



# Guide : Créer ta première route API

Je vais t'expliquer étape par étape comment créer une route simple pour afficher une liste de sources. On va procéder de manière progressive pour que tu comprennes bien chaque élément.



## Comprendre la structure actuelle

Regarde ton fichier `app.ts` ligne 62-68 :

```
const apiRouter = express.Router();

apiRouter.get('/', (_req: Request, res: Response) => {
  res.json({
    message: 'Sentinelle-Reputation API',
    version: config.API_VERSION,
    status: 'running',
  });
});
```

C'est déjà une route ! Quand tu visites `http://localhost:5000/api/v1/`, tu obtiens ce JSON. On va faire pareil, mais mieux organisé.



## Architecture à suivre

Pour une API professionnelle, on sépare les responsabilités :

```
Module Sources/
├── sources.routes.ts    → Définit les URLs (/api/v1/sources)
├── sources.controller.ts → Traite les requêtes et répond
├── sources.service.ts   → Logique métier (accès DB, calculs)
└── sources.types.ts     → Types TypeScript
```

### Pourquoi cette séparation ?

- **Routes** : "Quel chemin mène où ?"
- **Controller** : "Que faire avec cette requête ?"
- **Service** : "Comment obtenir/manipuler les données ?"



## Étape par étape : Créer le module Sources

### Étape 1 : Créer la structure de dossiers

```
mkdir -p src/modules/sources
```

### Étape 2 : Définir les types (`sources.types.ts`) 💡 Ce que tu apprends ici :

- On définit l'**interface** `Source` : c'est le "contrat" de nos données
- On crée aussi `SourcesResponse` pour structurer ce que l'API renvoie
- TypeScript va vérifier qu'on respecte ces contrats partout

```
// src/modules/sources/sources.types.ts
```

```
/**
 * 📋 Représente une source d'information
 *
 * Une source peut être un site web, un compte social media, etc.
 * que l'utilisateur souhaite monitorer
 */
export interface Source {
  id: string;
  name: string;
  url: string;
  type: 'website' | 'social_media' | 'news' | 'forum';
  isActive: boolean;
  createdAt: Date;
}
```

```
/**
 * 📦 Format de réponse de l'API
 *
 * On encapsule toujours les données dans un objet
 * pour pouvoir ajouter des métadonnées plus tard
 */
export interface SourcesResponse {
  success: boolean;
  data: Source[];
  count: number;
}
```

### Étape 3 : Créer le Service (**sources.service.ts**) 💡 Ce que tu apprends ici :

- Le **Service** gère les données (pour l'instant fictives)
- Les méthodes sont **async** car en vrai, on attendra la DB
- On exporte une **instance unique** (**sourcesService**) : tout le monde utilise le même objet

```
// src/modules/sources/sources.service.ts
```

```
import { Source } from './sources.types';
```

```
/**
```

```
 * 🎯 Service Sources
```

```
 *
```

```
 * Contient toute la logique métier liée aux sources.
```

```
 * Pour l'instant, on utilise des données "mock" (fictives).
```

```
 * Plus tard, tu remplaceras par des appels à Prisma (ta DB).
```

```
 */
```

```
class SourcesService {
```

```
  /**
```

```
   * Données fictives pour tester
```

```
   *
```

```
   * ➡️ Dans Phase 2, tu remplaceras par :
```

```
   * await prisma.source.findMany()
```

```
   */
```

```
  private mockSources: Source[] = [
```

```
    {
```

```
      id: '1',
```

```
      name: 'TechCrunch',
```

```
      url: 'https://techcrunch.com',
```

```
      type: 'news',
```

```
      isActive: true,
```

```
      createdAt: new Date('2024-01-15'),
```

```
    },
```

```
    {
```

```
      id: '2',
```

```
      name: 'BBC News',
```

```
      url: 'https://bbc.com/news',
```

```
      type: 'news',
```

```
      isActive: true,
```

```
        createdAt: new Date('2024-01-16'),
    },
    {
        id: '3',
        name: 'Reddit - Technology',
        url: 'https://reddit.com/r/technology',
        type: 'forum',
        isActive: false,
        createdAt: new Date('2024-01-17'),
    },
];
```

```
/**
```

```
 * Récupère toutes les sources
```

```
 *
```

```
 * @returns Tableau de sources
```

```
 */
```

```
async getAllSources(): Promise<Source[]> {
    // Simulation d'un appel async (DB)
    return new Promise((resolve) => {
        setTimeout(() => {
            resolve(this.mockSources);
        }, 100); // Simule 100ms de latence DB
    });
}
```

```
/**
```

```
 * Récupère uniquement les sources actives
```

```
 *
```

```
 * @returns Tableau de sources actives
```

```
 */
```

```
async getActiveSources(): Promise<Source[]> {
```

```

const allSources = await this.getAllSources();
return allSources.filter(source => source.isActive);
}

/**
 * Récupère une source par ID
 *
 * @param id - Identifiant de la source
 * @returns Source trouvée ou null
 */
async getSourceById(id: string): Promise<Source | null> {
  const allSources = await this.getAllSources();
  return allSources.find(source => source.id === id) || null;
}
}

