Change Request Tracker

DEVELOPER HANDBOOK

APRIL 11, 2019

Version 1.2

Developer Information

Wei Jie Zheng,
USCA Student

Table of Contents

1 Introduction

Change Request Tracker, i.e CRT, is a full-stack web project that included a API service, a web application, a desktop application, and third-party email service. The purpose of the project is to provide multiple methods for users to submit and track change requests. The project is developed by Wei Zheng during his Junior year at University of South Carolina Aiken as an Applied Computer Science student. The project is used as a capstone project proposed by Rural Sourcing Inc.

1.1 Techniques

1. Languages:
    a. **JavaScript** – Script Language.
    b. **Hypertext Markup Language 5** (HTML5) – Markup Language.
    c. **Casual Style Sheet 3** (CSS3) – Style Sheet Language.
2. Frameworks:
    a. **NodeJS** – Backend JavaScript Framework.
    b. **AdonisJS** – MVC Style NodeJs Framework.
    c. **VueJS** – Frontend Javascript Framework.
    d. **Bootstrap** – Frontend Design Framework.
3. Libraries:
    a. **KenxJS** – SQL Query Builder used in AdonisJS.
    b. **Vuex** – state management pattern for Vue.
    c. **Axios** – lightweight JS library used to perform HTTP requests.
    d. **Vuex-persistedstate** – persist and rehydrate Vuex states.
    e. **JQuery** – JS library that simplifies JavaScript tasks.
    f. **CKEditor5, vue-js-modal, Datatable, Select2, ChartJs, Bootstrap-dDatetime, DateRangeSeletorPicker, Pace, Lodash, Moment.**
4. Template:
    a. **AdminLTE**: an open source template built on top of Bootstrap 3.
5. Service:
    a. **Mailgun** –send, retrieve, and analysis emails.
    b. **Ngrok** – expose the local port to the public.

    c. **Github** – web-based hosting service for version control.

6. Other:
    a. **Visual Studio IDE** – extreme light weighted IDE developed by Microsoft Inc and supported by the communities.
    b. **MySQL** – free open source SQL database
    c. **npm & yarn** – JavaScript package managers.
    d. **Postmon** – developer tool used to test HTTP request.
    e. **PM2** – production process manager for Node.js applications. Allows keeping server application alive forever.
    f. **Babel** – JavaScript compiler. Able to transpile project to fit the older version of the browser.
    g. **Webpack -** bundle JavaScript files for usage in a browser.
    h. **PrettierJS** – editor format tool.
    i. **ESlint** – a linting utility for JavaScript.

1.2 Features

CRT system provides a notifications service and internal message service for everyone. The user is able to use the message service to interact with other users. And the notification service will record and notify the user for any changes made inside the system.

For clients only, CRT allows the client to submit change requests via email or using any provided applications. By visiting the CRT apps, the client is able to track the status of all submitted change requests and adding the messages to existing change requests.

For admins only, by visiting the CRT app, admins are able to generate a registration code to invite a new user, delete or update role of any client, submit change request on the behavior of the client, manage the status of all change requests submitted to the system, and review the weekly change requests status ratio chart.

1.3 Installation

[The project repository is hosted by GitHub. The source code is available at https://github.com/weijie0192/change-request-tracker]

Setup:

