FCM Dashboard Analysis & Integration Report

1. Executive Summary

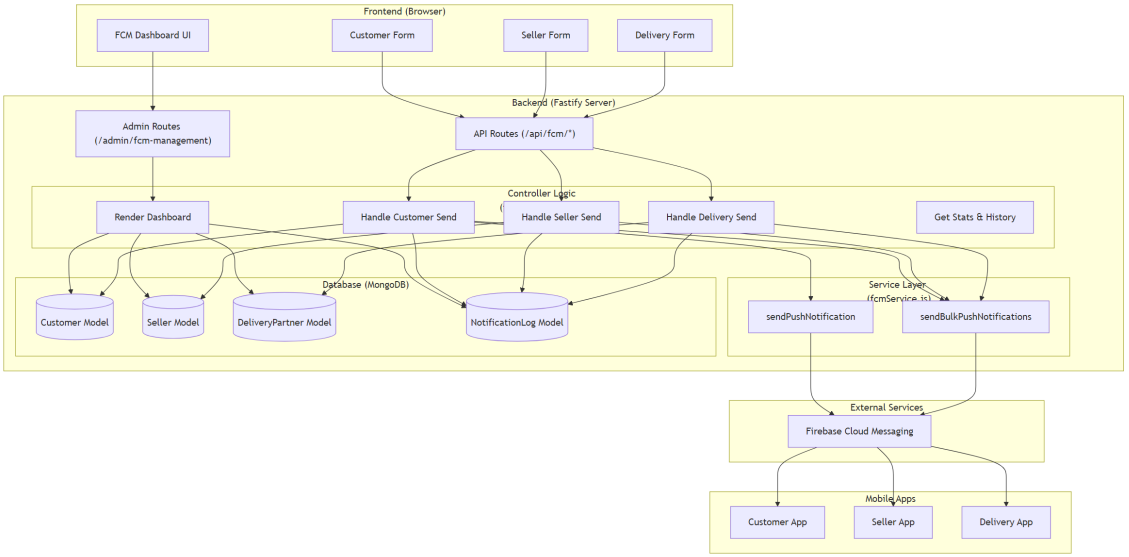
The FCM (Firebase Cloud Messaging) Dashboard is a centralized administrative interface designed to manage and send push notifications to three distinct user groups: **Customers**, **Sellers**, and **Delivery Partners**. It is built as a Server-Side Rendered (SSR) module within the Fastify backend, utilizing MongoDB for data persistence and the Firebase Admin SDK for message delivery.

2. Architecture Overview

The system follows a layered architecture:

1. **Presentation Layer**: A dynamic HTML dashboard rendered by the server (`fcmManagement.js`).
2. **API Layer**: Fastify routes handling form submissions and data retrieval (`routes.js`).
3. **Business Logic Layer**: Controllers processing requests, validating targets, and logging actions (`fcmManagement.js`).
4. **Service Layer**: A dedicated service wrapper around Firebase Admin SDK (`fcmService.js`).
5. **Data Layer**: MongoDB models for Users and Notification Logs.

System Architecture Diagram



4. Integration Details

4.1. Customer App Integration

- **Target Model**: `Customer`
- **Token Storage**: `fcmTokens` (Array of objects) or `fcmToken` (Legacy string).
- **Identifier**: Phone Number.
- **Sending Logic**:
  - **Broadcast**: Fetches all customers with valid tokens and uses `sendBulkPushNotifications` .
  - **Specific**: Looks up a customer by `phone` . If found, sends to their specific token.

4.2. Seller App Integration

- **Target Model**: `Seller`
- **Token Storage**: `fcmTokens` (Array of objects).
- **Identifier**: Phone Number.
- **Sending Logic**:
  - **Broadcast**: Fetches all sellers with tokens.
  - **Specific**: Looks up a seller by `phone`. Sends to all tokens registered for that seller (multi-device support).

4.3. Delivery App Integration

- **Target Model**: `DeliveryPartner`
- **Token Storage**: `fcmToken` (String) or `fcmTokens` (Array).
- **Identifier**: Email (Primary) or Phone.
- **Sending Logic**:
  - **Broadcast**: Fetches all partners.
  - **Specific**: Looks up by `email`. Sends to their token.

