<https://docs.google.com/forms/d/e/1FAIpQLSccIBF1zdj8Jwnt-RP2ozBcCQ4-eGyHVBpCEUdGVaFOvAJnTw/viewscore?viewscore=AE0zAgCVi0DZg8AwO_M8UBXEHIxC_eR8tm-GmV1MCBKLDlWgF8XuKf-4N6IqT6RlXNwc00E>

tôi có 1 model job tôi có 1 model stage , job và stage có quan hệ nhiều nhiều và đây là 3 model của 3 bảng này const { DataTypes } = require('sequelize');

const sequelize = require('../../config/db');

const Job = sequelize.define('Job', {

IDJob: {

type: DataTypes.INTEGER,

autoIncrement: true,

primaryKey: true

},

Status: {

type: DataTypes.STRING,

allowNull: false,

defaultValue: 'pending',

validate: {

isIn: [['pending', 'completed', 'canceled']]

}

},

IDUserAssign: DataTypes.INTEGER,

// IDUserPerform: { // Người thực hiện công việc

// type: DataTypes.INTEGER,

// allowNull: false,

// },

//tam thoi cho null de pass qua project

IDUserPerform: { // Người thực hiện công việc

type: DataTypes.INTEGER,

allowNull: true,

},

IDCreator: DataTypes.INTEGER,

TimeComplete: DataTypes.DATE,

TimeStart: DataTypes.DATE,

DescriptionJob: DataTypes.STRING,

approximateTime: DataTypes.DATE,

NameJob: DataTypes.STRING,

IDPriorityLevel: DataTypes.INTEGER,

Priority: DataTypes.STRING,

IDListFollower: DataTypes.STRING,

IDProject: DataTypes.INTEGER,

IDWorkFLow: DataTypes.INTEGER,

GroupID: DataTypes.INTEGER,

}, {

tableName: 'job',

timestamps: false

});

module.exports = Job; // models/JobStage.js

const { DataTypes } = require('sequelize');

const sequelize = require('../../config/db');

const Job = require('./Job\_Model');

const Stage = require('./Stage\_Model');

const User = require('./User\_Model');

const JobStage = sequelize.define('JobStage', {

ID: {

type: DataTypes.INTEGER,

allowNull: false,

primaryKey: true,

autoIncrement: true

},

IDJob: {

type: DataTypes.INTEGER,

allowNull: false,

references: {

model: Job,

key: 'IDJob'

},

onUpdate: 'CASCADE',

onDelete: 'CASCADE'

},

IDStage: {

type: DataTypes.INTEGER,

allowNull: false,

references: {

model: Stage,

key: 'IDStage'

},

onUpdate: 'CASCADE',

onDelete: 'CASCADE'

},

status: {

type: DataTypes.ENUM('completed', 'in\_progress', 'cancel'),

allowNull: false,

defaultValue: 'in\_progress'

},

completedAt: {

type: DataTypes.DATE,

allowNull: true

},

signatoryId: {

type: DataTypes.INTEGER,

},

description: {

type: DataTypes.TEXT,

allowNull: true

}

}, {

tableName: 'jobstage',

timestamps: true

}

);

module.exports = JobStage;

// models/Stage.js

const { DataTypes } = require('sequelize');

const sequelize = require('../../config/db');

const Stage = sequelize.define('Stage', {

IdStage: {

type: DataTypes.INTEGER,

autoIncrement: true,

primaryKey: true

},

NameStage: DataTypes.STRING,

DescriptionStatus: DataTypes.TEXT,

IDWorkFlow: DataTypes.INTEGER,

previousStage: DataTypes.INTEGER,

nextStage: DataTypes.INTEGER,

reviewer: DataTypes.STRING,

statusStage: DataTypes.STRING,

approximateTime: DataTypes.DATE,

timecompletedState: DataTypes.DATE,

IDRecipient: DataTypes.INTEGER,

EmailRecipient:DataTypes.STRING

}, {

tableName: 'stage',

timestamps: true

});

module.exports = Stage; stage thuộc về workFlow , có nghĩa là 1 workFlow có nhiều stage , khi tạo job tôi sẽ cho người dùng chọn nó thuộc về workFLow nào , tôi muốn làm chức năng khi hoàn thành job thì người duyệt stage sẽ là người kiểm duyệt job này có pass qua stage đó hay không , nếu stage không có người nhận thì người nhận job chính là người duyệt stage đó luôn , nếu pass qua tất cả stage thì job đó hoàn thành , còn nếu job đó đến 1 stage mà không không được chấp nhận vì làm sai thì job sẽ hoàn về stage trước đó , và người nhận phải hoàn thành lại và gửi lên lại stage trước stage bị hoàn về , thực hiện chức năng này như thế nào , lịch sử của job với stage sẽ được lưu ở bảng jobstage

Để thực hiện chức năng mà bạn mô tả, chúng ta cần điều chỉnh một chút trong thiết kế backend và frontend để đảm bảo rằng Job sẽ tự động gửi lên Stage đầu tiên của Workflow và chuyển sang các Stage tiếp theo theo đúng quy trình. Dưới đây là các bước cụ thể để xây dựng tính năng này.

