

---

# XEDIKE

Phó Nghĩa Văn

---



**CYBERSOFT**  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



**MYC**  **DER**

**timviec**  **it.com**

# ĐẶC TẢ PROJECT

## CÁC TÍNH NĂNG CƠ BẢN

1. Có 2 loại người cơ bản: passenger và driver (gọi chung là user)
2. User sẽ có một số thông tin cơ bản (email, password, name, phone,...)
3. Có chức năng đăng ký, đăng nhập, phân loại người dùng
4. Driver có ngoài những thông tin trên phải có thêm thông tin như license plate, car info, job, company, address, passportId ...
5. User và khách vãng lai có thể: xem danh sách user, xem thông tin một user (và thông tin chi tiết driver), danh sách chuyến đi, chi tiết một chuyến đi
6. Driver có quyền CRUD các chuyến đi và kết thúc các chuyến đi do mình tạo
7. Passenger có quyền book hoặc hủy book một chuyến đi
8. Passenger có quyền rate/comment tài xế đã đi cùng mình

# ĐẶC TẢ PROJECT

## CÁC TÍNH NĂNG MỞ RỘNG:

1. Có thêm user loại admin
2. Admin có quyền xem thống kê:
  - a. Tổng chi phí cho tất cả cuộc xe, chiết khấu, doanh thu, lợi nhuận, lợi nhuận ròng
  - b. Có bao nhiêu passenger, driver
  - c. Có tổng cộng bao nhiêu chuyến đi theo năm, tháng, ngày
  - d. Top Driver, top passenger đi được nhiều km nhất --> đặt ra chương trình tri ân, khuyến mãi, giảm giá, huy chương, ...
3. Tạo thêm các kênh chat (realtime với socket) (group user chat chung, group chat client, group chat driver, chat private 1vs1)

# CÁC BƯỚC THỰC HIỆN

## 1. DATABASE

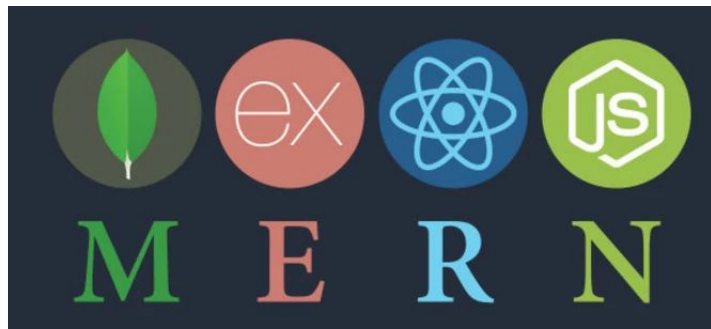
- Có những collection nào, mỗi collection có những field nào
- Mỗi quan hệ giữa các collection
- Dùng kỹ thuật nào để mô phỏng các mối quan hệ đó

## 2. BACKEND

- Tạo Schema/Model, kết nối đến database
- Thực hiện validation
- Cung cấp API (service) cho frontend

## 3. FRONTEND

- Thiết kế giao diện (UX/UI)
- Gọi service từ backend



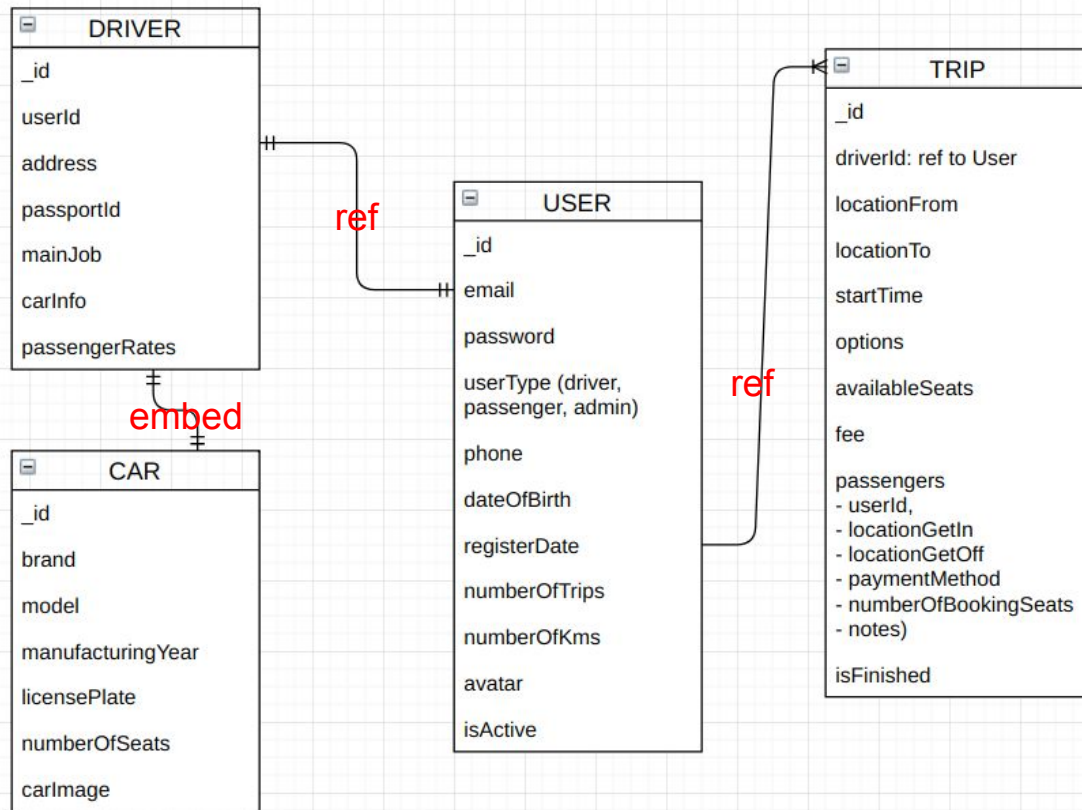
# DATABASE

- Có những collection nào, mỗi collection có những field nào ?
- Mỗi quan hệ giữa các collection ?
- Dùng kỹ thuật nào để mô phỏng các mối quan hệ đó ?

# MÔ HÌNH

Các đối tượng: User, Driver, Car, Trip

Phân loại người dùng:  
anonymous, passenger,  
driver, admin

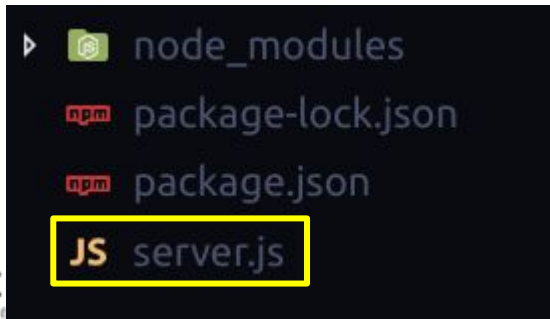


# BACKEND

- Tạo Schema/Model, kết nối đến database
- Thực hiện validation
- Cung cấp API (service) cho frontend

# KHỞI TẠO PROJECT

1. Khởi tạo project: **npm init**
2. Cài đặt các package cần thiết:  
express, mongoose, nodemon
3. Tạo file server.js
4. Thiết lập nodemon đến server.js



```
{  
  "name": "xedike",  
  "version": "1.0.0",  
  "description": "",  
  "main": "server.js",  
  "scripts": {  
    "start": "node server.js",  
    "server": "nodemon server.js"  
  },  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.16.4",  
    "mongoose": "^5.3.15",  
    "nodemon": "^1.18.8"  
  }  
}
```





# KHỞI TẠO PROJECT

1. Tạo server với express framework
2. Chạy server: **npm run server**

Chỉ định port tại terminal hoặc nhập port từ nơi host.  
(Trên linux/mac dùng "EXPORT",  
trên window dùng "SET")

```
export PORT=3000
```

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

const app = express();

