Multi-Tenant Isolation Requirements: CampusWatch

Introduction

Multi-tenant architecture is the use of a single logical software application or service to serve multiple customers. In this model, each customer is referred to as a tenant. In the project “CampusWatch”, the tenants are Colleges/Schools/Universities. Isolation of data, resources, access, and security is one of the major and critical tasks while distributing a system to multiple customers. The server side could be using the same hardware, but we need to abstract everything so that the customer doesn’t feel that way, kind of like using VMs in a PC.
In “CampusWatch”, data and security are critical factors. The project focuses on security and monitoring. Similarly, it requires resource isolation and allocation to handle a continuous stream of live video and the anomaly detection models.

Methodologies

I researched different articles online, like (Multi-Tenant Architecture: How It Works, Pros, and Cons)[1], (Tenant isolation in multi-tenant systems: What you need to know)[2], etc. Here, I got a better understanding of strategies for a Multi-Tenant Isolation system. I maintained the following 4 strategies:

1. Data Isolation

Each tenant's data must be logically separated to ensure privacy and security, as if each has its own "private, locked apartment" within the shared system. This aligns with the principle of **"Logical Data Separation"**.

[1] https://frontegg.com/guides/multi-tenant-architecture
[2] https://workos.com/blog/tenant-isolation-in-multi-tenant-systems

- **Logical Separation**: Each tenant's data must be **logically separated at the database level**. This can be achieved by either using a dedicated database per tenant (a **"silo"** model) or a single database with row-level security (RLS) or dedicated schemas per tenant. RLS ensures that each query automatically filters data to only show what belongs to the user's tenant, as detailed in many database security guides.
- **Data Encryption**: Where feasible, sensitive tenant data should be encrypted with unique keys for each tenant. This adds an extra layer of security, following the best practices for data at rest.
- **Backup and Restoration**: Backup procedures must be designed to support tenant-specific restoration. In the event of data loss or corruption, the platform must be able to restore a single university's data without affecting others.

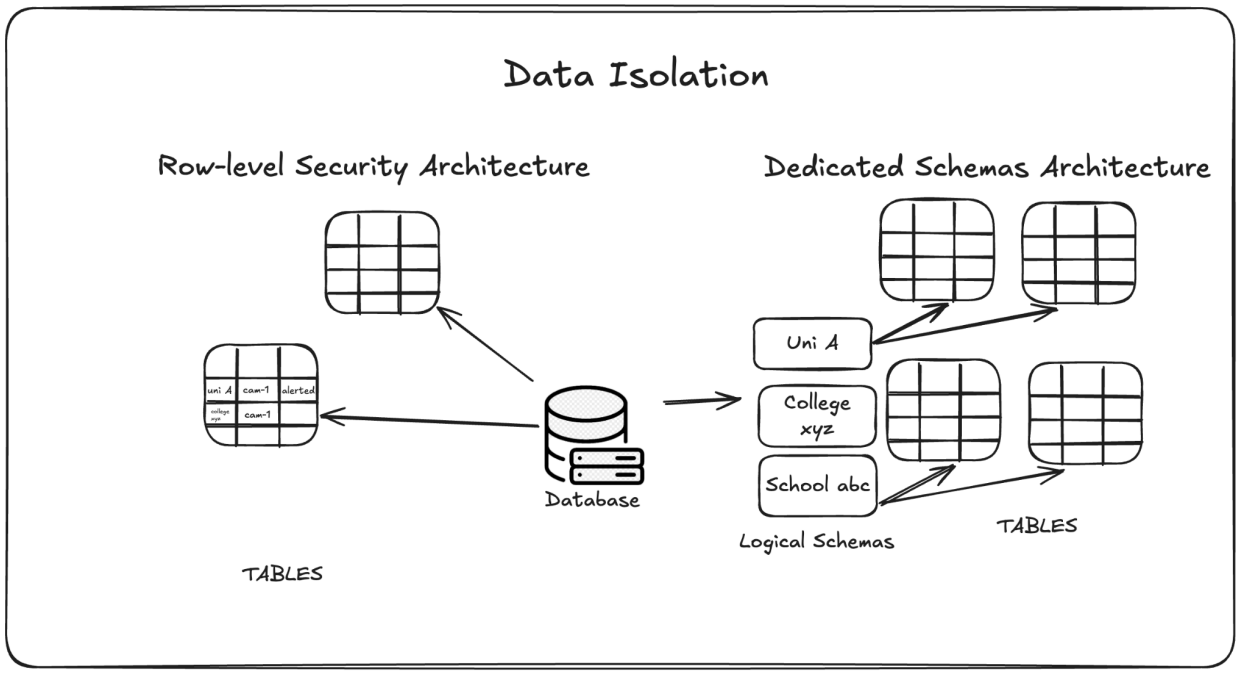

Figure: Data Logical Separation strategies

2. Resource Isolation

The platform must apply per-tenant resource controls to ensure fair performance and prevent one tenant's workload from impacting others. This is a core component of **"Shared Resource Isolation"**.

- **Per-Tenant Quotas**: Apply per-tenant quotas for **compute, storage, and network bandwidth**. This prevents a large campus with hundreds of cameras from consuming all available processing power, thus ensuring that smaller institutions receive the resources they need. Resource isolation can be accomplished using **containers** to keep resources separate.
- **Performance Stability**: The architecture must be designed to ensure one tenant's heavy workload cannot degrade another tenant's performance. This is typically managed through microservices, containerization, and a queuing system that handles resource-intensive tasks, like video analysis, independently for each tenant.
- **Independent Scaling**: The system should support resource scaling independently for each tenant. If one university experiences a sudden spike in campus activity, its resources can scale up automatically without requiring a full-system scale-out.

3. Access & Identity Management

Robust access controls are critical for maintaining the integrity of a multi-tenant system. This involves creating a **"Tenant-Aware Identity Management"** system that includes authentication and authorization.

- **Tenant Identity Validation**: All API requests must validate tenant identity at the service entry point. This is the first line of defense, ensuring that any request, whether for video feeds or incident logs, is authenticated and authorized for the correct tenant.
- **Tenant-Scoped RBAC**: **Role-Based Access Control (RBAC) must be tenant-scoped**. A security guard's permissions (e.g., view live camera feeds) are only valid for their specific university and do not grant them any access to another tenant's data. This is a fundamental concept in multi-tenant authorization models.
- **Audit Logs**: Audit logs must clearly separate tenant actions. Every action (e.g., a user viewing a camera, an admin changing a setting) must be logged with the corresponding tenant ID for traceability, compliance, and security forensics.

4. Compliance & Security

The platform must actively meet legal and security requirements for all its users. This is an essential aspect of a multi-tenant application's **"Governance and Security"**.

- **Regulatory Compliance**: The system must be designed to support compliance requirements like **GDPR** (General Data Protection Regulation), particularly in how it handles student and staff data.
- **Data Residency**: The system must provide data residency controls where required. This allows universities to specify that their data must be stored and processed within a specific geographic location to meet local regulations.
- **Security Verification**: The system and its isolation model should undergo periodic security testing, such as penetration testing, to verify that isolation between tenants is robust and effective against potential threats.

Tools: Google Docs, Excalidraw.

Research: Multi-tenancy architecture.