

1. Genyus Network - Facebook Data Migration	2
1.1 System Design	5
1.1.1 Architecture	6
1.1.2 Design Concept/Notebook	13
1.1.3 Installation on Local Machine	20
1.1.4 Reuse Plan	21
1.1.5 WordPress Backend API	22
1.2 Testing	26
1.2.1 Testing Overview	27
1.2.2 Testing Reports	29
1.3 Coding Standards & Version Control Policy	35
1.4 Resources	39
1.4.1 Checklists	40
1.4.1.1 General Process Documentation	41
1.4.1.2 Design Concept Checklist	42
1.4.1.3 Architecture Checklist	43
1.4.1.4 Executable Architecture Checklist	44
1.4.1.5 JIRA Checklist	45
1.4.1.6 Verification & Validation Checklist	46
1.4.1.7 Source Code & Version Control Checklist	47
1.4.1.8 Demo/Acceptance Session Checklist	48
1.4.1.9 Architecture Review Session Checklist	49
1.4.2 Submission Checklist	50
1.5 Handover Details	54
1.5.1 Acceptance Session	55
1.5.2 Client Feedback/Outstanding Issues	57
1.5.3 Facebook App Installation	58
1.5.4 Getting Facebook Approval	59
1.5.5 Migration App Installation	60
1.5.6 Source Code	61

# Genyus Network - Facebook Data Migration



## Project Description

A tool for content migration from Genyus Network's Facebook group to a new WordPress site.

GitHub: <https://github.com/SWEN90014-2021/quokka-webapp-prototype>

JIRA: <https://jira.cis.unimelb.edu.au:8444/projects/SWEN900142021FBQUOKKA/summary>

## Navigation

<a href="#">Communication</a>	<a href="#">Sprints</a>	<a href="#">System Design</a>	<a href="#">Testing</a>	<a href="#">Coding Standards &amp; Version Control Policy</a>	<a href="#">Resources</a>	<a href="#">Handover Details</a>
-------------------------------	-------------------------	-------------------------------	-------------------------	---	---------------------------	----------------------------------

## Client Details

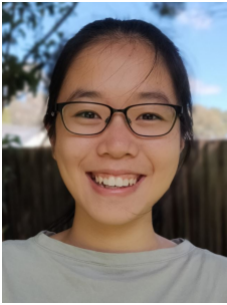

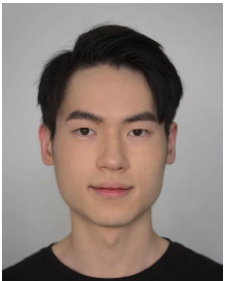

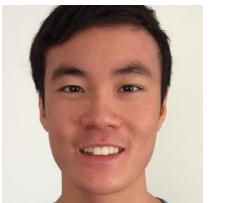
Name	Business	Contact	Role
Caleb Rixon	Genyus Network	<a href="mailto:caleb@genyusnetwork.com">caleb@genyusnetwork.com</a>	Chief Vision Officer

## Supervisor Details

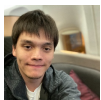
Name	Email & zoom
Glenn Jayaputera	Email: <a href="mailto:glenn.jayaputera@unimelb.edu.au">glenn.jayaputera@unimelb.edu.au</a> Individual (10:30 Wed): <a href="https://unimelb.zoom.us/j/82172004862?pwd=dlk1NGM1Zkcxd051OTJ6c1JrT1NvZz09">https://unimelb.zoom.us/j/82172004862?pwd=dlk1NGM1Zkcxd051OTJ6c1JrT1NvZz09</a> Workshop (14:15 Wed): <a href="https://unimelb.zoom.us/j/84594630699?pwd=YmZieDI4R1hzcV5VS8yYmNTbWx2dz09">https://unimelb.zoom.us/j/84594630699?pwd=YmZieDI4R1hzcV5VS8yYmNTbWx2dz09</a>

## Team FB Quokka Details

Student ID	Photo	Full name	Preferred name	Email	Role
------------	-------	-----------	----------------	-------	------

911806		Haoxin Li	Haoxin	<a href="mailto:haoxin1@student.unimelb.edu.au">haoxin1@student.unimelb.edu.au</a>	Product Owner
977720		Ray Chen	Ray	<a href="mailto:Juitengc@student.unimelb.edu.au">Juitengc@student.unimelb.edu.au</a>	Test Lead
945753		Yifei Yu	Daniel	<a href="mailto:yifeiy9@student.unimelb.edu.au">yifeiy9@student.unimelb.edu.au</a>	Development Environment Lead
1208254		Terran Kroft	Terran	<a href="mailto:tkroft@student.unimelb.edu.au">tkroft@student.unimelb.edu.au</a>	Architecture Lead
913503		Fei Yuan	Fei	<a href="mailto:yuanf3@student.unimelb.edu.au">yuanf3@student.unimelb.edu.au</a>	Scrum Master

## Recent space activity



### Terran Kroft

[Client Feedback/Outstanding Issues](#) updated about 4 hours ago • [view change](#)

[Source Code](#) updated about 4 hours ago • [view change](#)

[Migration App Installation](#) updated about 4 hours ago • [view change](#)

### Yifei Yu

## Space contributors

- [Terran Kroft](#) (3 hours ago)
- [Yifei Yu](#) (3 hours ago)
- [Ray Chen](#) (8 hours ago)
- [Haoxin Li](#) (8 hours ago)
- [Fei Yuan](#) (1 day ago)
- ...



[Facebook App Installation](#) updated about 4 hours ago •  
[view change](#)

[Migration App Installation](#) updated about 4 hours ago •  
[view change](#)

# System Design

# Architecture



This documentation should evolve from and replace your design concept. Note that *not all diagrams types or models are required*; this will depend on your project and should be confirmed with your supervisor.

## Table of Contents

- [Table of Contents](#)
- [4+1 View Model](#)
- [Use Case View](#)
- [Logical View](#)
- [Process View](#)
- [Implementation View](#)
- [Deployment View](#)
- [Executable Architecture](#)
  - [A series of screencasts demonstrating the features of our executable architecture](#)
    - [Web app](#)
    - [WordPress](#)
- [Slides for Architecture Review Sessions](#)

## 4+1 View Model

The system will be illustrated here using the following perspectives:

- [Use Case View](#)
- [Logical View](#)
- [Implementation View](#)
- [Process View](#)

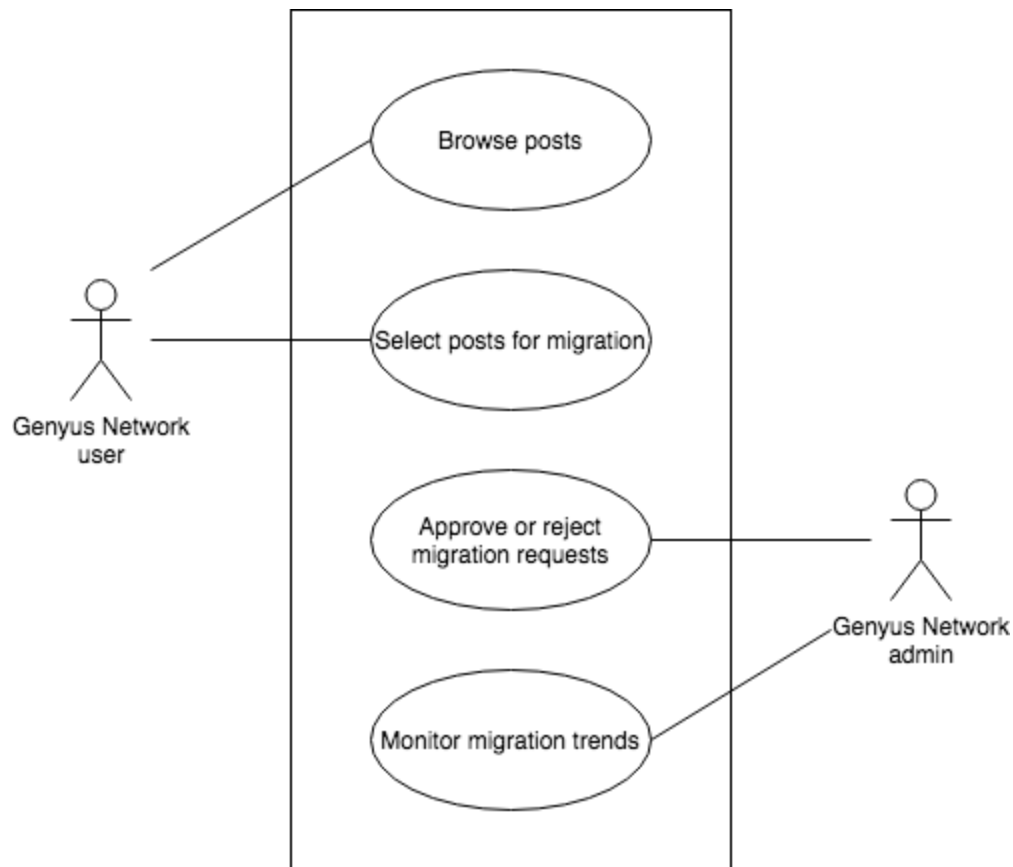
Not applicable:

- [Deployment View](#)

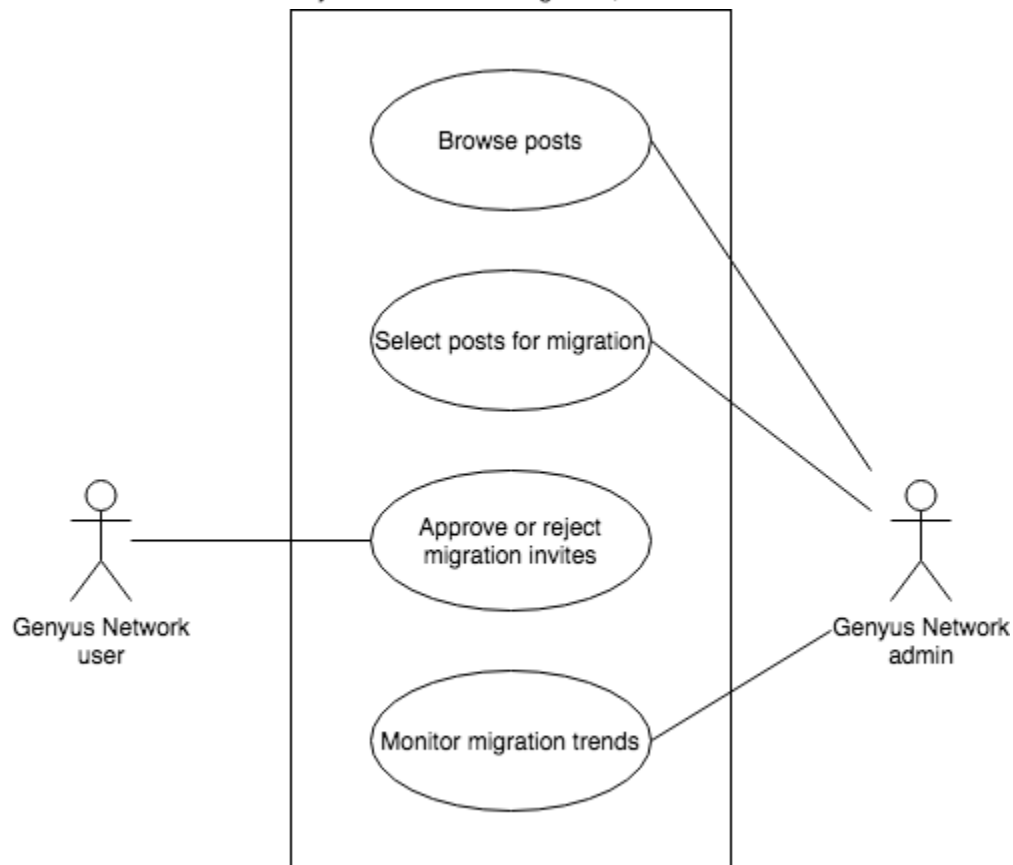
## Use Case View

These use case diagrams represent the high-level user scenarios for the user-driven (left) and admin-driven (right) data migration workflows. As agreed in the workshop on 08/09/2019, all teams will be focusing on implementing **user-driven migration** due to the high level of complexity that admin-driven migration would involve.

In user-driven migration, Genyus Network users can browse Facebook posts they have made on the Facebook group and select those they want to migrate, which generates migration requests. Genyus Network admins are able to review these migration requests, and either approve or reject them.



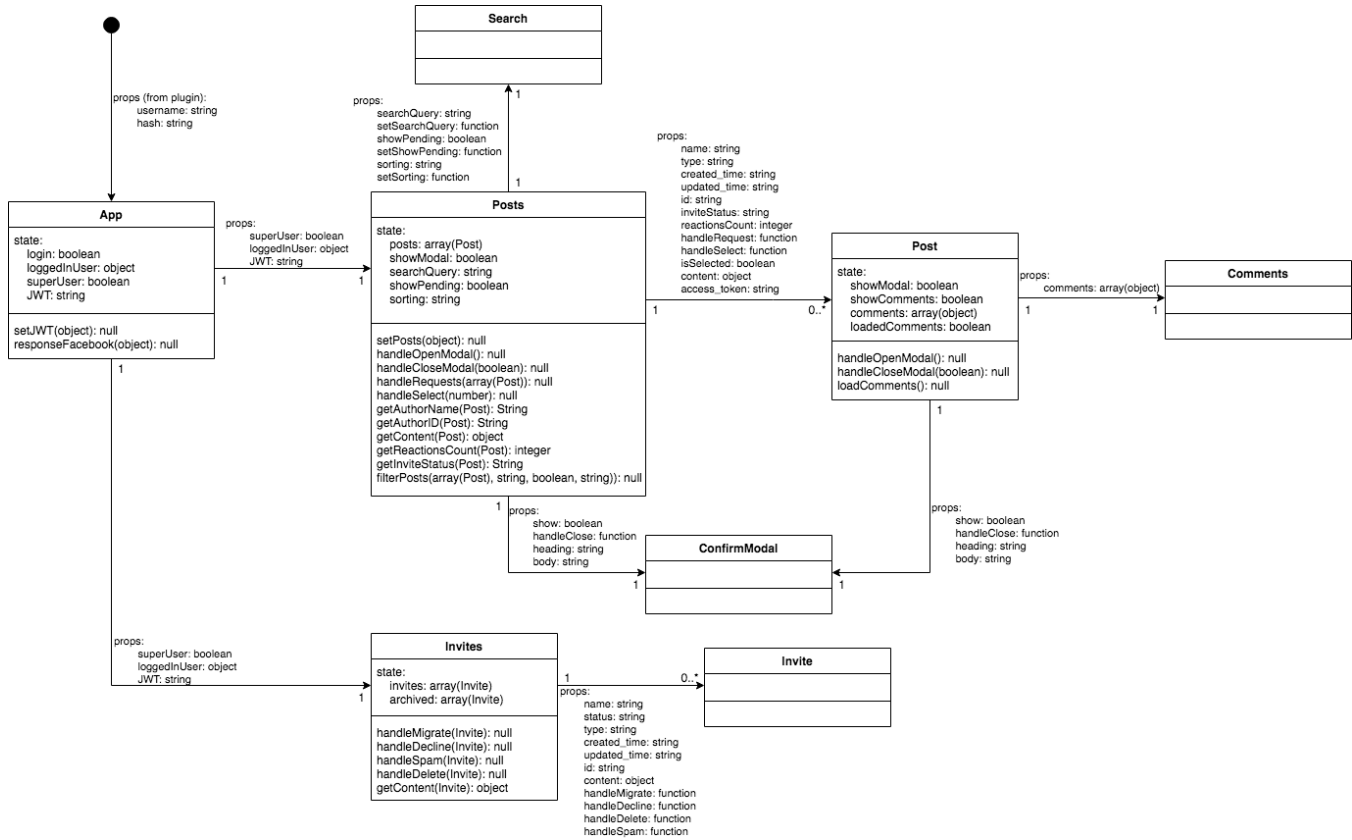
Genyus Network Data Migration, User-driven