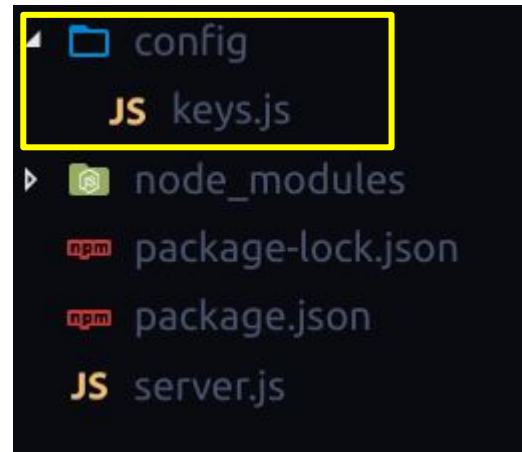
const port = process.env.PORT || 5000;
app.listen(port, () => {
  console.log(`Listening on port ${port}`)
})
```

```
> nodemon server.js

[nodemon] 1.18.8
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
Listening on port 5000
```

# KẾT NỐI DATABASE

1. Tạo file ./config/keys.js
2. Trong keys.js, export mongoURI



```
module.exports = {  
  ...  
  mongoURI: "mongodb://localhost:27017/xedike"  
}
```



CYBERSOFT  
ĐÀO TẠO CHUYÊN GIA L



PHUQUACK

university.com

# KẾT NỐI DATABASE

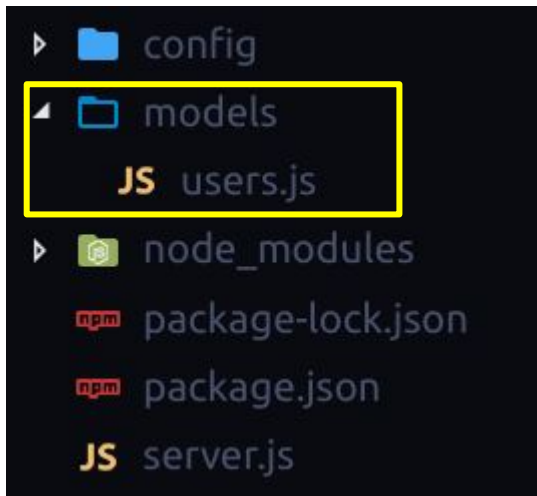
1. Import mongoURI
2. Connect to MongoDB

```
nodemon] restarting due to changes...  
nodemon] starting `node server.js`  
connected to MongoDB :))  
listening on port 5000
```



```
const express = require('express')  
const mongoose = require('mongoose');  
  
const {mongoURI} = require('./config/keys');  
  
mongoose.connect(mongoURI, {useNewUrlParser: true})  
  .then(console.log("Connected to MongoDB :))"))  
  .catch(console.log)  
  
const app = express();  
  
const port = process.env.PORT || 5000;  
app.listen(port, () => {  
  console.log(`Listening on port ${port}`)  
})
```

# TẠO MODEL USER



**CYBERSOFT**  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



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

const UserSchema = new mongoose.Schema({
  email: {type: String, required: true},
  password: {type: String, required: true},
  fullName: {type: String, required: true},
  userType: {type: String, required: true},
  phone: {type: Number, required: true},
  dateOfBirth: {type: Date, required: true},
  registerDate: {
    type: Date,
    default: new Date().getTime()
  },
  numberOfTrips: {type: Number},
  numberOfKms: {type: Number},
  avatar: {type: String},
  isActive: { type: Boolean, default: true }
})

const User = mongoose.model('User', UserSchema);
module.exports = {
  UserSchema, User
}
```

# BODY-PARSER

- Dùng để lấy dữ liệu từ form
- Là **Third-party middleware**
- Cài đặt:  
**npm install body-parser**
- Setup: trong **server.js**



```
const express = require('express')
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
```

```
const {mongoURI} = require('./config/keys');
```

```
mongoose.connect(mongoURI, {useNewUrlParser: true})
  .then(console.log("Connected to MongoDB :))"))
  .catch(console.log)
```

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

```
// middleware: body-parser
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
```

```
const port = process.env.PORT || 5000;
app.listen(port, () => {
  console.log(`Listening on port ${port}`)
})
```

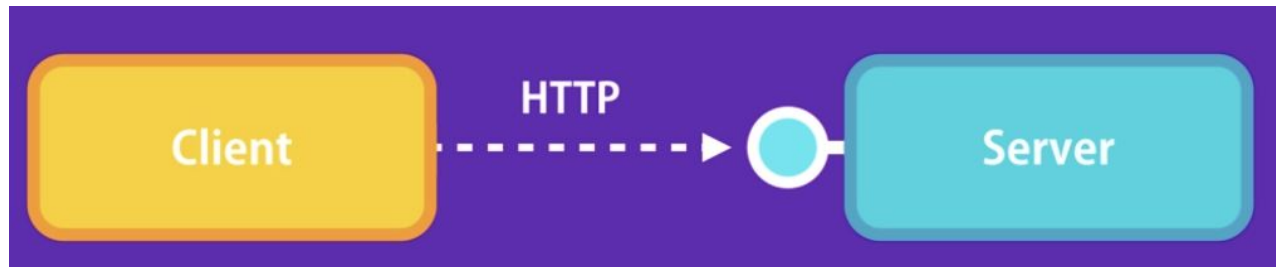


# REST API

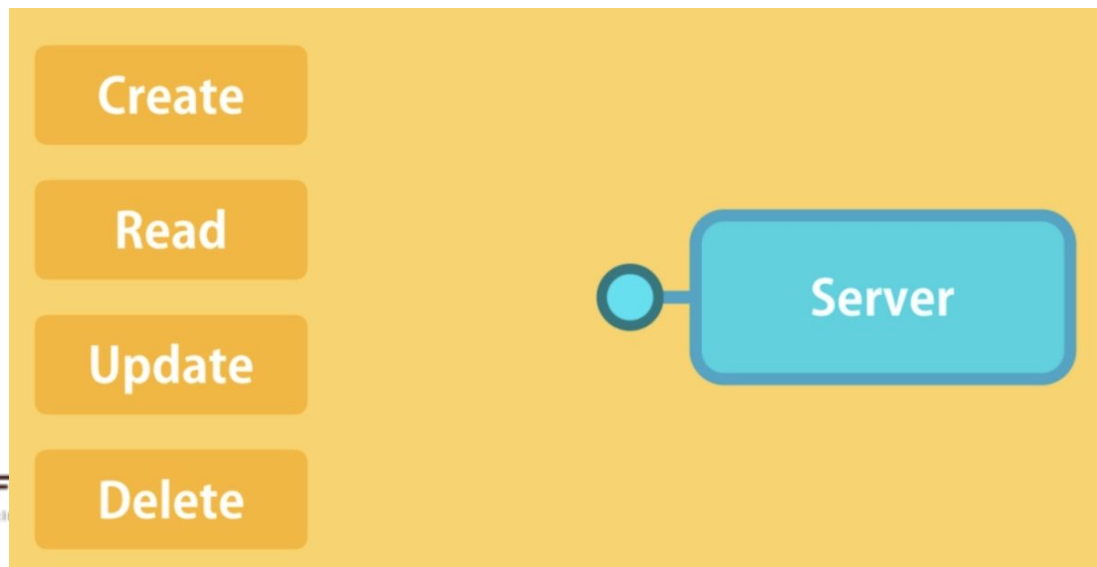
- API: **A**pplication **P**rogramming **I**nterface
- REST: **RE**presentational **S**tate **T**ransfer



# REST API



- API: **A**pplication **P**rogramming **I**nterface
- REST: **RE**presentational **S**tate **T**ransfer



# HTTP METHODS

## HTTP METHODS

```
GET /api/customers
GET /api/customers/1
PUT /api/customers/1
DELETE /api/customers/1
POST /api/customers
```

