**Handover Guidance File**

# Introduction

This document is designed to serve as a comprehensive handover guide for you as you assume full ownership of your digital product and its supporting infrastructure. It provides a structured framework for collecting assets, validating credentials, and confirming that all essential components are transferred securely and completely.

The intention is not only to ensure that you receive all technical deliverables, but also to equip you with the knowledge and best practices required to manage your systems independently.

* Task Goals

This guidance supports in achieving four critical objectives:

1. Ownership : Ensuring that you hold full administrative control over all project assets, accounts, and infrastructure.
2. Independence : Enabling you to test, validate, and deploy without reliance on the development team.
3. Risk-Free Handover : Identifying and eliminating gaps, red flags, or dependencies that could jeopardize continuity.
4. Confidence : Providing professional clarity through a structured process, so you can accept the project handover with assurance.

# Source Code Collection

You should collect the complete source code for every platform included in your project. At a minimum, each platform must be kept in a separate repository. Most importantly, the ownership of these repositories should belong to you or your company. Developers should only be added as collaborators, and you can decide the exact access level (read, write, or admin) based on what they need. Once their work is finished, you can easily remove their access while keeping your code fully secure.

If developers own the repositories, your project is at risk; they can remove your access or control critical parts of your system. By ensuring ownership stays with you, you protect your investment and maintain independence.

When reviewing your repositories, make sure that:

* Each repo is hosted under your company account on GitHub
* You have admin privileges (not just contributor rights).
* All needed branches (main, development, feature branches…) are present.
* Essential files are included:
  + README.md with setup & deployment instructions
  + Build scripts (package.json, Podfile, build.gradle, Dockerfiles if used)
  + gitignore files to prevent sensitive data leaks
* You have access to repos for:
  + OS App (Swift)
  + Android App (Kotlin)
  + Admin Panels (MEAN Stack, 2 repos)
  + Company Website (MEAN Stack)
  + Deployment pipelines or automation scripts (if used)

# Credentials & Access

For this part, you should handle and verify the most important elements related to security and ownership. Without this step, your project remains at risk of unauthorized access or hidden dependencies.

## AWS

You must confirm that AWS is fully under your company’s control and that no developer retains ownership or excessive privileges.

* Review all IAM users and roles:
  + Identify IAM users linked to your applications (programmatic access).
    - These users should only have the permissions needed for the app (for example, access to S3 or EC2).
    - They should not have console login access, since they are not meant to log in directly.
  + Identify IAM users assigned to developers.
    - Once the work is finished, reset their passwords or remove their access.
* Confirm that the root user is never used for daily work. It should only be used by you for billing or emergencies.
* Verify that you can add, update, or remove IAM users on your own.

## Twilio (Communication Services)

Twilio’s use depends on your project, may be used for SMS, voice, WhatsApp, or email. To confirm that you fully control it:

* Log into the Twilio Console with your company-owned account.
* Confirm that you can see the Account SID and Auth Token.
* Identify which service your project uses and test it:
  + If SMS → send yourself a test SMS.
  + If Email (SendGrid) → send a test email.
  + If Voice/WhatsApp → make a test call or send a WhatsApp message.
* Verify that any phone numbers, sender IDs, or email domains used in your project appear in your account.
* Check that callback URLs point to your servers, not external ones.
* Make sure you are the account owner, and developers are only added as users.

## TAP (Payment Gateway / API Platform)

* Log into TAP with your company account.
* Confirm you can see both Test and Live API keys.
* Run a small real transaction (or a test charge) to confirm live keys are active.
* Ensure keys are stored securely (in .env files, not hardcoded in code).
* Verify you can access TAP’s transaction logs and dashboards.

## Meta (Facebook/Instagram APIs)

* Log into the Meta Business Manager with your company account.
* Verify that the App ID and App Secret are registered under your business, not a developer’s personal account.
* Check that you can access the Meta Developer Console and view approval/review status for any live integrations.

## Domain Resgiter (GoDaddy)

* GoDaddy is only the registrar, confirm you own the domain in your GoDaddy account.
* DNS records are managed in AWS Route53, not in GoDaddy.
* In GoDaddy, you only need to verify that the nameservers point to AWS Route53.

# AWS Infrastructure Validation

AWS validation it’s about confirming that every service your system depends on is properly configured, secure, and cost-efficient. At this stage, you need to go beyond access and check whether the infrastructure is tuly ready.

