Back-end Document

Database Setup and Connection

Written by Jiaxin Mo

MongoDB

In this project, we followed the previous work of Swatkats to use MongoDB as our main database for storing textual information, including URL of photos and files. It was cheap to use, easy to extend and maintain. MongoDB Atlas provided a free cluster with 500MB storage. When we defined the attributes in our code, the data models could be automatically setup in database. This allowed us to modify changes easily.

Related operations:

1. Create a mongoDB account on Atlas.
2. Create a free cluster.
3. Install 'MongoDB for VS Code' on VSCode (if your IDE is VSCode).
4. Update connection URL in server.js.

Tips: download a mongoDB Compass to monitor the database easier.

There were two data models defined, USERS and PROFILES. USERS database was for storing user`s account information, such as name, login email and login password. PROFILES database stored user`s porfolio information, such as education, work experience, URL of photos and files, etc.

Setup

Server.js

```
const mongoURI = "mongodb+srv://<username>:<password>@<clustername>.mongodb.net/test?
retryWrites=true&w=majority"
```

Models

Please refer to **user.js**, **profile.js** in github repository.

Amazon Simple Storage Service (AWS S3)

AWS was used as our secondary database. Due to the limitation of small capacity of MongoDB, we selected AWS for its 5GB capacity to store a large number of pictures and PDF files. S3 provided a scalable storage in cloud for free, and was cost effective.

Related operations:

1. Create an AWS account, get the access key and security key.
2. Build two different buckets in your AWS S3, set the policies and permissions.
3. Install 'DOTENV' on VSCode (if your IDE is VSCode).
4. Create a .env file containing your AWS access key and security key at the same folder root of server.js.
5. Update bucket name and region in the files of services folder.

Tips: prepare a valid visa card for register.

Setup

.env

```
S3_ACCESS_KEY=<your access key>
S3_SECRET_ACCESS_KEY=<your secret key>
```

policies in AWS bucket

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicRead",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::saltyfish/*" //bucket name
        }
    ]
}
```

img-upload.js

```
aws.config.update({
    secretAccessKey: ACCESS_KEY,
    accessKeyId: ACCESS_ID,
    region: 'ap-southeast-2' //your region
})
const s3 = new aws.S3();
```

Services

Please refer to **img-upload.js**, **pdf-upload.js** in github repository.

US1 - Forget Password

Written by Zihan Ye

Node.js provides a module named nodemailer, which allows to send email easily.

If a client input his email in the Forget Email blank, the system will firstly get the password of this email from mangodb, then send it to the client's email.

Client Components

login.jsx

Controller

controller.js

Router

router.js

US2: Google/Facebook Login (Backend)

Written by Jiaxin Mo Xiaoyue LIU

1. Google Login

React has a package called as react-google-login. It provides a simple mechanism to add the Google login to backend authentication.

Firstly, we applied for an OAuthClientID from Google developer console. This ClientID was implemented to get the access_token when the user had a successful login. We built this token into our "jwtToken". Then some profile details objects of the user were accessed. We saved Google name as our USER name, email as email, and googleId as password, which were the neccessary information for our ePortfolio USER account. If the Google login is used for the first time, a new profile will be established, or an error message of "Email already registered" is sent. Otherwise, the profile page is loaded. When the token gets expired, it needs to refresh and generate a new token.

Please refer to these files in github repository.

Client components

login.jsx

Client actions

authActions.js

Controller

controller.js

Routes

router.js

---

2. Facebook Login

Facebook login also has a corresponding package in react, which is "react-facebook-login". It provides the necessary functions to establish a facebook connection. Unlike google login, facebook does not support the use of Facebook login service for websites whose domain name is http, because they think it is risky for facebook users to log in on such a website. Therefore, since we are using localhost, we need to change it to https in server.js.

First, we applied for a facebook developer account and obtained the app id, and used the app id to obtain the user's email, username and id from facebook. When a user logs in using facebook, the email, id and username provided by our facebook are used as the login email, password and name of the emailfolio website to create a new user. It should be noted that if the user has registered with the same email address before logging in to the e-folio website with facebook, when the user logs in with the corresponding facebook account, it will display "Email already registered" and cannot log in.

Please refer to these files in github repository.

Client components

login.jsx

Client actions

authActions.js

Controller

controller.js

Server

server.js

Routes

router.js

US3: Select section (Backend)

Written by Xiaoyue LIU

When the user clicks on the checkbox corresponding to each section, the system will record the display status of the section and store it in the database. Each section has corresponding attributes. When the user logs in again or refreshes, the data of the corresponding section in the database will be read again and set to the section and checkbox. The checkbox will also determine whether defaultchecked is required according to the corresponding section status in the database.

Component

profile.jsx

Public View Component

publicprofile.jsx

Routes

router.js

Controller

controller.js

US4: Reset Page (Backend)

Written by Jiaxin Mo

First, User account information is mounted from USER database with current login user ID. We set user`s name and email as default information shown in form. User can change any part of name, email or password, but not requiring all of them. When the user submit changes, it will push the update to both USER and PROFILE database with user ID.

Please refer to these files in github repository.

Components

reset.jsx

Routes

user.js

Controller

controller.js

US5 & US8: Download/delete Transcript File & Export ePortfolio (Backend)

Written by Jiaxin Mo

1. Download/delete Transcript File

The storage of the static transcript files in our app is based on AWS S3. When a user uploads a file to S3 bucket through the app, and S3 will generate a storage URL for it, which is saved in MongoDB for download purpose. The file must be a PDF format. The transcript URL is mounted from PROFILE database with user ID. If the transcript exists, the URL will be shown on both profile and public page. Users can click it, and download the file in a new blank page.

We set a "Delete" button for removing the file. If the user clicks on it, it will delete the specific object in S3 and clear the transcript item in MongoDB.

Please refer to these files in github repository.

Components

profile.jsx

publicprofile.jsx

Services

file-delete.js

2. Export ePortfolio

Exporting Portfolio utilizes Window print() method. When clicking on the "Print" button, it will open the print dialog of the browser. Users can modify the setting and print the current document or save as a PDF. We apply to define the contents to be printed.

Components

publicprofile.jsx

US7: Contact with user

In the public profile, if a user enter some message and clicks the submit button, the message would be sent to the profile owner from our official email account.

We use nodemailer to implement the sending email function. We suggest that the user should include his own contact information in the message.

Client Components

login.jsx

Controller

controller.js

Router

router.js