RC Users and Events license

- 1. Business goal
  - 1.1 Product assumptions
- 2. Architectural outline
  - 2.1. Main points
  - 2.2. User entity
  - 2.2. Current RCE database state
    - 2.2.1 Existing Users tables
  - 2.3. Proposed solution
    - 2.3.1. Solution outline
- 3. Implementation
  - 3.1. Data structure changes
    - 3.1.1. New/Updated RCE data structures
      - 3.1.1.1. New table RCAccounts
      - 3.1.1.2. New table RCUsers
      - 3.1.1.3. Updated table user_sessions
  - 3.2. Provisioning flows
    - 3.2.1. Create an RCE organization linked to the RC account
    - 3.2.2. Create RCE user on the fly after authorization
    - 3.2.3. Update/create user on demand
    - 3.2.4. Update user data on an event from RC
  - 3.3 Data retention job
    - 3.3.1. Periodical user data update job
      - 3.3.1.1. Avatar issue
    - 3.3.2. User anonymization
    - 3.3.3. Account deletion handling
  - 3.4 APIs
  - 3.5. GDPR
  - 3.6. Licensing model
  - 3.7. Additional points
    - 3.7.1. Data immutability
    - 3.7.2. Authorized sessions invalidation
  - 3.8. Operational dependencies
    - 3.8.1. Network
    - 3.8.1. Accessibility
    - 3.8.2. Application Credentials
- 4. Future plans
- 5. Open questions
  - 5.1 User anonymization details
- 6. Appendices

1. Business goal

The main goal is to cover integration of RCE into the RC ecosystem to allow having SSO and unified user management.

1.1 Product assumptions

The working assumption is that no legacy organizers creation can occur when the RC SSO is enabled.

2. Architectural outline

2.1. Main points

1. SSO and provisioning flows
2. An RC Account purchases an Events license. Information about the license to include the initial limit for the amount of Admins allowed under this license.
    a. Data on the account linked to the organization (table `organisers` in RCE database) should be cached in the RCE local database.
    b. The base user who creates the organization should be a system extension user (immutable on RC side), so the organization is not affected when such a user is removed for any reason.
    c. An organization and an organization_admin user (user creating the organization) should be created before the link to RC is set up.
    d. Admins amount limit can later be adjusted on the RCE side without any limitation.
        i. Admins and users limit is enforced on RCE side only.
3. A new user creation happens in RC and then user existence is ensured in RCE, after that users are linked
4. Users are added to the organization by the admin with a specific role.
5. Billing at this stage stays on the RCE side for legacy users and licenses not purchased from RC
6. If a user's role in the organization is removed, they lose the ability to manage the Events in this organization, have no RCE buttons/links in the UI, and can still attend the events.
    a. The link to the RC user should be removed in this case. The cached RC data should be removed as well.
7. Data on those linked extensions is loaded on demand from the Platform and cached in the RCE local database

    a. This data can also be loaded on an event.
8. Extension data stored in the RCE local database should be validated at least once a day
    a. If the extension is removed, the data in the RCE local database should be removed or anonymized depending on the use case and data usage
    b. if the whole account is removed all the account-related data in the RCE local database should be purged.
9. Handling of Avatar

2.2. User entity

The idea is that when an RCE user is linked to the RC extension, they have organizer privileges in RCE for their corresponding organization/Account.

Granting a role also creates RCE user, link to RC user and role record.

Revokinig the role also removes the link and cached RC user data.

Users in RCE without organizer privileges don't have to be linked to RC. They can attend any event based on RCE business rules.

2.2. Current RCE database state

2.2.1 Existing Users tables

Currently, the RCE database has the following user tables: (source)

Table	Description
`users`	Base users table, stores: id, personal data, authentication data, some payment, and even settings attributes
`admin_users`	Admin users, stores: id, authentication data
`organisers`	Organization table, stores organization related data
`organization_members`	Organization members, link table connecting users to an organization and stating their role, one of: • **organization_admin** • regular_member • event_admin

2.3. Proposed solution

2.3.1. Solution outline

1. Add a new table with RC Account data, with reference to the existing `organisers` table by organisers_id
2. Add a new table with RC Extensions data, with reference to the existing `users` table by user_id
3. Add a flow to create organization, taking in a requesting user and producing a base user, organization_admin user and the organization records, and connecting them to existing RC account.
4. Add a flow to create a user, add to the organization, load and store RC extension data, then link it to the Events user
    a. Add event handling process to do the same
5. Add a flow to create a user in RC, then create corresponding user in RCE and then link them.
6. Add a job to check existing RC users' data integrity
    a. flows to update, anonymize or delete records in the RCE database

3. Implementation

3.1. Data structure changes

3.1.1. New/Updated RCE data structures

3.1.1.1. New table RCAccounts

The table holds information on RC accounts with reference to a specific RCE organization.

