Infinure - Conversational Data Bot Platform

Complete Technical Specification for Development

📋 EXECUTIVE SUMMARY

Project: Enterprise conversational data analytics platform where business users connect their databases and chat with their data using natural language.

Key Requirements:

- Multi-tenant SaaS platform supporting multiple organizations
- Role-based responses (CEO vs Analyst get different insights from same data)
- Industry-specific configurations (Fintech, Healthcare, E-commerce, etc.)
- Enterprise-grade security and compliance (SOC 2, GDPR, HIPAA)
- Self-hosted Airbyte integration for 350+ data connectors
- Real-time chat interface with contextual AI responses

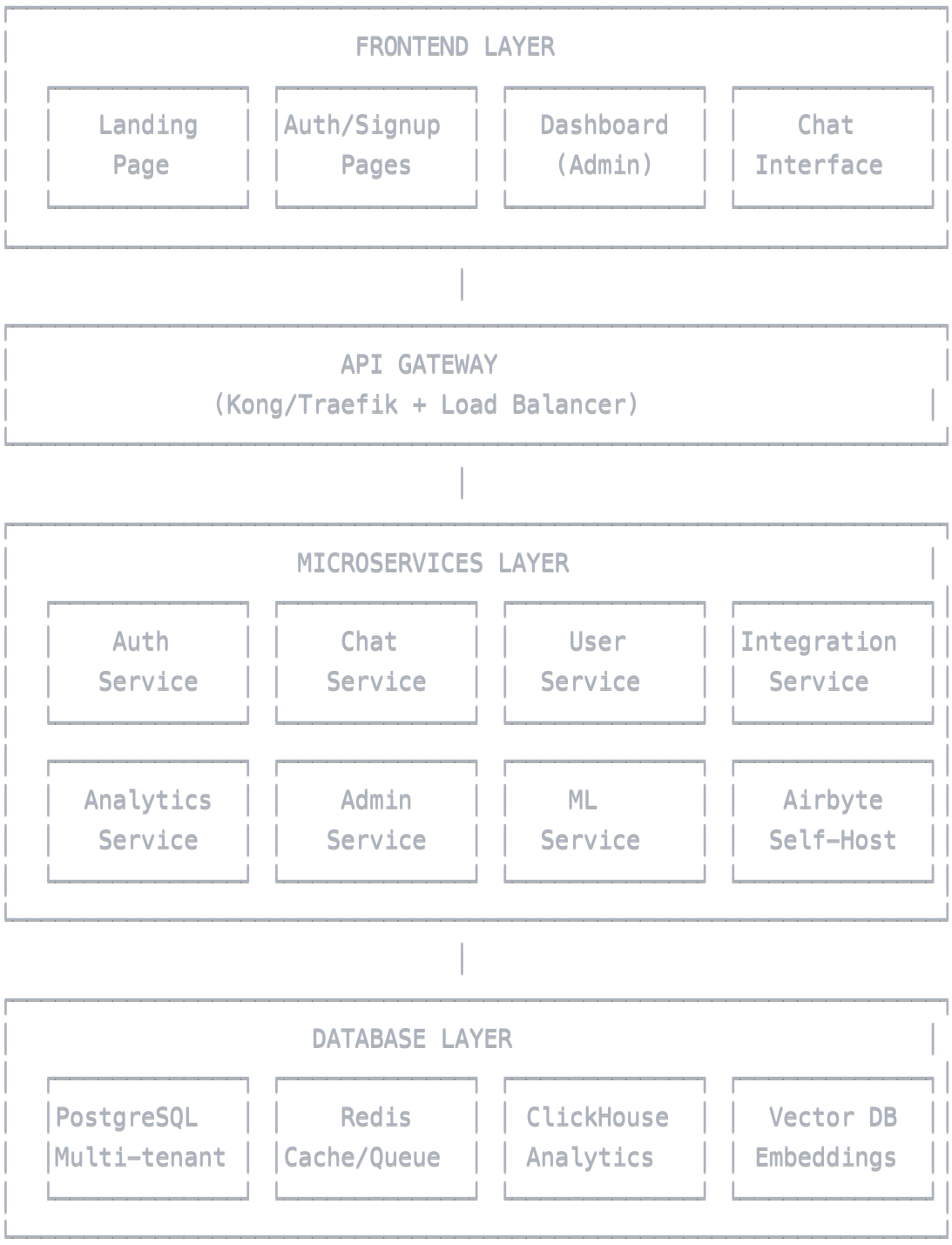
Architecture: Modern microservices with Next.js frontend, NestJS backend, PostgreSQL multi-tenant database, and Kubernetes deployment.

Integration Strategy: Airbyte Open Source self-hosted for data integrations, custom ML service integration for data scientist's AI logic.

---

1. SYSTEM ARCHITECTURE

1.1 High-Level Architecture



1.2 Technology Stack

Frontend Stack

Component	Technology	Version
Framework	Next.js	14+ (App Router)
Language	TypeScript	5.0+
UI Framework	React	18+
Styling	Tailwind CSS + Headless UI	Latest
State Management	Zustand + React Query	Latest
Forms	React Hook Form + Zod	Latest
Charts	Recharts + D3.js	Latest
Real-time	Socket.io-client	Latest
Testing	Vitest + Playwright	Latest

Backend Stack

Component	Technology	Version
Framework	NestJS	10+
Language	TypeScript	5.0+
API	GraphQL + REST hybrid	Latest
Authentication	JWT + Refresh Tokens	-
Real-time	Socket.io	Latest
Jobs	BullMQ + Redis	Latest
Email	SendGrid/Resend	Latest
Testing	Jest + Supertest	Latest

Database & Infrastructure

Component	Technology	Version
Primary DB	PostgreSQL	15+
Cache	Redis Cluster	7+
Analytics	ClickHouse	Latest
Vector DB	Pinecone/Chroma	Latest
Object Storage	AWS S3/MinIO	Latest
Container	Docker + Kubernetes	Latest
Monitoring	Prometheus + Grafana	Latest

---

2. DATABASE DESIGN

2.1 Multi-Tenant PostgreSQL Schema

sql

```sql
-- ORGANIZATIONS (Main tenant isolation)
CREATE TABLE organizations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    slug VARCHAR(100) UNIQUE NOT NULL,
    industry_type VARCHAR(100) NOT NULL,
    subscription_tier VARCHAR(50) NOT NULL DEFAULT 'starter',
    settings JSONB DEFAULT '{}',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- USERS (Role-based access control)
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID REFERENCES organizations(id) ON DELETE CASCADE,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255),
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    role VARCHAR(50) NOT NULL, -- 'ceo', 'director', 'manager', 'analyst', 'viewer'
    department VARCHAR(100),
    permissions JSONB DEFAULT '{}',
    last_login TIMESTAMP WITH TIME ZONE,
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- DATA INTEGRATIONS (Airbyte connections)
CREATE TABLE data_integrations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID REFERENCES organizations(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    type VARCHAR(100) NOT NULL, -- 'postgresql', 'mysql', 'salesforce', etc.
    airbyte_connection_id VARCHAR(255) NOT NULL,
    airbyte_source_id VARCHAR(255) NOT NULL,
    connection_config JSONB NOT NULL, -- encrypted credentials
    schema_mapping JSONB DEFAULT '{}',
    sync_frequency VARCHAR(50) DEFAULT 'hourly',
    is_active BOOLEAN DEFAULT true,
    last_sync TIMESTAMP WITH TIME ZONE,
    created_by UUID REFERENCES users(id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

