

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

*К защите допустить:*

И.О. заведующего кафедрой  
информатики

\_\_\_\_\_ С.И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту  
на тему

**РЕАЛИЗАЦИЯ ПРОСТОЙ БД: ПРОГРАММНЫЙ МОДУЛЬ,  
ПЕРЕНОСИМЫЙ БЕЗ  
ИНСТАЛЛЯЦИИ; ХРАНЕНИЕ ДАННЫХ СРЕДСТВАМИ ФАЙЛОВОЙ  
СИСТЕМЫ И Т.П. С ДОСТУПОМ ПОСРЕДСТВОМ API  
ПРОЦЕДУРНОГО ТИПА (БИБЛИОТЕКАМ, КЛАСС И Т.П.)**

БГУИР КП 1-40 04 01 01 024 ПЗ

Студент  
Руководитель

Д.В. Толстой  
Н.Ю. Гриценко

Минск 2024

## СОДЕРЖАНИЕ

Введение.....	5
1 Платформа программного обеспечения.....	6
1.1 Операционная система .....	6
2.2 Язык программирования .....	8
3 Теоретическое обоснование разработки программного продукта.....	21
3.1 Обоснование необходимости анализа.....	21
4 Проектирование функциональных возможностей программы .....	22
4.1 Функции программного обеспечения .....	22
Заключение .....	23
Список литературных источников .....	24

## ВВЕДЕНИЕ

В современном мире информационных технологий, с растущими требованиями к хранению и обработке данных, возникает необходимость в разработке простых и эффективных баз данных (БД). Такие базы данных должны быть легко переносимыми, не требовать сложной инсталляции и обеспечивать быстрый доступ к данным. Одним из решений такой задачи является создание программного модуля, который использует файловую систему для хранения данных и предоставляет доступ к ним через *API* процедурного типа.

Цель данного курсового проекта – разработать и реализовать простую базу данных, которая будет удовлетворять вышеуказанным требованиям. В рамках проекта будут исследованы различные подходы к созданию легковесных БД, анализироваться существующие решения и разрабатываться собственный программный модуль. Будет рассмотрена возможность хранения данных в файловой системе, методы обеспечения целостности и безопасности данных, а также разработка *API* для доступа к данным.

Проект включает в себя разработку библиотеки или класса, который будет предоставлять необходимый функционал для работы с БД, включая создание, чтение, обновление и удаление данных. Также будет рассмотрена возможность интеграции разработанного решения с существующими приложениями и системами.

В результате данного исследования будет создан простой, но функциональный программный модуль для работы с БД, который можно будет легко перенести и использовать в различных проектах без необходимости сложной инсталляции. Это позволит разработчикам сэкономить время и ресурсы при создании и интеграции систем хранения данных в свои проекты.

# 1 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 1.1 Операционная система

*Linux* (или *GNU/Linux*) – семейство бесплатных *Unix*-подобных многопользовательских операционных систем, основанных на ядре *Linux* и на программном обеспечении *GNU*. Широкое, в том числе коммерческое, распространение стало возможным в 1992 году благодаря лицензированию ядра *Linux* по свободной лицензии *GPL*. Одним из инициаторов *Linux* был финский программист Линус Торвалдс. Он по-прежнему играет координирующую роль в дальнейшей разработке ядра *Linux* и известен как «Великодушный пожизненный диктатор».

Модульная операционная система дорабатывается разработчиками программного обеспечения по всему миру. В разработке участвуют компании, некоммерческие организации и множество волонтеров. При использовании на компьютерах обычно используются так называемые дистрибутивы *Linux*. Дистрибутив объединяет ядро *Linux* с различным программным обеспечением в операционную систему, подходящую для конечного пользователя. Многие распространители и опытные пользователи адаптируют ядро под свои нужды.

*Linux* широко и разнообразно используется, например, на рабочих станциях, серверах, мобильных телефонах, маршрутизаторах, ноутбуках, встроенных системах, мультимедийных терминалах и суперкомпьютерах. Система *Linux* прочно обосновалась на рынке серверов, а также в мобильном секторе, и в то время играет небольшую, но растущую роль на рынке настольных компьютеров и ноутбуков. *Linux* используется многочисленными пользователями, включая частных пользователей, правительства, организации и предприятия.[1]

Ядро *Linux* представляет собой монолитное ядро, написанное на языке программирования *C* с использованием некоторых расширений *GNU-C*. Однако важные подпрограммы и критичные модули программируются на языке ассемблера для конкретного процессора. Ядро позволяет использовать только драйверы, необходимые для соответствующего оборудования. Кроме того, ядро также берёт на себя выделение процессорного времени и ресурсов для отдельных программ. С технической точки зрения, дизайн *Linux* сильно основан на модели *Unix*. [2]

Ядро *Linux* было перенесено на очень большое количество аппаратных архитектур. Их репертуар варьируется от довольно экзотических операционных сред, таких как карманный компьютер *iPAQ*, навигационные устройства от *TomTom* или даже цифровые камеры, до мейнфреймов, таких как *IBM System z*, а с некоторых пор также мобильных телефонов, таких как *Motorola A780*, и смартфонов с операционными системами, такими как *Android* или *Sailfish*. Несмотря на модульную концепцию, монолитная базовая архитектура ядра сохраняется. Ориентация оригинальной версии на широко

распространенные персональные компьютеры с процессором *x86* позволила обеспечить поддержку широкого спектра оборудования и доверить работу с драйверами даже неопытным программистам.