The cache period should be configurable. The default is 1 day.

Table columns :

Attribute	Type	Description
rc_account_id	String	RC account ID (initially PBX account ID) - *Unique constraint*
rc_id_domain	String	Defines the ID domain ("PBX" is for PBX)
brand_id	Number	RC brand ID for the account.
contracted_country_code	String	ISO alpha2 code of contracted country for the account.
organization_id	big int	RCE `organisers.id`. Reference to specific organization.
cache_ttl	DateTime	Defines when the cached data should be updated. Default is NOW + cache period

3.1.1.2. New table RCUsers

The table holds information on RC users with reference to a specific RCE user.

The cache period should be configurable. The default is 1 day.

Table columns :

Attribute	Type	Description
rc_account_id	String	RC account ID (initially PBX account ID) - *Unique constraint*
rc_id_domain	String	Defines the ID domain ("PBX" is for PBX)
rc_extension_id	String	RC extension ID
user_id	big int	Events `users.id`. References a specific user.
cache_ttl	DateTime	Defines when the cached data should be updated. Default is NOW + cache period

3.1.1.3. Updated table user_sessions

The table holds information on RCE user's sessions.

Table columns :

Attribute	Type	Description
user_id	Big int	RCE `users.id`. References a specific user.
session_id	String	RCW session ID
created_at	Timestamp	Timestamp of the record creation
updated_at	Timestamp	Timestamp of the record last update
rc_session_id	String	Authorized RC Session ID (new field)

rc_session_id is a new column to create a link between RCE and RC authentication session.

3.2. Provisioning flows

3.2.1. Create an RCE organization linked to the RC account

This flow creates a new RCE organization linked to the RC account and makes RC system extension user (i.e. primary Super Admin user) the owner of this organization.

This flow should be invoked manually or upon a certain event received from the RC core system. "RC Account ID" has to be known.

The flow works as follows:

1. Ensure that there is no current RCE organization linked to the given RC Account ID (if exists, fail the flow)
2. Find Account System user data: System User's ExtensionId is same as AccountId, call RC API Get Extension Info with the access token and provided AccountID and ExtensionId = AccountId
3. Ensure that there is no current RCE user linked to the given RC account system user found in step (2) (if exists, fail the flow)
4. Ensure that there is no current RCE user having the same email address as the RC account system user found in step (2)
    a. In case user exist link RC and RCE user entity
5. Create a record in the `users` table for RC account system user, create a corresponding record in `RCusers` table
6. Create a record in the `organisers` table referencing previously created user, create a corresponding record in `RCaccounts` table
7. Add RC linked RCE user as *organization_admin* to the organization.

