تعریف مسئله

طرح شماتیک دید کلی شرکت در لحظه برای مهندسان نرم‌افزار

ما به عنوان مهندسان نرم‌افزار در یک شرکت تولیدی بزرگ با چندین خط تولید، توده‌های مواد خام، انبار و حمل و نقل داخلی (خودکار و دستی)، به یک طرح شماتیک تک‌نما نیاز داریم که دیدگاه‌های لحظه‌ای از عملیات را ارائه دهد. ما در حال حاضر نرم‌افزارهایی برای نگهداری، تولید و مدیریت مواد خام داریم. هدف ما ادغام این سیستم‌ها در یک داشبورد واحد است.

در اینجا چند گزینه برای دستیابی به این هدف، با در نظر گرفتن سهولت طراحی، حداقل تعامل توسعه‌دهنده، اتصال داده‌های لحظه‌ای و نگهداری کم، ارائه شده است:

گزینه 1: داشبورد بصری با ویجت‌های تعبیه‌شده

- **توضیحات:** یک پلتفرم داشبورد مرکزی (به عنوان مثال، Power BI، Tableau، Grafana) به عنوان نمای اصلی عمل می‌کند. هر منطقه (خط تولید، توده مواد خام، منطقه ذخیره‌سازی و غیره) توسط یک ویجت بصری نمایش داده می‌شود. این ویجت‌ها داده‌ها را از سیستم‌های نرم‌افزاری موجود از طریق API یا اتصالات مستقیم پایگاه داده تعبیه می‌کنند.
- **معیارها:**
  - **سهولت طراحی:** بالا - رابط‌های کشیدن و رها کردن و ویجت‌های از پیش ساخته شده، ایجاد داشبورد را ساده می‌کنند.
  - **تعامل توسعه‌دهنده:** متوسط - تنظیم اولیه به کار توسعه‌دهنده برای پیکربندی اتصالات API و تبدیل داده‌ها نیاز دارد. تغییرات بعدی اغلب توسط مدیران داشبورد قابل انجام است.
  - **اتصال لحظه‌ای:** بالا - اکثر ابزارهای داشبورد از جریان‌های داده لحظه‌ای و فواصل تازه‌سازی پشتیبانی می‌کنند.
  - **نگهداری:** کم - تغییرات در طرح‌بندی یا تجسم به راحتی در داشبورد پیاده‌سازی می‌شوند. تغییرات API در سیستم‌های زیربنایی به دخالت توسعه‌دهنده نیاز دارد.
- **مثال:** می‌توان از Grafana با پلاگین‌ها برای اتصال به پایگاه‌های داده یا APIهای سیستم‌های تولید، نگهداری و مدیریت مواد خام استفاده کرد. ویجت‌ها معیارهای کلیدی مانند خروجی تولید، وضعیت دستگاه، سطوح سهام و فعالیت حمل و نقل را نشان می‌دهند.

گزینه 2: برنامه وب سفارشی با نقشه تعاملی

- **توضیحات:** یک برنامه وب سفارشی توسعه داده می‌شود که دارای یک نقشه تعاملی از تاسیسات است. هر عنصر روی نقشه (خط تولید، منطقه ذخیره‌سازی، کامیون و غیره) یک شیء قابل کلیک است که اطلاعات دقیق را از سیستم‌های نرم‌افزاری مربوطه نمایش می‌دهد.

- **معیارها:**
  - **سهولت طراحی:** متوسط - در مقایسه با داشبورد به تلاش توسعه بیشتری نیاز دارد. ادغام نقشه و منطق تجسم داده باید پیاده‌سازی شوند.
  - **تعامل توسعه‌دهنده:** بالا - درگیری قابل توجه توسعه‌دهنده برای توسعه اولیه و نگهداری مداوم مورد نیاز است.
  - **اتصال لحظه‌ای:** متوسط تا بالا - از طریق WebSockets یا رویدادهای ارسال شده از سرور قابل دستیابی است، اما نیاز به پیاده‌سازی دقیق دارد.
  - **نگهداری:** متوسط - تغییرات در نقشه، داده‌های نمایش داده شده یا ادغام‌ها به دخالت توسعه‌دهنده نیاز دارند.
- **مثال:** می‌توان از یک چارچوب جاوا اسکریپت (به عنوان مثال، Vue.js، Angular، React) برای ساخت برنامه استفاده کرد. یک کتابخانه نقشه‌برداری (به عنوان مثال، Leaflet، Google Maps) نقشه تعاملی را ارائه می‌دهد. داده‌ها از سیستم‌های موجود از طریق APIهای REST واکشی می‌شوند.

گزینه 3: رویکرد ترکیبی (داشبورد + نقشه)

- **توضیحات:** مزایای هر دو رویکرد را ترکیب می‌کند. یک داشبورد یک نمای کلی از معیارهای کلیدی ارائه می‌دهد، در حالی که یک نقشه تعاملی تعبیه‌شده امکان بررسی دقیق مناطق یا تجهیزات خاص را فراهم می‌کند.
- **معیارها:**
  - **سهولت طراحی:** متوسط - برای ادغام نقشه به مقداری تلاش توسعه نیاز است، اما از پلتفرم داشبورد برای نمای کلی استفاده می‌کند.
  - **تعامل توسعه‌دهنده:** متوسط - نسبت به یک برنامه کاملاً سفارشی، توسعه‌دهنده کمتری درگیر است، اما بیشتر از یک رویکرد داشبورد خالص.
  - **اتصال لحظه‌ای:** متوسط تا بالا - از طریق ترکیبی از ویژگی‌های داشبورد و تکنیک‌های ادغام نقشه قابل دستیابی است.
  - **نگهداری:** متوسط - مشابه رویکرد ترکیبی، با ملاحظات نگهداری برای داشبورد و کامپوننت نقشه.
- **مثال:** می‌توان از یک پلتفرم داشبورد به عنوان رابط اصلی استفاده کرد. یک نقشه تعاملی (ساخته شده با یک کتابخانه نقشه‌برداری) می‌تواند در یک پنل خاص تعبیه شود.

توصیه:

برای سناریوی ما، **داشبورد بصری با ویجت‌های تعبیه‌شده (گزینه 1)** بهترین تعادل را بین سهولت طراحی، حداقل تعامل توسعه‌دهنده، اتصال لحظه‌ای و نگهداری کم ارائه می‌دهد. این به ما امکان می‌دهد تا به سرعت یک نمای کلی جامع بدون سربار توسعه قابل توجه ایجاد کنیم. می‌توانیم با این رویکرد شروع کنیم و در صورت نیاز در آینده، راه‌حل‌های پیچیده‌تری (مانند رویکرد ترکیبی) را بررسی کنیم. تمرکز بر APIهای خوش تعریف برای سیستم‌های موجود ما برای هر یک از این گزینه‌ها بسیار مهم است.

معیارهای عمومی

معیار	توضیحات	سطح اولویت	دلیل
مقیاس‌پذیری	توانایی مقیاس‌بندی به صورت افقی (افزودن کاربران یا خطوط تولید بیشتر) و عمودی (مدیریت داده‌های پیچیده‌تر) بدون کاهش قابل توجه عملکرد.	بالا	سیستم شما باید بتواند حجم زیادی از داده‌ها و به‌روزرسانی‌های لحظه‌ای متعدد از منابع مختلف (خطوط تولید، کامیون‌ها، انبار) را مدیریت کند.
پردازش داده‌های لحظه‌ای	قابلیت پردازش و نمایش داده‌ها در زمان واقعی با حداقل تأخیر.	بالا	این نیاز اصلی پروژه شماست، جایی که تصمیمات باید بر اساس اطلاعات به‌روز از بخش‌های مختلف شرکت گرفته شوند.
ادغام با سیستم‌های موجود	میزان سهولت ادغام سیستم جدید با برنامه‌های کاربردی مبتنی بر وب موجود شما (به عنوان مثال، تولید، نگهداری، مدیریت مواد خام).	بالا	راه حل باید به طور یکپارچه با سیستم‌های نرم‌افزاری موجود کار کند و از جریان روان داده‌ها در همه بخش‌ها بدون ایجاد اختلال اطمینان حاصل کند.
رابط کاربری (UI) / تجربه کاربری (UX)	سهولت استفاده و وضوح در ارائه داده‌ها به ذینفعان مختلف (مدیریت، عملیات و غیره).	متوسط	یک رابط کاربری/تجربه کاربری کاربرپسند برای تصمیم‌گیری سریع ضروری است. با این حال، یک دید واضح از داده‌ها می‌تواند بر تصاویر پیچیده در صورت اولویت داشتن عملکرد غلبه کند.
دقت و سازگاری داده‌ها	اطمینان حاصل کنید که داده‌های جمع‌آوری شده از منابع مختلف (خطوط خودکار، کامیون‌ها، انبار) دقیق و سازگار در سراسر سیستم‌ها هستند.	بالا	اگر داده‌ها ناسازگار یا نادرست باشند، می‌تواند منجر به تصمیم‌گیری ضعیف و ناکارآمدی‌های عملیاتی شود.
بهره‌وری هزینه	هزینه کل پیاده‌سازی و نگهداری مداوم، شامل توسعه نرم‌افزار، زیرساخت و آموزش.	متوسط	در حالی که مهم است، هزینه باید در برابر معیارهای دیگر متعادل شود، به ویژه برای راه حلی که از عملکردهای حیاتی تجاری پشتیبانی می‌کند.

معیار	توضیحات	سطح اولویت	دلیل
امنیت و کنترل دسترسی	اطمینان از اینکه داده‌ها امن هستند و فقط پرسنل مجاز می‌توانند به اطلاعات خاصی دسترسی داشته باشند (به عنوان مثال، جزئیات تولید یا گزارش‌های نگهداری).	بالا	حفاظت از داده‌های شرکت و کنترل دسترسی به اطلاعات حساس برای یکپارچگی و حریم خصوصی عملیاتی بسیار مهم است.
انعطاف‌پذیری و سفارشی‌سازی	توانایی انطباق سیستم با نیازهای آینده، مانند افزودن خطوط تولید جدید یا گنجاندن فناوری‌های جدید (به عنوان مثال، هوش مصنوعی برای نگهداری پیش‌بینی‌کننده).	متوسط	با رشد و تکامل شرکت، سیستم باید بتواند بدون نیاز به اصلاحات اساسی، خود را تطبیق دهد.
نظارت و گزارش‌دهی سیستم	توانایی تولید آسان گزارش‌ها یا داشبوردهایی که وضعیت سیستم، شاخص‌های کلیدی عملکرد (KPI) و وضعیت کلی سیستم را نشان می‌دهند.	بالا	این به شناسایی گلوگاه‌ها، اطمینان از عملکرد سیستم مطابق انتظار و ارائه دید به عملیات مختلف کمک می‌کند.
قابلیت اطمینان و تحمل خطا	توانایی سیستم برای مدیریت ظریف شکست‌ها و ادامه کار یا بازیابی سریع از مشکلات.	بالا	در دسترس بودن بالا برای عملیات مداوم، به ویژه در یک محیط تولیدی که در آن خرابی می‌تواند منجر به خسارات قابل توجهی شود، بسیار مهم است.
ذخیره‌سازی و بازیابی داده‌ها	مدیریت کارآمد حجم زیادی از داده‌ها، با ذخیره‌سازی بهینه داده‌ها و بازیابی سریع برای تجزیه و تحلیل.	بالا	داده‌های خطوط تولید، کامیون‌ها و نگهداری باید به طور کارآمد برای اهداف تجزیه و تحلیل و گزارش‌دهی ذخیره شوند.
سازگاری پشته فناوری	فناوری انتخابی باید با تخصص تیم شما همسو باشد (به عنوان مثال، NET.، پلتفرم‌های ابری، چارچوب‌های فرانت‌اند مانند React).	بالا	این اطمینان می‌دهد که توسعه بدون شکاف‌های دانش قابل توجه به آرامی پیش می‌رود و از قابلیت‌های تیم موجود استفاده می‌کند.
نگهداری و پشتیبانی	سهولت نگهداری و ارتقاء سیستم در طول زمان، شامل رفع	متوسط	سیستم باید در طول چرخه عمر خود قابل نگهداری باشد، با حداقل خرابی و پشتیبانی طولانی مدت

معیار	توضیحات	سطح اولویت	دلیل
	اشکالات، درخواست‌های ویژگی و نگهداری عمومی سیستم.		برای رفع اشکالات و بهبود ویژگی‌ها.
تجسم داده	قابلیت نمایش داده‌های عملیاتی پیچیده در یک فرمت تصویری آسان برای درک (به عنوان مثال، داشبورد، نمودار، نقشه).	بالا	بازخورد بصری در زمان واقعی کلید شناسایی سریع مسائل در بخش‌های مختلف (تولید، انبار، حمل و نقل) خواهد بود.
عملکرد	توانایی سیستم برای مدیریت جریان‌های همزمان داده از چندین خط تولید، انبار و کامیون با حداقل تأخیر.	بالا	یک سیستم کند هدف نظارت در زمان واقعی را از بین می‌برد.
پشتیبانی فروشنده (در صورت خارجی)	در دسترس بودن پشتیبانی فنی و منابع جامعه در صورتی که قصد دارید از سیستم‌ها یا پلتفرم‌های شخص ثالث استفاده کنید.	متوسط	پشتیبانی فروشنده حیاتی است، اما شما باید اطمینان حاصل کنید که تیم شما می‌تواند بیشتر توسعه را به صورت داخلی انجام دهد یا در صورت نیاز به پشتیبانی دسترسی داشته باشد.

گزینه‌های عمومی

1. سیستم‌های SCADA (کنترل نظارتی و اکتساب داده)

سهولت طراحی: متوسط **حداقل تعامل توسعه‌دهنده:** متوسط **یکپارچه‌سازی داده‌های لحظه‌ای:** بالا **زمان نگهداری:** متوسط

سیستم‌های SCADA به طور گسترده در محیط‌های صنعتی برای نظارت و کنترل فرآیندها استفاده می‌شوند. آنها می‌توانند تجسم داده‌های لحظه‌ای، قابلیت‌های کنترلی و ثبت داده‌های تاریخی را فراهم کنند. ادغام سیستم‌های SCADA با نرم‌افزارهای موجود ممکن است به مقداری تلاش توسعه نیاز داشته باشد، اما قابل مدیریت است.

