**Phase 1: Resource Modelling**

**Initial Resources:**

* Users
* Job Postings,
* Workers Profiles,
* Applications,
* Reviews

**Additional Resources:**

1. PaymentTransactions
2. Skills
3. Chat

**Justifications for Additional Resources:**

1. Payment Transactions

* Provides evidence in case of payment-related conflicts between clients and workers.
* Enable monetisation features like platform commissions

1. Skills:

* Clients can find workers faster by filtering applicants by skill
* Provide a structured list of skills, thus preventing inconsistencies and redundancies (e.g, “electrician”, “electrician work”, “electrical tech”)

1. Conversation chat

* Clients and workers need a way to discuss job details, timelines, pricing, and expectations.
* In-app messaging keeps users engaged within the platform instead of switching to external tools like WhatsApp.

**Resource Attribute Design**Below is a table of attributes for each resource, classified properly and types

| Attribute | Type | Required | source | Notes |
| --- | --- | --- | --- | --- |
| Id | UUID | yes | System | Unique,auto-generated |
| Name | String | yes | user | Min 2, max 100 characters |
| email | String | yes | user | Valid email format |
| phone | String | optional | user | E.164 format |
| password | String | yes | user | Min 8 characters, hashed |
| role | Enum | yes | user | Client, worker |
| profile\_pic | String(URL) | optional | user | - |
| created\_at | DateTime | yes | system | ISO 8601 format |
| update \_at | DateTime | yes | system | ISO 8601 format |

**Job Postings**

| Attribute | Type | Required | Source | Notes |
| --- | --- | --- | --- | --- |
| Id | UUID | Yes | System | Auto-generated |
| Client\_Id | UUID | Yes | User | FK to users |
| title | String | Yes | User | Max 150 chars |
| description | Text | Yes | User | Max 200 chars |
| Category | String | Yes | User | Predefined list |
| Location | String | Yes | user | Max 150 |
| budget | Decimal | Yes | User | Min 0 |
| Created\_at | DateTime | Yes | System | Iso 8601 format |

**Worker Profiles**

| Attribute | Type | Required | Source | Notes |
| --- | --- | --- | --- | --- |
| Id | UUID | Yes | System | Auto-generated |
| User\_id | UUID | Yes | User/system | FK to usera |
| bio | String | Optional | User | Max 500 chars |
| Skills | List<UUID> | Yes | user | FK to skills |

**Applications**

| Attribute | Type | Required | Source | Notes |
| --- | --- | --- | --- | --- |
| Id | UUID | Yes | System | Auto-generated |
| worker\_id | UUID | Yes | System/user | Fk to worker Profile |
| job\_id | UUID | Yes | System/User | FK to job posting |
| Cover\_letter | Text | Optional | User | Max 1000 chars |
| Status | Enum | Yes | System | Pending, Accepted, Rejected |
| Applied\_at | DateTime | Yes | System | Iso 8601 |

**Reviews**

| Attribute | Type | Required | source | Notes |
| --- | --- | --- | --- | --- |
| Id | UUID | Yes | system | Auto-generated |
| author\_id | UUID | Yes | User | Fk to Users |
| target\_id | UUID | Yes | User | FK to Users |
| rating | integer | Yes | User | 1-5 |
| Comment | Text | Optional | User | Max 1000 chars |
| Created | DateTime | Yes | System | **-** |

**Payments**

| Attribute | Type | Required | Source | Notes |
| --- | --- | --- | --- | --- |
| Id | UUID | Yes | System | Auto-generated |
| Sender\_Id | UUID | Yes | System | FK to Users |
| Reciever\_Id | UUID | Yes | System | Fk to users |
| Method | String | Yes | User | “Mobile\_money”, “bank\_transfer”, etc |
| Amount | Decimal | Yes | System/User | >0 |
| Currency | String | Yes | System | Iso 4217 |
| Status | Enum | Yes | System | Pending, completed, failed |
| Created\_at | DataTime | Yes | System | - |

**Skills**

| Attribute | Type | Required | Source | Notes |
| --- | --- | --- | --- | --- |
| Id | UUID | Yes | System | Auto-generated |
| Name | String | Yes | Admin/User | Unique |

