/\))
8 import Effect (Effect)
9 import React.Basic.DOM (button, div_, p_, text)
10 import React.Basic.Events (EventHandler, handler_)
11 import React.Basic.Hooks (Component, component, useState')
12 import React.Basic.Hooks as React
13
14 type Props =
15   { label :: String
16   , onClick :: Int -> Effect Unit
17   , counterType :: CounterType
18   }
19
20 data CounterType = Increment | Decrement
21
22 counterTypeToString :: CounterType -> String
23
24 counterTypeFromString :: String -> Maybe CounterType
25 counterTypeFromString = case _ of
26   "incrementer" -> Just Increment
27   "decrementer" -> Just Decrement
28   _ -> Nothing

```

Рисунок 13 – Веб-приложение

```

30 mkCounter :: Component Props
31 mkCounter = component "Counter" \props -> React.do
32   count /\ setCount <- useState' 0
33   pure do
34     div_
35       [ button
36         { onClick: onClickHandler count Decrement setCount
37         , children: [ text "-" ]
38         } ,
39         p_ [text $ i count]
40       , button
41         { onClick: onClickHandler count Increment setCount
42         , children: [ text "+" ]
43         }
44     ]
45
46 onClickHandler :: (Int) -> (CounterType) -> (Int -> Effect Unit) -> EventHandler
47 onClickHandler count counterType setCount = handler_ do
48   let next = step count counterType
49   setCount next
50
51 step :: Int -> CounterType -> Int
52 step n counterType = case counterType of
53   Increment -> n + 1
54   Decrement -> n - 1

```

Рисунок 14 – Веб-приложение

3.2.4 Elm

Elm – это функциональный язык программирования со строгой типизацией для создания реактивных веб-приложений. Elm компилируется в JavaScript и исполняется в браузере привычным образом. В этом языке нет runtime exceptions, если проект собрался — значит он работает.

Процесс создания приложения:

- 1) установить Elm и настроить среду разработки. Elm может быть установлен через npm или скачан в виде дистрибутива с официального сайта,
- 2) создать новый Elm-проект. Для этого необходимо воспользоваться командой `elm init` в терминале,
- 3) определить модель данных. Модель данных в Elm представляет собой определение типа данных, которые будут использоваться в приложении. Для счетчика модель будет содержать единственное целочисленное поле,
- 4) определить сообщения. В Elm приложение реагирует на входящие сообщения, которые изменяют модель данных. Для счетчика сообщения могут быть "увеличить счетчик" и "уменьшить счетчик",
- 5) определить обновление модели. Функция `update` в Elm принимает текущую модель данных и сообщение, и возвращает обновленную модель. В случае счетчика, обновление будет состоять в изменении значения поля в модели в зависимости от полученного сообщения,
- 6) определить представление. В Elm представление отвечает за отображение модели на странице. Для счетчика представление может быть простым HTML-кодом с двумя кнопками и значением счетчика,
- 7) связать модель, сообщения, обновление и представление в одно целое. Для этого используется функция `program`, которая принимает все описанные выше компоненты и возвращает Elm-приложение,
- 8) запустить приложение. Для этого необходимо вызвать функцию `init` с начальной моделью и передать результат в функцию `program`. Это запустит приложение и отобразит его на странице.

Структура проекта представлена на рисунке 15.

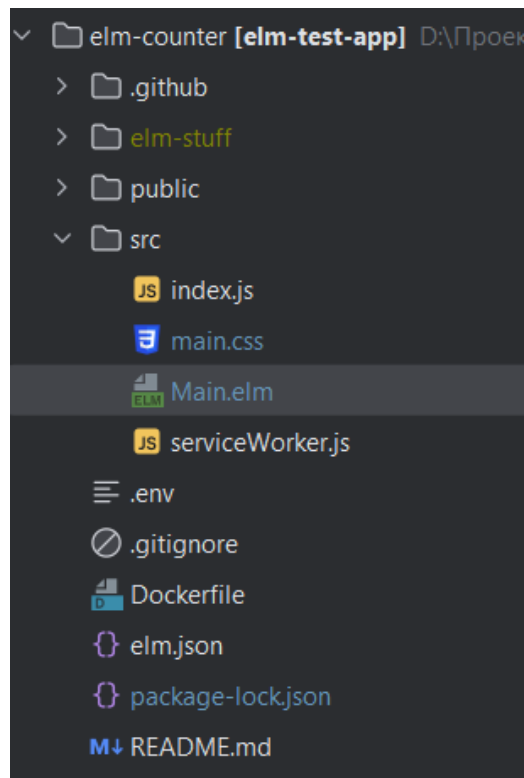


Рисунок 15 – Структура проекта

Код счетчика представлен на рисунке 16.

```

1  module Main exposing (..)
2  import Browser
3  import Html exposing (Html, button, div, text, p)
4  import Html.Events exposing (onClick)
5
6  main = Browser.sandbox { init = init, update = update, view = view }
7
8  type alias Model = Int
9
10 init : Model
11 init = 0
12
13 type Msg = Increment | Decrement
14
15 update : Msg -> Model -> Model
16 update msg model =
17   case msg of
18     Increment ->
19       model + 1
20     Decrement ->
21       model - 1
22
23 view : Model -> Html Msg
24 view model =
25   div []
26     [
27       button [ onClick Decrement ] [ text "-" ],
28       p [] [ text (String.fromInt model) ],
29       button [ onClick Increment ] [ text "+" ]
30     ]

```

Рисунок 16 – Веб-приложение

3.3 Сбор данных о производительности

3.3.1 TypeScript

Результат профилирования скорости выполнения кода на языке TypeScript представлен в таблице 4, на рисунке 18 можно увидеть flame graph производительности веб-приложения. Среднее время выполнения кода составило 208.6 мс.