2. پلتفرم‌های اینترنت اشیاء صنعتی (IIoT)

سهولت طراحی: بالا **حداقل تعامل توسعه‌دهنده:** بالا **یکپارچه‌سازی داده‌های لحظه‌ای:** بالا **زمان نگهداری:** کم

پلتفرم‌هایی مانند Microsoft Azure IoT یا Amazon AWS IoT راه‌حل‌های آماده‌ای را برای اتصال دستگاه‌ها و جمع‌آوری داده‌های لحظه‌ای ارائه می‌دهند. آنها اغلب با داشبوردهای بصری و ابزارهای تحلیلی ارائه می‌شوند. این پلتفرم‌ها نیاز به توسعه گسترده را به حداقل می‌رسانند و برای مقیاس‌پذیری و سهولت نگهداری طراحی شده‌اند.

3. داشبورد سفارشی مبتنی بر وب

سهولت طراحی: متوسط **حداقل تعامل توسعه‌دهنده:** کم **یکپارچه‌سازی داده‌های لحظه‌ای:** متوسط **زمان نگهداری:** متوسط

توسعه یک داشبورد سفارشی مبتنی بر وب به شما امکان می‌دهد راه حل را دقیقاً مطابق با نیازهای خود تنظیم کنید. با استفاده از فناوری‌های مدرن وب مانند React، Node.js و WebSocket، می‌توانید یک داشبورد تعاملی و لحظه‌ای ایجاد کنید که منابع داده مختلف را یکپارچه می‌کند. در حالی که ممکن است به توسعه اولیه بیشتری نیاز داشته باشد، اما کنترل کاملی بر طراحی و عملکرد به شما می‌دهد.

4. سیستم ERP با افزونه‌های لحظه‌ای

سهولت طراحی: متوسط **حداقل تعامل توسعه‌دهنده:** متوسط **یکپارچه‌سازی داده‌های لحظه‌ای:** متوسط **زمان نگهداری:** متوسط

سیستم‌های برنامه‌ریزی منابع سازمانی (ERP) مانند Oracle، SAP و Microsoft Dynamics راه‌حل‌های جامعی را برای مدیریت فرآیندهای تجاری ارائه می‌دهند. برخی از سیستم‌های ERP دارای افزونه‌ها یا ماژول‌های لحظه‌ای هستند که می‌توانند برای تجسم داده‌های لحظه‌ای یکپارچه شوند. این سیستم‌ها اغلب پشتیبانی و مستندات گسترده‌ای را ارائه می‌دهند که پیاده‌سازی را روان‌تر می‌کند.

5. فناوری دوقلوی دیجیتال (Digital Twin)

سهولت طراحی: متوسط **حداقل تعامل توسعه‌دهنده:** متوسط **یکپارچه‌سازی داده‌های لحظه‌ای:** بالا **زمان نگهداری:** متوسط

دوقلوهای دیجیتال یک نسخه مجازی از دارایی‌ها و فرآیندهای فیزیکی ایجاد می‌کنند و امکان نظارت و شبیه‌سازی در زمان واقعی را فراهم می‌کنند. ادغام دوقلوهای دیجیتال با حسگرهای اینترنت اشیاء و پلتفرم‌های تجزیه و تحلیل داده می‌تواند نمای جامعی از خطوط تولید، مدیریت مواد خام و حمل و نقل داخلی ارائه دهد. در حالی که تنظیمات ممکن است به مقداری تلاش اولیه نیاز داشته باشد، مزایای نظارت در زمان واقعی و تجزیه و تحلیل پیش‌بینی کننده قابل توجه است.

جدول خلاصه

گزینه	سهولت طراحی	حداقل تعامل توسعه‌دهنده	یکپارچه‌سازی داده‌های لحظه‌ای	زمان نگهداری
سیستم‌های SCADA	متوسط	متوسط	بالا	متوسط

گزینه	سهولت طراحی	حداقل تعامل توسعه‌دهنده	یکپارچه‌سازی داده‌های لحظه‌ای	زمان نگهداری
پلتفرم‌های اینترنت اشیاء صنعتی	بالا	بالا	بالا	کم
داشبورد سفارشی مبتنی بر وب	متوسط	کم	متوسط	متوسط
سیستم ERP با افزونه‌های لحظه‌ای	متوسط	متوسط	متوسط	متوسط
فناوری دوقلوی دیجیتال	متوسط	متوسط	بالا	متوسط

هر یک از این گزینه‌ها دارای نقاط قوت و چالش‌های خاص خود هستند. انتخاب شما به نیازهای خاص، زیرساخت موجود و بودجه شما بستگی دارد.

ابزارها/رویکردهای وب پالایش شده

- SVG / محتوای ثابت
- D3.js / Chart.js
- کتابخانه‌های مبتنی بر WebGL: PixiJS / Three.js
- Grafana / Power BI
- Leaflet برای نقشه‌ها
- D3.js + Leaflet برای نقشه‌ها

مقایسه کلی

در اینجا یک جدول مقایسه برای جایگزین‌های رویکرد فعلی شما (فایل‌های SVG متصل شده با تلاش توسعه‌دهنده) آورده شده است، که گزینه‌های مختلف را در جنبه‌های کلیدی مختلف ارزیابی می‌کند:

معیار	رویکرد فعلی (SVG + تلاش توسعه)	جایگزین 1 (React + D3.js/Chart.js)	جایگزین 2 (کتابخانه‌های مبتنی بر WebGL: PixiJS/Three.js)	جایگزین 3 (Grafana/Power BI)	جایگزین 4 (Leaflet for Maps + React)
مقیاس‌پذیری	مقیاس‌بندی با افزایش تعداد عناصر تحت نظارت دشوار است. ادغام دستی SVGها دشوار می‌شود.	بسیار مقیاس‌پذیر. React و D3.js می‌توانند مجموعه‌های داده بزرگ را به طور موثرتری مدیریت کنند و هندلینگ بهتری داشته باشند.	بسیار مقیاس‌پذیر برای نیازهای با کارایی بالا، به ویژه هنگام رندر کردن مجموعه‌های داده پیچیده یا بزرگ در زمان واقعی.	به خوبی مقیاس‌بندی می‌شود، اما معمولاً برای داشبوردهای نظارتی مناسب‌تر است تا تجسم‌های سفارشی.	مقیاس‌پذیر برای نقشه‌های تعاملی، به ویژه اگر داده‌های مکانی مورد نیاز باشد.
پردازش داده‌های لحظه‌ای	به‌روزرسانی‌های لحظه‌ای چالش‌برانگیز است. نیاز به هندلینگ دستی SVGها برای تغییرات وضعیت دارد.	به‌روزرسانی‌های لحظه‌ای را می‌توان به راحتی با استفاده از React با مدیریت وضعیت (مانند Redux) مدیریت کرد.	به‌روزرسانی‌های لحظه‌ای را با رندر و انیمیشن WebGL مدیریت می‌کند. برای تجسم‌های با کارایی بالا مناسب است.	برای نظارت و تجسم لحظه‌ای جریان‌های داده طراحی شده است.	می‌تواند به‌روزرسانی‌های لحظه‌ای را مدیریت کند، به ویژه برای داده‌های جغرافیایی و مبتنی بر مکان مفید است.
ادغام با سیستم‌های موجود	ادغام سفارشی ساخته شده است و می‌تواند با افزایش پیچیدگی، زمان‌بر شود.	ادغام آسان با برنامه‌های وب مدرن، به خصوص اگر از یک معماری مبتنی بر کامپوننت مانند React استفاده شود.	به خوبی با راه‌حل‌های سفارشی ادغام می‌شود، اما ممکن است برای ادغام با بک‌اند های سنتی به تلاش بیشتری نیاز داشته باشد.	می‌تواند با پایگاه‌های داده، APIها و سیستم‌های نظارتی مختلف ادغام شود، اما معمولاً به ابزارهای شخص ثالث برای ادغام عمیق نیاز دارد.	یکپارچه با نقشه‌ها ادغام می‌شود و می‌تواند داده‌ها را از APIها برای به‌روزرسانی‌های پویا بیرون بکشد، اگرچه ممکن است برای ادغام کامل برنامه به پیکربندی

معیار	رویکرد فعلی (SVG + تلاش توسعه)	جایگزین 1 (React + D3.js/Chart.js)	جایگزین 2 (کتابخانه‌های مبتنی بر WebGL: PixiJS/Three.js)	جایگزین 3 (Power BI/Grafana)	جایگزین 4 (React + Leaflet for Maps)
					اضافی نیاز باشد.
UI/UX	تعامل و انعطاف‌پذیری طراحی بدون تلاش قابل توجه محدود است.	انعطاف‌پذیری بالا برای UIهای مدرن و پویا با تجسم‌های غنی و تعاملی.	برای تجسم‌های بسیار تعاملی و پویا عالی است، اما به دانش پیشرفته برای بهینه‌سازی کامل UX نیاز دارد.	کامپوننت‌های صیقلی و از پیش ساخته شده برای تجسم فراهم می‌کند، اگرچه ممکن است سفارشی‌سازی کامل را محدود کند.	برای تجسم داده‌های جغرافیایی با رابط‌های مبتنی بر نقشه پاسخگو عالی است.
سفارشی‌سازی	بسیار قابل تنظیم است، اما با رشد سیستم می‌تواند دشوار شود.	بسیار قابل تنظیم است و امکان کنترل دقیق بر روی تجسم‌ها و طرح‌بندی را فراهم می‌کند.	برای عملکرد بصری و تجسم‌های پیچیده بسیار قابل تنظیم است، اما به تخصص توسعه بیشتری نیاز دارد.	سفارشی‌سازی محدود برای گردش‌های کاری سفارشی، اما در داشبورد انعطاف‌پذیری ارائه می‌دهد.	برای نقشه‌ها و تجسم‌های مبتنی بر مکان قابل تنظیم است، اما ممکن است برای سفارشی‌سازی کامل برنامه به کار بیشتری نیاز باشد.
دقت و سازگاری داده‌ها	نیاز به کدنویسی سفارشی برای اطمینان از همگام‌سازی و سازگاری در سراسر سیستم‌ها دارد.	جریان داده در زمان واقعی با به‌روزرسانی‌های قابل پیش‌بینی، دقت و سازگاری را تضمین می‌کند.	دقت در زمان واقعی با کنترل دقیق بر جریان داده و رندر.	سازگاری داده از طریق منابع داده خارجی مدیریت می‌شود، که بسته به بک‌اند می‌تواند بهینه شود.	می‌تواند برای همگام‌سازی فیدهای داده د زمان واقعی، به ویژه با داده‌های مکانی طراحی شود.
بهره‌وری هزینه	هزینه اولیه کم است، اما برای	هزینه توسعه اولیه متوسط،	هزینه اولیه بالاتر به دلیل	هزینه‌ها می‌توانند با	رایگان و متن‌باز برای اکثر موارد

معیار	رویکرد فعلی (SVG + تلاش توسعه)	جایگزین 1 (React + D3.js/Chart.js)	جایگزین 2 (کتابخانه‌های مبتنی بر WebGL: PixiJS/Three.js)	جایگزین 3 (Grafana/Power BI)	جایگزین 4 (Leaflet for Maps + React)
	نگهداری و گسترش نیاز به زمان قابل توجهی توسعه‌دهنده دارد.	اما کامپوننت‌های قابل استفاده مجدد می‌توانند زمان توسعه بلندمدت را کاهش دهند.	بهینه‌سازی عملکرد، اما می‌تواند تجسم‌های پیچیده را بدون مشکلات عملکرد قابل توجهی مدیریت کند.	استفاده مقیاس شوند، به ویژه با پلتفرم‌های مبتنی بر ابر (به عنوان مثال، Power BI) و هزینه‌های مجوز.	استفاده، اگرچه ممکن است برای ادغام به توسعه اضافی نیاز داشته باشد.
عملکرد	عملکرد محدود با SVGها، به ویژه با افزایش پیچیدگی داده.	عملکرد عالی برای مجموعه‌های داده بزرگ با بهینه‌سازی مناسب در D3.js.	عملکرد عالی برای تجسم‌های داده در زمان واقعی با پیچیدگی بالا یا حجم زیاد.	برای داشبوردهای نظارتی خوب است، اگرچه ممکن است عملکرد در زمان واقعی را برای تعاملات پیچیده ارائه ندهد.	عملکرد خوب برای تجسم‌های مبتنی بر نقشه اما ممکن است برای انواع دیگر تجسم‌ها به خوبی مقیاس‌بندی نشود.
قابلیت اطمینان و تحمل خطا	نیاز به هندلینگ دستی حالات خطا و تحمل خطا دارد، به طور بالقوه شکننده است.	مرزهای خطای React و کتابخانه‌های مدیریت وضعیت مانند Redux تحمل خطا و بازیابی را فراهم می‌کنند.	قابلیت اطمینان بالا، اما نیاز به مدیریت دقیق خطوط لوله رندر دارد. کتابخانه‌های WebGL ممکن است به مکانیسم‌های بازگشتی برای خطاها نیاز داشته باشند.	قابلیت اطمینان و تحمل خطای داخلی، به ویژه برای نظارت و داشبورد.	برای داده‌های مکانی قابل اعتماد است، اما نیاز به توجه به هندلینگ خطا در منطق برنامه سفارشی دارد.
نظارت و گزارش‌دهی سیستم	قابلیت‌های گزارش‌دهی محدود مگر	گزارش‌ها و داشبوردهای سفارشی را	برای گزارش‌دهی عمومی ایده‌آل	ویژگی‌های قدرتمند گزارش‌دهی و	گزارش‌دهی برای داده‌های جغرافیایی، با

معیار	رویکرد فعلی (SVG + تلاش توسعه)	جایگزین 1 (React + D3.js/Chart.js)	جایگزین 2 (کتابخانه‌های مبتنی بر WebGL: PixiJS/Three.js)	جایگزین 3 (Grafana/Power BI)	جایگزین 4 (Leaflet for Maps + React)
	اینکه ویژگی‌های سفارشی ساخته شوند.	می‌توان با تجسم‌های داده پویا در React/D3.js ساخت.	نیست، اما می‌توان از آن برای تجسم‌های تعاملی با تجزیه و تحلیل دقیق استفاده کرد.	نظارتی داخلی، به ویژه برای داده‌های زمان واقعی.	ابزارهایی برای تجزیه و تحلیل نقشه تعاملی، اگرچه در خارج از موارد استفاده مکانی محدود است.

