HR Recruitment & Employee Lifecycle Management System — Cursor-ready Step-by-Step

Stack chosen (recommended) - **Backend:** Node.js (TypeScript) + Express (or Next.js API routes) - **ORM:** Prisma - **Database:** PostgreSQL - **Frontend:** Next.js (React) + Tailwind CSS - **Auth:** NextAuth.js (or your preferred auth provider) - **File storage:** AWS S3 (recommended) or local storage for dev - **PDF generation:** Puppeteer (server-side HTML -> PDF) - **Resume viewer:** PDF.js (for PDF), convert DOCX -> PDF server-side if needed

> I picked this stack because it pairs well with Cursor, is productivity-friendly, and scales. If you prefer Django/DRF or Rails, tell me and I'll re-split prompts for that stack.

---

1 — High level architecture & workflow

- **Requisition** created by a Manager/HR → `requisition` table
- Candidates apply (bulk / individual) and are attached to a requisition → `candidates` table
- Candidate enters **Screening** stage with scoring and comments → `screenings` table
- If **Shortlisted** → move to **Interview**
- If **Rejected** → stays in Screening (terminal)
- **Interview** stage with scoring, comments, decision → `interviews` table
- If **Shortlisted** → move to **Offer**
- If **Hold / Rejected** → stay in Interview (terminal)
- **Offer** stage → auto-generate offer letter and store offer PDF → `offers` table
- **Onboarding** stage with checklist and document verification → `onboardings` table
- **Resignation** tab for employees → `resignations` table

All transitions are enforced server-side (only `SHORTLISTED` triggers movement forward).

---

2 — Database schema (Postgres / Prisma-ready)

