

CXP-Node-API

eFlexervices Philippines,  
Inc.

*Version 1.0*

January 29, 2022

---

## INTRODUCTION

The [CXP-Node-API repository](#) found at GitHub is a Node.js service to support the Victorious customer experience platform. The API intends to make a leaner and cleaner codebase that was derived from its original repository using C# and .NET Core frameworks. The [C# version of this project](#) was originally developed on September 28, 2020. Work on the Node repository started on August 12, 2021 and is presently being maintained.

Third-party libraries such as Amazon Web Services (AWS), Asana, Google APIs, and Hubspot are integrated in the project that extends the core functionality of the project, introducing Google Analytics and the ability to create, retrieve, and update tasks from Asana, among others.

Statistically wise, there have been **783** commits in the master branch of the C# repository and **8** commits in the master branch of the Node repository, with pending **303** commits unmerged from the fix-api branch.

This software documentation was created on January 20, 2022 and finished on January 29, 2022.

---

# TABLE OF CONTENTS

	Page
INTRODUCTION	2
TABLE OF CONTENTS	3
DEFINITION OF TERMS	5
FEATURES	6
Authentication	6
Mail	6
RESTful API	7
THIRD-PARTY SERVICES	8
Amazon Web Services (AWS)	8
Asana	8
Google APIs	8
HubSpot	9
SCOPE AND DELIMITATION	10
SOFTWARE DEVELOPMENT TOOLS	11
PROJECT	12
Downloads	12
Setup	12
Disable logging	12
package.json	12
Testing the API	14
Schema	15
API DOCUMENTATION	19
GET Methods	19
POST Methods	21
PUT Methods	23

---

DELETE Methods	25
REFERENCES	26
APPENDICES	27
Screenshots	27
.env variables	30
Terminal commands	32

---

## DEFINITION OF TERMS

Listed below are some of the terms that may be mentioned in the documentation to enhance the reader's understanding of the concepts.

Bearer tokens. Bearer authentication is an HTTP authentication scheme. It can be understood as “give access to the bearer of this token.” The bearer token is a cryptic string, usually generated by the server in response to a login request. The client must send this token in the Authorization header when making requests to protected resources.

CXP. Simply means “customer experience”. The CXP-Node-API project is an implementation of the customer experience platform of the client.

JSON Web Token (JWT). An open standard method of defining a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs are used as a form of authentication when using the API.

Tokens. Access tokens represent the authorization of a specific application to access user data. Access tokens must be kept confidential. Only the application, authorization server and resource server can utilize access tokens. Because tokens identify a user using the application, it must be secured as passing this to a non-secure network will make the application susceptible to attacks.

---

# FEATURES

## Authentication

The authentication used by the project is Passport, an authentication middleware that is widely used in the world today with over 500 strategies to use. The project requires users to be logged in to get access to other protected routes. A JWT is signed when a user makes a request to sign in. This token contains the following user information: **Id**, **Email**, **TenantId**, **PodId**, and **Role**. If a user is successfully logged in with the correct username and password, it returns the following response: a bearer token, the bearer token's expiration in local date and the User information (where the password fields are hidden). If a login is unsuccessful, either an incorrect password or email response is returned, depending on which field is incorrect.

The registration requires users to supply an email, password, user type and ds type. If an email is not found in the database, the registration is successful and returns the same response as when a user is logged in successfully. In addition, the password is hashed using the **bcrypt** package before being stored in the database.

One can verify the validity of a JWT by making a POST request to **/passport/auth/verifyjwt** route and pass a Token key with the value of the bearer token. This will return different responses that range from an expired token, a malformed token, an invalid signature, the token expiration (if token isn't expired yet), etc.

## Mail

The system features the ability to send mails. This is specifically used when a customer or Victorious member is invited using their email addresses. Inviting users this way is also another form of registration, as they become users of the system once they accept the invitations they received through email.

---

## RESTful API

REST stands for representational state transfer and is a set of architectural constraints, not a protocol or a standard. Making requests to endpoints isn't cumbersome as it will return a JSON representation of what the client needs. The **response.data** object, if accessed in a log (eg. `console.log`), contains helpful information to the user. Each response returned by the API will contain 4 keys: `message`, `error`, `statusCode` and `data`. There will be an array or an object in the `data` key if responses are successful, while in most cases, the value of this key is an empty array or null if there are no results returned by a GET request or there was an error, respectively.

It also conforms to best practices to avoid showing the password field on responses, as this is one crucial data that cannot be shown to the user as well as proper error handling if a certain resource cannot be found (404), a bad request (400), or system errors (500) among others.

---

## THIRD-PARTY SERVICES

Some of the third-party services used by the project is as follows:

### Amazon Web Services (AWS)

Amazon Web Services offers reliable, scalable, and inexpensive cloud computing services. AWS is a subsidiary of Amazon, providing on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis. File uploads, such as logos and contracts are uploaded into AWS S3 buckets, which is a public cloud storage resource available in Amazon Web Services' (AWS) Simple Storage Service (S3), an object storage offering. Some of the functions used are uploading files to S3, deleting files to S3, and getting a pre-signed URL request from a bucket.

