



شماره مستند: UML-TOOL-DOC-1405-001

نسخه: ۱.۰

تاریخ تهیه: ۱۶ شهریور ۱۴۰۵ – ساعت ۱۰:۳۰

طبقه‌بندی: محرمانه / فنی

TECHNICAL DOCUMENTATION

مستند فنی پروژه ابزار رسم نمودارهای UML

تحلیل معماری، ماژول‌ها، موتور رسم و ساختار داده

همکاران فنی و ذینفعان گرامی،

این مستند به‌عنوان مرجع فنی جامع پروژه «ابزار رسم نمودارهای UML» تهیه شده است. هدف از این مستند، تشریح دقیق معماری، ساختار کد، ماژول‌های کلیدی، الگوهای طراحی به‌کار رفته و قابلیت‌های فنی این ابزار می‌باشد تا تیم‌های توسعه و نگهداری بتوانند با دیدی کامل و یکپارچه، به تحلیل، توسعه و گسترش آن بپردازند.

پروژه «ابزار رسم نمودارهای UML» یک اپلیکیشن تحت وب کاملاً سمت کلاینت (Client-Side) است که با استفاده از تکنولوژی‌های HTML5، CSS3 و JavaScript خالص (Vanilla JS) توسعه یافته است. این ابزار بدون نیاز به هیچ‌گونه Backend یا فریم‌ورک خارجی، امکان رسم، ویرایش، ذخیره و بازیابی هشت نوع نمودار استاندارد UML را در محیطی تعاملی و حرفه‌ای فراهم می‌آورد.



نوع نمودار UML پشتیبانی شده



المان و رابطه قابل رسم



اجرا در سمت سرورگر (بدون Backend)

💡 **ارزش کلیدی:** این ابزار با تکیه بر HTML5 Canvas API و معماری ماژولار، یک محیط حرفه‌ای، سبک و کاملاً آفلاین برای مدل‌سازی سیستم‌های نرم‌افزاری در اختیار معماران و تحلیل‌گران قرار می‌دهد.

معماری این پروژه بر پایه الگوی **Single-Page Application (SPA)** و با رویکرد **State-Centric** طراحی شده است. تمامی وضعیت‌های برنامه در یک آبجکت مرکزی **state** نگهداری می‌شوند و هر تغییر در **state** منجر به بازترسیم کامل (Redraw) بوم می‌گردد.

Rendering Layer 🎨: موتور رسم مبتنی بر HTML5 Canvas 2D Context با قابلیت پشتیبانی از تم روشن و تاریک.

State Management 🗨️: یک آبجکت سراسری **state** که شامل لیست نمودارها، نمودار فعال، ابزار جاری، وضعیت Drag & Draw و تنظیمات UI است.

Interaction Layer 🖱️: مدیریت رویدادهای Keyboard و Mouse (mousedown, mousemove, mouseup, dblclick) برای تعامل با المان‌ها.

Persistence Layer 💾: سیستم ذخیره‌سازی مبتنی بر JSON File API (بدون نیاز به LocalStorage یا Backend).

UI Layer 🖼️: رابط کاربری مبتنی بر Properties Panel، Sidebar، Tab Bar، Modal ها و Properties Panel.

```
// ساختار کلی State Object
const state = {
  diagrams: [], // لیست همه نمودارها
  activeDiagramId: null, // شناسه نمودار فعال
  currentTool: 'select', // ابزار انتخاب‌شده
  selectedElement: null, // المان انتخاب‌شده
  isDragging: false, // وضعیت Drag
  isDrawingRelationship: false,
  darkMode: true, // تم تاریک/روشن
  gridType: 'lines', // نوع خط‌کشی بوم
  fileMetadata: { ... } // متادیتای پروژه
};
```

لایه	تکنولوژی	نقش در پروژه
ساختار	HTML5 Semantic	ساختار صفحه و المان‌های DOM
استایل	CSS3 + CSS Variables	طراحی RTL، تم داینامیک، گرادیانت‌ها
رسم گرافیکی	HTML5 Canvas 2D API	موتور رسم نمودارها، Grid و المان‌ها
منطق برنامه	Vanilla JavaScript (ES6+)	State Management، Event Handling، Drawing Engine
ذخیره‌سازی	File API + JSON	Export/Import پروژه به‌صورت فایل JSON
فونت	Vazirmatn (Google Fonts)	تایپوگرافی فارسی حرفه‌ای
استایل کمکی	Tailwind CSS (CDN)	Utility classes برای UI سریع

💡 **نکته معماری:** انتخاب JS Vanilla به جای فریم‌ورک‌هایی مانند React یا Angular، باعث کاهش چشمگیر حجم نهایی (Bundle Size)، اجرای سریع‌تر و عدم وابستگی به ابزارهای Build شده است. این رویکرد برای ابزارهای تخصصی و آفلاین، انتخاب بهینه‌ای است.