خلاصه جایگزین‌ها:

- **D3.js/Chart.js**: بهترین گزینه برای برنامه‌هایی است که به انعطاف‌پذیری در تجسم‌های داده تعاملی و لحظه‌ای و مقیاس‌پذیری نیاز دارند. اگر می‌خواهید کنترل کاملی بر UI خود داشته باشید و نیاز به مدیریت مجموعه‌های داده بزرگ دارید، انتخاب خوبی است.
- **کتابخانه‌های مبتنی بر WebGL (PixiJS/Three.js)**: ایده‌آل برای تجسم‌های با کارایی بالا و در زمان واقعی که نیاز به رندر پیچیده دارند (به عنوان مثال، تصاویر متحرک یا تعاملی سه بعدی). این یک گزینه خوب برای برنامه‌های کاربردی با کارایی بحرانی است.
- **Grafana/Power BI**: عالی برای داشبوردهای نظارتی و گزارش‌دهی که نیاز به نمایش داده‌های در زمان واقعی از سیستم‌های متعدد دارند. این سیستم کمتر از یک سیستم کاملاً سفارشی قابل تنظیم است، اما قابلیت‌های تجسم قوی را خارج از جعبه فراهم می‌کند.
- **Leaflet for Maps**: بهترین گزینه برای برنامه‌هایی که نیاز به تجسم داده‌های مکانی در زمان واقعی دارند. اگر سیستم شما شامل ردیابی مواد، کامیون‌ها یا دارایی‌ها با داده‌های مکانی است، Leaflet همراه با React می‌تواند یک راه‌حل قوی باشد.

معیارهای شرکت

در اینجا یک تجزیه و تحلیل دقیق از **معیارهایی** که تیم شما باید برای ساخت یک داشبورد تعاملی و سیستم کنترل برای یک خط تولید در نظر بگیرد، ارائه شده است. هر معیار توضیح داده شده و جدولی برای سهولت مراجعه ارائه شده است.

جزئیات معیارها

1. **سهولت توسعه اولیه**
   - اندازه‌گیری می‌کند که شروع پروژه از ابتدا چقدر سر راست است.
   - عوامل: در دسترس بودن چارچوب‌ها، کتابخانه‌ها و ابزارها؛ وضوح الزامات؛ تخصص تیم
   - مثال: استفاده از یک چارچوب مانند React یا Angular می‌تواند توسعه اولیه را سرعت بخشد.
2. **سهولت اصلاح**
   - اندازه‌گیری می‌کند که ایجاد تغییرات در سیستم پس از استقرار چقدر آسان است.
   - عوامل: مدولار بودن کد، مستندات و استفاده از الگوهای طراحی.
   - مثال: یک پایگاه کد خوش ساخت با اجزای قابل استفاده مجدد، اصلاحات را آسان‌تر می‌کند.
3. **حداقل زمان لازم برای اصلاح**
   - اندازه‌گیری می‌کند که پیاده‌سازی تغییرات یا به‌روزرسانی‌ها به چقدر زمان نیاز دارد.
   - عوامل: خوانایی کد، آزمایش خودکار و خطوط لوله CI/CD.
   - مثال: یک سیستم با آزمایش خودکار می‌تواند زمان اصلاح را به طور قابل توجهی کاهش دهد.
4. **تعداد توسعه‌دهندگان درگیر در طول توسعه**
   - اندازه تیم مورد نیاز برای ساخت و نگهداری سیستم را اندازه‌گیری می‌کند.
   - عوامل: پیچیدگی پروژه، سطح مهارت توسعه‌دهندگان و ابزارهای همکاری.
   - مثال: یک تیم کوچکتر با تخصص بالا ممکن است کارآمدتر از یک تیم بزرگتر و کم تجربه‌تر باشد.
5. **انعطاف‌پذیری در استفاده از دارایی‌های شخص ثالث**
   - اندازه‌گیری می‌کند که ادغام کتابخانه‌ها، APIها یا ابزارهای شخص ثالث چقدر آسان است.
   - عوامل: سازگاری، مجوز و پشتیبانی انجمن.
   - مثال: استفاده از یک کتابخانه نمودارسازی مانند Chart.js یا D3.js برای تجسم.
6. **ادغام با داده‌های زنده**
   - اندازه‌گیری می‌کند که سیستم چقدر می‌تواند داده‌های در زمان واقعی را از حسگرها یا APIها مدیریت کند.
   - عوامل: طراحی خط لوله داده، چارچوب‌های در زمان واقعی (به عنوان مثال، WebSockets) و مقیاس‌پذیری.
   - مثال: استفاده از WebSockets یا MQTT برای جریان داده در زمان واقعی.

7. **ادغام با پایگاه داده**
   - اندازه‌گیری می‌کند که سیستم چقدر با پایگاه‌های داده برای ذخیره و بازیابی داده‌ها ادغام می‌شود.
   - عوامل: نوع پایگاه داده (SQL در مقابل NoSQL)، ابزارهای ORM و بهینه‌سازی پرس و جو.
   - مثال: استفاده از PostgreSQL با یک ORM مانند Sequelize برای تعاملات کارآمد با پایگاه داده.

جدول معیارها

معیار	توضیحات	عوامل کلیدی	مثال
سهولت توسعه اولیه	شروع پروژه چقدر آسان است.	چارچوب‌ها، کتابخانه‌ها، تخصص تیم، وضوح الزامات.	استفاده از React یا Angular برای توسعه فرانت‌اند.
سهولت اصلاح	ایجاد تغییرات پس از استقرار چقدر آسان است.	مدولار بودن کد، مستندات، الگوهای طراحی.	کد مدولار با اجزای قابل استفاده مجدد.
حداقل زمان لازم برای اصلاح	زمان مورد نیاز برای پیاده‌سازی تغییرات.	خوانایی کد، آزمایش خودکار، خطوط لوله CI/CD.	آزمایش خودکار زمان اصلاح را کاهش می‌دهد.
تعداد توسعه‌دهندگان درگیر	اندازه تیم مورد نیاز برای توسعه و نگهداری.	پیچیدگی پروژه، سطح مهارت توسعه‌دهنده، ابزارهای همکاری.	تیم کوچکتر و با مهارت بالا برای کارایی.
انعطاف‌پذیری در استفاده از دارایی‌های شخص ثالث	ادغام ابزارهای شخص ثالث چقدر آسان است.	سازگاری، مجوز، پشتیبانی انجمن.	استفاده از Chart.js یا D3.js برای تجسم.
ادغام با داده‌های زنده	سیستم داده‌های در زمان واقعی را چقدر خوب مدیریت می‌کند.	طراحی خط لوله داده، چارچوب‌های در زمان واقعی (به عنوان مثال، WebSockets)، مقیاس‌پذیری.	استفاده از WebSockets یا MQTT برای جریان داده در زمان واقعی.
ادغام با پایگاه داده	سیستم چقدر با پایگاه‌های داده ادغام می‌شود.	نوع پایگاه داده (SQL/NoSQL)، ابزارهای ORM، بهینه‌سازی پرس و جو.	استفاده از PostgreSQL با Sequelize ORM.

توصیه‌ها برای تیم شما

1. **انتخاب پشته فناوری مناسب**:
   - فرانت‌اند: React (برای سهولت توسعه و اصلاح).
   - بک‌اند: Node.js با Express (برای مدیریت داده در زمان واقعی).
   - پایگاه داده: PostgreSQL (برای داده‌های ساختاریافته) یا MongoDB (برای داده‌های بدون ساختار).
   - داده‌های در زمان واقعی: WebSockets یا MQTT.
2. **تمرکز بر مدولار بودن**:
   - از اجزای قابل استفاده مجدد و معماری میکروسرویس‌ها برای آسان‌تر کردن اصلاحات استفاده کنید.
3. **خودکارسازی آزمایش و استقرار**:
   - خطوط لوله CI/CD را برای کاهش زمان اصلاح و اطمینان از کیفیت پیاده‌سازی کنید.
4. **استفاده از ابزارهای شخص ثالث**:
   - از کتابخانه‌هایی مانند D3.js، Chart.js یا Material-UI برای توسعه سریعتر استفاده کنید.
5. **برنامه‌ریزی برای مقیاس‌پذیری**:
   - سیستم را برای مدیریت مقادیر فزاینده‌ای از داده‌های زنده و کاربران طراحی کنید.

مقایسه بر اساس معیارهای شرکت

در اینجا مقایسه‌ای از ابزارهای ذکر شده بر اساس معیارهای ارائه شده آورده شده است:

معیار	SVG/محتوای ثابت	D3.js/Chart.js	کتابخانه‌های مبتنی بر WebGL (PixiJS/Three.js)	Grafana/Power BI	Leaflet برای نقشه‌ها
سهولت توسعه اولیه	آسان (استاتیک، منطق حداقلی)	متوسط (نیاز به درک نمودارها دارد)	پیچیده (نیاز به تخصص گرافیک و WebGL دارد)	متوسط (نیاز به تنظیم یکپارچه‌سازی دارد)	آسان (تنظیمات اصلی نقشه
سهولت اصلاح	بسیار آسان (محتوای ثابت	متوسط (پیکربندی	پیچیده (اصلاح کد WebGL	آسان (یکپارچه‌سازی	آسان (می‌توان با

معیار	SVG/محتوای ثابت	D3.js/Chart.js	کتابخانه‌های مبتنی بر WebGL (PixiJS/Three.js)	Grafana/Power BI	Leaflet برای نقشه‌ها
	ساده است)	نمودارها نیاز به تغییر دارد)	ممکن است دشوار باشد)	منابع داده می‌تواند انتزاعی شود)	پلاگین‌ها گسترش داد
حداقل زمان لازم برای اصلاح	بسیار کوتاه (به‌روزرسانی‌های ثابت)	متوسط (به‌روزرسانی‌های داده نمودار)	طولانی (تغییرات در مدل‌های سه بعدی زمان‌بر است)	کوتاه (یکپارچه‌سازی داده از طریق داشبورد انجام می‌شود)	کوتاه (افزودن لایه‌ها و نشانگرها)
تعداد توسعه‌دهندگان درگیر	1-2 (محتوای ثابت اساسی)	2-3 (تجسم‌ها نیاز به توسعه‌دهندگان دارند)	3+ (توسعه‌دهندگان گرافیک مورد نیاز است)	1-2 (متخصصان داشبورد و مهندسان داده)	1-2 (برای پیکربندی نقشه)
انعطاف‌پذیری در استفاده از دارایی‌های شخص ثالث	کم (محتوای ثابت، انعطاف‌پذیری محدود)	بالا (کتابخانه‌ها و پلاگین‌های زیادی در دسترس است)	بالا (می‌توان از دارایی‌های WebGL زیادی استفاده کرد)	بالا (پلاگین‌ها و یکپارچه‌سازی‌ها در دسترس است)	بالا (پلاگین‌های متعدد برای نقشه‌برداری
ادغام با داده‌های زنده	متوسط (نیاز به هندلینگ سفارشی برای داده‌های زنده دارد)	بالا (داده‌های در زمان واقعی برای نمودارها پشتیبانی می‌شود)	بالا (WebSocket یا خطوط لوله سفارشی مورد نیاز است)	بالا (پشتیبانی از داده‌های در زمان واقعی)	متوسط (نقشه‌ها می‌توانند با داده‌های در زمان واقعی به‌روز شوند)
ادغام با پایگاه داده	کم (یکپارچه‌سازی مستقیم نیست)	متوسط (نیاز به اتصالات سفارشی پایگاه داده دارد)	متوسط (نیاز به خطوط لوله داده سفارشی دارد)	بالا (می‌تواند مستقیماً از پایگاه‌های داده بیرون بکشد)	متوسط (یکپارچه‌سازی داده‌های مکانی)

بینش‌های سریع:

- **سهولت توسعه**: شروع با SVG/محتوای ثابت آسان‌ترین است، اما انعطاف‌پذیری ندارد. پیچیدگی D3.js/Chart.js و Grafana/Power BI متوسط‌تر است، در حالی که کتابخانه‌های مبتنی بر WebGL مانند PixiJS/Three.js پیچیده‌تر هستند و نیاز به تخصص خاص دارند.
- **انعطاف‌پذیری**: D3.js/Chart.js، کتابخانه‌های WebGL و ترکیب D3.js + Leaflet انعطاف‌پذیری بالایی را ارائه می‌دهند، به‌ویژه هنگام ترکیب تجسم‌ها و نقشه‌های سفارشی. Grafana/Power BI و Leaflet نیز انعطاف‌پذیری را ارائه می‌دهند اما به روش‌های مختلف (از طریق اکوسیستم‌های پلاگین و نقشه‌ها).
- **ادغام داده‌های زنده**: D3.js و کتابخانه‌های WebGL پتانسیل بالایی برای ادغام داده‌های زنده دارند و Grafana/Power BI برای داشبوردهای در زمان واقعی عالی است. Leaflet برای نقشه‌ها می‌تواند از به‌روزرسانی‌های در زمان واقعی پشتیبانی کند، اما محدودیت‌هایی در تجسم داده‌های پویا دارد.

هر ابزار نقاط قوت خود را دارد، بنابراین انتخاب درست به نیازهای خاص شما در مورد تعامل، پیچیدگی و مهارت‌های تیمی بستگی دارد.

