Internet Technology Lab

Name: Tonmoy Biswas

Roll No: 002110501133

Class: BCSE 3rd Year 2nd Sem

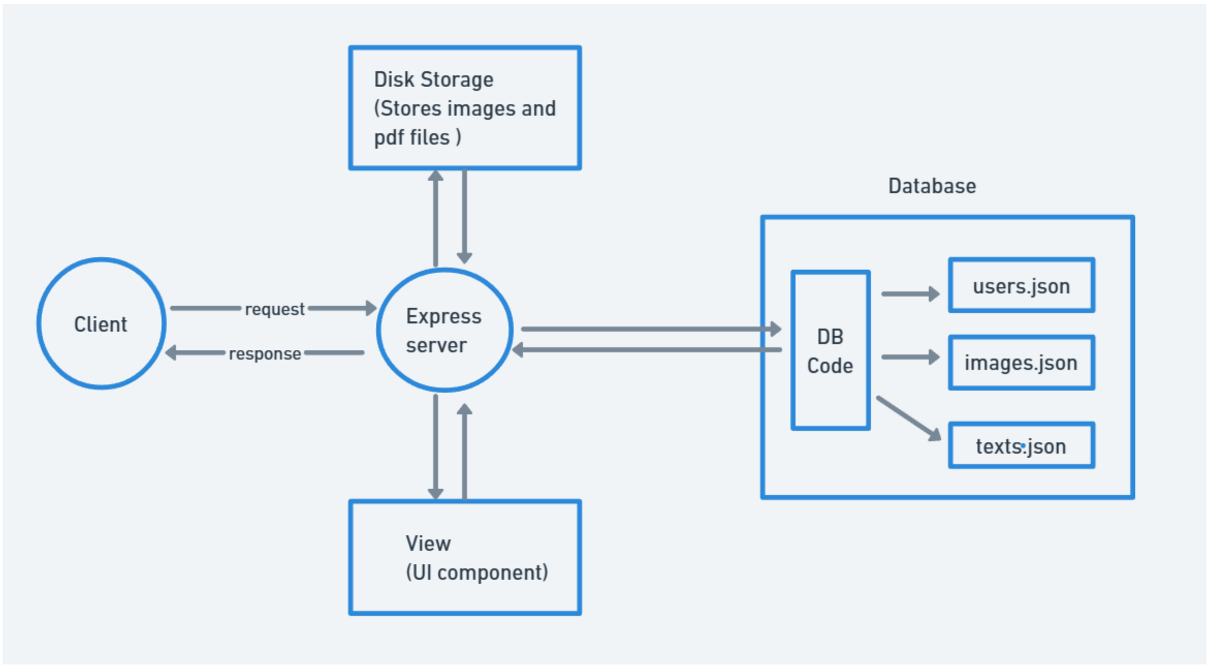
Section: A3

Assignment No: 2

Problem Statement:

Write a multi-client data repository using Node.JS. You can use the Express framework. Clients would upload text/image data to the server. A client may also get to view and download the collection. However, editing of the data is not allowed by any client.

Architecture Diagram:



Main Feature and Design Pattern:

- **Authentication:** User can sign up and login and access their own repository. Authentication keeps repository private and secure.
- **Upload Files:** User can upload their images and pdf files on the server.
- **View and Download Files**: User can view and their images and pdf files. Only authenticated and authorized user can access their files.
- Here we use *Dependency Injection Design pattern*. In the express app we inject several dependencies such as cookie parser, body parser, view engine etc.

Software Components:

We break down the application in 4 software components –

- **Server Component:**
  The server is implemented using Express library. It contains request handler for different request url and request method. It accept request from client and interact with Database , View and Disk Storage module.
  When client request for a web page it interact with Database module to get data and from

View module it get the UI components and populate it with the data fetched from Database.
It use Json Web Token(JWT) and Cookies for authentication.
When user upload files it store the raw file in the disk storage and store the path information of the file in the database.

- **View Component:**
  We use ejs view engine to render HTML. All the UI components are stored in the View Component. When client request for a web page express server interact with Database module to get data and from View module it get the UI components and populate it with the data fetched from Database.
- **Database Component:**
  All the data are stored in the database component. We implement the database using json files and middle layer between the express server and json file to handler database operation and manipulate json files. The images.json file stores the uploaded files' path, users.json file stores user information, texts.json file stores uploaded text(small notes).
- **Disk Storage:**
  We use disk storage to store the raw files

Code Snippets:

