MolBuilder Studio

Design Document

3D Molecule Building Suite

Version 1.0 -- February 13, 2026

Prepared by: MolBuilder Engineering Team
Organization: Materia Foundation / Wedge Dev Co.

Confidentiality: Internal Use

molbuilder.studio

1. Executive Summary

MolBuilder Studio is a browser-based, three-dimensional molecular editing and visualization suite designed to bring the capabilities of professional desktop chemistry tools into a modern web application. The project targets medicinal chemists, computational chemists, students, and researchers who need to rapidly build, visualize, analyze, and export molecular structures without installing specialized desktop software.

The studio represents the graphical centerpiece of the MolBuilder ecosystem, which includes a production API deployed on Railway (molbuilder.io), a published Python SDK on PyPI (molbuilder-client), and an existing React dashboard application. MolBuilder Studio is a separate, purpose-built application hosted at molbuilder.studio, designed from the ground up for interactive 3D molecular work.

Project Scope

Phase 1 (MVP) delivers the core studio experience: a SMILES-driven workflow where users enter molecular notation, the API parses and optimizes the 3D geometry, and the studio renders the result in an interactive Three.js viewport with atom selection, property inspection, and file export capabilities. The MVP comprises approximately 42 source files across 15 component categories, building to a 1.3 MB production bundle.

Key Design Decisions

- **Three.js over 3Dmol.js:** The existing frontend uses 3Dmol.js for passive visualization. The studio requires full editor control (raycasting, selection, custom shaders, imperative scene graph manipulation), which necessitates a lower-level 3D framework. React Three Fiber provides a declarative React wrapper over Three.js, enabling component-based scene construction.
- **Separate application:** Rather than extending the existing frontend/, the studio is a standalone Vite project in studio/ with its own deployment. This ensures the dashboard and studio can evolve independently with different dependency trees and release cadences.
- **API-first architecture:** All molecular computation (SMILES parsing, 3D coordinate generation, property calculation) happens server-side via the existing FastAPI endpoints. The studio is a pure rendering and interaction layer, keeping the client lightweight and computation-free.
- **Client-side export:** File export (MOL V2000, SDF, XYZ) is generated entirely in the browser from the 3D data already fetched from the API. This eliminates unnecessary round-trips and allows offline export after initial data load.

Success Criteria

- Zero lint errors, zero TypeScript errors, clean production build
- Functional register/login flow connecting to the production Railway API
- Interactive 3D rendering of any valid SMILES string with CPK-colored atoms and bonds
- Click-to-select atoms with visual feedback (emissive glow, HUD overlay)
- Accurate property display (molecular weight, LogP, Lipinski, functional groups)

- Valid MOL V2000 / SDF / XYZ file downloads from client-side generation

2. System Architecture

MolBuilder Studio operates within a distributed architecture spanning two cloud platforms and two domain names. The system follows a client-server model where the studio application is a static single-page application (SPA) served from Vercel's global CDN, communicating with a FastAPI backend hosted on Railway.

Architecture Diagram

```
molbuilder.studio (Vercel CDN)             molbuilder.io (Railway)
  +-----------------------------+            +-------------------------+
  |  studio/ (React + R3F)      |            |  saas/app/ (FastAPI)    |
  |                             |   HTTPS    |                         |
  |  Three.js 3D viewport      |<---------->|  /v1/molecule/*         |
  |  SMILES input + panels      |   /api/*   |  /v1/auth/*             |
  |  Auth (JWT via API)         |            |  /v1/billing/*          |
  |  Client-side MOL/SDF export |            |  SQLite + Volume        |
  +-----------------------------+            +-------------------------+
       |                                          |
       v                                          v
  Static assets (JS/CSS/HTML)              Persistent /data/ volume
  served from global edge nodes            (users DB, molecules DB,
  with SPA routing rewrites                 audit trail)
```

Request Flow

The following sequence describes the complete lifecycle of a molecule visualization request:

- **1. Authentication:** User registers via POST /v1/auth/register (returns API key) or logs in by exchanging an existing API key for a JWT via POST /v1/auth/token. The JWT is stored in Zustand state (memory only) while the API key is persisted in localStorage.
- **2. SMILES Submission:** User enters a SMILES string (e.g., 'c1ccccc1' for benzene). The studio sends POST /v1/molecule/from-smiles with the JWT in the Authorization header.
- **3. Server Processing:** The FastAPI backend parses the SMILES using the molbuilder core library, generates a molecule graph with atom positions and bond connectivity, optimizes 3D coordinates, and returns a MoleculeResponse with the molecule ID.
- **4. Parallel Data Fetch:** The studio immediately fires two parallel requests: GET /v1/molecule/{id}/3d (returns atom positions and bond topology) and GET /v1/molecule/{id}/properties (returns molecular weight, LogP, Lipinski properties, functional groups, etc.).
- **5. 3D Rendering:** The workspace store receives the 3D response and triggers a React re-render. The MoleculeScene component maps atoms to AtomMesh spheres and bonds to BondMesh cylinders, centered at the molecule's centroid for natural orbital rotation.
- **6. Interaction:** R3F's built-in raycasting detects pointer events on mesh components. Click events trigger atom selection in the editor store, which propagates emissive highlights and HUD overlays. OrbitControls handle camera orbit, pan, and zoom.

Network Architecture

In production, the studio is deployed to Vercel with API proxy rewrites defined in vercel.json. Requests to /api/* are proxied to the Railway backend at molbuilder-api-production.up.railway.app/api/*. This avoids CORS preflight overhead for same-origin requests in the browser. Direct CORS is also configured as a fallback.

In development, Vite's built-in proxy server handles the same /api/* rewriting, allowing the studio to run on localhost:5174 while transparently forwarding API requests to the production Railway endpoint.

Security Model

Authentication uses a two-token system. API keys are long-lived, HMAC-hashed secrets stored in the server's SQLite database. JWTs are short-lived (60-minute expiry) bearer tokens signed with HS256. The studio's auth store implements automatic token refresh: if a JWT will expire within 5 minutes, getValidToken() transparently exchanges the stored API key for a fresh JWT before any API call proceeds.

Rate limiting is enforced server-side per tier: Free (10 RPM), Pro (60 RPM), Team (120 RPM), Academic (30 RPM), Enterprise (300 RPM). Expensive endpoints (retrosynthesis, process engineering) have separate hourly quotas. Registration and token endpoints have IP-based rate limits (5 RPM and 20 RPM respectively) to prevent abuse.