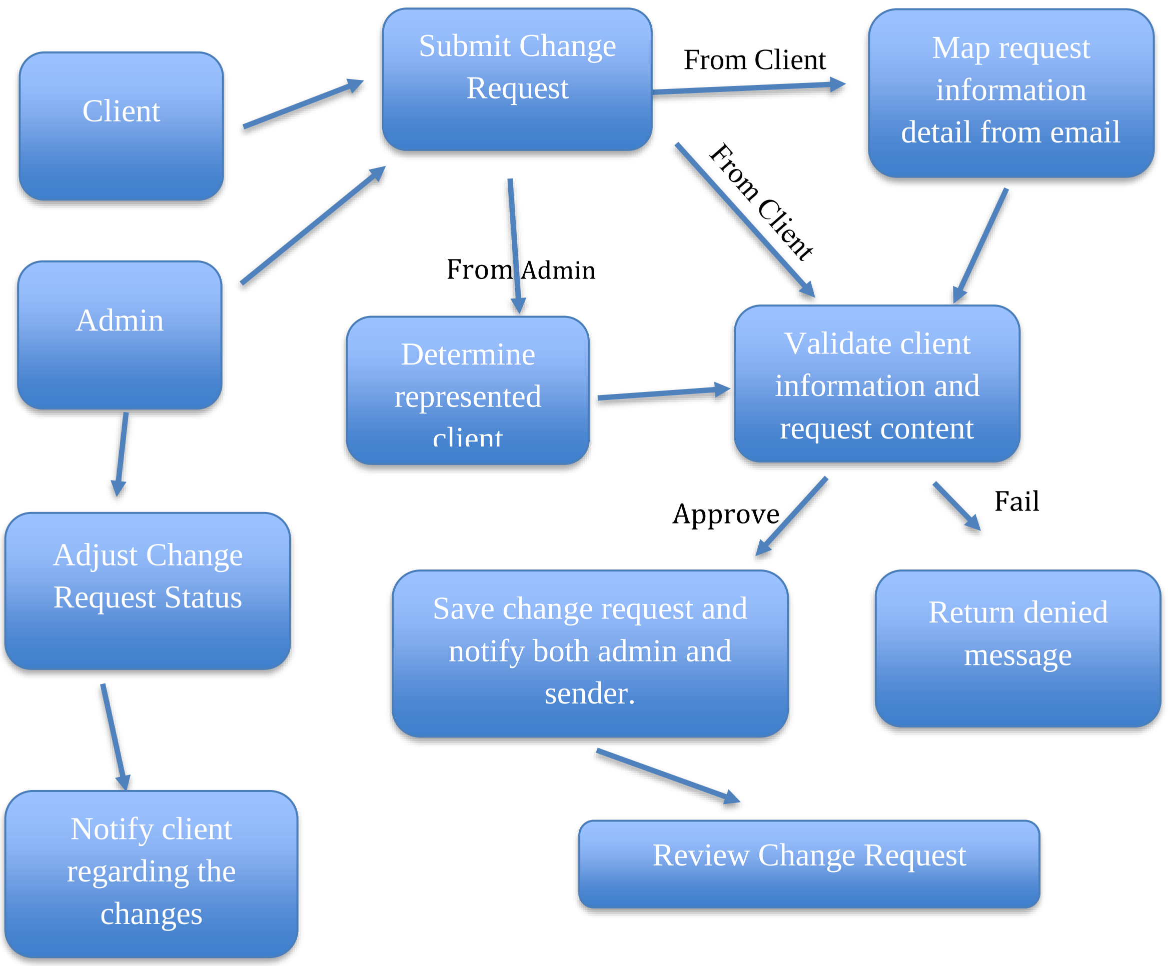
1. Install **Git**
2. Install **Node.js**, **npm**, and **yarn**
3. Run **npm i -g @adonisjs/cli** command to install AdonisJS cli
4. Install **XAMPP** for local server
5. Clone the entire project directory from GitHub using the command git **clone https://github.com/weijie0192/change-request-tracker**
6. Create .env file, using **.env-example** as the reference
7. Open server directory, then run the command **npm install** to install dependencies. Perform same instruction again under client directory
8. Open XAMPP and click start button for Apache and MySQL module. Then click Admin button under MySQL row to browse phpMyAdmin
9. Under the phpMyAdmin site, create a new database for the project
10. Return to server directory and run the command **adonis migration:run** to generate database tables
11. After migration, run the command **adonis seed** to generate default Developer account
12. Open server directory, then run the command line **adonis serve --dev** to deploy dev server
13. Open client directory located in main, then run the command line **npm run serve** to start local client-server
14. To visit the project website, enter the URL **localhost:8080** in any modern browser.
15. Login default account with the email **no-reply@rsicrt.com** and password **weijie0192**

