# Translation App

Technical Design Document

July 1, 2022

## Project Summary

Create a new Translation App to handle all internationalization functionality, including customer-specific overrides.

## Software Design Considerations

In version 1.17.0, we implemented internationalization in Ecosystem. At the time, Ecosystem seemed the best place to implement it, but after further consideration, we need to move it to its own service. We need to create a new Translation App and move the implementation from Ecosystem into this new app. Another obstacle we faced with the initial implementation was updating core translations while also preserving any customer-specific overrides. To work around this issue, we will maintain the core translations as JSON files in the repository and only use the database for customer-specific overrides.

## User Interface Design

For this initial phase, we will not include a user interface. In a future phase, we will add a user interface for managing customer-specific overrides.

## Service Code Specification

### API

The Translation App should contain a translations API for fetching translations and will implement the following endpoints.

- GET /translations/:appId/:locale
    - Returns all translations for the provided appId and locale. Any customer-specific overrides should be merged into the response.
- GET /translations/:appId/:locale/:translationKey
    - Returns the specific translation for the provided appId, locale, and translationKey. If there is a matching customer-specific override, the override should be returned.

There should also be an overrides API for managing customer-specific overrides and will implement the following endpoints.

- GET /overrides/:appId/:locale
    - Returns all overrides for the provided appId and locale.
- PUT /overrides/:appId/:locale/:translationKey
    - Adds a new override for the provided appId, locale, and translationKey. If one already exists, the value should be updated.
- DELETE /overrides/:appId/:locale/:translationKey
    - Deletes the override matching the provided appId, locale, and translationKey.

### Model

There should be a translations model and an overrides model to implement the logic and database operations for each of the API routes above.

### JSON Files

The core translations should be stored in JSON files. At the root of the project, there should be a locales directory. In the locales directory, there should be directories for each supported locale. In each locale directory, there should be a JSON file corresponding to each app containing translations.

Example:

- \locales
  - \ar
    - \berth-request-app.json
    - \ecosystem.json
    - \global.json
  - \en
    - \berth-request-app.json
    - \ecosystem.json
    - \global.json

### Cache

A cache should be created to keep the core translations from the JSON files in memory. This cache should be used in the models, as opposed to reading the JSON files for every operation. Examples of this type of cache can be found in the lib directory of ecosystem.

### App Implementations

Throughout all AVERT C2 apps, services, and libraries, any implementation of internationalization should be updated to use this new service. This includes dependency libraries such as node-app-core and client-app-core.

## Database Changes

### New Database

Create a new database in RethinkDB named “translation”. In this new database, create a new table named “sys_overrides”. These should be done via init scripts, similar to ecosystem.

### Data Structure

Each row in the overrides table should contain an appId, id, and translations object. The translations object should contain an object for each locale with overrides. The locale object(s) should contain key-value pair where the key is the translation key using dot-notation and the value is the translated string. If an app doesn’t require any overrides, there should not be a row for the app. Similarly, if an app doesn’t require overrides for a specific locale, that locale shouldn’t be in the translations object.

Example:

```
{
   "appId": "map-app",
   "id": "{RethinkDB generated GUID}",
   "translations": {
      "en": {
         "cameraWall.cameraSlot.cameraControls.hideControls": "Hide Controls"
      }
   }
}
```

### Old Table

The existing “sys_locales” table in the ecosystem database should be deleted. This should be done via an init script in ecosystem.

## Testing Requirements

### Existing Functionality

A full, in-depth test of all existing translations is not necessary, but you should test to make sure translations are working in various apps, both client-side and server-side.

### API

Test each new API endpoint to ensure they are working as expected and producing the correct responses.

### Overrides

Test fetching translations with and without overrides to ensure the core translations are returned unless there is an override.

## Notes

To create the new application, use the cb-app-scaffold project and follow the directions in its readme file.