When validating your AWS setup, make sure that:

* EC2(servers)
  + Review all EC2 instances and confirm the instance type (size/configuration) is suitable.
    - If the system is just delivered and not in production yet, consider downgrading to smaller instances to save costs until you have active users.
    - If load increases later, you can upgrade again.
  + Confirm that every EC2 instance linked to a domain has an Elastic IP attached. Avoid the default dynamic IPs that change after a reboot.
  + Request and receive the key pair (.pem file) for each instance.
    - Test SSH access immediately after receiving it:
      * For Ubuntu: ssh -i keypair\_path.pem ubuntu@ElasticIP
      * For Amazon Linux: ssh -i keypair\_path.pem ec2-user@ElasticIP
  + If apps are managed inside EC2 using PM2 (Node.js process manager), reboot the instance to confirm PM2 is configured to auto-restart apps after reboot.
  + Verify there are no unnecessary EC2 instances left running.
    - If you find test servers, confirm with developers before deleting them.
* S3 (Storage Buckets)
  + If S3 is used to store data, check whether the bucket structure is clean and logical as required
  + For S3 access from EC2, confirm that IAM roles and bucket policies are configured correctly.
    - The EC2 IAM role should only have the permissions required for the app (no excessive privileges).
  + If your S3 buckets are not meant to be public, ensure “Block Public Access” is enabled.
* IAM (Users & Roles)
  + If your S3 buckets are not meant to be public, ensure “Block Public Access” is enabled.
  + IAM Roles:
    - Confirm only the roles actually needed by your applications exist.
    - Review each role’s permissions to ensure they follow the principle of least privilege (only what’s necessary, nothing more).
* Route 53
  + The domain is listed in Route 53 as a hosted zone.
  + “A” records point to the Elastic IPs of EC2 instances.
  + CNAMEs and MX records (if email is configured) are correct.
  + TTL values are reasonable (not too long during early setup).
  + No old or unused records left behind.

# Local Environment Setup

Before you can confidently accept the project, you must be able to run each app locally on your own machine. This step gives you independence: if something fails, you’ll know it’s because of setup, not because the developers still hold hidden knowledge.

For this, you’ll need to prepare your environment with the same tools your developers used, then test each application one by one.

## Required Tools:

Install the following on your local machine:

* Xcode → Required for building and running the iOS app (Mac only).
* Android Studio → Required for building and running the Android app.
* Node.js & npm → Needed for running the Admin Panels and Company Website (MEAN stack).but make sure to ask the developers which version of Node.js they used. Installing the wrong version can cause build failures or unexpected errors when running the apps locally.
* MongoDB Compass → A GUI tool that lets you connect to the MongoDB database, explore collections, and confirm the structure.
* Postman → For testing APIs if needed

## Testing Each App

* iOS App (Swift)
  + Open the ios app repo in Xcode.
  + Confirm you can build and run the app in the iOS Simulator.
  + Test all the functionnality and validate everything works smoothly
* Android App (Kotlin)
  + Open the android app repo in Android Studio.
  + Confirm you can build and run the app in the Android Emulator.
  + Also test all the functionnality and validate everything works smoothly
* Admin Panels & Company Website (MEAN Stack)
  + Install dependencies with npm install. (after confirm with developer which version you should install)
  + Run locally with npm start or the provided script.
  + Open the app in your browser (usually http://localhost:3000 or http://localhost:4200). And test
* Database (MongoDB)
  + Ask developers for a connection string to your MongoDB database.
  + Use MongoDB Compass to connect.
    - Confirm that the database schema matches your expectations (for example, users, orders, products collections).
  + If remote access is enabled, verify you can connect from your local machine.
    - If not, confirm with developers how local access is managed.

