МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

Выпускная квалификационная работа

на тему

«Статический анализ в проверке корректности программного кода»

Обучающегося 4 курса
очной формы обучения
направление подготовки:
09.03.01 «Информатика и вычислительная техника»
направленность (профиль):
Технологии разработки программного обеспечения
Цирулика Ивана Александровича

Руководитель выпускной квалификационной работы:
Кандидат педагогических наук, доцент, доцент
Государев Илья Борисович

Санкт-Петербург
2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В 2021 году, почти все жители планеты сталкиваются с миром цифровых технологий. Интернет-магазины, сервисы записи к врачу, электронные документообороты, видеоигры, текстовые процессоры и многое другое — все это является программным обеспечением или даже целыми информационными системами. Если разбирать любое ПО, то в его основе, как-правило, будут обычные текстовые файлы – исходный код программы. Разработка ПО в настоящее время – очень высокобюджетный и долгий процесс, в котором, порой, задействованы сотни человек - разработчики, тестировщики, разного вида менеджеры основная задача которых – выпуск качественного программного продукта. Самая сложная часть данного процесса – написание исходного кода, ведь, по сути, это написание огромного размера текста, например исходный код операционных систем может насчитывать миллионы строк кода. Для обеспечения максимального качества осуществляется множественное тестирование программного обеспечения [6]. QC (контроль качества) отвечает за выполнение процесса тестирования (написание/прохождение кейсов, поиск/заведение дефектов и так далее) [8];

QA (обеспечения качества) отвечает за создание системы контроля, которая будет предотвращать появление багов уже на этапе разработки ПО, сокращая количество выявленных дефектов на этапе тестирования; грубо говоря – построение самого процесса тестирования [26].

Все это является проверкой корректности программного кода, но этот способ требует интерпретации или компиляции программного кода, т.е. несоответствия работы кода и ожидаемого результата будут определены только после написания кода и его выполнения – так называемый динамический анализ. Но существует и другой способ, это так называемый статический анализ кода.

В зависимости от используемого инструмента глубина анализа может варьироваться от определения поведения отдельных операторов до анализа,

включающего весь имеющийся исходный код. Способы использования полученной в ходе анализа информации также различны — от выявления мест, возможно содержащих ошибки, до формальных методов, позволяющих математически доказать какие-либо свойства программы (например, соответствие поведения спецификации).

Некоторые разработчики считают программные метрики и обратное проектирование формами статического анализа. Получение метрик и статический анализ часто совмещаются, особенно при создании встраиваемых систем. Поэтому очень важно ознакомить как можно больше разработчиков с инструментами статического анализа, так как это поможет сократить время на тестирование, получить более качественный код и приучить разработчиков к общепринятым спецификациям.

Актуальность ВКР обусловлена быстрым ростом популярности программирования и разработки ПО. А в частности, сценарии на языке JavaScript или Python становятся причиной ошибок в связи с динамической типизацией и другими особенностями языка, поэтому востребован статический анализ, позволяющий выявить ошибки до стадии выполнения. В том числе, любая база кода в конце концов становится огромной, поэтому простые ошибки, которые не были бы заметны при написании, могут стать препятствиями и добавить дополнительные часы отладки. Поэтому в игру вступают инструменты статического анализа кода, которые помогают разработчикам обнаружить такие проблемы.

Практическая значимость исследования заключается в создании удобного сервиса, который сможет обеспечить доступ к одному из инструментов статического анализа и соответственно способствовать росту числа

разработчиков, пользующихся инструментом для статического анализа и как следствие улучшение качества программного кода.

Вследствие этого **целью** данной выпускной квалификационной работы является создание веб-сервиса, в который будет интегрированы инструменты для статического анализа программного кода,

В данной работе описан процесс разработки и внедрения как клиентской, так и серверной части данного веб-сервиса.

Для достижения данной цели были сформулированы и выполнены следующие задачи:

1. Изучение средств для статического и динамического анализа и выбор наиболее подходящего и актуального решения на данный момент.
2. Разработка архитектуры веб-сервиса с учетом возможного масштабирования.
3. Проектирование базы данных для отображения связей всех участвующих в процессе сущностей.
4. Создание технического задания для упрощения разработки сервиса.
5. Разработка серверной части для имплементации функционала сервиса.
6. Верстка клиентской (frontend) части приложения для реализации клиент-серверного взаимодействия.

Объектом исследования в выпускной квалификационной работе являются инструменты для проверки корректности программного кода. Предметом ВКР является применение данного инструментария в процессе разработки программного обеспечения.

Выпускная квалификационная работа имеет структуру, состоящую из введения, двух глав и заключения.

В первой главе представлен анализ предметной области, в частности рассмотрены 2 основных подхода к проверке корректности кода, а также проведен анализ инструментов и технологий для реализации выбранного подхода.

Во второй главе представлены прикладные аспекты разработки сервиса – техническое задание данного ресурса в соответствии с ГОСТ [12], схема и принципы работы разрабатываемого приложения, схема взаимодействия сущностей в базе данных и макет пользовательского интерфейса.

ГЛАВА 1. АНАЛИЗ ПРОБЛЕМЫ И СУЩЕСТВУЮЩИХ РЕШЕНИЙ. ПОДГОТОВКА К РАЗРАБОТКЕ

1.1. Анализ проблемы

Проверка корректности программного кода – один из важнейших аспектов разработки ПО. Именно она позволяет выпускать качественные программные продукты, которые лишены неопределенного поведения при всевозможных действиях пользователя. Как упоминалось вj введении существует три разных подхода к проверке корректности кода:

1. Динамический анализ программного кода

Динамический анализ на практике можно реализовать по-разному. Первый и самый распространённый способ, к которому прибегают все разработчики неосознанно – запуск программы и проверка корректности ее отработки. Если сильно упрощать все сводится к следующему циклу (см. Рисунок 1)

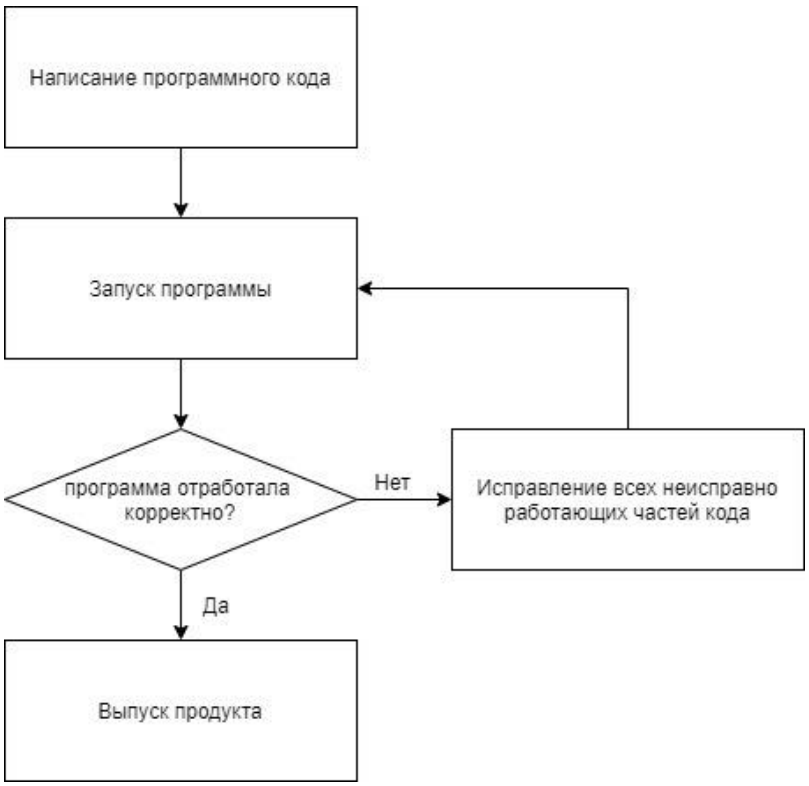

Рисунок 1. Схема упрощенного динамического анализа

Это самый неэффективный и временно затратный способ проверки – «решение в лоб». Существует куда более комплексный и научный подход в данном типе анализа – теория тестирования программного обеспечения, в котором программа выполняется не просто при помощи интерпретатора или компилятора, а при помощи специальных утилит, расширений языка программирования или его собственными средствами. Существует несколько уровней (этапов) тестирования ПО:

1. Модульное тестирование

Также называется компонентным тестированием. Такой вид тестирования является самым первым уровнем тестирования. На этом этапе тестированию подвергаются отдельные компоненты программы, такие как функции, классы, методы, структуры данных и другие. На данным этапе происходит локализация проблемы внутри одного модуля программы и последующее исправление неисправности [28].

2. Интеграционное тестирование

Данный вид тестирования выполняется после модульного тестирования и проверяет корректность взаимодействия между частями всей системы. Существует несколько подходов к осуществлению данного вида тестирования:

- **Снизу вверх.**

При выборе данного подхода в начале происходит сборка и тестирование самых низкоуровневых модулей, затем на уровень выше и так далее до тех пор, пока не будет достигнута вершина иерархии. Данный подход позволяет охватить весь тестируемый код, но требует готовности всех модулей на всех уровнях к прохождению тестов.

- **Сверху вниз.**

В данном подходе происходит сборка и тестирование от самых высокоуровневых модулей к модулям нижестоящего уровня. При данном подходе, как правило, используют некие стандартные «заглушки» для тех модулей нижестоящего уровня, которые еще не были охвачены тестами.

- **Большой взрыв**

Данный подход является самым хаотичным, так как происходит сборка всех модулей в один с последующим тестированием. Плюсом такого подхода является скорость осуществления проверки, но требуется очень детальная проработка самих тестов.