3. Technology Stack

The technology stack was selected to balance rendering performance, developer ergonomics, ecosystem maturity, and bundle size. Each choice is justified below with references to official documentation and community metrics.

Frontend Runtime & Framework

React 19

React 19 (released December 5, 2024, latest 19.2 as of October 2025) is Meta's UI library for building component-based user interfaces. Key features leveraged by the studio include concurrent rendering (prevents long Three.js scene updates from blocking UI), Suspense boundaries for lazy-loaded pages, and the mature hooks API (useState, useCallback, useMemo, useEffect) for managing component state and side effects.

React Team. 'React 19.2.' react.dev/blog, October 1, 2025. https://react.dev/blog/2025/10/01/react-19-2

TypeScript 5.7

The entire codebase is written in strict TypeScript with noUnusedLocals, noUnusedParameters, and noUncheckedIndexedAccess enabled. TypeScript provides compile-time type safety for API response types, store interfaces, component props, and Three.js vector math, catching entire categories of runtime errors before they reach the browser.

Vite 6

Vite is a next-generation frontend build tool created by Evan You (creator of Vue.js). It provides a development server with native ES module support and near-instant hot module replacement (HMR), plus a Rollup-based production bundler with tree-shaking and code splitting. Vite 6 requires Node.js 20.19+ and supports first-party plugins for React and Tailwind CSS. Build times for the studio are under 15 seconds.

Vite. Official documentation. https://vite.dev/

3D Rendering

Three.js

Three.js is a cross-browser JavaScript 3D graphics library that creates GPU-accelerated animations using WebGL, exposed via the HTML5 Canvas element. Created by Ricardo Cabello (mrdoob) and first released on GitHub in April 2010, it has grown to over 111,000 GitHub stars with 1,950+ contributors, making it the dominant JavaScript 3D library. Three.js abstracts WebGL's low-level shader programming into a scene graph of meshes, geometries, materials, lights, and cameras.

Three.js. GitHub repository. https://github.com/mrdoob/three.js. See also: https://en.wikipedia.org/wiki/Three.js

WebGL 2.0

WebGL (Web Graphics Library) is a cross-platform, royalty-free JavaScript API for rendering 2D and 3D graphics in the browser, maintained by the Khronos Group. WebGL 2.0, based on OpenGL ES 3.0, achieved universal major browser support on February 9, 2022 (Chrome, Firefox, Safari 15, Edge). Three.js uses WebGL as its primary rendering backend, with experimental WebGPU support in development.

Khronos Group. 'WebGL Specification.' https://www.khronos.org/webgl/. See also: https://en.wikipedia.org/wiki/WebGL

React Three Fiber (R3F)

React Three Fiber is a React renderer for Three.js maintained by the Poimandres (pmndrs) open-source collective (~30,100 GitHub stars). It allows declarative, component-based construction of 3D scenes using React's ecosystem. Instead of imperative Three.js code, developers write JSX: <mesh>, <sphereGeometry>, <meshStandardMaterial>, etc. R3F manages the render loop, raycasting, and scene lifecycle automatically.

The companion library @react-three/drei provides pre-built helpers including OrbitControls (camera interaction), Text (3D text rendering), and various geometry utilities. The studio uses R3F v9 with drei v10, both compatible with React 19.

Poimandres. 'React Three Fiber.' https://r3f.docs.pmnd.rs/. GitHub: https://github.com/pmndrs/react-three-fiber

State Management

Zustand

Zustand is a lightweight state management library for React, also maintained by the Poimandres collective (~57,000 GitHub stars). At under 1 KB minified, it provides a hook-based API with no providers, no boilerplate, and no reducers. State is created as a simple hook using the create() function, and components subscribe to specific slices of state via selectors, minimizing re-renders.

The studio uses three Zustand stores: auth-store (authentication state with localStorage persistence), workspace-store (loaded molecules and active selection), and editor-store (tool selection, viewport settings, hover/selection state). Zustand's getState() method enables state access outside React components, which is critical for the useMolecule hook's async API call chains.

Poimandres. 'Zustand.' https://github.com/pmndrs/zustand

Styling

Tailwind CSS v4

Tailwind CSS v4 (released January 2025) is a utility-first CSS framework that provides atomic CSS classes for rapid UI construction. Version 4 introduces a CSS-first configuration model using @theme directives instead of JavaScript config files, a new engine built on Lightning CSS (delivering 5x faster full builds and 100x faster incremental builds), automatic content detection, and native CSS features including cascade layers, @property registered custom properties, and color-mix().

The studio defines a custom dark theme via CSS custom properties in index.css, including colors for background (#0a0a0a), panels (#111111), borders (#262626), text tiers (primary, secondary, muted), and accent blue (#3b82f6). These variables are referenced through Tailwind utility classes like bg-bg, text-text-primary, and border-border.

Tailwind Labs. 'Tailwind CSS v4.0.' https://tailwindcss.com/blog/tailwindcss-v4

Backend & API

FastAPI

The MolBuilder API is built with FastAPI, a modern, high-performance Python web framework created by Sebastian Ramirez. Based on Starlette (ASGI) and Pydantic (data validation), FastAPI provides automatic interactive API documentation (Swagger UI at /docs), request validation via Python type hints, native async/await support, and OAuth2 + JWT authentication. With over 95,000 GitHub stars and 1 billion+ PyPI downloads, FastAPI is one of the most widely adopted Python API frameworks.

Ramirez, S. 'FastAPI.' https://fastapi.tiangolo.com/. GitHub: https://github.com/fastapi/fastapi

Deployment Platforms

Vercel (Frontend Hosting)

Vercel is a cloud platform optimized for frontend deployment, providing a global CDN, automatic HTTPS, serverless/edge functions, and preview deployments on every pull request. The studio's Vite build output (static HTML/JS/CSS) is served from Vercel's edge network with SPA routing rewrites and API proxy rules defined in vercel.json.

Vercel. https://vercel.com/

Railway (Backend Hosting)

Railway is a full-stack cloud platform for deploying web applications with automatic builds, persistent volumes, managed databases, and custom domains. The MolBuilder API runs as a Docker container on Railway with a persistent /data/ volume for SQLite databases (users, molecules, audit trail). Railway raised $100M in January 2026 for intelligent cloud infrastructure.

Railway. https://railway.com/. See also: SiliconANGLE, 'Railway gets $100M,' January 22, 2026.

