API Design Principles

- **RESTful**: Follows REST conventions (e.g., GET for retrieval, POST for creation, PUT for updates, DELETE for deletion).
- **Authentication**: JWT-based authentication for all endpoints except `/signup` and `/login`.
- **Input Validation**: Validate all inputs (e.g., email format, phone numbers) to prevent errors and attacks.
- **Error Handling**: Standardized error responses with HTTP status codes and messages.
- **Scalability**: Support pagination, rate limiting, and async processing for large datasets.
- **Content Type**: JSON for requests/responses; multipart/form-data for file uploads.

Authentication

- **JWT**: Issued on `/login`, included in `Authorization: Bearer <token>` header for protected endpoints.
- **Validation**: Middleware checks JWT validity and extracts `user_id` for user-specific operations.
- **Rate Limiting**: Apply limits (e.g., 100 requests/minute) to prevent abuse.

Base URL

- `https://api.africoda.com/v1` - Dev
- `http://localhost:9999/v1` - Prod

Phase 1: Core Features API Design

Features: Sign-up, login, contact upload, basic contact management, SMS sending.

1. Authentication Endpoints

POST /signup

- **Description**: Register a new user.
- **Request**:

```
{
  "username": "string",
  "email": "string",
  "password": "string",
}
```

- **Validation**:
  - `username`: 3-50 chars, unique.
  - `email`: Valid email, unique.
  - `password`: Min 8 chars, includes letters and numbers.
- **Response** (201 Created):

```
{
  "user_id": "integer",
  "username": "string",
  "email": "string",
  "created_at": "string (ISO 8601)"
}
```

- **Error** (400 Bad Request, 409 Conflict):

```
{
  "error": "Email already exists"
}
```

POST /login

- **Description**: Authenticate user and issue JWT.
- **Request**:

```
{
  "email": "string",
  "password": "string"
}
```

- **Response** (200 OK):

```
{
  "token": "string (JWT)",
  "user_id": "integer",
  "username": "string"
}
```

- **Error** (401 Unauthorized):

```
{
  "error": "Invalid credentials"
}
```

2. Contact Management Endpoints

POST /contacts/upload

- **Description**: Upload a CSV/Excel file to add contacts.
- **Auth**: JWT required.
- **Request**: `multipart/form-data`
  - `file`: CSV/Excel file (columns: name, phone, email, sms_consent).
- **Validation**:
  - File size < 10MB.
  - Phone numbers validated (e.g., E.164 format using `libphonenumber`).
  - `sms_consent` must be boolean if provided.
- **Response** (202 Accepted):

```
{
  "upload_id": "integer",
```

```
  "file_name": "string",
  "total_rows": "integer",
  "valid_rows": "integer",
  "invalid_rows": "integer",
  "feedback": {
    "row_5": "Invalid phone number",
    "row_10": "Missing name"
  },
  "uploaded_at": "string (ISO 8601)"
}
```

- **Processing**: Async via queue (e.g., RabbitMQ) to parse file, validate, and store in `contacts` and `upload_logs` tables.
- **Error** (400 Bad Request):

```
{
  "error": "Invalid file format"
}
```

GET /contacts

- **Description**: Retrieve user's contacts with pagination and filters.
- **Auth**: JWT required.
- **Query Params**:
  - `page`: Integer (default 1).
  - `limit`: Integer (default 50, max 100).
  - `search`: String (optional, search by name/phone).
  - `is_valid`: Boolean (optional, filter valid/invalid contacts).
- **Response** (200 OK):

```
{
  "contacts": [
    {
      "id": "integer",
      "name": "string",
```

```
      "phone": "string",
      "email": "string",
      "sms_consent": boolean,
      "created_at": "string (ISO 8601)",
      "updated_at": "string (ISO 8601)"
    }
  ],
  "pagination": {
    "page": "integer",
    "limit": "integer",
    "total": "integer"
  }
}
```

- **Error** (401 Unauthorized):

```
{
  "error": "Unauthorized"
}
```

GET /contacts/:id

- **Description**: Retrieve a single contact.
- **Auth**: JWT required.
- **Response** (200 OK):

```
{
  "id": "integer",
  "name": "string",
  "phone": "string",
  "email": "string",
  "sms_consent": boolean,
  "created_at": "string (ISO 8601)",
  "updated_at": "string (ISO 8601)"
}
```

- **Error** (404 Not Found):

```
{
  "error": "Contact not found"
}
```

PUT /contacts/:id

- **Description**: Update a contact's details.
- **Auth**: JWT required.
- **Request**:

```
{
  "name": "string",
  "phone": "string",
  "email": "string",
  "sms_consent": boolean
}
```

- **Response** (200 OK): Same as GET /contacts/:id.
- **Error** (400 Bad Request, 404 Not Found):

```
{
  "error": "Invalid phone number"
}
```

DELETE /contacts/:id

- **Description**: Soft delete a contact (set `is_valid` to FALSE or add `deleted_at`).
- **Auth**: JWT required.
- **Response** (204 No Content).
- **Error** (404 Not Found).

