**AI 接口测试系统建设方案**

**一、项目背景**

传统手动接口测试在复杂项目中效率低、易出错，无法满足快速迭代需求。构建集**需求分析、用例生成、用例管理、用例执行、缺陷分析**于一体的 AI 测试系统，是提升测试自动化与智能化的关键。

**二、系统建设目标**

**（一）用户功能目标**

|  |  |
| --- | --- |
| 目标分类 | 核心功能描述 |
| **文档处理** | 支持需求文档、接口文档、XMind 用例、图片等多格式文件上传与管理，触发对应解析引擎（llmindex 处理文本 / XMind，月之暗面Moonshot处理图片+docling+qianwen）。 |
| **需求分析** | 自动化解析文档生成**结构化字段明细表**（含接口 URL、请求方法、必填参数、业务场景等），支持历史分析结果查询、导出及关键词检索。 |
| **用例生成** | 基于选中需求智能生成测试用例，支持用户反馈修改（输入建议 / 选择同意 / 取消），经多智能体评审后输出标准化最终用例。 |
| **用例管理** | 管理测试用例，支持表格化展示、批量导出（Excel/XML）、版本对比（差异高亮）及标签分类（比如 "核心链路"" 边界值测试 "）。 |
| **用例执行** | 通过 Autogen-MCP 机制对接 Metersphere、Apifox 等平台，实现测试用例一键导入、执行状态同步（通过 / 失败 / 阻塞）、实时日志查看及用例增删改查。 |
| **缺陷分析** | 多平台获取失败用例的错误信息（HTTP 状态码、响应体、断言日志），汇总生成诊断报告 |