1.4 Problems / Solutions

Problem	Solution
AdminLTE element event does work when added dynamically.	Re-initiate element event after the elements were added.
Elements initializing process not work because synchronize conflict with the HTTP request.	Before initialize element, perform a10 milliseconds delay using setTimeout() after HTTP request is completed. This allows the system to setup incoming data from the server.
JQuery not found.	Declare a global '$' value for JQuery in main.js. Or setup $ in vue.config.js
Client browser overwhelmed when retrieving one million data from the server at once.	Use server-side process method.
Resources are not found when deployment to server	Config server proxy. Catch all resource folder.
Mail service incoming email cannot deliver to local server.	Use Ngrok relay server to expose the local server to the public.
Pick by week not supported by Bootstrap-datepicker library.	Modify change date event. When date change, transfer the highlight on a selected date to its parent week.
Some browser not supporting ES6 syntax and promises.	User a tool named babel to transpile ES6 code into ES5 which supported by these old browser.

2 Overview

2.1 Workflow


Client
Admin
Submit Change Request
From Client
Map request information detail from email
From Client
From Admin
Determine represented client
Validate client information and request content
Approve
Fail
Adjust Change Request Status
Save change request and notify both admin and sender.
Return denied message
Notify client regarding the changes
Review Change Request

2.2 Data Dictionary

User – individual with the ability to use the system. Every user has abilities to receive notifications and track notifications. Users can also interact with another user by using the in-site message system.

JWT Token – a token used by the server to identify user. Every time when a user successfully login, a token will be generated by the server and send back to the user. Each JWT token will be expired 24 hours after its creation.

Admin – a user who is authorized to manage the system. Admin has the right to review every change request in the system and make changes to them. Admin also has the ability to create a user by generating a registration code, change the role of a client, and delete a client account. Admin cannot submit change request directly, instead, they can only submit a change request on the behavior of a client.

Client (role)– a user who has very minimal authority in the system. A client can submit a change request directly under his/her name. Clients can only review change requests that have their identity on it. Email change request submission is also available for the client by sending emails to *change-request@rsicrt.com*

Developer – a user who give their best to assist develop the CRT system. Developer all the rights of a client and an admin. Developer's title can be downgraded by themselves, but their rights remain the same.

Change Request Email Submission – a feature for the client. The subject of the email will be used as change request title, the body of email used as change request detail, and the sender will use as the identity of the client. Before process into change request, the system will validate the email to match the following conditions: sender has a client account in the system, the email subject is not empty, and email body is not empty.

Change Request: a form that represents a client's proposal for an alteration to some production or the system itself. Change Request can only be submitted under the name of a client. The user can track the status of the submitted change request.

Change Request Content: A field that displays the title and the detail of change request. Admin has the authority to make changes in this field.

Change Request Message: After a change request is submitted, the client and admin are able to interact with each other regarding the issue about this change request by sending change request messages.

Change Request History: a field inside each change request detail that list every alternation made to this change request in descending order. History items are organized by dates.

Header Menu: Three menus that located on top of the web app. clicking the associated symbol located on the top of the CRT website. The three menus are message list, notification list, and flag list.

