Geekbrains

**Разработка веб-приложения «VMAS» по учету проведения технического обслуживания транспортных средств и контроля исполнения организационных работ для ГБУ МО «Автохозяйство»**

Программа: Разработчик

Специализация: Программист

Зуев Максим Михайлович

Москва

2025

**Содержание**

[Введение 3](#_Toc194596619)

[Глава 1**.** Анализ технологий и инструментов создания веб приложения. 5](#_Toc194596620)

[1.1Технологиивеб-разработки. 5](#_Toc194596621)

[1.2 Техническое задание на разработку веб-приложения. 11](#_Toc194596622)

[Глава 2. Разработка веб-приложения. 19](#_Toc194596623)

[2.1 Дизайн макет веб-приложения. 19](#_Toc194596624)

[2.2 Разработка веб-приложения. 26](#_Toc194596625)

[2.3 Запуск и тестирование веб-приложения. 34](#_Toc194596626)

[Заключение 43](#_Toc194596627)

[Список используемой литературы 44](#_Toc194596628)

[Приложение 45](#_Toc194596629)

**Введение**

Сегодня сложно представить динамичную, быстро развивающуюся компанию без использования IT – продуктов, так как информационные технологии (ИТ) участвуют практически во всех сферах бизнеса, от оптимизации процессов до прогнозирования.

Основной задачей информатизации компании является повышение эффективности управления ею, а также повышение производительности Труда и снижение издержек, увеличение капитализации компании и повышение конкурентоспособности продукции и услуг.

И одним из ключевых, направлений информатизации компании является веб-приложения. В настоящее время использование веб-приложений вышло на новый уровень и не умолило продолжает развиваться. Основными преимуществами веб-приложений стали: интерактивность, доступность, масштабируемость, обновляемость, безопасность.

Также стоит отметить появление и использование более продвинутого прогрессивного веб-приложения (англ. progressive web app, PWA) — технология в веб-разработке, которая визуально и функционально трансформирует сайт в приложение (мобильное приложение в браузере). PWA является гибридным решением и позволяет открыть приложение с помощью мобильного браузера. При этом полностью сохраняется функциональность нативного приложения.

Основной целью данной дипломной работы является (Разработка веб-приложения «VMAS» по учету проведения технического обслуживания транспортных средств и контроля исполнения организационных работ для ГБУ МО «Автохозяйство»). «VMAS» это сокращенная аббревиатура «Vehicle maintenance accounting system»- дословно(Система учета технического обслуживания транспортных средств).

Использование веб-приложения поможет компании решить несколько проблем:

* Предоставление доступа к информации – веб-приложение позволит пользователям получать необходимую информацию и функционал с любого устройства, из любой точки мира;
* Автоматизация процессов – веб-приложение позволит автоматизировать большинство процессов, решать задачи быстрее и эффективнее;
* Безопасность – данные централизованно хранятся на сервере, что минимизирует риски их потери;

Для достижения данной цели был поставлен ряд теоретических и практических задач:

* Проанализировать технологии реализации веб-приложений;
* Составить техническое задание на разработку с учетом пожеланий компании и поставленных задач;
* Создать дизайн-макет веб-приложения;
* Выполнить разработку веб-приложения и произвести его тестирование;

Таким образом, данная работа состоит из двух глав теоретической и практической.

* Теоретико-методологическую базу проекта составляют учебные пособия,

справочники, ресурсы в сети Интернет.

* Практическим результатом настоящей работы является готовое веб-приложение с полностью рабочим функционалом.

# 

# **Глава 1. Анализ технологий и инструментов создания веб приложения.**

В современном мире веб-приложения стали неотъемлемой частью компаний. Существует множество средств для создания веб-приложений, каждое из которых обладает своими особенностями, преимуществами и недостатками. Этот сравнительный анализ посвящен изучению различных сред создания веб-приложений с целью выявления оптимального инструмента для успешной разработки проекта.

**1.1 Технологии веб-разработки.**

В современном мире веб-разработка играет ключевую роль в бизнесе, образовании, развлечениях и многих других сферах. Веб-приложения стали неотъемлемой частью нашей повседневной жизни, предоставляя пользователям доступ к информации, услугам и продуктам. Компании всех размеров инвестируют в веб-разработку, чтобы улучшить взаимодействие с клиентами, повысить эффективность работы и оставаться конкурентоспособными.

Для создания современных веб-приложений используются различные технологии.

* HTML (HyperText Markup Language): Основной язык разметки для создания веб-страниц. Он определяет структуру и содержание веб-документов. HTML позволяет создавать заголовки, абзацы, списки, ссылки, изображения и другие элементы, которые составляют веб-страницу.
* CSS (Cascading Style Sheets): Язык стилей, используемый для описания внешнего вида веб-страниц, таких как цвета, шрифты и макеты. CSS позволяет отделить содержание веб-страницы от ее оформления, что упрощает разработку.
* JavaScript: Язык программирования, который позволяет добавлять интерактивные элементы на веб-страницы, такие как анимации, формы и динамическое обновление контента. JavaScript также используется для создания сложных веб-приложений, работающих в браузере.
* Базы данных: Системы управления базами данных (СУБД), такие как MySQL, PostgreSQL и MongoDB, используются для хранения и управления данными веб-приложений. Базы данных позволяют сохранять информацию о пользователях, продуктах, заказах и других данных, необходимых для работы веб-приложения.
* Серверные языки программирования: Языки, такие как PHP, Python, Ruby и Node.js, используются для написания серверной логики веб-приложений. Серверные языки позволяют обрабатывать запросы пользователей, взаимодействовать с базами данных и выполнять другие задачи на сервере.

Кроме основных технологий, существуют также другие инструменты и Фреймворки, которые помогают разработчикам создавать более сложные и функциональные веб-приложения. Например, TypeScript — это надстройка над JavaScript, которая добавляет статическую типизацию и улучшает качество кода. SASS и LESS — это препроцессоры CSS, которые упрощают написание стилей и добавляют возможности, отсутствующие в стандартном CSS.

У большинства современных веб-приложений есть два слоя: фронтенд для пользователей и бэкенд для технических действий. С фронтендом взаимодействуют обычные клиенты, а бэкенд находится под капотом и обрабатывает запросы от фронтенда.

**Frontend-разработка**

Фронтенд-разработка занимается созданием пользовательского интерфейса (UI) и взаимодействием пользователя с веб-приложением. Основные технологии фронтенд-разработки включают:

* HTML: Определяет структуру веб-страницы. HTML-теги используются для создания заголовков, абзацев, списков, изображений и других элементов, которые составляют веб-страницу.
* CSS: Определяет стиль и оформление веб-страницы. CSS позволяет задавать цвета, шрифты, размеры, отступы и другие параметры, которые определяют внешний вид веб-страницы.
* JavaScript: Добавляет интерактивность и динамическое поведение на веб-страницу и позволяет создавать анимации, обрабатывать события, взаимодействовать с сервером и выполнять другие задачи, которые делают веб-страницу более интерактивной.

Фреймворки JavaScript — это наборы готовых решений и шаблонов, которые предоставляют разработчику структуру для создания приложения. Фреймворки устанавливают общую структуру и порядок работы приложения, определяя, каким образом должен быть организован код.

Фреймворки бывают двух видов:

* Отвечающие за интерфейс и внешний вид (frontend).
* Отвечающие за внутреннюю логику, обработку серверных запросов и обращение к базам данных (Backend).

Использование Фреймворков помогает ускорить процесс разработки, упростить написание и поддержку кода, а также обеспечить стабильность и предсказуемость результатов.

Несколько самых лучших Фреймворков по версии Google:

* React: Библиотека JavaScript для создания пользовательских интерфейсов. Она позволяет разрабатывать компоненты, которые можно повторно использовать. React использует виртуальный DOM, что делает обновление интерфейса более эффективным.
* Angular: Фреймворк JavaScript для создания динамических веб-приложений. Он предоставляет мощные инструменты для работы с данными и маршрутизацией. Angular использует архитектуру MVC (Model-View-Controller), что упрощает организацию кода и разделение логики приложения.
* Vue.js: Прогрессивный Фреймворк JavaScript, который упрощает создание пользовательских интерфейсов и одностраничных приложений. Vue.js легко интегрируется с другими библиотеками и проектами, что делает его гибким и удобным для использования.

Фронтенд-разработчики также используют различные инструменты для повышения эффективности работы и улучшения качества кода. Например, Webpack — это модульный сборщик, который позволяет объединять и оптимизировать файлы JavaScript, CSS и других ресурсов. Babel — это транспайлер, который позволяет использовать современные возможности JavaScript в старых браузерах.

**Backend разработка**

Бэкенд-разработка занимается серверной частью веб-приложений, включая работу с базами данных, серверной логикой и API. Основные технологии бэкенд-разработки включают:

* Серверные языки программирования: PHP, Python, Ruby, Node.js и другие. Эти языки используются для написания серверной логики, обработки запросов пользователей, взаимодействия с базами данных и выполнения других задач на сервере.
* Базы данных: MySQL, PostgreSQL, MongoDB и другие. Базы данных позволяют сохранять и управлять данными веб-приложений, такими как информация о пользователях, продуктах, заказах и других данных.
* API: Интерфейсы для взаимодействия между различными частями веб-приложения и внешними сервисами. API позволяют обмениваться данными между клиентом и сервером, а также интегрировать веб-приложение с другими сервисами и платформами.

Фреймворки бывают двух видов:

* Отвечающие за интерфейс и внешний вид (frontend).
* Отвечающие за внутреннюю логику, обработку серверных запросов и обращение к базам данных (Backend).

Использование Фреймворков помогает ускорить процесс разработки, упростить написание и поддержку кода, а также обеспечить стабильность и предсказуемость результатов.

Несколько самых лучших Фреймворков по версии Google:

* Express.js: Минималистичный Фреймворк для Node.js, который упрощает создание серверных приложений и API. Express.js предоставляет инструменты для маршрутизации, обработки запросов и управления сессиями.
* Django: Высокоуровневый Фреймворк для Python, который предоставляет множество инструментов для разработки веб-приложений. Django включает в себя ORM (Object-Relational Mapping), систему аутентификации, админ-панель и другие компоненты, которые упрощают разработку веб-приложений.
* Ruby on Rails: Фреймворк для Ruby, который упрощает создание веб-приложений благодаря использованию соглашений и готовых решений. Ruby on Rails включает в себя инструменты для работы с базами данных, маршрутизацией, аутентификацией и другими задачами.

Бэкенд-разработчики также используют различные инструменты для повышения эффективности работы и улучшения качества кода. Например, Docker — это платформа для контейнеризации, которая позволяет создавать изолированные среды для разработки и развертывания приложений. Postman — это инструмент для тестирования API, который позволяет отправлять запросы и анализировать ответы.

**Full-stack разработка**

Фулл-стек разработка объединяет как фронтенд, так и бэкенд-разработку. Фулл-стек разработчики обладают знаниями и навыками в обеих областях, что позволяет им разрабатывать полноценные веб-приложения от начала до конца.

* MEAN Stack: Набор технологий, включающий MongoDB, Express.js, Angular и Node.js. Он позволяет разрабатывать веб-приложения на JavaScript от базы данных до пользовательского интерфейса. MEAN Stack обеспечивает высокую производительность и масштабируемость веб-приложений.
* MERN Stack: Похож на MEAN Stack, но вместо Angular используется React. MERN Stack сочетает в себе преимущества React и Node.js, что делает его популярным выбором для разработки современных веб-приложений.
* LAMP Stack: Набор технологий, включающий Linux, Apache, MySQL и PHP. Это классический стек для разработки веб-приложений, который обеспечивает стабильность и надежность. LAMP Stack широко используется для создания корпоративных веб-сайтов и приложений.

Фулл-стек разработчики обладают широким спектром знаний и навыков, что позволяет им эффективно работать над всеми аспектами веб-приложения. Они могут быстро переключаться между фронтенд и Бэкенд задачами, что ускоряет процесс разработки и улучшает качество конечного продукта. Фулл-стек разработчики также могут лучше понимать и решать проблемы, возникающие на стыке фронтенд и Бэкенд технологий.

Веб-разработка — это широкая и динамичная область, которая требует знаний и навыков в различных технологиях. Понимание основ фронтенд и бэкенд-разработки, а также фулл-стек подхода, поможет вам начать свой путь в этой увлекательной сфере. Независимо от того, хотите ли вы стать фронтенд, Бэкенд или фулл-стек разработчиком, важно постоянно учиться и следить за новыми тенденциями и технологиями в веб-разработке.

## 1.2 Техническое задание на разработку веб-приложения.

Ориентировочная структура ТЗ на создание сайта / веб-приложения.

**Аннотация**

В данном техническом задании сформулированы основные функциональные и технические требования к разрабатываемому сайту/веб-приложению.

Документ является основным источником требований для реализации проекта.

1. Цели создания веб-приложения.

* Разработка веб-приложения «VMAS» по учету проведения технического обслуживания транспортных средств и контроля исполнения организационных работ для ГБУ МО «Автохозяйство»

2. Целевая аудитория.

* Сотрудники компании.

3. Требования к веб-приложению.

3.1 Общие требования.

