

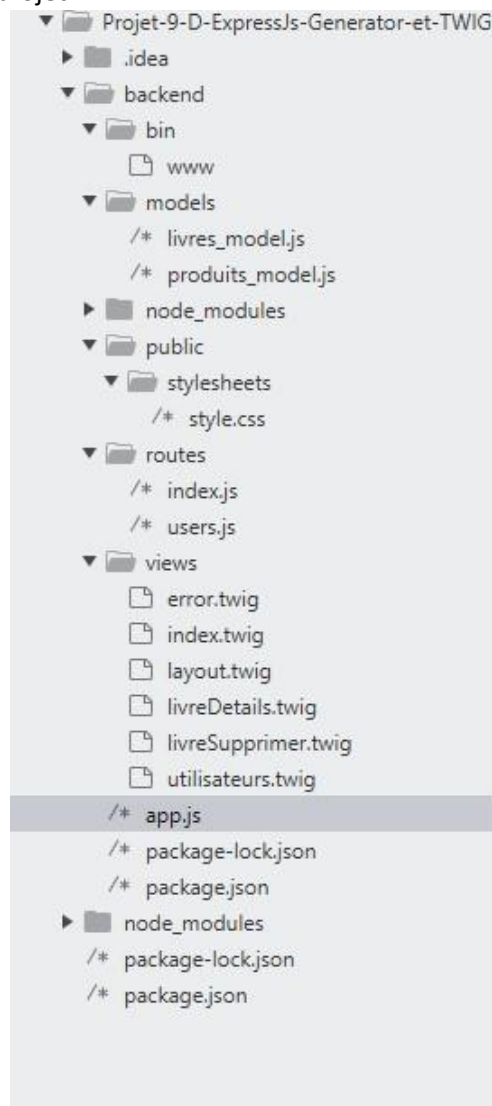
Projet 9 – D : ExpressJs Generator + Twig

Objectif :

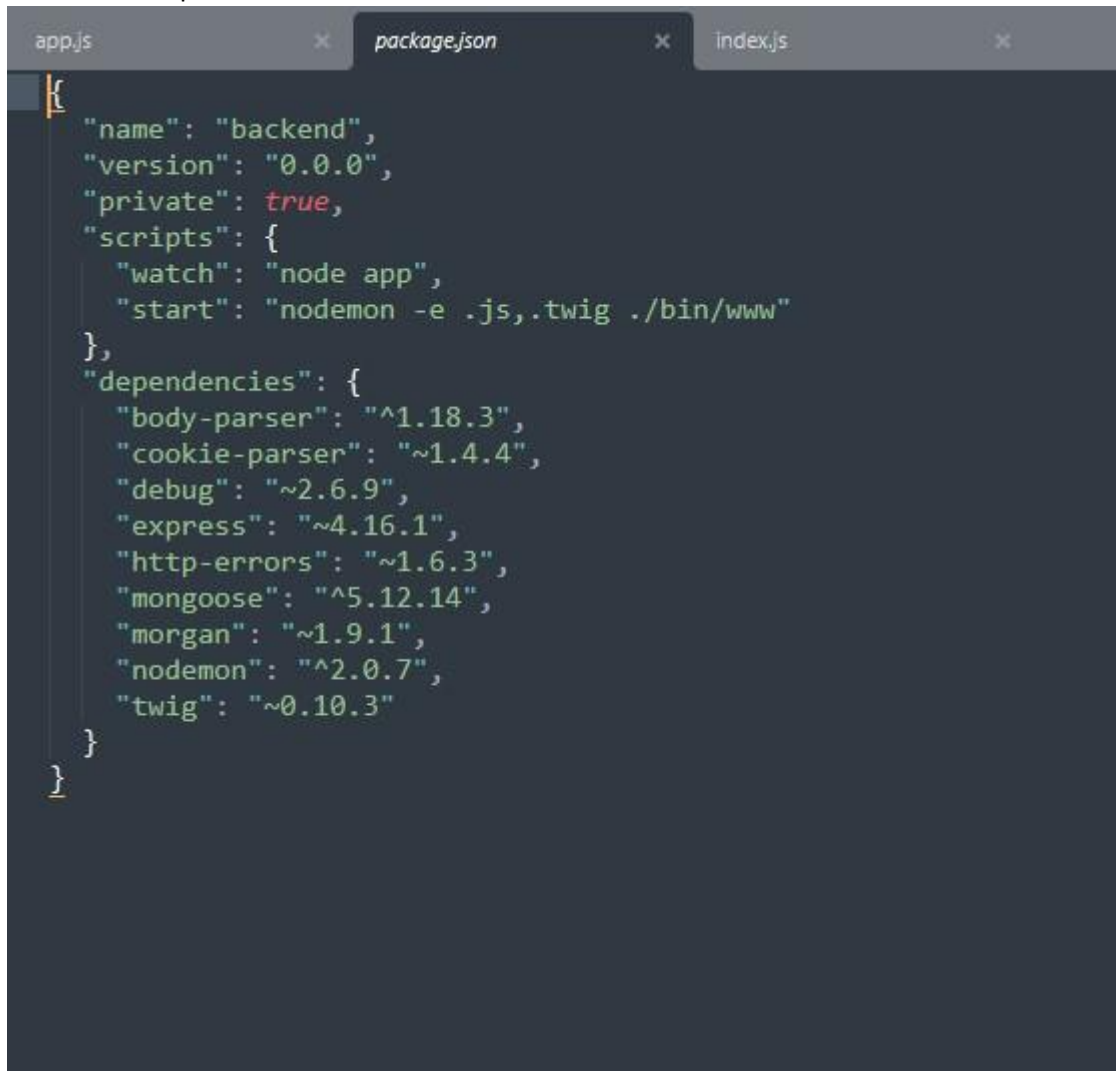
Construire une application web robuste à l'aide d'ExpressJs Generator et le gestionnaire de Template Twig. Cette application sera connectée à MongoDB et permettra de réaliser les 4 opérations de CRUD (GET, POST, PUT-PATCH, DELETE) via une API REST (Base de données ecommerce MongoDB + Collection livres)