### Asana

Asana is a web and mobile application designed to help teams organize, track, and manage their work. Asana simplifies team-based work management, providing a platform to manage your team's work, projects, & tasks online. There are lots of functions and routes that uses the Asana API, such as getting users by email, getting an Asana project by ID, marking tasks as complete, moving tasks to a section, adding subtasks, approving subtasks, updating tasks, getting stories for a task, getting tasks by section ID, fetching comments from a task, and fetching deliverables. The value of the [ASANA\\_PERSONAL\\_ACCESS\\_TOKEN in .env](#) is the token of the creator of this document. You have to replace this with your own.

### Google APIs

Google Cloud APIs are programmatic interfaces to Google Cloud Platform services. They are a key part of Google Cloud Platform, allowing you to easily add the power of everything from computing to networking to storage to machine-learning-based data analysis to your applications. Some of the services the project uses from Google APIs are Analytics Reporting API, Google Sheets API, and Search Console API.



---

## **HubSpot**

HubSpot is an American developer and marketer of software products for inbound marketing, sales, and customer service. It offers a full platform of marketing, sales, customer service, and CRM software, plus the methodology, resources, and support to help businesses. Some of the functions the project uses in their API are getting all closed won deals, getting a batch of companies by deals, and getting a batch of companies.

---

## SCOPE AND DELIMITATION

The scope of this system was to design and develop a RESTful API that utilizes all of the functions defined in the **server/services/** directory of the app. This includes returning responses properly when certain scenarios are met. For example, if a Goal Type Id is not found when making a request to the **/api/goalType/getById/{id}**, where **{id}** is an integer, it will return an error if the string is not a valid integer or if the ID is not found in the database.

The main focus of the system is to showcase best practices of using NodeJs whilst developing an API that is descriptive and helpful when making requests. For instance, the **response.data** object will contain 4 keys on protected routes: message (string), error (boolean), statusCode (integer), and data (array or object). For non-protected routes, where the user has to log in, the **response.data object** will contain an additional key of auth (boolean), which can be used to check if the user is authenticated or not.

The delimitation of this system is there are no test implementations placed in the system. Originally, the C# repository used the K6 test suite to develop the test on main routes, but this isn't implemented in the Node repository. However, when using the K6 test suite to test routes using the port of the Node repository from running a C# script, the tests passed on the main routes when it was last conducted on October 19, 2021.

The third-party API keys and credentials used in the project will be outdated if the accounts that possess the API keys are deleted. In such cases, routes and functions will not work, which could be troublesome since there are no backup API keys to use. For example, the AWS credentials are outdated, which made it impossible to implement an AWS Lambda job scheduling on the project. Cronjobs are currently implemented as routes found in the **server/routes/** directory. It wasn't implemented due to "unforeseen circumstances".

---

## SOFTWARE DEVELOPMENT TOOLS

The developer used the following tools in implementing the CXP-Node API project:

NodeJs 16.13.1. The main programming language used is NodeJs. It is used to write scripts and code that extend the capability of NodeJs onto the project. It is further supplemented by external NPM packages that make up the entire API.

pgAdmin 4. The database management system that's mainly used for Postgres, the world's most advanced Open Source database. It's designed to meet the needs of both novice and experienced Postgres users alike, providing a powerful graphical interface that simplifies the creation, maintenance and use of database objects.

PostgreSQL 12. The maintenance of the database, the information of users and merchants utilized MySQL, one of the most popular database management system languages for managing database content such as storing and retrieving data.

Postman 9.8.3. An application used for building, creating, and simulating API requests. The project must be locally started before sending a request.

Sequelize 6.6.5. A Node Package Manager (NPM) package that helps write functions that queries the database. Sequelize functions are mostly found under the **server/services/** directory of the project.

Visual Studio Code. The Integrated Development Environments (IDE) that the developer used to type and write codes for the project.

---

# PROJECT

## Downloads

The project uses the latest version of NodeJs. This is mandatory since there's one particular function in the codebase that's not supported for Node version < 15.0.0 which is the [replaceAll\(\)](#) function.

- ☐ Node: <https://nodejs.org/en/>
- ☐ pgAdmin 4: <https://www.pgadmin.org/download/>
- ☐ PostgreSQL 12: <https://www.postgresql.org/download/>
- ☐ Postman: <https://www.postman.com/>
- ☐ Visual Studio Code: <https://code.visualstudio.com/>

## Setup

After installation, go to PostgreSQL -> Login/Group Roles and create a new role. Use **victorious** as the name, and **enable all privileges** under the Privileges tab. Create a database called **victorious-node** under PostgreSQL 12 -> Databases.

Open Visual Studio Code and navigate to the directory where you want the project to be stored. Open the terminal and do a [git clone https://github.com/victoriousseo/CXP-Node-API.git](#). After cloning the repository, set up your [.env variables](#), run the [npm run migrate:seed](#) command, and finally run the app using the [npm run start](#) command and wait for a "Listening to port 3000" message to appear on the log.

## Disable logging

Since the project uses [Sequelize](#), the library logs every single query made to the database. You can temporarily disable this if you need to debug properly. In **server/config/database.js**, add a **logging: false** key-value pair to the **[env]** object then restart your server again.