3. Системное тестирование

Основной задачей данного этапа тестирования является проверка не только функциональных, но и общих требований в системе в целом. На данном этапе могут выявляться целый ряд дефектов, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и другие [30]

4. Приемочное тестирование

Последний этап тестирования, который проверяет готовность всей системы к эксплуатации. Данный этап является самым комплексным и важным, так как именно на нем происходит проверка корректности основных бизнес-операций системы. У приемочного тестирования существует всего 2 результата:

- В случае если проект не проходит весь набор тестов, он возвращается назад на доработку.
- В случае успешного прохождения продукт считается готовым, и заказчик осуществляет его ввод в эксплуатацию.

За счет использования тестов на всех этапах, современное ПО является высококачественным продуктом с почти полным отсутствием программных ошибок, но у всего есть своя цена. Тестирование должен кто-то проводить, тесты должны быть кем-то составлена – дополнительные инженеры ответственные за тестирования, подписки на сервисы автоматизации тестирования и многое другое. И, в связи с этим растет и цена данных программных продуктов.

2. Статический анализ программного кода

Статический анализ кода – это принципиально другой способ проверки работоспособности и корректности программного кода. Такой вид анализа обрабатывает лишь сами исходные файлы без непосредственного выполнения [22]. В основном он производится при помощи стороннего ПО, но существует один способ, который, проводится без участия компьютера:

Обзор кода. Один из старейших методов выявления дефектов кода, но тем не менее один из самых надежных. Он заключается в совместном внимательном чтении исходного кода и высказывании рекомендаций по его улучшению. В процессе чтения кода выявляются ошибки или участки кода, которые могут стать ошибочными в будущем. Также считается, что автор кода во время обзора не должен давать объяснений, как работает та или иная часть программы. Алгоритм работы должен быть понятен непосредственно из текста программы и комментариев. Если это условие не выполняется, то код должен быть доработан. Производится он как правило людьми; поскольку возможности анализа у человека намного шире нежели чем у компьютера, то такой анализ получатся куда более комплексным и понятным человеку. Как можно заметить, такой подход к анализу хоть и очень надежен, тем не менее требует очень больших человеческих затрат.

Остальные способы, как правило, основываются на результатах работы статического анализатора, полученных из исходного кода. В схеме работы статического анализатора можно выделить три основных шага:

1. Построение промежуточного представления (промежуточное представление также называют внутренним представлением или моделью кода).

2. Применение алгоритмов статического анализа, в результате работы которых модель кода дополняется новой информацией.
3. Применение правил поиска уязвимостей к дополненной модели кода.

Рассмотрим каждый их них подробнее:

1. Построение промежуточного представления

В зависимости от реализации статического анализатора он может работать с разными представлениями исходного кода, например, с непосредственно исходным текстом, деревом разбора, трехадресным кодом, графом потока управления, байт-кодом, потоком лексем, и так далее.[23]

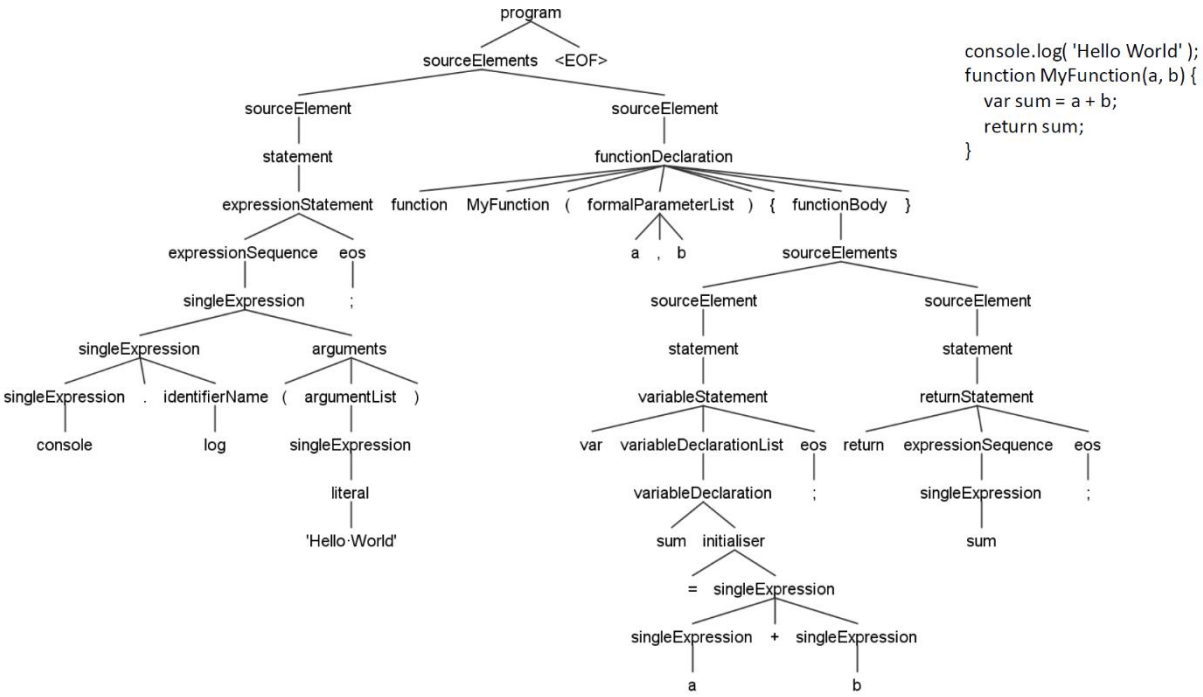

Рисунок 2 Пример промежуточного представления

Для построения дерева разбора (AST, Abstract Syntax Tree), как правило применяется Лексический и Синтаксический анализ. Лексический анализ разбивает текст программы на минимальные смысловые элементы (токены), на

выходе получая поток лексем, данный процесс также еще называют токенизацией. Синтаксический анализ осуществляет проверку на то, что поток лексем соответствует грамматике языка программирования, то есть полученный поток лексем является верным с точки зрения правил и грамматики языка. В результате синтаксического анализа происходит построение дерева разбора – структуры, которая моделирует исходный текст программы. Далее применяется семантический анализ, он проверяет выполнение более сложных условий, например, соответствие типов данных в инструкциях присваивания.

Дерево разбора можно использовать как внутреннее представление. Также из дерева разбора можно получить другие модели. Например, можно перевести его в трехадресный код, по которому, в свою очередь, строится граф потока управления (CFG). Обычно CFG является основной моделью для алгоритмов статического анализа.

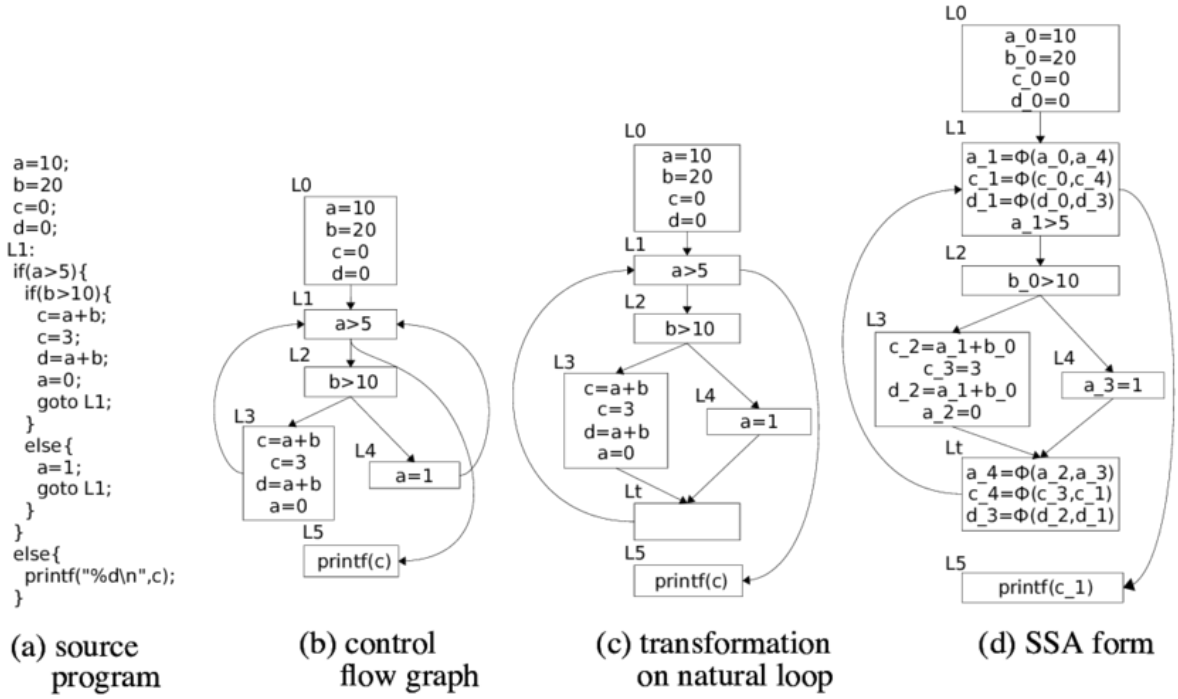

Рисунок 3 Преобразования, совершаемые статическим анализатором

2. Применение алгоритмов статического анализа

Существует несколько основных алгоритмов для статического анализа:

- Анализ потока данных

  Анализ потока данных — это метод сбора информации о возможном наборе значений, вычисленных в различных точках компьютерной программы. Граф потока управления программы (CFG) используется для определения тех частей программы, на которые может распространяться конкретное значение, присвоенное переменной. Собранная информация часто используется компиляторами при оптимизации программы. Канонический пример анализа потока данных - определение определений. Анализ потока данных -

- Символьное выполнение или абстрактная интерпретация
  В этом подходе происходит выполнение программы на абстрактных доменах, вычисление и распространение ограничений по данным в программе. С помощью такого подхода можно не просто находить уязвимость, но и вычислить условия на входные данные, при которых уязвимость является эксплуатируемой. Однако у такого подхода есть и серьезные минусы – при стандартных решениях на реальных программах алгоритмы экспоненциально взрываются, а оптимизации приводят к серьезным потерям в качестве анализа.