Etapes :

1. Installer ExpressJs : `npm install express --save`
2. Installer ExpressJs Generator : `npm install express-generator -g`
3. Générer un fichier package.json : `npm init`
4. Générer une application de backend avec le moteur de Template Twig :
5. `express --view=twig backend`
6. Installer les dépendances : `npm install`
7. Démarrer votre serveur via `npm start`
8. Exemple de structure du projet :



9. Installer les dépendances suivantes :



```
{
  "name": "backend",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "watch": "node app",
    "start": "nodemon -e .js,.twig ./bin/www"
  },
  "dependencies": {
    "body-parser": "^1.18.3",
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "mongoose": "^5.12.14",
    "morgan": "~1.9.1",
    "nodemon": "^2.0.7",
    "twig": "~0.10.3"
  }
}
```

- Body parser : Middleware d'analyse corporelle Node.js.

Analysez les corps des requêtes entrantes dans un middleware avant vos gestionnaires, disponibles sous la req.body propriété.

- http-errors : Créez facilement des erreurs HTTP pour Express
- Mongoose : <https://mongoosejs.com/> (Connexion à MongoDB)
- Morgan : Middleware d'enregistrement de requête http
- nodemon est un utilitaire d'interface de ligne de commande (CLI) développé par @rem. Il enveloppe votre application Node, surveille le système de fichiers et redémarre automatiquement le processus.

10. Dans votre fichier de configuration app.js :

- Stocker et importer vos dépendances (http-errors, express, cookie-parser, morgan, etc...)
- Importer vos fichiers de routes
- Connecter votre application a MongoDB via Mongoose (3 statuts : connected + error + disconnected)
- Configurer Twig et express :

```

// view engine setup TWIG
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'twig');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());

app.use(express.static(path.join(__dirname, 'public')));

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: true
}))

app.use('/', indexRouter);
app.use('/users', usersRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;

```

11. Créer un dossier modèle et un fichier livres_model.js
12. Importer mongoose et créer un Schéma
13. Exporter votre modèle
14. Dans votre dossier route => dans votre fichier index.js
 - Importer Express et Router()
 - Importer votre fichier livres_model.js

TESTER TOUTES VOS REQUETES HTTP AVEC POSTMAN :

