

**Уральский государственный экономический университет**

## **ОСНОВЫ WEB ПРОГРАММИРОВАНИЯ**

02.03.03 -Математическое обеспечение и администрирование информационных систем,  
направленность (профиль) -разработка и администрирование информационных систем

<http://vikchas.ru>

### **Лекция 6-1. «Современные тенденции в Web- программировании»**

**Часовских Виктор Петрович**  
доктор технических наук, профессор кафедры  
ШИиКМ, ФГБОУ ВО «Уральский государственный  
экономический университет

Екатеринбург 2026

Web-программирование — одна из самых быстроразвивающихся областей IT.

Если раньше сайты представляли собой в основном статические HTML-страницы, то сегодня Web-приложения конкурируют с настольными и мобильными приложениями.

Современные Web-сервисы позволяют:

- работать с документами;
- проводить видеоконференции;
- редактировать изображения и видео;
- управлять финансами;
- использовать искусственный интеллект;
- запускать сложные бизнес-системы прямо в браузере.

Главная тенденция последних лет — **превращение Web из набора страниц в полноценную платформу для разработки приложений.**

## Эволюция Web-разработки

### Web 1.0

Первый этап развития Web характеризовался статическими сайтами.

Особенности:

- HTML-страницы;
- минимум интерактивности;
- информация передавалась в одну сторону: от сайта к пользователю;
- JavaScript использовался ограниченно.

Примеры:

- личные страницы;
- каталоги;
- информационные сайты.

### Web 2.0

Следующий этап связан с появлением интерактивных сервисов.

## Особенности:

- пользователь стал не только читателем, но и автором контента;
- появились социальные сети, блоги, форумы;
- активно используются JavaScript, AJAX, API;
- интерфейсы стали динамическими.

## Примеры:

- YouTube;
- Facebook;
- Wikipedia;
- Google Docs.

**API** (Application Programming Interface) (еі рі аі ) — это интерфейс, который задаёт правила взаимодействия между клиентом и сервером: какие запросы можно отправлять, в каком формате, какие ответы получать. В клиент-серверной модели API выступает «языком общения»: клиент не лезет напрямую в базу данных или логику сервера, а делает запросы через чётко описанные точки входа (эндпоинты).

### Взаимодействие «клиент – сервер» через API

Клиент (браузер, мобильное приложение, скрипт, другая служба) формирует HTTP-запрос к конкретному эндпоинту API.

Запрос содержит метод (GET, POST и т. д.), URL, заголовки и, если нужно, тело с данными.

Сервер принимает запрос, проверяет авторизацию, валидирует данные, выполняет бизнес-логику (часто с обращением к БД), затем формирует ответ.

Ответ возвращается клиенту в виде HTTP-ответа с кодом состояния (200, 404, 500 и др.) и полезной нагрузкой (обычно JSON/XML).

## Web 3.0

Термин Web 3.0 трактуется по-разному. Чаще всего под ним понимают:

- децентрализацию;
- использование блокчейна;
- смарт-контракты;

- токены и цифровую идентичность;
- семантический Web;
- более активное использование искусственного интеллекта.

Важно понимать, что Web 3.0 — это не полностью сформированная технология, а скорее направление развития.

## Современная архитектура Web-приложений -1

Среди современных платформ Web-разработки следует отметить экосистему Microsoft: [ASP.NET](#) Core и Visual Studio. [ASP.NET](#) Core — это высокопроизводительный кроссплатформенный фреймворк для создания Web-приложений, REST API, микросервисов и облачных сервисов на языке C#. Он широко применяется в корпоративной разработке благодаря надёжности, безопасности, строгой типизации и развитым инструментам. Visual Studio, в свою очередь, предоставляет мощную среду разработки с удобной отладкой, тестированием, профилированием, работой с базами данных, Git, Docker и Azure. Вместе [ASP.NET](#) Core и Visual Studio образуют одну из наиболее зрелых и продуктивных платформ для профессиональной Web-разработки.

### [ASP.NET](#) Core и Visual Studio как мощная среда Web-разработки

Это современный кроссплатформенный фреймворк для создания Web-приложений, REST API, микросервисов, real-time приложений и облачных сервисов.

[ASP.NET](#) Core — это фреймворк для разработки серверной части Web-приложений на языке C# и платформе .NET.

С его помощью можно создавать:

- классические Web-сайты;
- Web-приложения;
- REST API;
- микросервисы;
- backend для мобильных приложений;
- real-time приложения с SignalR;
- корпоративные информационные системы;
- облачные приложения.

