Management Documentation

RESTFUL API

November 24, 2018

Qianyi Lu (ID: 1698774)
Peng Xu (ID: 1709403)
University of Adelaide

Contents

1 Introduction **4**
- 1.1 Scope . . . 4
- 1.2 Version . . . 4
- 1.3 Team Member . . . 4
- 1.4 Intended Audience . . . 4

2 End Points **5**
- 2.1 summary . . . 5
- 2.2 POST /task . . . 6
- 2.3 PATCH /task/:id . . . 7
- 2.4 GET /task/:id . . . 8
- 2.5 DELETE /task/:id . . . 9
- 2.6 GET /allocator . . . 10
- 2.7 POST /allocator . . . 11
- 2.8 POST /submission . . . 12
- 2.9 GET /submission/:id . . . 13

3 System Set Up **14**
- 3.1 Step 1 : Install required packages . . . 14
- 3.2 Step 2 : Reset envirnment variables . . . 15
- 3.3 Step 3 : Add app.js file . . . 15
- 3.4 Step 4 : Add .babelrc file . . . 15
- 3.5 Step 5 : Rewrite package.json . . . 15
- 3.6 Step 6 : Ensure your database is already started . . . 16
- 3.7 Step 7 : Run the system . . . 16

4 Related Files **17**

5 Database's design **18**
- 5.1 Set up tables in database . . . 18
- 5.2 Build up database(related file) . . . 19

6 Secure the database **20**

7 Test cases for our application **21**

Version History

Date	Version	Changer	Description
2018/10/08	1.0	Qianyi Lu	Document creation
2018/10/12	1.1	Qianyi Lu	Add introduction, end points
2018/11/12	1.2	Qianyi Lu	End points redesign
2018/11/19	1.3	Qianyi Lu	Code version change
2018/11/20	1.4	Qianyi Lu	Add system set up
2018/11/20	2.0	Qianyi Lu	Documentation completed
2018/10/08	3.0	Peng Xu	Relationship between tables design
2018/10/11	3.1	Peng Xu	Build up Database
2018/10/22	3.2	Peng Xu	Modify the database
2018/11/15-16	3.3	Peng Xu	Safety part for database
2018/11/19	3.4	Peng Xu	Set up test cases
2018/11/21-22	4.0	Peng Xu	Working in Documentation

1 Introduction

1.1 Scope

This project's purpose is to implement a RESTful API for processing tasks/jobs. The system will allow internal clients to create, update and delete tasks and receive task results. It will allow external to apply for tasks and submit task results.

1.2 Version

Currently this system is version 1.0 If new resources are added but the system itself does not change, it won't be count as a new version. eg. new task type is supported.

1.3 Team Member

The team contains two members. Qianyi Lu designed the api system, use cases and database structure, built up api functions and combined the system with database. Peng Xu figured out the specific tables and built up database tables, safety part of database and test cases.

1.4 Intended Audience

The intended audience of this project is Frisk who is the host company.

2 End Points

2.1 summary

Path	Method	Description
/api/v1/task	POST	Allow internal to create a task
/api/v1/task/{task id}	PATCH	Allow internal to update a task
/api/v1/task/{task id}	GET	Allow internal to check a task's detail
/api/v1/task/{task id}	DELETE	Allow internal to delete a task
/api/v1/allocator	GET	Allow external to get a assignment
/api/v1/allocator	POST	Allow external to accept a assignment
/api/v1/submission	POST	Allow external to submit a assignment
/api/v1/submission{task id}	GET	Allow internal to get a result of a task

2.2 POST /task

This request can only be used by internal clients to create new tasks. The created task type depends on client's id. If an client can create multiple type of task, it should choose the correct client id. If time_limit is not provided, system will use default value of related task type. time_limit should be provided in the form {number unit} eg. 1 hour, 30 mins.
For responses, a message will be sent back including err detail or success. If the task is created successfully, task detail will be provided in response including task_id, file, time_limit and description.

Parameters

Name	Location	Required	Type
client_id	header	yes	integer
file	body	yes	all types
type_id	body	yes	integer
time_limit	body	no	interval
description	body	no	string (up to 255 char)

Responses

Code	Description	Return Value
200	Creation succeed, no matter not required fields are provided or not	task_id, file, time_limit, description
403	Client is not a valid internal	
405	Some of the provided parameters are in wrong type. Or required parameters are not provided.	
503	Database system is crashed or connection pool is full	

2.3 PATCH /task/:id

This request can only be used by internal clients to update task detail file or time limit or description. At least one of them should be provided with a new value, otherwise the request will be rejected as no update is made. When a update is made, current assignment will be cancelled. Taks will be reprocessed even the update only occured on time limit. time_limit should be provided in the form {number unit} eg. 1 hour, 30 mins.
For responses, a message will be sent back including err detail or success. If the task is created successfully, task detail will be provided in response including task_id, updated file, updated time_limit and updated description.