Genyus Network Data Migration, Admin-driven

# Logical View

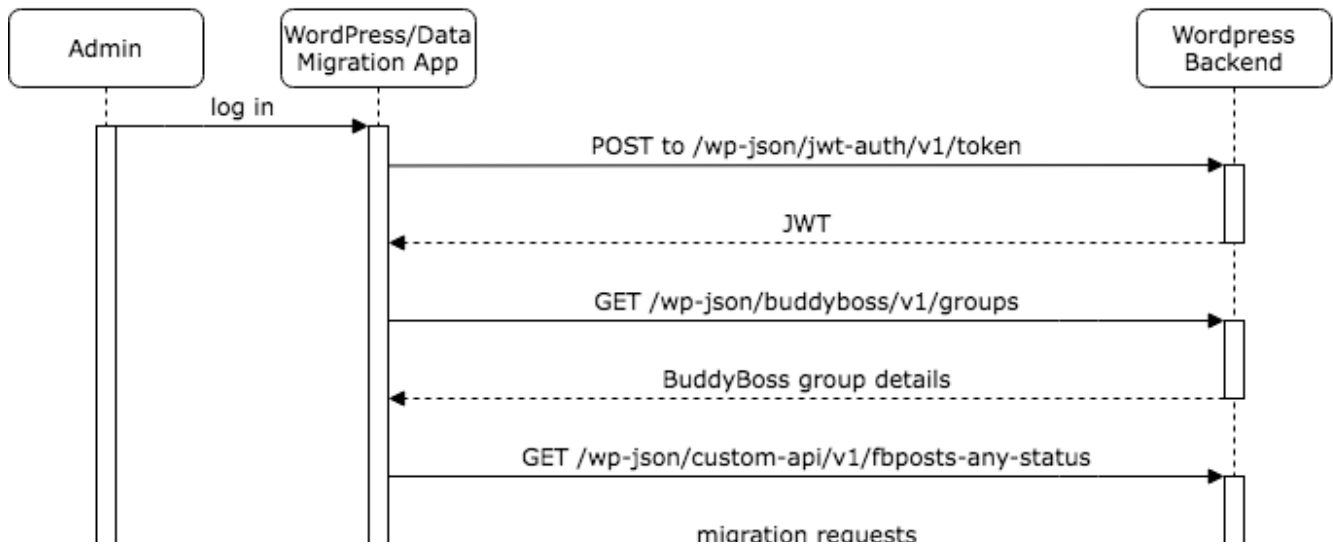
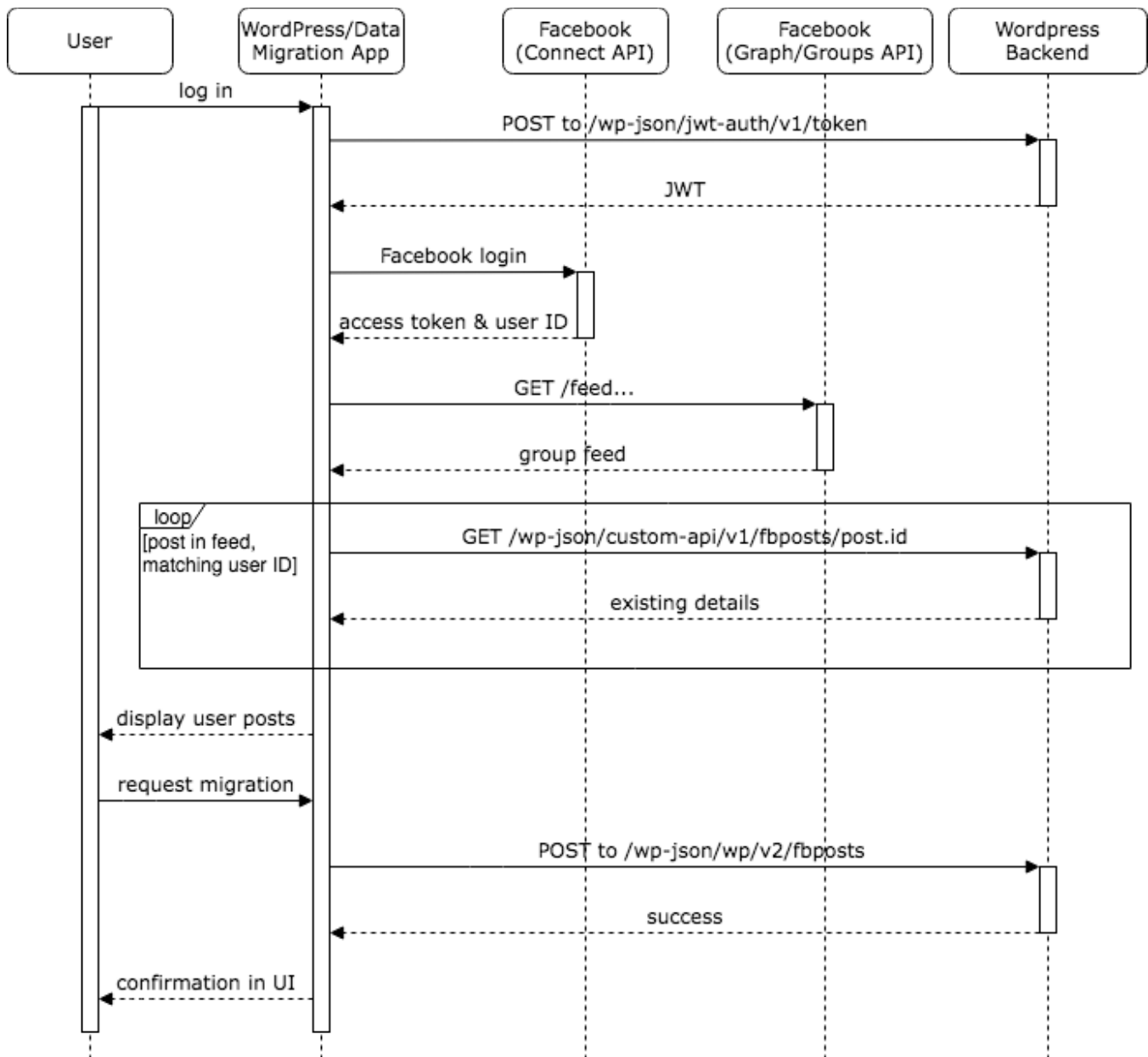
This diagram is not a strict UML design class diagram. Rather, the diagram is a representation of the component hierarchy within the react app (located in /wp-content/plugins/react-embed/src). The diagram summarises how the props and state are used and passed around by the components of the app.

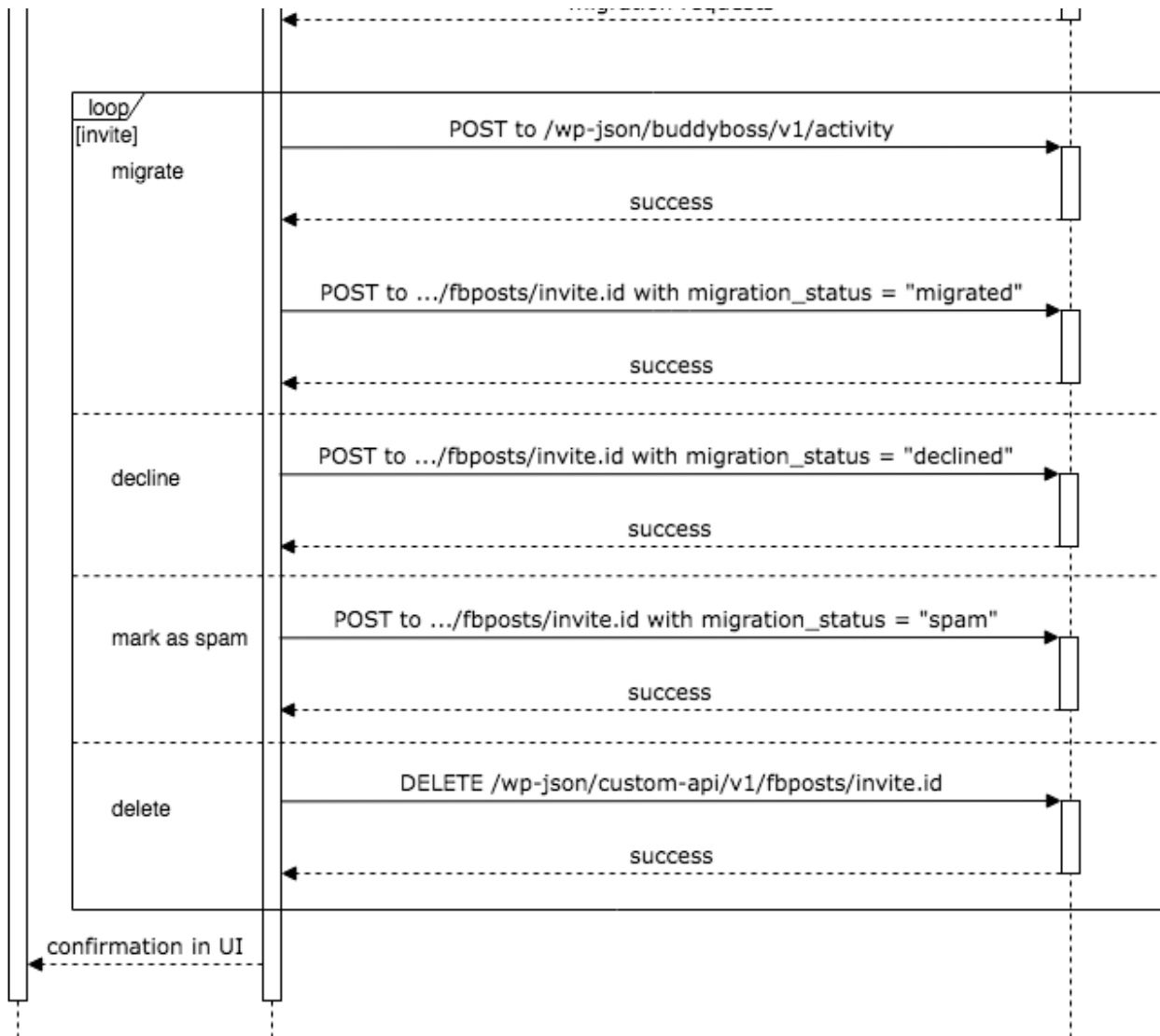


# Process View

These system sequence diagrams illustrate how user-driven migration is performed. As explained in the Use Case View, users initiate migration of posts, while admins review migration requests.







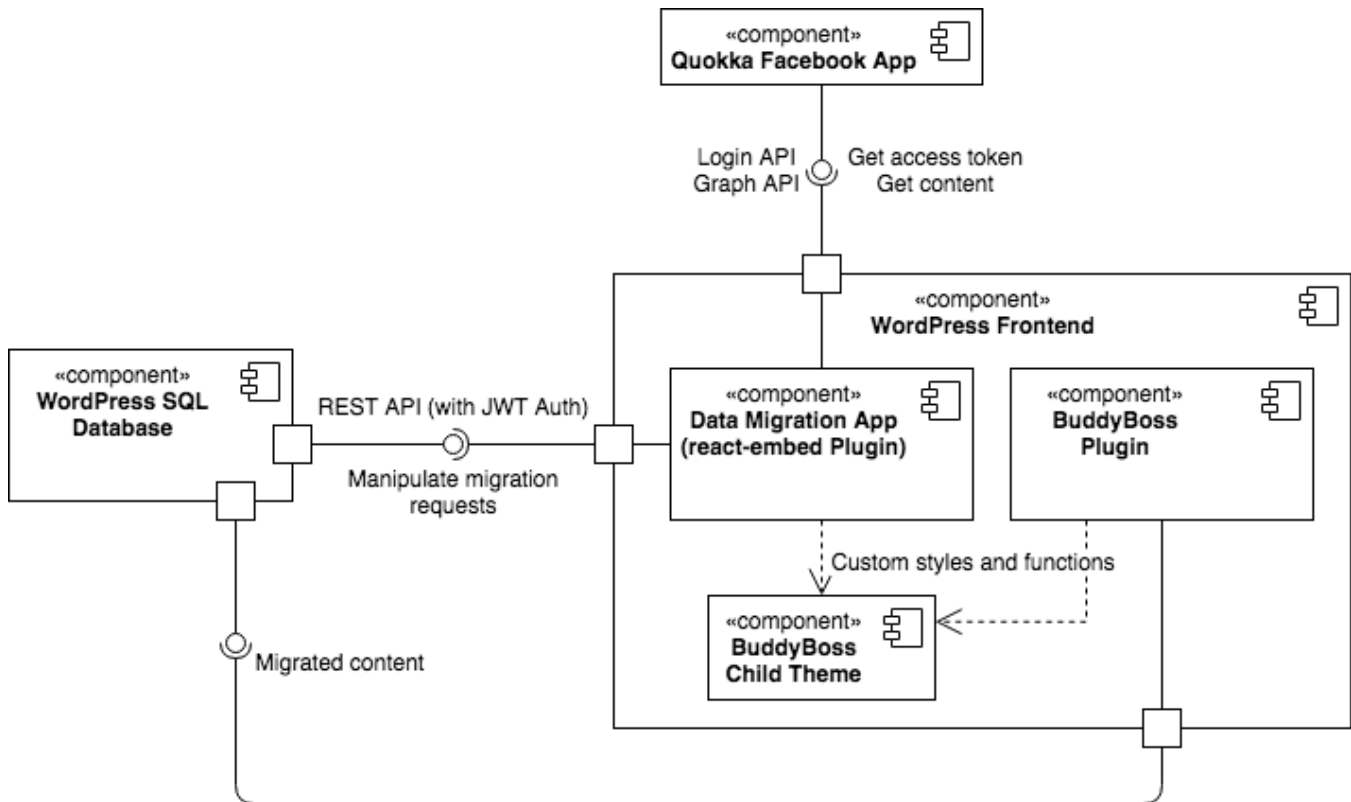
## Implementation View

This component diagram illustrates how the major parts of the system fit together. Some WordPress plugins that we reuse have been omitted to make the diagram clearer.

The system that we have prepared for demonstration is a WordPress site. The data migration process is handled by a **web app** embedded on a page using the **react-embed plugin**. The web app interacts with the **Quokka Facebook app**, which configures the APIs needed for logging in to a Facebook account and retrieving Facebook content. The web app also performs CRUD operations on migration requests saved in the **WordPress database**. These routes are secured by JSON web tokens (JWTs) that are provided to authenticated users.

The **BuddyBoss Child Theme** provides styling for the WordPress site. It also provides customised functionality that makes it easier to receive a JWT in a secure way.

The remaining frontend functionality comes from the **BuddyBoss plugin**, the same plugin that powers Genyus Network's new site. The plugin will allow us to demonstrate that the data has arrived at its destination when migration has been successful.



## Deployment View

The Deployment View is not been considered so far, because our project is primarily a proof-of-concept for demonstration purposes. We would require details about the existing deployment environment used in Genyus' minimal viable product (MVP), which we are unlikely to receive as it is currently still in the early stages of development.

## Executable Architecture

### Data Migration app (React)

- To get posts from Facebook, users will first authenticate with Facebook login, which returns an access token if successful
- The app retrieves users' posts from Facebook using Facebook's Graph API
- The app persists information about pending migration requests and migrated posts using the WordPress REST API

### WordPress backend

- Provides a custom REST API for the Data Migration app
- Uses several WordPress plugins for custom data fields, custom post types, JSON web tokens, etc.

### "Remaining bits of the WordPress site" - for demonstration purposes

- The Data Migration app is embedded within a page in the WordPress site
- Besides this page, the WordPress site also contains social network functionality from the BuddyBoss plugin and theme. Migrated content will be seen here.
- The site uses the built-in user management system & authentication provided by WordPress

### Quokka Facebook app

- The app configures the required settings and permissions that we need from the Facebook login API and Graph API
- Despite being called an "app", this is not a stand-alone application. Rather, it is just a collection of settings on the Facebook Developers' Platform.