**主页面原型图（参考）**



**（二）技术选型**

1. **前端层**：采用 **Vue3** + **Vite** 实现高效交互。
2. **后端层**：

**FastAPI**：前后端交互（**HTTP**）、大模型交互（**websocket**流式响应+全双工）。

**llamaIndex**：文档处理+检索增强+分块处理。

**AutoGen**：**AgentChat**构建多智能体交互协作分析流，**Autogen的消息机制**实现智能体间消息流转。

**text2sql**：自然语言转SQL查询。

**月之暗面Moonshot**：多模态数据转文本。

**DeepSeek**：核心文本处理、分析+输出。

**PydanticAI**：用例结构化输出。

**Milvus**：向量化文本的持久化存储

**Mysql**:结构化数据的持久化存储

1. **全流程闭环**：打通「需求分析→用例生成→执行验证→缺陷分析」数据链路

**三、用户交互设计**

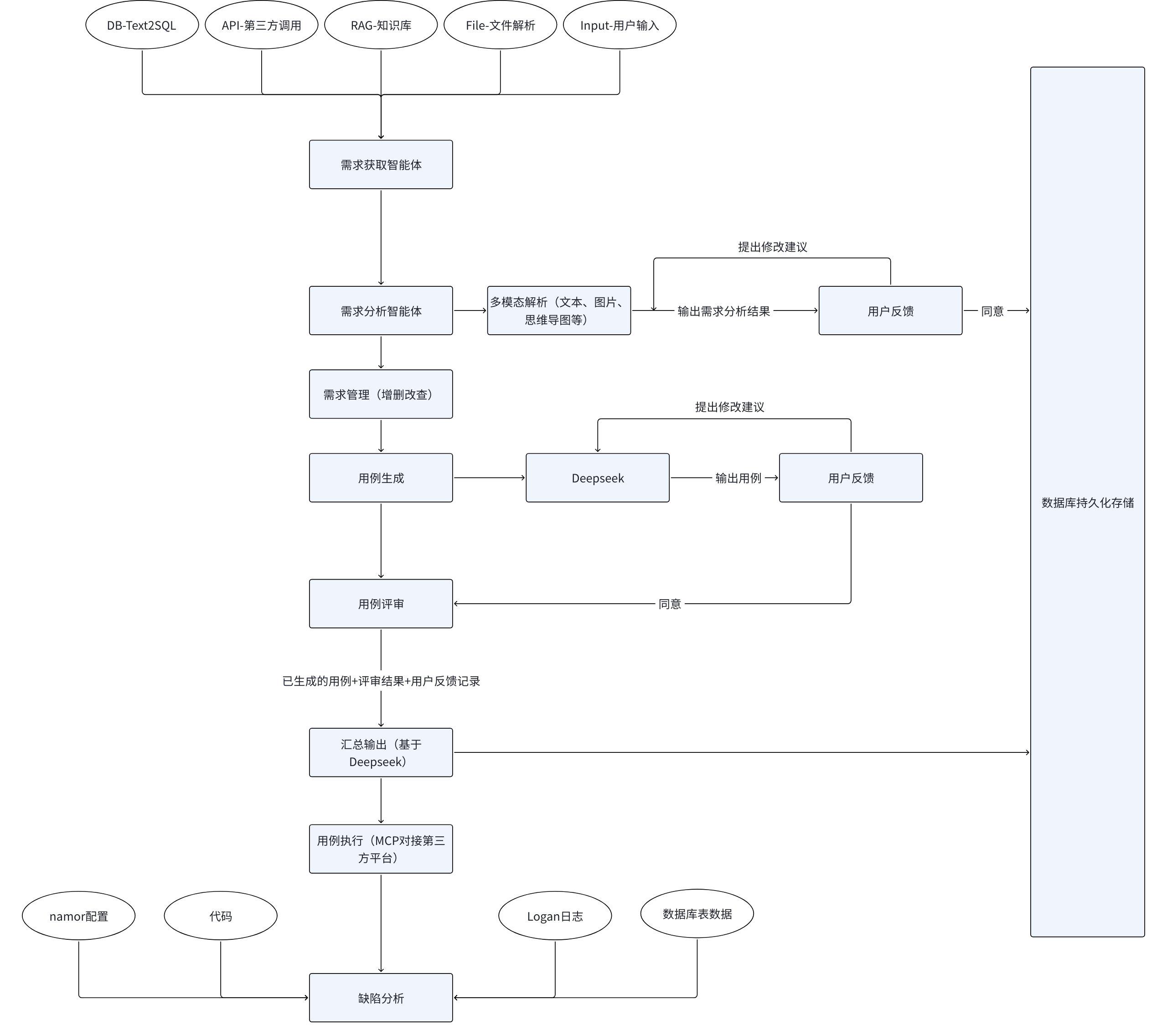
**（一）界面架构设计**

左侧固定导航栏包含：**项目管理 | 需求分析 | 需求管理 | 用例生成 | 用例管理 | 用例执行 | 缺陷分析**



**（二）业务操作流程**

**业务流程图**



**业务流程**

1. **数据输入与需求分析**

* 用户在**「项目管理」**创建项目后，通过**「需求分析」**模块上传多格式文档，Llamaindex和月之暗面Moonshot大模型自动识别：
* 文本解析为结构化字段（接口参数、业务规则）
* 图片、思维导图等转化为描述性文本
* 分析结果存入**「需求管理」**。