3. Применение правил поиска уязвимостей

В результате применения указанных выше алгоритмов промежуточное представление дополняется информацией, необходимой для поиска уязвимостей. Например, в модели кода появляется информация о том, каким переменным принадлежат определенные флаги, какие данные являются константными. Правила поиска уязвимостей формулируются в терминах модели кода. Правила описывают, какие признаки в итоговом промежуточном представлении могут говорить о наличии уязвимости.

Вывод:

Главный недостатком статического анализа по отношению к динамическому является наличие ложных срабатываний, достаточно большое потребление ресурсов и времени обработки на больших объемах кода, это связано в первую очередь с алгоритмами анализа, так как быстрый анализ не способен выявить множество уязвимостей.

Главное и ключевое преимущество статического анализа заключается в полном покрытии всего анализируемого кода. К тому же возможность внедрения статического анализа на самых ранних стадиях проекта, позволяет значительно уменьшить стоимость устранения дефектов программного обеспечение. Чем меньше время обнаружения дефекта, тем ниже будет обходиться стоимость ее последующего исправления. Так, исправление ошибки на этапе тестирования обойдется в десять раз дороже, чем на этапе конструирования (написания кода) [21, с.462]:

	Время обнаружения дефекта				
Время внесения дефекта	Выработка требований	Проектирование архитектуры	Конструирование (кодирование)	Тестирование	После выпуска ПО
Выработка требований	1	3	5-10	10	10-100
Проектирование архитектуры	-	1	10	15	25-100
Конструирование (кодирование)	-	-	1	10	10-25

Статический анализ

Рисунок 4 Средняя стоимость исправления дефектов в зависимости от времени их внесения и обнаружения

Инструменты статического анализа позволяют выявить большое количество ошибок на этапе конструирования (написания) программного кода, что очень

существенно снижает стоимость разработки всего проекта. Потому в дальнейшем мы сосредоточимся лишь на этом способе проверки корректности.

1.2. Обзор существующих решений проблемы

На данный момент существует достаточно большое множество инструментов статического анализа кода для языков JavaScript и Python. Они очень сильно отличается между собой, не только по используемым технологиям, но и по итоговому результату, поэтому рассмотрим некоторые наиболее известные и применяемые на данный момент [17]:

1. «DeepCode»

«DeepCode» — система анализа кода на базе глубинного обучения. В основе данного инструмента лежит не только применяемый в других статических анализаторах система семантического анализа, но и нейронная сеть, обучающаяся на основе всех публичных репозиториев GitHub

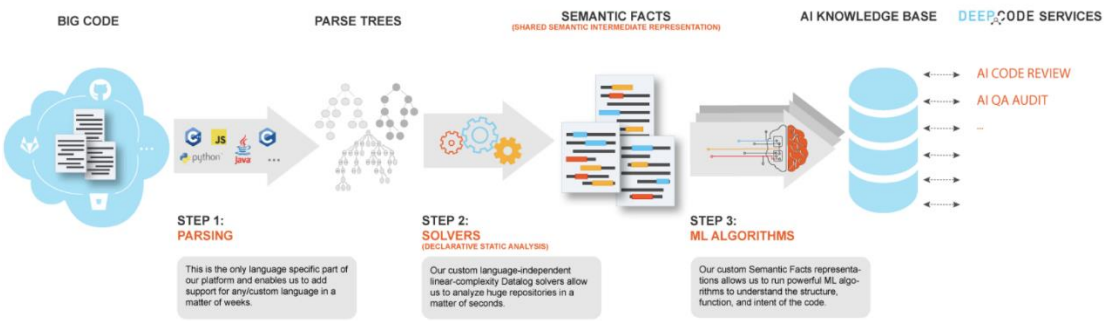

Рисунок 5. Схема функционирования «DeepCode»[7]

По итогу мы получаем инструмент, способный проводить анализ переданного ему кода не только со стороны банального синтаксиса и наличия фактически ошибок, но и с точки зрения того насколько полезно он выполняет свою работу (см. Рисунок

2). Например, приводится анализ всех предыдущей изменений и запросов на изменения в главной ветке на GitHub. Данный инструмент не только способен показывать разработчику количество ошибок, но и провести анализ нового функционала и проверить его совместимость с уже имеющейся базой кода. «DeepCode» можно применить несколькими способами: как плагин к некоторым IDE; при помощи официального веб-портала, можно дать доступ к своим репозиториям на портале «GitHub», и система сама проведет анализ; воспользоваться пакетным менеджером pip и установить локальный клиент, устанавливающий связь с инструментом на ресурсе разработчиков.

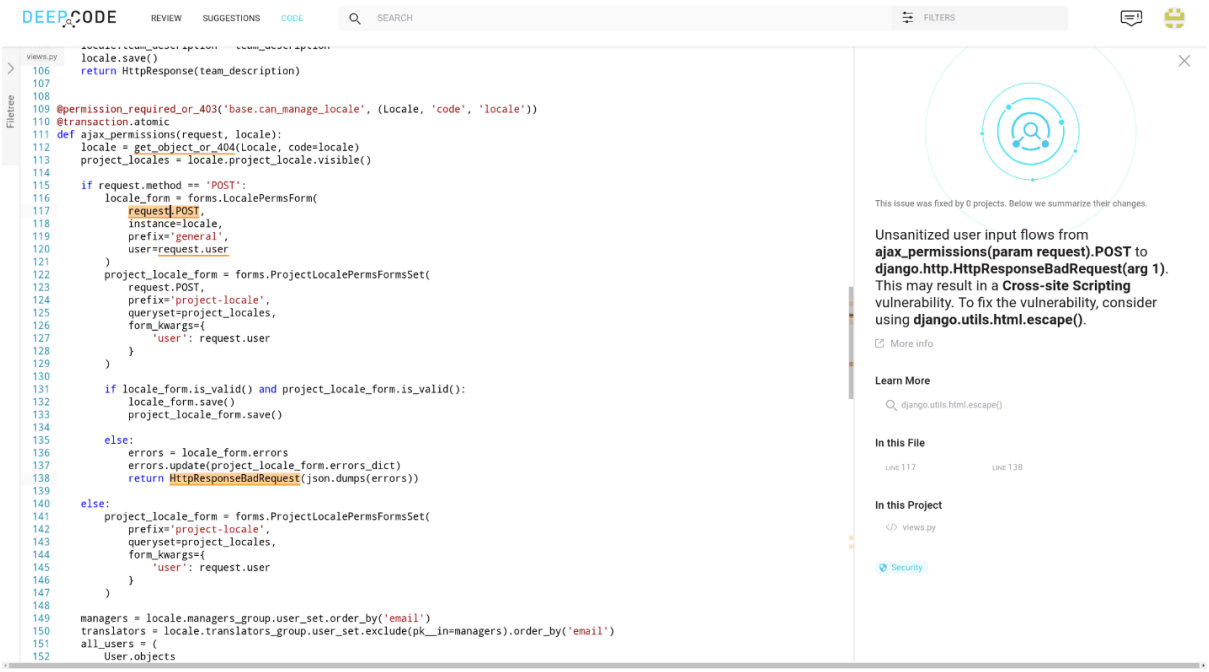

Рисунок 6 Пример использования DeepCode

2. «Kite»

«Kite» — представляет собой интегрирующуюся в IDE систему, состоящую из документации языка Python и JavaScirpt, а также средств для создания контекстных подсказок, вызываемых в процессе работы разработчика. Данный инструмент индексирует всю кодовую базу проекта, что почти всегда гарантирует вывод наиболее релевантных подсказок, раз в неделю происходит генерация отчета, в которой можно явно увидеть количество раз, в которых был использован предложенный вариант.

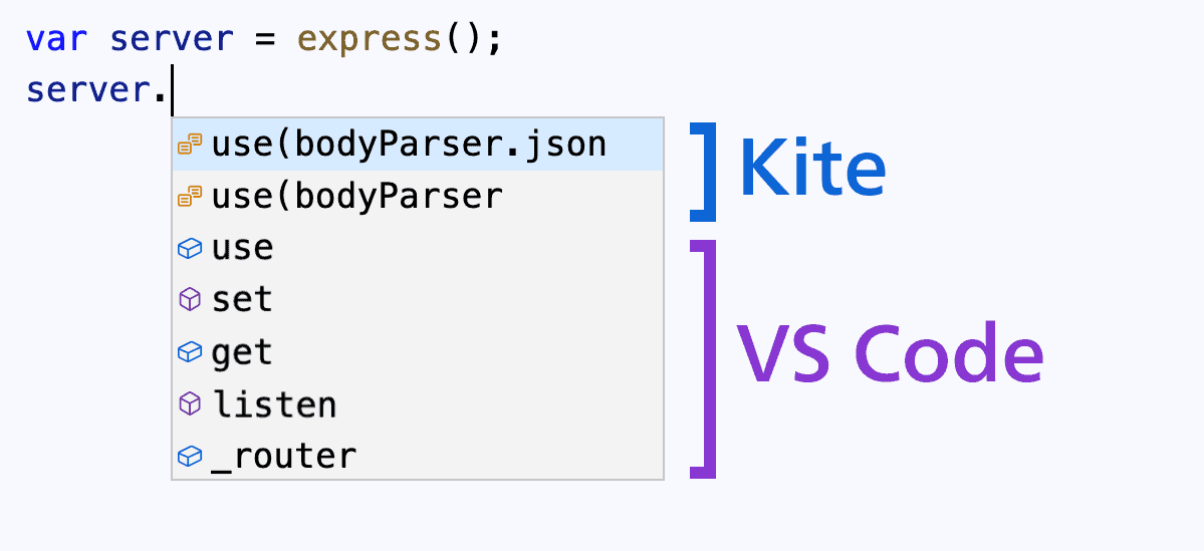

Рисунок 7. Сравнение встроенных подсказок “Visual studio code” и подсказок "Kite"

3. «Tabnine»

Tabnine представляет из себя плагин, встраиваемый в единую среду разработки или текстовый процессор. Его основная и главная идея заключается в том, что нейронная сеть, лежащая в его основе, обучается на общедоступных кодовых базах, и потому если определенные фрагменты кода могли быть использованы в других проектах, ИИ Tabnine мгновенно выдает предложения, которые он уже встречал во время обучения. Это сэкономит массу времени и позволит соответствовать лучшим практикам, избегая ненужных и досадных опечаток. Продвинутый искусственный интеллект и машинное обучение делают Tabnine инструментом для экономии времени, который является одним из самых популярных решений среди многих разработчиков на данный момент.