A series of screencasts demonstrating the features of our executable architecture

## Web app

## Slides for Architecture Review Sessions

The attached PDF are the slides we used in the Architecture Review Session with our supervisor, Glenn



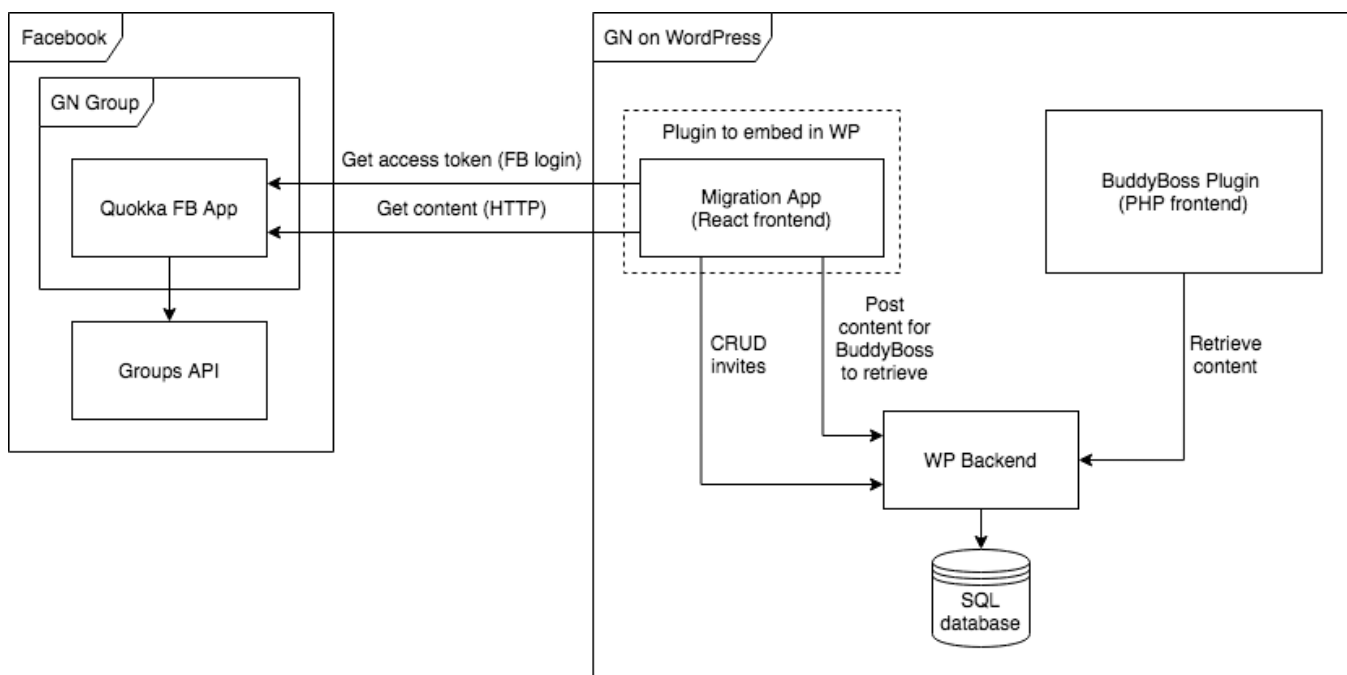
FB\_Quokka\_Arch...cture\_Demo.pdf

# Design Concept/Notebook

Version	Changes
1.0	Starting out with basic details
1.1	Update diagrams to fit updated design implementation
1.2	Add potential solution for WordPress import
1.3	Add testing
2.0	Add Sprint 2 Architectural Changes

## Concept Diagrams

### Sprint 2 Updated Concept



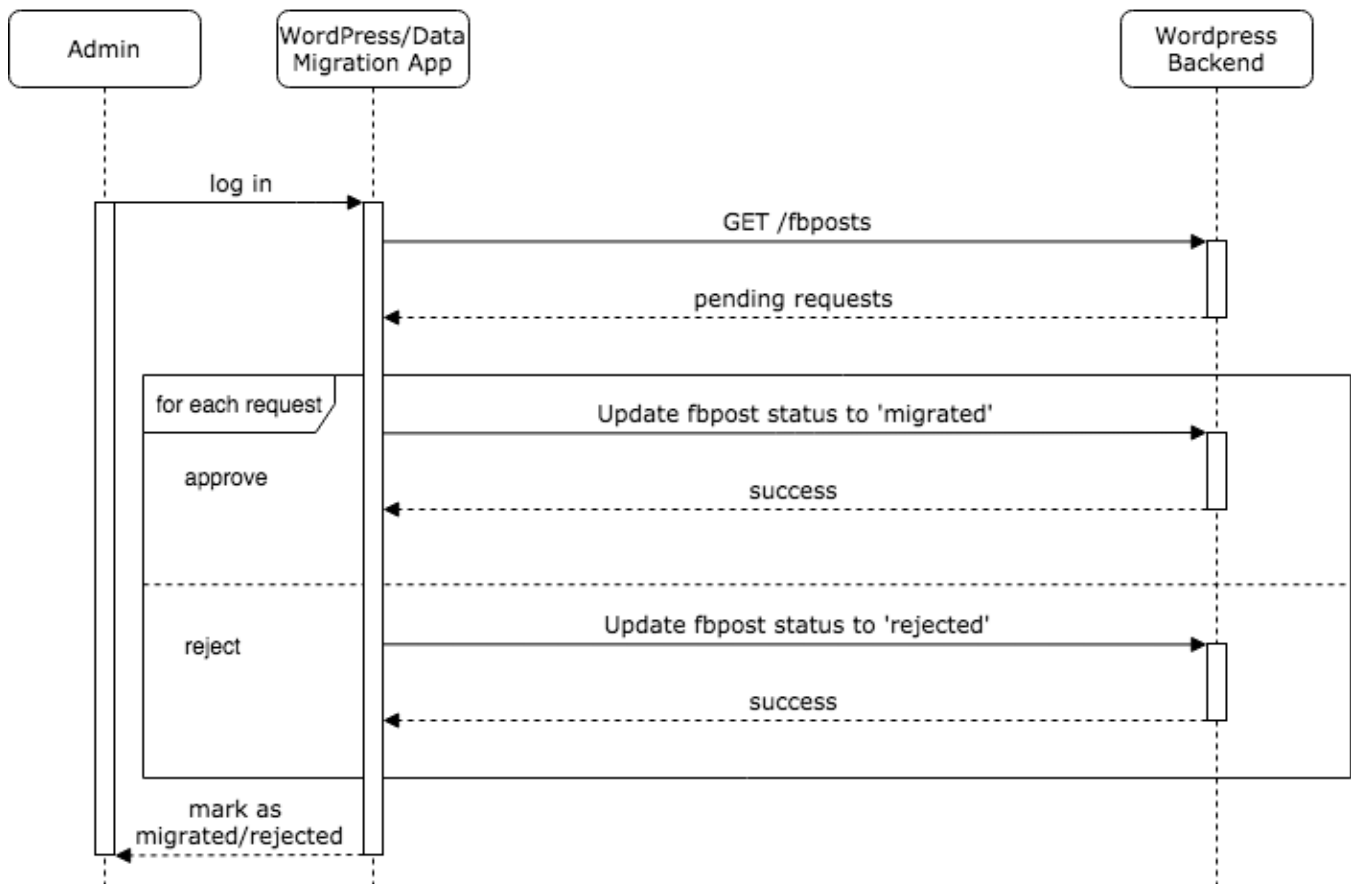
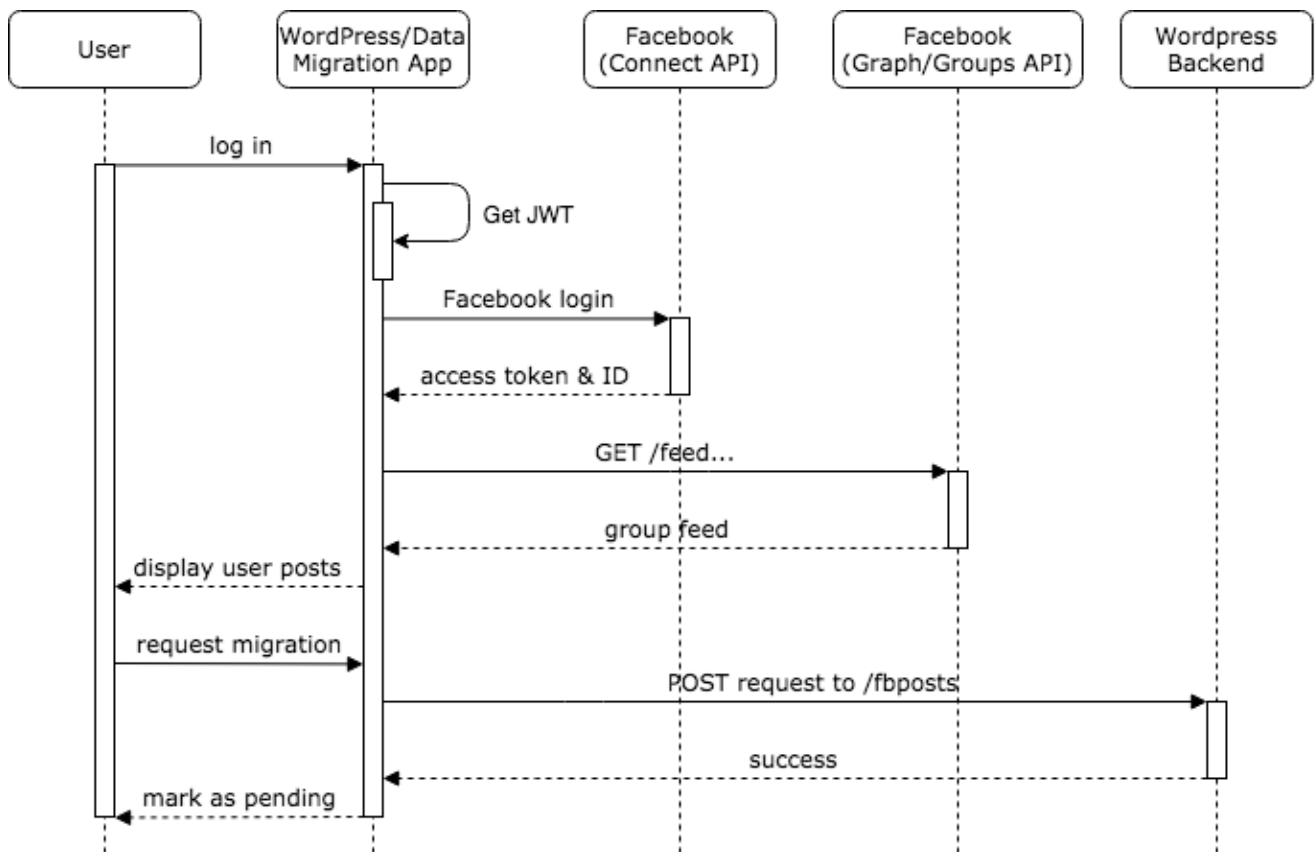
## Overview

Genyus Network (GN) is hosted on Wordpress and makes use of the **BuddyBoss** plugin, which adds social network features to a Wordpress site. Team Quokka's migration tool is a React app embedded in the same Wordpress site via a plugin that allows for React apps to be embedded (react-embed-app). The app logs into Facebook via Facebook's Connect (Login) API and uses the returned token to access data within a group via Facebook's Graph API (Groups API). This returns data from the Genyus Network Facebook group (GN Group), which also has the app installed.

## User-Initiated Approach

The workflow of a user in this approach is as follows:

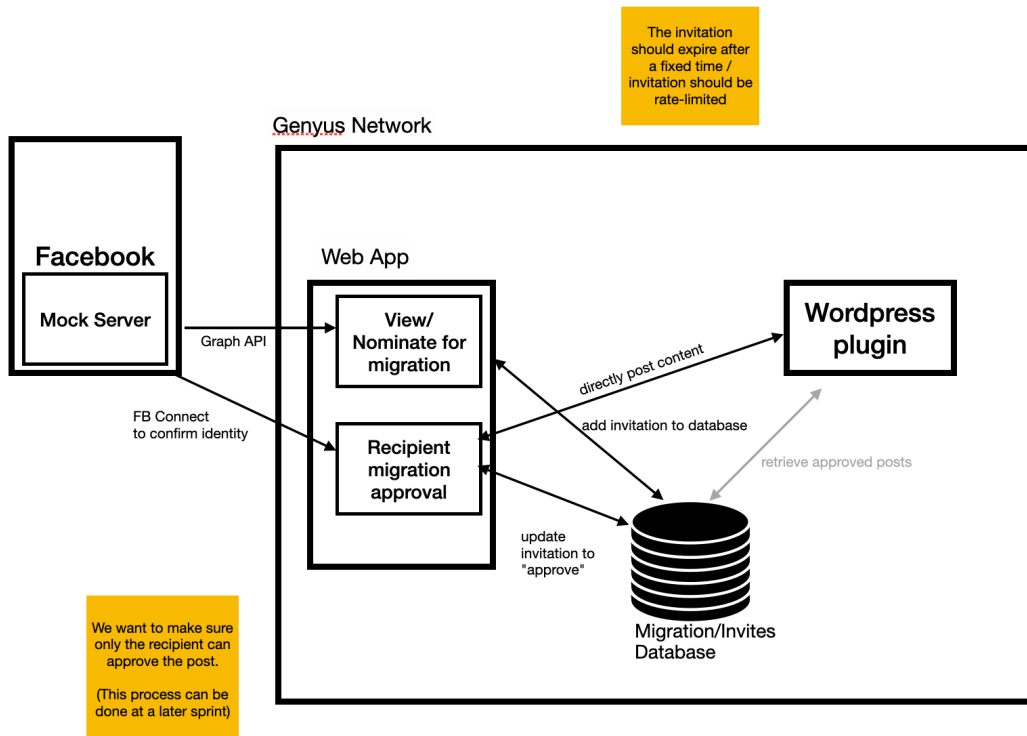
- User logs in to the Genyus Network with their Genyus Network credentials.
- User connects their GN account with Facebook through FB Login.
- The user will be shown a list of their posts in the group. They can select any of them to request migration.
- The admin will be notified when there are posts to migrate.
- If the admin approves the migration, the post will show up in BuddyBoss. Otherwise, it will not.



**Justification for Change**

The first sprint design had some issues that would make later development a bit more challenging. The web app was designed as a separate system between Facebook and Genyus Network on Wordpress, and acts essentially as middleware. It would also have to be involved with both Facebook and Wordpress/BuddyBoss authentication. Following design [Meeting 2021-09-01](#), the team decided to bring the migration app under Wordpress as an embedded page.

## Sprint 1 / Early Concept



Implementation detail on how the Web App and plugin will integrate with Genyus Network is out of scope.

In Concept 1, there are two distinct components to design: the web app and the WordPress plugin. The web app interfaces with Facebook via Graph API to access the posts for a given group (in this case, Genyus Network's public group). An admin can access the web app and view posts on the group and select a post for migration. At that point, the app generates an invite link (with expiry) and saves the invitation details to the database. The admin then sends the invite to the recipient (the poster of the post to be migrated), who will be authenticated and then approve/decline the invite. Authentication is likely to be done by "Sign in with Facebook" to ensure that the poster is the person approving the migration. If they approve the migration, the database entry will be updated indicating that the migration is approved. If not, the database entry will be updated to declined (or the entry can be deleted, but this allows for some logging). Once the post is approved, it will also contact the WordPress plugin via an API POST call.

The second component is a WordPress plugin that has access to the same invitations database. The plugin can poll the database for new invitations that have been approved, and convert the Facebook content into a WordPress post. This can also be set to be individually approved with additional formatting changes available (US2.1). (This component can be substituted to fit another plugin or another destination). Additionally, the WordPress plugin will also be able to make available a REST API for the web app to make API calls for direct updates once the invitation has been approved.



### Note

Keep in mind that there are aspects of the Genyus Network system that we do not yet have access to for the first sprint and thus parts would have to be stubbed. This includes but is not limited to:

- Q5 FB\_QuestionRound2.pdf - We do not know how the GN site will show content.
- Q14 FB\_QuestionRound2.pdf - A user cannot participate in GN without signing up as a member. We do not have access to how GN stores members.

Facebook App Review approval - for getting actual Facebook user names and pictures (the data returned will just be the content itself)

- Until App Review is granted (if it is granted) - there will only be one user class (admin) as it will not be possible for us to determine the original content creator.