Below is the relational schema. I include recommended column names and types. Use UUIDs for primary keys.

```
-- users (managers, hr, interviewers, employees)
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name TEXT NOT NULL,
  email TEXT UNIQUE NOT NULL,
  role TEXT NOT NULL, -- ENUM('ADMIN','HR','MANAGER','INTERVIEWER','EMPLOYEE')
```

```
  password_hash TEXT,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

CREATE TABLE requisitions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  start_date DATE,
  end_date DATE,
  manager_name TEXT,
  manager_id UUID REFERENCES users(id) ON DELETE SET NULL,
  position_title TEXT NOT NULL,
  job_description TEXT,
  number_of_openings INT DEFAULT 1,
  status TEXT NOT NULL DEFAULT 'OPEN', -- ENUM('OPEN','CLOSED')
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

CREATE TABLE candidates (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  requisition_id UUID REFERENCES requisitions(id) ON DELETE CASCADE,
  first_name TEXT,
  last_name TEXT,
  email TEXT,
  phone TEXT,
  resume_url TEXT,
  resume_filename TEXT,
  resume_mimetype TEXT,
  applied_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  status TEXT NOT NULL DEFAULT 'APPLIED', --
ENUM('APPLIED','SCREENING','INTERVIEW','OFFER','HIRED','ONBOARD','RESIGNED','REJECTED')
  current_stage TEXT,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

CREATE TABLE screenings (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  candidate_id UUID REFERENCES candidates(id) ON DELETE CASCADE,
  reviewer_id UUID REFERENCES users(id) ON DELETE SET NULL,
  scores JSONB, -- { "communication": 4, "skills": 3 }
  comments JSONB, -- { "communication": "good", "skills": "lacking" }
  decision TEXT NOT NULL, -- ENUM('SHORTLISTED','REJECTED')
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

CREATE TABLE interviews (
```

```
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  candidate_id UUID REFERENCES candidates(id) ON DELETE CASCADE,
  interviewer_id UUID REFERENCES users(id) ON DELETE SET NULL,
  scheduled_at TIMESTAMP WITH TIME ZONE,
  scores JSONB,
  comments JSONB,
  decision TEXT NOT NULL, -- ENUM('SHORTLISTED','HOLD','REJECTED')
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

CREATE TABLE offers (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  candidate_id UUID REFERENCES candidates(id) ON DELETE CASCADE,
  offered_by UUID REFERENCES users(id) ON DELETE SET NULL,
  offer_details JSONB, -- salary, benefits, ctc breakdown
  date_of_joining DATE,
  offer_pdf_url TEXT,
  status TEXT DEFAULT 'SENT', -- ENUM('SENT','ACCEPTED','REJECTED','EXPIRED')
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

CREATE TABLE onboardings (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  candidate_id UUID REFERENCES candidates(id) ON DELETE CASCADE,
  tasks JSONB, -- [{"task":"upload id", "done":true, "uploaded_url":"..."}]
  documents JSONB,
  progress INT DEFAULT 0,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

CREATE TABLE resignations (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  employee_id UUID REFERENCES users(id) ON DELETE SET NULL,
  resignation_date DATE,
  exit_interview_notes TEXT,
  checklist JSONB,
  final_settlement_amount NUMERIC(12,2),
  status TEXT DEFAULT 'PENDING', -- ENUM('PENDING','COMPLETED')
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

CREATE TABLE attachments (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  owner_type TEXT, -- 'candidate'|'offer'|'onboarding' etc
  owner_id UUID,
  url TEXT,
  filename TEXT,
  mimetype TEXT,
```

```
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

> Tip: `scores` and `comments` as JSONB lets you add / change criteria without new migrations. If you prefer typed columns for each score criterion, add them in the `screenings` and `interviews` tables.

---

3 — API design (REST style)

Auth: `/api/auth/*`

Requisitions - `GET /api/requisitions` — list (filter by status, manager) - `POST /api/requisitions` — create - `GET /api/requisitions/:id` — detail - `PUT /api/requisitions/:id` — update - `POST /api/requisitions/:id/close` — close job

Candidates - `GET /api/requisitions/:reqId/candidates` — list - `POST /api/requisitions/:reqId/candidates` — upload single (multipart/form-data) - `POST /api/requisitions/:reqId/candidates/bulk` — bulk upload (CSV + ZIP of resumes) - `GET /api/candidates/:id` — detail - `GET /api/candidates/:id/resume` — redirect to signed URL or stream file

Screenings - `POST /api/candidates/:id/screenings` — submit screening - `GET /api/candidates/:id/screenings` — list

Interviews - `POST /api/candidates/:id/interviews` — submit interview result - `GET /api/candidates/:id/interviews` — list

Offers - `POST /api/candidates/:id/offers` — create offer and generate PDF (server generates and stores PDF) - `GET /api/offers/:id` — details (download PDF)

Onboarding - `POST /api/candidates/:id/onboarding` — create/update onboarding checklist

Resignations - `POST /api/employees/:id/resignation` — create - `GET /api/resignations` — list

Status/Workflow - `GET /api/candidates/:id/status` — current status and timeline

---

4 — Frontend screens & components

1. **Dashboard / Requisition List** — show open/closed, counts, quick-create
2. **Requisition Create/Edit Modal** — fields from your spec + Close Job button

3. **Requisition Detail** — list of candidates attached
4. **Candidates List (per requisition)** — bulk upload + add candidate button
5. **Candidate Card / Detail** — basic info, resume viewer (PDF.js), timeline of actions (screening, interview, offers)
6. **Screening Form** — dynamic scoring fields, comments, radio buttons (Shortlist / Reject)
7. **Interview Form** — similar to screening, but radio options (Shortlist / Hold / Reject)
8. **Offer Creation Page** — form for offer details + preview offer letter + Generate PDF button
9. **Onboarding Page** — checklist UI (toggle complete), upload docs
10. **Resignation Tab** — for employees, with exit checklist
11. **Admin Settings** — manage scoring criteria, templates

UX notes: show **status chips** and timeline in candidate detail. Disable forward buttons unless server returns success and candidate status updates.

---

5 — Business rules & server validations (enforced server-side)

- Screening POST: if decision == `SHORTLISTED` → update candidate.status to `INTERVIEW` and current_stage to `INTERVIEW`.
- Screening POST: if `REJECTED` → candidate.status = `REJECTED` (or remains `SCREENING` if you want to keep it)
- Interview POST: if decision == `SHORTLISTED` → candidate.status = `OFFER` and current_stage = `OFFER`
- Interview POST: if decision == `HOLD` or `REJECTED` → candidate.status unchanged or set to `INTERVIEW` / `REJECTED`
- Offer creation: set offer.status = `SENT`. When candidate accepts (API call or signed offer), set offer.status = `ACCEPTED` and candidate.status = `HIRED` / create `onboarding` row.
- Only allow transitions that follow the pipeline. Reject attempts to jump stages.

---

6 — File handling & resume viewer

- For dev: accept file uploads (multipart), store in `uploads/` and save attachment record. For prod: generate S3 presigned upload URLs (PUT) and store S3 key.
- For resume viewer: prefer PDF. If user uploads DOCX, convert server-side (LibreOffice headless or `mammoth` to HTML then render PDF) and store PDF version too.
- Use `pdf.js` on frontend to render PDF from signed URL.

---

7 — Offer letter generation

- Build an HTML template with placeholders: `{{first_name}}`, `{{position_title}}`, `{{salary}}`, `{{doj}}`, etc.
- Server endpoint: `POST /api/candidates/:id/offers` receives offer details, renders HTML template with data (e.g., with Handlebars), launches Puppeteer to print to PDF, stores PDF in S3 (or server) and saves `offer_pdf_url`.

Example offer template placeholder snippet (server-side):