GET

POST

PUT

DELETE

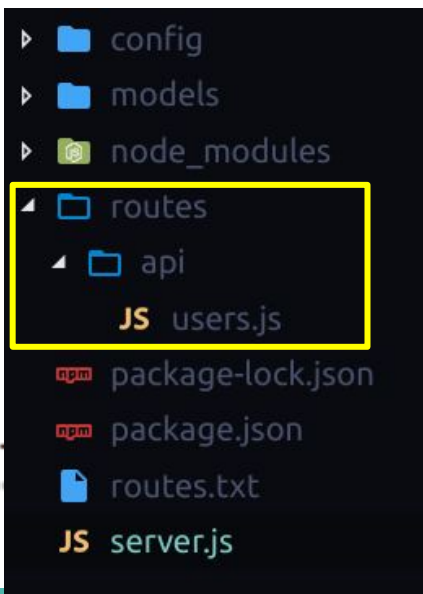


# CÁC API XÂY DỰNG

USERS	TRIPS
<p>POST /api/users/signup POST /api/users/signin GET /api/users/current GET /api/users/drivers/profile/:id POST /api/users/drivers/profile/:id</p>	<p>GET /api/trips POST /api/trips/create-trip PUT /api/trips/update-trip/:id DELETE /api/trips/remove-trip/:id  PUT /api/trips/book/:id PUT /api/trips/finish/:id PUT /api/trips/rate/:id</p>

# ROUTES CHO USER

1. Tạo cấu trúc thư mục như hình
2. Trong file users.js, khai báo router và export



```
const express = require('express');
const router = express.Router();

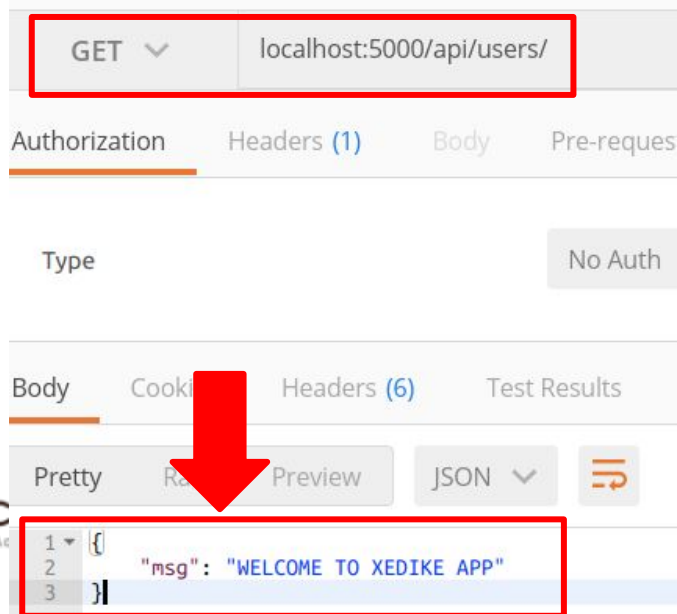
// load model
const User = require('../models/users');

router.get('/', (req, res) => {
  res.json({msg: 'WELCOME TO XEDIKE APP'})
})

module.exports = router;
```

# ROUTES CHO USER

1. Cấu hình routes trong **server.js**
2. Thực hiện test bằng **postman**



```
const express = require('express')
const mongoose = require('mongoose');
const bodyParser = require('body-parser');

const {mongoURI} = require('./config/keys');

mongoose.connect(mongoURI, {useNewUrlParser: true})
  .then(console.log("Connected to MongoDB :))"))
  .catch(console.log)

const app = express();

// middleware: body-parser
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

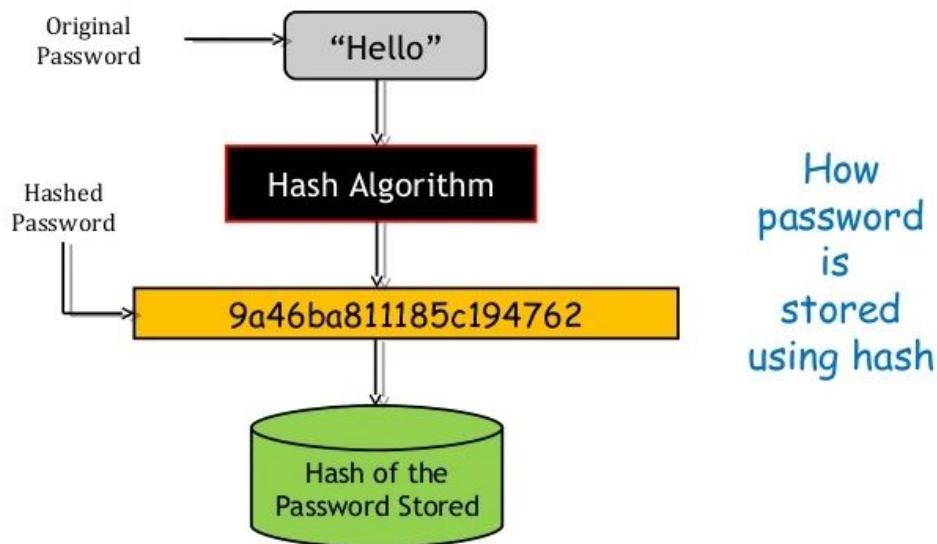
// routes
app.use('/api/users', require('./routes/api/users'))

const port = process.env.PORT || 5000;
app.listen(port, () => {
  console.log(`Listening on port ${port}`)
})
```

# REGISTER

1. Thực hiện tính năng register (tạo mới user)
2. Yêu cầu: password phải được hash trước khi lưu vào database tránh hacker dùng kỹ thuật **dictionary** hoặc **brute-forcing**

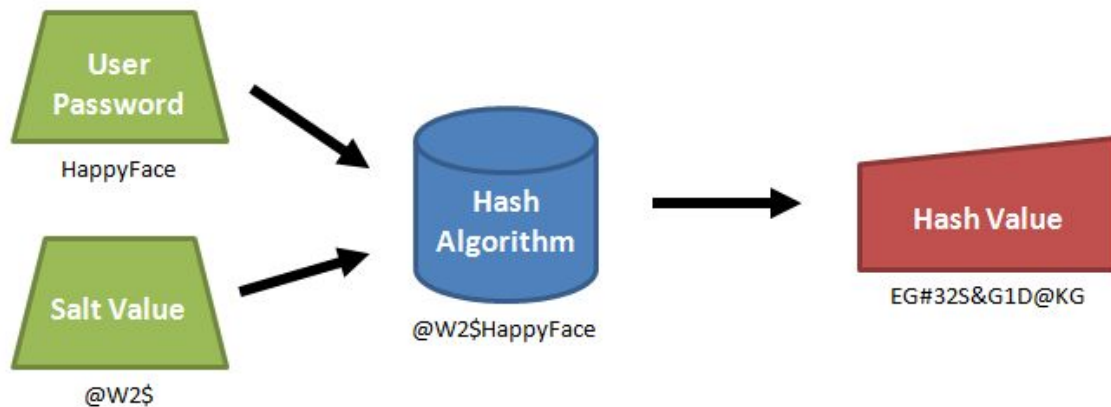
## Application: Hashing Password



# REGISTER

## 1. Một số cơ chế hash password:

- Không hash
- Hash với MD5 hoặc SHA1
- Hash với FIXED SALT hoặc PER USER SALT
- Hash với salt được tạo từ bcrypt. Ưu điểm: slow, config salt, random



# REGISTER

1. Cài đặt bcryptjs: **npm install bcryptjs**
2. Tạo route POST /api/users/register

```
//route:    /api/users/register  
//desc:     register an user  
//access:   public
```

Mô tả route

```
router.post('/register', (req, res) => {  
  const { email, password, fullName, userType, phone, dateOfBirth } = req.body  
  User.findOne({ $or: [{email}, {phone}] }) Kiểm tra email/phone đã tồn tại hay chưa  
    .then(user => {  
      if (user) return res.status(400).json({ error: "Email or Phone exists" })  
      Nếu không tồn tại thì tạo một instance mới  
      const newUser = new User({  
        email, password, fullName, userType, phone, dateOfBirth  
      })  
    })  
})
```

# REGISTER

1. Import bcryptjs:
2. **genSalt()**: tạo ra chuỗi salt để gắn vào password trước khi hash
3. **hash()**: hash chuỗi password-salt

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

```
    bcrypt.genSalt(10, (err, salt) => {  
        bcrypt.hash(newUser.password, salt, (err, hash) => {  
            if(err) throw err;  
            newUser.password = hash;  
            newUser.save()  
                .then(user => res.status(200).json(user))  
                .catch(console.log)  
        })  
    })  
})
```



