CaféConnect AI - Documentazione per Sviluppatori

Panoramica dell'Architettura

CaféConnect AI è una piattaforma modulare progettata per offrire esperienze conversazionali personalizzate in contesti commerciali come bar, caffetterie, ristoranti e negozi. L'architettura è basata sui principi di:

- **Estensibilità**: Facilità nell'aggiungere nuovi provider AI, funzioni e componenti UI
- **Configurabilità**: Tutti gli aspetti sono personalizzabili senza modificare il codice
- **Modularità**: Componenti disaccoppiati e facilmente sostituibili
- **Riutilizzabilità**: Pattern comuni e astrazioni che possono essere riusati in contesti diversi

Tecnologie Utilizzate

- **Frontend**: React, TypeScript, CSS
- **State Management**: React Context API
- **Persistenza Locale**: localStorage (per configurazioni e preferenze utente)
- **AI**: Integrazione con provider AI multipli (OpenAI, Claude, ecc.)

Struttura del Progetto

L'applicazione è organizzata secondo la seguente struttura:

```
src/
├── api/                # API mock per lo sviluppo locale
├── components/         # Componenti React
├── config/             # Sistema di configurazione centralizzato
├── contexts/           # React Contexts
├── hooks/              # React Hooks personalizzati
├── initialization/     # Logica di inizializzazione dell'app
├── services/           # Servizi core dell'applicazione
│   ├── ai/             # Servizi per integrazione AI
│   ├── catalog/        # Gestione catalogo (menu/prodotti)
│   ├── function/       # Registry funzioni custom
│   ├── theme/          # Gestione temi
│   ├── ui/             # Generazione componenti UI dinamici
│   └── user/           # Gestione contesto utente
├── styles/             # Fogli di stile globali
├── types/              # Definizioni TypeScript
└── utils/              # Utility generiche
```

Componenti Principali

ConfigManager

Il `ConfigManager` è il cuore del sistema di configurazione che permette la personalizzazione senza necessità di modificare il codice.

```typescript
// Esempio di utilizzo
import { configManager } from '../config/ConfigManager';

// Leggere la configurazione
const businessConfig = configManager.getSection('business');

// Aggiornare la configurazione
configManager.updateSection('ui', { enableSuggestions: true });
```

AppInitializer

Responsabile per l'inizializzazione ordinata di tutti i servizi:

```typescript
// Esempio di inizializzazione
import { appInitializer } from './initialization/AppInitializer';

// Inizializzazione con configurazione remota
await appInitializer.initialize('https://api.example.com/config');

// Reinizializzazione dopo modifiche
await appInitializer.reinitialize();
```

AIProviderRegistry

Sistema di registrazione per provider AI, permette di aggiungere nuovi provider senza modificare il codice esistente:

```typescript
// Registrazione di un nuovo provider
import { aiProviderRegistry } from './services/ai/AIProviderRegistry';
import { CustomAIProvider } from './services/ai/providers/CustomAIProvider';

aiProviderRegistry.registerProvider('custom', (config) => new CustomAIProvider(config));
```

FunctionRegistry

Sistema di registrazione per funzioni personalizzate:

typescript

```typescript
// Registrazione di una nuova funzione
import { functionRegistry } from './services/function/FunctionRegistry';

functionRegistry.registerFunction({
  name: 'get_special_offers',
  description: 'Ottiene le offerte speciali del giorno',
  parameters: { /* ... */ },
  handler: async (params) => {
    // Implementazione della funzione
    return { /* ... */ };
  }
});
```

Estensione dell'Applicazione

Aggiungere un Nuovo Provider AI

1. Creare una nuova classe che implementa l'interfaccia `IAIProvider`:

typescript

```typescript
// services/ai/providers/NewProvider.ts
import { IAIProvider, IMessageProvider } from '../interfaces/IAIProvider';
import { AIProviderConfig } from '../../../types/AIProvider';

export class NewProvider implements IAIProvider, IMessageProvider {
  name = 'New Provider';

  constructor(private config: AIProviderConfig) {
    console.log('New provider initialized with:', config);
  }

  providerName(): string {
    return this.name;
  }

  async sendMessage(prompt: string, options?: any): Promise<string> {
    // Implementazione dell'invio del messaggio
    // ...
    return 'Response from new provider';
  }
}
```

