

# Employee Management System

Yu Zhao, Xiaojing Wang, Sujeeth Nilakantan

GitHub: <https://github.com/zhaoyufrank/CloudComputing-Project>

## Project Introduction

This is an employee management project. The home page displays all employees, the next web page shows the detailed information of an employee. The user can create a new employee, update an employee's information and delete the employee from the system. The employee information includes name, gender, email, birthday, work department, start date, position, and so on.

We used React.js to build the front end, and Node.js and Cloud Function to build the backend. The project is built using Google Cloud components, such as Compute Engine, VM Instance, Firebase, Firestore and Cloud Functions.

## Service/Application Overview

- Describes how the user(s) will interact with the service
  1. If the user (admin) clicks the create button on the home page; the user is redirected to the 'create employee' page. The user inputs the employee related information and clicks on the save button; the input details are saved and stored in the Firestore database.
  2. If the user clicks on the edit button on the home page; the user is re-directed to the 'edit employee' page. The user can modify the existing employee details and click on the save button; the input details are saved and updated in the Firestore database.
  3. If the user clicks on an employee, the user will be re-directed to a new page which will display all the information of that employee.
  4. If the user clicks on the delete button, the application deletes the employee from the front end and the Firestore database

- What does the service enable?

The project can create, update, delete and display the details of employees

## Components Used

- Which components will you be using?

JavaScript, Node.js, React.js, Cloud Functions, Compute Engine, VM Instance, Cloud Firestore database

- What is the purpose of each one?

Programming language: JavaScript

Frontend: React.js

Backend: Node.js, Express, Cloud Functions

Database: Firestore

Compute service: Compute Engine

## Architecture

- How is the data flowing through the service?

The user enters the data (employee details) on the web page.

The front-end code (React.js) deployed in the VM instance on Google Compute Engine passes the data from the web page to the cloud function.

The cloud function in firebase contains the back-end code (Node.js), it writes the data to the employee database in Firestore.

- How will components connect to each other?

The front-end code (React.js) deployed in the VM instance on Google Compute Engine captures the data from the web page and passes it to the cloud function.

The cloud function in firebase contains the back-end code (Node.js), it writes the data to the employee database in Firestore.

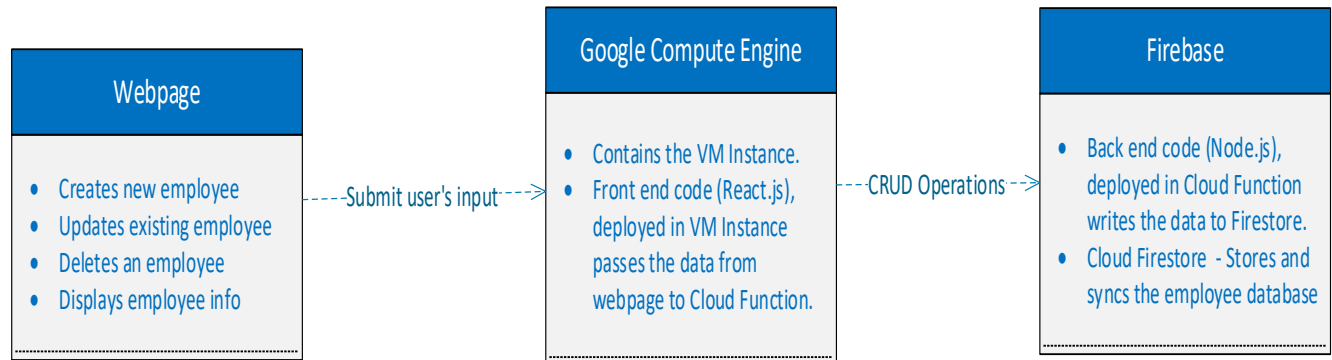
## Design

- What language, game framework, etc. are you using?

React.js, Node.js, JavaScript.