## package.json

Run the [npm install](#) command on your terminal. This will install the dependencies located in the package.json file on the root of the project. As of

---

this writing, the project is using the following packages found on the table below:

<b>PACKAGE</b>	<b>VERSION</b>	<b>DESCRIPTION</b>
@hubspot/api-client	5.0.0	Interact with HubSpot API
asana	0.18.6	Interact with Asana API
aws-sdk	2.991.0	Interact with AWS API
bcrypt	5.0.1	For hashing and comparing passwords
bunyan	1.8.15	Colorful logging
bunyan-format	0.2.1	Formats bunyan logs
compression	1.7.4	Compresses HTTP responses, useful for testing caching
dotenv	10.0.0	For .env file
dtrace-provider	0.8.8	Native DTrace providers for Node.js apps
express	4.17.1	Web framework for servers
google-spreadsheet	3.1.15	Manipulation of Google Sheets
googleapis	89.0.0	Interact with Google API
helmet	4.6.0	Secures app by setting various HTTP headers
http-errors	1.8.0	Error handling of views
if-env	1.0.4	For npm run start to run effectively
jsonwebtoken	8.5.1	Handling of JWTs
multer	1.4.3	Handling of file uploads
nodemailer	6.6.5	For sending of mails
passport	0.4.1	Passport authentication
passport-jwt	4.0.0	Uses JWT Passport strategies
passport-local	1.0.0	Uses Passport local strategies

---

pg	8.7.1	Non-blocking PostgreSQL client for Node.js.
pg-hstore	2.3.4	For serializing and deserializing JSON data to hstore format
sequelize	6.6.5	To write queries
sequelize-cli	6.2.0	To use Sequelize command line interface
sequelize-date-no-tz-postgres	1.0.0	Support for no timezone of DATE datatype
uuid	8.3.2	To generate unique UUIDs
eslint	7.32.0	Linting of code, used for best practices
eslint-config-airbnb-base	14.2.1	Config used for ESLint
eslint-plugin-import	2.24.0	ESLint plugin support
mocha	9.0.3	For testing, although unused
nodemon	2.0.12	Restarts the app automatically on file change

## Testing the API

You may now test the routes at Postman but you need to login at the **api/user/authenticate** route to get a bearer token first. Copy the token from the value "Token" key under data. To pass the bearer token on the Headers, click on Params -> Authorization of a request. Under Type, you may use either "Bearer Token" or "Inherit auth from parent" (this only applies if your routes are placed under folders/directories, but is extremely useful for many routes). If you're using the former, simply paste the token under the Token field then send the request.

Take note that the password for the test user is stored in plaintext as there's a bug regarding hashing it manually on the seeders file.

Import the collection located at **server/lib/CXP-Node-API.postman\_collection.json** to Postman so you don't have to re-create all the API routes from scratch.

---

## Schema

The project consists of the following models as well as their fields and datatypes.

MODEL	FIELDS
ActionArea	Id (int), Name (varchar(50))
AsanaDeliverableType	Id (int), Name (text), Gid (text), Description (text), HelpfulMaterial (text), Note (text), DeliverableTypeId (int)
Comment	Id (uuid), Title (varchar(200)), Description (varchar(500)), Active (bool), IsDeleted (bool), CreatedAt (date), ModifiedAt (date), DeletedAt (date), CreatedById (uuid), ModifiedById (uuid), CommentTypeId (int), ProjectId (uuid), GoalId (uuid)
CommentTemplate	Id (uuid), Name (varchar(200)), Title (varchar(200)), Description (varchar(500)), CreatedAt (date), ModifiedAt (date), CreatedById (uuid), ModifiedById (uuid), TenantId (uuid)
CommentTemplateTemplateType	Id (uuid), CommentTemplateId (uuid), CommentTemplateTypeId (int)
CommentTemplateType	Id (int), Name (varchar(100))
CommentType	Id (int), Name (varchar(100)), Description (text), GoalTypeId (int)
Customer	Id (uuid), TenantId (uuid), Name (varchar(200)), RefName (varchar(200)), RefId (varchar(200)), CreatedAt (date), ModifiedAt (date)
Deliverable	Id (uuid), AsanaGid (text), Name (text), Host (text), PermanentUrl (text), ViewUrl (text), CreatedAt (date), CreatedById (uuid), TaskId (uuid), ProjectId (uuid), DeliverableTypeId (int)
DeliverableType	Id (int), Name (text)
DsType	Id (int), Name (varchar(50))

ExceptionLog	Id (int), RequestTime (date), RequestId (text), AppName (text), Exception (text)
Goal	Id (uuid), ValidFrom (date), ValidTo (date), Goal (int), InitialValue (int), EndValue (decimal), GoalReachedAt (date), PerformanceIncreasedCount (int), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid), ProjectId (uuid), GoalTypeId (int)
GoalType	Id (int), Name (varchar(100))
GoogleAnalyticPagePath	Id (uuid), ProjectId (uuid), GoogleAnalyticTypeId (int), Path (text), Created (date)
GoogleAnalyticType	Id (int), Name (varchar(100))
GoogleAnalyticValue	Id (uuid), TenantId (uuid), Data (JSONb), Created (date), Filter (text), GoogleAnalyticTypeId (int), ProjectId (uuid)
ModuleType	Id (int), Name (varchar(100))
Notification	Id (uuid), Subject (varchar(200)), From (varchar(100)), To (varchar(100)), Message (text), NotificationGuid (uuid), ValidTill (date), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid), TemplateTypeId (int), TemplateId (int), UserId (uuid)
NotificationApp	Id (uuid), Title (text), Description (text), Action (varchar(100)), CreatedAt (date), CreatedById(uuid), NotificationAppTypeId (int), TenantId (uuid), TaskId (uuid), CommentId (uuid), GoalId (uuid)
NotificationAppType	Id (int), Name (varchar(100))
NotificationAppUserRead	Id (uuid), UserId (uuid), CreatedAt (date), NotificationAppId (uuid)
Pod	Id (uuid), Active (bool), Name (varchar(100)), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid), TenantId (uuid)
PodProject	Id (uuid), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid), PodId (uuid), ProjectId (uuid)



