React Окружение

WebPack
Vite
env переменные
Eslinter
micro-front-end

Сборщик

Сборщики (или билд-системы) — это инструменты, которые автоматизируют процесс подготовки вашего кода к производству. Они берут ваши исходные файлы (JavaScript, CSS, изображения и другие ресурсы) и преобразуют их в оптимизированные файлы, которые могут быть развернуты на сервере. Сборщики необходимы для React-приложений по нескольким причинам:

- Компиляция и транспиляция
- Упаковка модулей
- Оптимизация производительности
- Управление зависимостями
- Поддержка различных форматов
- Горячая перезагрузка
- Сборка для разных сред

Webpack

Webpack — это популярный сборщик модулей для JavaScript, который предназначен для упаковки модулей с зависимостями в один или несколько файлов, которые могут быть легко развернуты в браузере. Webpack позволяет управлять сложными приложениями и упрощает их поддержку, благодаря сборке всех исходных файлов в оптимизированный для выполнения пакет.

Преимущества Webpack

- Оптимизация: минификация, удаление неиспользуемого кода (tree shaking).
- Разделение кода: позволяет разбить код на несколько бандлов для улучшения производительности (код для разных страниц или функциональности можно загрузить только при необходимости).
- Hot Module Replacement: обновление модулей на лету без перезагрузки страницы.

Недостатки

- Сложность конфигурации: Webpack может быть сложным для понимания новичками, так как содержит много терминов и понятий.
- Время сборки: иногда процесс сборки может занимать значительное время для больших проектов.

https://webpack.js.org

Webpack + React 18

Настройка Webpack для React 18 требует установки нужных пакетов и создания конфигурационного файла для управления сборкой.

- **webpack, webpack-cli:** основной инструмент для сборки и интерфейс командной строки.
- **webpack-dev-server:** сервер разработки с поддержкой автообновления и горячей замены модулей.
- **html-webpack-plugin:** плагин для автоматического создания HTML-файлов и подключения бандлов.
- **babel-loader:** лоадер для использования Babel вместе с Webpack.
- **@babel/core:** основная библиотека Babel, выполняющая трансформацию кода.
- **@babel/preset-env:** пресет для поддержки современного JavaScript в старых браузерах.
- **@babel/preset-react:** пресет для поддержки JSX в React-приложениях.

npm install webpack webpack-cli webpack-dev-server html-webpack-plugin babel-loader @babel/core @babel/preset-env @babel/preset-react --save-dev

Флаг --save-dev нужен для установки пакетов только для режима разработки (для деплоя данные библиотеки не нужны)

Установка React: ***npm install react react-dom***

Структура проекта

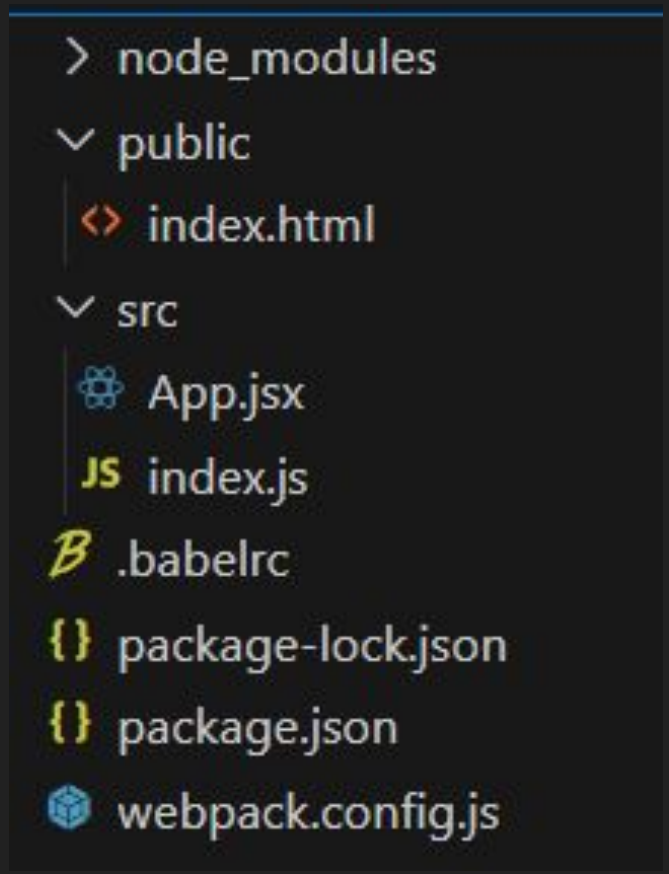
public: содержит статические файлы, такие как HTML.

src: основной исходный код приложения (JavaScript, компоненты).

webpack.config.js: настройки Webpack для сборки и разработки.

package.json: метаданные проекта, зависимости и скрипты.

