API Integration Guide

🔌 API Client Usage

Installation

```
# Install the API client package
yarn add @ai-platform/api-client
```

Basic Setup

```
import { createApiClient } from '@ai-platform/api-client';

const apiClient = createApiClient({
  baseURL: 'https://api.aiplatform.com',
  timeout: 10000,
});

// Set authentication token
apiClient.setAuthToken('your-jwt-token');
```

Authentication API

Register User

```
import { authApi } from '@ai-platform/api-client';

const result = await authApi.register({
  email: 'user@example.com',
  password: 'securePassword123',
  name: 'John Doe'
});

if (result.success) {
  const { user, tokens } = result.data;
  // Store tokens securely
  localStorage.setItem('accessToken', tokens.accessToken);
}
```

Login User

```
const result = await authApi.login({
  email: 'user@example.com',
  password: 'securePassword123'
});

if (result.success) {
  const { user, tokens } = result.data;
  // Update client with new token
  apiClient.setAuthToken(tokens.accessToken);
}
```

Refresh Token

```
const result = await authApi.refresh({
  refreshToken: localStorage.getItem('refreshToken')
});

if (result.success) {
  const { accessToken, refreshToken } = result.data;
  // Update stored tokens
  localStorage.setItem('accessToken', accessToken);
  apiClient.setAuthToken(accessToken);
}
```

AI Routing API

Get Available AI Agents

```
import { aiApi } from '@ai-platform/api-client';

const agents = await aiApi.getAgents({
  active: true,
  provider: 'openai'
});

console.log('Available agents:', agents.data);
```

Send Chat Message

```
const response = await aiApi.chat({
  message: 'Hello, how can you help me today?',
  agentId: 'gpt-4', // Optional: specify agent
  context: {
    conversationId: 'conv-123',
    userId: 'user-456'
  }
});

if (response.success) {
  const { message, agent, tokensUsed, cost } = response.data;
  console.log('AI Response:', message);
  console.log('Cost:', cost, 'credits');
}
```

User Management API

Get User Profile

```
import { userApi } from '@ai-platform/api-client';

const profile = await userApi.getProfile();
if (profile.success) {
  const { user, creditAccount } = profile.data;
  console.log('Current balance:', creditAccount.balance);
}
```

List Users (Admin only)

```
const users = await userApi.listUsers({
  page: 1,
  limit: 10,
  role: 'EMPLOYEE'
});

if (users.success) {
  const { users: userList, pagination } = users.data;
  console.log('Users:', userList);
  console.log('Total pages:', pagination.totalPages);
}
```

Billing API

Get Credit Balance

```
import { billingApi } from '@ai-platform/api-client';

const credits = await billingApi.getCredits();
if (credits.success) {
  const { balance, totalEarned, totalSpent } = credits.data;
  console.log(`Balance: ${balance} credits`);
}
```

Get Transaction History

```
const transactions = await billingApi.getTransactions({
  page: 1,
  limit: 20,
  type: 'DEBIT'
});

if (transactions.success) {
  const { transactions: txList, pagination } = transactions.data;
  txList.forEach(tx => {
    console.log(`${tx.type}: ${tx.amount} credits - ${tx.description}`);
  });
}
```

🔄 Error Handling

Standard Error Response

All API endpoints return a consistent error format:

```
{
  success: false,
  error: "Error message",
  code: "ERROR_CODE",
  details: {
    // Additional error information
  }
}
```

Error Handling Patterns

```
try {
  const result = await authApi.login(credentials);

  if (!result.success) {
    // Handle API error
    switch (result.code) {
      case 'INVALID_CREDENTIALS':
        showError('Invalid email or password');
        break;
      case 'ACCOUNT_LOCKED':
        showError('Account temporarily locked due to too many failed attempts');
        break;
      default:
        showError(result.error);
    }
    return;
  }

  // Handle success
  const { user, tokens } = result.data;

} catch (error) {
  // Handle network or unexpected errors
  if (error.response?.status === 429) {
    showError('Too many requests. Please try again later.');
  } else if (error.code === 'NETWORK_ERROR') {
    showError('Network error. Please check your connection.');
  } else {
    showError('An unexpected error occurred.');
  }
}
```

🔐 Authentication Integration

Token Management

```
class TokenManager {
  private accessToken: string | null = null;
  private refreshToken: string | null = null;

  setTokens(tokens: { accessToken: string; refreshToken: string }) {
    this.accessToken = tokens.accessToken;
    this.refreshToken = tokens.refreshToken;
    localStorage.setItem('accessToken', tokens.accessToken);
    localStorage.setItem('refreshToken', tokens.refreshToken);
  }

  getAccessToken(): string | null {
    return this.accessToken || localStorage.getItem('accessToken');
  }

  async refreshAccessToken(): Promise<boolean> {
    const refreshToken = this.refreshToken || localStorage.getItem('refreshToken');
    if (!refreshToken) return false;

    try {
      const result = await authApi.refresh({ refreshToken });
      if (result.success) {
        this.setTokens(result.data);
        return true;
      }
    } catch (error) {
      console.error('Token refresh failed:', error);
    }

    return false;
  }

  clearTokens() {
    this.accessToken = null;
    this.refreshToken = null;
    localStorage.removeItem('accessToken');
    localStorage.removeItem('refreshToken');
  }
}
```