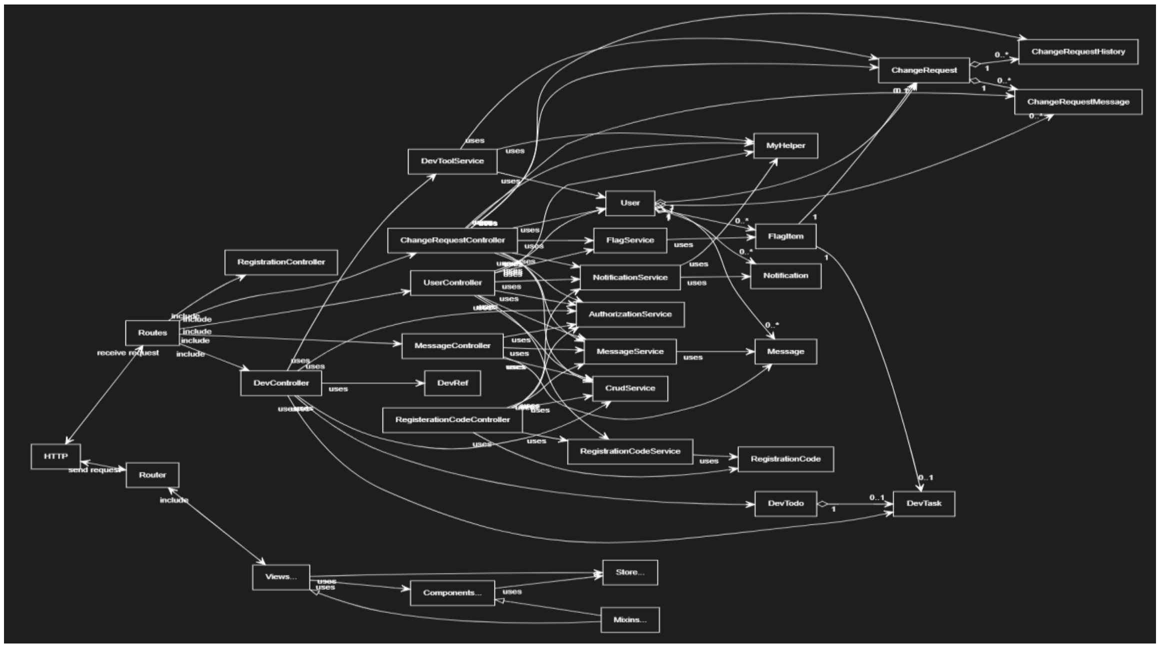
Sidebar: navigation panel located on the left side of the CRT website. Sidebar allows the user to navigate through the site. Each navigate item is linked to a component.

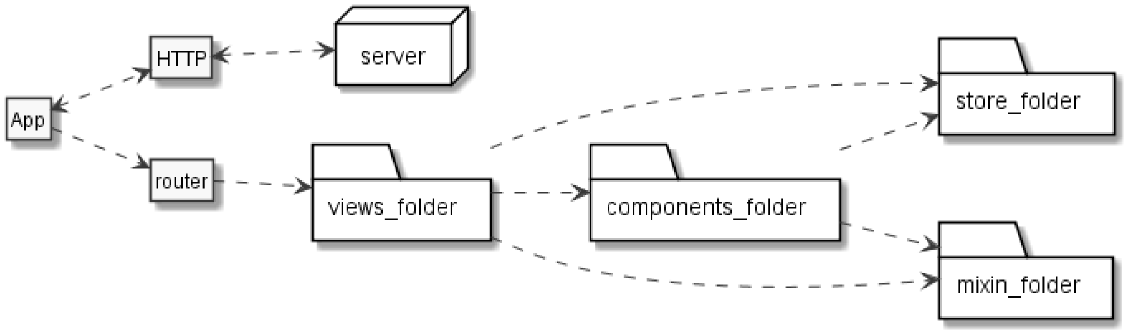
Control Bar: a panel that can be active by click gear icon located on the top right corner of CRT website. The panel is used to control the website theme and some layout setting.

Registration Code: a code used to register a user account. Only admin has the ability to create a registration code. During creation, admin can preset user information and set editability of preset information. The registration code will automatically send to receiver's email after it generated.