Таблица 4 – Измерения скорости выполнения кода на TypeScript

№ эксперимента	Время выполнения, мс
1	203
2	218
3	216
4	201
5	201
6	203
7	218
8	216
9	201
10	201

График потребления оперативной памяти можно увидеть на рисунке 17. В среднем потребление памяти составляет 9.6 МБ.

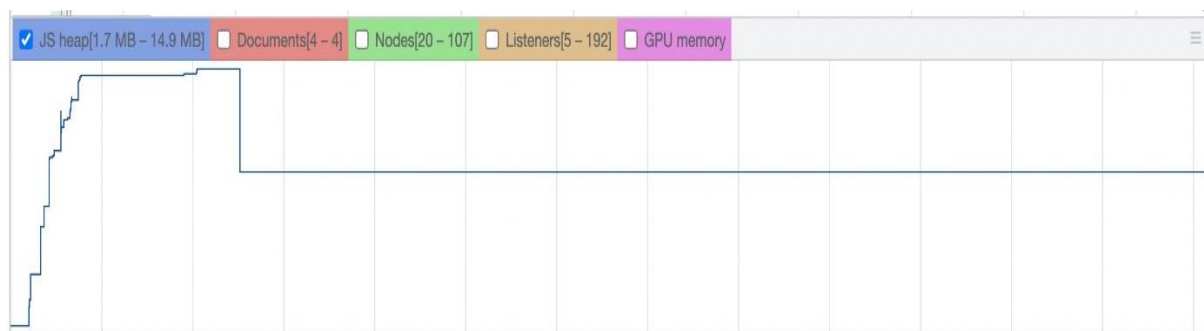


Рисунок 17 – График потребления памяти приложения на TypeScript

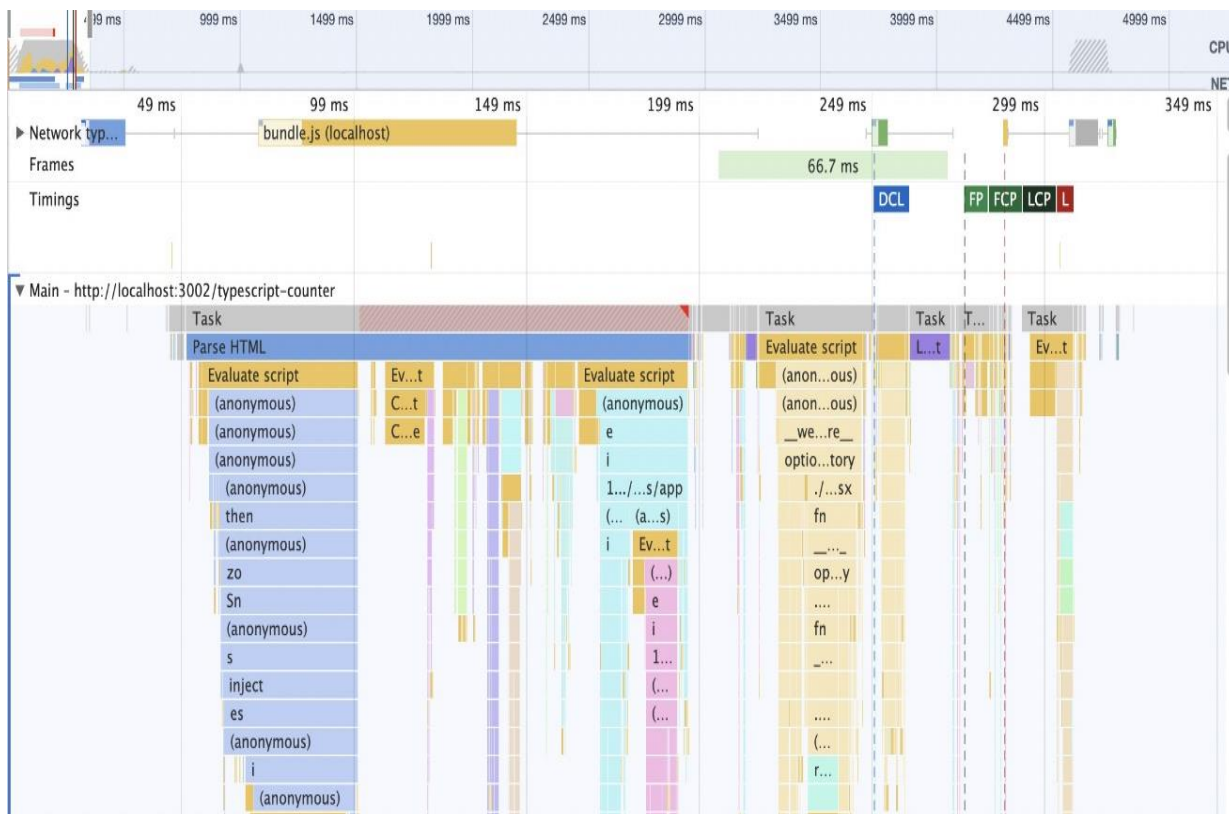


Рисунок 18 – Flame graph приложения на TypeScript

Использование процессора составило 0.25%. Результат представлен на рисунке 19.