**Conversation**

| Attribute | Type | Required | Source | Notes |
| --- | --- | --- | --- | --- |
| Id | UUID | Yes | System | System Generated |
| Created\_at | DateTime | Yes | System | - |

Tables related to Conversation:

* **ConversationParticipant**

| **Attribute** | **Type** | **Required** | **Source** | **Notes** |
| --- | --- | --- | --- | --- |
| Id | UUID | yes | System | System generated |
| conversation\_id | UUID | yes | System | FK to Conversation |
| user\_id | UUID | yes | System | FK to Users |

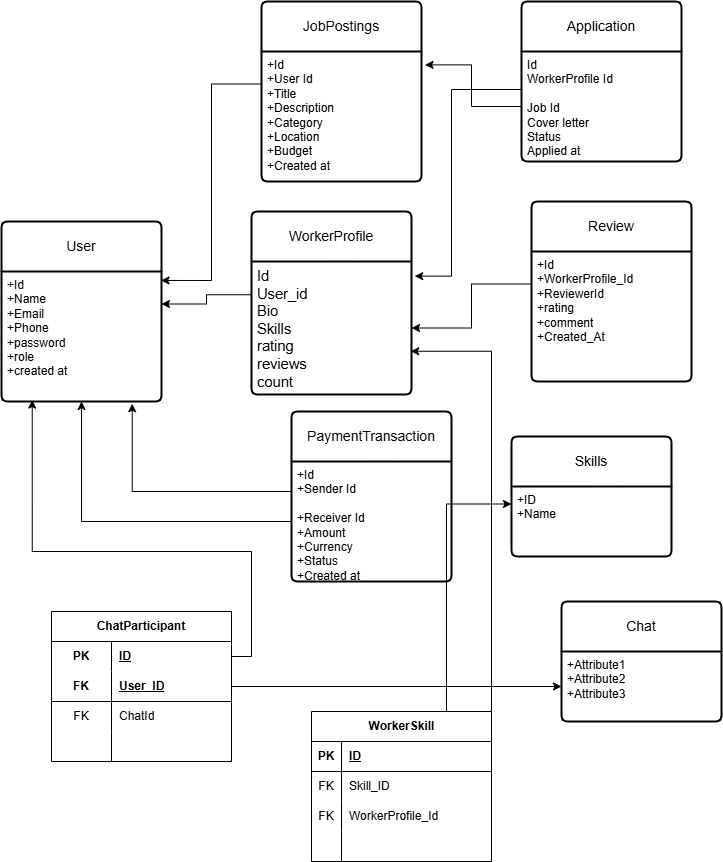
* **Message**

| **Attribute** | **Type** | **Required** | **Source** | **Notes** |
| --- | --- | --- | --- | --- |
| Id | UUID | yes | System | **-** |
| conversation\_Id | UUID | yes | System | FK to conversation |
| Sender\_ID | UUID | yes | System | FK to users |
| content | String | yes | User | **-** |
| created\_at | DateTime | yes | System | **-** |
| read\_at | DateTime | yes | System | **-** |

**Resource Relationships Summary**

| **Resource A** | **Resource B** | **Cardinality** | **Relationship Type** | **Notes** |
| --- | --- | --- | --- | --- |
| User | Worker Profile | 1:1 | Parent-Child | Each user has a single profile |
| Worker Profile | Skill | N:M | Peer | Shared skill tag |
| User | Job Posting | 1:N | Parent-Child | A user can post many jobs |
| JobPosting | Application | 1:N | Parent-Child | Jobs receive many applications |
| WorkerProfile | Application | 1:N | Parent-Child | Worker can apply to many jobs |
| WorkerProfile | Review | 1:N | Parent-Child | Workers receive many reviews |
| User | PaymentTransaction | 1:N | Peer | Users involved in many transactions |
| User | Chat | N:M | Peer | Users message each other |

**ERD Diagram:**

****

# **Phase 2: API Design Excellence**

### Endpoint Design (Leg by Endpoint Designer) Comprehensive Endpoint List (Restful)

### 

**Users endpoint**

| Resource | Endpoint | Method | Description |
| --- | --- | --- | --- |
| Users | /admin | GET | Admin access |
|  | /users | POST | Create a new user |
|  | /users/{userId} | GET | Retrieve a specific user |
|  | /users/{userId} | PUT | Update a user completely |
|  | /users/{userId} | PATCH | Update a user partially |
|  | /users/{userId} | DELETE | Delete a user |

