

---

TO  
Fannect

AUTHORED BY  
Jasvinderjit Singh

DATE  
27 Jan 2014

---

The logo for FANNECT, featuring the word "FANNECT" in white, bold, sans-serif capital letters. A small red arrow points to the right, positioned between the 'N' and 'E'.

# Technical Guidance Document

---

*Fannect Mobile Application*

# Table of Contents

---

<b>Introduction .....</b>	<b>3</b>
Purpose.....	3
Scope.....	3
Audience .....	3
<b>Design Overview .....</b>	<b>4</b>
Architectural Goals .....	4
Risks/Constraints.....	4
<b>Topology Diagram .....</b>	<b>5</b>
Request/Response .....	5
Publish .....	6
<b>Application Architecture .....</b>	<b>6</b>
User Graph .....	6
Team Graph.....	7
Data Types.....	7
Non-Relational Data .....	7
Messages.....	7
Logging .....	10
Scheduled Tasks .....	10
Data Usage .....	10
<b>Technology Selection - Backend .....</b>	<b>11</b>
Services .....	11
Solutions .....	11

# Introduction

---

## Purpose

The purpose of this document is to provide a technical guidance on the implementation of the Fannect mobile application and backend services.

## Scope

All documentation and recommendations are based on the requirements gathered during the Discovery phase of the Fannect mobile application project.

## Audience

The intended audiences of this document are Fannect stakeholders and the DEG project team - business analysts and engineers working on the project.

*This portion of the page intentionally left blank.*

# Design Overview

---

## Architectural Goals

**The architectural goals of the mobile application are:**

- Provide an optimal end-user experience
- Operate under potentially low-bandwidth conditions or no connectivity
- Minimize battery consumption

**The architectural goal of the backend services are:**

- Provide a scalable backend
- Provide a highly available backend
- CDN will be used to host static content

**The security goals are:**

- Only minimal required customer data will be collected and stored
- All sensitive customer information will be encrypted and stored securely
- All communications between end-user devices and the backend will be done over a secure channel (HTTPS)
- Application interactions with client information on third-party services (e.g. Facebook, Twitter) will be between the device and the third-party service. No user identifiable information will be sent back to the backend servers.
- Opt-in two-factor authentication

These goals will be achieved by using industry best practices, standard technologies, and IaaS/SaaS services in the cloud.

## Risks/Constraints

Constraints of the application are:

- Number of notifications. The number of notification sent in response to a “post” could potentially be large. Prioritizing and limiting the recipients may be crucial to maintain application responsiveness.
- Message size: Large attachments, e.g. photographs, could affect the performance of the application in low bandwidth conditions.
- Device memory limitations. The limited amount of memory on devices may limit the amount of information stored/cached on the device. This may require the device to be online to access images, older postings, etc. once they are no longer in cache.
- Device bandwidth limitations. The device may be used in areas where there is no connectivity or in low-bandwidth conditions. The mobile application needs to be able to adapt to these conditions, such as: upload/download low-resolution images, display a progress bar if a task is going to take longer than expected, and the ability to cancel a task.