Dependency Summary

Package	Version	Purpose	License
react	^19.0.0	UI framework	MIT
react-dom	^19.0.0	DOM renderer	MIT
react-router-dom	^7.1.0	Client-side routing	MIT
three	^0.172.0	3D graphics engine	MIT
@react-three/fiber	^9.0.0	React renderer for Three.js	MIT
@react-three/drei	^10.0.0	R3F helper components	MIT
zustand	^5.0.3	State management	MIT
clsx	^2.1.0	Class name utility	MIT
typescript	^5.7.3	Type checker	Apache-2.0
vite	^6.0.0	Build tool	MIT
tailwindcss	^4.0.0	CSS framework	MIT
eslint	^9.18.0	Linter	MIT

4. 3D Rendering Pipeline

The 3D rendering pipeline transforms API response data (atom positions, bond topology) into an interactive, visually accurate molecular scene. This chapter describes each stage of the pipeline, from data ingestion to pixel output.

Molecular Representation Models

Ball-and-Stick Model

The studio's default rendering style is the ball-and-stick model, one of the oldest and most intuitive representations of molecular structure. In this model, atoms are depicted as spheres and covalent bonds as cylindrical rods connecting them. The first physical ball-and-stick models were created by August Wilhelm von Hofmann in 1865, using colored croquet balls and brass tubes at London's Royal Institution.

The ball-and-stick model strikes a balance between spatial accuracy and visual clarity: it conveys three-dimensional geometry and connectivity while leaving interstitial space visible (unlike space-filling models). The studio also supports stick-only rendering (thinner cylinders, no visible atom spheres) and space-filling rendering (van der Waals scaled spheres with no explicit bonds).

Wikipedia. 'Ball-and-stick model.' https://en.wikipedia.org/wiki/Ball-and-stick_model. See also: Hofmann, A.W. (1865). Royal Institution lectures.

CPK Color Convention

Atoms are colored according to the CPK (Corey-Pauling-Koltun) convention, a standardized color scheme for distinguishing chemical elements in molecular visualization. The convention is named after Robert Corey and Linus Pauling, who designed space-filling calotte models at Caltech in the 1950s, and Walter Koltun, who patented an improved version at UC Berkeley in 1965.

Standard CPK colors include: hydrogen (white, #FFFFFF), carbon (gray, #909090), nitrogen (blue, #3050F8), oxygen (red, #FF0D0D), fluorine (green, #90E050), sulfur (yellow, #FFFF30), phosphorus (orange, #FF8000), and chlorine (green, #1FF01F). The studio implements the full CPK color table for 60 elements in lib/cpk-colors.ts, with a magenta fallback (#FF1493) for unknown elements.

Wikipedia. 'CPK coloring.' https://en.wikipedia.org/wiki/CPK_coloring. Colors derived from molbuilder/core/element_properties.py.

Van der Waals Radii

Atom sphere sizes are derived from van der Waals radii -- the radius of an imaginary hard sphere representing the distance of closest approach for non-covalently-bonded atoms. Named after Johannes Diderik van der Waals (1910 Nobel Prize in Physics), these radii define the effective 'size' of atoms in molecular visualization. The studio uses Bondi's 1964 consensus values, scaled down for ball-and-stick rendering: hydrogen 0.25 A, carbon 0.4 A, nitrogen 0.38 A, oxygen 0.36 A, with a 0.4 A default for unlisted elements.

Bondi, A. 'van der Waals Volumes and Radii.' J. Phys. Chem., 1964, 68(3), 441-451. See also: https://en.wikipedia.org/wiki/Van_der_Waals_radius

Scene Graph Construction

The MoleculeScene component reads the active molecule's 3D structure from the workspace store and constructs a Three.js scene graph. The process involves:

- **1. Centroid calculation:** The arithmetic mean of all atom positions is computed. The entire molecule group is translated by the negative centroid, centering the molecule at the world origin for natural orbital rotation around the molecule's center of mass.
- **2. Hydrogen filtering:** If showHydrogens is false in the editor store, hydrogen atoms and their bonds are excluded from the scene. This is a common visualization preference for clarity in larger molecules.
- **3. Atom mesh generation:** Each visible atom becomes an AtomMesh component -- a <mesh> with <sphereGeometry> (32x32 segments) and <meshStandardMaterial> colored by the CPK map. Selected atoms receive an emissive blue glow (emissiveIntensity=0.5), and hovered atoms scale up by 1.15x with a subtle emissive tint.
- **4. Bond mesh generation:** Each visible bond becomes a BondMesh component. Single bonds are a single cylinder, double bonds are two parallel offset cylinders (0.1 A apart), and triple bonds are three cylinders (center + two at 0.14 A offset). Each bond is split at its midpoint into two half-cylinders, one colored as atom A and one as atom B.

Bond Geometry Mathematics

Positioning a cylinder between two arbitrary 3D points requires computing the midpoint (for position), length (for cylinder height), and orientation quaternion (to rotate the cylinder from its default Y-axis alignment to the bond direction). The computeBondTransform function in lib/bond-geometry.ts performs this calculation:

```
function computeBondTransform(posA, posB):
    direction = normalize(posB - posA)
    length = distance(posA, posB)
    midpoint = (posA + posB) / 2
    quaternion = Quaternion.setFromUnitVectors(Y_AXIS, direction)
    return { position: midpoint, quaternion, length }
```

For double and triple bonds, a perpendicular offset vector is computed using cross-product geometry. The function computeBondOffset finds a vector perpendicular to the bond axis by crossing the bond direction with an arbitrary reference vector (X-axis, or Y-axis if the bond is near-parallel to X). This offset is then scaled and applied to both endpoint positions to create parallel cylinders.

Lighting Model

The viewport uses a three-light setup optimized for molecular visualization:

- Ambient light (intensity 0.4): Provides baseline illumination so no face is completely dark, simulating ambient environmental light.
- Primary directional light (position [10, 10, 10], intensity 0.8): The main light source, creating highlights and depth cues on atom spheres through specular reflection.
- Fill directional light (position [-5, -5, -10], intensity 0.3): A dimmer light from the opposite direction, softening shadows and revealing details on the back side of the molecule.

Atom materials use meshStandardMaterial with roughness=0.4 and metalness=0.1, producing a subtle glossy appearance appropriate for scientific visualization. Bond materials use roughness=0.5 for a slightly more matte finish.

Camera & Interaction Controls

The viewport camera is a perspective camera positioned at [0, 0, 15] with a 50-degree field of view. OrbitControls from @react-three/drei provides three interaction modes:

- Orbit: Left mouse button rotates the camera around the molecule center.
- Pan: Right mouse button or arrow keys translate the camera laterally.
- Zoom: Scroll wheel moves the camera closer/further from the target.

