c18n

A Community Discussion Platform

| S. No. | Contributor | Comments | Date | Reviewed By | Review Date |
| --- | --- | --- | --- | --- | --- |
| 1 | Jitender Singh | Introduction, Architecture dagram | Jul/25/2021 |  |  |
| 2 | Jitender Singh | User Registration, Authorisation, Discovery | Jul/25/2021 |  |  |
| 3 | Jitender Singh | Users APIs | Jul/25/2021 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Introduction 4

High Level design 5

Design Components 7

Users (Registration, Authentication, Authorisation and discovery) 7

Communities 8

Apis, Schema and Data Flow 9

Users APIS 9

Communities APIs 13

Design choices / improvements 13

Security concerns / guidelines 13

Challenges 14

Future Scope 14

This page is intentionally left blank.

# Introduction

The community discussion platform code named **c18n** (naming convention borrowed from other [numeronyms](https://en.wikipedia.org/wiki/Numeronym) like k8s, i18n, l10n etc.) is a community platform to support user discussions and direct messaging between users or user-groups.

The primary goal is provide a social network service where users across the globe can connect to each other, form communities based on their interest areas, create user-group with people they know directly or indirectly and engage in discussions, or send direct messages.

The **c18n** platform intends to build a strong user base by providing them with a rich feature set including (but not limited to) community creation, rich media posts containing text, images, audio, video and other interactive media, user discovery, likes, comments and shares, direct and group messaging, user discussion threads, custom profiles and building of user circles / friends networks.

As the user base is expected to grow to millions of users worldwide, the c18n platform aims to make use of a highly scalable, fault tolerant and low latency architecture by using various streaming and big data solutions.

For the proof of concept app for **c18n**, the focus area will be limited to a system that provides an end to end implementation of the community posts and direct user (or user-group) messaging. We also intend to provide a set of custom scripts to make it easy to generate the required data and a functional UI to visualise the user experience.

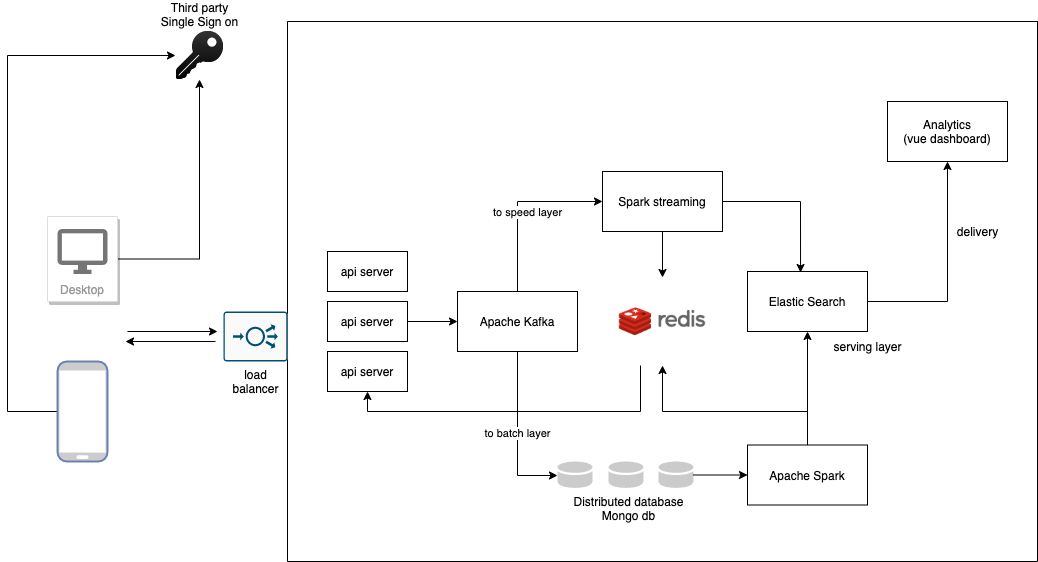
The PoC app is expected to have all the primary components running their simplest configuration such that the entire **c18n** PoC app can be set up and run on a single node (or a single node cluster.)