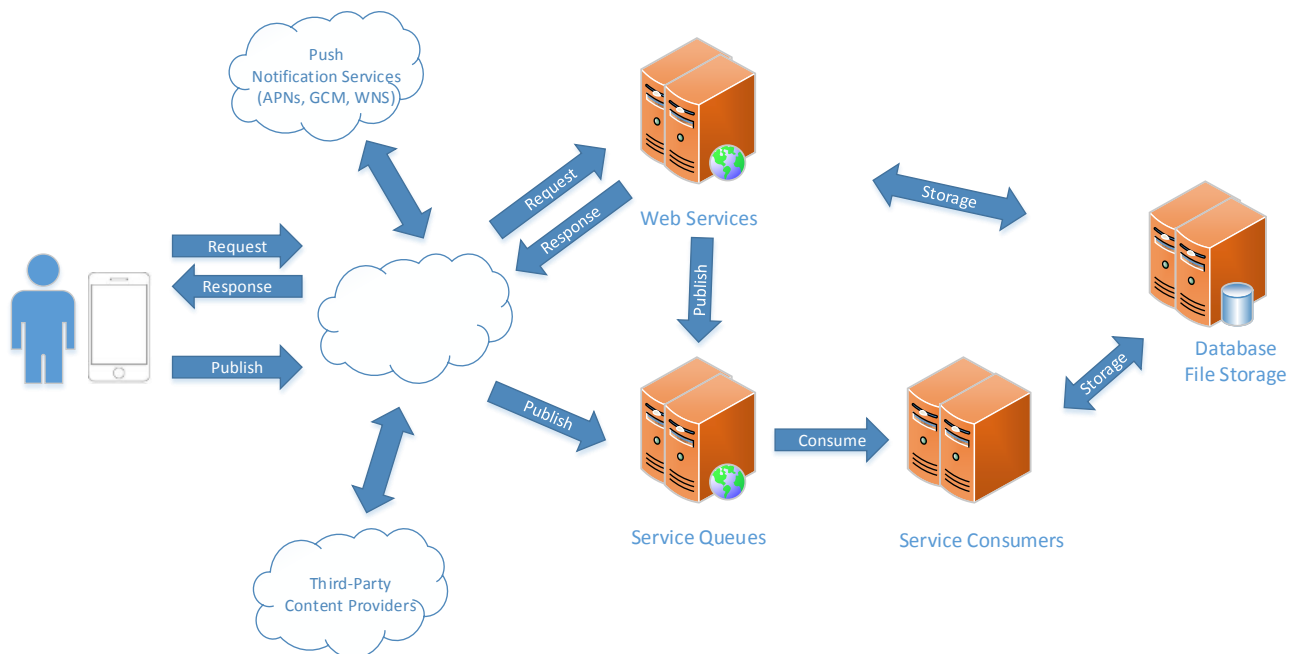
# Topology Diagram

A high level overview of the application environment is shown in

Figure 1: Topology Diagram

Figure 1.

Figure 1: Topology Diagram



The Fannect mobile application will interact with the backend by either making a request and waiting for a response (Request/Response) or by posting a message (Publish.)

## Request/Response

The request/response interaction will be used when an immediate response is needed from the backend. In this scenario, the mobile application interacts directly with one or more load-balanced Web Front-End servers (WFEs) that process the request and return a response. By having a pool of WFEs, the solution is able to scale up or down depending on the workload.

Example use-cases are:

- Authentication
- Retrieve the user profile
- Requesting new/unread messages
- Static content (e.g. pictures)
- Dynamic content (e.g. team lists, game scores, etc.)

## Publish

The publish interaction will be used when no response from the backend is required. This mechanism allows the mobile application to publish information to the backend service buses and immediately return control to the user. The service bus queues the message(s) and makes them available to one or more service consumers to process it. By having multiple service consumers, the solution is able to scale up or down depending on the workload.

Example use-cases are:

- Posting messages/pictures
- Posting comments
- Fetching content from third-party content providers (e.g. team schedules, game scores, etc.)

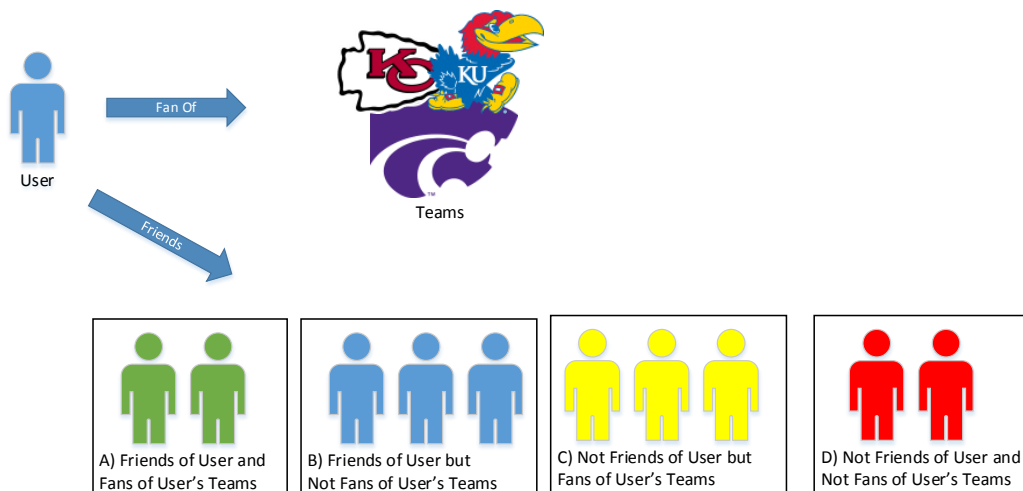
## Application Architecture

### User Graph

The User Graph is shown in Figure 2. Users have the following relationships with other users and teams:

- Each user is a fan of one or more teams
- Each user has zero or more friends
- Other users are related to the user by being
  - Friends of the user and fans of one or more of the user's teams
  - Friends of the user but not a fan of any of the user's teams
  - Not a friend but a fan of one or more of the user's teams
  - Unrelated – not a friend and not a fan of any of the user's teams

Figure 2: User Graph



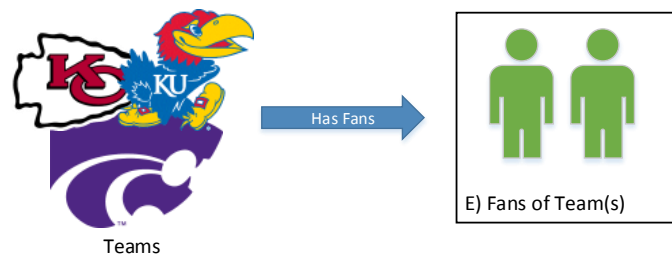
## Team Graph

The Team Graph is shown in

Figure 3. Teams have the following relationships:

- Teams can have zero or more fans

Figure 3: Team Graph



## Data Types

### Date

All date only values will not contain a time component and thus not require a UTC/Local time conversion.

