Claims Management Platform (CMP) API Documentation

By Carl Ngan

Introduction

This is the documentation for the API of Claims Management Platform (CMP). The platform allows users (authorized personnel) to create, read, update, delete, and manage claims intelligently and efficiently. The purpose of exposing an API is so that any amount of apps, interfaces, or devices can interact with the same set of data without having to directly connect to the database. This provides a layer of security, guarantees abstraction, and maintains data

integrity. As of March 22, 2016, this is nowhere near production ready, but sufficient for my upcoming interview with Mitchell, fulfilling all the requirements -- at least I hope it does.

Related links

- The API - http://api.cmp.carlngan.com/
- The Data Management Portal (DMP - the user interface) - http://api.dmp.carlngan.com/

Last updated

March 23, 2016

Backlog

Below are the features that would be nice to have (or things that I would do) if I were to make this a production ready product one day:

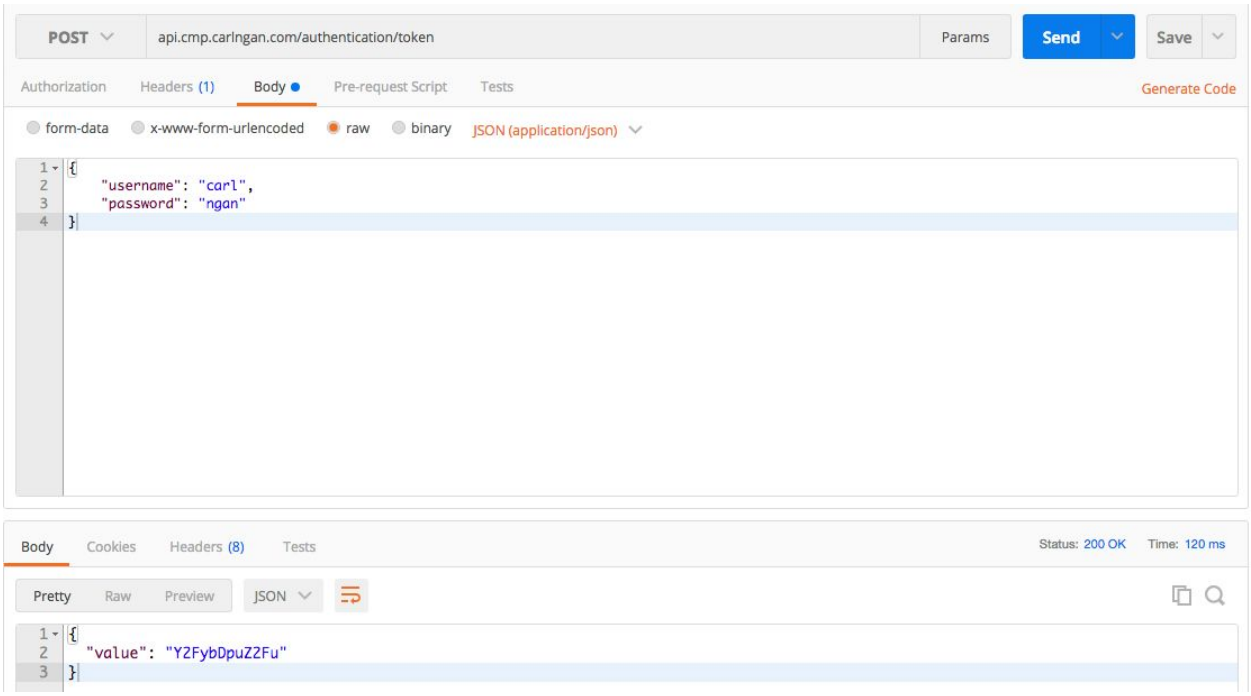
- Run HTTPS with cert
- Implement a real OAuth2 server, handle real access and refresh tokens. Accept multiple grant types.
- Implement true elasticsearch.
- Allow the API to communicate (receive and serve) both JSON and XML.
- Write more thorough tests for each API endpoints and think of more corner cases to ensure robustness of product.
- Build an error directory so users can look up their error code for more explanation and possible causes/solutions.

Usage

Although this API is exposed to the public, the endpoints for interacting with claims are protected and any app making a request must present the API server with a token. Since we will be interacting with the API directly in this documentation, I will be using Postman (https://www.getpostman.com/) to demonstrate basic usage. A more technical documentation can be found below in the "Developers" section.