# High Level design

Fig. 1 - High Level Design

The figure above shows the proposed high level design diagram for the **c18n** app.

The design diagram primarily caters to the following use cases, and is subjected to changes with the introduction of new requirements / features.

* User registration
* User login
* User discovery
* Community creation
* Community discovery
* Joining a community
* Inviting users to join a community
* Accept / decline an invitation
* User-Group creation
* User-Group discovery
* Joining a user-group
* Inviting users to join a user-group
* Accept / Decline an invitation
* Community post containing
  + text
  + images
  + videos
* Share a community post
* Comment on a community post
* Like a community post
* Direct message to a user
* Reply to a user message
* Message (broadcast) to a user-group
* Comment / Reply on a user-group message
* Like a user-group message
* Reporting / Analytics
* Ad hoc queries

The application is supposed to have multiple scalable api server components, each of components catering to one particular type of resource and its associated use cases.

The api servers for Users, Communities, User-Groups, Messaging etc will be deployed via kubernetes deployments and each one of the deployments can be scaled up / down independently, depending upon the incoming traffic volume.

We intend to use HTTP based apis for use-cases related to user authentication, creation of user-groups, communities and other non-heavy workflows, while use-cases requiring heavy media flow, longer connections and instant user notifications (e.g. messages, posts, likes, shares) shall be implemented via web-sockets.

The user facing servers shall be further split into application backend servers and frontend servers to make it easy to develop the two independently and separate the presentation layer from the backend apis. The api (backend) server thus will only accept requests from the frontend servers.

We intend to use python/flask for implementing the backend api servers, while the frontend shall be using Vue.js

Message Queue - Apache Kafka

Apache Kafka shall be used to decouple the user facing servers from the core application components. In practice, some of the low volume apis may (for example community creation, user creation ) may bypass Kafka and directly save the data to a database. This shall simplify the workflows that require events to be processed in specific order (for example, a community must be created first before posting a message to it.)

Further, the components responsible for creating a resource may emit an event like - “a new user has been created”, or “a new community has been created”, enabling the Kafka consumers to pick up and take appropriate action on it (e.g. we may want to build a list of users that may be interested in joining this community, and send them a **suggested community** notification.)

The api (backend) servers shall act as Kafka producers and shall push incoming event streams such as new messages / posts, likes, share to the Kafka server.

Batch Layer - Mongo db

We shall make use of a distributed database for storing the event metadata (for example, text content / urls of posts and messages, likes, shares and other information related to creation and subscriptions of various communities, user-groups and their mappings to the users of the c18n platform.

We intend to use an append only database in accordance with the guidelines for implementing the lambda architecture, which shall accept data from Kafka to serve as the batch layer.

For the PoC, we plan to make use of a mongo db server owing to its ease of setup. For a real production case we may want to explore Hadoop hdfs/hbase or Cassandra.

For generating the batch views, we may use custom queries triggered via external scripts or through Elastic search.

Speed Layer - Kafka Streams / Apache Spark

We shall be using Apache Spark or Kafka stream for the speed layer of the lambda architecture.

This component shall be responsible for generating the speed view and save the view to the serving layer, (to be implemented via Elastic Search)

TBD - Yet to explore both the options and need to figure out which one is more suitable for the PoC.

Serving Layer - Elastic Search

We plan to use elastic search for combining the speed and batch views and serve the results to both the api servers and in turn the end users, and to the analytics dashboard.

Cache - Redis

In order to speed up the query and reduce the load on the database, we plan to use a redis cache server. Redis shall be used to hold the recent events data, for example currently logged in user's info, recently created communities, recent posts belonging to large user communities, or recents posts by users having large following.

RDBMS / Graph databases

A relational database may be used to hold user information, roles and access permissions. As we do not expect the total number of users to cross a few billions, a traditional database may work fine for this, however if there is a need to build a user - user relationship networks, followers / fans or a friends of friends network, we may want to explore a graph database for such cases.

For the PoC, any information related to dummy users shall be stored in the Mongo db itself.