4. «ESLint»

ESLint является одним из самых новых инструментов [31]. Он был разработан, чтобы быть легко расширяемым, поставляется с большим количеством пользовательских правил, и его легко установить больше в виде плагинов. Он дает краткий вывод, но по умолчанию используется имя правила, поэтому всегда известно, какие правила вызывают сообщения об ошибках. Документация ESLint

немного неполная. Список правил легко отслеживается и сгруппирован в логические категории, но инструкции по настройке немного запутаны в некоторых местах. Тем не менее, он предлагает ссылки на интеграцию редакторов, плагины и примеры в одном месте.

5. «JSLint»

«JSLint» был первым статическим анализатором для JavaScript. Его можно запустить через официальный сайт или воспользоваться программной надстройкой для запуска на локальных проектах. JSLint определяет специальное подмножество JavaScript, более строгое чем описанное в третьем издании спецификации ECMAScript, это подмножество тесно связано с рекомендациями, которые Дуглас Крокфорд, дал в девятой главе книги JavaScript — сильные стороны [15]. JSLint не гарантирует правильность программы, но выявляет синтаксические ошибки и ряд потенциальных проблем, касающихся как оформления кода, так и семантики программы.

6. «JSHint»

JSHint — это инструмент, разработанный сообществом, который обнаруживает ошибки и потенциальные проблемы в коде JavaScript [20]. JSHint находится в открытом доступе и всегда будет оставаться таковым. JSHint сканирует программу, написанную на JavaScript, и сообщает о часто встречающихся ошибках и потенциальных багах. Потенциальной проблемой может быть синтаксическая ошибка, ошибка из-за неявного преобразования типов, утечка переменной или что-то другое.

1.3. Анализ существующих решений

1. «DeepCode»

DeepCode — инструмент, в котором машинное обучение присутствует в качестве компонента что делает его невероятно эффективным в отличие от

стандартных средств, которые используют лишь классические методы. Количество правил валидации этого инструмента действительно очень впечатляет - более 250 тысяч правил. Этот инструмент обучается на основе изменений, вносимых разработчиками в исходный код открытых проектов (миллион репозиториев) так что актуальность его подсказок и советов всегда остается максимальной.

С точки зрения поддержания качества кода – это наиболее мощный и современный инструмент. Наличие AI/ML-обзора кода, помогающего следовать лучшим практикам кодирования, — это отличное звено для добавления в CI/CD конвейер. Рекомендации, полученные на основе сканирований более чем миллионов других репозиториев, на которых DeepCode обучается, дает очень большую уверенность в том, что полученный в итоге код безопасен и хорошо написан. Кроме того, возможность интеграции данного инструмента в собственные репозитории Github без особых сложностей, а также расширение для Visual Studio Code отлично подходит для сканирования проблем в собственных проектах в реальном времени.

Достоинства:

- Использование технологий ИИ
- Огромная обучающая выборка
- Легкость интеграции в собственные проекты
- Поддержка большого количества языков программирования
- Актуальные подсказки во время написания кода

Недостатки:

- Нет возможности использования самого ядра для встраивания анализатора непосредственно в коде
- Платный тариф при использовании большой командой

2. «Kite»

Kite тоже является инструментов который не только использует уже ставшими стандартом принципы работы статического анализатора, но и самые современные технологии. Разработчики обучили новую модель глубокого обучения на 22 миллионах файлов JavaScript с открытым исходным кодом, чтобы гарантировать работу с самыми современными и популярными библиотеками и фреймворками, такими как «React», «Vue», «Angular». Главным плюсом такого подхода к анализу является его постоянная актуальность, несмотря на быстрое развитие технологий и экосистемы языка JavaScript – любая новая технологий и способ ее применения, появившийся в открытом источнике, с достаточно высокой скоростью будет обработан ядром данного инструмента и добавлен в общую базу советов. Kite для JavaScript бесплатен и работает только на локальных устройствах.

Достоинства:

- Актуальные советы по автозаполнению
- Поддержка сразу JavaScript и Python.
- Поддержка самых передовых библиотек языка

Недостатки:

- Только локальное использование
- Нет возможности использовать API

3. Tabnine

TabNine тоже создан на базе технологий машинного обучения. В поддерживаемый им набор языков входит не только Python и JavaScript но и 18 других языков, например Haskell или C++. Данный инструмент использует специально адаптированную для программного кода модель GPT-2 для обработки естественного языка. Данный инструмент не только работает с синтаксическими конструкциями языка программирования, но и помогает с написаниями документации и комментариев внутри исходного кода. Выборка для обучения состояла из 2 млн файлов с GitHub. В результате TabNine научился улавливать

закономерности и успешно дополнять их самостоятельно. Единственный способ его интеграции представлен в виде плагина. Возможность установки данного плагина есть во всех самых популярных редакторах - Microsoft Visual Studio Code и Sublime Text.

Достоинства:

- Актуальные советы по автозаполнению
- Легкость использования
- Поддержка большого количества языков программирования

Недостатки:

- Только локальное использование внутри редактора кода
- Нет возможности использовать API

4. Eslint

Из рассмотренных мной линтеров является наиболее современным и новым. Выполнить первичную настройку можно при помощи пакетного менеджера «Node.js». При первом запуске создается конфигурационный файл, в котором располагаются правила валидации, указывается среда, в которой Javascript-код применяется, а также указывается в соответствии с какой версией EcmaScript будет выполнена валидация.

Дополнительно есть возможность вызывать и выполнять анализ прямо внутри программного Java Script кода и благодаря API получать форматированный результат статического анализа прямо во время выполнения программы, причем ESLint позволяет выбрать один из множества форматов, например html, xml или json [4].

Достоинства:

- Гибкость и глубина настройки
- Есть возможность использовать инструменты анализа внутри JavaScript кода напрямую через API

- Понятный результат
- Лучшая из возможных поддержка ES6, а также единственный инструмент для поддержки JSX

Недостатки:

- Обязательно наличие конфигурационного файла
- Низкая скорость обработки

5. JSLint

JSLint – является самым старым из рассмотренных инструментов. Дуглас Крокфорд создал его в 2002 году для обеспечения того стиля написания кода, которые по его опыту, представляют собой лучшие практики JavaScript. Если стиль написания кода совпадает с его стилем, то JSLint может быть хорошим инструментом. К тому же его использование является наиболее простым, необходимо установить его, и он сразу готов к работе.

Главным недостатком является то, что JSLint невозможно настроить и расширить. Невозможно отключить многие функции вообще, а многим из них не хватает документации. Официальный сайт не очень полезен, например, ему не хватает информации о том, как его интегрировать с редактором.

Достоинства:

- Легкость установки и использования

Недостатки:

- JSLint не имеет файла конфигурации, что может вызывать сложности если вам нужно изменить настройки
- Очень грубая настройка, нет возможностей отключить многие функции
- Нет возможности использовать API
- Нет возможности добавить свои правила для валидации
- Нет четкого обозначения какое именно правило нарушено

6. JSHint

JSHint был создан как более настраиваемая версия JSLint (форком которого он является). Есть возможность выполнить настройку каждого правила и поместить их в файл конфигурации, что упрощает использование JSHint в больших проектах. JSHint также имеет отличнейшую документацию для каждого из правил валидации, поэтому однозначно можно понять, что они делают. Выполнить его интеграцию в редакторы кода, тоже не составляет особо труда.

Небольшой недостаток JSHint заключается в том, что по умолчанию у него используется очень простая конфигурация. Поэтому перед использованием нужно выполнить небольшую настройку, чтобы сделать его полезным. По сравнению с ESLint, определение правил, которые необходимо изменить, для включения или отключения определенных сообщений об ошибках.

Достоинства:

- Широкий выбор настроек
- Поддержка конфигурационного файла
- Поддержка многих библиотек для простой установки.
- Базовая поддержка ES6

Недостатки:

- Сложность определения правила, вызвавшего ошибку
- Имеет два типа опций: принудительное и простое (что может быть использовано для более строгой настройки JSHint или для подавления его предупреждений). Это может сделать конфигурацию немного запутанной
- Нет возможности использовать API
- Поддержка пользовательских правил

Выводы по главе 1

На основе анализа всех рассматриваемых инструментов, был сделан вывод что на данный момент не существует веб-ресурса, который максимально просто позволял бы программисту воспользоваться средствами статического анализа, потому было принято решение разработать такой сервис, пользуясь технологической базой библиотеки «Eslint» в сочетании с платформой «Node.js».