Damping is enabled (dampingFactor=0.1) for smooth, physically-plausible camera deceleration after user input. Clicking empty space (onPointerMissed) clears the atom selection.

Three.js. 'OrbitControls.' https://threejs.org/docs/#examples/en/controls/OrbitControls

5. Component Architecture

The studio follows a layered component architecture with clear separation of concerns. Components are organized into six categories: viewport (3D rendering), panels (data display/input), toolbar (navigation/tools), layout (grid structure), auth (authentication), and ui (shared primitives).

Component Hierarchy

```
App
+-- Routes
    +-- AuthGuard
    |   +-- StudioPage
    |       +-- StudioLayout (CSS Grid)
    |           +-- MenuBar          (File | Edit | View menus)
    |           +-- ToolPalette      (Select | Rotate | Measure)
    |           +-- Viewport         (R3F Canvas)
    |           |   +-- MoleculeScene
    |           |   |   +-- AtomMesh[]   (per atom)
    |           |   |   +-- BondMesh[]   (per bond)
    |           |   +-- OrbitControls
    |           |   +-- ViewportOverlay  (hover HUD)
    |           |   +-- SelectionOutline (selection info)
    |           +-- Sidebar
    |           |   +-- SmilesPanel
    |           |   +-- PropertiesPanel
    |           |   +-- AtomsPanel
    |           |   +-- ExportPanel
    |           +-- StatusBar
    +-- LoginPage
    |   +-- LoginForm
    +-- RegisterPage
        +-- RegisterForm
```

Viewport Components

Viewport.tsx

The root 3D container. Wraps an R3F <Canvas> with perspective camera, three lights, MoleculeScene, and OrbitControls. Overlays ViewportOverlay and SelectionOutline as absolute-positioned HTML divs on top of the canvas. Shows a 'No molecule loaded' placeholder when no structure data is available.

MoleculeScene.tsx

Reads the active molecule from workspace-store and the selection state from editor-store. Computes the centroid of all atoms and wraps the scene in a <group> translated by the negative centroid. Maps visible atoms to AtomMesh components and visible bonds to BondMesh components. Handles hydrogen visibility filtering.

AtomMesh.tsx

Renders a single atom as a Three.js sphere. Props include the atom data (symbol, position, index), selection state, label visibility, and render style. Implements onClick for selection (with shift-click for multi-select), onPointerOver/Out for hover effects (cursor change, scale animation, emissive tint), and optional Text label above the atom showing the symbol and index.

BondMesh.tsx

Renders a single bond as one or more cylinders (1 for single, 2 for double, 3 for triple). Uses computeBondTransform and computeBondOffset from bond-geometry.ts to position and orient each cylinder. Each bond is split into two half-cylinders at the midpoint, colored by the respective atom's CPK color. Hidden in spacefill render mode.

Panel Components

SmilesPanel.tsx

Input panel for SMILES strings. Features a text input with a Parse button, error display via Alert component, and six preset example buttons (benzene, aspirin, caffeine, ethanol, ibuprofen, paracetamol). Uses the useMolecule hook to trigger the parse-fetch-render pipeline. Clicking an example button both sets the input value and triggers parsing.

PropertiesPanel.tsx

Displays computed molecular properties from the API in a two-column grid layout. Shows formula, molecular weight (Da), LogP, hydrogen bond donors/acceptors, rotatable bonds, topological polar surface area (TPSA), and heavy atom count. Features a Lipinski Rule of Five badge (Pass/Fail with violation count) and a tag cloud of functional groups.

AtomsPanel.tsx

A scrollable table listing all atoms with their index, symbol, and 3D coordinates (X, Y, Z in Angstroms). Rows are clickable for atom selection (with shift-click for multi-select), and selected rows are highlighted with an accent background. Respects the hydrogen visibility toggle. Collapsed by default to save sidebar space.

ExportPanel.tsx

Three download buttons for MOL V2000, SDF, and XYZ file formats. Each button triggers the corresponding function from the useExport hook, which generates the file content client-side and triggers a browser download. Buttons are disabled when no structure is loaded.

PanelContainer.tsx

A reusable collapsible wrapper for sidebar panels. Features a header row with the panel title and a chevron icon that toggles between expanded and collapsed states. Panels can be configured to start open (default) or closed via the defaultOpen prop.

Toolbar Components

MenuBar.tsx

A horizontal menu bar with three dropdown menus (File, Edit, View). File offers export actions and sign-out. Edit provides Clear Selection. View toggles hydrogen visibility, atom labels, and render style (ball-and-stick, stick, spacefill) with checkmark indicators for active options. Menus open on click and switch on hover when already open. Clicking outside closes the active menu.

ToolPalette.tsx

A narrow vertical strip (48px wide) with tool buttons for Select, Rotate, and Measure modes. The active tool is highlighted with an accent blue background. Tool keyboard shortcuts (V, R, M) are shown in tooltips.

StatusBar.tsx

A bottom bar displaying the current molecule name (or SMILES if unnamed), atom count, bond count, molecular formula, active tool, and render style. Uses monospace font at 10px for a compact, IDE-like appearance.

Layout

StudioLayout.tsx

The master layout component uses CSS Grid with three rows and three columns:

```
grid-template-rows:    40px  1fr   28px
grid-template-columns: 48px  1fr   320px

+---------------------------------------------------+
|  MenuBar (col-span-3, 40px height)                |
+------+-------------------------------+-----------+
| Tool |                               | Sidebar   |
| Pal. |    3D Viewport (fills)        | (320px)   |
| 48px |                               | Scrollable|
+------+-------------------------------+-----------+
|  StatusBar (col-span-3, 28px height)              |
+---------------------------------------------------+
```

The viewport fills all available space between the tool palette and sidebar. The sidebar is scrollable with overflow-y-auto for molecules with many properties or atoms. The entire layout fills the viewport (100vh x 100vw) with overflow:hidden on the body to prevent scrollbars.

6. State Management Design

The studio uses three Zustand stores to manage application state. Each store is a standalone module that can be accessed both inside React components (via the hook) and outside (via getState()). This dual-access pattern is critical for async operations like API calls that run outside the React render cycle.

Auth Store (auth-store.ts)

Manages authentication state: API key, JWT, email, tier, and authentication status. The API key and email are persisted to localStorage under the key 'molbuilder_studio_auth'. The JWT is held in memory only (never persisted) for security.