[ASP.NET](#) Core является развитием более ранней технологии [ASP.NET](#), но в отличие от старых версий он является:

- кроссплатформенным;
- высокопроизводительным;
- модульным;
- хорошо подходящим для облачной разработки;
- открытым и активно развиваемым.

[ASP.NET](#) Core широко применяется в бизнесе и корпоративной разработке благодаря следующим преимуществам:

- высокая производительность;
- строгая типизация языка C#;
- развитая экосистема .NET;
- встроенная поддержка безопасности;
- удобная работа с базами данных через Entity Framework Core;
- поддержка REST API и gRPC;
- возможность создания real-time приложений через SignalR;
- хорошая интеграция с облаком Microsoft Azure;
- развитые инструменты для тестирования и отладки;
- долгосрочная поддержка со стороны Microsoft.

[ASP.NET](#) Core часто используется в крупных организациях, банках, государственных системах, образовательных платформах, ERP/CRM-системах и корпоративных порталах.

Важной частью экосистемы Microsoft является среда разработки **Visual Studio**.

**Visual Studio** — это мощная интегрированная среда разработки, которая предоставляет разработчику полный набор инструментов для создания, отладки, тестирования и публикации приложений.

## Возможности Visual Studio

Visual Studio позволяет:

- писать код на C#, JavaScript, TypeScript, HTML, CSS;
- создавать проекты [ASP.NET](#) Core;

- работать с базами данных;
- использовать встроенный отладчик;
- проводить профилирование производительности;
- писать и запускать тесты;
- работать с Git;
- управлять NuGet-пакетами;
- публиковать приложения на сервер или в облако;
- интегрироваться с Docker и Azure.

Особенно важно, что Visual Studio предоставляет очень удобные средства отладки: точки останова, просмотр переменных, пошаговое выполнение кода, анализ исключений и профилирование.

## Visual Studio Code и Visual Studio

Важно различать две среды:

- **Visual Studio** — полноценная профессиональная IDE, особенно мощная для .NET и ASP.NET-разработки.
- **Visual Studio Code** — лёгкий редактор кода с расширениями, популярный для frontend-разработки, Node.js, Python и многих других технологий.

Обе среды активно используются в современной Web-разработке, но для [ASP.NET](#) Core особенно удобна именно Visual Studio.

## Пример современной Web-архитектуры на [ASP.NET](#) Core

Типичный проект может выглядеть так:

- frontend: React, Angular, Vue или Razor Pages;
- backend: [ASP.NET](#) Core Web API;
- база данных: SQL Server, PostgreSQL или MySQL;
- ORM: Entity Framework Core;
- авторизация: [ASP.NET](#) Core Identity, OAuth2, OpenID Connect;

- real-time: SignalR;
- контейнеризация: Docker;
- облако: Microsoft Azure;
- CI/CD: GitHub Actions или Azure DevOps.

Такой стек хорошо подходит для серьёзных промышленных систем.

## Современная архитектура Web-приложений - 2

Современное Web-приложение обычно состоит из двух основных частей:

- **Frontend** — **клиентская часть**, которая работает в браузере.
- **Backend** — **серверная часть**, отвечающая за бизнес-логику, данные, авторизацию и взаимодействие с внешними сервисами.

Также важную роль играют:

- базы данных;
- API;
- системы кэширования;
- облачные сервисы;
- DevOps-инфраструктура;
- средства мониторинга.

### Клиент-разработка

Frontend — это часть приложения, с которой взаимодействует пользователь.

Современный frontend включает:

- HTML;
- CSS;
- JavaScript;
- TypeScript;
- фреймворки и библиотеки;
- инструменты сборки;

- тестирование;
- оптимизацию производительности.

## JavaScript и TypeScript

### JavaScript

JavaScript остается основным языком Web-разработки. Он используется:

- в браузере;
- на сервере через Node.js;
- в мобильной разработке;
- в desktop-приложениях через Electron;
- в serverless-функциях.

Современный JavaScript включает:

- модульную систему;
- асинхронное программирование;
- промисы;
- async/await;
- классы;
- стрелочные функции;
- работу с API браузера.

### TypeScript

TypeScript — это надмножество JavaScript, добавляющее статическую типизацию.

Преимущества TypeScript:

- меньше ошибок на этапе разработки;
- удобная поддержка больших проектов;
- автодополнение и подсказки в IDE;
- более понятные интерфейсы между частями системы;
- упрощение рефакторинга.

