JavaScript библиотеки и фреймворки

Библиотеки
Фреймворки
пакетный менеджер

Что такое библиотека

В JavaScript, библиотека — это набор готовых функций, методов и классов, которые упрощают выполнение часто встречающихся задач. Библиотеки позволяют разработчикам использовать повторно используемый код для таких операций, как манипуляции с DOM, обработка событий, выполнение сетевых запросов и многое другое.

Примеры популярных JavaScript-библиотек:

- **jQuery**: Обеспечивает упрощенную работу с DOM, обработку событий, анимацию и AJAX-запросы.
- **Lodash**: Предоставляет утилиты для работы с массивами, объектами и другими структурами данных.
- **Axios**: Библиотека для выполнения HTTP-запросов, обычно используется для работы с API.
- **Moment.js**: Упрощает работу с датами и временем

Как подключать библиотеку

Чтобы подключить библиотеку на фронтенде с использованием нативного JavaScript, есть несколько вариантов:

- Использование CDN (Content Delivery Network)
- Локальное подключение:
- Использование модульного подхода (ES Modules):
- Использование пакетных менеджеров (npm, Yarn) с инструментами сборки

Использование CDN (Content Delivery Network):

Это самый простой способ. Вы можете подключить библиотеку, используя тег <script> и указав URL-адрес библиотеки.

<script src="https://example.com/library.js"></script>

Убедитесь, что вы размещаете тег <script> перед закрывающим тегом </body>, чтобы страница сначала загрузила содержимое, а затем библиотеки.

Локальное подключение:

Если вы скачали библиотеку на свой сервер или локальный компьютер, вы можете подключить ее так же, как и любой другой файл JavaScript.

<script src="/path/to/library.js"></script>

Убедитесь, что путь к файлу корректный относительно вашей HTML-страницы.

Использование модульного подхода (ES Modules):

Если библиотека поддерживает ES Modules, вы можете подключить ее с использованием атрибута type="module".

```
<script type="module">
  import { someFunction } from 'https://example.com/library.js';
  someFunction();
</script>
```

Обратите внимание, что для использования import через URL, библиотека и ваш браузер должны поддерживать ES Modules.

Использование пакетных менеджеров (npm, Yarn) с инструментами сборки:

Если вы используете инструменты сборки, такие как Webpack, Parcel или Rollup, вы можете установить библиотеку через npm или Yarn и импортировать ее в своем JavaScript-коде.

npm install library-name

Затем импортируйте библиотеку в вашем скрипте:

```
import { someFunction } from 'library-name';

someFunction();
```

Пакетный менеджер

Пакетный менеджер — это инструмент, который автоматизирует процесс установки, обновления, конфигурации и удаления программных пакетов. В контексте веб-разработки пакетные менеджеры управляют библиотеками и зависимостями, которые необходимы для работы вашего проекта.

Основные функции пакетного менеджера включают:

- **Установка пакетов:** Позволяет легко устанавливать библиотеки и их зависимости, которые требуются вашему проекту. Например, вместо того чтобы искать и загружать библиотеку вручную, вы можете просто использовать команду для установки.
- **Обновление пакетов:** Пакетные менеджеры могут управлять версиями установленных пакетов и облегчают обновление до новых версий.
- **Удаление пакетов:** Пакетные менеджеры позволяют легко удалять пакеты и их зависимости, которые больше не нужны.
- **Управление зависимостями:** Пакеты часто зависят от других пакетов, и пакетные менеджеры автоматически управляют этими зависимостями, чтобы убедиться, что все необходимые пакеты установлены.

npm

npm (Node Package Manager) — это стандартный пакетный менеджер для среды выполнения Node.js, который используется для управления пакетами JavaScript. npm позволяет разработчикам находить, устанавливать, удалять, обновлять и публиковать пакеты, которые могут содержать как код JavaScript, так и информацию о проекте.

Основные функции npm:

- **Управление пакетами:** npm позволяет устанавливать и удалять пакеты, а также управлять их версиями и зависимостями.
- **Создание и публикация пакетов:** npm предоставляет возможности для создания собственных пакетов и их публикации в публичном или частном реестре.
- **Скрипты npm:** Через файл package.json можно настроить скрипты, которые автоматизируют задачи, такие как сборка, тестирование или развертывание проекта.

Как работать с npm:

Инициализация проекта: ***npm init*** - Эта команда создаст файл package.json, содержащий информацию о вашем проекте и его зависимостях.

Установка пакетов:

- **Локальная установка:** Устанавливает пакет только для текущего проекта и сохраняет его в node_modules. ***npm install package-name***
- **Глобальная установка:** Устанавливает пакет глобально для всей системы. ***npm install -g package-name***
- **Установка пакетов из package.json:** ***npm install*** -Если файл package.json содержит зависимости, команда npm install установит все пакеты, указанные в этом файле.

