Module de Personnalisation - MetaSign

Vue d'ensemble

Le module de personnalisation gère les profils utilisateurs et l'adaptation personnalisée de l'apprentissage de la Langue des Signes Française (LSF). Il fournit des services pour créer, mettre à jour et analyser les profils d'apprentissage individuels.

Architecture

```
personalization/
├── ProfileManager.ts                  # Gestionnaire principal des profils
├── interfaces/
│   ├── IUserProfileManager.ts         # Interface du gestionnaire
│   └── IUserProfileStorage.ts         # Interface de stockage
├── __tests__/
│   └── ProfileManager.test.ts         # Tests unitaires
├── index.ts                           # Point d'entrée du module
└── README.md                          # Documentation
```

Services Disponibles

🧬 UserProfileManager

Responsabilité : Gestion complète des profils utilisateurs avec personnalisation adaptative

Fonctionnalités principales :

- Création et récupération de profils utilisateurs
- Mise à jour des compétences basée sur les performances

- Analyse des préférences d'apprentissage
- Suivi de la progression
- Calcul de compatibilité de contenu
- Cache intelligent avec TTL

💾 InMemoryUserProfileStorage

Responsabilité : Stockage en mémoire pour les tests et développement

Fonctionnalités :

- Stockage temporaire des profils
- Operations CRUD complètes
- Idéal pour les tests unitaires

Utilisation

Installation et Configuration

```typescript
import {
    createUserProfileManager,
    createInMemoryStorage,
    IUserProfileStorage
} from '@/ai/services/learning/human/personalization';

// Utilisation basique avec stockage en mémoire
const storage = createInMemoryStorage();
const profileManager = createUserProfileManager(storage);

// Ou avec stockage personnalisé
class DatabaseStorage implements IUserProfileStorage {
    // Implémentation personnalisée...
}

const customStorage = new DatabaseStorage();
const manager = createUserProfileManager(customStorage);
```

Gestion des Profils

```typescript
// Créer ou récupérer un profil
const profile = await profileManager.getOrCreateProfile('user-123');

// Mettre à jour le profil
const updatedProfile = await profileManager.updateProfile('user-123', {
    skillLevel: CompetencyLevel.INTERMEDIATE,
    interests: ['mathematics', 'geometry']
});

// Analyser les préférences
const preferences = await profileManager.analyzePreferences('user-123');
```

Suivi des Performances

```typescript
// Données de performance
const performanceData: UserPerformanceData = {
    userId: 'user-123',
    exercises: [
        {
            id: 'ex1',
            conceptId: 'basic-signs',
            successRate: 0.85,
            completionTime: 120,
            attempts: 2
        }
    ],
    quizzes: [],
    interactionPatterns: {
        hesitations: ['complex-grammar'],
        repeatedMistakes: ['finger-spelling'],
        avoidedTopics: []
    },
    lastActivity: new Date()
};

// Mettre à jour les compétences
const updatedProfile = await profileManager.updateSkills('user-123', performanceData);
```

Suivi de Progression

typescript

```typescript
// Données de progression
const progressData: ProgressData = {
    userId: 'user-123',
    activityId: 'lesson-basic-greetings',
    completionStatus: 'completed',
    score: 85,
    timeSpent: 300,
    masteredConcepts: ['greeting-signs', 'polite-expressions'],
    timestamp: new Date()
};

// Enregistrer la progression
await profileManager.trackProgress('user-123', progressData);
```

Calcul de Compatibilité

typescript

```typescript
// Calculer la compatibilité avec du contenu
const compatibility = await profileManager.calculateContentCompatibility(
    'user-123',
    ['mathematics', 'geometry', 'algebra']
);

console.log(`Compatibilité: ${compatibility * 100}%`);
```

Types Principaux

ExtendedUserProfile

typescript

```typescript
interface ExtendedUserProfile extends UserProfile {
    userId: string;
    skillLevel: CompetencyLevel;
    skills?: Record<string, number>;
    interests?: string[];
    preferences?: LearningPreferences;
    history?: LearningHistory;
    metadata?: UserMetadata;
}
```

LearningPreferences

typescript

```typescript
interface LearningPreferences {
    preferredPace: number;
    preferredLearningStyle: LearningStyle;
    preferredContentTypes: string[];
    goalOrientation: 'mastery' | 'performance' | 'exploration';
    pacePreference: 'slow' | 'moderate' | 'fast';
    assistanceLevel: number;
    adaptivityLevel: number;
    requiresStructure: boolean;
    prefersFeedback: boolean;
}
```

