# JuaJobs API Design Documentation

## Overview

JuaJobs is a gig economy platform designed to connect skilled workers across Africa (plumbers, electricians, artisans, etc.) with clients seeking their services. This API documentation outlines the core functionality of the platform, focusing on job postings, worker profiles, and job applications.

## Business Context

In many African markets, skilled workers face challenges in finding consistent work, while clients struggle to find reliable, vetted professionals for their needs. JuaJobs bridges this gap by providing:

1. **Digital presence** for skilled workers who traditionally rely on word-of-mouth
2. **Verification and review systems** to build trust
3. **Location-based matching** to connect local workers with nearby opportunities
4. **Secure payment options** adapted to local preferences
5. **Mobile-first experience** to account for the mobile-dominant African internet landscape

## Core Resources

### 1. Users

Users are divided into two primary roles:

- **Clients**: Post jobs and hire workers
- **Workers**: Create profiles and apply for jobs

Both types share common user attributes but have role-specific functionalities.

### 2. Job Postings

Job postings are created by clients and represent work opportunities.

### 3. Worker Profiles

Worker profiles showcase skills, experience, and availability of service providers.

### 4. Applications

Applications represent a worker's interest in a specific job posting and the subsequent hiring workflow.

### 5. Reviews

Reviews provide feedback on both workers and clients after job completion.

### 6. Job filters

Search options for jobs (location, category, entry level, skills).

### 7. Messages

Chats between employers and workers.

### 8. Payment

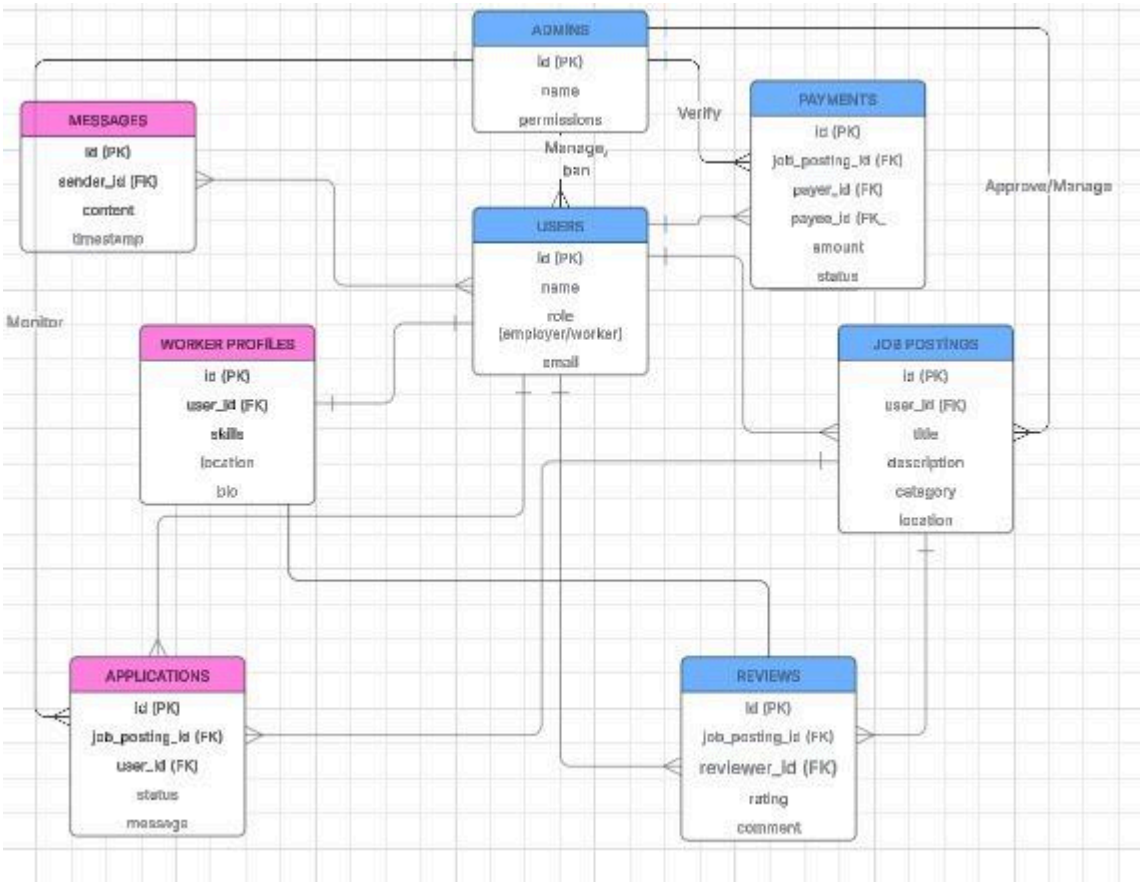
Payment tracking for job transactions.