* Размещение на хостинге Beget;
* Закрепление за веб-приложением доменного имени «www.qmox.ru»;
* Установка последний версии сертификата SSL;
* Использование СУБД;
* Разработанное на Node.JS
* C использованием Фреймворка React последней версии;
* Создание почтового клиента info@qmox.ru;

3.2 Ролевая модель.

|  |  |  |
| --- | --- | --- |
| **Роль** | **Возможности и полномочия** | **Кем создается** |
| Не зарегистрированный пользователь | * Взаимодействие с публичной частью веб-приложения «почта» | Администратором,  Помощником Администратора |
| Зарегистрированный пользователь | * Взаимодействие с публичной частью веб-приложения * Авторизация | Администратором |
| Администратор | * Взаимодействие с панелью управления и СУБД | Администратором |
| Помощник Администратора | * Взаимодействие с панелью управления и СУБД | Администратором |

3.3 Функциональные требования к публичной части (только авторизованные пользователи).

* Просмотр данных.
* Изменение данных.
* Просмотр рабочих задач.
* Изменение рабочих задач.
* Просмотр статусов.
* Изменение статусов.

3.4. Функциональные требования к личному кабинету клиента.

* Создание учетной записи сотрудника – только Администратором или помощником администратора.
* Только вход в свою учетную запись по средству ввода логина и пароля.

3.5. Функциональные требования к панели управления (администрирования).

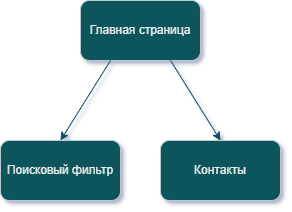
* Доступ только у Администратора или помощника администратора.

3.6. Требования к структуре веб-приложения.

* Просмотр данных.
* Изменение данных.
* Просмотр рабочих задач.
* Изменение рабочих задач.
* Просмотр статусов.
* Изменение статусов.

3.6.1. Структура публичной части веб-приложения

Структура разделов публичной части веб-приложения приведена на рисунке ниже.



|  |  |  |  |
| --- | --- | --- | --- |
| Раздел / подраздел | Содержимое | Функциональные требования | Требования к контенту |
| Фильтрованный поик | Список из СУБД | * Просмотр данных. * Изменение данных. * Просмотр рабочих задач. * Изменение рабочих задач. * Просмотр статусов. * Изменение статусов. | * Заполняется СУБД * Корректируется пользователем * Корректируется администратором * Корректируется помощником администратора |
| Контакты | Контакты | Согласно компании | Выводится в подвале |

3.7 Архитектура веб-приложения.



3.8. Требования к интеграциям

* СУБД

3.9. Технические требования

* Размещение на хостинге Beget;
* Закрепление за веб-приложением доменного имени «www.qmox.ru»;
* Установка последний версии сертификата SSL;
* Использование СУБД;
* Разработанное на Node.JS
* C использованием Фреймворка React последней версии;
* Создание почтового клиента [info@qmox.ru](mailto:info@qmox.ru);

3.10. Требования к дизайну

* Минималистичный стиль
* Логотип веб-приложения
* Логотип компании

3.11. Требования к контенту

* СУБД Заказчика

3.12. Требования к внутренней SEO оптимизации

* Не требуется

3.13. Требования к к системам аналитики

* Не требуется

3.14. Дополнительные требования

* Корректировки после передачи Заказчику

3.14.1. Поисковые подсказки

* Не требуется

3.14.2. Требования к уведомлениям

* Изменение данных.
* Изменение рабочих задач.
* Изменение статусов.
* Уведомление push веб-приложение
* Уведомление Email

3.14.3. Требования к управлению изменениями

Все изменения кода сайта должны внедряться с использованием системы контроля версий — git после предварительного тестирования на тестовом сервере.

4. Требования к документации

Приводится перечень разрабатываемой документации и требований к ней.

5. Стадии и этапы разработки

В данном разделе приведена последовательность этапов реализации проекта (состав этапов зависит от конкретного проекта).

5.1. Прототипирование

На данном этапе необходимо выполнить:

* Создание интуитивно понятного и удобного интерфейса для требуемых типов устройств;
* Проектирование и разработка прототипа;
* Usability-тестирование прототипа.

Артефакты:

* Рабочий прототип, который будет отражать основные функции и возможности сайта.

5.2. Создание дизайна

* Согласно макету Figma

5.3. Верстка и разработка

На данном этапе необходимо выполнить:

* Разработка веб-интерфейса сайта с заданным функционалом согласно макету;
* Разработка серверной части сайта и базы данных;
* Интеграция со смежными системами.

Артефакты:

* Полнофункциональная версия сайта.

5.4. Тестирование

На данном этапе необходимо выполнить:

* Тестирование сайта, исправление выявленных ошибок, оптимизация.

Выполняются следующие виды тестирования:

* Unit тестирование;
* Интеграционное тестирование;
* UAT тестирование;
* Нагрузочное тестирование;
* Тестирование безопасности.

Артефакты:

* Работоспособный сайт(веб-приложение), прошедший испытания и готовый к эксплуатации.

5.5. Документирование

На данном этапе необходимо выполнить:

* Разработку комплекта документации, согласно требованиям данного технического задания.

Артефакты:

* Программная и эксплуатационная документация.

5.6. Обучение

На данном этапе необходимо выполнить:

* Обучения персонала Заказчика работе и администрированию сайта.

Артефакты:

* Обученный персонал Заказчика.

# **Глава 2. Разработка веб-приложения.**

## 2.1 Дизайн макет веб-приложения.

Дизайн-макет – это визуальный макет приложения. Он отображает весь контент в веб-приложении и расположенные элементы интерфейса. Макеты интерфейсов создаются с помощью специального программного обеспечения. Figma – онлайн-сервис для разработки интерфейсов с возможностью организации совместной работы в режиме реального времени.

На этапе проектирования продумывается структура всего приложения, взаимодействие элементов системы. Для построения структуры веб-приложения нужно отталкиваться от основного пользовательского сценария. Согласно техническому заданию, основная цель создания системы: «учет проведения технического обслуживания транспортных средств и контроля исполнения организационных работ».

Шаги для реализации сценария:

* Анализ дизайна веб-приложения в зависимости от сферы деятельности;
* Анализ цветовой палитры веб-приложения;
* Анализ юзабилити веб-приложения;

К дополнительным сценариям относятся:

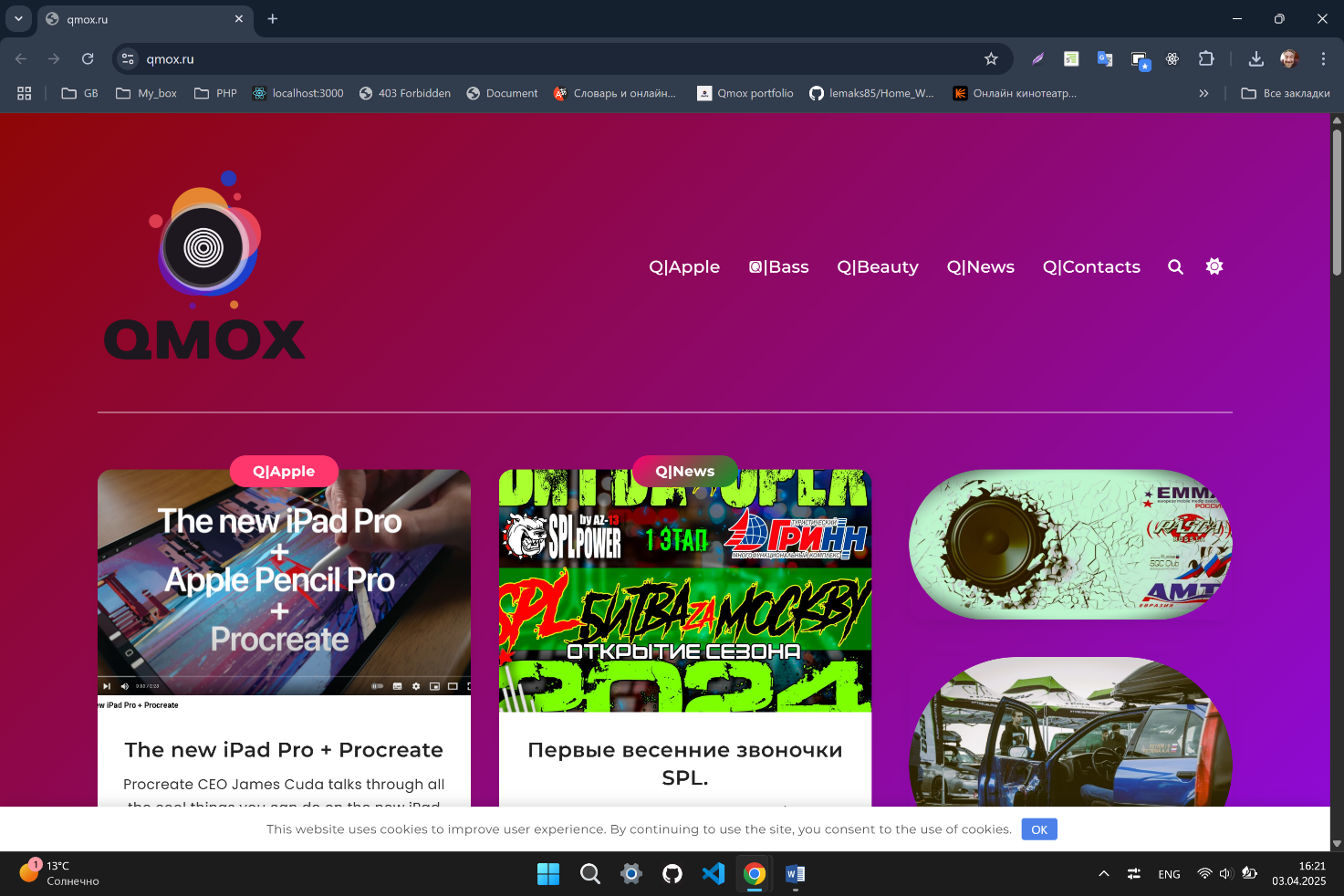
* Форма обратной связи;
* Форма для залогинивания;
* Отображение статусов в веб-приложении;

Мы пользуемся веб-приложениями практически каждый день. Они дают нам доступ к разнообразным сервисам и функциям через интернет. Успех веб-приложения во многом зависит от его дизайна, и важно, чтобы он был не только эстетически привлекательным, но и функциональным, удобным и интуитивно понятным.

Внешне веб-приложение выглядит, как обычный сайт, но между ними есть отличия по многим параметрам. Поэтому при разработке веб-приложений особое внимание уделяют дизайну пользовательского интерфейса и пользовательского опыта (UI/UX).

Чтобы разобраться между этими двумя на первый взгляд очень похожими технологиями, изучим их и сравним.

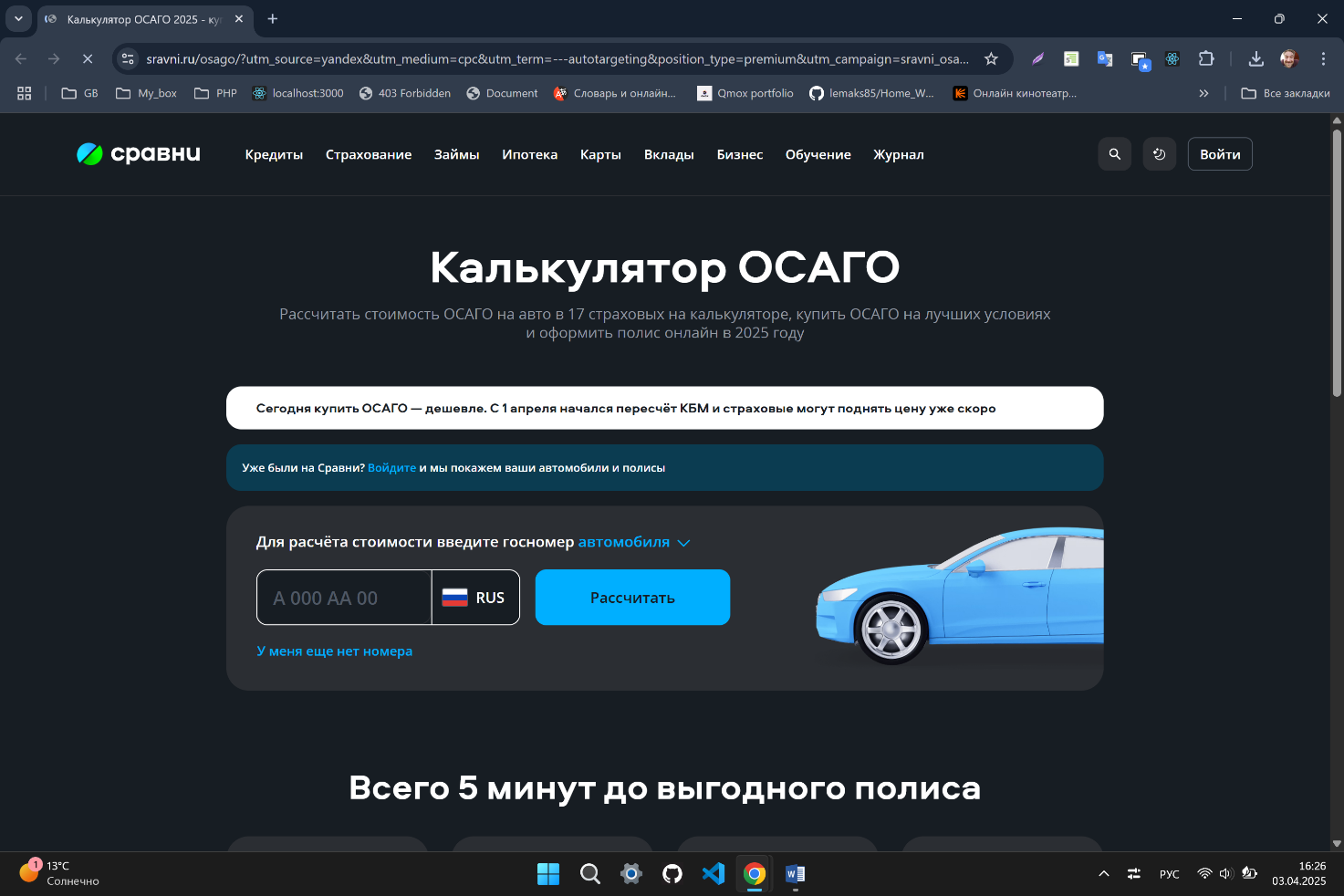
**Веб-сайт** — это набор интернет-страниц, которые открываются в браузере и объединены одной темой, например, посвящены конкретной компании или бренду. На сайте представлено много текстового и визуального контента, он помогает пользователю больше узнать о бизнесе или продукте.