**1. Cập nhật Backend**

**a. Gửi Job lên Stage đầu tiên của Workflow**

Khi người dùng tạo Job và chọn Workflow, chúng ta sẽ xác định Stage đầu tiên trong Workflow và tự động tạo bản ghi trong bảng JobStage.

javascript

Sao chép mã

// routes/job.js

const express = require('express');

const router = express.Router();

const { Job, JobStage, Stage, Workflow } = require('../models');

// Tạo job và gửi lên stage đầu tiên của workflow

router.post('/jobs', async (req, res) => {

const { workflowId, ...jobData } = req.body;

try {

// Tạo Job mới

const job = await Job.create(jobData);

// Lấy stage đầu tiên của workflow

const firstStage = await Stage.findOne({

where: { IDWorkFlow: workflowId },

order: [['IdStage', 'ASC']], // Sắp xếp để lấy stage đầu tiên

});

// Tạo bản ghi JobStage cho stage đầu tiên

await JobStage.create({

IDJob: job.IDJob,

IDStage: firstStage.IdStage,

status: 'in\_progress',

});

res.status(201).json({ message: 'Job created and sent to first stage', job });

} catch (error) {

res.status(500).json({ message: error.message });

}

});

**b. Chuyển Job đến Stage tiếp theo**

Khi người duyệt quyết định duyệt Job, Job sẽ được chuyển đến Stage tiếp theo. Bạn cần cập nhật logic trong API duyệt Job:

javascript

Sao chép mã

// Duyệt job tại stage

router.post('/jobs/:jobId/review-stage/:stageId', async (req, res) => {

const { jobId, stageId } = req.params;

const { accepted } = req.body;

try {

const jobStage = await JobStage.findOne({

where: {

IDJob: jobId,

IDStage: stageId,

},

});

if (!jobStage) {

return res.status(404).json({ message: 'JobStage not found' });

}

// Nếu được chấp nhận

if (accepted) {

jobStage.status = 'completed';

await jobStage.save();

// Kiểm tra xem có stage tiếp theo không

const nextStage = await Stage.findOne({

where: { IDWorkFlow: jobStage.IDStage }, // Lấy IDWorkflow từ stage hiện tại

order: [['IdStage', 'ASC']],

offset: 1, // Bỏ qua stage hiện tại

});

if (nextStage) {

// Tạo bản ghi JobStage cho stage tiếp theo

await JobStage.create({

IDJob: jobId,

IDStage: nextStage.IdStage,

status: 'in\_progress',

});

res.status(200).json({ message: 'Job accepted, moved to next stage' });

} else {

// Nếu không còn stage nào, đánh dấu job hoàn thành

jobStage.status = 'completed';

await jobStage.save();

res.status(200).json({ message: 'Job completed' });

}

} else {

// Nếu không chấp nhận, chuyển job về stage trước đó

await revertJobToPreviousStage(jobId, stageId);

res.status(200).json({ message: 'Job reverted to previous stage' });

}

} catch (error) {

res.status(500).json({ message: error.message });

}

});

**c. Hàm hoàn về stage trước đó**

Tạo hàm revertJobToPreviousStage để hoàn về stage trước đó:

javascript

Sao chép mã

async function revertJobToPreviousStage(jobId, currentStageId) {

// Lấy tất cả JobStage của job

const jobStages = await JobStage.findAll({

where: { IDJob: jobId },

order: [['createdAt', 'DESC']],

});

// Tìm index của stage hiện tại

const currentIndex = jobStages.findIndex(js => js.IDStage === currentStageId);

// Nếu có stage trước đó

if (currentIndex > 0) {

const previousJobStage = jobStages[currentIndex - 1];

previousJobStage.status = 'in\_progress'; // Đặt lại trạng thái

await previousJobStage.save();

}

}

**2. Cập nhật Frontend**

**a. Gửi Job lên Stage**

Khi người dùng tạo Job mới, giao diện sẽ tự động gửi lên Stage đầu tiên mà không cần lựa chọn.

html

Sao chép mã

<!-- Mẫu tạo Job mới -->

<form id="createJobForm">

<input type="text" name="NameJob" placeholder="Tên Job" required />

<input type="hidden" name="workflowId" value="ID\_CỦA\_WORKFLOW" />

<button type="submit">Tạo Job</button>

</form>

<script>

document.getElementById('createJobForm').onsubmit = async function (e) {

e.preventDefault();

const formData = new FormData(this);

const response = await fetch('/api/jobs', {

method: 'POST',

body: JSON.stringify(Object.fromEntries(formData)),

headers: { 'Content-Type': 'application/json' },

});

const result = await response.json();

alert(result.message);

