中山大學

SUN YAT-SEN UNIVERSITY

软件工程

Yatmosphere 智能家居控制系统需求分析

团 队 Yatmosphere

学 院 计算机学院

专 业 计算机科学与技术

2025 年 6 月 3 日

目录

1 系统背景与目标

1.1 系统背景

随着物联网和智能设备的迅速普及，全球智能家居市场持续高速增长。据统计，全球智能家居市场预计在 2025 年将达到 1740 亿美元。用户对家庭自动化的需求不断提升，希望通过语音、手机或自动化规则，实现灯光、温控、安全等设备的互联控制。然而，现有智能家居系统往往碎片化严重，不同品牌、不同协议的设备缺乏统一平台，安全性和可扩展性也亟待提高。因此，开发一个统一的智能家居控制平台—— “Yatmosphere” 具有重要意义，不仅能提升用户体验，还能提高系统安全与运行效率。

全屋智能设备一体化控制系统旨在为用户提供一个集中、统一的界面和接口，实现对家中各种智能设备（如智能灯光、空调、窗帘、安防设备等）的便捷控制与管理，提升家居智能化体验和生活品质。

项目的英文名称定为 **Yatmosphere**。“Yatmosphere” 灵感来自 “atmosphere”（大气层），寓意系统将为智能家居营造一个统一而智能的 “环境氛围”，实现对家庭设备的全局感知和控制。Yatmosphere 智能家居控制系统旨在打造一个融合多种智能设备的统一管理平台，为用户提供智能化、便利化的家居环境控制体验。团队通过物联网技术，实现对家用电器、照明、安防等设备的远程监控与集中控制，提升生活质量和安全性。同时，项目代码托管于 GitHub 和团队协作文档。

1.2 系统目标

为了确保 Yatmosphere 智能家居控制系统在功能完备、性能可靠、安全稳定、易于维护和用户体验方面均达到预期，系统目标可划分为以下五大方面，并在每个方面下进行详细阐述。

1.2.1 功能完备性

Yatmosphere 旨在提供覆盖家居中所有智能设备的全生命周期管理能力，从设备注册、配置、分组到删除都应一站式完成。系统需要具备直观的设备控制面板，支持对灯光、空调、窗帘、安防等不同类型设备的开关、亮度、温度等参数自由调节，并能实时展示在线/离线状态及历史数据曲线。场景与自动化编排功能则应允许用户通过可视化拖拽方式定义复杂的 if–then 规则，结合时间、设备状态和传感器阈值等多种触发条件，实现一键执行或定时调度，同时保留完整的执行日志以便回溯。用户权限管理模块则需支持多级角色（如管理员、家主、普通用户、访客）与精细化权限分配，动态控制前端界面元素的可见性与操作能力，确保不同角色仅能访问其被授权的功能与设备。

1.2.2 性能与可用性

为了应对家庭中大量设备与并发用户的实时互动需求，系统必须在低延迟与高吞吐两方面均表现优异。具体而言，从前端发出控制命令到设备执行并反馈状态的全链路延迟应控制在 200 毫秒以内，实时状态更新从设备端变化到前端显示不得超过 300 毫秒；同时，系统应支撑至少 500 个并发 WebSocket/SSE 连接，MQTT 中间件需能稳定处理每分钟 5000 条以上的消息发布与订阅。当出现单个节点故障时，系统应依托主从或集群架构自动完成故障切换，保证整体可用率不低于 99.9%；并配备完善的健康检测与报警机制，实时监控 API 网关、数据库和 MQTT Broker 的运行状态，及时告警和自愈，确保用户在任何时刻都能顺畅使用。

1.2.3 安全与合规

鉴于智能家居系统直接关乎用户的隐私与人身安全，Yatmosphere 必须在认证、授权、数据传输和存储等环节全方位加固。所有 API 与 WebSocket/SSE 通信均需通过 HTTPS/TLS 加密，MQTT 通信亦需采用 TLS 隧道并校验客户端证书；用户身份验证应基于 JWT 或 OAuth2，结合多因素认证可选机制，以防止未授权访问；系统需对每一次关键操作（如登录登出、设备配置变更、场景脚本编辑）进行审计日志记录，便于事后追踪与分析。数据库中对用户密码等敏感字段进行强加密存储，并定期进行渗透测试与安全评估，及时修补潜在漏洞，确保系统始终符合国内外安全合规标准。