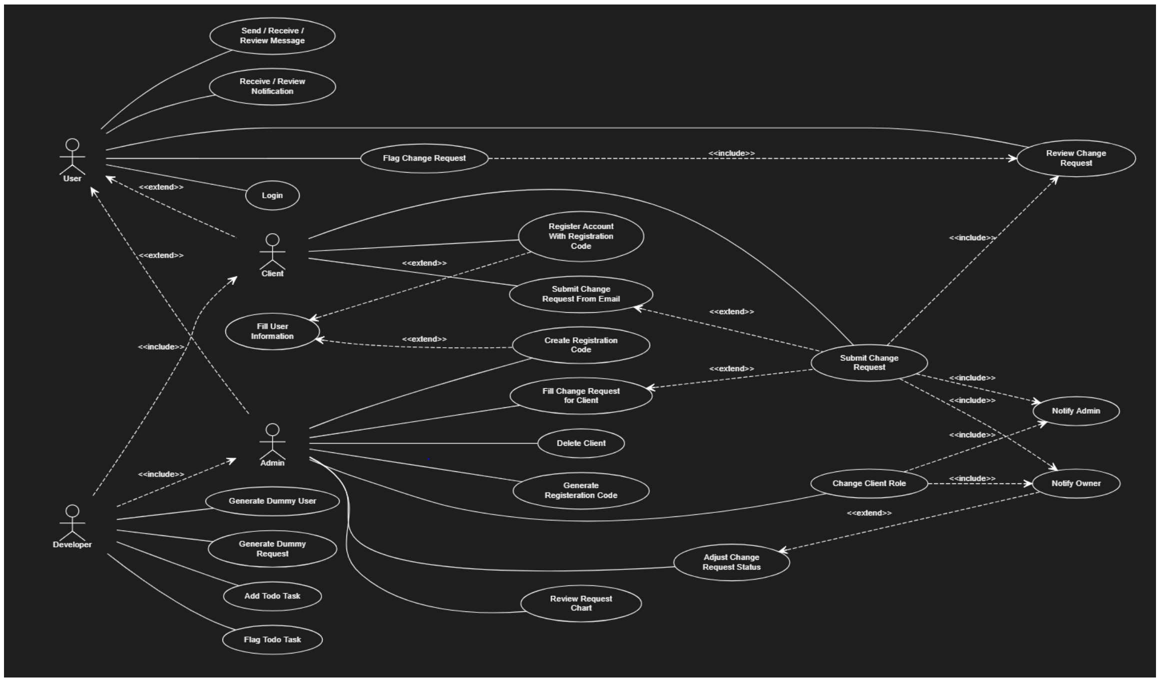
Http: a transfer protocol to transfer data between the front-end and the back-end. Front-end will send HTTP request to the server and the server will return http response.

2.3 Class Diagram (Entire Back-End and Shortened Front-End)





2.4 Use Case Diagrams



3 Back-End Layer

The back-end is a portion of the system where remain unseen by users. Every data user saw is processed and retrieved from the back-end. The back-end server handles database query, data process, data hash, and data. The back-end of CRT is a RESTful API service build on top of HTTP protocol. The entire server is stateless and will only respond when an HTTP request is submitted.

3.1 API Routes

[server/start/routes.js]

The server API routes are used to map incoming request. In this project, each route will have “**api**’’ prefix and pair up with a controller method.

3.1.1 Unauthenticated Routes

The user is not expected to provide JWT Token in each request header.

Route:
POST /auth/register

Description:
Register a user account

Body:

```
{
   "email",
   "password",
   "code",
   "first_name",
   "mid_initial,
   "last_name"
}
```

Route:
POST /auth/login

Description:
Login user account. Return the user ID token

Body:

```
{
   "email",
   "password"
}
```

Route:
GET /regist-code/verify/:code

Description:
Validate if the registration code exist in the CRT database

Route:
POST /change-request/mail-submit/:key

Description:
Take incoming emails from Mailgun service

Body:

```
{
   Mail_JSON
}
```

Route:
POST /change-request/mail-request-info/:key
 	
Description:
Process request from email and return change request information
 	
Body:
`{`
`   Mail_JSON`
`}`

3.1.2 Authenticated Routes

The user is expected to provide JWT Token in each request header, else the request will be denied.