The database will store the invitation id (or link, depending on actual implementation) to match the user with invites, invitation status (=rejected, accepted, pending), as well as the post itself. Accepted posts will remain in the database to allow extension into logging if necessary. Pending posts can expire after a fixed time, and rejected posts will also be removed from the database after a fixed time (to allow the recipient to change their mind or to prevent more invitations from being sent).

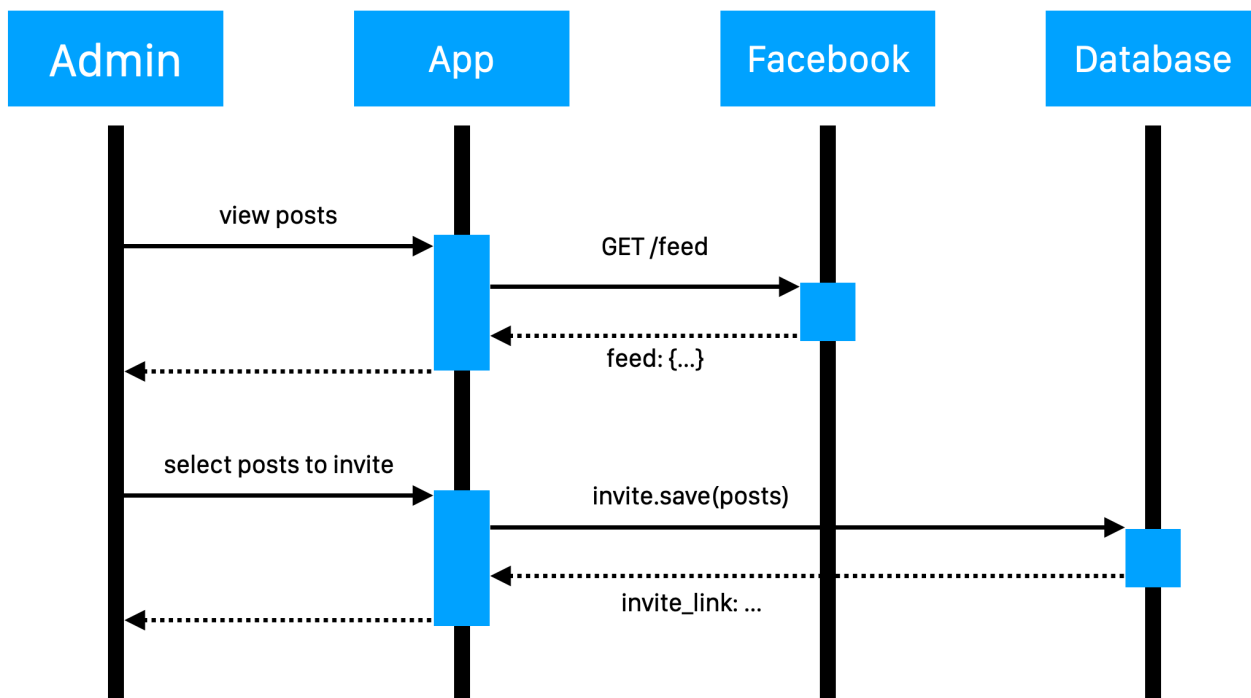
The database will store information such as the graph api identifier to the post itself, the recipient, and a status code. The post\_id is stored as we do not want to explicitly save post data until the invitation has been approved. While the post itself is already visible within the group, and given this project involves migration, it feels more appropriate to wait for the user to approve of the post before we even save it in our database.

name	type	notes
inviteID	string	identifier for Facebook group post currently identical to the Facebook group post identifier from Graph API
status	enum(string)	migration status of the post with values: PENDING, ACCEPTED, REJECTED
inviteCreationDate	date	the date when the migration has been created

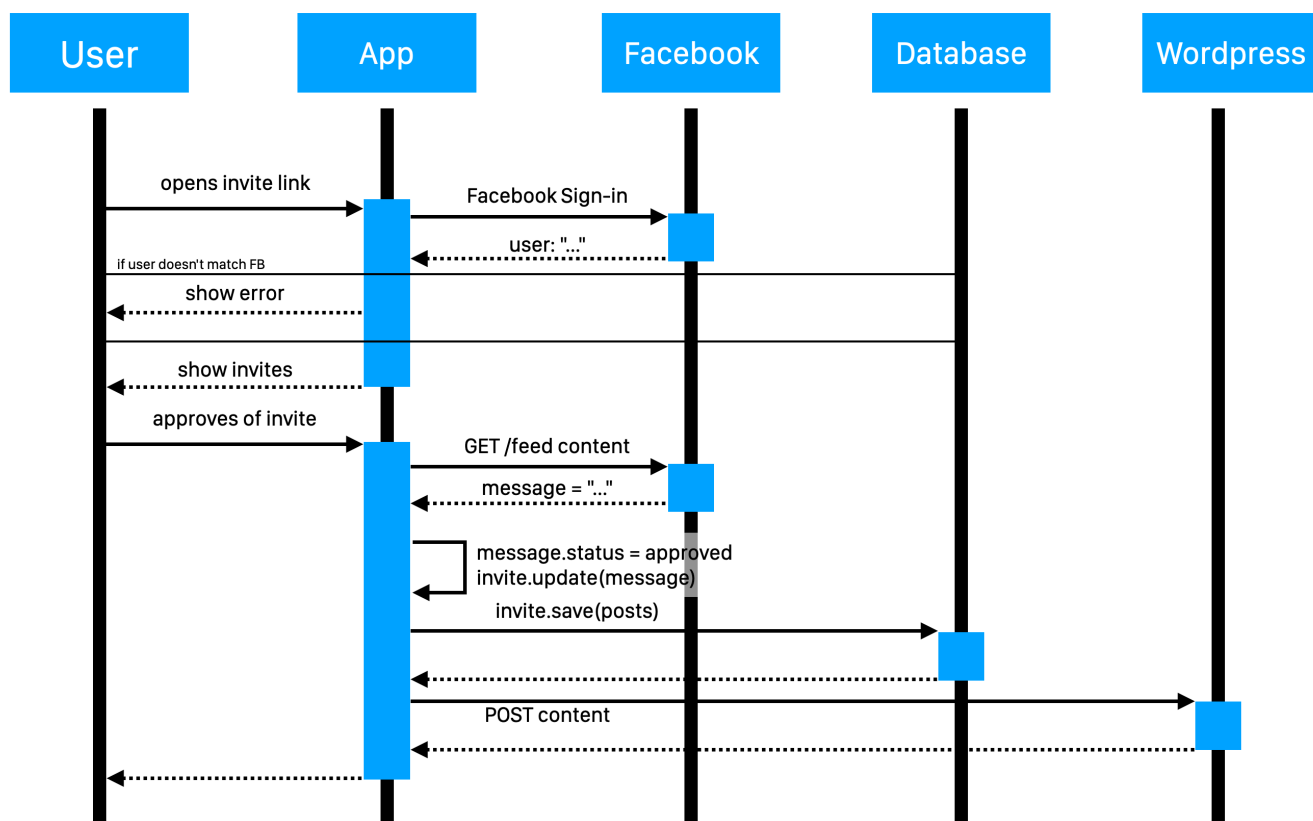
This schema will be updated as development continues.

## Sequence Diagrams

These sequence diagram explain the process detailed above, from a user's and admin's perspective.







## Types of Users

User Type	Description	Permissions
Administrator	Moderator on the Genyus Network.	<ul style="list-style-type: none"> <li>View all posts (within 90 days)</li> <li>Send invitation to migrate</li> <li>View pending invites</li> <li>Access to WordPress plugin</li> </ul>
User	Participant on the Genyus Network. They may already have posted content on the GN Facebook group.	<ul style="list-style-type: none"> <li>Approve invitations sent to them</li> <li>Select posts they have access to for migration</li> </ul>

## Synchronization

This section details how the system will respond to changes in the original Facebook post (i.e. a comment or edit to the post itself)

Facebook posts can be updated from time to time by the user through means of editing posts. If this is done, the admin would be able to query all the posts that are in the database (as archived/migrated posts) to check if the last\_edited field from the Graph API response has changed, and if so, automatically update the post in the database and edit a flag to indicate that there is a change.

## Comments

Further investigation into how comments will be migrated over will be necessary, but this information will be saved in the database (either anonymized or otherwise)

When a post is approved by the recipient, the original post will be saved. The comments will also be saved. *Once App Review authorization is granted*, we will be able to store the names of the commenters. As it is much more challenging to keep comment threads updated, they can be saved and admins may pick good discussion starters to migrate (manually for now, implementation details may have to wait until a better understanding of the destination system) them as anonymized discussion starters. It may not be feasible to maintain identical comment threads nor may it be beneficial.

## Design decision and reasoning

A big challenge to this project is understanding how each component fits into the final envisioned product - the new Genyus Network community. Some of the systems in the separately-developed community we do not have access to at the moment nor do we know much about its implementation as it is bespoke. This design avoids that issue as it is able to work as a separate and standalone tool that can be integrated with the final product if necessary; it can also remain as a separate tool. Within the project requirements itself, the separation of the web app component and the Wordpress component also allow for parallel development. If migration needs to be done to another platform in the future, another tool/plugin can be written that also connects to the database. This reduces coupling greatly (in this case, all the components still have to talk to the database, but the web app can be easily substituted for another system and it would not affect the plugin or any other component).

We also open up the plugin's REST API to the web app so it is able to receive posts directly. This removes the need for polling, but that can still be done from the plugin's perspective if necessary at a later stage.

## Technologies

### Facebook Export

Retrieving data from Facebook is done with Facebook's [Graph API](#) as this is Facebook's only supported method of accessing Facebook resources. While it may be possible to scrape data directly from Facebook, Facebook actively obfuscates and prevents scraping and investing time into this alternative may not be a good use of time. Scraping from Facebook may also be dubious from a legal perspective according to the [terms of service](#).

Getting data from Graph API requires a Facebook Developer account which provides an API key and an access token. Not all permissions are granted and some may need to be requested.



#### LIMITATION

Tools using Graph API may be subject to several limitations:

- The "developer" of the app (in the most general terms) has to be an admin of the group in question.
- Data retention policies of Facebook restrict feed data to be within the past 90 days.

Facebook login will be the choice of identity management as the origin of the posts is Facebook. However, Facebook does require an App Review before allowing apps to have access to the name/photo of the users who post content in groups, which takes time. To prevent this from being a blocker, it is suggested that the executable architecture following the end of Sprint 1 automatically approves of all posts without user authentication.

### WordPress Import

Importing data from web app could be achieved in two ways

#### Method 1 - REST API POST request

Receiving data from HTTP POST request will require to construct a custom post API route in WordPress. The functionality is provided by WordPress. The major benefit of using customer API is it is free and with no limitation, however, since the entire solution was constructed by the team, security is one of the main concern that will need to be addressed.

#### Method 2 - WordPress plugins

[WordPress plugins](#) can be used to add functionality to individual WP sites. As the new Genyus Network platform is to be run on a bespoke WordPress site, it may be possible to integrate the importer as a WordPress plugin.

Potential candidate of the plugin is Zapier. Zapier allow us to link WordPress app with external database (e.g. MongoDB). Hence, data sharing with the Web App could be made possible from WordPress by exchanging and extracting information via the shared database. After extracting the data form external database, the data will be reconstructed into "post" type, a special type in WordPress, and stored in internal database. One of the benefits of the plugin is that Zapier handles the entire process automatically and without the need to know how to code. However, due to the limitation customization more research and testing are require to integrate with exiting WordPress prototype and client's system.

### Stack choice

The WordPress plugin has to be developed in PHP as this is a requirement for WordPress plugins.

The migration tool web app will be developed as a separate component using the MERN stack. The reason for this is that the members of the team have experience in the stack and reduces the time needed to be brought to speed with multiple new technologies (especially with the need to learn PHP).

The MongoDB database will be the interface where the plugin and web app interact. This allows the app and the plugin to be developed in parallel without each component depending on the other for testing (we can use stubs). MongoDB Atlas is an online hosted MongoDB database that can be used to simplify the local development environment. This database is separate from WordPress' internal database which stores posts (or more, in GN's case possibly).

## Testing

Facebook is a fortress. Their data protection policies prevent us from accessing the data we would like from the API, such as post authors and anything other than the user. We are using Express to mock a server that returns a JSON payload similar to what Facebook returns, but with missing data filled in manually to what is expected from Facebook. This will allow us to test.

# Installation on Local Machine

## Setting up WordPress environment

1. Install Local By Flywheel from <https://www.localwp.com/> on your local machine
2. Create a local WordPress site using Local By Flywheel.
3. Name the new site 'QuokkaNetwork'
4. Set the username/password to be admin/admin
5. Clone the GitHub repository on the same level as 'quokkanetwork/app/public' folder.
6. Copy all the files/folders (apart from the files/folders that are available inside the GitHub repository) from the 'quokkanetwork/app/public' folder to the cloned GitHub repository folder.
7. Set hidden files on system to be visible (to ensure `.git/` and `.gitignore` are copied).
8. Delete the 'public' folder.
9. Rename the cloned GitHub repository folder as 'public'.
10. Run the website
11. Go to the Admin dashboard and activate the following plugins:
  - ACF to REST API
  - Admin Columns
  - Advanced Custom Fields
  - BuddyBoss Platform
  - BuddyBoss Platform Pro
  - Child Theme Configurator
  - Custom Post Type UI
  - JWT Auth
  - [react-embed](#)
12. Go to the Appearance and activate '**BuddyBoss Theme Child**' theme.
13. Import Custom Post Type:
  - Hover your mouse onto 'CPT UI' on the left menu, a drop down menu will pop up, and then click onto the 'Tools'
  - There is an 'Import Post Types' section, and copy and paste the content in the 'database/Custom Post Type/CPT UI.txt' onto the 'Import Post Types' section.
  - Click 'Import'.
14. Import Custom Fields:
  - Hover your mouse onto 'Custom Fields' on the left menu, a drop down menu will pop up, and then click onto the 'Tools'
  - There is an 'Import Field Groups' section, click onto the 'Choose file' and select the 'database/Custom Fields/acf-export-{date of import}.json' file.
  - Click 'Import'.
15. Go to 'quokkanetwork/app/public/wp-content/react-embed' folder:
  - Run **npm install** to install the React dependencies.
  - Run **npm run build** to generate static assets for the WordPress plugin.
16. Create a new page
  - Click on 'Pages' on the left menu.
  - Click on the 'Add New' button.
  - Name this page 'Data Migration' (you can name it whatever you like).
  - Insert '[react-app-embed]' in the content field.
  - Click on 'Publish' to publish the page.
17. Add the 'Data Migration' page to the menu:
  - Hover your mouse onto 'Appearance' on the left menu, a drop-down menu will pop up, and then click onto the 'Menus'.
  - Select the 'Data Migration' page and add it to the menu.

## Web app development instructions

1. The web app can still be run separately by executing ``npm start`` in the ``/react-embed`` directory.
2. The page will be live on ``localhost:3000``

## Testing functionality with WordPress

1. Before opening the embedded page on WordPress, ``npm run build`` to build a version that can be embedded.

# Reuse Plan

## Existing "Solutions" for Reference

The [FBomatic](#) Wordpress plugin allows users to migrate group content as individual posts. There is a demo available at the link which spins up a temporary WP blog.

- This solution does not take into account of the important step of **user approval** as it is simply a FBWP migration tool.
- However this can serve as inspiration.

The 'Custom Post Type UI', 'Advanced Custom Fields', and 'ACF to REST API' WordPress plugins allow developers to create a custom post type, custom fields, custom taxonomies.

- This solution allows fast database schema creation.
- The API for the custom post type is created automatically.

The 'JWT Authentication for WP-API' WordPress plugin secures the API endpoints by using JSON Web Token Authentication.

## Plugin substitutes

As part of early prototypes, we will see if it may be possible to directly POST updates from the web app to WordPress rather than through a PHP plugin. If this is possible and does not reduce functionality of the system, we can reuse the web app to also handle posts. As the components are separate, this can be also tested in parallel to a plugin.

## Email

We may want to allow the app to automatically send emails on our behalf (for invitations, alerts, etc.) This can be an option for sending invitations. In Cal's Question Response 2, it is indicated that the invite link can also be copy-and-pasted so this might not be necessary.