ГЛАВА 2. РАЗРАБОТКА ПРИЛОЖЕНИЯ

Для успешной разработки данного приложения необходимо пройти через ряд устоявшихся шагов:

2.1. Техническое задание

2.1.1. ОБЩИЕ ПОЛОЖЕНИЯ

2.1.1.1. Наименование системы

2.1.1.1.1. Полное наименование системы

Полное наименование системы: Веб-сервис для проверки работоспособности программного кода.

2.1.1.2. Наименование организаций – Заказчика и Разработчика

2.1.1.2.1. Разработчик

Разработчик: Цирулик Иван Александрович

2.1.1.3. Плановые сроки начала и окончания работы

2.1.1.3.1. Общие даты

Плановый срок начала работ по созданию веб-сервиса – 01 апреля 2021 года.

Плановый срок окончания работ по созданию веб-сервиса – 7 мая 2022 года.

2.1.1.3.2. Стадии и этапы разработки

- Разработка документации - до 01.04.2020
- Разработка тестовой версии с ограниченным функционалом - до 12.04.2021
- Тестирование и сбор обратной связи - до 14.04.2021
- Разработка тестовой версии ПО с расширенным функционалом - до 12.10.2021
- Тестирование и сбор обратной связи - до 15.05.2022

- Разработка финальной версии ПО - до 22.09.2022
- Начало процесса введения в эксплуатацию - до 7.05.2023

2.1.1.4. Основания для разработки

Основанием для разработки является возможная научная ценность данного проекта, а также прикладное применение в процессе разработки ПО.

2.1.1.5. Порядок оформления и предъявления заказчику результатов работ по созданию системы

ПО тестируется путем общего доступа к веб-сервису. Все замечания записываются и предоставляются разработчику для дальнейшего исправления.

Работа принимается, если она была выполнена в установленный срок и у заказчика не осталось замечаний к исправлению, относящихся к требованиям к программе в данном ТЗ. В обратном случае разрабатывается новое ТЗ с новыми сроками работы и перечисленными исправлениями.

2.1.1.6. Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ

При разработке автоматизированной системы и создании проектно-эксплуатационной документации Исполнитель должен руководствоваться требованиями следующих нормативных документов:

– ГОСТ 19.201-78. ТЕХНИЧЕСКОЕ ЗАДАНИЕ. ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И ОФОРМЛЕНИЮ;

– ГОСТ 34.602-89. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания;

2.1.2. НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ

2.1.2.1. Назначение системы

ПО предназначено для помощи в написании программ на языке JavaScript

2.1.2.2. Цели создания системы

Основными целями создания веб-сервиса являются:

– Повышение качества программного кода за счет статического анализа кода и выдачи своевременных предупреждений;
– Повышение читаемости программного кода за счет приведения программного кода к общепринятым нотациям;
– Упрощение написания программного кода за счет авто подсказок.

2.1.3. ТРЕБОВАНИЯ К СИСТЕМЕ

2.1.3.1. Требования к функциям (задачам), выполняемым системой

– ПО должно быть выполнено с использованием клиент-серверной архитектуры для возможности удаленного использования с помощью браузера;

– ПО должно обеспечивать возможность администрирования. В частности добавление, изменение, удаление пользователей. Установка пользователям прав доступа к документам;

– ПО должно обеспечивать функции авторизации для идентификации пользователя;

– ПО должно обеспечивать возможность загрузки файла на сервер;

– ПО должно обеспечивать доступ к уже загруженным на сервер файлам и результатам их анализа;

– ПО должно быть разработано с учетом возможной высокой нагрузки;

– ПО должно включать в себя REST API для возможной будущей интеграции в другие сервисы.

2.1.3.2. Требования к надежности

Система должна сохранять работоспособность и обеспечивать восстановление своих функций при возникновении следующих внештатных ситуаций:

– при сбоях в системе электроснабжения аппаратной части, приводящих к перезагрузке ОС, восстановление программы должно происходить после перезапуска ОС и запуска исполняемого файла системы;

– при ошибках в работе аппаратных средств (кроме носителей данных и программ) восстановление функции системы возлагается на ОС;

– при ошибках, связанных с программным обеспечением (ОС и драйверы устройств), восстановление работоспособности возлагается на ОС. Для защиты аппаратуры от бросков напряжения и коммутационных помех должны применяться сетевые фильтры.

2.2. Архитектура и принципы работы приложения

Для разработки данного веб-сервиса будет использована клиент-серверная архитектура. Она заключается в наличии единого вычислительного сервера, являющегося центральным узлом иерархии, который занимается обработкой запросов от клиентов (дочерних узлов).

Общая схема работы представляет собой следующую цепочку событий (см. рисунок 8), которая состоит из 5 основных страниц ресурса; рассмотрим их подробнее:

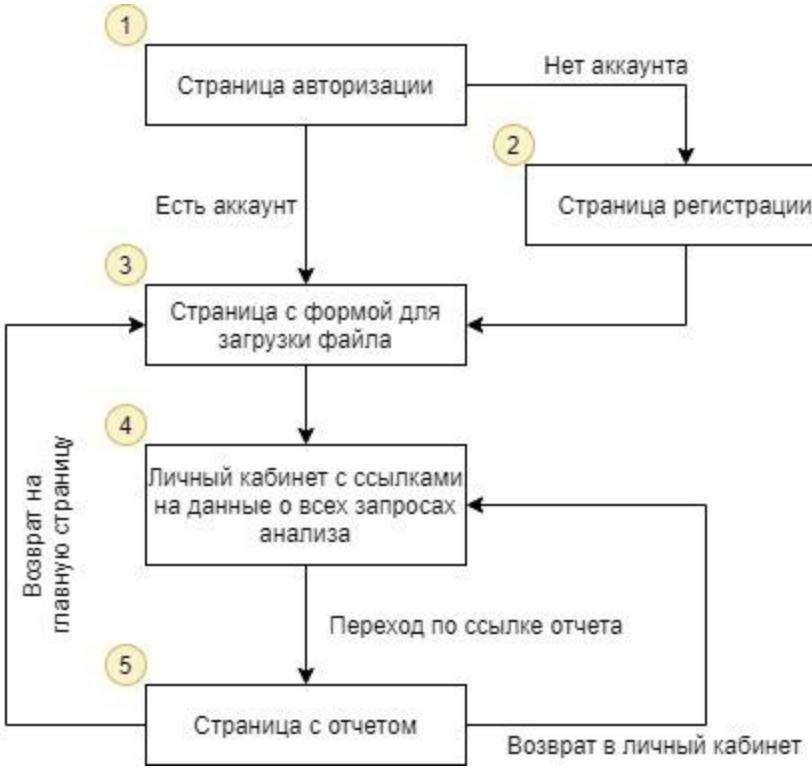

Рисунок 8. Обобщенная схема работы сервиса

Первое что возвращает сервер при новом запросе – страница авторизации (*шаг 1*), она необходима для того, чтобы была возможность однозначно определять пользователей, а также сортировать и возвращать результаты анализа только их кода. В случае если у пользователя еще нет учетной записи на данном портале он переходит на страницу регистрации (*шаг 2*), либо если он уже зарегистрирован, то сразу переходит на страницу с основным функционалом веб-ресурса (*шаг 3*)

Страница регистрации пользователя представляет собой html-форму, в которой новый пользователь может ввести свои персональные данные и после отправки их на сервер, будет выполнена запись в базу данных, для возможности дальнейшей авторизации. После валидации всех данных формы, будет выполнено перенаправления пользователя на страницу с функционалом сайта (*шаг 3*)

Весь функционал веб-сервиса, фактически, представлен одной html страницей, на которой находится форма загрузки файлов, написанных на языке JavaScript. После отправки заполненной формы на сервер, будет выполнена не

только проверка прикрепленного кода, но и ряд операций в базе данных, которые позволят однозначно установить какой пользователь сделал данный запрос. Подобные меры позволят не только удобно отображать множество запросов на множество пользователей, но и обеспечит конфиденциальность проверенному программному коду, так как доступ к файлу и отчету будет только у пользователя, сделавшего запрос. После окончания результатов проверки, пользователя перенаправляют в его личный кабинет, где он может увидеть все свои предыдущие результаты проверки (*шаг 4*), а также перейти непосредственно к каждой из них (*шаг 5*)

Последний шаг представляет собой автоматически сгенерированный библиотекой «ESLint» html отчет, в котором подробно изложены какие правила написания кода были нарушены, а также представлен результат анализа на наличие уязвимостей.

2.3. База данных

В данном веб-сервисе можно обойтись без сложной структуры базы данных, по причине того, что существует всего 2 сущности: Пользователь и Отчет, рассмотрим их подробнее:

У каждой сущности пользователя помимо атрибутов, которые пользователь введет на странице регистрации, существует уникальный ID, который позволяет однозначно установить все персональные данные, он является первичным ключом (PK) у данной сущности.

У сущности, представляющей отчет, помимо уникального ID являющегося первичным ключом, и полей, предоставляющих информацию о анализируемом файле, файле отчета, а также времени и дате проверки, существует поле, которое является внешним ключом (FK), ссылающееся на пользователя, который сделал

запрос на проведения анализа. Причем связь определена как многие-к-одному, что говорит о том, что у каждого отчета может быть лишь один пользователь, но у одного пользователя может быть много отчетов [25].

