Onboarding in a SaaS platform

Fullstack Devs Evaluation Task

Introduction

This document outlines the evaluation task for full-stack developers contributing to a multi-tenant SaaS platform built on Laravel. The objective is to implement a structured, environment-aware onboarding and provisioning flow for new tenants, while adhering to clean separation of logic across multiple runtime environments.

The SaaS application is architected around a strict isolated-database multitenancy model, powered by the Spatie Multitenancy package. Each tenant operates within its own database, with no shared data or authentication layers. The platform is divided into three clearly defined environments—**root**, **landlord**, and **tenant**—each serving a distinct purpose, and each requiring precise routing and scoping behavior.

Before any feature implementation begins, it is critical to understand the role and behavior of each environment, how tenant resolution and subdomain routing is handled, and how connections to tenant-specific databases are dynamically configured and activated.

This task is not purely academic—it reflects production-level architecture used in scalable SaaS systems. Developers are expected to demonstrate not only technical implementation, but thoughtful decisions around structure, state management, validation, security, and background processing.

The remainder of this document outlines the environments in detail, followed by the implementation task itself, the expected flow, and final delivery expectations.

Tech Stack

The following technologies are expected to be used for this implementation:

- **Laravel** (latest stable release)
- **Spatie Multitenancy** (Isolated DB mode)
- **Laravel Queues** (Redis or database driver)
- **MySQL** (for both landlord and tenant databases)
- **Session/Cookie Handling** (for onboarding state)
- **Virtual Host / Local Domain Configuration** (e.g., `myapp.test`, `company1.myapp.test`)
- **Frontend**: While React with Inertia.js is preferred to align with the broader application stack, alternative frontend approaches are acceptable as long as the core functionality and architectural requirements are met.

Environment Overview

The system is organized into three distinct runtime environments. Each serves a different operational role and requires precise handling of routing, database resolution, and service scope.

Root environment

Domain example: `https://myapp.test`

Purpose:
The root environment is the public-facing entry point of the application. It is responsible for handling global access such as the multi-step onboarding/sign-up flow for new organizations (tenants). This environment operates completely outside the context of any tenant and does not activate a tenant-specific database.

Key characteristics:

- No tenant context is resolved in this environment.
- The application uses the default Laravel connection configured for the application (commonly referred to as the "landlord" or global database).
- Routes in this environment include the landing page, sign-up pages, and terms/legal policies.
- All email uniqueness and onboarding flows are validated at a global level using the landlord database.

Landlord environment

Domain example: `https://landlord.myapp.test`

Purpose:
This is the internal administrative layer of the platform used by system administrators or platform operators. It manages the lifecycle of tenants—tracking their creation status, listing existing tenants, and optionally performing manual overrides or deletions. The landlord environment interfaces exclusively with the global landlord database and never activates a tenant context.

Key characteristics:

- Entirely scoped to the landlord database connection.

- No tenant identification or scoping is performed here.
- Authentication is handled for internal users (platform admins), not tenant users.
- Routes in this environment may include dashboards for tenant management, error logs, and provisioning analytics.
- Access to this environment must be guarded and restricted to authorized users.

Tenant environment

Domain pattern: `https://{tenant}.myapp.test`

Purpose:
This is the environment in which isolated tenants operate. Each tenant is assigned a unique subdomain that is mapped to its own isolated database. Once a tenant is provisioned during onboarding, it becomes accessible under its designated subdomain. All business logic, data access, and user authentication in this environment are scoped strictly to the tenant's context.

Key characteristics:

- Tenant resolution is performed by inspecting the request's subdomain.
- Upon resolution, the application uses Spatie's `Tenant::makeCurrent()` to bind the request lifecycle to the tenant's context.
- A unique database connection is configured for each tenant using runtime overrides (typically via Laravel's configuration system).
- All routes within this environment are protected and require authentication against the tenant-specific users table.

- No data from other tenants is accessible under any circumstances, regardless of access method or user roles.