```sql
-- CONVERSATIONS (Chat history)
CREATE TABLE conversations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID REFERENCES organizations(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    title VARCHAR(255),
    context JSONB DEFAULT '{}', -- user role, department, industry context
    is_archived BOOLEAN DEFAULT false,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- MESSAGES (Individual chat messages)
CREATE TABLE messages (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    conversation_id UUID REFERENCES conversations(id) ON DELETE CASCADE,
    role VARCHAR(20) NOT NULL, -- 'user', 'assistant', 'system'
    content TEXT NOT NULL,
    metadata JSONB DEFAULT '{}', -- query results, charts data, sources
    tokens_used INTEGER DEFAULT 0,
    processing_time_ms INTEGER DEFAULT 0,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- AUDIT LOG (Security and compliance)
CREATE TABLE user_activities (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID REFERENCES organizations(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE SET NULL,
    action VARCHAR(100) NOT NULL,
    resource_type VARCHAR(100),
    resource_id UUID,
    metadata JSONB DEFAULT '{}',
    ip_address INET,
    user_agent TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- INDUSTRY CONFIGURATIONS (Role/industry-specific settings)
CREATE TABLE industry_configs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    industry_type VARCHAR(100) NOT NULL,
    config_name VARCHAR(100) NOT NULL,
    config_value JSONB NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    UNIQUE(industry_type, config_name)
```

```
);

-- INDEXES for performance
CREATE INDEX idx_users_org_id ON users(organization_id);
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_conversations_user_id ON conversations(user_id);
CREATE INDEX idx_messages_conversation_id ON messages(conversation_id);
CREATE INDEX idx_activities_org_id ON user_activities(organization_id);
CREATE INDEX idx_integrations_org_id ON data_integrations(organization_id);
```

2.2 ClickHouse Analytics Schema

sql

```
-- Query analytics and performance tracking
CREATE TABLE query_analytics (
    timestamp DateTime64(3),
    organization_id String,
    user_id String,
    user_role String,
    query_type String,
    query_complexity Float32,
    execution_time_ms UInt32,
    tokens_used UInt32,
    success Bool,
    error_type String,
    data_sources Array(String)
) ENGINE = MergeTree()
ORDER BY (organization_id, timestamp)
PARTITION BY toYYYYMM(timestamp);
```

---

3. SECURITY ARCHITECTURE

3.1 Authentication & Authorization

typescript

```
// Multi-factor Authentication Configuration
interface AuthConfig {
  mfa: {
    required: boolean;
    methods: ['totp', 'sms', 'email'];
  };
  passwordPolicy: {
    minLength: 12;
    requireNumbers: true;
    requireSymbols: true;
    requireUppercase: true;
    requireLowercase: true;
    preventReuse: 12;
  };
  sessionConfig: {
    maxDuration: '8h';
    refreshThreshold: '15m';
    maxConcurrentSessions: 3;
  };
}

// Role-Based Access Control
interface RolePermissions {
  ceo: {
    data_access: 'all';
    sensitive_data: true;
    user_management: true;
    billing: true;
    integrations: true;
  };
  director: {
    data_access: 'department';
    sensitive_data: true;
    user_management: 'department';
    billing: false;
    integrations: 'department';
  };
  manager: {
    data_access: 'team';
    sensitive_data: false;
    user_management: 'team';
    billing: false;
    integrations: false;
  };
  analyst: {
    data_access: 'assigned';
```

```
    sensitive_data: false;
    user_management: false;
    billing: false;
    integrations: false;
  };
}
```

3.2 Data Encryption

typescript

```typescript
// AES-256-GCM encryption for sensitive data
class EncryptionService {
  async encryptSensitiveData(data: any, organizationId: string): Promise<string> {
    const key = await this.getOrganizationKey(organizationId);
    return await encrypt(JSON.stringify(data), key, 'aes-256-gcm');
  }

  async decryptSensitiveData(encryptedData: string, organizationId: string): Promise<a
    const key = await this.getOrganizationKey(organizationId);
    const decrypted = await decrypt(encryptedData, key, 'aes-256-gcm');
    return JSON.parse(decrypted);
  }
}

// Field-level encryption for GDPR compliance
interface EncryptedField {
  value: string; // encrypted
  algorithm: 'aes-256-gcm';
  keyId: string;
  created_at: Date;
}
```

3.3 Compliance Requirements

typescript

```typescript
interface ComplianceFramework {
  // SOC 2 Type II
  accessControls: boolean;
  systemMonitoring: boolean;
  changeManagement: boolean;
  riskAssessment: boolean;

  // GDPR
  dataRetentionPeriod: '7_years';
  rightToBeDeleted: boolean;
  dataPortability: boolean;
  consentManagement: boolean;
  dataProcessingLog: boolean;

  // HIPAA (for healthcare clients)
  accessLogs: boolean;
  dataEncryption: boolean;
  auditTrails: boolean;
  businessAssociateAgreements: boolean;
}
```

---

4. AIRBYTE INTEGRATION SERVICE

4.1 Kubernetes Deployment

yaml

```yaml
# k8s/airbyte/airbyte-deployment.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: airbyte-env
  namespace: infinure
data:
  DATABASE_HOST: "infinure-postgres"
  DATABASE_PORT: "5432"
  DATABASE_DB: "airbyte"
  DATABASE_USER: "airbyte"
  WEBAPP_URL: "https://integrations.infinure.com"
  API_URL: "https://api-integrations.infinure.com"
  TEMPORAL_HOST: "airbyte-temporal:7233"
  TRACKING_STRATEGY: "logging"
  WORKER_ENVIRONMENT: "kubernetes"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: airbyte-server
  namespace: infinure
spec:
  replicas: 2
  selector:
    matchLabels:
      app: airbyte-server
  template:
    metadata:
      labels:
        app: airbyte-server
    spec:
      containers:
      - name: airbyte-server
        image: airbyte/server:0.63.15
        envFrom:
        - configMapRef:
            name: airbyte-env
        - secretRef:
            name: airbyte-secrets
        resources:
          requests:
            memory: "2Gi"
            cpu: "1000m"
          limits:
            memory: "4Gi"
```

```
      cpu: "2000m"
  ports:
  - containerPort: 8001
  livenessProbe:
    httpGet:
      path: /api/v1/health
      port: 8001
    initialDelaySeconds: 60
  readinessProbe:
    httpGet:
      path: /api/v1/health
      port: 8001
    initialDelaySeconds: 10
```

4.2 Airbyte Management Service

typescript

```typescript
// integration-service/src/services/airbyte-manager.ts
import { AirbyteApi } from '@airbyte/api';

