Alivio Technology Pvt Ltd — Partner (Vendor/ Technician) Onboarding Roadmap

> Purpose: A complete, production-ready roadmap to build a vendor/technician self-onboarding portal and admin system. Frontend: **Next.js + Tailwind**. Backend: **Node.js + Express**. Database: **MongoDB** with **Prisma** as the ORM. Monorepo approach (pnpm/ turborepo). File storage: S3-compatible (AWS S3 / DigitalOcean Spaces). Payment gateway: Razorpay / Paytm / Stripe (choose by region).

Table of contents

Free OTP Service (SMS + Email) — Temporary Free Setup

For development/testing, you can use free or zero-cost OTP systems before integrating paid services.

Free SMS OTP Options (Temporary Only)

1. **Fast2SMS Trial** (offers limited free SMS credits)
2. **TextLocal Free Trial** (10–20 free SMS credits)
3. **Mock OTP System** (Recommended for development)

4. Generate OTP in backend
5. Send OTP via console log instead of SMS
6. Accept OTP for verification

Free Email OTP

Use **Nodemailer + Gmail SMTP (Free)** - Gmail allows ~100 emails/day in testing - Enable "App Password" in Gmail for secure SMTP usage

OTP Flow

1. Generate 6-digit OTP on backend
2. Save OTP in a temporary collection (expire in 5 minutes)
3. Send:
4. SMS → mock SMS (console log) or Fast2SMS trial
5. Email → using Gmail SMTP (Nodemailer)
6. Verify OTP → compare hashed OTP + expiry time

API Endpoints

- `POST /auth/send-otp`
- `POST /auth/verify-otp`

Example Backend Code Snippet (Node.js)

```
// send-otp.js
const otp = Math.floor(100000 + Math.random() * 900000);
console.log("OTP for", phone, "=", otp); // Free SMS (Mock mode)
await Otps.create({ phone, otp, expires: Date.now() + 5*60*1000 });

// send email otp using nodemailer
const transporter = nodemailer.createTransport({
  service: "gmail",
  auth: { user: process.env.GMAIL, pass: process.env.APP_PASSWORD }
});
await transporter.sendMail({
  to: email,
  subject: "Your OTP",
  html: `<h2>Your OTP is <b>${otp}</b></h2>`
});
```

---

1) Goals & success criteria

- Allow technicians/vendors to self-onboard with a multi-step form + document uploads.

- Enforce mandatory legal T&C acceptance and a non-refundable ₹200 payment before submission finalization.
- Admins can approve/reject, assign zone/batch, generate employee IDs, and offer kit/shirt purchases.
- Integrations for email + WhatsApp notifications and payment webhooks.
- Referral scheme: ₹500 paid on successful referral activation.
- All parts interconnected and auditable.

Success criteria: validated user registration, working payments, admin approval flow, delivery emails for kit purchases, referrals tracked & paid.

---

2) High-level user flows

Candidate (Vendor) flow

1. Landing page → Click **Become a Partner**.
2. Multi-step form:
3. Step 1: Basic info (Name, Phone, Address, Dist dropdown, Pin, DOB).
4. Step 2: Role info (Technician type select, Educational Qualification dropdown, Year of Experience).
5. Step 3: Bank details (Bank dropdown, IFSC, A/C No, Re-enter A/C No).
6. Step 4: Upload documents: Aadhaar, PAN, Edu Cert, Driving License, RC Book, Police Verification, Photo, Cancel Cheque.
7. Step 5: Terms & legal acceptance (checkbox + digital signature placeholder) — mandatory.
8. Step 6: Payment ₹200 (non-refundable). Use payment gateway.
9. On successful payment: generate `applicationId`, show success UI, send email & WhatsApp with application id and next steps.
10. Candidate waits for admin approval (status: Pending → Approved/Rejected).

Kit / T-shirt purchase during onboarding

- Optional add-ons shown on payment page. Each purchase triggers separate payment line items and post-payment emails with delivery ETA.