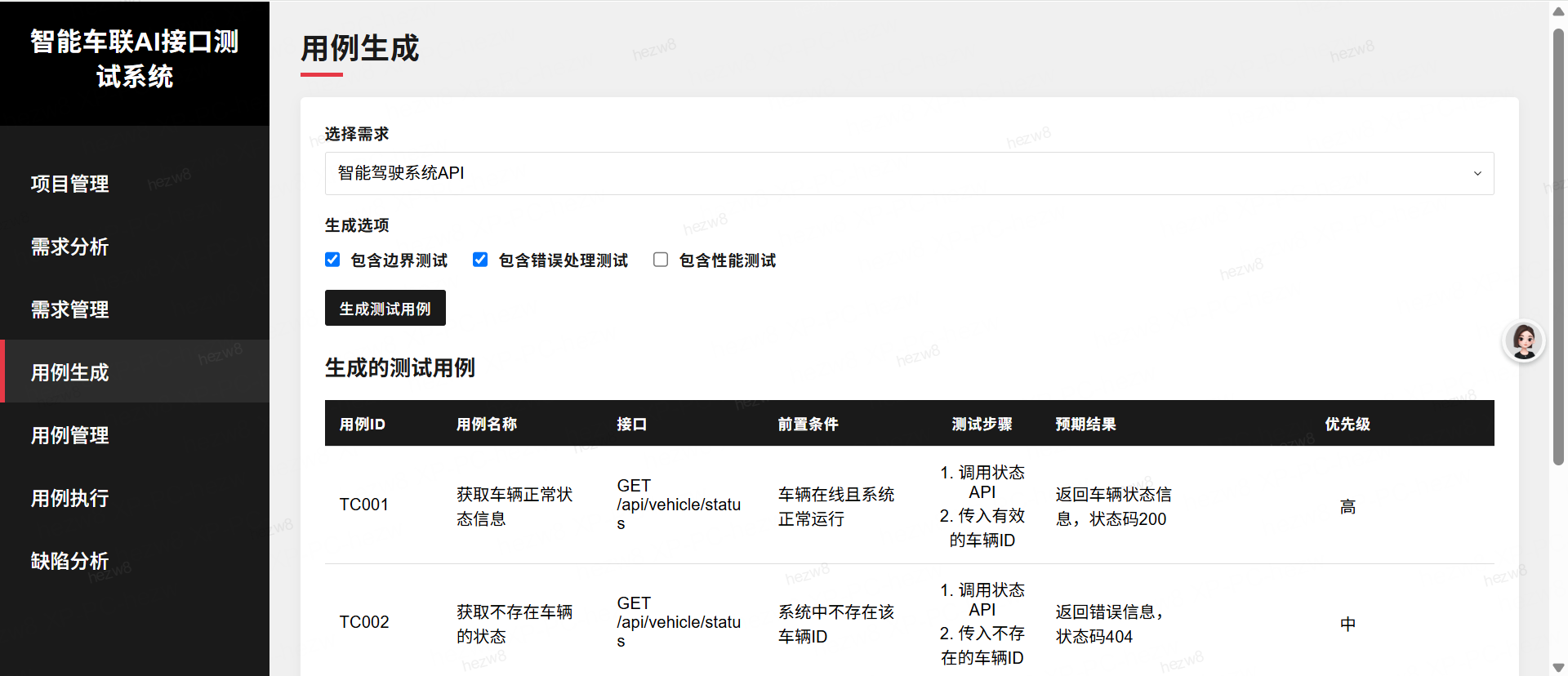


**输出结构化数据示例**：

|  |
| --- |
| json {  "project\_info": {  "项目名称": "按车型和OS版本下发颠簸点数据",  "项目版本": "V1.0",  "需求来源": "《20240809-车管平台-智能底盘需求设计文档》",  "负责人": "测试负责人",  "创建时间": "2025-05-15",  "背景": "1. SR渲染XOS 5.5.0引入bug导致渲染效果不好（飞书文档外暂不展示）；2. 数据需添加wasRisky字段灰度能力；3. OCM仅支持10个VIN灰度，采用智能底盘调用ota开放平台接口方案"  },  "需求分析树": [  {  "需求层级": "一级需求",  "需求ID": "REQ-001",  "需求名称": "颠簸点数据下发控制",  "需求描述": "服务端具备按车型、OS版本、VIN码控制颠簸点数据下发范围的能力",  "关联文档": ["《车端导航数据获取接口设计》"],  "子需求": [  {  "需求层级": "二级需求",  "需求ID": "REQ-001-01",  "需求名称": "wasRisky字段灰度能力",  "需求描述": "根据车辆VIN、车型、OS版本集合，动态控制wasRisky字段返回值",  "测试点集合": [  {  "测试点ID": "TP-001-01-01",  "测试点名称": "验证8种集合组合下wasRisky值",  "需求映射": "REQ-001-01",  "测试类型": "接口测试",  "输入参数": ["A、B、C集合不同组合（如A={a}, B={b}, C={c}）"],  "预期输出": {  "业务逻辑": "按规则返回wasRisky=true/false（如A∩B∩(U-C)范围内返回true）",  "数据库校验": "odps颠簸图层中对应车辆wasRisky字段按规则更新"  },  "关联接口": ["导航态获取颠簸点接口", "巡航态获取颠簸点接口"],  "优先级": "P0",  "备注": "需验证集合C的匹配规则（TODO：前缀匹配/精确匹配）"  }  ]  },  {  "需求层级": "二级需求",  "需求ID": "REQ-001-02",  "需求名称": "VIN灰度列表管理",  "需求描述": "支持VIN列表的添加、清除及配置缓存刷新",  "测试点集合": [  {  "测试点ID": "TP-001-02-01",  "测试点名称": "正向场景：添加单个VIN到灰度列表",  "需求映射": "REQ-001-02",  "接口信息": {  "接口ID": "API-001",  "接口名称": "VIN灰度列表导入接口",  "接口路径": "http://mock.test.xiaopeng.local/project/1634/interface/api/215482",  "请求方法": "POST",  "示例参数": '{"vinList": ["LMVHFEFZ3JA994663"]}'  },  "预期输出": {  "状态码": 200,  "数据库校验": "vin\_list表新增对应VIN记录"  },  "关联文档": ["预发布环境请求示例（需替换cookie）"],  "优先级": "P1"  },  {  "测试点ID": "TP-001-02-02",  "测试点名称": "反向场景：清除灰度列表",  "需求映射": "REQ-001-02",  "接口信息": {  "接口ID": "API-002",  "接口名称": "VIN灰度列表清除接口",  "接口路径": "http://mock.test.xiaopeng.local/project/1634/interface/api/215488",  "请求方法": "POST"  },  "预期输出": {  "状态码": 200,  "数据库校验": "vin\_list表对应VIN记录清空"  }  }  ]  }  ]  }  ],  "接口矩阵": [  {  "接口ID": "API-001",  "接口名称": "VIN灰度列表导入接口",  "所属模块": "数据灰度管理",  "请求方法": "POST",  "参数列表": [  {  "参数名": "vinList",  "参数类型": "array",  "是否必填": "是",  "示例值": ["LMVHFEFZ3JA994663"],  "需求来源": "需求文档-灰度列表管理"  }  ],  "响应字段": {  "code": { "类型": "int", "说明": "200=成功" },  "message": { "类型": "string", "说明": "操作结果描述" }  },  "关联测试点": ["TP-001-02-01"]  },  {  "接口ID": "API-003",  "接口名称": "导航态获取颠簸点接口",  "所属模块": "数据下发",  "接口路径": "/cv2/vmp/intelligentChassisData/bumpPoint/listByArea",  "请求方法": "GET",  "关联测试点": ["TP-001-01-01"]  }  ],  "流程覆盖矩阵": [  {  "流程图名称": "运营灰度流程",  "流程节点": ["VIN灰度导入", "车型灰度配置", "OS版本排除"],  "覆盖测试点": [  {  "测试点ID": "TP-001-02-01",  "节点映射": "VIN灰度导入",  "验证内容": "接口正确接收VIN列表并存储"  },  {  "测试点ID": "TP-001-01-01",  "节点映射": "车型灰度配置",  "验证内容": "按车型集合B过滤wasRisky字段"  }  ]  }  ],  "需求-测试点追溯表": [  {  "需求ID": "REQ-001-01",  "需求名称": "wasRisky字段灰度能力",  "覆盖测试点数量": 1,  "未覆盖需求点": ["cduVersion为空时的处理逻辑（已知问题）"]  }  ],  "配置项管理": [  {  "配置项名称": "bump-point.data-gray.vehicleTypeCodeSet",  "值示例": "EA,ED",  "说明": "车型编码集合配置"  },  {  "配置项名称": "bump-point.data-gray.excludeCduVersionSet",  "值示例": "5.4.5,5.5.0",  "说明": "OS版本排除集合配置"  }  ] } |