# REGISTER

POST localhost:5000/api/users/register

Authorization Headers (1) Body Pre-rec

☐ form-data ☒ x-www-form-urlencoded

Key	
<input checked="" type="checkbox"/> email	hackagon@gmail.com
<input checked="" type="checkbox"/> password	hackagon
<input checked="" type="checkbox"/> fullName	Phó Nghĩa Văn
<input checked="" type="checkbox"/> userType	passenger
<input checked="" type="checkbox"/> phone	0963228338
<input checked="" type="checkbox"/> dateOfBirth	02-02-1822
New key	Value

```
{  "registerDate": "2018-12-12T07:36:06.316Z",  "isActive": true,  "_id": "5c10ba745c59e002e81b26c6",  "email": "hackagon@gmail.com",  "password": "$2a$10$czHua7NY0QaYx051BNrgsu0QrD/HuazSncGei9gQy/65XKJ.o1",  "fullName": "Phó Nghĩa Văn",  "userType": "passenger",  "phone": 963228338,  "dateOfBirth": "1822-02-01T17:00:00.000Z",  "__v": 0}
```

Chuỗi password-salt được hash

R

timviec  it.com



# LOGIN

1. Thực hiện tính năng login
2. So sánh password được nhập với password trong chuỗi hash password-salt
3. Nếu so sánh trùng khớp, trả về chuỗi **token** để xác nhận rằng user đã đăng nhập vào hệ thống



# LOGIN

```
//route:    /api/users/login  
//desc:     login  
//access:   public
```

Mô tả route

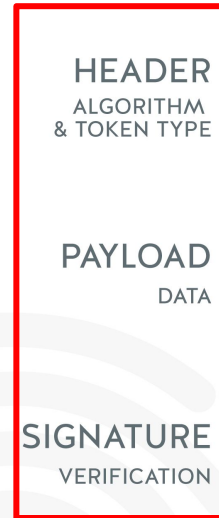
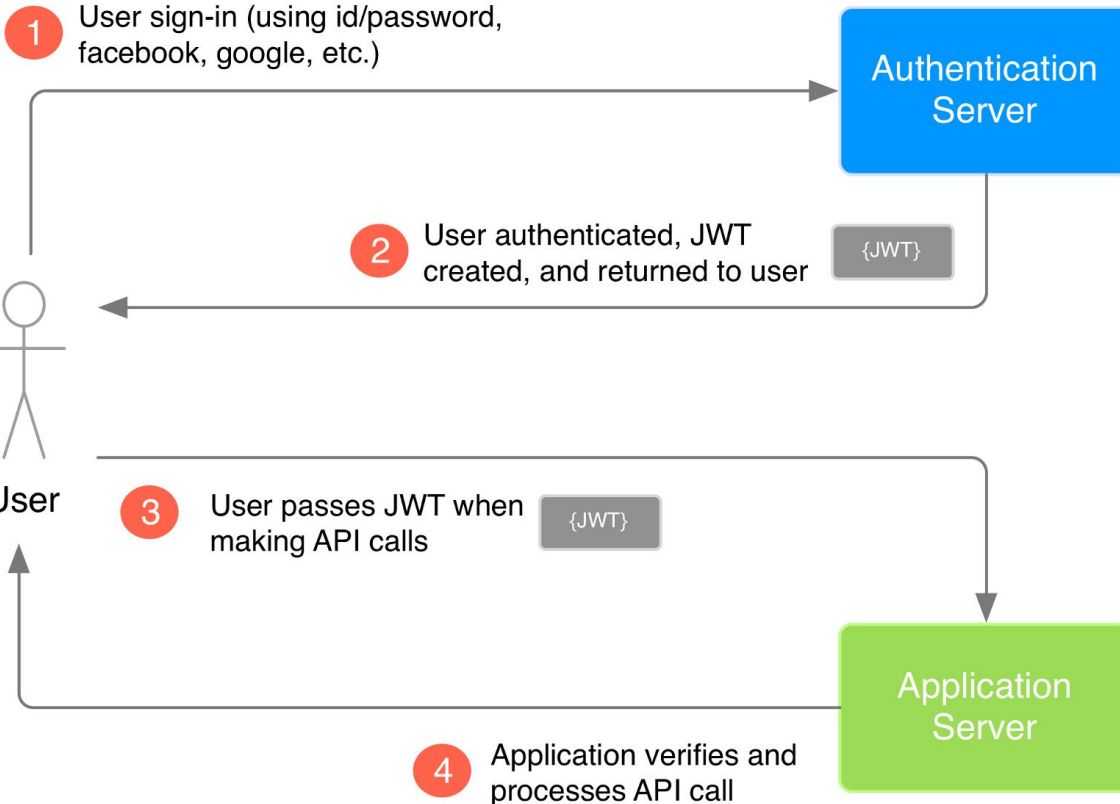
```
router.post('/login', (req, res) => {  
  const {email, password} = req.body;  
  User  
    .findOne({email})  
    .then(user => {  
      if(!user) return res.status(404).json({email: 'Email does not exist'})  
      bcrypt.compare(password, user.password)  
        .then(isMatch => {  
          if(!isMatch) res.status(400).json({password: "Password incorrect"})  
          res.status(200).json({msg: "Success"})  
        })  
    })  
})
```

So sánh password nhập vào và password được hash



# LOGIN - JSON WEB TOKEN

JWT  
JSON WEB TOKEN



```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

+

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}
```

+

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload), secretKey)
```

# LOGIN - JSON WEB TOKEN

1. Cài đặt jsonwebtoken:  
**npm install jsonwebtoken**
2. Do cấu trúc jsonwebtoken khá đơn giản và phổ biến nên rất dễ bị decode. Do đó, **không nên để những thông tin nhạy cảm lên tại payload**



```
const payload = {
  id: user._id,
  email: user.email,
  fullName: user.fullName,
  userType: user.userType
}
jwt.sign(
  payload,
  "secretKey", secretOrKey: string nào
  (err, token) cũng được => {
    res.status(200).json({
      success: true,
      token: 'Bearer '+token
    })
  })
)
```

# LOGIN - JSON WEB TOKEN

POST localhost:5000/api/users/login

Authorization

Headers (2)

Body

Pre-request Script

Tests

☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary

	Key	Value
<input checked="" type="checkbox"/>	email	hackagon@gmail.com
<input checked="" type="checkbox"/>	password	hackagon
	New key	Value



```
{
  "success": true,
  "token": "BearereyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVjMTBiYTc0NWUwMDJlODFmZjZjNiIsImVtYWlsIjoiaGFja2FnbnI2NSAzZ1haWwY29tDQ2MDMzMd9.Jl5Ne0sAWHcaMZtBCcjEZsiTzfZN-Ofz1ms3acft0c"
}
```

# AUTHENTICATION

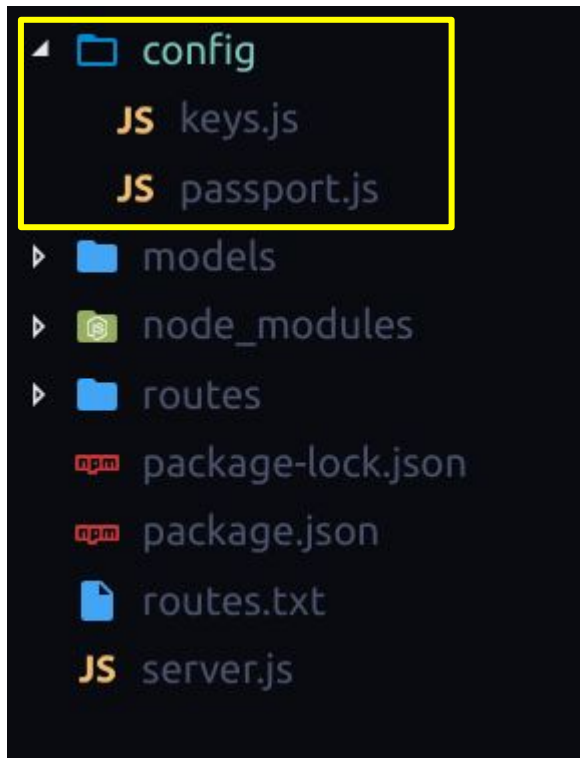
1. **Passport** một trong những module phổ biến nhất của Nodejs hỗ trợ bạn authentication, bản chất là một **middleware**
2. Cài đặt: **npm install passport passport-jwt**
3. Tại server.js, cấu hình middleware passport