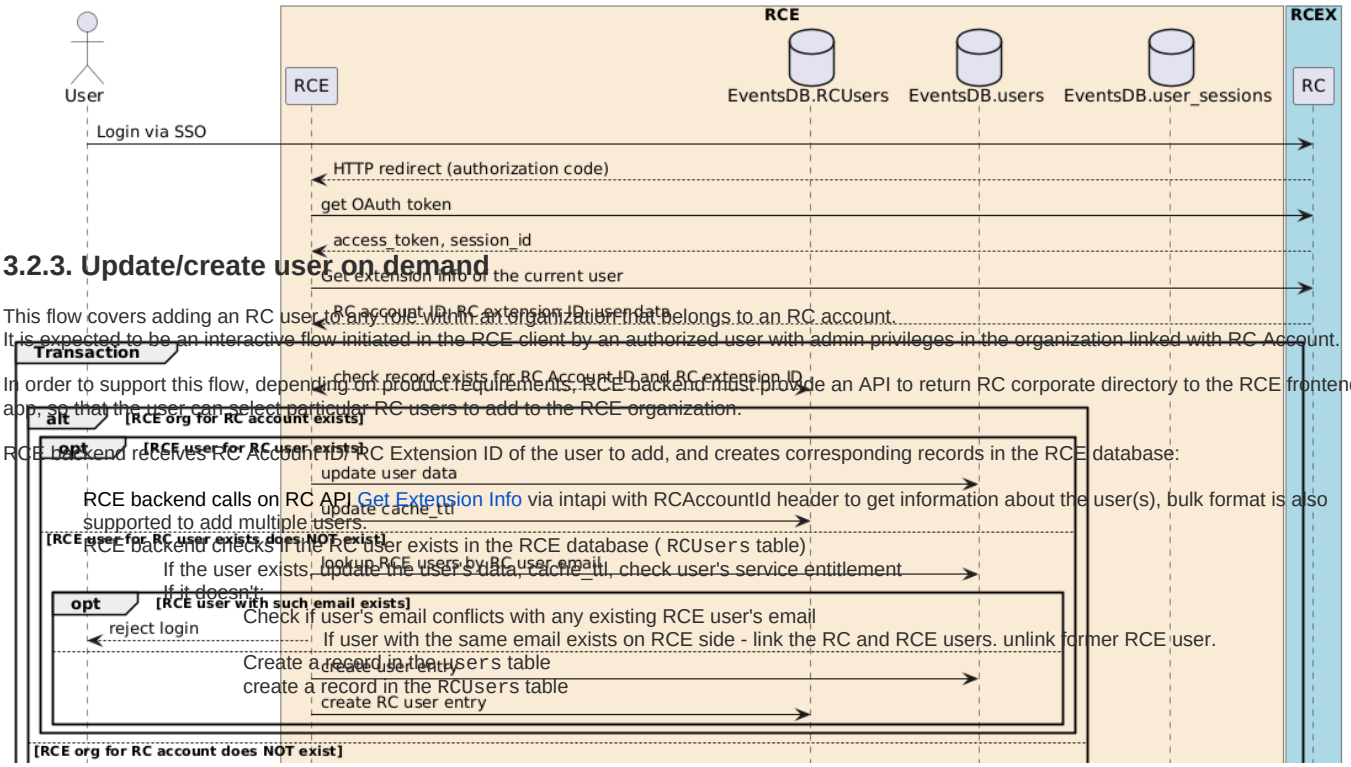
8. Make this organization owner user immutable (i.e. disallow deletion or change of privileges)

3.2.2. Create RCE user on the fly after authorization

This flow allows creating RCE user right after successful authorization in the RCE client application.

RCE client is supposed to initiate OAuth 2.0 Authorization Code flow (see https://developers.ringcentral.com/guide/authentication/auth-code-flow).

- After successfully completing interactive authentication, the RCE backend receives an authorization code through HTTP redirect to the redirect URI registered for RCE frontend app.
- RCE backend exchanges authorization code to access token
  - Extract `session_id` parameter from the response – it will hold the ID of the RC authorization session
- RCE backend calls RC API Get Extension Info with the access token
  - `id` represents RC user (extension) ID
  - `account.id` represents RC account ID
  - `contact` section contains first/last name and email
  - `type` contains extension type (only xxxUser extension types should be supported)
- RCE backend checks if the RCE organization corresponding to RC account ID exists
  - If the org does not exist:
    - Check RC entitlement of the account/extension for the RCE
    - if there is no entitlement for RCE: (user with free license case)
      - Check if the user's email conflicts with any existing RCE user's email – reject login in this case (will be handled via customer support)
      - Ensure RCE user creation
      - Create a record in the `RCUsers` table with RC User's data
      - Update user's contact data
      - Update `cache_ttl`
      - Create for the user RCE organization with free license
  - If the org exists do the following (as a transaction)
    - If the RCE user corresponding to RC user exists
      - Check user's entitlements
      - Update user's contact data
      - Update `cache_ttl`
    - If the RCE user does NOT exist
      - Check if the user's email conflicts with any existing RCE user's email – reject login in this case (will be handled via customer support)
      - Create a record in the `RCUsers` table with RC User's data
    - Store RC session ID along with RCE session information
    - Create an RCE authorization token and return it to the client



