### Turning a single consumer web-based Platforms into a SaaS

A web-based gaming platform (=gPlatform) is currently providing it’s services to one single gaming site (=gSite).

The services provided by gPlatform to gSite include hosting web-games and backoffice for managing players that sign-up and play on gSite.

We want to make gPlatform into a SaaS that can be sold to other gaming sites as subscription-based service.

Each new gaming company operating a gaming site, will have it’s own dedicated domain - for example:

Company A will have a domain cool-games.com

Company B will have a domain luck-games.co.uk

etc

Currently at gPlatform, users are identified by using email as a unique key.

#### Give a short, clear explanation for every question below:

1. **How can we design the system in a way that every Company will be able to serve games on their gaming site from their domain?**

Firstly, implement a multi-tenancy architecture where each company (gaming site) operates within its own isolated environment on the platform. Next, configure the platform to support custom domains for each company, allowing them to use their own domain (e.g., cool-games.com, luck-games.co.uk). Then, set up domain routing and DNS configuration so that each domain points to the respective company's instance of the platform. Ensure SSL certificates are managed and configured for each custom domain to provide secure connections. Implement a centralized backend that supports scalable hosting of web games and provides a backoffice for managing players across all gaming sites. Develop APIs or integrations that allow each company to manage their specific games, player accounts, and other customized features on their domain. Finally, maintain robust data isolation and security measures to protect each company's data and ensure compliance with privacy regulations across different gaming domains.

1. **What modification should be done to the users table at gPlatform to support this change?**

Firstly, add a tenant\_id column to the users table to associate each user with a specific gaming domain (or tenant). Next, ensure that the email column is part of a unique constraint that includes tenant\_id, allowing each domain to have unique user emails. Then, establish a foreign key constraint on the tenant\_id column referencing a tenants table, which holds information about each gaming domain. Afterward, update your application logic to include tenant\_id when querying or manipulating user data, ensuring operations are scoped to the correct domain. Lastly, consider implementing tenant-specific configurations or settings in the users table to accommodate varying needs across different gaming domains.

1. **Considering we have 1 backend cluster that serves all companies, how can we validate a user login on one gaming domain in such a way that it does not give access to a different gaming domain? (i.e. authenticating on site A, grants access to site A only)**

First, ensure each gaming domain has a unique identifier within your system. Next, establish separate authentication mechanisms or endpoints for each domain, such as /auth/{domain}/login, ensuring requests are routed and processed accordingly. Then, implement a credential store that segregates user data by domain, preventing unauthorized access between domains. After setting up domain-specific authentication, integrate token-based authentication where each token includes domain-specific information, ensuring requests are contextually validated. Subsequently, employ strict request routing and authorization checks that verify domain-specific access rules before processing any requests. Finally, regularly audit and update your authentication and authorization mechanisms to maintain robust security across all gaming domains.