## API Architect

**A-Resource Identification:**

1-Users: People using the platform (employers + workers).
2-Worker Profiles: Detailed information about workers (name, skills, location, bio).
3-Job Postings: Jobs posted by employers.
4-Applications: Job applications submitted by workers.
5-Reviews: Feedback and comments left by users after job completion.
6-Admins: Manage safety, jobs, and payments. Verify and confirm the job's authenticity.
7-Job Filters: Search options for jobs (location, category, entry level, skills).
8-Messages: Chats between employers and workers.
9-Payments: Payment tracking for job transactions.

**Entity Relationship Diagram:**




**C-HTTP status codes & error handling:**

| Status Code | Meaning | Use |
|---|---|---|
| 200 OK | Success | GET request completed successfully |
| 201 Created | Resource created | POST requests created a new job, user, review, etc. |
| 204 No Content | Success with no data | DELETE succeeded, no content to return |
| 400 Bad Request | Invalid input or missing | Required field missing or invalid JSON payload |
| 401 Unauthorized | Not logged on or invalid token | User must be logged in or token is missing/invalid |
| 403 Forbidden | Not allowed | User is logged in but lacks permission (e.g., not an admin) |

| | | |
|---|---|---|
| 404 Not Found | Resource not found | No user, job, or application with that ID |
| 409 Conflict | Already exists | Trying to register with an already used email |
| 422 Unprocessable Entity | Invalid input format | JSON field is the wrong type, invalid format (e.g. invalid email) |
| 500 Internal Server Error | Something broke | Unexpected server crash or bug |

**D-API Versioning:**

The versioning approach we selected is URL-based (URI) versioning because it's:

-Clear and visible in browser and logs.
-Simple for routing and documentation.
-Most widely adopted in RESTful API design.
-Easier for clients to specify which version they use.
-Encourages predictable version management and scalability.

Deprecation Policy:

-Announce deprecated versions via response headers and documentation.
-Provide a minimum 6-month notice before removing deprecated versions.
-Include "Depreciation" and "Sunset" headers

Backward Compatibility Guidelines:

-Non-breaking changes (adding optional fields) stay within the same version.
-Breaking changes (removing a field or changing a response structure) require a new version (v2).
-Maintain full support for old versions until officially sunset.

Change Management Framework:

-Proposal: Propose changes via internal API RFC (Request For Comments) doc
-Review: By the API Governance Group
-Test: For backward compatibility
-Versioning: Version and document the change
-Release: Behind a feature toggle if needed.

## 1. User Endpoints

| Method | Endpoint | Description | Authentication |
|---|---|---|---|
| POST | /api/users/register | Register a new user | None |
| POST | /api/users/login | Login and obtain JWT token | None |
| GET | /api/users/me | Retrieve own profile | Authenticated |
| PUT | /api/users/me | Update user profile | Authenticated |
| DELETE | /api/users/me | Deactivate own account | Authenticated |

---

## 2. Job Postings

| Method | Endpoint | Description | Authentication |
|---|---|---|---|
| GET | /api/jobs | List all jobs (with filters) | Optional |
| POST | /api/jobs | Post a new job | Client |
| GET | /api/jobs/{id} | Get job by ID | Optional |
| PUT | /api/jobs/{id} | Update job post | Client |
| DELETE | /api/jobs/{id} | Delete job | Client |

---

## 3. Worker Profiles

| Method | Endpoint | Description | Authentication |
|---|---|---|---|
| GET | /api/workers | List worker profiles | Optional |
| GET | /api/workers/{id} | Get profile by ID | Optional |
| PATCH | /api/workers/{id} | Update own profile | Worker |

---

## 4. Applications

| Method | Endpoint | Description | Authentication |
|---|---|---|---|
| POST | /api/jobs/{jobId}/applications | Apply to a job | Worker |
| GET | /api/users/me/applications | View own applications | Worker |
| DELETE | /api/applications/{id} | Withdraw application | Worker |

---

## 5. Reviews

| Method | Endpoint | Description | Authentication |
|---|---|---|---|
| POST | /api/reviews | Submit a review | Authenticated |
| GET | /api/reviews/{id} | View review by ID | None |
| GET | /api/workers/{id}/reviews | List reviews for worker | None |
| GET | /api/clients/{id}/reviews | List reviews for client | None |