# Handover Readiness checklist

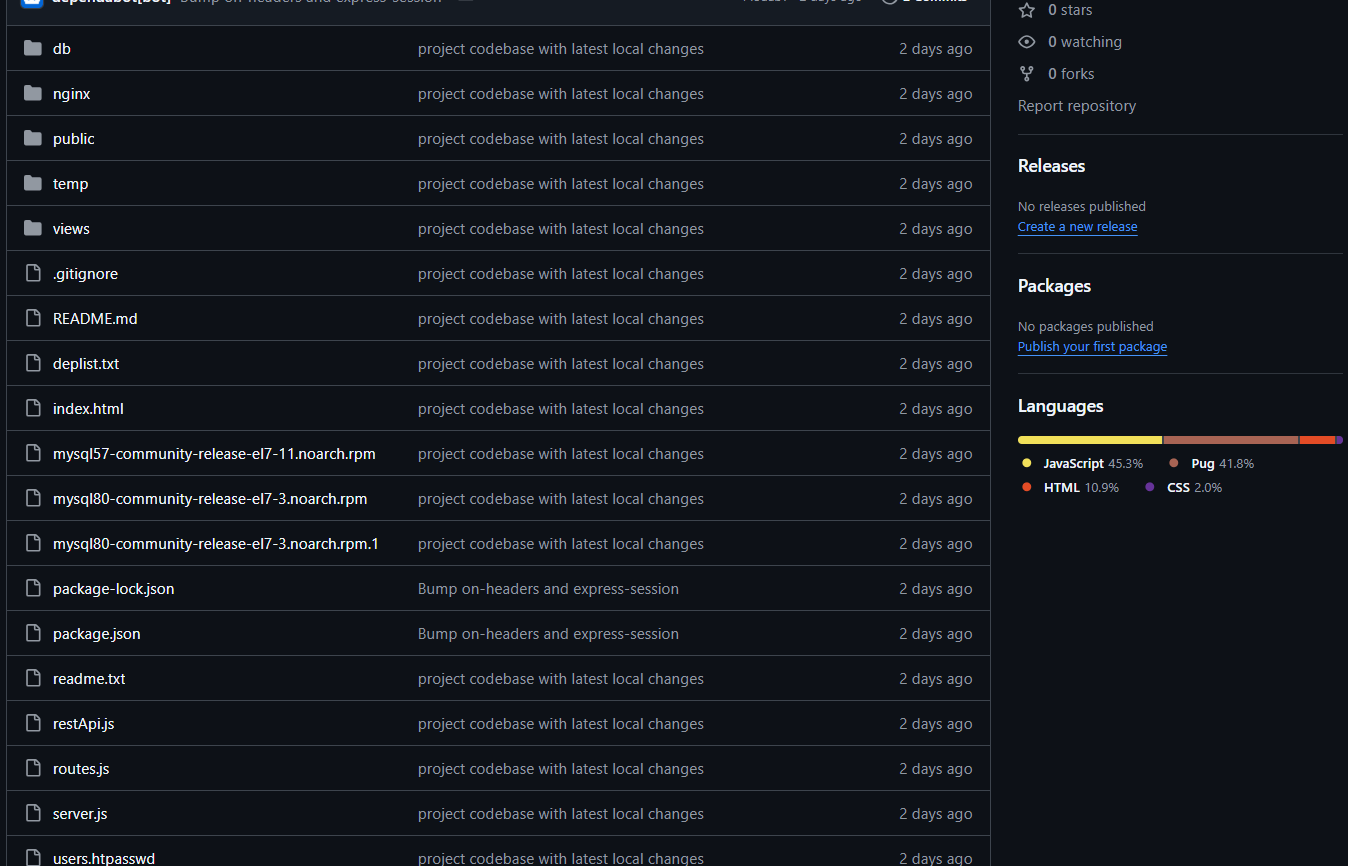
This is your final review before signing off the project. At this point, you should not assume anything, verify every item with your own access and tests. If something is missing or unclear, pause the handover and request it from your developers immediately.

* Source code
  + All repositories (iOS, Android, Admin Panels, Company Website) are hosted under your company account.
  + Documentation exists (README.md, deployment notes, build scripts).
  + .env files are not included or incluide just a templates only in the repos (no real secret values committed).
  + The real .env values are delivered securely, stored outside GitHub, and a versioned copy is kept for future updates.
* Credentials & Access
  + AWS IAM users access confirmed
  + API keys and secrets (Twilio, TAP, Meta) are delivered, tested, and stored securely.
  + Domain registrar (GoDaddy) is under your ownership, and nameservers point to AWS Route53.
* AWS Infrastructure
  + All EC2 instances reviewed: right sizing (downgraded if needed), Elastic IPs attached.
  + SSH key pairs received and tested (Ubuntu and Amazon Linux logins confirmed).
  + PM2 configured to auto-restart apps after reboot.
  + Only required EC2 instances remain (test servers removed or approved).
  + S3 buckets structured correctly, with public access blocked unless explicitly required.
  + IAM roles checked, only necessary roles exist, with least-privilege permissions.
  + Route 53 reviewed : all DNS records (A, CNAME, MX) are correct and match your infrastructure.
* Local Environment
  + Required tools installed (Xcode, Android Studio, Node.js in the right version, MongoDB Compass).
  + iOS app builds and runs in the simulator.
  + Android app builds and runs in the emulator.
  + Admin Panels and Company Website run locally without errors.
  + MongoDB Compass connects to the database
* Final Security Steps
  + All developer IAM users removed or passwords rotated.
  + All API keys, tokens, and database credentials rotated after handover.
  + Credentials stored in a company-controlled vault (not plain text).
  + Backup of repositories and credentials secured in client-owned storage.