- What are the major code processes/workflows that will control functionality? And a description of major objects in the design

For the major code processes/workflows and description of major objects, I drew the following UML to clarify them.



## Implementation Plan

- What parts of the application and/or design do you plan to implement in what order?

The order in which we plan to implement the components of the application are

1. Compute service – Google Compute Engine
2. VM Instance
3. A web page – Front end
4. A serverless execution environment - Cloud Functions
5. A cloud database - Firestore

- Who is responsible for which parts of implementation?

Yu - Front-end and Back-end code

Xiaojing - Google Compute Engine, Code Deployment

Sujeeth - Firestore, Cloud Functions

- How will you know you are on schedule for finishing and/or a planned milestones timeline?

No.	Task	Due Date
1	Set up the cloud environment	10/16/2021
2	Home page	10/20/2021
3	Create employee page	10/28/2021
4	Mid-point demo	10/28/2021
5	Update employee page	12/02/2021
6	Delete employee function	12/02/2021
7	Test the web service	12/02/2021
8	Final presentation	12/02/2021

## Test Plan

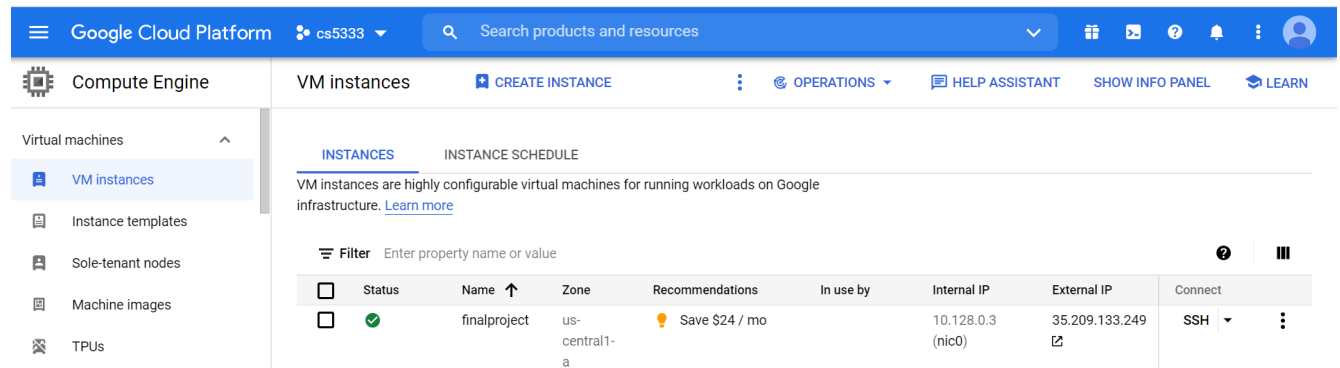
- How will you verify that the service game is working correctly?

The user will create/update/delete new employee details on the webpage. Upon submission, if the input details are created/updated/deleted in the Firestore database accurately and the new employee information is displayed correctly on the webpage, then we can conclude that the web service is working correctly.

## Demo:

Our source code is in GitHub: <https://github.com/zhaoyufrank/CloudComputing-Project>