```
const passport = require('passport')
```

```
// Middleware: passport  
app.use(passport.initialize());  
require('./config/passport')(passport);
```

# AUTHENTICATION

Kiểm tra xem passport mà browser cung cấp có khớp với dữ liệu được lưu trong database không. Nếu khớp sẽ trả về callback done(null, user). Done lấy dữ liệu từ payload trả về cho passport.authenticate()



```
const {User} = require('../models/users');
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;

const opts = {};
opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
opts.secretOrKey = 'secretKey'; Key tương tự key jwt phần login

module.exports = (passport) => {
  passport.use(
    new JwtStrategy(opts, (jwtPayload, done) => {
      User.findById(jwtPayload.id)
        .then(user => {
          if(user) return done(null, user)
          return done(null, false)
        })
        .catch(console.log)
    })
  )
}
```



# AUTHENTICATION

1. Tạo route GET  
/api/users/current
2. Lấy dữ liệu từ payload  
(trả về cho done) để  
hiển thị cho client

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

```
//route:    /api/users/current  
//desc:     login  
//access:   private (passenger, driver, admin)  
router.get(['/current'],  
  passport.authenticate('jwt', {session: false}),  
  (req, res) => {  
    res.status(200).json({  
      id: req.user._id,  
      email: req.user.email,  
      fullName: req.user.fullName,  
      userType: req.user.userType,  
    })  
  })  
})
```



# UPLOAD AVATAR

1. Sử dụng package **multer**: `npm install multer` để upload hình ảnh

```
const multer = require('multer')

const storage = multer.diskStorage({
  destination: function(req, file, cb){
    cb(null, './uploads/')
  },
  filename: function(req, file, cb){
    let type = "";
    if(file.mimetype === "application/octet-stream") type = ".jpg"
    cb(null, new Date().toISOString() + "-" + file.originalname + type)
  }
})

const upload = multer([storage])
```

**destination:** tạo folder lưu trữ hình ảnh

**filename:** đặt tên cho hình ảnh được upload. Trong demo này, đặt tên theo cú pháp: time + filename + type (định dạng jpg/jpeg/png)



# UPLOAD AVATAR

1. Tạo API để upload avatar
2. Sử dụng **upload.single** như middleware

```
//route:    /api/users/upload-avatar
//desc:     upload an avatar
//access:   private (passenger, driver, admin)
router.post('/upload-avatar',
  upload.single('avatar'),
  passport.authenticate('jwt', { session: false }),
  (req, res) => {
    console.log(req.file)
    User
      .findById(req.user.id)
      .then(user => {
        user.avatar = req.file.path
        return user.save()
      })
      .then(user => res.status(200).json(user))
      .catch(console.log)
  })
})
```

# UPLOAD AVATAR

1. Cấu hình middleware trong server.js để có thể chạy hiển thị được hình ảnh trên trình duyệt

POST localhost:5000/api/users/upload-avatar/5c10ba745c59e002e81b26c6

Authorization Headers (1) Body Pre-request Script Tests

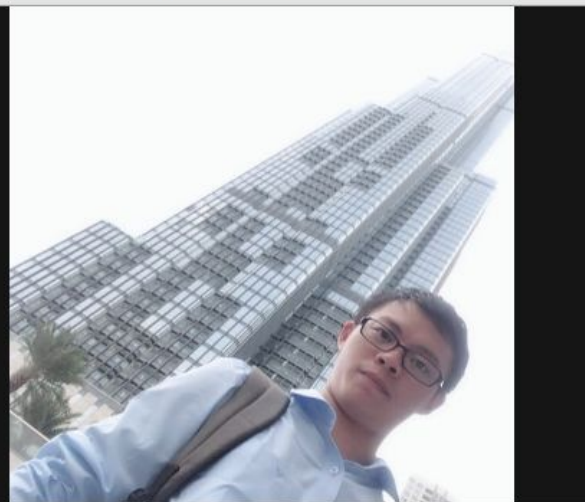
☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary

Key	Value
<input checked="" type="checkbox"/> avatar	File Choose Files hackagon.jpeg

localhost:5000/uploads/2018-12-14T07:31:46.683Z-hackagon.jpeg

Frontend CyberSoft GV ReactJS RoboLab NodeJS JAVA

```
// Middleware: upload image
app.use(['/uploads', express.static('uploads')])
```



# USER

## CÁC BẠN TỰ HOÀN THIỆN CÁC PHẦN SAU:

### 1. GET /api/users

- a. Mô tả: Lấy danh sách users
- b. PUBLIC: tất cả mọi người dùng (kể cả người dùng vắng lai) đều có thể access

### 2. GET /api/users/:userId

- a. Mô tả: Lấy thông tin chi tiết của một user
- b. PUBLIC: tất cả mọi người dùng (kể cả người dùng vắng lai) đều có thể access

### 3. POST/PUT /api/users/update

- a. Mô tả: sửa thông tin chi tiết của một user
- b. PRIVATE (chỉ có user đang đăng nhập vào hệ thống mới có thể sửa thông tin của họ)



# USER

CÁC BẠN TỰ HOÀN THIỆN CÁC PHẦN SAU:

## 4. POST/DELETE /api/users/delete

- a. Mô tả: delete một user
- b. PRIVATE (chỉ có user đang đăng nhập vào hệ thống mới có thể xóa tài khoản của họ)



# DRIVER

CÁC BẠN TỰ HOÀN THIỆN CÁC PHẦN SAU:

1. Tạo Schema/Model cho **Driver** (các thuộc tính lấy trong **slide 6**). Sử dụng kỹ thuật **referencing** để mô phỏng mối quan hệ giữa User và Driver
2. POST /api/users/drivers/create-profile
  - a. Mô tả: tạo thông tin chi tiết cho một driver
  - b. PRIVATE: chỉ có userType=driver mới được access
3. GET /api/user/drivers/profile/:userId
  - a. Mô tả: lấy thông tin chi tiết của một tài xế
  - b. PUBLIC: tất cả mọi người dùng (kể cả người dùng vắng lai) đều có thể access



# DRIVER

CÁC BẠN TỰ HOÀN THIỆN CÁC PHẦN SAU:

4. PUT/POST /api/users/drivers/update-profile

- a. Mô tả: update profile của driver
- b. PRIVATE: chỉ có userType=driver + đang đăng nhập mới được access

5. DELETE/POST /api/users/drivers/delete-profile

- a. Mô tả: delete profile của driver
- b. PRIVATE: chỉ có userType=driver + đang đăng nhập mới được access



# DRIVER

CÁC BẠN TỰ HOÀN THIỆN CÁC PHẦN SAU:

6. Tạo Schema/Model cho **Car** (các thuộc tính lấy trong **slide 6**). Sử dụng kỹ thuật **embedding** để mô phỏng mối quan hệ giữa Driver và Car
7. GET /api/users/drivers/:driverId/cars
  - a. Mô tả: lấy danh sách xe hơi của một tài xế
  - b. PUBLIC: tất cả mọi người dùng (kể cả người dùng vắng lai) đều có thể access
8. POST /api/users/drivers/add-car (POST)
  - a. Mô tả: Thêm xe hơi vào danh sách xe hơi của tài xế
  - b. PRIVATE: chỉ có userType=driver + đang đăng nhập mới được access





# DRIVER

CÁC BẠN TỰ HOÀN THIỆN CÁC PHẦN SAU:

9. POST/PUT /api/users/drivers/update-car/:carId
  - a. Mô tả: edit thông tin của xe hơi
  - b. PRIVATE: chỉ có userType=driver + đang đăng nhập mới được access
10. POST/DELETE /api/users/drivers/update-car/:carId
  - a. Mô tả: xóa xe hơi ra khỏi danh sách xe hơi của tài xế
  - b. PRIVATE: chỉ có userType=driver + đang đăng nhập mới được access



# TRIP

CÁC BẠN TỰ HOÀN THIỆN CÁC PHẦN SAU:

1. Tạo Schema/Model cho **Trip** (các thuộc tính lấy trong **slide 6**). Sử dụng kỹ thuật **referencing** để mô phỏng mối quan hệ giữa User và Trip
2. GET /api/trips/
  - a. Mô tả: Lấy danh sách chuyến đi
  - b. PUBLIC: tất cả mọi người dùng (kể cả người dùng vắng lai) đều có thể access
3. GET /api/trips/:tripId
  - a. Mô tả: tạo một chuyến đi
  - b. PRIVATE: chỉ có userType=driver + đang đăng nhập mới được access



# TRIP

CÁC BẠN TỰ HOÀN THIỆN CÁC PHẦN SAU:

4. POST /api/trips/

- a. Mô tả: thêm mới một chuyến đi
- b. PRIVATE: chỉ có userType=driver + đang đăng nhập mới được access

5. PUT/POST /api/trips/:tripId

- a. Mô tả: update thông tin chuyến đi
- b. PRIVATE: chỉ có userType=driver + đang đăng nhập mới được access

6. DELETE/POST /api/trips/:tripId

- a. Mô tả: delete chuyến đi
- b. PRIVATE: chỉ có userType=driver + đang đăng nhập mới được access



# TRIP

CÁC BẠN TỰ HOÀN THIỆN CÁC PHẦN SAU:

7. POST /api/trips/book/:tripId

- a. Mô tả: User (passenger) book 1 chuyến đi
- b. PRIVATE: chỉ có userType=passenger + đang đăng nhập mới được access

8. POST /api/trips/cancel/:tripId

- a. Mô tả: User (passenger) hủy book 1 chuyến đi
- b. PRIVATE: chỉ có userType=passenger + đang đăng nhập mới được access

9. POST /api/trips/finish/:tripId

- a. Mô tả: Driver kết thúc một chuyến đi
- b. PRIVATE: chỉ có userType=passenger + đang đăng nhập mới được access



# TRIP

CÁC BẠN TỰ HOÀN THIỆN CÁC PHẦN SAU:

## 10. POST /api/trips/rates/:tripId

- a. Mô tả: user (passenger) đánh giá về **tài xế**
- b. PRIVATE: chỉ có userType=passenger + đang đăng nhập mới được access

**Chú ý:** Các Route này có thể thay đổi tùy theo cách làm của học viên, cũng như khi implement phần Frontend

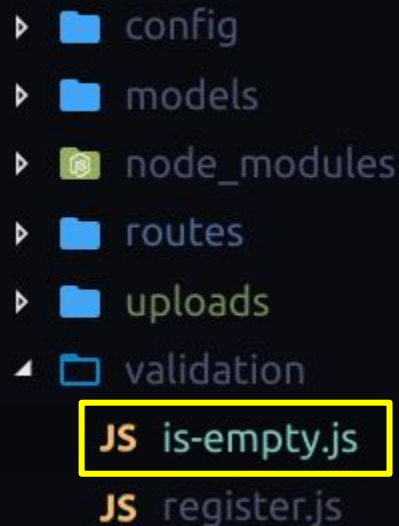


# VALIDATION

- Sử dụng package **validator**: **npm install validator**
- Tự viết tay module **isEmpty** như sau