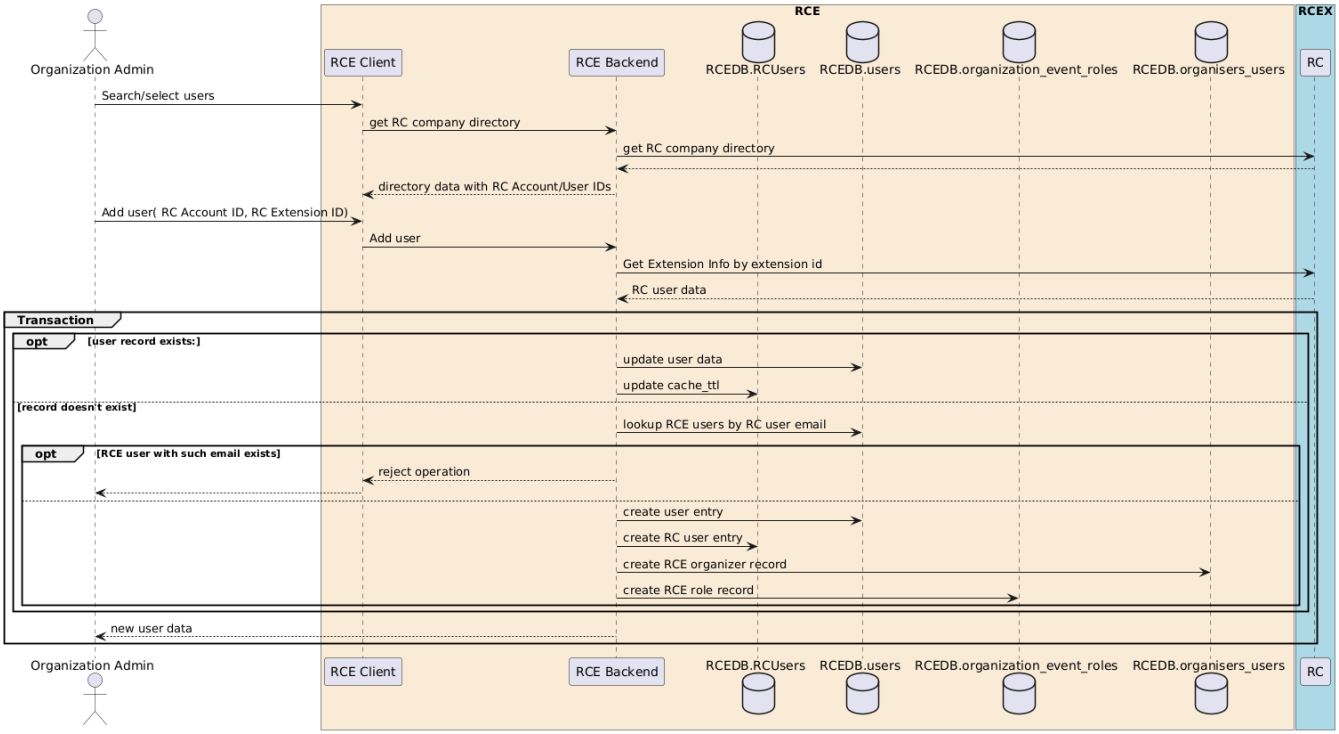
3.2.3. Update/create user on demand

This flow covers adding an RC user to any role within an organization that belongs to an RC account.
It is expected to be an interactive flow initiated in the RCE client by an authorized user with admin privileges in the organization linked with RC Account.

In order to support this flow, depending on product requirements, RCE backend must provide an API to return RC corporate directory to the RCE frontend app, so that the user can select particular RC users to add to the RCE organization.

RCE backend receives RC Account ID, RC Extension ID of the user to add, and creates corresponding records in the RCE database:

RCE backend calls on RC API Get Extension Info via intapi with RCAccountId header to get information about the user(s), bulk format is also supported to add multiple users.
RCE backend checks if the RC user exists in the RCE database (`RCUsers` table)
If the user exists, update the user's data, cache_ttl, check user's service entitlement
Check if user's email conflicts with any existing RCE user's email
If user with the same email exists on RCE side - link the RC and RCE users. unlink former RCE user.
Create a record in the `users` table
create a record in the `RCUsers` table



3.2.4. Update user data on an event from RC

Create a flow of events from RC to RCE streaming updates of accounts/extensions from RC to RCE database.

The idea is that the RCE backend subscribes to the dedicated Kafka topic, which delivers messages about account/user data changes from the RC core system. This works as a firehose: the RCE backend needs to process only messages about RC accounts/users whose data is stored in the RCE database.

On event a message is sent of the following format:

```
{
  "uuid":{{UUID}},
  "event":"/restapi/v1.0/account/~/extension/{{EXTENSIONID}}",
  "timestamp": {{TIMESTAMP}},
  "subscriptionId": "{{SUBSCRIPTIONID}}",
  "ownerId": {{OWNERID}},
  "body":{
      "extensionId": {{EXTENSIONID}},
      "eventType":"Update",
      "hints": [ "ExtensionInfo", "ExtensionInfoShort"]
  }
}
```

Information on account is passed in the event headers as RCAccountId

3.3 Data retention job

3.3.1. Periodical user data update job

Periodical (e.g. daily) scheduled job should run on the `RCUsers` table, and check entitlement, and update all records where cache_ttl is older than now.