# Case Study: ShirtClub Reference

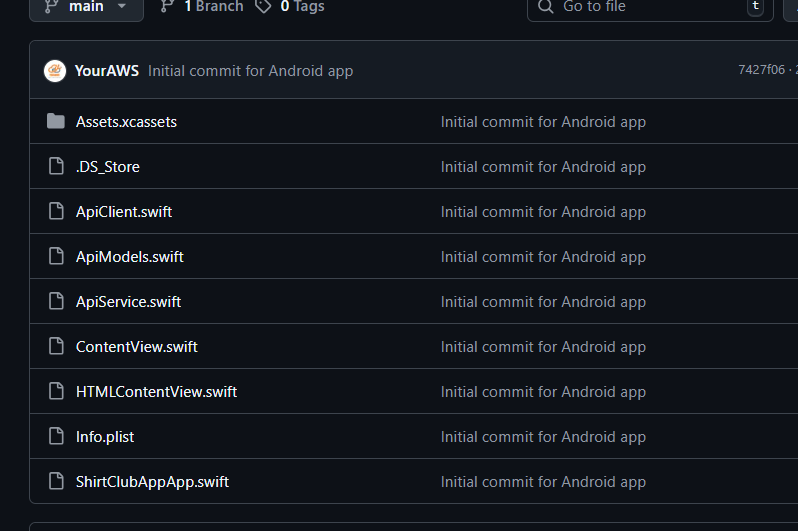
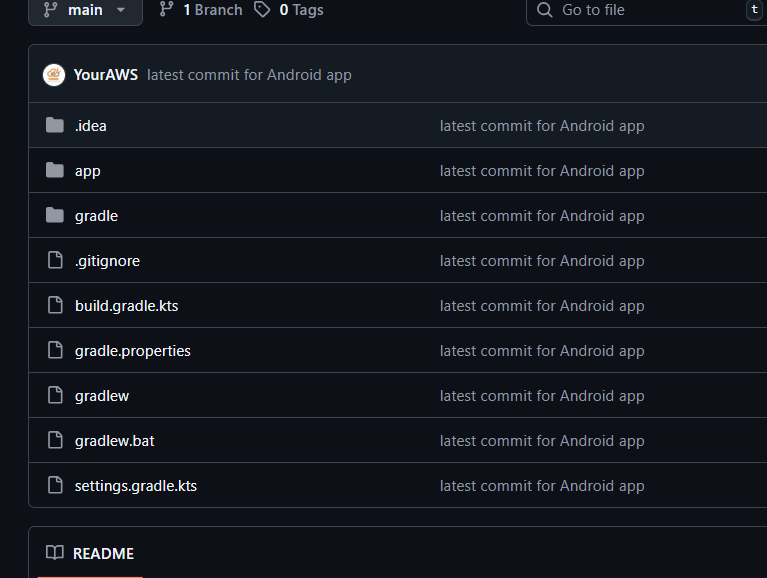
## Code source hosted in github

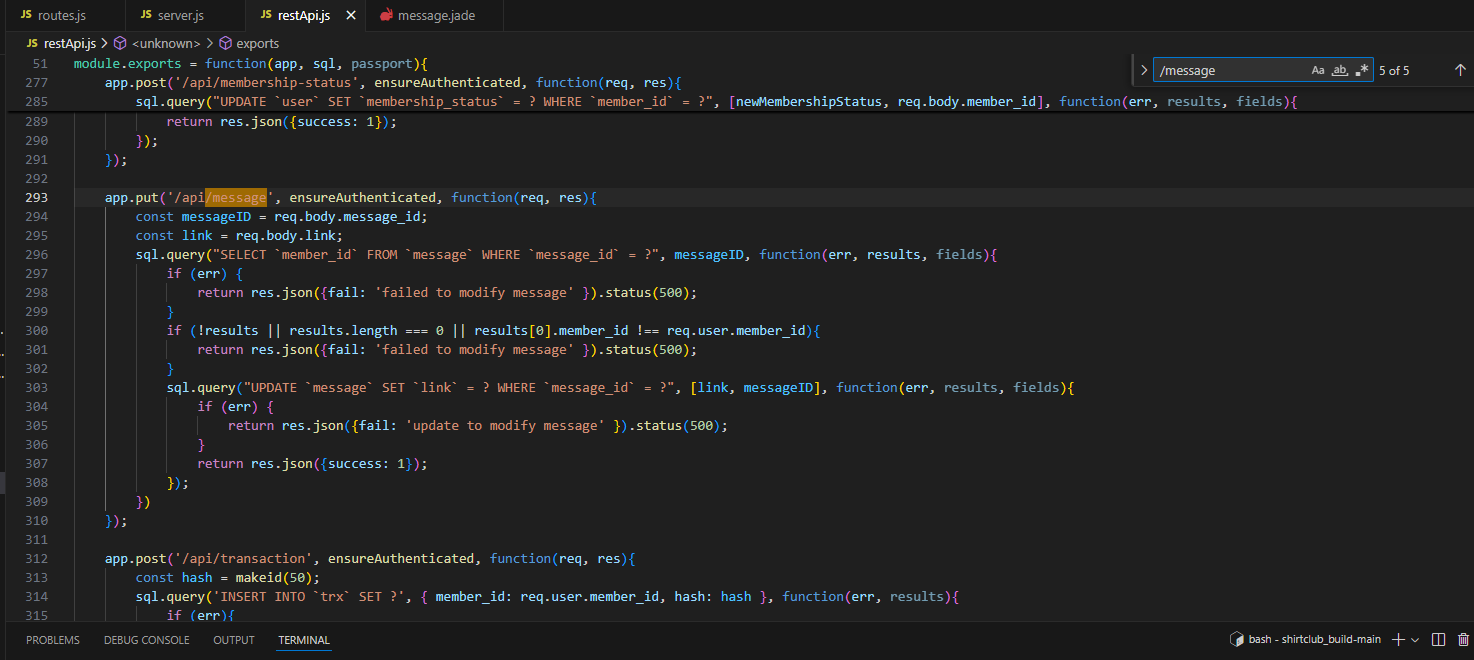
All your applications should exist in separate GitHub repositories, each dedicated to a specific platform. For example, in the ShirtClub case :



