ĐẠI HỌC BÁCH KHOA HÀ NỘI

Trường Công nghệ thông tin và Truyền thông

BÁO CÁO

Nghiên cứu tốt nghiệp 2

Sinh viên thực hiện: Trần Tiến Ngọc

Mã số sinh viên: 20205009

Lớp Việt-Nhật 01 – K65

Giảng viên hướng dẫn: TS. Nguyễn Tuấn Dũng

Hà Nội, 1-2024

Nội dung

1.Cấu trúc một trang web

Về mặt kỹ thuật không nói đến phần thiết kế hay test và bảo trì: em thấy cần có các yếu tố để đưa 1 sản phẩm web lên mạng gồm có:

theo mô hình client server sẽ gồm có là client và server:

Front end(client-side): Là phần tạo giao diện trên trình duyệt, sử dụng các HTML, CSS, JS để lập trình front end: các framework hỗ trợ: AngularJS, ReactJS và VueJS

Back end(server side): xử lý logic giao tiếp giữa trình duyệt và CSDL thường sử dụng: Java, PHP, Ruby, Python và NodeJS các framework hỗ trợ: Laravel, Ruby on Rails, Django và ExpressJS.

Database: Cơ sở dữ liệu là nơi lưu trữ các thông tin và dữ liệu của sản phẩm web, thiết kế và quản lý cơ sở dữ liệu, xây dựng các câu truy vấn.

DevOps: Chuẩn hóa môi trường từ local đến product, deploy sản phẩm.

2. Reactjs: làm việc với hook

1. useRef

Function trả về một object với thuộc tính current bằng giá trị được truyền vào. Object này có thể thay đổi và tồn tại xuyên suốt vòng đời của component. Có tác dụng như một biến toàn cục sẽ không bị reset sau khi component re-render.

```
function App() {
  const [count, setCount] = useState(60)
  let timerId = useRef()
  const h1ref = useRef()
  console.log(h1ref.current)
  const handleStart = () => {
    timerId.current = setInterval(() => {
      setCount(pre => pre - 1)
    }, 1000)
  }

  const handleStop = () => {
    clearInterval(timerId.current)
  }



  return (
    <div>
      <h1 ref={h1ref}>{count}</h1>       You, now • Uncommitte
      <button onClick={handleStart}>start</button>
      <button onClick={handleStop}>stop</button>
    </div>
  );
}
```

Chức năng thứ 2 của useRef là tham chiếu tới một element ví dụ như có thể lấy ra thẻ h1 với h1ref

2. useCallback

```
const [count, setCount] = useState(0);
const [countOther, setCountOther] = useState(0);

const increase = () => setCount(count + 1);
const decrease = () => setCount(count - 1);

const increaseOther = () => setCountOther(countOther + 1);
const decreaseOther = () => setCountOther(countOther + 1);

return (
  <>
    <div>Count: {count}</div>
    <button onClick={increase}>+</button>
    <button onClick={decrease}>-</button>

    <div>Count other: {countOther}</div>
    <button onClick={increaseOther}>+</button>
    <button onClick={decreaseOther}>-</button>
  </>
)
```

useCallback sử dụng để tối ưu hoá quá trình render trong function component. Ví dụ như trong đoạn code trên, chỉ cần 1 thành phần counter thay đổi thì cả 4 hàm viết bên trên đều bị khởi tạo lại.

```
const increase = useCallback(() => setCount(count + 1), [count]);
const decrease = useCallback(() => setCount(count - 1), [count]);

const increaseOther = useCallback(() => setCountOther(countOther + 1), [countOther]);
const decreaseOther = useCallback(() => setCountOther(countOther + 1), [countOther]);
```

Sử dụng useCallback, đầu vào gồm 2 đối số: function, mảng dependance. Mỗi khi dependance thay đổi thì mới gọi hàm.
useCallback chỉ được khuyến khích sử dụng nếu trong component có các hàm xử lí phức tạp , tốn nhiều tài nguyên

