Mern Stack CRUD + Function

This is a very easy project that realised the full-stack function of C-R-U-D, by using MERN Stack (MongoDB + Express + React + Node). There are only 5 files in Backend and 8 files in Frontend in this project.

By using Mern Stack, we can run both back-end and front-end parts in Visual Studio Code, so it will be easy to handle this full-stack project.

The reason why we chose Firebase to connect with React is because we **don’t need** to set up **Backend** program.

In this part, we need to allow a user to

* **CREATE - Add a pallet for shipping (add to our Firestore database)** – Allow the user to specify the goods on the pallet and the overall weight (KG).
* **READ - Display all pallets** in our warehouse**.**
* **UPDATE - Edit a pallet** – allow the user to change the details for an existing pallet in our Firestore database.
* **DELETE - Delete a pallet** – remove this pallet from our Firestore database

The following section explains the function that firebase used in each component.

# server.js

const express = require("express");

const cors = require("cors");

const mongoose = require("mongoose");

// Import **Express Framework**, **CORS**(Cross-Origin Resource Sharing) middleware, **Mongooose** Model

const app = express();

// Create an instance of Express Application

app.use(cors());

// Allow Express Application to Share Cross-Origin Resource

app.use(express.json());

// Allow Express Application to parse incoming Request with JSON payloads, and to process JSON data in the request body.

require('dotenv').config();

// 1 – Import “dotenv” package

// 2- Read the file of “.env”, put their data into “**process.env**” object.

const port = process.env.PORT || 5000;

const mongoUrl = process.env.ATLAS\_URI;

mongoose

  .connect(mongoUrl, {

    useNewUrlParser: true,

  })

// 1- Connect to MongoDB with the “mongoUrl”

// 2- Use the new, updated URL Parser from MongoDB

  .then(() => {

    console.log("MongoDB database connection established successfully");

  })

  .catch((e) => console.log(e));

app.use(express.urlencoded({ extended: false }));

// All encoded URL will be put into key-value data. This allows Express Application to use a easier library(‘querystring’) to parse Request and Response

const exercisesRouter = require('./routes/exercisesRoute');

const usersRouter = require('./routes/userRoute');

app.use('/exercises', exercisesRouter);

app.use('/user', usersRouter);

// Set

app.listen(port, () => {

  console.log(`Server is running on port: ${port}`);

});

1. Create an Express Application

# App.js

The first page of this program, it depends whether t**heAuthUser** has logged in successfully or not.

It determines the next page to display by whether the user has authenticaded.

**Success => PalletDashboard + SignOut**

**Failed => Signin.js**

Authentication status is failed by default. **useState(null)**

# fbconfig.js

**initializeApp(firebaseConfig)**

Initializes the **Firebase Application instance** according to the parameters defined by firebaseConfig,

**getApp()**，

Get the Firebase application instance in your project.

After we have an application instance, we can access **Firebase authentication**, **database operations,** **storage**, and other services.

const db = initializeFirestore(firebaseApp,

  {

      experimentalForceLongPolling : true

  });

**initializeFirestore(firebaseApp)** 根据获取好的firebase**应用程序实例**，初始化Firebase的**实时数据库服务**。这样React中的Firebase应用程序能够使用实时数据库服务。

Initialize Firebase's **real-time database service** according to the acquired **Firebase Application instance**. This allows the Firebase application in React to use the real-time database service.

**experimentalForceLongPolling**

Forces the use of **long polling** as the transport method for communicating with the **Firestore server.**

Long polling is a technique for implementing real-time communication, which simulates real-time by means of a continuous connection between the client and the server. Typically, the Firebase SDK automatically selects the best transport mechanism (such as **WebSockets**) based on the environment and browser support. However, WebSockets may not work due to network limitations or environment-specific restrictions in some cases. In those cases, experimentalForceLongPolling can be used to force the connection between React and Firestore Server.

# Signin.js

**getAuth(firebaseApp)**

Gets an **instance** of the **Authentication** service, passing an initialized Firebase application -- **firebaseApp**, as an argument.

**e.preventDefault()**

Many elements and events have default behavior in the browser. For example, forms often try to submit to the server when the user clicks the submit button. Disabling the default submit behavior of a form gives you complete control over the event handling. This method is useful for handling link click events, form submit events, and so on.

**signInWithEmailAndPassword(auth, email, password)**

This is the function used in the **Firebase Authentication Service** for **user login via email and password**。

There are three parameters that needs to be passed:

* **auth:** An **instance of the authentication service**, which can be obtained via the **getAuth** method.
* **email:** user’s email
* **password:** user’s password

If the authentication **succeeds**, the callback function in the **then()** receives a **UserCredential** object which contains the user's information.

If the authentication fails, the callback function in the catch() receives an object which contains an **error message.**

# Signout.js

**SignOut()**

This function is used to **log out the current use**r in the Firebase Authentication Service. It serves to **terminate the current user's session** and log them out of the application.

# PalletDashboard.js

Get all the pallets information from Firestore Database.

图形用户界面

描述已自动生成

**collection(query, (snapshot)=>{})**

  const q = query(

      collection(db, "pallets"),

      where("userID", "==", currentUser.uid)

    );

In Firebase's Firestore, the query function is used to create and execute queries against collections in your Firestore database.

It the above code, Firebase will find a **database** named **“db”** with the **collection** named **“pallets”**, and it must comply with the user login authentication.