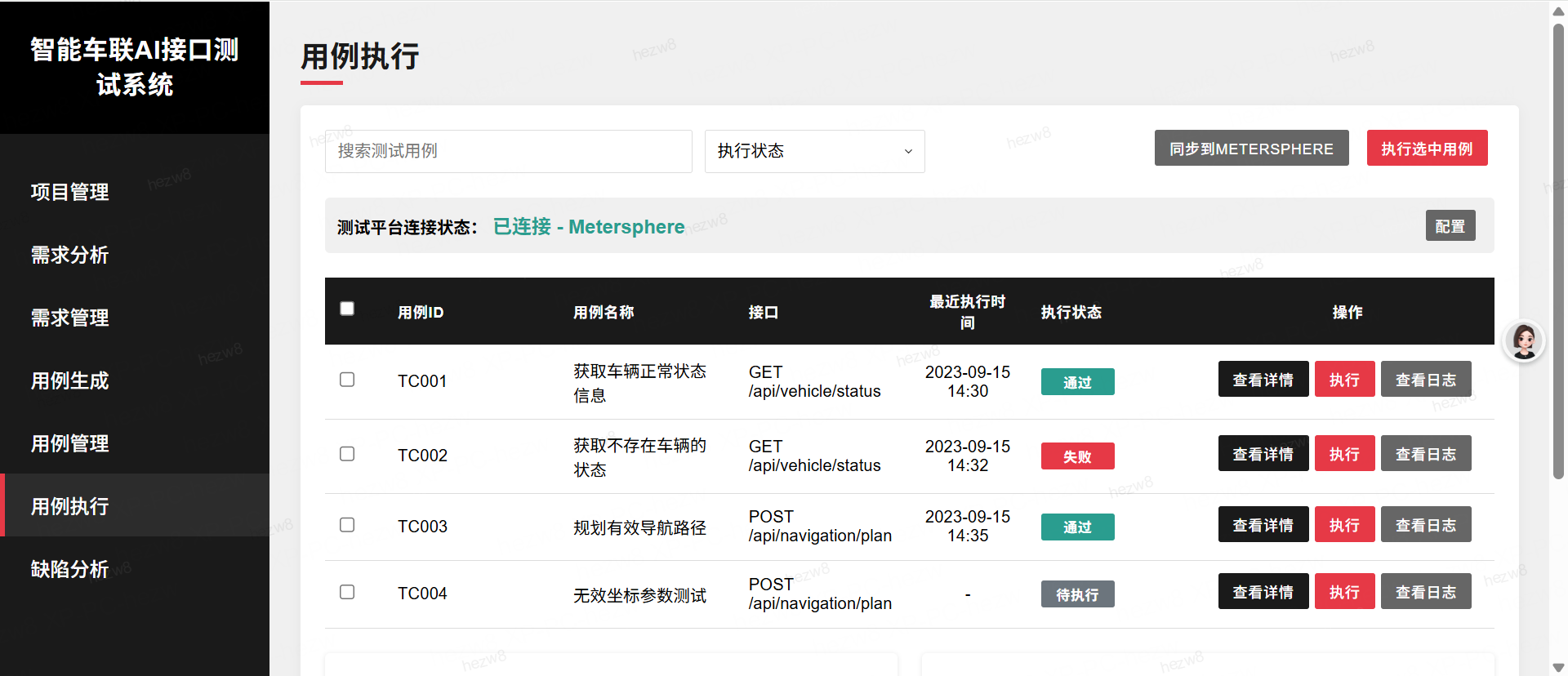
1. **用例生成与多轮迭代**

* 在**「用例生成」**选中需求→**「用例生成智能体」**按模板生成表格化用例
* 反馈弹窗支持：
* ✅ **同意**：进入**[用例评审智能体]**，输出完整性评分与修改建议
* ⚙ **修改**：用户输入建议（如 "补充参数为空的异常测试"），DeepSeek 二次生成用例
* ❌ **取消**：返回需求列表重新选择
* 评审通过后，**[汇总输出智能体]**合并需求分析结果、反馈记录、评审建议，生成带唯一 ID 的最终用例，存入**「用例管理」**。