```
const isEmpty = value =>
  value === undefined ||
  value === null ||
  (typeof value === 'object' && Object.keys(value).length === 0)
  (typeof value === 'string' && value.trim().length === 0)

module.exports = isEmpty;
```



# VALIDATION

- Thực hiện Register validation
- Các field cần thực hiện validation
  - Email: Không được empty, giá trị input là thuộc dạng email
  - Password: Không được empty, min: 6, max: 30
  - Confirmed Password: Phải match với Password
  - Full name: không được empty, min: 2, max: 30
  - User type: không được empty, chỉ có 3 loại là passenger, driver và admin
  - Phone: không được empty, dạng string, min: 10, max: 11
  - Date of birth: không được empty, dạng date



# VALIDATION

- Demo validation email
- Học viên tự thực hiện validate với các field còn lại với các phương thức: isEmpty, isLength, isEmail, equals,...



```
const validator = require('validator');
const isEmpty = require('./is-empty');

module.exports = function validateRegisterInput(data){
  let errors = {};

  if(validator.isEmpty(data.email)){
    errors.email = "Required"
  }

  if(!validator.isEmail(data.email)){
    errors.email = "Email is invalid"
  }

  return {
    errors,
    isValid: isEmpty(errors)
  }
}
```

MYCODER

timviec.com



# VALIDATION

```
// validation
const validateRegisterInput = require('../validation/register');
```

- Áp dụng Register validation

```
router.post('/register', (req, res) => {
  const {errors, isValid} = validateRegisterInput(req.body);

  // check validation
  console.log(isValid)
  if(!isValid) return res.status(400).json(errors);

  const { email, password, fullName, userType, phone, dateOfBirth } = req.body
  User.find({ $or: [{email}, {phone}] })
    .then(users => {
      if (users.length > 0) {
        for(let i=0; i<users.length; i++){
          if(users[i].email === email) errors.email = "Email already exist"
          if(users[i].phone === phone) errors.phone = "Phone already exist"
        }
        return res.status(400).json(errors)
      }
    })
}
```

Bổ sung  
thêm errors



# VALIDATION

Các bạn tự thực hiện validation cho những phần sau

1. Login
2. Thêm profile cho driver
3. Thêm car cho driver
4. Thêm mới một chuyến đi



# ACCESS-CONTROL-ALLOW-ORIGIN

Mặc định trình duyệt (chrome) sẽ tự động chặn những request từ phía Client (frontend) nhằm gia tăng tính bảo mật. Do đó, để phía Client có thể request đến service của Backend thì cần phải bổ sung đoạn code sau trong file **server.js**

```
// allow CORS
app.use(function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Methods", "GET, PUT, POST, DELETE");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept, Authorization");
  next();
});
```



# FRONTEND

- Thiết kế giao diện (UX/UI)
- Gọi service từ backend
- Sử dụng react-router-dom và redux, redux-thunk

# CÀI ĐẶT

- Cài đặt: **(npx) create-react-app client**
- Để khởi động một lúc backend và frontend, sử dụng package: **concurrently**
- Cài đặt: `npm install concurrently`
- Cấu hình package.json