<https://practicalprogramming.fr/postman>

- Créer une première route à l'aide router et la méthode GET

- Créer une promesse dans laquelle vous aller boucler sur votre collection (schéma et / ou modèle) a l'aide de la fonction find()
- Ex : let livres = await livreModel.find()
- Si la requête fonctionne dans votre réponse (render()) appelé le nom de la vue twig (view/index.twig) + passé les valeurs de vos livres dans un objet

```
//Import du fichier models/livres_model.js
let livresModel = require('../models/livres_model');

/* GET home page. = fonction asynchrone async + await */
router.get('/', async function(req, res, next) {

  //Boucle de lecture de la collection produit = forEach()
  let produits = await produitsModel.find();

  //Appel de livre Schema
  let livres = await livresModel.find();
  //let produits = res.json()

  //Response + nom de la vue twig = views/index.twig + options clé/valeur produits: (cle pour la vue twig et produits = variable await + Model + find
  res.render('index', {
    title: 'ExpressJS + TWIG',
    produits: produits,
    books: livres
  });
});
```

- Répéter les opérations pour effectuer votre CRUD

```
//Afficher les details d'un livre + recup de id en paramètre url
router.get('/livres/:id', async (request, response) => {
  //On utilise ici la methode findOne + options _id = query id dans url
  let livresDetails = await livresModel.findById({
    '_id': request.params.id,
  })
  //Appel de la vues views/livreDetails.twig + cle / valeur livre = variable livreDetails = model + findOne
  response.render('livreDetails', {
    title: "La page détails",
    livre : livresDetails
  })
})

//Ajouter un livre
router.post('/ajouter-livre', async (request, response) =>{
  let savelivre = new livresModel(request.body);
  savelivre.save()
    .then(item => {
      response.redirect('/')
      console.log(item)
    })
    .catch(err => {
      response.status(400).send('erreur ajout du livre' + err)
    })
})

//Supprimer un livre
router.get('/livres/supprimer/:id', async (request, response) =>{
  let deletelivre = await livresModel.findByIdAndRemove({
    '_id': request.params.id,
  })
  console.log(deletelivre)

  response.render('livreSupprimer', {
    title: "livre supprimer",
    deletelivre: deletelivre
  })
})

//Export du module router utilisé dans app.js
module.exports = router;
```

- Dans vos vues aux extensions .twig
- Hérité du layout de base {% extends layout.twig %}
- Dans votre block body {% block body %} : Créer un formulaire avec 4 champs pour ajouter un livre et un bouton de soumission

- Spécifié à la balise <form> l'action (route ajouter-livre ci-dessus qui déclenche une requête http de type post) et la méthode = POST.
- Créer une boucle de lecture à l'aide de Twig {% for livre in books %} définit dans votre fichier index.js res.render books : livres (votre schéma)
- Pour chaque élément utilisé l'interpolation Twig {{ livre.nomLivre }} etc...
- Le fichier views/index.twig :

```

app.js      index.js      index.twig      livres_model.js
{% extends 'layout.twig' %}

{% block body %}
  <h4 class="blue-text card-panel center">{{ title }}</h4>
  <div class="card-panel col s12">
    <!--Ajouter un livre-->
    <form action="/ajouter-livre" method="post">
      <h3 class="orange-text center lighten-1 card-panel">AJOUTER UN LIVRE</h3>
      <div class="form-group">
        <label>Nom du Livre</label>
        <input class="form-control" type="text" name="nomLivre">
      </div>
      <div class="form-group">
        <label>Description du Livre</label>
        <input class="form-control" type="text" name="descriptionLivre">
      </div>
      <div class="form-group">
        <label>Prix du livre</label>
        <input class="form-control" type="text" name="prixLivre">
      </div>
      <div class="form-group">
        <label>Image du Livre</label>
        <input class="form-control" type="text" name="imageLivre">
      </div>
      <div class="form-group">
        <button class="col s6 waves-effect waves-light btn orange lighten-2" type="submit">Ajouter</button>
        <button type="reset" class="col s6 waves-effect waves-light btn purple lighten-2">vider les champs</button>
      </div>
    </form>
  </div>
  <!--Fin du formulaire-->

  <div class="row card-panel">
    {% for livre in books %}
      <div class="card col s4 offset-s1">
        <div class="card-image">
          
        </div>
        <div class="card-content">
          <p>{{ livre.nomLivre }}</p>
          <p>Description :</p>
          <p>{{ livre.nomLivre }}</p>
          <p>Référence : {{ livre._id }}</p>
          <p>{{ livre.prixLivre }} €</p>
          <div class="card-action col s12">
            <a href="livres/{{ livre._id }}" class="btn purple center lighten-2" style="width: 100%">Détails</a>
            <br>
            <a href="livres/supprimer/{{ livre._id }}" class="btn center red lighten-2" style="width: 100%">Supprimer</a>
          </div>
        </div>
      </div>
    {% endfor %}
  </div>
{% endblock %}