@Injectable()
export class AirbyteManager {
  private airbyteApi: AirbyteApi;

  constructor(
    private encryptionService: EncryptionService,
    private auditService: AuditService
  ) {
    this.airbyteApi = new AirbyteApi({
      baseURL: process.env.AIRBYTE_API_URL,
      bearerToken: process.env.AIRBYTE_API_TOKEN
    });
  }

  async createOrganizationWorkspace(organizationId: string): Promise<WorkspaceResult> {
    try {
      const workspace = await this.airbyteApi.workspaces.create({
        name: `org-${organizationId}`,
        displayName: `Organization ${organizationId}`,
        email: `admin@org-${organizationId}.infinure.com`,
        anonymousDataCollection: false,
        news: false,
        securityUpdates: true
      });

      // Save workspace mapping in our database
      await this.saveWorkspaceMapping(organizationId, workspace.workspaceId);

      await this.auditService.log({
        organizationId,
        action: 'WORKSPACE_CREATED',
        resourceId: workspace.workspaceId
      });

      return { workspaceId: workspace.workspaceId, status: 'created' };
    } catch (error) {
      throw new BadRequestException(`Failed to create workspace: ${error.message}`);
    }
  }

  async setupDataSource(
    organizationId: string,
    sourceConfig: DataSourceConfig
```

```
): Promise<ConnectionResult> {
  const workspaceId = await this.getWorkspaceId(organizationId);

  // 1. Encrypt credentials
  const encryptedCredentials = await this.encryptionService.encryptSensitiveData(
    sourceConfig.credentials,
    organizationId
  );

  // 2. Create Source in Airbyte
  const source = await this.airbyteApi.sources.create({
    workspaceId,
    name: sourceConfig.name,
    sourceDefinitionId: await this.getSourceDefinitionId(sourceConfig.type),
    connectionConfiguration: sourceConfig.credentials // Will be handled by Airbyte'
  });

  // 3. Create Destination (organization's data warehouse schema)
  const destination = await this.createOrganizationDestination(organizationId, works

  // 4. Discover source schema
  const catalog = await this.discoverSchema(source.sourceId);

  // 5. Create Connection with proper namespacing
  const connection = await this.airbyteApi.connections.create({
    sourceId: source.sourceId,
    destinationId: destination.destinationId,
    syncCatalog: catalog,
    schedule: {
      scheduleType: 'cron',
      cronExpression: this.getCronExpression(sourceConfig.syncFrequency || 'hourly')
    },
    namespaceDefinition: 'destination',
    namespaceFormat: `org_${organizationId}_${sourceConfig.name.toLowerCase()}`
  });

  // 6. Save to our database
  await this.saveIntegrationRecord({
    organizationId,
    name: sourceConfig.name,
    type: sourceConfig.type,
    airbyteConnectionId: connection.connectionId,
    airbyteSourceId: source.sourceId,
    connectionConfig: encryptedCredentials,
    syncFrequency: sourceConfig.syncFrequency
  });
```

```
    return {
      connectionId: connection.connectionId,
      sourceId: source.sourceId,
      status: 'configured'
    };
  }

  async triggerSync(connectionId: string): Promise<SyncResult> {
    const job = await this.airbyteApi.jobs.create({
      connectionId,
      jobType: 'sync'
    });

    return {
      jobId: job.id,
      status: job.status,
      startedAt: job.createdAt
    };
  }

  private getCronExpression(frequency: string): string {
    const schedules = {
      'hourly': '0 * * * *',
      'daily': '0 2 * * *',
      'weekly': '0 2 * * 0',
      'monthly': '0 2 1 * *'
    };
    return schedules[frequency] || schedules['daily'];
  }
}
```

4.3 Available Connectors (350+)

typescript

```typescript
// integration-service/src/connectors/connector-registry.ts
export const AIRBYTE_CONNECTORS = {
  // Databases
  databases: {
    'postgres': 'decd338e-5647-4c0b-adf4-da0e75f5a750',
    'mysql': 'b4389032-b9bb-4c1e-9a2e-3ad4c87b3e0c',
    'mongodb': 'b5ea17b1-f170-46dc-bc31-cc744ca984c1',
    'oracle': '8be1cf83-fde1-477f-a4ad-318d23c9f3c6',
    'sqlserver': 'b5ea17b1-f170-46dc-bc31-cc744ca984c1',
    'redis': 'b687c099-6f50-4d89-ab7f-e9b2e4c9c7b1',
    'elasticsearch': 'c1b6c8c8-c8c8-4c8c-8c8c-c8c8c8c8c8c8'
  },

  // Data Warehouses
  dataWarehouses: {
    'snowflake': 'b1c2c3d4-e5f6-7890-abcd-ef1234567890',
    'bigquery': 'c2d3e4f5-g6h7-8901-bcde-f23456789012',
    'redshift': 'd3e4f5g6-h7i8-9012-cdef-345678901234',
    'databricks': 'e4f5g6h7-i8j9-0123-def0-456789012345'
  },

  // SaaS Business Tools
  saas: {
    'salesforce': 'b7e4c55d-7867-4dcb-a856-c021a5c2a5a8',
    'hubspot': 'f7864902-5566-4a22-be62-b64bd4b8252c',
    'stripe': 'e094cb9a-26de-4645-8761-65c0c425d1de',
    'shopify': 'c08cfb82-2c8b-4c14-9e77-fa1faf2c21c7',
    'google-analytics': 'aea4e1d7-3e29-4c8b-9c1b-7e3c3e3c3e3c',
    'facebook-marketing': 'e7778cfc-e97c-4458-9ecb-b4f2bbd8431c',
    'linkedin-ads': 'e02df5cd-3c3d-4b85-a8a6-e2e1e1e1e1e1',
    'mailchimp': 'c8c8c8c8-c8c8-4c8c-8c8c-c8c8c8c8c8c8',
    'intercom': 'd9d9d9d9-d9d9-4d9d-9d9d-d9d9d9d9d9d9',
    'zendesk': 'e0e0e0e0-e0e0-4e0e-e0e0-e0e0e0e0e0e0'
  },

  // File Sources
  files: {
    's3': 'b3bfc7b5-6b6b-4b6b-8b6b-b6b6b6b6b6b6',
    'google-sheets': 'a4a4a4a4-a4a4-4a4a-a4a4-a4a4a4a4a4a4',
    'csv': 'e5e5e5e5-e5e5-4e5e-e5e5-e5e5e5e5e5e5',
    'json': 'f6f6f6f6-f6f6-4f6f-f6f6-f6f6f6f6f6f6',
    'ftp': 'g7g7g7g7-g7g7-4g7g-g7g7-g7g7g7g7g7g7'
  }
};

export class ConnectorRegistry {
```

```
  getConnectorsByIndustry(industry: string): string[] {
    const industryMappings = {
      'fintech': ['postgres', 'stripe', 'salesforce', 'plaid', 'quickbooks'],
      'healthcare': ['postgres', 'salesforce', 'epic', 'cerner', 'mongodb'],
      'ecommerce': ['shopify', 'google-analytics', 'facebook-marketing', 'stripe', 'ma
      'saas': ['hubspot', 'intercom', 'mixpanel', 'amplitude', 'salesforce']
    };

    return industryMappings[industry] || Object.keys(AIRBYTE_CONNECTORS.databases);
  }
}
```

---

5. MICROSERVICES ARCHITECTURE

5.1 Auth Service

typescript