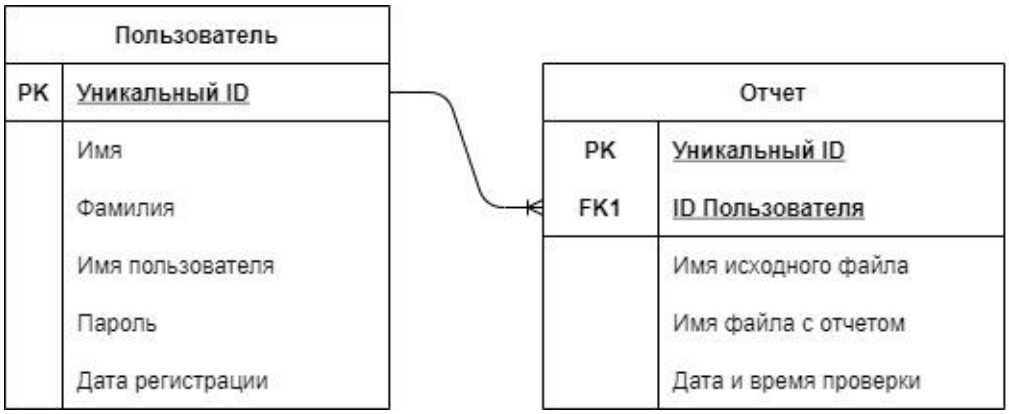

Рисунок 9. Модель базы данных

2.4. Пользовательский интерфейс

Для полноценного функционирования нам понадобится создать прототип всего 5 страниц.

Для начала создадим прототип страницы Авторизации: он должен включать себя форму для ввода имени пользователя и пароля.

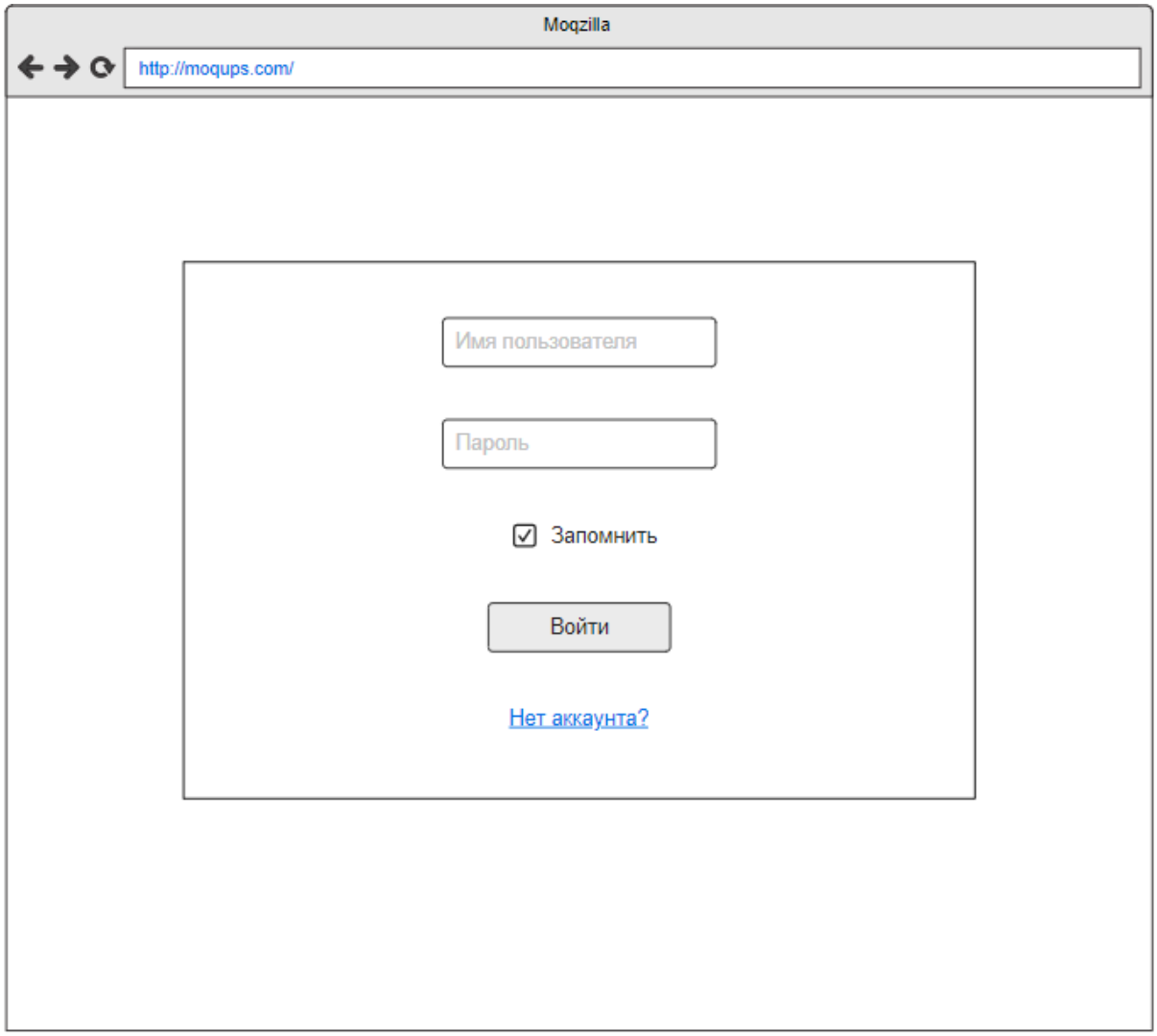

Рисунок 10. Макет страницы авторизации

Затем создадим страницу регистрации, на ней должна быть возможность ввести данные, для большей персонализации пользователя, и обратиться к серверу для отправки введённых данных.

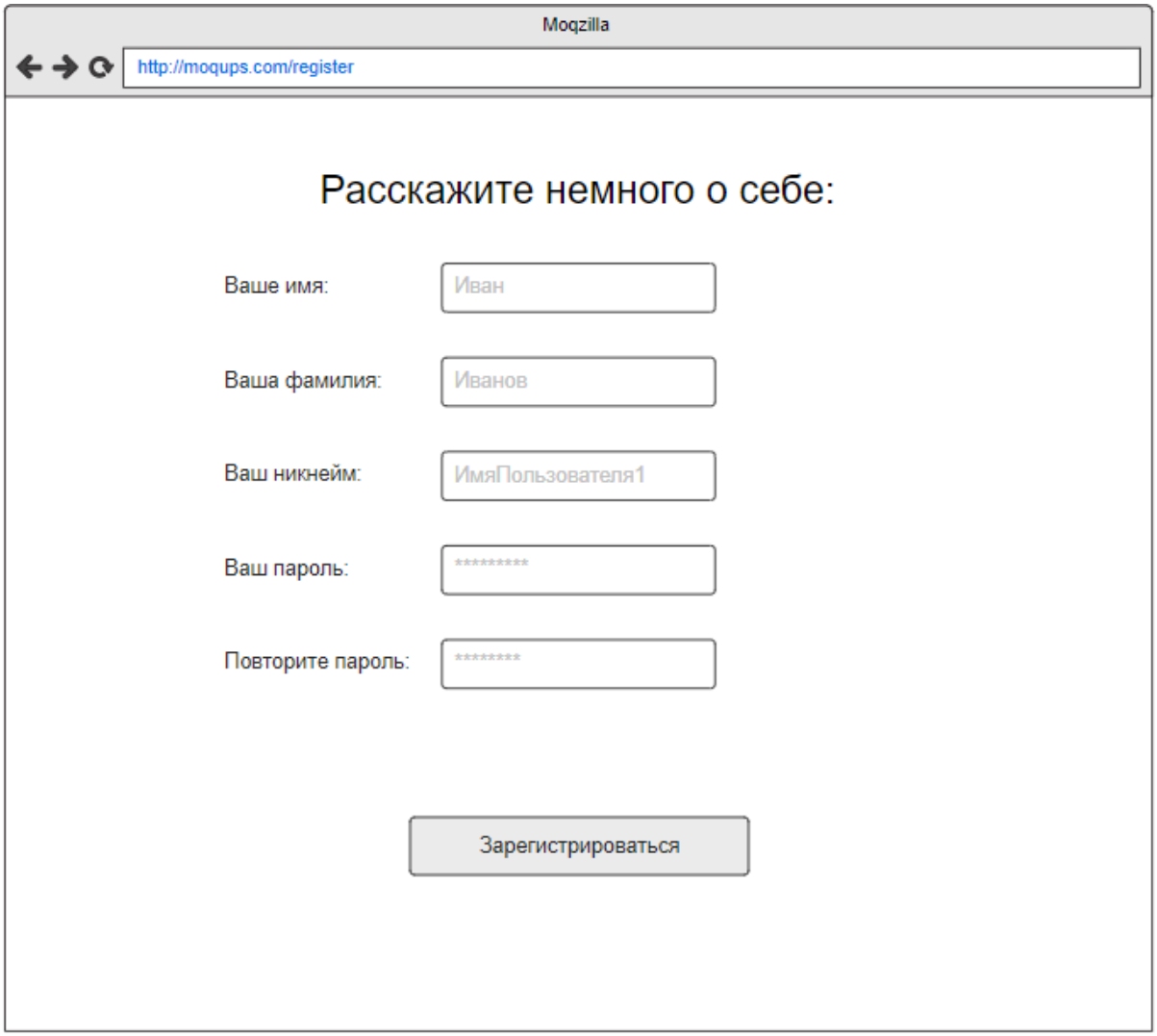

Рисунок 11. Макет страницы регистрации

Также необходимо сделать прототип основной страницы, через которую и будет происходить все взаимодействие с сервером и вызываться инструмент статического анализа. В целом можно обойтись и простой формой, для отправки файла, но стоит предусмотреть и возможность ввода текста непосредственно через форму.