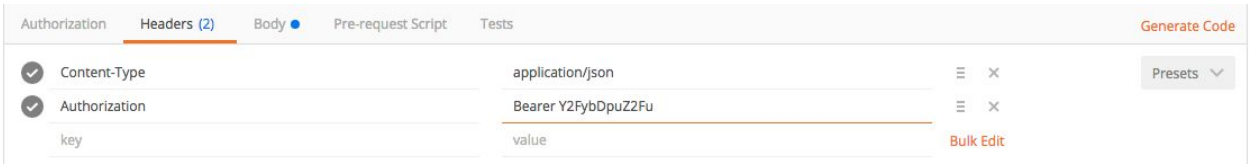
Authentication

As mentioned above, we need a token to make http requests to the API. To get a token, send a POST request to api.cmp.carlngan.com/authentication/token with headers: Content-Type: application/json and body: username: carl, password: ngan .



We will get an object in return that has the key: value, which will contain our access token.

To make requests, we would need to include the token in header of every request we make. Simply add the header: Authorization: Bearer + {token}



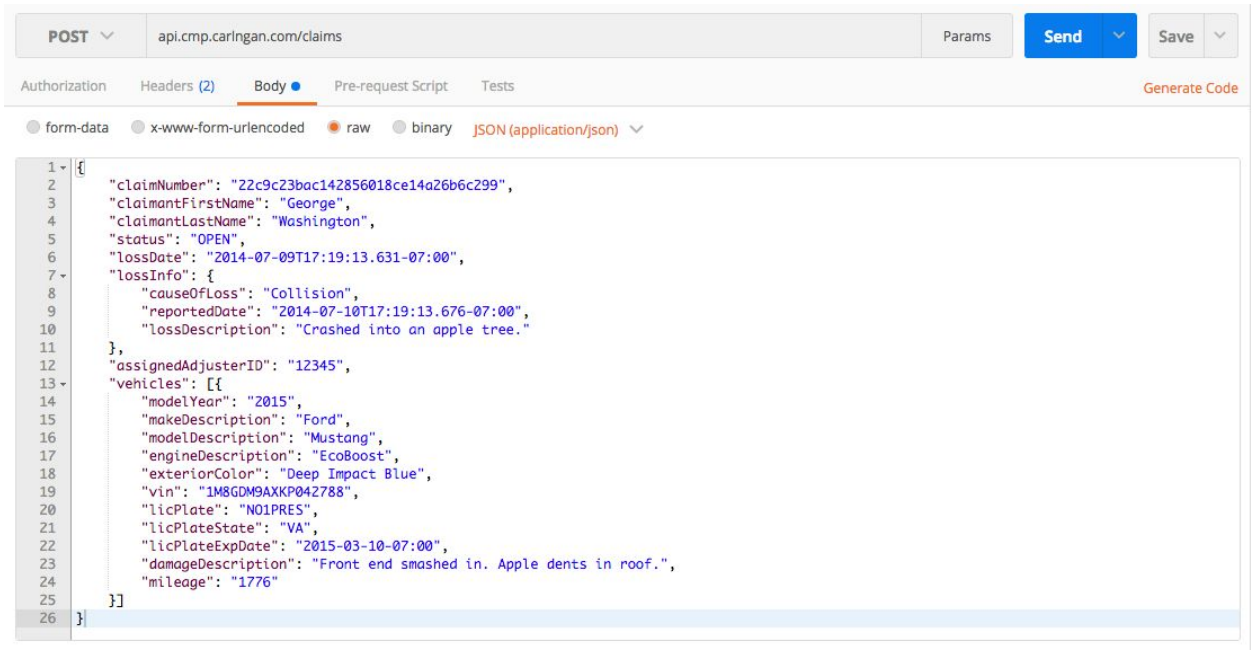
To logout, which supposedly destroys our access token, simply send a GET request to api.cmp.carlngan.com/authentication/logout