Route: GET /util/flag **Description:** Return list of a flagged item owned by the current user (e.g. flagged change request)
Route: GET /util/notification **Description:** Return all unread notifications and notifications from the last 3 days of the current user
Route: GET /util/msg **Description:** Return all unread and marked messages of the current user
Route: POST /util/notification/paginate **Description:** Return paginated notifications list of the current user. Used in notification datatable

Body:

```
{
   DataTable_JSON
}
```

Route:
PATCH /util/notification/clear-new/:target

Description:
Set the select notification from unread to read

Route:
GET /user

Description:
Return information of current user

Route:
GET /user/:email

Description:
Return a user by email

Route:
POST /user/search/:role

Description:
Return list of users filtered by the search data

Body:

```
{
   "term",
   "page"
}
```

Route:
POST /user/datatable

Description:
Return list of users. Used to perform datatable server process

Body:

```
{
   DataTable_JSON
}
```

Route:
DELETE /user/:id

Description:
Delete user by id

Route:
PATCH /user/:id

Description:
Update user by id

Body:

```
{
   "role": "Client || Admin || Developer"
}
```

Route:
POST /regist-code

Description:
Create new registration code

Body:

```
{
   "allowEdit",
   "content",
   "email",
   "first_name",
   "last_name",
   "mid_initial",
   "role": "Developer || Admin || Client"
}
```

Route:
GET /message/:id

Description:
Return message by id

Route:
POST /message/list

Description:
Return all message of the current user in the paginated formula

Body:

```
{
   "limit",
   "page",
   "search",
   "type": "inbox || sent || archive"
}
```

Route:
POST /message

Description:
Create new message

Body:

```
{
   "content",
   "title",
   "receiver": "[ "Name (Email)" ]"
}
```

Route:
PATCH /message/clear-new

Description:
Mark the message as read

Route:
PATCH /message/archive

Description:
Toggle the message archive status

Body:

```
{
   "isArchived",
   "list"
}
```

Route:
PATCH /message/:id

Description:
Update message by id

Body:

```
{
   "isRead",
   "isArchived",
   "isBookmark"
}
```

Route:
POST /change-request/list

Description:
Return change request list of the current user filtered by request tabs

Body:

```
{
   "method": "tab",
   "tab": "active || all || Cancelled || To Do || In Progress || Complete"
}
```

Route:
POST /change-request/admin/list

Description:
Return all change request in the system and filter by the request

Body:

```
{
   "method",
   "date",
   "id",
   "clientsName": "[ Name ]",
   "status": "Cancelled || To Do || In Progress || Complete",
   "tab": "active || all || Cancelled || To Do || In Progress || Complete"
}
```

Route:
GET /change-request/chart/:range

Description:
Return change request status ratio of requested date range. Used in ChartJS

Route:
GET /change-request/:id

Description:
Return change request by id

Route:
POST /change-request/

Description:
Create new change request

Body:

```
{
   "client",
   "message",
   "details",
   "title"
}
```

Route:
PATCH /change-request/:id

Description:
Update change request content by id

Body:

```
{
   "status": "Cancelled || To Do || In Progress || Complete",
   "details",
   "title"
}
```

