体系结构设计文档(SAD)

小组名称：_____软工加油队_____

小组成员：____付翔宇、冯彬、孟维汉、翟江浩、杨雨泽_____

项目名称：_____人才招募系统_____

年　级：☐ 一年级 ☐二年级 ☑三年级

山东大学制

目录

1.引言

1.1 文档目的

SAD 文档在整个项目生命周期中也发挥着版本控制和设计回溯的重要作用，有助于在迭代开发过程中追踪架构变更，提升团队协作效率。本体系结构设计文档（SAD）旨在全面描述人才招募系统的软件结构，帮助开发人员和管理人员验证软件的正确性、一致性与完整性，提供软件设计、确认、验证和成本估计的基准。系统涵盖用户信息管理、岗位推荐、企业信息发布、人才筛选等功能，支持多终端访问，具备良好的横向扩展能力，能够应对用户数量快速增长的情况。

1.2 背景与范围

本系统为人才招募系统，由 SDU2025“软工加油队”项目组开发，服务于有求职需求的个人与人才需求的企业。
前端框架：Vue 2.6.11；后端环境：NodeJS v16.18.0 + Express；
数据库：MySQL；接口测试工具：Postman。

1.3 定义、属于与缩略词

SAD: Software Architecture Document
API: Application Programming Interface
OSS: Object Storage Service
JWT: JSON Web Token
SQL: Structured Query Language

1.4 参考资料

IEEE-42010
《计算机软件文档编制规范》GB/T8567-2006
《规格需求分析》（SRS）

1.5 文档结构

本文件详细说明软件结构，包括体系结构、数据结构和接口设计，确保文档内容的正确性、完整性、一致性与可验证性。为了适应未来潜在的业务发展，本架构还应支持微服务改造与容器化部署能力，如通过 Docker + Kubernetes 实现弹性扩容。

2.架构目标与约束

2.1 架构设计目标

提供直观、易用的用户界面，实现清晰的前后端分离。
后端提供完善的 API 接口，确保系统安全性、可靠性、高效性与扩展性。
数据处理模块具备防止 SQL 注入和安全验证机制，保证数据完整性和隐私性。

2.2 系统关键质量属性

性能：响应时间≤3000ms；更新≤2000ms；传输≤1000ms；计算≤20000ms。
安全性：敏感数据加密、权限控制、用户认证（JWT）。
可扩展性：支持水平扩展。
可靠性：提供异常处理机制，支持并发请求。

2.3 技术与业务约束

技术限制：Windows10/11，MySQL 数据库，Vue+SpringBoot 架构。
法规遵循：确保数据传输与存储符合安全规范。

3.架构概览

3.1 系统背景与整体功能

系统服务包括用户注册登录、岗位信息发布与查询、简历管理、企业信息维护、管理员审核管理。此外，前端开发过程中采用了 Vue Router 进行路由控制，使用 Axios 处理 HTTP 请求，配合 Element Plus 快速构建响应式 UI 界面，有效提升用户体验。

3.2 架构风格与模式

采用前后端分离架构（Vue + SpringBoot），符合分层架构设计思想。
模块划分为表现层、业务逻辑层、数据访问层。

3.3 技术栈概览

前端：Vue.js + Element Plus
后端：Spring Boot + Node.js

数据库：MySQL 8.0
工具链：IntelliJ IDEA, Postman, Redis
存储服务：阿里云 OSS

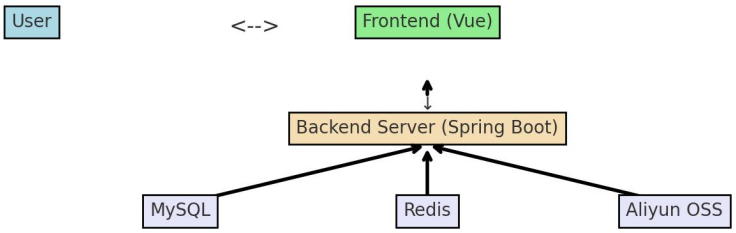
系统边界清晰，各模块之间通过定义良好的接口协议进行交互，确保模块间低耦合、高内聚，为后续维护和升级提供便利。

4.系统上下文

4.1 外部系统和接口

MySQL：提供数据存储与访问支持。
Redis：用于缓存和用户会话管理。
OSS：用于存储简历和头像等静态文件。

4.2 系统边界图



4.3 主要用户与交互关系

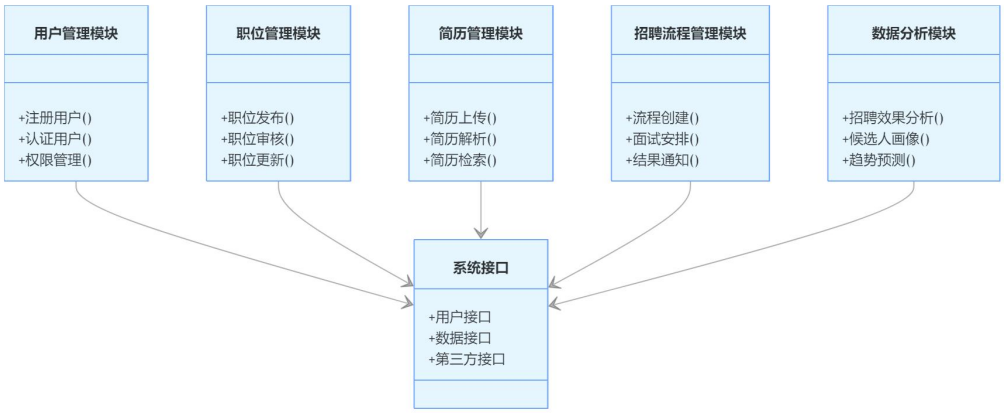
个人用户：注册、登录、简历管理、岗位查询。
企业用户：岗位发布、企业信息管理、人才查询。
管理员：用户审核、权限管理。

5.架构视图

5.1 逻辑视图

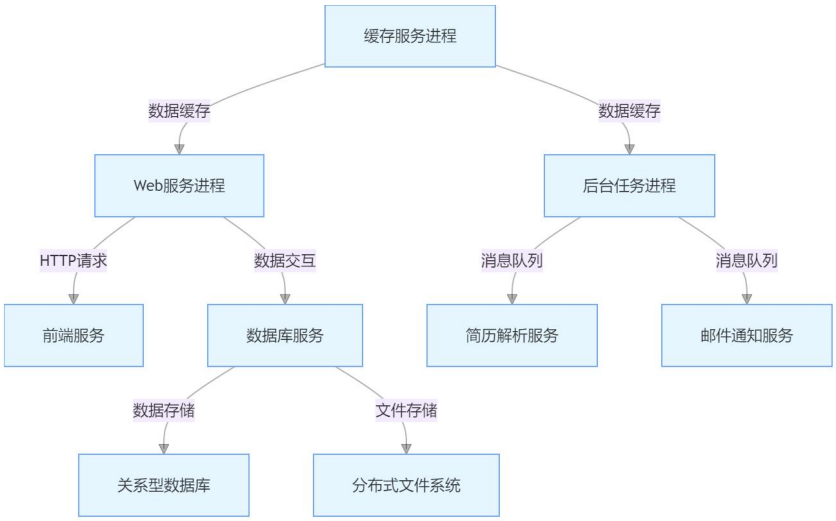
逻辑视图作为系统架构的核心抽象，通过分层模块化设计展现人才招募系统的功能组织

结构。系统以领域驱动设计思想为指导，将业务划分为用户管理、职位管理、简历管理、招聘流程管理和数据分析五大核心领域。用户管理领域采用多角色认证授权模型，通过统一身份服务实现求职者、招聘方和管理员的权限隔离；职位管理领域构建了基于行业分类、职能标签和职级体系的多维职位模型，支持智能搜索与推荐；简历管理领域集成自然语言处理技术，实现简历内容的自动解析、关键词提取和语义匹配；招聘流程管理领域通过工作流引擎支持自定义招聘阶段配置，实现从简历筛选、面试安排到录用决策的全流程数字化管理；数据分析领域采用维度建模构建招聘数据仓库，通过 OLAP 分析和数据可视化技术提供招聘效率、质量和成本的多维度分析洞察。各领域通过定义清晰的领域服务接口实现松耦合协作，共同构成系统的核心业务能力。



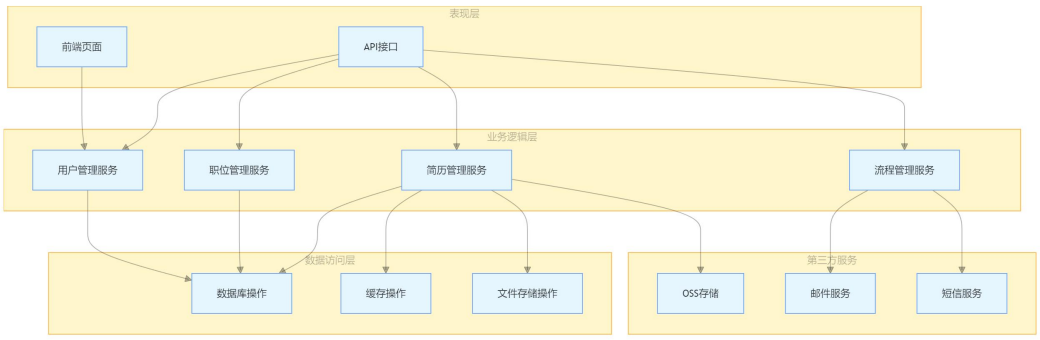
5.2 进程视图

进程视图聚焦于系统运行时的并发处理与资源分配策略，采用微服务架构模式实现高并发场景下的弹性伸缩。前端采用多进程架构设计，主渲染进程负责 UI 交互处理，WebWorker 进程承担简历文件解析等耗时操作，Service Worker 进程实现离线缓存和消息推送功能，提升用户体验。后端系统基于容器编排平台构建分布式微服务集群，Web 服务器集群通过负载均衡器接收外部请求，路由至对应的业务微服务；业务微服务采用异步处理模式，通过消息队列实现与后台任务服务的解耦通信，支持简历解析、邮件通知等耗时操作的异步执行；数据层采用读写分离架构，主库处理事务性写操作，从库集群承担查询负载，结合分布式缓存技术提升数据访问性能。系统通过熔断器、限流阀和自动扩缩容机制保障高可用性，通过分布式日志聚合和 APM 工具实现全链路监控。



