AutoMapGenie-Product Requirements Document (PRD)

Product Name: AutoMapGenie

Purpose:

AutoMapGenie is a tool that automates and simplifies API field mapping, reducing the manual effort Product Analysts and Managers spend on analyzing API structures, comparing fields, and creating usable field mappings for development teams.

Product Description:

AutoMapGenie is designed to alleviate the time-consuming and often repetitive task of API field mapping. It allows users to input source and target API structures (in JSON or XML formats), compare them visually, and automatically generate field mappings. Users can manually refine mappings and export the results in various formats such as JSON or Excel. The intuitive interface, coupled with features like dark/light mode, error handling, and responsive design, ensures a smooth experience across different devices. This tool significantly improves productivity for Product Analysts, Managers, and Developers by eliminating non-productive tasks while ensuring accurate and efficient API integration.

Business Objectives:

- **Increase Productivity:** Automate API analysis and mapping to save time for Product Analysts and Developers.
- **Reduce Errors:** Minimize manual errors in API field mapping through automated comparison and validation.
- **Enhance Developer Efficiency:** Provide developers with clear and accurate mappings for seamless integration.
- **Promote Product-Led Growth (PLG):** Deliver a tool that provides immediate value to users, encouraging adoption through word of mouth and organic growth.
- **Monetization Potential:** Offer subscription-based or tiered pricing models for advanced features, catering to different business sizes and needs.

User Problem Statement:

Product Analysts and Managers spend an excessive amount of time analyzing complex API structures, understanding their fields, and manually creating field mappings for development teams. This process is tedious, error-prone, and often seen as non-productive, yet it is essential for accurate API integration. Developers, in turn, rely on these mappings for successful implementation, making the accuracy and efficiency of this process critical.

Value Proposition:

AutoMapGenie automates the entire API field mapping process, providing an easy-to-use interface for quick comparison, mapping generation, and export. It saves time, reduces manual errors, and enhances collaboration between Product Analysts and Developers. Unlike manual methods or generic tools, AutoMapGenie is purpose-built for API mapping, offering specialized features that directly address the pain points of API integration workflows.

Target Audience & Market Analysis

User Personas:

1. **Product Analyst (PA): Sarah, 28**
    - **Demographics:** Female, 28 years old, works at a mid-sized SaaS company, 3 years of experience in product analysis.
    - **Psychographics:** Detail-oriented, analytical thinker, values efficiency and accuracy in data handling, often overwhelmed by the repetitive nature of API analysis tasks.
    - **Technographics:** Uses tools like Postman, Excel, and internal API documentation systems. Comfortable with JSON and XML structures.
    - **Pain Points:** Manual API field mapping is time-consuming and error-prone. Needs a faster way to generate accurate mappings for developers.
    - **Needs:** A tool that automates API comparison and mapping, allowing her to focus on higher-value tasks.

2. **Product Manager (PM): David, 35**
    - **Demographics:** Male, 35 years old, works at a large enterprise software company, 8 years of experience in product management.
    - **Psychographics:** Strategic thinker, time-conscious, focused on project delivery and team efficiency.
    - **Technographics:** Familiar with API documentation tools, project management software (Jira, Confluence), and collaboration tools.
    - **Pain Points:** API mapping takes too long, delaying project timelines. Needs seamless communication and handoff between product and development teams.
    - **Needs:** A solution that reduces the back-and-forth between analysts and developers by providing accurate, ready-to-use mappings.

3. **Software Developer: Priya, 30**
    - **Demographics:** Female, 30 years old, backend developer at a fast-growing startup, 5 years of coding experience.

    - **Psychographics:** Problem-solver, values clean code and clear documentation, often frustrated by incomplete or incorrect API mappings.
    - **Technographics:** Works with various APIs, uses tools like Postman, Swagger, and GitHub. Comfortable with JSON, XML, and YAML.
    - **Pain Points:** Relies on accurate API mappings for integration; manual errors slow down development.
    - **Needs:** Clear, precise field mappings that can be quickly implemented into code without backtracking.

Market Segmentation:

- **Segment 1:** Mid-sized SaaS companies with growing product and development teams needing efficient API integration tools.
- **Segment 2:** Large enterprises with complex API ecosystems and a high volume of integrations.
- **Segment 3:** Fast-growing startups that need to move quickly with minimal errors in API integrations.
- **Segment 4:** Freelance Product Analysts and Developers handling multiple projects and seeking time-saving tools.

Competitor Analysis:

Competitor	Strengths	Weaknesses	Differentiation for AutoMapGenie
Postman	API testing, extensive community support	No dedicated field mapping tool	Focused on field mapping, not just testing
MuleSoft Anypoint	Comprehensive integration platform	High cost, steep learning curve	Lightweight, affordable, easy to use
Swagger/OpenAPI	API documentation and testing	Limited mapping capabilities	Dedicated mapping functionality
Manual Excel Mapping	Familiarity, customizable	Time-consuming, error-prone, not scalable	Automated, fast, and accurate

Goals and Objectives

Business Goals (SMART):

- Increase adoption by **20% within the first 6 months** post-launch through a freemium model with essential features available for free and premium features behind a paywall.