Все версии ядра *Linux* заархивированы на *kernel.org*. Версия, которую можно найти там, гарантированно является соответствующим эталонным ядром. На этом факте основаны так называемые дистрибутивные ядра, а дополнительные функции добавляются отдельными дистрибутивами *Linux*. Особенностью является схема нумерации версий, состоящая из четырех цифр, разделенных точками, например 2.6.14.1. Такая нумерация предоставляет информацию о точной версии и, таким образом, о возможностях соответствующего ядра. Из четырех чисел последнее меняется при исправлении ошибок и оптимизации кода, но не при введении новых функций или других серьезных изменениях. По этой причине его редко упоминают, например, при сравнении версий ядра. Предпоследнее, третье число меняется по мере добавления новых функций. То же самое относится к первым двум номерам, но для них изменения новые функции должны быть более радикальными. Начиная с версии 3.0 (август 2011 года) второе число опускается.

Несмотря на большую безопасность по сравнению с самой распространенной операционной системой *Windows*, возможность параллельной установки и большой выбор бесплатного программного обеспечения, *Linux* лишь изредка используется на настольных компьютерах. Хотя интерфейс наиболее популярных «сборок» *Linux* выглядит аналогично *Windows* или *macOS*, они отличаются различными системными функциями. Поэтому неопытному пользователю может потребоваться определённый период обучения.

Благодаря совместимости с другими *Unix*-подобными системами *Linux* особенно быстро зарекомендовал себя на рынке серверов. Поскольку множество часто используемых и необходимых серверных программ, таких как веб-серверы, серверы баз данных и групповое программное обеспечение, были доступны для *Linux* на раннем этапе, бесплатно и в основном без ограничений, доля рынка неуклонно росла. *Linux* считается стабильным и простым в обслуживании, он также отвечает особым требованиям, предъявляемым к серверной операционной системе. Модульная структура системы *Linux* также позволяет использовать компактные выделенные серверы. Кроме того, перенос *Linux* на самые разные аппаратные компоненты привел к тому, что *Linux* поддерживает все известные серверные архитектуры. В январе 2017 года не менее 34 % всех веб-сайтов были доступны с использованием сервера *Linux*. Поскольку не все серверы *Linux* идентифицируют себя как таковые, фактическая доля может быть значительно выше, до 65 %.[3]

Существуют специально оптимизированные дистрибутивы *Linux* для смартфонов и планшетов. В дополнение к функциям телефонии и *SMS* они предлагают различные функции *PIM*, навигации и мультимедиа. Работа

обычно осуществляется с помощью мультитач или с помощью пера. *Android* также рассматривается как дистрибутив *Linux*, имея с ним много общего. С конца 2010 года системы *Linux* захватили лидерство на быстрорастущем рынке смартфонов, и в настоящее время их рыночная доля превышает 80 %.

Среди других областей применения *Linux*: автомобильные бортовые компьютерные системы и суперкомпьютеры.

Несмотря на большую безопасность по сравнению с самой распространенной операционной системой *Windows*, возможность параллельной установки и большой выбор бесплатного программного обеспечения, *Linux* лишь изредка используется на настольных компьютерах. Хотя интерфейс наиболее популярных «сборок» *Linux* выглядит аналогично *Windows* или *macOS*, они отличаются различными системными функциями. Поэтому неопытному пользователю может потребоваться определённый период обучения.[4]

## 2.2 Язык программирования

*JavaScript* – это интерпретируемый язык программирования, который используют для написания *frontend*- и *backend*-частей сайтов, а также мобильных приложений. Часто в текстах и обучающих материалах название языка сокращают до *JS*. Это язык программирования высокого уровня, то есть код на нем понятный и хорошо читается.

*JS* поддерживают все популярные браузеры. Во *frontend*-части сайтов язык используют для создания интерактива (анимаций, всплывающих форм, автозаполнения), так как он связан с *HTML* и *CSS* и может ими манипулировать. В *backend*-части с языком *JavaScript* работают на платформе *Node.js*. С ее помощью, например, разрабатывают серверные веб-приложения и подключают библиотеки. В поисковике *Google* на *JavaScript* работает строка автозаполнения, а *Netflix*, *Uber*, *eBay* используют его в своем *backend*. Уже 6 лет *JS* – самый популярный язык среди разработчиков по версии *GitHub*. [5]

События, в результате которых появился *JavaScript*, разворачивались в течение шести месяцев, с мая по декабрь 1995 года. Компания *Netscape Communications* уверенно прокладывала себе путь в области веб-технологий. Её браузер *Netscape Communicator* успешно отвоевывал позиции у *NCSA Mosaic*, первого популярного веб-браузера. *Netscape* была создана людьми, принимавшими участие в разработке *Mosaic* в ранние 90-е. Теперь, с деньгами и независимостью, у них было всё необходимое для поиска способов дальнейшего развития веб-технологий. Именно это послужило толчком для рождения *JavaScript*.