.babelrc: настройки Babel для трансформации кода.



Настройка Babel

Babel поможет транспилировать современный JavaScript и JSX в код, понятный большинству браузеров.

Чтобы не импортировать React в каждом файле, начиная с React 17, можно использовать "новую JSX-трансформацию".

Важная часть здесь — "runtime": "automatic". Эта опция включает новую трансформацию, позволяя не импортировать React в каждом файле.

```
{
    "presets": [["@babel/preset-env"], ["@babel/preset-react", {"runtime":"automatic"} ]]
}
```

Создание Webpack конфигурации

path — встроенный модуль Node.js для работы с путями файлов.

HtmlWebpackPlugin — плагин для автоматического создания HTML-файла с подключёнными бандлами.

entry — начальная точка для сборки (входной файл).

output — параметры выходного файла

JavaScript и JSX: обрабатываются babel-loader для поддержки современного JS и JSX.

CSS: файлы обрабатываются style-loader и css-loader, чтобы можно было импортировать CSS в JS.

Изображения: обрабатываются через asset/resource для сохранения их в сборку.

Указывает расширения, которые можно опустить при импорте, например, .js и .jsx.

static — указывает, где находятся статические файлы (dist).

hot — включена горячая перезагрузка (HMR).

port — сервер будет доступен по порту 3000.

Создаёт карты исходников для более удобной отладки кода в браузере.

```
const path = require("path");
const HtmlWebpackPlugin = require("html-webpack-plugin");

module.exports = {
  entry: "./src/index.js", // Точка входа
  output: {
    path: path.resolve(__dirname, "dist"),
    filename: "bundle.js",
    clean: true, // Очистка папки dist перед сборкой
  },
  mode: "development", // Режим разработки (можно изменить на production для продакшн-сборки)
  module: {
    rules: [
      {
        test: /\.jsx?$/, // Обработка .js и .jsx файлов
        exclude: /node_modules/,
        use: {
          loader: "babel-loader",
        },
      },
      {
        test: /\.css$/, // Обработка CSS файлов
        use: ["style-loader", "css-loader"],
      },
      {
        test: /\.(png|jpg|jpeg|gif|svg)$/, // Обработка изображений
        type: "asset/resource",
      },
    ],
  },
  resolve: {
    extensions: [".js", ".jsx"], // Поддержка расширений файлов
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: "./public/index.html",
    }),
  ],
  devServer: {
    static: "./dist",
    hot: true, // Поддержка горячей перезагрузки
    port: 3000, // Порт сервера разработки
  },
  devtool: "source-map", // Карты исходников для отладки
};
```

Обновление package.json

Добавим скрипты для удобного запуска сборки и дев-сервера:

Теперь можно использовать команду npm start для запуска сервера разработки, и npm run build для сборки проекта для продакшн.

```
"scripts": {
  "start": "webpack serve --mode development",
  "build": "webpack --mode production"
},
```

Vite

Vite — это современный сборщик приложений для веб-разработки, который был создан Эваном Ю, разработчиком Vue.js. Он отличается высокой производительностью и удобством использования, особенно для современных JavaScript-фреймворков, таких как Vue, React и Svelte.

Мгновенная сборка: Vite использует нативную поддержку ES-модулей в браузерах, что позволяет загружать модули по мере необходимости. Это означает, что Vite не требует полной сборки приложения при каждом изменении — он обрабатывает только измененные файлы, что значительно сокращает время ожидания.

Использование Rollup: Vite использует Rollup для финальной сборки, что позволяет эффективно оптимизировать выходной код. Rollup известен своей способностью удалять неиспользуемый код (tree shaking) и производить компактные бандлы.