```

```

// Export une instance unique (Singleton pattern)
export const sourcesService = new SourcesService();

```

#### Étape 4 : Créer le Controller (**sources.controller.ts**) 💡 Ce que tu apprends ici :

- Le **Controller** reçoit la requête HTTP (**req**) et prépare la réponse (**res**)
- Structure **try/catch** pour gérer les erreurs
- On utilise **next(error)** pour déléguer les erreurs au middleware global
- Codes HTTP : 200 (OK), 404 (Not Found)

```

// src/modules/sources/sources.controller.ts

```

```

import { Request, Response, NextFunction } from 'express';
import { sourcesService } from '../sources.service';
import { SourcesResponse } from '../sources.types';
import { logger } from '@/infrastructure/logger';

/**

```

\* 🎮 Controller Sources

\*

\* Fait le lien entre les routes HTTP et la logique métier.

\* Chaque méthode correspond à une route.

\*/

```
class SourcesController {
```

```
  /**
```

```
   * GET /api/v1/sources
```

```
   *
```

```
   * Récupère toutes les sources
```

```
   *
```

```
   * @example
```

```
   * curl http://localhost:5000/api/v1/sources
```

```
  */
```

```
  async getAllSources(req: Request, res: Response, next: NextFunction) {
```

```
    try {
```

```
      logger.info('Fetching all sources');
```

```
      // 1. Appeler le service pour obtenir les données
```

```
      const sources = await sourcesService.getAllSources();
```

```
      // 2. Construire la réponse formatée
```

```
      const response: SourcesResponse = {
```

```
        success: true,
```

```
        data: sources,
```

```
        count: sources.length,
```

```
      };
```

```
      // 3. Envoyer la réponse avec status 200 (OK)
```

```
      res.status(200).json(response);
```

```
    } catch (error) {
```

```

// 4. En cas d'erreur, passer au gestionnaire d'erreur global
logger.error('Error fetching sources:', error);
next(error);
}
}

/**
 * GET /api/v1/sources/active
 *
 * Récupère uniquement les sources actives
 *
 * @example
 * curl http://localhost:5000/api/v1/sources/active
 */
async getActiveSources(req: Request, res: Response, next: NextFunction) {
  try {
    logger.info('Fetching active sources');

    const sources = await sourcesService.getActiveSources();

    const response: SourcesResponse = {
      success: true,
      data: sources,
      count: sources.length,
    };

    res.status(200).json(response);

  } catch (error) {
    logger.error('Error fetching active sources:', error);
    next(error);
  }
}

```

```
}
```

```
/**
```

```
 * GET /api/v1/sources/:id
```

```
 *
```

```
 * Récupère une source spécifique par son ID
```

```
 *
```

```
 * @example
```

```
 * curl http://localhost:5000/api/v1/sources/1
```

```
 */
```

```
async getSourceById(req: Request, res: Response, next: NextFunction) {
```

```
  try {
```

```
    // Récupérer l'ID depuis les paramètres de l'URL
```

```
    const { id } = req.params;
```

```
    logger.info(`Fetching source with id: ${id}`);
```

```
    const source = await sourcesService.getSourceById(id);
```

```
    // Si la source n'existe pas, renvoyer 404
```

```
    if (!source) {
```

```
      return res.status(404).json({
```

```
        success: false,
```

```
        message: `Source with id ${id} not found`,
```

```
      });
```

```
    }
```

```
    // Source trouvée, renvoyer 200
```

```
    res.status(200).json({
```

```
      success: true,
```

```
      data: source,
```

```
    });
```

```

    } catch (error) {
      logger.error(`Error fetching source by id:`, error);
      next(error);
    }
  }
}
}

```

// Export une instance unique

```
export const sourcesController = new SourcesController();
```

## Étape 5 : Créer les Routes (**sources.routes.ts**) 💡 Ce que tu apprends ici :

- Le **Router** mappe les URLs vers les méthodes du controller
- `.bind(sourcesController)` : nécessaire pour garder le contexte `this`
- **Ordre important** : routes spécifiques (`/active`) avant routes dynamiques (`/:id`)