This setup follows security best practices:

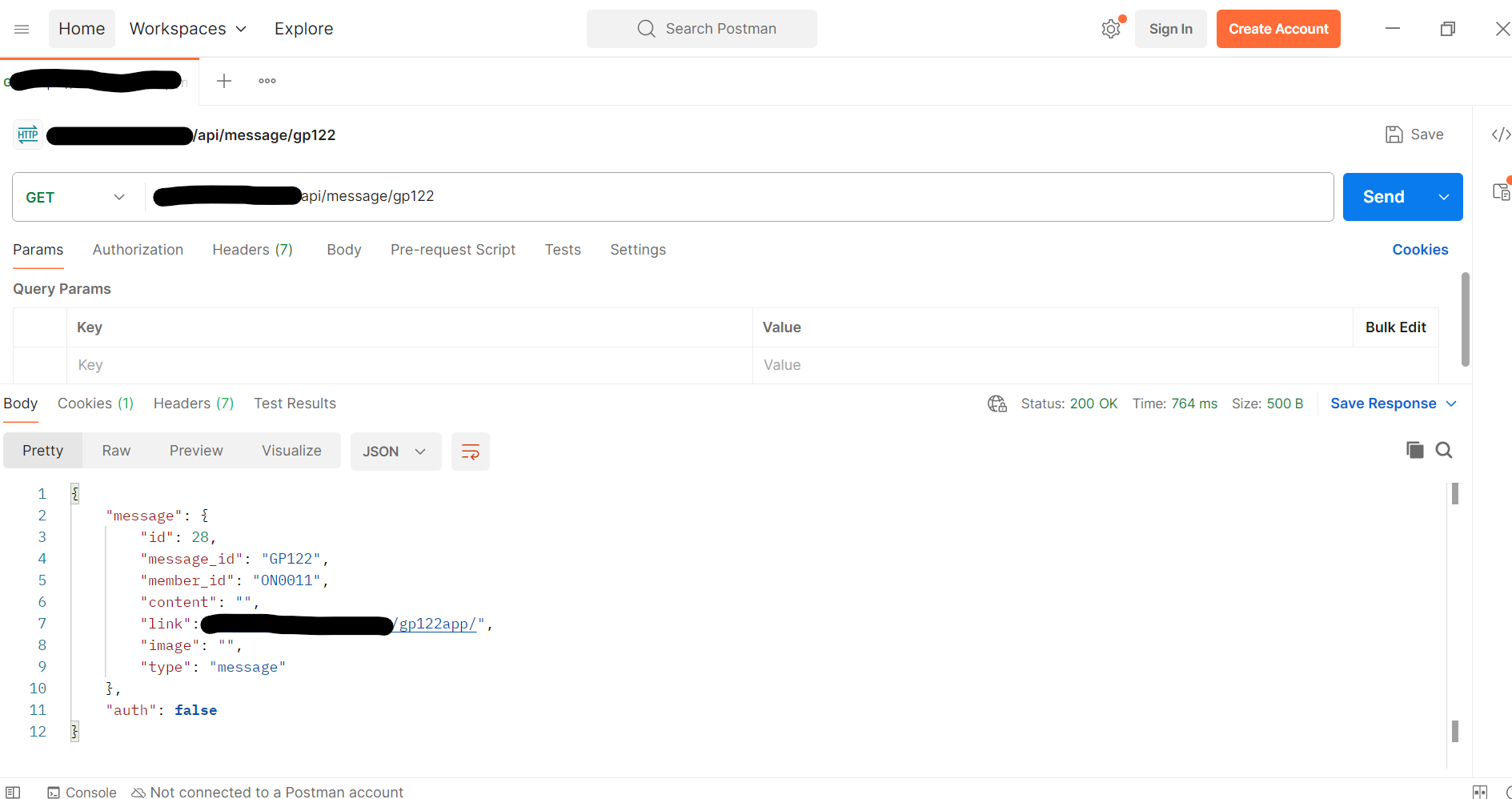
* .env files are not included with real values in the repositories
* The repositories are owned by the company account, while developers are added only as collaborators.
* Every platform (Website, Android app, iOS app) has its own repository, hosted in GitHub for both backup and deveopment workflow.