PodUser	Id (uuid), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid), PodId (uuid), UserId (uuid)
Project	Id (uuid), Name (varchar(100)), AsanaProjectId (varchar(200)), AsanaRefName (varchar(200)), AsanaOwnerName (varchar(200)), AsanaTeamName (varchar(200)), AsanaCreatedAt (date), GoogleAnalyticsPropertyId (text), GoogleAnalyticsViewId (varchar(200)), Domain (text), GoogleDriveFolderId (text), GoogleDriveFolderName (text), ContractFileName (varchar(200)), ContractLocation (varchar(254)), LogoLocation (varchar(200)), AsanaSectionDone (varchar(200)), AsanaSectionRevision (varchar(200)), MaxBackfills (int), NumBackfills (int), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid), CustomerId (uuid), ProjectStatusId (int), TenantId (uuid), MainContactId (uuid)
ProjectStatus	Id (int), Name (varchar(200))
RequestLog	Id (int), RequestTime (date), RequestId (text), AppName (text), JsonRequest (JSONb)
Role	Id (int), RoleName (varchar(100)), UserTypeId (int)
TargetKeyword	Id (uuid), Name (text), TargetPageId (uuid), ProjectId (uuid)
TargetKeywordPositioning	Id (uuid), Date (date), Position (decimal), Delta (decimal), TargetKeywordId (uuid), ProjectId (uuid)
TargetPage	Id (uuid), Name (text), Url (text), ProjectId (uuid)
Task	Id (uuid), AsanaGid (text), Name (text), Description (text),
TaskComment	Id (uuid), AsanaGid (text), Text (text), CreatedAt (date), (date), CreatedById(uuid), TaskId (uuid)
Template	Id (int), Name (varchar(100)), Template (text), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid), TenantId (uuid), TemplateTypeId (int)

---

TemplateType	Id (int), Name (varchar(100))
Tenant	Id (uuid), Active (bool), Name (varchar(200)), Address (varchar(100)), Phone (varchar(50)), Email (varchar(100)), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid)
TenantModule	Id (uuid), TenantId (uuid), ModuleTypeId (int)
TenantSettings	Id (uuid), Name (text), Val1 (text), Val2 (text), TenantId (uuid)
TotalKeywordsRanked	Id (uuid), Date (date), Top3 (decimal), Top10 (decimal), Top100 (decimal), Over100 (decimal), TargetPageId (uuid), ProjectId (uuid)
User	Id (uuid), TenantId (uuid), FirstName (varchar(100)), LastName (varchar(100)), FirstLastName (varchar(200)), Title (varchar(100)), Email (varchar(100)), PasswordHash (varchar(100)), PasswordSalt (varchar(100)), FunFacts (text), IsPrimaryContact (bool), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid), CustomerId (uuid), DsTypeId (int), UserTypeId (int)
UserEmailSettings	Id (uuid), Value (bool), UserEmailSettingsTypeId (int), UserId (uuid)
UserEmailSettingsType	Id (int), Name (text)
UserPic	Id (uuid), Location (varchar(200)), IsNotifPic (bool), BackColor (varchar(30)), OrderNumber (int), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid), UserId (uuid)
UserRole	Id (uuid), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid), RoleId (int), UserId (uuid)
UserType	Id (int), Name (varchar (100), CreatedAt (date), ModifiedAt (date), CreatedById(uuid), ModifiedById (uuid)

---

---

## API DOCUMENTATION

All API routes are accessed with **localhost:3000:/api/** eg. **localhost:3000/api/user/authenticate**. Only required fields are listed under the Params / Body column. Routes starting with **tenant/** are not used in the C# repository but are implemented in the Node repository.

### GET Methods

ROUTE	PARAMS / BODY
/comment?projectId={projectId}&commentTypeId={commentTypeId}	projectId (uuid), commentTypeId (int)
/comment/goals?goalIds={goalIds}	goalIds (uuid, separated by comma for multiple Goal Ids)
/comment/home?projectId={projectId}	projectId (uuid)
/commentTemplate/{commentTemplateId}	commentTemplateId (uuid)
/commentTemplate/getAll	N/A
/commentType/getAll	N/A
/commentType/getById/{commentTypeId}	commentTypeId (int)
/customer?Search={search}&OrderByProperty={orderByProperty}&OrderByDirection={orderByDirection}&Take={take}&Skip={skip}	search (string), orderByProperty (string), orderByDirection (string), take (int), skip (int)
/customer/{customerId}	customerId (uuid)
/customer/hubspot/{searchString}	searchString (string)
/goal?projectId={projectId}	projectId (uuid)
/goalType/getAll	N/A
/goalType/getById/{goalTypeId}	goalTypeId (int)