5. Workflow Deep Dive

Sending a Notification (Step-by-Step)

1. **User Action**: Admin selects a target (e.g., "Specific Customer") and fills the form on the dashboard.
2. **Request**: The browser sends a `POST` request to `/api/fcm/send-to-customers` with JSON payload: `{ target, phone, title, message, type }`.
3. **Validation**: The backend (`sendToCustomers` in `fcmManagement.js`) validates the input.
4. **Token Retrieval**:
   - If `target === 'all'`, it queries MongoDB for all users of that type with tokens.
   - If `target === 'specific'`, it queries for the specific user by phone/email.
5. **Firebase Dispatch**:
   - The system calls `fcmService.sendPushNotification` (for single) or `sendBulkPushNotifications` (for multiple).
   - `fcmService` constructs the FCM payload (Title, Body, Data, Android Channel ID `goatgoat_notifications`).
   - It uses `admin.messaging().send()` or `sendEachForMulticast()`.
6. **Logging**: The action is logged to `NotificationLog` (though the current implementation in `sendToCustomers` etc. doesn't explicitly save to `NotificationLog` in the snippet, the `sendDashboardNotification` function does. *Note: There is a discrepancy in the code where the specific handlers might not be logging to DB, while the generic `sendDashboardNotification` does. This is a potential improvement point.*).
7. **Response**: The API returns success/failure stats to the frontend, which displays a toast notification.

6. Code Snippets

Route Registration (`src/routes/index.js`)

```
import { adminFcmRoutes, apiFcmRoutes } from '../features/fcm-dashboard/routes.js';

// ... inside registerRoutes
```

```
console.log('Registering FCM dashboard routes...');
await fastify.register(adminFcmRoutes, { prefix: '/admin' });
await fastify.register(apiFcmRoutes, { prefix: '/api/fcm' });
```

Notification Log Retrieval (`src/features/fcm-dashboard/fcmManagement.js`)

```
// Fetching the recent history for the dashboard
historyLogs = await NotificationLog.find()
    .sort({ createdAt: -1 }) // Sort by newest first
    .limit(50)               // Limit to 50 entries
    .lean();
```