1.2.4 可维护性与可扩展性

为适应智能家居生态和互联网技术的高速演进，系统设计须坚持模块化、契约化和插件化原则。前后端分离的三层架构（表现层、业务层、数据访问层）能够让各模块独立升级与部署，开发团队可对设备管理、场景引擎、权限管理、状态订阅等功能进行单独维护。所有 RESTful API 均使用 Swagger/OpenAPI 规范定义并自动生成文档，同时结合契约测试（Contract Testing）确保前后端接口的一致性；设备协议适配层应设计为可插拔插件，后续可快速接入 Z-Wave、BLE、Thread 等新协议；场景规则引擎亦应支持脚本扩展或第三方规则库接入，使得功能扩展不依赖核心代码改动，从而大幅降低后期维护成本。

1.2.5 用户体验与可访问性

Yatmosphere 不仅追求技术先进，更要保证用户使用的便捷与舒适。界面设计需遵循统一的视觉规范和交互流程，对 PC、平板、手机等设备采用响应式布局，并提供 PWA 安装、离线模式以及推送通知等常用功能。系统发布 Beta 版本后应至少开展两轮用户可用性测试，收集真实用户的操作数据与反馈，并在正式上线前调整优化。为兼顾不同用户群体需求，界面还需满足 WCAG 2.1 AA 标准，提供键盘导航、屏幕阅读器支持和高对比度模式等无障碍功能，确保视力、行动不便等用户都能顺畅访问和操作。

2 目标用户与用户画像

本节聚焦本系统的主要使用群体，通过精准画像与需求洞察，确保功能设计和交互流程贴合不同角色的真实场景与价值诉求。并力求通过精准画像与深度需求洞察，为每类用户提供契合场景和驱动业务价值的功能，以下为可能存在的相关问题：

用户类型	年龄范围	痛点与需求	用户故事
家庭主人 (Home Owner)	30–55 岁	设备多且分散；希望一键切换全屋模式；重视安全与隐私	“出门上班前，我希望一键启动离家模式，关闭所有灯光、锁门并开启安全监控，这样我能放心离开。”
家庭成员 (Family Member)	10–70 岁	操作复杂；不同终端使用习惯差异大；需要简化界面与语音交互	“我不太懂技术，想用语音说‘打开客厅灯’，而不用找手机点开应用操作。”
访客/租客 (Guest/Tenant)	18–65 岁	临时授权管理难；无法快速获取临时密码；担心权限滥用	“来朋友家做客时，我想给他一个只能开关客厅灯的临时权限，用完后自动失效。”
技术爱好者 (Tech Enthusiast)	20–45 岁	希望二次开发与自定义集成;需要开放 API 与插件机制	“我想把智能家居系统接入我的 HomeKit，并用脚本自动调节灯光色温。”
运维/管理员 (Operator/Admin)	25–50 岁	大规模设备监控与故障排查困难；需要批量操作与日志审计	“当物业管理员时，我希望能批量查看所有房间的设备在线状态并导出故障报告。”

2.1 家庭主人 (Home Owner)

家庭主人往往是房屋的决策者和主要使用者，对智能家居系统的便捷性、安全性和全屋场景模式有最高要求。他们需要在出行、回家、休息等不同场景下一键切换预定义模式，并实时查看全屋设备状态与安全监控画面。系统应提供“一键离家”“一键回家”“就寝模式”等快捷操作，将灯光、窗帘、安防、空调等设备自动联动，简化操作流程；同时，所有数据在本地和云端均加密存储，并支持日志审计与授权分享，确保家庭主人对整个系统拥有绝对的控制权与可追溯性。

2.2 家庭成员 (Family Member)

家庭成员包括配偶、子女及其他常住人员，他们对系统的操作要求更加直观与多样。青少年和中青年成员偏好手机 App 和语音交互，长辈则可能更习惯触摸屏和一键物理按键。系统需通过响应式设计和可视化大按钮减少学习成本，并提供基于角色的界面简化，将核心常用操作（如单灯开关、场景切换）置于首页，同时隐藏高级配置功能，保证家庭成员在无需复杂培训下即可安全便捷地使用。