Основатель *Netscape Communications* и бывший участник команды *Mosaic* Марк Андрессен (рисунок 1) считал, что веб должен стать более динамичным. Анимации, взаимодействие с пользователями и другие виды интерактивности должны стать неотъемлемой частью интернета будущего. Веб нуждался в лёгком скриптовом языке (или языке сценариев – прим. ред.),

способном работать с *DOM*, который в те дни не был стандартизирован. Существовало одно «но», являвшееся на тот момент серьёзным вызовом: этот язык не должен был предназначаться для крупных разработчиков и прочих людей, имевших отношение к инженерной стороне вопроса. *Java* в те дни уже активно развивалась и твёрдо заняла эту нишу. Таким образом, новый скриптовый язык должен был предназначаться для совершенно иной аудитории — дизайнеров. Очевидно, что веб был статичным, а *HTML* был достаточно молод и прост в освоении даже для тех, кто не имел ничего общего с программированием. Следовательно, всё, что должно было стать частью браузера и сделать веб более динамичным, должно быть максимально понятным для далёких от программирования людей. Из этого предположения родилась идея *Mocha*, который должен был стать тем самым простым, динамичным и доступным скриптовым языком.



Рисунок 1 – Марк Адриссен

И тут в нашей истории появляется Брендан Айк (рисунок 2), отец *JavaScript*. Айк должен был разработать для *Netscape* «*Scheme*» для браузера. *Scheme* — это динамичный, мощный и функциональный диалект языка программирования *Lisp* с максимально упрощённым синтаксисом. Вебу требовалось что-то подобное: простое в освоении, динамичное, немногословное и мощное. Айк не стал упускать шанса поработать над тем, что ему нравилось, и присоединился к команде.





Рисунок 2 – Брэндан Айк

Перед командой была поставлена задача подготовить работающий прототип в кратчайшие сроки. *Sun Microsystems* заканчивала работу над своим языком программирования *Java*, на тот момент называвшимся *Oak*, и *Netscape Communications* была уже готова заключить с компанией контракт, чтобы сделать *Java* доступным в своем браузере. Так зачем же понадобился *Mocha* (первое название *JavaScript*)? Зачем нужно было создавать абсолютно новый язык программирования при наличии готовой альтернативы? Дело в том, что *Java* не был предназначен для той аудитории, на которую ориентировался *Mocha* – скриптеры, любители, дизайнеры. *Java* был слишком большим и навороченным для того, чтобы выполнять эту роль. Основная идея заключалась в том, что *Java* должен был предназначаться для крупных разработчиков и профессиональных программистов, в то время, как *Mocha* должен был использоваться для небольших скриптовых задач. Другими словами, *Mocha* должен был стать скриптовым компаньоном для *Java* по принципу, аналогичному тому, как взаимодействуют *C/C++* и *Visual Basic* на платформе *Windows*.

Инженеры *Netscape* приступили к детальному изучению *Java*. Они даже начали разрабатывать собственную виртуальную машину *Java*, однако проект быстро свернули, так как она не могла достичь идеальной совместимости с виртуальной машиной *Sun Microsystems*.

Проблема скорейшего выбора языка стояла как никогда остро. Возможными кандидатами были *Python*, *Tcl* и *Scheme*. Айк должен был действовать быстро. По сравнению с конкурентами у него были два преимущества: свобода в определении набора необходимых возможностей и прямая связь с заказчиком. К несчастью, имело место и очевидное неудобство: для принятия огромного количества важных решений времени практически не



было. *JavaScript, a.k.a. Mocha*, был рождён именно в таких условиях. В течение нескольких недель был подготовлен рабочий прототип, который затем был интегрирован в *Netscape Communicator*.

То, что должно было стать аналогом *Scheme* для браузера, вылилось в нечто совершенно иное. Рукой Айка управляли необходимость закрыть сделку с *Sun* и сделать *Mocha* скриптовым компаньоном для *Java*. Синтаксис должен был быть максимально близким *Java*. Помимо этого, от *Java* была унаследована семантика для большого количества устоявшихся идиом. Таким образом, *Mocha* был совсем не похож на *Scheme*. Он выглядел, как динамический *Java*, под оболочкой которого скрывался гибрид *Scheme* и *Self*.

Прототип *Mocha* был интегрирован в *Netscape Communicator* в мае 1995 года. Через очень короткий промежуток времени он был переименован в *LiveScript*, так как в тот момент слово *live* выглядело очень привлекательным с точки зрения маркетологов. В декабре 1995 года сделка между *Netscape Communications* и *Sun* была закрыта: *Mocha/LiveScript* был переименован в *JavaScript* и преподносился в качестве скриптового языка для выполнения небольших клиентских задач в браузере, в то время, как *Java* был полноценным профессиональным языком программирования для разработки сложных веб-компонентов.

