Serveur JSON et Système d'Upload d'Images

Vue d'ensemble

Ce projet intègre maintenant un serveur JSON pour simuler les données de l'application et un système d'upload d'images qui stocke les URLs dans le serveur JSON.

Fonctionnalités

1. Serveur JSON (json-server)

- **Port**: 3000
- **Base de données**: `db.json`
- **Routes personnalisées**: `routes.json`
- **Données mockées**: Utilisateurs, catégories, produits

2. Service d'Upload d'Images

- **Upload d'images**: Supporte JPEG, PNG, GIF, WebP (max 5MB)
- **Stockage des URLs**: Les URLs des images sont stockées dans le serveur JSON
- **Génération d'URLs**: Génération automatique d'URLs d'images via Picsum
- **Gestion des métadonnées**: Taille, type MIME, date d'upload

Démarrage du projet

Développement (Recommandé)

```
npm run start:dev
```

PROF

Cette commande démarre à la fois:

- Le serveur JSON sur http://localhost:3000
- L'application Angular sur http://localhost:4201

Manuellement

```
# Terminal 1 - Démarrer le serveur JSON
npm run json-server

# Terminal 2 - Démarrer l'application Angular
ng serve --port 4201
```

En arrière-plan

```
npm run json-server:bg
```

Structure des données

db.json

```
{
  "users": [...],      // Utilisateurs
  "categories": [...], // Catégories de produits
  "products": [...],   // Produits avec images
  "uploads": []        // Images uploadées
}
```

Endpoints API

Utilisateurs

- `GET /users` - Tous les utilisateurs
- `GET /users/:id` - Utilisateur spécifique
- `GET /users?role=SELLER` - Utilisateurs par rôle

Catégories

- `GET /categories` - Toutes les catégories
- `GET /categories/:id` - Catégorie spécifique

Produits

- `GET /products` - Tous les produits
- `GET /products?sellerId=2` - Produits par vendeur
- `GET /products?status=APPROVED` - Produits approuvés
- `GET /products?category=Electronique` - Produits par catégorie
- `POST /products` - Créer un produit
- `PATCH /products/:id` - Mettre à jour un produit

PROF

Upload d'images

- `GET /uploads` - Toutes les images uploadées
- `POST /uploads` - Uploader une nouvelle image
- `DELETE /uploads/:id` - Supprimer une image

Utilisation du service d'upload

Dans un composant Angular

```
```

```
import { ImageUploadService } from '../core/services/image-
upload.service';

export class MonComposant {
  constructor(private imageUploadService: ImageUploadService) {}

  onFileSelected(event: any) {
    const file: File = event.target.files[0];

    if (this.imageUploadService.isValidImageFile(file)) {
      this.imageUploadService.uploadImage(file, userId).subscribe({
        next: (imageUrl: string) => {
          console.log('Image uploadée:', imageUrl);
          // Utiliser l'URL de l'image
        },
        error: (error) => {
          console.error('Erreur d\'upload:', error);
        }
      });
    }
  }

  // Upload multiple images
  onFilesSelected(event: any) {
    const files: File[] = Array.from(event.target.files);

    this.imageUploadService.uploadImages(files, userId).subscribe({
      next: (imageUrls: string[]) => {
        console.log('Images uploadées:', imageUrls);
      },
      error: (error) => {
        console.error('Erreur d\'upload:', error);
      }
    });
  }
}
```

PROF

Dans le template HTML

```
<input type="file" accept="image/*" (change)="onFileSelected($event)">
<button (click)="fileInput.click()">Choisir une image</button>
<input #fileInput type="file" accept="image/*"
(change)="onFileSelected($event)" style="display: none;">
```

Configuration

Limites d'upload

- **Taille maximale**: 5MB par image
- **Types supportés**: JPEG, PNG, GIF, WebP

Modification des limites

Dans `image-upload.service.ts`:

```
// Modifier la taille maximale
getMaxFileSize(): number {
  return 10 * 1024 * 1024; // 10MB
}

// Modifier les types supportés
isValidImageFile(file: File): boolean {
  const validTypes = ['image/jpeg', 'image/jpg', 'image/png',
'image/gif', 'image/webp', 'image/svg+xml'];
  // ...
}
```

Intégration avec les produits

Le système d'upload est conçu pour fonctionner avec les produits:

1. **Upload d'images** pour un produit
2. **Stockage des URLs** dans le produit
3. **Persistance** via le serveur JSON

```
// Exemple d'ajout de produit avec images
addProductWithImages(productData: any, images: File[]) {
  this.imageUploadService.uploadImages(images).subscribe({
    next: (imageUrls: string[]) => {
      const product = {
        ...productData,
        images: imageUrls,
        createdAt: new Date(),
        updatedAt: new Date()
      };

      this.mockDataService.addProduct(product).subscribe({
        next: (newProduct) => {
          console.log('Produit créé avec images:', newProduct);
        }
      });
    }
  });
}
```

Développement

Ajout de nouvelles données

Pour ajouter de nouvelles données mockées, modifiez le fichier `db.json`:

```
{
  "products": [
    {
      "id": 9,
      "name": "Nouveau produit",
      "images": ["https://picsum.photos/400/400?random=123"],
      // ... autres propriétés
    }
  ]
}
```

Routes personnalisées

Pour ajouter des routes personnalisées, modifiez `routes.json`:

```
{
  "/api/products/search": "/products?q=:query"
}
```

Dépannage

Problèmes courants

1. **Port 3000 déjà utilisé**

PROF

```
# Changer le port du serveur JSON
npx json-server --watch db.json --port 3001
```

2. **CORS issues**
   - Le serveur JSON est configuré pour accepter les requêtes cross-origin
   - Vérifiez que l'URL du serveur JSON correspond dans `ApiService`
3. **Données non persistantes**
   - Les modifications sont sauvegardées dans `db.json`
   - Redémarrez le serveur JSON pour voir les changements

Logs

Pour voir les logs du serveur JSON:

```
npm run json-server
```

Les requêtes HTTP seront affichées dans la console.

Production

Pour la production, remplacez le serveur JSON par une vraie API REST:

1. Remplacez les URLs dans `ApiService`
2. Implémentez les endpoints correspondants
3. Gérez l'upload d'images côté serveur
4. Configurez la persistance des données

Sécurité

⚠ **Important**: Ce serveur JSON est destiné au développement uniquement. Pour la production:

- Implémentez une authentification
- Validez les uploads côté serveur
- Sécurisez les endpoints
- Gérez les permissions utilisateurs