3. SMS Sending Endpoints

POST /sms/send

- **Description**: Send SMS to contacts or phone numbers.

- **Auth**: JWT required.
- **Request**:

```
{
  "contact_ids": ["integer"], // Optional
  "phones": ["string"], // Optional
  "message": "string" // Max 160 chars for SMS
}
```

- **Validation**:
  - At least one of `contact_ids` or `phones` required.
  - Check `sms_consent` for contacts in `contact_ids`.
  - Validate phone numbers in `phones`.
- **Response** (202 Accepted):

```
{
  "message_id": "integer",
  "status": "pending",
  "sent_at": "string (ISO 8601)"
}
```

- **Processing**: Async via queue to Twilio/Nexmo; log to `message_history`.
- **Error** (400 Bad Request):

```
{
  "error": "No valid recipients"
}
```

GET /sms/history

- **Description**: Retrieve real-time message history with pagination.
- **Auth**: JWT required.
- **Query Params**:
  - `page`: Integer (default 1).

    - `limit`: Integer (default 50, max 100).
    - `status`: Enum (pending, sent, delivered, failed, optional).
    - `start_date`, `end_date`: ISO 8601 (optional).
- **Response** (200 OK):

```
{
  "messages": [
    {
      "id": "integer",
      "contact_id": "integer",
      "phone": "string",
      "message_content": "string",
      "status": "string",
      "sent_at": "string (ISO 8601)",
      "twilio_message_id": "string"
    }
  ],
  "pagination": {
    "page": "integer",
    "limit": "integer",
    "total": "integer"
  }
}
```

- **Error** (401 Unauthorized).

Phase 2: Advanced Features API Design

Features: Grouping, message templates, data cleaning tools.

4. Group Management Endpoints

POST /groups

- **Description**: Create a new contact group.
- **Auth**: JWT required.
- **Request**:

```
{
  "name": "string"
}
```

- **Response** (201 Created):

```
{
  "id": "integer",
  "name": "string",
  "created_at": "string (ISO 8601)"
}
```

- **Error** (400 Bad Request):

```
{
  "error": "Group name required"
}
```

GET /groups

- **Description**: Retrieve user's groups with pagination.
- **Auth**: JWT required.
- **Query Params**: `page`, `limit`, `search` (by name).
- **Response** (200 OK):

```
{
  "groups": [
    {
      "id": "integer",
      "name": "string",
      "created_at": "string (ISO 8601)",
      "updated_at": "string (ISO 8601)"
    }
  ],
  "pagination": { ... }
}
```

PUT /groups/:id

- **Description**: Update group name.
- **Auth**: JWT required.
- **Request**:

```
{
  "name": "string"
}
```

- **Response** (200 OK): Same as GET /groups/:id.

DELETE /groups/:id

- **Description**: Delete group (cascades to `contact_groups`).
- **Auth**: JWT required.
- **Response** (204 No Content).

POST /groups/:id/contacts

- **Description**: Add contacts to a group.
- **Auth**: JWT required.
- **Request**:

```
{
  "contact_ids": ["integer"]
}
```

- **Response** (200 OK):

```
{
  "group_id": "integer",
  "added_contacts": ["integer"]
}
```

- **Error** (400 Bad Request):

```
{
  "error": "Invalid contact IDs"
}
```

DELETE /groups/:id/contacts

- **Description**: Remove contacts from a group.
- **Auth**: JWT required.
- **Request**:

```
{
  "contact_ids": ["integer"]
}
```

- **Response** (204 No Content).

5. Message Template Endpoints

POST /templates

- **Description**: Create a message template with variables.
- **Auth**: JWT required.
- **Request**:

```
{
  "name": "string",
  "content": "string", // e.g., "Hi {FirstName}, join our sale on {Date}!"
  "variables": {
    "FirstName": "string",
    "Date": "date"
  }
}
```

- **Response** (201 Created):

```
{
  "id": "integer",
```

```
  "name": "string",
  "content": "string",
  "variables": { ... },
  "created_at": "string (ISO 8601)"
}
```

- **Error** (400 Bad Request):

```
{
  "error": "Invalid variables"
}
```

GET /templates

- **Description**: Retrieve user's templates with pagination.
- **Auth**: JWT required.
- **Query Params**: `page`, `limit`, `search` (by name).
- **Response** (200 OK):

```
{
  "templates": [
    {
      "id": "integer",
      "name": "string",
      "content": "string",
      "variables": { ... },
      "created_at": "string (ISO 8601)",
      "updated_at": "string (ISO 8601)"
    }
  ],
  "pagination": { ... }
}
```

PUT /templates/:id

- **Description**: Update a template.
- **Auth**: JWT required.