Field	Type	Persisted	Description
apiKey	string | null	Yes	Long-lived API key
jwt	string | null	No	Short-lived JWT (60 min)
jwtExpiresAt	number | null	No	JWT expiry timestamp (ms)
email	string | null	Yes	User email address
tier	string | null	Yes	Subscription tier
isAuthenticated	boolean	Derived	true if apiKey is set

Key methods: setApiKey() persists credentials and sets isAuthenticated. login() exchanges an API key for a JWT. logout() clears all state and localStorage. getValidToken() is the primary method called before API requests -- it checks if the current JWT has more than 5 minutes remaining, and if not, transparently refreshes it using the stored API key.

Workspace Store (workspace-store.ts)

Manages the collection of loaded molecules and tracks which one is active. Uses a Map<string, MoleculeEntry> keyed by molecule ID (from the API). Each entry contains the original SMILES, name, MoleculeResponse, and optional 3D structure and properties (which arrive asynchronously after the initial parse).

Field	Type	Description
molecules	Map<string, MoleculeEntry>	All loaded molecules by ID
activeId	string | null	ID of the currently displayed molecule

Methods: addMolecule() creates a new entry and sets it as active. setStructure() and setProperties() update an existing entry with async data. removeMolecule() deletes an entry and clears activeId if it was the active molecule. getActive() returns the full MoleculeEntry for the active molecule.

Editor Store (editor-store.ts)

Manages the interactive editing state: active tool, selection sets, hover targets, and viewport settings. This store is read by 3D viewport components (AtomMesh, MoleculeScene) and written by interaction handlers and toolbar controls.

Field	Type	Default	Description

tool	Tool enum	SELECT	Active tool mode
selectedAtoms	Set<number>	(empty)	Selected atom indices
selectedBonds	Set<string>	(empty)	Selected bond keys (i-j)
hoveredAtom	number | null	null	Currently hovered atom
hoveredBond	string | null	null	Currently hovered bond
showHydrogens	boolean	true	Hydrogen visibility
showLabels	boolean	false	Atom label visibility
renderStyle	RenderStyle	ball-and-stick	Rendering mode

Selection methods support multi-select via the shift key: selectAtom(idx, multi) either replaces the selection or toggles the atom in/out of the existing set. Multi-selection enables measuring distances between atoms and bulk operations in future phases.

7. Molecular Data & SMILES Notation

SMILES Overview

SMILES (Simplified Molecular-Input Line-Entry System) is the primary input method for the studio. Developed by David Weininger at the USEPA Mid-Continent Ecology Division Laboratory in the 1980s, SMILES encodes molecular structures as compact ASCII strings. For example, 'c1ccccc1' represents benzene (aromatic six-membered ring), 'CCO' represents ethanol, and 'CC(=O)Oc1ccccc1C(=O)O' represents aspirin.

SMILES notation captures: atom types (elements), bond orders (single, double, triple, aromatic), ring closures (numbered pairs), branching (parentheses), charges, and stereochemistry (@ for chirality, / and \ for cis/trans). The MolBuilder API parses SMILES into a molecular graph with atom symbols, formal charges, hybridization states, and bond orders, then generates optimized 3D coordinates through force-field minimization.

Weininger, D. 'SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules.' J. Chem. Inf. Comput. Sci., 1988, 28(1), 31-36. https://en.wikipedia.org/wiki/Simplified_Molecular_Input_Line_Entry_System

API Data Model

The API returns molecular data in three structured responses, mirrored exactly as TypeScript interfaces in the studio's api/types.ts:

MoleculeResponse

```
{
  "id": "mol_abc123",       // Unique molecule ID
  "name": "benzene",        // Resolved name (or SMILES if unknown)
  "smiles": "c1ccccc1",     // Canonical SMILES
  "num_atoms": 12,          // Total atom count (including H)
  "num_bonds": 12           // Total bond count
}
```

Molecule3DResponse

```
{
  "id": "mol_abc123",
  "atoms": [
    { "index": 0, "symbol": "C", "position": [1.21, 0.70, 0.00],
      "hybridization": "sp2", "formal_charge": 0 },
    ...
  ],
  "bonds": [
    { "atom_i": 0, "atom_j": 1, "order": 1.5, "rotatable": false },
    ...
  ]
}
```

MoleculePropertiesResponse

```
{
  "id": "mol_abc123",
  "smiles": "c1ccccc1",
  "formula": "C6H6",
  "molecular_weight": 78.11,
  "logp": 1.56,
  "hbd": 0,                     // Hydrogen bond donors
  "hba": 0,                     // Hydrogen bond acceptors
  "rotatable_bonds": 0,
  "tpsa": 0.0,                  // Topological polar surface area
  "heavy_atom_count": 6,
  "lipinski_violations": 0,
  "lipinski_pass": true,
  "functional_groups": ["aromatic ring"]
}
```

Lipinski's Rule of Five

The properties panel prominently displays Lipinski's Rule of Five compliance. Published by Christopher Lipinski in 1997, this rule provides a rapid estimate of oral bioavailability for drug candidates. A compound is considered drug-like if it satisfies at least three of the following four criteria (all multiples of 5):

- **Molecular weight:** <= 500 Daltons
- **LogP:** <= 5 (octanol-water partition coefficient)
- **Hydrogen bond donors:** <= 5 (sum of OH and NH groups)
- **Hydrogen bond acceptors:** <= 10 (sum of N and O atoms)

Compounds violating two or more rules are predicted to have poor oral absorption or membrane permeation. The MolBuilder API computes these properties server-side and returns the violation count and pass/fail status.

Lipinski, C.A., Lombardo, F., Dominy, B.W., Feeney, P.J. 'Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings.' Adv. Drug Deliv. Rev., 1997, 23(1-3), 3-25. DOI: 10.1016/S0169-409X(96)00423-1

8. File Export Formats

The studio generates three standard chemical file formats entirely client-side from the 3D structure data already fetched from the API. This eliminates server round-trips for export and enables offline downloading after initial data load.

MOL V2000 Format

The MOL file format (also known as MDL Molfile) is part of the CTfile (Chemical Table file) family maintained by BIOVIA (a subsidiary of Dassault Systemes). The V2000 variant is a text-based format with fixed-width fields for atom coordinates, bond connectivity, and molecular properties. Originally created at Molecular Design Limited (MDL), the format has been the de facto standard for chemical structure exchange since the 1980s.