Route:
POST /change-request/search/:target

Description:
Return list of change requests filtered by the search data

Body:

```
{
   "term",
   "page"
}
```

Route:
DELETE /change-request/:id/unflag

Description:
Delete change request in flagged list

Route:
POST /change-request/:id/flag

Description:
Add change request into flag list

Route:
GET /change-request/:id/msg/:num

Description:
Return requested number of messages in requested change request id

---

Route:
POST /change-request/:id/msg

Description:
Create a message for a change request

Body:

```
{
   "content"
}
```

Route:
DELETE /change-request/msg/:id

Description:
Delete change request message by message id

Route:
GET /change-request/:id/hist

Description:
Return history of requested change request id

Route:
GET /dev

Description:
Return all developer to-do groups

Route:
POST /dev/todo

Description:
Create new to-do group

Body:

```
{
   "content",
   ",description"
}
```

Route:
DELETE /dev/todo/:id

Description:
Delete to-do group by id

Route:
PATCH /dev/todo/:id

Description:
Update to-do group by id

Body:

```
{
   "content"
}
```

Route:
POST /dev/todo/:id/task

Description:
Add a task to to-do group by id

Body:

```
{
   "content",
   ",description"
}
```

Route:
DELETE /dev/task/:id

Description:
Delete task by id

Route:
PATCH /dev/task/:id

Description:
Update task by id

Body:

```
{
   "content"
}
```

Route:
PATCH /dev/task/complete/:id

Description: Set task to complete **Body:** `{ "isCompleted" }`
Route: DELETE /dev/todo/:id **Description:** Delete to-do group
Route: GET /test/generate/user/:num **Description:** Generate number of dummy users
Route: GET /test/generat/cr/:num **Description:** Generate number of dummy change requests
Route: GET /test/correctCR **Description:** Refresh change request data field after the data got mess up by testing

3.2 Authentication

The CRT support three role types: developer, admin, and role. Refers to *2.2 Data Dictionary* for more information about each role. The CRT server uses AdonisJS authentication provider middleware to authenticate the user. Every authenticated route will require the user to have JWT token in their request header. Some functions like generate registration code will perform a user role verify before processing.

3.3 Model / Database

CRT uses code first migration method with the help of KnexJS. Each migration file is used to create a database table. And each database table is linked to an lucid model. And each model is used as a storage for database queries.

3.4 HTTP Controllers

HTTP controller is classes that used to handle all incoming and outgoing data from server. When an http request is received, the controller will select service to process the request and then return the result as http response. CRT server has six controllers: UserController, ChangeRequestController, MessageController, RegistrationController, DevController, and UtilityController.

3.5 Services

Service are used to handle business logic, these logic included if-else statements, query selections, and etc. CRT server has eight services: RegistrationService, FlagService, MessageService, NotificationService, UserService, ChangeRequestService, MailService, and DevService.

3.6 Helpers

Helpers are classes with static methods that used to perform utility jobs like model mapping, calculation, and verification. CRT has two helper classes: MapHelper to map data and VerificationHelper to perform verifications.

3.7 Exceptions

CRT has three user-defined custom exceptions: InvalidAccessException, RegistCodeNotExistException, and ResourceNotExistException.

4 Front-End Layer

Front-end layer consists of the everything user sees or able to interact with. Front-end app runs in client side; therefore, the speed of the app will depend on user's device.

4.1 Vue Routers

[client/src/router.js]

Vue router uses browser URL to identify which page user wants to see. Each Vue router route is linked to a view component. Vue router also handles event before the user enters the router or after user enters the router. In CRT, when a user enters a route, Vue router will perform a verification to verify if the user has the right to enter the page.

Routes	Description
/login	Login page where the user can enter their username and password to access the system or enter a registration code to access register page.
/register	Register page where the user can fill in information to register an account.
/	Default page when the user logins. The page displays dashboard to the user.
/mailbox	Mailbox page where users can review all received or sent messages. The user is also able to compose a new message on this page.
/notifications	The page that displays all notifications for the current user.
/dev/todo	Developer To-Do page. Only developers are allowed to enter. Display all developer to-do tasks.

