1. An introduction of your application, including why you want to develop the application and the main functions of your application.

   Our application transfers data distinguished by regional association into interactive 3D mode. The three data we used for this application are '107年成人保護級家庭暴力案件統計', '交通事故每千人死傷數' and '全國身心障礙機構一覽表'.

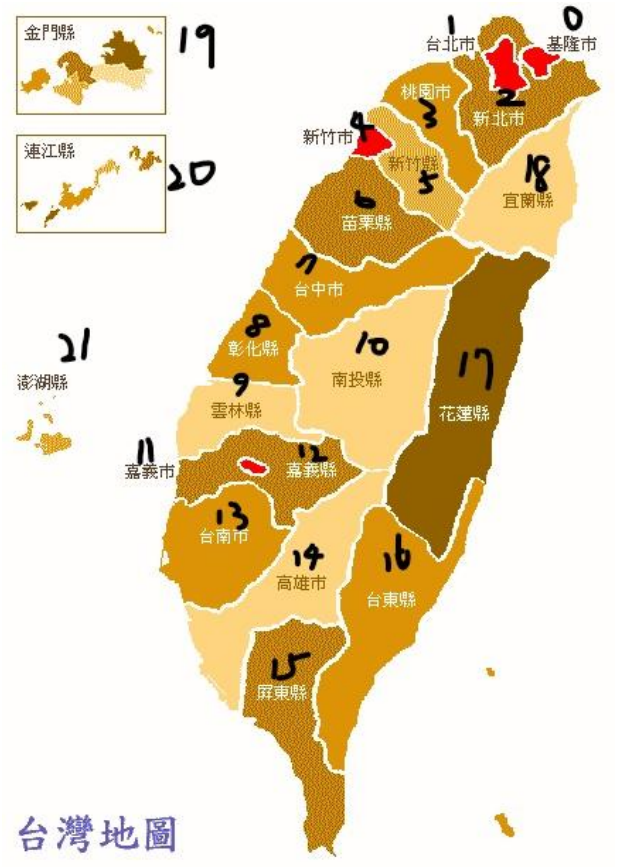
2. Database design - describe the schema of all your tables in the database, including keys and index, if applicable (why you need the keys, or why you think that adding an index is or is not helpful).

   For this application, we chose mongodb as our database server, which is well-known as a NoSQL platform(means that we have no keys in this DB by definition, but from the conceptual aspect, {province, region} can define a certain field).
   The reason behind this decision is that this application only consists of one collection schema for displaying the 3D data visualization. As for the index mongodb will generate a '_id' for each field in a collection, so I didn't do further setups on the index.

   The schema in mongodb(using mongoose):

```
import mongoose from "mongoose";

const DataSchema = new mongoose.Schema({
    province: {
        type: String,
        required: true
    },
    region: {
        type: String,
        required: true
    },
    data: {
        type: Number,
        required: true
    },
    province_id: {
        type: Number,
        required: true
    },
    region_id: {
        type: Number,
        required: true
    },
    ratio: {
        type: Number,
        required: true
    },
    occupation: {
        type: Number,
        required: true
    }
})

export default mongoose.model('Data', DataSchema, 'datas');
```

region_id:
https://docs.google.com/spreadsheets/d/13UjmIDXJpniDhp_0FWFkJJJAbH53LHYOxVxIC-WsSkA/edit#gid=0
province_id:



3. Database design - describe the normal form of all your tables. If the tables are not in BCNF, please include the reason for it (performance trade-off, etc.).

   For NoSQL database, there's a discrepancy between “Embedded Data Model” and “Normalized Data Model” . “Embedded Data Model” embedded some smaller collections in a collection, and “Normalized Data Model” split all the collections apart.

   Our schema pertains to the “Normalized Data Model”, using this instead of the Embedded model can improve write speed. The implementation of the DB is to delete all the fields and re-post the data fetched from the internet, so the increase of the write speed will improve the performance of updating the collection.