2.3 访客/租客 (Guest/Tenant)

访客和短租租客在临时入住期间对智能家居系统的需求集中于临时授权与易用性。他们只需最基础的开关控制和场景调用，而无需访问家庭内部其他敏感设备。系统应支持“一次性密码”或“时限授权”机制，将访客权限自动绑定至客厅灯、空调等指定设备，并在授权到期后自动撤销，既保证了便捷性，也维护了家庭安全。

2.4 技术爱好者 (Tech Enthusiast)

技术爱好者具备一定的开发能力与 IoT 协议知识，他们期望系统具有高度开放性，能够通过 API、命令行或插件接入第三方服务。系统需提供完善的 RESTful/OpenAPI 文档、WebSocket 事件订阅接口与插件开发框架，允许技术爱好者自定义新设备适配、二次开发场景脚本或对接智能语音助手，从而形成生态化的社区驱动创新环境。

2.5 运维/管理员 (Operator/Admin)

在物业或企业场景下，运维人员和管理员负责大规模设备的部署、监控与故障排查。他们需要支持批量注册与分组管理、实时在线状态监控以及集中告警与日志导出功能。系统应提供管理后台的全局视图和导出工具，并支持通过脚本或命令行执行批量操作，帮助运维人员快速定位故障并生成报告，确保大规模部署时的高效运维与服务稳定性。

3 应用场景及业务流程

Yatmosphere 智能家居系统主要应用于以下典型场景：

- 离家/回家模式切换：用户外出或归家时，一键触发全屋设备联动，包括灯光、空调、安防、窗帘等，实现安全与舒适自动切换。
- 定时/环境触发自动化：基于预设时间、光照强度、温湿度或人流等条件自动执行场景，例如清晨自动打开窗帘并调整空调至舒适温度。
- 临时访客授权：对访客或租客提供临时设备访问权限，自动过期，确保安全性同时提升便捷。

本节重点梳理系统在三个典型场景（家庭模式切换、定时/环境自动化、访客临时授权）的应用流程，并通过流程图展示各环节关键功能和交互方式，最后给出整体业务流转视图。

3.1 家庭模式切换场景

3.1.1 应用流程

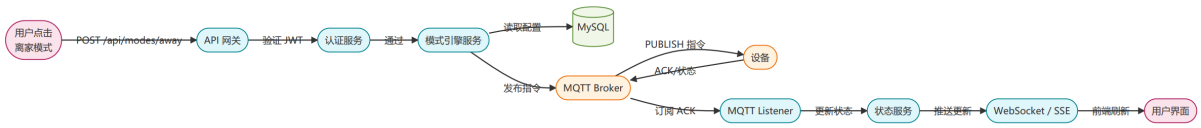

图 1: 家庭模式切换场景

当用户在移动端或 Web 界面点击“离家模式”按钮后，前端立即向后台的模式控制接口发送带有 JWT 的 POST 请求。接口网关接收请求后，会调用认证服务验证用户身份与权限，通过后将控制权交给模式引擎服务。模式引擎查询数据库中预先配置的“离家模式”场景模板，读取包括关灯、锁门、激活安防等一系列有序动作。随后，模式引擎将每个动作封装为 MQTT 控制消息，按配置顺序依次发布到对应设备主题。设备收到消息后执行相应操作，并通过 MQTT Broker 将执行结果（ACK 或最新状态）回传至后端的 MqttListener。MqttListener 将这些反馈写入状态服务并通过 WebSocket/SSE 实时推送给前端，前端界面在用户侧刷新显示整个模式切换的进度与结果，最终在几百毫秒内完成所有设备的联动，确保用户出行时全过程可视可控。

3.1.2 关键功能与交互方式说明

- **一键模式触发**：前端调用模式引擎 API，携带模式 ID 和用户权限 Token，进入身份校验与授权环节。
- **模式配置读取**：模式引擎服务从 MySQL 中加载预定义的设备动作序列及场景联动规则。
- **MQTT 指令发布**：按配置顺序对各设备发布控制命令，保证命令按用户预期依次执行。
- **ACK 及状态回传**：设备执行完毕后向 MQTT Broker 回报执行结果，后端通过 MqttListener 读取并存储最新状态。
- **实时状态推送**：状态服务将更新通过 WebSocket/SSE 通道广播到前端，前端界面实时显示模式切换进度及结果。