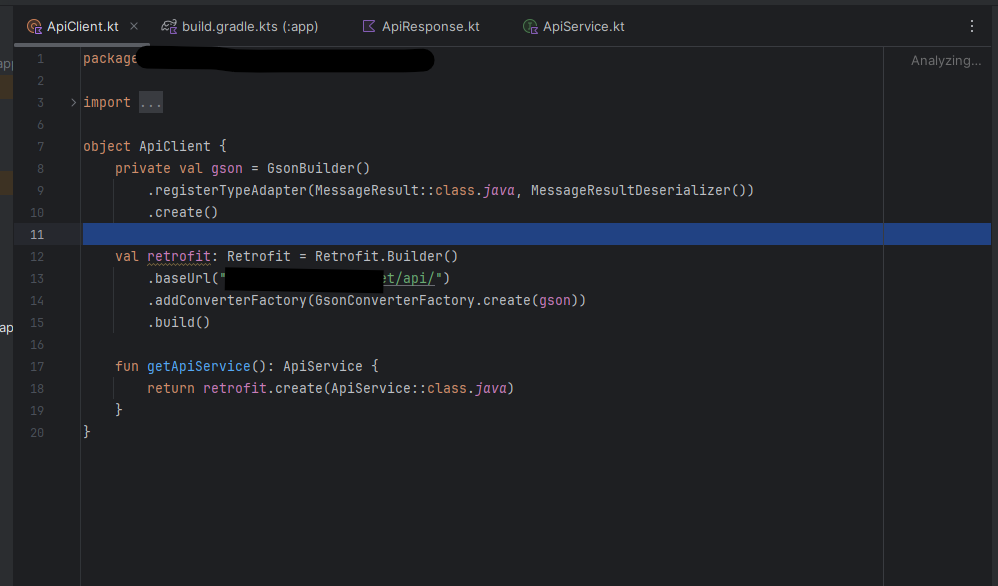
 In addition, the ShirtClub project also included a custom API, which retrieved information from the database and provided endpoints for use by the Android and iOS applications

For example, as shown in the screenshot, there was an endpoint that could search for a T-shirt message in the database. The client validated this API in two ways:

By sending requests through Postman.



