🧪 Tests API Porsche - Architecture Refactorisée

📋 Vue d'ensemble

Cette suite de tests a été complètement refactorisée selon les principes **SOLID** et les bonnes pratiques **AGILE** pour remplacer l'ancien fichier monolithique de 1581 lignes.

✅ Améliorations clés

- **Modularité**: Architecture en dossiers avec séparation des responsabilités
- **Réutilisabilité**: Client API centralisé et fixtures partagées
- **Maintenabilité**: Tests isolés et indépendants
- **Testabilité**: Utilisation de Jest avec assertions strictes
- **Configuration**: Variables d'environnement et configuration centralisée
- **Couverture**: Tests complets avec génération de rapports

🏗️ Architecture

```
tests/
├── setup/                    # Configuration et utilitaires
│   ├── config.js            # Configuration centralisée
│   ├── fixtures.js          # Données de test réutilisables
│   └── helpers.js           # Fonctions utilitaires
│
├── utils/                    # Utilitaires transversaux
│   └── api-client.js        # Client HTTP réutilisable
│
└── integration/             # Tests d'intégration
    ├── admin/               # Tests partie Admin
    │   ├── auth.test.js     # Authentification admin
    │   ├── couleurs.test.js # CRUD couleurs
    │   └── voitures.test.js # CRUD voitures + réservations
    │
    └── user/                # Tests partie User
        ├── auth.test.js     # Authentification user
        └── commandes.test.js # Panier et commandes
```

🚀 Installation

```
# Installer les dépendances de test
npm install --save-dev jest @jest/globals

# Ou si déjà fait
npm install
```

🎯 Utilisation

Lancer tous les tests

```
npm test
```

Tests en mode watch (développement)

```
npm run test:watch
```

Tests avec couverture de code

```
npm run test:coverage
```

Tests par catégorie

```
# Tests admin uniquement
npm run test:admin

# Tests user uniquement
npm run test:user

# Tests d'authentification uniquement
npm run test:auth
```

Tests legacy (ancien système)

```
npm run test:legacy
```

⚙️ Configuration

Variables d'environnement

Créer un fichier `.env.test` (optionnel):

```
# API
TEST_API_URL=http://localhost:3000
TEST_TIMEOUT=5000
```

```
# Credentials Admin
ADMIN_EMAIL=admin@porsche.com
ADMIN_PASSWORD=Admin123!@#

# Configuration des tests
TEST_RETRY_ATTEMPTS=3
TEST_RETRY_DELAY=1000
TEST_CLEANUP=true
```

Modification de la configuration

Éditer `tests/setup/config.js` pour personnaliser:

- URL de l'API
- Credentials de test
- Délais et timeouts
- Comportement du cleanup

📝 Écrire de nouveaux tests

Exemple de test simple

```
import { describe, test, expect, beforeAll } from "@jest/globals";
import { apiClient } from "../../utils/api-client.js";
import { config } from "../../setup/config.js";

describe("Mon nouveau test", () => {
  beforeAll(async () => {
    await apiClient.login(config.credentials.admin, true);
  });

  test("should do something", async () => {
    const response = await apiClient.get("/endpoint");
    expect(response.status).toBe(200);
  });
});
```

Utiliser les fixtures

```
import {
  createUserFixture,
  couleursExterieurFixtures,
} from "../../setup/fixtures.js";

const userData = createUserFixture();
const couleurData = couleursExterieurFixtures[0];
```

Utiliser les assertions personnalisées

```
import { assertions } from "../../setup/helpers.js";

assertions.isValidUser(user);
assertions.isValidVoiture(voiture);
assertions.isValidCouleur(couleur);
```

🔧 API Client

Le client API (`api-client.js`) fournit des méthodes dédiées pour chaque endpoint:

```
// Authentification
await apiClient.login(credentials, isAdmin);
await apiClient.register(userData);

// Couleurs
await apiClient.createCouleurExterieur(data);
await apiClient.getAllCouleursExterieur();
await apiClient.updateCouleurExterieur(id, data);

// Voitures
await apiClient.createVoiture(data);
await apiClient.getAllVoitures();

// Réservations
await apiClient.createReservation(userId, data);
await apiClient.checkDisponibilite(voitureId, date);

// Commandes
await apiClient.getOrCreatePanier();
await apiClient.addLigneCommande(data);
await apiClient.viderPanier();
```

📊 Principes SOLID appliqués

Single Responsibility (SRP)

- **config.js**: Gestion de la configuration uniquement
- **api-client.js**: Communication HTTP uniquement
- **fixtures.js**: Données de test uniquement
- **helpers.js**: Utilitaires réutilisables uniquement

Open/Closed (OCP)

- Le client API est extensible sans modification
- Nouveaux tests ajoutables sans modifier l'existant

Liskov Substitution (LSP)

- Les fixtures peuvent être utilisées de manière interchangeable
- Les assertions sont polymorphes

Interface Segregation (ISP)

- Client API avec méthodes spécifiques par ressource
- Pas de dépendances inutiles entre modules

Dependency Inversion (DIP)

- Tests dépendent d'abstractions (config, fixtures)
- Pas de hardcoding dans les tests

🆚 Comparaison Ancien vs Nouveau

Critère	Ancien	Nouveau
Lignes de code	1581 lignes (1 fichier)	~400 lignes/fichier (8 fichiers)
Maintenabilité	❌ Difficile	✅ Facile
Réutilisabilité	❌ Code dupliqué	✅ DRY respecté
Isolation	❌ Tests dépendants	✅ Tests indépendants
Framework	❌ Custom	✅ Jest
Configuration	❌ Hardcodé	✅ Variables d'env
Assertions	❌ Basiques	✅ Strictes et typées
Couverture	❌ Aucune	✅ Rapports détaillés

🪲 Debugging

Verbose mode

```
npm test -- --verbose
```

Run un seul fichier

```
npm test -- tests/integration/admin/auth.test.js
```

Run un seul test

```
```

```
npm test -- -t "should login successfully"
```

Voir les handles ouverts

```
npm test -- --detectOpenHandles
```

📈 Métriques

Les tests fournissent des métriques détaillées:

- **Temps d'exécution** par test
- **Taux de succès** global
- **Couverture de code** par fichier
- **Branches non testées**

🤝 Contribution

Ajouter un nouveau test

1. Créer un fichier `*.test.js` dans le dossier approprié
2. Importer les dépendances nécessaires
3. Structurer avec `describe` et `test`
4. Utiliser `beforeAll`/`afterAll` pour setup/cleanup
5. Ajouter des assertions strictes

Bonnes pratiques

✅ **À faire**:

- Tests isolés et indépendants
- Utiliser les fixtures existantes
- Nettoyer après les tests (afterAll)
- Assertions explicites
- Noms de tests descriptifs

❌ **À éviter**:

- Tests dépendants de l'ordre d'exécution
- Hardcoding de données
- Tests sans assertions
- Timeouts trop courts
- Ignorer les erreurs

📚 Ressources

- Jest Documentation

- SOLID Principles
- API Testing Best Practices

🔐 Sécurité

⚠ **Important**: Ne jamais commit de credentials réels dans les fichiers de config ou de test.

Utiliser des variables d'environnement pour les informations sensibles.

📞 Support

Pour toute question ou problème:

1. Vérifier les logs des tests
2. Vérifier la configuration dans `config.js`
3. S'assurer que le serveur est démarré
4. Vérifier les credentials admin

Développé avec ❤️ suivant les principes SOLID et AGILE