3.2 定时/环境自动化场景

3.2.1 应用流程

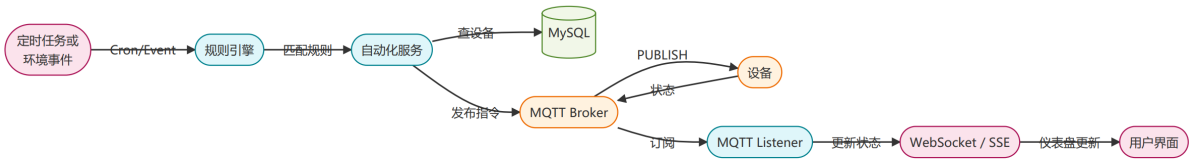

图 2: 定时/环境自动化场景

在用户预先配置好自动化场景（例如每天早上 7 点自动开窗帘并调整空调）后，系统会将定时任务或环境传感器事件注册到规则引擎。定时触发器或传感器一旦检测到设定的时间点或阈值变化，便向规则引擎发送事件。规则引擎根据事件类型检索所有与之匹配的场景规则，生成一份执行计划列表，并将其传递给自动化服务。自动化服务会批量读取涉及的设备清单，然后通过 MQTT Broker 并行或顺序地广播控制消息，触发设备动作。每个设备在执行完毕后依旧通过 MQTT 将最新状态上报至后端，状态服务收到后对外推送到前端仪表盘和通知中心。用户可在仪表盘中实时看到温湿度曲线、光照强度或人流等数据的变化，以及自动化任务的执行日志，整个流程无需用户干预即可完全自动化运行。

3.2.2 关键功能与交互方式说明

- **规则引擎触发**：支持 Cron 定时、光照/温湿度/人流等多种传感器事件作为触发条件。
- **自动化策略匹配**：规则引擎根据触发条件检索已配置的场景规则并生成执行计划。
- **批量指令发布**：自动化服务对场景内所有设备并行或按序发布 MQTT 控制命令，确保联动一致性。
- **状态回馈与监控**：设备通过 MQTT 回传执行结果，后端统一收集并通过状态服务更新至前端仪表盘。
- **历史记录归档**：所有自动化执行日志和设备反馈状态持久化存储，便于审计与回溯。

3.3 访客临时授权场景

3.3.1 应用流程

当房主需要为访客或短租租客开放部分设备权限时，会在前端填写访客姓名、授权设备列表与生效时长，并提交给权限管理服务。权限管理服务将该信息写入数据库，生成一个范围受限且带有过期时间的临时 Token，并返回给房主端显示。访客使用该 Token 在自己的界面登录后，前端在每次发起设备控制请求时都会携带此临时 Token，API 网关调用认证服务对 Token 进行合法性与权限范围校验，通过后才允许访问被授权的设备控制接口。访客的控制命令同样通过后端的控制服务发布至 MQTT Broker，设备执行后将 ACK 和最新状态经过后端推送至访客界面。系统后台会定时扫描过期 Token 并自动撤销对应权限，确保访客会话安全、简便且可审计。