Первая версия *JavaScript* заложила все те фундаментальные особенности, которыми этот язык знаменит и поныне. В частности, его объектная модель и функциональные особенности уже присутствовали в первой версии.

Трудно сказать, как развивались бы события, если бы Аик не успел предоставить рабочий прототип вовремя. *Python*, *Tcl*, *Scheme*, рассматривавшиеся в качестве альтернативы, были совершенно не похожи на *Java*. *Sun* было бы трудно принять в качестве языка-компаньона для *Java* варианты, в корне отличавшиеся от него. С другой стороны, *Java* долгое время был важной частью веба. Если бы *Sun* не являлись определяющим фактором, у *Netscape* было бы намного больше свободы в выборе языка. Но стала бы *Netscape* разрабатывать собственный язык или воспользовалась одним из существующих? Этого мы никогда не узнаем.

Когда *Sun* и *Netscape* закрыли сделку, и *Mocha/LiveScript* был переименован в *JavaScript*, встал ребром очень важный вопрос: что будет с конкурентами? Хотя *Netscape* и набирал популярность, становясь самым используемым браузером, *Microsoft* занималась активной разработкой *Internet Explorer*. С самых первых дней *JavaScript* показал настолько удивительные возможности в плане взаимодействия с пользователем, что соперничающим браузерам не оставалось ничего иного, кроме как в кратчайшие сроки найти готовые решения, представлявшие собой рабочие реализации *JavaScript*. В тот момент (и ещё достаточно долго после этого) веб-стандарты оставались достаточно слабыми. Поэтому *Microsoft* разработала свою реализацию *JavaScript*, назвав ее *JScript*. Убрав из названия слово *Java*, они смогли избежать возможных проблем с владельцами торговой марки. Однако, *JScript*

отличался не только названием. Небольшие различия в реализации – в частности, подход к некоторым *DOM* функциям – оставили рябь, которая будет ощущаться ещё долгие годы. Бои за *JavaScript* шли на гораздо большем количестве фронтов, чем названия и таймлайны, и многие причуды этого языка появились благодаря им. Первая версия *JScript* появилась в *Internet Explorer 3.0*, увидевшем свет в августе 1996 года.

Реализация *JavaScript* получила свое собственное название и в *Netscape*. Версия, выпущенная вместе с *Netscape Navigator 2.0*, была известна, как *Mocha*. Осенью 1996 года Айк переписал большую часть *Mocha*, чтобы разобраться с техническими огрехами и недоработками, возникшими, как следствие спешки при разработке. Новая версия была названа *SpiderMonkey*. Это название используется по сей день в *JavaScript*-движке браузера *Firefox*, внука *Netscape Navigator*.

В течение нескольких лет *JScript* и *SpiderMonkey* были единственными движками *JavaScript*. Особенности обоих движков, не всегда совместимые, определили вектор развития веба на ближайшие годы.

Первой большой переменной для *JavaScript* после его выпуска стала стандартизация *ECMA*. *ECMA* — ассоциация, созданная в 1961 году с целью стандартизации информационных и коммуникационных систем.

Работа над стандартизацией *JavaScript* началась в ноябре 1996 года. Стандарту, над которым работала группа *TC-39*, был присвоен идентификационный номер *ECMA-262*. К тому моменту *JavaScript* активно использовался на многих веб-страницах. В этом пресс-релизе 1996 года указано количество в 300000 страниц, использующих *JavaScript*.

Стандартизация стала для молодого языка не только важным шагом, но и серьезным вызовом. Она открыла *JavaScript* для большей аудитории и дала возможность сторонним разработчикам принимать участие в развитии языка. Она также помогла держать других разработчиков в рамках. В те времена бытовало опасение, что *Microsoft* или кто-либо ещё могут слишком сильно отклониться от оригинальной реализации языка, что могло привести к фрагментации.

Из-за проблем с торговой маркой *ECMA* не могла использовать *JavaScript* в качестве названия. После непродолжительных дебатов было решено, что описанный стандартом язык программирования будет назван *ECMAScript*. На сегодняшний день *JavaScript* это всего лишь коммерческое название *ECMAScript*. [6]

Первый стандарт *ECMAScript* был основан на версии *JavaScript*, входившей в состав *Netscape Navigator 4* и не включал в себя важные особенности, такие как регулярные выражения, *JSON*, исключения и важные методы для встроенных объектов. Тем не менее, в браузере он работал намного лучше. Версия 1 была выпущена в июне 1997 года.

Вторая версия, *ECMAScript 2*, была выпущена в июне 1998 года, чтобы исправить несостыковки между *ECMA* и стандартом *ISO* для *JavaScript* (*ISO/IEC 16262*) и не включала в себя никаких изменений самого языка.

Интересной особенностью этой версии *JavaScript* было то, что интерпретатор должен был самостоятельно решать, что делать с неотловленными ошибками (и в большинстве случаев оставленными как неклассифицированные). Причиной этому стало то, что исключения ещё не были частью языка на тот момент.