Referral

- Candidate enters referral code on signup. Admin verifies after successful activation; system awards ₹500 to referrer wallet when referred vendor completes onboarding and is approved/active.

Admin flow

- Admin dashboard: list of pending applications, filters by district/zone/type, quick actions: Approve / Reject.
- Approve flow: assign Zone, Zone Code, Batch; option to auto-generate Employee ID; optionally send WhatsApp & email; allow admin to mark kit/shirt purchase items and track purchase payment status.

---

3) Data model (collections + Prisma snippets)

> Note: Prisma supports MongoDB. Some features (transactions across collections) behave differently; plan accordingly.

Primary models

- `User` (auth & basic contact)
- `Application` (onboarding details)
- `Document` (file metadata)
- `Zone` (zone metadata)
- `Batch` (training batches)
- `Employee` (post-approval user record)
- `Payment` (gateway payments & line items)
- `Referral` (track referrals)
- `Order` (kit/shirt purchases)
- `AuditLog`

Example Prisma model (abridged)

```
generator client { provider = "prisma-client-js" }
datasource db { provider = "mongodb" url = env("DATABASE_URL") }

model User {
  id       String @id @default(auto()) @map("_id")
  email    String? @unique
  phone    String @unique
  password String?
  role     String // 'admin' | 'vendor' | 'employee'
  createdAt DateTime @default(now())
}

model Application {
  id            String @id @default(auto()) @map("_id")
  applicationId String @unique
  userId        String
  name          String
  phone         String
  address       String
  district      String
  pinCode       String
  technicianType String
  dob           DateTime
  education     String
  experienceYrs Int
  bankName      String
```

```
  bankIFSC      String
  bankAccount   String
  documents     Document[]
  referralCode  String?
  status        String @default("PENDING") // PENDING, APPROVED, REJECTED
  createdAt     DateTime @default(now())
}

model Document {
  id           String @id @default(auto()) @map("_id")
  applicationId String
  type         String
  url          String
  uploadedAt   DateTime @default(now())
}

model Payment {
  id             String @id @default(auto()) @map("_id")
  applicationId  String
  gateway        String
  amount         Float
  currency       String
  status         String
  gatewayPayload Json?
  createdAt      DateTime @default(now())
}

model Referral {
  id String @id @default(auto()) @map("_id")
  referrerUserId String
  referredPhone String
  applicationId String?
  rewardPaid Boolean @default(false)
  createdAt DateTime @default(now())
}
```

---

4) API endpoints (REST style)

Public / Auth

- `POST /api/auth/send-otp` — send OTP for phone login/signup
- `POST /api/auth/verify-otp` — verify OTP

Application (onboarding)

- `POST /api/application` — create partial application (save step progress)
- `GET /api/application/:id` — get application details
- `PUT /api/application/:id` — update (form progress)
- `POST /api/application/:id/upload` — upload document
- `POST /api/application/:id/submit` — final submit (requires T&C accepted & payment init)

Payment

- `POST /api/payments/init` — create order with gateway
- `POST /api/payments/webhook` — handle gateway webhook (payment status)
- `GET /api/payments/:id` — payment status

Admin

- `GET /api/admin/applications?status=pending` — list
- `POST /api/admin/applications/:id/approve` — approve (payload: zoneId, batchId, generateEmployeeId, assignKits)
- `POST /api/admin/applications/:id/reject` — reject with reason
- `POST /api/admin/orders/:id/mark-shipped` — mark kit/shirt shipped

Referral

- `POST /api/referral/apply` — attach referral at registration
- `POST /api/referral/settle` — admin trigger to pay referrer (or automate)

Webhooks

- `/webhook/payment`
- `/webhook/whatsappDelivery` (if provider supplies)

All admin routes protected by RBAC (JWT + role check).

---

5) Frontend architecture & pages/components

Stack: Next.js (app or pages router), Tailwind CSS, TypeScript, React Hook Form / Zod for validation.

