**Étapes pour gérer un fil d'actualités d'entreprise via des API REST en partant d'une architecture MVC et d'une page de connexion**

**1. Mise en place de l'architecture MVC**

* **Modèle (Model)** : représente les données des actualités (titre, contenu, date, auteur, commentaires, statut de validation, tags, filtres). Il interagit avec la base de données pour stocker et récupérer les actualités.
* **Vue (View)** : interface utilisateur, ici la page de connexion, la page d'affichage du fil d'actualités, les formulaires de commentaires, les notifications, les filtres et les tags.
* **Contrôleur (Controller)** : gère les requêtes, relie les vues aux modèles, et orchestre les actions comme l'ajout, la modification, la suppression, la validation, le commentaire, le filtrage ou le taggage d'une actualité.

**2. Création de la page de connexion**

* Créer une vue HTML/CSS/JS pour la connexion des employés.
* Le contrôleur reçoit les données du formulaire (email, mot de passe).
* Le modèle vérifie les identifiants dans la base de données.
* Si les identifiants sont valides, une session est créée ou un token JWT est généré.

**3. Mise en place de l'authentification**

* Utiliser des sessions ou des tokens JWT pour sécuriser les accès.
* Le contrôleur vérifie l'authentification avant d'autoriser l'accès au fil d'actualités et aux API REST.

**4. Création des API REST pour le fil d'actualités**

* Définir les routes REST :
  + GET /actualites : récupérer toutes les actualités
  + GET /actualites/{id} : récupérer une actualité spécifique
  + POST /actualites : créer une nouvelle actualité
  + PUT /actualites/{id} : mettre à jour une actualité
  + DELETE /actualites/{id} : supprimer une actualité
  + POST /actualites/{id}/commentaires : ajouter un commentaire
  + GET /actualites/{id}/commentaires : récupérer les commentaires
  + POST /actualites/{id}/valider : valider une actualité (action réservée aux RH)
  + GET /actualites?tag={tag} : filtrer les actualités par tag
  + GET /actualites?filtre={filtre} : filtrer les actualités par critère
* Chaque route est gérée par un contrôleur qui appelle les méthodes du modèle.

**5. Intégration avec la base de données**

* Le modèle utilise une base NoSQL facile à apprendre et déployable sur la plateforme [www.alwasdata.com](https://www.alwasdata.com), comme MongoDB.
* Les contrôleurs transmettent les données reçues des API aux modèles.

**6. Sécurisation des API**

* Vérifier les tokens JWT ou les sessions à chaque requête.
* Implémenter des rôles (ex : administrateur, employé, RH) pour gérer les permissions d'ajout, de suppression, de validation ou de commentaire d'actualités.
* Utiliser des middlewares pour la vérification des rôles et des permissions.
* Valider les entrées utilisateur avec des bibliothèques comme Joi ou express-validator.
* Protéger les routes sensibles avec des middlewares d'authentification.
* Vérifier que le compte utilisateur est actif avant d'autoriser l'accès aux routes protégées.

**7. Notifications**

* Implémenter un système de notifications (ex : WebSocket, polling) pour informer les employés des nouvelles actualités ou des réponses à leurs commentaires.
* Afficher les notifications dans l'interface utilisateur.

**8. Tableau de bord RH**

* Créer une interface dédiée aux RH pour visualiser les actualités en attente de validation, les statistiques de publication, les commentaires en attente de modération, et les tags les plus utilisés.
* Intégrer des graphiques (ex : nombre d'actualités par mois, taux de validation, répartition par tag).
* Permettre la validation ou la suppression rapide d'une actualité depuis le tableau.
* Ajouter des filtres pour trier les actualités par département, auteur ou statut.

**9. Structuration technique et choix des technologies**

* **Frontend** : [React.js](https://React.js) pour une interface dynamique et réactive.
* **Backend** : [Node.js](https://Node.js) avec Express pour gérer les API REST.
* **Base de données** : MongoDB ou autre base NoSQL facile à apprendre, déployée sur [www.alwasdata.com](https://www.alwasdata.com).
* **Authentification** : JWT pour les tokens, avec middleware Express pour la vérification.
* **Notifications** : WebSocket via [Socket.io](https://Socket.io) pour les mises à jour en temps réel.
* **Documentation** : Swagger pour documenter les endpoints.
* **Déploiement** : Docker pour la conteneurisation, hébergement sur Heroku.
* **Sécurité** : HTTPS, validation des entrées, gestion des rôles et permissions.

**10. Scripts backend et configuration**

* **Installation** :
  + npm init -y
  + npm install express mongoose dotenv jsonwebtoken bcryptjs cors joi [socket.io](https://socket.io) swagger-ui-express
* **Structure des dossiers** :
  + /models : schémas Mongoose (Actualite, Utilisateur, Commentaire)
  + /controllers : logique métier pour chaque entité
  + /routes : définition des routes REST
  + /middlewares : vérification JWT, rôles, validation des entrées, vérification de compte actif
  + /config : connexion MongoDB via mongoose
  + /views : pages HTML pour la connexion et l'affichage
  + [server.js](https://server.js) : point d'entrée de l'application
* **Exemples de schémas MongoDB** :

// Utilisateur.js

const mongoose = require('mongoose');

const UtilisateurSchema = new mongoose.Schema({

nom: String,

email: { type: String, unique: true },

motDePasse: String,

role: { type: String, enum: ['employe', 'rh', 'admin'], default: 'employe' },

actif: { type: Boolean, default: true }

});

module.exports = mongoose.model('Utilisateur', UtilisateurSchema);

* **Middlewares** :