5.3 开发视图

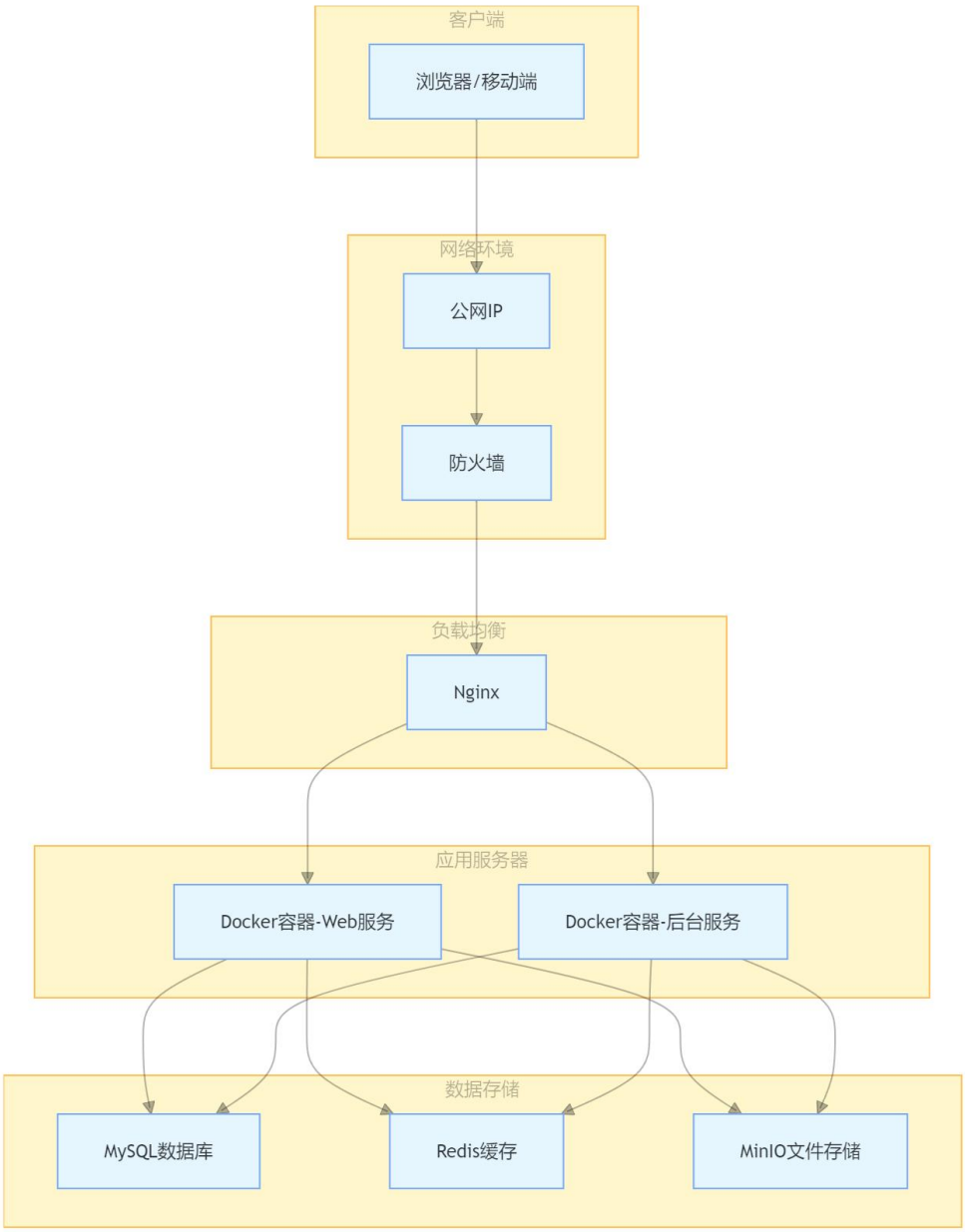
开发视图以模块化设计思想为指导，构建层次分明的代码组织结构和高效协作的开发工具链。前端采用组件化开发模式，基于 Vue.js/React 框架构建视图层，通过状态管理库实现数据流动的单向性和可追溯性，利用 TypeScript 的静态类型检查提升代码质量；后端采用六边形架构模式，清晰分离核心业务逻辑与外部依赖，通过 Spring Boot/Spring Cloud 构建微服务框架，实现服务注册发现、配置中心和 API 网关等基础设施；数据访问层采用 ORM 框架实现对象关系映射，结合 MyBatis-Plus 等工具提升数据操作效率；测试层构建包含单元测试、集成测试和端到端测试的多层次测试体系，通过自动化测试框架保障代码质量。开发流程遵循 Git Flow 分支策略，通过 Jenkins/GitLab CI 实现持续集成，结合 Docker 和 Kubernetes 实现自动化部署，形成从代码提交到生产环境的完整 DevOps 流水线。



5.4 部署视图

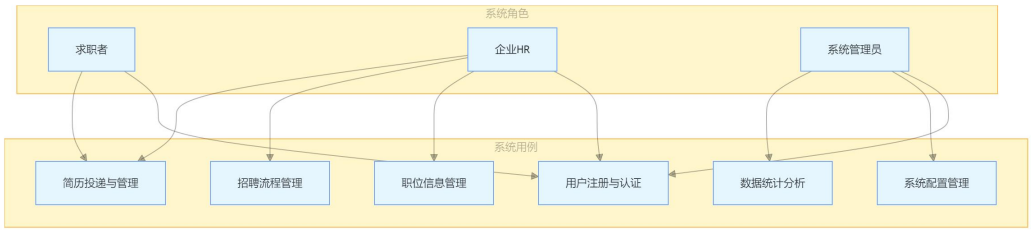
部署视图采用云原生架构设计，构建高可用、可扩展的分布式部署体系。系统采用三级

部署架构，接入层通过 CDN 分发静态资源，减轻服务器负载，利用负载均衡器实现流量的智能分发和健康检查，结合 Web 应用防火墙提供安全防护；应用层基于 Kubernetes 容器编排平台部署微服务集群，通过水平 Pod 自动扩缩容 (HPA) 和集群自动扩缩容 (CA) 实现资源的动态分配，利用服务网格技术实现微服务间的通信治理；数据层采用主从复制、分片集群等技术构建高可用数据库架构，结合分布式文件系统实现简历附件等非结构化数据的可靠存储，通过定期备份和灾备演练保障数据安全。监控层采用 Prometheus 和 Grafana 构建全方位监控体系，实现系统性能、应用指标和业务数据的实时监控与告警；日志层通过 ELK Stack 实现日志的收集、存储和分析，为问题排查和业务决策提供数据支持。



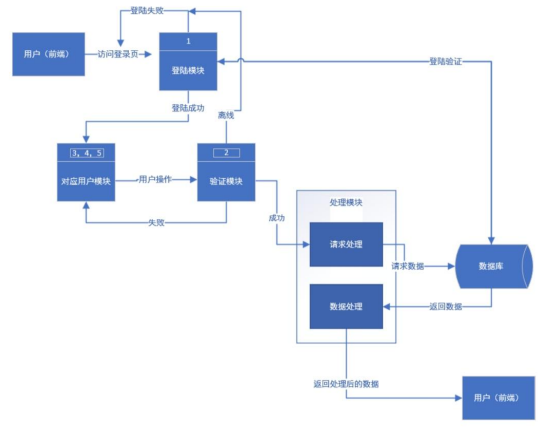
5.5 用例视图

用例视图以用户需求为中心，通过 actor 与 use case 的交互关系完整展现系统的功能边界。系统定义了求职者、招聘方和系统管理员三类主要角色，每个角色对应特定的业务场景。求职者通过系统完成从注册登录、简历创建到职位搜索、申请投递的全流程操作，系统支持多维度职位筛选和智能推荐，提供申请状态实时跟踪和面试通知提醒功能；招聘方通过企业认证后可发布和管理职位，利用系统提供的简历搜索和筛选工具快速定位合适候选人，通过自定义招聘流程模板实现面试安排、评估和录用决策的数字化管理；系统管理员负责维护系统的基础数据和运营规则，包括用户管理、企业审核、职位分类维护等操作，通过系统监控功能实时掌握系统运行状态，利用数据分析工具生成各类运营报表，为系统优化提供决策支持。各用例之间通过扩展、包含和泛化等关系形成有机整体，共同支撑人才招募业务的高效运转。



6.接口设计

6.1 内部模块接口



为确保内部模块间交互的高效性与稳定性，采用清晰且规范的接口设计。函数调用接口遵循单一职责原则，每个函数专注于完成一项特定任务，减少函数功能的复杂性。消息队列接口则保证异步通信的可靠性，支持高并发场景下的模块解耦。

以人才招募系统中简历筛选模块与面试安排模块为例。简历筛选模块提供

filterResumes(jobId, criteria) 函数，其中 jobId 为职位 ID（整数类型）， criteria 为筛选标准（JSON 格式字符串，包含学历、工作经验等筛选条件），该函数返回符合条件的简历 ID 列表（数组类型）。面试安排模块监听简历筛选模块发送到消息队列的简历筛选结果消息，消息结构为 { "jobId": 123, "selectedResumeIds": [1, 3, 5] } ，面试安排模块接收到消息后，根据预设规则安排面试。

对于函数调用接口，使用自定义异常类来处理可能出现的错误。如当 filterResumes 函数中数据库查询失败时，抛出 DatabaseQueryException ，并在调用方进行捕获和处理，向用户返回友好的错误提示。对于消息队列接口，设置消息重试机制，当消息处理失败时，可在一定时间间隔后自动重试一定次数，若仍失败则记录错误日志并通知管理员。

6.2 外部系统接口

严格遵循 RESTful 设计原则，接口地址具有可读性和语义化。例如获取职位详情接口为 /jobs/{jobId} ，其中 {jobId} 为职位的唯一标识，通过路径参数传递。请求方法按照标准规范使用，GET 用于获取资源，POST 用于创建资源，PUT 用于更新资源，DELETE 用于删除资源。

在与招聘信息聚合平台对接时，我方系统作为客户端发起请求。以获取聚合平台上的热门职位列表为例，向 https://aggregation-platform/api/hot - jobs 发送 GET 请求，请求头中包含认证信息（如 API 密钥），请求体为空。聚合平台响应数据格式为 JSON，例如 { "jobs": [{ "id": 1, "title": "Java 开发工程师", "company": "ABC 公司" }, { "id": 2, "title": "前端开发工程师", "company": "DEF 公司" }] } 。同时，为应对第三方系统可能的响应延迟或不可用情况，设置合理的超时时间和熔断机制，当超时或连续多次请求失败时，触发熔断，不再向该第三方系统发送请求，并向用户返回提示信息。