Блог, в котором вы читаете эту статью, расположен на обычном многостраничном ресурсе.

Так на сайте <https://www.qmox.ru> есть страницы услуг, блог, кейсы с описанием различных проектов. Также есть раздел с мероприятиями, статьями о компании в ведущих СМИ и отзывами клиентов.

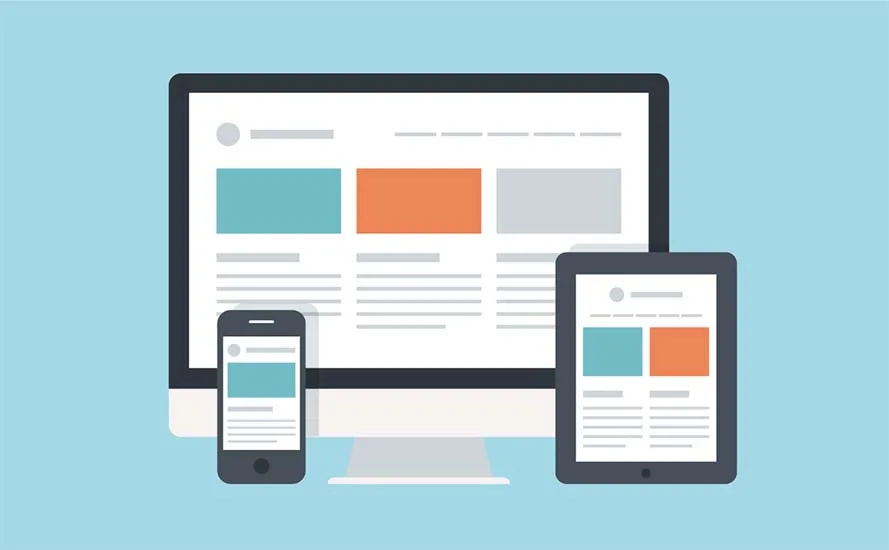
Веб-приложения имеют похожий дизайн с мобильными, но они запускаются в браузере. От обычных сайтов их отличает то, что пользователи могут управлять данными и по-разному взаимодействовать с интерфейсом.



Часто веб-приложения можно загрузить на компьютер прямо с сайта компании. Кроме того, веб-приложения создаются как веб-версии мобильных приложений. Именно так появились мессенджеры, где первой была версия для смартфона.

Примеры веб-приложений:

* CRM-системы — Битрикс24 и другие;
* маркетплейсы и [интернет-магазины](https://webfly.ru/cases/internet_magaziny/) — Вайлдберриз, Ozon, Спортмастер;
* мессенджеры — WhatsApp, Telegram;
* электронная почта — Mail.ru, Gmail, Yandex;
* личные кабинеты — Инвитро, Ростелеком, [любые b2b-порталы](https://webfly.ru/cases/b2b_portaly_kabinety/);
* редакторы для работы с документами: текстом, таблицами, презентациями — экосистема продуктов Google;
* онлайн-сервисы банков — Сбер-Онлайн, интернет-банк Тинькофф;
* агрегаторы объявлений — Авито, Юла;
* соцсети — вКонтакте, YouTube, Нельзяграм.



Всего у продукта может быть 3 версии приложения: для компьютера, для мобильных устройств или веб-версия. Однако у всех этих модификаций будет единая база данных, общий контент, история и другие идентичные функции.

Веб-приложения предназначены для выполнения различных действий: размещения объявлений, покупок, постановки задач, создания документов и прочего. При их дизайне и разработке используются другие подходы.

**Пользовательский опыт (UX)**

UX-дизайн нужен, чтобы обеспечить пользователям положительный опыт. Это достигается благодаря удобной навигации, логичной структуре и интуитивно понятным интерфейсам. Также важно продумать отклик приложения на действия пользователей. Например, добавить визуальную индикацию загрузки, уведомления об успешных или ошибочных действиях и подсказки для дальнейших шагов.

При разработке приложения важно изучить целевую аудиторию. Это позволит учесть потребности и ожидания пользователей на каждом этапе взаимодействия с приложением.

**Интерфейс пользователя (UI)**

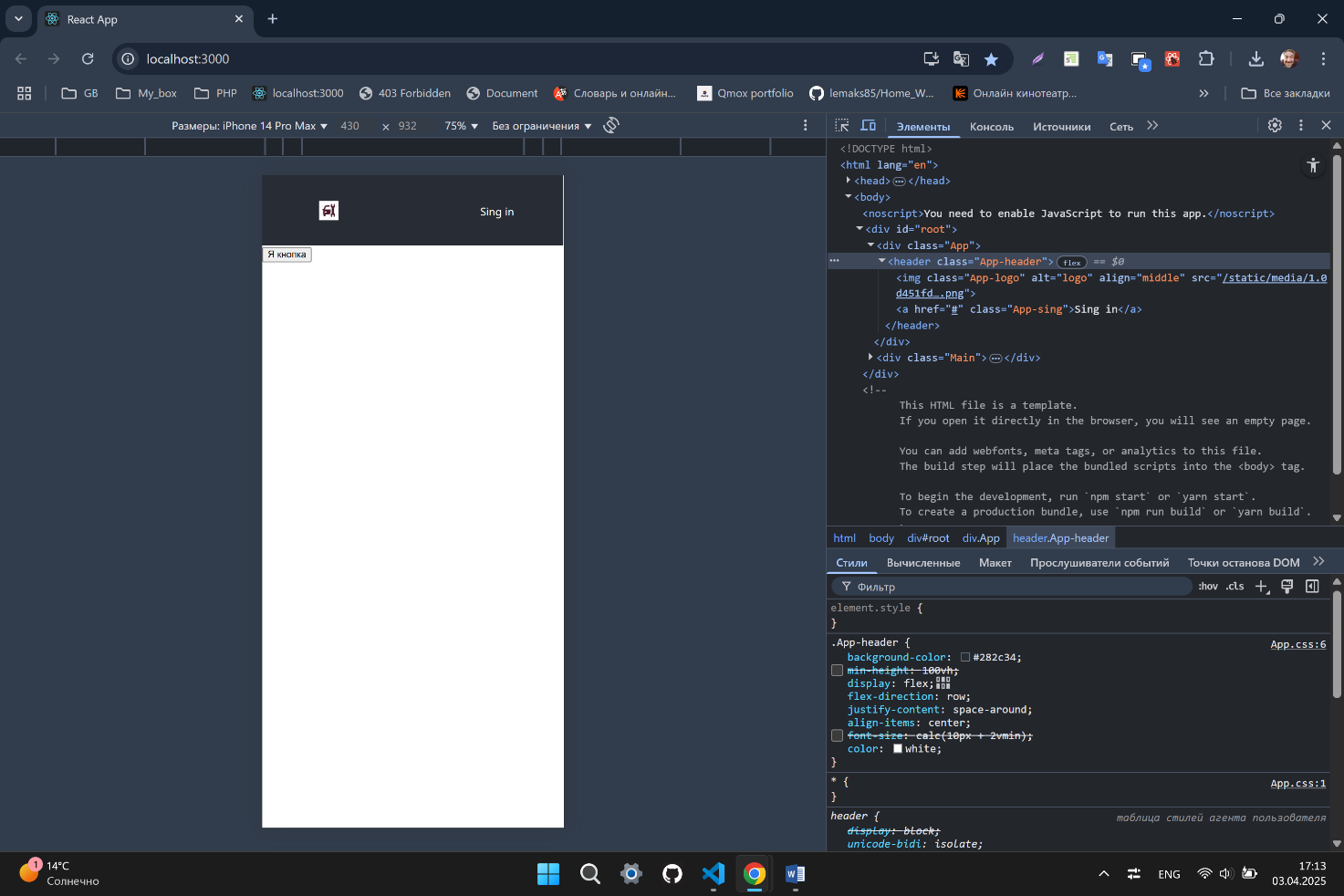
UI-дизайн — это то, как выглядит веб-приложение. Он включает в себя выбор цветовой палитры, шрифтов, иконок и остальных элементов интерфейса. Основная цель UI — создать привлекательный дизайн, который отражает стиль компании и нравится пользователям.

**Адаптивный дизайн**

Веб-приложениями должно быть удобно пользоваться с любых устройств: персональных компьютеров, ноутбуков с экранами разных размеров, планшетов и мобильных телефонов. Для этого применяется адаптивный дизайн, он автоматически подстраивает интерфейс под размер экрана, разрешение устройства и браузер. Это особенно важно для проектов, у которых нет мобильного приложения.

**Скорость загрузки**

Быстрая загрузка страниц и элементов интерфейса имеет решающее значение для удержания пользователей. Если это время превышает 2-3 секунды, то посетитель закроет вкладку и уйдёт. Чтобы улучшить быстродействие веб-приложения, можно сжать изображения, применить кэширование и сократить количество запросов к серверу.



Руководствуются выше описанным и опираясь на техническое задание, было произведено UX-исследование. Данное исследование помогло определить сценарии поведения пользователей. Так как веб-приложение разрабатывалось во основном для работы со смартфона была полностью переработана структуру ресурса, продуман пользовательский путь и выявлены положительные и отрицательные моменты взаимодействия с продуктом.

В начале работы над проектом были определены функции, разделы и страницы будущего веб-приложения

* Была разработана форма Sing in и размещена в удобном для пользователя месте экрана;
* Били продуманны формы поиска транспортного средства по его «гаражному номеру» и «по государственному регистрационному номеру»;
* Была разработана форма вывода данных транспортного средства согласно техническому заданию
* Была разработана форма для внесения изменений в СУБД;

Исходя из анализа статей о цветовой палитре и ее влияния на психическое и моральное состояние пользователя, были соблюдены все рекомендации медик научных институтов.

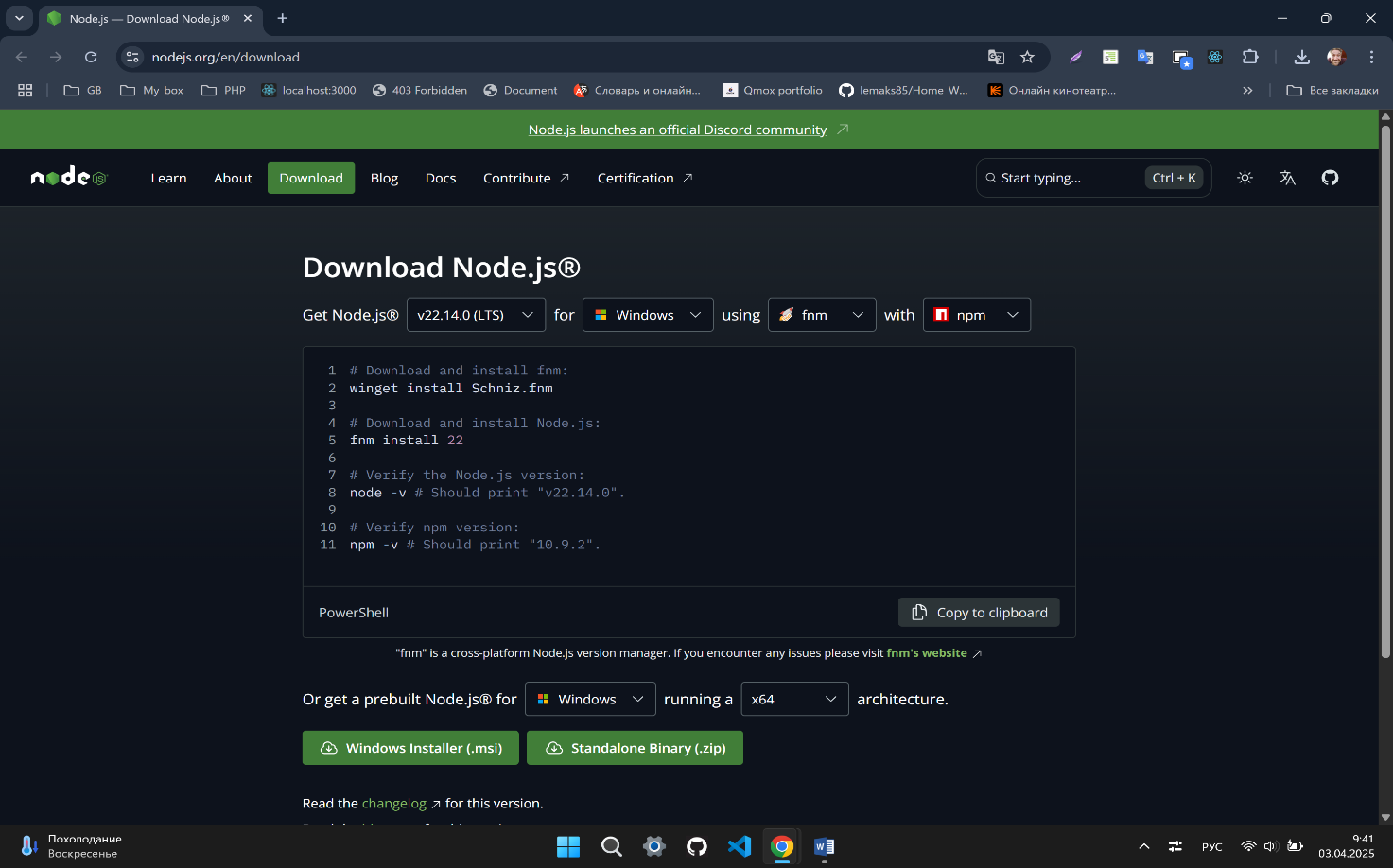
Проведено натуральное исследование юзабилити веб-приложения. Порядка 150 человек оставили положительные отзывы.