После *ECMAScript 2* работа продолжилась и первые большие изменения языка увидели свет. Новая версия включала в себя:

- 1 Регулярные выражения.
- 2 Блок *do-while*.
- 3 Исключения и *try/catch* блоки.
- 4 Больше встроенных функций для строк и массивов.
- 5 Форматирование численных выходных данных.
- 6 Операторы *in* и *instanceof*.
- 7 Улучшенная обработка ошибок.

*ECMAScript 3* был выпущен в декабре 1999 года.

Эта версия *ECMAScript* получила очень широкое распространение. Все крупные браузеры того времени поддерживали её и продолжали поддерживать в течение многих лет. Даже сегодня многие транспайлеры в качестве выходного языка могут указывать этот стандарт. Это сделало *ECMAScript 3* фундаментом для многих библиотек, даже когда были выпущены более поздние версии стандарта.

Хоть *JavaScript* использовался практически повсеместно, он всё ещё оставался клиентским языком программирования. Многие из его нововведений позволили ему приблизиться к тому, чтобы вырваться из этой клетки.

*Netscape Navigator 6*, выпущенный в ноябре 2000 года, поддерживал *ECMAScript 3*. Спустя почти полтора года был выпущен *Firefox*, браузер, основанный на кодовой базе *Netscape Navigator* и также поддерживавший *ECMAScript 3*. Бок о бок с *Internet Explorer* эти браузеры делали всё возможное для дальнейшего роста и развития *JavaScript*.

*AJAX* (*asynchronous JavaScript and XML*) — технология, появившаяся на свет в годы *ECMAScript 3*. Хоть она и не являлась частью стандарта, *Microsoft* встроила некоторые расширения для *JavaScript* в *Internet Explorer 5*. Одним из таких расширений была функция *XMLHttpRequest* в виде управляющего элемента *ActiveX XMLHttpRequest*. Эта функция позволяла браузеру выполнять асинхронные *HTTP*-запросы серверу, тем самым позволяя страницам обновляться на лету. Хотя само название *AJAX* было придумано значительно позже, сама техника активно использовалась в то время.

Применение *XMLHttpRequest* оказалось успешным, и годами позже было стандартизировано группами *WHATWG* и *W3C*.

Постоянная эволюция функциональности, разработчики, вносящие что-то новое в язык и встраивающие эти новинки в свои браузеры, до сих пор являются основополагающими факторами в развитии *JavaScript* и связанных с ним стандартов, таких, как *CSS* и *HTML*. Связь между отдельными группами в

те дни была очень слабая, что привело к задержкам и фрагментации. Честно говоря, в наши дни разработка *JavaScript* организована куда лучше благодаря процедурам, позволяющим заинтересованным группам вносить свои предложения.

К сожалению, следующие несколько лет не принесли *JavaScript* ничего хорошего. Вместе с началом работы над *ECMAScript 4* в сообществе, разбившемся на группы, начались разногласия. Одна группа утверждала, что *JavaScript* необходимо сделать языком для разработки крупных приложений. Эта группа предлагала множество новых опций большого объёма, требовавших внесения кардинальных изменений. Другая группа находила подобный вектор развития недопустимым. Отсутствие компромиссов и сложность некоторых предлагавшихся улучшений отодвигали выход *ECMAScript 4* всё дальше и дальше.

Работа над *ECMAScript 4* началась вскоре после выхода третьей версии в 1999 году. Большое количество интересных нововведений обсуждалось в *Netscape*. Однако интерес к ним со временем иссяк, и в 2003 году работа над новой версией *ECMAScript* остановилась. Был выпущен промежуточный отчёт, и некоторые разработчики, такие, как *Adobe (ActionScript)* и *Microsoft (JScript.NET)* использовали его в качестве основы для собственных движков. В 2005 году *AJAX* и *XMLHttpRequest* смогли вновь разжечь интерес к новой версии *JavaScript* и *TC-39* возобновила работу. Проходили годы и набор нововведений рос всё больше и больше.

Комитет, разрабатывавший *ECMAScript 4*, включал в себя *Adobe*, *Mozilla*, *Opera* (неофициально) и *Microsoft*. *Yahoo* вошла в игру, когда большинство решений по стандартам и возможностям были уже приняты, прислав Дугласа Крокфорда, влиятельного *JavaScript*-разработчика, который тут же раскритиковал большинство новшеств, получив мощную поддержку со стороны представителя *Microsoft*.

То, что началось как сомнения, быстро переросло в сильную оппозицию к *JavaScript*. *Microsoft* наотрез отказывалась утверждать любую часть *ECMAScript 4* и была готова к любым действиям, включая судебные тяжбы, чтобы не дать стандарту возможности быть утвержденным. К счастью, члены комитета смогли избежать судебных разбирательств. Однако из-за разногласий *ECMAScript 4* продолжал топтаться на месте.

Решения, принятые на этой встрече:

1 Сфокусироваться на работе над *ES3.1* при полном сотрудничестве всех участников и выпустить две совместимые версии в начале следующего года.