### Date/time

All date/time values will be stored in UTC on the device and on the backend. The date/time values will be converted to local time on the device when displaying.

## Non-Relational Data

All user interactions (event data) can be efficiently stored and retrieved using a non-relational databases (a NoSQL database). All user interactions are converted into messages and sent to the backend services and queued in a service bus queue. Service consumers then pickup these messages and

1. Log messages as events
2. Add the messages to the appropriate feed(s)
3. Update summary data

For example, when a user plays the "Predict the Score" game, the user's prediction is sent to the backend service. The backend service will log the event against the user and update all the appropriate activity feeds.

## Messages

Messages are used to send information from users to the backend service or from the backend service to the user.

### Device Activation

When a user runs the application for the first time, the application will contact the backend service and retrieve a unique device ID (FDEVID). This FDEVID will be used by the backend service to uniquely identify the device. The device will send the FDEVID along with other information whenever it communicates with the backend service.

### Account Registration

When the user uses the application for the first time, they can either create a Fannect account or use it anonymously. In either case, the user will have to provide minimal information to login. Upon successful creation of the account, the user

is assigned a unique Fannect account ID (FAID). The account will be tagged as either a “Full” account or a “Guest” account.

### User Login

When a user logs in to the application, the application will send the FDEVID along with the credentials to the backend service for authentication. Once authenticated, the backend service will associate the user's account (FAID) with the device (FDEVID). A user can be associated with multiple devices by simultaneously logging from multiple devices.

The user can either authorize a device, if they intend to use it as their primary device, or flag it as one-time use. If the user authorizes the device, the user is permanently associated with the device (FDEVID).

### Push Notification

In a typical push notification environment (such as APNs and GCM), the Fannect application, after registering with the push notification service, will send its assigned device token (PNDEVID) to the backend service. The backend service will associate the PNDEVID with the FDEVID. The backend service will use the PNDEVID when it needs to send a push notification to the device.

### Application State Notification

When a user launches the Fannect application, the application will send a message, along with the FDEVID, to the backend service to log its state. This information will be tracked by the backend service so that it can be used to relay state to other users or used by other backend services.

### Message Format

Messages will be in JSON format. Whenever possible, large resources will be referenced using URLs instead of serializing them in the message. This allows the recipient to fetch the resource as needed, or not at all.

Each message will be assigned a unique ID (MID) by the backend system when it is received.

### Message Prioritization

To provide an engaging user experience, messages and notifications need to be prioritized so that those who are actively participating receive the message first. This requires the backend system to maintain a user state that tracks:

- If the user is logged in.
- The device(s) the user is logged in from and the “relevance” of each device.
- The activities the user is performing. For example, the match the user has checked into.

### Messages Distribution

Across all the different features/functions in the Fannect Application, there are four types of message distribution scenarios:

1. From a user to their friends. The user's message is sent to the backend service. The backend service then sends notification messages to the users' friends. In addition, the message is added to each recipient's activity feed. The order in which the friends are notified is optimized for delivery to those who are currently actively using the application.
2. From a user to a team. The user's message is sent to the backend service. The backend service then sends notification messages to the fans of the targeted team. In addition, the message is added to each recipient's activity feed. The order in which the fans are notified is optimized for delivery to those who are currently actively using the application.



3. From the backend service to users. The backend service can also trigger push notifications on its own based on certain conditions. In addition, a message is also added to each recipient's activity feed. For example, when the final score of a user's team is available.
4. From the application to the backend service. The application, independent of a user's action, can send messages to the backend service to, for example, update its state, refresh a feed the user is viewing, or check for new messages.

Since a user can be logged in from multiple devices, push notifications are sent to all authorized devices associated with the user. For one-time use devices, only those the user is currently logged-in on will receive notifications.

When a message is received, it is queued on a Service Queue. These messages can originate from the user, the application, or from the backend service. The messages are then distributed to available Service Consumers for processing. If the message is intended for other users, the following sequence of events will occur:

1. The target recipients of the message are determined
2. Based on the user state, the users are prioritized in the order of most important to least
3. Based on the recipient priority,
  - a. The messages are distributed to the recipients in the backend.
  - b. Push notifications are sent to the recipient's devices.