3. useMemo

Truyền vào 2 tham số 1 hàm và các đối số của hàm đó ở trường thứ 2. Điều này làm cho useMemo có thể không phải tính toán lại hàm nếu các tham số không đổi so với lần gọi hàm trước.

```
const [count, setCount] = useState(0);
const [wordIndex, setWordIndex] = useState(0);

const words = ["hey", "this", "is", "cool"];
const word = words[wordIndex];

const computeLetterCount = word => {
  let i = 0;
  while (i < 1000000000) i++;
  return word.length;
};

const letterCount = useMemo(() => computeLetterCount(word), [word]);
```

4. useReducer

Là một bản nâng cấp so với useState, trong trường hợp nhiều action có thể làm thay đổi state theo những giá trị khác nhau ta có thể sử dụng useReducer:
Để sử dụng đc useReducer ta cần viết trước một hàm reducer

```
const reducer = (state, action) => {
  switch (action.type) {
    case "COMPLETE":
      return state.map((todo) => {
        if (todo.id === action.id) {
          return { ...todo, complete: !todo.complete };
        } else {
          return todo;
        }
      });
    default:
      return state;
  }
};
```

Hàm này sẽ có 2 tham số là state và action. Qua đó dựa vào action.type để quyết định return về một state cụ thể.
Ví dụ như trong đoạn code trên sẽ cần truyền vào action là 1 object có 2 thuộc tính {type,id}. Tác dụng là set lại trạng thái cho todo có id là action.id.
Tiếp theo ta sẽ sử dụng hook useReducer như sau:

```
const [todos, dispatch] = useReducer(reducer, initialTodos);
const handleComplete = (todo) => {
  dispatch({ type: "COMPLETE", id: todo.id });
};
return (
  <>
    {todos.map((todo) => (
      <div key={todo.id}>
        <label>
          <input
            type="checkbox"
            checked={todo.complete}
            onChange={() => handleComplete(todo)}
          />
          {todo.title}
        </label>
      </div>
    ))}
  </>
)
```

Tương tự như useState tuy nhiên 2 tham số đầu vào của useReducer là hàm reducer và giá trị khởi tạo ban đầu. Hàm dispath để thực hiện hàm reducer với action đầu vào tương ứng

3. Nodejs với MongoDB

1. Định nghĩa

1. Định nghĩa:Node.js là một môi trường thực thi cho JavaScript ở phía server, được cung cấp bởi V8 JavaScript engine của Chrome.

2. NPM

Node package manager: là một kho lưu trữ các libs, packages của hệ sinh thái node js. Hỗ trợ tối đa cho việc install, uninstall, update, control version các packages trong dự án NodeJs.

1 số câu lệnh cơ bản:

Install package.json dependencies	npm install
Install a package	npm i <package>
Uninstall a package	npm un <package>
Update a package	npm up <package>
List globally installed packages	npm list -g –depth=0
Uninstall global package	npm -g uninstall <name>
Upgrade npm on Windows	npm-windows-upgrade
Update global packages	npm outdated -g –depth=0
	npm update -g <package> <package> <package>
List available scripts to run	npm run
Update npm	npm install -g npm@latest
Installed version	npm list

3. Mô hình MVC

1. Khái niệm:

Mô hình Model-View-Controller (MVC) là một mẫu kiến trúc phân tách một ứng dụng thành ba thành phần logic chính Model, View và Controller. Do đó viết tắt MVC. Mỗi thành phần kiến trúc được xây dựng để xử lý khía cạnh phát triển cụ thể của một ứng dụng. MVC tách lớp logic nghiệp vụ và lớp hiển thị ra riêng biệt.

1. View: client là nơi mà người dùng cuối nhìn thấy và thao tác, qua đó tương tác với ứng dụng.
2. Controller: phần xử lý, với những tương tác của người dùng, bộ phận này sẽ tiến hành xử lý logic
3. Model: lưu trữ dữ liệu và logic liên quan: truy vấn, xử lý dữ liệu:,...