2 Начать работу над следующим шагом после *ES3.1*, который будет включать в себя синтаксические расширения, но более скромные, нежели те, которые предлагались для *ES4* как в плане семантических, так и синтаксических инноваций.

3 Ради всеобщего блага было решено отказаться от некоторых из предложений *ES4*: пакетов, пространства имен и ранней связки. Они были признаны бесполезными для веба. Это решение – ключ к Гармонии.

4 Прочие идеи и цели *ES4* были перефразированы с целью достичь консенсуса внутри комитета, например, понятие классов, основанное на имеющихся концептах *ES3*, совмещённых с предлагаемыми улучшениями *ES3.1*.

В результате ушло почти восемь лет на то, чтобы закончить разработку *ECMAScript 4*. Тяжелый урок для всех, кто принимал участие.

Слово «Гармония», появившееся в перечисленных решениях, стало названием проекта в будущих версиях. *Harmony* станет альтернативой, с которой все будут согласны. После выхода *ECMAScript 3.1* (в виде версии 5, как будет рассказано чуть ниже) *ECMAScript Harmony* стала тем местом, где обсуждаются все новые идеи относительно *JavaScript*.

В 2008 году, после долгих боев, развернувшихся вокруг *ECMAScript 4*, сообщество сфокусировалось на работе над *ECMAScript 3.1*, отправив *ECMAScript 4* на свалку. В 2009 году *ECMAScript 3.1* был полностью завершён и одобрен всеми участниками комитета. Так как *ECMAScript 4* считался своеобразным вариантом *ECMAScript* несмотря на отсутствие хоть какого-либо релиза, было решено переименовать *ECMAScript 3.1* в *ECMAScript 5*, чтобы избежать недоразумений.

*ECMAScript 5* стал одной из самых поддерживаемых версий *JavaScript*, став также целью компиляции многих транспайлеров. *ECMAScript 5* получил полную поддержку в браузерах *Firefox 4* (2011), *Chrome 19* (2012), *Safari 6* (2012), *Opera 12.10* (2012) и *Internet Explorer 10* (2012).

*ECMAScript 5* был достаточно скромным улучшением *ECMAScript 3*, включавшим в себя:

- 1 Геттеры/сеттеры.
- 2 Разделители-запятые в массивах и объектах.
- 3 Возможность использовать зарезервированные слова в качестве свойств объекта.
- 4 Новые методы объектов (*create*, *defineProperty*, *keys*, *seal*, *freeze*, *getOwnPropertyNames* и т.д.).
- 5 Новые методы массивов (*isArray*, *indexOf*, *every*, *some*, *map*, *filter*, *reduce* и т.д.).
- 6 *String.prototype.trim* и доступ к свойствам.
- 7 Новые методы *Date* (*toISOString*, *now*, *toJSON*).
- 8 Привязывание функций.
- 9 *JSON*.
- 10 Неизменяемые глобальные объекты (*undefined*, *NaN*, *Infinity*).
- 11 Строгий режим.
- 12 Другие небольшие изменения (*parseInt* игнорирует ведущие нули, функции в *throw* имеют значение и т.д.).

Ни одно из этих изменений не требовало внесения изменений в синтаксис. Геттеры и сеттеры в то время уже неофициально поддерживались некоторыми браузерами. Новые методы объектов должны были улучшить большое программирование, дав программистам больше инструментов для

проверки соблюдения определённых инвариантов (*Object.seal*, *Object.freeze*, *Object.createProperty*). Строгий режим также стал мощным инструментом в этой области, позволив избежать большого числа ошибок. Дополнительные методы массивов улучшили определённые функциональные паттерны (*map*, *reduce*, *filter*, *every*, *some*). Ещё одним большим нововведением является *JSON*: формат данных, основанный на *JavaScript*, который теперь поддерживается нативно благодаря *JSON.stringify* и *JSON.parse*. Другие изменения касаются небольших улучшений, основанных на практическом опыте. В целом, *ECMAScript 5* был небольшим улучшением, которое приукрасило *JavaScript* в плане юзабилити как для небольших скриптов, так и для более объемных проектов. Тем не менее, большое количество хороших идей, предлагавшихся для *ECMAScript 4*, так и не были реализованы и ждали своего возвращения в *ECMAScript Harmony*.

*ECMAScript 5* получил обновление в 2011 году под названием *ECMAScript 5.1*. Этот релиз вносил ясность в некоторые неоднозначные пункты стандарта, но никаких новых возможностей в нем не было. Все новые возможности были запланированы для следующего большого релиза *ECMAScript*.<sup>[7]</sup>

План *ECMAScript Harmony* стал основой для последующих улучшений *JavaScript*. Многие идеи из *ECMAScript 4* канули в лету ради всеобщего блага, однако некоторые были пересмотрены. *ECMAScript 6*, позже переименованный в *ECMAScript 2015*, должен был принести большие перемены. Почти все обновления, так или иначе влиявшие на синтаксис, были отложены именно для этой версии. К 2015 году комитет, наконец, смог побороть все внутренние разногласия, и *ECMAScript 6* увидел свет. Большинство производителей браузеров уже работали над поддержкой этой версии, однако до сих пор не все браузеры имеют полную совместимость с *ECMAScript 2015*.