این ابزار هشت نوع نمودار استاندارد UML را پشتیبانی می‌کند که هر کدام در رجیستری `DIAGRAM_TYPES` تعریف شده‌اند. این رجیستری به‌عنوان **Single Source of Truth** برای ابزارها، گروه‌ها و قابلیت‌های هر نمودار عمل می‌کند.



Use Case Diagram

مدل‌سازی Actorها، Use Caseها، System Boundary و روابط Include/Extend/Generalization.



State Machine Diagram

State، Initial/Final Stateها، Composite State و Transition با پرچسب اختیاری.



Activity Diagram

Action، Decision/Merge، Fork/Join و جریان‌های Control/Object Flow.



Component Diagram

Package، Interface (Provided/Required)، Component و روابط Dependency/Connector.



Deployment Diagram

.Deploy/Manifest روابط و Node، Device، Execution Environment، Artifact



Class Diagram

.UML روابط کامل و Attributes/Methods با Class، Interface، Abstract، Enumeration



Sequence Diagram

.Sync/Async/Return/Self و پیام‌های Participant، Activation، Combined Fragment



Swimlane Diagram

.Swimlane افقی/عمودی با قابلیت Fork/Join و Resize، Action، Decision درون آن

```
// الگوی طراحی - نمونه تعریف یک نمودار Registry
const DIAGRAM_TYPES = {
  usecase: {
    id: 'usecase',
    name: 'نمودار مورد کاربرد (Use Case)',
    icon: '🔴',
    tools: ['select', 'actor', 'useCase', 'boundary', ...],
    groups: ['selection', 'uc-elements', 'uc-relations', ...]
  },
  // سایر نمودارها ...
};
```

موتور رسم، قلب تپنده این پروژه است و بر پایه **Immediate Mode Rendering** طراحی شده است. در این رویکرد، در هر فریم (یا پس از هر تغییر state) کل بوم پاک‌سازی شده و تمامی المان‌ها از نو رسم می‌شوند. این رویکرد اگرچه در نگاه اول پرهزینه به نظر می‌رسد، اما به دلیل سادگی، قابلیت اطمینان بالا و عدم نیاز به مدیریت پیچیده Diff، برای این نوع ابزارها ایده‌آل است.

چرخه رسم (Render Loop):

۱. پاک‌سازی بوم: `ctx.clearRect()` و رسم پس‌زمینه بر اساس تم.

۲. رسم Grid: بر اساس نوع انتخابی (Lines/Dots/None) با استفاده از CSS Variables.

۳. رسم المان‌ها: فراخوانی توابع اختصاصی هر نمودار (مانند `drawUseCase`, `drawClassDiagram`).

۴. رسم روابط: پس از المان‌ها، روابط با محاسبه مرکز المان‌های مبدأ و مقصد رسم می‌شوند.

۵. رسم Selection: المان انتخاب‌شده با خط‌چین آبی‌رنگ مشخص می‌شود.

۶. رسم Relationship Preview: در حالت رسم رابطه، خط موقتی تا موقعیت ماوس نمایش داده می‌شود.

// چرخه اصلی رسم

```
function draw() {
  const d = getActive(), tc = getTC();
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ctx.fillStyle = tc.canvasBg;
  ctx.fillRect(0, 0, canvas.width, canvas.height);
  drawGrid(ctx, canvas.width, canvas.height, tc, false);
  if (!d) return;
  drawDiag(ctx, d, tc); // Dispatch اختصاصی
}
```

💡 **بهینه‌سازی:** برای جلوگیری از رسم‌های مکرر غیرضروری، از تکنیک `requestAnimationFrame` و

`scheduleResize()` استفاده شده است تا `Resize` رویدادها به‌صورت هوشمند مدیریت شوند.