- **Request**: Same as POST /templates.
- **Response** (200 OK): Same as GET /templates/:id.

DELETE /templates/:id

- **Description**: Delete a template.
- **Auth**: JWT required.
- **Response** (204 No Content).

POST /sms/send-template

- **Description**: Send SMS using a template to contacts or groups.
- **Auth**: JWT required.
- **Request**:

```
{
  "template_id": "integer",
  "contact_ids": ["integer"], // Optional
  "group_ids": ["integer"], // Optional
  "phones": ["string"], // Optional
  "variables": {
    "FirstName": ["string"], // Matches contact_ids or phones
    "Date": "string"
  }
}
```

- **Validation**:
  - Check `sms_consent` for contacts.
  - Ensure `variables` match template's expected variables.
- **Response** (202 Accepted):

```
{
  "message_id": "integer",
  "status": "pending",
  "sent_at": "string (ISO 8601)"
}
```

- **Processing**: Async via queue; replace variables and send via Twilio/Nexmo.

6. Data Cleaning Endpoints

POST /data/clean

- **Description**: Clean uploaded contact data (e.g., standardize phones, remove duplicates).
- **Auth**: JWT required.
- **Request**:

```
{
  "upload_id": "integer", // Optional, clean specific upload
  "contact_ids": ["integer"], // Optional, clean specific contacts
  "operations": ["standardize_phone", "remove_duplicates", "fix_email"]
}
```

- **Response** (202 Accepted):

```
{
  "cleaning_id": "integer",
  "operations": ["string"],
  "details": {
    "rows_affected": "integer",
    "changes": {
      "row_5": "Phone standardized to +1234567890"
    }
  },
  "performed_at": "string (ISO 8601)"
}
```

- **Processing**: Async via queue; log to `data_cleaning_logs`.

GET /data/cleaning-logs

- **Description**: Retrieve cleaning operation logs with pagination.
- **Auth**: JWT required.
- **Query Params**: `page`, `limit`, `upload_id` (optional).

- **Response** (200 OK):

```
{
  "logs": [
    {
      "id": "integer",
      "upload_id": "integer",
      "operation": "string",
      "details": { ... },
      "performed_at": "string (ISO 8601)"
    }
  ],
  "pagination": { ... }
}
```

7.User preference and Settings

POST /user/settings

- **Description**: Create or update user settings.
- **Auth**: JWT required.
- **Request**:

```
{
  "preferred_provider_id": "integer",
  "default_language": "string",
  "timezone": "string",
  "receive_email_notifications": boolean,
  "receive_sms_notifications": boolean
}
```

- **Validation**:
  - `preferred_provider_id`: Must exist in `sms_providers`.
  - `default_language`: Valid ISO 639-1 code (e.g., “en”).
  - `timezone`: Valid IANA timezone (e.g., “Africa/Accra”).

- **Response** (200 OK):

```
{
  "id": "integer",
  "user_id": "integer",
  "preferred_provider_id": "integer",
  "default_language": "string",
  "timezone": "string",
  "receive_email_notifications": boolean,
  "receive_sms_notifications": boolean,
  "updated_at": "string (ISO 8601)"
}
```

- **Error** (400 Bad Request):

```
{ "error": "Invalid provider ID" }
```

GET /user/settings

- **Description**: Retrieve user settings.
- **Auth**: JWT required.
- **Response** (200 OK): Same as POST response.
- **Error** (404 Not Found):

```
{ "error": "Settings not found" }
```

POST /user/preferences

- **Description**: Create or update user preferences.
- **Auth**: JWT required.
- **Request**:

```
{
  "theme": "string",
  "pagination_limit": "integer",
```

```
  "date_format": "string"
}
```

- **Validation**:
  - `theme`: Enum (“light”, “dark”).
  - `pagination_limit`: 10–100.
  - `date_format`: Valid format (e.g., “DD-MM-YYYY”).
- **Response** (200 OK):

```
{
  "id": "integer",
  "user_id": "integer",
  "theme": "string",
  "pagination_limit": "integer",
  "date_format": "string",
  "updated_at": "string (ISO 8601)"
}
```

- **Error** (400 Bad Request):

```
{ "error": "Invalid theme" }
```

GET /user/preferences

- **Description**: Retrieve user preferences.
- **Auth**: JWT required.
- **Response** (200 OK): Same as POST response.

GET /notifications

- **Description**: Retrieve user notifications with pagination and filters.
- **Auth**: JWT required.
- **Query Params**:
  - `page`: Integer (default 1).
  - `limit`: Integer (default 50, max 100).