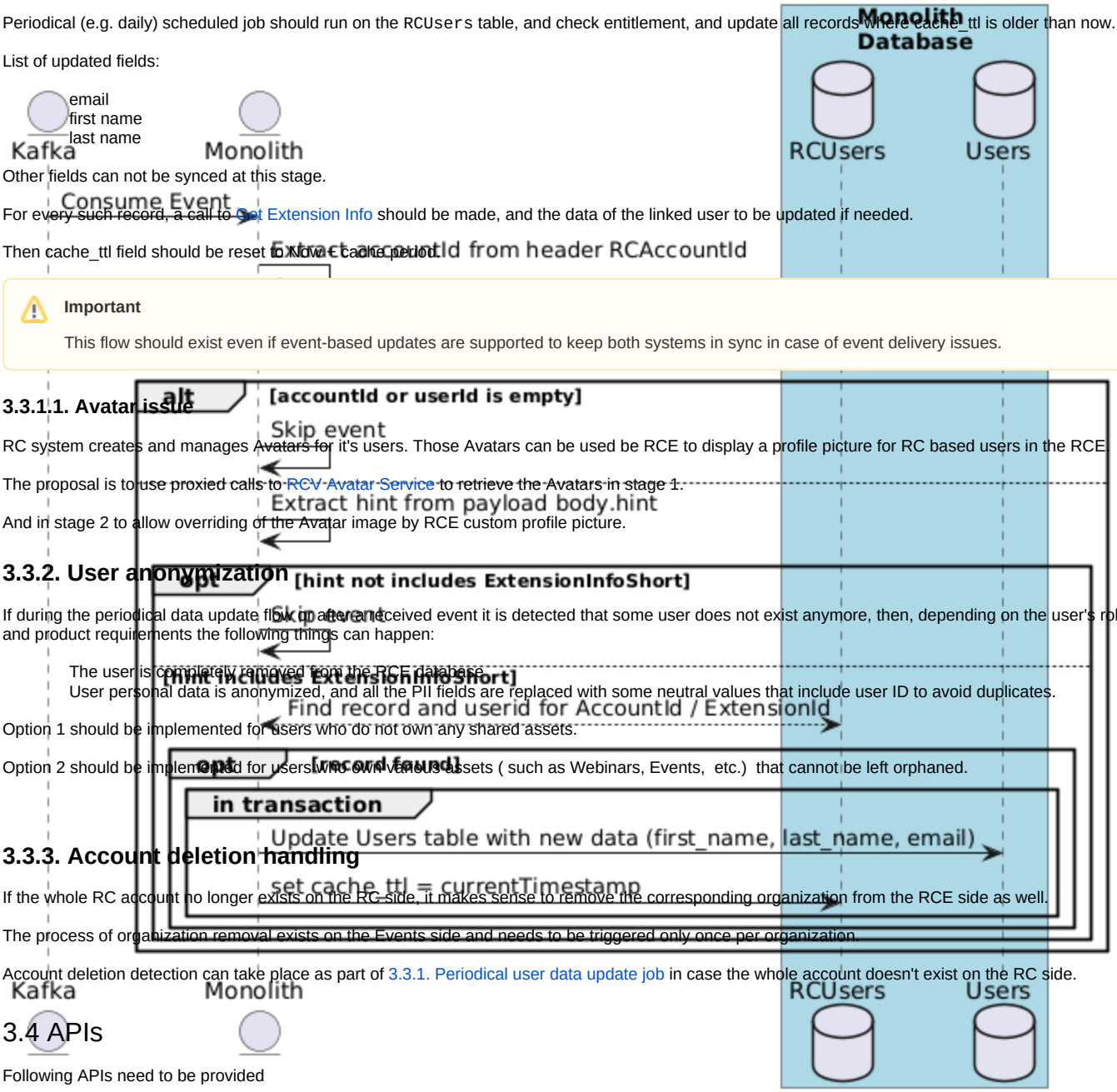
List of updated fields:

- email
- first name
- last name

Other fields can not be synced at this stage.

For every such record, a call to Get Extension Info should be made, and the data of the linked user to be updated if needed.

Then cache_ttl field should be reset to Now + cache period.



> **Important**
>
> This flow should exist even if event-based updates are supported to keep both systems in sync in case of event delivery issues.