Отсутствие предварительной сборки: В отличие от Webpack, который требует предварительной сборки всех модулей перед началом разработки, Vite работает с нативными модулями. Это избавляет от временных затрат на сборку и позволяет разработчикам сразу начать работать.

Эффективное обновление: Vite обеспечивает быстрый механизм горячей перезагрузки модулей (Hot Module Replacement), который заменяет только измененные модули в реальном времени без необходимости полной перезагрузки страницы. Это делает процесс разработки более интерактивным и отзывчивым.

Простота настройки: Vite имеет более простую конфигурацию по сравнению с Webpack, что позволяет быстрее настраивать и запускать проекты. Это уменьшает время, затрачиваемое на начальную настройку проекта.

https://vitejs.dev

Vite + React 18

Сборка проекта React на Vite — это простой и быстрый процесс.

npm create vite@latest my-react-app --template react

Установка шаблона приложения React

Если вы хотите создать проект React на Vite вручную, а не с помощью команды npm create vite, вы можете сделать это следующим образом:

Установка React: ***npm install react react-dom***

Установка Vite и плагина: ***npm install vite @vitejs/plugin-react --save-dev***

Настройка vite

файл конфигурации Vite:

Плагин @vitejs/plugin-react для Vite нужен для обеспечения поддержки специфических возможностей React

- Поддержка JSX
- Поддержка react-refresh
- Поддержка react/jsx-dev-runtime

Добавление скриптов в package.json:

vite preview: Эта команда запускает локальный сервер, который позволяет вам просматривать ваш собранный проект, созданный с помощью npm run build. Это полезно для тестирования вашего приложения перед его развертыванием на продакшене.

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  plugins: [react()],
});
```

```
"scripts": {
  "dev": "vite",
  "build": "vite build",
  "serve": "vite preview"
},
```

Структура проекта

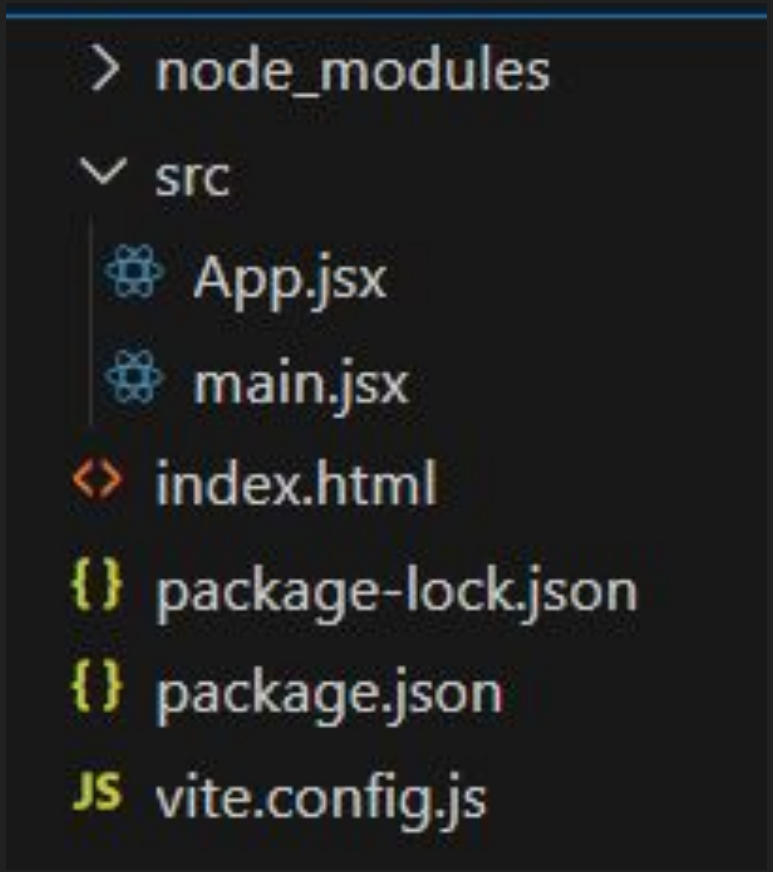
index.html: основной HTML-документ приложения, содержащий корневой элемент, в который будет монтироваться React-приложение.

src: основной исходный код приложения, включая компоненты React (например, App.jsx и main.jsx).

vite.config.js: настройки Vite для сборки и разработки, включая подключение плагинов, таких как @vitejs/plugin-react.

package.json: метаданные проекта, включая название, версию, зависимости и скрипты для запуска разработки, сборки и предпросмотра приложения.

node_modules: папка, содержащая установленные зависимости проекта, управляемые npm.



env - переменные окружения

.env файлы на фронтенде используются для хранения конфиденциальной информации и параметров конфигурации, которые могут меняться в зависимости от окружения (например, разработка, тестирование, продакшн). Они позволяют разработчикам централизованно управлять такими параметрами, как:

- API ключи;
- URL для доступа к серверу;
- Настройки фич (фич-флаги).

Зачем использовать .env файлы на фронтенде

- **Безопасность:** Избегается хардкодинг ключей и конфиденциальной информации в коде.
- **Гибкость:** Удобство при переключении между разными окружениями — достаточно создать .env.development и .env.production.
- **Удобство:** Все параметры конфигурации собраны в одном месте, что упрощает их управление.

Webpack + env

Для интеграции переменных окружения с Webpack потребуется использовать пакеты dotenv и dotenv-webpack

npm install dotenv dotenv-webpack --save-dev

Создайте файл .env в корне вашего проекта. Этот файл будет содержать ваши переменные окружения. Важно, чтобы переменные начинались с REACT_APP_, так как это принято для фронтенд приложений и позволяет легко идентифицировать переменные, которые будут публично доступны.

Настройте Webpack с dotenv-webpack