1. **用例执行与平台对接**

* 在「用例管理」选中最终用例→「用例执行」模块选择目标平台（Apifox/Metersphere/ 自定义）
* Autogen 通过 MCP 机制自动转换用例格式（如将 "预期结果" 映射为 Metersphere 的断言脚本），支持：
* 批量导入 / 单个导入
* 用例执行+结果获取
* 执行进度实时监控（百分比 + 日志流）



1. **缺陷分析**

* 用例执行失败时可进行缺陷分析：
* 提取错误信息（状态码、响应体等）
* DeepSeek 结合历史缺陷库、执行结果、项目代码、数据库、namor配置+Logan日志定位原因
* 生成可视化报告（失败原因+修复建议）

**四、核心技术解析**

**（二）后端服务层**

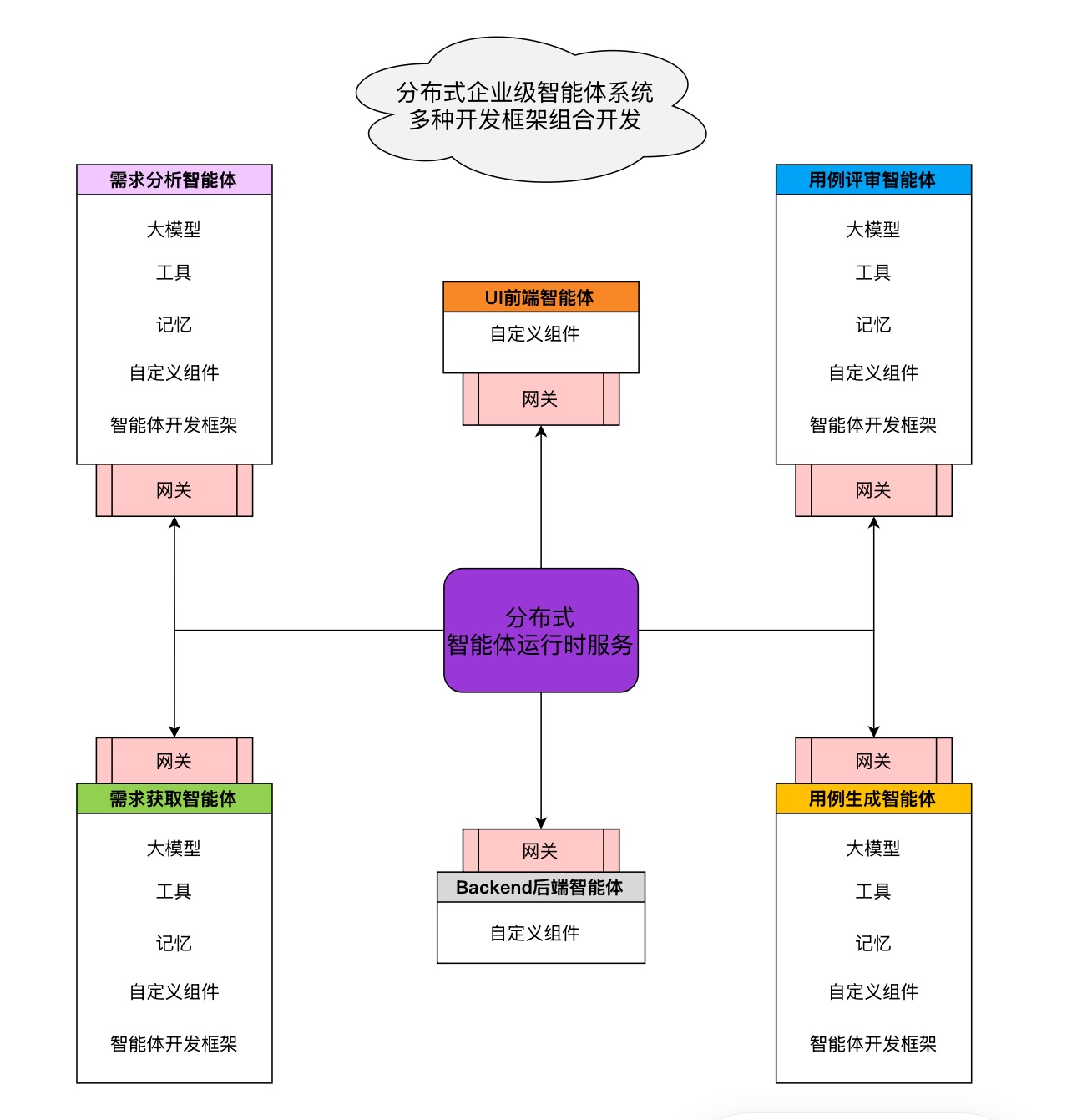
* 采用 **FastAPI** 构建接口，结合 **AutoGen（AgentChat + 消息机制）**实现任务自动化流转
* 利用 **llamaIndex** 管理知识库，通过 **text2sql** 实现文本到 SQL 的转换
* 对接多模态识别的 **LLM（月之暗面 Moonshot）**将输入数据转化为文本
* 核心文本 **LLM（DeepSeek）**进行需求分析并存储到数据库。
* 需求分析阶段，[需求分析智能体]调用[查漏补缺智能体]，提示用户补充必要信息。
* 生成用例阶段，从数据库获取需求，通过 **AutoGen 消息机制+AgenChat**依次完成测试点提取、接口用例生成、用例评审和结果汇总，最终用 **PydanticAI** 结构化输出用例
* 生成的用例通过 **AutoGen** 的 **MCP** 机制对接到用例管理平台（如 **APIFox**、**MeterSphere**）进行导入和执行。
* 缺陷分析基于项目代码、执行结果，数据库、namor配置和Logan中台日志，接入**Deepseek**实现错误原因的多维度分析（接口层 / 业务层 / 断言层）。