Parameters

Name	Location	Required	Type
client_id	header	yes	integer
file	body	no	all types
type_id	body	no	integer
time_limit	body	no	interval
description	body	no	string (up to 255 char)

Responses

Code	Description	Return Value
200	Update succeed.	task_id, file, time_limit, description
403	Client is not a valid internal or it's not the creater of this task	
405	Some of the provided parameters are in wrong type. Or required parameters are not provided.	
503	Database system is crashed or connection pool is full	

2.4 GET /task/:id

This request can only be used by internal clients to get a task's detail file or time limit or description. And this client must be the creater of this task.
For responses, a message will be sent back including err detail or success. If the task detail is sent successfully, task detail will be provided in response including task_id, file, time_limit and description.

Parameters

Name	Location	Required	Type
client_id	header	yes	integer

Responses

Code	Description	Return Value
200	Task detail is sent.	task_id, file, time_limit, description
403	Client is not a valid internal or it's not the creater of this task	
405	Client id is in wrong type or not provided.	
503	Database system is crashed or connection pool is full	

2.5 DELETE /task/:id

This request can only be used by internal clients to delete a task.
For responses, a message will be sent back including err detail or success.

Parameters

Name	Location	Required	Type
client_id	header	yes	integer

Responses

Code	Description
200	deleted succeed.
403	Client is not a valid internal or it's not the creater of this task
405	Client id is in wrong type or not provided.
503	Database system is crashed or connection pool is full

2.6 GET /allocator

This request can only be used by exernal clients to ask for a new preassignment. The assigned task type depends on client's id. If an client can deal with multiple type of task, it should send several requests with different client ids. Maximum waiting time is 30 seconds(can be change), if time out, the prassignment will be cancelled.
For responses, a message will be sent back including err detail or success. If the task is preassigned successfully, task detail will be provided in response including assignment_id, file, time_limit and description. If the client already has a task, task detail will be resent. Notice the returning assignment_id is only an id help clients recording their work, no request require clients provide this id.

Parameters

Name	Location	Required	Type
client_id	header	yes	integer

Responses

Code	Description	Return Value
200	Preassign succeed or task detail resent	assignment_id, file, time_limit, description
403	Client is not a valid external	
405	Some of the provided parameters are in wrong type. Or required parameters are not provided.	
503	Database system is crashed or connection pool is full	

2.7 POST /allocator

This request can only be used by external clients to accept or refuse its preassignment. The assignment's id depends on client's id. When the assignment is accepted, the real time limit will start to count.
For responses, a message will be sent back including err detail or success.

Parameters

Name	Location	Required	Type
client_id	header	yes	integer
acception	body	yes	Boolean or 'true'(string) or 'false'(string)

Responses

Code	Description
200	Accepted or refused.
403	Client is not a valid external.
405	Some of the provided parameters are in wrong type. Or required parameters are not provided.
503	Database system is crashed or connection pool is full

2.8 POST /submission

This request can only be used by external clients to submit its assignment's result. The assignment's id depends on client's id. When a commission field is provided and its value is true, the assignment will be closed. If commission field is not provided, client can resend results for many times until commission is provided or time limit meets. In this situation it will still be treated as a completed assignment but client cannot get a new assignment before committing the submission.
For responses, a message will be sent back including err detail or success.

Parameters

Name	Location	Required	Type
client_id	header	yes	integer
file	body	no	all types
commission	body	no	Boolean or 'true'(string) or 'false'(string)
description	body	no	string (up to 255 char)

Responses

Code	Description
200	Accepted or refused.
403	Client is not a valid external.
405	Some of the provided parameters are in wrong type. Or required parameters are not provided.
503	Database system is crashed or connection pool is full

2.9 GET /submission/:id

This request can only be used by internal clients to get a task's result. It must be the creator of the task. Get the result won't close the task, so client can ask for result many times. If a client think the task result is not satisfied, it could sent a request to reset the task (eg. change description) A new result will be processed. If a client think the result is satisfied, it should send another request to delete the task.
For responses, a message will be sent back including err detail or success. Processed file will be sent if the task finished processing.

