Email Marketing API Specification

Health Check

Check service health status

```
GET /ping
```

Contacts

Add a new contact

```
POST /contacts
```

Payload example:

```
{
    "email": "contact@example.com",  (required)
    "firstName": "John",
    "lastName": "Doe",
    "phone": "+1234567890",
    "tags": ["tag1", "tag2", "tag3"],
    "autoVerify": true,
    "alertAdmin": true,
    "customFields": {
        "preferredLanguage": "English",
        "dateOfBirth": "2000-01-01",
        "nickname": "JD",
        ...  // other custom fields can be added as needed
    },
    "lists": ["listId1", "listId2", ...]  // Initial list subscriptions
}
```

Response:

```
{
    "contactId": "1234567",
    "email": "contact@example.com",
    "firstName": "John",
    "lastName": "Doe",
    "phone": "+1234567890",
    "tags": ["tag1", "tag2", "tag3"],
    "verified": true,
    "verificationStatus": "verified",
    "verificationAttempts": 0,
    "customFields": {
        "preferredLanguage": "English",
        "dateOfBirth": "2000-01-01",
        "nickname": "JD",
        ...  // other custom fields as returned from the input
    },
    "subscriptions": [
        {"listId": "listId1"},
        {"listId": "listId2"},
        ...
    ]
}
```

Retrieve a contact by email or contactId

```
GET /contacts?email=contact@example.com
```

or

```
GET /contacts?contactId=1234567
```

Response:

```
200 OK
```

```
{
    "contactId": "1234567",
    "email": "contact@example.com",
    "firstName": "John",
    "lastName": "Doe",
    "phone": "+1234567890",
    "tags": ["tag1", "tag2", "tag3"],
    "verified": false,
    "verificationStatus": "notStarted",
    "verificationAttempts": 0,
    "customFields": {
        "preferredLanguage": "English",
        "dateOfBirth": "2000-01-01",
        "nickname": "JD",
        ...  // other custom fields as stored for the contact
    }
}
```

or

```
404 Not Found
```

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

Update contact details

```
PUT /contacts/{contactId}
```

Payload (optional fields, based on what needs updating):

```
{
    "email": "newemail@example.com",
    "firstName": "Johnny",
    "lastName": "Doe",
    "phone": "+0987654321",
    "tags": ["newtag1", "newtag2"],
    "customFields": {
        "preferredLanguage": "Spanish",
        "nickname": "Johnny"
        ...  // other custom fields can be updated as needed
    }
}
```

Verify a contact's email through double opt-in

```
POST /contacts/{contactId}/verify
```

Payload:

```
{
    "token": "unique_verification_token"
}
```

Response (upon successful verification):

```
{
    "message": "Verification successful",
    "verified": true,
    "verificationStatus": "verified"
}
```

Subscriptions

Add a contact's subscription to a list using either email or contactId

```
POST /lists/{listId}/subscriptions
```

Payload (either "email" or "contactId" should be provided):

```
{
    "email": "example@example.com"
}
```

or

```
{
    "contactId": "1234567"
}
```

Remove a contact's subscription from a list

```
DELETE /lists/{listId}/subscriptions/{contactId}
```

Unsubscribe a contact from all lists

```
POST /unsubscribe
```

Payload:

```
{
    "email": "example@example.com"
}
```

or

```
{
    "contactId": "1234567"
}
```

Persistence Requirements

All contacts and subscription data should be stored in DynamoDB as well as Pinpoint.

Test Cases

Test cases should be written in Jest.

Testing for `createContact` Endpoint

1. Positive Tests (Happy Path)

- **Basic Contact Creation**:
  - Submit a well-formed contact with minimum required fields and ensure the contact is created successfully.
- **Full Data Contact Creation**:
  - Submit a contact with all optional and required fields filled in to ensure all fields are correctly processed and stored.
- **Admin Notification**:
  - When the `alertAdmin` flag is set to true, verify that an admin is notified (e.g., through checking if an SNS event is triggered).

2. Negative Tests (Error Cases)

- **Missing Required Fields**:
  - Attempt to create a contact with one or more required fields missing. The API should return an error.
- **Invalid Email Format**:
  - Submit a contact with an improperly formatted email address.
- **Invalid Phone Number Format**:
  - Submit a contact with an improperly formatted phone number (not adhering to the format you've specified).
- **Duplicate Email**:
  - Create a contact with a unique email, then try to create another with the same email. Application should return an error if duplicate.

3. Edge Cases

- **Maximum Data Limits**:
  - If there are limits on the size or number of tags, custom fields, or other array-based properties, try to exceed them.

- **Special Characters**:
  - Submit strings with special characters or non-Latin characters to ensure they're processed and stored correctly.
- **Empty Strings**:
  - Submit empty strings for various fields to ensure they're handled correctly.