- Reduce the time Product Analysts spend on API field mapping by **at least 50%** within the first 3 months of use.
- Achieve a customer satisfaction (CSAT) score of **90% or above** within the first year through an intuitive UX and reliable performance.
- Generate **$100,000 in annual recurring revenue (ARR)** by the end of the first year through tiered subscription plans.

User Goals:

- Quickly analyze and compare API structures without manual intervention.
- Automatically generate accurate field mappings that can be manually adjusted if necessary.
- Export mappings in multiple formats (JSON, Excel) for easy sharing and implementation.
- Enjoy a seamless, user-friendly experience with error handling and visual feedback.

Success Metrics (KPIs):

- **Adoption Metrics:**
  - Number of new signups per month.
  - Daily and monthly active users.
- **Engagement Metrics:**
  - Average time spent on the app during API mapping sessions.
  - Number of mappings generated and exported per user.
- **Financial Metrics:**
  - Revenue from subscription plans.
  - Customer acquisition cost (CAC) and customer lifetime value (LTV).

Features and Functionality

Feature Prioritization Matrix:

Feature	Priority	Impact	Feasibility
Input API structures (JSON/XML)	High	High	High
API structure comparison	High	High	Medium
Automated field mapping	High	High	Medium
Manual adjustment of mappings	Medium	High	Medium
Export to JSON/Excel	High	High	High

Dark/Light mode toggle	Medium	Medium	High
Error handling	High	High	Medium

Feature Descriptions:

1. API Input Interface

- **Description:** Users can paste or type API structures into code editors, with format selection for JSON/XML.
- **User Stories:**
  - *As a Product Analyst, I want to easily input API structures so that I can begin the comparison process.*
- **Acceptance Criteria:**
  - Code editors must support JSON and XML formats.
  - Users can switch between dark and light modes.
- **Technical Considerations:**
  - Integration with code editor libraries like Monaco Editor.

2. API Comparison

- **Description:** System analyzes and compares source and target API structures.
- **User Stories:**
  - *As a Product Manager, I want to see a clear comparison of API structures so that I can identify differences and similarities.*
- **Acceptance Criteria:**
  - System must display a dialog with detailed breakdowns of both APIs.
- **Technical Considerations:**
  - Implement parsing logic for JSON and XML structures.

3. Automated Field Mapping

- **Description:** Generates field mappings between source and target APIs automatically.
- **User Stories:**
  - *As a Developer, I want to get accurate field mappings automatically so that I can integrate APIs faster.*
- **Acceptance Criteria:**
  - Generated mappings must appear in a table format with editable fields.
- **Technical Considerations:**
  - Use algorithms to match fields based on names, types, and structures.

4. Manual Mapping Adjustments

- **Description:** Users can manually edit field mappings for accuracy.

- **User Stories:**
  - *As a Product Analyst, I want to adjust field mappings manually so that I can ensure they meet project requirements.*
- **Acceptance Criteria:**
  - Fields must be editable directly within the table.

5. Export Options

- **Description:** Users can export mappings as JSON or Excel files.
- **User Stories:**
  - *As a Developer, I want to export mappings so that I can share them with my team and implement them easily.*
- **Acceptance Criteria:**
  - Users must be able to download JSON and Excel files with proper structure.

6. Error Handling

- **Description:** Validates JSON/XML formats and displays error messages.
- **User Stories:**
  - *As a Product Analyst, I want to see clear error messages when something goes wrong so that I can correct my input.*
- **Acceptance Criteria:**
  - System must detect invalid formats and show specific error messages.
  - System should identify json and xml formats correctly

Technical Specifications

System Architecture Diagram:

(A visual representation can be created upon request, but here's a textual breakdown for now.)

- **Frontend:** Built with React and TypeScript, ensuring a modular and maintainable codebase.
- **Build Tool:** Vite serves as the development and build tool, known for its speed and optimized output.
- **UI Layer:** Tailwind CSS for utility-first styling and shadcn/ui for pre-built, accessible components.
- **Data Processing:** Handles JSON and XML inputs using `fast-xml-parser` and processes exports via `XLSX`.
- **State Management:** Managed through React Hooks, ensuring a lightweight and efficient state flow.
- **Theme Management:** Dark/Light mode toggling through `next-themes`.
- **Development Tools:** TypeScript ensures type safety, ESLint enforces code quality, and SWC accelerates compilation.

API Integrations:

- **None currently required,** as the application processes user-input data locally. However, future integrations with API documentation tools (like Swagger or Postman) could enhance the product.

Data Model:

- The application processes data dynamically without a persistent data storage model since all operations are handled in-memory during the session.
- Data structures include:
    - **Source API Structure:** JSON/XML object tree.
    - **Target API Structure:** JSON/XML object tree.
    - **Mapping Table:** JSON array with source-target field pairs.