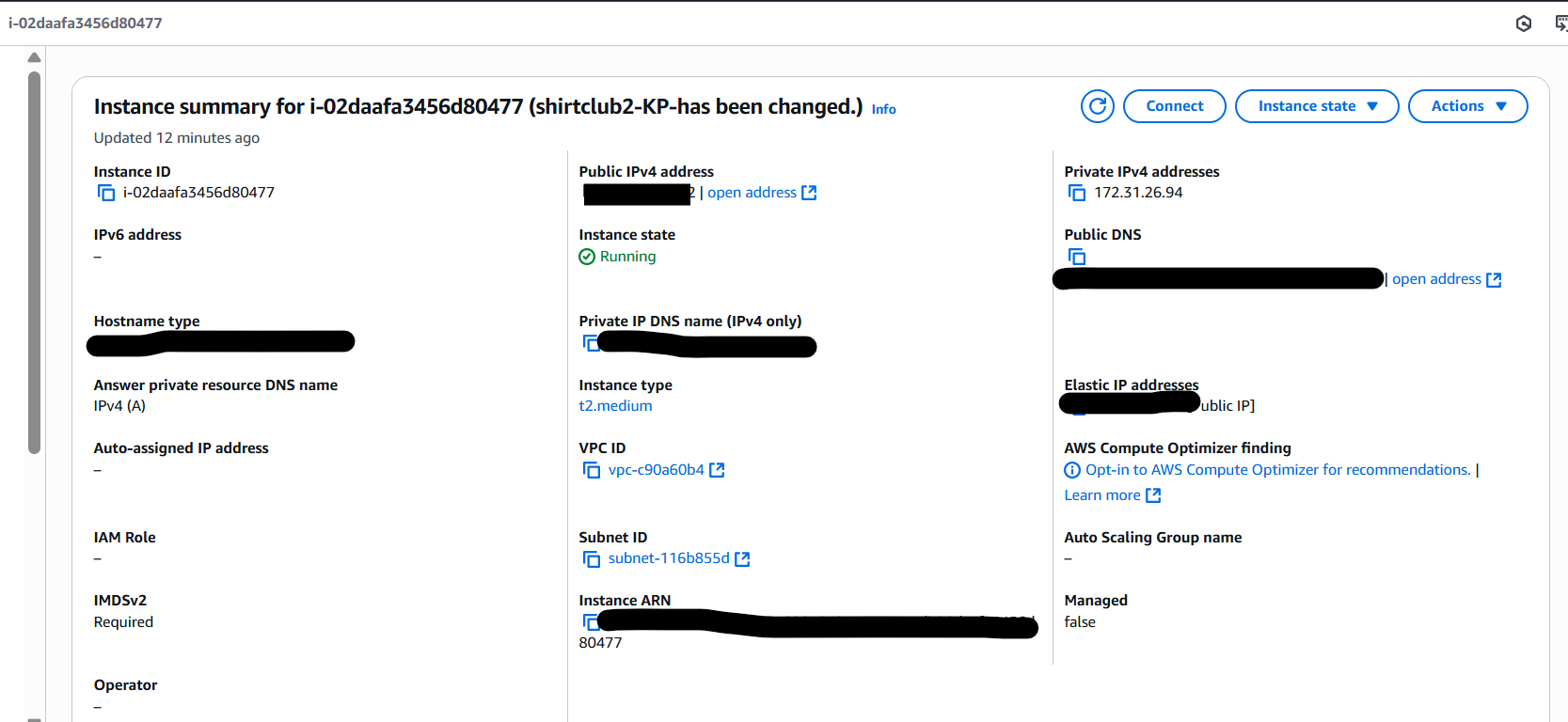
or by running the Android app in Android Studio or (ios app in Xcode) and testing the endpoint directly inside the application.