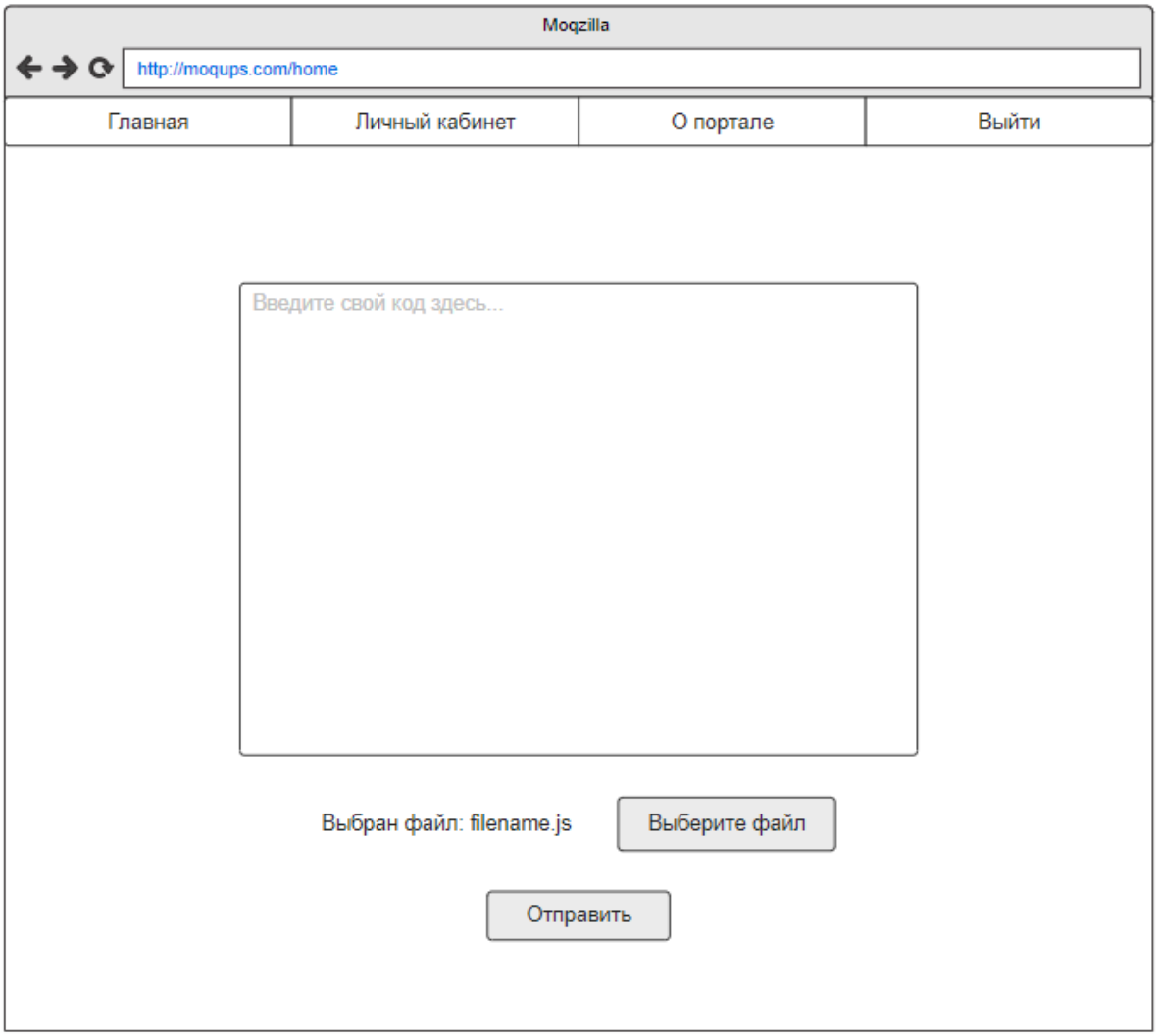

Рисунок 12 Макет страницы с функционалом

Последним прототипом необходимым для функционирования веб-сервиса, будет макет личного кабинета, который и осуществляет доступ к результатам работы анализатора.

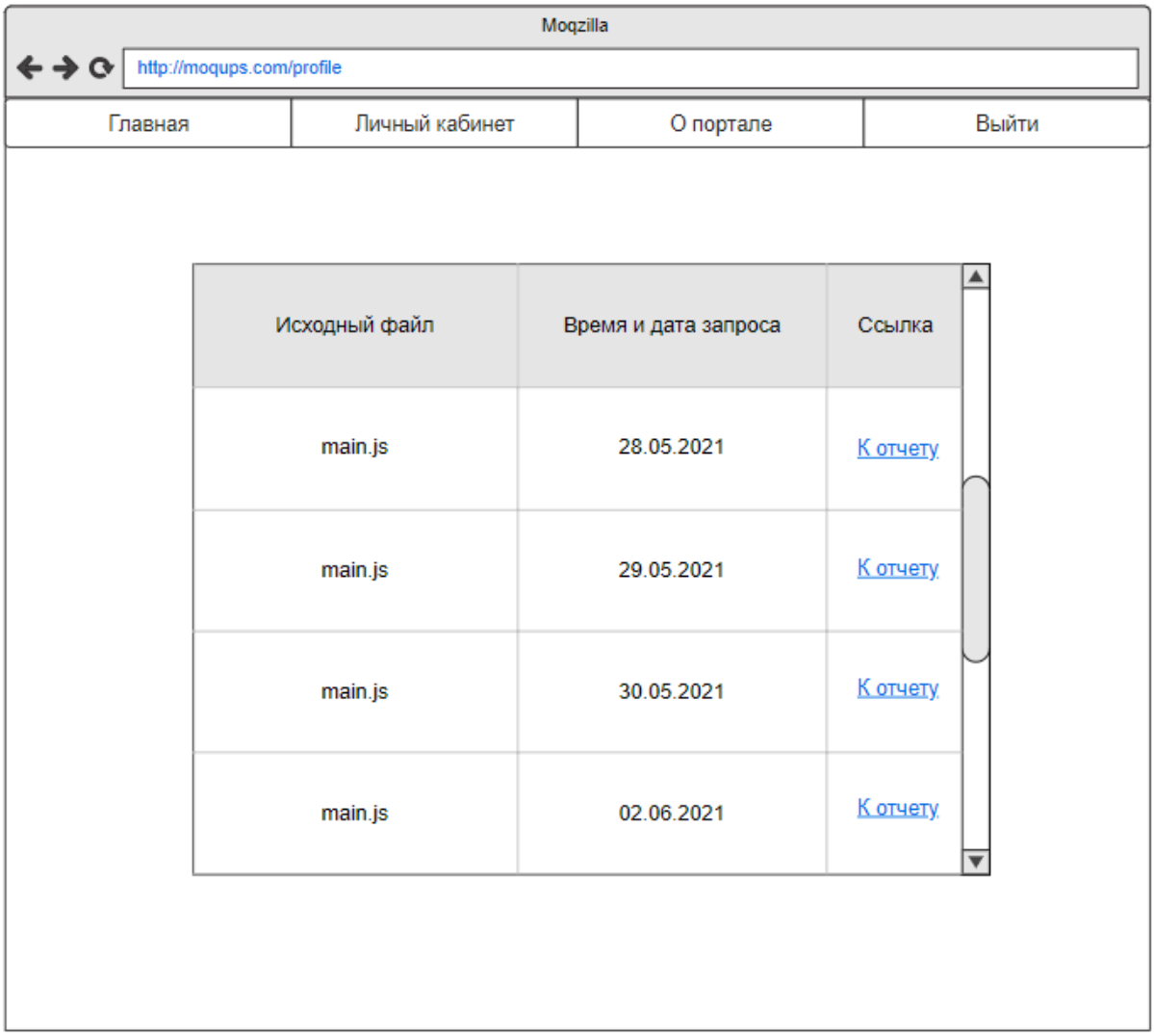

Рисунок 13. Макет личного кабинета пользователя

Прототипировать интерфейс отчета не является необходимым, так как встроенный в анализатор шаблон и так удовлетворяет всем требованиям.

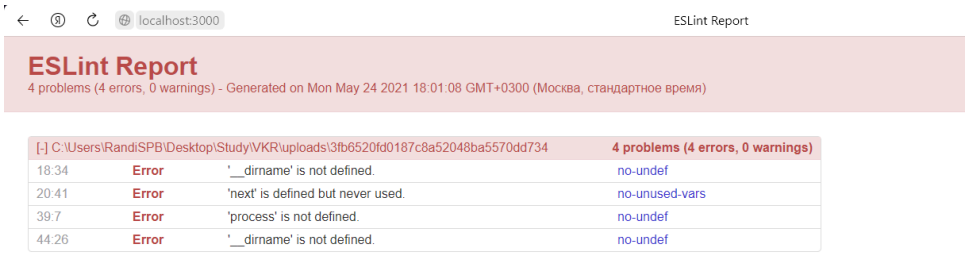

Рисунок 14. Пример результата работы анализатора

Выводы по главе 2

При разработке данного веб-сервиса были выполнены следующие шаги:

1. Разработано техническое задание
2. Определена архитектура данного сервиса
3. Описаны принципы работы ресурса с пользователем
4. Спроектирована база данных
5. Осуществлено прототипирование пользовательского интерфейса
6. Реализован веб-сервер при помощи “Express.JS” и имплементирована часть функционала указанного в ТЗ

ЗАКЛЮЧЕНИЕ

При работе над данной ВКР автором был разработан веб-сервис предоставляющий доступ к инструментам статического анализа. В процессе выполнения были выполнены следующие задачи:

1. Изучены средства для статического и динамического анализа и выбрано наиболее подходящее и актуальное на данный момент решение.
2. Разработана архитектуры веб-сервиса с учетом возможного масштабирования.
3. Спроектирована база данных для отображения связей всех участвующих в процессе сущностей.
4. Создано техническое задание для упрощения разработки сервиса.
5. Разработана серверная часть для имплементации функционала сервиса.
6. Создан прототип пользовательского интерфейса

Таким образом цель данной ВКР было достигнута.