### Job Postings endpoint

| Resource | Endpoint | Method | Description |
| --- | --- | --- | --- |
| Job posting | /Admin/jobs | GET | List all job postings |
|  | /{userId}/jobs/ | POST | Create a job posting |
|  | /{userId}/jobs/{id}/ | GET | Retrieve a job posting |
|  | /{userId}/jobs/{id}/ | PUT | Fully update a job posting |
|  | /{userId}/jobs/{id}/ | PATCH | Partially update a job posting |
|  | /{userId}/jobs/{id}/ | DELETE | Delete a job posting |

### 

### 

### 

### **Worker Profiles Endpoints**

| Endpoint | Method | Description |  |
| --- | --- | --- | --- |
| /users/workers | GET | List all worker profiles |  |
| /users/workers | POST | Create a worker profile |  |
| /users/workers/{id | GET | Retrieve a worker profile |  |
| /users/workers/{id} | PUT | Fully update a profile |  |
| /users/workers/{id} | PATCH | Partially update a profile |  |
| /users/workers/{id} | DELETE | Delete a profile |  |

### 

### **Applications Endpoints**

| Endpoint | Method | Description |
| --- | --- | --- |
| /user/applications | GET | List applications |
| /user/applications | POST | Submit an application |
| /user/applications/{id} | GET | Retrieve an application |
| /user/applications/{id} | PUT | Update an application |
| /user//applications/{id} | Delete | Delete an application |

### 

### **Reviews Endpoints**

| Endpoint | Method | Description |
| --- | --- | --- |
| /worker\_profiles/{worker\_id}/reviews | POST | Create a new review (e.g. user reviewing a worker) |
| /worker\_profiles/{worker\_id}/reviews | GET | Get all reviews received by a user |
| /reviews/{review\_id} | GET | Get a specific review |
| /reviews/{review\_id} | DELETE | Delete a review |

**Payments Endpoints**

| Endpoint | Method | Description |
| --- | --- | --- |
| /users/{sender\_id}/transactions | POST | Initiate a payment from user to another |
| /users/{sender\_id}/transactions | GET | List all initiated payments by a user |
| /users/{receiver\_id}/transactions/received | GET | List all payments received by a user |
| /transactions/{transaction\_id} | GET | Get a specific transaction by ID |

### **Conversation**

| Endpoint | Method | Description |
| --- | --- | --- |
| /users/{receiver\_id}/messages | POST | Send a message to another user (starts conversation if it doesn't exist) |
| /users/{user\_id}/conversations | GET | List conversations the user |
| /conversations/{conversation\_id}/messages | GET | Get all messages in a specific conversation |
| /messages/{message\_id} | DELETE | Delete a message |

### 

### **Skills endpoint**

| Endpoint | Method | Description |
| --- | --- | --- |
| /Admin/user | GET | List available skills |
| /Admin/user | GET | Add a new skill |

**2. Query Parameters & Filtering (Led by UX Analyst)**

We explore how query parameters and filtering mechanisms were implemented within the system’s API to enhance user experience, efficiency, and data retrieval precision. This feature set, led by the UX Analyst, was designed with a user-centric mindset to ensure that clients can interact with the API flexibly, optimally, and intuitively.

### **Filtering**

Filtering allows users to narrow down results by specifying what they want to see.

**Example:** If a user wants to find plumbing jobs in Nairobi that pay between 500 and 1000 shillings, the request would look like this:

**GET /jobs?category=plumbing&location=nairobi&price\_min=500&price\_max=1000**

**Filterable fields include:**

* **Jobs:** category, location, price range
* **Workers:** skill, location, rating
* **Users:** role (admin/user), status (active/inactive)

### **Sorting**

Sorting lets users choose the order of the results, for example, by newest jobs or highest-rated workers.

**How it works:**

* Use sort=fieldname to sort in ascending order.
* Use sort=-fieldname for descending order.

**Example:**

| GET /api/jobs?sort=price,-createdAt |
| --- |

This means: sort by price (low to high), then by creation date (newest first).