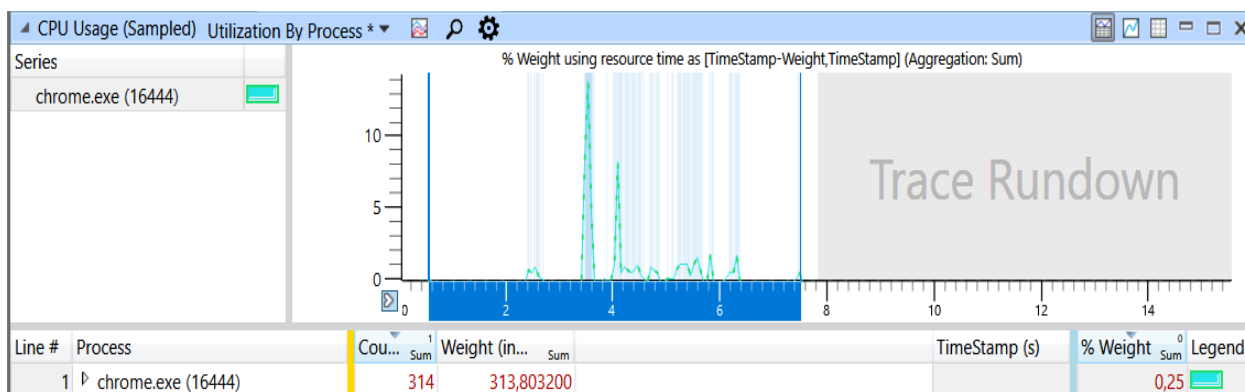


Рисунок 19 – Результат использования процессора приложением на TypeScript

3.3.2 ReScript

Результат профилирования скорости выполнения кода на языке ReScript представлен в таблице 5, на рисунке 21 можно увидеть flame graph производительности веб-приложения. Среднее время выполнения кода составило 155.2 мс.

Таблица 5 – Измерения выполнения кода на ReScript

№ эксперимента	Время выполнения, мс
1	159
2	155
3	147
4	162
5	153
6	159
7	155
8	147
9	162
10	153

График потребления оперативной памяти можно увидеть на рисунке 20. В среднем потребление памяти составляет 13.3 МБ.

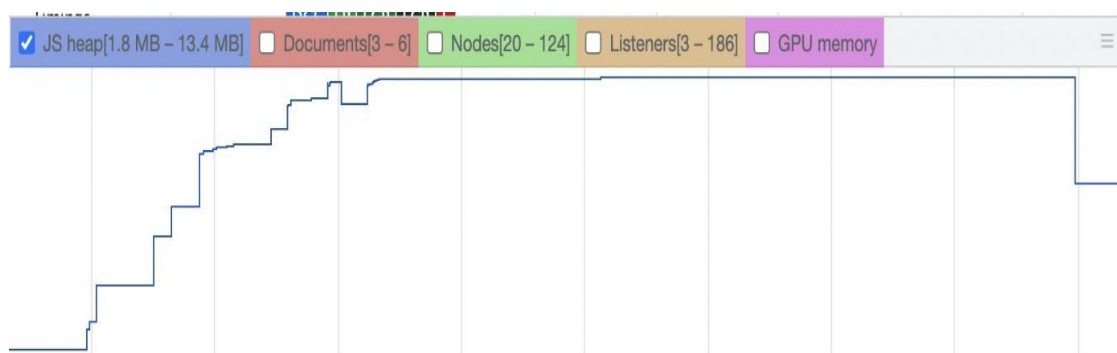


Рисунок 20 – График потребления памяти приложения на ReScript

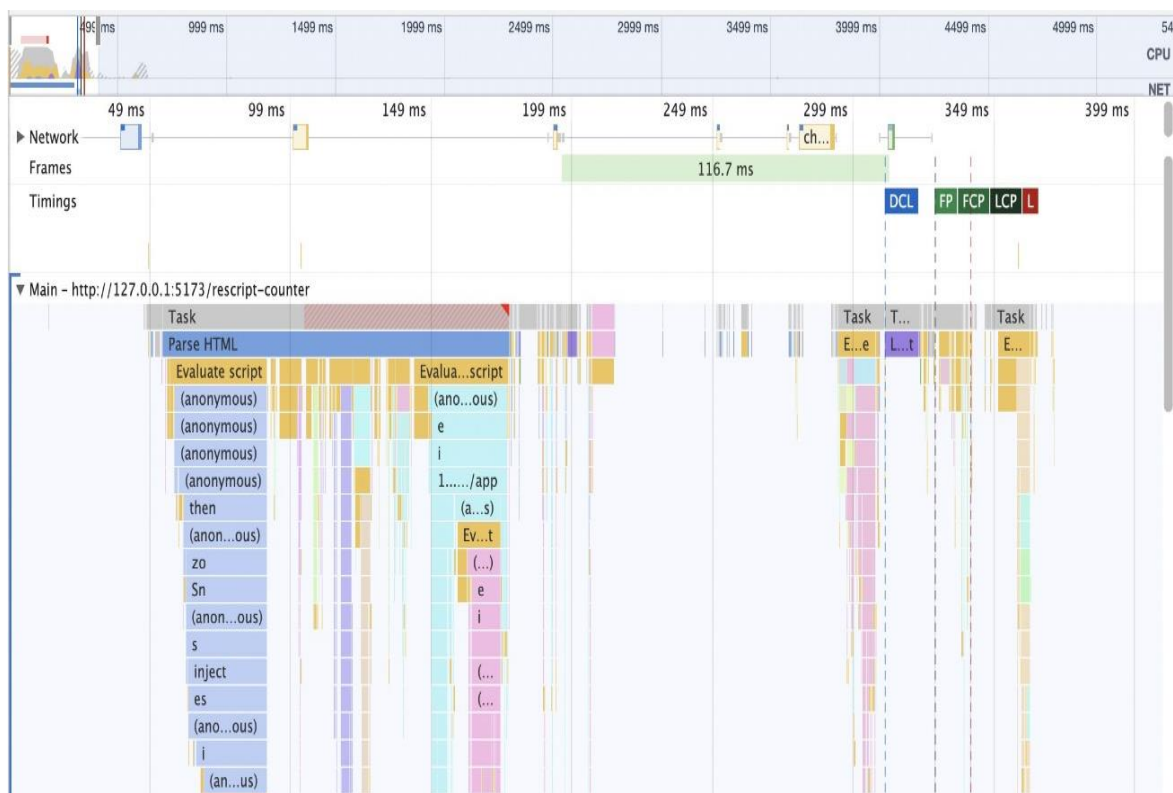


Рисунок 21 – Flame graph приложения на ReScript

Использование процессора составило 0.44%. Результат представлен на рисунке 22.