```typescript
// auth-service/src/auth.controller.ts
@Controller('auth')
@ApiTags('Authentication')
export class AuthController {
  constructor(
    private authService: AuthService,
    private mfaService: MFAService
  ) {}

  @Post('login')
  @ApiOperation({ summary: 'User login with MFA support' })
  async login(@Body() loginDto: LoginDto): Promise<AuthResponse> {
    // 1. Validate credentials
    const user = await this.authService.validateUser(loginDto.email, loginDto.password

    // 2. Check if MFA is required
    if (user.organization.mfaRequired) {
      const mfaToken = await this.mfaService.generateMFAChallenge(user.id);
      return {
        requiresMFA: true,
        mfaToken,
        supportedMethods: user.mfaMethods
      };
    }

    // 3. Generate tokens
    const tokens = await this.authService.generateTokens(user);

    // 4. Log successful login
    await this.auditService.log({
      organizationId: user.organizationId,
      userId: user.id,
      action: 'LOGIN_SUCCESS'
    });

    return {
      accessToken: tokens.accessToken,
      refreshToken: tokens.refreshToken,
      user: this.sanitizeUser(user)
    };
  }

  @Post('sso/:provider')
  @ApiOperation({ summary: 'SSO login (SAML, OIDC)' })
  async ssoLogin(@Param('provider') provider: string, @Body() ssoDto: SSODto): Promise
    // Support for Azure AD, Okta, Google Workspace
```

```
    return await this.authService.processSSOLogin(provider, ssoDto);
  }

  @Post('mfa/verify')
  @ApiOperation({ summary: 'Verify MFA token' })
  async verifyMFA(@Body() mfaDto: MFAVerificationDto): Promise<AuthResponse> {
    const verification = await this.mfaService.verifyMFA(mfaDto.mfaToken, mfaDto.code)

    if (!verification.valid) {
      throw new UnauthorizedException('Invalid MFA code');
    }

    const tokens = await this.authService.generateTokens(verification.user);
    return {
      accessToken: tokens.accessToken,
      refreshToken: tokens.refreshToken,
      user: this.sanitizeUser(verification.user)
    };
  }
}
```

5.2 Chat Service

typescript

```typescript
// chat-service/src/chat.service.ts
@Injectable()
export class ChatService {
  constructor(
    private mlService: MLService, // Data scientist's service
    private contextService: ContextService,
    private analyticsService: AnalyticsService
  ) {}

  async processQuery(
    query: string,
    userId: string,
    organizationId: string,
    conversationId?: string
  ): Promise<ChatResponse> {
    const startTime = Date.now();

    try {
      // 1. Get user context (role, department, permissions)
      const userContext = await this.contextService.getUserContext(userId);

      // 2. Get organization context (industry, data sources)
      const orgContext = await this.contextService.getOrganizationContext(organization

      // 3. Build complete context for ML service
      const chatContext: ChatContext = {
        user: {
          role: userContext.role,
          department: userContext.department,
          permissions: userContext.permissions
        },
        organization: {
          industryType: orgContext.industryType,
          availableDataSources: orgContext.dataSources,
          complianceRequirements: orgContext.compliance
        },
        conversation: {
          history: await this.getConversationHistory(conversationId, 10),
          context: {}
        }
      };

      // 4. Call ML service (data scientist's implementation)
      const mlResponse = await this.mlService.processQuery({
        query,
        context: chatContext,
```

```
      organizationId
    });

    // 5. Post-process response based on user role
    const customizedResponse = await this.customizeResponseForRole(mlResponse, userC

    // 6. Save message to database
    const conversation = await this.ensureConversation(conversationId, userId, organ
    await this.saveMessages(conversation.id, [
      { role: 'user', content: query },
      { role: 'assistant', content: customizedResponse.content, metadata: customizedR
    ]);

    // 7. Track analytics
    await this.analyticsService.trackQuery({
      organizationId,
      userId,
      userRole: userContext.role,
      queryType: mlResponse.queryType,
      executionTime: Date.now() - startTime,
      tokensUsed: mlResponse.tokensUsed,
      success: true
    });

    return {
      content: customizedResponse.content,
      metadata: customizedResponse.metadata,
      conversationId: conversation.id,
      sources: mlResponse.sources
    };

  } catch (error) {
    await this.analyticsService.trackQuery({
      organizationId,
      userId,
      userRole: userContext?.role || 'unknown',
      executionTime: Date.now() - startTime,
      success: false,
      error: error.message
    });

    throw error;
  }
}

private async customizeResponseForRole(
  mlResponse: MLResponse,
```

```
    userRole: UserRole
  ): Promise<CustomizedResponse> {
    switch (userRole) {
      case 'ceo':
        return this.createExecutiveSummary(mlResponse);
      case 'director':
        return this.createDepartmentalView(mlResponse);
      case 'manager':
        return this.createOperationalView(mlResponse);
      case 'analyst':
        return this.createDetailedAnalysis(mlResponse);
      default:
        return this.createBasicView(mlResponse);
    }
  }

  private createExecutiveSummary(response: MLResponse): CustomizedResponse {
    return {
      content: response.content,
      metadata: {
        ...response.metadata,
        summary: this.generateExecutiveSummary(response.data),
        kpis: this.extractKPIs(response.data),
        recommendations: this.generateStrategicRecommendations(response.data),
        charts: this.createExecutiveCharts(response.data)
      }
    };
  }
}

// Context interfaces
interface ChatContext {
  user: {
    role: 'ceo' | 'director' | 'manager' | 'analyst' | 'viewer';
    department: string;
    permissions: string[];
  };
  organization: {
    industryType: string;
    availableDataSources: DataSource[];
    complianceRequirements: string[];
  };
  conversation: {
    history: Message[];
    context: Record<string, any>;
```

```
    };
  }
```

5.3 Integration API Controller

typescript

```typescript
// integration-service/src/controllers/integration.controller.ts
@Controller('integrations')
@ApiTags('Data Integrations')
@UseGuards(JwtAuthGuard, RoleGuard)
export class IntegrationController {

  @Post('sources')
  @Roles('admin', 'manager')
  @ApiOperation({ summary: 'Create new data source connection' })
  async createDataSource(
    @Body() createSourceDto: CreateSourceDto,
    @GetOrganization() organizationId: string,
    @GetUser() user: User
  ): Promise<SourceResult> {
    // Validate organization access
    await this.validateOrganizationAccess(organizationId, user);

    // Create data source in Airbyte
    const result = await this.airbyteManager.setupDataSource(
      organizationId,
      createSourceDto
    );

    // Register in audit log
    await this.auditService.log({
      organizationId,
      userId: user.id,
      action: 'DATA_SOURCE_CREATED',
      resourceId: result.sourceId,
      metadata: {
        sourceType: createSourceDto.type,
        sourceName: createSourceDto.name
      }
    });

    return result;
  }

  @Get('sources')
  @ApiOperation({ summary: 'Get organization data sources' })
  async getDataSources(
    @GetOrganization() organizationId: string
  ): Promise<DataSource[]> {
    return await this.airbyteManager.getOrganizationSources(organizationId);
  }
```

