MULTI-TENANT SAAS BASICS

TechRacine Solutions Academy

https://techracine.com

https://academy.techracine.com

Study Guide

Table of Contents

Introduction to SaaS

SaaS stands for Software as a Service.

SaaS applications are hosted online and accessed through web browsers.

Users subscribe to software instead of installing it locally.

Modern SaaS products are widely used in CRM systems, accounting software, HR platforms, and AI applications.

Examples include Google Workspace, Microsoft 365, Salesforce, and QuickBooks Online.

What is Multi-Tenant Architecture

Multi-tenant architecture allows multiple customers to use the same software application.

Each customer is called a tenant.

Although tenants share the same application, their data remains isolated and secure.

Multi-tenant systems reduce infrastructure and maintenance costs.

Single Tenant vs Multi-Tenant

In single-tenant systems, each customer has a separate application instance and database.

In multi-tenant systems, multiple customers share the same application instance.

Multi-tenant systems are more scalable and cost-effective.

Single-tenant systems provide stronger isolation but require higher infrastructure cost.

Benefits of Multi-Tenant Applications

1. Lower hosting and infrastructure costs.

2. Easier software updates.

3. Centralized maintenance.

4. Better scalability.

5. Faster onboarding of customers.

6. Efficient resource usage.

Real-World SaaS Examples

Many modern applications use multi-tenant architecture.

Examples include Salesforce, Shopify, Slack, QuickBooks Online, and HubSpot.

These applications support thousands of customers using shared infrastructure.

Tenant Identification

Applications must identify which tenant a user belongs to.

Tenant identification can be based on domain, subdomain, token, or tenant ID.

Proper tenant identification is critical for security.

```
Example:

academy1.app.com
academy2.app.com
```

Database Design for Multi-Tenant Systems

Database design is very important in multi-tenant systems.

Most applications include tenant_id columns in tables.

Data must always be filtered using tenant_id.

Improper filtering can expose customer data.

```
students table:

student_id
tenant_id
student_name
```

Shared Database vs Separate Database

Shared database architecture stores all tenant data in the same database.

Separate database architecture creates individual databases for each tenant.

Shared databases reduce cost.

Separate databases improve isolation and customization.

Authentication and Authorization

Authentication verifies user identity.

Authorization controls what users can access.

Multi-tenant systems must ensure users access only their tenant data.

Tenant-Based Security

Tenant isolation is one of the most important security requirements.

Applications must validate tenant ownership in every request.

API security and database filtering are critical.

Multi-Tenant Application Flow

Users log in to the application.

The application identifies the tenant.

Tenant-specific data is loaded.

All operations are restricted to the tenant context.

Subscription and Billing Concepts

Most SaaS products use subscription-based billing.

Customers may subscribe monthly or yearly.

Applications may support multiple pricing plans and feature restrictions.

Scalability and Performance

Multi-tenant applications must support many customers simultaneously.

Caching, database optimization, and load balancing improve performance.

Scalability is a major requirement for SaaS products.

Logging and Monitoring

Applications should maintain tenant-specific logs.

Monitoring helps identify performance issues and security problems.

Centralized logging improves debugging and maintenance.

Deployment Concepts

SaaS applications are usually hosted in cloud environments.

CI/CD pipelines help automate deployments.

Docker and Kubernetes are commonly used in SaaS platforms.

Challenges in Multi-Tenant Systems

Data isolation complexity.

Performance bottlenecks.

Tenant customization challenges.

Security risks.

Scalability issues.

Database management complexity.

Best Practices

Always use tenant_id filtering.

Implement strong authentication.

Use scalable database design.

Separate business logic properly.

Maintain proper logging and auditing.

Design APIs securely.

Summary

Multi-tenant SaaS architecture is widely used in modern software products.

It allows multiple customers to share the same application securely.

Proper tenant isolation, database design, and scalability are critical.

Understanding multi-tenant concepts is important for SaaS product development.