Особое внимание было уделено разработке логотипа веб-приложения.

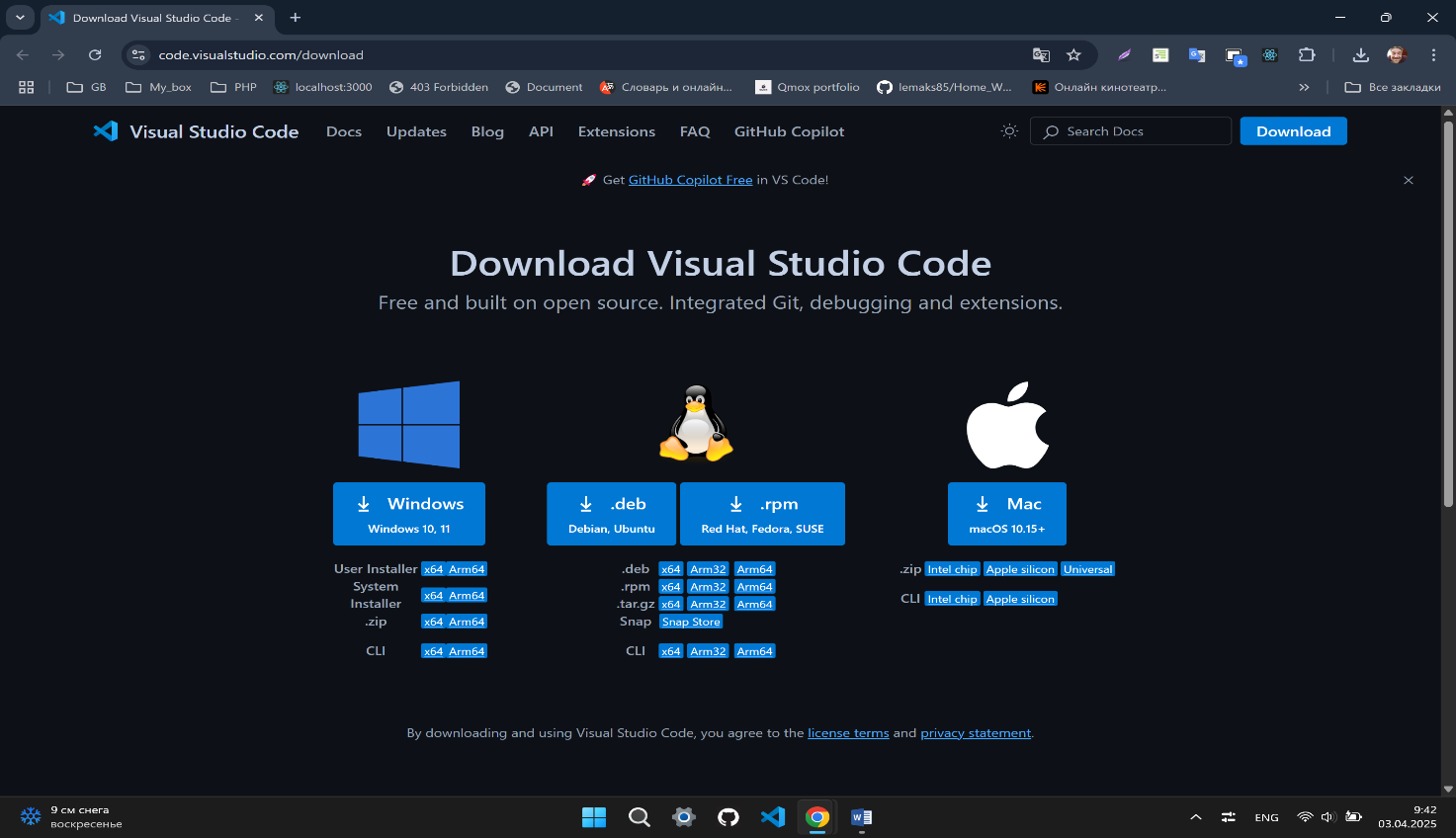


## 2.2 Разработка веб-приложения.

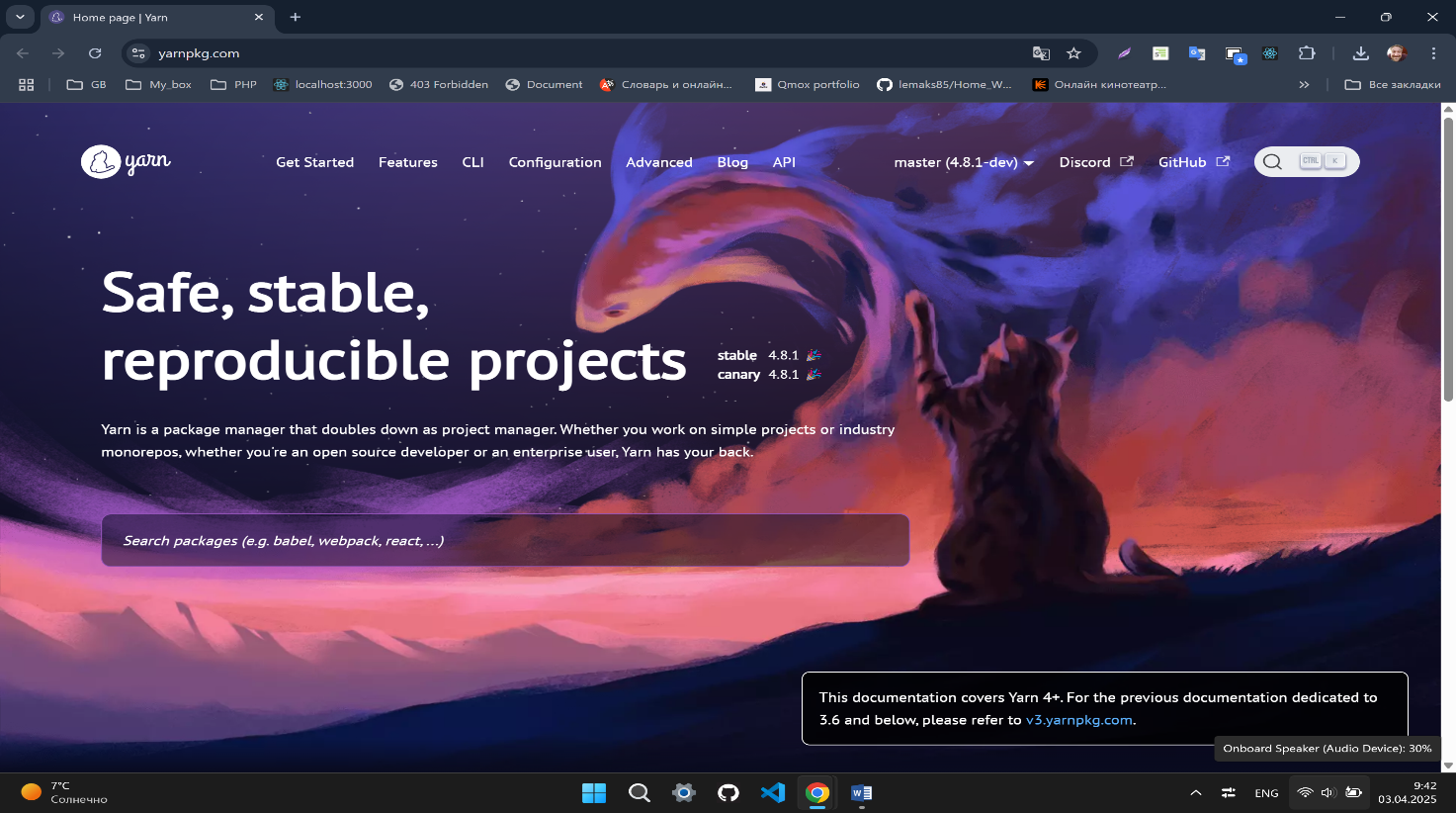
Устанавливаем последнюю актуальную версию Node JS.



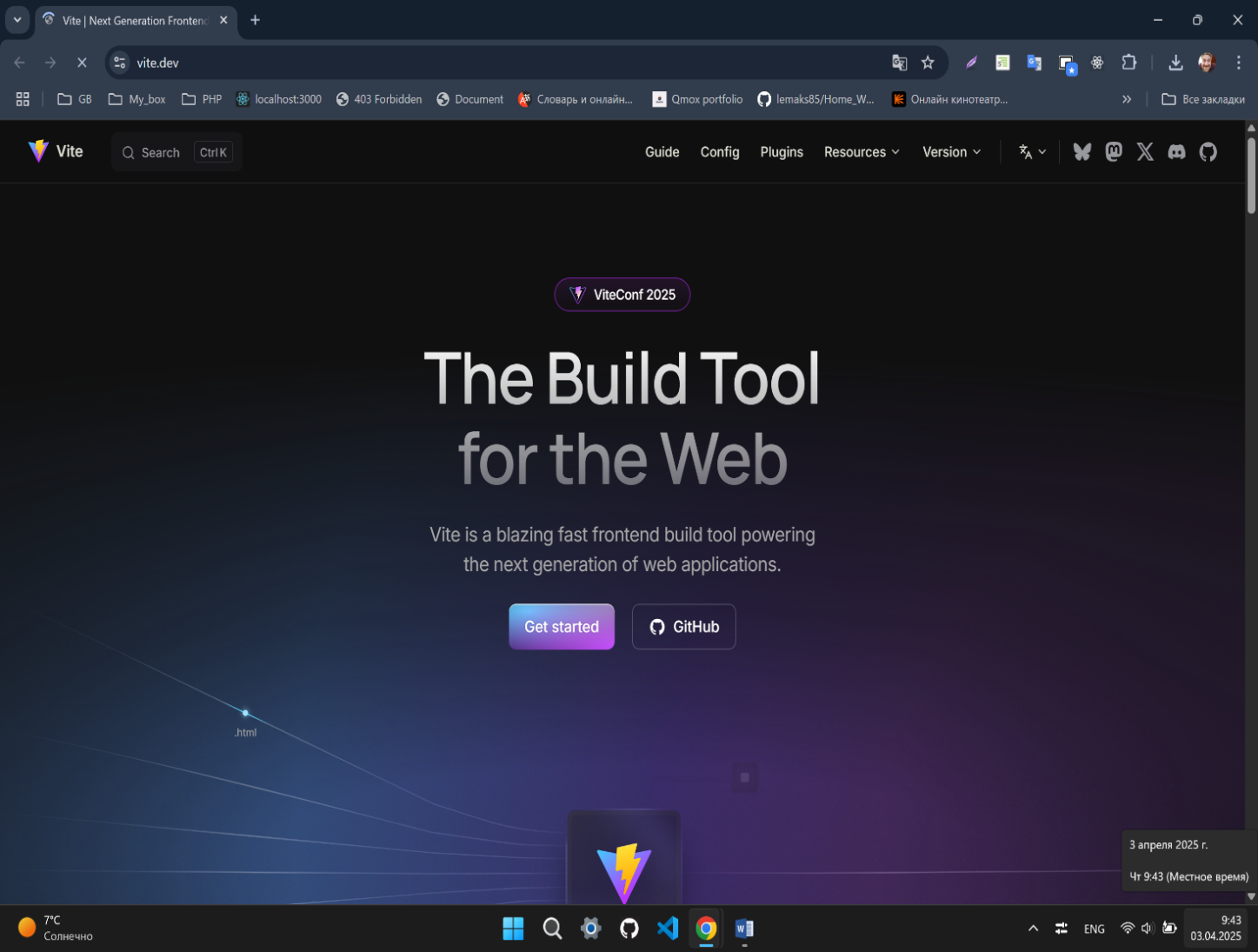
Устанавливаем последнюю актуальную версию VS Code.