The studio generates V2000 MOL files with the following structure:

```
molecule_name                        <- Name from API response
  MolBuilder3D                       <- Program identifier
                                     <- Comment line (blank)
  6 6  0  0  0  0  0  0  0  0999 V2000  <- Counts line
    1.2100    0.7000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0  <- Atom block
    ...                              <- (x, y, z, symbol, mass diff, charge, ...)
  1  2  2  0  0  0  0                <- Bond block (atom_i, atom_j, order, ...)
  ...
M  END                               <- Terminator
```

Key implementation details: Atom indices are converted from 0-based (API) to 1-based (MOL spec). Bond orders are rounded to the nearest integer and clamped to [1, 3]. Coordinates are formatted to 4 decimal places in 10-character fixed-width fields.

BIOVIA. 'CTfile Formats, BIOVIA Databases 2020.' https://discover.3ds.com/sites/default/files/2020-08/biovia_ctfileformats_2020.pdf. See also: https://en.wikipedia.org/wiki/Chemical_table_file

SDF Format

The Structure-Data Format (SDF) extends the MOL format to include associated data fields. An SDF file begins with a complete MOL block, followed by data items in the format '> <FIELD_NAME>' / value / blank line, and terminated by '$$$$'. The studio appends formula, molecular weight, SMILES, and LogP as data fields when properties are available.

XYZ Format

The XYZ format is a minimal plain-text format for atomic coordinates, developed in 1990 at the University of Minnesota as part of the XMol visualization software. It consists of: line 1 = atom count, line 2 = comment (molecule name), followed by one line per atom with the element symbol and X, Y, Z coordinates in Angstroms. The format intentionally omits connectivity information, making it lightweight and universally supported by quantum chemistry codes (Gaussian, ORCA, etc.).

Wikipedia. 'XYZ file format.' https://en.wikipedia.org/wiki/XYZ_file_format

9. Chemical Property Analysis

The studio's Properties panel displays a comprehensive set of molecular descriptors computed server-side by the MolBuilder core library. These descriptors support drug discovery workflows, SAR (structure-activity relationship) analysis, and educational applications.

Displayed Properties

Property	Unit	Description
Molecular Formula	--	Hill system notation (e.g., C6H6)
Molecular Weight	Daltons	Sum of atomic weights
LogP	dimensionless	Octanol-water partition coefficient
HBD	count	Hydrogen bond donor count (NH, OH)
HBA	count	Hydrogen bond acceptor count (N, O)
Rotatable Bonds	count	Non-ring single bonds with > 1 heavy neighbor
TPSA	Angstrom^2	Topological polar surface area
Heavy Atom Count	count	Non-hydrogen atom count
Lipinski Violations	count	Number of Rule of Five violations
Lipinski Pass	boolean	True if <= 1 violation
Functional Groups	list	Detected functional groups (e.g., hydroxyl)

Drug-Likeness Assessment

The Lipinski compliance indicator serves as a rapid drug-likeness screen. In drug discovery, ADMET (Absorption, Distribution, Metabolism, Excretion, and Toxicity) properties are critical determinants of clinical success. Undesirable ADMET properties are a leading cause of clinical trial failure, making early-stage computational filtering essential. The Rule of Five focuses on the absorption/permeability dimension of ADMET, filtering compounds that are unlikely to be orally bioavailable.

Wikipedia. 'ADME.' https://en.wikipedia.org/wiki/ADME. See also: Lipinski, C.A. et al. (1997). Adv. Drug Deliv. Rev., 23(1-3), 3-25.

10. Authentication & Security

Authentication Flow

The studio implements a two-step authentication flow that mirrors the existing frontend application:

- **1. Registration:** POST /v1/auth/register with email address. The API creates a free-tier account and returns an API key (format: mb_...). The key is displayed once and stored in localStorage. Server-side, the API key is HMAC-hashed before storage (defense-in-depth over plain SHA256).
- **2. Login / Token Exchange:** POST /v1/auth/token with the API key in the request body. The API validates the key, generates a JWT (HS256, 60-minute expiry), and returns it with the expiry duration. The JWT is stored in Zustand memory state (never localStorage) and used as a Bearer token for all subsequent API requests.
- **3. Auto-Refresh:** Before each API call, getValidToken() checks if the JWT will expire within 5 minutes. If so, it transparently re-exchanges the stored API key for a fresh JWT. If the exchange fails (key revoked/expired), the user is logged out automatically.

RBAC (Role-Based Access Control)

The API enforces three roles: admin (full access), chemist (standard usage), and viewer (read-only). The /register endpoint only creates free/chemist keys. Admin keys are bootstrapped on deployment via the admin_bootstrap_email environment variable. The studio does not expose role management UI in Phase 1; all studio users operate at their assigned tier and role.

Rate Limiting

Tier	RPM (General)	Expensive/Hour	Price
Free	10	5	$0
Pro	60	30	$49/mo
Team	120	60	$199/mo
Academic	30	15	$0
Enterprise	300	200	$499/mo

CORS Configuration

Cross-Origin Resource Sharing (CORS) is configured on the Railway API to accept requests from the studio domain. The allowed_redirect_hosts setting in saas/app/config.py includes molbuilder.studio and www.molbuilder.studio. Additionally, the CORS_ORIGINS environment variable on Railway must be set to include https://molbuilder.studio.

In production, Vercel's API proxy rewrites (/api/* -> Railway) make most requests same-origin, bypassing CORS entirely. Direct CORS headers are a fallback for any requests that bypass the proxy.

11. Deployment Architecture

Vercel Configuration

The studio is deployed as a static Vite build on Vercel. The vercel.json file defines two rewrite rules:

```
{
  "rewrites": [
    {
      "source": "/api/:path*",
      "destination": "https://molbuilder-api-production.up.railway.app/api/:path*"
    },
    {
      "source": "/((?!assets/).*)",
      "destination": "/index.html"
    }
  ]
}
```

- **API Proxy:** All requests to /api/* are proxied to the Railway backend. This avoids CORS preflight requests and keeps the browser's origin consistent.
- **SPA Fallback:** All non-asset routes are rewritten to index.html, enabling client-side routing via react-router-dom. The negative lookahead (?!assets/) ensures that static assets (JS, CSS, images) are served directly from the CDN.

Vercel deployment settings: root directory = studio, build command = npm run build, output directory = dist, Node.js version = 22. Custom domain: molbuilder.studio.

Railway Configuration

The MolBuilder API runs on Railway as a Docker container defined by saas/Dockerfile. Key configuration includes a persistent /data/ volume for SQLite databases, environment variables for secrets (JWT_SECRET_KEY, API_KEY_HMAC_SECRET, STRIPE_SECRET_KEY, etc.), and a railway.toml file for deployment settings.