### 

### 

### **Pagination**

Pagination breaks large sets of data into smaller pages so it’s easier to view and faster to load.

**Parameters:**

* **page**: Which page to show (starts from 1)
* pageSize: How many items to show on each page

**Example:**

| GET /user/jobs?page=2&pageSize=10 |
| --- |

**Response:**

| {  "data": [...],  "meta": {  "page": 2,  "pageSize": 10,  "totalItems": 95,  "totalPages": 10  } } |
| --- |

### **Selecting Specific Fields (Sparse Fieldsets)**

Sometimes, users don’t need all the data – just a few fields. They can request specific fields to reduce loading time.

**Example:**

| GET /Admin/jobs?fields=title, location, price |
| --- |

This returns only the job title, location, and price.

### **2.5 Search**

Users can search using keywords like "electrician" or "Nairobi" to find relevant results.

**Example:**

| GET /jobs?search=plumber+nairobi |
| --- |

The system will return jobs that match those keywords in the title or description.

### **Location-Based Filtering**

This is useful when users want to find services near them. They can search by location (latitude and longitude) and radius.

**Example:**

| GET /user/workers?near=1.2921,36.8219&radius=10 |
| --- |

This returns workers within 10 kilometres of that location.

### **2.7 Handling Errors**

If a user makes a mistake in the query, the system will help by giving a clear error message.

**Example:**

| {  "error": "Invalid filter: 'locatn'. Did you mean 'location'?" } |
| --- |

### 

### **Phase 3: Documentation Excellence**

### **2. API Style Guide**

Establishing a consistent API style guide is crucial for maintainability and ease of use. Here are the recommended conventions:

#### **Naming Conventions**

* **Resources**: Use plural nouns (e.g., **/users, /jobs**).
* **Nested Resources:** Use nesting to indicate ownership or tight relationships.  
  Example:
* **/users/{user\_id}/applications**
* **/worker\_profiles/{worker\_id}/reviews**
* **/users/{user\_id}/messages**
* **HTTP Methods**:

| **Method** | **Usage** | **Example** |
| --- | --- | --- |
| **GET** | Retrieve a list or a specific resource | GET /job\_postings,  GET /users/{id} |
| **POST** | Create a new resource | POST /applications |
| **PUT** | Replace an entire ressource | PUT /users/{id} |
| **PATCH** | Partially update a resource | PATCH /users/{id} |
| **DELETE** | Delete a resource | DELETE /messages/{id} |

#### **Formatting Standards**

* **Dates**: ISO 8601 format (**YYYY-MM-DD).**
* **Currency**: 3-letter ISO codes (e.g., **USD, RWF, EUR**) with decimal amounts.
* **Booleans**: Lowercase **true** or **false**.
* **Phone Numbers**: E.164 format (**+250785000066**)

**Status Codes and Responses**

| **Code** | **Meaning** |
| --- | --- |
| 200 OK | Successful GET, PUT, or PATCH |
| 201 Created | Resource successfully created |
| 204 No Content | Successful DELETE |
| 400 Bad Request | Validation or input error |
| 401 Unauthorized | Auth required or invalid |
| 403 Forbidden | Not allowed |
| 404 Not Found | Resource does not exist |
| 422 Unprocessable Entity | Semantic errors (e.g. invalid enum) |
| 500 Internal Server Error | Server-side failure |

**Standard Error Format**

Every error response should follow this format:

| {  "error": {  "message": "Invalid email address",  "code": 400  } } |
| --- |

**API Versioning**

Use **URI-based versioning** for breaking changes:  
 **/v1/users, /v1/job\_postings**

For backward-compatible changes (e.g. adding fields), no version bump is needed.

#### **Documentation Templates for consistency**

* Consistent request/response examples.
* Clear descriptions for endpoints, parameters, and responses.
* Standardised error messages and status codes.
* Interactive documentation (e.g., Swagger UI) is encouraged

#### **Extension Guidelines**

* Adhere to REST conventions for new features.
* Use versioning for breaking changes or major upgrades.

**3. Developer Experience Documentation**

#### **Getting Started Guide**

1. **Sign Up**: Register
2. **Obtain API Key**: Retrieve your JWT(JSON Web Token) token upon registration.