Database strategy

The system uses an isolated database approach, meaning that each tenant has their own dedicated database. This allows for:

- Strong data isolation across tenants
- Easier backup and restore per tenant
- Enhanced scalability and flexibility for custom features per tenant

Tenant databases are dynamically configured at runtime based on the tenant's metadata (retrieved from the landlord database). The configuration is injected into Laravel's database connection system before any database interactions take place. Once configured, this connection is used for the lifetime of the request cycle through the tenant environment.

Tenant identification

Tenant identification is handled via subdomain parsing. The application uses custom middleware or the Spatie multitenancy package's built-in tenant finder to resolve a subdomain like `company1.myapp.test` into a Tenant model instance from the landlord database. This model is then used to configure the database and other scoped services for the duration of the request.

If the subdomain does not match any existing tenant or the tenant is in a failed/provisioning state, the request is rejected or redirected.

The Task

This task focuses on implementing the multi-step sign-up and onboarding flow for new tenants within the **root environment**. The goal is to enable organizations to register for the platform, provide necessary account and company details, and be provisioned as isolated tenants upon completion of the flow.

The developer is expected to:

- Implement the user-facing sign-up interface in the root environment.
- Persist onboarding data progressively across multiple steps.
- Handle all necessary validations and edge cases.
- Initiate and manage the tenant creation process through a queued job.
- Ensure seamless transition from onboarding completion to tenant access via subdomain.

The root environment is publicly accessible at `https://myapp.test`, and all logic in this task must be executed without activating any tenant context.

This task is limited to the onboarding and provisioning layer only. It does not include implementation within the landlord or tenant environments but assumes familiarity with their behavior, as described in the architecture documentation.

Functional Scope

The onboarding flow must include the following five steps:

1. Account Information
2. Password Setup

3. Company Details
4. Billing Information
5. Confirmation and Provisioning

Each step must save data persistently to the landlord database and prevent users from skipping or manipulating the flow. On completion, the system will dispatch a background job to provision the tenant's environment and redirect the user to the newly created subdomain.

Step-by-Step Breakdown

Step 1: Account Information

Objective
Collect basic user identity details required to initiate an onboarding session.

Form Fields

- Full Name
- Email Address

Validations

- Email must be unique across all onboarding sessions and existing tenants (checked against the landlord database)
- Required fields: full name and email address
- Valid email format

Technical Notes

- On success, create an `OnboardingSession` record in the landlord database
- Generate a signed or tokenized continuation URL for step progression and resumption
- Associate the session with the browser via session or cookie

State Handling

- Session data is persisted in the database
- Users who abandon onboarding must be able to resume from where they left off

---

Step 2: Password Setup

Objective
Collect and confirm a secure password for the account being created.

Form Fields

- Password
- Confirm Password

Validations

- Required: both fields
- Minimum 8 characters
- At least one lowercase, one uppercase, one number, one special character
- Password and confirmation must match

Technical Notes

- Password should be hashed immediately using Laravel's hashing mechanism
- Stored in the `OnboardingSession` model, not yet tied to an actual user
- Do not expose hashed or plain text passwords in any response

State Handling

- Step loads with previously stored data (if resuming), though passwords should not be pre-filled

---

Step 3: Company Details

Objective

Collect essential company identity details and reserve a unique subdomain.

Form Fields

- Company Name
- Subdomain
- Optional: industry, company size, logo

Validations

- Subdomain must be unique across existing tenants and onboarding sessions
- Allowed characters: lowercase alphanumeric and dashes
- Must not match any reserved keywords (e.g., admin, www, landlord)
- Required: company name and subdomain

Technical Notes

- Normalize subdomain inputs (lowercase, trimmed)
- Save company metadata to the existing onboarding session
- Reserved keywords should be defined in configuration and enforced globally

State Handling

- Data is saved to the onboarding session
- Must support review or edits prior to final submission

Step 4: Billing Information

Objective
Collect billing-related metadata (no real payment processing required for this phase)

Form Fields

- Billing Name
- Address
- Country
- Phone Number

Validations

- All fields required
- Phone number should follow E.164 format or a validated international format
- Country field should support validation against a controlled list (if implemented)

Technical Notes

- Placeholder billing only; payment gateway is not part of this task
- Data is stored in the landlord database alongside other onboarding details

State Handling

- Continue using persistent state tied to the onboarding session
- Allow editing of previous steps before confirmation

Step 5: Confirmation and Tenant Provisioning

Objective

Trigger the backend process to fully provision the tenant's environment.