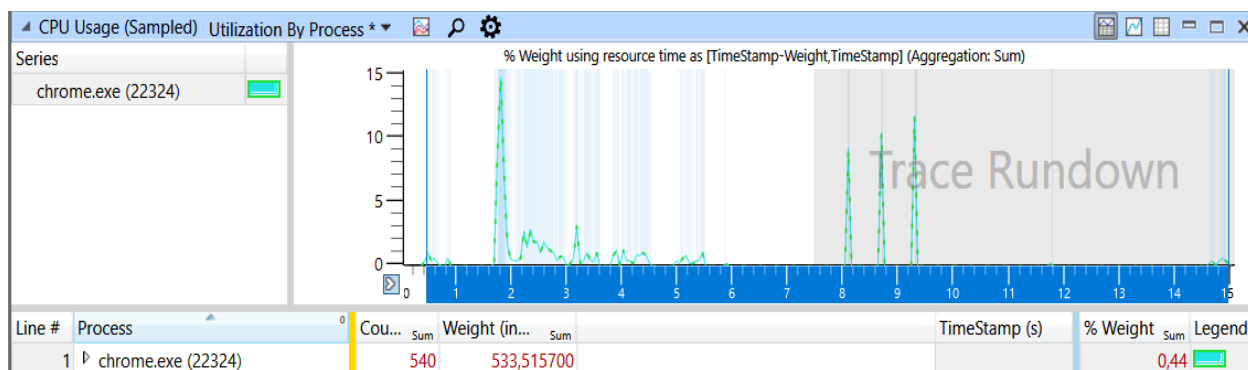


Рисунок 22 – Результат использования процессора приложением на ReScript

3.3.3 PureScript

Результат профилирования скорости выполнения кода на языке PureScript представлен в таблице 6, на рисунке 23 можно увидеть flame graph производительности веб-приложения. Среднее время выполнения кода составило 170.8 мс.

График потребления оперативной памяти можно увидеть на рисунке 22. В среднем потребление памяти составляет 11 МБ.

Таблица 6 – Измерения выполнения кода на PureScript

№ эксперимента	Время выполнения, мс
1	168
2	169
3	167
4	161
5	189
6	168
7	169
8	167
9	161
10	189

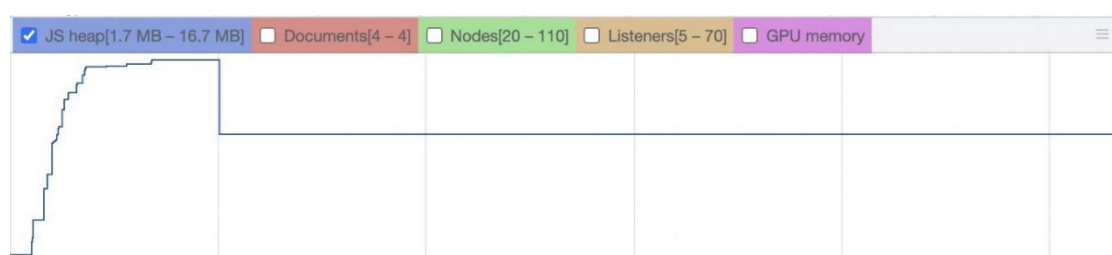


Рисунок 23 – График потребления памяти приложения на PureScript

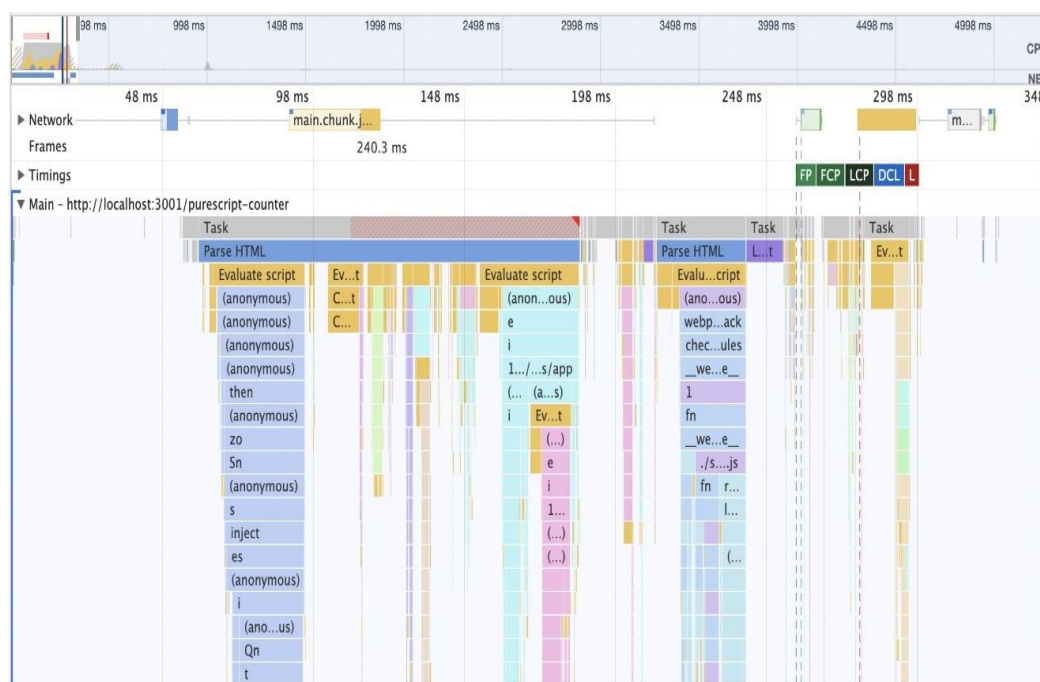