```
const path = require("path");
const HtmlWebpackPlugin = require("html-webpack-plugin");
const Dotenv = require("dotenv-webpack");

plugins: [
  new HtmlWebpackPlugin({
    template: "./public/index.html",
  }),
  new Dotenv({
    path: "./.env", // путь к вашему .env файлу
  }),
],
```

```
const apiTestKey = process.env.REACT_APP_TEST_KEY;

const App = () => <h1>{apiTestKey}</h1>;

export default App;
```

.env

```
1  REACT_APP_TEST_KEY="test message"
```

Vite + env

В Vite переменные env работают “из под коробки”

В Vite все переменные окружения, которые должны быть доступны в коде фронтенда, должны начинаться с префикса VITE_. Это необходимо для безопасности, так как Vite позволяет различать переменные, которые должны быть публичными, и те, что предназначены только для использования на сервере.

Вы также можете создавать различные .env файлы для разных окружений:

- .env — общий файл для всех окружений.
- .env.development — для окружения разработки.
- .env.production — для окружения продакшн.

При запуске Vite вы можете указать режим с помощью флага --mode.
Например: ***vite --mode production***

Чтобы получить доступ к переменным окружения в коде, используйте объект import.meta.env.
Например: ***const apiUrl = import.meta.env.VITE_API_URL***

```
const App = () => {
  const message = import.meta.env.VITE_TEST_MESSAGE;

  return <h1>{message}</h1>;
};

export default App;
```

.env

.env

```
1  VITE_TEST_MESSAGE="123456_env"
```

ESLinter

ESLint (или "ESLint-er") — это популярный линтер для JavaScript и TypeScript, предназначенный для анализа исходного кода и выявления ошибок, недочетов или проблем с качеством. Он помогает поддерживать единый стиль кода, предотвращать баги и следовать лучшим практикам программирования.

Основные возможности ESLint:

- **Анализ кода на ошибки** — обнаруживает потенциальные баги в коде.
- **Проверка кодстайла** — помогает соблюдать стандарты кодирования (например, расположение скобок, отступы).
- **Настраиваемые правила** — ESLint поддерживает кастомизацию, можно включить или отключить конкретные правила, использовать сторонние конфигурации или написать свои собственные.
- **Автоисправление** — многие ошибки могут быть исправлены автоматически с помощью команды eslint --fix.

ESLint обычно интегрируется с редакторами кода (такими как VS Code), чтобы разработчики могли видеть предупреждения и ошибки прямо при написании кода.

Webpack + ESLinter

Для начала, установите ESLint и необходимые плагины для React:

npm install eslint eslint-webpack-plugin eslint-plugin-react --save-dev

- **eslint** — основной пакет ESLint.
- **eslint-webpack-plugin** — плагин для интеграции ESLint с Webpack.
- **eslint-plugin-react** — плагин для поддержки правил, специфичных для React.

Инициализация ESLint

npx eslint --init

ESLint задаст вам несколько вопросов для настройки, таких как:

- Тип проекта (React).
- Используемый синтаксис (ES6+).
- Формат конфигурационного файла (JSON, YAML, JavaScript).

Это создаст файл .eslintrc.json, который выглядит примерно так:

```
{
  "env": {
    "browser": true,
    "es2021": true
  },
  "extends": [
    "eslint:recommended",
    "plugin:react/recommended"
  ],
  "parserOptions": {
    "ecmaVersion": "latest",
    "sourceType": "module"
  },
  "plugins": [
    "react"
  ],
  "rules": {
    // Можно добавить свои правила или изменить существующие
  }
}
```

Настройка Webpack

Импортируйте плагин в webpack.config.js:

```
const ESLintPlugin = require('eslint-webpack-plugin');
```

Добавьте плагин в массив plugins:

```
module.exports = {
  // ...остальная конфигурация Webpack
  plugins: [
    new ESLintPlugin({
      extensions: ['js', 'jsx'],
      exclude: ['node_modules']
    })
  ],
};
```

Теперь вы можете запускать ESLint командой: ***npx eslint src***

Для более удобной работы интегрируйте ESLint с вашим редактором кода (например, VS Code), установив расширение ESLint, чтобы ошибки отображались в реальном времени.

Vite + ESLinter

Сначала установите ESLint и плагин для поддержки React:

npm install eslint eslint-plugin-react --save-dev

- **eslint** — основной пакет для линтинга.
- **eslint-plugin-react** — плагин для поддержки правил, специфичных для React.

Инициализация ESLint

npx eslint --init

Вам будут заданы вопросы, чтобы настроить ESLint:

- Какой тип проекта вы используете? Выберите "React".
- Какой синтаксис JavaScript используете? Выберите ES6+.
- Какую среду развертывания вы используете? Выберите "Browser".
- Хотите использовать ES Modules? Да.
- Используете ли вы TypeScript? Нет (если используете JavaScript).
- Какую конфигурацию хотите использовать? Выберите "JSON", "YAML" или "JavaScript" по своему предпочтению.

В результате у вас будет файл конфигурации .eslintrc.json, например:

```
{
  "env": {
    "browser": true,
    "es2021": true
  },
  "extends": [
    "eslint:recommended",
    "plugin:react/recommended"
  ],
  "parserOptions": {
    "ecmaVersion": "latest",
    "sourceType": "module"
  },
  "plugins": [
    "react"
  ],
  "rules": {
    // Здесь можно добавить свои правила
  }
}
```

Настройка

Настройка Vite для использования ESLint (опционально)

npm install vite-plugin-eslint --save-dev

Настройка скриптов в package.json

Запуск ESLint: ***npm run lint***

Чтобы видеть ошибки в реальном времени, добавьте расширение ESLint в ваш редактор (например, в VS Code), и ESLint будет подсвечивать ошибки при написании кода.

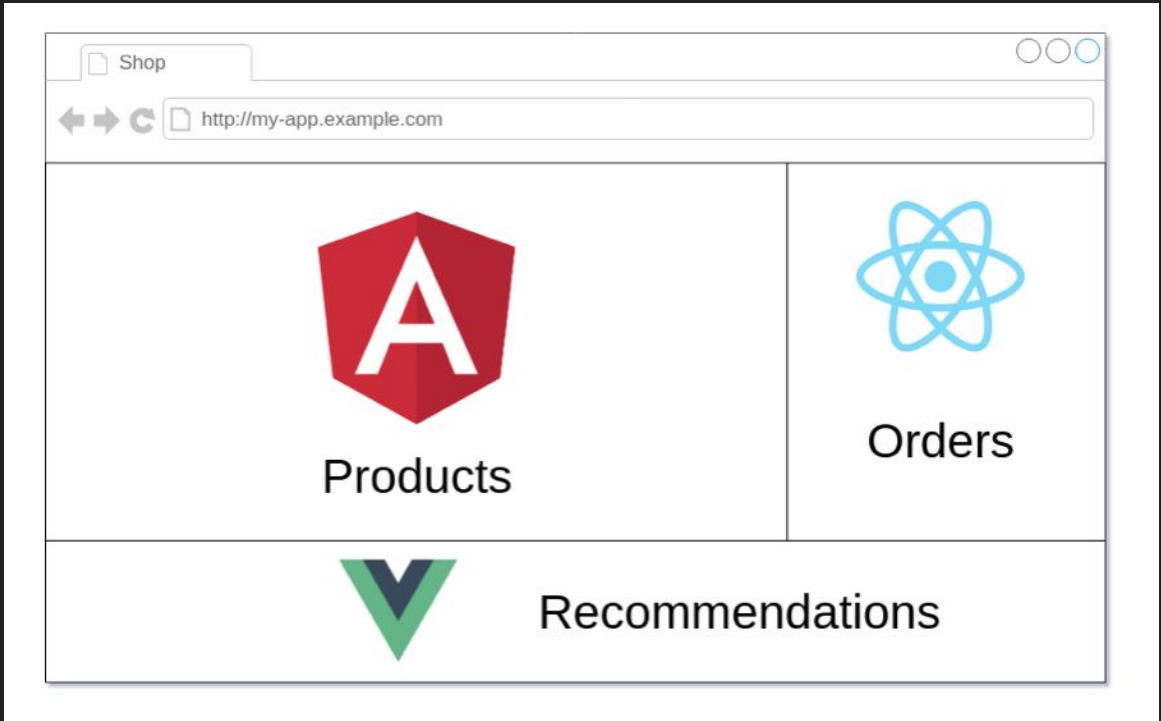
```
"scripts": {
  "lint": "eslint src --ext .js,.jsx"
}
```

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import eslintPlugin from 'vite-plugin-eslint';

export default defineConfig({
  plugins: [
    react(),
    eslintPlugin({
      cache: false,
    }),
  ],
});
```