**Authentication Flow**

**POST /auth/login**

**Request:**

| **{  "email": "user@example.com",  "password": "\*\*\*\*\*\*\*\*" }** |
| --- |

**Response:**

| **{  "token": "JWT\_TOKEN" }** |
| --- |

#### **Interactive Examples**

interactive examples for common API operations:

* Create a Job Posting (Client):

| POST /users/{user\_id}/job\_postings |
| --- |

Body:

| {  "title": "Design a logo",  "description": "Need a creative logo for my bakery",  "category": "Design",  "location": "Nairobi",  "budget": 100 } |
| --- |

* Applying for a Job(Worker)

| **POST /users/{user\_id}/applications** |
| --- |

Body:

| {  "job\_id": "abc123",  "cover\_letter": "I'm a professional designer with 5 years of experience." } |
| --- |

* Sending a chat message.

| POST /users/{sender\_id}/messages |
| --- |

Body:

| {  "receiver\_id": "xyz456",  "content": "Hi! I'm interested in working with you on future projects." } |
| --- |

#### **Use-Case Based Documentation**

**Use Case 1: Hiring a Worker**

1. Client posts a job: POST /users/{id}/job\_postings
2. Workers Apply: POST /users/{id}/applications
3. Client reviews applications: **GET /job\_postings/{job\_id}/applications**
4. Payment after agreement: POST /users/{id}/payments

**Use Case 2: Sending a Message**

1. Worker sees a job and sends a message: POST /users/{id}/messages
2. The conversation is automatically initiated or resumed
3. Both users can fetch messages via:

GET /conversations/{conversation\_id}/messages

### **Phase 4: Security & Compliance Design**

### **1. Authentication Methods for the Target Market**

Given our platform targets users in regions with varying levels of digital infrastructure and devices (e.g., Africa), our authentication system must balance security, simplicity, and accessibility.

#### The Recommended Authentication Methods are **:**

* **Email + Password (Primary)** Reliable and familiar to most users. Strong password policy required(minimum length, complexity). Supports email verification on signup for added trust
* **Phone Number + OTP** Ideal for users without email access or with low literacy levels. Can be added using SMS-based OTP.

### **2. Token-Based Authentication Flows**

We will use JWT (JSON Web Token) for stateless and scalable authentication.

#### **JWT Flow:**

1. **Login Request**:  
   * Endpoint: POST /auth/login
   * User submits email and password:

| {  "email": "user@example.com",  "password": "\*\*\*\*\*\*\*\*" } |
| --- |

1. **Authentication Logic**:  
   * Server validates credentials.
   * If valid, generate **access token** (short-lived) and **refresh token** (longer-lived).
2. **Response**:  
   * Returns both tokens:

| {  "access\_token": "JWT\_ACCESS\_TOKEN",  "refresh\_token": "JWT\_REFRESH\_TOKEN" } |
| --- |

**4. Authenticated Requests**:

* Client sends access token in headers:

### **3. Refresh Token Strategy**

#### **Why It's Important:**

Mobile users may experience unstable internet connections. A robust refresh token system helps avoid forced logouts.

#### **Design:**

* **Access Token**: Valid for 15–30 minutes.
* **Refresh Token**: Valid for 7–30 days.
* Stored securely (e.g., HTTPOnly cookie or encrypted local storage).
* **Endpoint**: POST /auth/refresh

Input:

| {  "refresh\_token": "JWT\_REFRESH\_TOKEN" } |
| --- |

Response:

| {  "access\_token": "NEW\_JWT\_ACCESS\_TOKEN" } |
| --- |

* Invalidate refresh tokens on logout or suspicious behavior.

### **4. Security Requirements for API Consumers**

To ensure secure API consumption:

#### **- Authentication:**

* All protected endpoints must require **Authorization headers** with a valid JWT.
* Use **HTTPS** exclusively to prevent token leakage.

#### **-Validation:**

* Validate JWT signatures using a **secure secret** or **public key** (if using asymmetric encryption).
* Implement token **expiration**.

#### **-Rate Limiting**

* Apply rate limits to **/auth/login** and **/auth/refresh** endpoints.
* Lock accounts after repeated failed login attempts.