4. MongoDB

4.1 Các thuật ngữ hay sử dụng

- document : là đơn vị cơ bản nhất để lưu trữ dữ liệu trong mongodb. Tương ứng với một record trong các kiểu dữ liệu quan hệ. Một tài liệu được lưu bằng đối tượng JSON gồm các cặp key, value.

```
{
    "_id": "6108f386b1b70f6a7a3d3a8c",
    "name": "John",
```

```
    "age": 30,
    "address": {
    "street": "123 Main St",
    "city": "New York",
    "state": "NY",
    "zip": "10001"
    }
}
```

1. Collection: một nhóm các document tương ứng với một table trong cơ sở dữ liệu quan hệ. Được đặt tên và lưu trữ các document có cùng cấu trúc.
2. _id: một trường bắt buộc trong document, có thể hiểu là khóa chính. Nếu thêm mới document vào mongodb sẽ được sinh tự động một _id đại diện và là duy nhất trong csdl
4.2 Kết nối và truy vấn trong mongodb sử dụng mongoose
   - Connect:

```
const connectDatabase = async () => {
    try {
        await mongoose.connect("mongodb://localhost:27017/StudyControl")
        console.log("connected")
    }
    catch (error) {
        console.error(error)
    }
}
connectDatabase()
```

- Tạo collection bằng Schema:

```
const UserSchema = new Schema({
    username: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true
    },
    createAt: {
        type: Date,
        default: Date.now
    }
})
module.exports = mongoose.model('users', UserSchema)
```

- Tạo User và viết vào Database:

```
const newUser = new User({
    username: 'Ngoc',
    password: 'ngocars2002'
});

const writeDB = async () => {
    await User.create(newUser);
}
writeDB()
```

- Tìm kiếm

```
const findDB = async () => {
    const findItems = await User.find({})
    console.log(findItems);
}
```

- Update:

```
User.updateOne(
    { username: 'Ngoc', password: 'ngocars2002' },
    {
        password: 'chubedan'
    }
)
    .then(data => console.log(data))
    .catch(error => console.error(error))
```

Parameter:

{}:filter

{}:update

- Delete

```
const deleteOne = async () => {
    await User.deleteOne({ username: 'Ngoc' })
}

deleteOne()
```

4. Aggregate Populute với MongoDB

1. Aggregate

Là truy vấn nâng cao trong mongodb cho phép truy vấn tương tự như việc kết nối nhiều bảng trong SQL, tương tự GROUP BY trong SQL.

Xử lý dựa trên aggregate pipeline

Một số Operation cơ bản trong aggregate:

$project : chỉ định các field mong muốn truy vấn.

$match : chọn document mong muốn truy vấn.

$limit: giới hạn số lượng document

$skip : bỏ qua document nhất định

$group: nhóm các document theo điều kiện nhất định

$sort: sắp xếp document

$unwind : thực hiện thao tác mở rộng trên một mảng , tạo một ouput document cho mỗi giá trị trong mảng đó

$out : ghi kết quả sau khi thực hiện trên pipeline vào một collection.

Ví dụ:

```
const result = await User.aggregate([
        {
            $group: {
                _id: "$age",
                count: { $sum: 1 }
            }
        }, {
            $limit: 5
        },
        {
            $project: {
                _id: 0,
                age: "$_id",
                count: 1
            }
        },
        { $sort: { age: 1 } }
```

```
    ])
```

2. Populate

- Saving ref

```
const newCard = new Card({
    _id: new mongoose.Types.ObjectId(),
    card_id: 2911,
    name: "Vietcombank"
})

const newUser = new User({
    username: 'ngoc123',
    password: '123',
    age: 20,
    salary: 40000,
    card: newCard._id
})

const writeDb = async () => {
    await Card.create(newCard);
    await User.create(newUser);
}
writeDb()
```

- Population

```
const populateDb = async () => {
    const populateData = await User.find({ age: 20 })
        .populate('card')
    console.log(populateData)
}
populateDb()
```