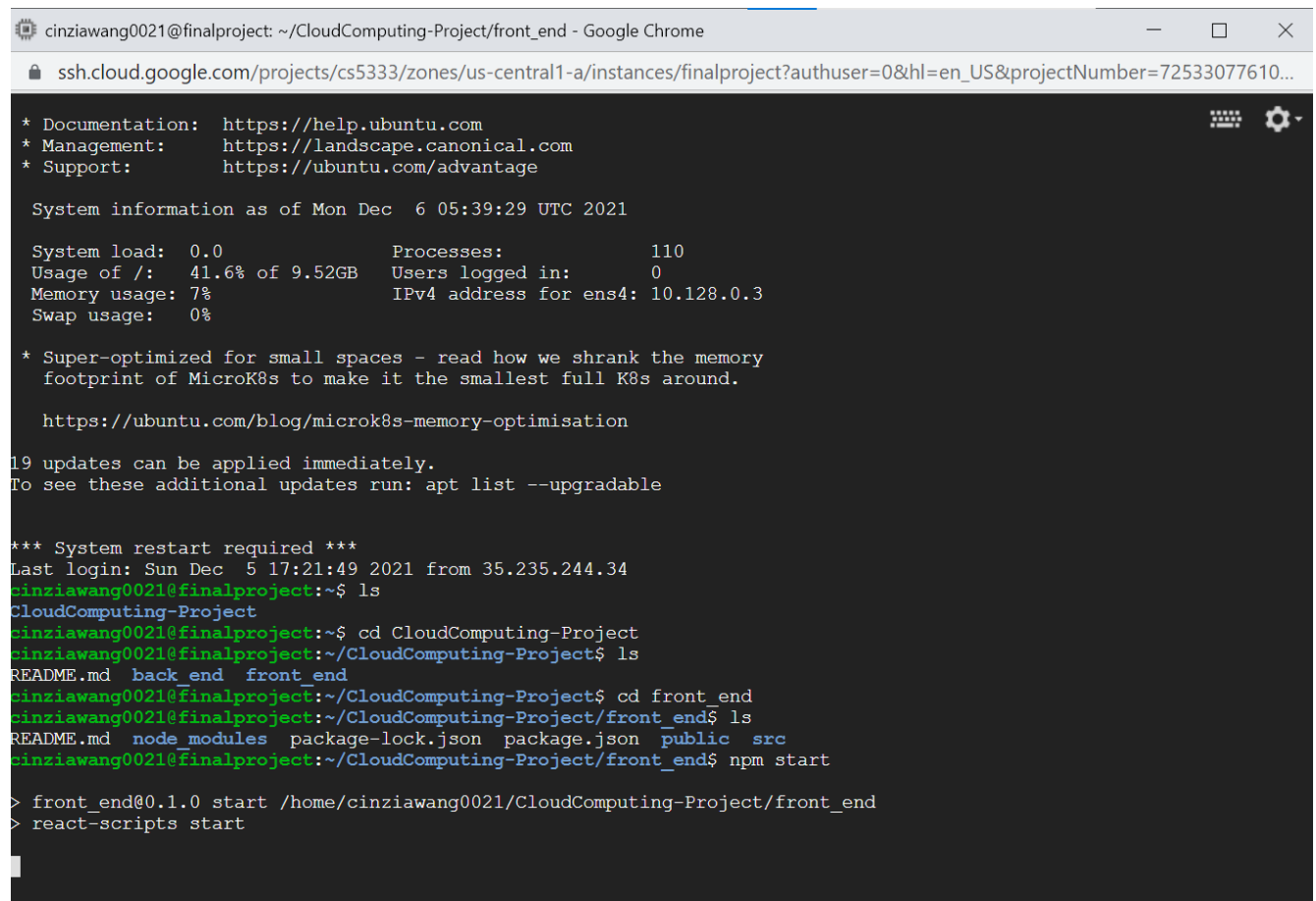
1. Create a project instance, the operating system is Ubuntu 20.0



The screenshot shows the Google Cloud Platform console. The left sidebar has a menu with 'Virtual machines' expanded, showing 'VM instances' (selected), 'Instance templates', 'Sole-tenant nodes', 'Machine images', and 'TPUs'. The main area is titled 'VM instances' and has a 'CREATE INSTANCE' button. Below this, there's a table of instances. The table has columns: Status, Name, Zone, Recommendations, In use by, Internal IP, External IP, and Connect. One instance is listed: 'finalproject' in the 'us-central1-a' zone, with internal IP '10.128.0.3' and external IP '35.209.133.249'. The 'Connect' column shows an 'SSH' button.

Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
Running	finalproject	us-central1-a	Save \$24 / mo		10.128.0.3 (nic0)	35.209.133.249	SSH

2. Login in and run project.



The screenshot shows a terminal window with the following content:

```
cinziawang0021@finalproject: ~/CloudComputing-Project/front_end - Google Chrome
ssh.cloud.google.com/projects/cs5333/zones/us-central1-a/instances/finalproject?authuser=0&hl=en_US&projectNumber=72533077610...

* Documentation: https://help.ubuntu.com
* Management:   https://landscape.canonical.com
* Support:      https://ubuntu.com/advantage

System information as of Mon Dec 6 05:39:29 UTC 2021

System load: 0.0          Processes:              110
Usage of /:  41.6% of 9.52GB Users logged in:          0
Memory usage: 7%          IPv4 address for ens4: 10.128.0.3
Swap usage:  0%

* Super-optimized for small spaces - read how we shrank the memory
  footprint of MicroK8s to make it the smallest full K8s around.

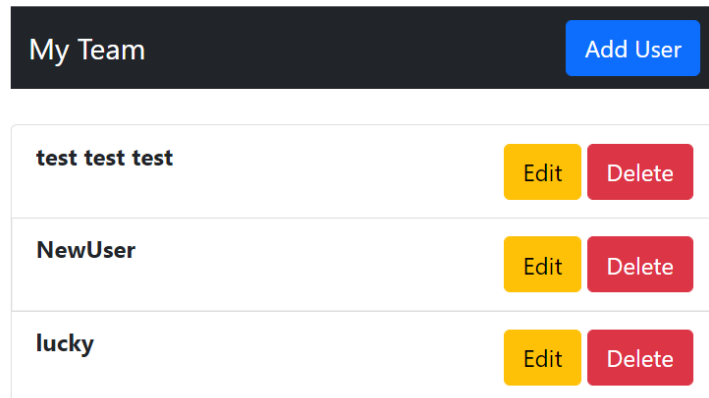
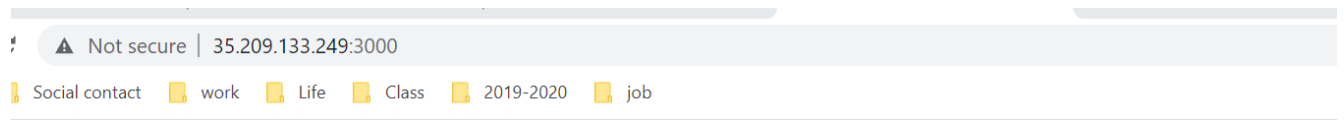
https://ubuntu.com/blog/microk8s-memory-optimisation

19 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

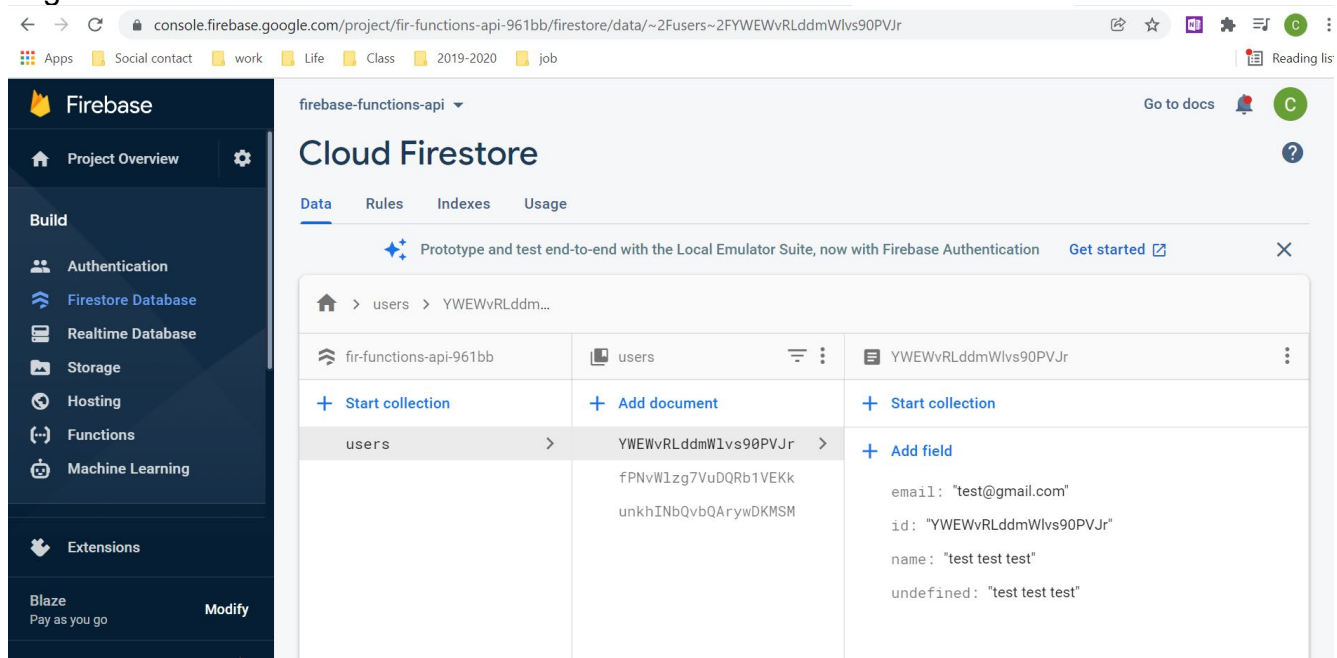
*** System restart required ***
Last login: Sun Dec 5 17:21:49 2021 from 35.235.244.34
cinziawang0021@finalproject:~$ ls
CloudComputing-Project
cinziawang0021@finalproject:~$ cd CloudComputing-Project
cinziawang0021@finalproject:~/CloudComputing-Project$ ls
README.md  back_end  front_end
cinziawang0021@finalproject:~/CloudComputing-Project$ cd front_end
cinziawang0021@finalproject:~/CloudComputing-Project/front_end$ ls
README.md  node_modules  package-lock.json  package.json  public  src
cinziawang0021@finalproject:~/CloudComputing-Project/front_end$ npm start

> front_end@0.1.0 start /home/cinziawang0021/CloudComputing-Project/front_end
> react-scripts start
```