除使用 HTTPS 协议外，还采用 OAuth 2.0 协议进行授权认证。在与外部系统建立连接前，我方系统先引导用户在外部系统进行授权，获取授权码后，通过交换流程获取访问令牌（Access Token），后续请求携带访问令牌进行身份验证，确保只有经过授权的请求才能访问外部系统资源。

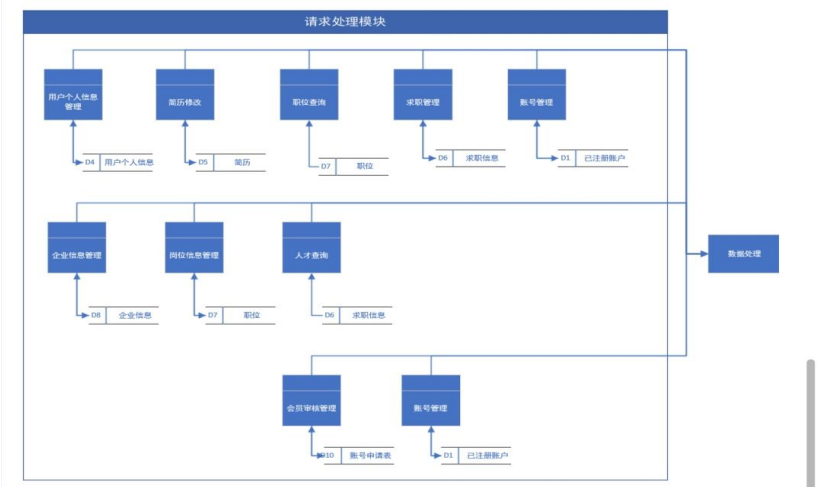
6.3 接口协议规范与数据格式

HTTP 协议版本采用 HTTP/2 或更高版本，利用其多路复用、头部压缩等特性提升接口性能。在请求头和响应头中，严格按照规范设置必要字段，如 Content - Type 字段在发送 JSON 格式数据时设置为 application/json ，Accept 字段表明接收数据格式偏好等。同时，遵循 RESTful API 状态码规范，2xx 系列表示成功（如 200 表示请求成功，201 表示资源创建成功），4xx 系列表示客户端错误（如 400 表示请求错误，401 表示未授权），5xx 系列表示服务器错误（如 500 表示服务器内部错误）。

JSON 数据格式定义明确的字段类型和约束。例如在用户注册接口的请求体中，定义 { "username": "string", "password": "string", "email": "string", "phone": "string" } ， 其 中 username 长度限制在 3 - 20 个字符， password 长度至少 6 位且包含数字和字母等规则。在服务端接收数据时，使用 JSON Schema 进行数据验证，若数据不符合定义规则，立即返回错误响应，告知客户端错误原因。

7.数据架构

7.1 数据模型



以人才招募系统的核心业务流程为基础构建 ER 图。“求职者” 实体包含属性如姓名、身份证号、联系方式、邮箱等；“招聘企业” 实体包含企业名称、统一社会信用代码、企业地址、联系电话等属性；“职位” 实体包含职位 ID、职位名称、薪资范围、工作地点、岗位职责、任职要求等属性；“简历” 实体包含简历 ID、求职者 ID、教育经历、工作经历、项目经验等属性。实体间关系通过联系来表示，“求职者” 与 “简历” 之间的一对多关系通过在 “简历” 实体中添加求职者 ID 外键来体现；“招聘企业” 与 “职位” 的一对多关系通过在 “职位” 实体中添加企业 ID 外键来体现；“求职者” 和 “职位” 的多对多关系通过创建 “应聘记录” 关联表，包含求职者 ID、职位 ID、应聘时间等字段来实现。

从面向对象编程角度设计类图。以 “求职者” 类为例，定义私有属性如 name （姓名）、 idNumber （身份证号）、 contactInfo （联系方式）等，通过公共的访问器（getter）和修改器（setter）方法进行属性访问和修改，同时定义业务方法如 applyForJob(Job job) 用于求职者申请职位，方法内部实现申请逻辑，如检查是否已申请过该职位、更新应聘记录等。“职位” 类包含 title （职位名称）、 salaryRange （薪资范围）等属性，以及 postJob() （发布职位）、 closeJob() （关闭职位）等方法。类与类之间的关系通过关联、聚合等方式表示，与 ER 图中的实体关系相对应，确保数据模型在不同视角下的一致性。

7.2 数据存储方案

对于人才招募系统中结构化、强事务性的数据，选择 MySQL 作为存储数据库。创建多个表来对应 ER 图中的实体，如 job_seekers 表存储求职者信息，包含字段 id （主键）、 name 、 id_number 等； recruiting_companies 表存储招聘企业信息，包含字段 company_id （主键）、 company_name 、 credit_code 等； jobs 表存储职位信息，包含字段 job_id （主键）、 company_id （外键，关联 recruiting_companies 表的 company_id ）、 job_title 等。利用 MySQL 的索引机制，在经常查询的字段上创建索引，如在 jobs 表的 job_title 字段上创建普通索引，提高职位搜索的查询效率。同时，合

理设置数据库的事务隔离级别，如采用 REPEATABLE READ 级别，保证在事务执行过程中数据的一致性和可重复性。

Redis 主要用于缓存热点数据和频繁访问的数据，以提升系统响应速度。例如将热门职位信息缓存到 Redis 中，设置合理的缓存过期时间（如 1 小时）。当用户请求热门职位列表时，先从 Redis 中查询，若存在则直接返回，若不存在再从 MySQL 数据库中查询并将结果存入 Redis 缓存。此外，对于用户登录后的会话信息，也存储在 Redis 中，利用其原子操作特性实现高效的会话管理，如设置会话过期时间、更新会话状态等操作。

用户上传的简历文件存储在本地文件系统（如基于 Linux 的 ext4 文件系统）中。在服务器上创建专门的简历存储目录，按照一定规则（如按日期或用户 ID 分区）进行文件存储。在数据库中（如 MySQL 的 resumes 表）记录简历文件的相关元信息，包括 resume_id（主键）、 job_seeker_id （关联求职者 ID）、 file_path （文件路径）、 file_name （文件名）、 upload_time （上传时间）等。同时，为保障文件安全，设置文件系统的访问权限，只有相关服务进程具有读写权限，防止文件被非法访问或篡改。

7.3 属于一致性与同步策略

在 MySQL 数据库中，利用事务机制确保同一事务内操作的数据一致性。例如在求职者申请职位的操作中，涉及在 job_applications 表中插入一条应聘记录，同时更新求职者的应聘次数等操作，将这些操作封装在一个事务中。若其中任何一个操作失败，事务回滚，保证数据不会出现部分更新的不一致情况。此外，通过外键约束来维护表与表之间的数据一致性，如 jobs 表中的 company_id 外键引用 recruiting_companies 表的 company_id ，当删除 recruiting_companies 表中的企业记录时，若 jobs 表中存在相关职位记录，根据外键约束策略（如 CASCADE 级联删除或 RESTRICT 限制删除）进行相应处理，确保数据的完整性。

对于数据库（MySQL）与文件系统间的数据同步，采用基于事件驱动的机制。当用户上传新简历时，系统先将简历文件保存到文件系统指定目录，然后在 MySQL 数据库的 resumes 表中插入一条记录，包含文件路径等元信息。同时，发送一条消息到消息队列，通知相关服务（如简历解析服务）进行后续处理。若用户删除简历，在数据库中删除对应记录后，通过消息队列通知文件系统删除对应的简历文件。对于 MySQL 与 Redis 间的数据同步，当 MySQL 中职位信息发生更新（如职位薪资调整）时，通过数据库触发器或应用程序监听机制，捕获数据更新事件，然后删除 Redis 中对应的缓存数据，当下次请求该职位信息时，重新从 MySQL 中查询并更新 Redis 缓存，保证数据的一致性。

8.安全架构

8.1 身份验证与授权

除用户名密码和验证码外，引入基于时间同步的一次性密码（TOTP）技术作为多因素认证方式之一。用户在首次登录时，可选择绑定手机应用（如 Google Authenticator）作为身份验证器。系统为用户生成一个密钥，用户将密钥导入到身份验证器应用中，应用根据时间戳和密钥生成动态验证码。用户登录时，除输入用户名、密码和静态验证码外，还需输入

身份验证器应用中显示的动态验证码。系统在服务端通过相同的算法验证动态验证码的有效性，增加身份验证的安全性。同时，对于一些高风险操作（如修改用户重要信息、大额资金交易等），强制要求用户进行 MFA 验证，降低账户被盗用的风险。

在人才招募系统中，详细定义多种角色及其权限。求职者角色拥有查看职位信息、投递简历、修改个人基本信息等权限；招聘企业管理员角色拥有发布职位、查看应聘简历、筛选候选人、安排面试等权限；系统管理员角色拥有最高权限，包括管理用户账号、配置系统参数、监控系统运行状态、审核违规操作等。通过数据库中的权限表来存储角色与权限的映射关系，每个角色对应一组权限集合。在用户进行操作时，系统根据用户所属角色查询权限表，判断用户是否具有相应操作权限，若没有权限则返回权限不足的提示信息，禁止操作执行。同时，支持角色的动态分配和权限的灵活调整，以适应不同业务场景和组织架构变化。

8.2 数据加密机制

在系统服务器端配置 SSL/TLS 证书，支持 HTTPS 协议通信。SSL/TLS 协议通过握手过程建立安全连接，在握手阶段，客户端和服务器端协商加密算法套件（如 AES - 256 - GCM + SHA384 + ECDHE - RSA 等），服务器端向客户端发送数字证书进行身份验证，客户端验证证书有效性后，双方交换密钥材料，生成会话密钥。后续数据传输过程中，使用会话密钥对数据进行加密和解密。为确保安全性，定期更新 SSL/TLS 证书，及时修复证书漏洞，并关注加密算法的安全性，随着技术发展适时更换更安全的加密算法套件。同时，配置 HSTS（HTTP Strict Transport Security）策略，强制客户端在一定时间内只能通过 HTTPS 访问系统，防止中间人攻击和协议降级攻击。

对于需要加密存储的敏感数据（如用户身份证号、银行卡号等），采用 AES - 256 加密算法（高级加密标准，256 位密钥长度）。在数据写入数据库前，使用预先生成并安全保管的密钥对数据进行加密，加密模式选择 CBC（Cipher Block Chaining，密码块链接）模式，增加加密的安全性。解密过程则在读取数据时，使用相同密钥进行反向操作。密钥管理方面，采用密钥管理系统（KMS）进行密钥的生成、存储、更新和销毁。密钥定期更换，旧密钥在一定时间内保留用于数据解密历史记录，同时严格控制密钥的访问权限，只有授权的系统组件才能获取和使用密钥。