سیستم تم این پروژه بر پایه **CSS Custom Properties (Variables)** طراحی شده است. با تغییر کلاس `body.dark`، تمامی متغیرهای رنگی به صورت آنی تغییر می‌کنند و موتور رسم نیز از طریق تابع `getTC()` این مقادیر را در زمان اجرا (Runtime) از DOM استخراج می‌کند.

متغیر CSS	نقش	مثال مقدار (Dark Mode)
<code>canvas-bg--</code>	پس‌زمینه بوم	<code>#0f1724</code>
<code>actor-stroke--</code>	رنگ خط Actor	<code>cbd5e0#</code>
<code>class-fill--</code>	رنگ زمینه Class	<code>#1e2d40</code>
<code>seq-msg--</code>	رنگ پیام‌های Sequence	<code>cbd5e0#</code>
<code>selected--</code>	رنگ المان انتخاب‌شده	<code>#667eea</code>

✓ **مزیت این رویکرد:** افزودن یک تم جدید (مانند تم آبی، سبز یا High-Contrast) تنها با تغییر چند متغیر CSS امکان‌پذیر است و هیچ تغییری در منطق JavaScript نیاز نیست.

برای تشخیص اینکه ماوس روی کدام المان قرار دارد، از الگوی **Hit Testing** استفاده می‌شود. تابع `hitTest(x, y)` بر اساس نوع نمودار فعال، الگوریتم‌های متفاوتی را اجرا می‌کند:

Actor: محاسبه فاصله اقلیتی (Bounding Box 50×80 پیکسل).

Use Case (بیضی): استفاده از معادله استاندارد بیضی $(x^2/a^2) + (y^2/b^2) \geq 1$.

State/Action/Component (مستطیل با گوشه گرد): بررسی Bounding Box.

Decision (لوزی): استفاده از معادله لوزی $|x| + |y| \leq \text{size}$.

Class: بررسی بر اساس ارتفاع محاسبه‌شده داینامیک (`computedH`).

Swimlane: علاوه بر Hit Test بدنه، بررسی ۸ Handle برای Resize.

⚠ **چالش فنی:** در نمودار Class، ارتفاع به‌صورت داینامیک بر اساس تعداد Attributes و Methods محاسبه می‌شود. این مقدار در هر Draw در `cls._computedH` ذخیره می‌گردد تا در Hit Test نیز قابل استفاده باشد.

سیستم Persistence این پروژه کاملاً مبتنی بر **JSON File API** است و از **LocalStorage** یا **IndexedDB** استفاده نمی‌کند. این تصمیم معماری، مزایای زیر را به همراه دارد:

قابلیت اشتراک‌گذاری آسان: فایل JSON به راحتی قابل ارسال از طریق ایمیل، Git یا پیام‌رسان‌ها است.

نسخه‌بندی (Versioning): فایل‌ها می‌توانند در Git نگهداری و Diff شوند.

عدم وابستگی به مرورگر: با پاک‌سازی Cache مرورگر، داده‌ها از بین نمی‌روند.

ساختار قابل توسعه: فرمت JSON به راحتی قابل گسترش است.

// ساختار فایل خروجی JSON

```
{
  "version": "5.1",
  "fileTitle": "عنوان پروژه",
  "author": "هادی خزاعی اصل",
  "organization": "رایان پرداز کاوش",
  "createdAt": "2026-09-07T10:30:00.000Z",
  "preferences": { "gridType": "lines", "darkMode": true },
  "diagrams": [ /* آرایه نمودارها */ ]
}
```

Backward Compatibility 💡 در هنگام Load فایل، کد با استفاده از `|| operator`، فیلدهای اختیاری را مقداره‌ی پیش‌فرض می‌کند تا فایل‌های نسخه‌های قدیمی‌تر نیز بدون خطا بارگذاری شوند.

در این پروژه، چندین الگوی طراحی شناخته شده به صورت عملی پیاده سازی شده اند:

الگو	محل کاربرد	توضیح
Registry Pattern	DIAGRAM_TYPES	مرجع مرکزی تعریف انواع نمودار و قابلیت های آنها
State Pattern	آبجکت state	مدیریت وضعیت سراسری برنامه
Strategy Pattern	توابع drawUseCase, ... ,drawClassDiagram	انتخاب استراتژی رسم بر اساس نوع نمودار
Command Pattern	مدیریت ابزارها (Tools)	هر ابزار یک رفتار مشخص را در mousedown/mouseup اجرا می کند
Observer Pattern	Event Listeners	واکنش به رویدادهای Mouse، Keyboard و UI
Factory Pattern	() createNewDiagram	ساخت ساختار اولیه نمودار بر اساس نوع
Singleton Pattern	Canvas Context، State Object	تنها یک نمونه از منابع حیاتی در طول اجرا

📁 مدیریت چند نمودار (Multi-Tab): امکان باز کردن هم‌زمان چندین نمودار در تب‌های مجزا با امکان سوئیچ سریع.