Дальнейшее развитие данного сервиса подразумевает интеграцию других сервисов (VK.com, Facebook и другие) для более удобной авторизации, проектирование REST API для возможности обращения через HTTP-запросы, а также улучшение дизайна и интерфейса для более удобного взаимодействия пользователя с сервером.

ГЛОССАРИЙ

ECMAScript — это встраиваемый расширяемый не имеющий средств ввода-вывода язык программирования, используемый в качестве основы для построения других скриптовых языков. [10]

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

IDE — (англ. Integrated development environment — IDE), также единая среда разработки, ECP — комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО). Среда разработки включает в себя: текстовый редактор, Транслятор (компилятор и/или интерпретатор), средства автоматизации сборки, отладчик. [16]

Node.js — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения [27].

Visual Studio Code — редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений.

Байт-код — стандартное промежуточное представление, в которое может быть переведена компьютерная программа автоматическими средствами. По сравнению с исходным кодом, удобным для создания и чтения человеком, байт-код — это компактное представление программы, уже прошедшей синтаксический и семантический анализ. В нём в явном виде закодированы типы, области видимости и другие конструкции. С технической точки зрения байт-код

представляет собой машинно-независимый код низкого уровня, генерируемый транслятором из исходного кода.[9, с.272]

В разработке ПО, CI/CD или CICD [5] — это комбинация непрерывной интеграции (continuous integration) и непрерывного развертывания (continuous delivery или continuous deployment) программного обеспечения в процессе разработки [3][2]. CI/CD объединяет разработку, развёртывание и команду, ускоряя процесс сборки, тестирования и развёртывания приложения.

Динамический анализ кода — анализ программного обеспечения, производящийся при помощи выполнения программ на реальном или виртуальном процессоре (в отличие от статического анализа). Утилиты динамического анализа могут требовать загрузки специальных библиотек, перекомпиляцию программного кода. Некоторые утилиты могут инструментировать исполняемый код в процессе исполнения или перед ним. Для большей эффективности динамического анализа требуется подача тестируемой программе достаточного количества входных данных, чтобы получить более полное покрытие кода. Также следует позаботиться о минимизации воздействия инструментов на исполнение тестируемой программы (включая временные характеристики). [14]

Иску́сственный интелле́кт (ИИ; англ. artificial intelligence, AI) — свойство интеллектуальных систем выполнять творческие функции, которые традиционно считаются прерогативой человека (не следует путать с искусственным сознанием, ИС); наука и технология создания интеллектуальных машин, особенно интеллектуальных компьютерных программ. [19]

Исходный код — текст компьютерной программы на каком-либо языке программирования или языке разметки, который может быть прочтён человеком. В обобщённом смысле — любые входные данные для транслятора. Исходный код транслируется в исполняемый код целиком до запуска программы при помощи компилятора или может исполняться сразу при помощи интерпретатора. [18]

Качество программного обеспечения (Software Quality) — весь объём признаков и характеристик программ, который относится к их способности удовлетворять установленным или предполагаемым потребностям.[13]

Машинное обучение (англ. machine learning, ML) — класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение за счёт применения решений множества сходных задач. Для построения таких методов используются средства математической статистики, численных методов, математического анализа, методов оптимизации, теории вероятностей, теории графов, различные техники работы с данными в цифровой форме.[29]

Непрерывная интеграция (CI, англ. Continuous Integration) — практика разработки программного обеспечения, которая заключается в постоянном слиянии рабочих копий в общую основную ветвь разработки (до нескольких раз в день) и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем.[1]

Непрерывное развертывание (HP) – это расширение непрерывной доставки, которое автоматически развертывает каждую сборку, которая проходит полный цикл тестирования.

Плаги́н — независимо компилируемый программный модуль, динамически подключаемый к основной программе и предназначенный для расширения и/или использования её возможностей. Плагины обычно выполняются в виде библиотек общего пользования. [24]

Статический анализ кода — анализ программного обеспечения, производимый (в отличие от динамического анализа) без реального выполнения исследуемых программ. В большинстве случаев анализ производится над какой-либо версией исходного кода.

Тестирование программного обеспечения — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом.[11]

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Alessandro Del Sole // Visual Studio Code Succinctly. // SyncFusion Inc., 2016. - 128 с.
2. Continuous integration vs. continuous delivery vs. continuous deployment // atlassian.com URL: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment (дата обращения: 22.05.2021).
3. Heller, Martin. Continuous integration is not always the right answer. Here's why., TechBeacon (20 апреля 2021).
4. Node.js API // eslint.org URL: https://eslint.org/docs/developer-guide/nodejs-api (дата обращения: 05.04.2021).
5. 5 common pitfalls of CI/CD—and how to avoid them // infoworld.com URL: https://www.infoworld.com/article/3113680/5-common-pitfalls-of-cicd-and-how-to-avoid-them.html (дата обращения: 28.04.2021).
6. ISO/IEC TR 19759:2005 // https://www.iso.org/standard/33897.html (дата обращения: 02.04.2020).
7. ML в помощь: инструменты для разработчика с использованием ИИ // https://habr.com/ru/company/plesk/blog/490280/ (дата обращения: 06.04.2020).
8. Software Quality Objectives for Source Code // https://web.archive.org/web/20120312224540 URL: https://web.archive.org/web/20120312224540/http://www.erts2010.org/Site/0ANDGY78/Fichier/PAPIERS%20ERTS%202010/ERTS2010_0035_final.pdf (дата обращения: 02.04.2020).
9. Terence Parr Language Implementation Patterns — Pragmatic Bookshelf. December 2009. - 370 с. - ISBN 978-1-934356-45-6

10. Zakas N. ECMAScript // Professional JavaScript for Web Developers. — 2nd ed. — USA, Canada: Wiley Publishing, Inc, 2009. — P. 3—7. — ISBN 978-0-470-22780-0.
11. ГОСТ Р 56920-2016 СИСТЕМНАЯ И ПРОГРАММНАЯ ИНЖЕНЕРИЯ. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. ЧАСТЬ 1. ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ // protect.gost.ru URL: http://protect.gost.ru/v.aspx?control=8&id=195502 (дата обращения: 05.04.2021).
12. ГОСТ Р ИСО/МЭК 15271-02 Процессы жизненного цикла программных средств // docplace.ru URL: https://docplace.ru/gostr34/gostriso1527102/ (дата обращения: 05.04.2021).
13. ГОСТ Р ИСО/МЭК 9126-93 // docs.cntd.ru URL: https://docs.cntd.ru/document/1200009076 (дата обращения: 25.05.2021).
14. Динамический анализ кода // pvs-studio.com URL: https://pvs-studio.com/ru/blog/terms/0070/ (дата обращения: 25.05.2021).
15. Дуглас Крокфорд. Прил. B. JSLint // JavaScript. Сильные стороны = JavaScript: The Good Parts. — Питер, 2012. — С. 141—152. — 174 с. — (Бестселлеры O'Reilly). — ISBN 978-5-459-01263-7.
16. Интегрированная среда разработки // ru.wikipedia.org URL: https://ru.wikipedia.org/wiki/Интегрированная_среда_разработки (дата обращения: 26.05.2021).
17. Использование машинного обучения в статическом анализе исходного кода программ // habr.com URL: https://habr.com/ru/company/pvs-studio/blog/484208/ (дата обращения: 25.05.2021).
18. Исходный код программы // life-prog.ru URL: https://life-prog.ru/view_zam2.php?id=225&cat=5&page=17 (дата обращения: 25.05.2021).

19. Компьютер учится и рассуждает (ч. 1) // Компьютер обретает разум = Artificial Intelligence Computer Images / под ред. В. Л. Стефанюка. — Москва: Мир, 1990. — 240 с. — ISBN 5-03-001277-X (рус.); ISBN 0705409155 (англ.).
20. Линтер — это чистый код для ленивых // thecode.media URL: https://thecode.media/linter/ (дата обращения: 25.05.2021).
21. Макконнелл С. Совершенный код. / Пер. с англ. — М. : Издательство «Русская редакция», 2010. — 896 стр. ISBN 978-5750200641
22. О статическом анализе кода в теории и на практике // info-comp.ru URL: https://info-comp.ru/static-code-analysis (дата обращения: 25.05.2021).
23. О статическом анализе начистоту // habr.com URL: https://habr.com/ru/company/solarsecurity/blog/439286/ (дата обращения: 25.05.2021).
24. ОРФОГРАФИЧЕСКИЙ АКАДЕМИЧЕСКИЙ РЕСУРС "АКАДЕМОС" // orfo.ruslang.ru (дата обращения: 18.04.2021)
25. Руководство по проектированию реляционных баз данных (7-9 часть из 15) [перевод] // habr.com URL: https://habr.com/ru/post/193380/ (дата обращения: 25.05.2021).
26. Синицын С. В., Налютин Н. Ю. Верификация программного обеспечения. — М.: БИНОМ, 2008. — 368 с. — ISBN 978-5-94774-825-3.
27. Сухов К. К. Node.js. Путеводитель по технологии. — «ДМК», 2015. — С. 416. — ISBN 978-5-97060-164-8.
28. Тестирование. Фундаментальная теория // qa-guide.ru URL: https://qa-guide.ru/forums/topic/teorija_testirovanija/ (дата обращения: 25.05.2021).
29. Флах П. Машинное обучение. — М.: ДМК Пресс, 2015. — 400 с. — ISBN 978-5-97060-273-7.
30. Фундаментальная теория тестирования // habr.com URL: https://habr.com/ru/post/549054/ (дата обращения: 07.04.2021).

31. Холиварный рассказ про линтеры // habr.com URL: https://habr.com/ru/company/oleg-bunin/blog/433480/ (дата обращения: 08.05.2021).

ПРИЛОЖЕНИЯ