- Nodemailer is a node.js module that allows emails to be sent.
- [Emailjs](#) is another module that does the same

## Identity

For identity management for migration, Facebook Login (<https://developers.facebook.com/docs/facebook-login/web>) is an option as posters in the group already have Facebook and just need to sign in to prove they are the poster.

- This requires additional approval from Facebook.

It is understood that Genyus Network has its own user management system that we are able to build upon once we have a working architecture. Their identity management will then be used for determining user permissions (admin user/GN participant user/not logged in)

## Environment

- [Local](#) by Flywheel allows WP instances to be run.
- [BuddyBoss](#) platform, a free WP plugin that the new Genyus Network site uses, allows social network functionality to be added to WP. We will be reusing this plugin for demonstration purposes, to show the end result of the data migration process. However, it is unlikely that we will be given access to the exact details of the client's modifications and configuration of BuddyBoss.
- ~~MongoDB's Atlas platform is a cloud database platform which will make it easier for us to test and view database entries, rather than requiring our environments to be set up with MongoDB.~~ We will be instead be using the backend SQL database provided by WordPress instead, due to smoother integration with the target system. This can be considered component reuse as we do not need to set up a new database, only set up new tables for an existing one.

# WordPress Backend API

This section documents all the WordPress API endpoints.

## JWT Authentication

The WordPress API endpoints are secured by the JWT authentication. A JWT Token is required when a user wants to make a post, update or delete request.

### To get the JWT Token using a plain text password:

```
POST http://genyusnetwork.local/wp-json/jwt-auth/v1/token
Content-Type: application/json
```

```
{
  "username": "<user's username>",
  "password": "<user's password>"
}
```

### To get the JWT Token using hashed password:

```
POST http://genyusnetwork.local/wp-json/jwt-auth/v1/token
Content-Type: application/json
```

```
{
  "username": "<user's username>",
  "password": "<user's hashed password>"
  "custom_auth": "true"
}
```

This will return a JWT Token for the current user. When you want to make a post, update or delete a request, you need to append the JWT Token in the request header.

## FBposts

The FBposts is a database table that stores the migrated Facebook posts.

### To get all of the posts in FBposts:

```
GET http://genyusnetwork.local/wp-json/wp/v2/fbposts
```

### To get a specific post using WordPress post id:

```
GET http://genyusnetwork.local/wp-json/custom-api/v1/fbposts/<WordPress post id>
Authorization: Bearer <user's JWT Token>
```

### To get a specific post using Facebook post id:

```
GET http://genyusnetwork.local/wp-json/custom-api/v1/fbposts/<Facebook post id>
Authorization: Bearer <user's JWT Token>
```

### To add a new post:

```
POST http://genyusnetwork.local/wp-json/wp/v2/fbposts
Content-Type: application/json
Authorization: Bearer <user's JWT Token>
```

```
POST http://genyusnetwork.local/wp-json/wp/v2/fbposts
Content-Type: application/json
Authorization: Bearer <user's JWT Token>
```

```
{
  "title": "<Facebook Post ID>",
  "content": "<FaceBook Post Content>",
  "status": "publish",
  "fields": {
    "created_time": "<Format: 15-08-2021 3:30 pm>",
    "file_links": "",
    "author_id": "",
    "updated_time": "<Format: 19-08-2021 5:30 pm>",
    "author_name": "",
    "type": "",
    "post_content": "<FaceBook Post Content>",
    "post_id": "<Facebook Post ID>",
    "migration_status": "<Choose between 'migrated', 'pending review', 'declined', or 'spam'>",
    "rejection_reason": "<Leave it blank in this case>"
  }
}
```

```
}
}
```

### To update a post using WordPress post id:

POST http://genyusnetwork.local/wp-json/wp/v2/fbposts/<WordPress post id>  
 Content-Type: application/json  
 Authorization: Bearer <user's JWT Token>

```
{
  "title": "<Facebook Post ID>",
  "content": "<FaceBook Post Content>",
  "status": "publish",
  "fields": {
    "created_time": "<Format: 15-08-2021 3:30 pm>",
    "file_links": "",
    "author_id": "",
    "updated_time": "<Format: 19-08-2021 5:30 pm>",
    "author_name": "",
    "type": "",
    "post_content": "<FaceBook Post Content>",
    "post_id": "<Facebook Post ID>",
    "migration_status": "<Choose between 'migrated', 'pending review', 'declined', or 'spam'>",
    "rejection_reason": "<Insert the rejection reason if the post is getting declined or marked as a
spam>"
  }
}
```

### To update a post using Facebook post id:

POST http://genyusnetwork.local/wp-json/wp/v2/fbposts/<Facebook post id>  
 Content-Type: application/json  
 Authorization: Bearer <user's JWT Token>

```
{
  "post_title": "<Facebook Post ID>",
  "post_content": "<FaceBook Post Content>",
  "post_status": "publish",
  "meta_input": {
    "created_time": "<Format: 15-08-2021 3:30 pm>",
    "file_links": "",
    "author_id": "",
    "updated_time": "<Format: 19-08-2021 5:30 pm>",
    "author_name": "",
    "type": "",
    "post_content": "<FaceBook Post Content>",
    "post_id": "<Facebook Post ID>",
    "migration_status": "<Choose between 'migrated', 'pending review', 'declined', or 'spam'>",
    "rejection_reason": "<Insert the rejection reason if the post is getting declined or marked as a
spam>"
  }
}
```

### To delete a specific post using WordPress post id:

DELETE http://genyusnetwork.local/wp-json/wp/v2/fbposts/<WordPress post id>  
 Authorization: Bearer <user's JWT Token>

### To delete a specific post using Facebook post id:

DELETE http://genyusnetwork.local/wp-json/wp/v2/fbposts/<Facebook post id> Authorization: Bearer <user's JWT Token>

## Buddy Boss API:

### To upload a post to Buddy Boss without group:

POST <http://quokkanetwork.local/wp-json/buddyboss/v1/activity>  
Content-Type: application/json  
Authorization: Bearer <user's JWT Token>

```
{
  "component": "activity",
  "type": "activity_update",
  "content": "this is content"
}
```

### To upload a post to Buddy Boss with group:

POST <http://quokkanetwork.local/wp-json/buddyboss/v1/activity>  
Content-Type: application/json  
Authorization: Bearer <user's JWT Token>

```
{
  "component": "groups",
  "primary_item_id": 1,
  "type": "activity_update",
  "content": "this is content"
}
```

note:

primary\_item\_id: the associate group id for the post

### To upload a post to Buddy Boss with image(s):

POST <http://quokkanetwork.local/wp-json/buddyboss/v1/activity>  
Content-Type: application/json  
Authorization: Bearer <user's JWT Token>

```
{
  "component": "groups",
  "primary_item_id": 1,
  "type": "activity_update",
  "content": "this is content 15",
  "bp_media_ids": [270, 271, 272]
}
```

note:

bp\_media\_ids: An array of images id associate to the post, ex: "bp\_media\_ids": [<image\_id\_1>, <image\_id\_2>, <image\_id\_3>.....]

### To upload image:

POST <http://quokkanetwork.local/wp-json/buddyboss/v1/media/upload>  
Content-Type: application/json  
Authorization: Bearer <user's JWT Token>

```
{
  "file": image_file_in_binary
}
```

note:

file: in binary format

### To get group info:

GET <http://quokkanetwork.local/wp-json/buddyboss/v1/groups>  
Authorization: Bearer <user's JWT Token>

```
{
}
```



**To update a post:**

PATCH <https://genyusnetwork.flywheelsites.com/wp-json/buddyboss/v1/activity/:id>

Authorization: Bearer <user's JWT Token>

Content-Type: application/json

```
{  
  "user_id": 5  
}
```

note:

user\_id: the author id of the Facebook post that you are going to assign to

:id: the post id that was created by admin

# Testing

[Testing Overview](#)

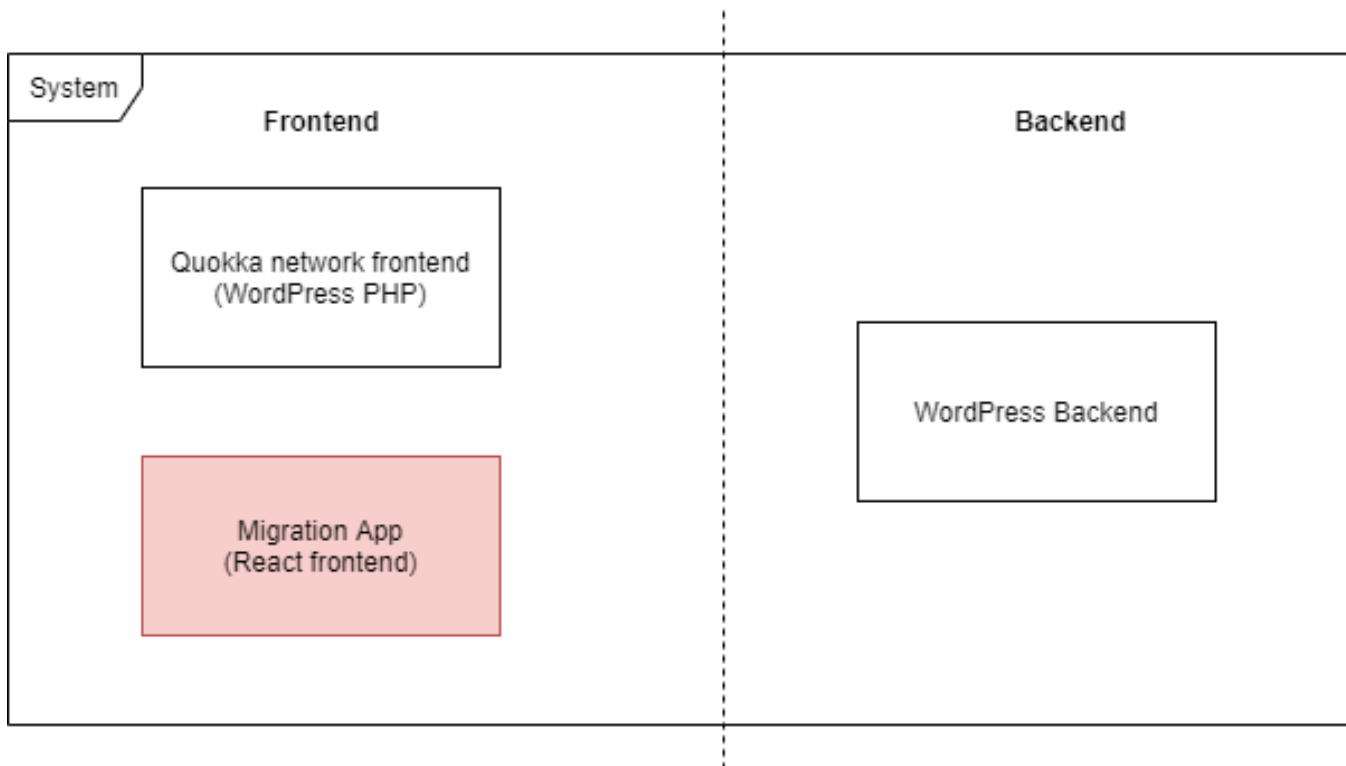
[Testing Reports](#)

# Testing Overview

## 1) Table of Content

- 1) Table of Content
- 2) Overview and Justification
  - Migration App:
- 3) Testing Rules
  - Who will write tests, for which section?
  - How failed tests are handled?

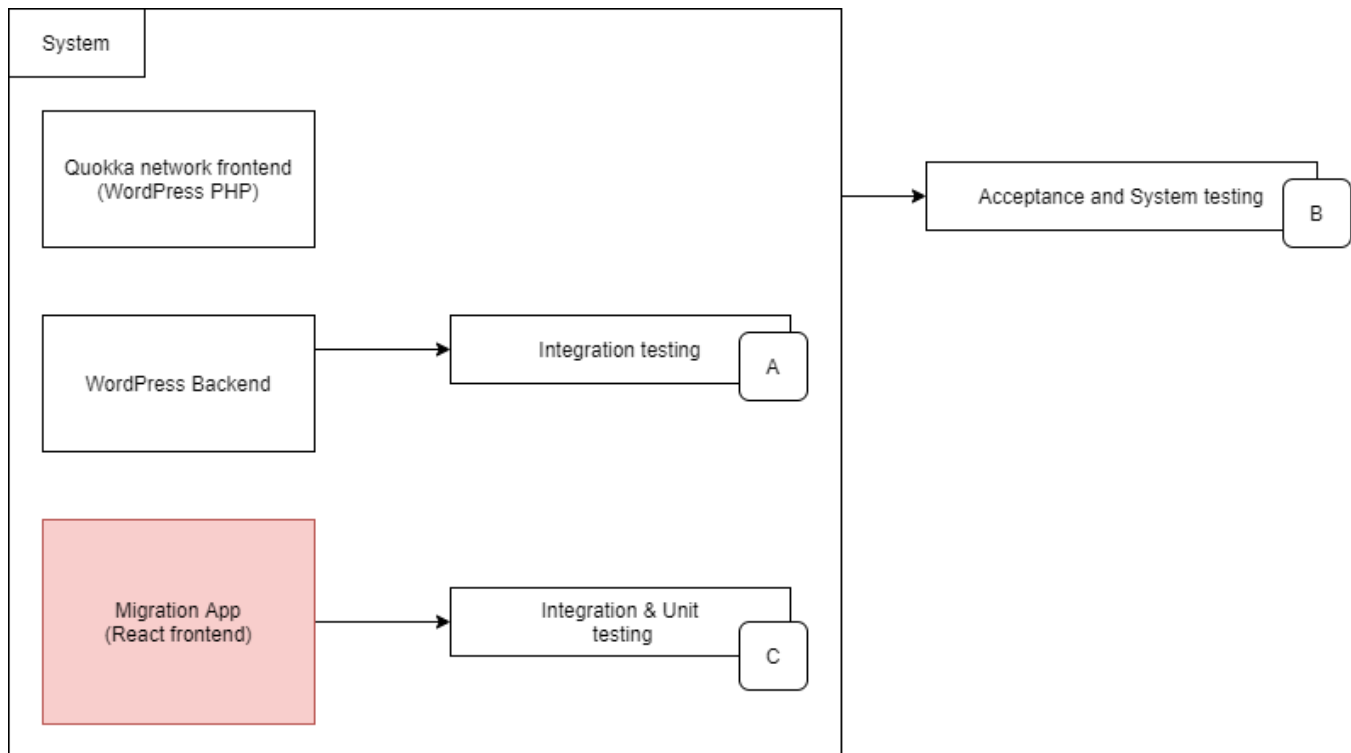
## 2) Overview and Justification



Our system could be distinguish into three different components

- **Quokka network Frontend:**
  - It is design to mimic the client's actual website for the purpose of demonstrating that our solution, migration App, could be integrated to the current Genyus network. The structure and design concept of our Quokka network frontend is similar to client's Genyus network which most of the functionalities are powered by WordPress themes and BuddyBoss plugins. Therefore, no code is require for this components , hence integration and unit test could not be performed.
  - Some of the key functionalities are
    - Displaying the Facebook posts
    - Provided user login interface
- **WordPress Backend:**
  - The main purpose of the backend is provided API end points for the frontends to access the database for CRUD operations. Currently, most of the functionalities in the backend were achieved by WordPress plugins hence unit test could not be performed. However, integration testing will still be conducted for each API end points to ensure each plugin is working as indented.
  - Some of the key functionalities are
    - Storing, retrieving, updating and deleting Facebook post
    - Provided JWT access token to migration app
    - Provided user login functionality to Genyus network frontend
- **Migration App:**
  - Migration app is the place where most of the migration functionality happened. Unlike other components it is designed and created by our team, hence it is also the focal point of our testing. For more information please referred to section C Migration App testing below.
  - we will take condition coverage into consideration when contracting the testing suite or possibly multi-condition coverage.
- **Entire System:**
  - Automated system testing and acceptance testing will be performed for our system which will include all the components above. The aim of the testing is to ensure all the components are working as indented and the test will focus on the key functionalities as listed above.

The picture below is a summary of the testing we will conducted for each components. Each section will be explained in more detailed in the [Testing Reports](#) page.