2. Registrare il provider nel sistema:

typescript

```typescript
// In initialization/AppInitializer.ts
import { NewProvider } from '../services/ai/providers/NewProvider';

// All'interno del metodo registerAIProviders:
aiProviderRegistry.registerProvider('newprovider', (config) => new NewProvider(config));
```

Aggiungere una Nuova Funzione

1. Definire e registrare la funzione:

typescript

```typescript
functionRegistry.registerFunction({
  name: 'calculate_discount',
  description: 'Calcola lo sconto applicabile a un ordine',
  parameters: {
    type: 'object',
    properties: {
      userId: {
        type: 'string',
        description: 'ID dell\'utente'
      },
      orderTotal: {
        type: 'number',
        description: 'Totale dell\'ordine'
      }
    },
    required: ['userId', 'orderTotal']
  },
  handler: async (params) => {
    // Logica per calcolare lo sconto
    const discount = params.orderTotal > 50 ? 0.1 : 0.05;
    return {
      discountPercent: discount * 100,
      discountAmount: params.orderTotal * discount,
      finalTotal: params.orderTotal * (1 - discount)
    };
  },
  uiMetadata: {
    displayType: 'card',
    cardTemplate: 'discount-card'
  }
});
```

Aggiungere un Nuovo Componente UI

1. Creare il componente React:

tsx

```tsx
// components/ui/DiscountCard.tsx
import React from 'react';

interface DiscountCardProps {
  discountPercent: number;
  discountAmount: number;
  finalTotal: number;
  id: string;
  onAction?: (action: string, payload: any) => void;
}

export const DiscountCard: React.FC<DiscountCardProps> = ({
  discountPercent,
  discountAmount,
  finalTotal,
  id,
  onAction
}) => {
  return (
    <div className="discount-card" id={id}>
      <div className="discount-header">
        <h3>Il tuo sconto</h3>
      </div>
      <div className="discount-content">
        <div className="discount-value">{discountPercent}%</div>
        <p>Hai risparmiato: {discountAmount.toFixed(2)}€</p>
        <p>Totale finale: {finalTotal.toFixed(2)}€</p>
      </div>
    </div>
  );
};
```

2. Registrare il componente nel registry:

typescript

```typescript
// components/ui/registry/ComponentRegistration.tsx
import { DiscountCard } from '../DiscountCard';

// All'interno della funzione registerComponents
uiComponentRegistry.register('discountCard', (component, onAction) => (
  <DiscountCard
    discountPercent={component.data.discountPercent}
    discountAmount={component.data.discountAmount}
    finalTotal={component.data.finalTotal}
    id={component.id}
    onAction={onAction}
  />
));
```

API Chiave

ConfigManager

typescript

```typescript
// Ottieni l'intera configurazione
getConfig(): AppConfig;

// Ottieni una sezione specifica
getSection<K extends keyof AppConfig>(section: K): AppConfig[K];

// Aggiorna una sezione
updateSection<K extends keyof AppConfig>(section: K, data: Partial<AppConfig[K]>): void;

// Carica configurazione remota
loadConfig(configUrl: string): Promise<void>;

// Carica configurazione locale
loadLocalConfig(config: Partial<AppConfig>): void;
```

AIService

typescript

```typescript
// Invia un messaggio e ottieni una risposta
sendMessage(message: string, userContext: UserContext): Promise<AIResponse>;

// Ottieni la cronologia della conversazione
getConversationHistory(): Message[];

// Resetta la conversazione
resetConversation(): void;

// Cambia il provider AI
changeProvider(provider: string, config: AIProviderConfig): void;
```

FunctionRegistry

typescript

```typescript
// Registra una nuova funzione
registerFunction(functionDef: FunctionDefinition): void;

// Ottieni tutte le funzioni registrate
getAllFunctions(): FunctionDefinition[];

// Esegui una funzione
executeFunction(functionName: string, parameters: any, useMock?: boolean): Promise<FunctionCall
```

