■# GoatGoat Environment & Build Configuration Guide (Customer, Seller, Delivery Apps)

This document explains how the **Customer/Delivery app** (`C:\client`) and the **Seller app** (`C:\Seller App 2\SellerApp2`) decide **which backend environment** to talk to (staging vs production), and how to build APKs for both emulators and physical devices safely.

The goal:

- During development and QA, **all apps should point to the staging stack**.
- Later, when production is ready, we can flip **a single environment flag (`APP_ENV`)** so specific builds use the production domain.

---

1. Server endpoints and environments

We have two backend environments exposed via nginx:

- **Production**: `https://goatgoat.tech`
- **Staging**: `https://staging.goatgoat.tech`

Both servers expose:

- REST API: `https:///api/...`
- Admin / FCM: `https:///admin/...`
- Socket.IO: `https://` (same host, no `/api` or `/admin` suffix)

All apps use these domains. The only question is **which one** they pick for a given build.

---

2. Customer + Delivery app (`C:\client`)

2.1 Config file

File: `src/service/config.tsx`

```
import { Platform } from 'react-native';
import Config from 'react-native-config';

// GOATGOAT API CONFIGURATION - MULTI-ENVIRONMENT SUPPORT
const DEVELOPMENT_IP = '192.168.1.10'; // Local dev IP (only used for local server)
// SERVER ENDPOINTS
```

```
const PRODUCTION_URL = 'https://goatgoat.tech';        // Production server
const STAGING_URL    = 'https://staging.goatgoat.tech'; // Staging server
const LOCAL_URL      = `http://${DEVELOPMENT_IP}:3000`; // Local dev server

// ENVIRONMENT DETECTION
const IS_DEVELOPMENT = __DEV__;

// ENVIRONMENT SELECTION WITH APP_ENV OVERRIDE
// APP_ENV comes from .env.* files via react-native-config.
//  - APP_ENV=production → production
//  - anything else or missing → staging (safe default)
const APP_ENV = Config.APP_ENV || 'staging'; // 'staging' | 'production'
const ENVIRONMENT: 'staging' | 'production' =
  APP_ENV === 'production' ? 'production' : 'staging';

// ENVIRONMENT-SPECIFIC URL GENERATION
const getBaseURL = () => {
  switch (ENVIRONMENT) {
    case 'production':
      return `${PRODUCTION_URL}/api`;

    case 'staging':
    default:
      return `${STAGING_URL}/api`;
  }
};

const getSocketURL = () => {
  switch (ENVIRONMENT) {
    case 'production':
      return PRODUCTION_URL;
    case 'staging':
    default:
      return STAGING_URL;
  }
};

export const BASE_URL   = getBaseURL();
export const SOCKET_URL = getSocketURL();
export const GOOGLE_MAP_API = Config.GOOGLE_MAP_API || 'YOUR_DEFAULT_API_KEY';
export const BRANCH_ID      = Config.BRANCH_ID || '68a1a76e2c93ad61799983b3';

// DEBUG LOGGING (dev only)
if (__DEV__) {
  console.log('=== GOATGOAT API CONFIGURATION ===');
  console.log('Build Type:', __DEV__ ? 'DEBUG BUILD' : 'RELEASE BUILD');
  console.log('APP_ENV:', APP_ENV);
  console.log('Environment:', ENVIRONMENT);
  console.log('BASE_URL:', BASE_URL);
  console.log('SOCKET_URL:', SOCKET_URL);
  console.log('====================================');
}

export const ENVIRONMENT_INFO = {
  environment: ENVIRONMENT,
  appEnv: APP_ENV,
  isDevelopment: IS_DEVELOPMENT,
  platform: Platform.OS,
  baseUrl: BASE_URL,
  socketUrl: SOCKET_URL,
};
```

Key points:

- **Only `APP_ENV` decides staging vs production**.
- `APP_ENV` is read from `.env.*` via `react-native-config`.
- Default (when `APP_ENV` is not set or not `"production"`):
- `ENVIRONMENT = 'staging'`
- `BASE_URL = https://staging.goatgoat.tech/api`
- `SOCKET_URL = https://staging.goatgoat.tech`