مقایسه رتبه‌بندی شده

بر اساس مقایسه ابزارهای ارائه شده، در اینجا رتبه‌بندی بر اساس مقیاس 100 امتیازی آورده شده است، که عواملی مانند سهولت توسعه، انعطاف‌پذیری، ادغام داده‌های زنده و ادغام با پایگاه‌های داده را در نظر می‌گیرد:

ابزار	سهولت توسعه اولیه	سهولت اصلاح	حداقل زمان برای اصلاح	اندازه تیم مورد نیاز	انعطاف‌پذیری با دارایی‌های شخص ثالث	د
D3.js + Leaflet برای نقشه‌ها	75 (متوسط، نیاز به ترکیب دارد)	75 (هر دو کتابخانه نیاز به کار دارند)	70 (نیاز به هماهنگی بین D3 و Leaflet دارد)	85 (نیاز به 2-3 توسعه‌دهنده دارد)	90 (نقاط قوت هر دو کتابخانه را ترکیب می‌کند)	90 در + به D3
Grafana/Power BI	80 (راه‌اندازی ساده، نیاز به یکپارچه‌سازی داده دارد)	85 (سفارشی‌سازی آسان برای داشبوردها)	85 (می‌تواند به سرعت با تغییرات داده به‌روز شود)	70 (اندازه تیم کم)	90 (اکوسیستم قوی برای پلاگین‌ها و یکپارچه‌سازی‌ها)	95 دا زم ط اس

ابزار	سهولت توسعه اولیه	سهولت اصلاح	حداقل زمان برای اصلاح	اندازه تیم مورد نیاز	انعطاف‌پذیری با دارایی‌های شخص ثالث	د
D3.js/Chart.js	70 (منحنی یادگیری برای تجسم‌های پیچیده دارد)	80 (نمودارها مدولار و انعطاف‌پذیر هستند)	80 (به‌روزرسانی سریع با تغییرات داده)	75 (نیاز به 2-3 توسعه‌دهنده برای سفارشی‌سازی کامل دارد)	85 (اکوسیستم عالی برای تجسم داده‌ها)	80 دا زم تلا
WebGL (PixiJS/Three.js)	60 (پیچیدگی بالا، منحنی یادگیری شیب‌دار)	60 (نیاز به تخصص در گرافیک دارد)	50 (تنظیم تجسم‌ها می‌تواند زمان‌بر باشد)	90 (نیاز به توسعه‌دهندگان گرافیک متخصص دارد)	85 (بسیار قابل تنظیم با دارایی‌ها)	85 ه س دا دا
Leaflet برای نقشه‌ها	80 (تنظیمات اولیه نقشه آسان است)	85 (می‌توان با پلاگین‌ها گسترش داد)	75 (به‌روزرسانی سریع برای لایه‌های نقشه)	70 (نیاز به 1-2 توسعه‌دهنده برای یکپارچه‌سازی کامل دارد)	85 (انعطاف‌پذیر با پلاگین‌ها و دارایی‌های شخص ثالث)	75 به اس وا
SVG/محتوای ثابت	90 (شروع با آن بسیار آسان است)	90 (تغییرات ساده محتوای ثابت)	90 (اصلاح سریع)	50 (1-2 توسعه‌دهنده برای محتوای اساسی)	50 (انعطاف‌پذیری محدود)	50 قا ز

رتبه‌ها:

1. **D3.js + Leaflet برای نقشه‌ها - 85 امتیاز**
   - به دلیل انعطاف‌پذیری، توانایی مدیریت داده‌های زنده و یکپارچه‌سازی با پایگاه‌های داده، بهترین انتخاب است. کمی پیچیده است اما برای داشبوردها و نقشه‌برداری پیشرفته ارزش تلاش را دارد.
2. **Grafana/Power BI - 85 امتیاز**
   - بهترین برای راه‌اندازی سریع داشبوردهای در زمان واقعی با حداقل تلاش است و برای تجسم داده‌ها قدرتمند است. اگر به بینش‌ها و گزارش‌دهی سریع نیاز دارید، عالی است.

3. **D3.js/Chart.js - 81 امتیاز**

- انعطاف‌پذیر و برای تجسم داده‌ها خوب است، اما ممکن است برای مدیریت داده‌های زنده نیاز به تلاش بیشتری داشته باشد. ترکیب D3.js و Chart.js برای ایجاد نمودارهای پیچیده ایده‌آل است.

4. **WebGL (PixiJS/Three.js) - 75 امتیاز**

- پتانسیل انعطاف‌پذیری و گرافیکی بالایی دارد، اما پیچیده است و نیاز به تخصص ویژه دارد. بهترین انتخاب برای یکپارچه‌سازی داده‌های ساده‌تر یا اصلاحات سریع نیست.

5. **Leaflet برای نقشه‌ها - 75 امتیاز**

- برای تجسم‌های ساده نقشه و قابلیت‌های متوسط در زمان واقعی خوب است، اما فاقد ویژگی‌های مدیریت داده پیشرفته است که برخی از گزینه‌های دیگر دارند.

6. **SVG/محتوای ثابت - 70 امتیاز**

- برای محتوای ثابت بسیار آسان است، اما فاقد انعطاف‌پذیری، یکپارچه‌سازی در زمان واقعی و توانایی مدیریت داده‌های پیچیده یا تعامل است.

نتیجه‌گیری:

اگر به‌روزرسانی‌های در زمان واقعی و انعطاف‌پذیری برای پروژه شما حیاتی هستند، D3.js + Leaflet برای نقشه‌ها یا Grafana/Power BI بهترین انتخاب‌ها خواهند بود. اگر پروژه شما به نقشه‌برداری پیشرفته اما با مدیریت داده ساده‌تر نیاز دارد، Leaflet انتخاب محکمی است. برای گرافیک سه بعدی و تجسم‌های پیچیده،

ابزارهای WebGL قدرتمند هستند اما ممکن است برای نیازهای ساده‌تر زیاده‌روی باشد.

D3.js

D3.js (اسناد مبتنی بر داده) یک کتابخانه قدرتمند جاوا اسکریپت است که برای ایجاد تجسم‌های داده‌ای پویا و تعاملی در مرورگرهای وب استفاده می‌شود. این به توسعه‌دهندگان اجازه می‌دهد تا داده‌ها را به عناصر HTML متصل کنند، که سپس می‌توان بر اساس آن داده‌ها، آن‌ها را دستکاری کرد. D3.js امکان ایجاد تجسم‌های پیچیده مانند نمودارها، نقشه‌ها و انیمیشن‌ها را فراهم می‌کند و آن را به طور گسترده برای تجسم داده‌ها در برنامه‌های وب مورد استفاده قرار می‌دهد.

برخی از ویژگی‌های کلیدی D3.js عبارتند از:

- **اتصال داده**: D3 به شما امکان می‌دهد داده‌ها را به عناصر DOM متصل کنید، و ایجاد و به‌روزرسانی تجسم‌ها را به صورت پویا بر اساس تغییر داده‌ها آسان می‌کند.
- **رندر SVG و Canvas**: از SVG (گرافیک برداری مقیاس‌پذیر) و Canvas برای رندر عناصر گرافیکی استفاده می‌کند و به شما کنترل کاملی بر طراحی و استایل می‌دهد.
- **تعامل**: D3.js از ایجاد تجسم‌های تعاملی، مانند تولتیپ‌ها، جلوه‌های هاور و رویدادهای کلیک پشتیبانی می‌کند.
- **انیمیشن‌ها**: انیمیشن‌های روان را برای انتقال بین حالات داده ارائه می‌دهد، و امکان تجسم‌های جذاب‌تر و آموزنده‌تر را فراهم می‌کند.

این کتابخانه به ویژه برای ساخت نمودارها و گراف‌های سفارشی و تعاملی (مانند نمودارهای میله‌ای، نمودارهای خطی و نمودارهای پراکندگی) که می‌توانند به صورت پویا بر اساس ورودی کاربر یا تغییر داده‌ها به‌روز شوند، محبوب است.

برای شبیه‌سازی یک شبکه برق نیروگاه ساده با D3.js، از جمله ولتاژ خط و وضعیت سوئیچ (باز/بسته)، می‌توانیم از D3.js برای تجسم شبکه استفاده کنیم، جایی که گره‌ها نشان‌دهنده نیروگاه‌ها و خطوط شبکه هستند و لبه‌های بین گره‌ها نشان‌دهنده خطوط انتقال هستند. ولتاژ خط را می‌توان با ضخامت یا رنگ لبه‌ها نشان داد و وضعیت سوئیچ (باز/بسته) را می‌توان با استفاده از برچسب‌ها یا آیکون‌ها نشان داد.

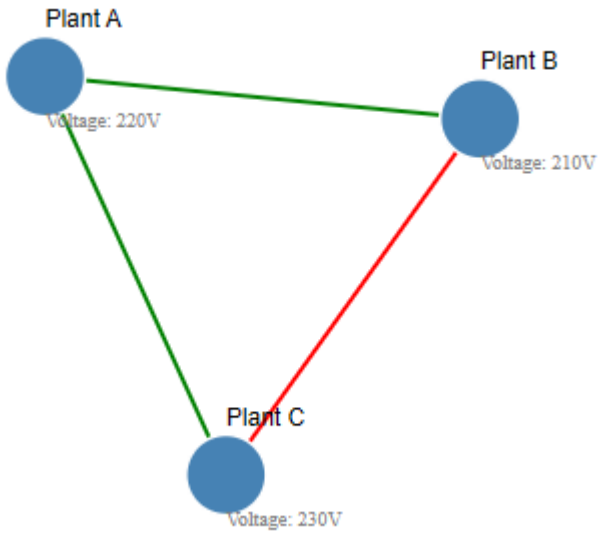
در زیر یک مثال اساسی از نحوه تنظیم چنین تجسمی با استفاده از D3.js آورده شده است:

مثال HTML + D3.js

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>شبیه‌سازی شبکه برق</title>
<script src="https://d3js.org/d3.v7.min.js"></script>
<style>
.node {
  fill: steelblue;
  stroke: #fff;
  stroke-width: 1.5px;
}

.link {
  stroke: #999;
  stroke-width: 2px;
}

.open {
  stroke: red;
  stroke-width: 3px;
}

.close {
  stroke: green;
  stroke-width: 3px;
}

.label {
  font-size: 12px;
  font-family: Arial, sans-serif;
  fill: black;
  pointer-events: none;
}

.voltage-label {
  font-size: 10px;
  fill: gray;
}
</style>
</head>
<body>

<svg width="800" height="600"></svg>
```

```
<script>
// تعریف داده‌ها برای شبکه
] = const nodes
,{ A", voltage: 220, x: 100, y: 100 نیروگاه" :id: 1, name }
,{ B", voltage: 210, x: 400, y: 150 نیروگاه" :id: 2, name }
 { C", voltage: 230, x: 250, y: 400 نیروگاه" :id: 3, name }
;[

] = const links
,{ "source: 1, target: 2, status: "close }
 ,{ "source: 2, target: 3, status: "open }
 { "source: 3, target: 1, status: "close }
;[

// تنظیم کانتینر SVG
;const svg = d3.select("svg")

// ایجاد شبیه‌سازی برای شبکه برق
const simulation = d3.forceSimulation(nodes)
force("link", d3.forceLink(links).id(d => d.id).distance(200)).
force("charge", d3.forceManyBody().strength(-1000)).
;force("center", d3.forceCenter(400, 300)).

// افزودن لینک‌ها (خطوط شبکه)
const link = svg.selectAll(".link")
data(links).
enter().append("line").
attr("class", d => "link " + (d.status === "open" ? "open" : "close")).
;style("stroke-width", 2).

// افزودن گره‌ها (نیروگاه‌ها)
const node = svg.selectAll(".node")
data(nodes).
enter().append("circle").
attr("class", "node").
;attr("r", 20).

// افزودن برچسب‌ها برای گره‌ها
svg.selectAll(".label")
data(nodes).
enter().append("text").
attr("class", "label").
attr("x", d => d.x).
attr("y", d => d.y - 25).
;text(d => d.name).
```

```
// افزودن برچسب‌های ولتاژ
svg.selectAll(".voltage-label")
  .data(nodes)
  .enter().append("text")
  .attr("class", "voltage-label")
  .attr("x", d => d.x)
  .attr("y", d => d.y + 25)
  .text(d => `Voltage: ${d.voltage}V`);

// به‌روزرسانی موقعیت‌ها بر اساس شبیه‌سازی
simulation.on("tick", () => {
  link
    .attr("x1", d => d.source.x)
    .attr("y1", d => d.source.y)
    .attr("x2", d => d.target.x)
    .attr("y2", d => d.target.y);

  node
    .attr("cx", d => d.x)
    .attr("cy", d => d.y);

  svg.selectAll(".label")
    .attr("x", d => d.x)
    .attr("y", d => d.y - 25);

  svg.selectAll(".voltage-label")
    .attr("x", d => d.x)
    .attr("y", d => d.y + 25);
});
</script>

</body>
</html>
```



تجزیه و تحلیل:

1. **گره‌ها**: نشان دهنده نیروگاه‌ها هستند. هر گره دارای `voltage`، `name`، `id` و `x`، `position` (`y` است.)
2. **لینک‌ها**: نشان دهنده خطوط انتقال برق بین نیروگاه‌ها هستند. هر لینک دارای `target` و `status` (یا `open` یا `close`) است.
3. **شبیه‌سازی D3**: یک طرح‌بندی نیرومند برای شبیه‌سازی رفتار شبکه استفاده می‌شود، جایی که گره‌ها جذب یا دفع می‌شوند و لینک‌ها (خطوط) بین آنها ایجاد می‌شوند.
4. **ولتاژ خط**: ولتاژ در نزدیکی هر گره (نیروگاه) نمایش داده می‌شود و رنگ یا ضخامت لبه‌ها می‌تواند برای نشان دادن وضعیت شبکه برق تنظیم شود.
5. **وضعیت سوئیچ**: خطوط انتقال رنگ‌های متفاوتی دارند (`قرمز` برای باز و `سبز` برای بسته`) برای نشان دادن وضعیت سوئیچ شبکه برق.

سفارشی‌سازی‌ها:

- می‌توانید مقادیر ولتاژ را به صورت پویا تغییر دهید یا آنها را بر اساس شرایط مختلف تنظیم کنید.
- وضعیت سوئیچ (باز/بسته) با تغییر رنگ خطوط متصل کننده گره‌ها منعکس می‌شود.

این تجسم یک نمای کلی از سیستم شبکه نیروگاهی را ارائه می‌دهد، نشان می‌دهد که گره‌ها چگونه به هم متصل شده‌اند، سطوح ولتاژ آنها و وضعیت خطوط انتقال.

برای رفع مشکل و گنجاندن یک نقشه خارجی مانند Leaflet، همراه با افزودن یک پاپ‌آپ به گره‌ها، چندین تنظیمات انجام می‌دهم:

1. **استفاده از Leaflet برای نقشه**: ما طرح‌ریزی `d3.geoMercator` را با عملکرد نقشه Leaflet جایگزین خواهیم کرد.

2. **عملکرد پاپ‌آپ**: یک رویداد کلیک به گره‌ها اضافه می‌کنم که هنگام کلیک جزئیات را نشان می‌دهد.
3. **ادغام نقشه با D3 برای ترسیم گره‌ها و لینک‌ها**: عناصر D3 (مانند گره‌ها و لینک‌ها) را در بالای نقشه Leaflet لایه می‌کنیم.

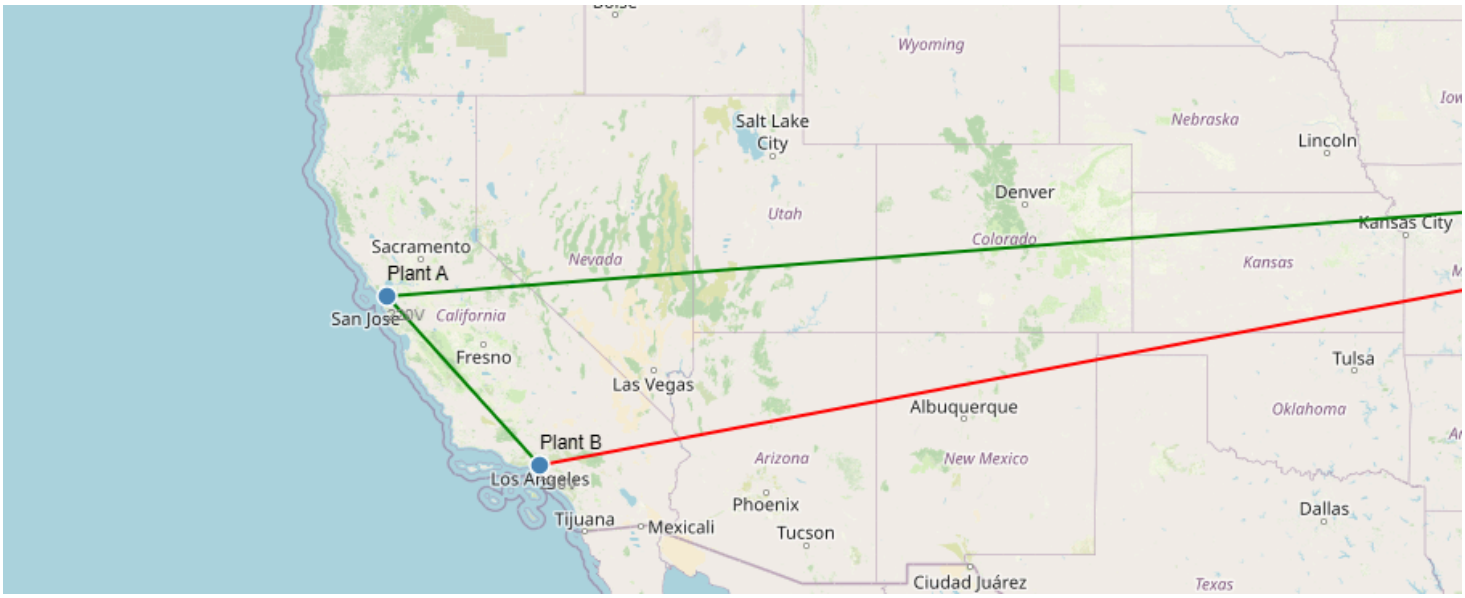
در اینجا کد به‌روزرسانی شده آورده شده است:

```html
<DOCTYPE html!>
<"html lang="en>
<head>
<"meta charset="UTF-8>
<"meta name="viewport" content="width=device-width, initial-scale=1.0>
<title/>شبیه‌سازی شبکه برق بر روی نقشه<title>
<script src="https://d3js.org/d3.v7.min.js"></script>
script src="https://cdn.jsdelivr.net/npm/topojson-client@3.1.0/dist/topojson->
<client.min.js"></script
<script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
</ "link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css>
<style>
body { margin: 0; }
map { width: 100%; height: 100vh; position: absolute; top: 0; left: 0; }.
} node.
;fill: steelblue
;stroke: white
;stroke-width: 1.5px
{
} link.
;stroke: #999
;stroke-width: 2px
{
} open.
;stroke: red
;stroke-width: 3px
{
} close.
;stroke: green
;stroke-width: 3px
{
} label.
;font-size: 12px
;font-family: Arial, sans-serif
;fill: black
{
} voltage-label.
;font-size: 10px
;fill: gray
{
<style/>
<head/>
<body>

<div id="map" class="map"></div>

<script>
```

```
// تعریف داده‌های نیروگاه‌ها و خطوط انتقال
const nodes = [
  { id: 1, name: "نیروگاه A", coordinates: [-122.4194, 37.7749], voltage: 220 }, // سان فرانسیسکو
  { id: 2, name: "نیروگاه B", coordinates: [-118.2437, 34.0522], voltage: 210 }, // لس آنجلس
  { id: 3, name: "نیروگاه C", coordinates: [-73.9352, 40.7306], voltage: 230 }  // نیویورک
];

const links = [
  { source: 1, target: 2, status: "close" },
  { source: 2, target: 3, status: "open" },
  { source: 3, target: 1, status: "close" }
];

// инициализация Leaflet карта
const map = L.map('map').setView([37.7749, -122.4194], 5); // پیش فرض به سان فرانسیسکو

// добавить слой плитки из OpenStreetMap
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
  attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
}).addTo(map);

// создание ссылок (линий передачи) с использованием D3
const svg = d3.select(map.getPanes().overlayPane).append('svg')
  .attr("width", map.getSize().x)
  .attr("height", map.getSize().y);

const g = svg.append("g").attr("class", "leaflet-zoom-hide");

// Функция для проецирования географических координат в экранные координаты
function latLonToPixel(lat, lon) {
  return map.latLngToLayerPoint([lat, lon]);
}

// Create links (transmission lines)
const link = g.selectAll(".link")
  .data(links)
  .enter().append("line")
  .attr("class", d => "link " + (d.status === "open" ? "open" : "close"))
  .style("stroke-width", 2)
  .attr("x1", d => latLonToPixel(nodes[d.source - 1].coordinates[1],
    nodes[d.source - 1].coordinates[0]).x)
  .attr("y1", d => latLonToPixel(nodes[d.source - 1].coordinates[1],
    nodes[d.source - 1].coordinates[0]).y)
```

```
        attr("x2", d => latLonToPixel(nodes[d.target - 1].coordinates[1],.
                                      nodes[d.target - 1].coordinates[0]).x)
        attr("y2", d => latLonToPixel(nodes[d.target - 1].coordinates[1],.
                                     ;nodes[d.target - 1].coordinates[0]).y)

                                       Создание узлов (электростанции) //
                                        const node = g.selectAll(".node")
                                                        data(nodes).
                                              enter().append("circle").
                                                 attr("class", "node").
                                                        attr("r", 6).
    attr("cx", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).x).
    attr("cy", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).y).
                                     } on("click", function (event, d).
                        Показать всплывающее окно при нажатии на узел //
                                                ` = const popupContent
                                    <strong>${d.name}</strong><br>
                                         ;`Voltage: ${d.voltage}V
                                                         ()L.popup
                      setLatLng([d.coordinates[0], d.coordinates[1]]).
                                             setContent(popupContent).
                                                        ;openOn(map).
                                                                ;({

                                           добавление подписей для узлов //
                                                    g.selectAll(".label")
                                                        data(nodes).
                                                 enter().append("text").
                                                 attr("class", "label").
     attr("x", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).x).
attr("y", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).y - 10).
                                                    ;text(d => d.name).

                                         добавление меток напряжения //
                                          g.selectAll(".voltage-label")
                                                        data(nodes).
                                                 enter().append("text").
                                               attr("class", "voltage-label").
     attr("x", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).x).
attr("y", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).y + 15).
                                                 ;text(d => `${d.voltage}V`).

                                Управление масштабированием и перемещением карты //
                                               ;map.on("zoomend", updatePositions)
                                               ;map.on("moveend", updatePositions)

                                                    } ()function updatePositions
```

```
    Обновление позиций ссылок и узлов при изменении масштаба или перемещения //
                                                                    link
        attr("x1", d => latLonToPixel(nodes[d.source - 1].coordinates[1],.
                                      nodes[d.source - 1].coordinates[0]).x)
        attr("y1", d => latLonToPixel(nodes[d.source - 1].coordinates[1],.
                                      nodes[d.source - 1].coordinates[0]).y)
        attr("x2", d => latLonToPixel(nodes[d.target - 1].coordinates[1],.
                                      nodes[d.target - 1].coordinates[0]).x)
        attr("y2", d => latLonToPixel(nodes[d.target - 1].coordinates[1],.
                                     ;nodes[d.target - 1].coordinates[0]).y)

                                                                    node
     attr("cx", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).x).
    ;attr("cy", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).y).

                                                          g.selectAll(".label")
      attr("x", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).x).
;attr("y", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).y - 10).

                                                       g.selectAll(".voltage-label")
      attr("x", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).x).
;attr("y", d => latLonToPixel(d.coordinates[1], d.coordinates[0]).y + 15).
                                                                        {