3.3.1.1. Avatar issue

RC system creates and manages Avatars for it's users. Those Avatars can be used be RCE to display a profile picture for RC based users in the RCE.

The proposal is to use proxied calls to RCV Avatar Service to retrieve the Avatars in stage 1.

And in stage 2 to allow overriding of the Avatar image by RCE custom profile picture.

3.3.2. User anonymization

If during the periodical data update flow or after a received event it is detected that some user does not exist anymore, then, depending on the user's role and product requirements the following things can happen:

1. The user is completely removed from the RCE database.
2. User personal data is anonymized, and all the PII fields are replaced with some neutral values that include user ID to avoid duplicates.

Option 1 should be implemented for users who do not own any shared assets.

Option 2 should be implemented for users who own various assets (such as Webinars, Events, etc.) that cannot be left orphaned.

3.3.3. Account deletion handling

If the whole RC account no longer exists on the RC side, it makes sense to remove the corresponding organization from the RCE side as well.

The process of organization removal exists on the Events side and needs to be triggered only once per organization.

Account deletion detection can take place as part of 3.3.1. Periodical user data update job in case the whole account doesn't exist on the RC side.

3.4 APIs

Following APIs need to be provided

Method	Endpoint	Parameters	Result	Description

POST	/rce/1.0 /create_linked_organization	• organization data
		• linked account id
			One of:
			• Success
			  ◦ Organization id
			• Error
			  ◦ Error number
			  ◦ Error description
				Creates organization on RCE side linked to an existing RC account

POST	/rce/1.0 /create_linked_user	• authorization code	One of:
			• Success
			  ◦ user data
			• Error
			  ◦ Error number
			  ◦ Error description
				As result of authorization on RC side, creates a new linked user on RCE side and add them into existing RCE organization.

3.5. GDPR

GDPR requests will be handled manually at this point.

But there should be endpoints implementing anonymizaion on a specific user record by user id, that could be used in the manual anonymization process and in the process of user anonymization due to user removal handling.

3.6. Licensing model

Organization purchases a license from RC side. This information is forwarded to the RCE with initial limit amount for allowed Admins.

From this limit for amount of Admins is managed on RCE side.

A users get assigned a role by Admin. User that with a role can access and act according to their role in the RCE.

Users without a role can not manage events on RCE regardless of their previous access and data they created.

Organizations with license assigned from RC are not billed on RCE side.

3.7. Additional points

3.7.1. Data immutability

All the synced data of RC accounts and users should not be editable on the Events side as RC becomes the source of truth for those records.

3.7.2. Authorized sessions invalidation

Since the validity of an RC user's authorization session is only checked during login flow, and all subsequent requests from the RCE frontend to the backend are made with the RCE JWT, it is important to develop a mechanism to ensure that RC user's session is still alive while this user continues to work with RCE.

Here is the proposal:

- In phase 1 RCE just stores RC OAuth session ID after user's login as described in 3.2.2
- In the next phases RC will provide a dedicated Kafka topic to listen for session invalidation events. It will work in a firehose mode: the RCE backend will need to process only events related to OAuth sessions that are currently active on the RCE side (the rate of the events will be up to 3K per minute at peak time).
  Upon receiving such an event, the RCE backend will need to invalidate the RCE user's session associated with the RC session.

3.8. Operational dependencies

3.8.1. Network

Both systems RC and Events should have network interconnectivity.

3.8.1. Accessibility

The following services of respective parties should be mutually accessible

- API gateway
- Kafka

3.8.2. Application Credentials

As a minimum, the following application credentials must be created on RC side:

- client_id/client_secret for a frontend RCE client to be able to go through RingCentral login flow, receive OAuth tokens, and call RingCentral external APIs.
  There also should be a redirect URI registered for this application to receive the OAuth authorization code as described in 3.2.2
- client_id/client_secret for the RCE backend to access RingCentral internal APIs

4. Future plans

The next stage of development should account for options to either federate a few RCE organizations under one RC account or to have several RC accounts linked to one RCE organization.

Both options should be assessed by product and architecture teams.

5. Open questions

5.1 User anonymization details

The list of PII fields needs to be created and the process of replacing those fields should be described.

6. Appendices

Additional product assumptions can be found here:
https://docs.google.com/document/d/1VDcUgVseDODJ5CRL1mJtQvTFFI7iMtPNsCUFBjMVmrM/edit

Agreements on open questions:

RCE gaps and open questions