---

/metrics/{projectId}/keywords	projectId (uuid)
/metrics/{projectId}/targetPages	projectId (uuid)
/pod/{podId}	podId (uuid)
/pod/getAll	N/A
/project?Search={search}&OrderBy={orderBy}&Direction={direction}&Take={take}&Skip={skip}	search (string), orderBy (string), direction (string), take (int), skip (int)
/project/{projectId}	projectId (uuid)
/project/{projectId}/comments?Search={search}&Active={active}&OrderBy={orderBy}&Direction={direction}&Take={take}&Skip={skip}	projectId (uuid), search (string), active (bool), orderBy (string), direction (string), take (int), skip (int)
/project/{projectId}/contract	projectId (uuid)
/project/{projectId}/deliverables	projectId (uuid)
/project/{projectId}/goals	projectId (uuid)
/project/{projectId}/goals/latest	projectId (uuid)
/project/asana/{asanaProjectGid}	asanaProjectGid (string)
/project/getForCustomer?customerId={customerId}	customerId (uuid)
/projectStatus	N/A
/role/getAll	N/A
/role/getById/{roleId}	roleId (int)
/task?projectId={projectId}	projectId (uuid)
/task/{taskId}/assistance	taskId (uuid)
/tenant/getAll	N/A
/tenant/getById/{tenantId}	tenantId (uuid)

---

/tenant/getPaged?Page={page}&Take={take}	page (int), take (int)
/tenant/toggleActive/{tenantId}	tenantId (uuid)
/user?Take={take}&Skip={skip}	take (int), skip (int)
/user/{userId}	userId (uuid)
/user/{userId}/pictures	userId (uuid)
/user/customer/{customerId}?Take={take}&Skip={skip}	customerId (uuid), take (int), skip (int)
/user/getUserByNotificationGuid?guid={notificationGuid}	notificationGuid (uuid)
/user/victorious?Take={take}&Skip={skip}	take (int), skip (int)
/user/withoutPod?Take={take}&Skip={skip}	take (int), skip (int)
/userType/getAll	N/A

## POST Methods

ROUTE	PARAMS / BODY
/comment	Title (string), Description (string), GoalId (uuid), ProjectId (uuid), Active (bool), CommentTypeId (int)
/commentTemplate/create	Name (string), Title (string), Description (string), Type (array of int)
/commentTemplate/getPaged	Types (array of int), Take (int), Skip (int)
/goal	ValidFrom (date), ValidTo (date), Goal (int), InitialValue (int), EndValue (int), PerformanceIncreasedCount (int), ProjectId (uuid), GoalTypeId (int), UserId (uuid)

---

/goal/eCommerceMetric	ProjectId (uuid), From (date), To (date)
/goal/googleConnectionCheck	webPropertyId (string), viewId (string), domain (string)
/goal/organicSearch	ProjectId (uuid), From (date), To (date)
/goal/pagePath	ProjectId (uuid), GoogleAnalyticTypeId (int)
/metrics/targetKeywordPositioning	ProjectId (uuid), From (date), To (date)
/metrics/totalKeywordsRanked	ProjectId (uuid), From (date), To (date)
/notificationApp/create	NotificationAppTypeId (int)
/notificationApp/getPaged	Types (array of int), ProjectId (uuid), Take (int), Skip (int)
/notificationApp/notificationReadSet	UserId (uuid), NotificationAppId (uuid)
/pod/create	TenantId (uuid), Name (string), Active (bool)
/project	HubSpotCustomerId (string), HubSpotCustomerName (string), CustomerName (string), ProjectName (string), Logo (file upload)

---

/tenant/create	Active (bool), Name (string)
/user/{userId}/picture	userId (uuid), BackColor (string), OrderNumber (string), File (file upload)
/user/{userId}/resendInvitation	userId (uuid), NotificationId (uuid), DsTypeId (int)
/user/authenticate	email (string), password (string)
/user/inviteCustomer	FirstName (string), LastName (string), Email (string), RoleId (int), CustomerId (uuid)
/user/inviteVictorious	FirstName (string), LastName (string), Email (string), RoleId (int)

## PUT Methods

ROUTE	PARAMS / BODY
/comment/{commentId}	commentId (uuid), Title (string), ProjectId (uuid), Active (bool), CommentTypeId (int)
/commentTemplate/update	Id (uuid), Name (string), Title (string), Description (string), Type (array of int)
/goal/{goalId}	goalId (uuid), Id (uuid), ValidFrom (date), ValidTo (date), ProjectId (uuid), GoalTypeId (int)
/notificationApp/update	Id (uuid), NotificationAppTypeId (int)
/pod/project	PodId (uuid), ProjectId (uuid), Remove (bool)
/pod/update	Id (uuid), Name (string), Active (bool)
/pod/user	PodId (uuid), UserId (uuid), Remove