### 3) Testing Rules

#### Who will write tests, for which section?

- At first, we wanted to let people test each other's code, i.e. the author of the code ideally should not also test the code. This is due to the fact that the programmer will likely to repeat the same mistake or forget the same things when they write tests, as they write the code. In addition, testing is a destructive process where the main goal is to find faults, whereas programming is a constructive process where the main goal is to build a system. It can be challenging to switch from one perspective to the other.
- For testing the migration app, we followed the principle by asking Haoxin to learn React and test the app written in React, by Terran and Fei. By doing so, we ensure that the focal point of testing - Migration App, is tested by a separate tester.
- However, due to time constraints and resource constraints, it is infeasible to ask Haoxin, Fei or Terran to test the backend which is constructed by Yifei and Ray. As results, Yifei and Ray completed backend testing themselves to ensure it can be completed on time.

#### How failed tests are handled?

- Firstly, the testers should examined the bug, ensuring that the tests are written correctly.
- If tests are indeed written correctly, the developers will be notified of the failed tests and they will try to fix them.
- If developers cannot resolve the failed tests either, it will be documented in Jira as an outstanding issue.

Throughout the course of the project, failed tests have been able to be resolved by Step 2. Step 3 is never reached.

# Testing Reports

## 1) Version History

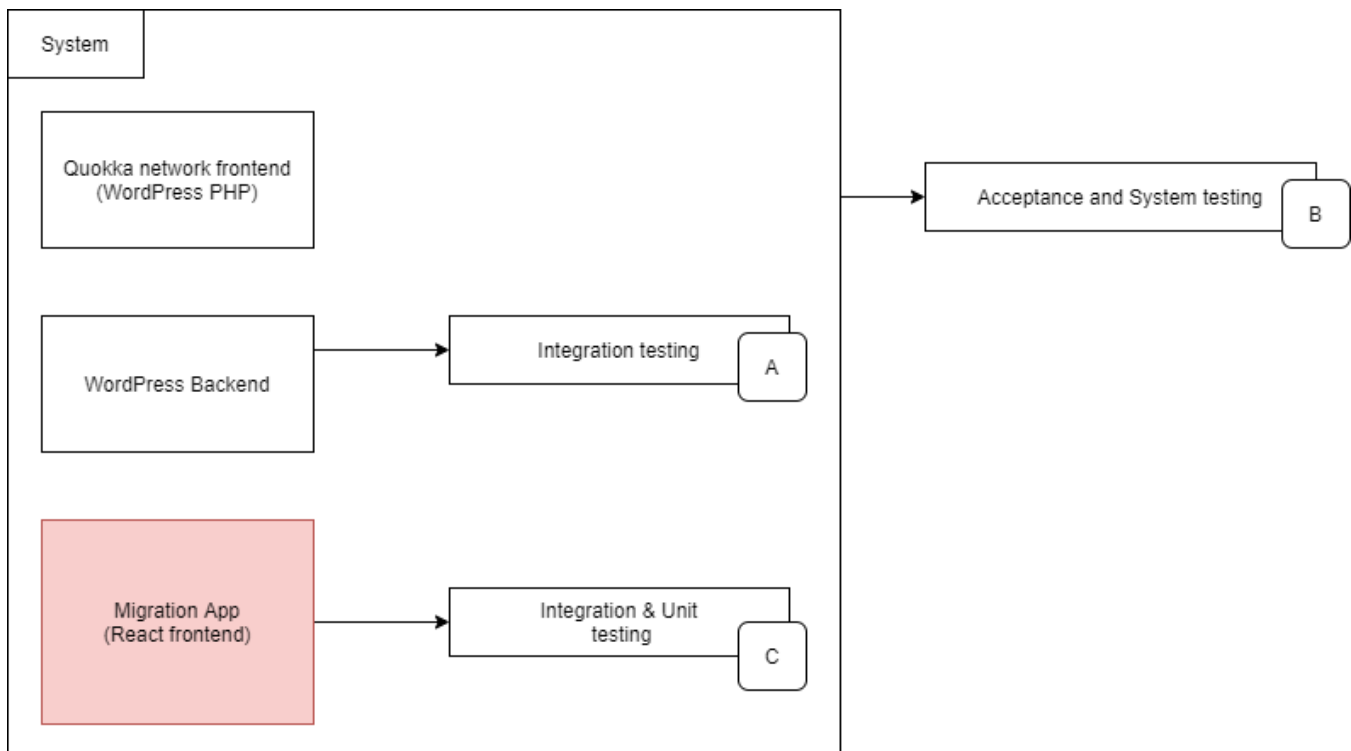
Version	Changes
1.0	Starting out with basic details
1.1	Adding details for individual section testing (including reports)
1.2	Modifying the testing for Migration App
1.3	Update section C

## 2) Table of Contents

- [1\) Version History](#)
- [2\) Table of Contents](#)
- [3\) Overview of sections](#)
  - [A. Integration testing for WordPress Backend](#)
  - [B. System testing for the entire system](#)
  - [C. Migration App testing \(unit and integration tests of components\)](#)

## 3) Overview of sections

The picture below is a summary of the testing we will conducted for each components



### A. Integration testing for WordPress Backend

Testing build version: 1.0

Program build version: Sprint 3 final delivered version

**Aim(s):**

- The focal point for this testing is API endpoints.
- To test all the API endpoints that the frontend website is using.
- Integration testing is performed.

#### Tool(s):

The tools that we are using for our backend API testing are Postman and Newman.

- Postman: An application used for API testing
- Newman: A command-line collection runner for Postman

#### Testing:

We have created two collections of tests:

- Auth: The first collection is called 'Auth', it is to test the JWT authentication.
  - The test includes:
    - Get the JWT token using the correct username and password.
    - Get the JWT token using an incorrect username or an incorrect password.
    - Get the JWT token using the correct username and hashed password.
    - Get the JWT token using an incorrect username or an incorrect hashed password.
- FBposts: The second collection is called 'FBposts', it is to test our database that stores the migrated Facebook posts.
  - The test includes:
    - Create a post with a valid JWT token.
    - Create a post with an invalid JWT token.
    - Get all posts.
    - Get a specific post using Facebook post ID or using WordPress post ID.
    - Update a specific post using Facebook post ID or using WordPress post ID.
    - Delete a specific post using Facebook post ID or using WordPress post ID.

#### How to reproduce tests:

The tests can be run with typing commands. They are not manual tests. In order to reproduce the tests results, please refer to the instructions and command on the GitHub testing/wp\_backend\_integration\_testing/README.md file.

#### Testing Report(s):

Please **download** the attachment files below for testing details.

##### Auth

##### FBposts



Auth-2021-10-22...1-22-314-0.html



FBposts-2021-10...1-58-841-0.html

## B. System testing for the entire system

Testing build version: 2.0

Program build version: Sprint 3 final delivered version

#### Aim(s):

- The focal point of this testing is to test the entire system in the perspective of end users by simulate how the user will use our website.

#### Tool(s):

- CodeceptJS

CodeceptJS have been used to construct and perform the system testing for our project. The reason we chose CodeceptJS as our system testing framework is because it allows us to easily construct a human readable script to simulate and mimic the action of an end user using our system. Once we input the instruction to CodeceptJS, it will search through the HTML source files and located the object which we would like to interact with such as clicking a button or selecting a post. As shown in the example test case below, each of the test is written as a linear scenario of the user's actions on our website and the character "I" at the start of each action represent the actor that perform the following action.

Test case	Testing Inputs
Verify Migration Page is Accessible	<pre>I.amOnPage('/'); I.fillField('Email Address', BOB_USERNAME); I.fillField('Password', BOB_PASS); I.click('Log In'); I.see('Welcome to Quokka Network!');</pre>
Verify Facebook Login Functionality	<pre>I.amOnPage('data-migration/'); I.see('LOGIN WITH FACEBOOK'); I.click('Login with Facebook'); I.switchToNextTab(); I.see('Log in to use your Facebook account'); I.see('Email address or phone number:'); I.fillField('email', BOB_FB_EMAIL); I.fillField('pass', secret(BOB_FB_PASS)); I.click('Log In'); I.switchToNextTab(); I.see('Bob Dillon');</pre>

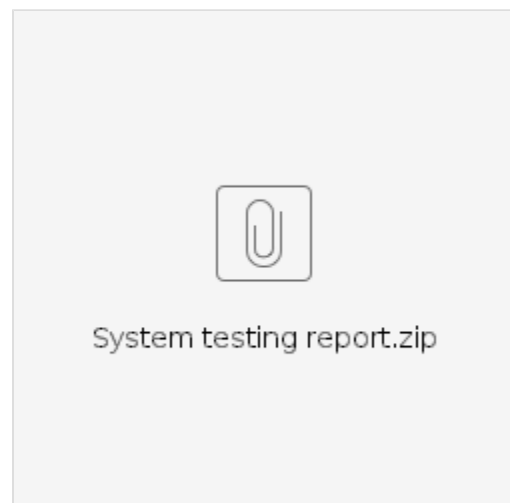
#### How to reproduce tests:

The test needs to be run with type commands. They are not manual tests. In order to reproduce the tests results, please refer to the instructions and command on GitHub README file.

#### Two collections of testing have been built

- Quokka-migration-APP-testing
  - The aim of this collection is to simulate a normal user using our migration App to migrate the posts and an admin approves/rejects the migration requests made by the normal user.
  - Two users has been involve in this testing Bob and admin, Bob represents a public user who does not have admin privilege and try to perform various action when using the Quokka migration APP. Admin simulates the admin user who main purpose in this testing is to accept or reject the migration request made from Bob.
- WordPress-backend
  - The Aim of this testing is to verify if the posts, migrated from previous testing (Quokka-migration-APP-testing), have been successfully displayed in Quokka network.
  - Two users has been involve in this testing Bob and Alex, Bob is the user form previous testing collection and he is also a member of Quokka group. His purpose of the testing is to verify if the posts, both public and private group posts, have been displayed in Quokka network correctly. Alex, an anonymous user who is not a member of Quokka group, thus unable to access to the Facebook posts posted to Quokka group by Bob, however he could access to the posts that are posted to public group. His purpose of the testing is to verify if the private group has function correctly.

#### Testing report



#### Demo video



System testing video.mkv

## C. Migration App testing (unit and integration tests of components)

Testing build version: 2.0

Program build version: Sprint 3 final delivered version

### Aim(s):

The main goals of testing Migration App (built with React) are to:

- Ensure that an individual component behave correctly when users perform an action or when the component(s) is rendered.
- Ensure that the interactions between components are as expected when users perform an action or when the component(s) is rendered.
- Achieve at least 50% statement coverage.

### Tool(s):

We have decided to use Jest and React Testing Library to test the components developed in the migration app.

- Jest: Jest is a JavaScript test runner recommended by React official documentation. It is very popular and Create React App (a tool with which our frontend React app is built) already includes Jest. ([Jest Official Website](#))
- React Testing Library: The library provides a set of helpers to test our React App. It is not a replacement with Jest - it needs to work with Jest. It is also the recommended choice on React official documentation and come with Create React App. ([React Testing Library Introduction](#))

There are two main approaches of testing React app: one with Enzyme, a library that allows shallow rendering (i.e. renders one component only, no children component will be rendered) and checking the internal states of components ([Reference](#)), . The other choice is using React Testing Library which encourages testers to test the app in a way that resembles how the user uses the app in production, and it discourage shallow rendering. React Testing Library is officially recommended by React, hence it became our choice in testing ([React - Testing Overview](#)) . The main reason behind the recommendation is that in production environment, the user will never interact with an isolated component with no children component rendered, and the user will also never be able to adjust the internal states of the components. The tests should resemble the way users interact with the app, and by doing that we can be more confident in our app.

### How to reproduce tests:

The tests can be run with typing commands. They are not manual tests. In order to reproduce the tests results, please refer to the instructions and command on GitHub README file.

### Distinction between unit and integration tests

The distinction between a unit test and an integration test can be challenging to establish in testing React application. Testers have different views on such matter, and React recommends that it is up to the team to decide, based on the specific project at hand. We have decided to define:

- Unit := a single, isolated component
- Integration := interactions between more than 1 components

Note that more than half of our tests are integration tests. This is due to the following reasons:



- Testing a component (e.g. Posts) which renders at least one children component in production often requires an integration test. Otherwise, mocking of the child component would be required but that is against the principle of React Testing Library.
- Kent C. Dodds, the creator of React Testing Library, has recommended to write lots of integration tests as they offer the best trade off between cost and return ([Write tests. Not too many. Mostly Integration.](#)).

## Code Coverage:

The goal of achieving at least 50% statement coverage has been achieved. The figure below illustrate the coverage achieved for different components.

### All files

80.14% Statements (238/287) 62.99% Branches (88/127) 80.73% Functions (86/389) 81.48% Lines (228/279)

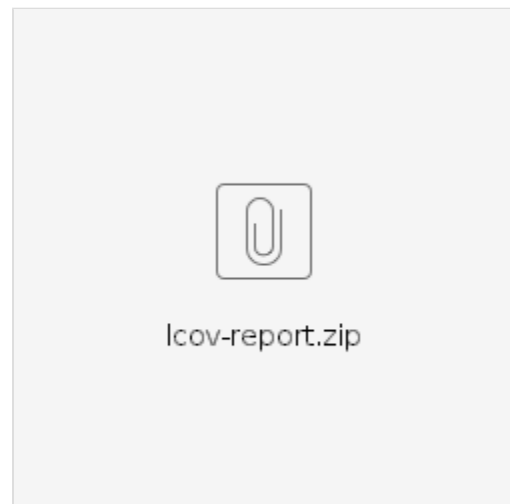
Press n or / to go to the next uncovered block, b, p or k for the previous block.

File	Statements	Branches	Functions	Lines
ConfirmModal.js	100%	5/5	100%	0/0
Invite.js	100%	23/23	78.57%	11/14
Invites.js	83.12%	64/77	75%	9/12
PhotoRow.js	50%	6/12	10%	1/10
Post.js	75%	12/16	93.75%	15/16
Posts.js	77.4%	113/146	58.67%	44/75
Search.js	87.5%	7/8	100%	0/0

To gain further details such as which statements are not tested, which branches have been missed in the test report, here are steps:

1. Download the zip file.
2. Extract files.
3. Double click on the HTML file called "index" (It is NOT inside the src folder).
4. This file should be open in the browser, and all file names will be clickable to view the details of the test report.

Statements, functions and lines that are not covered will be colour-coded. In order to understand the meaning behind the colours, please refer to the link [Meanings of the colour.](#)





# Coding Standards & Version Control Policy

## Definition of Done (DoD)

The definition of done is an agreed upon the guidelines/rules below that must be met before an user story is consider as done.

- If applicable, the code must met the coding standards below.
- If applicable, the code must work intended as the requirement(s) set out in the user story.
- All tests must be passed and new commit should not break any existing features/systems.
- Artefacts or code must be peer-reviewed.
- If applicable, API documentation should be completed.
- UI designs should check with acceptance test.
- Once the above guidelines have met, the code must be merged into development branch.

## Estimation

Different estimation techniques were attempted. The **Fibonacci** sizes were used in the first two sprints. This was changed to a more generic T-Shirt/Small-Medium-Large approach in the third sprint as it was simpler and our tasks were reasonably granular that splitting into these categories made estimation easier to complete.

## Coding Standards

This document contains the guidelines for writing code for the team SWEN90014-2021-FB-Quokka. The point of style guide is to uniform the appearance of source code. The benefit of such guideline will improve readability, maintainability for the programming code and further enhanced the communicability between members.

### General

#### Commenting

- Ensure comments are clear and readable.
- Ensure comments are accurate and up-to-date.
- In general, insert comment on the line above the code it is describing.
- Avoid over-commenting or stating the obvious.

#### Code documentation

- Each system component must be documented with a README file containing relevant information such as:
- A description of what the component does
- Instructions for setting up and using the component

### Javascript

In general, refer to Google's [style guide](#). Selected points include:

- Indentation: Use **2 spaces**. Tab characters are not used for indentation.
- Variable declaration: Use the **const** keyword by default. If a variable needs to be reassigned, use the let keyword. The var keyword must **not** be used.
- Semicolons: Every statement must be terminated with a semicolon.