    - `status`: Enum (“unread”, “read”, “dismissed”, optional).
    - `type`: Enum (“sms_delivery”, “upload_status”, etc., optional).
- **Response** (200 OK):

```
{
  "notifications": [
    {
      "id": "integer",
      "type": "string",
      "message": "string",
      "status": "string",
      "channel": "string",
      "related_entity_id": "integer",
      "related_entity_type": "string",
      "created_at": "string (ISO 8601)",
      "read_at": "string (ISO 8601)"
    }
  ],
  "pagination": {
    "page": "integer",
    "limit": "integer",
    "total": "integer"
  }
}
```

PUT /notifications/:id

- **Description**: Update notification status (e.g., mark as read).
- **Auth**: JWT required.
- **Request**:

```
{
  "status": "string" // "read" or "dismissed"
}
```

- **Response** (200 OK): Same as GET /notifications single item.

- **Error** (400 Bad Request):

```
{ "error": "Invalid status" }
```

POST /oauth/login

- **Description**: Authenticate via OAuth (e.g., Google) and issue JWT.
- **Request**:

```
{
  "provider": "string", // e.g., "google"
  "code": "string" // OAuth authorization code
}
```

- **Processing**:
  - Exchange `code` for access/refresh tokens via provider's API (e.g., Google OAuth).
  - Store tokens in `oauth_tokens`.
  - Link or create `users` record with `oauth_id` and `oauth_provider`.
  - Issue JWT.
- **Response** (200 OK):

```
{
  "token": "string (JWT)",
  "user_id": "integer",
  "username": "string"
}
```

- **Error** (401 Unauthorized):

```
{ "error": "Invalid OAuth code" }
```

Modified Endpoints

- **POST /sms/send** and **POST /sms/send-template**:

    - Use `user_settings.preferred_provider_id` if `provider_id` is not specified in the request.
    - Check `receive_sms_notifications` to send delivery notifications via SMS.
- **GET /sms/history**:
    - Include `notification_id` in response to link to related notifications.

Integration with SMS Providers (BMS, mNotify, Hubtel)

- **User Settings**:
    - `preferred_provider_id` defaults SMS sends to BMS, mNotify, or Hubtel.
    - `receive_sms_notifications` controls whether users get SMS alerts (e.g., delivery confirmations) via the selected provider.
- **Notifications**:
    - Delivery updates from mNotify/Hubtel (via webhooks) or BMS (if supported) create entries in `notifications` linked to `message_history`.
    - Example: mNotify webhook updates `message_history.status` to “delivered” and creates an in-app notification.
- **OAuth**:
    - Users logging in via Google OAuth can still select a preferred provider and configure notifications.

Compliance and Security

- **GDPR**:
    - `receive_email_notifications` and `receive_sms_notifications` ensure user consent for notifications.
    - Soft delete (`users.is_active`, `contacts.is_valid`) supports data erasure requests.
    - Encrypt `oauth_tokens.access_token` and `refresh_token` using AES-256.
- **TCPA**:
    - Check `user_settings.receive_sms_notifications` before sending SMS notifications.
    - Continue enforcing `contacts.sms_consent` for SMS campaigns.
- **Security**:

- Use parameterized queries to prevent SQL injection.
- Validate OAuth tokens with provider APIs (e.g., Google's tokeninfo endpoint).
- Rate-limit `/oauth/login` to prevent abuse.

Additional Notes

Scalability

- **Async Processing**: Use RabbitMQ or AWS SQS for `/contacts/upload`, `/sms/send`, and `/data/clean` to handle large datasets.
- **Pagination**: Applied to GET endpoints (`/contacts`, `/sms/history`, `/groups`, `/templates`, `/data/cleaning-logs`) to limit response size.
- **Caching**: Cache `templates` and `groups` in Redis for faster access.

Security

- **Input Sanitization**: Use libraries like `validator.js` to sanitize inputs.
- **Rate Limiting**: Apply to `/login`, `/signup`, and `/sms/send` to prevent abuse.
- **SQL Injection**: Use parameterized queries (as shown in artifact).
- **File Uploads**: Validate file types and scan for malware (e.g., using AWS S3 virus scanning).

Compliance

- **TCPA**: Check `sms_consent` in `/sms/send` and `/sms/send-template`.
- **GDPR**: Support data deletion via DELETE endpoints; log user consent in `users.consent_opt_in`.
- **Audit Logs**: Store all actions in `upload_logs` and `data_cleaning_logs` for traceability.

Error Handling

- **Standard Format**:

```
{
  "error": "string",
```

```
  "details": "string" // Optional
}
```

- **Common Codes**:
  - 400: Invalid input.
  - 401: Unauthorized (invalid/missing JWT).
  - 403: Forbidden (e.g., accessing another user’s data).
  - 404: Resource not found.
  - 429: Too many requests.
  - 500: Server error.