4. From the data sources to the database - describe the data source and the original format.

   Our data sources '107年成人保護級家庭暴力案件統計' and '全國身心障礙福利機構一覽表' are from https://data.gov.tw/ . It's a website providing open data to the public. By means of the data, the public can strengthen the power of supervising the government.

   the original format of '107年成人保護級家庭暴力案件統計':
   https://data.gov.tw/dataset/147093
   The original format is composed by 36 attributes, including OWNERCITYCODEOWNERCITYCODE, INFOERTYPE, INFOUNIT, OTHERINFOERTYPE, OTHERINFOUNIT, CLIENTID, SEXID, DBDATE, DIDTYPE, DEDUCATION, DMAIMED, DOTHERMAIMED, DOTHERMAIMED2, DOCCUPATION, DOTHEROCCUPATION, RELATIONSHIP, OTHERRELATIONSHIP, HELPACTION, OTHERHELPACTION, FOLLOWACTION, OTHERFOLLOWACTION, CASEEVAL, OCCURTIME, RECEIVETIMEYEAR, RECEIVETIMEMONTH, and TOWNCODE. After calculating the occurance number of every town,we only kept OWNERCITYCODEOWNERCITYCODE (renamed it as province), TOWNCODE (renamed it as region), and data (occurance number). Furthermore, we added province_id, region_id, ratio, and occupation. Attribute province_id and region_id can be referred to by the following resources. Besides, the advanced usage of URL forms ratio and occupation column.

   the original format of '全國身心障礙機構一覽表':
   https://data.gov.tw/dataset/12061
   It contains 行政區、名稱、屬性、地址、電話、傳真、服務對象、服務內容、設立日期、總計、全日型住宿人數、夜間型住宿人數、日間照顧人數、部分時制照顧人數, but for our needs, we only kept the 行政區(province)、地址(region)、總計(data) and added region_id、province_id、ratio、occupation to use.

   the data source of 交通事故每千人死傷數:
   https://roadsafety.tw/Dashboard/Custom?type=%E9%84%89%E9%8E%AE%E5%B8%82%E5%8D%80%E6%AF%8F%E5%8D%83%E4%BA%BA%E6%AD%BB(%E5%82%B7)%E6%95%B8
   It's a website that can search for the number of injuries and deaths resulting from traffic accidents. Since we couldn't find the original table of the data, we selected the province and region on the website, and recorded the results from each region. After getting those data, we calculate the ratio to represent the height of the data in the 3D model, and we divide each number by the maximum of all the numbers to get the ratio. Lastly, we add several columns, which are region_id, province_id, and occupation according to above resources for advanced usage.

5. From the data sources to the database - describe the methods of importing the original data to your database and strategies for updating the data, if you have one.

   After the api fetches the .csv file, we transform the data into the form of the schema, through 'post' from api, the data will be saved into DB.

   About renewing the data field, I set a timer to delete the data and post them again. Because I did not know which data field to update, directly deleting then posting again seems feasible.

```
setTimeout(updateDB, 1800000);
```
(180000ms = 30 minutes)

```
function updateDB(){
  axios.delete('http://34.230.89.214:8000/deleteData')
      .then((response) => {
        console.log(response.data);
      })
      .catch((err) => {
        console.log(err);
      })
  axios.post('http://34.230.89.214:8000/updateData1')
      .then((response) => {
        console.log(response.data);
      })
      .catch((err) => {
        console.log(err);
      })
  axios.post('http://34.230.89.214:8000/updateData2')
      .then((response) => {
        console.log(response.data);
      })
      .catch((err) => {
        console.log(err);
      })
  axios.post('http://34.230.89.214:8000/updateData3')
      .then((response) => {
        console.log(response.data);
      })
      .catch((err) => {
        console.log(err);
      })
}
```

6. Application with database - explain why your application needs a database.

   Databases can be used to go beyond basic data storage and transactions to analyze huge volumes of data from multiple systems. In addition, due to the flexibility and scalability, we can extend our data and create an organized format as data changes over time.

7. Application with database - includes the queries that are performed by your application, how your application performed these queries (connections between application and database), and what are the cooperating functions for your application.

For ‘get’, I used the function find() in mongoose with filter ‘{}’ to get all the field from (Data1/ Data2/Data3)

```
app.get('/Data1', async(req, res) => {
        var condition = {};

        db.datas.find(condition)
          .then(data => {
            res.json(data);
          })
          .catch(err => {
            res.status(500).send({
              message:
                err.message || "Some error occurred while retrieving datas."
            });
          });
});
```

For ‘delete’, I used deleteMany with filter ‘{}’ to delete all the data in the DB.