Making Requests

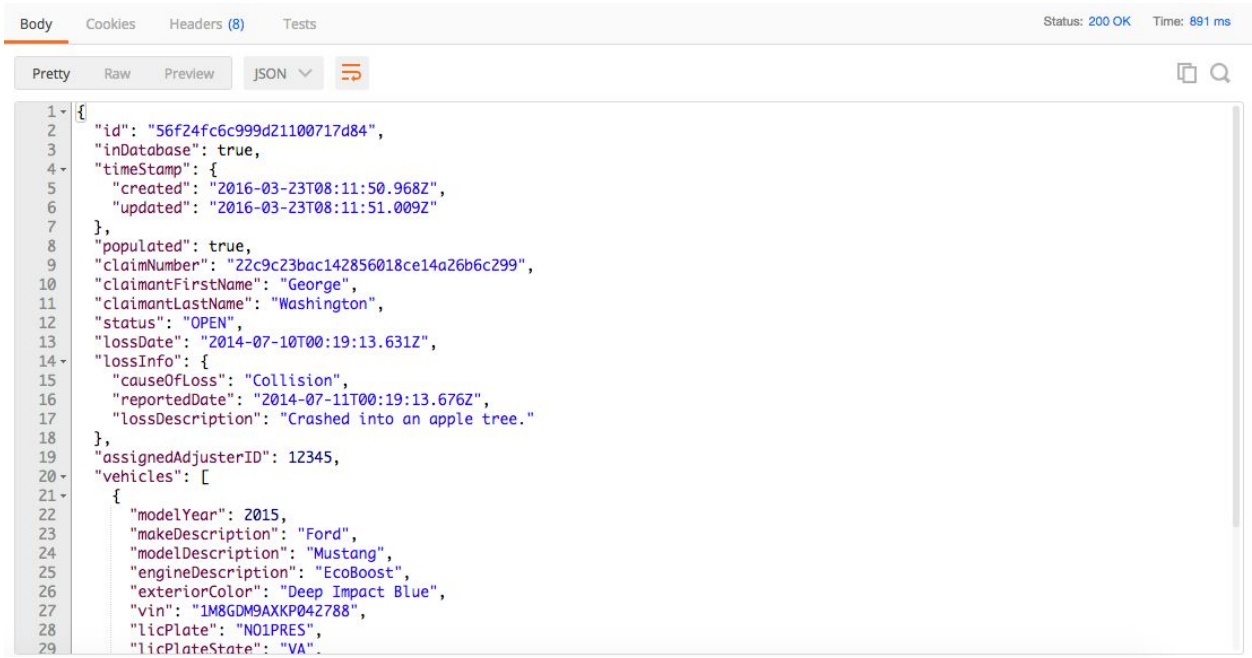
Now that we have an access token, we can start making http requests. We will only be going over the basic usage of these requests. To see all possible parameters and methods to interact with these requests, see our swagger doc at: api-docs.cmp.carlngan.com

Create a claim

To create a claim, we can send a POST request to api.cmp.carlngan.com/claims with a body like the following:



What we will get in response will be something like:



Note: The only field that is required is the claimNumber -- in fact we can create a claim with only a claimNumber. Keep in mind that claimNumber must be unique and the API will give us an error if we try to create a claim with a claimNumber that already exists in the database. We will talk about errors later on.

Update a claim

To update a claim, we must have the id of the claim object. If we only have the claimNumber, we can lookup (find) that claim, and then update. If we change the claimNumber, the new claimNumber must not already exist in the database, otherwise the API will return us a friendly error.
To create a claim, we can send a PUT request to api.cmp.carlngan.com/claims/{claimID} with a body like the following:

You should get a response like:

Delete a claim

To delete a claim, we simply need the id of the claim. Again, we can lookup a claim using find as long as we have some information about the claim.
To delete, we send a DELETE request to api.cmp.carlngan.com/claims/{claimID}

The response is simply 200 OK if no errors were encountered.

Validate a claim

In the interest of maintaining data integrity, we can validate a claim in XML format against the provided XSD by sending a POST request to api.dmp.carlngan.com/claims/validate with an object with the key xmlContent as json.

Find a claim

If we want to find one particular claim, we only need to know the claim object id, or the claimNumber.
If we know the object id, we send a GET request to api.cmp.carlngan.com/claims/id={claimId}.
If we know the claimNumber, we send a GET request to api.cmp.carlngan.com/claims/claimNumber={claimNumber}.

Find claims

If we want to see a list of claims, or find claims that fall under a set of criteria, we send a GET request to api.cmp.carlngan.com/claims with the possible query string parameters:

- include: indicate which fields to include or populate separated by commas(,) and no space after the comma. Example: api.cmp.carlngan.com/claims?include=claimNumber,claimantFirstName,status Default, or if left blank, will include claimNumber and claimantFirstName
- exclude: works just like include, except we specify which fields to omit. We can specify either include, exclude, none at all, but not both. In the event both is specified, include will take priority. By default nothing is excluded.
- paginate: we can specify whether we want the API to paginate our results. Default is true. If false is specified, all results will be returned.
- perPage: assuming we have pagination turned on, we can specify how many results we want per page.