**五、分阶段实施计划**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 阶段 | 时间 | 核心目标 | 关键任务列表 | 技术要点 | 交付成果 |
| Q2 | 4-6 月 | * **最小闭环 Demo 验证** | 1. 实现需求分析和用例生成两大核心模块的基础功能（分析）  2. 支持文本、文档和 XMind 解析，生成基础用例  3.前端平台基础搭建，可使用上述功能 | * Vue3+FastAPI+llamaIndex+text2sql+Autogen+Deepseek | 可使用的最小闭环Demo |
| Q3 | 7-9 月 | * **部署上线** * **用例执行 + 缺陷分析落地** * **多模态技术实现** | 1. 接入月之暗面，实现图片 OCR 解析  2. 新增用户反馈 / 评审历史存储与展示  3. 优化用例生成模板，支持类 Excel 表格编辑 | * 新增月之暗Moonshot * Autogen+text2sql+Mysql实现数据持久化存储，前后端支持用户反馈机制 * 新增llmindex 分块处理和索引优化 * 技术栈迭代 | 可使用更丰富的功能 |
| Q4 | 10-12 月 | * **稳定性优化** * **用户反馈功能完善** * **技术架构优化，提高用例生成准确率** * **分块上传机制实现** | 1. 开发 MCP 接口适配层，对接 Metersphere/Apifox  2. 实现缺陷分析引擎，关联执行日志与错误定位  3. 全流程闭环（需求分析→生成→执行→缺陷分析） | * 服务器 * Autogen-MCP 协议开发 * 三方平台接入（Logan、namor） | 可上线运行的正式版本（含 7 大模块） |

**六、更长远的规划（性能提升）**

分布式部署，提升响应速度



**协调点**

1. 第三方平台，namor，logan等的接入
2. 提测流程优化：提测内容的标准化构建（应尽量适配AI的结构化输入）
3. 测试工具的切换(metersphere -> apifox)