Internationalization

Technical Design Document

November 22, 2021

Project Summary

This enhancement will provide for any language and international formatting support across all AVERT C2 applications to include all static web site text, string templating (i.e. emails, alerts), and for data stored in Elasticsearch and RethinkDB. The initial implementation will provide the framework for support as well as specific support for English and Arabic languages. Some languages, including Arabic, require right-to-left support, so this will also be required for this enhancement.

Software Design Considerations

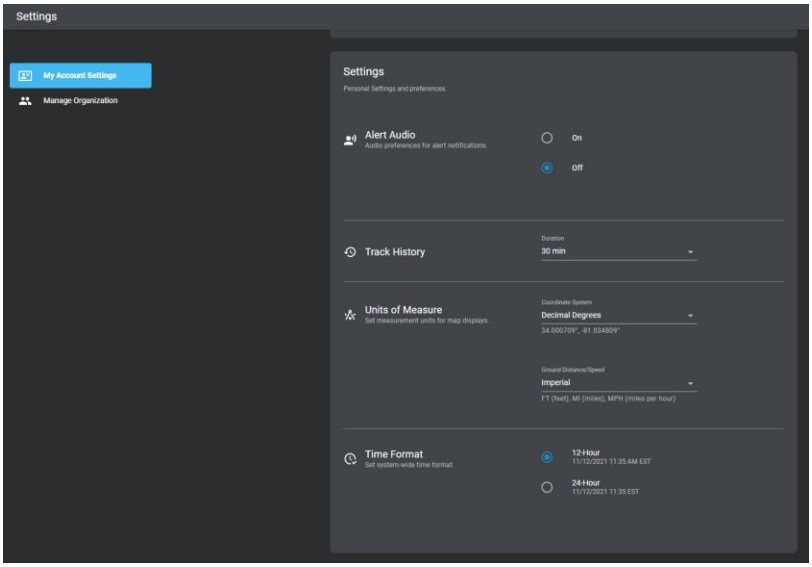
Considering our current client architecture makes heavy use of redux, we will want to use a redux solution. The react-i18n-redux library looks like a good fit, so documentation will reflect using that library.

Typically, translations are stored in files. AVERT C2 is built on an application framework that isolates applications and each will need its own set of translations. We are opting to store translations in the database and categorize by app ID and expose via rest API.

User Interface Design

User Locale Setting

There will need to be a new setting in the My Account Settings page in the Settings app to allow the user to choose their locale. Add a new section with the title “Language” and description “Set language preferences.” The locale setting should be a drop-down component and the values should be determined by the supported locales available in the database. The option to change locale in settings-app from here:



Right-To-Left

Apply right to left styling for Arabic and other languages that require it. A list of those languages can be found here: https://www.worldatlas.com/articles/which-languages-are-written-from-right-to-left.html

User Interface Code Specification

Reference Implementation

The implementation at the following address is the general architecture we are looking to implement and serves as a good reference.

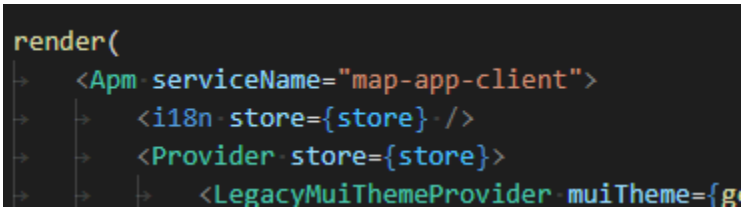
https://medium.com/i18n-and-l10n-resources-for-developers/react-redux-tutorial-internationalization-with-react-i18n-redux-dcc6724baa49

Loading Translations

Client code will leverage the translations and other formatting using the react-i18n-redux library. Base support for internationalization should be implemented in orion-components and imported into all projects with a user interface. This means it should be imported into all projects that utilize orion-components.

Translations will be loaded via the translations API (see Service Code Specification). You will need to add proxy methods to client-app-core for accessing the API from the client. These can be added to applicationService.js. Client-app-core is already installed with orion-components and you can import into i18n component to load translations at initialization.

Add a new component to orion-components that accepts the redux store as an argument and sets up i18n fetching the translations as described in previous paragraph by importing the applicationService from client-app-core and initializing i18n with those translations. The application ID should be derived from the route to the application. For example, in this URL *https://localhost/settings-app/my-account-settings,* "settings-app" is the ID for the application. This is a convention we follow with all applications. This component should be imported into the root index.js file in each client application (app-src directory) and nested in Apm component like so.



Export the react-i18n-redux components (Translate, Localize) from the new component to import into applications where needed. The instantiating and initializing of this new component must occur before any rendering occurs.