```
<html>
  <body>
    <h1>Offer Letter</h1>
    <p>Dear {{first_name}} {{last_name}},</p>
    <p>We are pleased to offer you the position of {{position_title}} at
{{company_name}}.</p>
    <p>Joining Date: {{date_of_joining}}</p>
    <p>CTC: {{ctc}}</p>
    <p>Sincerely,</p>
    <p>{{offered_by_name}}</p>
  </body>
</html>
```

---

8 — Testing & seed data

- Create seed scripts for: 2-3 requisitions, 10 candidates (mix of statuses), a few screenings, interviews and one offer.
- Unit test server validations (transition guards) and integration tests for upload + generate offer.

---

9 — Cursor prompts — split and ready to paste (follow these in order)

Each prompt below is written so you can paste it into Cursor step-by-step. After each prompt, commit, run migrations, and verify before moving to the next.

Prompt 1 — Initialize repository & dependencies

```
Create a new TypeScript Node + Next.js project named `hr-ems`. Install the
following dependencies and devDependencies and initialize a git repo.
```

```
Dependencies:
- next react react-dom
- prisma @prisma/client
- express (if using separate server) or use Next.js API routes
- pg
- aws-sdk (or @aws-sdk/* v3)
- multer (for dev file uploads)
- jsonwebtoken / next-auth (for auth)
- puppeteer
- pdfjs-dist (for client viewer usage)

DevDependencies:
- typescript ts-node
- eslint prettier
- jest / vitest (testing)
- @types/node @types/react etc

Create a `README.md` with commands: `dev`, `build`, `start`, `prisma:migrate`,
`prisma:generate`.

Make the initial commit.
```

Prompt 2 — Add Prisma schema (paste this)

```
Create `prisma/schema.prisma` with a PostgreSQL datasource and the models
corresponding to the SQL schema provided earlier. Use UUIDs for IDs. Make sure
to add `@@map` if you want different table names. Run `npx prisma migrate dev --
name init` and `npx prisma generate`.

Paste in the prisma models equivalent of the SQL schema (requisition, candidate,
screening, interview, offer, onboarding, resignation, user, attachment) and
create the migration.
```

(Note: in Cursor paste the actual Prisma schema — see example below if you want it prewritten. If you want, I can paste the Prisma schema in the next step.)

Prompt 3 — Create basic auth and user model

```
Implement authentication with NextAuth.js (or JWT-based). Create user seeding
script with roles ADMIN, HR, MANAGER, INTERVIEWER, EMPLOYEE. Protect API routes
so that only authorized roles can perform actions (e.g., only HR/Manager can
create requisitions, only Interviewer/HR can submit screenings/interviews).
```

```
Make one admin user via the seed script and commit.
```

Prompt 4 — Requisition APIs + UI

```
Create backend endpoints for Requisition CRUD following the API design.
Implement `POST /api/requisitions/:id/close` which sets `status = 'CLOSED'`.
Create a React page `/requisitions` that lists requisitions and a modal form `/
requisitions/new` to create one. Use Tailwind for styling. Add client-side
validation and server-side checks.