// src/modules/sources/sources.routes.ts

```
import { Router } from 'express';
import { sourcesController } from '../sources.controller';
```

/\*\*

\* 🏠 Routes Sources

\*

\* Définit les endpoints HTTP pour gérer les sources

\*/

```
const router = Router();
```

/\*\*

\* @route GET /api/v1/sources

\* @desc Récupérer toutes les sources

\* @access Public (pour l'instant)

\*

\* 📝 Note : Plus tard, tu ajouteras un middleware d'authentification :

```

* router.get('/', authenticateUser, sourcesController.getAllSources);
*/

router.get('/', sourcesController.getAllSources.bind(sourcesController));

/**
 * @route GET /api/v1/sources/active
 * @desc Récupérer uniquement les sources actives
 * @access Public
 *
 * ⚠ IMPORTANT : Cette route DOIT être AVANT /:id
 * Sinon Express pensera que "active" est un ID !
 */

router.get('/active', sourcesController.getActiveSources.bind(sourcesController));

/**
 * @route GET /api/v1/sources/:id
 * @desc Récupérer une source par son ID
 * @access Public
 *
 * :id est un paramètre dynamique (peut être 1, 2, abc, etc.)
 */

router.get('/:id', sourcesController.getSourceById.bind(sourcesController));

export default router;

```

## Étape 6 : Connecter le module à l'app (app.ts)

Maintenant, modifie ton fichier `app.ts` pour importer et utiliser ces routes :##  Tester ton API

```

import express, { Application, Request, Response, NextFunction } from 'express';
import cors from 'cors';
import helmet from 'helmet';
import compression from 'compression';
import cookieParser from 'cookie-parser';

```

```
import { config } from './config/app';
import { errorHandler, notFoundHandler } from '@shared/middlelware/error.middleware';
import { logger } from './infrastructure/logger';
```

```
// 📦 Import des routes des modules
```

```
import sourcesRoutes from './modules/sources/sources.routes';
```

```
export const createApp = (): Application => {
```

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

```
  // ===== Security & Basic Middleware =====
```

```
  // Helmet pour sécuriser les headers HTTP
```

```
  app.use(helmet());
```

```
  // CORS
```

```
  app.use(cors({
    origin: [config.CLIENT_URL, config.ADMIN_URL, config.LANDING_URL],
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'OPTIONS'],
    allowedHeaders: ['Content-Type', 'Authorization'],
  }));
```

```
  // Compression des réponses
```

```
  app.use(compression() as any);
```

```
  // Parse JSON bodies
```

```
  app.use(express.json({ limit: '10mb' }));
```

```
  // Parse URL-encoded bodies
```

```
  app.use(express.urlencoded({ extended: true, limit: '10mb' }));
```

```
// Parse cookies
app.use(cookieParser() as any);

// ===== Logging Middleware =====
app.use((req: Request, res: Response, next: NextFunction) => {
  const start = Date.now();

  res.on('finish', () => {
    const duration = Date.now() - start;
    logger.info('HTTP Request', {
      method: req.method,
      path: req.path,
      statusCode: res.statusCode,
      duration: `${duration}ms`,
      ip: req.ip,
    });
  });

  next();
});

// ===== Health Check =====
app.get('/health', (_req: Request, res: Response) => {
  res.json({
    status: 'ok',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    environment: config.NODE_ENV,
  });
});
```

```
// ===== API Routes =====  
  
const apiRouter = express.Router();  
  
// Route d'accueil de l'API  
apiRouter.get('/', (_req: Request, res: Response) => {  
  res.json({  
    message: 'Sentinelle-Reputation API',  
    version: config.API_VERSION,  
    status: 'running',  
    endpoints: {  
      sources: `/api/${config.API_VERSION}/sources`,  
      health: '/health',  
    },  
  });  
});
```

```
// 🔗 Montage des routes des modules  
apiRouter.use('/sources', sourcesRoutes);
```

```
// 📝 Pour ajouter d'autres modules plus tard :  
// apiRouter.use('/users', usersRoutes);  
// apiRouter.use('/alerts', alertsRoutes);
```

```
// Mount API routes  
app.use(`/api/${config.API_VERSION}`, apiRouter);
```

```
// ===== Error Handling =====
```

```
// 404 handler - doit être avant le error handler  
app.use('*', notFoundHandler);
```

```
// Global error handler (doit être en dernier)
```

N'hésite pas à me poser des questions sur ce que tu ne comprends pas ! 😊