4. Security and Robustness Tests

- **Cross-Site Scripting (XSS)**:
  - Ensure that no malicious scripts can be stored in your database through any fields.
- **Rate Limiting**:
  - Rapidly send multiple requests to create contacts to ensure your API has proper rate limiting and doesn't crash or slow down.

5. External System Integration Tests

- **Pinpoint Integration**:
  - Ensure that contacts are correctly created in Amazon Pinpoint, especially if that's where you're primarily storing them.
- **SNS Integration for Admin Notification**:
  - If `alertAdmin` is true, ensure that SNS (for example) is correctly triggered and delivers the notification.

6. Stateful Tests

- **Auto-verify**:
  - If the `autoVerify` field is set to true, ensure that the verification process runs and that the contact's status reflects this.
- **Subscriptions**:
  - If an array of lists (subscriptions) is provided, ensure the contact is added to those lists in Amazon Pinpoint.

7. Cleanup Tests

- **Database Cleanup**:
  - After the test execution, ensure that test data is removed or isolated so it doesn't interfere with other tests or real data.

Testing for `getContact` Endpoint

1. Positive Tests (Happy Path)

- **Retrieve Existing Contact**:
    - Create a contact, then use the `getContact` endpoint to retrieve it by its email. The data returned should match the created contact.

2. Negative Tests (Error Cases)

- **Retrieve Nonexistent Contact**:
    - Try to retrieve a contact with an email that does not exist in the system. The API should return a 404 error.
- **Invalid Email Format**:
    - Submit an improperly formatted email address to the `getContact` endpoint. The API should return an appropriate error response.

3. Edge Cases

- **Special Characters**:
    - Create a contact with special or non-Latin characters in its email, then try to retrieve it. Ensure the email is processed correctly and the contact is returned.
- **Case Sensitivity**:
    - Determine if your system treats emails as case-sensitive. If it does, test with varying cases to ensure consistent behavior.

4. External System Integration Tests

- **Pinpoint Integration**:
    - After creating a contact in Pinpoint, ensure that the `getContact` endpoint correctly retrieves the contact with its data intact.

5. Stateful Tests

- **Data Integrity**:
    - Make updates to a contact (such as changing its name or adding tags) and then use the `getContact` endpoint to ensure the data is consistent with the updates.

6. Cleanup Tests

- **Database Cleanup**:
  - After the test execution, ensure that test data is removed or isolated so it doesn't interfere with other tests or real data.

Testing for `updateContact` Endpoint

1. Positive Tests (Happy Path)

- **Update Existing Contact**:
  - Create a contact, then use the `updateContact` endpoint to update fields like `firstName` , `lastName` , etc. Verify the returned and stored data reflects the updates correctly.

2. Negative Tests (Error Cases)

- **Update Nonexistent Contact**:
  - Try to update a contact using an ID that doesn't exist in the system. The API should return a 404 error.
- **Invalid Data Submission**:
  - Submit data with invalid formats, like an incorrectly formatted email address or phone number. The API should return validation errors.

3. Edge Cases

- **Partial Updates**:
  - Submit only some fields in the update payload and ensure that only those fields are updated, while others remain unchanged.
- **Empty Data Submission**:
  - Send an update request with no data to see how the system handles it. Ideally, it should either reject the request or make no changes.

4. External System Integration Tests

- **Pinpoint Integration**:
  - Update a contact and then verify with Pinpoint to ensure that the changes are reflected there as well.

5. Stateful Tests

- **Data Integrity After Multiple Updates**:
  - Update a contact multiple times in succession and ensure that the final state of the contact reflects all changes correctly and in order.

6. Cleanup Tests

- **Database Cleanup**:
  - After test execution, ensure that test data is reverted or isolated, so it doesn't interfere with subsequent tests or actual data.

Testing for `verifyContact` Endpoint

1. Positive Tests (Happy Path)

- **Successful Verification**:
  - Create a contact, generate a verification token, then use the `verifyContact` endpoint with the token. Verify the returned and stored data reflects the contact as verified.

2. Negative Tests (Error Cases)

- **Invalid Verification Token**:
  - Use the `verifyContact` endpoint with an invalid token. The API should return a verification failure.
- **Verify Nonexistent Contact**:
  - Try to verify a contact using an ID that doesn't exist in the system. The API should return a 404 error.
- **Expired Verification Token**:
  - If tokens have an expiration period, test with an old token to ensure the API handles expired tokens correctly.

3. Edge Cases

- **Already Verified Contact**:
  - Verify a contact and then attempt to verify the same contact again. The system should handle this gracefully, either by ignoring the second request or by sending a notification that

    - the contact is already verified.