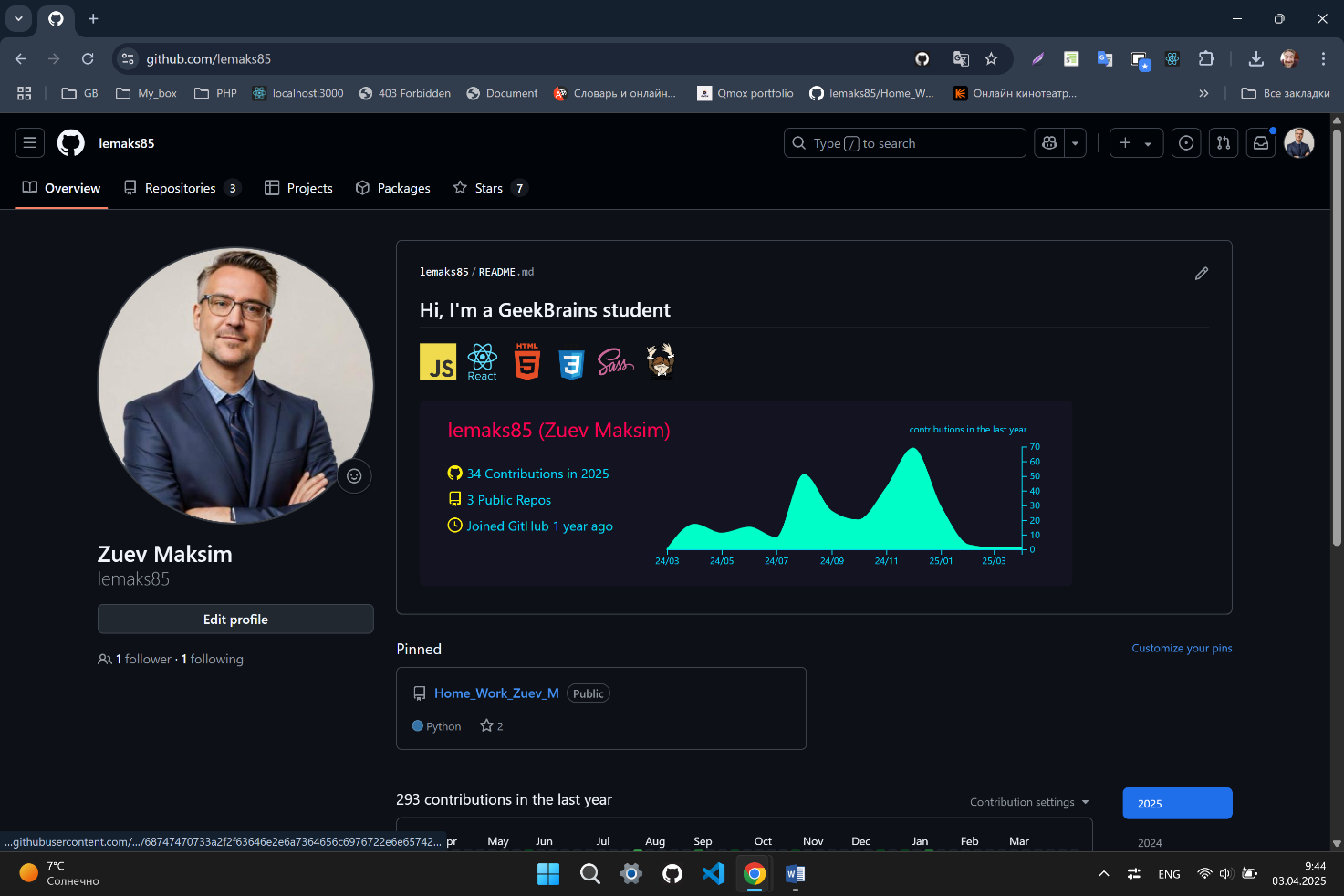
Устанавливаем последнюю актуальную версию пакетного менеджера Yarn.



Устанавливаем последнюю актуальную версию Vite.



Устанавливаем последнюю актуальную версию GitHub Desktop.



Все выше перечисленное программное обеспечение, поможет быстрее и эффективнее

выполнить разработку веб-приложения.

Основная часть веб-приложения будет разработана в среде Node JS в связке с React JS так считаю, что данная среда является более востребованной по состоянию на 2025 год и имеет неоспоримые преимущества:

* **Высокая скорость**. JavaScript-код, который выполняется в среде Node.js, может быть в несколько раз быстрее, чем написанный на языках вроде Ruby или Python.
* **Универсальность и гибкость**. В Node.js выполняется код, который написан на JavaScript. Это означает, что frontend-разработчики, которые уже используют JavaScript в браузере, могут писать и клиентский, и серверный код на привычном языке программирования, не изучая инструмент с нуля.
* **Большое количество модулей и библиотек**. Экосистема Node.js стремительно развивается благодаря менеджеру пакетов NPM, который содержит более 500 000 модулей и библиотек open-source, которые находятся в свободном доступе.
* **Работа на движке Chrome V8**. Node.js работает на JavaScript-движке V8 от Google, который делает JavaScript очень быстрым.
* **Данные в реальном времени.** Node.js имеет событийно-ориентированную архитектуру, что делает его идеальным для приложений, работающих с данными, поступающими в реальном времени.
* **Ускорение процесса разработки.** Использование React и Node.js обеспечивает высокую окупаемость инвестиций, а также экономию времени и денег.
* **Масштабируемость.** React.js и Node.js позволяют создавать веб-приложения, которые реагируют на различных устройствах. Это особенно полезно при работе над крупными проектами.
* **Один язык для фронтенда и бэкенд.** Разработчикам не нужно изучать сложные языки бэкенд, такие как Ruby или Python. Они могут использовать один язык для React-разработки пользовательского интерфейса и Node.js-разработки на стороне сервера.
* **Высокая производительность.** Node.js — идеальная платформа для высокопроизводительных приложений. При использовании Node.js с React можно создавать мощные веб-приложения, не беспокоясь о медленном времени отклика или загрузки страниц.
* **Одностраничные приложения.** Node.js и React обеспечивают идеальную платформу для разработки мощных одностраничных приложений.

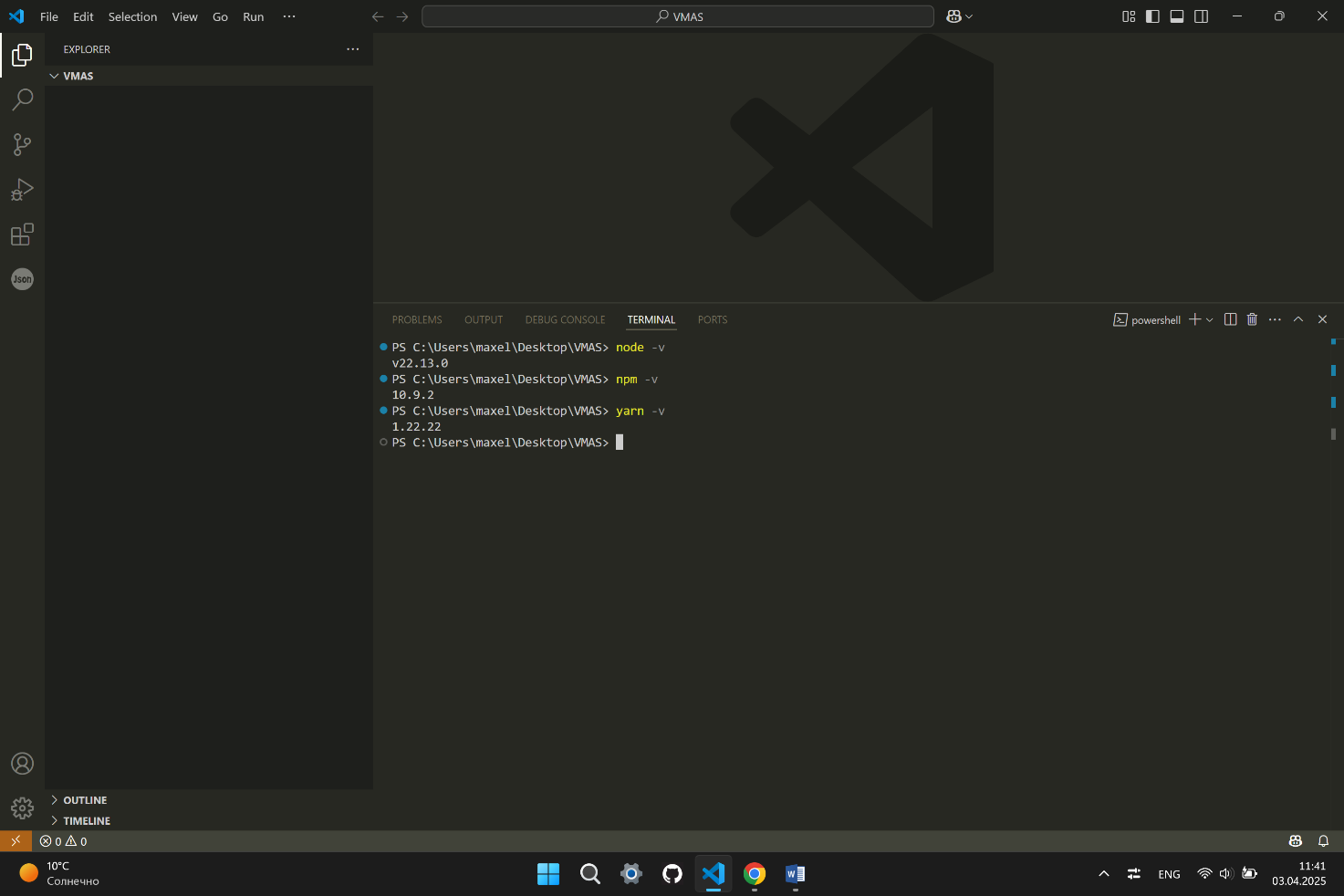
IDE для разработки веб-приложения будем использовать VS Code в связке с дополнительными расширениями, так как данная IDE обладает рядом преимуществ для новичков в IT отрасли:

* **Простота использования**. Интерфейс редактора интуитивно понятен даже новичку. 1
* **Малый вес**. По сравнению с IDE редактор быстро работает и не требователен к ресурсам компьютера.
* **Кроссплатформенность**. Версии VS Code есть для всех популярных операционных систем, что позволяет работать вне зависимости от ОС и удобнее переносить проекты с одного устройства на другое.
* **Поддержка разных языков**.  Редактор поддерживает огромное количество языков программирования, и их легко подключить с помощью плагинов.
* **Гибкость**. У VS Code множество расширений и удобных функций, его можно легко и гибко настроить под себя в зависимости от потребностей разработчика и технологий, с которыми он работает.
* **Наличие горячих клавиш**. Они позволяют ускорить процесс написания кода и сделать его более удобным.
* **Контроль версий**. VS Code поддерживает систему контроля версий Git, прямо из редактора можно закоммитить изменения или откатить коммит, добавить к нему комментарий или отправить в удалённый репозиторий.