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成图形用户界面, 文本, 应用程序

描述已自动生成**db.userID currentUser.uid**

**onsnapshot()**

It is the function used by Firebase to listen to document and collection changes in real time. **If a document/collection is changed (created, updated or deleted)**, the listener is triggered and exposed to the corresponding change data.

querySnapshot.forEach((doc) => {

        palletsArrayFromFirebase.push({ ...doc.data(), id: doc.id });

* **doc.data()**  - the content of the document
* **doc.id**  - the id of the document

***Collection name***

* ***document.id***
* ***- document content***

图形用户界面, 应用程序

描述已自动生成

# DisplayPallet.js

**deleteDoc(doc(db, collection, id))**

deleteDoc(doc(db, "pallets", id));

This function is a call that **removes the specified document** from the "pallets" collection **according to the ID of the document**.

* **db** - represents a Firebase database instance.
* "**pallets"** - the name of the collection.
* **id** - the **ID** of the document to be deleted

**updateDoc(doc(db, collection, id),newData)**

    await updateDoc(doc(db, "pallets", docu.id), {

      delivered: !docu.delivered

    });

This is a function used to update the data of the **specified document**. According to **docu.id,** the updated data includes:

**description：**remain the same

**weight：**remain the same

**delivered：**change the data according to the input

**createdAt：**remain the same

**userID：**remain the same

The function of **updateDoc** can only update some of the fields while leaving others unchanged.

# AddPallet.js

**form onSubmit={handleSubmit}>**

A <form> is an HTML element used to create a form that can contain input fields, buttons, and other elements. When the user fills out the form and submits it, data can be sent or certain actions can be triggered.

* **onSubmit：**

**onSubmit** is an event listener property used to catch form submission events.

The **onSubmit event** is triggered when the user submits the form (usually by clicking the submit button or pressing the enter key).

* **handleSubmit：**

handleSubmit is a function that defines the action that should be performed when a form is submitted.In React, this function is typically used to process form data, such as validating input, sending data to the server, or updating the application state.

In React, form elements are typically bound to the component's **State**, which makes it easy to collect and manage user input data. The handleSubmit function accesses the **State** data and uses it when the user submits the form.

**addDoc(collection(db, "pallets"), newData)**

It is a function in Firebase to **add a new document** in the specified collection. There are 2 parameters to be passed:

* **db** - represents a Firebase database instance.
* **“pallets**” – nane of the collection.
* **newData – an object contains the data that needs to be stored in the new document.**

**Auto-generated Document ID:** When a document is added using **addDoc**, Firestore automatically generates a unique ID for the new document, as opposed to manually specifying the document ID (such as setDoc).

**Aynchronours function:** addDoc is an asynchronous function that returns a Promise. you can use the .then() and .catch() methods to handle the success or failure of the add operation.

# EditPallet.js

    await updateDoc(doc(db, "pallets", palletToEdit.id), {

      description,

      weight,

      delivered: palletToEdit.delivered,

      createdAt: palletToEdit.createdAt,

      userID: palletToEdit.userID

    });

This code uses the Firebase Firestore service and shows how to asynchronously update a specific document in the Firestore database:

1. **Asynchronous update operation:** the keyword **“await”** is used to wait for an asynchronous operation to complete. This means that the code waits for the updateDoc function to complete the update operation before continuing with the code that follows. It is often used to handle time-consuming network requests such as database operations.

**2）doc(db, "pallets", palletToEdit.id)**

* + **db** - represents a Firebase database instance.
  + "**pallets"** - the name of the collection.
  + **palletToEdit.id** - the **ID** of the document to be updated
  + **newData** - The object to be updated and the key in the object to be updated. If the key is specified in newData, this function will assign the new value to that key in the object. If the key is not specified in newData, the value of the key remains the same.

**3）Update data:** This is a function used to update the data of the specified document. According to palletToEdit.id, the updated data includes:

**description：**change the data according to the input

**weight：**change the data according to the input

**delivered：**remain the same

**createdAt：**remain the same

**userID：**remain the same

The function of **updateDoc** can only update some of the fields while leaving others unchanged.

**const bcypt = require (“bcyptjs”)**

bcrypt 是一种哈希加密算法，用于实现密码的安全存储和验证。

必须通过**异步**才能实现。

1. **引入 bcryptjs 模块：**通过 `require("bcryptjs")`，Node.js 将加载 `bcryptjs` 模块。这是 Node.js 中引入外部模块或库的标准方式。`bcryptjs` 是一个流行的 JavaScript 库，用于密码哈希。

2. **创建 bcrypt 变量**：将 `bcryptjs` 模块的导出内容分配给变量 `bcrypt`。这样，你就可以在你的代码中通过 `bcrypt` 变量使用 `bcryptjs` 模块提供的所有功能和方法。

3. **密码加密功能**：`bcryptjs` 提供了一系列用于加密和验证密码的方法。例如，它可以将用户密码转换为哈希值，以安全地存储在数据库中。这对于保护用户密码免受未经授权的访问非常重要。

4. **安全性**：`bcryptjs` 使用 bcrypt 加密算法，这是一个安全的方式来存储密码。即使两个用户有相同的密码，由于 bcrypt 的工作原理（使用盐和哈希），它们的存储哈希将是不同的，增加了安全性。