micro-front-end

Микро-фронтенды (micro frontends) — это архитектурный подход к разработке веб-приложений, при котором фронтенд разбивается на более мелкие, независимые части, каждая из которых разрабатывается и развертывается автономно. Этот подход напоминает микросервисную архитектуру на серверной стороне, но применяется к фронтенду.



Основные идеи микро-фронтендов:

Разделение на независимые модули: Приложение делится на независимые компоненты или модули, каждый из которых разрабатывается разными командами. Эти модули могут быть реализованы с использованием разных технологий и инструментов, если это необходимо.

Автономная разработка: Каждая команда, работающая над своим микро-фронтендом, имеет возможность использовать разные технологии и инструменты, что позволяет выбрать наиболее подходящее решение для конкретной задачи.

Независимое развертывание: Каждый микро-фронтенд может быть развернут независимо от других частей приложения, что позволяет обновлять отдельные модули без необходимости развертывать все приложение.

Общая интеграция: В конечном итоге все микро-фронтенды интегрируются на уровне пользовательского интерфейса, создавая единую, цельную систему. Интеграция может быть выполнена разными способами: с помощью фреймворков, таких как Single-SPA, или через серверные рендеринг и контейнерные приложения.

Примеры реализации включают использование таких фреймворков, как **Single-SPA, Module Federation** в Webpack 5, или подходы на основе **iFrame**.

Плюсы и минусы микро-фронтенда

Преимущества микро-фронтендов:

- **Масштабируемость:** Разные команды могут работать параллельно над своими частями приложения, что ускоряет разработку.
- **Независимость:** Модули могут быть изменены и развернуты независимо, что улучшает устойчивость и гибкость.
- **Поддержка старых технологий:** Можно постепенно мигрировать к новым технологиям, не переписывая сразу все приложение.
- **Гибкость в выборе технологий:** Возможность использовать разные технологии для разных частей приложения.

Недостатки микро-фронтендов:

- **Сложность интеграции:** Интеграция нескольких частей в единое приложение может быть сложной задачей, особенно когда модули используют разные технологии.
- **Оверхед инфраструктуры:** Требуется больше усилий для настройки CI/CD, обеспечения консистентного UX, организации коммуникации между модулями.
- **Повторение зависимостей:** Разные микро-фронтенды могут использовать одни и те же библиотеки, что увеличивает размер конечного пакета.

Ресурсы

оф. документация Webpack - ТЫК

оф. документация Vite - ТЫК

оф. документация ESLint - ТЫК

vite plugin federation (микро-фронтэнд) - ТЫК