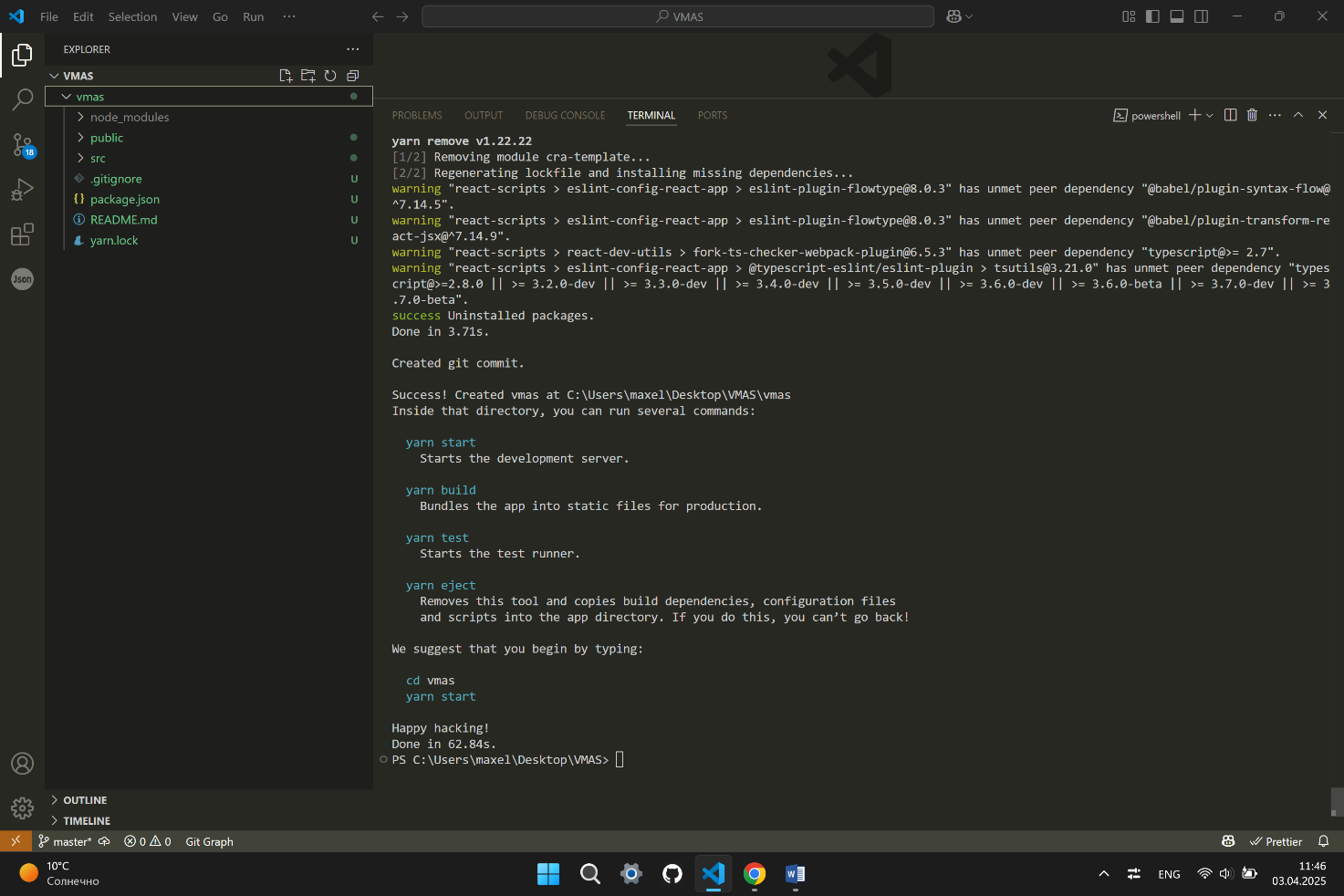
Каждый уважающий себя разработчик или команда разработчиков не могут обойтись без системы контроля версий Git разработанной Линусом Торвальдсом. Созданная им система используется по сегодняшний день и показала себя с лучшей стороны. Вот несколько преимуществ:

* **Упрощение совместной работы**. Несколько людей могут одновременно работать над одним проектом, а система отслеживает, кто и что сделал. Это помогает предотвратить ошибки, улучшить код и сделать процесс разработки более эффективным.
* **Хранение проектов**. В GitHub файлы проектов сохраняются в репозиториях. Изменения кода сохраняются внутри хранилища в виде коммитов и веток.
* **Показ портфолио**. Раздел Repositories демонстрирует реализованные проекты и технические навыки разработчика, профиль на GitHub становится публичным портфолио программиста.
* **Доступ к ресурсам**. На платформе есть богатое сообщество разработчиков и энтузиастов, которые могут помочь командам и отдельным пользователям.
* **Работа с проектами с открытым исходным кодом**. Создавая такой проект на GitHub, разработчик даёт всем разрешение использовать контент проекта, исходный код и документы по дизайну.
* **Использование для личных проектов**. Например, можно создать репозиторий, где можно делиться кодом, изменять файлы и отслеживать изменения.
* **Применение в корпоративных проектах**. Организации могут использовать GitHub Enterprise Server для разработки программного обеспечения и управления исходным кодом внутри своих команд.

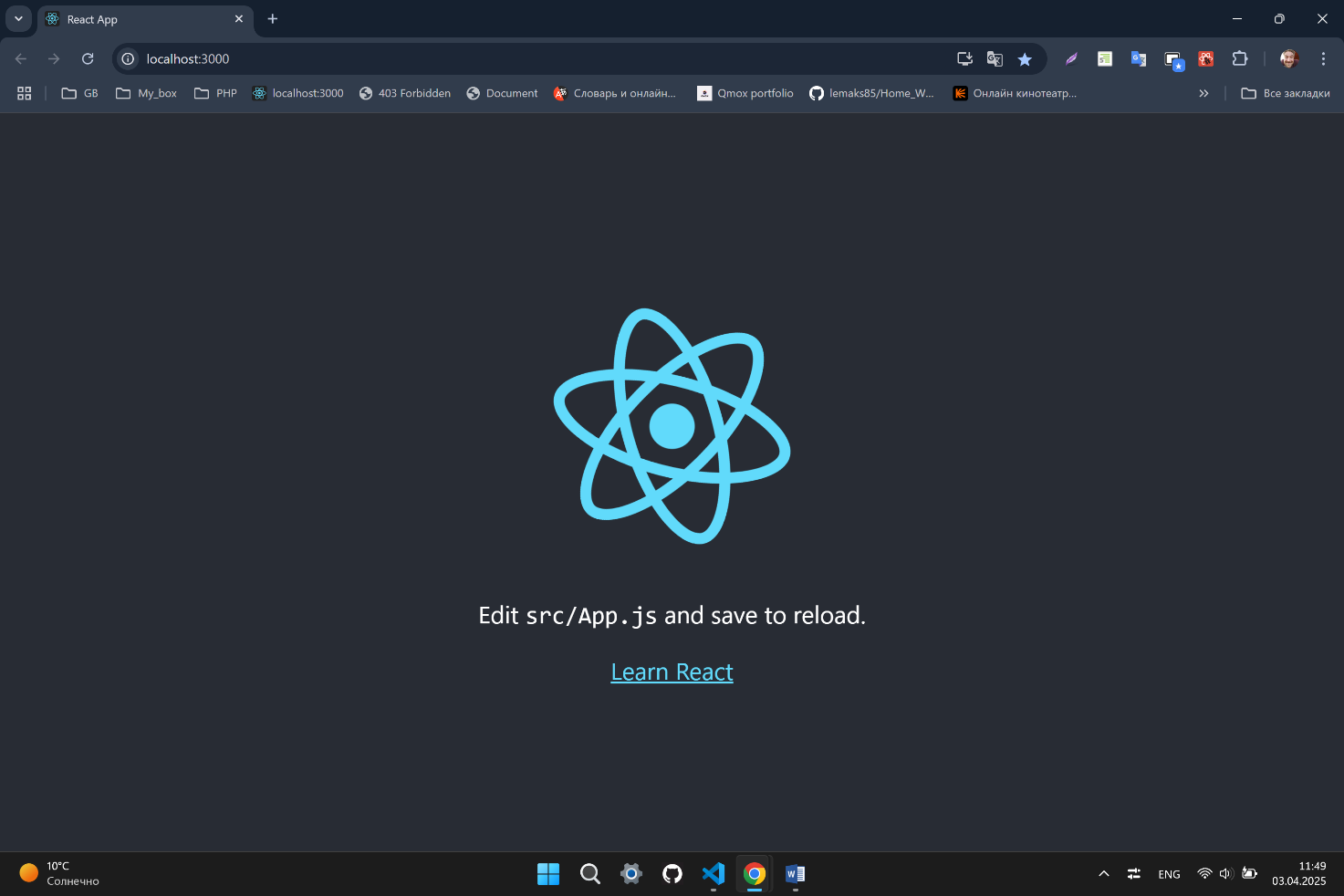
После того как у нас все установлено и проверены через терминал все версии установленного ранее программного обеспечения, переходим к созданию проекта и его зависимостей.



Далее создаем сам проект «VMAS»



Производим тестовый запуск нашего проекта, чтобы убедится, что все работает исправно и нет ошибок.



Наш проект запустился, ошибок нет переходим к разработке самого веб-приложения.

## 2.3 Запуск и тестирование веб-приложения.

Даже в случае самой базовой конфигурации по мере добавления новых страниц и функциональности тестировать веб-приложение становится все сложнее. И чтобы помочь начинающим в этом нелегком деле, я написал небольшое вводное руководство.

Интернет является наиболее продвинутой информационной технологией и

платформой разработки приложений, от которой сегодня уже зависит львиная доля экономики всей планеты.

Проще всего представить себе интернет как сеть взаимосвязанных узлов, называемых серверы и клиенты, которые обмениваются данными по установленным протоколам вроде TCP/IP, HTTP, FTP и т.д. Рекомендую почитать о [зарождении](https://home.cern/science/computing/birth-web) (англ.) этой технологии и ознакомиться с ее [техническими особенностями](http://info.cern.ch/hypertext/WWW/Summary.html) (англ.).

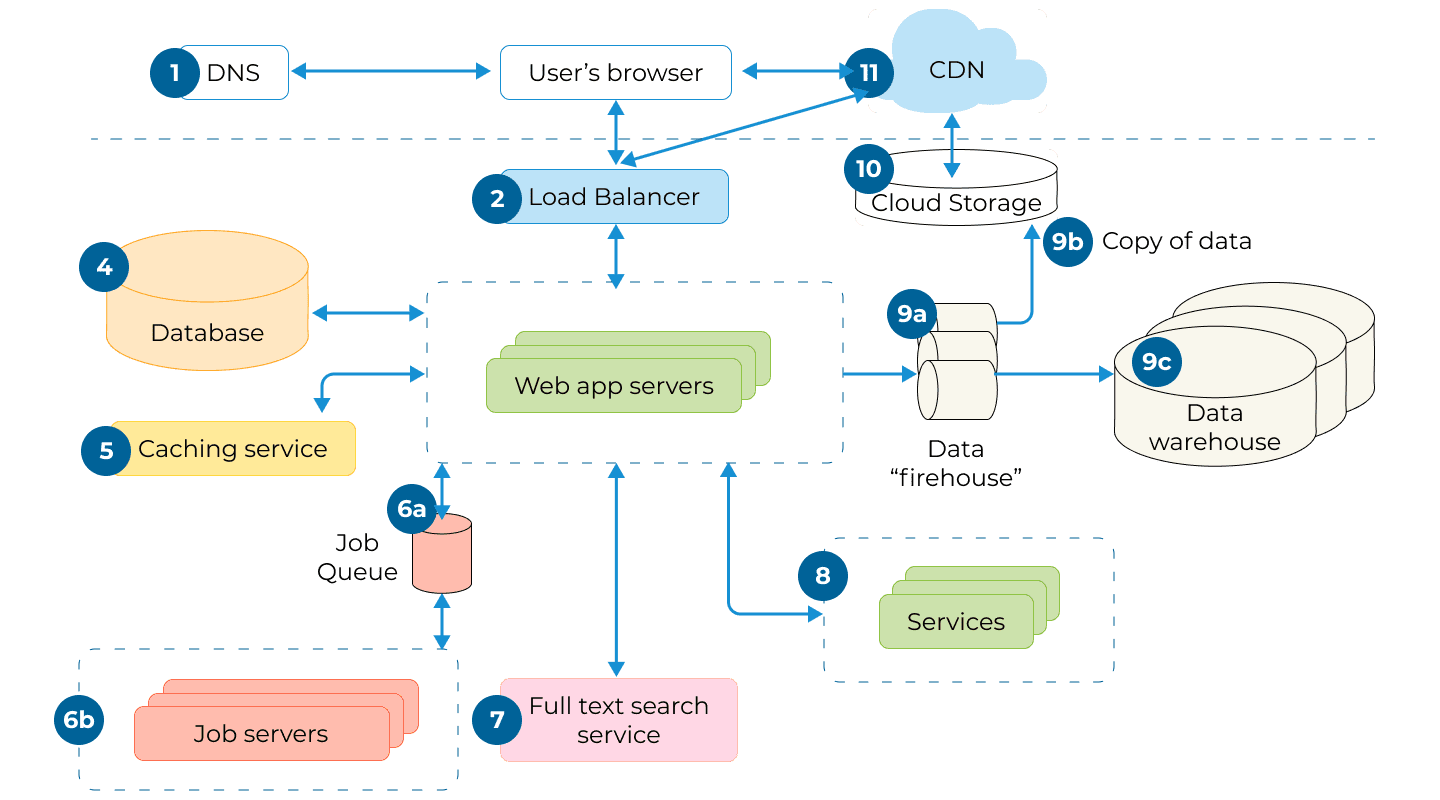
При разработке для веб по современным стандартам важно использовать актуальный набор подходящих инструментов и архитектуру, на всех уровнях поддерживающую CI/CD (непрерывное развертывание и доставку) и QA (обеспечение качества).  
  
**Структура веб-приложений**

Веб-приложение, или веб-сайт, представляет собой пакет, загруженный на клиентской стороне и содержащий множество слоев взаимозависимых модулей, построенных на основе веб-технологий:

* *HTML* – язык разметки гипертекста, представляющий основанную на тегах систему разметки документов, определяющую структуру и отдельные компоненты того, что в итоге компилируется в объектную модель документа, или DOM.
* *CSS* – каскадная таблица стилей, являющаяся Фреймворком для стилизации приложения, с помощью которого идентифицируют и оформляют различные части DOM видимой области страницы. CSS предоставляет возможности вроде выбора элементов по их ID, классу и отношению к другим элементам DOM.
* *JavaScript* – высокоуровневый интерпретируемый скриптовый язык, с помощью которого прописывается и выполняется все поведение приложения.

Функциональность этих основных технологий дополнительно расширяется слоями других технических решений, диапазон которых поистине огромен. Сюда относятся различные JS-фреймворки вроде Angular, React и Vue, CSS-инструменты предварительной/пост-обработки вроде Less и Sass, а также всевозможные механизмы HTML-шаблонизации.