CatalogService

typescript

```typescript
// Ottieni elementi menu filtrati per il momento della giornata
getMenuItems(): Promise<MenuItem[]>;

// Ottieni tutti gli elementi menu
getAllMenuItems(): Promise<MenuItem[]>;

// Ottieni prodotti filtrati per categoria
getProducts(category?: string): Promise<Product[]>;

// Aggiorna il catalogo
refreshCatalog(): Promise<void>;
```

ThemeService

```typescript
// Ottieni il tema corrente
getCurrentTheme(): ThemeColors;

// Applica un tema personalizzato
applyTheme(theme: Partial<ThemeColors>): void;

// Genera un tema da un colore primario
generateThemeFromColor(primaryColor: string): ThemeColors;
```

Best Practices

1. **Utilizzo dei servizi**:
   - Accedi ai servizi tramite i singleton esportati o il hook `useServices`
   - Non creare istanze multiple dei servizi
2. **Estensibilità**:
   - Usa i registry per registrare nuovi componenti
   - Mantieni le interfacce coerenti quando estendi i servizi
3. **Gestione configurazione**:
   - Utilizza il ConfigManager per tutte le configurazioni
   - Non hardcodare valori che potrebbero cambiare
4. **Performance**:
   - Usa React.memo e useMemo per evitare re-render non necessari
   - Limita l'uso di localStorage alle configurazioni essenziali
5. **Debugging**:
   - I servizi hanno log dettagliati per facilitare il debugging
   - Usa modalità mock durante lo sviluppo (enableLocalData: true)

Deployment

Configurazione per Ambiente di Produzione

1. Crea un file di configurazione personalizzato:

json

```
{
  "business": {
    "name": "Il Tuo Business",
    "type": "cafe",
    "logo": "/your-logo.svg",
    "theme": {
      "primaryColor": "#3b5998",
      "secondaryColor": "#f56565",
      "backgroundColor": "#f7fafc",
      "textColor": "#2d3748"
    }
  },
  "ai": {
    "defaultProvider": "openai",
    "providers": {
      "openai": {
        "displayName": "OpenAI",
        "models": [
          { "id": "gpt-4", "name": "GPT-4" },
          { "id": "gpt-3.5-turbo", "name": "GPT-3.5 Turbo" }
        ],
        "defaultModel": "gpt-3.5-turbo"
      }
    },
    "systemPrompt": "Sei un assistente AI per {business.name}..."
  },
  // Altre configurazioni...
}
```

2. Imposta la variabile d'ambiente `REACT_APP_CONFIG_URL` per puntare al tuo file di configurazione
3. Costruisci l'applicazione:

bash

```
npm run build
```

Troubleshooting

Problemi comuni

1. **L'inizializzazione fallisce**:
   - Verifica che il file di configurazione sia accessibile
   - Controlla che la sintassi JSON sia valida

2. **Provider AI non disponibile**:
   - Verifica che il provider sia registrato correttamente
   - Controlla l'API key nella configurazione
3. **Componenti UI non visualizzati**:
   - Assicurati che il componente sia registrato nel UIComponentRegistry
   - Verifica che il placement sia corretto ('inline', 'sidebar', 'bottom')
4. **Funzioni non disponibili all'AI**:
   - Controlla che la funzione sia registrata nel FunctionRegistry
   - Verifica che sia inclusa nella lista enabledFunctions

Contribuire al Progetto

Workflow di Sviluppo

1. Clona il repository
2. Installa le dipendenze: `npm install`
3. Avvia il server di sviluppo: `npm start`
4. Implementa le tue modifiche
5. Esegui i test: `npm test`
6. Invia una pull request

Standard di Codice

- Segui le linee guida di TypeScript
- Usa il pattern di design singleton per i servizi
- Documenta le funzioni pubbliche con JSDoc
- Mantieni la compatibilità con le interfacce esistenti

Documentazione

Aggiorna questa documentazione quando:

- Aggiungi nuovi servizi o componenti
- Modifichi le interfacce esistenti
- Implementi nuove funzionalità significative