8.3 安全威胁与防护措施

对于 SQL 注入攻击，除采用参数化查询外，还使用预编译语句。在使用编程语言（如 Java 结合 JDBC）与 MySQL 数据库交互时，使用 PreparedStatement 类来构建 SQL 语句，将用户输入作为参数传递，而不是直接拼接在 SQL 语句中。同时，对用户输入进行严格的合法性校验，如限制输入长度、检查输入字符集（只允许数字、字母、特定符号等）。对于 NoSQL 数据库（如 Redis）可能存在的注入风险，同样避免直接使用用户输入构建命令，采用安全的 API 调用方式。此外，定期进行 SQL 注入漏洞扫描，使用专业的扫描工具（如 sqlmap）对系统进行全面检测，及时发现并修复潜在漏洞。

在前端页面渲染用户输入内容时，采用白名单过滤机制。使用安全的 HTML 解析库（如 DOMPurify）对用户输入进行处理，只允许白名单内的 HTML 标签和属性存在，过滤掉可能包含恶意脚本的标签（如 <script> 标签）和危险属性（如 onclick 等事件属性）。同时，在 HTTP 响应头中设置 Content - Security - Policy （CSP）策略，限制页面可以加载的资源

来源，例如设置 default - src'self'; script - src'self' ，表示只允许从当前站点加载脚本，防止外部恶意脚本注入。对于用户提交的富文本内容（如职位描述中的格式文本），采用服务器端渲染（SSR）技术，在服务器端对内容进行安全处理后再发送到客户端，减少客户端渲染时可能出现的 XSS 风险。

针对 OWASP Top 10 中的其他威胁，如跨站请求伪造（CSRF），采用 CSRF 令牌机制。在用户登录后，服务器为每个用户会话生成一个唯一的 CSRF 令牌，存储在用户的会话中，并在表单或 AJAX 请求中包含该令牌。服务器在处理请求时，验证请求中的 CSRF 令牌是否与会话中的令牌一致，若不一致则拒绝请求。对于不安全的直接对象引用，在访问敏感资源（如用户个人信息、特定职位详情等）时，进行严格的权限验证和对象所有权验证，确保用户只能访问自己有权限访问的对象。同时，定期对系统进行安全审计，包括日志审计、代码审计等，及时发现和处理潜在的安全问题。

9.可扩展性与可维护性设计

9.1 模块解耦设计

在模块接口设计中，遵循接口隔离原则，避免模块依赖不必要的接口功能。例如在人才招募系统的简历解析模块与其他模块交互时，简历解析模块提供专门的简历解析接口，如 parseResume(resumeFile) ，仅负责将简历文件解析为结构化数据，不涉及其他无关业务逻辑。其他模块（如简历筛选模块）仅依赖该解析接口获取解析后的简历数据，而不依赖简历解析模块的其他内部实现细节。这样，当简历解析模块内部算法升级或更换解析库时，只要接口保持不变，不会影响到依赖它的其他模块，降低模块间的耦合度。

采用依赖注入（DI）技术实现模块间的解耦。以 Spring 框架为例，在配置文件或通过注解方式定义模块间的依赖关系。例如，面试安排模块依赖求职者服务模块获取求职者信息，通过依赖注入，将求职者服务模块的实例注入到面试安排模块中。这样，面试安排模块不需要直接创建求职者服务模块的实例，而是由容器负责创建和注入。当需要替换求职者服务模块的实现（如切换到新的服务实现版本或不同的服务提供商）时，只需要在配置中进行修改，而无需修改面试安排模块的代码，提高了系统的可维护性和可扩展性。

9.2 插件化/微服务策略

人才招募系统采用插件化架构实现功能的动态扩展与灵活部署，通过定义标准的扩展点接口和插件加载机制，允许第三方开发者或内部团队开发独立的功能模块。系统核心框架负责提供基础服务（如用户认证、数据访问）和插件管理机制，插件则通过实现预定义的接口（如简历解析器、消息通知器）集成到系统中。这种松耦合设计使得系统能够在不修改核心代码的情况下引入新功能，例如集成第三方背调服务、AI 面试评估工具或特定行业的职位分类体系。插件通过统一的插件注册中心进行管理，支持热插拔特性，可在运行时动态安装、卸载或更新，极大提升了系统的灵活性和可维护性。

系统采用微服务架构模式实现服务的自治与独立部署，将核心业务拆分为用户服务、职位服务、简历服务、招聘流程服务等多个独立的微服务。每个微服务专注于单一业务能力，拥有独立的数据库和开发团队，通过轻量级 API（如 RESTful 接口或 gRPC）进行通信。服

务间采用异步消息机制（如 Kafka/RabbitMQ）实现解耦，确保业务流程的弹性和可扩展性。例如，简历投递操作会触发消息事件，通知招聘流程服务更新候选人状态，同时触发邮件服务发送确认通知。微服务通过服务注册与发现机制（如 Consul/Eureka）实现自动注册和动态发现，结合 API 网关（如 Spring Cloud Gateway/Kong）提供统一的入口点，实现请求路由、负载均衡和权限校验。

为确保微服务集群的稳定运行，系统实施全面的服务治理策略。采用断路器模式（如 Hystrix/Sentinel）防止级联故障，当服务出现异常时自动熔断并执行降级逻辑；通过限流与负载保护机制（如令牌桶算法）控制服务流量，避免资源耗尽；实现分布式事务管理（如 TCC 模式或 Saga 模式）保障跨服务业务操作的一致性；引入服务监控与告警系统（如 Prometheus+Grafana）实时监控服务运行状态，设置关键指标阈值触发告警。此外，通过分布式链路追踪（如 Zipkin/jaeger）实现请求全链路可视化，帮助快速定位故障点，提升系统可观测性。

系统采用 Docker 容器技术实现微服务的标准化打包和部署，将每个微服务及其依赖打包为独立容器，确保环境一致性。通过 Kubernetes 进行容器编排管理，实现服务的自动部署、扩缩容和健康检查。Kubernetes 的 Horizontal Pod Autoscaler (HPA) 根据 CPU 使用率或自定义指标自动调整服务实例数量，应对流量波动；Node Autoscaler 则根据集群资源使用情况动态调整底层计算节点数量，优化资源利用率。通过 Helm 图表管理应用部署配置，实现环境间的配置复用和版本化管理，结合 CI/CD 流水线实现代码提交到生产环境的自动化部署流程。

9.3 配置与版本管理方案

系统采用集中化配置管理方案，将应用配置与代码分离，实现配置的统一管理和动态更新。使用配置中心（如 Nacos/Apollo）存储所有微服务的配置信息，支持按环境（开发 / 测试 / 生产）、按服务进行配置隔离。配置中心提供配置版本控制、灰度发布和回滚功能，确保配置变更的安全性和可追溯性。应用启动时自动从配置中心拉取配置，运行时通过配置监听机制实时感知配置变更并动态刷新，无需重启服务。对于敏感配置（如数据库密码、API 密钥），采用加密存储和传输机制，结合 Kubernetes Secret 或 Vault 实现密钥的安全管理。

系统实施严格的版本管理策略，确保代码、依赖和基础设施的可追溯性和一致性。代码仓库采用 Git 进行版本控制，遵循 Git Flow 分支模型，主分支（master/main）用于生产发布，开发分支（develop）用于集成开发，功能分支（feature/）用于新功能开发，发布分支（release/）用于版本发布准备，热修复分支（hotfix/*）用于紧急修复。每个微服务独立进行版本管理，使用语义化版本号（如 v1.2.3）标识版本变更。依赖管理采用锁定机制，通过 package-lock.json 或 pom.xml 文件精确记录所有依赖的版本信息，确保构建环境的一致性。

系统采用蓝绿部署和金丝雀发布相结合的策略实现平滑发布。蓝绿部署通过准备两套完全相同的生产环境（蓝环境和绿环境），每次发布时切换流量到新版本环境，确保发布过程中服务始终可用。金丝雀发布则先将新版本部署到少量服务器，让部分用户访问新版本进行验证，收集反馈后再逐步扩大到全量用户。每个发布版本都关联唯一的构建标识，支持快速回滚到上一个稳定版本。通过自动化测试和冒烟测试确保新版本的质量，结合灰度发布策略降低发布风险。

系统采用基础设施即代码理念管理部署环境，使用 Terraform 定义和配置云基础设施资源，实现基础设施的版本化和自动化管理。Terraform 配置文件存储在代码仓库中，与应用代码一同进行版本控制，确保基础设施的变更可追溯。通过 Kubernetes manifests 或 Helm charts 定义应用部署配置，实现应用部署的标准化和自动化。CI/CD 流水线集成基础设施验

证和测试环节，确保基础设施变更与应用代码变更的一致性。通过 GitOps 实践，将生产环境状态与 Git 仓库状态保持同步，任何环境变更都通过提交代码触发，提升部署过程的透明度和可控性。

建立完善的配置与版本审计机制，记录所有配置变更和版本发布历史。配置中心保存配置变更的完整历史，包括变更人、变更时间、变更内容和变更原因，支持配置变更的回滚和审计查询。CI/CD 流水线记录每次构建和部署的详细信息，包括代码版本、依赖版本、构建时间和部署环境。通过集成日志聚合和监控系统，实现配置变更和版本发布的实时监控和告警，确保任何异常变更都能及时发现和处理。定期进行配置合规性检查，确保生产环境配置符合安全和合规要求。

10.架构决策记录

10.1 决策列表与原因

决策编号	决策内容	原因	责任人	日期
ADR-001	采用前后端分离架构	1. 解耦前端展示与后端逻辑，提升开发效率； 2. 支持独立部署与技术栈升级（如前端 Vue.js，后端 Node.js）	付翔宇	2025-05-26
ADR-002	数据库采用 MySQL + Redis + OSS	1. MySQL 存储结构化数据，支持事务与复杂查询； 2. Redis 缓存高频数据，提升响应速度； 3. OSS 存储非结构化文件，降低存储成本	冯彬	2025-05-26
ADR-003	身份认证使用 JWT 令牌机制	1. 无状态设计，支持分布式部署； 2. 结合 BCrypt 加密密码，提升安全性	孟维汉	2025-05-26
ADR-004	接口设计遵循 RESTful 规范	1. 统一接口格式，便于前后端联调； 2. 支持 Swagger 自动生成文档，提升协作效率	翟江浩	2025-05-26
ADR-005	采用弹性计算云服务器部署	1. 按需扩展资源，应对流量波动； 2. 利用云服务商高可用性保障（如阿里云 ECS + 负载均衡）	杨雨泽	2025-05-26