3. Use external IP address and port 3000 to access the application.



4. Login to Firestore.



5. Next, we will display three functions of this application

- Add a user

# Add New User

Name

Helen

Email

Helen@gmail.com

Submit

Cancel

## Cloud Firestore

Data Rules Indexes Usage

Prototype and test end-to-end with the Local Emulator Suite, now with Firebase Authentication [Get started](#)

users > VZqxTBqDDJPF...

fir-functions-api-961bb

users

VZqxTBqDDJPFcGQTyzIH

+ Start collection

+ Add document

+ Start collection

users

VZqxTBqDDJPFcGQTyzIH

+ Add field

email: "Helen@gmail.com"

name: "Helen"

- Edit the user information

# Edit User

Name

Helenupdate

Email

Helenupdate@gmail.com

Update

Cancel

# Cloud Firestore

Data Rules Indexes Usage

Prototype and test end-to-end with the Local Emulator Suite, now with Firebase Authentication [Get started](#)

users > VZqxTBqDDJPF...

fir-functions-api-961bb

users

VZqxTBqDDJPFcGQTyzIH

+ Start collection

+ Add document

+ Start collection

users

VZqxTBqDDJPFcGQTyzIH

+ Add field

YWEWvRLddmWlvs90PVJr  
fPNvWlZg7VuDQRb1VEKk  
unkhINbQvbQArYwDKMSM

email: "Helenupdate@gmail.com"  
id: "VZqxTBqDDJPFcGQTyzIH"  
name: "Helenupdate"  
undefined: "Helenupdate@gmail.com"