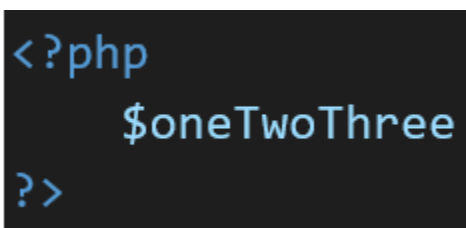
Our departures from Google's style guide:

- String literals: Delimit using **double quotes**, rather than single quotes.

### PHP

#### Naming convention

- Variable names: camel case



```
<?php
    $oneTwoThree
?>
```

- Constant: Capitalized with underscores

```
<?php
    $ONE_TWO_THREE
?>
```

#### Statements

- Start braces on the same line as if/for/while/else statement
- Use tabs to indent new lines

```
<?php
    if ($foo == true) {
        return $foo;
    } else {
        $foo = false;
    }
?>
```

- space after parameter

```
<?php
    myFunc($parameter1, $parameter2, $parameter2)
?>
```

#### Operators

- Space between operator

```
<?php
    $foo = $a + $b + $c
?>
```

#### Comments

- Comment above the code
- Start with lowercase letter
- Single line starts with `"/"`
- Multiple lines start with `"/**/"`

```

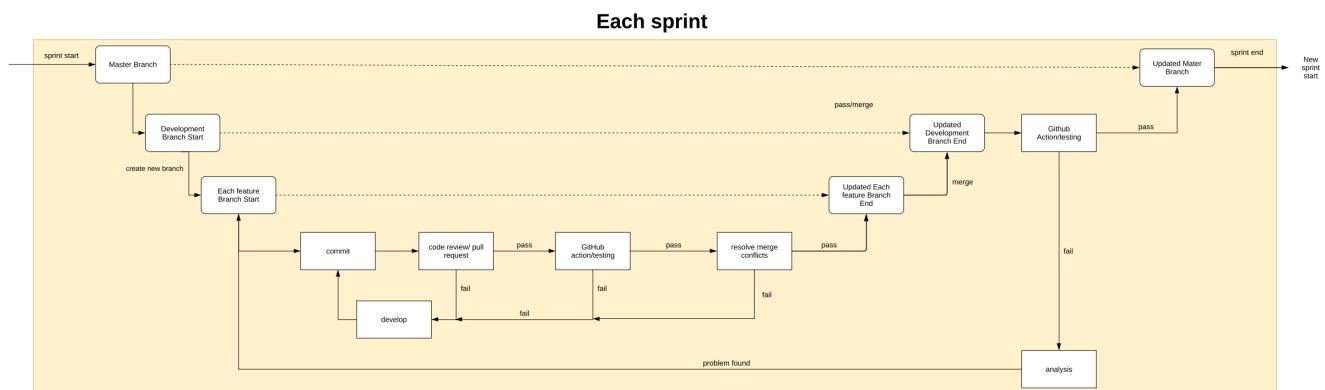
<?php
    //single line comment
    $foo = $a + $b + $c

    /*
        multiple lines comments
        please comments in this style
    */
    $foo = $a + $b + $c
?>

```

In general, refer to the official PSR-12 [style guide](#).

## Version Control Policy



## Commits and releases

- "Commit small, Commit often".
- Write clear and concise commit messages in the **imperative mood**. Where applicable, **refer to the relevant issue on JIRA**.
- Make a tagged release at the end of each Sprint.

## Branching

- Maintain a high quality, up-to-date main branch.
- Use feature branches for all new features and bug fixes.
- Merge feature branches into the main branch using pull requests.

## Code Review

In order for a pull request to be merged, it must

- Pass all relevant tests
- Be reviewed by (at least one) team member for sanity checking. (which team member(s) are involved in reviewing will be subject to availability)



## Resources

# Checklists



# General Process Documentation

1. Team roles, the individuals filling them, and the high-level responsibilities of the roles.
2. The team will hold a team meeting at least once per week. A document should be maintained for each such meeting, including agenda (describing the main discussion topics and built collaboratively before the meeting) and meeting minutes (describing the main decisions made and the action items assigned to specific team members or for entry in JIRA).
3. Up-to-date risk tracking table (maintained and updated as required by the team at team meetings)
4. Separate to their regular team meetings, the team will hold and document sprint planning, sprint review and sprint retrospective meetings for each sprint.
5. To ensure team members are aware of each other's progress, the team will hold stand-up meetings at least twice per week, either face-to-face or electronically. These meetings do not need to be documented.
6. *All product related communication from the Client should be captured in the Project Requirements Confluence space.*

# Design Concept Checklist

1. A high-level design concept diagram (block diagram or UML component diagram), describing the high-level components of the system, including external systems, and their relationships (interfaces/interactions). (This should eventually evolve into a representation of your architecture.)
2. A list of the main technologies supporting both the overall concept and components: this must include the implementation technologies, but may also include development environment and support technologies, such as for testing.
3. An associated reuse plan indicates for each component in the design concept, which can be provided through reuse, which are to be developed for purpose, and which will be provided through a mix of these; the details of libraries, components, or other elements being reused need to be included.
4. References to learning resources to bring the team up to speed on any of the items above, e.g. tutorial on a development environment, reference for a framework, documentation for a component library.
5. The equivalent of a design notebook: at least the reasons for the choices made in deciding on the above elements, but better still the process undertaken in deciding this. That is, while it is useful to know why you ended up choosing a component or technology, it can be more informative to also know what alternatives were considered, and how all the options were evaluated.

# Architecture Checklist

1. Documented architecture in Confluence in the 4+1 style. This documentation should evolve from and replace your design concept. Note that *not all diagrams types or models are required*; this will depend on your project and should be confirmed with your supervisor.
2. Your documentation should not attempt to capture details of frameworks or other fixed elements that you have adopted; it is sufficient to reference these elements. You should concentrate on the elements of the system which your team has developed and how the system fits together as a whole.
3. An architecture should always take basic design principles such as low coupling and high cohesion into account, to support maintainability and extensibility. In addition, where your project has critical attributes or other special requirements, it should be clearly documented how the architecture accounts for these.
4. It must be possible to navigate from the architecture documentation to the code, that is, naming and other information in the architecture should allow a reader to find the corresponding code elements in Bitbucket.
5. Your architecture should pass the "new team member" test. If a new team member joined your team, you should be able to use the architecture diagrams to explain the system and assign tasks to the new team member, who should then be able to understand the context of their assigned task, the interfaces they need to work to, the reuse elements already decided and the rationale for these, and be able to find the relevant elements in Bitbucket.

# Executable Architecture Checklist

*Context:* Your team should have a design concept and reuse plan documented in your team confluence space. You need to flesh out this design concept to define, construct and test your system architecture as completely as possible with the constraints of the sprint.

*Your aim is to ensure that:*

1. All of the high-level subsystems and components of your architecture are identified (high-level here requires judgement and justification). This includes:
  - a. External subsystems
  - b. Components you plan to re-use
  - c. Components you plan to develop
2. All the interfaces between these components should be defined: which operations are supported, what data is transferred, which component controls the interaction.
3. Your bitbucket repository should contain a tagged, runnable build which can exercise, in a manner similar to that required by the target system:
  - a. All re-use components
  - b. All external subsystems
  - c. Drivers/stubs, or limited capability versions, of all subsystems/components that you plan to develop
4. Your repository should also include a suite of system and integration tests which exercise all the interfaces in a manner similar to that required by the target system, as well as
  - a. expected test results, and
  - b. actual test results from running the tests (build version used in running test should be clearly identified).
5. Where your executable architecture does not support the full interface required of the target system, a (concise) justification should be provided in terms of risk, difficulty/complexity, and scope.

# JIRA Checklist

1. A product backlog (with higher-level user stories) will be maintained in Jira, along with a lower-level sprint backlog.
2. Tasks in the sprint backlog, will have a sufficiently low level of granularity.
3. Tasks in the sprint backlog, will have an estimation.
4. Team members will take ownership of tasks proactively (just in time) and will be responsible for moving them along the lanes of the board in a timely fashion.
5. Documented estimation process (in Confluence)
6. Documented workflow (in Confluence)

# Verification & Validation Checklist

1. Testing -- your team's approach to testing should be documented in Confluence including:
  - a. What are the rules for who will test artefacts the team produces? (For example, is the developer unit testing? Integration testing? System Testing?)
  - b. What general approach is required for testing? (e.g. What degree of coverage at the different levels? What degree of automation?)
  - c. How are test results to be documented? (Test results should be captured and reproducible.)
  - d. How are test failures to be handled? (Are they all to be logged in JIRA? Should the tester try to fix bugs first?)
2. Reviewing -- your team's approach to reviewing should be documented in confluence including:
  - a. Which artefacts need to be reviewed and under what circumstances?
  - b. What style of review is required for different artefacts/circumstances? (e.g. walkthrough, inspection, audit)
  - c. What are the details of the review to be conducted? (See, for example, code reviews in the GitHub checklist)
3. Other V&V activities: where activities other than testing and reviews have been used (e.g. prototyping in target technology), this should be documented in Confluence.

# Source Code & Version Control Checklist

1. Good quality code: coding standard identified (in Confluence) and followed
2. Test code and cases (being clearly identified)
3. Instructions or infrastructure to build and execute the system, including running the test cases
4. Sensible log messages in all commits, including references to Jira issues when relevant
5. Documented commit and branching policy (in Confluence)
6. Well defined code review policy (inc. nomination of reviewers, timeframes for review, checklist for review)
7. Sensible policy on tagging for system versions (to identify milestone versions in general and, for instance, to being able to repeat tests)

# Demo/Acceptance Session Checklist

- Every team must have a presentation
- Closed sessions (i.e. presenting team only)
- Plan for 25 minutes
- Attendees:
  - All team members
  - Team supervisor
  - (2nd Examiner)
  - Client or rep. attendance is strongly preferred
- Main Focus: Use cases (mostly non-technical aspects)
- You (students) run the session
- The session content should include:
  - Introduction
  - Final Scope of your project
    - Delivered vs Not Delivered
    - Committed to vs Not Committed to
  - Quality Assurance
    - Briefly describe why Client can be confident in your system that it is fit-for-purpose
- The session content should include:
  - Testing Report
    - Summary statistics: Categories and outcomes
    - Outstanding issues
  - Demonstration
    - Team representative performs a live demonstration (playing a video record is not acceptable)
    - Team **must record** a video the live demo during the session for a submission
- The session content should include:
  - Delivery
    - Revisit outstanding issues (including from demo)
    - Handover (if the client accepted) or agree on handover arrangements
  - Client provide feedback
  - All outstanding issues and client feedback should be recorded in Confluence
- Demo capability (what users can do on the system), not functionality (what can the system do)
- Use realistic examples (based on user stories)
- Talk through what is happening during the demo
- Ensure you can set/reset easily for demos
- Pre-create elements to avoid delays
- Be prepared
  - Avoid issues if at all possible
  - Pre-announce if they can't be avoided
- Don't argue with the Client: just record



# Architecture Review Session Checklist

- Team supervisor will run the session
- The session content includes:
  - Overview of architecture
    - incl. identifying incomplete elements
  - Outline how Client can deploy your system
    - What are you delivering (form, content)?
    - What action needs to be taken?
  - Questions regarding proposed changes
    - Illustrate response using architecture
    - Show relevant parts of code

# Submission Checklist

	Checklist Entry	Notes	Status	Relevant Link
<b>Requirements Artefacts</b>				
	The Confluence requirements space for a project holds all requirements for all teams doing that project.			<a href="#">Home</a>
	There are no requirements in the Confluence development space of a team.			
	There is a dedicated page in the requirements space for the requirement called "Requirements" that will contain sub-pages for all the requirements artefacts; initially this content will be the resolved requirements, made up of artefacts assembled from the SWEN90006 requirement and reflecting the baseline for the SWEN90014 project.			<a href="#">Requirements</a>
	The requirements should be up-to-date, that is, reflect the current understanding of the project requirements.			In the requirement space, latest version and older version of User Stories, Acceptance Criteria and Acceptance Tests can be found
	In the requirements page there will be a requirements artefact corresponding to each of: background, goal model, personas, user stories (including acceptance criteria), quality requirements, paper prototype <sup>[1]</sup> and digital prototype.			<a href="#">Background</a> <a href="#">Goal Model</a> <a href="#">Personas</a> <a href="#">User Stories</a> <a href="#">Acceptance Criteria</a> <a href="#">Architectural and Critical Requirements</a> <a href="#">Prototypes</a>
	The requirements page will contain a sub-page called "Requirements resolution" which will list, for each student involved in the resolution process, their name, their team, and (briefly) the part of the process for which they were responsible. This should include the reviewers (from the client and technical perspective) for each team.			<a href="#">Requirements Resolution</a>
	The digital prototype will be used in client meetings to illustrate the functionality that will be developed in the following sprint. Comments or feedback from the client must be incorporated into the digital prototype, as it will act as a sounding board and guide the development of the real system, supporting consistency in terms of interaction and look and feel.	Not Applicable. <ul style="list-style-type: none"> <li>The first meeting with the client was to clarify all the questions among all project teams and update our client on the latest finding with Facebook restrictions. Hence the digital prototype was not used among all three teams.</li> <li>In the second meeting with the client, we used the system-under-development to discuss with the client about future features and potential improvements. Digital prototype was not used either.</li> </ul>		
	Detected issues in the requirements will have notes against them until fixed. The fixing may involve resolving ambiguities with the client.			
	If a team negotiates a team-specific deviation from the specification with their client (e.g. based on their choice of technology), they should note the nature and details of the deviation in the page of each affected requirements artefact.	Not Applicable.		
	Architecture and other critical requirements should be explicitly identified. (The way these have been addressed in the architecture should also be clear and explicit.)			<a href="#">Architectural and Critical Requirements</a>
	Acceptance tests are part of the requirements documentation. (The acceptance session demo conducted at the end of the semester should be drawn from these acceptance tests and <b>must</b> follow a narrative.)			<a href="#">Team Quokka's Acceptance Tests</a>
<b>General Process Documentation</b>				
	Team roles, the individuals filling them, and the high-level responsibilities of the roles.			<a href="#">Genyus Network - Facebook Data Migration</a>
	The team will hold a team meeting at least once per week. A document should be maintained for each such meeting, including agenda (describing the main discussion topics and built collaboratively before the meeting) and meeting minutes (describing the main decisions made and the action items assigned to specific team members or for entry in JIRA).			<a href="#">Team Meetings</a>
	Up-to-date risk tracking table (maintained and updated as required by the team at team meetings)			<a href="#">Risk Tracking</a>
	Separate to their regular team meetings, the team will hold and document sprint planning, sprint review and sprint retrospective meetings for each sprint.			<a href="#">Sprints</a>

	To ensure team members are aware of each other's progress, the team will hold stand-up meetings at least twice per week, either face-to-face or electronically. These meetings do not need to be documented.			
	<i>All product related communication from the Client should be captured in the Project Requirements Confluence space.</i>	We still need the notes for the final		<a href="#">Client Communication</a> <a href="#">Client Meetings</a>

## Sprint 1

Sprint 1 is already complete.

	Sprint 1 will include any required updates to requirements artifacts.			
	Sprint 1 will include further refinement of your design concept.			
	Sprint 1 will include an executable architecture.			<a href="#">Executable Architecture</a>
	Process documentation (in Confluence), including: <ol style="list-style-type: none"> <li>1. Records of meeting: significant decisions and actions</li> <li>2. All product related communication from the client</li> </ol>			
	A product backlog (with higher-level user stories) will be maintained in Jira, along with a lower-level sprint backlog.			

## Sprint 2

Sprint 2 is already complete.