10.2 替换方案评估

决策项	替代方案	优点	缺点	最终选择原因
数据库选型	MongoDB 替代 MySQL	适合非结构化数据，查询灵活	不支持事务，复杂查询性能低	系统以结构化数据为主，需保证数据一致性
身份认证方式	Session-Cookie	兼容性好，实现简单	依赖服务器存储，分布式场景维护成本高	JWT 更适合前后端分离架构，支持跨域访问
前端框架选择	React 替代 Vue.js	生态成熟，适合大型应用	学习曲线较陡，项目初期开发效率低	Vue.js 更轻量,适合中小型项目快速迭代
部署方式	物理服务器自建	数据可控性高	硬件成本高，扩展灵活性差	云服务器成本更低，且提供自动容灾能力

10.3 决策影响分析

10.3.1 技术影响：

前后端分离架构引入的协作挑战：
前后端解耦设计虽提升开发并行效率,但增加了跨团队接口联调复杂度。需通过 Swagger 接口契约标准化 明确请求参数、响应格式及错误码规范，配合自动化测试工具（如 Postman）降低联调成本，确保前后端逻辑无缝对接。

JWT 认证机制的状态管理：
基于 JWT 的无状态认证需实现令牌生命周期管理，包括：过期策略：设置合理 TTL（如 2 小时），结合 Refresh Token 机制自动刷新访问令牌，避免用户频繁登录；安全加固：采用 HTTPS 传输令牌，禁止令牌在 Cookie 中明文存储，防范 CSRF 攻击。

10.3.2 运维影响：

云服务依赖下的数据可靠性保障：
系统部署依赖阿里云等第三方云服务商,需构建 多层级数据备份体系：实时备份：通过 OSS 跨区域复制功能，实现简历文件等非结构化数据的异地容灾；定期全量备份：每日凌晨对 MySQL 数据库进行全量备份，并通过冷存储（如阿里云 Archive Storage）降低长期存储成本。
Redis 集群的高可用性架构：
为避免缓存层单点故障，需部署 Redis 主从复制集群，并结合 Sentinel 实现自动故障转移（Failover）。同时，通过监控工具（如 Prometheus）实时追踪缓存命中率、内存利用率等

指标，动态调整集群节点数量。

10.3.3 成本影响：

云资源的弹性成本控制：
云服务采用 “按需付费” 模式，开发测试阶段可通过低成本实例（如阿里云突发性能实例）控制支出；生产环境需建立流量监控预警机制（如设置 ECS CPU 利用率超过 80% 时自动扩容），避免因突发流量导致费用异常激增。
开源技术的隐性成本权衡:
采用 Vue.js、Node.js 等开源技术栈虽规避了商业授权费用，但需投入专项资源进行技术培训（如组织 Vue 组件开发实战培训），并建立组件库复用机制，提升开发效率的同时降低重复开发成本。

11.架构评估与验证

11.1 评估方法

11.1.1 场景设计

性能压力场景：
模拟 500 并发用户 执行 “关键词搜索职位” 操作，通过 JMeter 压测工具验证接口响应时间的 95% 分位值是否≤800ms，重点关注 Elasticsearch 全文检索引擎的吞吐量及 MySQL 数据库的索引命中率。
安全攻击模拟场景：
使用 OWASP ZAP 等渗透测试工具，向系统注入 SQL 注入 payload（如 ' OR 1=1--）、XSS 脚本（如 <script>alert('hack')</script>），验证：后端是否通过参数预编译（如 MyBatis 的 #{} 占位符）防御 SQL 注入；前端是否启用 Content Security Policy（CSP）响应头拦截非法脚本执行。
高可用性故障场景：
人为断开主服务器网络连接，模拟数据中心级故障，验证 负载均衡器（如阿里云 SLB）是否在 5 分钟内完成流量切换 至备用节点，并通过 Prometheus 监控系统确认应用层服务恢复时间≤10 分钟。

11.1.2 评估步骤

阶段 1：架构可视化描述
基于文档中的 系统拓扑图 和 模块交互流程图，通过时序图（Sequence Diagram）动态演示用户登录、简历解析等核心流程，向评估团队（含开发、测试、运维）讲解架构分层设计（前端层、后端服务层、数据存储层）及跨层通信机制（如 RESTful API、消息队列）。

阶段 2：场景具象化开发
收集并优先级排序 30+ 关键场景，例如：异常场景：简历文件损坏导致解析失败时的错误处理逻辑；边界场景：企业用户同时发布 100 个岗位时的接口限流机制；针对高优先级场景编写 场景描述文档，明确输入条件、预期输出及评估通过标准。
阶段 3：量化属性分析
性能维度：通过 JMeter 生成《压测报告》，对比 “设计目标值” 与 “实测值”，如接口平均响应时间、吞吐量（TPS）、错误率；
安全维度：基于 OWASP ZAP 扫描结果生成《安全漏洞清单》，标注漏洞等级（高 / 中 / 低）及修复建议；
可用性维度：通过 Chaos Engineering（混沌工程）工具模拟硬件故障，记录系统恢复时间（MTTR）及服务不可用时长（MTBD），验证是否符合 SLA 承诺（如可用性≥99.9%）。

11.2 与质量目标对齐分析

质量属性	目标值	实现机制	验证结果
性能	页面加载时间≤2s	前端资源压缩、CDN 加速、Redis 缓存热门岗位数据	压测结果显示 95% 请求≤1.8s
安全性	密码加密强度≥BCrypt	采用 BCrypt 加盐哈希算法，禁止明文传输	安全审计未发现弱加密漏洞
可用性	系统可用性≥99.9%	云服务器多可用区部署，搭配负载均衡与自动故障转移	模拟故障时切换时间为 3 分钟
可维护性	模块复用率≥60%	采用模块化设计（如认证模块、简历解析模块），支持独立升级	代码评审显示模块耦合度低

11.3 风险识别与缓解策略

风险类型	具体风险	缓解策略	责任人
技术风险	第三方库漏洞(如 JWT 库)	定期扫描依赖库安全漏洞，及时更新至稳定版本	付翔宇、冯彬
性能风险	高并发下数据库连接池耗尽	配置连接池最大连接数限制，引入队列异步处理非核心请求	孟维汉、翟江浩
安全风险	用户数据泄露	敏感字段加密存储（AES-256），接口访问日志脱敏处理	杨雨泽、付翔宇
团队风险	成员技术栈不熟悉 Vue.js	组织内部技术培训，提供 Vue.js 实战案例与文档	翟江浩、孟维汉

12.CSCI 体系结构设计

12.1 体系结构

12.1.1 程序(模块)划分

系统采用前后端分离架构与分层模块化设计，核心模块划分如下：

前端模块（表现层）

1、用户界面模块：基于 Vue.js + Element Plus 实现响应式 UI，负责用户交互界面展示。

2、路由控制模块（Vue Router）：管理页面路由跳转与导航逻辑。

3、HTTP 通信模块（Axios）：处理前端与后端的数据交互请求。

4、状态管理模块：使用 Vuex/Pinia 实现前端数据的集中管理与共享。

后端模块（业务逻辑层 / 数据访问层）

1、用户管理模块：用户注册、登录、权限认证（JWT）及多角色管理（求职者、企业用户、管理员）。

2、职位管理模块：职位发布、审核、更新、搜索及推荐逻辑（关键词匹配、智能推荐算法）。

3、简历管理模块：简历上传、解析（NLP 技术）、检索、匹配及存储（OSS 文件系统）。

4、招聘流程管理模块：面试安排、录用决策、流程状态跟踪（工作流引擎支持）。

5、数据接口模块：提供 RESTful API 接口，遵循 Swagger 规范。

6、第三方服务模块：对接 Redis（缓存）、MySQL（数据库）、OSS（文件存储）。

数据存储模块

1、关系型数据库（MySQL）：存储结构化数据（用户信息、职位信息、应聘记录等）。

2、缓存数据库（Redis）：缓存高频访问数据（热门职位、用户会话）。

3、对象存储（OSS）：存储非结构化数据（简历文件、企业 LOGO 等）。

12.1.2 程序(模块)层次结构关系

前端层（Vue）

1、调用 RESTful API 接口

2、通过 HTTP 协议与后端交互

后端层

1、表现层（API 接口）：接收前端请求，返回数据响应

2、业务逻辑层：处理核心业务规则（简历解析、职位匹配）

3、数据访问层：操作数据库与第三方服务（MySQL 查询、Redis 缓存读写）

数据层

1、MySQL：存储事务性数据，支持复杂查询与事务管理

2、Redis：提供缓存与会话管理，提升响应速度

3、OSS：存储静态文件，支持高并发访问与弹性扩展

12.2 全局数据结构说明

12.2.1 常量

常量名称	类型	值 / 描述
MAX_USERNAME_LENGTH	整数	20（用户名最大长度）
MIN_PASSWORD_LENGTH	整数	6（密码最小长度，需含数字与字母）
JWT_EXPIRE_TIME	字符串	"2h"（JWT 令牌过期时间）
SEARCH_KEYWORD_MAX_LEN	整数	50（职位搜索关键词最大长度）
CV_PARSE_TIMEOUT	整数	30000（毫秒，简历解析超时时间）
API_VERSION	字符串	"v1"（接口版本号）

12.2.2 变量和数据结构

1. 用户实体（User）

```
interface User {
  userId: string; // 用户 ID（UUID 生成）
  username: string; // 用户名（唯一标识）
  password: string; // 加密密码（BCrypt 哈希）
  role: "job_seeker" | "company" | "admin"; // 角色类型
  contactInfo: {
    phone: string;
    email: string;
  }; // 联系方式
  createTime: Date; // 注册时间}
```

2. 职位实体（Job）

```
interface Job {
  jobId: string; // 职位 ID
  companyId: string; // 企业 ID（外键关联）
  title: string; // 职位名称
  salaryRange: {
    min: number;
```

```
    max: number;
  }; // 薪资范围
  location: string; // 工作地点
  description: string; // 职位描述
  requirements: string[]; // 任职要求（关键词数组）
  status: "active" | "closed"; // 职位状态
  publishTime: Date; // 发布时间}
```

3. 简历实体（Resume）

```
interface Resume {
  resumeId: string; // 简历 ID
  userId: string; // 求职者 ID（外键关联）
  fileName: string; // 文件名
  filePath: string; // OSS 存储路径
  parsedData: {
    name: string; // 姓名
    education: string[]; // 教育经历
    experience: string[]; // 工作经历
    skills: string[]; // 技能标签
  }; // 解析后的结构化数据
  uploadTime: Date; // 上传时间}
```