**Authorization Framework**

**1. Role-Based Access Control**

We implement Role-Based Access Control (RBAC) to determine user permissions based on their role. This ensures actions are scoped to a user’s responsibilities on the platform.

**Defined Roles & Permissions**

| **Role** | **Capabilities** |
| --- | --- |
| Worker | - Create & update their profile  - Apply to job postings  - View their applications  - View payment receipts |
| Client | - Post new job listings  - Edit/delete own jobs  - View applications to their jobs  - Make payments |
| Admin | - Moderate all content  - Access all data  - System analytics  - Override user permissions |

**Special Rules**

- Profile edits are restricted to owners only (verified via user ID matching)

- Applications can't be modified after the job closure dates

- Payment records are visible to both transaction parties

**2. Ownership Verification**

For sensitive or user-generated resources, ownership checks are enforced.

| **Resource** | **Ownership Rule** |
| --- | --- |
| **Profile** | Only the profile owner (user\_id) can update it. |
| **Job Post** | Only the posting client can update/delete the job. |
| **Application** | Only the worker who applied can view/withdraw it (before deadline). |
| **Review** | Only the author can delete/edit it within 7 days. |
| **Payment** | Both sender and receiver have access. |

**Example ownership check:**

| **if request.user.id != resource.owner\_id:  raise PermissionDenied("You can only access your own data.")** |
| --- |

**3. Endpoint Protection Matrix**

| **Endpoint** | **Allowed Roles** | **Ownership Requirement** |
| --- | --- | --- |
| GET /profiles/{id} | Owner, Admin | Profile must belong to user |
| POST /jobs/{id}/applications | Worker | None |
| GET /jobs/{id}/applications | Job Owner, Admin | Client must own the job |
| DELETE /reviews/{id} | Admin | None |

**4. Technical Implementation**

**1. JWT Claims Structure**

JWT tokens include role and permission claims to enforce access at runtime**.**

| {  "sub": "user\_123",  "roles": ["worker"],  "perms": ["profile:write", "job:apply"] } |
| --- |

**2. Middleware Checks**

| def check\_ownership(request\_user, resource\_user\_id):  if request\_user.id != resource\_user\_id and "admin" not in getattr(request\_user, "roles", []):  return JsonResponse({  "error": "forbidden",  "message": "You do not own this resource"  }, status=403)  return None |
| --- |

5. **Error Response Conventions**

| **{  "error": "forbidden",  "message": "Job editing requires 'client' role",  "required\_role": "client",  "current\_role": "worker" }** |
| --- |

Use consistent **403** and **401** status codes:

* **401** Unauthorized: Token missing or invalid.
* **403** Forbidden: Valid token, but insufficient role or ownership.

**Data Privacy & Compliance**

**1. Sensitive Data Handling**

**Protected Data Types:**

* Personal Identifiers

(Phone, Email)

- Storage: AES-256 encryption

- Display: Partial masking (+254\*\*\*789, ex\*\*\*@domain.com)

- Access: Owner + admin only

* Financial Information

- Processed via Flutterwave/Paystack (PCI-DSS compliant)

- No raw card/bank details stored

* Location Data

- Precise GPS → Truncated to nearest district for non-admin views(Kigali, Gasabo)

- Job radius matching: ±3km precision

**2. Consent Workflows**

* Required Consents (Signup):

| POST /api/signup {  "accept\_data\_processing": true, # Mandatory  "accept\_marketing": false # Optional } |
| --- |

* User Management:

| PATCH /users/{id}/preferences {  "privacy": {  "share\_profile": true, # Default: false  "location\_precision": 2 # 1=City, 2=District, 3=Exact  } } |
| --- |

**3. Regional Compliance**

| Country | Key Requirements Implemented | API Endpoint |
| --- | --- | --- |
| Kenya | Data Protection Act 2019 | DELETE /users/{id}` (30-day Kendra |
| Nigeria | NDPR audit trails | GET /users/{id}/access\_logs |
| Rwanda | MINICT data localization | Encrypted local storage |

**4. Data Minimization**

* Public API Responses:

| {  "job": {  "title": "Plumbing Repair",  "budget": 5000,  "location": "Nairobi West" // Approximated  } } |
| --- |