Рисунок 24 – Flame graph приложения на PureScript

Использование процессора составило 0.83%. Результат представлен на рисунке 25.

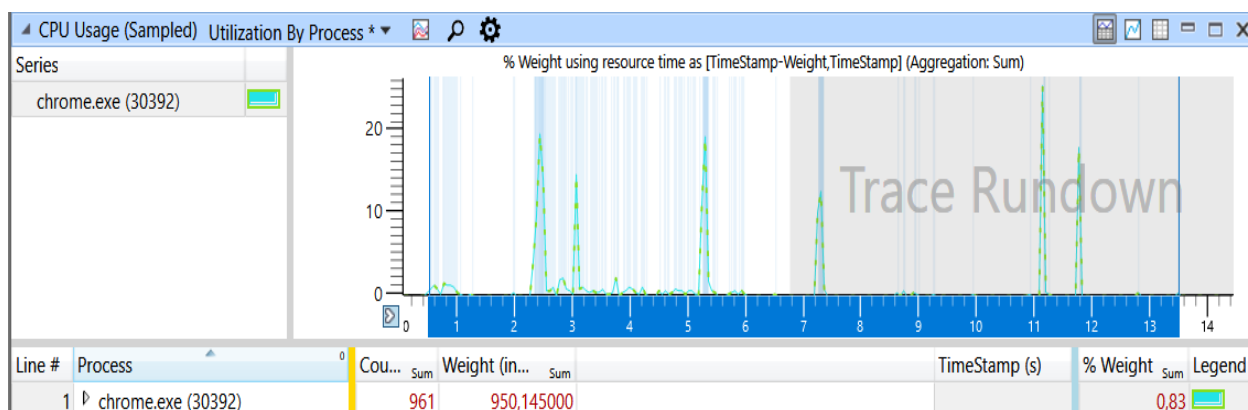


Рисунок 25 – Результат использования процессора приложением на PureScript

3.3.4 Elm

Результат профилирования скорости выполнения кода на языке Elm представлен в таблице 7, на рисунке 27 можно увидеть flame graph производительности веб-приложения. Среднее время выполнения кода составило 192 мс.

Таблица 7 – Измерения выполнения кода на Elm

№ эксперимента	Время выполнения, мс
1	191
2	201
3	183
4	182
5	203
6	191
7	201
8	183
9	182
10	203

График потребления оперативной памяти можно увидеть на рисунке 26. В среднем потребление памяти составляет 10.4 МБ.