/dev/tool	Only developers are allowed to enter. The page is a place where a developer can generate dummy data or perform testing.
/admin/user-list	Only admins are allowed to enter. The page is a place where admin manages all users and make changes to the client account.
/admin/chart	Only admins are allowed to enter. The page will display the change request status ratio in the selected week.
/admin/generate-code	Only admins are allowed to enter. The page allows admin to create a new registration code by filling out the form.
/admin/change-request	Only admins are allowed to enter this page. The page displays all change requests in the system.
/admin/change-request/search	Only admins are allowed to enter this page. The page allows admin to search change request by entering date range, request id, client name, or request status. The page also displays a table of the result.
/change-request/entry	The page where the client or admin can fill out a form to create a change request.
/change-request	The page displays all change requests owned by the current user.
/change-request/:id	The page displays change request detail. Only admins or the request owner can enter this page.
/change-request/:id/content	The nested page that displays the selected change request title and detail. Admin is able to modify request title and detail under this page.
/change-request/:id/message	The nested page that displays the selected change request messages. Client and admin

	are able to interact with each other by sending messages on this page.
/change-request/:id/history	The nested page that displays the selected change request history.

4.2 Main / App

Main.js is the beginning of the entire VueJS app. In main.js, we define our libraries and global variables on the top of the file.

App.vue is the parent of all views and components. In CRT, App.vue contains two components, Main component that displays the sidebar, controlbar, and header; and Auth component that displays Login page or Register page. App.vue also include HTTP request methods to retrieve current user’s information.

4.3 Assets

Assets is the directory where all external resources are stored. External resources included an image, libraries, template, custom JavaScript file, and custom CSS file.

4.4 Stores

Stores is used to manage data state. Vue store is supported by Vuex library. With the help of stores, data can be shared across every component and will be saved as cookies in a local browser.

4.5 Mixins

Mixins contain methods for use by other components. Mixins is very similar to inheritance but without having a parent-child relationship. There are three mixins module declare in CRT. Helper.js to provide utility methods, rowEvent.js provide table row click event, and sharedList provide reusable methods for navigation menu list and dashboard.

4.6 Components

Components are reusable blocks of code that can have both structure and functionality. Components are separated into three parts: Html template, Javascript for functions, and CSS for styles. There are nine components, only count components inside the components directory, used in the CRT system.

4.7 Views

Views are the component that has a URL route linked to it. Typically view is consider a page where user sees. Refer to *3.1 Vue Router* for more information about each individual view.

5 Mail Service

- CRT uses Mailgun to provide mail service. Mailgun offers a complete cloud-based email service for sending, receiving and tracking email. Incoming email will first be received by Mailgun server, then re-route to CRT back-end server through HTTP request POST method to ***/api/change-request/mail-submit***.
- Rsicrt.com is the domain address for the CRT mail service.
- The user will receive a registration code from ***no-reply@rsicrt.com***.
- The client is able to submit a change request by sending an email to ***change-request@rsicrt.com***.

6 Reference

- Helps from Google – https://www.google.com
- Helps from Stack Overflow -https://stackoverflow.com/
- Helps from GitHub communities - https://github.community/
- Helps from YouTube video - https://www.youtube.com/watch?v=dfEZlcPvez8
- AdonisJS doc - https://adonisjs.com/docs/4.1/
- VueJS doc - https://vuejs.org/v2/guide/
- Vue guides from other site –
  - https://dev-notes.eu/2018/05/passing-data-between-vue-components/
  - https://markus.oberlehner.net/blog/vue-router-page-transitions/
  - https://alligator.io/vuejs/component-lifecycle/
- Query Builder - https://knexjs.org/#Builder-where
- Icon search - https://fontawesome.com/icons?d=gallery
- AdminLTE Template - https://adminlte.io/themes/AdminLTE/index2.html
- Prettier formatter installation guide - https://prettier.io/
- Select2 doc - https://select2.org/
- Mailgun doc - https://documentation.mailgun.com/en/latest/api_reference.html
- Datatable - https://www.datatables.net/examples/index