- Population nhiều lớp

```
const populateDb = async () => {
    const populateData = await User.find({ age: 20 })
        .populate({
            path: 'card',
            populate: { path: 'chi_nhanh' }
        })

}
populateDb()
```

5. Express và các vấn đề liên quan

1. Routing

app.METHOD(path, handle)

```
const app = express()
```

method: get, post, put, delete,…

handle: Hàm xử lý khi đi đến đúng router

```
app.get('/', (req, res) => {
  res.send('Hello World!')
})
```

2. Restful API

Tiêu chuẩn dùng trong thiết kế API cho các ứng dụng web

- API : Tương tác giữa hai hay nhiều thành phần (client, server) để lấy dữ liệu từ bên này gửi cho bên kia
- REST: là dạng chuyển đổi dữ liệu, một kiểu kiến trúc để viết API. Thay vì gửi URL, REST gửi một yêu cầu HTTP như GET, POST, DELETE, vv đến một URL để xử lý dữ liệu.
- Chức năng quan trọng nhất của **REST** là quy định cách sử dụng các HTTP method (như GET, POST, PUT, DELETE...) và cách định dạng các URL cho ứng dụng web để quản các resource.
- Authentication request: RESTful API không sử dụng session và cookie, nó sử dụng một access_token với mỗi request

```
router.get('/users', getUsers)
router.post('/create', createUser)
router.put('/update/:id', updateUser)
router.delete('/delete/:id', deleteUser)
```

3. Method (CRUD)

- GET

```
router.get('/users', getUsers)
```

```
module.exports.getUsers = async (req, res) => {
    const users = await UserModel.find({})
    res.send(users)
}
```

Method GET(Read): Truy suất tài nguyên từ DB và trả về response

- POST

```
router.post('/create', createUser)
```

```
module.exports.createUser = async (req, res) => {
    const newUser = req.body;
    console.log(req.body)
    try {
        await UserModel.create(newUser);
        res.status(201).send({ message: 'User created' });
    }
    catch (err) {
        console.log("something went wrong")
        res.status(500).send({ message: 'Something went wrong' });
    }
}
```

Method POST(Create)

- PUT

```
router.put('/update/:id', updateUser)
```

```
module.exports.updateUser = async (req, res) => {
    const updateUser = req.body
    const { id } = req.params
    console.log()
    try {
        await UserModel.findByIdAndUpdate(id, updateUser)
        res.status(201).send({ message: 'User updated' });
    }
    catch (err) {
        console.log("something went wrong")
        res.status(500).send({ message: 'Something went wrong' });
    }
};
```

Method PUT(update)

- DELETE

```
router.delete('/delete/:id', deleteUser)
```

```
module.exports.deleteUser = async (req, res) => {
    const { id } = req.params
```

```
    try {
        await UserModel.findByIdAndDelete(id)
        res.status(201).send({ message: 'User deleted' });
    }
    catch {
        res.status(500).send({ message: 'Something went wrong' });
    }
}
```

Method DELETE

4. Middleware

Tuy ở trên nói rằng các method thường gửi dữ liệu đến DB tuy nhiên, sau khi nhận được request ta thường không cập nhật vào DB ngay mà sẽ tiến hành một số bước kiểm tra hoặc các thao các khác, vì vậy mà middleware được sử dụng.

Chức năng chính của middleware:

- Cần xác thực người dùng để quyết định xem họ có được phép truy cập đến route hiện tại hay không.
- Yêu cầu đăng nhập
- Chuyển hướng người dùng
- Thay đổi/chuẩn hoá các tham số
- Xử lý request đầu vào và response được tạo ra,...

```
1   const jwt = require('jsonwebtoken')
2
3   const verifyToken = (req, res, next) => {
4           const authHeader = req.header('Authorization')
5           const token = authHeader && authHeader.split(' ')[1]
6
7           if (!token)
8                   return res
9                           .status(401)
10                          .json({ success: false, message: 'Access token not found' })
11
12          try {
13                  const decoded = jwt.verify(token, process.env.ACCESS_TOKEN_SECRET)
14
15                  req.userId = decoded.userId
16                  next()
17          } catch (error) {
18                  console.log(error)
19                  return res.status(403).json({ success: false, message: 'Invalid token' })
20          }
21  }
22
23  module.exports = verifyToken
```

```
router.get('/', verifyToken, async (req, res) => {
        try {
                const user = await User.findById(req.userId).select('-password')
                if (!user)
                        return res.status(400).json({ success: false, message: 'User not found' })
                res.json({ success: true, user })
        } catch (error) {
                console.log(error)
                res.status(500).json({ success: false, message: 'Internal server error' })
        }
})
```