Commit this feature and run it locally.
```

Prompt 5 — Candidate upload (single and bulk)

```
Implement candidate upload endpoints:
- Single upload: `POST /api/requisitions/:reqId/candidates` accepting multipart/
form-data: `firstName`, `lastName`, `email`, `phone`, `resume` (file). Save
resume to local `uploads/` folder in dev and store metadata in `candidates` and
`attachments`.
- Bulk upload: `POST /api/requisitions/:reqId/candidates/bulk` should accept a
CSV (rows -> firstName,lastName,email,phone,resumeFilename) plus ZIP file of
resumes. Implement CSV parsing and ZIP extraction (use `adm-zip`), match
filenames, and create candidate rows.

On the frontend: a Candidate list page with `Add Candidate` modal and `Bulk
Upload` modal. Show upload progress and success errors.

Commit and test with sample files.
```

Prompt 6 — Resume viewer and conversion

```
Add resume viewing functionality in the candidate detail page. If the resume is
PDF, render via `pdf.js` in the page. If the upload is docx, create a server-
side conversion step to convert to PDF using LibreOffice headless (or `mammoth`
-> HTML -> Puppeteer PDF). Return the PDF URL and render.

Add a `View Resume` button on candidate cards and candidate detail.
```

Prompt 7 — Screening API & UI + business rules

```
Create POST `api/candidates/:id/screenings` which accepts JSON: { reviewerId,
scores: {criteria1: number, criteria2: number}, comments: {criteria1:"..."},
decision: 'SHORTLISTED'|'REJECTED' }

Server logic:
- Validate reviewer role.
- Insert into `screenings` table.
- If `decision === 'SHORTLISTED'`, update candidate.status = 'INTERVIEW' and
candidate.current_stage = 'INTERVIEW'.

On frontend: candidate detail page should show `Start Screening` action if
candidate.status is `APPLIED` or `SCREENING`. Implement screening form with
dynamic criteria (for now send two sample criteria). Commit and test.
```

Prompt 8 — Interview API & UI + business rules

```
Create POST `api/candidates/:id/interviews` with payload similar to screening
but decision is 'SHORTLISTED'|'HOLD'|'REJECTED'. Server logic:
- Save interview result.
- If `SHORTLISTED` → update candidate.status = 'OFFER' and
candidate.current_stage = 'OFFER'.
- If `HOLD` or `REJECTED` → do not move forward (status keep as `INTERVIEW` or
`REJECTED`).

Frontend: Interview form on candidate detail; show interviewer, schedule fields.
Commit and test.
```

Prompt 9 — Offer generation endpoint + template

```
Create POST `api/candidates/:id/offers` that accepts { offeredBy, offerDetails:
{ctc, salaryBreakup, benefits}, dateOfJoining }.

Server responsibilities:
1. Validate candidate is in `OFFER` status.
2. Populate HTML template with candidate + offer data.
3. Use Puppeteer to render HTML to PDF.
4. Save PDF to S3/local and store `offer_pdf_url` in `offers` table with status
'SENT'.

Frontend: Offer creation screen with preview. Add `Download Offer` action after
generation.
```

```
Commit and test generation for 1 candidate.
```

Prompt 10 — Onboarding checklist implementation

```
Create endpoints to create/update onboarding checklist for the candidate who
accepted the offer. The onboarding `tasks` is JSON array of tasks with `task`,
`done`, `uploadedUrl`.

Frontend: Onboarding page should show checklist UI (checkbox + upload per task)
and progress bar.