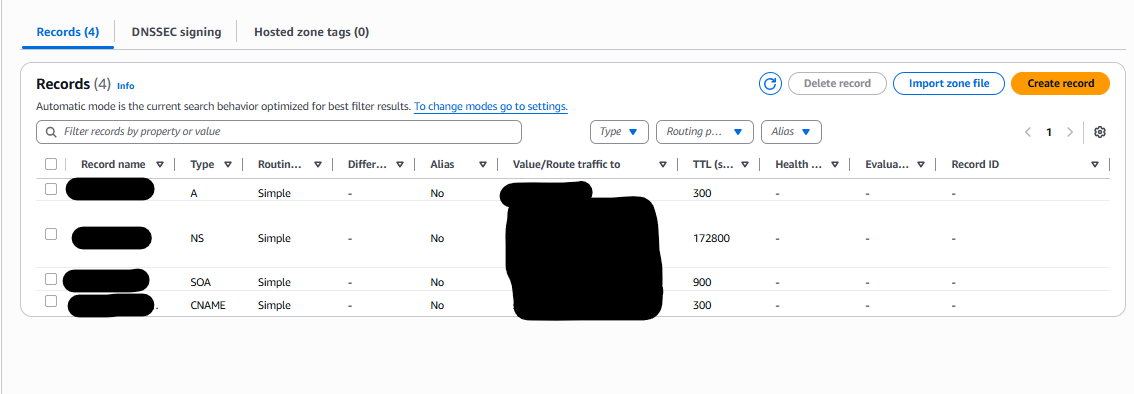


## Credentials & Access

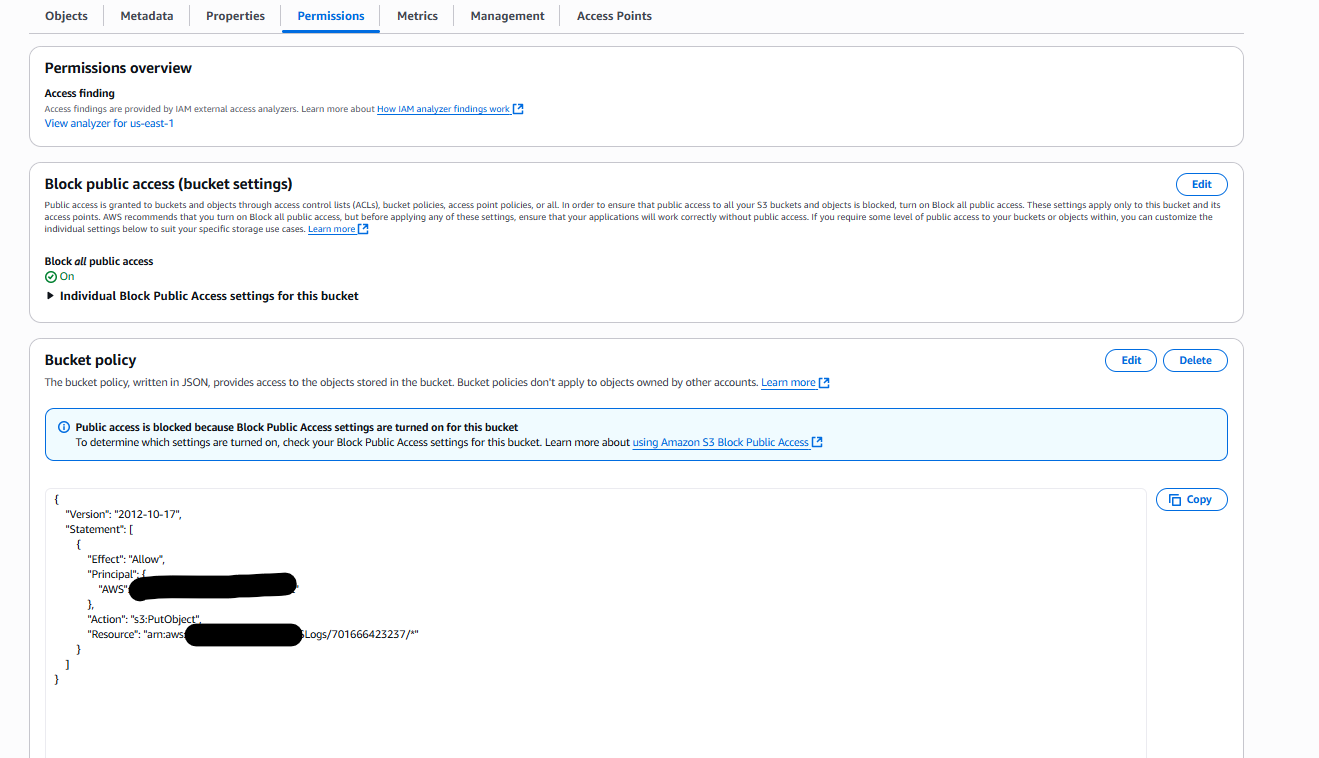
* We provided all credentials securely through a password manager.
* The ShirtClub client validated by:
  + Logging into their AWS root account.
  + Confirming the Cloudflare domain pointed to AWS Route53.

## AWS Infrastructure Validation

*  EC2 instances were reviewed together. The ShirtClub client checked:
  + Instance type was correctly sized.
  + Elastic IPs were attached to production servers.
  + SSH key pairs worked (the client tested access with ssh -i commands).
* In IAM, the client reset my passwords after the handover and confirmed that only the necessary roles remained active, each with limited permissions.
* For Route53, since ShirtClub used a domain management setup similar to your case, the client was able to review the DNS records in Route53 and confirm that everything was configured correctly



* For S3 buckets, public access was not required. Therefore, it was kept blocked, and we configured the appropriate bucket policy according to the client’s needs



## Local Environment Setup

* The ShirtClub client installed required tools (Xcode, Android Studio, Node.js, WAMP server for mysql database).
* They cloned each repo and followed the README instructions to:
  + Run the iOS app in Xcode simulator.
  + Run the Android app in Android emulator.
  + Launch the Website and Admin Panel locally with npm start.
* Using WAMP server, they connected to the database and visually confirmed the schema.

## Final Review & Sign-Off

* The client went through a handover readiness checklist with us: repos, credentials, AWS, local builds, and database access.
* Once every item was confirmed, they removed our collaborator access, rotated sensitive credentials, and backed up everything in their own systems.
* At this point, they were fully independent and confident in taking ownership of their project.

## Outcome

ShirtClub’s validation process shows how a client can successfully confirm ownership of all code, credentials, infrastructure, tools, and sign off only after every step is tested. You can use the same approach as a blueprint for your own handover.

# Conclusion

This guidance file is your complete reference for taking ownership of your project. By following it step by step, you will:

* Collect and secure all source code, credentials, and assets.
* Validate that your AWS infrastructure is fully under your control.
* Prepare your own local environment to test and run every application.
* Use proven best practices to avoid common risks and oversights.
* Benchmark your project against a real case study (ShirtClub) to understand what “good” validation looks like.

The goal of this document is independence. Once you complete these steps, you will no longer rely on developers to confirm ownership or explain what is running — you will know, test, and control it yourself.