Управление пакетами

- **Удаление пакета:** ***npm uninstall package-name***
- **Обновление пакета:** ***npm update package-name***

Преимущества npm:

Широкий выбор пакетов: npm предоставляет доступ к огромной библиотеке пакетов и модулей, что упрощает разработку и ускоряет процесс создания приложений.

Автоматическое управление зависимостями: npm автоматически устанавливает зависимости, обеспечивая, чтобы все нужные пакеты были доступны.

Интеграция с экосистемой JavaScript: npm тесно интегрирован с экосистемой Node.js и JavaScript, поддерживая последние версии и стандарты.

Пакетные скрипты: npm позволяет создавать и управлять скриптами для автоматизации задач, что делает его мощным инструментом для DevOps и CI/CD процессов.

Недостатки npm:

Проблемы с производительностью: Ранее npm имел проблемы с производительностью при установке пакетов, особенно в больших проектах. Многие из этих проблем решены в более поздних версиях.

Проблемы с безопасностью: Поскольку npm позволяет любому публиковать пакеты, существует риск установки уязвимых или злонамеренных пакетов.

Конфликты версий: npm позволяет устанавливать разные версии одного и того же пакета в различных частях проекта, что иногда может вызывать конфликты и ошибки.

Иногда сложно отслеживать зависимости: В больших проектах с множеством зависимостей иногда бывает трудно понять, какие пакеты зависят от других и как обновление одного пакета может повлиять на весь проект.

Публикация библиотеки на npm

Создайте учетную запись на npm: Если у вас ее еще нет, зарегистрируйтесь на официальном сайте npm.

Войдите в свою учетную запись через терминал: *npm login*

Убедитесь, что имя вашего пакета уникально, иначе npm не позволит его опубликовать. Для публикации используйте команду: *npm publish*

После успешной публикации вы сможете найти свой пакет на npmjs.com.

Управление и обновление пакета

Обновление пакета:

Если вы вносите изменения в свой пакет, не забудьте обновить версию в package.json. Версии должны соответствовать семантическому версионированию.

После этого повторно публикуйте пакет:

npm version patch # или minor/major, в зависимости от изменений

npm publish

Удаление пакета:

Если вам нужно удалить пакет, используйте команду:

npm unpublish package-name

Учтите, что unpublish нельзя использовать для версий, которые были опубликованы более 24 часов назад.

Рекомендации по публикации библиотеки:

Документация: Обязательно предоставьте документацию по использованию вашей библиотеки, чтобы другие разработчики могли легко понять, как с ней работать.

Лицензия: Убедитесь, что вы добавили файл лицензии (например, LICENSE) в ваш проект. Обычной практикой является использование лицензии MIT.

README: Добавьте файл README.md, чтобы описать, что делает ваш пакет, как его установить и использовать.

Настройка webpack для нативного HTML + JS + CSS

Создайте структуру проекта:

mkdir my-webpack-project (создать папку проекта)

cd my-webpack-project (открыть проект)

mkdir src (создать папку src)

touch src/index.html src/index.js src/styles.css (в папке src создать файлы)

Установите Webpack и необходимые зависимости:

npm init -y (устанавливаем пакетный менеджер)

npm install --save-dev webpack webpack-cli webpack-dev-server html-webpack-plugin css-loader style-loader (устанавливаем все необходимое для webpack-а

Создайте конфигурационный файл Webpack:

Создайте файл webpack.config.js в корневой директории проекта и добавьте следующее:

Настройте скрипты в package.json:

Измените раздел scripts в package.json, чтобы включить команды для запуска Webpack:

```
"scripts": {
  "build": "webpack",
  "start": "webpack serve"
}
```

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

module.exports = {
  mode: "development",
  entry: "./src/index.js",
  output: {
    filename: "bundle.js",
    path: path.resolve(__dirname, "dist"),
  },
  module: {
    rules: [
      {
        test: /\.css$/i,
        use: ["style-loader", "css-loader"],
      },
    ],
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: "./src/index.html",
    }),
  ],
  devServer: {
    static: path.resolve(__dirname, "dist"),
    port: 3000,
  },
};
```

Контент в файлы:

html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Webpack Project</title>
</head>
<body>
  <h1>Hello Webpack!</h1>
  <script src="bundle.js"></script>
</body>
</html>
```

javascript

```javascript
import './styles.css';

console.log('Hello from JavaScript!');
```

css

```css
body {
  background-color: #f0f0f0;
  color: #333;
  font-family: Arial, sans-serif;
}
```

Запуск проекта:

Для разработки:

npm start

Для сборки проекта в продакшн:

npm run build

Что такое фреймворк

Фреймворк в JavaScript — это более структурированный и комплексный инструмент, чем библиотека. Фреймворки предоставляют базовую архитектуру и набор инструментов, которые помогают разработчикам создавать приложения, следуя определенным принципам и шаблонам. Они обычно включают в себя библиотеки, шаблоны, утилиты и средства управления состоянием приложения.

Основные особенности фреймворков:

Архитектурный каркас: Фреймворки предлагают стандартный способ организации кода, что упрощает его понимание и поддержку.