	Sprint 2 will include any required updates to requirements artifacts.			
	Sprint 2 will include further refinement of your design concept.			
	Sprint 2 should be built on a complete, executable version of your architecture(+).			
	Sprint 2 will deliver identified capability to the client; your Sprint review should be able to demonstrate to the Client a combination of this capability together with static elements (e.g. forms, dialogues) developed in the target technology but not yet integrated.			<a href="#">Client End-Of-Sprint-2 Demo Meeting Notes</a>
	Process documentation (in Confluence), including: <ol style="list-style-type: none"> <li>1. Records of meeting: significant decisions and actions</li> <li>2. All product related communication from the client</li> <li>3. Other process documentation as specified in the referenced checklists</li> </ol>			
	Sprint 2 should make full use of JIRA capability for tracking project work.	We forgot to start sprint 2 on time in JIRA (1 day late)		
	Sprint 2 should include repeatable testing, with details of all testing which has been conducted being captured.			<a href="#">Testing</a>

## Sprint 3

	Sprint 3 will include any required updates to requirements artifacts.			
	Sprint 3 will include further refinement of your design concept.			
	Sprint 3 should be built on a complete, executable version of your architecture.			
	Sprint 3 will deliver identified capability to the client; your Acceptance Session should be able to demonstrate this capability and its fitness-for-purpose to the Client.			
	Process documentation as defined in the General Process Documentation checklist and other referenced checklists.			
	Sprint 3 should make full use of JIRA capability for tracking project work. Please refer to the JIRA checklist for details.			
	Documented V&V as per the Verification and Validation checklist.			
	Any materials produced specifically in support of your architecture session.			<a href="#">Architecture#SlidesforArchitectureReviewSessions</a>
	Any materials produced specifically in support of your acceptance session.			
	A video of the planned demonstration component of your acceptance session. <ul style="list-style-type: none"> <li>• During your acceptance session, you are expected to provide a <i>live demo</i> of your final system.</li> <li>• Your team is expected to <i>video this demo</i> from the audience perspective; a screen capture is <i>not</i> acceptance.</li> <li>• The video should be uploaded (<i>not a link, for example to an external streaming service or dropbox, but a playable video file</i>) to your confluence space.</li> </ul>			<a href="#">Acceptance Session</a>
	Appropriate documentation is provided to suit your specific project: this might include deployment documentation, developer documentation, user documentation, or other.			<a href="#">Installation on Local Machine</a> <a href="#">Migration App Installation</a> <a href="#">WordPress Backend API</a>
	All relevant project artefacts required to continue development and/or deploy the system must be delivered to the Client, and a record of when and how this was done must be provided in Confluence.			<a href="#">Handover Details</a>

			<p>Artefacts to be delivered to client:</p> <p>Relevant sections of this confluence (testing, system design, handover)</p> <p>Source code (in repository)</p>
--	--	--	---

## Design Concept

	A high-level design concept diagram (block diagram or UML component diagram), describing the high-level components of the system, including external systems, and their relationships (interfaces /interactions). (This should eventually evolve into a representation of your architecture.)		<a href="#">System Design</a>
	A list of the main technologies supporting both the overall concept and components: this must include the implementation technologies, but may also include development environment and support technologies, such as for testing.		
	An associated reuse plan indicates for each component in the design concept, which can be provided through reuse, which are to be developed for purpose, and which will be provided through a mix of these; the details of libraries, components, or other elements being reused need to be included.		<a href="#">Reuse Plan</a>
	References to learning resources to bring the team up to speed on any of the items above, e.g. tutorial on a development environment, reference for a framework, documentation for a component library.		There are some links in many of the pages under <a href="#">System Design</a>
	The equivalent of a design notebook: at least the reasons for the choices made in deciding on the above elements, but better still the process undertaken in deciding this. That is, while it is useful to know why you ended up choosing a component or technology, it can be more informative to also know what alternatives were considered, and how all the options were evaluated.		<a href="#">Design Concept /Notebook</a>

## Architecture

	Documented architecture in Confluence in the 4+1 style. This documentation should evolve from and replace your design concept. Note that <i>not all diagrams types or models are required</i> ; this will depend on your project and should be confirmed with your supervisor.		<a href="#">Architecture</a>
	Your documentation should not attempt to capture details of frameworks or other fixed elements that you have adopted; it is sufficient to reference these elements. You should concentrate on the elements of the system which your team has developed and how the system fits together as a whole.		
	An architecture should always take basic design principles such as low coupling and high cohesion into account, to support maintainability and extensibility. In addition, where your project has critical attributes or other special requirements, it should be clearly documented how the architecture accounts for these.		
	It must be possible to navigate from the architecture documentation to the code, that is, naming and other information in the architecture should allow a reader to find the corresponding code elements in Bitbucket.		
	Your architecture should pass the “new team member” test. If a new team member joined your team, you should be able to use the architecture diagrams to explain the system and assign tasks to the new team member, who should then be able to understand the context of their assigned task, the interfaces they need to work to, the reuse elements already decided and the rationale for these, and be able to find the relevant elements in Bitbucket.		

## Executable Architecture

	All of the high-level subsystems and components of your architecture are identified (high-level here requires judgement and justification). This includes: <ol style="list-style-type: none"> <li>1. External subsystems</li> <li>2. Components you plan to re-use</li> <li>3. Components you plan to develop</li> </ol>	All		
		Complete		
	All the interfaces between these components should be defined: which operations are supported, what data is transferred, which component controls the interaction.	As		
	Your bitbucket repository should contain a tagged, runnable build which can exercise, in a manner similar to that required by the target system: <ol style="list-style-type: none"> <li>1. All re-use components</li> <li>2. All external subsystems</li> <li>3. Drivers/stubs, or limited capability versions, of all subsystems/components that you plan to develop</li> </ol>	Part		
		Of		
	Your repository should also include a suite of system and integration tests which exercise all the interfaces in a manner similar to that required by the target system, as well as <ol style="list-style-type: none"> <li>1. expected test results, and</li> <li>2. actual test results from running the tests (build version used in running test should be clearly identified).</li> </ol>	Sprint		
		1		
	Where your executable architecture does not support the full interface required of the target system, a (concise) justification should be provided in terms of risk, difficulty/complexity, and scope.			

## JIRA

	A product backlog (with higher-level user stories) will be maintained in Jira, along with a lower-level sprint backlog.			
	Tasks in the sprint backlog, will have a sufficiently low level of granularity.			
	Tasks in the sprint backlog, will have an estimation.			
	Team members will take ownership of tasks proactively (just in time) and will be responsible for moving them along the lanes of the board in a timely fashion.			
	Documented estimation process (in Confluence)			Coding Standards & Version Control Policy
	Documented workflow (in Confluence)			

## Verification and Validation

	<b>Testing -- your teams approach to testing should be documented in confluence including:</b>			Testing
	What are the rules for who will test artefacts the team produces? (For example, is the developer unit testing? Integration testing? System Testing?)			
	What general approach is required for testing?(E.g. What degree of coverage at the different levels? What degree of automation?)			
	How are test results to be documented? (Test results should be captured and reproduceable.)			
	How are test failures to be handled? (Are they all to be logged in JIRA? Should the tester try to fix bugs first?)			
	<b>Reviewing -- your teams approach to reviewing should be documented in confluence including:</b>			Coding Standards & Version Control Policy
	Which artefacts need to be reviewed and under what circumstances?			Testing
	What style of review is required for different artefacts/circumstances? (E.g. walkthrough, inspection, audit)			
	What are the details of the review to be conducted? (See, for example, code reviews in the Bitbucket checklist)			
	<b>Other V&amp;V activities: where activities other than testing and reviews have been used (e.g. prototyping in target technology), this should be documented in Confluence.</b>	No additional activities have been used in this project		

## Source Code and Version Control

	Good quality code: coding standard identified (in Confluence) and followed			Coding Standards & Version Control Policy
	Test code and cases (being clearly identified)			
	Instructions or infrastructure to build and execute the system, including running the test cases			
	Sensible log messages in all commits, including references to Jira issues when relevant			
	Documented commit and branching policy (in Confluence)			Coding Standards & Version Control Policy
	Well defined code review policy (inc. nomination of reviewers, timeframes for review, checklist for review)			^
	Sensible policy on tagging for system versions (to identify milestone versions in general and, for instance, to being able to repeat tests)			Sprint milestones tagged on the GitHub repository.

# Handover Details

This section will document handover instructions and setup required.

## Table of Contents

## Notes

As the development team does not have access to the live Genyus Network implementation of BuddyBoss, we have implemented our own and deployed a "test" version. There may be differences between the "test" version and the Genyus Network implementation, which may result in parts that may not be fully compatible. Here are some assumptions that have been made in our test version:

- Users with **admin** privilege are able to access the Requests page. Standard users are only able to access the Migrate page.
- Discussion groups or rooms make use of BuddyBoss' groups, which is reachable through a BB endpoint. The admin will make the decision to migrate the post to a specific group and knows the original posters' memberships.
- There is no integration between the BuddyBoss account and Facebook login.

# Acceptance Session

Below is a recording of team Quokka's Acceptance session with the clients Cal and Ian Rixon, supervisor Glenn Jayaputera, course coordinator Philip Dart.

Due to the size of the recording, it has been split into 4 clips.



trim 1.mp4



trim 2.mp4



trim 31.mp4



trim 32.mp4

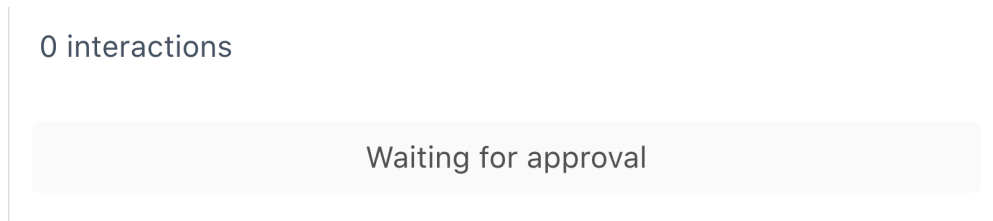


# Client Feedback/Outstanding Issues

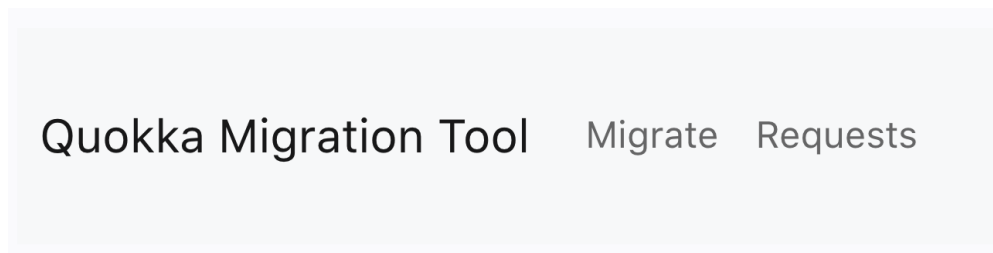
The following feedback and features were mentioned during the final Acceptance Demo. Due to the lack of time between the demo and end-of-course, they have not been implemented.

## Feedback from the Demo session:

- Waiting for approval needs to be more clear, add a red colour?



- The migrate/requests button needs to be more clear, add a bounding box?



- Add a dropdown list of rejection reasons instead of a text area.

## Functions not yet being implemented:

- The ability to migrate video content.
- The ability to migrate comments.
- A landing page that has a welcome/instruction video in it.

# Facebook App Installation

Before installing the Facebook Group App, the app needs to go through the Facebook App review. The Facebook app can only be installed after it is being approved.

To install the Facebook App:

- Go into the Genyus Network Facebook group setting.
- Click on the Manage Advanced Settings.
- Add the Facebook App.

# Getting Facebook Approval

## Limitations imposed by Facebook

Facebook imposes several limits to its API regardless of app type. These limits may affect the usage of the migration app. Some mitigation measures are possible, but are out of scope in the deliverable.

- The Groups API only shows posts made within the last 90 days. Editing or relinking to a post will not change this limitation.
  - Potential mitigation measure: caching or storing past Facebook posts by users separately.
- The Groups API only shows posts made **after** the app is installed into a group. This means that all existing posts may not be accessible by the application.
  - There is no way of mitigating this.
- Users of the app need to approve the permissions *before* their name/profile is identifiable by a post.
  - Avoided with the "user-initiated" workflow

There is also a limitation in the current handover limitation due to the lack of App Review (process to be documented in a separate section). For posts to be visible to the API, a user needs to be **admin** of the Facebook group in which the app is installed as *well as* **admin** of the application itself. This is because it is not possible to install an app that has not been approved to a group, and being admin of both the group and the app is the workaround.

## Getting Facebook Approval

The app is a Facebook Groups app, so it needs to be installed onto a group for it to take effect. In order to do so, the app needs to go through an App Review process to gain Facebook's approval. The development team is unable to do so ourselves due to the need for the app to be fully completed as well as the potential need for legal contracts between the app and Facebook.

The full Facebook app review process is detailed here: <https://developers.facebook.com/docs/app-review/>

Before gaining Facebook approval, a Facebook Developer account is needed. The following steps guide how this is done:

1. create a Facebook developers account on [developers.facebook.com](https://developers.facebook.com)
2. Create an App with the Create App button. Select *Business App*. Contact details are needed as well as a name for the app, as well as optional Business manager account.
3. Facebook will also require privacy policies and related legal documentation.
4. Verification with Facebook can be done in the App settings - this involves sending Facebook some identification documents to verify the developer account's identity
5. Terms of Services, Privacy Settings, and other similar links can be added in the settings app
6. Add the **Facebook Login** product. Under Allowed Domains for the JavaScript SDK, add the domain of the final website (which is likely <https://genyusnetwork.com>)
7. Once this is done, the App Review tab guides the rest of the process. The permissions to request are **groups\_access\_member\_info**.
8. As the development team are unable to complete App Review ourselves, we are unable to provide instructions on how to install an app in a group. However, Facebook documentation is available and should suffice.

# Migration App Installation

The migration app is built as a React app. The Facebook app ID and the Facebook group ID will need to be changed, before packaging the app up with ``npm run build``.

1. Setup the react\_embed plugin:
  - Change the Facebook app ID:
    - Go to GitHub and clone the GitHub repository.
    - Locate the 'react\_embed' plugin folder.
    - Open the 'src/APP.js' file.
    - Search for 'appld' keyword and change the Facebook app ID to Genyus Network Facebook app ID. This app ID is acquired from Facebook Developers portal, as described in [Getting Facebook Approval](#).
  - Change the Facebook Group ID:
    - Locate the 'react\_embed' plugin folder.
    - Open the 'src/POSTS.js' file.
    - Search for '<https://graph.facebook.com/v11.0/2658160797819865/feed>' and change '2658160797819865' in that URL to the Genyus Network Facebook Group ID, or any other group ID that you would want to test with.
  - Build the app with ``npm install`` and ``npm run build``.
2. Install and activate the react\_embed plugin onto the Genyus Network website (primary deliverable).
  - Go to the WordPress admin dashboard.
  - Hover your mouse onto 'Plugins' on the left menu, a drop-down menu will pop up, and then click onto the 'Upload Plugin' button.
  - Choose the 'react\_embed' plugin folder that you just located and upload it.
  - After it is uploaded, activate the plugin.
3. Install and activate additional plugins:
  - ACF to REST API
  - Advanced Custom Fields
  - JWT Auth
  - Custom Post Type UI
4. Import Custom Post Type:
  - Hover your mouse onto 'CPT UI' on the left menu, a drop-down menu will pop up, and then click onto the 'Tools'.
  - There is an 'Import Post Types' section, and copy and paste the content in the 'database/Custom Post Type/CPT UI.txt' onto the 'Import Post Types' section.
  - Click 'Import'.
5. Import Custom Fields:
  - Hover your mouse onto 'Custom Fields' on the left menu, a drop-down menu will pop up, and then click onto the 'Tools'.
  - There is an 'Import Field Groups' section, click onto the 'Choose file' and select the 'database/Custom Fields/acf-export-{date of import}.json' file.
  - Click 'Import'.
6. Create a new page
  - Click on 'Pages' on the left menu.
  - Click on the 'Add New' button.
  - Name this page 'Data Migration' (you can name it whatever you like).
  - Insert '[react-app-embed]' in the content field.
  - Click on 'Publish' to publish the page.
7. Add the 'Data Migration' page to the menu:
  - Hover your mouse onto 'Appearance' on the left menu, a drop-down menu will pop up, and then click onto the 'Menus'.
  - Select the 'Data Migration' page and add it to the menu.

There may need to be some additional configurations to ensure that users have the correct roles so that the authentication token works as expected.

# Source Code

The handover repository can be found here: <https://github.com/itemic/fb-quokka-delivery>. This is different from the course project repo in that it organizes the files in a different hierarchy and does not contain plugins that we did not design.

This is a private repo, so please get in touch with team Quokka if you need access to this as well.