Pages

- `/` — marketing/landing
- `/become-partner` — onboarding wizard container
- `/become-partner/step/[1..6]` — step pages or subcomponents
- `/application/status/[applicationId]` — status page
- `/admin/*` — admin dashboard (protected)

Key components

- `WizardForm` (handles step state, autosave)
- `Input`, `Select`, `FileUpload`, `DatePicker`, `BankAccountInput` (validate IFSC format)
- `DocumentUploader` (preview + progress)
- `TermsModal`, `DigitalSignature` placeholder
- `PaymentWidget` (Razorpay/Stripe integration)
- `AdminTable`, `ApplicationDetail`, `AssignZoneModal`

UX details: - Autosave every step (localStorage + backend draft `POST /application`) - Show progress bar and validation per step - Disable final submit until all mandatory docs present and T&C accepted

---

6) Monorepo layout & dev tooling

Use `pnpm` workspaces or `turborepo`:

```
/alivio-monorepo
  /apps
    /web      (Next.js frontend)
    /admin    (Next.js admin or same web with route protection)
    /api      (Express app)
  /packages
    /ui       (shared React components)
    /lib      (shared types, prisma client wrapper)
    /config   (eslint, tailwind)
  package.json
  pnpm-workspace.yaml
```

Dev tooling: ESLint, Prettier, Husky (pre-commit), lint-staged, TypeScript, Vitest/Jest.

---

7) File uploads & document handling

- Store files in S3-compatible storage; keep only metadata in DB.
- Use presigned URLs for direct upload from client (reduces server cost & memory usage).
- On upload completion, client notifies backend with file key to attach to application.
- Validate file types, sizes, and run basic malware scan (ClamAV or external service) on server side if possible.
- Keep audit trail of uploads (who/when).

Document naming: `applications/{applicationId}/{documentType}_{timestamp}.{ext}`

---

8) Payment flow

- Use gateway SDK to generate an order from backend (`POST /payments/init`) including `applicationId` and amount (₹200 + line items for kits).
- Return order info to client to open payment modal.
- Gateway calls webhook to `/webhook/payment` after payment success/failure — verify signature, update `Payment` record, update `Application.status` if payment is the last required action.
- On successful payment:
- create `Payment` record
- generate `applicationId` (if not already)
- send confirmation email & WhatsApp

Kit purchases: treat each kit as a separate `Order` with line items and triggers for shipping flow and automated emails.

---

9) Admin workflows & automation

- Admin UI shows pending apps. Approve action will:
- Change application status to `APPROVED` and create `Employee` record.
- Generate `employeeId` (pattern `ALV-<ZoneCode>-<sequential>`).
- Optionally assign to a `Batch` and schedule training.
- Trigger welcome messages (email & WA) with Employee ID and next steps.
- If kit/shirt purchased: generate order, mark payment as received, and send order confirmation + delivery ETA.

Automation hooks: - onPaymentSuccess (payment webhook) → send receipts, mark orders, trigger notification - onApproval → send welcome messages & set `active:true`

---

10) Referral program

- On signup, user can supply `referralCode` (mapped to `User.id` or `referrerCode`).
- Referral condition: when referred application reaches `APPROVED` and `active` (post-training), set `Referral.rewardPaid=true` and enqueue payout.
- Keep referral wallet in `User` record or external payout system.
- Admin can view pending referral payouts and mark them settled.

---

11) Security, validation & compliance

- Auth: OTP or email+password. Use short-lived JWTs with refresh tokens. Store refresh tokens hashed.
- Sensitive data: encrypt bank account numbers at rest (e.g., use application-level encryption or KMS).
- PCI: avoid storing card data. Use gateway tokens.
- Input validation: Zod / Joi server-side + React Hook Form + Zod client-side.

- Rate limiting: express-rate-limit on sensitive endpoints.
- Helmet, CORS, CSRF protections on web endpoints.
- Audit logs for approvals, payments, and document uploads.
- GDPR/Local privacy: dataset retention policy (delete drafts after X months).