```typescript
  @Post('sources/:sourceId/sync')
  @Roles('admin', 'manager', 'analyst')
  @ApiOperation({ summary: 'Trigger manual data sync' })
  async triggerSync(
    @Param('sourceId') sourceId: string,
    @GetOrganization() organizationId: string,
    @GetUser() user: User
  ): Promise<SyncResult> {
    // Validate source ownership
    await this.validateSourceOwnership(sourceId, organizationId);

    const connectionId = await this.getConnectionBySourceId(sourceId);
    const result = await this.airbyteManager.triggerSync(connectionId);

    await this.auditService.log({
      organizationId,
      userId: user.id,
      action: 'SYNC_TRIGGERED',
      resourceId: sourceId
    });

    return result;
  }

  @Get('sources/:sourceId/status')
  @ApiOperation({ summary: 'Get data source sync status' })
  async getSourceStatus(
    @Param('sourceId') sourceId: string,
    @GetOrganization() organizationId: string
  ): Promise<SourceStatus> {
    await this.validateSourceOwnership(sourceId, organizationId);

    return await this.airbyteManager.getSourceStatus(sourceId);
  }

  @Get('connectors')
  @ApiOperation({ summary: 'Get available connectors by industry' })
  async getAvailableConnectors(
    @GetOrganization() organizationId: string
  ): Promise<ConnectorInfo[]> {
    const organization = await this.organizationService.findById(organizationId);
    return this.connectorRegistry.getConnectorsByIndustry(organization.industryType);
  }
}
```

6. FRONTEND ARCHITECTURE

6.1 Project Structure

```
src/
├── app/                          # Next.js App Router
│   ├── (auth)/                   # Auth routes
│   │   ├── login/
│   │   │   └── page.tsx
│   │   ├── signup/
│   │   │   └── page.tsx
│   │   └── sso/
│   │       └── [provider]/
│   │           └── page.tsx
│   ├── (dashboard)/              # Main app routes
│   │   ├── chat/
│   │   │   ├── page.tsx          # Chat interface
│   │   │   └── [conversationId]/
│   │   │       └── page.tsx
│   │   ├── integrations/
│   │   │   ├── page.tsx          # Integration management
│   │   │   ├── new/
│   │   │   │   └── page.tsx
│   │   │   └── [integrationId]/
│   │   │       ├── page.tsx
│   │   │       └── settings/
│   │   │           └── page.tsx
│   │   ├── analytics/
│   │   │   └── page.tsx          # Usage analytics
│   │   ├── settings/
│   │   │   ├── page.tsx          # Organization settings
│   │   │   ├── users/
│   │   │   │   └── page.tsx
│   │   │   └── billing/
│   │   │       └── page.tsx
│   │   └── layout.tsx            # Dashboard layout
│   ├── (marketing)/              # Public routes
│   │   ├── page.tsx              # Landing page
│   │   ├── pricing/
│   │   │   └── page.tsx
│   │   ├── about/
│   │   │   └── page.tsx
│   │   └── layout.tsx            # Marketing layout
│   ├── api/                      # API routes
│   │   ├── auth/
│   │   ├── webhooks/
│   │   └── health/
│   ├── globals.css
│   └── layout.tsx                # Root layout
├── components/
```

```
│   ├── ui/                    # Design system components
│   │   ├── button.tsx
│   │   ├── input.tsx
│   │   ├── modal.tsx
│   │   ├── chart.tsx
│   │   └── data-table.tsx
│   ├── chat/                  # Chat-specific components
│   │   ├── chat-interface.tsx
│   │   ├── message-list.tsx
│   │   ├── message-item.tsx
│   │   ├── chat-input.tsx
│   │   └── typing-indicator.tsx
│   ├── integrations/          # Integration components
│   │   ├── connection-card.tsx
│   │   ├── connector-grid.tsx
│   │   ├── sync-status.tsx
│   │   └── credential-form.tsx
│   ├── analytics/             # Analytics components
│   │   ├── metrics-dashboard.tsx
│   │   ├── usage-chart.tsx
│   │   └── performance-stats.tsx
│   └── layout/                # Layout components
│       ├── sidebar.tsx
│       ├── header.tsx
│       └── breadcrumbs.tsx
├── lib/
│   ├── auth.ts                # Auth utilities
│   ├── api.ts                 # API client
│   ├── websocket.ts           # WebSocket client
│   ├── utils.ts               # General utilities
│   └── validators.ts          # Zod schemas
├── hooks/
│   ├── use-auth.ts
│   ├── use-chat.ts
│   ├── use-integrations.ts
│   └── use-analytics.ts
├── stores/
│   ├── auth-store.ts          # Zustand auth store
│   ├── chat-store.ts          # Chat state
│   └── ui-store.ts            # UI state
└── types/
    ├── auth.ts
    ├── chat.ts
    ├── integrations.ts
    └── analytics.ts
```

6.2 Key Frontend Components

typescript

```tsx
// components/chat/ChatInterface.tsx
'use client';

import { useState, useEffect, useRef } from 'react';
import { useChat } from '@/hooks/use-chat';
import { useAuth } from '@/hooks/use-auth';
import { MessageList } from './MessageList';
import { ChatInput } from './ChatInput';
import { TypingIndicator } from './TypingIndicator';

interface ChatInterfaceProps {
  conversationId?: string;
}

export function ChatInterface({ conversationId }: ChatInterfaceProps) {
  const { user } = useAuth();
  const {
    messages,
    isLoading,
    isTyping,
    sendMessage,
    loadConversation
  } = useChat(conversationId);

  const messagesEndRef = useRef<HTMLDivElement>(null);

  useEffect(() => {
    if (conversationId) {
      loadConversation(conversationId);
    }
  }, [conversationId, loadConversation]);

  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  }, [messages]);

  const handleSendMessage = async (content: string) => {
    if (!content.trim()) return;

    await sendMessage({
      content,
      context: {
        userRole: user?.role,
        department: user?.department,
        organizationId: user?.organizationId
      }
```

```
  });
};

const getPlaceholderByRole = (role: string) => {
  const placeholders = {
    ceo: "Ask about overall business performance, strategic metrics, or company-wide
    director: "Ask about departmental performance, team metrics, or operational insi
    manager: "Ask about team performance, project metrics, or resource allocation..."
    analyst: "Ask about specific data points, detailed analysis, or technical metric
  };
  return placeholders[role] || "Ask me anything about your data...";
};

return (
  <div className="flex flex-col h-full max-h-screen">
    {/* Chat Header */}
    <div className="flex-shrink-0 border-b border-gray-200 p-4">
      <div className="flex items-center justify-between">
        <div>
          <h1 className="text-lg font-semibold text-gray-900">
            Data Chat
          </h1>
          <p className="text-sm text-gray-500">
            Role: {user?.role} • Department: {user?.department}
          </p>
        </div>
        <div className="flex items-center space-x-2">
          {/* Status indicators, controls, etc. */}
        </div>
      </div>
    </div>