Помимо слоя фронтенда в большинстве приложений также присутствует серверная сторона, или бэкенд, с API, построенным на микросервисах и базах данных. Серверная часть содержит все данные и бизнес-логику, абстрагируя всю эту информацию в соответствующие контракты, к которым фронтенд может обращаться через HTTP-методы, используя нужную форму запроса и учетные данные.



В зависимости от применяемых инструментов и внутренней сути сайта ему потребуется подходящая стратегия хостинга и инфраструктура. Сайты могут размещаться на различных системах, которые в общем можно разделить на два вида:

*Статические веб-хосты*: используются для статических сайтов и представляют собой платформы доставки на базе хранилищ файлов, предоставляющие домены, электронные адреса, DNS и прочие возможности вроде SSL, шифрования и интеграции сторонних сервисов. В качестве примеров можно привести Godaddy, Hostinger и прочие.

* *Динамические веб-хосты*: служат для размещения динамических веб-приложений. К наиболее популярным относятся облачные платформы вроде AWS, Google Cloud, Azure, Salesforce и IBM cloud, которые предоставляют в качестве сервисов различные вычислительные возможности, включая виртуальные машины, базы данных, масштабирование ресурсов по требованию и т.д. Эти платформы являются стандартом для развертывания веб-приложений в бизнес-целях ввиду их высокой производительности и безопасности. Кроме того, они предлагают к использованию передовые технологии ИИ и машинного обучения.

Веб-приложение, даже при небольшой начальной конфигурации по мере добавления в него страниц, контента и функциональности будет постепенно разрастаться. И однажды, когда будет достигнут определенный порог сложности, значительно затруднится управление приложением, а также отслеживание потребления и выделения ресурсов.  
  
 В зависимости от типа создаваемого приложения разработчик может использовать для него различную структуру:

* *Одностраничное приложение*. Состоит из одной модели документа, в рамках которой реализуется вся функциональность. Такие проекты задействуют большой объем функциональной логики, упаковываемой и отправляемой на компьютер клиента с соответствующими оптимизациями безопасности и производительности. В качестве известных примеров можно назвать Gmail, Facebook, GitHub и пр.
* *Многостраничное приложение*. Наиболее частый выбор, при котором приложение разделяется на множество страниц, доступных по различным путям URL. Создаются такие приложения с помощью серверных фреймворков и механизмов шаблонизации. При этом они имеют характерное преимущество в безопасности перед одностраничными решениями.
* *Прогрессивное веб-приложение*. Это современный способ применения WebView на мобильных устройствах для выполнения веб-приложений в качестве нативных, используя service worker, манифесты и оболочку.

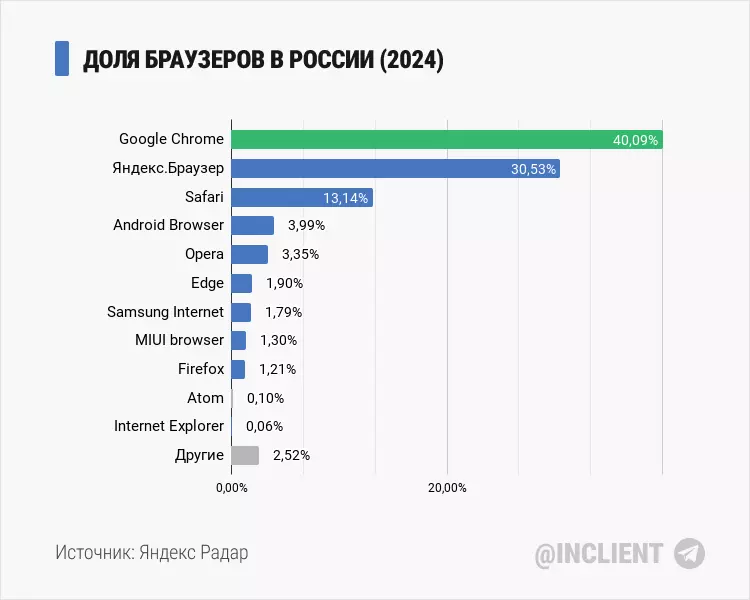
Хотя какой бы подход вы не выбрали, для обеспечения эффективности рабочего потока DevOps в каждом случае важно будет реализовать план тестирования как на уровне разработки, так и на уровне эксплуатации приложения.

**Тестирование веб-приложений**

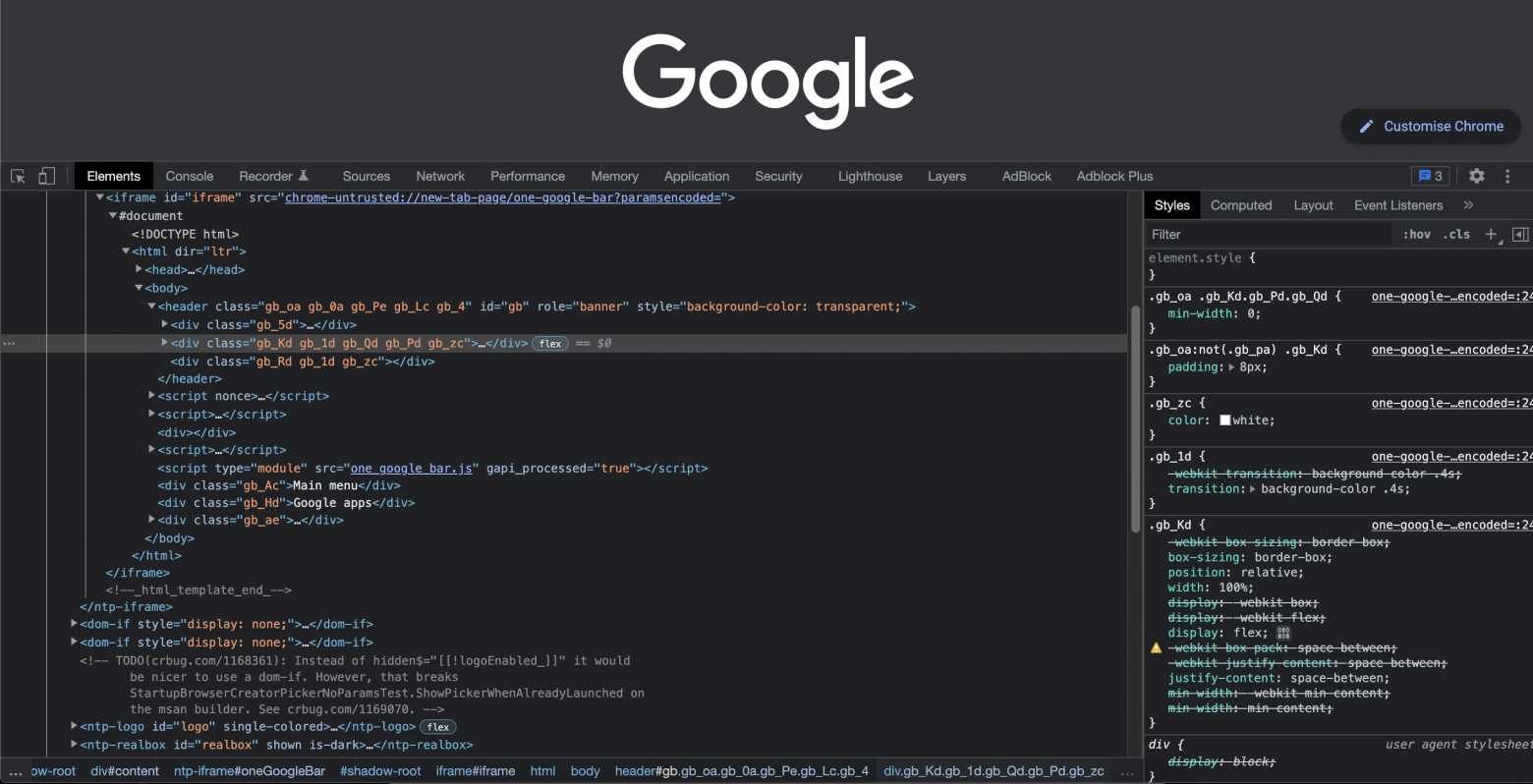
Хороший план тестирования сайта включает в себя стратегию, задачи тестирования, подход, расписание тестов и среду их выполнения. Стратегия должна быть выстроена так, чтобы сайт в итоге отвечал всем бизнес-требованиям и соответствовал своему назначению.  
  
Что касается подхода к тестированию, то он должен включать в себя:

* *Модульное тестирование*: проверку отдельных частей базы кода с помощью модульных тестов. Подробнее об этом можете почитать в руководстве по JavaScript (англ.) и документации Python (англ.). При этом современные подходы программирования по принципу no-code/low-code и инструменты на основе ИИ позволяют написание модульных тестов частично автоматизировать.
* *Интеграционное тестирование*: подразумевает проверку различных сегментов кода сайта в виде независимых функций или модулей с помощью тестовой программы или прочих инструментов, активируемых при слиянии кода с родительским репозиторием. Все ведущие хостинги вроде GitHub, Gitlab и Bitbucket имеют встроенную поддержку реализации CI/CD.
* *Системное тестирование*: тестирование сайта на уровне пользовательского интерфейса и функционала вроде авторизации, регистрации и прочих потоков. В ходе этого процесса проверяется корректность совместной работы различных элементов сайта. Чаще всего с этой целью используется Selenium, наиболее популярный фреймворк для автоматизации действий браузера.
* *Приемочное тестирование*: как правило, это последняя стадия тестирования, на которой полностью собранное приложение с данными проверяется в продакшн-среде или среде интеграции (стейджинг). Этот этап включает в себя тестирование с привлечением фактических или мнимых пользователей. При этом наиболее эффективным способом управления процессом утверждения изменений в быстро меняющемся UI продукта является автоматизированное визуальное тестирование.
* *Тестирование производительности*: при постепенном увеличении числа пользователей необходимо обеспечить, чтобы сервера справлялись с нагрузкой в пиковые часы использования приложения. Помимо этого, данный этап также подразумевает поддержание сквозной безопасности в каждой точке контакта между сайтом и пользователями с помощью правильного использования HTTP-заголовков и анализа метаданных.

**Роль браузеров**

Для просмотра любого сайта необходим браузер, который запускается в операционной системе устройства. Эти особые приложения разрабатываются несколькими компаниями и обычно являются бесплатными.  
  
  
  
  
 Браузеры играют важную роль, не только тем, что делают сайты доступными для пользователей, но и тем, что помогают разработчикам, предоставляя широкий арсенал инструментов для тестирования и отладки различных аспектов веб-приложений.  
  
 Большинство браузеров предлагают в качестве вспомогательного интерфейса инструменты разработки, позволяя при желании заглянуть в код отрисовываемых страниц приложений. Доступ к этим инструментам обычно можно получить через правый клик по странице и выбор опции «Просмотреть код элемента», либо комбинацией клавиш Ctrl+Chift+I.  
  
Ниже я опишу некоторые основные компоненты инструментов разработчика в Chrome:

* *Elements*: проводник по элементам, который предоставляет доступ к скомпилированной DOM с множеством функций для добавления/удаления компонентов и установки состояний вроде наведения (hover), фокуса (focus) и т.д.
* *Console*: журнал вывода консоли при выполнении JavaScript. Очень полезен при отладке. Эту вкладку также можно использовать для выполнения фрагментов JS-кода на активной странице и просмотра его вывода.
* *Sources*: в этой вкладке вы увидите список всех файлов исходного кода, загруженных текущей страницей. Справа от нее находится отладчик скриптов, с помощью которого можно расставлять точки останова и корректировать выполнение сайта в реальном времени.
* *Network*: эта вкладка логирует все сетевые вызовы к странице и от нее, отражая большой список подробностей вроде типа, статуса, запроса/ответа, тайминга и т.д. Здесь также есть возможность эмуляции сценариев сетевой доступности с помощью функции ограничения скорости (throttling).
* *Performance*: на этой вкладке у вас есть возможность проводить запись и анализ быстродействия страницы во времени.
* *Memory*: эта вкладка позволяет делать моментальные снимки кучи, визуализировать использование памяти во времени, обнаруживать ее утечки, просматривать размеры объектов и т.д.
* *Application*: с помощью этого инструмента можно оценивать, редактировать и отлаживать service worker, кэш и прочие параметры.
* *Security*: предоставляет краткую сводку по валидности сертификата SSL.

  
  
  
 В браузере также присутствует встроенная панель эмуляции устройств, которая позволяет эмулировать сценарии пользовательского интерфейса различных устройств, предлагая список предустановленных профилей разрешения, сетевой задержки, масштабирования, поворота экрана, а также возможность устанавливать собственное разрешение для тестирования отзывчивости.  
 Раздел разработчика в Chrome постоянно получает новые и более совершенные инструменты вроде Lighthouse, Recorder и прочих, которые предоставляют углубленные возможности анализа общего состояния приложения.  
 Более того, помимо инструментов разработчика, браузеры дают доступ ко множеству API для взаимодействия с вводом/выводом, камерой, геолокацией, аудио/речью, сетями и не только. DOM и многие другие компоненты активной сессии доступны на уровне объектов.