4. 应聘记录实体（Application）

```
interface Application {
  applicationId: string; // 应聘记录 ID
  userId: string; // 求职者 ID
  jobId: string; // 职位 ID
  status: "applied" | "screening" | "interviewing" | "hired" | "rejected"; // 应聘状态
  applyTime: Date; // 申请时间}
```

12.3CSCI 部件

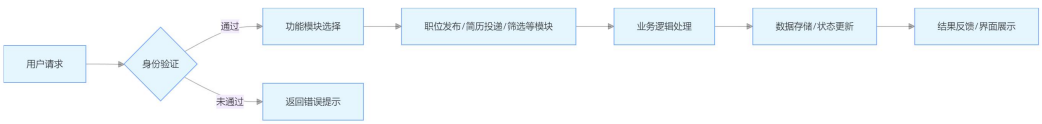
CSCI 部件名称	描述	技术实现	依赖关系
前端应用	用户交互界面与路由逻辑	Vue.js + Vue Router + Element Plus	后端 API 接口、OSS 静态资源

CSCI 部件名称	描述	技术实现	依赖关系
后端服务	核心业务逻辑与 API 接口实现	Spring Boot + Node.js + Express	MySQL、Redis、OSS、消息队列
用户认证服务	身份验证、JWT 生成与校验	Spring Security + JWT Token	无
简历解析服务	简历文件解析与结构化数据提取	NLP 库（如 spaCy）+ 自定义解析算法	OSS（读取简历文件）、Redis（缓存）
职位推荐服务	基于关键词匹配的职位推荐算法	TF-IDF 算法 + 协同过滤算法	职位数据（MySQL）、用户行为日志
数据存储服务	数据库与缓存管理	MySQL 8.0 + Redis 6.x + 阿里云 OSS	无
消息队列服务	异步通信与事件驱动（如面试通知）	RabbitMQ/Kafka	后端服务、第三方通知服务
监控与日志服务	系统性能监控、日志收集与分析	Prometheus + Grafana + ELK Stack	各业务服务组件

12.4 执行概念

12.4.1 执行控制流图示

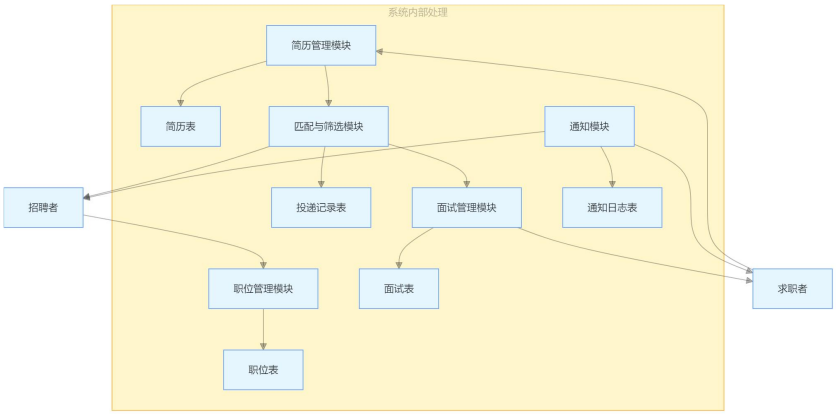
执行控制流图展示系统中各个功能模块的调用关系和执行顺序。主要模块包括用户登录、职位发布、简历投递、简历筛选、面试安排和录用通知等。控制流从用户请求开始，经过身份验证后导向不同功能模块，各模块按业务逻辑顺序执行，并通过返回值或状态码控制流程转向。



12.4.2 数据流图示

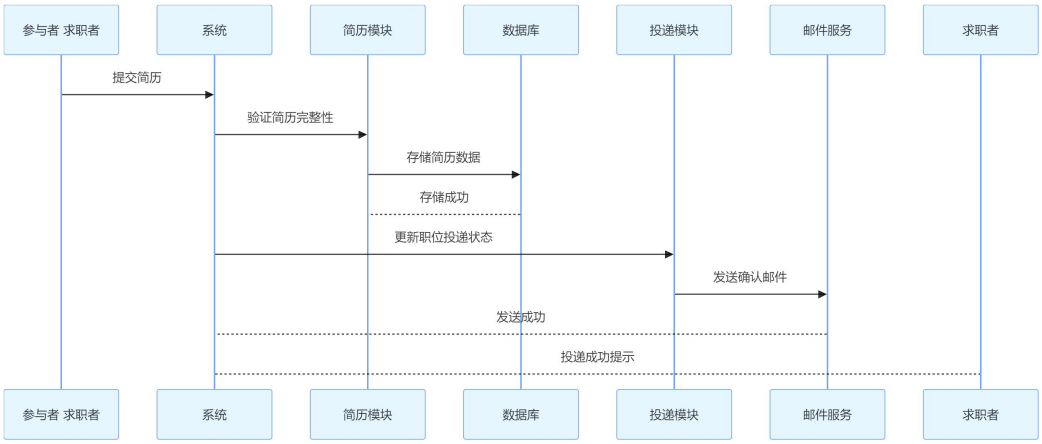
数据流图描述系统中数据的流动和处理过程。外部实体（如求职者、招聘者）与系统之间存在数据输入输出，系统内部数据在数据存储（数据库）和处理过程（如简历分析、匹配算法）之间流动。关键数据流包括职位信息、简历内容、筛选结果、面试安排等。
顶层数据流：

分层数据流：



12.4.3 动态控制序列图示

动态控制序列图展示特定场景下系统组件之间的交互时序。例如，求职者投递简历场景：求职者提交简历后，系统验证简历完整性，存储简历数据，更新职位投递状态，发送确认邮件，整个过程按时间顺序展示各组件的消息传递和状态变化。



12.5 接口设计

12.5.1 数据库表与访问接口

数据库设计包含以下核心表结构：

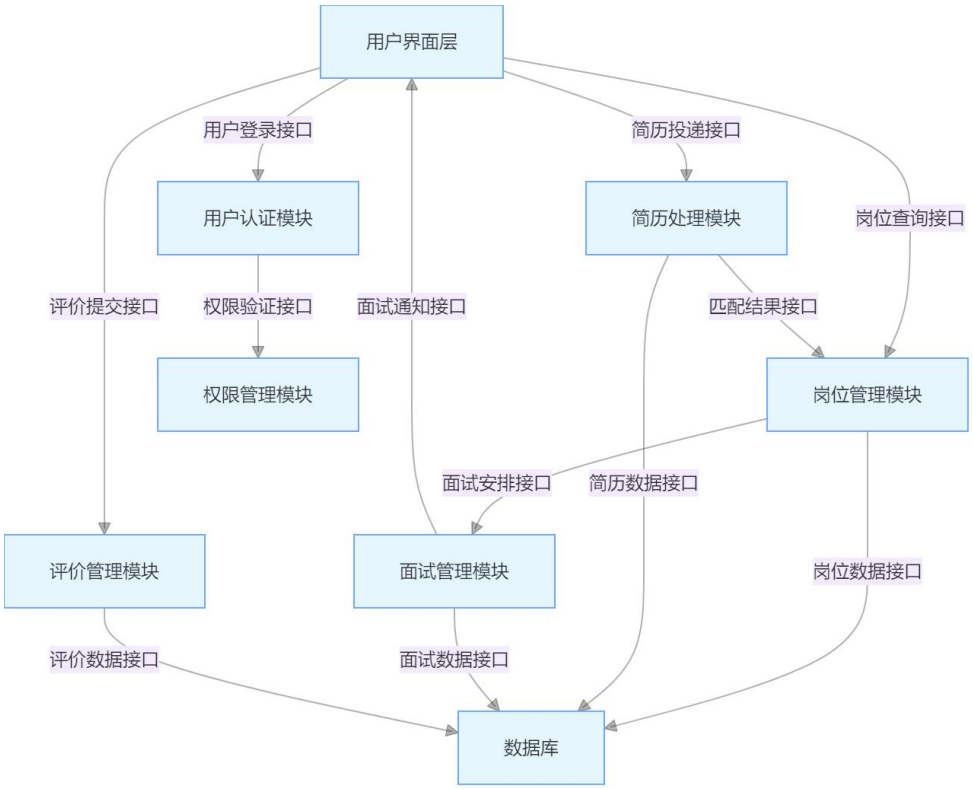
表名	字段名	数据类型	约束	说明
users	user_id	INT	主键、自增	用户唯一标识
	username	VARCHAR(50)	非空、唯一	用户名
	password_hash	VARCHAR(100)	非空	密码哈希值
	user_type	ENUM	非空	取值：job_seeker/recruiter
	contact_info	VARCHAR(100)	非空	联系方式
	create_time	DATETIME	默认	注册时间

表名	字段名	数据类型	约束	说明
jobs	job_id	INT	主键、自增	职位唯一标识
	recruiter_id	INT	外键	关联招聘者 ID
	job_title	VARCHAR(100)	非空	职位名称
	job_description	TEXT	非空	职位描述
	salary_range	VARCHAR(50)	非空	薪资范围
	location	VARCHAR(50)	非空	工作地点
	publish_time	DATETIME	默认	发布时间
	deadline	DATETIME	非空	申请截止时间

表名	字段名	数据类型	约束	说明
Resumes	**resume_id**	**INT**	**主键、自增**	**简历唯一标识**
	job_seeker_id	**INT**	**外键**	**关联求职者 ID**
	personal_info	**TEXT**	**非空**	**个人信息**
	education	**TEXT**	**非空**	**教育背景**
	work_experience	**TEXT**	**非空**	**工作经历**
	skills	**TEXT**	**非空**	**技能列表**
	self_assessment	**TEXT**	**非空**	**自我评价**
	create_time	**DATETIME**	**默认**	**创建时间**

访问接口采用 RESTful 风格设计，主要接口包括：
用户认证接口：登录、注册、注销
职位管理接口：创建职位、更新职位、删除职位、查询职位
简历管理接口：创建简历、更新简历、查询简历
投递管理接口：提交投递、查询投递记录、更新投递状态
面试管理接口：安排面试、更新面试结果、查询面试信息