---

## 6. Job Filters

| Method | Endpoint | Description | Authentication |
|---|---|---|---|
| GET | /api/job-filters | List saved job filters | Authenticated |
| POST | /api/job-filters | Save new job filter | Authenticated |
| DELETE | /api/job-filters/{id} | Delete a saved filter | Authenticated |

---

## 7. Messages

| Method | Endpoint | Description | Authentication |
|---|---|---|---|
| GET | /api/conversations | List user's conversations | Authenticated |
| POST | /api/conversations | Start a new conversation | Authenticated |
| GET | /api/conversations/{id}/messages | Get messages in a conversation | Authenticated |
| POST | /api/conversations/{id}/messages | Send message in conversation | Authenticated |

---

## 8. Payments

| Method | Endpoint | Description | Authentication |
|---|---|---|---|
| POST | /api/payments/initiate | Initiate a job payment | Client |
| GET | /api/payments/{id} | Get payment status | Client/Worker |
| POST | /api/payments/{id}/confirm | Confirm receipt of payment | Worker |

| | | | |
|---|---|---|---|
| GET | /api/users/me/wallet | Get user wallet balance & transaction history | Authenticated |

# Query Parameters & Filters

### Job Listing Filters – `GET /api/jobs`

| Parameter | Type | Description |
|---|---|---|
| location | string | City or region |
| category | string | Job category |
| minBudget | number | Minimum budget |
| maxBudget | number | Maximum budget |
| postedSince | string | ISO date string for recently posted jobs |
| sort | string | `budget_desc`, `date_asc` |
| page | number | Page number for pagination |
| limit | number | Items per page |

---

### Worker Profile Filters – `GET /api/workers`

| Parameter | Type | Description |
|---|---|---|
| skill | string | Filter by skill name |
| location | string | Filter by worker location |
| ratingAbove | number | Minimum average rating |
| availableNow | boolean | Show only currently |

| | | |
|---|---|---|
| | | available workers |
| sort | string | `rating_desc`, `location_asc` |

---

### Saved Job Filters – `GET /api/job-filters`

| Parameter | Type | Description |
|---|---|---|
| name | string | Name of saved filter |
| createdAfter | string | ISO date string to filter recent saves |

---

### Conversations – `GET /api/conversations`

| Parameter | Type | Description |
|---|---|---|
| unreadOnly | boolean | Only return conversations with unread messages |

---

### Advanced Query Features

| Feature | Example Endpoint | Description |
|---|---|---|
| Full-text search | `/api/jobs/search?q=plumber+lagos` | Searches job title and description |
| Geo-filtering | `/api/jobs?lat=-1.9&lng=30.1&radius=15` | Jobs within a geo-radius |
| Sparse fieldsets | `/api/workers?fields=id,name,skills` | Return only specified fields |
| Field expansion | `/api/jobs?expand=client,reviews` | Include nested related data |

# Token-Based Authentication (OAuth2 + JWT)

## Why We Chose It:

- Mobile-first compatibility: Ideal for mobile clients and modern frontends.
- Stateless architecture: No need for session storage on the server.
- Scalability: Suitable for distributed, cloud-native systems.

## Authentication Flow:

- User logs in using email or phone number + password.
- Server validates credentials and returns:
    - Access Token – short-lived, used for API access.
    - Refresh Token – longer-lived, used to renew access tokens.

## Key Endpoints:

| Method | Endpoint | Description |
| --- | --- | --- |

| | | |
|---|---|---|
| POST | /auth/login | Authenticate and get tokens |
| POST | /auth/refresh | Refresh the access token |
| POST | /auth/logout | Invalidate current token |
| POST | /auth/register | Register a new user |

### JWT Payload (Claims):

- sub: User ID
- role: User role (client / worker / admin)
- exp: Expiration time
- iat: Issued-at timestamp

### Security Practices:

- Use HTTPS only — never expose tokens over insecure connections.
- Store tokens in HTTP-only cookies or secure local storage.
- Rate-limit login attempts to prevent brute-force attacks.
- Rotate secret keys used for signing JWTs regularly.

# Authorization Design: Role-Based Access Control (RBAC)

### Defined Roles:

- Client: Can post jobs, view applicants, and review workers.
- Worker: Can browse jobs, apply, and manage their profile.
- Admin: Full access across the platform (e.g., verifying users).

### Endpoint Access Summary by Role

#### `POST /jobs`

- **Client**: Allowed – can post job listings.

- **Worker**: Not allowed – workers cannot create jobs.
- **Admin**: Allowed – can manage job postings.