};

</script>

**b. Duyệt Job**

Giao diện cho việc duyệt Job vẫn giống như trước, với việc thêm thông tin về trạng thái.

html

Sao chép mã

<!-- Danh sách Job chờ duyệt -->

<ul id="jobList">

<!-- Các job sẽ được render ở đây -->

</ul>

<script>

async function renderJobs() {

const response = await fetch('/api/jobs');

const jobs = await response.json();

const jobListElement = document.getElementById('jobList');

jobListElement.innerHTML = '';

jobs.forEach(job => {

const li = document.createElement('li');

li.innerHTML = `

${job.NameJob}

<button onclick="reviewJob(${job.IDJob}, true)">Duyệt</button>

<button onclick="reviewJob(${job.IDJob}, false)">Không chấp nhận</button>

`;

jobListElement.appendChild(li);

});

}

async function reviewJob(jobId, accepted) {

const stageId = ...; // Lấy stageId từ context hiện tại

const response = await fetch(`/api/jobs/${jobId}/review-stage/${stageId}`, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ accepted }),

});

const result = await response.json();

alert(result.message);

renderJobs(); // Cập nhật lại danh sách job sau khi duyệt

}

renderJobs(); // Gọi hàm render khi trang được tải

</script>

**Kết luận**

Với các cập nhật này, bạn sẽ có một hệ thống hoàn chỉnh cho phép Job tự động gửi lên Stage đầu tiên của Workflow, chuyển tiếp đến các Stage tiếp theo khi được duyệt, và hoàn về Stage trước đó nếu bị từ chối. Điều này sẽ giúp quản lý quy trình duyệt Job một cách hiệu quả và rõ ràng.

## 1. Cập nhật Backend

### a. Tạo Route để Tạo Job và Gửi lên Stage Đầu tiên

**File:** backend/routes/job.js

javascript

Sao chép mã

// backend/routes/job.js

const express = require('express');

const router = express.Router();

const { Job, JobStage, Stage, Workflow } = require('../models');

// Tạo job và gửi lên stage đầu tiên của workflow

router.post('/jobs', async (req, res) => {

const { workflowId, ...jobData } = req.body;

try {

// Tạo Job mới

const job = await Job.create(jobData);

// Lấy stage đầu tiên của workflow

const firstStage = await Stage.findOne({

where: { IDWorkFlow: workflowId },

order: [['IdStage', 'ASC']], // Sắp xếp để lấy stage đầu tiên

});

if (!firstStage) {

return res.status(404).json({ message: 'First stage not found in the workflow' });

}

// Tạo bản ghi JobStage cho stage đầu tiên

await JobStage.create({

IDJob: job.IDJob,

IDStage: firstStage.IdStage,

status: 'in\_progress',

});

res.status(201).json({ message: 'Job created and sent to first stage', job });

} catch (error) {

res.status(500).json({ message: error.message });

}

});

module.exports = router;

### b. Tạo Route để Duyệt Job và Chuyển Stage

**File:** backend/routes/job.js (tiếp tục thêm vào file này)

javascript

Sao chép mã

// Duyệt job tại stage

router.post('/jobs/:jobId/review-stage/:stageId', async (req, res) => {

const { jobId, stageId } = req.params;

const { accepted } = req.body;

try {

const jobStage = await JobStage.findOne({

where: {

IDJob: jobId,

IDStage: stageId,

},

});

if (!jobStage) {

return res.status(404).json({ message: 'JobStage not found' });

}

// Nếu được chấp nhận

if (accepted) {

jobStage.status = 'completed';

await jobStage.save();

// Lấy workflow của stage hiện tại

const currentStage = await Stage.findByPk(stageId);

const workflowId = currentStage.IDWorkFlow;

// Lấy stage tiếp theo trong workflow

const nextStage = await Stage.findOne({

where: { IDWorkFlow: workflowId, IdStage: { [Op.gt]: currentStage.IdStage } },

order: [['IdStage', 'ASC']],

});

if (nextStage) {

// Tạo bản ghi JobStage cho stage tiếp theo

await JobStage.create({

IDJob: jobId,

IDStage: nextStage.IdStage,

status: 'in\_progress',

});

res.status(200).json({ message: 'Job accepted, moved to next stage' });

} else {

// Nếu không còn stage nào, đánh dấu job hoàn thành

await job.update({ status: 'completed' });

res.status(200).json({ message: 'Job completed' });

}

} else {

// Nếu không chấp nhận, chuyển job về stage trước đó

await revertJobToPreviousStage(jobId, stageId);

res.status(200).json({ message: 'Job reverted to previous stage' });

}

} catch (error) {

res.status(500).json({ message: error.message });

}

});

const { Op } = require('sequelize'); // Đảm bảo bạn đã import Op từ sequelize

module.exports = router;