12.5.2 接口图与接口标识



接口名称	接口功能描述	输入参数	输出参数	调用方	被调用方
用户登录接口	验证用户登录信息，生成登录令牌	用户名、密码	登录状态（成功 / 失败）、登录令牌	用户界面层	用户认证模块
岗位查询接口	根据用户输入条件查询岗位信息	查询条件（如岗位名称、薪资范围、工作地点等）	岗位列表数据	用户界面层	岗位管理模块
简历投递接口	接收用户投递的简历，保存至数据库	简历文件、投递岗位 ID	投递结果（成功 / 失败）	用户界面层	简历处理模块
权限验证接口	验证用户操作权限	操作类型、用户 ID、登录令牌	权限状态（允许 / 禁止）	用户认证模块	权限管理模块

岗位数据接口	实现岗位信息的增删改查操作	岗位操作类型（新增、修改、删除、查询）、岗位数据	操作结果（成功 / 失败）、查询结果	岗位管理模块	数据库
简历数据接口	实现简历信息的存储、读取和更新	简历操作类型（存储、读取、更新）、简历数据	操作结果（成功 / 失败）、读取结果	简历处理模块	数据库
匹配结果接口	将简历匹配结果返回给岗位管理模块	无	匹配后的简历列表及匹配度	简历处理模块	岗位管理模块
面试安排接口	保存面试安排信息至数据库	面试安排数据（时间、地点、面试官、候选人 ID 等）	操作结果（成功 / 失败）	岗位管理模块	面试管理模块
面试通知接口	向用户发送面试通知	面试通知内容、用户 ID	发送结果（成功 / 失败）	面试管理模块	用户界面层
面试数据接口	实现面试信息的存储、读取和更新	面试操作类型（存储、读取、更新）、面试数据	操作结果（成功 / 失败）、读取结果	面试管理模块	数据库
评价提交接口	接收用户提交的评价信息，保存至数据库	评价数据（评价内容、评价对象 ID、评价者 ID 等）	提交结果（成功 / 失败）	用户界面层	评价管理模块
评价数据接口	实现评价信息的存储、读取和更新	评价操作类型（存储、读取、更新）、评价数据	操作结果（成功 / 失败）、读取结果	评价管理模块	数据库

13. CSCI 详细设计

13.1 用户登入模块设计

13.1.1 模块功能概述

用户登入模块是人才招募系统的核心入口，承担着用户身份信息收集、登录流程引导及基础输入校验的功能。该模块需支持个人用户与企业用户两类角色的登录操作，覆盖账号密码登录的全生命周期管理，包括界面展示、输入交互、格式校验及请求发起。设计上需兼顾易用性与安全性，确保不同角色用户能快速完成登录，同时防止非法用户通过界面漏洞获取敏感信息。

13.1.2 角色与场景分析

1. 个人用户场景

个人用户以手机号或邮箱作为账号，输入密码后登录系统，主要用于管理简历、查询岗位、投递申请等操作。需考虑用户可能忘记密码、账号未注册等异常情况，界面需提供“忘记密码”链接及“立即注册”引导。

2. 企业用户场景

企业用户使用企业专属账号（如企业 ID 或认证邮箱）登录，用于发布职位、管理招聘流程、筛选人才等。企业账号需先通过管理员审核，因此登录模块需校验账号状态（如“未认证”“已禁用”），并在登录失败时提示用户进行认证或联系管理员。

13.1.3 界面交互设计

13.1.3.1 页面布局

整体结构：采用响应式设计，适配 PC 端与移动端。PC 端登录页面居中显示登录表单，背景可采用浅色系搭配企业 LOGO；移动端采用全屏布局，表单占据屏幕中上部，底部设置注册找回密码链接。

表单元素：

用户类型选择：通过单选按钮或下拉菜单区分“个人用户”与“企业用户”，默认选中“个人用户”。

账号输入框：提示文字为“手机号/邮箱”（个人用户）或“企业账号/认证邮箱”（企业账号），支持键盘输入及粘贴，移动端自动唤起数字或邮箱键盘。

密码输入框：提示文字为“密码”，默认隐藏输入内容，右侧设置“显示密码”按钮（点击后切换为明文），符合无障碍设计规范。

登录按钮：按钮文字为“立即登录”，点击前为灰色不可用状态，需满足所有输入项格式正确后变为可点击状态（如蓝色高亮）。

辅助链接：在按钮下方排列“忘记密码？”“还没有账号？立即注册”，点击后跳转至对应页面。

13.1.3.2 交互流程细节

焦点管理：用户进入页面时，自动聚焦于“账号输入框”；按下 Tab 键可依次切换焦点至密码框、用户类型选择、登录按钮。

输入校验：

实时校验：用户输入时触发格式验证，如手机号需符合“1[3-9]\d{9}”正则表达式，邮箱需包含“@”符号，企业账号需匹配数据库中已存在的企业标识。

提交前校验：点击登录按钮时，再次校验所有字段：若账号为空，提示“请输入账号”；若密码长度小于 6 位，提示“密码需至少 6 位”；若用户类型未选择，默认按个人用户处理（需二次确认或自动识别）。

加载状态：点击登录按钮后，按钮文字变为“登录中...”，并禁用重复点击，防止多次提交请求。

13.1.4 数据传输设计

前端请求：使用 Axios 库发送 POST 请求至后端登录接口，请求地址为“/api/login”，请求体包含以下字段：

```
{
  "userType": "individual/company", // 用户类型
  "identifier": "user@example.com", // 账号（手机号/邮箱/企业账号）
  "password": "encoded_password", // 前端 Base64 编码后的密码（非加密，仅防明文传输）
  "clientType": "web/mobile" // 客户端类型，用于日志记录
}
```

安全增强：前端不对密码进行加密（避免加密算法泄露），仅进行 Base64 编码，真正的密码校验由后端完成；使用 HTTPS 协议确保数据传输过程中不被中间人窃取。

13.1.5 异常场景处理

1. 账号未注册：后端验证账号不存在时，返回错误码“USER_NOT_EXIST”，前端提示“该账号未注册，请先注册”，并高亮显示“立即注册”链接。
2. 账号被锁定：若用户连续 5 次输入错误密码，后端锁定账号 30 分钟，返回“账号已锁定，请 30 分钟后重试”，并触发短信通知用户（需用户已绑定手机）。
3. 企业账号未认证：企业用户登录时，若账号状态为“未认证”，返回“请先完成企业认证”，并引导至认证页面。

13.2 验证模块设计

13.2.1 模块功能定位

验证模块是登录流程的核心逻辑层，负责验证用户身份的合法性，涵盖账号存在性校验、密码正确性验证、权限匹配及多因素认证（MFA）等功能。该模块需与数据库、缓存系统及第三方认证服务（如短信验证码）交互，确保只有合法用户才能获得访问权限，并防范暴力破解、越权访问等安全威胁。

13.2.2 核心验证流程

13.2.2.1 账号存在性校验

1. 数据查询：根据用户类型（individual/company）选择对应的数据库表：

个人用户：查询 job_seekers 表，条件为 phone_number = :identifier OR email = :identifier 。

企业用户：查询 recruiting_companies 表，条件为 company_account = :identifier OR verified_email = :identifier 。

2. 异常处理：若查询结果为空，直接返回“账号或密码错误”（不明确提示“账号不存在”，避免泄露用户注册信息）。

13.2.2.2 密码正确性验证

1. 哈希比对：从数据库中获取用户存储的密码哈希值（使用 BCrypt 算法生成），将前端传入的密码进行 BCrypt 哈希处理，与数据库值比对。
2. BCrypt 参数设置：工作因子（cost factor）设为 12，确保哈希计算耗时合理（约 50-100ms），抵御彩虹表攻击。
3. 错误处理：若比对失败，记录错误日志（包含 IP 地址、尝试时间、用户类型），并返回通用错误提示；若连续失败次数达到阈值（如 5 次），触发账号锁定机制（见 13.2.4 节）。

13.2.2.3 权限匹配

角色识别：从用户表中获取 role 字段（如 individual_user 、company_admin 、system_admin ），验证该角色是否具备登录权限（如禁用状态的用户 is_active = 0 则拒绝登录）。

企业认证状态：对于企业用户，需额外校验 is_verified 字段（是否通过管理员审核），未通过认证的企业账号需引导至认证流程。

13.2.2.4 多因素认证（MFA）

1. 触发条件：

用户首次登录新设备（通过 IP 地址、User-Agent 等信息识别）。
系统检测到异常登录行为（如异地登录、高频次登录请求）。
用户主动开启 MFA 功能（在账户设置中绑定手机或邮箱）。

2. 验证方式：

TOTP 动态验证码：用户使用 Google Authenticator 等应用生成 6 位动态码，后端通过时间同步算法验证（误差允许±30 秒）。

短信验证码：调用阿里云短信服务发送 6 位数字验证码，用户输入后与 Redis 中存储的验证码（有效期 5 分钟）比对。

邮箱验证码：发送包含链接的验证邮件，用户点击链接后完成验证（适用于无手机用户）。

13.2.3 数据库与缓存设计

1. 用户表结构：

job_seekers 表字段： id （主键）、 phone_number 、 email 、 password_hash （BCrypt 哈希值）、 role 、 is_active 、 mfa_secret （TOTP 密钥）。

2. Redis 缓存：

存储临时验证码：如短信验证码 SMS_CODE_${phone_number} ，有效期 300 秒，值为随机 6 位数字。存储账号锁定状态： LOCKED_ACCOUNT_${identifier} ，值为锁定截止时间戳，

有效期 1800 秒（30 分钟）。

13.2.4 安全防护机制

13.2.4.1 暴力破解防御

1. IP 限流：使用 Nginx 的 limit_req 模块，对同一 IP 地址的登录请求进行速率限制，如“每分钟最多允许 20 次请求”，超出后返回 429 Too Many Requests。

2. 账号锁定：连续 5 次密码错误后，锁定账号并记录 LOCKED_ACCOUNT_${identifier} 到 Redis，同时发送告警邮件至管理员。解锁方式包括：

自动解锁：达到锁定时间后自动清除缓存。

手动解锁：系统管理员在后台管理界面操作解锁。

13.2.4.2 密码策略

1. 复杂度要求：强制用户密码包含至少 6 位字符，且包含数字、字母（大小写混合）、特殊符号（如!@#$%）中的至少两种。
2. 定期更换：要求用户每 90 天更换一次密码，到期前 15 天发送邮件提醒。
3. 密码历史限制：禁止用户重复使用前 5 次用过的密码，避免旧密码泄露导致安全风险。

13.3 请求处理模块设计

13.3.1 模块功能定义

请求处理模块是连接前端、验证模块与后端服务的桥梁，负责协调登录流程中的数据流转，完成身份验证后的授权处理、会话状态维护及响应生成。该模块需实现 JWT 令牌生成、Redis 会话存储、权限信息传递等功能，确保用户登录后能安全、高效地访问系统资源。