**`GET /jobs/{id}`**

- **Client**: Allowed – can view job details.
- **Worker**: Allowed – can view job details.
- **Admin**: Allowed – has full visibility.

**`POST /jobs/{id}/apply`**

- **Client**: Not allowed – clients cannot apply for jobs.
- **Worker**: Allowed – workers can apply for available jobs.
- **Admin**: Allowed – for management or testing purposes.

**`GET /applications/{id}`**

- **Client**: Allowed, but only if they own the job tied to the application.
- **Worker**: Allowed, but only for their own applications.
- **Admin**: Fully allowed – has access to all applications.

**`DELETE /users/{id}`**

- **Client**: Not allowed – no user management permissions.
- **Worker**: Not allowed – cannot delete accounts.
- **Admin**: Allowed – can delete user accounts as part of admin controls.

## Ownership Checks:

- A worker can only modify their own profile.

- A client can only access their posted jobs and related applications.

# Data Privacy & Compliance

## PII Handling:

- Encrypt sensitive data at rest (e.g., phone numbers, email addresses).
- Avoid logging Personally Identifiable Information (PII).
- Design API responses to return only what's necessary (data minimization).

**Example: Safe Public Worker Profile**

```
{
  "id": "w123",
  "name": "Jane W.",
  "skills": ["Plumbing", "Carpentry"],
  "rating": 4.7
  // Contact info hidden unless hired
}
```

## Consent & Compliance:

- Display Terms & Conditions and Privacy Policy during sign-up.
- Track explicit user consent (e.g., timestamped agreement log).
- Align data handling with local laws:
    - Kenya: Data Protection Act (2019)
    - Nigeria: NDPR

## Documentation Excellence

```yaml
openapi: 3.0.3
info:
  title: JuaJobs API
  description: >
    RESTful API for JuaJobs - a gig economy platform connecting skilled workers with clients across Africa.
    This documentation is designed to help developers understand and consume the API effectively.
  version: 1.0.0

servers:
  - url: https://api.juajobs.africa/v1
```

```yaml
    description: Production server

paths:
  /users:
    get:
      summary: Get all users
      responses:
        '200':
          description: A list of users
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/User'

  /users/{userId}:
    get:
      summary: Get user by ID
      parameters:
        - name: userId
          in: path
          required: true
          schema:
            type: string
      responses:
        '200':
          description: User details
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'

  /profiles/{userId}:
    get:
      summary: Get user profile
      parameters:
        - name: userId
          in: path
          required: true
          schema:
            type: string
      responses:
```

```yaml
      '200':
        description: User profile
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Profile'

/applications:
  post:
    summary: Submit job application
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Application'
    responses:
      '201':
        description: Application submitted successfully

/reviews:
  post:
    summary: Submit a review
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Review'
    responses:
      '201':
        description: Review submitted

/messages:
  post:
    summary: Send a message
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Message'
    responses:
```

```yaml
        '201':
          description: Message sent

  /payments:
    post:
      summary: Process a payment
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Payment'
      responses:
        '200':
          description: Payment processed

  /jobs/filter:
    post:
      summary: Filter jobs based on criteria
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/JobFilter'
      responses:
        '200':
          description: Filtered job results
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object

components:
  schemas:
    User:
      type: object
      properties:
        id:
          type: string
        name:
```

```yaml
        type: string
      email:
        type: string
      role:
        type: string
      createdAt:
        type: string
        format: date-time

  Profile:
    type: object
    properties:
      userId:
        type: string
      bio:
        type: string
      skills:
        type: array
        items:
          type: string
      experience:
        type: string
      location:
        type: string

  Application:
    type: object
    properties:
      userId:
        type: string
      jobId:
        type: string
      coverLetter:
        type: string
      resumeUrl:
        type: string

  Review:
    type: object
    properties:
      reviewerId:
        type: string
      revieweeId:
```

```
        type: string
      rating:
        type: integer
        format: int32
      comment:
        type: string

  Message:
    type: object
    properties:
      senderId:
        type: string
      recipientId:
        type: string
      content:
        type: string
      timestamp:
        type: string
        format: date-time

  Payment:
    type: object
    properties:
      payerId:
        type: string
      payeeId:
        type: string
      amount:
        type: number
        format: float
      jobId:
        type: string
      method:
        type: string

  JobFilter:
    type: object
    properties:
      category:
        type: string
      location:
        type: string
      experienceLevel:
```

```yaml
          type: string
        budgetRange:
          type: object
          properties:
            min:
              type: number
              format: float
            max:
              type: number
              format: float

  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT

security:
  - bearerAuth: []
```