	(bool)
/project/{projectId}	projectId (uuid), ProjectName (string), Logo (file upload), deleteLogo (bool)
/project/{projectId}/asana	projectId (uuid), AsanaProjectId (string), AsanaRefName (string), AsanaOwnerName (string), AsanaTeamName (string), AsanaCreatedAt (date)
/project/{projectId}/contract	projectId (uuid), File (file upload)
/project/{projectId}/ga	projectId (uuid), Domain (string), GoogleAnalyticsPropertyId (string), GoogleAnalyticsViewId (string), GoogleDriveFolderId (string)
/project/{projectId}/gDrive?folder={googleDriveFolderId}	projectId (uuid), googleDriveFolderId (string)
/project/{projectId}/logo	projectId (uuid), Logo (file upload)
/project/{projectId}/podAndContact	projectId (uuid), PodId (uuid), MainContactId (uuid), Contract (file upload)
/project/ga/validate?property={googleAnalyticsPropertyId}&view={googleAnalyticsViewId}	googleAnalyticsPropertyId (string), googleAnalyticsViewId (string)
/task/{taskId}/approve	taskId (uuid)
/task/{taskId}/complete	taskId (uuid)
/task/{taskId}/feedback	taskId (uuid)
/tenant/update	Id (uuid)
/user/{userId}	Id (uuid), FirstName (string), LastName (string), Email (string), TenantId (uuid), RoleId (int), DsTypeId (int), UserTypeId (int)
/user/{userId}/pictures/{userPicId}	userId (uuid), userPicId (uuid), File



---

	(file upload)
/user/{userId}/pictures/{userPicId}/color	userId (uuid), userPicId (uuid), BackColor (string)
/user/changePassword	CurrentPassword (string), NewPassword (string), UserId (uuid)
/user/emailSettings	Type (int), Value (bool)
/user/resetPassword	Email (string)
/user/setupPassword	NotificationGuid (uuid), Password (string), KeepLoggedIn (bool)

## DELETE Methods

ROUTE	PARAMS / BODY
/comment/{commentId}	commentId (uuid)
/commentTemplate/{commentTemplateId}	commentTemplateId (uuid)
/goal/{goalId}	goalId (uuid)
/notificationApp/{notificationAppId}	notificationAppId (uuid)
/pod/{podId}	podId (uuid)
/project/{projectId}	projectId (uuid)
/project/{projectId}/contract	projectId (uuid)
/project/{projectId}/logo	projectId (uuid)
/tenant/delete/{tenantId}	tenantId (uuid)
/user/{userId}	userId (uuid)
/user/{userId}/pictures/{userPicId}	userId (uuid), userPicId (uuid)

---

## REFERENCES

<https://jwt.io/introduction>

<https://swagger.io/docs/specification/authentication/bearer-authentication/>

<https://www.npmjs.com/package/jsonwebtoken>

<https://www.redhat.com/en/topics/api/what-is-a-rest-api>

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/String/replaceAll](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replaceAll)

<https://stackoverflow.com/questions/70080928/replaceall-function-doesnt-work-in-node-js>

# APPENDICES

## Appendix A

### Screenshots

pgAdmin

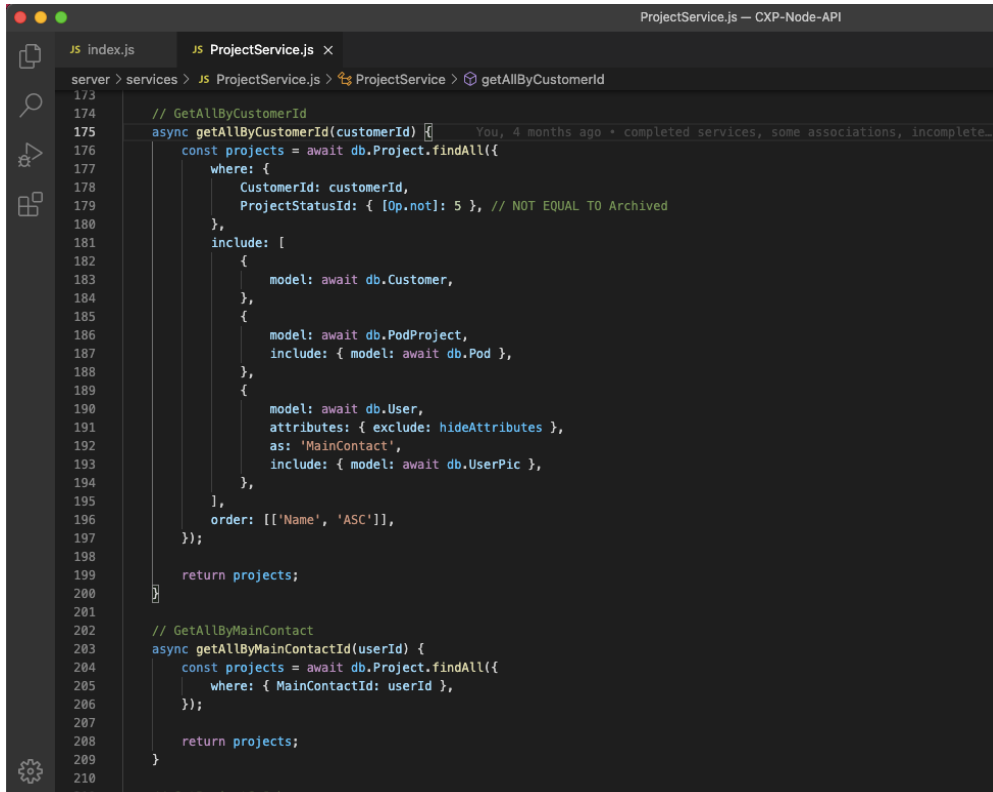
The screenshot displays the pgAdmin 4 web interface. On the left, a 'Browser' pane shows a tree of database objects, with 'Project' selected. The main area is divided into a 'Query Editor' and a 'Data Output' section. The 'Query Editor' contains the following SQL query:

```
1 SELECT * FROM public."Project"  
2 ORDER BY "Id" ASC
```