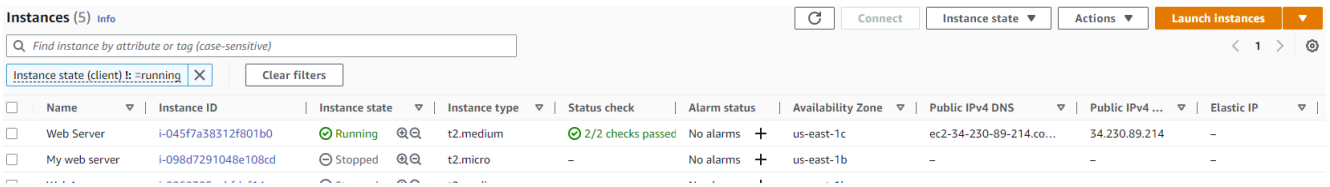
```
app.delete('/deleteData', async function(req, res){
  clearCollections();
  res.send('Delete data in DB');
});
async function clearCollections() {
  const collections = mongoose.connection.collections;

  await Promise.all(Object.values(collections).map(async (collection) => {
      await collection.deleteMany({});
  }));
}
```

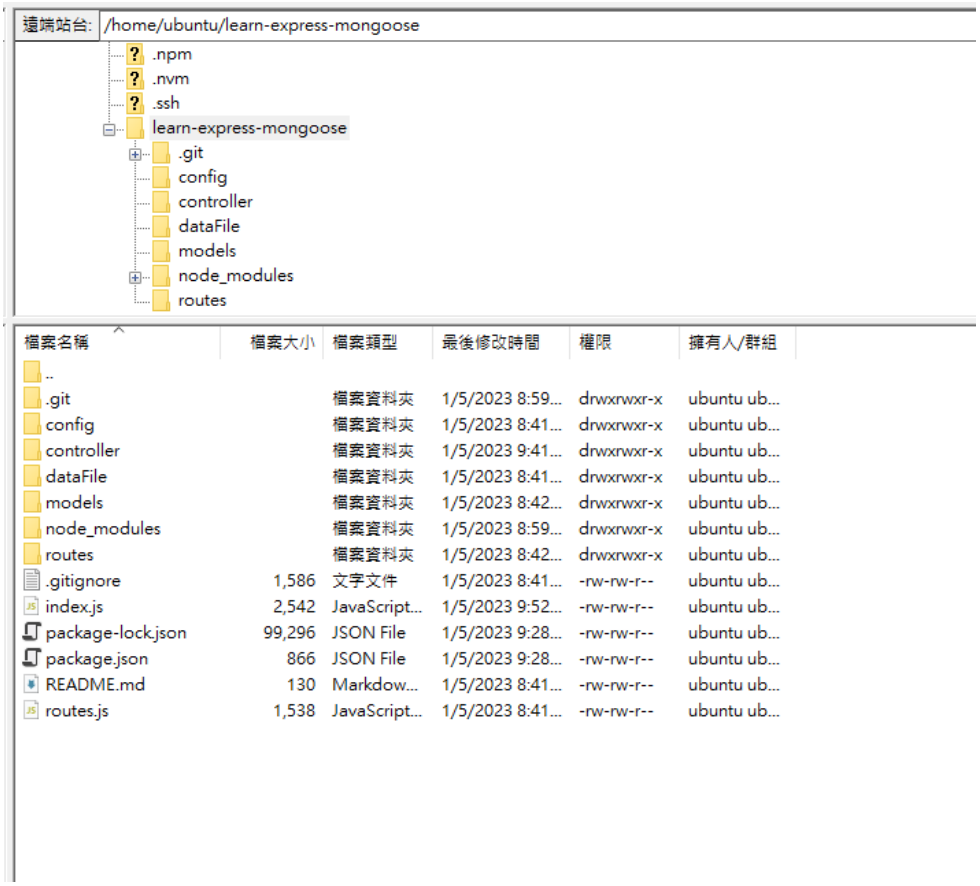
The implementation with api in frontend.

```
mounted(){
  axios.get('http://34.230.89.214/Data1')
    .then((response) => {
      //console.log(response.data);
        this.options[0].data = response.data;
    })
    .catch((err) => {
      console.log(err);
    });

  axios.get('http://34.230.89.214/Data2')
    .then((response) => {
      //console.log(response.data);
      this.options[1].data = response.data;
    })
    .catch((err) => {
      console.log(err);
    });

  axios.get('http://34.230.89.214/Data3')
    .then((response) => {
      //console.log(response.data);
      this.options[2].data = response.data;
    })
    .catch((err) => {
      console.log(err);
    })
  setTimeout(updateDB, 1800000);
},
```

8. All the other details of your application that you want us to know.

Implementation:
Frontend: Vue3, Threejs, axios
Backend: Express, nodejs, mongoose
Database: Mongodb
Server: AWS EC2 ubuntu-focal-20.04





we have recorded a video about application and some explain
link of the video: https://youtu.be/PdxhdXAPpGY