When the user receives the notification or the application polls for updates for a feed, the new messages that have been added to their feed will be fetched and displayed on the device. To provide a quick response, the messages will be paginated and fetched in descending order by date. Only those messages that are visible will be fetched. Additional messages will be fetched as necessary.

## Activity Feeds

Each user has their own activity feed storage. The activity feed storage has a copy of all messages targeted for the user's activity feed. Any related information to an item in the feed (e.g. comments, tagging, boo/cheer counts) will be maintain in a shared structure.

When the application requests messages for a user's activity feed, the request will include the current message's ID and the number of messages to retrieve. Based on the request, the next N messages are selected directly from the user's activity feed storage. This provides a quick mechanism for retrieving the messages without having to perform a lot of complex queries. For each message, any related information is tagged on and returned to the application.

Another benefit of duplicating the messages is that as user relationships changes overtime (e.g. creating new friends, unfriending someone or team), the historical view is maintained. For example, if User A and User B are friends and User B sends a message, the message will be added to User A's activity feed. Later, if User A unfriends User B, all old messages from User B will remain in User A's feed – messages sent after User A had unfriended User B will no longer be sent to User A's feed. However, if a user deletes a message, it will be removed from all the feeds.

## News Feeds

News feed information is segmented into Headlines, Teams, and Social Streams. Feeds are created as necessary to provide quick retrieval of information.

## Logging

Logging will play an important part in analyzing the mobile application performance and of the backend services.

### Mobile Application

The “snappiness” of the mobile application will be tracked. Factors that affect the performance of the application are the available bandwidth, available/used memory, available/used storage, CPU utilization, amount of time it takes to render a screen, number of times the “Please Wait” pop-up is displayed, and the number of time the user aborts a task because of processing time.

In addition, all errors will be logged. The error log will be sent to the backend for storage and analysis.

### Backend Services

Key metrics will be logged and used to analyze the overall performance of the environment. Factors that affect the performance of the backend and REST web service response times, Service Bus queue average lengths, Service Consumer average processing time. The server loads (CPU and memory usage), if applicable, will also be logged.

This information will be used to scale up/down the various services, trace bottlenecks, and plan for future growth.

## Scheduled Tasks

The backend will include multiple scheduled tasks that will run at regular intervals to retrieve data from third-party external services. The third-party services include:

- Facebook
- Twitter
- Instagram
- Game schedules and scores
- Game preview and statistics
- News

The information will be cached in the backend.

## Data Usage

It is important to minimize data usage at all times. This can be achieved by:

- Fetch only what is needed. Only the items that will fit on the screen should be fetched. As the user scrolls down the list, additional items are fetched.
- Keep the feed light. The feed should only contain pertinent data that can be consumed immediately. All references to resources such as images should be retrieve as needed.
- Use thumbnails. Low resolution version of the images should be retrieved first. Retrieve the high resolution image only on request.
- Caching. The feed and resources that have already been retrieved should be cached to prevent another roundtrip to the backend.
- Limit upload photo sizes. Based on the data connectivity available, the user should be provided with an appropriate set of choices for the image to be uploaded. The images should be scaled down on the device before uploading to the backend. High resolution images may be maintained on the device for later uploading.

# Technology Selection - Backend

## Services

The service stack required to implement the backend are:

### REST Web Services

Load balanced REST web services will be used to implement a high performance and highly scalable solution for hosting the request/response model. These services can be scaled up or down depending on the load and are load balanced across multiple instances for redundancy.

### Service Bus

Service Buses will be used to implement the publish communication model. Messages received from the application or generated by the backend will be queued using service buses. These messages will then be forwarded to subscribers (Service Consumers) for processing.

### Relational Database

Relational databases will be used to store all relational data. This includes account profile information, team information, game schedules, etc.

### NoSQL/Non-relation Database

NoSQL/non-relational databases are ideal for storing structured, non-relational data such as messages in activity and news feeds. The data will be partitioned by the feeds to allow for quick access by feed.

### Blob/File Storage

Blob/file storage will be used to store content uploaded by users and static content generated by the backend. This content will be accessible through a CDN.

### Global CDN

Global CDN will be used to distribute content and services across multiple regions around the world.

## Solutions

	Amazon AWS	Windows Azure	Open Source
<b>REST Web Services</b>	Elastic Compute Cloud	Cloud Service/Web Sites	Apache/Nginx
<b>Service Bus</b>	Simple Queue Service	Service Bus Queues	RabbitMQ
<b>Relational Database</b>	Relational Database Service	SQL Database	MySQL
<b>NoSQL Database</b>	SimpleDB	Table Storage Service	MongoDB
<b>File Storage</b>	Enterprise Block Storage	BLOB Storage	Local storage
<b>Global CDN</b>	CloudFront	Traffic Manager	CDN
<b>Mobile Notification Service</b>	Simple Notification Service	Mobile Services	Service provider APIs