ở cuối hàm middleware nếu được chấp nhận gửi đi tiếp ta sẽ cần gọi hàm next() để đi tới những middleware kế tiếp hoặc function xử lý cuối cùng.

Middleware cũng có thể được sử dụng global với từ khóa use ở trong index.js

```
app.use(express.json())
app.use(express.urlencoded({ extended: true }))
app.use(cors());

app.use('/api', userRoute)
```

như ở đây các hàm bên trên sẽ chạy xử lý trước khi gọi API

5. Authentication and Authorization

Authentication	Authorization
Authentication xác nhận danh tính của bạn để cấp quyền truy cập vào hệ thống.	Authorization xác định xem bạn có được phép truy cập tài nguyên không.
Đây là quá trình xác nhận thông tin đăng nhập để có quyền truy cập của người dùng.	Đó là quá trình xác minh xem có cho phép truy cập hay không.
Nó quyết định liệu người dùng có phải là những gì anh ta tuyên bố hay không.	Nó xác định những gì người dùng có thể và không thể truy cập.
Authentication thường yêu cầu tên người dùng và mật khẩu.	Các yếu tố xác thực cần thiết để authorization có thể khác nhau, tùy thuộc vào mức độ bảo mật.

Authentication	Authorization
Authentication là bước đầu tiên của authorization vì vậy luôn luôn đến trước.	Authorization được thực hiện sau khi authentication thành công.
Ví dụ, sinh viên của một trường đại học cụ thể được yêu cầu tự xác thực trước khi truy cập vào liên kết sinh viên của trang web chính thức của trường đại học. Điều này được gọi là authentication.	Ví dụ, authorization xác định chính xác thông tin nào sinh viên được phép truy cập trên trang web của trường đại học sau khi authentication thành công.

6. Axios

Là một thư viện HTTP client, vì vậy cách viết tương đồng với restfulAPI

```
 const data = await axios.get(`${apiURL}/users`)
            console.log(data)
            setUserList(data.data)
const register = async (event) => {
        event.preventDefault()
        if (registerForm.password === registerForm.confirmPassword) {
            const response = await axios.post(`${apiURL}/create`, { username:
registerForm.username, password: registerForm.password }
            )
        }
        setUpdateUI(pre => !pre)
    }
```

```
const update = async (event) => {
        event.preventDefault()
        const response = await axios.put(`${apiURL}/update/${updateId}`,
{ username: registerForm.username, password: registerForm.password })
        setUpdateId(null)
        setUpdateUI(pre => !pre)
    }
```

```
const deleteUser = async (id) => {
        const response = await axios.delete(`${apiURL}/delete/${id}`)
        setUpdateUI(pre => !pre)
    }
```

Trong response, Axios trả về một promise để xử lý response object hoặc object error, response object thường có:

- data: body của response
- status: mã HTTP status code của response, vd 200 hay 404
- statusText: đoạn text của HTTP status
- headers: chứ header của request
- config: cấu hình của request
- request: đối tượng XMLHttpRequest (XHR)

6. Kết quả đạt được

- Nắm rõ quá trình tạo nên một trang web
- Nắm rõ mô hình client-server và MVC
- Nắm được cách code Javascript, React và Nodejs
- Học và có thể code một MERN app (Mongodb, Express, React, Nodejs)
- Phát triển được trang web đầu tư chứng khoán đơn giản