```
"scripts": {  
  "client-install": "npm install --prefix client",  
  "start": "node server.js",  
  "server": "nodemon server.js",  
  "client": "npm start --prefix client",  
  "dev": "concurrently \"npm run server\" \"npm run client\" "  
}
```

install dependencies một lúc cả backend lẫn frontend

Chạy lệnh **npm run dev**

# TỔ CHỨC THƯ MỤC

- Tổ chức thư mục
- Cài đặt **react-router-dom**
- Trong App.js:

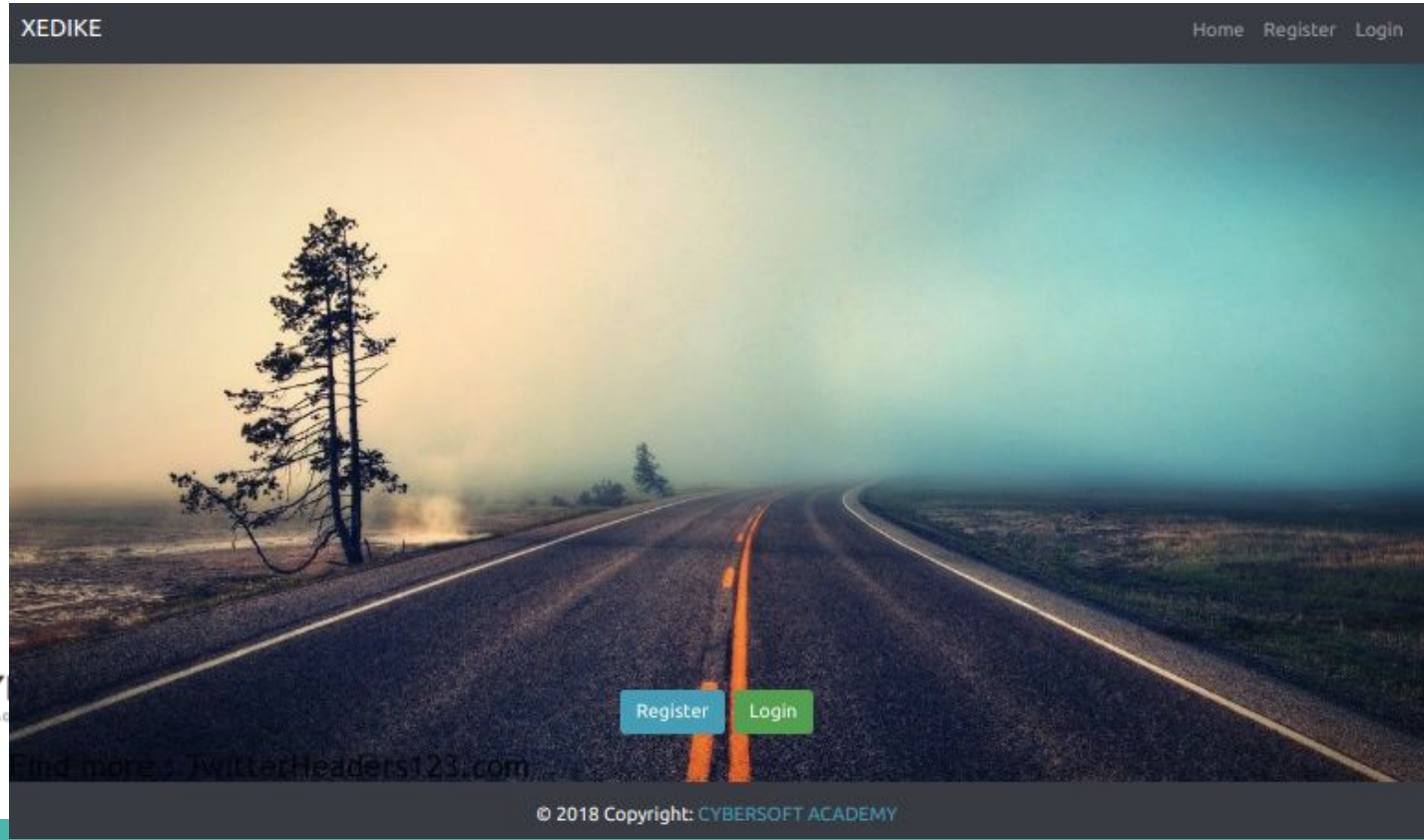
```
├─ src
├─ Components
│  └─ auth
│     ├── JS Login.js
│     ├── JS Register.js
│     └─ layouts
│        ├── JS Footer.js
│        ├── JS Landing.js
│        └─ JS Navbar.js
```

```
class App extends Component {
  render() {
    return (
      <Router>
        <div className="App">
          <Navbar />

          <Route exact path="/" component={Landing} />
          <Route exact path="/register" component={Register} />
          <Route exact path="/login" component={Login} />

          <Footer />
        </div>
      </Router>
    );
  }
}
```

# NAVBAR - LANDING - FOOTER



# REGISTER

- Thiết kế component register
- Thêm sự kiện onChange và onSubmit

## REGISTER NEW USER



# LOGIN

- Thiết kế component login
- Thêm sự kiện onChange và onSubmit

## LOGIN

# AXIOS

- Sử dụng package **axios** để gọi service từ phía backend
- Cài đặt trong Client: **npm install axios**
- Tại ./auth/Register dùng axios để gọi server register
- **Chú ý:** phần này chỉ test, chưa kết hợp với redux

```
onSubmit = (event) => {  
  event.preventDefault()  
  const {email, password, password2, fullName, phone, userType, dateOfBirth} = event.target.elements  
  const newUser = {  
    email, password, password2, fullName, phone, userType, dateOfBirth  
  }  
}
```

```
Axios.post('http://localhost:5000/api/users/register', newUser)  
  .then(console.log)  
  .catch(err => console.log(err.response.data))
```



# AXIOS

- Khi không điền bất cứ thông tin nào trong Register form + Click submit, ta sẽ nhận được chuỗi JSON trong console của browser

```
dateOfBirth: "Required"
email: "Email is invalid"
fullName: "Name must be between 2 and 30 characters"
password: "Password must be between 6 and 30 characters"
password2: "Required"
phone: "Phone must be between 10 and 11 characters"
userType: "Choose passenger or driver"
```

# DISPLAY ERROR

- Sử dụng package **classnames** để hiển thị errors khi validation thất bại
- Cài đặt: **npm install classnames**
- Sử dụng:

```
<div className="form-group" >
  <input type="text"
    className={classnames("form-control form-control-lg", {'is-invalid': errors.email})}
    name="email"
    placeholder="Enter email"
    onChange={this.onChange}
    value={this.state.email}
  />
  {errors.email && (<div className="invalid-feedback">{errors.email}</div>)}
</div>
```

# DISPLAY ERROR

- Kết quả: khi validation fail sẽ hiện kết quả như hình bên
- Học viên tự hoàn thiện validation cho các field còn lại
- Học viên tự hoàn thiện validation cho các component Login, Driver Profile, Trip, Car



## REGISTER

Enter email

Email is invalid

Enter password

Password must be between 6 and 30 characters

Enter confirmed password

Required

Enter full name

Name must be between 2 and 30 characters

Select user type:

Choose passenger or driver

Enter phone

Phone must be between 10 and 11 characters

mm/dd/yyyy

Required

Submit



# REDUX

- Sử dụng **redux** để dễ dàng truyền dữ liệu giữa các Component
- Tuy nhiên, quá trình gọi service là một **async-action**, để giải quyết vấn đề này cần phải cài đặt thêm một trong các middleware sau: **redux-thunk**, **redux-saga**, ...
- Trong bài này, chúng ta sẽ sử dụng **redux-thunk**
- Cài đặt: `npm install redux redux-thunk`



# REDUX

- Tổ chức thư mục như hình bên
- Trong **./store.js**

```
import {createStore, applyMiddleware, compose} from 'redux'  
import thunk from 'redux-thunk'  
import rootReducer from './reducers';
```

Đưa rootReducer vào store

```
const initialState = {};
```

```
const middleware = [thunk]
```

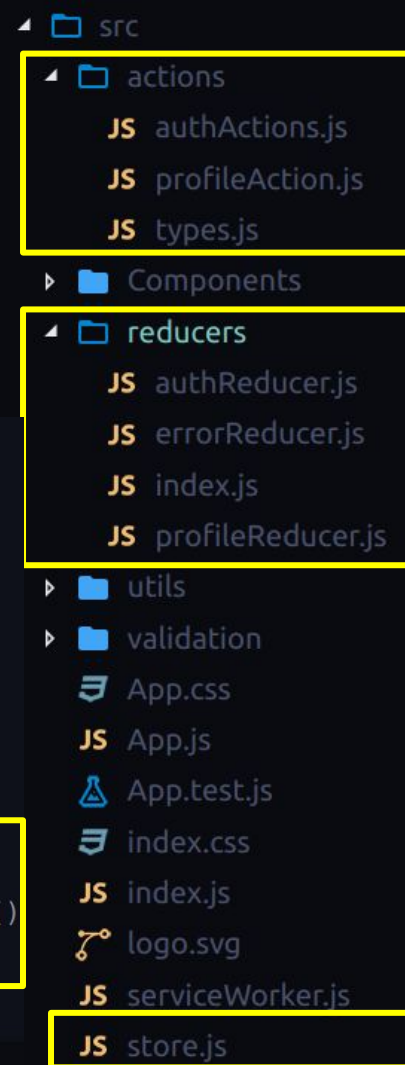
```
const store = createStore(  
  rootReducer,  
  initialState,
```

Cấu hình redux-think

```
  compose(  
    applyMiddleware(...middleware),  
    window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()  
  )  
)
```

Cho phép sử dụng Redux devtools extension trên Chrome