- Delete a user

My Team

Add User

test test test

Edit

Delete

NewUser

Edit

Delete

lucky

Edit

Delete



# Cloud Firestore

Data Rules Indexes Usage

Prototype and test end-to-end with the Local Emulator Suite, now with Firebase Authentication [Get started](#)

users > YWEWvRLddm...

fir-functions-api-961bb	users	YWEWvRLddmWlvs90PVJr
<a href="#">+ Start collection</a>	<a href="#">+ Add document</a>	<a href="#">+ Start collection</a>
users >	YWEWvRLddmWlvs90PVJr >	<a href="#">+ Add field</a>
	fPNvW1zg7VuDQRb1VEKk unkhINbQvbQARYwDKMSM	email: "test@gmail.com" id: "YWEWvRLddmWlvs90PVJr" name: "test test test" undefined: "test test test"

## Code

Front end:

- Built the front end by react.js framework.
- Used the “axios” to call the cloud functions URI deployed on the Google Firebase platform.
- Created the Add User page, Edit User page, List User pages and some other components.

Edit User page

```
JS user.js JS EditUser.js X JS AddUser.js JS UserList.js
front_end > src > components > JS EditUser.js > ...
1  import React, { useState, useEffect } from "react";
2  import { Form, FormGroup, Label, Input, Button } from "reactstrap";
3  import { Link, useHistory } from "react-router-dom";
4  import axios from 'axios';
5
6
7  // Edit user function, need to get the information from the database,
8  // then update the information of a user, and then use put function to save to database.
9  // After saving to database, redirect the page to home page.
10 export const EditUser = (props) => {
11   const [selectedUser, setSelectedUser] = useState({
12     id: "",
13     name: "",
14   });
15   const history = useHistory();
16   const currentUserId = props.match.params.id;
17
18   // Get the user information by id from the database, and show in this page.
19   useEffect(() => {
20     const getUser = async () => {
21       const userId = currentUserId;
22       const res = await axios.get(`https://us-central1-fir-functions-api-961bb.cloudfunctions.net/user/${userId}`);
23       const user = res.data
24       setSelectedUser(user);
25     }
26     getUser()
27   }, [currentUserId]);
28
29   // Get the change information from the input
```

## Add User page

```
JS user.js JS EditUser.js JS AddUser.js X JS UserList.js
front_end > src > components > JS AddUser.js > AddUser
1 import React, { useState } from "react";
2 import { Form, FormGroup, Label, Input, Button } from "reactstrap";
3 import { Link, useHistory } from "react-router-dom";
4 import axios from 'axios';
5
6 // Add user to the database, and redirect to the home page
7 export const AddUser = () => {
8   const [name, setName] = useState("");
9   const [email, setEmail] = useState("");
10  const history = useHistory();
11
12  // After getting all the input information, using RESTfull API post to save to the database.
13  // Redirect to the home page.
14  const onSubmit = async (e) => {
15    e.preventDefault();
16    const newUser = {
17      name,
18      email,
19    };
20    await axios.post(`https://us-central1-fir-functions-api-961bb.cloudfunctions.net/user`, newUser );
21    history.push("/");
22  };
23
24  return (
25    <Form onSubmit={onSubmit}>
26      <h1>Add New User</h1>
27      <FormGroup>
28        <Label>Name</Label>
29        <Input
30          type="text"
```

## List User page

```
JS user.js JS EditUser.js JS AddUser.js JS UserList.js X
front_end > src > components > JS UserList.js > UserList > users.map() callback
1 import React, { useState, useEffect } from "react";
2 import { Link } from "react-router-dom";
3 import { ListGroup, ListGroupItem, Button } from "reactstrap";
4 import axios from 'axios';
5
6 // List the user in the firestore database, through the axios API call,
7 // We also do the remove function in here with the related user ID.
8 export const UserList = () => {
9
10  const [users, setUsers] = useState([]);
11
12  // list all the users get from database, when the users is changed, rerender the page.
13  useEffect( () => {
14    const getUsers = async () => {
15      const res = await axios.get(`https://us-central1-fir-functions-api-961bb.cloudfunctions.net/user`);
16      const users = res.data;
17      setUsers(users);
18    }
19
20    getUsers()
21  }, [users]);
22
23  // Delete the user by sending the function to the backend.
24  const removeUser = async (userId, e) => {
25    e.preventDefault();
26    await axios.delete(`https://us-central1-fir-functions-api-961bb.cloudfunctions.net/user/${userId}`)
27  }
28
29  return (
30    <ListGroup className="mt-4">
```