---

12) Email & WhatsApp automations

- Email provider: SendGrid / Mailgun / SES.
- WhatsApp: Meta WhatsApp Cloud API or Twilio WhatsApp Business API for messaging.

Automated messages: - OTPs - Payment success receipts & invoice - Application received with `applicationId` - Application approved: include Employee ID, Zone, Batch, training schedule - Kit purchase: payment success + delivery ETA - Referral payout notified

Design templates as transactional templates in provider; keep placeholders for personalization.

---

13) Testing strategy

- Unit tests for critical business logic (payment handling, referral settlement, ID generation).
- Integration tests for API flows (supertest / jest).
- E2E tests for onboarding flow (Cypress / Playwright).
- Manual QA checklist for document uploads and payment reconciliation.

---

14) CI / CD & deployment

- Use GitHub Actions (or GitLab CI) pipelines for lint→test→build→deploy.
- Frontend: Deploy Next.js to Vercel (or Netlify). Use preview deployments for PRs.
- Backend: Deploy Express via container (ECS/Fargate / DigitalOcean App Platform / Heroku) or serverless functions (if you prefer). Keep a single production DB.
- Use environment secrets (GitHub Secrets / Vault). Manage Prisma Migrations carefully for MongoDB (Prisma has `db push` style workflows).
- Scheduled job worker (BullMQ/Redis or native queue) for asynchronous tasks (sending emails, settling referral payouts, shipping updates).

---

15) Observability & monitoring

- Centralized logs (Winston → Papertrail / Datadog / LogDNA).
- Error tracking: Sentry.
- Health checks for API and queue workers.
- Metrics: Prometheus / Grafana or cloud provider metrics.

---

16) Phased rollout (MVP → v1 → v2)

MVP (must-have): - Multi-step form with fields and basic validation - Document upload (S3 presigned urls) - Terms & conditions acceptance - ₹200 payment flow (gateway integration) + webhook - Application generation and email/whatsapp notification - Admin: list pending apps + approve/reject + assign zone + generate employee id - Referral code capture & basic tracking

v1 (important enhancements): - Kit & T-shirt purchase flow with order management and shipping notifications - Autosave drafts & resume - Secure bank data encryption - Robust RBAC and audit logs

v2 (nice-to-have): - Training scheduling calendar & calendar invites - Mobile friendly admin app - Bulk import/export of applications - Referral payouts automation via payouts API (bank/UPI/third-party) - Analytics dashboards (acceptance rate, time-to-approve, revenue)

---

17) Checklist & next steps (developer handoff)

- [ ] Create monorepo skeleton (pnpm + turborepo)
- [ ] Implement Prisma models and generate client
- [ ] Implement auth (OTP) and basic user model
- [ ] Build onboarding wizard UI + autosave
- [ ] Implement S3 presigned upload flow + document metadata
- [ ] Integrate payment gateway & implement webhook handler
- [ ] Build admin dashboard with approve/reject flows
- [ ] Integrate email & WhatsApp notifications
- [ ] Implement referral tracking and admin dashboard
- [ ] Write unit & integration tests for payment and approval flows
- [ ] Configure CI/CD and production deployment

Appendix: Example Employee ID generation logic (pseudo)

- Pattern: `ALV-{ZoneCode}-{YYYY}{seq}`
- On approval: find or create counter for zone & year (atomic increment), then format ID.

```
// pseudocode
const nextSeq = await counters.findOneAndUpdate({ zone, year }, { $inc: { seq:
1 } }, { upsert:true, returnNew:true });
const empId = `ALV-${zoneCode}-${new Date().getFullYear()}$
{String(nextSeq.seq).padStart(4,'0')}`;
```

If you want, I can now: - produce a **detailed DB schema** in Prisma file for the full domain, - create **OpenAPI** spec for all endpoints, - scaffold a monorepo skeleton (file tree + basic code) using Next.js + Express + Prisma.

Tell me which of those you want next and I will scaffold it in the repository structure of the roadmap.