* Private (Owner/Admin) Responses:

| {  "job": {  "client\_contact": "j\*\*\*@email.com",  "exact\_location": "1.2345,36.7890"   } } |
| --- |

**5. Retention & Deletion**

- **Active accounts**: Data retained indefinitely

- **Deleted accounts**: Soft-deleted instantly → Anonymized after 30 days

- **Audit logs**: 2-year retention

**6. Technical Implementation Notes:**

1. **Field Tagging**:All PII fields tagged in the database schema (`is\_pii=true`)

2. **Log Hygiene**:Automated nightly scans for unmasked PII in logs

3. **Breach response protocol**:

- Notify affected users within 72 hours

- Rotate encryption keys

**Phase 5: Market-Specific Design Considerations**

Led by: All Team Members

In designing the JuaJobs API, it is essential to consider the diverse markets and infrastructure conditions across Africa. This phase focuses on three critical areas: localisation, performance optimisation in low-connectivity environments, and integration with various regional payment systems.

### **1. Localisation Strategy**

**a. Multi-Language Support**

To serve a linguistically diverse user base, the API will support multiple languages. This will be implemented using the Accept-Language HTTP header, allowing clients to specify the preferred language (e.g., en, fr, sw, am). Default responses will be in English if no preference is specified.

**b. Currency, Date, and Unit Formatting**

Standardisation is key for clarity. Currency values will follow the ISO 4217 standard (e.g., KES for Kenyan Shillings, NGN for Nigerian Naira). Dates will be formatted in ISO 8601 (YYYY-MM-DD) for consistency. Units (distance, weight, etc.) will be localised as per the user’s region.

**c. Region-Specific Data Requirements**

Some countries may require capturing additional identity information (e.g., national ID numbers, province, or region). The API will accommodate these as optional fields based on the user’s selected country.

**d. Cultural Adaptation in API Behaviour**

Examples of culturally aware adaptations include:

* Offering job categories relevant to specific regions.
* Customising review systems to account for cultural norms (e.g., stars vs. thumbs up).
* Supporting local job types (e.g., boda-boda drivers in East Africa).

### **2. Connectivity & Performance Optimisation**

**a. Batch Operations**

To optimise for users in low-bandwidth areas, the API supports batch endpoints. For example, clients can send multiple job applications or retrieve multiple profiles in a single request using a /batch endpoint.

**b. Caching Strategies**

Caching will be implemented using:

* **ETag headers** to reduce redundant data transfer.
* **Cache-Control** headers to define caching behaviour for static data (e.g., job categories, regions).

**c. Payload Optimisation**

The API supports:

* **Sparse fieldsets**: Clients can specify only the fields they need using a fields query parameter.
* **Pagination and limit parameters** to reduce the amount of data sent per request.
* **Compression** via GZIP to minimise bandwidth use.

**d. Offline-First Capabilities**

For users who operate in intermittent connectivity conditions:

* The API will support **queueing of write operations** client-side, which can be synced when the user is back online.
* Clients can cache responses for non-sensitive GET requests, with TTL (time-to-live) headers guiding expiration.

**3. Payment Integration Design**

### **Integration Points with Payment Systems**

* **Mobile Money APIs**: Map integration with providers like MTN MoMo, Airtel Money...
* **Bank Transfers**: Identify API endpoints for local banks or third-party payment aggregators (like Flutterwave or Paystack).
* **Fallback/Alternative Options**: Consider wallet systems or card payments.

### **API Endpoints Design**

I will design RESTful API endpoints to support:

* POST /payments/initiate – Start a payment process
* GET /payments/status/:transactionId – Check transaction status
* POST /payments/webhook – Handle payment confirmation callbacks
* GET /payments/history – List past transactions for a user/employer

Each endpoint will support multiple payment methods and provide clear response codes.

### **Transaction State Management**

* Use a **finite state machine** model for transactions:  
  + PENDING, PROCESSING, SUCCESS, FAILED, CANCELLED
* Implement retry logic for transient failures
* Log all transaction state changes for traceability

### **Security Considerations**

* Encrypt sensitive data (e.g., using AES or RSA)
* Use secure tokens for authentication (JWT or OAuth2)
* Implement IP whitelisting and rate limiting for callbacks
* Ensure PCI-DSS compliance for card payments
* Protect against replay attacks and fraud