Exercices API Node JS

Structure commune à chaque exercice

```
project-root/
├─ controllers/
│   ├─ <entity>Controller.js
├─ routes/
│   ├─ <entity>Routes.js
├─ database.json
├─ database.csv
├─ app.js
└─ package.json
```

- `controllers/` : contient la logique métier (CRUD, traitement des données, appels aux fichiers JSON/CSV, modules natifs).
- `routes/` : définit les endpoints Express et fait appel aux controllers.
- `database.json` : base de données JSON principale.
- `database.csv` : base de données CSV secondaire (à synchroniser ou utiliser selon l'exercice).

Exercice 1 – Bibliothèque de livres

Routes (`routes/bookRoutes.js`)

```
router.post   ('/books',       bookController.createBook);
router.get    ('/books',       bookController.getAllBooks);
router.get    ('/books/:id',    bookController.getBookById);
```

```
router.put    ('/books/:id',     bookController.updateBook);
router.delete ('/books/:id',     bookController.deleteBo
```

Méthodes (`controllers/bookController.js`)

- **createBook** – lit `database.json`, ajoute un nouveau livre (générer `id`), sauvegarde JSON + append CSV.
- **getAllBooks** – lit JSON, renvoie liste et génère à la volée `database.csv`.
- **getBookById** – lit JSON, filtre par `id`.
- **updateBook** – lit JSON, modifie l'objet, réécrit JSON + synchronise CSV.
- **deleteBook** – supprime du JSON, réécrit + synchronise CSV.

Exercice 2 – Carnet d'adresses

Routes (`routes/contactRoutes.js`)

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

router.post   ('/contacts',     contactController.createContact);
router.get    ('/contacts',     contactController.getAllContacts);
router.get    ('/contacts/:id', contactController.getContactById);
router.put    ('/contacts/:id', contactController.updateContact);
router.delete ('/contacts/:id', contactController.deleteContact);
router.get    ('/status',       contactController.getStatus);

module.exports = router;
```

Méthodes (`controllers/contactController.js`)

- **createContact** – génère `id` avec `crypto`, stocke JSON + append CSV.
- **getAllContacts** – lit JSON et renvoie tableau.

- **getContactById** – lit JSON, renvoie contact.
- **updateContact** – modifie contact dans JSON et CSV.
- **deleteContact** – supprime contact de JSON et CSV.
- **getStatus** – utilise `os` pour retourner infos système + nombre de contacts.

Exercice 3 – Journalisation d'événements

Routes (`routes/eventRoutes.js`)

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

router.post   ('/events',            eventController.createEvent);
router.get    ('/events',            eventController.getAllEvents);
router.get    ('/events/:id',        eventController.getEventById);
router.put    ('/events/:id',        eventController.updateEvent);
router.delete ('/events/:id',        eventController.deleteEvent);
router.get    ('/events/compress-logs',eventController.compressLogs);

module.exports = router;
```

Méthodes (`controllers/eventController.js`)

- **createEvent** – ajoute événement JSON + CSV, écrit log via un `stream.Writable`.
- **getAllEvents** – renvoie tous les événements JSON et logue la requête.
- **getEventById** – filtre JSON, logue.
- **updateEvent** – met à jour JSON/CSV, logue.
- **deleteEvent** – supprime JSON/CSV, logue.
- **compressLogs** – compresse le fichier de logs en `.gz` via `zlib`.

Exercice 4 – Catalogue produits

Routes (routes/productRoutes.js)

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

router.post   ('/products',          productController.createProduct);
router.get    ('/products',          productController.getAllProducts);
router.get    ('/products/:id',      productController.getProductById);
router.put    ('/products/:id',      productController.updateProduct);
router.delete ('/products/:id',      productController.deleteProduct);
router.get    ('/products/promos',   productController.getProductsWithPromo
s);

module.exports = router;
```

Méthodes (controllers/productController.js)

- **createProduct** – stocke dans JSON + CSV.
- **getAllProducts** – renvoie JSON.
- **getProductById** – filtre JSON.
- **updateProduct** – met à jour JSON/CSV.
- **deleteProduct** – supprime JSON/CSV.
- **getProductsWithPromos** – fait un `http.get` vers une URL externe (gérée via `url.parse` pour ajouter query params), fusionne la promo reçue avec chaque produit avant renvoi.

Exercice 5 – Gestion de tâches

Routes (routes/taskRoutes.js)

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

router.post   ('/tasks',       taskController.createTask);
router.get    ('/tasks',       taskController.getAllTasks);
router.get    ('/tasks/:id',   taskController.getTaskById);
router.put    ('/tasks/:id',   taskController.updateTask);
router.delete ('/tasks/:id',   taskController.deleteTask);

module.exports = router;
```

Méthodes (`controllers/taskController.js`)

- **createTask** – utilise `util.promisify` pour lire/écrire JSON+CSV, émet un événement `taskCreated`.
- **getAllTasks** – lit JSON.
- **getTaskById** – filtre JSON.
- **updateTask** – modifie JSON/CSV, émet `taskUpdated`.
- **deleteTask** – supprime JSON/CSV, émet `taskDeleted`.

Exercice 6 – Conversion de fichiers

Routes (`routes/convertRoutes.js`)

```
const express = require('express');
const router = express.Router();
const convertController = require('../controllers/convertController');
const multer = require('multer');
const upload = multer({ dest: 'tmp/' });

router.post   ('/convert', upload.single('file'), convertController.convertFile);
router.get    ('/convert/:id',                    convertController.downloadConverted);
```

```
router.put    ('/convert/:id',          convertController.reprocessFile);
router.delete ('/convert/:id',           convertController.deleteConversion);

module.exports = router;
```

Méthodes (`controllers/convertController.js`)

- **convertFile** – stocke en `tmp/`, lit via `fs.createReadStream`, transforme en majuscules avec un `Transform` stream, calcule 2 hashes MD5 (`crypto`), enregistre méta JSON+CSV, renvoie `id` + hashes.
- **downloadConverted** – renvoie le flux du fichier converti.
- **reprocessFile** – refait la conversion (stream + hash) et met à jour JSON+CSV.
- **deleteConversion** – supprime fichiers et métadonnées JSON+CSV.