🌙 تم تاریک/روشن: سوئیچ آبی با حفظ ترجیحات در فایل پروژه.

🗂️ کلیدهای میانبر: Ctrl+S (ذخیره)، Ctrl+N (جدید)، Delete (حذف)، G (Grid)، B (Sidebar).

🎨 Properties Panel: ویرایش زنده ویژگی‌های المان انتخاب‌شده در پایین صفحه.

📄 Class Editor Modal: ویرایشگر پیشرفته Attributes و Methods با Static، Visibility، و Type.

📏 Resize Handles: قابلیت تغییر اندازه Swimlane با ۸ Handle در گوشه‌ها و اضلاع.

🖨️ Export to PNG: خروجی با کیفیت از نمودار فعلی با قابلیت انتخاب رنگ پس‌زمینه یا شفافیت.

📐 Grid System: سه حالت (Lines, Dots, None) برای کمک به چیدمان دقیق المان‌ها.

🔗 Relationship Drawing: رسم روابط با Drag از مبدأ به مقصد و Prompt برای برچسب و Multiplicity.

دسته	نقاط قوت فعلی	پیشنهادهای بهبود
معماری	بدون وابستگی خارجی، سبک و سریع	ماژولار کردن کد با ES Modules برای نگهداری بهتر
عملکرد	اجرای روان در نمودارهای کوچک و متوسط	پیاپی سازی Dirty Rectangle Rendering برای نمودارهای بزرگ
UX	رابط کاربری شهودی با RTL کامل	افزودن Undo/Redo با Command History
ذخیره سازی	خروجی JSON قابل اشتراک	افزودن Auto-Save در LocalStorage به عنوان Backup
توسعه پذیری	Registry Pattern برای افزودن نمودار جدید	افزودن نمودارهای Package و Communication
خروجی	PNG با کیفیت	افزودن Export به PDF، SVG و XMI استاندارد

پروژه «ابزار رسم نمودارهای UML» یک نمونه موفق از توسعه یک اپلیکیشن تخصصی و حرفه‌ای با استفاده از تکنولوژی‌های پایه وب است. این ابزار با تکیه بر معماری State-Centric، موتور رسم مبتنی بر Canvas و سیستم Registry منعطف، توانایی مدل‌سازی کامل سیستم‌های نرم‌افزاری را در محیطی آفلاین و سبک فراهم می‌آورد.

نتیجه‌گیری معماری

این پروژه نشان می‌دهد که برای ساخت ابزارهای تخصصی، لزومی به استفاده از فریم‌ورک‌های سنگین نیست. با طراحی دقیق معماری، انتخاب الگوهای مناسب و تمرکز بر تجربه کاربری، می‌توان محصولی در سطح Enterprise تولید کرد که کاملاً مستقل، قابل نگهداری و توسعه‌پذیر باشد.

💡 **ارزش نهایی:** ابزاری کاملاً آفلاین، سبک، حرفه‌ای و مستقل که نیاز هر معمار و تحلیل‌گر نرم‌افزار را برای مدل‌سازی سریع و دقیق برطرف می‌سازد.

🔧 **گام بعدی:** برای توسعه نسخه‌های آتی، پیشنهاد می‌شود مازول‌های Undo/Redo، Export به فرمت‌های استاندارد (XMI, SVG) و افزودن نمودارهای Interaction اضافه گردد.

این مستند محرمانه بوده و صرفاً جهت استفاده داخلی تیم فنی تهیه شده است. هرگونه کپی‌برداری یا افشا بدون اجازه کتبی ممنوع است.

طراحی، تحلیل و توسعه: هادی خزاعی اصل

تاریخ تهیه: ۱۶ شهریور ۱۴۰۵ - ساعت ۱۰:۳۰ | نسخه: ۱.۰ | طبقه‌بندی: محرمانه / فنی