Automatic Token Refresh

```
import { apiClient } from '@ai-platform/api-client';

// Setup automatic token refresh
apiClient.interceptors.response.use(
  (response) => response,
  async (error) => {
    if (error.response?.status === 401 && !error.config._retry) {
      error.config._retry = true;

      const refreshed = await tokenManager.refreshAccessToken();
      if (refreshed) {
        // Retry original request with new token
        error.config.headers.Authorization = `Bearer ${tokenManager.getAccessToken()}`;
        return apiClient.request(error.config);
      } else {
        // Refresh failed, redirect to login
        tokenManager.clearTokens();
        window.location.href = '/login';
      }
    }
    return Promise.reject(error);
  }
);
```

📡 Real-time Integration

WebSocket Connection

```
import { notificationApi } from '@ai-platform/api-client';

class NotificationManager {
  private ws: WebSocket | null = null;

  connect(userId: string, token: string) {
    this.ws = new WebSocket(`wss://api.aiplatform.com/notifications?token=${token}&user
Id=${userId}`);

    this.ws.onopen = () => {
      console.log('Connected to notification service');
    };

    this.ws.onmessage = (event) => {
      const notification = JSON.parse(event.data);
      this.handleNotification(notification);
    };

    this.ws.onclose = () => {
      console.log('Disconnected from notification service');
      // Attempt reconnection
      setTimeout(() => this.connect(userId, token), 5000);
    };
  }

  private handleNotification(notification: any) {
    switch (notification.type) {
      case 'CREDIT_LOW':
        showNotification('Credit balance is low', 'warning');
        break;
      case 'AI_RESPONSE_READY':
        updateChatMessage(notification.data);
        break;
      default:
        console.log('Unknown notification:', notification);
    }
  }

  disconnect() {
    if (this.ws) {
      this.ws.close();
      this.ws = null;
    }
  }
}
```

🔧 Advanced Integration Patterns

Pagination Helper

```
class PaginatedApi<T> {
  constructor(
    private apiCall: (params: any) => Promise<{ data: { items: T[]; pagination: any } }
>
  ) {}

  async *fetchAll(baseParams: any = {}): AsyncGenerator<T, void, unknown> {
    let page = 1;
    let hasMore = true;

    while (hasMore) {
      const result = await this.apiCall({ ...baseParams, page, limit: 50 });
      const { items, pagination } = result.data;

      for (const item of items) {
        yield item;
      }

      hasMore = page < pagination.totalPages;
      page++;
    }
  }
}

// Usage
const userApi = new PaginatedApi(userApi.listUsers);
for await (const user of userApi.fetchAll({ role: 'EMPLOYEE' })) {
  console.log(user.name);
}
```

Batch Operations

```
class BatchProcessor<T, R> {
  constructor(
    private processor: (items: T[]) => Promise<R[]>,
    private batchSize: number = 10
  ) {}

  async process(items: T[]): Promise<R[]> {
    const results: R[] = [];

    for (let i = 0; i < items.length; i += this.batchSize) {
      const batch = items.slice(i, i + this.batchSize);
      const batchResults = await this.processor(batch);
      results.push(...batchResults);
    }

    return results;
  }
}

// Usage
const batchProcessor = new BatchProcessor(
  async (users) => Promise.all(users.map(user => userApi.updateUser(user.id, user))),
  5
);

await batchProcessor.process(usersToUpdate);
```

📊 Rate Limiting

Client-side Rate Limiting

```typescript
class RateLimiter {
  private requests: number[] = [];

  constructor(
    private maxRequests: number,
    private windowMs: number
  ) {}

  async canMakeRequest(): Promise<boolean> {
    const now = Date.now();
    this.requests = this.requests.filter(time => now - time < this.windowMs);

    if (this.requests.length >= this.maxRequests) {
      const oldestRequest = Math.min(...this.requests);
      const waitTime = this.windowMs - (now - oldestRequest);
      await new Promise(resolve => setTimeout(resolve, waitTime));
    }

    this.requests.push(now);
    return true;
  }
}

// Usage
const rateLimiter = new RateLimiter(100, 60000); // 100 requests per minute

async function makeApiCall() {
  await rateLimiter.canMakeRequest();
  return await apiClient.get('/some-endpoint');
}
```

🧪 Testing API Integration

Mock API Responses

```
import { jest } from '@jest/globals';
import { authApi } from '@ai-platform/api-client';

// Mock the API module
jest.mock('@ai-platform/api-client');

const mockedAuthApi = authApi as jest.Mocked<typeof authApi>;

describe('Authentication Integration', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });

  test('should handle successful login', async () => {
    const mockResponse = {
      success: true,
      data: {
        user: { id: '1', email: 'test@example.com', name: 'Test User' },
        tokens: { accessToken: 'token', refreshToken: 'refresh' }
      }
    };

    mockedAuthApi.login.mockResolvedValue(mockResponse);

    const result = await handleLogin('test@example.com', 'password');

    expect(mockedAuthApi.login).toHaveBeenCalledWith({
      email: 'test@example.com',
      password: 'password'
    });

    expect(result.success).toBe(true);
  });
});
```

This integration guide provides comprehensive examples for working with the AI Employee Platform APIs, including error handling, authentication, real-time features, and testing patterns.