The react-i18n-redux library also requires you add the i18nReducer reducer with the name "i18n" to be combined into existing reducer(s). This can be accomplished by adding the reducer in orion-components in src/services/reducers and exporting from /src/Services/index.js. Then import into each application and combine in app-src/reducers/index.js.

Applying Localization

While obviously all static text that is displayed in client for every application that has client code will require localization, there may be some not so obvious places. These are called out here for clarification.

- Rule text generation in rule-app

User Locale

The user's locale can be retrieved via Accept-Language header or other best practice and/or in the Settings app, the user can set their preferred locale, which would override the value determined by the browser.

Service Code Specification

Approach

Translations will be stored in the ecosystem RethinkDB database by locale and application ID. There will be a special ID “global” to store all non-app specific translations. Mostly, those should be leveraged in orion-components and possibly in node-app-core and client-app-core. Translations retrieval will be accomplished via a new rest API in the ecosystem project by locale and application ID. At the time of retrieval, global translations should be merged with app-specific translations, making all the translations children of the application ID or “global”.

API

API routes should be created to support retrieving a single language for an application or all languages in the following formats.

Load all translations for an application:
GET /ecosystem/api/applications/[appId] /translations

Load translations for a specified culture:
GET /ecosystem/api/applications/[appId] /translations/[culture code]

Model

A new method will need to be added to the application model in ecosystem to support the two new API routes. This method should have two arguments: appId and culture. The appId should be required, but the culture argument should be optional. This method will retrieve the translations from the database, filtered by the appId argument and filtered by the culture argument if provided.

Localization on the Server

There are scenarios where static text needs to be localized on the server. For this we will leverage the i18n package found at https://www.npmjs.com/package/i18next. AVERT C2 servers leverage the node-app-core library to load common functionality to all applications. Since every application is ultimately a candidate for localization, this functionality should be implemented in the node-app-core package and made available as a separate import into applications that require it. Loading of translations will follow a model similar to the app-config module’s loading of configuration data in node-app-core. The module will leverage the API created for retrieving translations by application and will initialize and export the i18next object required for applying translations and formatting. The file will be named localize.js.

Server Localization vs Client Localization

Client localization is derived from the user’s language preference either from a setting in AVERT C2 or the user’s system locale as provide via Accept-Language header. Server localization will be set on the server in ecosystem configuration and derived and set at initialization in the localize component in node-app-core. This will ensure that server localized data will be published (email/pdf) or stored (activity) in the language defined for the installation.

Email/PDF Templates

Various templates for generating email messages and PDF files across a few applications will need to be modified to leverage translations where applicable. Move templates text into sys_locales as key in appropriate app. Import localize from node-app-core and leverage for translations and formatting.

The following applications/files have been identified to require localization:

- ecosystem: email templates in config.json
- replay-app: email templates in logic/portable.js
- reports-app: pdfTemplates/*, api/reports.js email templates
- berth-schedule-app: pdfTemplates/*

Activities

Various parts of the system generate activities that can be leveraged for rules and maintain a history of operations performed on an entity and/or events they are involved in. The summary of an activity is stored on the activity and in associated notifications and will be translated prior to persisting and will be stored in language of the base system locale.

The following applications/files have been identified to require localization:

- ecosystem: Activities are generated within most of the model classes /models/*. A global search for generateObject in VSCode should reveal the location of all of them.
- rules-app: AlertGenerator.jsx (client activity), loiterActivityGenerator.js
- integration-app: bosch-camera-analytics-transformer.js, imex-alarm-transformer.js, vigilant-lpr-transformer.js, onaco-transformer.js, ngfr-mqtt-client.js, vigilant-lpr-transformer.js, weather-notification-transformer.js
- berth-schedule-app: vesselBerthingEventProcessor.js, vesselActivityModel.js

Database Changes

sys_locales

Store translations leveraging i18n file format, but further categorizing by application.

Schema

Property	Type	Description
id	string	RethinkDB auto-generated id property
locale	string	Known language identifier of language used in this translation
appId	string	Application translations are relevant to
translations	object	The label that will appear in the base map selection control

RethinkDB Internationalization Support

Tested Arabic support in RethinkDB and storage and filtering worked. Need to assess impact to dates.

Elasticsearch Internationalization Support

https://towardsdatascience.com/designing-an-optimal-multi-language-search-engine-with-elasticsearch-3d2de1b9636d

Possibility of requiring different analyzers to support different languages. However, we don't currently make heavy use of anything but keyword so language specific analyzers may not be required or even relevant. Need to research and test.