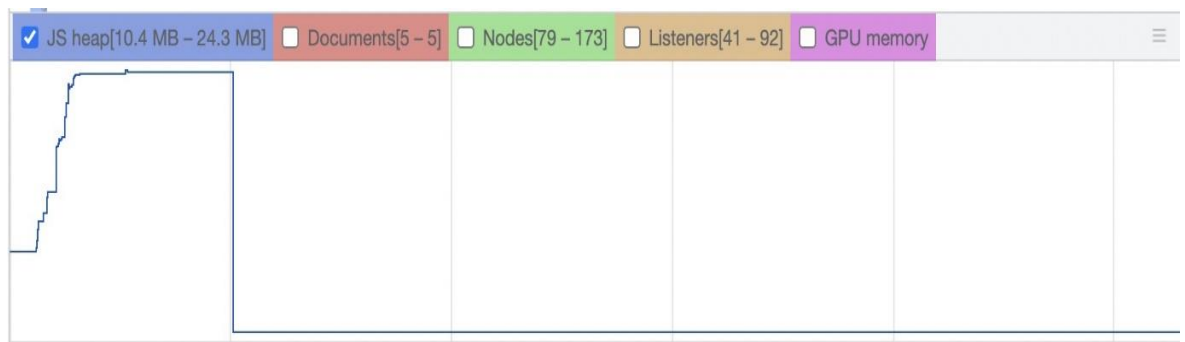


Рисунок 26 – График потребления памяти приложения на Elm

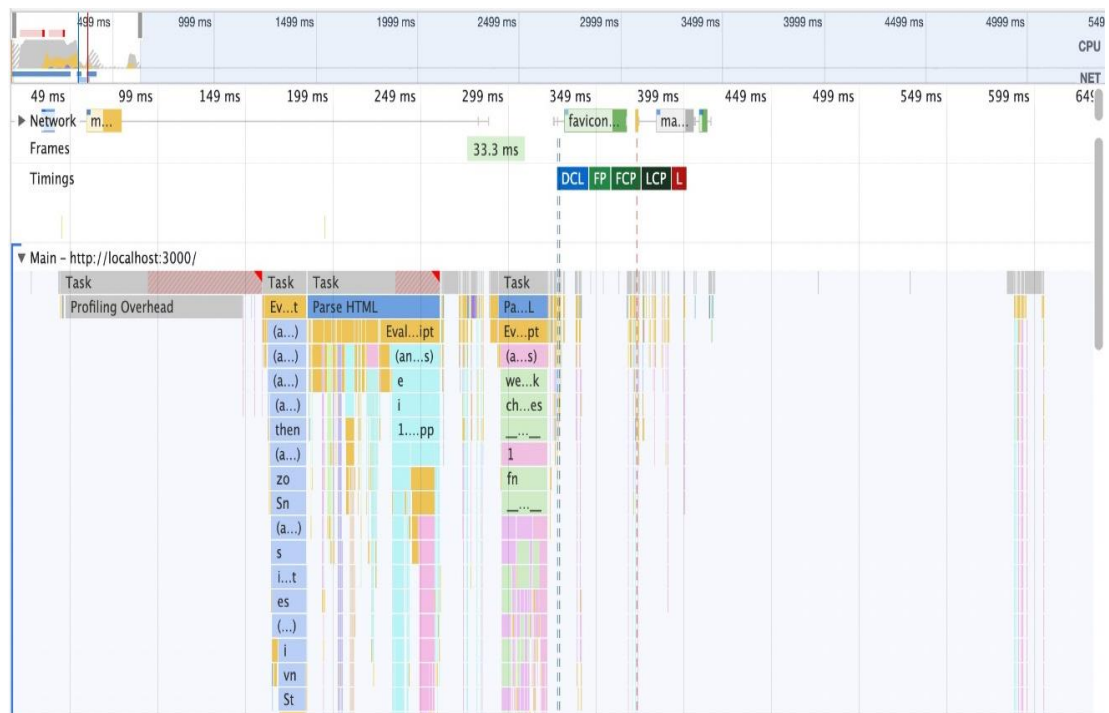


Рисунок 27 – Flame graph приложения на Elm

Использование процессора составило 0.35%. Результат представлен на рисунке 28.

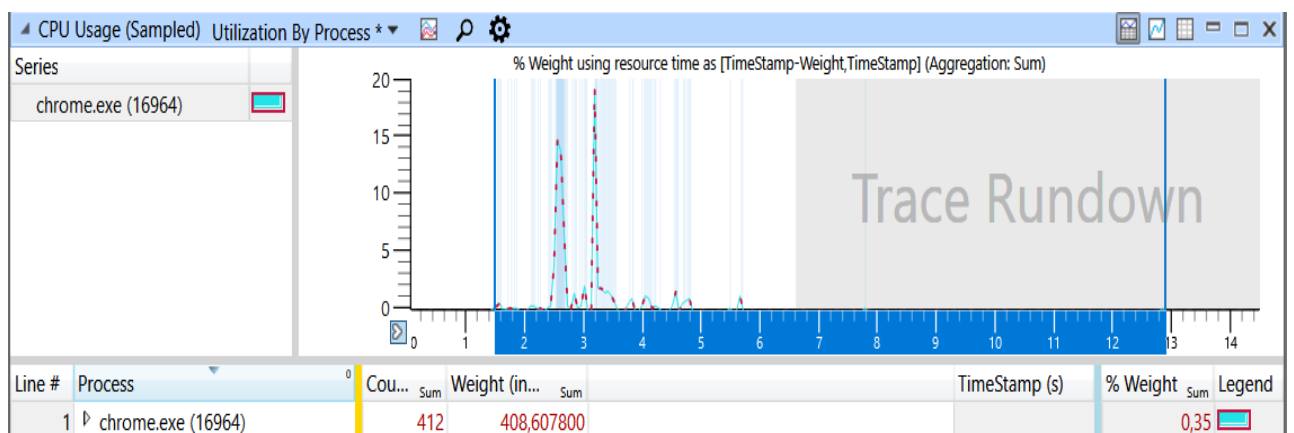


Рисунок 28 – Результат использования процессора приложением на Elm

3.3 Итоговая таблица и рекомендации

Из глав 1 и 3 можно составить таблицу результатов (таблица 8).

