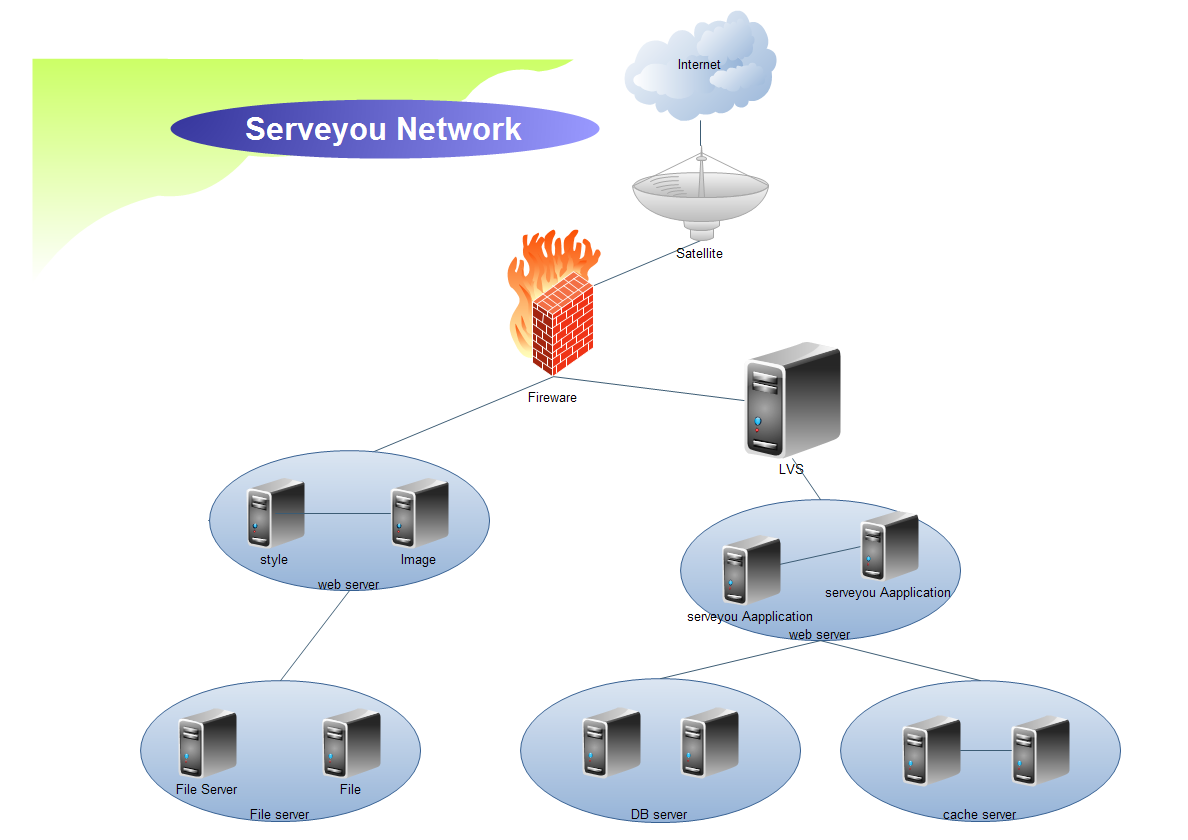
**概要设计说明书**

**用户个性化设计**

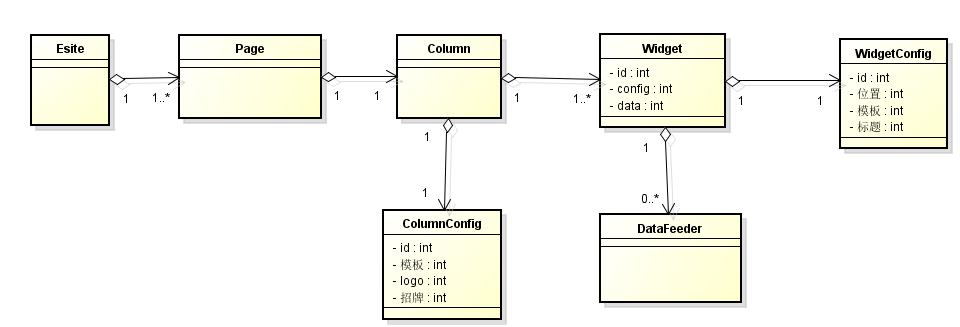
1. **项目说明**
2. **主要目的**
3. 用户个性化页面保存。
4. 提高系统的可用性和稳定性。
5. 为将来业务发展打好坚实的架构基础。
6. **网络拓扑**

****

1. **业务/数据流程图**
2. **领域模型**
3. **用户个性化布局**

****

**结构模型**

****

**说明：**

1、整体采用两栏式结构，总体分为：顶部、侧边栏、主区域、底部，总共四大块。

2、采用widget（板块）为最小单位，规划布局，即每个页面就是许多widget以不同方式的组合，每个大板块可包含一个或者多个widget，widget的布局原则上可任意调整。

3、页面布局（暂且不考虑）和样式存储于数据库中，考虑采用json格式数据存储。

4、每个栏目页的数据流程均大致相同，详见下面的页面渲染数据流程图。

5、架构需要考虑到以后的编辑模式扩展。

6、通过jQuery.load()，数据加载完毕后，通过调用miui中api进行个性化的改变，其中注入mergeColumns、sortBy 、moveRow 等datagrid的个性化设置中进行改变，针对于全局性的css样式更改，是通过UI工程师准备模板，全方面的改变

1. **页面渲染流程图**

****

**说明：**

1. SpringControl取得当前请求的栏目页设置，读取配置所要渲染的页面和取数dataFeeder，进行最终页面渲染。
2. DataFeeder：每个widget页面都或多或少需要从数据源中取得数据进行渲染，datafeeder就充当了这个数据源的角色。一个widget可能会包含一个或多个datafeeder。由于一个栏目页可能包含重复的datafeeder，所以要使用DataCollector进行统一管理，并将取得的数据放入DataContext，如果DataContext中已经有相同的datafeeder了，则不用重新取数
3. 由于一个栏目中，可能会出现两块内容一样的版块，如左侧与主内容区域，在DataFeeder需要考虑屏蔽重复取数问题。
4. 展示层面，取数据库的数据都要加cache处理。

**业务数据模型**

****

****

1. **数据库表结构设计**

****

1. **关键技术要点**
2. **应用划分**
3. **图片服务**

Nginx模块完成，对于静态服务器的加载

1. **缓存**

采用分布式缓存memcached，pv较高，页面中所有展示信息都要进行cache缓存。客户填写率较低的功能点，需要对null值也进行cache（缓存一个Object空对象），防止缓存没有命中而频繁访问数据库

1. **其他**