UserPerformanceData

```typescript
interface UserPerformanceData {
    userId: string;
    exercises: Array<{
        id: string;
        conceptId: string;
        successRate: number;
        completionTime: number;
        attempts: number;
    }>;
    quizzes: Array<{
        id: string;
        conceptIds: string[];
        score: number;
        timeSpent: number;
    }>;
    interactionPatterns: {
        hesitations: string[];
        repeatedMistakes: string[];
        avoidedTopics: string[];
    };
    lastActivity: Date;
}
```

Configuration

Options par défaut

```typescript
const DEFAULT_PROFILE_CONFIG = {
    cacheTTL: 15 * 60 * 1000,    // 15 minutes de cache
    masteryThreshold: 0.8,       // Seuil de maîtrise (80%)
    skillThreshold: 0.6          // Seuil de compétence pour intérêts
};
```

Configuration avancée

```typescript
// Configuration du gestionnaire
const manager = new UserProfileManager(storage);

// Les seuils peuvent être ajustés via les méthodes privées
// ou par héritage pour des besoins spécifiques
```

Tests

Exécution des tests

```bash
# Tests unitaires
npm test src/ai/services/learning/human/personalization/__tests__/

# Tests avec couverture
npm run test:coverage -- src/ai/services/learning/human/personalization/

# Tests spécifiques
npm test ProfileManager.test.ts
```

Structure des tests

Les tests couvrent :

- Création et récupération de profils
- Mise à jour des compétences
- Gestion des préférences
- Calcul de compatibilité
- Stockage et récupération
- Gestion des erreurs

Intégration avec MetaSign

Dépendances

- `@/ai/utils/Logger` - Système de logging
- `@/ai/services/learning/types` - Types du module d'apprentissage

Services connectés

- **AdaptiveLearningSystem** - Utilise les profils pour l'adaptation
- **LearningMetricsCollector** - Fournit les données de performance
- **RealTimeAdapter** - Consomme les profils en temps réel

Bonnes Pratiques

1. Gestion des erreurs

typescript

```typescript
try {
    const profile = await profileManager.getProfile(userId);
} catch (error) {
    if (error.message.includes('not found')) {
        // Créer un nouveau profil
        const newProfile = await profileManager.getOrCreateProfile(userId);
    } else {
        // Gérer l'erreur
        logger.error('Erreur profil:', error);
    }
}
```

2. Mise à jour efficace

typescript

```typescript
// Préférer les mises à jour partielles
await profileManager.updateProfile(userId, {
    lastActive: new Date(),
    experience: profile.experience + 10
});

// Plutôt que de récupérer et réécrire tout le profil
```

3. Cache et performance

typescript

```typescript
// Le cache est automatique, mais peut être invalidé si nécessaire
// Les profils sont mis en cache pendant 15 minutes par défaut
```

4. Stockage personnalisé

```typescript
class DatabaseUserProfileStorage implements IUserProfileStorage {
    async saveProfile(profile: ExtendedUserProfile): Promise<void> {
        // Implémentation avec base de données réelle
        await this.db.profiles.upsert(profile);
    }

    async getProfile(userId: string): Promise<ExtendedUserProfile | null> {
        return await this.db.profiles.findByUserId(userId);
    }

    // ... autres méthodes
}
```

Extensibilité

Ajouter de nouveaux types de données

typescript

```typescript
// Étendre ExtendedUserProfile
interface MyCustomProfile extends ExtendedUserProfile {
    customField: string;
    additionalMetrics: Record<string, number>;
}

// Créer un gestionnaire personnalisé
class CustomProfileManager extends UserProfileManager {
    // Méthodes personnalisées
}
```

Analyses personnalisées

typescript

```typescript
class AnalyticsProfileManager extends UserProfileManager {
    async analyzeAdvancedPatterns(userId: string) {
        const profile = await this.getProfile(userId);
        // Analyses personnalisées
        return customAnalysis;
    }
}
```

Support et Contribution

- **Documentation :** Voir les interfaces TypeScript pour les détails d'API
- **Tests :** Suivre les patterns existants dans `__tests__/`
- **Types :** Respecter `exactOptionalPropertyTypes: true`
- **Logging :** Utiliser Logger.getInstance() pour tous les logs

- **Erreurs :** Toujours wrapper les erreurs avec un contexte approprié

Le module de personnalisation est conçu pour être extensible, performant et facile à intégrer avec le reste du système MetaSign.