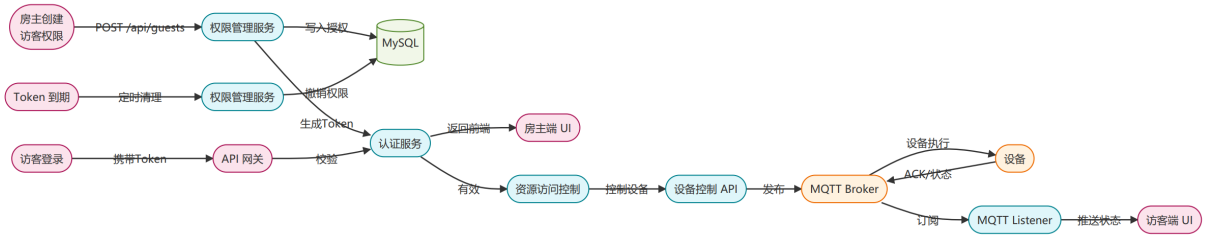

图 3: 访客临时授权场景

3.3.2 关键功能与交互方式说明

- **权限创建与存储**：房主通过后台 UI 提交访客姓名、授权设备列表及生效时长，权限管理服务写入 MySQL 并生成临时 Token。
- **Token 校验**：访客在前端输入 Token 登录，API 网关调用 AuthService 验证 Token 有效性与权限范围。
- **资源受限访问**：授权服务基于 Token 限定访客只能调用指定设备的控制接口，其他资源一律拒绝。
- **执行与回馈**：访客控制命令经 MQTT 发布至设备，后端通过状态服务和 WebSocket/SSE 将执行结果实时推送给访客界面。
- **自动过期与清理**：系统定时任务监测 Token 有效期，权限到期后自动从数据库删除并禁止进一步访问。

4 平台及技术架构

4.1 系统架构

Yatmosphere 的整体业务流转由「前端界面 → API 网关 → 认证/授权服务 → 业务逻辑层 → 数据存储与消息中间件 → 设备」以及「设备 → 消息中间件 → 后端状态服务 → HTTP 接口 → 前端界面」两条主链路组成。用户所有触发行为先走 API 网关并通过安全校验，然后进入相应的业务流程（如模式引擎、自动化服务或权限管理），业务层既与 MySQL 数据库交互又与 MQTT Broker 通信，实现对设备的指令发布与状态接收。设备执行结果通过 MQTT 回传到后端，后端状态服务将最新状态写入数据库，并通过 HTTP 接口供前端查询或定期轮询获取，实现端到端的交互与展示。整个系统在高性能并发与安全可控的基础上，保证用户体验的流畅与配置运维的高效。