Выход *ECMAScript 2015* стал причиной резкого роста популярности транспайлеров, таких, как *Babel* или *Traceur*. Благодаря тому, что производители этих транспайлеров следили за работой технического комитета, у многих людей появилась возможность испытать преимущества *ECMAScript 2015* задолго до его выхода.

Некоторые из основных возможностей *ECMAScript 4* были реализованы в этой версии с несколько иным подходом. Например, классы в *ECMAScript 2015* — это нечто большее, чем просто синтаксический сахар поверх прототипов. Подобный подход облегчает разработку и внедрение новых возможностей.

Краткий список новых возможностей включает в себя:

- 1 *Let* (лексическая) и *const* (неизменяемая) привязки.
- 2 Стрелочные функции (короткие анонимные функции) и лексическое *this*.
- 3 Классы (синтаксический сахар поверх прототипов).



4 Улучшения объектных литералов (вычисляемые ключи, укороченные определения методов и т.д.).

5 Шаблонные строки.

6 Промисы.

7 Генераторы, итерируемые объекты, итераторы и *for..of*.

8 Параметры функций по умолчанию и оператор *rest*.

9 *Spread*-синтаксис.

10 Деструктуризация.

11 Модульный синтаксис.

12 Новые коллекции (*Set*, *Map*, *WeakSet*, *WeakMap*).

13 Прокси и *Reflect*.

14 Тип данных *Symbols*.

15 Типизированные массивы.

16 Наследование классов.

17 Оптимизация хвостовой рекурсии.

18 Упрощённая поддержка *Unicode*.

19 Двоичные и восьмеричные литералы.

Все эти возможности открыли *JavaScript* для ещё большего количества программистов и внесли существенный вклад в большое программирование.

Некоторых может удивить, как могло такое количество новых возможностей проскочить мимо процесса стандартизации, во время которого был загублен *ECMAScript* 4. Хотелось бы отметить, что большинство наиболее агрессивных инноваций *ECMAScript* 4, таких, как пространства имён или опциональное типирование, были забыты и к ним больше не возвращались, в то время, как другие были переосмыслены с учётом возникших возражений. Работа над *ECMAScript* 2015 была очень тяжёлой и заняла почти шесть лет (и даже больше, учитывая время, необходимое на реализацию). Но сам факт того, что технический комитет *ECMAScript* смог справиться с таким трудным заданием, стал добрым знаменем.

В 2016 году увидело свет небольшое обновление *ECMAScript*. Эта версия стала результатом нового процесса подготовки, принятого в TC-39. Все новые предложения должны пройти через четыре стадии. Предложение, достигшее четвёртой стадии, имеет все шансы быть включенным в следующую версию *ECMAScript* (однако комитет имеет право отложить его для более поздней версии). Таким образом, каждое предложение разрабатывается индивидуально (разумеется, с учётом его взаимодействия с другими предложениями), не тормозя разработку *ECMAScript*.

Если предложение готово к включению в стандарт, и достаточное количество других предложений достигло четвёртой стадии, в свет выходит новая версия *ECMAScript*.

Версия, выпущенная в 2016 году, была очень маленькой. Она включала в себя:

1 Оператор возведения в степень (\*\*).

2 *Array.prototype.includes*.

3 Несколько незначительных поправок (генераторы не могут быть использованы с *new* и т.д.).

Самым важным предложением, достигшим четвёртой стадии, является *async/await*. это расширение синтаксиса для *JavaScript*, которое делает работу с промисами более приятной.

Другие предложения, достигшие четвёртой стадии, совсем небольшие:

1 *Object.values* и *Object.entries*.

2 Выравнивание строк.

3 *Object.getPrototypeOfDescriptors*.

4 Разделители-запятые в параметрах функций.

Все эти предложения предназначены для релиза 2017 года, однако комитет имеет право отложить их до следующего релиза. Но даже одно лишь дополнение в лице *async/await* будет потрясающим.

Будущее на этом не заканчивается. Давайте посмотрим на некоторые другие предложения, чтобы получить представление о том, что ждёт нас впереди. Вот несколько самых интересных:

1 *SIMD API*.

2 Асинхронные итераторы (*async/await* + итерация).

3 Стрелочные генераторы.

4 Операции с 64-битными целыми числами.

5 Области (изоляции состояний).

6 Общая память и *Atoms*.

*JavaScript* всё больше становится похож на язык общего назначения. Но есть ещё одна большая деталь в будущем *JavaScript*, которая внесёт свои коррективы.

Огромное количество библиотек и фреймворков, появившихся после выхода *ECMAScript 5*, а также общее развитие языка, сделали *JavaScript* интересной целью для других языков. Для больших кодовых структур функциональная совместимость является ключевой потребностью. Возьмите, к примеру, игры. Самым распространённым языком, на котором пишутся игры, является *C++*, благодаря чему их можно портировать на большое количество архитектур. Тем не менее портирование для браузера *Windows* или консольной игры считалось невыполнимой задачей. Однако это стало возможным благодаря стремительному развитию и небывалой эффективности сегодняшних виртуальных машин *JavaScript*. Именно для выполнения подобных задач на свет появились инструменты вроде *Emscripten*.