Back end:

- Used the Node.js with express framework to create the back end.
- Used the RESTful API to create GetAll, GetById, EditUser, DeleteById functions.
- Deployed the cloud functions to the Google firebase platform.
- Connected to the Firestore database and do the CRUD operations on the database.

User functions 1

```
Go Run Terminal Help user.js - CloudComputing-Project - Visual Studio Code


JS user.js X JS EditUser.js JS AddUser.js JS UserList.js

back_end > functions > controller > JS user.js > ...
1  const functions = require("firebase-functions");
2  const express = require("express");
3  const cors = require("cors");
4
5  // Connect to the firestore database
6  const admin = require("firebase-admin");
7  admin.initializeApp();
8  const db = admin.firestore();
9  |
10 // Create the express app, and use cors for outside connecting
11 const userApp = express();
12 userApp.use(cors({origin: true}));
13
14 // Restful api get fuction is to list all the users in the database
15 // Send the data to the frontend, with status code 200
16 userApp.get("/", async (req, res) => {
17   const snap = await db.collection("users").get();
18
19   let users = [];
20   snap.forEach( doc => {
21     let id = doc.id;
22     let data = doc.data();
23
24     users.push({id, ...data});
25   });
26   res.status(200).send(JSON.stringify(users));
27 });
28
29 // Get by ID fuction is getting the user by the passing parameter ID in the database
30 // Send the data to the frontend, with status code 200
```

## User functions 2

```
JS user.js  X  JS EditUser.js  JS AddUser.js  JS UserList.js
back_end > functions > controller > JS user.js > ...
31  userApp.get("/:id", async(req, res)=>{
32      const snap = await db.collection("users").doc(req.params.id).get();
33      const userid = snap.id;
34      const userdata = snap.data();
35
36      res.status(200).send(JSON.stringify({id: userid, ...userdata}));
37  })
38
39  // Put function is getting new user information from front end, and save to the database to
40  // replace the older user information
41  userApp.put("/:id", async (req, res)=>{
42      const body = req.body;
43      await db.collection("users").doc(req.params.id).update(body);
44      res.status(200).send();
45  })
46
47  // Delete the user by id, and return status code with no body.
48  userApp.delete("/:id", async(req, res)=>{
49      await db.collection("users").doc(req.params.id).delete();
50      res.status(200).send();
51  })
52
53  // Save to new user to the database
54  userApp.post("/", async(req, res) => {
55      const user = req.body;
56      await db.collection("users").add(user);
57      res.status(201).send();
58  });
59
60  exports.user = functions.https.onRequest(userApp);
```

## Deployed cloud functions

 **Firebase**

Project Overview

Build

Authentication

Firestore Database

Realtime Database

Storage

Hosting

Functions

Extensions

firebase-functions-api

Go to docs

?

# Functions

Dashboard Health Logs Usage

Protect your Functions resources from abuse, such as billing fraud or phishing [Configure App Check](#)

Function	Trigger	Region	Runtime	Memory	Timeout
user	Request HTTP https://us-central1-fir-functions-api-961bb.cloudfunctions.net/user	us-central1	Node.js 14	256 MB	60s

## **Teammate Evaluation**

**Yu Zhao:** Project Introduction, Service/Application Overview, Components Used, Front-end and Back-end code

**Xiaojing Wang:** Architecture, Design, Google Compute Engine, Code Deployment

**Sujeeth Nilakantan:** Implementation Plan, Test Plan, Firestore, Cloud Functions