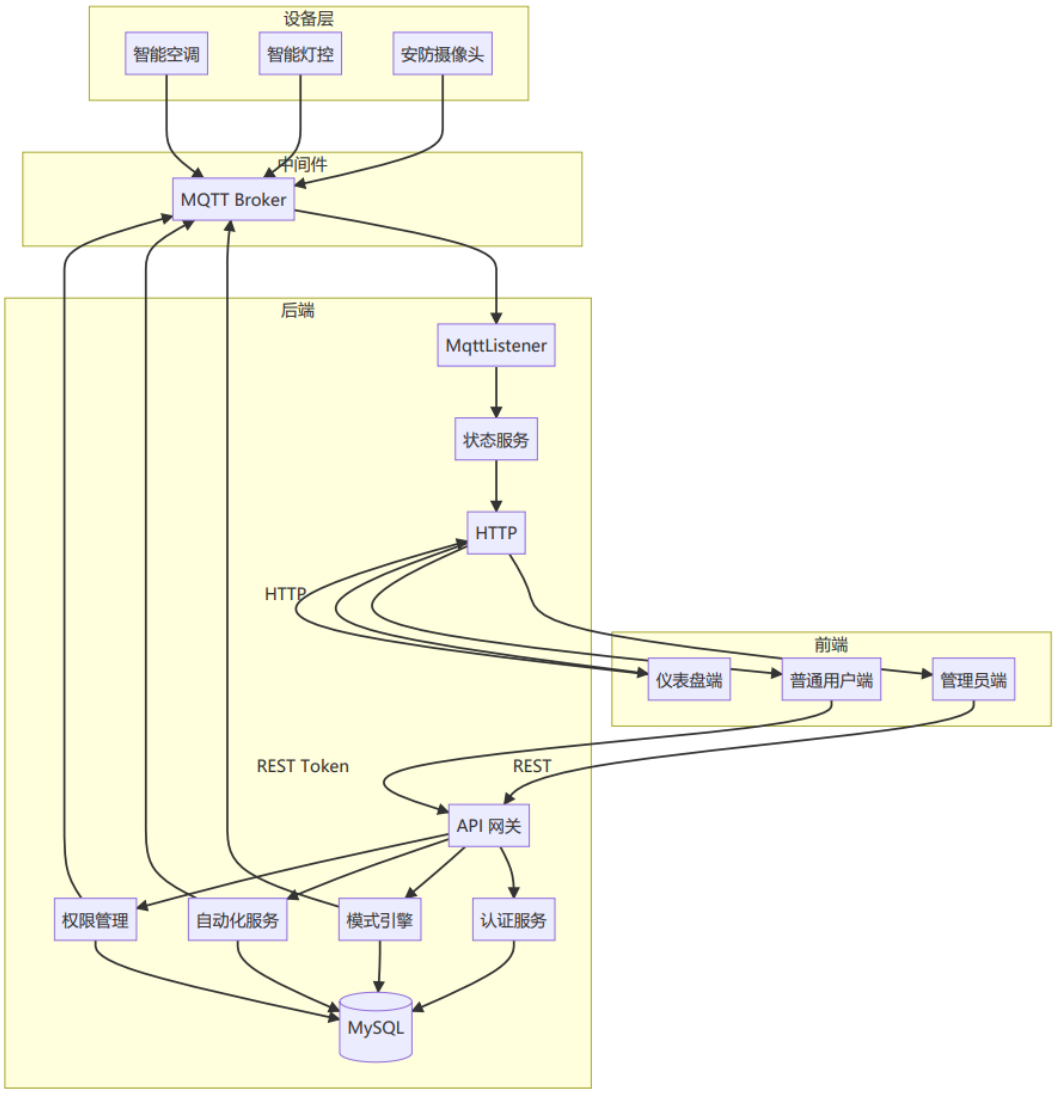

图 4: 系统架构

4.2 硬件平台

4.2.1 客户端与设备

Yatmosphere 客户端覆盖 Web、iOS/Android 应用及语音助手，各终端需支持 TLS 1.3、MQTT-over-WebSocket 等协议。智能设备包括 Zigbee、Wi-Fi、Bluetooth Mesh 等协议网关与终端传感器，最低硬件配置为 64 KB RAM、MCU 主频 80 MHz，并预装轻量级 MQTT 客户端固件。所有设备通过边缘网关接入，网关配置 4 核 CPU、8 GB 内存，用于本地消息缓存、初步处理与策略下发。

4.2.2 服务器与边缘

云端采用 16 核 ×16GB 内存 Ubuntu 22.04 x64 的系统，并根据业务需求横向扩展；高并发场景下可选配 NVIDIA T4 GPU 节点，用于深度学习模型推理与语音识别。边缘节点部署在用户侧或局部机房，负责本地 MQTT Broker、WebSocket 缓存、定时任务触发与离线数据同步。云与边缘之间建立双向 TLS 隧道，并根据 RTT 与负载动态切换流量，兼顾实时性与稳定性。

4.3 网络环境

4.3.1 混合接入与隔离

平台支持家庭 Wi-Fi、4G/5G 等多种接入方式，优先使用带宽高、时延低的本地网络，网络抖动时自动切换至 4G/5G 回落模式。云与边缘、服务间通信通过 IPsec VPN 或 VPC Peering 相互隔离，配合网络 ACL 与安全组策略，防止未经授权的横向访问，并在关键链路部署 WAF 与 DDoS 防护，保证外部接口安全可用。

4.3.2 性能保障与时延控制

对外带宽需保证 100Mbps，平均网络往返延迟 <50ms；内部网络建议 10Gbps 直连，时延 <5ms。边缘与云中心 RTT 控制在 <20ms，满足对实时控制命令（<250ms 闭环）和状态更新（<300ms）的严格要求，通过 QoS 策略和网络预留带宽优先级，确保用户体验无感切换。

4.4 技术架构

4.4.1 容器化与编排

设备层服务（MQTT）以 Docker 容器交付，服务发现与健康检查。集群部署多可用区副本，Controller Manager 与 Scheduler 高可用，结合 Horizontal Pod Autoscaler 根据 CPU/内存及自定义指标动态扩缩容，实现“弹性伸缩、无感扩容”。

4.4.2 数据存储与缓存

系统对不同场景下的数据选型精细：用户、权限等事务性数据开启主从复制与分片；设备与规则日志写入 SQL 以支持灵活查询；高频传感器与状态时序数据用于报表和告警；缓存热点会话、Token 和最近设备状态，以毫秒级响应频繁读请求。所有存储实例均运行在集群模式，配合自动故障转移与快照备份，保证数据安全与高可用。