4. External System Integration Tests

- **Pinpoint Integration**:
    - After successfully verifying a contact, verify with Pinpoint or any other connected system to ensure that the changes are reflected there as well.

5. Stateful Tests

- **Multiple Verification Attempts**:
    - Try to verify a contact with wrong tokens multiple times before using the correct token. Ensure that the `verificationAttempts` count is updated correctly.

6. Cleanup Tests

- **Database Cleanup**:
    - After test execution, ensure that test data is reverted or isolated so it doesn't interfere with subsequent tests or actual data.

Testing for `addContactToList` Endpoint

1. Positive Tests (Happy Path)

- **Add Subscription with Email**:
    - Create a contact and add their subscription to a list using their email. Verify the list now contains that contact's subscription.
- **Add Subscription with ContactId**:
    - Create a contact and add their subscription to a list using their `contactId`. Verify the list now contains that contact's subscription.

2. Negative Tests (Error Cases)

- **Invalid Email Format**:
    - Try to add a subscription using an email in an invalid format. The API should return a validation error.
- **Nonexistent Email**:

    - Try to add a subscription using an email that doesn't exist in the system. The API should return an error indicating that the email doesn't match any existing contact.
- **Invalid ContactId**:
    - Use an invalid `contactId` format or a non-existing `contactId` to add a subscription. The API should return a validation or not-found error.
- **Already Subscribed Contact**:
    - After adding a subscription to a list for a contact, attempt to add the same contact to the list again. The API should return an error indicating that the contact is already subscribed.

3. Edge Cases

- **Missing Payload Data**:
    - Attempt to add a subscription without providing either an email or a `contactId`. The API should handle this with a suitable error message.

4. External System Integration Tests

- **Integration with Pinpoint**:
    - After successfully adding a subscription, verify with Pinpoint or any other connected system to ensure that the changes are reflected there as well.

5. Cleanup Tests

- **Database Cleanup**:
    - Ensure that after test execution, test data is reverted or isolated, so it doesn't interfere with other tests or actual operational data.

Testing for `removeContactFromList` Endpoint

1. Positive Tests (Happy Path)

- **Remove Subscription with Valid ListId and Email**:
    - Create a contact, add their subscription to a list, and then remove the subscription using the given `listId` and contact's `email`. Verify the list no longer contains that contact.
- **Remove Subscription with Valid ContactId**:
    - Create a contact, add their subscription to a list, and then remove the subscription using the given contact's `contactId`. Verify the list no longer contains that contact.

2. Negative Tests (Error Cases)

- **Invalid ListId**:
    - Try to remove a subscription using an invalid `listId` format or a non-existing `listId`. The API should return a validation or not-found error.
- **Non-subscribed Contact**:
    - Attempt to remove a contact who isn't subscribed to the given list. The API should return an error indicating that the contact isn't subscribed.

3. Edge Cases

- **Remove Subscription without Providing Contact Identifier**:
    - Attempt to remove a subscription without specifying a `listId` or contact identifier (`email` / `contactId`). The API should handle this with a suitable error message.

4. External System Integration Tests

- **Integration with Pinpoint**:
    - After successfully removing a subscription, verify with Pinpoint or any other connected system to ensure that the changes are reflected there as well.

5. Cleanup Tests

- **Database Cleanup**:
    - Ensure that after test execution, test data is reverted or isolated, so it doesn't interfere with other tests or actual operational data.

Testing for `removeContactFromAllLists` Endpoint

1. Positive Tests (Happy Path)

- **Remove Contact with Valid ContactId**:
    - Create a contact and add their subscription to the specified list. Then, remove the contact from that list using their `contactId`. Verify that the contact is no longer subscribed to the list.

2. Negative Tests (Error Cases)

- **Non-existent Contact**:
    - Attempt to remove a contact from the list using a non-existent `contactId`. The API should return a not-found or validation error.
- **Contact Not Subscribed to the Specified List**:
    - Create a contact but don't add them to the specified list. Attempt to remove this contact from the list using their `contactId`. The API should return a validation error indicating that the contact isn't subscribed to the list.
- **Invalid List**:
    - Using a valid `contactId`, try to remove the contact from a list that doesn't exist. The API should return a not-found or validation error for the invalid list.

3. Edge Cases

- **Remove Already Removed Contact**:
    - Create a contact, add them to the list, and then remove them. Now, try to remove them again from the same list using the `contactId`. The API should handle this gracefully, either by indicating that the contact is not in the list or by silently succeeding without any changes.
- **Concurrent Removals**:
    - Simultaneously (or in quick succession) attempt to remove the same contact from the list using their `contactId`. This tests for race conditions. The system should handle it without errors, and the end result should be that the contact is removed from the list.