    {/* Messages */}
    <div className="flex-1 overflow-y-auto p-4">
      <MessageList
        messages={messages}
        userRole={user?.role}
        isLoading={isLoading}
      />
      {isTyping && <TypingIndicator />}
      <div ref={messagesEndRef} />
    </div>

    {/* Chat Input */}
    <div className="flex-shrink-0 border-t border-gray-200 p-4">
      <ChatInput
        onSendMessage={handleSendMessage}
```

```
          placeholder={getPlaceholderByRole(user?.role)}
          disabled={isLoading}
        />
      </div>
    </div>
  );
}
```

typescript

```tsx
// components/integrations/ConnectorGrid.tsx
'use client';

import { useState } from 'react';
import { useIntegrations } from '@/hooks/use-integrations';
import { ConnectorCard } from './ConnectorCard';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Badge } from '@/components/ui/badge';

interface ConnectorGridProps {
  industryType: string;
}

export function ConnectorGrid({ industryType }: ConnectorGridProps) {
  const [searchQuery, setSearchQuery] = useState('');
  const [selectedCategory, setSelectedCategory] = useState<string>('all');

  const {
    availableConnectors,
    isLoading,
    createConnection
  } = useIntegrations();

  const categories = [
    { id: 'all', name: 'All Connectors', count: availableConnectors.length },
    { id: 'databases', name: 'Databases', count: availableConnectors.filter(c => c.cat
    { id: 'saas', name: 'SaaS Tools', count: availableConnectors.filter(c => c.categor
    { id: 'files', name: 'File Sources', count: availableConnectors.filter(c => c.cate
  ];

  const filteredConnectors = availableConnectors.filter(connector => {
    const matchesSearch = connector.name.toLowerCase().includes(searchQuery.toLowerCas
                          connector.description.toLowerCase().includes(searchQuery.toLo
    const matchesCategory = selectedCategory === 'all' || connector.category === selec
    return matchesSearch && matchesCategory;
  });

  const recommendedConnectors = availableConnectors.filter(connector =>
    connector.recommendedFor?.includes(industryType)
  );

  return (
    <div className="space-y-6">
      {/* Search and Filters */}
      <div className="flex flex-col sm:flex-row gap-4">
```

```
  <div className="flex-1">
    <Input
      placeholder="Search connectors..."
      value={searchQuery}
      onChange={(e) => setSearchQuery(e.target.value)}
      className="w-full"
    />
  </div>
  <div className="flex gap-2">
    {categories.map(category => (
      <Button
        key={category.id}
        variant={selectedCategory === category.id ? 'default' : 'outline'}
        size="sm"
        onClick={() => setSelectedCategory(category.id)}
      >
        {category.name}
        <Badge variant="secondary" className="ml-2">
          {category.count}
        </Badge>
      </Button>
    ))}
  </div>
</div>

{/* Recommended Connectors */}
{recommendedConnectors.length > 0 && (
  <div>
    <h3 className="text-lg font-medium text-gray-900 mb-4">
      Recommended for {industryType}
    </h3>
    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
      {recommendedConnectors.map(connector => (
        <ConnectorCard
          key={connector.id}
          connector={connector}
          onConnect={() => createConnection(connector)}
          isRecommended
        />
      ))}
    </div>
  </div>
)}

{/* All Connectors */}
<div>
  <h3 className="text-lg font-medium text-gray-900 mb-4">
```

```
            All Available Connectors
          </h3>
          {isLoading ? (
            <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
              {[...Array(9)].map((_, i) => (
                <div key={i} className="h-40 bg-gray-200 animate-pulse rounded-lg" />
              ))}
            </div>
          ) : (
            <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
              {filteredConnectors.map(connector => (
                <ConnectorCard
                  key={connector.id}
                  connector={connector}
                  onConnect={() => createConnection(connector)}
                />
              ))}
            </div>
          )}
        </div>
      </div>
    );
}
```

6.3 Real-time WebSocket Integration

typescript

```typescript
// lib/websocket.ts
import { io, Socket } from 'socket.io-client';

class WebSocketManager {
  private socket: Socket | null = null;
  private organizationId: string | null = null;
  private userId: string | null = null;

  connect(organizationId: string, userId: string, token: string) {
    if (this.socket?.connected) {
      this.disconnect();
    }

    this.organizationId = organizationId;
    this.userId = userId;

    this.socket = io(process.env.NEXT_PUBLIC_WS_URL!, {
      auth: { token },
      query: { organizationId, userId },
      transports: ['websocket']
    });

    this.setupEventListeners();
  }

  private setupEventListeners() {
    if (!this.socket) return;

    this.socket.on('connect', () => {
      console.log('WebSocket connected');
    });

    this.socket.on('disconnect', () => {
      console.log('WebSocket disconnected');
    });

    this.socket.on('message', (message: ChatMessage) => {
      // Handle incoming chat messages
      this.handleIncomingMessage(message);
    });

    this.socket.on('typing', (data: TypingData) => {
      // Handle typing indicators
      this.handleTypingIndicator(data);
    });
```

```
    this.socket.on('integration_status', (data: IntegrationStatus) => {
      // Handle integration sync status updates
      this.handleIntegrationStatus(data);
    });

    this.socket.on('user_activity', (activity: UserActivity) => {
      // Handle real-time user activity updates
      this.handleUserActivity(activity);
    });
  }

  sendMessage(conversationId: string, content: string) {
    if (!this.socket?.connected) return;

    this.socket.emit('send_message', {
      conversationId,
      content,
      organizationId: this.organizationId,
      userId: this.userId
    });
  }

  joinConversation(conversationId: string) {
    if (!this.socket?.connected) return;
    this.socket.emit('join_conversation', { conversationId });
  }

  leaveConversation(conversationId: string) {
    if (!this.socket?.connected) return;
    this.socket.emit('leave_conversation', { conversationId });
  }

  setTyping(conversationId: string, isTyping: boolean) {
    if (!this.socket?.connected) return;

    this.socket.emit('typing', {
      conversationId,
      isTyping,
      userId: this.userId
    });
  }

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

```
  }

  private handleIncomingMessage(message: ChatMessage) {
    // Update chat store with new message
    window.dispatchEvent(new CustomEvent('websocket:message', {
      detail: message
    }));
  }

  private handleTypingIndicator(data: TypingData) {
    window.dispatchEvent(new CustomEvent('websocket:typing', {
      detail: data
    }));
  }

  private handleIntegrationStatus(status: IntegrationStatus) {
    window.dispatchEvent(new CustomEvent('websocket:integration_status', {
      detail: status
    }));
  }

  private handleUserActivity(activity: UserActivity) {
    window.dispatchEvent(new CustomEvent('websocket:user_activity', {
      detail: activity
    }));
  }
}

export const wsManager = new WebSocketManager();
```