                                                                <script/>

                                                                  <body/>
                                                                  <html/>
```



توضیح تغییرات:

1. **ادغام Leaflet**: جاوا اسکریپت و CSS Leaflet را برای رندر کردن نقشه گنجانده‌ام و یک لایه کاشی OpenStreetMap اضافه کرده‌ام.
2. **عملکرد نقشه و بزرگنمایی**: از Leaflet `map.latLngToLayerPoint()` برای تبدیل مختصات جغرافیایی به مختصات پیکسلی استفاده می‌کنیم و گره‌ها و لینک‌ها را به درستی روی نقشه قرار می‌دهیم.
3. **عملکرد بازشو**: هنگام کلیک بر روی یک گره، یک پنجره بازشو ظاهر می‌شود و نام و ولتاژ نیروگاه را نشان می‌دهد.
4. **موقعیت‌یابی لینک**: خطوط انتقال (لینک‌ها) با استفاده از عناصر `line` از D3 کشیده می‌شوند و موقعیت‌های آنها بر اساس بزرگنمایی و حرکت فعلی نقشه به‌روزرسانی می‌شوند.

اکنون باید نقشه‌ای نمایش داده شود، نیروگاه‌ها به عنوان گره‌های قابل کلیک رندر شوند و خطوط انتقال بین آنها نشان داده شوند و با کلیک بر روی یک گره، جزئیات ظاهر شوند.

Grafana

Grafana یک پلتفرم متن باز است که برای نظارت، تجسم و تجزیه و تحلیل داده‌های سری زمانی استفاده می‌شود. این ابزار یک رابط قدرتمند و انعطاف‌پذیر برای ایجاد داشبوردهایی فراهم می‌کند که می‌توانند معیارها و لاگ‌های بلادرنگ را نمایش دهند، و آن را به طور گسترده برای نظارت بر برنامه‌ها، زیرساخت و معیارهای تجاری مورد استفاده قرار می‌دهد.

ویژگی‌های کلیدی Grafana:

1. **داشبوردها**: Grafana به کاربران اجازه می‌دهد داشبوردهای قابل تنظیم و تعاملی ایجاد کنند که می‌توانند داده‌ها را در قالب‌های مختلف مانند نمودارها، جداول، نقشه‌های حرارتی و موارد دیگر تجسم کنند. این داشبوردها را می‌توان برای نمایش مرتبط‌ترین معیارها برای یک سیستم یا برنامه خاص تنظیم کرد.

2. **ادغام منبع داده**: Grafana از طیف گسترده‌ای از منابع داده، از جمله پایگاه‌های داده سری زمانی مانند Prometheus، InfluxDB، Graphite، Elasticsearch و موارد دیگر پشتیبانی می‌کند. همچنین می‌تواند به منابع داده دیگر مانند پایگاه‌های داده SQL، ابزارهای نظارت ابری و حتی APIهای REST متصل شود.

3. **هشدار**: Grafana قابلیت‌های هشداردهی قدرتمندی را ارائه می‌دهد. شما می‌توانید بر اساس معیارهای خود هشدارها را تنظیم کنید و آنها را از طریق کانال‌هایی مانند ایمیل، Slack یا Webhookها ارسال کنید، هر زمان که شرایط خاصی برآورده شد (به عنوان مثال، هنگامی که یک معیار از آستانه فراتر رفت).

4. **گزینه‌های تجسم**: Grafana از انواع تجسم‌ها مانند نمودارهای خطی، نمودارهای میله‌ای، نمودارهای دایره‌ای، جداول و موارد دیگر پشتیبانی می‌کند. این ابزار دارای یک سیستم پلاگین است که به شما امکان می‌دهد انواع تجسم اضافی را نیز اضافه کنید.

5. **پرس و جو و کاوش**: با Grafana، کاربران می‌توانند داده‌ها را پرس و جو کرده و تجسم‌های مختلف را برای بررسی دقیق‌تر عملکرد، روندها و ناهنجاری‌ها در طول زمان کاوش کنند. این ابزار از پرس و جوهای پیچیده با گزینه‌های فیلتر و جمع‌بندی قدرتمند پشتیبانی می‌کند.

6. **دسترسی و اشتراک گذاری کاربر**: Grafana از احراز هویت کاربر، نقش‌ها و مجوزها پشتیبانی می‌کند، و کنترل اینکه چه کسی می‌تواند داشبوردها را مشاهده یا تغییر دهد را آسان می‌کند. داشبوردها همچنین می‌توانند با سهامداران به اشتراک گذاشته شوند یا به صورت عمومی در دسترس قرار گیرند.

7. **پلاگین‌ها**: Grafana دارای اکوسیستم بزرگی از پلاگین‌ها است که امکان گسترش عملکردها مانند منابع داده اضافی، انواع پانل و برنامه‌های کاربردی برای موارد استفاده خاص را فراهم می‌کند.

موارد استفاده رایج برای Grafana:

- **نظارت بر زیرساخت**: نظارت بر سلامت سرور، استفاده از منابع (CPU، حافظه، دیسک) و عملکرد شبکه.
- **نظارت بر برنامه**: ردیابی عملکرد برنامه‌های وب، میکروسرویس‌ها و پایگاه‌های داده در زمان واقعی.
- **معیارهای تجاری**: تجسم و تجزیه و تحلیل معیارهای کلیدی کسب و کار مانند داده‌های فروش، ترافیک وب سایت و تعامل کاربر.
- **نظارت امنیتی**: تجزیه و تحلیل لاگ‌ها و رویدادهای امنیتی برای تهدیدات یا ناهنجاری‌های احتمالی.

به طور خلاصه، Grafana یک ابزار ضروری برای سازمان‌هایی است که نیاز به ردیابی، تجسم و درک حجم زیادی از داده‌های سری زمانی دارند، و به اطمینان از عملکرد روان سیستم‌ها کمک می‌کند و امکان تصمیم‌گیری مبتنی بر داده را فراهم می‌سازد.

Solar PV System

lammeter Solar PV System, You can monitor the energy flow of the solar pv system.

Three.js

Three.js یک کتابخانه جاوا اسکریپت محبوب و متن باز است که توسعه دهندگان را قادر می سازد تا با استفاده از WebGL (کتابخانه گرافیکی وب) گرافیک های سه بعدی را در مرورگرهای وب ایجاد و رندر کنند. WebGL یک API سطح پایین برای رندر کردن گرافیک های دو بعدی و سه بعدی تعاملی است و Three.js با انتزاع جنبه های پیچیده WebGL به یک API قابل دسترس تر، کار با آن را آسان تر می کند.

ویژگی های کلیدی Three.js:

1. **رندر سه بعدی**: Three.js یک چارچوب برای رندر صحنه های سه بعدی در مرورگر فراهم می کند. این شامل پشتیبانی از هندسه ها (مانند مکعب ها، کره ها و اشکال سفارشی)، نورپردازی، بافت ها و مواد است که به شما امکان می دهد اشیاء و محیط های سه بعدی واقعی یا سبک دار ایجاد کنید.

2. **انتزاع WebGL**: در حالی که WebGL به خودی خود قدرتمند است، استفاده مستقیم از آن می تواند دشوار باشد. Three.js این را با ارائه انتزاعات سطح بالاتری که کار با آنها بسیار آسان تر است، مانند مدیریت خودکار سایه بان ها، کنترل های دوربین و خطوط لوله رندر، ساده می کند.

3. **دوربین ها و کنترل ها**: Three.js از انواع مختلف دوربین ها (مانند دید پرسپکتیو و ارتوگرافیک) پشتیبانی می کند و کنترل های داخلی مانند پیمایش اول شخص یا چرخش برای تعامل با صحنه های سه بعدی را ارائه می دهد.

4. **نورپردازی و سایه ها**: این شامل انواع مختلف گزینه های نورپردازی (محیط، جهت دار، نقطه ای، نورافکن و غیره) و پشتیبانی از سایه ها است که امکان ایجاد جلوه های واقع گرایانه در محیط های سه بعدی را فراهم می کند.

5. **مواد و بافت ها**: Three.js طیف وسیعی از مواد را ارائه می دهد که ظاهر اشیا را تعریف می کنند (مانند رنگ، درخشندگی و بازتابندگی). همچنین به شما امکان می دهد بافت ها را روی مدل های سه بعدی برای تصاویری با جزئیات بیشتر اعمال کنید، مانند اعمال تصاویر روی سطوح.

6. **انیمیشن ها**: می توانید اشیا را در Three.js با استفاده از فریم های کلیدی، درون یابی و سایر تکنیک ها متحرک کنید. این شامل متحرک کردن موقعیت های دوربین، تبدیل اشیا و حتی انیمیشن های اسکلتی پیچیده برای مدل های سه بعدی است.

7. **مدل ها و مش ها**: Three.js می تواند مدل های سه بعدی را در قالب های مختلف (مانند OBJ، GLTF یا FBX) بارگیری و نمایش دهد، و ادغام محتوای سه بعدی پیچیده در برنامه های وب شما را آسان می کند.

8. **سیستم های ذره ای**: می توانید جلوه های ذره ای (مانند انفجار، دود و آتش) را در Three.js ایجاد کنید که برای افزودن عناصر بصری و پویا به صحنه های شما مفید است.

9. **چند سکویی**: از آنجایی که Three.js مستقیماً در مرورگر کار می کند، به شما امکان می دهد برنامه های سه بعدی چند سکویی ایجاد کنید که می توانند روی دستگاه های مختلف، از جمله رایانه های رومیزی، تبلت ها و دستگاه های تلفن همراه اجرا شوند.

10. **پشتیبانی از VR و AR**: Three.js از واقعیت مجازی (VR) و واقعیت افزوده (AR) از طریق استفاده از WebVR و WebXR پشتیبانی می کند و ساخت تجربیات سه بعدی فراگیر را ممکن می سازد.

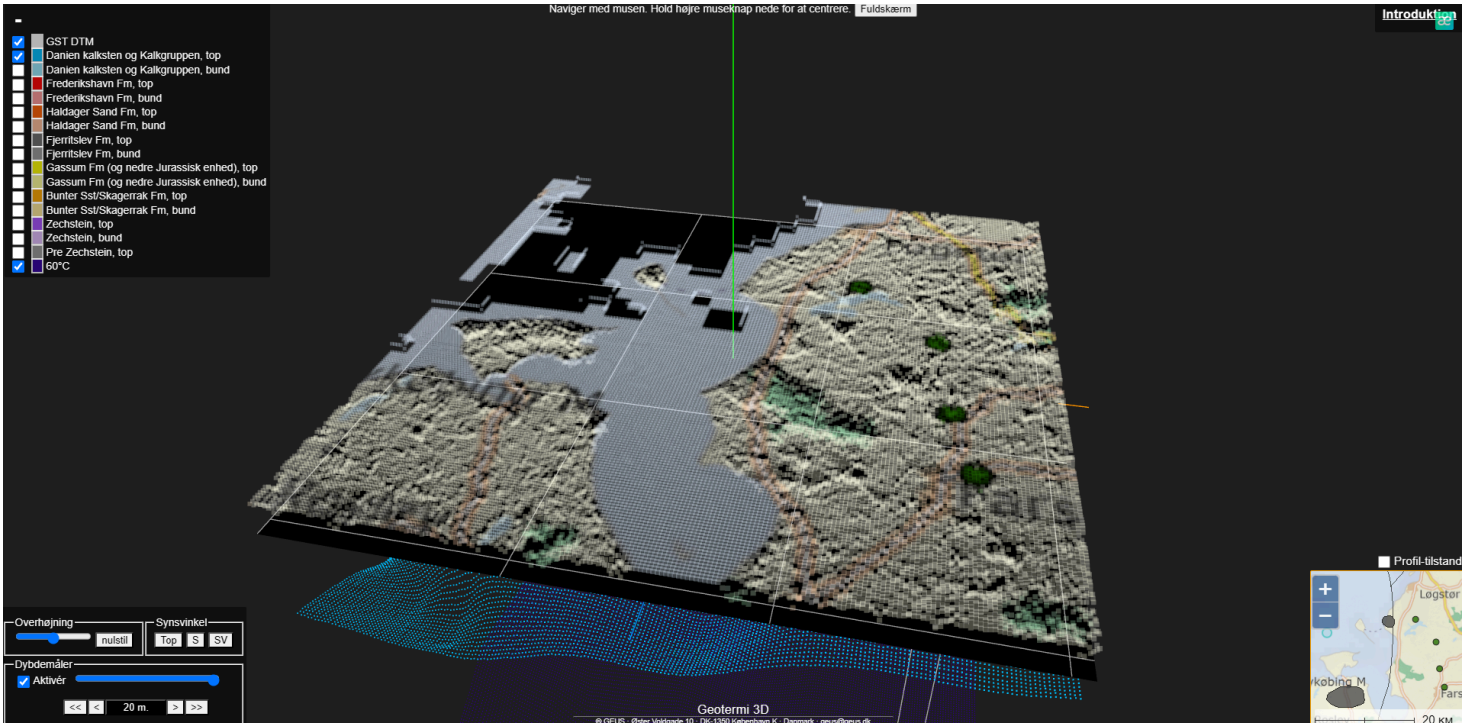
موارد استفاده رایج برای Three.js:

- **بازی های سه بعدی**: ساخت بازی ها یا شبیه سازی های سه بعدی تعاملی که در مرورگر اجرا می شوند.
- **تجسم داده**: نمایش مجموعه داده های سه بعدی پیچیده، مانند داده های جغرافیایی، ساختارهای مولکولی یا سایر تجسم های علمی.
- **تجسم معماری**: ایجاد بازدیدهای تعاملی از مدل های سه بعدی ساختمان ها، فضاهای داخلی یا محیط های شهری.
- **تجسم محصول**: به مشتریان اجازه می دهد تا با مدل های سه بعدی محصولات (مانند اتومبیل، مبلمان یا ابزارها) در یک نمایشگر مبتنی بر وب به صورت بلادرنگ تعامل داشته باشند.
- **هنر تعاملی**: طراحی هنر مولد یا تجربیات سه بعدی خلاقانه برای وب سایت ها، اینستالیشن ها یا پروژه های چند رسانه ای.

مثال مورد استفاده:

یک مثال معمول می تواند ایجاد یک نقشه سه بعدی تعاملی باشد که در آن بتوانید بچرخانید، بزرگنمایی کنید و روی قسمت های مختلف کلیک کنید تا اطلاعات بیشتری کسب کنید. مثال دیگر در تجارت الکترونیکی است، جایی که کاربران می توانند یک مدل سه بعدی از یک محصول (مانند کفش یا ماشین) را در زمان واقعی بچرخانند تا آن را از زوایای مختلف مشاهده کنند.

به طور کلی، Three.js یک انتخاب عالی برای هر کسی است که به دنبال اضافه کردن گرافیک و تجسم های سه بعدی به برنامه های وب خود است. این ابزار بسیاری از پیچیدگی های رندر سه بعدی را از بین می برد و به توسعه دهندگان این امکان را می دهد تا بر ساخت تجربیات جذاب تمرکز کنند.



Leaflet

Leaflet یک کتابخانه جاوا اسکریپت سبک و متن‌باز است که برای ایجاد نقشه‌های تعاملی در برنامه‌های وب استفاده می‌شود. این کتابخانه یک راه آسان برای ادغام و نمایش نقشه‌ها با ویژگی‌های مختلف، مانند نشانگرها، پنجره‌های بازشو، بزرگ‌نمایی و غیره را به روشی ساده و کارآمد ارائه می‌دهد.

ویژگی‌های کلیدی Leaflet:

1. **نمایش نقشه**: Leaflet به شما امکان می‌دهد نقشه‌ها را در برنامه‌های وب خود جاسازی کنید. می‌توانید از ارائه‌دهندگان کاشی مختلف (مانند OpenStreetMap، Google Maps یا Mapbox) برای نمایش کاشی‌های نقشه استفاده کنید.

2. **نشانگرها و پنجره‌های بازشو**: می‌توانید نشانگرها را به نقشه اضافه کنید که می‌توانند تعاملی باشند. نشانگرها می‌توانند هنگام کلیک، پنجره‌های بازشو را نمایش دهند و اطلاعات یا داده‌های اضافی را نشان دهند.

3. **بزرگ‌نمایی و پیمایش**: Leaflet از بزرگ‌نمایی و پیمایش پشتیبانی می‌کند و به کاربران امکان می‌دهد به راحتی در اطراف نقشه حرکت کنند. همچنین از کنترل‌های تعاملی برای بزرگ‌نمایی و تغییر نوع نقشه پشتیبانی می‌کند.

4. **کنترل لایه**: Leaflet به شما امکان می‌دهد با چندین لایه نقشه (مانند ماهواره، زمین یا نماهای خیابان) کار کنید و کاربران می‌توانند بر اساس ترجیحات خود بین این لایه‌ها جابجا شوند.

5. **موقعیت جغرافیایی**: از موقعیت جغرافیایی پشتیبانی می‌کند، بنابراین می‌توانید موقعیت فعلی کاربر را روی نقشه ردیابی و نمایش دهید.

6. **لایه‌های سفارشی**: Leaflet به شما امکان می‌دهد لایه‌های سفارشی، از جمله پوشش‌های تصویر، لایه‌های برداری (به عنوان مثال، چندضلعی‌ها و خطوط) یا حتی داده‌های GeoJSON خارجی را برای نمایش اطلاعات جغرافیایی اضافه کنید.

7. **مدیریت رویداد**: دارای یک سیستم قوی برای مدیریت رویدادها است، بنابراین می‌توانید اقدامات سفارشی را روی رویدادهای نقشه مانند کلیک، بزرگ‌نمایی یا حرکات ماوس اضافه کنید.

8. **دوستانه موبایل**: Leaflet به گونه‌ای طراحی شده است که پاسخگو باشد و به خوبی روی دستگاه‌های دسکتاپ و موبایل کار می‌کند و حتی در صفحه‌های کوچک تعاملات روانی را ارائه می‌دهد.

9. **پلاگین‌ها**: Leaflet دارای یک اکوسیستم غنی از پلاگین‌ها است که عملکرد آن را گسترش می‌دهند و ویژگی‌هایی مانند نقشه‌های حرارتی، مسیریابی، زمین‌کدگذاری و حتی تجسم‌های سه‌بعدی را اضافه می‌کنند.

10. **قابل تنظیم**: تقریباً هر جنبه‌ای از نقشه، از ظاهر نشانگرها گرفته تا رفتار کنترل‌ها و لایه‌های نقشه، قابل تنظیم است.

موارد استفاده رایج برای Leaflet:

- **نقشه‌های تعاملی**: برای نمایش نقشه‌های پویا با ویژگی‌های تعاملی مانند بزرگ‌نمایی، پیمایش و نشانگرهای قابل کلیک استفاده می‌شود.
- **برنامه‌های مبتنی بر مکان**: ایده‌آل برای برنامه‌هایی که نیاز به نمایش داده‌های مبتنی بر مکان دارند، مانند پلتفرم‌های املاک، ردیابی تحویل یا برنامه‌های گردشگری.
- **تجسم داده**: می‌توانید داده‌های جغرافیایی را تجسم کنید، مانند ترسیم نقاط روی نقشه، نشان دادن مناطق با نقشه‌های حرارتی یا رسم چندضلعی‌ها برای نمایش مناطق مورد علاقه.
- **تجزیه و تحلیل مکانی**: Leaflet می‌تواند برای کارهای مکانی پیشرفته‌تر، مانند رسم مسیرها، محاسبه فواصل یا نشان دادن داده‌های ارتفاع استفاده شود.

مثال مورد استفاده:

- **وب‌سایت املاک**: یک وب‌سایت که املاک موجود را روی نقشه نمایش می‌دهد. کاربران می‌توانند نقشه را بزرگ‌نمایی و کوچک‌نمایی کنند، روی نشانگرها کلیک کنند تا جزئیات ملک را ببینند و املاک را بر اساس مکان فیلتر کنند.
- **برنامه آب و هوا**: یک برنامه آب و هوا که الگوهای آب و هوایی، مانند طوفان‌ها یا ناهنجاری‌های دمایی را روی یک نقشه تعاملی نمایش می‌دهد.
- **برنامه‌ریز مسیر**: یک سرویس برای ترسیم مسیرهای پیاده‌روی یا رانندگی، نمایش مسیر روی نقشه و ارائه زمان‌های تخمینی رسیدن.

مزایای Leaflet:

- **سبک**: این یک کتابخانه مینیمالیستی و سریع است که آن را برای برنامه‌های وب دوستانه موبایل مناسب می‌کند.
- **آسان برای استفاده**: API آن به آسانی قابل استفاده است و می‌توانید تنها با چند خط کد شروع کنید.
- **بسیار قابل تنظیم**: Leaflet با اکوسیستم گسترده پلاگین و توانایی ادغام لایه‌ها و کنترل‌های سفارشی، بسیار قابل تنظیم است.

به طور خلاصه، Leaflet یک انتخاب عالی است زمانی که به یک راه‌حل سبک و انعطاف‌پذیر برای افزودن نقشه‌های تعاملی به پروژه‌های وب خود نیاز دارید.

در اینجا یک نمونه کد Leaflet.js برای تولید یک نقشه ساده با سه گره (نشان دهنده نقاط روی یک شبکه برق) و مسیرها (نشان دهنده خطوط شبکه بین آنها) با برچسب هایی که ولتاژ را روی هر مسیر و نام گره نشان می دهد، آمده است.