The 'Data Output' section shows the results of the query in a table format. The table has 8 columns: Id, Name, AsanaProjectId, AsanaRefName, AsanaOwnerName, AsanaTeamName, and AsanaCreateTimestamp. The first row of data is as follows:

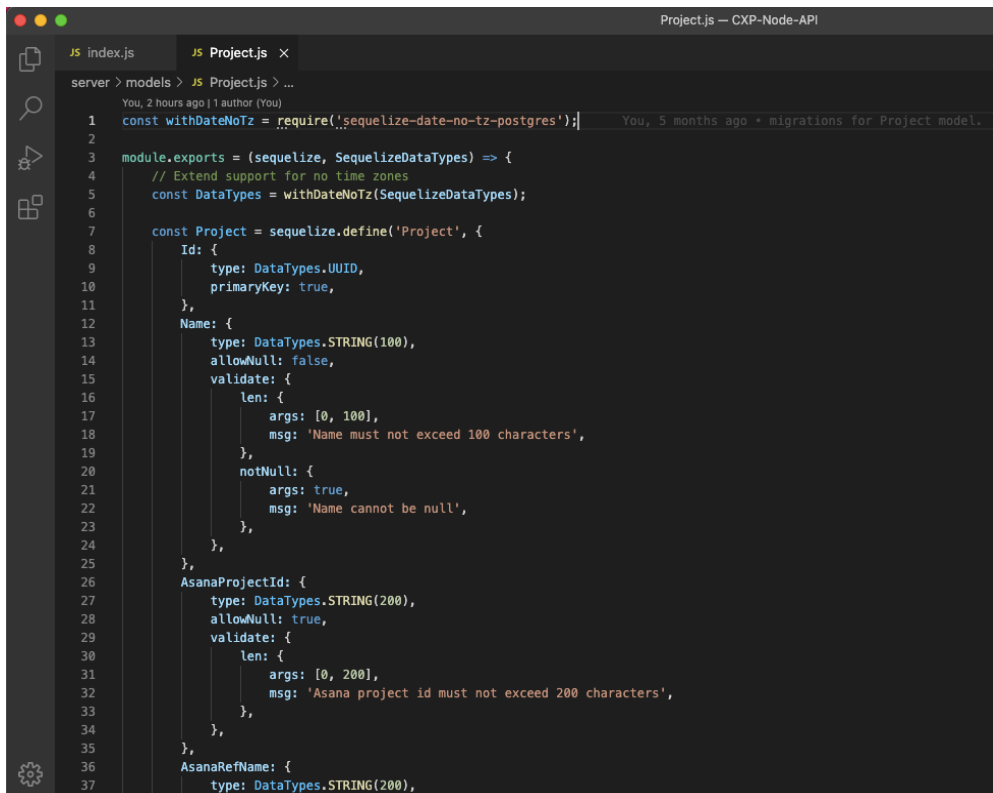
Id	Name	AsanaProjectId	AsanaRefName	AsanaOwnerName	AsanaTeamName	AsanaCreateTimestamp
d3ec8c1d-81cd-4918-be55-1e3d4a895d2a	project name	1200167151046149	[null]	[null]	[null]	[null]

## Sample Sequelize function



```
ProjectService.js — CXP-Node-API
server > services > JS ProjectService.js > ProjectService > getAllByCustomerId
173
174 // GetAllByCustomerId
175 async getAllByCustomerId(customerId) {
176   const projects = await db.Project.findAll({
177     where: {
178       CustomerId: customerId,
179       ProjectStatusId: { [Op.not]: 5 }, // NOT EQUAL TO Archived
180     },
181     include: [
182       {
183         model: await db.Customer,
184       },
185       {
186         model: await db.PodProject,
187         include: { model: await db.Pod },
188       },
189       {
190         model: await db.User,
191         attributes: { exclude: 'hideAttributes' },
192         as: 'MainContact',
193         include: { model: await db.UserPic },
194       },
195     ],
196     order: [['Name', 'ASC']],
197   });
198   return projects;
199 }
200
201
202 // GetAllByMainContact
203 async getAllByMainContactId(userId) {
204   const projects = await db.Project.findAll({
205     where: { MainContactId: userId },
206   });
207   return projects;
208 }
209
210
211 // GetProjectInfo
```

## Sample Sequelize model definition



```
Project.js — CXP-Node-API
server > models > JS Project.js > ...
You, 2 hours ago | 1 author (You)
1 const withDateNotZ = require('sequelize-date-no-tz-postgres');
2
3 module.exports = (sequelize, SequelizeDataTypes) => {
4   // Extend support for no time zones
5   const DataTypes = withDateNotZ(SequelizeDataTypes);
6
7   const Project = sequelize.define('Project', {
8     Id: {
9       type: DataTypes.UUID,
10      primaryKey: true,
11    },
12    Name: {
13      type: DataTypes.STRING(100),
14      allowNull: false,
15      validate: {
16        len: {
17          args: [0, 100],
18          msg: 'Name must not exceed 100 characters',
19        },
20        notNull: {
21          args: true,
22          msg: 'Name cannot be null',
23        },
24      },
25    },
26    AsanaProjectId: {
27      type: DataTypes.STRING(200),
28      allowNull: true,
29      validate: {
30        len: {
31          args: [0, 200],
32          msg: 'Asana project id must not exceed 200 characters',
33        },
34      },
35    },
36    AsanaRefName: {
37      type: DataTypes.STRING(200),
```