2.2 Recommended .env configuration

In `C:\client`:

- `.env.development` (used for debug builds):

```env
APP_ENV=staging
```

- `.env.staging`:

```env
APP_ENV=staging
```

- `.env.production` (for real production builds later):

```env
APP_ENV=staging # SAFE FOR NOW. Change to production when ready.
```

When you are ready to ship production builds, change **only** `.env.production` to:

```
APP_ENV=production
```

This way:

- All dev / QA builds continue to use **staging**.
- Only production CI / release builds (which use `.env.production`) will talk to `https://goatgoat.tech`.

2.3 Debug vs Release builds

- **Debug on emulator / device (Metro)**

- Command: `npx react-native run-android` (or via Gradle with debug variant).

- Uses `.env.development` → `APP_ENV=staging`.

- JS bundle is loaded from Metro, but APIs go to **staging**.

- **Release APK for testers (`./gradlew assembleRelease`)**

- Uses `.env.production`.

- With `APP_ENV=staging` in `.env.production`, the APK will **still talk to staging**.

- When you later set `APP_ENV=production` there, those release builds will talk to production.

---

3. Seller app (`C:\Seller App 2\SellerApp2`)

3.1 Environment selection

File: `src/config/environment.ts`

This file controls the Seller app's base URL. It has been rewritten to use the same domains and the same `APP_ENV` idea.

```
/**
 * ENVIRONMENT CONFIGURATION (Seller App)
 *
 * Uses APP_ENV to select between staging and production servers:
 * - APP_ENV=staging    → https://staging.goatgoat.tech
 * - APP_ENV=production → https://goatgoat.tech
 *
 * Default is STAGING to keep all builds safe during development.
 */

const environments = {
  staging: {
    // Staging via nginx proxy (same as customer app)
    API_BASE_URL: 'https://staging.goatgoat.tech',
    FCM_ENDPOINT: '/admin/fcm-management',
    DEBUG_MODE: true,
    ENVIRONMENT: 'staging',
  },
  production: {
    // Production via nginx proxy (same as customer app)
    API_BASE_URL: 'https://goatgoat.tech',
    FCM_ENDPOINT: '/admin/fcm-management',
    DEBUG_MODE: false,
    ENVIRONMENT: 'production',
  },
};

/**
 * ENVIRONMENT SELECTION
 *
 * APP_ENV is read from process.env (Metro / build env):
```

```
 * - 'production' → production config
 * - anything else (or missing) → staging config
 */

const appEnvFromProcess = (process.env.APP_ENV || '').toLowerCase();
const appEnv: 'staging' | 'production' =
  appEnvFromProcess === 'production' ? 'production' : 'staging';

const config = environments[appEnv];

console.log(`Seller Environment: ${config.ENVIRONMENT.toUpperCase()}`);
console.log(`Seller API Base URL: ${config.API_BASE_URL}`);

export default config;
```

3.2 Seller API and Socket.IO URLs

File: `src/config/index.ts`

```
import environment from './environment';

const API_BASE_URL   = `${environment.API_BASE_URL}/api`;
const SELLER_API_URL = `${API_BASE_URL}/seller`;

export const API_ENDPOINTS = {
  LOGIN:          `${SELLER_API_URL}/login`,
  VERIFY_OTP:     `${SELLER_API_URL}/verify-otp`,
  RESEND_OTP:     `${SELLER_API_URL}/resend-otp`,
  LOGOUT:         `${SELLER_API_URL}/logout`,
  PROFILE:        `${SELLER_API_URL}/profile`,
  UPDATE_PROFILE: `${SELLER_API_URL}/profile`,
  STORE_REGISTER: `${SELLER_API_URL}/register`,
  // ... other endpoints ...
};

export const CONFIG = {
  API_TIMEOUT: 30000,
  OTP_RESEND_DELAY: 30,
  TOKEN_REFRESH_THRESHOLD: 5 * 60 * 1000,
  GOOGLE_MAPS_API_KEY: 'AIzaSyDOBBimUu_eGMwsXZUqrNFk3puT5rMWbig',
} as const;

// Socket.IO base URL (matches API host; no /api suffix)
export const SOCKET_URL = environment.API_BASE_URL;
```

So:

- `APP_ENV` **not** `"production"` → Seller uses **staging.goatgoat.tech** for both REST and Socket.IO.
- `APP_ENV=production` → Seller uses **goatgoat.tech** for both.