```
export default store;
```



# REDUX

Kết nối từ **Component**  
đến **store** thông  
**Provider**

```
import React from 'react';  
import ReactDOM from 'react-dom';  
import './index.css';  
import App from './App';  
import * as serviceWorker from './serviceWorker';  
import {Provider} from 'react-redux';
```

```
import store from './store';
```

```
ReactDOM.render(  
  <Provider store={store}>  
    <App />  
  </Provider>  
,
```

```
  document.getElementById('root'));
```





# REDUX

Tại `./reducer/index.js`  
sử dụng  
`combineReducers` để  
gôm tất cả các  
reducers lại thành  
**rootReducer** và được  
đưa vào store

```
import {combineReducers} from 'redux';  
import authReducer from './authReducer';  
import errorReducer from './errorReducer';  
import profileReducer from './profileReducer';  
  
export default combineReducers ({  
  auth: authReducer,  
  errors: errorReducer,  
  profile: profileReducer  
})
```

# REDUX

Tại `./actions/types.js`: tạo các hằng số sau:

```
export const SET_CURRENT_USER = 'SET_CURRENT_USER'
export const GET_ERRORS = 'GET_ERRORS'
export const GET_PROFILE = 'GET_PROFILE'
export const PROFILE_NOT_FOUND = 'PROFILE_NOT_FOUND'
export const CLEAR_CURRENT_PROFILE = 'CLEAR_CURRENT_PROFILE'
export const GET_PROFILES = 'GET_PROFILES'
export const PROFILE_LOADING = 'PROFILE_LOADING'
export const GET_DRIVER_PROFILE = 'GET_DRIVER_PROFILE'
```



CYBERSOFT  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



MYCODER

timviec*it*.com

# REGISTER

- Tạo action **registerUser**
- Sử dụng **axios** để gọi service
- Khi **rejected** sẽ thực hiện **GET\_ERRORS**

```
// register
export const registerUser = (userData, history) => {
  return (dispatch) => {
    axios
      .post('http://localhost:5000/api/users/register', userData)
      .then(res => history.push('/login'))
      .catch(err => {
        dispatch({
          type: GET_ERRORS,
          payload: err.response.data
        })
      })
  })
}
```

Sử dụng withRouter tại Component sử dụng mới sinh ra đối tượng **history**

# REGISTER

Tại  
./reducers/authReducer.js

```
import { GET_ERRORS } from '../actions/types';

const initialState = {};

export default function(state = initialState, action) {
  switch (action.type) {
    case GET_ERRORS:
      return action.payload;
    default:
      return state;
  }
}
```

GET\_ERRORS trả về chuỗi  
json error

# REGISTER

Tại ./components/auth/Register.js:

- import {connect} để kết nối component với store
- **import {withRouter} from 'react-router-dom'** để tạo đối tượng **history**

```
const mapStateToProps = (state) => {  
  return {  
    auth: state.auth,  
    errors: state.errors  
  }  
}
```

```
export default connect(mapStateToProps, {registerUser})(withRouter(Register));
```

# REGISTER

Tại ./components/auth/Register.js:

- Gọi **registerUser** trong **onSubmit(...)**

```
onSubmit = (event) => {  
  event.preventDefault()  
  const {email, password, password2, fullName, phone, us  
  const newUser = {  
    email, password, password2, fullName, phone, userT  
  }  
  
  this.props.registerUser(newUser, this.props.history);  
}
```



# LOGIN

- Khi login thành công, cần đưa chuỗi jsonwebtoken vào localStorage và header của những lần gọi service khác
- Tại ./utils/setAuthToken.js, tạo hàm setAuthToken(...)

```
import axios from 'axios'

const setAuthToken = token => {
  if(token){
    axios.defaults.headers.common['Authorization'] = token;
  } else {
    delete axios.defaults.headers.common['Authorization'];
  }
}

export default setAuthToken;
```

Gắn header vào khi login thành công

Xóa header khi hết thời gian login, hoặc khi logout





# LOGIN

Tại authActions.js

```
// set Current user
export const setCurrentUser = (decoded) => {
  return {
    type: SET_CURRENT_USER,
    payload: decoded
  }
}
```

```
// login - get jwt
export const login = (userData) => {
  return (dispatch) => {
    // Sử dụng package jwt-decode để decode chuỗi jwt
    import jwtDecode from 'jwt-decode'

    axios
      .post('http://localhost:5000/api/users/login', userData)
      .then(res => {
        const {token} = res.data;
        localStorage.setItem('jwtToken', token);
        setAuthToken(token);
        const decoded = jwtDecode(token);
        dispatch(setCurrentUser(decoded))
      })
      .catch(err => {
        dispatch({
          type: GET_ERRORS,
          payload: err.response.data
        })
      })
  }
}
```

Khi đăng nhập thành công sẽ đưa chuỗi jwt lên localStorage và set header cho những hành động cần "đăng nhập"

Khi đăng nhập thất bại sẽ thực hiện GET\_ERRORS



# LOGIN

Tại `./reducers/authReducer.js`

- Giá trị mặc định của `isAuthenticated` là `false`.



```
import { SET_CURRENT_USER } from '../actions/types';
import isEmpty from '../validation/is-empty';

const initialState = {
  isAuthenticated: false,
  user: {}
}

export default function(state = initialState, action){
  switch (action.type) {
    case SET_CURRENT_USER:
      return {
        ...state,
        isAuthenticated: !isEmpty(action.payload),
        user: action.payload
      }

    default:
      break;
  }

  return state;
}
```

# LOGIN

- Nếu đăng nhập thành công sẽ quay lại trang chủ
- Nếu đăng nhập thất bại sẽ hiển thị lỗi (tương tự như hiển thị error khi validation register thất bại)

```
const mapStateToProps = (state) => {  
  return {  
    auth: state.auth,  
    errors: state.errors  
  }  
}  
  
export default connect(mapStateToProps, {login})(Login);
```

```
onSubmit = (event) => {  
  event.preventDefault()  
  const {email, password} = this.state  
  this.props.login({email, password})  
}  
  
componentWillReceiveProps = (nextProps) => {  
  if(nextProps.auth.isAuthenticated){  
    this.props.history.push('/')  
  }  
  if(nextProps.errors){  
    this.setState({errors: nextProps.errors})  
  }  
}
```

# LOGOUT

Action logout được thực thi khi:

- Người dùng click logout (component **Navbar.js**)
- khi hết thời gian đăng nhập (theo yêu cầu là 1 tiếng) (component **App.js**)



```
//logout
export const logout = () => {
  return (dispatch) => {
    localStorage.removeItem('jwtToken')
    setAuthToken(false)
    dispatch(setCurrentUser({}))
    dispatch(clearCurrentProfile())
  }
}
```

```
if(localStorage.jwtToken){
  setAuthToken(localStorage.jwtToken)
  const decoded = jwtDecode(localStorage.jwtToken)
  store.dispatch(setCurrentUser(decoded));
  const currentTime = Date.now()
  if(decoded.iat + 3600 > currentTime){
    store.dispatch(logout())
    // store.dispatch(clearCurrentProfile());
    window.location.href = "/login"
  }
}
```

# NHỮNG PHẦN CÒN LẠI ...

Những phần quan trọng để hoàn tất project cuối khóa đã được trình bày trong slide. Những tính năng còn lại (tương tự) sẽ do học viên tự hoàn thành và nộp lại cho trung tâm Cybersoft Academy.

Giảng viên có thể hướng dẫn code thêm một số phần không có trong slide

CHÚC CÁC BẠN HOÀN THÀNH TỐT PROJECT XEDIKE VỚI NHIỀU NIỀM VUI




# TỔNG KẾT KIẾN THỨC

- MongoDB:
  - Xây dựng mô hình
  - Tạo Schema, model, instance, query,... bằng package mongoose
- NodeJS:
  - Tạo nodejs server bằng express
  - API, HTTP
  - jsonwebtoken, bcrypt, passport authentication
  - Validation
- ReactJS:
  - react-router-dom
  - redux và redux-thunk







Thank  
you!