13.3.2 核心处理流程

13.3.2.1 请求接收与解析

1. 接口定义：后端提供 RESTful 接口 POST /api/login，使用 Spring Boot 或 Express 框架接收请求，通过注解或中间件解析请求体中的 JSON 数据。

2. 参数校验：对 userType、identifier、password 等必填字段进行非空校验，若缺失则返回 400 Bad Request，并提示“缺少必要参数”。

13.3.2.2 调用验证模块

1. 依赖注入：通过 Spring 依赖注入或 Express 中间件将验证模块实例注入请求处理逻辑，调用 verifyLogin(userType, identifier, password) 方法。

2. 结果处理：

若验证成功，获取用户 ID、角色、权限等信息。

若验证失败，根据错误类型返回对应的 HTTP 状态码（如 401 Unauthorized、403 Forbidden）。

13.3.2.3 JWT 令牌生成

1. 令牌结构：采用 JSON Web Token（JWT）实现无状态认证，令牌包含三部分：

Header：指定签名算法（如 HS256）和令牌类型，示例：

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload：存储用户身份信息及元数据，示例：

```
{
  "sub": "user_123",          // 用户唯一标识（个人用户 ID 或企业 ID）
  "role": "individual_user", // 用户角色
  "permissions": ["view_jobs", "apply_job"], // 权限列表
  "exp": 1750000000,          // 过期时间戳（当前时间+2 小时）
  "iat": 1749993600,          // 签发时间戳
  "iss": "talent-system"      // 签发者
}
```

Signature：通过 HS256 算法对 Header 和 Payload 进行签名，密钥存储在服务器端环境变量中，确保不可泄露。

2. 生成逻辑：使用 jsonwebtoken 库（Java）或 jsonwebtoken.io （Node.js）生成令牌，设置过期时间为 2 小时，通过 Refresh Token 机制实现令牌续期。

13.3.2.4 会话状态存储

1. Redis 存储：将 JWT 令牌存入 Redis，以 JWT_TOKEN_${user_id} 为键，值为令牌字符串，过期时间与 JWT 有效期一致（2 小时）。存储目的：支持服务器端主动失效令牌（如用户注销时删除 Redis 记录）。防止 JWT 令牌被篡改（需结合 HTTPS 使用，Redis 存储为额外安全层）。

2. 无状态模式：若系统严格遵循 JWT 无状态设计，可省略 Redis 存储，直接通过 JWTPayload 验证用户权限，但需注意令牌无法主动失效的问题（需依赖令牌过期时间）。

13.3.2.5 响应生成

1. 成功响应：
- HTTP 状态码：200 OK。
- 响应体：

```
{
   "status": "success",
   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...", // JWT 令牌
   "userInfo": {
      "userId": "user_123",
      "username": "张三",
      "role": "individual_user",
      "companyName": "XX 科技有限公司" // 企业用户专属字段
   },
   "expiresIn": 7200 // 令牌剩余有效期（秒）
}
```

2. 失败响应：
- 账号或密码错误：401 Unauthorized，响应体 {"error": "invalid_credentials"} 。
- 权限不足：403 Forbidden，响应体 {"error": "permission_denied"} 。
- 账号锁定：423 Locked，响应体 {"error": "account_locked", "retryAfter": 1800}（提示剩余锁定时间）。

13.3.3 安全与性能优化

13.3.3.1 JWT 安全加固

1. HTTPS 强制使用：在 Nginx 或 Spring Boot 中配置 HSTS（HTTP Strict Transport Security），强制客户端通过 HTTPS 访问，防止令牌在 HTTP 传输中被截取。

2. Refresh Token 机制：

生成 Refresh Token（长有效期，如 7 天），存储于 HttpOnly Cookie 中，避免 XSS 攻击。

当 JWT 过期时，前端调用 /api/refresh-token 接口，传入 Refresh Token 获取新的 JWT，无需用户重新登录。

3. 令牌黑名单：用户注销或账号被盗时，将 JWT 加入 Redis 黑名单（键为 BLACKLIST_TOKEN_${token}），有效期与原令牌一致，后续请求检测到黑名单令牌时拒绝访问。

13.3.3.2 性能优化

1. 数据库连接池：使用 HikariCP（Java）或 Node.js 的 mysql2/pool 创建数据库连接池，最大连接数设为 50，最小空闲连接 5，避免高并发下连接耗尽。

2. Redis 集群：在生产环境中部署 Redis 主从集群，通过 Sentinel 实现自动故障转移，提升缓存服务的可用性和读写性能。

3. 异步处理：将非核心操作（如登录日志记录）通过消息队列（如 RabbitMQ）异步处理，避免阻塞登录主流程。登录成功后，发送消息至“登录日志队列”，由消费者服务异步写入数据库。

13.3.4 模块间协作流程

1. 用户在前端输入账号密码，点击登录。
2. 前端发送登录请求至后端请求处理模块。
3. 请求处理模块解析参数，调用验证模块进行身份验证。
4. 验证模块查询数据库，校验账号密码、权限及 MFA（如有）。
5. 验证通过后，请求处理模块生成 JWT 令牌，并可选存入 Redis。
6. 请求处理模块返回包含令牌的响应给前端。
7. 前端存储令牌（建议存入 LocalStorage 或内存中，避免 Cookie 泄露风险），并跳转至首页或目标页面。
8. 后续接口请求中，前端在请求头中携带令牌：`Authorization: Bearer <JWT_TOKEN>`。
9. 后端全局拦截器验证令牌有效性（解析 JWT 并校验签名、过期时间、权限），通过后允许访问资源。

13.3.5 异常场景处理

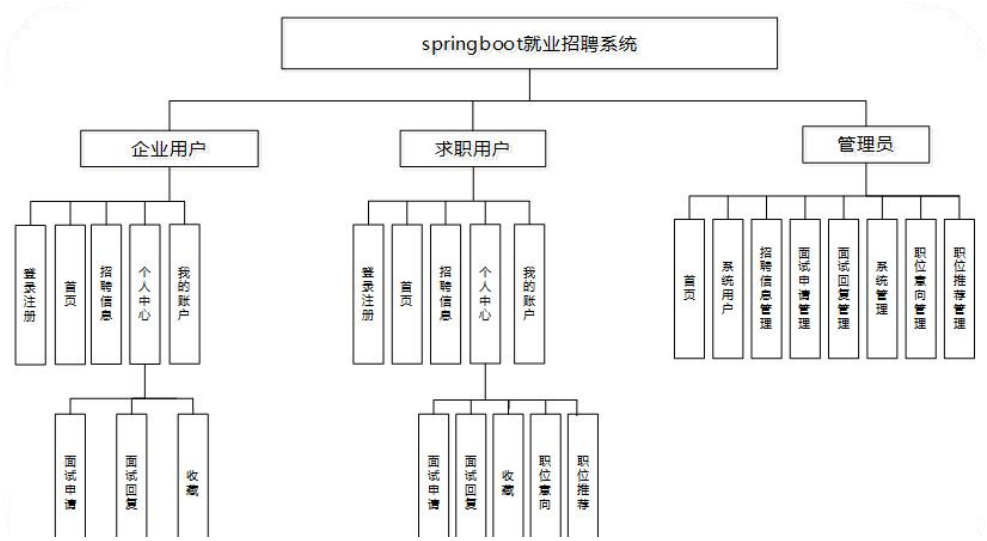
1. JWT 解析失败：若令牌被篡改或格式错误，返回 401 Unauthorized，提示“无效访问令牌”。
2. 令牌过期：返回 401 Unauthorized，提示“令牌已过期，请重新登录”，并触发前端跳转至登录页面或自动刷新令牌（若存在 Refresh Token）。
3. 权限不足：用户尝试访问超出自身角色权限的资源（如普通用户访问管理员接口），返回 403 Forbidden，提示“你没有权限执行此操作”。

附录

A.缩略词表

缩略词	全称	说明
ADR	Architectural Decision Record	架构决策记录
ATAM	Architecture Tradeoff Analysis Method	架构权衡分析方法
BCrypt	Blowfish Cryptographic Function	密码哈希算法
CDN	Content Delivery Network	内容分发网络
JWT	JSON Web Token	身份验证令牌
OSS	Object Storage Service	对象存储服务（如阿里云 OSS）
RESTful	Representational State Transfer	一种接口设计规范

B.UML 图汇总



C.更改记录

版本号	日期	修改内容	修改人
V1.0	2025-05-10	初始版本发布，完成基础架构设计	付翔宇
V1.1	2025-05-18	优化简历解析算法，提升识别准确率至 95%	孟维汉、杨雨泽
V1.2	2025-05-30	增加 Redis 集群支持，提升并发处理能力	冯彬、翟江浩

D.补充材料

样例接口文档（部分节选）
请求：

```
GET /api/positions?keyword=Java HTTP/1.1
Host: api.talent-system.com
Authorization: Bearer <JWT_TOKEN>
```

响应：

```
{
  "code": 200,
  "data": [
    {
      "positionId": "POS-20253421",
      "companyName": "山大测试 company",
      "salary": 15000,
      "matchScore": 85   // 基于 TF-IDF 算法计算的匹配得分
    }
  ]}
```

系统在 1000 并发用户压力下，搜索接口响应时间、吞吐量及资源利用率均符合预期目标，具备处理峰值流量的能力。建议定期进行负载测试，监控长期运行下的性能衰减情况。
安全测试报告
测试目的：识别系统潜在安全漏洞，验证身份认证机制的健壮性。
测试工具：OWASP ZAP 2.14.0（开源网络安全扫描工具）
测试范围：用户登录接口（POST /api/login）的认证逻辑。

漏洞发现：
漏洞类型：暴力破解风险（Brute Force Attack）
漏洞描述：登录接口未限制同一账号的密码错误尝试次数，攻击者可通过自动化工具（如 Hydra）循环尝试密码组合，存在账户锁定风险。
修复方案：
添加验证码机制：当同一账号连续 5 次登录失败时，强制要求输入图形验证码（如 Google reCAPTCHA）；验证码需具备时效性（如 2 分钟内有效），且每次验证后随机刷新。
接口限流策略：通过 Nginx 或 Spring Boot Actuator 对登录接口设置速率限制，如“同一 IP 地址每分钟最多发起 20 次登录请求”。
复测结果：验证流程：使用 OWASP ZAP 模拟 100 次 /s 的登录请求，携带随机错误密码；结果显示：第 5 次错误尝试后，接口返回验证码要求，攻击请求被有效拦截，漏洞已修复。