# Design Components

The PoC app intends to use the following components **exposed via the api servers** -

### Users (Registration, Authentication, Authorisation and discovery)

User registration can be done via a hosted user authentication service, which provides a form based interface for new users to supply user information (email / password / phone / username etc) and validate them via email or a secondary channel (eg. Sms / voice call)

In addition to this, c18n platform will also provide the option to use a third party authentication, so that users can use their existing accounts (Google / Facebook / okta etc) to join the c18n platform.

Once signed up and validated via email / phone, the users can authenticate themselves to the c18n platform and start accessing its services. Both the hosted and third party auth services will make use of JWT token to validate the authenticity of the users. The token may have an expiry period set to 7 days by default, and may need to re-login once the JWT token is expired.

All the users will be given a default **regular user** role. In future, the c18n platform may also support additional roles (say, **premium users**, **api** **developers** etc).

In addition to the above, there will be one of more users assigned with **administrator** roles, having access to various subsystems (e.g. analytics dashboard) and control permissions (user or content deletion/deactivation etc). A separate authorisation table shall be used to decide the user roles and permissions.

For the **c18n** PoC app, we are going to make use of okta third party authentication for signing up new users. In addition to this, the PoC app will also allow batch creation of dummy users via a user creation api.

For the PoC app, users signing up via okta are going to be treated as admin users and will have the permissions to use the analytics dashboard, batch create dummy users or impersonate other users.

In addition to this, a user discover api will allow users to search other users by name and any other public info.

### Communities

The communities module will handle use cases related to creation, discovery and user mappings to a community.

Future scope of this module may also include features like assigning tags to communities and extend the community discovery / search by associated tags. Further, we may introduce the concept of transferring the community ownership to other users, providing one or more users with a moderator role and management of various community specific workflows (e.g. limiting the number of posts per user, block / kick out certain users, moderation of posts etc.)

# Apis, Schema and Data Flow

All the above listed component for the **c18n** platform are expected to use REST guidelines to implement their apis. We will use json as the primary format for data exchange between the different components as well as to/from end users.

The apis will use the format /api/<version>/<resource-pluralized-name>/[action if any]

Following the standard REST guidelines, we will be using POST requests for resource creation, PUT / PATCH requests for resource update, GET for retrieval, while DELETE shall be used for soft-deletion or deactivating a resource. (Related, see Security Concerns and Guidelines section.)

Below listed table present the various apis associated to the modules listed earlier, along with their sample request / response formats -

Users APIS

|  | Create a new user |
| --- | --- |
| Request | POST /api/v1/users/new  Accept: application/json, text/plain, \*/\*  Accept-Encoding: gzip, deflate, br  Connection: keep-alive  Host: api-server:80  User-Agent: browser UA  Content-Type: application/x-www-form-urlencoded  Content-Length: 76  username=<username>&email=<[username@domain.com](mailto:username@domain.com)>&password=<password> |
| Response | HTTP/1.0 200 OK  Content-Type: application/json  Content-Length: 397  server: API Web Server  Date: Sun, 25 Jul 2021 11:21:25 GMT  Location: /api/v1/users/<user-resource-id>  {  "\_id": "user-resource-id",  "email": “username@domain.com",  "exp": 1627797085.108836,  "iat": 1627192285.108832,  "name": "username",  "sub": "user-jwt-token",  "token": "user-jwt-token,  "ver": 1  } |
| Comments | In the PoC app we will be using the above api for creating dummy users in the system, the post request here would require okta validated JWT token to ensure the admin privileges. |
| Data Flow | Front End / User App -> Users api server |

|  | Retrieve a user |
| --- | --- |
| Request | GET /api/v1/users/  or  GET /api/v1/users/<user-resource-id>  Accept: application/json, text/plain, \*/\*  Accept-Encoding: gzip, deflate, br  Connection: keep-alive  Host: api-server:80  User-Agent: browser UA  Content-Length: 0 |
| Response | HTTP/1.0 200 OK  Content-Type: application/json  Content-Length: 397  server: API Web Server  Date: Sun, 25 Jul 2021 11:21:25 GMT  Location: /api/v1/users/<user-resource-id>  {  "\_id": "user-resource-id",  "email": “username@domain.com",  "exp": 1627797085.108836,  "iat": 1627192285.108832,  "name": "username",  "sub": "user-jwt-token",  "token": "user-jwt-token,  "ver": 1  } |
| Comments | The first api may be accessible to regular users to fetch their own profile information, the second format can be made available to admin users to obtain info on any of the platform’s users. |
| Data Flow | Front End / User App -> Users api server |