Таблица 8 – Итоговая сравнительная таблица

Критерий	TypeScript	ReScript	PureScript	Elm
Потребление ресурсов процессора, %	0.25	0.44	0.83	0.35
Потребление оперативной памяти, МБ	9.6	13.3	11	10.4
Время выполнения, мс	208.6	155.2	170.8	192
Размер скомпилированного кода, КБ	150	143	292	36
Время компиляции, сек	5.32	2.08	8.49	4.92
Популярность, место	1	4	2	3
Инструменты и экосистема, место	1	3	4	2
Документация и материалы для обучения, место	1	3	4	2
Постепенная типизация	Есть	Нет	Нет	Нет

По результатам таблицы и по опыту разработки приложений можно выделить следующие рекомендации:

1) если в проект, написанный на JavaScript, нужно добавить типизацию, то лучше всего для решения этой задачи подойдет TypeScript,

который имеет возможность добавлять типы постепенно. У других инструментов такой возможности нет, поэтому для внедрения типизации нужно будет полностью переписать проект,

2) если необходимо создать новое веб-приложение со статической типизацией, но при этом ни у кого в команде нет навыков работы с PureScript, ReScript, Elm или другими функциональными языками, то стоит выбрать TypeScript, потому что он имеет более легкий порог входа, огромное количество обучающих материалов, а также большое сообщество разработчиков,

3) для проектов, в которых необходима высокая производительность, стоит выбрать ReScript, потому что он компилируется в наиболее быстрый JavaScript код,

4) для проектов, в которых размер приложения играет критическую роль, стоит выбрать Elm, потому что он имеет наименьший размер скомпилированного кода,

5) для проектов, в которых необходима быстрая скорость компиляции, стоит выбрать ReScript.

ЗАКЛЮЧЕНИЕ

В первом разделе работы проведены анализ российской и зарубежной научной литературы по теме исследования, анализ информационных ресурсов для разработчиков, с целью усвоения практического опыта в рамках разработки веб-приложений с использованием различных подходов к статической типизации. Также было описано текущее состояние проблемы темы исследования и определены наиболее распространенные инструменты для статической типизации в JavaScript согласно рейтингу сообщества программистов и заключениям экспертов данной области. Выбранные инструменты, а именно – TypeScript, PureScript, Elm и ReScript, послужили основой для последующего сравнения их качественных и количественных характеристик.

Во втором разделе выпускной квалификационной работы были выявлены основные критерии для сравнения качества рассматриваемых инструментов. Данные критерии были проанализированы, и отобраны наиболее значимые из них. В результате была разработана система критериев для проведения сравнительного анализа инструментов и составлена таблица с кратким описанием выделенных критериев.

В заключительном разделе выпускной квалификационной работы разработано четыре веб-приложения, написанных на следующих языках программирования: TypeScript, ReScript, PureScript и Elm. Также было проведено тестирование разработанных веб-приложений на производительность, данные были собраны, предварительно обработаны и визуализированы. Кроме того, был проведен сравнительный анализ подходов к реализации статической типизации в JavaScript. По результатам анализа были выделены преимущества и недостатки каждого из рассмотренных подходов, описаны их особенности, ситуации в которых они могут раскрыться, а также разработаны рекомендации по выбору наиболее подходящего инструмента.

В ходе выполнения выпускной квалификационной работы были решены

все поставленные задачи и выполнено индивидуальное задание в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вандеркам Дэн Эффективный TypeScript: 62 способа улучшить код. – СПб.: Питер, 2020. — 288 с
2. Натан Розенталс Изучаем TypeScript 3. – М.: ДМК Пресс, 2019. – 624 с.
3. Борис Черный Профессиональный TypeScript. Разработка масштабируемых JavaScript- приложений. – СПб.: Питер, 2021. — 352 с.
4. TypeScript: Typed JavaScript at Any Scale [Электронный ресурс] – URL: <https://www.typescriptlang.org/> (дата обращения: 05.04.2023).
5. Flow: A Static Type Checker for JavaScript [Электронный ресурс] – URL: <https://flow.org/en/> (дата обращения: 05.04.2023).
6. Статическая типизация [Электронный ресурс] – URL: <https://ru.reactjs.org/docs/static-type-checking.html> (дата обращения: 10.02.2023).
7. Stackoverflow 2021 Developer Survey [Электронный ресурс] – URL: <https://insights.stackoverflow.com/survey/2021> (дата обращения: 10.02.2023).
8. Stackoverflow 2022 Developer Survey [Электронный ресурс] – URL: <https://survey.stackoverflow.co/2022/> (дата обращения: 10.02.2023).
9. Сучков А.А., Гек Д.К., Багаева А.П. Использование ReactJS в современной web-разработке // Актуальные проблемы авиации и космонавтики: сборник материалов V Международной научно-практической конференции, посвященной Дню космонавтики. 2019. Т. 2. С. 378–380.
10. Сергеев Е.И.Б, Прасолов А.П. Сравнение языков программирования TypeScript и JavaScript в разработке современных веб- приложений / Міжнародний науковий журнал №7 2016. с.118-121/.
11. Разновидности JavaScript. Рейтинги [Электронный ресурс] – URL: <https://2023.stateofjs.com/ru/javascript-flavors/> (дата обращения: 09.04.2023).
12. Справочник React. Статическая типизация [Электронный ресурс] Режим доступа: <https://reactdev.ru/handbook/static-type-checking/> (дата обращение 21.04.2023)

13. Опыт перевода большого проекта с Flow на TypeScript [Электронный ресурс] – URL: <https://habr.com/ru/company/directum/blog/462055/> (дата обращения: 23.04.2023).
14. Why use static types in JavaScript? (A 3-part primer on static typing with Flow) [Электронный ресурс] – URL: <https://medium.com/free-code-camp/why-use-static-types-in-javascript-part-1-8382da1e0adb> (дата обращения: 23.04.2023).