---

7. INDUSTRY-SPECIFIC CONFIGURATIONS

7.1 Industry Templates

typescript

```
// config/industries.ts
export const INDUSTRY_CONFIGS = {
  fintech: {
    name: 'Financial Technology',
    sensitiveFields: [
      'account_balance', 'transaction_amount', 'credit_score',
      'ssn', 'bank_account', 'routing_number'
    ],
    regulatoryCompliance: ['PCI-DSS', 'SOX', 'GDPR', 'CCPA'],
    defaultQuestions: [
      'Show me this month\'s transaction volume by payment method',
      'What\'s our customer acquisition cost by channel?',
      'Analyze fraud patterns in recent transactions',
      'Compare revenue growth vs industry benchmarks',
      'Show top performing products by profit margin'
    ],
    chartTypes: ['financial_timeline', 'risk_matrix', 'compliance_dashboard', 'convers
    kpis: [
      'Monthly Recurring Revenue', 'Customer Acquisition Cost', 'Lifetime Value',
      'Churn Rate', 'Transaction Volume', 'Fraud Rate'
    ],
    dataSources: ['stripe', 'plaid', 'salesforce', 'postgres', 'quickbooks']
  },

  healthcare: {
    name: 'Healthcare',
    sensitiveFields: [
      'patient_id', 'medical_record_number', 'diagnosis', 'treatment',
      'ssn', 'date_of_birth', 'address', 'phone_number'
    ],
    regulatoryCompliance: ['HIPAA', 'GDPR', 'FDA', 'HITECH'],
    defaultQuestions: [
      'Patient satisfaction trends this quarter',
      'Resource utilization by department',
      'Clinical outcomes analysis for diabetes patients',
      'Average length of stay by condition',
      'Staff productivity metrics'
    ],
    chartTypes: ['patient_flow', 'clinical_metrics', 'resource_allocation', 'outcome_t
    kpis: [
      'Patient Satisfaction Score', 'Average Length of Stay', 'Readmission Rate',
      'Staff Utilization', 'Clinical Quality Indicators', 'Cost per Patient'
    ],
    dataSources: ['epic', 'cerner', 'salesforce', 'postgres', 'mongodb']
  },
```

```
  ecommerce: {
    name: 'E-commerce',
    sensitiveFields: [
      'customer_email', 'payment_info', 'personal_address',
      'credit_card', 'phone_number', 'purchase_history'
    ],
    regulatoryCompliance: ['PCI-DSS', 'GDPR', 'CCPA', 'COPPA'],
    defaultQuestions: [
      'Top selling products this month',
      'Customer lifetime value analysis',
      'Inventory turnover by category',
      'Conversion rate by traffic source',
      'Average order value trends'
    ],
    chartTypes: ['sales_funnel', 'product_performance', 'customer_segments', 'inventor
    kpis: [
      'Conversion Rate', 'Average Order Value', 'Customer Lifetime Value',
      'Cart Abandonment Rate', 'Inventory Turnover', 'Return Rate'
    ],
    dataSources: ['shopify', 'google-analytics', 'facebook-marketing', 'stripe', 'mail
  },

  saas: {
    name: 'Software as a Service',
    sensitiveFields: [
      'user_email', 'payment_info', 'api_keys', 'usage_data', 'personal_data'
    ],
    regulatoryCompliance: ['GDPR', 'CCPA', 'SOC2', 'ISO27001'],
    defaultQuestions: [
      'Monthly recurring revenue trends',
      'User engagement metrics by feature',
      'Churn analysis by customer segment',
      'Product adoption rates',
      'Support ticket trends'
    ],
    chartTypes: ['mrr_trends', 'user_engagement', 'churn_analysis', 'feature_adoption'
    kpis: [
      'Monthly Recurring Revenue', 'Annual Recurring Revenue', 'Churn Rate',
      'Net Revenue Retention', 'Customer Acquisition Cost', 'Product Qualified Leads'
    ],
    dataSources: ['hubspot', 'intercom', 'mixpanel', 'amplitude', 'salesforce']
  }
};

export class IndustryConfigService {
  getIndustryConfig(industryType: string): IndustryConfig {
    return INDUSTRY_CONFIGS[industryType] || INDUSTRY_CONFIGS['saas'];
```

```
  }

  getRecommendedConnectors(industryType: string): string[] {
    const config = this.getIndustryConfig(industryType);
    return config.dataSources;
  }

  getSensitiveFields(industryType: string): string[] {
    const config = this.getIndustryConfig(industryType);
    return config.sensitiveFields;
  }

  getComplianceRequirements(industryType: string): string[] {
    const config = this.getIndustryConfig(industryType);
    return config.regulatoryCompliance;
  }
}
```

7.2 Role-Based Response Customization

typescript

```typescript
// services/response-customizer.ts
export class ResponseCustomizer {
  customizeResponse(
    response: MLResponse,
    userContext: UserContext,
    industryType: string
  ): CustomizedResponse {
    const { role, department } = userContext;

    switch (role) {
      case 'ceo':
        return this.createExecutiveSummary(response, industryType);
      case 'director':
        return this.createDepartmentalView(response, department, industryType);
      case 'manager':
        return this.createOperationalView(response, department);
      case 'analyst':
        return this.createDetailedAnalysis(response);
      default:
        return this.createBasicView(response);
    }
  }

  private createExecutiveSummary(response: MLResponse, industry: string): CustomizedRe
    const industryConfig = INDUSTRY_CONFIGS[industry];

    return {
      ...response,
      format: 'executive',
      sections: {
        summary: this.generateExecutiveSummary(response.data, industry),
        kpis: this.extractIndustryKPIs(response.data, industryConfig.kpis),
        recommendations: this.generateStrategicRecommendations(response.data, industry
        risks: this.identifyBusinessRisks(response.data, industry),
        charts: this.createExecutiveCharts(response.data, industryConfig.chartTypes)
      },
      tone: 'strategic',
      detailLevel: 'high-level'
    };
  }

  private createDepartmentalView(
    response: MLResponse,
    department: string,
    industry: string
  ): CustomizedResponse {
```

```
    return {
      ...response,
      format: 'departmental',
      sections: {
        summary: this.generateDepartmentalSummary(response.data, department),
        metrics: this.extractDepartmentalMetrics(response.data, department),
        comparisons: this.createDepartmentalComparisons(response.data, department),
        actionItems: this.generateDepartmentalActions(response.data, department),
        charts: this.createDepartmentalCharts(response.data, department)
      },
      tone: 'operational',
      detailLevel: 'detailed'
    };
  }

  private createDetailedAnalysis(response: MLResponse): CustomizedResponse {
    return {
      ...response,
      format: 'analytical',
      sections: {
        methodology: this.explainAnalysisMethodology(response.data),
        findings: this.presentDetailedFindings(response.data),
        dataQuality: this.assessDataQuality(response.data),
        limitations: this.identifyAnalysisLimitations(response.data),
        recommendations: this.generateTechnicalRecommendations(response.data),
        rawData: this.includeRawDataSample(response.data)
      },
      tone: 'technical',
      detailLevel: 'comprehensive'
    };
  }
}
```