When all required documents are verified, mark candidate.status = 'ONBOARD' or
'HIRED' depending on your naming.
```

Prompt 11 — Resignation flow

```
Create employee resignation endpoints (`POST /api/employees/:id/resignation`)
which accept resignation_date, exit_interview_notes, checklist,
final_settlement_amount. Save record and provide admin UI to process final
settlement and close checklist.

Frontend: Resignation tab under employee profile with checklist and status.
```

Prompt 12 — Notifications & timeline

```
Add a `candidate_activity` table (or event logs) to store timeline events:
'Applied', 'Screened', 'Interviewed', 'OfferSent', 'OfferAccepted',
'OnboardCompleted', 'Resigned'. Write server code to create events whenever a
stage transition occurs.

Frontend: Display timeline on candidate detail and show toast notifications for
important events (offer sent/accepted).
```

Prompt 13 — Authz, roles and permissions enforcement

```
Add middleware or API guards so only allowed roles can perform actions.
Examples:
- Create requisition: HR/MANAGER/ADMIN
- Submit screening: HR/INTERVIEWER/ADMIN
- Submit interview: INTERVIEWER/HR/ADMIN
```

```
- Create offer: HR/ADMIN

Add role-aware UI (hide actions user is not allowed to do).
```

Prompt 14 — Search, filters, and CSV export

```
Add server-side filters for candidate lists by status, requisition, date range,
and full-text search on name/email. Add CSV export endpoint for candidate lists
and requisition reports.
```

Prompt 15 — Testing, seeding, and QA

```
Write tests for:
- Requisition create/close
- Candidate upload (single & bulk) and resume viewing
- Screening and Interview transitions (guarded)
- Offer generation (end-to-end: create -> generate PDF)

Create seed data for 3 requisitions and 10 candidates with mixed statuses.
```

Prompt 16 — Docker & deployment

```
Create Dockerfile(s) for the app and a docker-compose.yml with Postgres, Prisma
migrations, and the app. Add a production checklist and instructions to deploy
to Vercel (frontend) + Fly/Heroku/DigitalOcean (backend + Postgres + S3). Ensure
environment variables and secrets management are documented.
```

Prompt 17 — Final QA checklist to run before launch

```
1. Test all transitions with role-based users.
2. Upload and view 10 different resume types (pdf, docx, doc).
3. Generate 5 offer letters and download PDFs.
4. Complete onboarding for a candidate and mark progress.
5. Create resignation and complete final settlement flow.
6. Run load test on candidate bulk upload.
7. Backup DB and snapshot S3 storage.
```

---

10 — Helpful small scripts & examples (paste into Cursor if you want)

Prisma model (short sample)

```
model Requisition {
  id              String   @id @default(uuid())
  startDate       DateTime?
  endDate         DateTime?
  managerName     String?
  managerId       String?  @db.Uuid
  positionTitle   String
  jobDescription  String?
  numberOfOpenings Int     @default(1)
  status          String   @default("OPEN")
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt
  candidates      Candidate[]
}

model Candidate {
  id            String   @id @default(uuid())
  requisition   Requisition? @relation(fields: [requisitionId], references:
[id])
  requisitionId String?
  firstName     String?
  lastName      String?
  email         String?
  phone         String?
  resumeUrl     String?
  appliedAt     DateTime @default(now())
  status        String   @default("APPLIED")
  screenings    Screening[]
  interviews    Interview[]
  offers        Offer[]
}
```

Offer generation (server snippet idea)

```
1. Render Handlebars template to HTML with candidate + offer data.
2. Launch Puppeteer in headless mode: const pdf = await page.pdf({ format:
'A4' });
3. Upload `pdf` buffer to S3 and store url.
```

Wrap-up & next steps

I've split the work so you can paste each prompt into Cursor one-by-one and ship incrementally. Follow the prompts in order and run the migrations / tests after each major commit. If you want, I can:

- Paste exact Prisma schema text into the next step.
- Produce a prefilled Handlebars offer template.
- Provide a ready-to-run Docker Compose sample.

Tell me which of those you'd like next and I'll add it as a separate prompt block for Cursor.