Parameters

Name	Location	Required	Type
client_id	header	yes	integer

Responses

Code	Description	Return Value
200	Task detail is sent.	task_id, file, description
403	Client is not a valid internal or it's not the creater of this task	
405	Client id is in wrong type or not provided.	
503	Database system is crashed or connection pool is full	

3 System Set Up

3.1 Step 1 : Install required packages

Please run following commands in command.txt file under your system root in order.

npm init

This is used to instail a package manager for JavaScript

npm install express

This is used to instail the express structure manager

npm install pg-promise

This is used to instail the pg-promise for connecting with postgreSQL database

npm install babel-cli
npm install babel-preset-es2015

This is used to instail the promise manager. Currently the system is written by SE6 promise but express only support to SE5. So use babel to change the promise to SE5.

npm install nodemon

This is used to instail the nodemon which can automatically restart system

npm install body-parser

This is used to instail the parser to handle the request body before come into processing

npm install later

This is used to instail the later to build up timer function

npm install dotenv

This is used to instail the dotenv to read environment variables from .env file

3.2 Step 2 : Reset envirnment variables

Please copy and paste the .env file in to your root. Then change the variables in it to fix your own system. If you already have the same name variables locally, please set the local stored variables too.

- REST_HOST should be the host address your system run on.
- REST_USER should be the user you used to connect to the system.
- REST_PASSWORD should be the password of your user.
- REST_SERVER should be the your server name of database.
- REST_DATABASE restfuldb should be the name of database.
- REST_DBPORT restfuldb should be the port of database.
- REST_DBMAX restfuldb should be the maximum connection limit to visit database in the same time.
- REST_DBTIME should be the maximum connection time limit to visit database. The unit is 0.001 second.
- REST_PORT restfuldb should be the port of API.
- REST_TIMERATE should be the time rate of running auto check timer function which will cancel the time out assignments. The written form is every {number} {time unit}
- REST_ACCEPTLIM should be the maximum time limit for accetping a preassignment.

3.3 Step 3 : Add app.js file

Please copy and paste app.js file into your root.

3.4 Step 4 : Add .babelrc file

Please copy and paste .babelrc file into your root. If your are using windows system please also copy and paste babel.cmd file to root.

3.5 Step 5 : Rewrite package.json

Please add the following line in the"scripts": {} part under your root. package.json file should already be there created by your ***npm install express*** command. Remember to add comma.

"start": "nodemon app.js –exec babel-node –"

3.6 Step 6 : Ensure your database is already started

This system is not going to start your postgreSQL database system or creating tables, so you should make sure you have already started the server.

3.7 Step 7 : Run the system

Please use following command to run the system. All log will be displayed in console, or you can output it to other file by add command line.

npm run start

4 Related Files

1. app.js
2. command.txt
3. .babelrc
4. .env
5. babel.cmd
6. package.json (Just for cooperation with your own file, don't copy it.)

5 Database's design

The following is the process of the illusion of our early design. Most of them has been came true in our database

- Create the task which will be recorded in our task table.
- Check whether the task information what we needs in our task table.
- Rapidly find out the task id according to the request from client side and searching the file whether stored in our task table, if not, insert the file from client side to it. The file would be ignored when there has a file with id in task table.
- Suppose we have several id number for a file. When those file has same id, it will be appeared in the meantime if the database receive the request.
- For the last process, we are concerning about the circle of the each response from database to client. Suppose we have three clients(client1, client2 and client3). If any situation influence on client1, which was lead to the client 1 can not receive the response on time. The client2 will receive this response if it matched. So the next chance for client1 to receive is what we caring in.

5.1 Set up tables in database

logtype

All tasks are updated to this table after created. After a client sending a request, check whether there is a waiting task to assign. If there is, link that request with this table.

permission

Allow different levels of visitors to do different tasks, their work is limited to their own permissions.

state

Determine the current state of the visitor, and assigning a task to him through the state.

type

Reading three kinds of files which distinguished by their own type and we set up the time limit for each process.

client

The table will record the each current client when they accessed and represent them and set the type of file.

task

The task table is most base one which provide the each client matched their own task and set the limitation in time.

assignment

Each assignment was inherited from task and it will be handled in here.

response

It get used to client id and task id to handle the response.

Related file:

- Specific Table.pdf
- Table's Relationship.doc

5.2 Build up database(related file)

Our database based on postgres sql (file: Database.sql)

6 Secure the database

Establishing a Username and Password provides security for the database. The contents of the database must be changed before the developer logs in. We set up the user and password through terminal.

LOGIN means creator can log in our database and CREATEDB represent it has a right to build up the database.

- CREATE ROLE role1 WITH PASSWORD '*****' LOGIN CREATEDB;

7 Test cases for our application

At present, we have worked out seven large-scale running tests, each of which focuses on checking different methods(get, post, patch and delete), especially testing client identity, which is involved in every project to ensure that clients with different permissions to do it, which leads to no conflict in our application.

- Test for create task(included 12 requests)
- Test for external to accept task(included 9 requests)
- Test for external to get a task(included 8 requests)
- Test for external submit a task(included 9 requests)
- Test for internal get a result of task(included 8 requests)
- Test for internal update a task(included 8 requests)
- Test for internal to delete task(included 7 requests)

Related file:
Test cases.pdf(our test cases was built up based on postman)