|  | Update a user |
| --- | --- |
| Request | PATCH /api/v1/users/<user-resource-id>  Accept: application/json, text/plain, \*/\*  Accept-Encoding: gzip, deflate, br  Connection: keep-alive  Host: api-server:80  User-Agent: browser UA  Content-Length: 123  {  "name": “new-user-name",  } |
| Response | HTTP/1.0 200 OK  Content-Type: application/json  Content-Length: 397  server: API Web Server  Date: Sun, 25 Jul 2021 11:21:25 GMT  Location: /api/v1/users/<user-resource-id>  {  "\_id": "user-resource-id",  "email": “username@domain.com",  "exp": 1627797085.108836,  "iat": 1627192285.108832,  "name": “new-user-name“,  "sub": "user-jwt-token",  "token": "user-jwt-token,  "ver": 1  } |
| Comments | Api may allow only specific fields (e.g. name, display-name, location etc) to be updated. |
| Data Flow | Front End / User App -> Users api server |

|  | Search user(s) |
| --- | --- |
| Request | GET /api/v1/users/search?name=foo&location=bar  Accept: application/json, text/plain, \*/\*  Accept-Encoding: gzip, deflate, br  Connection: keep-alive  Host: api-server:80  User-Agent: browser UA  Content-Length: 0 |
| Response | HTTP/1.0 200 OK  Content-Type: application/json  Content-Length: 397  server: API Web Server  Date: Sun, 25 Jul 2021 11:21:25 GMT  [{  "\_id": “user-resource-id",  "name": “user-name“,  "location": “http://api-server/api/v1/users/<user-resource-id>“,  },  {  "\_id": "user2-resource-id",  "name": “user2-name“,  "location": “<http://api-server/api/v1/users>/<user2-resource-id>“,  },  …  ] |
| Comments | All the GET apis that are expected to return more than 1 resources must be paginated.  Related - see Security Concerns and guidelines. |
| Data Flow | Front End / User App -> Users api server |

Communities APIs

|  | Create a new user |
| --- | --- |
| Request | POST /api/v1/users/new  Accept: application/json, text/plain, \*/\*  Accept-Encoding: gzip, deflate, br  Connection: keep-alive  Host: api-server:80  User-Agent: browser UA  Content-Type: application/x-www-form-urlencoded  Content-Length: 76  username=<username>&email=<[username@domain.com](mailto:username@domain.com)>&password=<password> |
| Response | HTTP/1.0 200 OK  Content-Type: application/json  Content-Length: 397  server: API Web Server  Date: Sun, 25 Jul 2021 11:21:25 GMT  Location: /api/v1/users/<user-resource-id>  {  "\_id": "user-resource-id",  "email": “username@domain.com",  "exp": 1627797085.108836,  "iat": 1627192285.108832,  "name": "username",  "sub": "user-jwt-token",  "token": "user-jwt-token,  "ver": 1  } |
| Comments | In the PoC app we will be using the above api for creating dummy users in the system, the post request here would require okta validated JWT token to ensure the admin privileges. |
| Data Flow | Front End / User App -> Users api server |

# Design choices / improvements

# Security concerns / guidelines

Under no circumstances(\*) shall the data be deleted from the system and maximum efforts shall be made to preserve the history of every resource creation, update and access.

\* except where the data policies (e.g. GDPR) or law-enforcement require the data to be deleted upon expiry or upon user request.

All the GET apis that are expected to return more than 1 resources (e.g. search users, search communities etc) must have pagination support.

The apis may support a configurable default value (say 10 records per query), while a non-configurable / hardcoded max value must restrict the number of items per search queries to an allowed maximum (say max 100 records per query).

# Challenges

# Future Scope