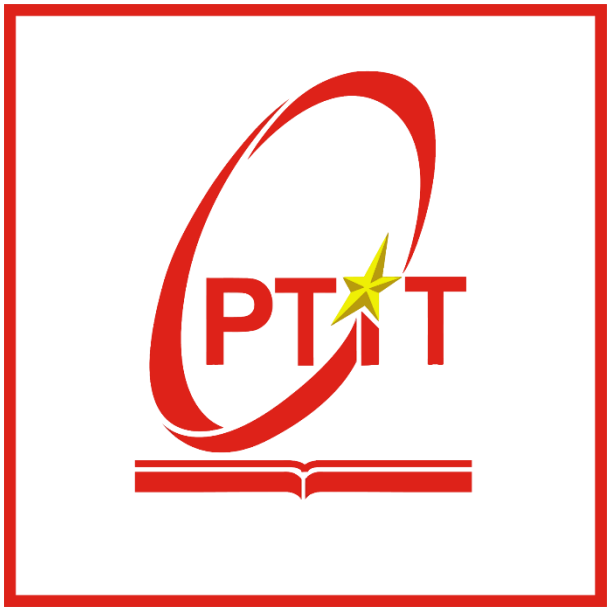
BỘ THÔNG TIN VÀ TRUYỀN THÔNG HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO THỰC TẬP CƠ SỞ TUẦN 11

Triển khai dự án (phần 4)

Giảng viên hướng dẫn: TS. Kim Ngọc Bách

Sinh viên thực hiện:

Lê Quang Thanh – B22DCVT509

I. Triển khai code backend

1. Cấu hình và Khởi tạo:

- Khai báo các hàm quản lí :

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const nodemailer = require('nodemailer');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 5000;
```

- Khởi tạo Middleware:

```
app.use(express.json());
app.use(cors({
  origin: 'http://localhost:3000',
  credentials: true
}));
```

-Middleware xác thực người dùng JWT:

```
// Middleware xác thực JWT
function authMiddleware(req, res, next) {
  const token = req.header('x-auth-token');
  if (!token) return res.status(401).json({ message: 'Không có token, truy cập bị từ chối
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET || 'movieappsecret');
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Token không hợp lệ' });
  }
}
```

- Kết nối với MongoDB:

```
mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log('MongoDB connected'))
.catch(err => console.log('MongoDB connection error:', err));
```

2. Các Models (Schema):

a) User Model - Quản lý người dùng:

```
// User Model
const UserSchema = new mongoose.Schema({
  wishlist: [
    {
      movieId: String,
      title: String,
      poster_path: String,
      vote_average: Number,
      release_date: String,
      media_type: String
    }
  ],
  username: {
    type: String,
    required: true,
    unique: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});

const User = mongoose.model('User', UserSchema);
```

b) Comment Model - Quản lý bình luận:

```
const CommentSchema = new mongoose.Schema({
  movieId: { type: String, required: true },
  user: {
    id: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
    username: String
  },
  content: { type: String, required: true },
  parentId: { type: mongoose.Schema.Types.ObjectId, ref: 'Comment', default: null },
  createdAt: { type: Date, default: Date.now },
  likes: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }]
});
const Comment = mongoose.model('Comment', CommentSchema);
```

c) Notification Model - Quản lý thông báo:

```
const NotificationSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  message: { type: String, required: true },
  createdAt: { type: Date, default: Date.now },
  read: { type: Boolean, default: false }
});
const Notification = mongoose.model('Notification', NotificationSchema);
```

d) OTP Model - Quản lý mã xác thực:

```
/ OTP Model
onst OtpSchema = new mongoose.Schema({
 email: { type: String, required: true },
 otp: { type: String, required: true },
 expiresAt: { type: Date, required: true }
);
onst Otp = mongoose.model('Otp', OtpSchema);
```

3. API Endpoints:

a) Authentication APIs:

/api/register - Đăng ký tài khoản mới

/api/login - Đăng nhập

/api/auth/send-otp - Gửi mã OTP

/api/auth/verify-otp - Xác thực OTP

/api/auth/reset-password - Đặt lại mật khẩu

b) User APIs:

/api/user - Lấy thông tin user

/api/user/profile - Cập nhật thông tin user

/api/wishlist - CRUD danh sách phim yêu thích

c) Comment APIs:

/api/comments/:movieId - Lấy comments của phim

/api/comments - Thêm comment mới

/api/comments/:commentId - Xóa comment

/api/comments/:id/like - Like/unlike comment

d) Notification APIs:

/api/notifications - Lấy/thêm thông báo

/api/notifications/:id - Xóa thông báo

II. Các chức năng nổi bật

1. Xây dựng xác thực OTP với nodemailer:

a. Quy trình hoạt động tổng thể:

- User nhập email -> Kiểm tra email tồn tại
- Gửi OTP -> Tạo OTP -> Lưu DB -> Gửi email
- User nhập OTP -> Kiểm tra OTP hợp lệ
- User đặt mật khẩu mới -> Cập nhật mật khẩu -> Xóa OTP

b. Cấu hình nodemailer

- Tải thư viện nodemailer

```
"dependencies": {
  "bcryptjs": "^2.4.3",
  "cors": "^2.8.5",
  "dotenv": "^16.3.1",
  "express": "^4.18.2",
  "jsonwebtoken": "^9.0.2",
  "mongoose": "^8.0.3",
  "nodemailer": "^7.0.3"
},
```

- Cấu hình nodemailer:

```
const transporter = nodemailer.createTransp
  service: 'gmail',
  auth: {
    user: process.env.EMAIL_USER,  // Emai
    pass: process.env.EMAIL_PASS   // Mật k
  }
});
```

c. Các luồng hoạt động :

- API gửi OTP

```
app.post('/api/auth/send-otp', async (req,
  try {
    const { email } = req.body;
```

- Tạo mã OTP ngẫu nhiên 6 số và thời gian hết hạn

```
const otp = Math.floor(100000 + Math.random() * 900000).toString();

// 2. Tạo thời gian hết hạn (5 phút)
const expiresAt = new Date(Date.now() + 5 * 60 * 1000);
```

- Xóa OTP cũ nếu có

```
// 3. Xóa OTP cũ nếu có
await Otp.deleteMany({ email });
```

- Gửi email chứa OTP

```
  await transporter.sendMail({
    from: process.env.EMAIL_USER,  // Email gửi đi
    to: email,                     // Email người nhận
    subject: 'Mã OTP xác thực Movie App',
    text: `Mã OTP của bạn là: ${otp}. Mã có hiệu lực trong 5 phút.`
  });

  res.json({ message: 'Đã gửi mã OTP về email.' });
} catch (err) {
  console.error('Lỗi gửi OTP:', err);
  res.status(500).json({ message: 'Không thể gửi OTP.' });
}
;
```

- Cơ chế xác thực của OTP:

```
pp.post('/api/auth/verify-otp', async (req, res) => {
 try {
   const { email, otp } = req.body;

   // 1. Tìm record OTP trong database
   const record = await Otp.findOne({ email, otp });
   if (!record) {
     return res.status(400).json({ message: 'OTP không đúng' });
   }

   // 2. Kiểm tra thời gian hết hạn
   if (record.expiresAt < new Date()) {
     return res.status(400).json({ message: 'OTP đã hết hạn' });
   }

   res.json({ message: 'OTP hợp lệ' });
 } catch (err) {
   res.status(500).json({ message: 'Không thể xác thực OTP' });
 }
);
```

- Reset mật khẩu sau khi xác thực OTP:

```
app.post('/api/auth/reset-password', async (req, res) => {
  try {
    const { email, otp, newPassword } = req.body;

    // 1. Kiểm tra OTP có hợp lệ
    const record = await Otp.findOne({ email, otp });
    if (!record || record.expiresAt < new Date()) {
      return res.status(400).json({ message: 'OTP không hợp lệ hoặc đã hết hạn' });
    }

    // 2. Tìm user và cập nhật mật khẩu
    const user = await User.findOne({ email });
    const salt = await bcrypt.genSalt(10);
    user.password = await bcrypt.hash(newPassword, salt);
    await user.save();