15. Dynamic typing vs. static typing [Электронный ресурс] – URL: https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html/ (дата обращения: 23.04.2023).
16. Делаем TypeScript строже. Доклад Яндекса [Электронный ресурс] URL: <https://habr.com/ru/company/yandex/blog/532240/> (дата обращения: 23.04.2023).
17. Типы данных JavaScript и структуры данных [Электронный ресурс] – URL: https://developer.mozilla.org/ru/docs/Web/JavaScript/Data_structures (дата обращения: 05.04.2023).
18. PureScript [Электронный ресурс] – URL: <https://www.purescript.org/> (дата обращения: 05.04.2023).
19. Elm [Электронный ресурс] – URL: <https://elm-lang.org> (дата обращения: 23.04.2023).
20. ReScript [Электронный ресурс] – URL: <https://rescript-lang.org/docs/manual/latest/introduction> (дата обращения: 23.04.2023).
21. The 2022 State of the Octoverse [Электронный ресурс] – URL: <https://octoverse.github.com> (дата обращения: 02.02.2023).
22. NPM Trends [Электронный ресурс] – URL: <https://www.npmtrends.com> (дата обращения: 02.02.2023).
23. State of JS 2022 [Электронный ресурс] – URL: <https://2022.stateofjs.com/en-US/> (дата обращения: 02.02.2023).
24. JetBrains Webstorm – [Электронный ресурс]. – URL: <https://www.jetbrains.com/ru-ru/webstorm/> (дата обращения: 15.02.2023).

25. JavaScript – [Электронный ресурс]. – URL: <https://developer.mozilla.org/ru/docs/Glossary/JavaScript> (дата обращения: 15.02.2022).
26. Справочные материалы по анализу производительности [Электронный ресурс]. – URL: <https://docs.microsoft.com/ru-ru/microsoftedge/devtools-guide-chromium/evaluate-performance/reference> (дата обращения: 28.11.2023).
27. Chrome Devtools [Электронный ресурс]. – URL: <https://techblog.sdstudio.top/chrome-devtools-poleznosti-pri-razrabotke/> (дата обращения: 01.05.2023).
28. Разработка React-приложений с использованием ReasonReact [Электронный ресурс] URL: <https://habr.com/ru/companies/ruvds/articles/424965/> (дата обращения: 23.04.2023).
29. Why use static types in JavaScript? (A 3-part primer on static typing with Flow) [Электронный ресурс] – URL: <https://medium.com/free-code-camp/why-use-static-types-in-javascript-part-1-8382da1e0adb> (дата обращения: 23.04.2023).
30. Dynamic typing vs. static typing [Электронный ресурс] – URL: https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html/ (дата обращения: 23.04.2023).
31. Elm – забава или серьёзный инструмент? [Электронный ресурс] URL: <https://habr.com/ru/articles/696718/> (дата обращения: 23.04.2023).
32. Webpack [Электронный ресурс] – URL: <https://webpack.js.org> (дата обращения: 02.02.2023)
33. Vite [Электронный ресурс] – URL: <https://vitejs.dev/> (дата обращения: 02.02.2023)
34. State of JS 2021 [Электронный ресурс] – URL: <https://2021.stateofjs.com/en-US/> (дата обращения: 02.02.2022).
35. State of JS 2020 [Электронный ресурс] – URL: <https://2020.stateofjs.com/en-US/> (дата обращения: 02.02.2022).

36. Подготовка веб-разработчиков к использованию статической типизации в JavaScript / Крисанова Е.О.// Научно-методический журнал «Современное образование: традиции и инновации» №1/2022. – С.182-187.
37. JS: правильный путь [Электронный ресурс]. – Режим доступа: <http://jstherightway.org/ru-ru/> (дата обращения: 11.04.2023).
38. Проектирование программного обеспечения [Электронный ресурс]. – Режим доступа: <https://habr.com/post/74330/> (дата обращения: 13.02.2023).
39. ГОСТ Р ИСО/МЭК 25010-2015. «Информационные технологии (ИТ). Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов» (действует с 1 июня 2016 г.).
40. ГОСТ Р ИСО/МЭК 25040-2014. «Информационные технологии (ИТ). Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Процесс оценки» (действует с 1 июня 2015 г.).
41. Справочные материалы по анализу производительности [Электронный ресурс]. – URL: <https://docs.microsoft.com/ru-ru/microsoft-edge/devtools-guide-chromium/evaluate-performance/reference> (дата обращения: 01.05.2023).
42. Гудков П.А. Методы сравнительного анализа // Учеб. пособие. – Пенза: Изд-во Пенз. гос. ун-та, 2008 – 81 с.
43. Feldman R. Elm in Action. – second edition. – Manning Publications Co., 2020. – 302 p.
44. Ликбез по типизации в языках программирования [Электронный ресурс] URL: <https://habr.com/ru/articles/161205/> (дата обращения: 23.04.2023).
45. Статическая и динамическая типизация [Электронный ресурс] URL: <https://habr.com/ru/articles/308484/> (дата обращения: 23.04.2023).
46. Основы разработки на языке Elm (руководство по инструментарию для начинающих) [Электронный ресурс] URL: <https://habr.com/ru/articles/302154/> (дата обращения: 23.04.2023).

47. Как я писал приложение на Elm [Электронный ресурс] URL: <https://habr.com/ru/articles/314050/> (дата обращения: 23.04.2023).