```html
<DOCTYPE html!>
<"html lang="en>
<head>
<"meta charset="UTF-8>
<"meta name="viewport" content="width=device-width, initial-scale=1.0>
<title/>نقشه شبکه برق<title>
</"link rel="stylesheet" href="https://unpkg.com/leaflet/dist/leaflet.css>
<style>
} map#
;height: 500px
;width: 100%
{
<style/>
<head/>
<body>

<div id="map"></div>

<script src="https://unpkg.com/leaflet/dist/leaflet.js"></script>
<script>
Initialize the map //
const map = L.map('map').setView([51.505, -0.09], 13);  // Centered on a
random location

Add a tile layer (OpenStreetMap in this case) //
} ,'L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png
attribution: '© <a
'href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors
;addTo(map).({

Define the nodes with coordinates (latitude, longitude) //
] = const nodes
,{ 'ولتاژ پایین' :lat: 51.505, lng: -0.09, voltage ,'گره 1' :name }
,{ 'ولتاژ پایین' :lat: 51.515, lng: -0.10, voltage ,'گره 2' :name }
{ 'ولتاژ پایین' :lat: 51.525, lng: -0.08, voltage ,'گره 3' :name }
;[

Function to add nodes to the map //
} <= nodes.forEach(node
;const marker = L.marker([node.lat, node.lng]).addTo(map)
;(`{node.voltage}$ :ولتاژ<b>${node.name}</b><br>`)marker.bindPopup
;()marker.bindTooltip(node.name).openTooltip
;({

Function to draw lines between nodes //
] = const path
,[nodes[0].lat, nodes[0].lng]
```

```
                                                ,[nodes[1].lat, nodes[1].lng]
                                                 [nodes[2].lat, nodes[2].lng]
                                                                          ;[

                                    Create a polyline (path) and style it to resemble a power grid line //
                                                                } ,const powerLine = L.polyline(path
                                                                          ,'color: 'red
                                                                             ,weight: 4
                                                                           opacity: 0.6
                                                                             ;addTo(map).({

                                                Add voltage labels on each segment of the path (edge) //
                                                                    } for (let i = 0; i < path.length - 1; i++)
                                                ;const midLat = (path[i][0] + path[i + 1][0]) / 2
                                                ;const midLng = (path[i][1] + path[i + 1][1]) / 2

                                                                                L.marker([midLat, midLng])
                                                                    (`<b/>ولتاژ پایین: ولتاژ<b>`)bindPopup.
                                                                                        ;addTo(map).
                                                                                                 {

                                                                                      <script/>
                                                                                            <body/>
                                                                                            <html/>
```

تصویر نتیجه یک شبکه ساده

شرح:

1. **تنظیم نقشه**: نقشه در یک مکان تصادفی (`51.505, -0.09`) متمرکز شده است، اما می توانید مختصات را به هر چیزی که شبکه شما نیاز دارد تغییر دهید.
2. **گره‌ها**: سه گره با نام (`گره 1` , `گره 2` , `گره 3`) و برچسب گذاری ولتاژ پایین تعریف شده اند.
3. **نشانگرها**: هر گره با یک نشانگر نشان داده می شود که یک پنجره بازشو با نام و ولتاژ را نشان می دهد.
4. **مسیرها (خطوط شبکه برق)**: یک چندخطی کشیده شده است که گره ها را به هم متصل می کند و یک خط شبکه برق (به رنگ قرمز) را نشان می دهد. هر بخش بین گره ها دارای یک برچسب ولتاژ است.
5. **برچسب های ولتاژ**: بخش های مسیر دارای یک پنجره بازشو هستند که "ولتاژ پایین" را نشان می دهد (برای سادگی). در صورت نیاز می توان این برچسب ها را برای نشان دادن ولتاژهای مختلف برای هر بخش تغییر داد.

در صورت نیاز، مختصات و سبک را مطابق با مورد استفاده خود تنظیم کنید.

بهبود-SVG

در نظر بگیرید ما مهندسان نرم‌افزار هستیم و فرض کنید در یک شرکت بزرگ تولید کننده کالا هستیم و دارای خطوط تولید متعدد، توده‌های مواد خام، انبار و حمل و نقل داخلی (با استفاده از خطوط خودکار یا دستی با استفاده از کامیون) هستیم، و برای هر بخش از شرکت نرم‌افزار داریم، مثال نگهداری، تولید، مدیریت مواد خام، در حال حاضر از فایل‌های SVG به عنوان شماتیک‌های شرکت و با استفاده از Asp.Net استفاده می‌کنیم و این فایل‌های SVG را در صفحات Razor جاسازی می‌کنیم، اما با هر تغییر SVG مجبوریم کد C# را در فایل svg در فایل cshtml دوباره جاسازی کنیم، آیا راه بهتری برای انجام این کار وجود دارد؟

به نظر می‌رسد شما در تلاش برای مدیریت فایل‌های SVG هستید که نمایانگر اجزای مختلف سیستم تولید شما (مانند خطوط تولید، توده‌های مواد خام، انبار و غیره) هستند و آن‌ها را مستقیماً در صفحات Razor جاسازی می‌کنید. در حالی که جاسازی فایل‌های SVG به عنوان بخشی از کد C# امکان‌پذیر است، اما این رویکرد از نظر نگهداری و مقیاس‌پذیری بهترین نیست، به خصوص با رشد سیستم یا تغییرات مکرر SVGها.

در اینجا چند رویکرد بهتر وجود دارد که می‌تواند گردش کار شما را بهبود بخشد و به جدا کردن کد C# شما از فایل‌های SVG کمک کند:

1. فایل‌های SVG را جدا کنید و از تگ‌های تصویر یا SVG درون‌خطی استفاده کنید

به جای جاسازی کل کد SVG در صفحات Razor خود، می‌توانید فایل‌های SVG را به طور جداگانه ذخیره کنید و از طریق تگ‌های `<img>` یا SVG درون‌خطی (با استفاده از تگ `<svg>` مستقیماً در HTML) به آن‌ها ارجاع دهید.

برای مثال:

```html
<img src="/images/production-line.svg" alt="خط تولید" />
```

این کار صفحات Razor شما را تمیزتر نگه می‌دارد و SVGها را از منطق کسب و کار جدا می‌کند. اگر نیاز به دستکاری پویای SVG دارید، می‌توانید آن را از طریق جاوا اسکریپت در DOM بارگیری کنید.

2. بارگیری پویای SVG با جاوا اسکریپت (AJAX یا Fetch)

اگر نیاز به تغییر یا تعامل پویا با SVGهای خود بر اساس شرایط خاصی دارید (به عنوان مثال، به‌روزرسانی‌های وضعیت در زمان واقعی)، می‌توانید محتوای SVG را از طریق درخواست‌های AJAX یا `fetch` در صفحه بارگیری کنید و DOM را با جاوا اسکریپت دستکاری کنید. این کار به صفحه Razor شما اجازه می‌دهد تمیز بماند، در حالی که همچنان به شما انعطاف‌پذیری برای تعامل با عناصر SVG را می‌دهد.

مثال با جاوا اسکریپت:

```javascript
fetch('/images/production-line.svg')
  .then(response => response.text())
  .then(svg => {
    document.getElementById('svg-container').innerHTML = svg;
  });
```

به این ترتیب، فایل‌های SVG شما همچنان به طور جداگانه ذخیره می‌شوند، اما می‌توانید آن‌ها را در صورت نیاز به صورت پویا در صفحه خود بارگیری کنید.

3. از SVG Sprites استفاده کنید

اگر آیکون‌ها یا نمودارهای SVG زیادی دارید و باید در مکان‌های مختلف در سراسر صفحه دوباره استفاده شوند، می‌توانید از یک SVG sprite sheet استفاده کنید. این کار به شما امکان می‌دهد چندین SVG را در یک فایل بسته‌بندی کنید، درخواست‌های HTTP را کاهش دهید و نشانه‌گذاری خود را ساده کنید.

مثال:

```html
<svg class="icon">
  <use xlink:href="/images/sprite.svg#production-line"></use>
</svg>
```

می‌توانید این sprites را با استفاده از ابزارهایی مانند SVGOMG تولید کنید یا آن‌ها را به طور خودکار به عنوان بخشی از فرآیند ساخت خود بسازید.

4. از کتابخانه‌های جاوا اسکریپت برای تعامل پویای SVG استفاده کنید

برای یک تنظیم پیچیده‌تر، استفاده از کتابخانه‌های جاوا اسکریپت مانند SVG.js یا D3.js را برای دستکاری عناصر SVG خود در نظر بگیرید. این کتابخانه‌ها به شما امکان می‌دهند تجسم‌های پویا، تعاملی و مبتنی بر داده ایجاد کنید. آن‌ها می‌توانند به شما در به‌روزرسانی بخش‌هایی از SVG خود (به عنوان مثال، خطوط تولید یا مسیرهای حمل و نقل) بر اساس داده‌های زمان واقعی کمک کنند.

5. کامپوننت‌های وب را در نظر بگیرید (اختیاری)

اگر نیاز به کپسوله کردن SVG و منطق آن دارید، می‌توانید از کامپوننت‌های وب استفاده کنید. با کامپوننت‌های وب، می‌توانید عناصر سفارشی قابل استفاده مجدد ایجاد کنید که هم رندر و هم منطق مرتبط با SVGهای شما را مدیریت می‌کنند.

مثالی از یک عنصر سفارشی برای یک SVG:

```html
<my-production-line status="active"></my-production-line>
```

کامپوننت `my-production-line` می‌تواند SVG مربوطه را بارگیری کند، استایل‌ها را اعمال کند و تعامل را ارائه دهد، و منطق را از صفحه Razor شما جدا کند.

6. از یک چارچوب CSS برای تعامل استفاده کنید (مانند SCADA)

اگر SVG شما برای نظارت بر خطوط تولید استفاده می‌شود و شامل به‌روزرسانی‌های زنده زیادی است، ممکن است بخواهید یکپارچه‌سازی فایل‌های SVG خود در یک سیستم SCADA (کنترل نظارتی و اکتساب داده) یا چارچوب تجسم مشابه را در نظر بگیرید، که می‌تواند وضعیت‌های سیستم را ردیابی کند و به‌روزرسانی‌های در زمان واقعی را فعال کند.

نتیجه گیری

برای اینکه راه حل شما قابل نگهداری‌تر باشد:

1. **فایل‌های SVG را از کد C# جدا کنید**—SVGها را به عنوان فایل ذخیره کنید و به صورت پویا از طریق جاوا اسکریپت یا تگ‌های `<svg>` درون‌خطی به آن‌ها ارجاع دهید.
2. **از جاسازی SVGها به طور مستقیم در نماهای Razor خودداری کنید** مگر اینکه کاملاً ضروری باشد.
3. **از جاوا اسکریپت برای مدیریت به‌روزرسانی‌های پویا به SVGها** بر اساس تغییرات وضعیت سیستم استفاده کنید.

این کار کد شما را ساده می‌کند و امکان تفکیک بهتر دغدغه‌ها بین UI و منطق کسب و کار را فراهم می‌کند، در حالی که به شما انعطاف‌پذیری برای مدیریت تغییرات و تعاملات SVG به طور موثرتر را می‌دهد.

آیا Jquery می تواند در این کار به ما کمک کند؟

بله، **jQuery** قطعا می تواند در این کار کمک کند، به خصوص اگر از قبل از آن در پروژه خود استفاده می کنید. jQuery چندین ویژگی را ارائه می دهد که می تواند کار با فایل های SVG را در مورد شما ساده کند. در زیر چند روش وجود دارد که jQuery می تواند به بهبود نحوه مدیریت و دستکاری SVG ها در برنامه وب شما کمک کند:

1. بارگیری و تزریق پویای SVG ها

اگر می خواهید SVG ها را به صورت پویا از فایل های خارجی بارگیری کنید و بدون بارگیری مجدد کل صفحه، آنها را در صفحه خود وارد کنید، `.load()` یا jQuery `$.get()` می تواند مفید باشد.

مثالی با استفاده از `$.get()` برای بارگیری یک SVG و تزریق آن به یک ظرف:

```javascript
} get('/images/production-line.svg', function(data).$
    ;html(data).('svg-container#')$
;({
```

این کار فایل SVG را از سرور واکشی می کند و آن را در عنصر با ID `svg-container` وارد می کند. می توانید از این برای بارگیری هر SVG به صورت پویا در صفحه استفاده کنید.

2. دستکاری عناصر SVG

پس از تزریق SVG، ممکن است بخواهید عناصر آن را دستکاری کنید (مانند تغییر رنگ یک خط نشان دهنده یک خط تولید یا به روز رسانی وضعیت ماشین آلات). jQuery به شما این امکان را می دهد که به راحتی عناصر SVG را با استفاده از انتخابگرها و متدها، درست مانند هر عنصر HTML دیگر، انتخاب و دستکاری کنید.

مثال:

```javascript
} ()ready(function.(document)$
    // تغییر رنگ یک مسیر خاص در داخل SVG
;css('fill', '#FF0000').('svg-container svg path#production-line#')$
;({
```

می توانید قسمت های خاصی از SVG را با استفاده از شناسه ها، کلاس ها یا سایر ویژگی های آن هدف قرار دهید و سپس استایل های CSS یا سایر دستکاری ها را اعمال کنید. این امر به ویژه در صورتی مفید است که نیاز به تغییر وضعیت یک SVG بر اساس داده های زمان واقعی داشته باشید (به عنوان مثال، وضعیت دستگاه).

3. ایجاد انیمیشن در SVG ها با jQuery

اگر می خواهید برخی از انیمیشن های اساسی را به SVG خود اضافه کنید (به عنوان مثال، متحرک کردن یک مسیر برای نشان دادن فعالیت در خط تولید خود)، می توانید از متد jQuery `.animate()` در ترکیب با ویژگی های SVG استفاده کنید.

مثال: متحرک کردن یک مسیر خط تولید برای شبیه سازی فعالیت:

```javascript
;animate({ strokeDashoffset: 0 }, 2000).('production-line#')$
```

این کار استروک یک مسیر SVG را متحرک می کند، به عنوان مثال، شبیه سازی یک خط که در طول زمان "پر" می شود.

می توانید از jQuery برای ایجاد عناصر SVG به صورت بصری تعاملی یا منعکس کننده تغییرات در وضعیت تولید استفاده کنید.

4. مدیریت رویدادها بر روی عناصر SVG

jQuery مدیریت رویداد را ساده می کند، که می تواند مفید باشد اگر می خواهید کاربران با SVG تعامل داشته باشند (به عنوان مثال، کلیک کردن بر روی یک خط تولید برای دریافت جزئیات بیشتر).

مثالی از اتصال یک رویداد کلیک به یک عنصر SVG:

```javascript
$('svg-container svg path#').on('click', function() {
    alert('شما بر روی یک خط تولید کلیک کردید!');
});
```

این شنونده رویداد زمانی فعال می شود که روی هر `path` در داخل SVG کلیک شود، و می توانید آن را برای انجام اقدامات مختلف بر اساس تعاملات کاربر با SVG سفارشی کنید.

5. به روز رسانی SVG ها بر اساس داده های زمان واقعی

اگر سیستم شما داده های زمان واقعی را مدیریت می کند (مانند آمار تولید، به روز رسانی وضعیت یا داده های حسگر)، می توانید از jQuery برای به روز رسانی دوره ای عناصر SVG بر اساس داده های جدید استفاده کنید، مانند تغییر رنگ یک عنصر SVG برای منعکس کردن وضعیت فعلی.

مثالی از نظرسنجی یا به روز رسانی SVG:

```javascript
setInterval(function() {
    $.get('/get-production-status', function(status) {
        if (status === 'active') {
            $('production-line#').css('fill', '#00FF00');
        } else {
            $('production-line#').css('fill', '#FF0000');
        }
    });
}, 5000);
```

در اینجا، شما هر 5 ثانیه وضعیت یک خط تولید را واکشی می کنید و رنگ آن را بر این اساس به روز می کنید. این باعث می شود SVG تعاملی و همگام با وضعیت واقعی سیستم باشد.

6. مدیریت اندازه و مقیاس بندی SVG

گاهی اوقات، SVG ها ممکن است بسته به اندازه ظرف نیاز به تغییر اندازه یا مقیاس بندی پویا داشته باشند. از jQuery می توان برای تنظیم عرض و ارتفاع SVG بر اساس اندازه نمای دید یا ظرف اصلی استفاده کرد.

مثالی از تغییر اندازه پویا یک SVG:

```javascript
$(window).resize(function () {
    var width = $('#svg-container').width();
    $('#svg-container svg#').attr('width', width);
});
```

مزایای استفاده از jQuery برای دستکاری SVG

- **سینتکس ساده**: سینتکس jQuery نسبتاً آسان برای استفاده است، که می تواند به دستکاری سریع SVG ها بدون نیاز به جاوا اسکریپت پیچیده کمک کند.
- **سازگاری بین مرورگرها**: jQuery بسیاری از ویژگی های خاص مرورگر را مدیریت می کند، بنابراین اگر برنامه شما نیاز به اجرا در مرورگرهای مختلف دارد، jQuery می تواند بسیاری از مشکلات احتمالی رندر SVG را هموار کند.
- **سهولت ادغام**: اگر از قبل از jQuery در برنامه خود استفاده می کنید، افزودن عملکرد برای دستکاری SVG ها ساده است و همه چیز را سازگار نگه می دارد.

نتیجه گیری

استفاده از **jQuery** در پروژه شما قطعا می تواند کار با SVG ها را با موارد زیر آسان تر کند:

- بارگیری پویا فایل های SVG.
- دستکاری عناصر SVG (مانند تغییر رنگ، انیمیشن و تعاملات).
- به روز رسانی SVG ها بر اساس داده های زمان واقعی یا اقدامات کاربر.

با این حال، شایان ذکر است که در حالی که jQuery قدرتمند است، برای دستکاری ها یا تعاملات SVG پیچیده تر، ممکن است در نهایت به برخی محدودیت ها برخورد کنید. اگر تعاملات SVG شما پیچیده تر شود، می توانید برای موارد استفاده پیشرفته تر، کتابخانه های تخصصی مانند **SVG.js** یا **D3.js** را نیز بررسی کنید.

اگر از قبل از jQuery در پروژه خود استفاده می کنید، این برای این کار مناسب است، به خصوص برای دستکاری های ساده و به روز رسانی های پویای SVG.

SVG برای طراحی پیچیده

ایجاد طرح‌های پیچیده با SVG (گرافیک‌های برداری مقیاس‌پذیر) یک رویکرد قدرتمند است، به‌ویژه هنگامی که با کتابخانه‌های جاوا اسکریپت مانند D3.js ترکیب شود. SVG یک فرمت انعطاف‌پذیر و مستقل از وضوح برای گرافیک‌های برداری ارائه می‌دهد، و آن را برای طرح‌های پیچیده‌ای که نیاز به مقیاس‌پذیری و تعامل دارند، ایده‌آل می‌کند.

استفاده از SVG برای طرح‌های پیچیده:

1. **ساختار و مقیاس‌پذیری:** SVG به شما امکان می‌دهد شکل‌ها، مسیرها و الگوهای پیچیده را تعریف کنید، که می‌توان آن‌ها را بدون از دست دادن کیفیت به هر اندازه‌ای مقیاس کرد. این امر به ویژه برای طرح‌های واکنش‌گرا که نیاز به سازگاری با اندازه‌های صفحه نمایش مختلف دارند، مفید است.
2. **تعامل‌پذیری:** عناصر SVG بخشی از DOM (مدل شیء سند) هستند که امکان دستکاری مستقیم با CSS و جاوا اسکریپت را فراهم می‌کند. این امر اضافه کردن ویژگی‌های تعاملی مانند جلوه‌های هاور، انیمیشن‌ها و به‌روزرسانی‌های پویا را تسهیل می‌کند.
3. **ادغام با فناوری‌های وب:** SVG به طور یکپارچه با HTML و CSS ادغام می‌شود و امکان استایل‌دهی یکنواخت و ادغام آسان در صفحات وب را فراهم می‌کند. این سازگاری فرآیند توسعه را ساده می‌کند و از سازگاری در بخش‌های مختلف یک برنامه وب اطمینان می‌دهد.

مقایسه SVG با D3.js:

در حالی که SVG یک زبان نشانه‌گذاری برای توصیف گرافیک‌های برداری است، D3.js یک کتابخانه جاوا اسکریپت است که DOM را بر اساس داده‌ها دستکاری می‌کند و امکان ایجاد تجسم‌های پویا و تعاملی را فراهم می‌کند. D3.js از SVG (در میان سایر فناوری‌ها) برای رندر کردن گرافیک‌ها استفاده می‌کند و یک انتزاع سطح بالاتر برای طرح‌های مبتنی بر داده ارائه می‌دهد.

مزایای استفاده از SVG:

- **سادگی:** برای طرح‌های ایستا یا نسبتاً تعاملی، SVG یک رویکرد سرراست را بدون نیاز به کتابخانه‌های اضافی ارائه می‌دهد.
- **عملکرد:** برای طرح‌هایی با تعداد محدودی از عناصر، SVG می‌تواند کارآمدتر باشد، زیرا نیازی به سربار یک کتابخانه جاوا اسکریپت ندارد.

مزایای استفاده از D3.js:

- **اتصال داده:** D3.js در اتصال داده‌ها به عناصر DOM عالی است و آن را برای ایجاد تجسم‌های پیچیده و مبتنی بر داده ایده‌آل می‌کند.
- **تعامل‌پذیری پیشرفته:** D3.js پشتیبانی گسترده‌ای از انیمیشن‌ها، انتقال‌ها و تعاملات پیچیده ارائه می‌دهد که پیاده‌سازی آن‌ها با SVG ساده می‌تواند چالش برانگیز باشد.
- **جامعه و اکوسیستم:** D3.js دارای یک جامعه بزرگ و انبوهی از منابع، از جمله پلاگین‌ها و نمونه‌ها است که می‌تواند توسعه را تسریع بخشد.

ملاحظات:

- **منحنی یادگیری:** D3.js به دلیل عملکرد گسترده و دستور زبان پیچیده، دارای منحنی یادگیری شیب‌دارتری است. برای استفاده مؤثر از آن، نیاز به درک عمیق‌تری از جاوا اسکریپت و SVG دارید.
- **عملکرد با مجموعه‌داده‌های بزرگ:** برای برنامه‌های کاربردی که شامل مجموعه‌داده‌های بزرگ هستند یا نیاز به به‌روزرسانی‌های در زمان واقعی دارند، D3.js ممکن است با چالش‌های عملکردی مواجه شود، زیرا برای رندر کردن به DOM متکی است. در چنین مواردی، فناوری‌هایی مانند HTML5 Canvas ممکن است عملکرد بهتری ارائه دهند.

نتیجه‌گیری:

برای طرح‌های ایستا یا نسبتاً تعاملی، SVG به تنهایی یک گزینه مناسب است که سادگی و ادغام مستقیم با فناوری‌های وب را ارائه می‌دهد. با این حال، برای تجسم‌های پیچیده و مبتنی بر داده که نیاز به تعامل‌پذیری پیشرفته و به‌روزرسانی‌های پویا دارند، D3.js یک راه‌حل قوی‌تر ارائه می‌دهد. انتخاب بین SVG و D3.js به الزامات خاص پروژه شما بستگی دارد، از جمله پیچیدگی طرح، نیاز به تعامل‌پذیری و ملاحظات عملکردی.

طراح D3.js

هیچ "طراح بصری" بومی در خود D3.js وجود ندارد - از آنجایی که D3 اساساً یک کتابخانه مبتنی بر کد است - اما چندین ابزار و چارچوب شخص ثالث به شما اجازه می دهند تا تجسم های پیچیده را با استفاده از یک رابط کشیدن و رها کردن یا رابط اشاره و کلیک طراحی کنید. این ابزارها می توانند خروجی SVG (و گاهی اوقات حتی کد سازگار با D3) را تولید کنند، که می تواند نیاز به تغییرات دستی کد و استقرار مجدد هنگام تنظیم طرح ها را کاهش دهد.

گزینه های محبوب

- Charticulator **Charticulator** که توسط Microsoft Research توسعه یافته است، یک ابزار مبتنی بر وب است که به شما امکان می دهد تجسم های سفارشی و تعاملی را بدون نوشتن کد ایجاد کنید. شما نمودار خود را به صورت بصری طراحی می کنید و سپس تجسم حاصل را به عنوان یک SVG یا به عنوان کدی که می توانید در یک پروژه D3.js ادغام کنید، صادر می کنید. این کار برای طراحان تکرار بر روی تجسم های پیچیده را بدون نیاز به تخصص برنامه نویسی عمیق آسان تر می کند. منبع: https://charticulator.com/

- RAWGraphs **RAWGraphs** یک چارچوب تجسم داده متن باز است که یک رابط کاربر پسند برای ایجاد نمودارهای پیچیده ارائه می دهد. پس از طراحی تجسم خود از طریق رابط آن، می توانید نتیجه را به عنوان یک SVG صادر کنید یا حتی آن را در یک صفحه وب جاسازی کنید. در حالی که RAWGraphs در درجه اول بر تولید SVG های استاتیک تمرکز دارد، خروجی می تواند به عنوان نقطه شروعی باشد که می توانید آن را با D3.js برای رفتار پویا بیشتر تقویت کنید. منبع: https://rawgraphs.io/

- Vega **Vega Editor / Vega-Lite** و Vega-Lite زبان های سطح بالاتری هستند که بر روی D3.js ساخته شده اند و به شما امکان می دهند تجسم ها را در یک قالب اظهاری مشخص کنید. Vega Editor یک رابط تعاملی ارائه می دهد که در آن می توانید مشخصات تجسم را تنظیم کنید و بلافاصله نتیجه را ببینید. این انتزاع می تواند نیاز به کدنویسی سطح پایین D3 را کاهش دهد در حالی که همچنان قدرت D3 را در زیر کاپوت ارائه می دهد. منبع: https://vega.github.io/editor/

مزایا و معایب

- **کاهش تعامل توسعه دهنده:** این ابزارها به طراحان یا تحلیلگران اجازه می دهند بدون نیاز به غوطه ور شدن در کد D3 زیربنایی، تجسم را تنظیم کنند. این جداسازی می تواند تکرار را سرعت بخشد و چرخه تغییرات کد و استقرار مجدد را برای هر تغییر طراحی کوچک کاهش دهد.

- **قابلیت سفارشی سازی در مقابل سهولت استفاده:** در حالی که طراحان بصری می توانند فرآیند را ساده کنند، ممکن است همان سطح کنترل را مانند راه حل های D3 کدگذاری شده با دست ارائه ندهند. برای برنامه های بسیار پیچیده یا بسیار تعاملی، ممکن است همچنان نیاز به استفاده از کد سفارشی برای رفتار پیشرفته داشته باشید.

- **یکپارچه سازی:** هنگامی که تجسم خود را صادر می کنید (به عنوان یک SVG یا حتی به عنوان کد D3.js)، می توانید آن را در برنامه وب خود ادغام کنید. این به شما امکان می دهد عناصر طراحی شده بصری را با تعامل برنامه ای ترکیب کنید - به شما این امکان را می دهد تا قسمت هایی از تجسم را بر اساس داده های زمان واقعی به روز یا متحرک کنید.

نتیجه گیری

استفاده از ابزارهایی مانند RAWGraphs، Charticulator یا Vega Editor می تواند به پر کردن شکاف بین طراحی و توسعه با ارائه یک رابط بصری برای ایجاد طرح های پیچیده و مبتنی بر داده SVG کمک کند. آنها به غیر توسعه دهندگان اجازه می دهند تا تجسم ها را نمونه سازی و تکرار کنند، در حالی که هنوز مسیری برای ادغام و بهبود آن طرح ها با D3.js در صورت نیاز به تعامل پیشرفته ارائه می دهند. این رویکرد ترکیبی می تواند گردش کار شما را ساده کند و نیاز به استقرار مجدد مکرر کل محصول را برای تنظیمات طراحی به حداقل برساند.

منابع و لینک ها

d3js

1. https://main.dp23i3r580zal.amplifyapp.com/
2. https://urmilaj.github.io/world-power-plants/
3. https://d3-graph-gallery.com/index.html
4. https://observablehq.com/@d3/raster-tiles

grafana

1. https://blog.networkprofile.org/power-monitoring-setup-iotawatt-grafana/

leaflet

1. https://www.ise.fraunhofer.de/en/press-media/news/2017/interactive-map-of-germany-with-power-plant-sites.html

SCADA

1. https://en.wikipedia.org/wiki/SCADA#:~:text=SCADA%20systems%20are%20used%20to,the%20basis%20of%20modern%20society
2. https://www.youtube.com/watch?v=nlFM1q9QPJw

Threejs

1. https://data.geus.dk/geoterm/get_3d.jsp?
11&layers=dtm,KalkGruppen_TopDybde,iso60&bbox=495279,6285628.435,525905,6316407.565

تشکر ویژه از:

1. /https://chatgpt.com
2. /https://chat.deepseek.com
3. /https://aistudio.google.com
4. /https://bard.google.com
5. /https://copilot.microsoft.com