Actions on Submit

- Final validation of the entire onboarding session
- Dispatch a queued provisioning job that will:
    - Create a tenant record in the landlord database
    - Create an isolated tenant database
    - Run migrations
    - Seed initial user using onboarding data
    - Apply default configuration (timezone, locale, etc.)

Technical Notes

- Provisioning jobs must be idempotent and retry-safe
- Track provisioning status (e.g., pending, completed, failed) on the tenant record
- On success, redirect user to tenant's subdomain login page

- Optionally send confirmation email

Edge Case Handling

- Duplicate email or subdomain conflicts
- Abandoned onboarding flows (with expiration policy)
- Provisioning job failures (with retry logic and logging)
- Subdomain normalization issues (case sensitivity, symbol restrictions)

Technical Requirements

- The flow must persist state at every step in the landlord database
- Access to each step must be conditional based on completion of previous steps
- Resumable via secure token-based URLs or session state
- Provisioning must occur via Laravel queues, using supported drivers (Redis, DB, etc.)
- All onboarding logic must remain fully outside the tenant context

Completion Checklist

The implementation will be considered complete when the following outcomes are achieved across all environments:

1. **Root environment** (`https://myapp.test`)
    - Fully functional, validated multi-step onboarding flow
    - Onboarding sessions are persistently stored and resumable
    - Successful onboarding triggers a queued tenant provisioning job

- Redirect to tenant domain upon provisioning completion

2. **Landlord environment** (`https://landlord.myapp.test`)
   - Admin-only interface displaying a list of all tenants (Updating and Deleting records is optional)
   - Tenant metadata reflects correct status post-provisioning
3. **Tenant environment** (`https://{tenant}.myapp.test`)
   - Functional login screen for tenant-specific users
   - Authentication checks the tenant's isolated database
   - On successful login, user is directed to a welcome page scoped to the tenant
   - All queries are correctly tenant-scoped via subdomain resolution
4. **General system behavior**
   - Multi-tenancy operates on an isolated database model using Spatie's package
   - All onboarding and tenant creation logic is correctly separated by environment
   - Queue system is configured, tested, and handles provisioning reliably
   - Virtual host routing works for all environment types in local development
5. **Demonstration**
   - A brief **video demonstration** is recorded showing:
     - The full onboarding and login experience
     - Code walkthrough of the implementation
     - Explanation of architectural decisions and job handling

Submission Guidelines

- **Deadline**:
  Submit your completed project within **7 days** of receiving this task.
- **Submission Format**:
  - Share a link to a **Git repository** (GitHub) containing:
    - The full source code
    - A clear and concise `README.md`
    - The recorded video (either embedded or linked)
    - Any additional documentation
- **Access Requirements**:
  Ensure the repository is **public** or that access is granted to our reviewers (@ahmed-z0).

> ⚠️ Developers are encouraged to use AI tools such as **ChatGPT**, **Gemini** (preferably **Gemini 2.5 Pro**), or any assistant of their choice to support their implementation. This evaluation is not about recalling syntax — it's about delivering scalable, maintainable solutions using modern development practices.
>
> That said, submissions that reflect blind **copy-pasting** or a **lack of understanding** will be rejected immediately. Use AI to enhance your workflow — not to substitute critical thinking or architectural awareness.

Summary

This task is designed to validate the developer's ability to implement environment-aware functionality within a multi-tenant Laravel SaaS architecture. The focus is on building a robust onboarding and provisioning flow in the root environment, while maintaining strict separation of concerns across the root, landlord, and tenant environments.

The outcome of this task serves as the foundation for scaling the application to support dynamic, isolated SaaS tenants—each with their own workspace and data. It also sets a precedent for clean architectural boundaries, production-readiness, and developer discipline within a growing codebase.

*************************🌟 Good Luck! 🌟*************************