Интеграция: Они часто включают в себя интегрированные решения для различных аспектов разработки, таких как маршрутизация, управление состоянием, взаимодействие с сервером и работа с данными.

Обязательные соглашения: Фреймворки обычно предполагают следование определенным соглашениям и шаблонам проектирования, что может ограничивать гибкость, но взамен упрощает разработку и поддержку.

Примеры популярных JavaScript-фреймворков:

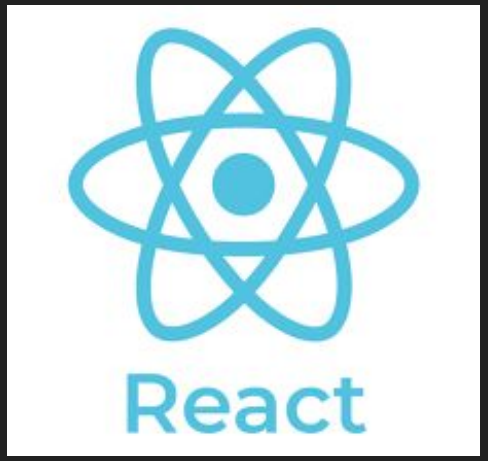
React: Несмотря на то, что часто называется библиотекой, React может быть частью фреймворка, если используется с такими инструментами, как React Router и Redux.

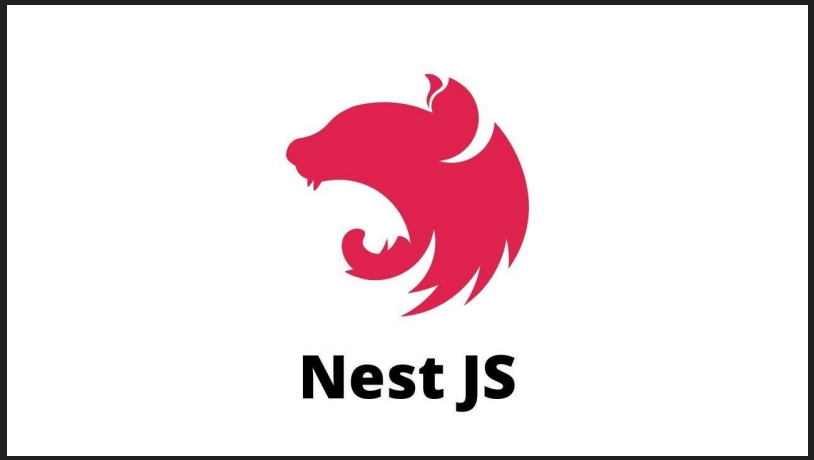
Angular: Комплексный фреймворк от Google, который включает в себя всё, что нужно для разработки SPA (одностраничных приложений).

Vue.js: Прогрессивный фреймворк, который можно использовать как для создания простых компонентов, так и для разработки сложных приложений.

Express.js: Фреймворк для Node.js, предназначенный для создания веб-приложений и API.

Фреймворки помогают стандартизировать и ускорить процесс разработки, предоставляя проверенные решения и лучшие практики.





EDET EMMANUEL ASUQUO

Фреймворки на фронт-энде предоставляют:

Компонентный подход: Позволяет разбивать интерфейс на переиспользуемые компоненты. Это упрощает разработку, тестирование и поддержку приложения.

Управление состоянием: Фреймворки часто включают инструменты для управления состоянием приложения (например, Redux для React или Vuex для Vue), что облегчает работу с данными и их синхронизацию.

Маршрутизация: Предоставляют решения для управления маршрутами и навигацией в приложении. Это позволяет легко организовывать и изменять URL-адреса и отображать разные компоненты на основе текущего маршрута.

Реактивность: Некоторые фреймворки (например, Vue) предлагают реактивные системы, которые автоматически обновляют представление при изменении данных.

Интеграция с инструментами сборки: Включают поддержку инструментов сборки, таких как Webpack или Vite, для упрощения процесса сборки и оптимизации приложений.

Модульная структура: Поддержка модульной архитектуры для упрощения организации и управления кодом.

Поддержка шаблонов: Обеспечивают средства для создания шаблонов (например, JSX для React или шаблоны для Vue), что упрощает генерацию HTML-кода и связывание его с данными.

Инструменты для тестирования: Включают встроенные или совместимые инструменты для тестирования компонентов и функциональности приложения.

Библиотеки и плагины: Предлагают расширяемость через библиотеки и плагины для добавления дополнительного функционала.

Поддержка современных стандартов: Обеспечивают поддержку последних веб-стандартов и технологий, таких как ES6+ и CSS-препроцессоры.

Ресурсы

node js - ТЫК

npm - ТЫК

WebPack - ТЫК

lodash, библиотека типовых вспомогательных функций - ТЫК

axios, библиотека для работы с HTTP запросами - ТЫК

JQuery - библиотека для упрощения работы JavaScript-а - ТЫК

Документация react-а на русском - ТЫК

официальный сайт react-а - ТЫК