Auto-deployment from GitHub is configured via .github/workflows/deploy.yml, which triggers on pushes to main that modify saas/ or molbuilder/ directories. The RAILWAY_TOKEN secret in the GitHub repository enables the deployment workflow.

DNS & Domain Architecture

Domain	Host	Purpose
molbuilder.studio	Vercel	3D Studio application
molbuilder.io	Railway	API backend
app.molbuilder.io	Vercel	Dashboard frontend

12. CI/CD Pipeline

Studio CI Workflow

The studio has a dedicated GitHub Actions workflow at .github/workflows/studio-test.yml that runs on pushes and pull requests modifying files under studio/. The workflow performs three checks:

- **Lint:** ESLint with TypeScript-ESLint, React hooks, and React Refresh rules.
- **Typecheck:** tsc --noEmit with strict mode, noUnusedLocals, noUnusedParameters.
- **Build:** tsc -b && vite build -- full production build verification.

```
name: Studio CI
on:
  push:
    paths: ["studio/**"]
  pull_request:
    paths: ["studio/**"]
jobs:
  lint-typecheck-build:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: studio
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: "22"
          cache: "npm"
          cache-dependency-path: studio/package-lock.json
      - run: npm ci
      - run: npm run lint
      - run: npm run typecheck
      - run: npm run build
```

Related Workflows

Workflow	Trigger	Jobs
test.yml	Push/PR (any path)	Ruff lint + pytest (3 suites x 3 Python vers.)
studio-test.yml	Push/PR (studio/**)	ESLint + tsc + Vite build
frontend-test.yml	Push/PR (frontend/**)	ESLint + tsc + Vite build
deploy.yml	Push to main (saas/, molbuilder/)	Railway deployment
release-sdk.yml	Tag sdk-v*	PyPI publish via OIDC trusted publisher

13. Project File Structure

The studio project contains 42 source files organized into a clear directory hierarchy. All source code is under src/ with subdirectories for each concern.

```
studio/
|-- public/
|   +-- favicon.svg             # Molecule icon (3 atoms + 3 bonds)
|-- src/
|   |-- main.tsx                # ReactDOM.createRoot entry point
|   |-- App.tsx                 # Router: / (studio), /login, /register
|   |-- index.css               # Tailwind @theme + dark theme variables
|   |-- vite-env.d.ts           # Vite/ImportMeta type declarations
|   |-- api/
|   |   |-- client.ts           # ApiClient class + singleton factory
|   |   |-- types.ts            # All API request/response interfaces
|   |   |-- auth.ts             # register(), getToken() direct fetch
|   |   +-- molecule.ts         # parseSmilesApi, get3dApi, getPropertiesApi
|   |-- stores/
|   |   |-- auth-store.ts       # Zustand: JWT + API key + localStorage
|   |   |-- workspace-store.ts  # Zustand: molecule collection + active ID
|   |   +-- editor-store.ts     # Zustand: tool, selection, viewport settings
|   |-- hooks/
|   |   |-- useApiClient.ts     # Memoized ApiClient with JWT getter
|   |   |-- useMolecule.ts      # Parse -> fetch 3D + props -> store
|   |   +-- useExport.ts        # Generate MOL/SDF/XYZ + download
|   |-- components/
|   |   |-- ui/                 # 6 shared primitives
|   |   |   |-- Button.tsx      # Variants: primary/secondary/ghost/danger
|   |   |   |-- Input.tsx       # With label + error display
|   |   |   |-- Card.tsx        # Card + CardHeader + CardTitle
|   |   |   |-- Badge.tsx       # Color-coded tags
|   |   |   |-- Alert.tsx       # Info/success/warning/error messages
|   |   |   +-- Tabs.tsx        # Context-based tab system
|   |   |-- auth/               # 3 auth components
|   |   |   |-- AuthGuard.tsx   # Route protection + token validation
|   |   |   |-- LoginForm.tsx   # API key input + JWT exchange
|   |   |   +-- RegisterForm.tsx # Email registration + key display
|   |   |-- viewport/           # 6 viewport components (ALL NEW)
|   |   |   |-- Viewport.tsx    # R3F Canvas + lights + controls
|   |   |   |-- MoleculeScene.tsx# Maps data -> AtomMesh + BondMesh
|   |   |   |-- AtomMesh.tsx    # Sphere + CPK color + selection glow
|   |   |   |-- BondMesh.tsx    # Single/double/triple cylinders
|   |   |   |-- ViewportOverlay.tsx  # Hover HUD (atom info)
|   |   |   +-- SelectionOutline.tsx # Selection summary HUD
|   |   |-- toolbar/            # 3 toolbar components (ALL NEW)
|   |   |   |-- MenuBar.tsx     # File | Edit | View dropdown menus
|   |   |   |-- ToolPalette.tsx  # Select | Rotate | Measure buttons
|   |   |   +-- StatusBar.tsx   # Molecule info + tool/style display
|   |   |-- panels/             # 5 panel components (ALL NEW)
|   |   |   |-- SmilesPanel.tsx  # SMILES input + example molecules
|   |   |   |-- PropertiesPanel.tsx  # MW, LogP, Lipinski, FGs
|   |   |   |-- AtomsPanel.tsx   # Atom table (index, symbol, coords)
|   |   |   |-- ExportPanel.tsx  # MOL/SDF/XYZ download buttons
|   |   |   +-- PanelContainer.tsx   # Collapsible panel wrapper
|   |   +-- layout/
|   |       +-- StudioLayout.tsx # CSS Grid master layout
|   |-- lib/
|   |   |-- constants.ts         # API_BASE_URL, APP_NAME
|   |   |-- cpk-colors.ts       # CPK color map + atom radii
|   |   |-- cn.ts                # Class name merger utility
|   |   |-- bond-geometry.ts     # Cylinder transform math (quaternion)
|   |   |-- mol-export.ts       # MOL V2000 string generator
|   |   |-- sdf-export.ts       # SDF string generator
|   |   +-- xyz-export.ts       # XYZ string generator
|   |-- types/
|   |   +-- editor.ts            # Tool enum, RenderStyle, ViewportSettings
|   +-- pages/
|       |-- StudioPage.tsx       # Main studio page (auth-guarded)
|       |-- LoginPage.tsx        # Centered login form
|       +-- RegisterPage.tsx     # Centered register form
|-- package.json                 # Dependencies + scripts
|-- vite.config.ts               # Vite + React + Tailwind + proxy
|-- tsconfig.json                # Project references
|-- tsconfig.app.json            # App compiler options (strict)
|-- tsconfig.node.json           # Node config (vite.config.ts)
|-- eslint.config.js             # ESLint + TS-ESLint + React rules
|-- vercel.json                  # Vercel deployment + rewrites
+-- .gitignore                   # node_modules, dist, .env
```