// verifyToken.js

const jwt = require('jsonwebtoken');

module.exports = function(req, res, next) {

const token = req.header('Authorization');

if (!token) return res.status(401).send('Accès refusé');

try {

const verified = jwt.verify(token, process.env.JWT\_SECRET);

req.user = verified;

next();

} catch (err) {

res.status(400).send('Token invalide');

}

};

// checkRole.js

module.exports = function(role) {

return function(req, res, next) {

if (req.user.role !== role) return res.status(403).send('Accès interdit');

next();

};

};

// checkActif.js

const Utilisateur = require('../models/Utilisateur');

module.exports = async function(req, res, next) {

try {

const utilisateur = await Utilisateur.findById(req.user.\_id);

if (!utilisateur || !utilisateur.actif) return res.status(403).send('Compte inactif');

next();

} catch (err) {

res.status(500).send('Erreur serveur');

}

};

* **Routes REST** :

// utilisateurRoutes.js

const express = require('express');

const router = express.Router();

const utilisateurController = require('../controllers/utilisateurController');

const verify = require('../middlewares/verifyToken');

const checkActif = require('../middlewares/checkActif');

const checkRole = require('../middlewares/checkRole');

router.post('/register', utilisateurController.register);

router.post('/login', utilisateurController.login);

router.get('/me', verify, checkActif, utilisateurController.getProfile);

router.put('/me', verify, checkActif, utilisateurController.updateProfile);

router.get('/admin/utilisateurs', verify, checkActif, checkRole('admin'), utilisateurController.getAll);

router.put('/admin/utilisateurs/:id/actif', verify, checkActif, checkRole('admin'), utilisateurController.toggleActif);

module.exports = router;

// actualiteRoutes.js

const express = require('express');

const router = express.Router();

const actualiteController = require('../controllers/actualiteController');

const verify = require('../middlewares/verifyToken');

const checkActif = require('../middlewares/checkActif');

const checkRole = require('../middlewares/checkRole');

router.get('/', verify, checkActif, actualiteController.getAll);

router.get('/:id', verify, checkActif, actualiteController.getById);

router.post('/', verify, checkActif, actualiteController.create);

router.put('/:id', verify, checkActif, actualiteController.update);

router.delete('/:id', verify, checkActif, actualiteController.remove);

router.post('/:id/valider', verify, checkActif, checkRole('rh'), actualiteController.validate);

router.get('/search/tag', verify, checkActif, actualiteController.searchByTag);

router.get('/search/filtre', verify, checkActif, actualiteController.searchByFiltre);

module.exports = router;

// commentaireRoutes.js

const express = require('express');

const router = express.Router();

const commentaireController = require('../controllers/commentaireController');

const verify = require('../middlewares/verifyToken');

const checkActif = require('../middlewares/checkActif');

router.post('/:actualiteId', verify, checkActif, commentaireController.create);

router.get('/:actualiteId', verify, checkActif, commentaireController.getAll);

module.exports = router;

**11. Test et documentation**

* Utiliser Postman ou Swagger pour tester les API.
* Documenter les endpoints, les paramètres et les réponses.

**Contenu de la collection Postman**

La collection inclut :

| **📌 Nom de la requête** | **Méthode** | **URL** | **Test intégré** |
| --- | --- | --- | --- |
| Connexion | POST | /api/auth/connexion | ✅ Sauvegarde du token |
| Statistiques dashboard | GET | /api/dashboard/statistiques | ✅ Utilise le token |
| Actualités dashboard | GET | /api/dashboard/actualites | ✅ Utilise le token |
| Commentaires dashboard | GET | /api/dashboard/commentaires | ✅ Utilise le token |
| Graphiques dashboard | GET | /api/dashboard/graph | ✅ Utilise le token |
| Inscription | POST | /api/auth/inscription | ❌ (pas besoin de token) |
| Déconnexion | POST | /api/auth/deconnexion | ✅ (optionnel) |

**12. Déploiement**

* Héberger l'application sur Heroku.
* S'assurer que les API REST sont accessibles et sécurisées.
* Mettre en place un système de mise à jour automatique du fil d'actualités sur l'interface employé.

**13. Fonctionnalités avancées à envisager**

**1. Système de recherche intelligent**

* Recherche plein texte sur les titres et contenus des actualités
* Suggestions automatiques de tags ou filtres
* Historique des recherches pour chaque utilisateur

**2. Système de vote ou de réactions**

* 👍 / 👎 ou ❤️ / 😡 pour chaque actualité
* Statistiques sur les réactions par département ou période
* Classement des actualités les plus engageantes

**3. Archivage automatique**

* Archivage des actualités après une certaine durée (ex. 90 jours)
* Accès à une section “Archives” avec filtres temporels

**4. Multi-langue / internationalisation**

* Interface et contenu traduisibles (français, anglais, etc.)
* Détection automatique de la langue ou choix manuel

**5. Gestion des brouillons**

* Permettre aux employés de sauvegarder une actualité sans la publier
* Interface RH pour visualiser et relancer les brouillons en attente

**6. Modération collaborative**

* RH peuvent déléguer la modération à des référents par département
* Historique des validations et des suppressions

**7. Export et reporting**

* Export des actualités et commentaires en CSV ou PDF
* Rapports mensuels automatiques envoyés aux RH

**8. Intégration avec d'autres outils**

* Slack, Teams ou email pour notifier les nouvelles actualités
* SSO (Single Sign-On) avec les comptes d’entreprise

**9. Accessibilité et UX**

* Mode sombre / clair
* Accessibilité pour les personnes malvoyantes (polices, contrastes, navigation clavier)