Сегодня TypeScript стал стандартом во многих коммерческих Web-проектах.

# Современные клиентские (frontend)-фреймворки

Наиболее популярные технологии:

- **React**
- **Vue**
- **Angular**
- **Svelte**

## **React**

React — библиотека для создания пользовательских интерфейсов.

Особенности:

- компонентный подход;
- виртуальный DOM;
- большое сообщество;
- богатая экосистема;
- активное использование в крупных компаниях.

React часто используется вместе с:

- Next.js;
- Redux;
- Zustand;
- React Query;
- Tailwind CSS.

## **Vue**

Vue — прогрессивный frontend-фреймворк.

Особенности:

- низкий порог входа;
- удобная реактивность;
- хорошая документация;
- подходит как для небольших, так и для крупных проектов.

## Angular

Angular — полноценный фреймворк от Google.

Особенности:

- TypeScript «из коробки»;
- строгая структура проекта;
- встроенные инструменты;
- подходит для крупных корпоративных приложений.

## Svelte

Svelte отличается тем, что переносит большую часть работы на этап компиляции.

Преимущества:

- меньше кода в браузере;
- высокая производительность;
- простой синтаксис;
- удобная реактивность.

## Single Page Application и Server Side Rendering

### SPA

SPA — Single Page Application, **одностраничное приложение**.

Особенности:

- страница не перезагружается полностью;
- интерфейс обновляется динамически;
- взаимодействие с сервером происходит через API;
- пользователь получает опыт, похожий на desktop-приложение.

Преимущества:

- высокая интерактивность;
- плавная навигация;

- удобство разработки сложных интерфейсов.

Недостатки:

- большая первоначальная загрузка;
- сложность оптимизации;
- зависимость от JavaScript.

## SSR

SSR — Server Side Rendering, **серверная подготовка страниц**.

Смысл SSR в том, что HTML формируется на сервере и отправляется пользователю уже готовым.

Преимущества:

- лучше SEO;
- быстрее первый показ страницы;
- удобнее для контентных сайтов;
- лучше работает на слабых устройствах.

Популярные SSR-фреймворки:

- Next.js;
- Nuxt;
- Angular Universal;
- SvelteKit.

## SSG

SSG — Static Site Generation.

**Страницы генерируются заранее** на этапе сборки.

Преимущества:

- высокая скорость;
- безопасность;
- простота хостинга;
- низкая стоимость эксплуатации.

Примеры инструментов:

- Next.js;
- Astro;
- Hugo;
- Gatsby;
- Jekyll.

## Серверная (backend) разработка

Backend отвечает за:

- хранение и обработку данных;
- авторизацию;
- бизнес-логику;
- API;
- безопасность;
- интеграцию с внешними сервисами.

**ASP.NET Core занимает важное место среди backend-фреймворков и особенно популярен в корпоративной среде.**

Другие средства (языки) backend-разработки:

- JavaScript / TypeScript;
- Python;
- Java;
- C#;
- Go;
- PHP;
- Ruby;
- Rust.

## Node.js и Backend на JavaScript

Node.js позволил использовать JavaScript на сервере.

Преимущества Node.js:

- единый язык на frontend и backend;
- высокая производительность для I/O-задач;
- большая экосистема npm;
- удобство для real-time приложений.

Популярные backend-фреймворки:

- Express.js;
- NestJS;
- Fastify;
- Кoa.

Node.js часто используется для:

- REST API;
- GraphQL API;
- чатов;
- микросервисов;
- serverless-функций;
- realtime-сервисов.

## **API: REST, GraphQL, gRPC**

### **REST API**

REST — наиболее распространенный стиль построения API.

Особенности:

- работа через HTTP;
- ресурсы представлены URL-адресами;
- используются методы GET, POST, PUT, PATCH, DELETE;
- данные часто передаются в JSON.

Преимущества:

- простота;
- распространенность;

- хорошая поддержка инструментами.

## **GraphQL**

GraphQL позволяет клиенту самому указывать, какие данные ему нужны.

Преимущества:

- меньше лишних данных;
- удобство для сложных интерфейсов;
- единая точка входа;
- строгая схема данных.

Недостатки:

- сложнее кэширование;
- выше сложность настройки;
- нужны дополнительные меры безопасности.

## **gRPC**

gRPC часто используется для взаимодействия микросервисов.

Особенности:

- высокая производительность;
- бинарный формат Protocol Buffers;
- строгие контракты;
- поддержка streaming.

Чаще применяется во внутренней инфраструктуре, а не напрямую в браузере.