4.5 性能与安全性分析

4.5.1 响应时间分析

- 端到端时延拆分：客户端 → API 网关：平均网络时延 <50ms；API 网关 → Auth-Service（鉴权）<10ms；鉴权 → 业务服务（Chat/Alert）<100ms ；业务服务 → ModelInference：流式推理 <100ms；服务整合 → 客户端返回：总体 <300ms。
- 关键路径加速：对话首包（用户请求到首条回复）路径采用预热模型实例和 GPU 异步调用，确保首包 <200ms。数据查询接口通过读写分离和列存索引，保障单条查询 <100ms。

4.5.2 安全性分析

- 威胁与风险识别：网络层攻击：DDoS、流量劫持；应用层威胁：SQL 注入、XSS、RPC 劫持；数据层风险：敏感数据泄露、未授权访问。
- 安全防护策略：网络防护：部署 WAF、DDOS 防护、IP Blacklist 和 GeoIP 白名单；应用加固：统一输入校验和输出编码，使用安全框架（Spring Security 或 JWT 防篡改）；数据加密：传输层 TLS 1.3，存储层使用 AES-256 加密并通过 HSM 管理密钥；身份认证与授权：OAuth2.0+MFA，多租户隔离及最小权限原则的 RBAC；审计与合规：日志不可篡改（WORM 存储）、定期安全审计和渗透测试，符合 HIPAA/GDPR/PIPL 合规要求。

5 非功能需求与性能

为了确保系统在实际场景下既能稳定提供核心功能，又能满足高可用、快速响应、良好扩展与可维护性，并符合行业安全标准，特提出以下非功能性需求：

5.1 性能要求

5.1.1 响应时间

- 前端控制界面（Dashboard）对用户交互动作（如开关灯、调节温度等）的点击响应应在 100 毫秒以内完成路由跳转与界面更新。

- 从前端发起 HTTP 请求至后端 Spring Boot 接口并返回结果的往返时延（TTFB）不得超过 200 毫秒；在局域网环境下，平均时延应控制在 50–100 毫秒之间。
- MQTT 消息从前端（或后端）发布到对应虚拟设备，再由设备反馈到后端并回传前端的全链路延迟不超过 500 毫秒，以保证用户感知的实时性。

5.1.2 吞吐量与并发

- 系统应能支持至少 300 并发连接（前端用户会话）并发发起操作请求（平均每秒 300 次请求），高峰期压力下响应成功率不低于 80%。
- 后端 Spring Boot 服务须能处理每秒至少 1000 条 MQTT 消息，以应对多个设备层同时发起的大量状态更新或数据上报。

5.1.3 数据存储与读写

- 数据库（如 MySQL 或 PostgreSQL）单机读写性能需达到每秒 200 次 CRUD 操作，在 1 万条设备历史记录规模下，单次查询时间不超过 100 毫秒。
- 日志文件写入需异步化，避免高并发阻塞主业务线程。单天日志量控制在 500MB 以内，定期归档与压缩。

5.2 可用性与可靠性

5.2.1 系统可用性

- 后端服务（HTTP API 和 MQTT Broker）年可用性需达到 99.9%（年度宕机时间不超过 8.76 小时）。
- 前端静态资源应部署到 CDN 或使用反向代理，保证页面加载时间始终不超过 2 秒。

5.2.2 故障恢复

- 核心组件须具备自动重启与自愈机制，故障后 30 秒内自动切换至备用实例或重启，1 分钟内恢复服务。
- 关键业务操作（如设备状态变更、用户登录注册）应采用幂等设计，并结合 ACK 与重试机制，保障数据一致性。

5.2.3 监控与告警

- 部署 Prometheus+Grafana 实时监控系统关键指标，设定阈值告警。
- 服务响应失败率超过 1%、CPU 利用率超 80%、或数据库连接池耗尽时，通过邮件与钉钉机器人即时通知。

5.3 可扩展性

5.3.1 水平扩展

- 后端应用应无状态，支持通过 Kubernetes 或 Docker Compose 容器化快速扩容。单节点满载后，3 分钟内扩容至 N+1 节点并加入负载均衡。
- MQTT Broker（如 EMQX）需配置集群，支持至少 50 台虚拟设备同时连接，节点可动态加入/移除，线性扩展性能。