Token Aggregation Logic (`src/features/fcm-dashboard/fcmManagement.js`)

```
// Fetching active tokens from different collections
const customers = await Customer.find({ 'fcmTokens.0': { $exists: true } })
    .limit(50) // Note: No sort, gets first 50 found
    .select('name phone email fcmTokens fcmToken createdAt')
# FCM Dashboard Analysis & Integration Report

## 1. Executive Summary
The FCM (Firebase Cloud Messaging) Dashboard is a centralized administrative interface
designed to manage and send push notifications to three distinct user groups: **Customers**,
**Sellers**, and **Delivery Partners**. It is built as a Server-Side Rendered (SSR) module
within the Fastify backend, utilizing MongoDB for data persistence and the Firebase Admin
SDK for message delivery.

## 2. Architecture Overview

The system follows a layered architecture:
1.  **Presentation Layer**: A dynamic HTML dashboard rendered by the server
(`fcmManagement.js`).
2.  **API Layer**: Fastify routes handling form submissions and data retrieval
(`routes.js`).
3.  **Business Logic Layer**: Controllers processing requests, validating targets, and
logging actions (`fcmManagement.js`).
4.  **Service Layer**: A dedicated service wrapper around Firebase Admin SDK
(`fcmService.js`).
5.  **Data Layer**: MongoDB models for Users and Notification Logs.
*   **Token Storage**: `fcmTokens` (Array of objects) or `fcmToken` (Legacy string).
*   **Identifier**: Phone Number.
*   **Sending Logic**:
    *   **Broadcast**: Fetches all customers with valid tokens and uses
`sendBulkPushNotifications`.
    *   **Specific**: Looks up a customer by `phone`. If found, sends to their specific
token.

### 4.2. Seller App Integration
*   **Target Model**: `Seller`
```

```
*   **Token Storage**: `fcmTokens` (Array of objects).
*   **Identifier**: Phone Number.
*   **Sending Logic**:
    *   **Broadcast**: Fetches all sellers with tokens.
    *   **Specific**: Looks up a seller by `phone`. Sends to all tokens registered for that seller (multi-device support).

### 4.3. Delivery App Integration
*   **Target Model**: `DeliveryPartner`
*   **Token Storage**: `fcmToken` (String) or `fcmTokens` (Array).
*   **Identifier**: Email (Primary) or Phone.
*   **Sending Logic**:
    *   **Broadcast**: Fetches all partners.
    *   **Specific**: Looks up by `email`. Sends to their token.

## 5. Workflow Deep Dive

### Sending a Notification (Step-by-Step)

1. **User Action**: Admin selects a target (e.g., "Specific Customer") and fills the form on the dashboard.
2. **Request**: The browser sends a `POST` request to `/api/fcm/send-to-customers` with JSON payload: `{ target, phone, title, message, type }`.
3. **Validation**: The backend (`sendToCustomers` in `fcmManagement.js`) validates the input.
4. **Token Retrieval**:
    *   If `target === 'all'`, it queries MongoDB for all users of that type with tokens.
    *   If `target === 'specific'`, it queries for the specific user by phone/email.
5. **Firebase Dispatch**:
    *   The system calls `fcmService.sendPushNotification` (for single) or `sendBulkPushNotifications` (for multiple).
    *   `fcmService` constructs the FCM payload (Title, Body, Data, Android Channel ID `goatgoat_notifications`).
    *   It uses `admin.messaging().send()` or `sendEachForMulticast()`.
6. **Logging**: The action is logged to `NotificationLog` (though the current implementation in `sendToCustomers` etc. doesn't explicitly save to `NotificationLog` in the snippet, the `sendDashboardNotification` function does. *Note: There is a discrepancy in the code where the specific handlers might not be logging to DB, while the generic `sendDashboardNotification` does. This is a potential improvement point.*).
7. **Response**: The API returns success/failure stats to the frontend, which displays a toast notification.

## 6. Code Snippets

### Route Registration (`src/routes/index.js`)
```javascript
import { adminFcmRoutes, apiFcmRoutes } from '../features/fcm-dashboard/routes.js';

// ... inside registerRoutes
console.log('Registering FCM dashboard routes...');
```
```

```
await fastify.register(adminFcmRoutes, { prefix: '/admin' });
await fastify.register(apiFcmRoutes, { prefix: '/api/fcm' });
```

Notification Log Retrieval (`src/features/fcm-dashboard/fcmManagement.js`)

```
// Fetching the recent history for the dashboard
historyLogs = await NotificationLog.find()
    .sort({ createdAt: -1 }) // Sort by newest first
    .limit(50)               // Limit to 50 entries
    .lean();
```

Token Aggregation Logic (`src/features/fcm-dashboard/fcmManagement.js`)

```
// Fetching active tokens from different collections
const customers = await Customer.find({ 'fcmTokens.0': { $exists: true } })
    .limit(50) // Note: No sort, gets first 50 found
    .select('name phone email fcmTokens fcmToken createdAt')
    .lean();

// ... similar queries for Seller and DeliveryPartner ...

// Normalizing data for the view
customers.forEach(c => {
    // ... logic to extract tokens and push to allTokens array ...
    allTokens.push({
        type: 'Customer',
        name: c.name || 'Unknown',
        // ...
    });
});
```

Sending Logic (`src/features/fcm-dashboard/fcmManagement.js`)

```
export async function sendToCustomers(request, reply) {
    const { target, phone, title, message, type } = request.body;

    if (target === 'all') {
        // ... fetch all tokens ...
        const result = await sendBulkPushNotifications(tokens, { title, body: message, data:
{ type } });
        return reply.send({ success: true, details: result });
    } else {
        // ... fetch specific customer ...
        const result = await sendPushNotification(customer.fcmToken, { title, body: message,
data: { type } });
        return reply.send({ success: true, details: result });
    }
}
```

Service Layer (`src/services/fcmService.js`)

```
export const sendPushNotification = async (fcmToken, payload) => {
    const message = {
        token: fcmToken,
        notification: { title: payload.title, body: payload.body },
        android: { notification: { channelId: 'goatgoat_notifications' } }
    };
    return await admin.messaging().send(message);
};
```