Server Code:

```
const express=require('express');

const cors=require('cors');
const multer=require('multer');
// const bodyParser=require('body-parser');
const path=require('path');
const expressLayouts=require('express-ejs-layouts');
const cookieParser = require('cookie-parser');
const jwt=require('jsonwebtoken');
require('dotenv').config();
const middleware=require("./utility/middleware");
const fs=require('fs');
const db=require("./utility/db")




const port=8000;
const secret=process.env.SECRET;


const app=express();
app.use(express.json());
app.use(express.urlencoded());
app.use(cookieParser());
app.use(cors());
app.use(expressLayouts);
app.use(express.static('public'));
// app.use(bodyParser());


app.set('views',path.join(__dirname,'views'));
app.set('view engine','ejs');


// Set the storage engine
const storage = multer.diskStorage({
    destination: function (req, file, cb) {
        cb(null, './public/images/');  // Uploads will be stored in the
'uploads' directory
    },
    filename: function (req, file, cb) {
        cb(null, file.fieldname + '-' + Date.now() +
path.extname(file.originalname));
    }
});

// Initialize Multer with the storage engine
const upload = multer({ storage: storage });

//......................................page
request........................
//register page
app.get("/",function(req,res){
    res.render("register.ejs",{signin:req.authorized});
});

app.get("/file-upload-page",middleware.authCheck,function(req,res){

    res.render("file_upload_page.ejs",{signin:req.authorized});
});

app.get("/text-upload-page",middleware.authCheck,function(req,res){

    res.render("text_upload_page.ejs",{signin:req.authorized});
});

app.get("/file-show-page",middleware.authCheck,function(req,res){
    // const images=["/images/myImage-1705932093864.png"]
    const images=db.getImage(req.user.email);
    // console.log("images",images);
    res.render("file_show_page.ejs",{images,signin:req.authorized});
});

app.get("/text-show-page",middleware.authCheck,async function(req,res){
    // const textdb=fs.readFileSync("./public/storage/texts.json","utf-8");
    // console.log(JSON.parse(textdb)["tonmoy"]);
    // const texts=["nice text 1","nice text 1","nice text 1","nice text
1","nice text 1","nice text 1","nice text 1"];
    const texts=db.getText(req.user.email);
    res.render("text_show_page.ejs",{texts,signin:req.authorized});
});


//...................api........................
app.post('/upload-image',middleware.authCheck, upload.single('myImage'), (req,
res) => {
    // Multer has added the 'file' object to the request
    const file = req.file;

    if (!file) {
        return res.status(400).send("No file uploaded");
    }

    // You can access file properties like file.path, file.filename, etc.
    // Perform additional actions (e.g., save the file path in a database)

    // res.send('File uploaded!');
    // return res.status(200).json({ok:true,message:"File uploaded
successfully"});
    // console.log("file",file);
    const filepath=`/images/${file.filename}`;
    db.addImage(req.user.email,filepath);
    return res.redirect("/file-show-page")
});

app.post("/upload-text",middleware.authCheck,function(req,res){
    // console.log(req.body);
    db.addText(req.user.email,req.body.text);
    res.redirect("/text-show-page");
});

app.get("/logout",function(req,res){
    res.clearCookie("jwt");
    res.redirect("/");
});

app.post("/register",async function(req,res){
    // console.log(req.body);
    const password=db.getUserPassword(req.body.email);
    if(password){
        if(password!=req.body.password){
            return res.send("wrong password")
        }
    }
    else{
        db.addUser(req.body.email,req.body.password);
    }
    const token= jwt.sign({email:req.body.email},secret);
    res.cookie("jwt",token);
    res.redirect("/text-show-page");
})

app.listen(port,function(err){
    if(!err){
        console.log(`The server is running on port ${port}`);
    }
    else{
        console.log(`Error: ${err.message}`);
    }
})
```

Auth Check Middleware:

```
const jwt=require('jsonwebtoken');

require('dotenv').config();


module.exports.authCheck=async function(req,res,next){
    try{
        const secret=process.env.SECRET;
        if(!req.cookies.jwt){
            return res.redirect("/");
        }
        const user=jwt.verify(req.cookies.jwt,secret);
        req.authorized=true;
        req.user=user;
        // console.log("req.user",req.user);
        // console.log("user",user);
        next();
    }
    catch(err){
        res.clearCookie("jwt");
        return res.redirect("/");
    }
}
```

Database implementation:

```
const fs=require('fs');
const userpath="./public/storage/users.json";
const imagepath="./public/storage/images.json";
const textpath="./public/storage/texts.json";


function addUser(email,password){
    let users=fs.readFileSync(userpath,"utf-8");
    users=JSON.parse(users);
    users[email]=password;
    fs.writeFileSync(userpath,JSON.stringify(users));
}

function getUserPassword(email){
    let users=fs.readFileSync(userpath,"utf-8");
    users=JSON.parse(users);
    // console.log(users);
    return users[email];
    // fs.writeFileSync(userpath,JSON.stringify(users));
}

function addImage(email,url){
    let images=fs.readFileSync(imagepath,"utf-8");
    images=JSON.parse(images);
    if(!images[email]){
      images[email]=[];
    }
    images[email].push(url);
    fs.writeFileSync(imagepath,JSON.stringify(images));
}

function addText(email,text){
    let texts=fs.readFileSync(textpath,"utf-8");
    texts=JSON.parse(texts);
    if(!texts[email]){
      texts[email]=[];
    }
    texts[email].push(text);
    fs.writeFileSync(textpath,JSON.stringify(texts));
}

function getImage(email){
    let images=fs.readFileSync(imagepath,"utf-8");
    images=JSON.parse(images);
    if(!images[email]){
        return [];
    }
    return images[email];
    // if(!images[email]){
    //   images[email]=[];
    // }
    // images[email].push(url);
    // fs.writeFileSync(imagepath,JSON.stringify(images));
}

function getText(email){
    let texts=fs.readFileSync(textpath,"utf-8");
    texts=JSON.parse(texts);
    if(!texts[email]){
        return [];
    }
    return texts[email];
    // if(!texts[email]){
    //   texts[email]=[];
    // }
    // texts[email].push(text);
    // fs.writeFileSync(textpath,JSON.stringify(texts));
}

module.exports={
    addUser,
    getUserPassword,
    addImage,
    getImage,
    addText,
    getText
};
```

Discussion:

There is some scope of optimization in the application. We can use external services to store and serve files optimally like AWS S3.
The passwords stored in the database are in its original format. It leads to security issue. We can store the passwords in the database in hashed form. We can also use external database service which will store the data in a optimized way. We can create a cluster of express servers and introduce a load balancer between them to improve performance and reduce response time.