Быстро сориентировавшись в ситуации, *Mozilla* начала работу над тем, чтобы сделать *JavaScript* подходящей целью для компиляторов. Так на свет появился *Asm.js* – подмножество *JavaScript*, идеально подходящее в качестве подобной цели. Виртуальные машины *JavaScript* могут быть оптимизированы для распознавания этого подмножества и производства кода, намного лучшего, чем тот, который генерируют текущие виртуальные машины. Благодаря *JavaScript* браузеры медленно становятся новой целью для компиляторов

И всё же существуют огромные ограничения, которые не в состоянии преодолеть даже *Asm.js*. В *JavaScript* необходимо внести такие изменения, которые расходятся с его текущим предназначением. Нужно что-то совершенно иное для того, чтобы сделать веб достойной целью для других языков программирования. И именно для этого предназначен *WebAssembly* – низкоуровневый язык программирования для веба. Любая программа может быть скомпилирована в *WebAssembly* при помощи подходящего компилятора и затем запущена в подходящей виртуальной машине (виртуальные машины *JavaScript* могут предоставить необходимый уровень семантики). Первые версии *WebAssembly* имеют стопроцентную совместимость со спецификацией *Web.js*. *WebAssembly* обещает не только более быстрое время загрузки (байт-код обрабатывается быстрее, чем текст), но и возможность оптимизации, недоступной в *Asm.js*. Представьте себе интернет с идеальной функциональной совместимостью между *JavaScript* и вашим языком программирования.

На первый взгляд это может помешать росту *JavaScript*, но на самом деле всё совершенно иначе. Благодаря тому, что другие языки и фреймворки получают функциональную совместимость с *JavaScript*, он сможет продолжать свое развитие в качестве языка общего назначения. И *WebAssembly* является необходимым инструментом для этого.

В настоящий момент *dev*-версии *Chrome*, *Firefox* и *Microsoft Edge* имеют начальную поддержку *WebAssembly* и способны проигрывать демо-приложения.

История *JavaScript* длинна и полна неожиданных поворотов. Изначально предложенный в качестве «*Scheme* для веба», он позаимствовал свой синтаксис у *Java*. Его первый прототип был разработан за несколько недель. Подстраиваясь под требования рынка, он сменил три названия менее чем за два года, после чего был стандартизирован и получил название, более подходящее для кожного заболевания. После трёх успешных релизов язык варился в адских котлах почти восемь лет. Затем, благодаря успеху одной-единственной технологии (*AJAX*), сообщество смогло побороть противоречия и возобновить разработку. Версия 4 была заброшена, а небольшое обновление, известное, как версия 3.1, было переименовано в версию 5. Версия 6 провела в разработке много лет (опять), но на этот раз комитет успешно закончил работу, сменив цифру в названии на 2015. Это было очень большое обновление, и его реализация заняла много времени. В результате *JavaScript* получил второе дыхание. Сообщество оживилось как никогда до этого. Благодаря *Node.js*, *V8* и другим проектам *JavaScript* поднялся на высоты, о которых разработчики первой версии даже не задумывались, а благодаря *Asm.js* и *WebAssembly* он взлетит ещё выше. Активные предложения, пребывающие в разных стадиях, делают будущее *JavaScript* чистым и безоблачным. Пройдя долгий путь, полный неожиданных поворотов и препятствий, *JavaScript* остаётся одним из самых успешных языков в истории

программирования. И это – лучшее доказательство его надежности. Всегда ставьте на *JavaScript*.

### **3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА**

#### **3.1 Обоснование необходимости анализа**

## **4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ**

### **4.1 Функции программного обеспечения**



## **ЗАКЛЮЧЕНИЕ**

## СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Importance of Linux OS [Электронный ресурс]. – Режим доступа: <https://www.sevenmentor.com/importance-of-linux-os>. – Дата доступа: 29.03.2024.

[2] Architecture of linux operating system [Электронный ресурс]. – Режим доступа: <https://www.geeksforgeeks.org/architecture-of-linux-operating-system/>. – Дата доступа: 29.03.2024.

[3] What is Linux Used For? [Электронный ресурс]. – Режим доступа: <https://www.lenovo.com/za/en/faqs/operating-systems/what-is-linux-used-for>. – Дата доступа: 29.03.2024.

[4] Linux. [Электронный ресурс]. – Режим доступа: <https://znanierussia.ru/articles/Linux>. – Дата доступа: 29.03.2024

[5] JavaScript. [Электронный ресурс]. – Режим доступа: <https://blog.skillfactory.ru/glossary/javascript/>. – Дата доступа: 29.03.2024

[6] Краткая история JavaScript. Часть 1. [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/companies/livotyping/articles/324196/>. – Дата доступа: 29.03.2024

[7] Краткая история JavaScript. Часть 2. [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/companies/livotyping/articles/324506/>. – Дата доступа: 29.03.2024