## Sample Sequelize migration

```
20210821050210-Project.js — CXP-Node-API
JS index.js JS 20210821050210-Project.js x
server > migrations > JS 20210821050210-Project.js > ...
You, 2 hours ago | 2 authors (You and others)
You, 5 months ago • migrations for Project model.
1 const withDateNoTz = require('sequelize-date-no-tz-postgres');
2
3 module.exports = {
4   up: async (queryInterface, Sequelize) => {
5     // Extend support for no time zones
6     const DataTypes = withDateNoTz(Sequelize);
7
8     await queryInterface.createTable('Project', {
9       Id: {
10         type: DataTypes.UUID,
11         primaryKey: true,
12       },
13       Name: {
14         type: DataTypes.STRING(100),
15         allowNull: false,
16       },
17       AsanaProjectId: {
18         type: DataTypes.STRING(200),
19         allowNull: true,
20       },
21       AsanaRefName: {
22         type: DataTypes.STRING(200),
23         allowNull: true,
24       },
25       AsanaOwnerName: {
26         type: DataTypes.STRING(200),
27         allowNull: true,
28       },
29       AsanaTeamName: {
30         type: DataTypes.STRING(200),
31         allowNull: true,
32       },
33       AsanaCreatedAt: {
34         type: DataTypes.DATE_NO_TZ,
35         allowNull: true,
36       },
37       GoogleAnalyticsPropertyId: {
```

## Sample routes

```
index.js — CXP-Node-API
JS index.js x
server > routes > projects > JS index.js > <unknown> > exports > router.get('/') callback
You, 3 months ago • add pagination functions and refactor
41 router.get('/', async (request, response, next) => {
42   try {
43     const {
44       Search, OrderBy, Direction, Take, Skip, Active,
45     } = request.query;
46     const paginationParams = {
47       Search, OrderBy, Direction, Take, Skip, Active,
48     };
49
50     const customerId = await userResolver.getCustomerId();
51     const podId = await userResolver.getPodId();
52     const isVictoriousAdmin = await userResolver.isVictoriousAdmin();
53     const allProjects = await projects.getPaged(
54       paginationParams, isVictoriousAdmin, customerId, podId,
55     );
56
57     return response.json(api.success(allProjects));
58   } catch (event) {
59     return next(event);
60   }
61 });
62
63 router.post('/', upload.single('Logo'), async (request, response, next) => {
64   let customer;
65
66   try {
67     if (request.body.CustomerId) {
68       customer = await customers.getById(request.body.CustomerId);
69     } else { // Create new customer if CustomerId is not passed
70       const hubspotCustomer = {
71         Name: request.body.CustomerName,
72         RefName: request.body.HubSpotCustomerName,
73         RefId: request.body.HubSpotCustomerId,
74         TenantId: await userResolver.getTenantId(),
75       };
76       customer = await customers.createCustomer(hubspotCustomer);
77     }
78     customer.Name = request.body.CustomerName;
```

---

## Appendix B

### **.env variables**

# Replace these values depending on your pgAdmin settings

HOST\_NAME\_DEV=

PORT\_DEV=

USERNAME\_DEV=

PASSWORD\_DEV=

DB\_DSN\_DEV=

# Victorious

VICTORIOUS\_EMAIL=

VICTORIOUS\_PASSWORD=

# -- Used in TenantSettings seeder

SMTP\_USERNAME=

SMTP\_PASSWORD=

SMTP\_HOST=

SMTP\_PORT=

# Passport

JWT\_SECRET\_OR\_KEY=

# Asana

ASANA\_PERSONAL\_ACCESS\_TOKEN=

# Google

GOOGLE\_SERVICE\_ACCOUNT\_EMAIL=

GOOGLE\_PRIVATE\_KEY=

GOOGLE\_IMPERSONATED\_ACCOUNT=

GOOGLE\_CUSTOMER\_FOLDER\_LOCATION=

GOOGLE\_SAMPLE\_SPREADSHEET\_ID=

# – Google Search Console

GSC\_SERVICE\_ACCOUNT\_EMAIL=

GSC\_PRIVATE\_KEY=

# -- Used in Project seeder

GA\_VIEW\_ID=

GA\_PROPERTY\_ID=

---

GOOGLE\_DRIVE\_FOLDER\_ID=

# Hubspot

HUBSPOT\_API\_KEY=

### Terminal commands

- `npm run start` - starts the app at localhost:3000 unless a variable called **PORT** is set on .env file
- `npm run lint` - performs linting and checks for errors and warnings on code based on rules defined by .eslintrc.js
- `npm run migrate` - performs migrations on unmigrated files
- `npm run migrate:seed` - runs all migrations and seeders
- `npm run migrate:undo` - undos all migrations and seeders
- `npm run migrate:prod` - performs migrations on production environment
- `npm run docker:build` - creates a container using Docker
- `npm run docker:run` - creates and runs a container on port 3000 and removes it when the program is exited
- `npm run docker:stop` - stops running container
- `npm run docker:push` - shares your image to the Docker Hub
- `npm run test` - runs a test on the project using the **mocha** package, although not implemented