3.3 Setting APP_ENV for Seller builds

For SellerApp2, `APP_ENV` is read from `process.env.APP_ENV` (Node/Metro/build env).

Examples (PowerShell / cmd, from `C:\Seller App 2\SellerApp2`):

- **Dev / staging build (recommended default):**

```powershell
# Debug run on device/emulator
set APP_ENV=staging && npx react-native run-android
```

- **Release APK that still talks to staging:**

```powershell
set APP_ENV=staging && .\gradlew assembleRelease
```

- **Later, production build:**

```powershell
set APP_ENV=production && .\gradlew assembleRelease
```

If `APP_ENV` is not set at all, the Seller app defaults to **staging**, which is safe.

---

4. Summary: keeping everything on staging now

To ensure **all three apps** (customer, delivery, seller) talk to **staging** right now:

1. **Customer/Delivery app (`C:\client`):**
- In `.env.development`, `.env.staging`, `.env.production`, set:

```env
APP_ENV=staging
```

- Build / run commands (examples):

```bash
# Debug on emulator or device (Metro)
npx react-native run-android
```

Release APK for testers (still talking to staging)
./gradlew assembleRelease
```

2. **Seller app (`C:\Seller App 2\SellerApp2`):**
- When running or building, set `APP_ENV=staging`:

```powershell
# Debug run
set APP_ENV=staging && npx react-native run-android

# Release APK for testers
set APP_ENV=staging && .\gradlew assembleRelease
```

- If `APP_ENV` is omitted, Seller still defaults to **staging**.

At this point **no build will talk to production** unless you explicitly set `APP_ENV=production`.

---

5. How to switch to production later (controlled rollout)

When the production backend is fully ready and tested, you can selectively move builds to production.

5.1 Customer/Delivery app

- Change only `.env.production` in `C:\client`:

```env
APP_ENV=production
```

- Keep `.env.development` and `.env.staging` as:

```env
APP_ENV=staging
```
```

- Your CI / release pipeline, which uses `.env.production`, will now build APKs that talk to **goatgoat.tech**.

- Local dev / internal QA builds continue to hit **staging.goatgoat.tech**.

5.2 Seller app

- For production Seller builds, set `APP_ENV=production` only in the environment where you run the **release** build:

```powershell
set APP_ENV=production && .\gradlew assembleRelease
```

- For internal QA Seller builds, continue using `APP_ENV=staging`.

This allows a **phased rollout** where you can:

- Keep development + QA traffic on staging.

- Turn on production traffic only for specific, controlled builds.

---

6. Notes on Metro vs bundled JS (APK on devices)

- **Debug builds** (Metro):

- JS bundle is loaded from Metro (port 8081).

- API URLs still follow `APP_ENV` (staging/production).

- **Release / test APKs** (`assembleRelease`):

- JS bundle is packaged into the APK.

- They do **not** need Metro.

- They use the configured `BASE_URL` and `SOCKET_URL` based on `APP_ENV`.

For physical device testing without Metro, always use a build where the JS bundle is packaged (`assembleRelease` or a debug build with `bundleInDebug: true`), and control the backend via `APP_ENV`.

This setup keeps the logic simple:

- **Toggle `APP_ENV` to control backend env.**

- **Default to staging** everywhere to avoid accidental production calls.

- Use the **same domain structure** (`staging.goatgoat.tech` / `goatgoat.tech`) across customer, delivery, and seller apps.

You can hand this document to anyone responsible for DevOps / builds, and they will know exactly where and how to configure environment selection.