14. Build Metrics & Performance

Production Build Output

Chunk	Raw Size	Gzipped	Contents
three-*.js	689.51 KB	176.99 KB	Three.js core library
r3f-*.js	489.22 KB	160.14 KB	R3F + drei + fiber
index-*.js	39.66 KB	14.38 KB	React + Router + Zustand
StudioPage-*.js	23.13 KB	7.62 KB	Studio components
RegisterPage-*.js	2.22 KB	1.06 KB	Registration form
LoginPage-*.js	1.45 KB	0.79 KB	Login form
index-*.css	17.96 KB	4.34 KB	Tailwind CSS output
TOTAL	~1.3 MB	~365 KB	All assets

Three.js and R3F are the dominant bundle contributors at ~1.18 MB raw (~337 KB gzipped). Manual chunk splitting ensures these heavy libraries are loaded lazily and cached independently by the browser. The studio-specific code (StudioPage chunk) is only 23 KB, demonstrating that the application logic is lightweight relative to the 3D engine.

Performance Characteristics

- Build time: ~15 seconds (Vite 6 with Rollup)
- TypeScript compilation: ~3 seconds
- ESLint: ~2 seconds
- Initial page load (gzipped): ~365 KB total transfer
- Time to interactive: depends on network (CDN-served static assets)
- 3D frame rate: 60 FPS for molecules up to ~500 atoms on modern GPUs
- Parallel API calls: 3D structure + properties fetched simultaneously after parse

15. Phase Roadmap

Phase 1: MVP (Current)

Phase 1 delivers the core studio experience: SMILES-driven visualization with interactive 3D rendering, property analysis, and file export. Status: COMPLETE. All 42 source files implemented with 0 lint errors, 0 type errors, and a clean production build.

Feature	Status
Project scaffolding (Vite + React + TS + Tailwind	Complete
Three.js 3D viewport (atoms as spheres, bonds as c	Complete
Interactive camera (orbit, zoom, pan via OrbitCont	Complete
Click-to-select atoms/bonds with highlight	Complete
SMILES input -> API parse -> 3D render	Complete
Properties panel (formula, MW, LogP, Lipinski, FGs	Complete
Client-side file export (MOL V2000, SDF, XYZ)	Complete
Auth flow (register + login + JWT)	Complete
Studio layout (toolbar + tool palette + viewport +	Complete
Status bar (molecule name, atom/bond count, formul	Complete
Vercel deployment config for molbuilder.studio	Complete
CORS update for molbuilder.studio origin	Complete
CI workflow (lint + typecheck + build)	Complete

Phase 2: Interactive Editing

Phase 2 introduces direct 3D molecular editing capabilities, transforming the studio from a viewer into a true editor:

- Click-to-add atoms in 3D (raycasting on a reference plane to determine position)
- Click-and-drag to create bonds between atoms
- Delete atoms/bonds via backspace/delete key
- Undo/redo stack using the command pattern in editor-store
- Measurement tool: click two atoms for distance, three for angle, four for dihedral
- Bond rotation tool: click a rotatable bond, drag to rotate the dihedral angle
- New API endpoint: POST /v1/molecule/from-mol (parse MOL string server-side)

Phase 3: Full Suite

Phase 3 evolves the studio into a comprehensive molecular engineering platform:

- 2D sketcher integration (Ketcher by EPAM, or custom HTML5 canvas)
- Synchronized 2D <-> 3D views (edit in 2D, see live 3D update)

- Template insertion panel (functional groups, amino acids, common fragments)
- Retrosynthesis integration (D3.js tree in a dockable panel)
- Process engineering integration (reactor design, cost estimation)
- Multi-molecule workspace with tabs (compare side-by-side)
- WebSocket support for real-time collaboration (requires API changes)
- Custom shader materials (electron density surfaces, orbital rendering)

16. References

[1] Weininger, D. 'SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules.' J. Chem. Inf. Comput. Sci., 1988, 28(1), 31-36.

[2] Lipinski, C.A., Lombardo, F., Dominy, B.W., Feeney, P.J. 'Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings.' Adv. Drug Deliv. Rev., 1997, 23(1-3), 3-25. DOI: 10.1016/S0169-409X(96)00423-1

[3] Bondi, A. 'van der Waals Volumes and Radii.' J. Phys. Chem., 1964, 68(3), 441-451.

[4] BIOVIA. 'CTfile Formats, BIOVIA Databases 2020.' Dassault Systemes. https://discover.3ds.com/sites/default/files/2020-08/biovia_ctfileformats_2020.pdf

[5] Three.js. Official documentation and source code. https://threejs.org/. GitHub: https://github.com/mrdoob/three.js

[6] Poimandres. 'React Three Fiber.' Documentation: https://r3f.docs.pmnd.rs/. GitHub: https://github.com/pmndrs/react-three-fiber

[7] Poimandres. 'Zustand: Bear necessities for state management.' GitHub: https://github.com/pmndrs/zustand

[8] Khronos Group. 'WebGL Specification.' https://www.khronos.org/webgl/

[9] React Team. 'React 19.' https://react.dev/. Blog: https://react.dev/blog

[10] Tailwind Labs. 'Tailwind CSS v4.0.' https://tailwindcss.com/blog/tailwindcss-v4

[11] Ramirez, S. 'FastAPI.' https://fastapi.tiangolo.com/. GitHub: https://github.com/fastapi/fastapi

[12] Vercel. 'Vercel Platform.' https://vercel.com/

[13] Railway. 'Railway Platform.' https://railway.com/

[14] Wikipedia. 'CPK coloring.' https://en.wikipedia.org/wiki/CPK_coloring

[15] Wikipedia. 'Ball-and-stick model.' https://en.wikipedia.org/wiki/Ball-and-stick_model

[16] Wikipedia. 'Van der Waals radius.' https://en.wikipedia.org/wiki/Van_der_Waals_radius

[17] Wikipedia. 'XYZ file format.' https://en.wikipedia.org/wiki/XYZ_file_format

[18] Wikipedia. 'Chemical table file.' https://en.wikipedia.org/wiki/Chemical_table_file

[19] Wikipedia. 'ADME.' https://en.wikipedia.org/wiki/ADME

[20] Vite. 'Next Generation Frontend Tooling.' https://vite.dev/

Table of Contents