5.3.2 数据库扩展

- 数据库支持读写分离，通过主从复制或分片，将读请求分散至只读从库，保障高写入量下的查询性能。

5.4 可维护性

5.4.1 模块化与解耦

- 前端 Vue 项目采用组件化开发，统一代码规范（ESLint+Prettier），逻辑清晰、职责单一，公共逻辑集中管理。
- 后端分层架构（Controller、Service、Mapper、Entity），模块通过接口解耦，便于单元测试与后续扩展。

5.4.2 文档与注释

- 所有公开接口（REST API、MQTT Topic 规范）使用 OpenAPI 或 Markdown 文档详细描述。
- 核心业务代码需有必要注释，特别是调度逻辑、消息处理等，保障新成员一周内掌握。

5.4.3 自动化测试与 CI/CD

- 后端编写单元测试（JUnit 5）与集成测试，业务逻辑覆盖率不少于 80%。

- 配置 CI/CD 流水线（如 GitHub Actions 或 Jenkins），实现提交即自动构建、测试、部署，测试未通过禁止合并。

5.5 安全性

5.5.1 身份认证与授权

- 后端使用 JWT 或 Spring Security 完成认证，所有 REST API 校验 Token 合法性与角色权限，防止越权。
- MQTT 通信采用用户名/密码或证书认证，Topic 层级配置 ACL，限制访问权限。

5.5.2 数据加密与传输安全

- 前后端通信强制 HTTPS，证书由可信 CA 签发；MQTT 启用 TLS（8883 端口）。
- 敏感数据如密码必须使用 BCrypt、PBKDF2 或 Argon2 哈希，不得明文存储。

5.5.3 安全漏洞防护

- 后端接口防范 SQL 注入、XSS、CSRF，参数化查询，前端对用户输入过滤转义。
- 定期安全扫描与依赖库升级，修复已知漏洞。

5.6 可用性与用户体验

5.6.1 界面友好性

- 前端设计响应式，适配桌面与移动端。重要控件尺寸不少于 44×44px，操作精准。
- 通信失败或设备离线及时提示（弹窗/Toast），提供重试按钮与操作反馈。

5.6.2 易用性测试

- 核心功能上线前，进行不少于 10 人次易用性测试，优化布局与提示，保证新用户 1 分钟内完成注册并控制设备。

5.7 兼容性与可移植性

5.7.1 浏览器兼容

- 前端适配主流浏览器（Chrome、Firefox、Edge）最新版及前两版，使用 Polyfill 或 Babel 降级兼容，IE11 以下显示提示。

5.7.2 跨平台部署

- 后端服务支持 Linux（Ubuntu 20.04+）和 Windows Server 2019+，Docker 容器化部署，开发生产环境一致。
- 虚拟设备层兼容 Linux 与 macOS，Python 依赖用虚拟环境管理，避免系统库冲突。

6 总结

本需求分析文档围绕 Yatmosphere 智能家居控制系统的总体目标、用户画像、应用场景及业务流程、平台及技术架构、性能与安全性分析及非功能需求等方面展开全面剖析。首先，通过对家庭主人、家庭成员、访客/租客、技术爱好者和运维管理员五大核心用户群体的深度画像，明确了各角色在系统使用过程中的痛点与核心需求，为后续功能设计提供了精准定位；随后，结合“家庭模式切换”、“定时/环境自动化”和“访客临时授权”三大典型场景，从用户发起到后台指令下发、设备执行、状态回传和前端实时展示的业务闭环，详细梳理了端到端的交互逻辑与关键接口，确保整个流程在高并发与低延迟下依旧平稳可控。在技术层面，文档提出了基于前端—接入层—核心微服务—中间件—存储—边缘网关的六层解耦架构，通过 Mermaid 展示了系统全景图，并针对容器化部署、服务编排、通信机制、存储选型、缓存策略、模型推理等核心模块进行了细致阐述。在非功能需求部分，明确提出了系统在性能、可用性、安全性、扩展性与可维护性方面的具体要求，为项目后续设计开发与实施提供坚实依据。

通过本需求分析，Yatmosphere 项目团队将能够在满足用户需求与业务目标的基础上，实现高效、可靠、安全和可持续演进的智能家居解决方案。