```

15. Créer les vues error.twig, livreDetails.twig, livreSupprimer.twig

16. Pour error.twig :

```
{% extends 'layout.twig' %}
```

```
{% block body %}
```

```
<h1>{{message}}</h1>
```

```
<h2>{{error.status}}</h2>
```

```
<pre>{{error.stack}}</pre>
```

{% endblock %}

17. Pour livreDetail.twig : dans votre fichier index.js on passe l'id en paramètre de URL et on utilise la fonction findById fournie par Mongoose

```
//Afficher les details d'un livre + recup de id en paramètre url
router.get('/livres/:id', async (request, response) => {
  //On utilise ici la methode findOne + options _id = query id dans url
  let livresDetails = await livresModel.findById({
    '_id': request.params.id,
  })
  //Appel de la vues views/livreDetails.twig + cle / valeur livre = variable livreDetails = model + findOne
  response.render('livreDetails', {
    title: "La page détails",
    livre : livresDetails
  })
})
```

18. Pour l'affichage dans votre vue Twig on utilise livre : l'objet passé dans votre render pour afficher chaque élément
19. Exemple : {{ livre.nomLivre }}
20. Pour ajouter un livre : le formulaire grâce a l'attribut action appel l'url ajouter-livre et la fonction suivante dans votre index.js :

```
//Ajouter un livre
router.post('/ajouter-livre', async (request, response) =>{
  let saveLivre = new livresModel(request.body);
  saveLivre.save()
    .then(item => {
      response.redirect('/')
      console.log(item)
    })
    .catch(err => {
      response.status(400).send('erreur ajout du livre' + err)
    })
})
```

- Lors de l'instance du model on passe en paramètre request.body capable de récupérer les données du formulaire.
- En effet les requêtes http sont composée de 2 éléments, des options d'entête (headers) et le corps de la requête(body), ExpressJs m'est à disposition request.body pour récupérer des valeurs d'un formulaire via la méthode POST
- Request.body :
Contient des paires clé-valeur de données soumises dans le corps de la demande. Par défaut, il n'est pas défini et est renseigné lorsque vous utilisez un middleware d'analyse corporelle tel que express.json() ou express.urlencoded() qui sont définis dans votre fichier de configuration ExpressJs (ici app.js)
- On utilise également la fonction save() fournie par Mongoose pour persister les données, on effectue ensuite une promesse qui redirige si tout fonctionne où nous affiche une erreur (résolve ou reject).
- Supprimer un livre : comme pour les détails, on passe l'id du livre en paramètre de URL et dans une requête asynchrone, on utilise findByIdAndRemove() fournis par Mongoose et request.params.id pour récupérer l'id :

https://mongoosejs.com/docs/api.html#model_Model.findByIdAndRemove


```
//Supprimer un livre
router.get('/livres/supprimer/:id', async (request, response) =>{
  let deleteLivre = await livresModel.findByIdAndRemove({
    '_id': request.params.id,
  })
  console.log(deleteLivre)

  response.render('livreSupprimer', {
    title: "livre supprimer",
    deleteLivre: deleteLivre
  })
})
})
```

- On utilise ensuite deleteLivre directement dans notre vue Twig pour supprimer une valeur de notre collection

```
{% extends 'layout.twig' %}

{% block body %}
<h4 class="blue-text card-panel center">{{title}}</h4>

<div class="card-panel col s12">
  <h2 class="green-text card-panel">Supprimer le livre : {{ dump(deleteLivre.nomLivre) }}</h2>
  <ul class="collection">
    <li class="collection-item">
      
    </li>
    <li class="collection-item">{{ deleteLivre.nomLivre }}</li>
    <li class="collection-item">
      <p>Description :</p>
      <p>{{ deleteLivre.nomLivre }}</p>
    </li>
    <li class="collection-item">
      <p>Référence : {{ deleteLivre._id }}</p>
    </li>
    <li class="collection-item">{{ deleteLivre.prixLivre }} €</li>
    <li>
      <a href="/" class="btn purple lighten-2">Retour</a>
    </li>
  </ul>
  <a href="/" class="btn green lighten-1">CONFIRMER</a>
</div>

{% endblock %}
```

Aller + loin :

- Ajouter un formulaire dans une fenêtre modal pour éditer vos livres
- (Dans index.js créer une requête http méthode PUT pour imposer la modification de tous les champs + le modale dans index.twig
- Créer une autre collection et répéter les opérations de CRUD