**Тестирование локального развертывания**

Прежде чем переходить к настройке потока тестирования в браузере, рекомендуется реализовать модульный тест в приложении, чтобы убедиться в надежности и чистоте базы кода.  
 Когда разработчики создают сайт, они проверяют все изменения и обновления в браузере на рабочем компьютере. После предварительного функционального тестирования следующим важным шагом идет проверка совместимости устройства и браузера, что позволяет понять, как приложение выглядит и работает на их разных комбинациях.  
 В связи с тем, что разработчику недоступен весь спектр устройств для тестирования, отловить всевозможные баги становится сложно. Несмотря на то, что в веб-среде установлены определенные стандарты и спецификации, при их трактовке различными браузерами возникают разночтения, чему способствуют отличия в реализации браузерных движков и используемых интерпретаторов JS-кода. В целом современный браузер сам по себе предоставляет инструментарий для открытия и отладки любого аспекта приложения.

**Автоматизированное визуальное тестирование**

Для пользователя работа с приложением является визуальным опытом, и его перемещение по сайту определяется особенностями дизайна, установленными в ходе бизнес анализа и проектирования. Для того, чтобы итоговый сайт оставался верен заложенным идеям дизайна, необходимо обеспечить подходящий режим визуального тестирования и утверждения вносимых в базу кода изменений.  
 И главную роль в этом процессе играет команда по обеспечению качества (QA team). С десяток лет назад визуальное тестирование выполнялось в основном вручную, но сегодня оно все больше автоматизируется. Один из эффективных способов его реализации состоит в выполнении визуальных тестов в потоке CI/CD.  
 Вы можете добавить членов команды и настроить выполнение тестов на продакшн-среду, где каждую сборку можно будет оценить на визуальные отличия, а команда сможет согласованно утверждать изменения.

Вот мы и расписали в общих чертах путь ручного тестирования на локальной машине с использованием автоматизации, параллельно обсудив применение визуальных тестов для контроля изменений UI. После успешной реализации всех этих этапов ваш сайт будет готов к отправке в продакшн.  
 Но это еще не значит, что тестирование закончено. Проделав все описанное, вы лишь убедитесь, что при правильной конфигурации всех тестовых кейсов и плавном протекании потока CI/CD вам потребуется меньше времени проводить за проверкой и исправлением ошибок, вместо этого занимаясь разработкой нового функционала. Для того же, чтобы получить эффективную систему тестирования, посоветую вам учитывать следующие нюансы:

* Прогоны тестов на всех этапах и на всех уровнях должны отслеживаться, поскольку автоматизация поможет лишь с выполнением и обнаружением ошибок, а для понимания серьезности неполадок и поиска способа их исправления необходима хорошая коммуникация внутри команды.
* Информационные панели обеспечивают отличный способ мониторинга состояния сборок и результатов тестовых прогонов, но, если их будет слишком много, это только приведет к излишнему усложнению и трате времени. В связи с этим рекомендуется по возможности уровни тестирования интегрировать.
* Любое изменение в бизнес-спецификациях или дизайне должно исходить от руководства, проходя этап разработки и в конечном итоге попадая в тестовые скрипты. В грамотном процессе создания приложения перемещение информации по принципу сверху вниз оказывается плавным и быстрым.

Существует множество подходов к тестированию сайтов, и здесь можно применить разные комбинации инструментов, а с появлением новых облачных технологий веб-тестирование стало еще богаче на функционал и надежнее. Возможность сотрудничать с командой на одной платформе для построения планов тестирования дает превосходный результат, поскольку все участники имеют доступ к общим ресурсам.

# **Заключение**

Целью нашей работы была разработка веб-приложения «VMAS» для учета технического обслуживания транспортных средств. Результатом работы стала демонстрация лучших практик для создания веб-приложений, собранных на основе личного опыта и современных тенденций в мире веб-разработки, а также были рассмотрены достоинства и недостатки существующих популярных инструментов и подходов.

В ходе работы был представлен тщательный сравнительный анализ Фреймворков для разработки веб-приложений с полным разбором преимуществ и недостатков, исходящих из целей поставленной нами задачи. Был аргументирован и выбран Фреймворк React JS. Помимо этого, были реализованы и другие задачи проекта, а именно:

* спроектировать инфраструктуру веб-приложения;
* разработать клиентскую часть веб-приложения согласно макетам;
* разобрать серверную часть веб-приложения согласно техническому заданию;
* разместить статический контент веб-приложения на сервере Beget;

В ходе тестирования веб-приложения были найдены и устранены мелкие недочёты и пожелания сотрудников компании. Было услышано много слов благодарности и успехов в работе.

Во время работы над настоящим проектом были получены практические умения в исследовании предметной области, описания проектного решения, построения моделей.

Исходя из вышеперечисленного, можно сделать вывод, что цель работы, а именно – разработка веб-приложения «VMAS» была достигнута в полном объёме и с должным уровнем аналитического подхода к современной ситуации в IT-индустрии. А исходя из вышеперечисленных современных реалий, можно сделать заключение, что веб-приложение «VMAS» будет пользоваться популярностью среди специалистов и не только, а значит, обладает высоким потенциалом для развития.

# **Список используемой литературы**

Великолепный PWA — Статья на vc.ru про PWA

Научная статья опубликованная в журнале «Известия Санкт-Петербургского государственного экономического университета» С.Л.Торосян «Роль информационных технологий в бизнес-проекте»

Научная статья журнала «Вестник науки» Терехина С.В. «Современные информационные технологии в бизнесе»

Документация Google по Angular. URL: https://angular.io/docs

RxJS for Reactive programming. URL: https://angular.io/guide/rx-library

Документация Facebook по React. URL: https://react.dev

Количество скачиваний React, Angular и Vue за последний год. URL: https://npmtrends.com/angular-vs-react-vs-vue

Server React DOM APIs. URL: https://react.dev/reference/react-dom/server

Progressive Javascript Framework. URL: https://vuejs.org

RawGit. JS Framework Benchmark URL: https://rawgit.com/krausest/js-framework-benchmark/master/webdriver-ts-results/table.html

Web Apps Performance Track Metrics by Google Chrome Labs. URL: https://perf-track.web.app

Современный учебник JavaScript: сайт. – URL: https://learn.javascript.ru

Документация по TypeScript c официального сайта: сайт. – URL: https://www.typescriptlang.org/

Основы CSS: сайт. – URL: https://developer.mozilla.org/ru/docs/Learn/Getting\_started\_with\_the\_web/ CSS\_basics

Документация по Angular c официального сайта: сайт. – URL: https://angular.io

Документация по Vue.js: сайт. – URL: <https://vuejs>.

Документация по React c официального сайта: сайт. – URL: https://ru.reactjs.org Документация по Redux: сайт. – URL: https://reactdev.ru/redux/

Документация по Next.js c официального сайта: сайт. – URL: https://nextjs.org

Документация по Docker c официального сайта: сайт. – URL: https://www.docker.com/get-started Документация разработчика Sentry c официального сайта: сайт. – URL: https://develop.sentry.dev/ Документация по GitHub c официального сайта: сайт. – URL: https://docs.github.com/en/actions 12.IDE NodeJS: сайт. – URL: https://nodejs.org/en

Знакомство с Content Delivery Network: сайт. – URL: https://habr.com/ru/company/webzilla/blog/236511 Докеризация веб-приложения Node.js NodeJS: сайт. – URL: https://nodejs.org/ru/docs/guides/nodejs-docker-webapp

# **Приложение**

**App.js**

import logo from './1.png';

import './App.css';

*function* App() {

  return (

    <div className="App">

      <header className="App-header">

        <img src={logo} className="App-logo" alt="logo" align="middle" />

          <a href="#" className="App-sing">Sing in</a>

      </header>

    </div>

  );

}

export default App;

**Index.js**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import Main from './Main';

import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

<Main />

</React.StrictMode>

);

reportWebVitals();

Форма авторизации https://danshin.ms/simple-login-react-app/

// Импортируем необходимые модули из библиотеки react

import { createContext, useContext, useState } from "react";

import { BrowserRouter as Router, Route, Routes, Navigate, Outlet, useLocation, useNavigate, Link } from 'react-router-dom';

// Определяем тип контекста с двумя полями: isAuthenticated и setAuth

type AuthContextType = {

isAuthenticated: boolean; // флаг, показывающий, аутентифицирован ли пользователь

setAuth: (auth: boolean) => void; // функция для изменения значения isAuthenticated

};

// Создаем контекст с типом AuthContextType и начальными значениями по умолчанию

const AuthContext = createContext<AuthContextType>({

isAuthenticated: false,

setAuth: () => { },

});

// Компонент App является корневым компонентом приложения

function App() {

const [isAuthenticated, setAuth] = useState<boolean>(false); // определяем состояние isAuthenticated

return (

// оборачиваем компонент в Router, чтобы использовать роутинг

<Router>

{/\* используем контекст для передачи значения isAuthenticated и функции setAuth вниз по иерархии компонентов \*/}

<AuthContext.Provider value={{ isAuthenticated, setAuth }}>

<Routes>

{/\* обычные маршруты \*/}

<Route path="/" element={<Main />} />

<Route path="/login" element={<Login />} />

{/\* защищённые маршруты \*/}

<Route element={<PrivateRoute />}>

<Route path='/admin' element={<Admin />} />

{/\* другие защищённые маршруты \*/}

</Route>

</Routes>

</AuthContext.Provider>

</Router>

);

}

// Компонент PrivateRoute используется для защиты определенных маршрутов в приложении.

const PrivateRoute = () => {

const { isAuthenticated } = useContext(AuthContext); // используем контекст для получения значения isAuthenticated

const location = useLocation(); // получаем текущий маршрут с помощью хука useLocation()

return (

// если пользователь авторизован, то рендерим дочерние элементы текущего маршрута, используя компонент Outlet

isAuthenticated === true ?

<Outlet />

// если пользователь не авторизован, то перенаправляем его на маршрут /login с помощью компонента Navigate

// свойство replace указывает, что текущий маршрут будет заменен на новый, чтобы пользователь не мог вернуться

// обратно, используя кнопку "назад" в браузере.

:

<Navigate to="/login" state={{ from: location }} replace />

);

}

// Компонент Login отображает страницу авторизации и обрабатывает вход пользователя.

const Login = () => {

const { setAuth } = useContext(AuthContext); // используем контекст для получения значений isAuthenticated и setAuth

const navigate = useNavigate(); // используем хук useNavigate для навигации по маршрутам

const location = useLocation(); // используем хук useLocation для получения текущего маршрута

// получаем маршрут, на который нужно перенаправить пользователя после авторизации

const from = location.state?.from?.pathname || '/';

return (

<>

<div>Login</div>

<button type={'button'} onClick={() => {

setAuth(true); // устанавливаем флаг isAuthenticated в true

navigate(from, { replace: true }); // перенаправляем пользователя на страницу, которую он запрашивал до авторизации

}}>Login</button>

</>

);

}

// Компонент Main отображает главную страницу приложения.

const Main = () => {

return (

<>

<div>Main</div>

<Link to='/admin'>Go to Admin Page</Link>

</>

);

}

// Компонент Admin отображает страницу для администратора.

const Admin = () => {

return (

<>

<div>Admin</div>

<Link to='/main'>Go to Main Page</Link>

</>

);

}

// Компонент App является корневым компонентом приложения и содержит маршруты и контекст для авторизации.

export default App;

#Создаем новое React приложение с TypeScript шаблоном в директории dms-auth-simple с помощью команды npx create-react-app

npx create-react-app dms-auth-simple --template typescript

#Переходим в созданную директорию

cd dms-auth-simple/

# создаём структуру папок и файлы, необходимые для проекта

mkdir ./src/components/

touch ./src/components/Navbar.tsx

touch ./src/components/PrivateRoute.tsx

mkdir ./src/context/

touch ./src/context/AuthProvider.tsx

mkdir ./src/hooks/

touch ./src/hooks/useAuth.tsx

mkdir ./src/pages/

touch ./src/pages/Admin.tsx

touch ./src/pages/Login.tsx

touch ./src/pages/Logout.tsx

touch ./src/pages/Main.tsx

mkdir ./src/routes

touch ./src/routes/routes.tsx

#Устанавливаем библиотеку react-router-dom с помощью команды npm install

npm install --save react-router-dom

import { Route, Routes } from 'react-router-dom';

import Login from '../pages/Login';

import Main from '../pages/Main';

import Admin from '../pages/Admin';

export const useRoutes = () => {

return (

<Routes>

<Route path="/" element={<Main />} />

<Route path="/login" element={<Login />} />

<Route path="/admin" element={<Admin />} />

</Routes>

)

}

export default useRoutes

import { Route, Routes } from 'react-router-dom';

import { PrivateRoute } from '../components/PrivateRoute';

import Main from '../pages/Main';

import Login from '../pages/Login';

import Admin from '../pages/Admin';

import Logout from '../pages/Logout';

export const useRoutes = () => {

return (

<Routes>

<Route index element={<Main />} />

<Route path="/" element={<Main />} />

<Route path="/login" element={<Login />} />

<Route element={<PrivateRoute />}>

<Route path='/admin' element={<Admin />} />

<Route path="/logout" element={<Logout />} />

</Route>

</Routes>

)

}

export default useRoutes