Technology Stack:

- **Frontend Framework:**
    - React (^18.3.1) with TypeScript for a modern, type-safe development environment.
- **Build Tool:**
    - Vite for optimized builds and fast development cycles.
- **UI Components & Styling:**
    - `shadcn/ui`, Tailwind CSS, Lucide React for icons.
- **Data Processing & File Handling:**
    - `fast-xml-parser (^4.5.1)` for XML, `XLSX (^0.18.5)` for Excel.
- **State Management & UI Utilities:**
    - React Hooks, `@radix-ui` components, `next-themes` for theme toggling.
- **Development Tools:**
    - TypeScript, ESLint, SWC.
- **Utilities:**
    - `clsx`, `tailwind-merge`, `class-variance-authority` for styling and component management.

Hosting/Deployment:

- **Deployment Pipeline:**
    - Hosted on **Netlify** for seamless CI/CD and easy deployment.
- **Performance Considerations:**
    - Vite ensures minimal build sizes and fast load times.
    - Lazy loading components to optimize performance.

Constraints and Assumptions

Budget:

- Estimated budget of **20000 Rs for initial development and launch**, covering:
    - Developer salaries, tools, and deployment costs.
    - Marketing and user acquisition expenses. - this is doubtful as this is an internal tool !

Timeline:

- **Phase 1:** MVP Development – 1 week
- **Phase 2:** Beta Testing and Feedback –3 days
- **Phase 3:** Launch and Marketing – 2 weeks
- **Phase 4:** Post-launch Enhancements – Ongoing

Resources:

- **Team Members:**
    - 1 Frontend Developer
    - 1 Product Manager
    - 1 UX Designer
    - 1 QA Engineer
- **Third-party Resources:**
    - Netlify for deployment.
    - OpenAI/ChatGPT for user support and documentation assistance (optional future integration).
    - Lovable.io for tool development
    - Cursor to enhance the UI and UX functionalities of the tool

Technical Limitations:

- Limited to JSON and XML formats initially.
- No backend support for persistent storage or multi-user collaboration in MVP.
- Performance may be affected with extremely large API structures due to in-browser processing.

Regulatory/Compliance Requirements:

- Ensures compliance with data privacy regulations like **GDPR** as no user data is stored.
- Open-source libraries used are compliant with MIT and similar licenses.

Assumptions:

- Users are familiar with JSON and XML structures.
- Users have access to API documentation for accurate inputs.
- Market demand exists for lightweight API mapping tools without heavy integrations.
- Early adopters will provide feedback for continuous improvement.

Appendix
JSON and XML samples to test

1. Ecommerce samples

JSON REST Simple

```
{
  "products": [
    { "id": 1, "name": "Laptop", "price": 899.99, "category": "Electronics" },
    { "id": 2, "name": "Smartphone", "price": 699.99, "category": "Electronics" },
    { "id": 3, "name": "Headphones", "price": 199.99, "category": "Accessories" },
    { "id": 4, "name": "Backpack", "price": 49.99, "category": "Fashion" },
    { "id": 5, "name": "Coffee Maker", "price": 79.99, "category": "Home Appliances" }
  ]
}
```

JSON REST Nested

```
{
  "store": {
    "name": "Tech World",
    "location": "Online",
    "products": [
      {
        "id": 1,
        "name": "Gaming Laptop",
        "details": {
          "brand": "Alienware",
          "specs": {
            "CPU": "i9",
            "RAM": "32GB",
            "Storage": "1TB SSD"
          }
        }
      },
      {
        "id": 2,
        "name": "Smart TV",
```

```
      "details": {
        "brand": "Samsung",
        "features": ["4K", "HDR", "Smart Hub"]
      }
    }
  ]
 }
}
```

XML SOAP Simple

```
<?xml version="1.0" encoding="UTF-8"?>
<Products>
  <Product>
    <Id>1</Id>
    <Name>Laptop</Name>
    <Price>899.99</Price>
    <Category>Electronics</Category>
  </Product>
  <Product>
    <Id>2</Id>
    <Name>Smartphone</Name>
    <Price>699.99</Price>
    <Category>Electronics</Category>
  </Product>
</Products>
```

XML SOAP Nested

```
<?xml version="1.0" encoding="UTF-8"?>
<Store>
  <Name>Gadget Haven</Name>
  <Location>Online</Location>
  <Products>
    <Product>
      <Id>1</Id>
      <Name>Gaming Laptop</Name>
      <Details>
        <Brand>Alienware</Brand>
        <Specs>
          <CPU>i9</CPU>
          <RAM>32GB</RAM>
          <Storage>1TB SSD</Storage>
        </Specs>
      </Details>
    </Product>
    <Product>
      <Id>2</Id>
      <Name>Smart TV</Name>
      <Details>
```

```
        <Brand>LG</Brand>
        <Features>
          <Feature>4K</Feature>
          <Feature>OLED</Feature>
          <Feature>AI ThinQ</Feature>
        </Features>
      </Details>
    </Product>
  </Products>
</Store>
```