    // 3. Xóa OTP đã sử dụng
    await Otp.deleteMany({ email });

    res.json({ message: 'Đổi mật khẩu thành công!' });
  } catch (err) {
    res.status(500).json({ message: 'Không thể đổi mật khẩu.' });
  }
```

2. Xây dựng chức năng đăng nhập đăng ký với JWT Authentication

a. Giao diện người dùng

- Sử dụng form chung cho cả đăng nhập và đăng ký
- Có thể chuyển đổi giữa 2 form thông qua nút toggle
- Form đăng nhập có 2 trường: email và password
- Form đăng ký có 4 trường: username, email, password và confirmPassword
- Có link "Quên mật khẩu" trong form đăng nhập

b. Validation

- Kiểm tra username không được trống khi đăng ký
- Validate email phải đúng định dạng
- Password phải có ít nhất 6 ký tự
- Confirm password phải khớp với password khi đăng ký
- Hiển thị lỗi validation ngay dưới mỗi field

c. Các luồng đăng ký và đăng nhập:

- Luồng đăng ký:

- Frontend:

```
// 1. User điền form đăng ký
const handleRegister = async (e) => {
  e.preventDefault();
  try {
    // 2. Gửi request đến API
    const res = await axios.post('/api/register', {
      username,
      email,
      password
    });

    // 3. Lưu token vào localStorage
    localStorage.setItem('token', res.data.token);

    // 4. Cập nhật trạng thái đăng nhập
    login(res.data.user);
  } catch (err) {
    setError(err.response.data.message);
  }
}
```

- Backend:

```
// 1. Kiểm tra email đã tồn tại
let user = await User.findOne({ email });
if (user) {
  return res.status(400).json({ message: 'Email đã được sử dụng' });
}

// 2. Kiểm tra username đã tồn tại
user = await User.findOne({ username });
if (user) {
  return res.status(400).json({ message: 'Username đã tồn tại' });
}

// 3. Mã hóa mật khẩu
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(password, salt);
```

```
// 4. Tạo user mới
user = new User({
  username,
  email,
  password: hashedPassword
});
await user.save();

// 5. Tạo JWT token
const token = jwt.sign(
  { id: user._id },
  process.env.JWT_SECRET,
  { expiresIn: '1d' }
);

// 6. Trả về token và thông tin user
res.status(201).json({
  token,
  user: {
    id: user._id,
    username: user.username,
    email: user.email
  }
});
} catch (err) {
```

- Luồng đăng nhập:

- Frontend:

```
const handleLogin = async (e) => {
  e.preventDefault();
  try {
    // 1. Gửi request đăng nhập
    const res = await axios.post('/api/login', {
      email,
      password
    });

    // 2. Lưu token
    localStorage.setItem('token', res.data.token);

    // 3. Cập nhật context
    login(res.data.user);
  } catch (err) {
    setError(err.response.data.message);
  }
}
```

- Backend:

```
// 1. Tìm user theo email
const user = await User.findOne({ email });
if (!user) {
  return res.status(400).json({ message: 'Email không tồn tại' });
}

// 2. Kiểm tra mật khẩu
const isMatch = await bcrypt.compare(password, user.password);
if (!isMatch) {
  return res.status(400).json({ message: 'Mật khẩu không đúng' });
}

// 3. Tạo JWT token
const token = jwt.sign(
  { id: user._id },
  process.env.JWT_SECRET,
  { expiresIn: '1d' }
);
```

```
    // 4. Trả về token và thông tin user
    res.json({
      token,
      user: {
        id: user._id,
        username: user.username,
        email: user.email
      }
    });
  } catch (err) {
    res.status(500).json({ message: 'Lỗi server' });
  }
});
```

d. Auth Middleware: Bảo vệ các route cần xác thực

```
function authMiddleware(req, res, next) {
  const token = req.header('x-auth-token');
  if (!token) {
    return res.status(401).json({ message: 'Không có token' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(401).json({ message: 'Token không hợp lệ' });
  }
}
```

III. Tổng thể

- Backend được xây dựng theo mô hình MVC (Model-View-Controller) đơn giản:

  - Models: Định nghĩa cấu trúc dữ liệu

- Controllers: Logic xử lý trong các route handlers
- Views: Được handle bởi React frontend
- Các tính năng bảo mật:
  + Authentication với JWT
  + Password hashing
  + Rate limiting (nếu cần thêm)
  + Input sanitization
  + Error handling
  + Secure headers

- Backend cung cấp đầy đủ các API cần thiết cho một ứng dụng xem phim với các chức năng:

+ Quản lý người dùng
+ Danh sách yêu thích
+ Hệ thống bình luận
+ Thông báo
+ Xác thực email
+ Streaming video