- page: pagination setting: we can tell the API which page to give us if there are multiple pages in our results.
- sort: specify the order to sort in. 1 is for ascending and -1 is for descending. Default is 1
- sortBy: specify the field to sort by. Default is sort by lossDate
- search: provide a term to search results. Currently only searching by claimantFirstName is implemented.
- rangeStart: use this to search for a range of lossDate results. rangeEnd must also be specified. The API prefers an ISO time string, but it will attempt to convert any time formats.
- rangeEnd: use this to search for a range of lossDate results. rangeStart must also be specified.
- status: search claims that have a particular status. Default includes all statuses.

Count claims

Sometimes we only want to know how many claims fall under a certain criteria, but do not necessary need the whole list of claims. In that case, we send a GET request to api.cmp.carlngan.com/claims/count with query parameters. The parameters we can pass in are the same as the above section: Find Claims.

Dealing with Errors

Whether we miss validations, pass in wrong values, or interact with the API in unexpected ways, the API will intelligently return us error objects specifying the error code, the error message, and error = true. The error message is usually a brief sentence saying what is wrong and what the user should do to fix the error. The error code allows the user to lookup the error for a more detailed when our error documentation is built :)

Developers

Running the project

1. Navigate to our desired folder(it should be blank), or create a new folder for this project (i.e. cmp-api)
3. cd into the folder: `cd cmp-api`
4. Clone the repo into this folder: `git clone https://github.com/carlngan/cmp-api .`
5. `npm install` or `sudo npm install`
6. Make a file called ".env" -- `vim .env`
7. Paste the following content:
```
```

EXPRESS_SECRET=CARL
NODE_ENV=development
PORT=3001

```

9. `npm start`

10. We can test locally by making HTTP requests to "localhost:3001"

## Environments

Our environment needs to be defined because it a very dangerous to let the server infer. We use a different set of data for each environment.

## Databases

We have three databases: one for production, one for development, and one for testing. Hence it is very important to set NODE_ENV correctly. We use mongolab (http://mlab.com) for our databases. The credentials are found in ./config/database.json

## Directory layout

./bin/
- www //entry point to the app. Make server and call /app.js

./config/
- database.json //specify database credentials

./modules/
- authentication/
  - api/
    - Authentication.js
  - src/
    - AccessToken.js
    - AuthFactory.js
    - AuthMiddleware.js
- claims/
  - api/
    - claims.js
  - schemas/
    - claims.js
  - src/
    - Claim.js
    - ClaimFactory.js
```

```
                - ClaimLossInfo.js
    - errors/                   //classes and objects dealing with errors
    - generics/src/             //classes and objects that are generic
./test/
    - claims/           //folder for files that we need for claims tests
    - authentication.js         //tests for authentication
    - claims.js                 //tests for CRUD claims
./.env                  //declare environment variables -- must have this file
./.gitignore            //specify which files and files to ignore for git
./app.js                //entry point to the app after ./bin/www. Server level configurations should be defined here
./circle.yml            //settings for circle continuous integration
./package.json          //npm dependency manager
./README.md             //project readme file
```

Coding convention

I try to follow the airbrb javascript guide (https://github.com/airbnb/javascript). I can use ES6 features because this is only a server sided app and we can guarantee the server running in a certain environment. Sadly, this cannot be done on the browser side yet.

Testing

I use mocha (https://mochajs.org/) for running my tests. I only test the API endpoints and not the individual methods.

Endpoint Documentation

I use swagger (http://swagger.io/) for documenting the endpoints. The interface also has a nice feature for us to test the endpoints while seeing all possible parameters to use.

Continuous Integration

When a commit is push, CircleCI (https://circleci.com/gh/carlngan/cmp-api) will automatically run our tests to make sure our build is not broken. If we pushed to master, circleci will automatically deploy the app for us.

Deployment

For this project, I use Heroku (http://heroku.com) for deployment since it integrates well with circle and github. Personally, I use AWS (https://aws.amazon.com/) for my own projects.

Final words

This was a fun project overall, though I wish I had more time to work on it. I was given the project about 2 weeks ago and had only gotten the chance to start working on it 2 days ago. Finishing up with school (finals) is not an excuse, but it did consume much more time than I had expected. Although I have not started coding until 2 days ago, I have been thinking about this project, from the end user's perspective. I think I have over engineered certain components and made it more complex than it needs to be for myself. If I had the chance to do this project again, I would really prioritize a list and do what is more important/critical first, and less on trying to design and build a production ready product.