---

8. MONITORING & OBSERVABILITY

8.1 Application Metrics

typescript

```typescript
// monitoring/metrics.service.ts
import { Injectable } from '@nestjs/common';
import { register, Counter, Histogram, Gauge } from 'prom-client';

@Injectable()
export class MetricsService {
  private readonly httpRequestsTotal = new Counter({
    name: 'infinure_http_requests_total',
    help: 'Total number of HTTP requests',
    labelNames: ['method', 'route', 'status_code', 'organization_id']
  });

  private readonly httpRequestDuration = new Histogram({
    name: 'infinure_http_request_duration_seconds',
    help: 'Duration of HTTP requests in seconds',
    labelNames: ['method', 'route', 'organization_id'],
    buckets: [0.1, 0.5, 1, 2, 5, 10]
  });

  private readonly chatQueryDuration = new Histogram({
    name: 'infinure_chat_query_duration_seconds',
    help: 'Duration of chat queries in seconds',
    labelNames: ['organization_id', 'user_role', 'query_type'],
    buckets: [1, 2, 5, 10, 20, 30, 60]
  });

  private readonly activeConnections = new Gauge({
    name: 'infinure_websocket_connections_active',
    help: 'Number of active WebSocket connections',
    labelNames: ['organization_id']
  });

  private readonly airbyteActiveConnections = new Gauge({
    name: 'infinure_airbyte_connections_active',
    help: 'Number of active Airbyte connections',
    labelNames: ['organization_id']
  });

  private readonly airbyteSyncFailures = new Counter({
    name: 'infinure_airbyte_sync_failures_total',
    help: 'Total number of Airbyte sync failures',
    labelNames: ['organization_id', 'source_type', 'error_type']
  });

  recordHttpRequest(method: string, route: string, statusCode: number, organizationId:
    this.httpRequestsTotal.inc({
```

```
      method,
      route,
      status_code: statusCode.toString(),
      organization_id: organizationId
    });
  }

  recordHttpDuration(method: string, route: string, organizationId: string, duration:
    this.httpRequestDuration.observe({
      method,
      route,
      organization_id: organizationId
    }, duration);
  }

  recordChatQuery(organizationId: string, userRole: string, queryType: string, duratio
    this.chatQueryDuration.observe({
      organization_id: organizationId,
      user_role: userRole,
      query_type: queryType
    }, duration);
  }

  updateActiveConnections(organizationId: string, count: number) {
    this.activeConnections.set({ organization_id: organizationId }, count);
  }

  recordAirbyteSyncFailure(organizationId: string, sourceType: string, errorType: stri
    this.airbyteSyncFailures.inc({
      organization_id: organizationId,
      source_type: sourceType,
      error_type: errorType
    });
  }

  async getMetrics(): Promise<string> {
    return await register.metrics();
  }
}
```

8.2 Health Check System

typescript

```typescript
// health/health.controller.ts
import { Controller, Get } from '@nestjs/common';
import { HealthCheckService, HealthCheck, TypeOrmHealthIndicator } from '@nestjs/termi

@Controller('health')
export class HealthController {
  constructor(
    private health: HealthCheckService,
    private db: TypeOrmHealthIndicator,
    private airbyteHealth: AirbyteHealthIndicator,
    private redisHealth: RedisHealthIndicator
  ) {}

  @Get()
  @HealthCheck()
  check() {
    return this.health.check([
      () => this.db.pingCheck('database'),
      () => this.redisHealth.pingCheck('redis'),
      () => this.airbyteHealth.pingCheck('airbyte'),
      () => this.checkMLService(),
      () => this.checkStorageService()
    ]);
  }

  @Get('detailed')
  @HealthCheck()
  detailedCheck() {
    return this.health.check([
      () => this.db.pingCheck('database'),
      () => this.redisHealth.pingCheck('redis'),
      () => this.airbyteHealth.detailedCheck('airbyte'),
      () => this.checkMLService(),
      () => this.checkStorageService(),
      () => this.checkExternalAPIs()
    ]);
  }
}
```

8.3 Alerting Configuration

yaml

```yaml
# monitoring/alerts.yml
groups:
- name: infinure.rules
  rules:
  - alert: HighErrorRate
    expr: rate(infinure_http_requests_total{status_code!~"2.."}[5m]) > 0.1
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "High error rate detected"
      description: "Error rate is {{ $value | humanizePercentage }} for the last 5 min

  - alert: ChatQueryLatencyHigh
    expr: histogram_quantile(0.95, rate(infinure_chat_query_duration_seconds_bucket[5m
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: "High chat query latency"
      description: "95th percentile latency is {{ $value }}s"

  - alert: AirbyteConnectionDown
    expr: infinure_airbyte_connections_active < 1
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "Airbyte connection failure"
      description: "No active Airbyte connections detected"

  - alert: DatabaseConnectionFailure
    expr: up{job="infinure-postgres"} == 0
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "Database connection failure"
      description: "PostgreSQL database is not accessible"

  - alert: HighMemoryUsage
    expr: (1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)) > 0.85
    for: 5m
    labels:
      severity: warning
    annotations:
```

```
      summary: "High memory usage"
      description: "Memory usage is above 85%"

  - alert: SecurityViolation
    expr: increase(infinure_security_violations_total[5m]) > 0
    for: 0m
    labels:
      severity: critical
    annotations:
      summary: "Security violation detected"
      description: "{{ $value }} security violations in the last 5 minutes"
```

---

9. DEPLOYMENT ARCHITECTURE

9.1 Kubernetes Manifests

yaml

```yaml
# k8s/namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: infinure
  labels:
    name: infinure
---
# k8s/backend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: infinure-backend
  namespace: infinure
spec:
  replicas: 3
  selector:
    matchLabels:
      app: infinure-backend
  template:
    metadata:
      labels:
        app: infinure-backend
    spec:
      containers:
      - name: backend
        image: infinure/backend:latest
        resources:
          requests:
            memory: "1Gi"
            cpu: "500m"
          limits:
            memory: "2Gi"
            cpu: "1000m"
        env:
        - name: NODE_ENV
          value: "production"
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: infinure-secrets
              key: database-url
        - name: REDIS_URL
          valueFrom:
            secretKeyRef:
              name: infinure-secrets
```

```
            key: redis-url
        - name: JWT_SECRET
          valueFrom:
            secretKeyRef:
              name: infinure-secrets
              key: jwt-secret
        ports:
        - containerPort: 3000
        livenessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 5
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: infinure-backend-service
  namespace: infinure
spec:
  selector:
    app: infinure-backend
  ports:
  - protocol: TCP
    port: 80
    targetPort: 3000
  type: ClusterIP
```

9.2 Frontend Deployment

yaml

```yaml
# k8s/frontend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: infinure-frontend
  namespace: infinure
spec:
  replicas: 2
  selector:
    matchLabels:
      app: infinure-frontend
  template:
    metadata:
      labels:
        app: infinure-frontend
    spec:
      containers:
      - name: frontend
        image: infinure/frontend:latest
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        env:
        - name: NEXT_PUBLIC_API_URL
          value: "https://api.infinure.com"
        - name: NEXT_PUBLIC_WS_URL
          value: "wss://ws.infinure.com"
        ports:
        - containerPort: 3000
        livenessProbe:
          httpGet:
            path: /api/health
            port: 3000
          initialDelaySeconds: 30
        readinessProbe:
          httpGet:
            path: /api/health
            port: 3000
          initialDelaySeconds: 5
---
apiVersion: v1
kind: Service
```

```
  metadata:
    name: infinure-frontend-service
    namespace: infinure
  spec:
    selector:
      app: infinure-frontend
    ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
    type: ClusterIP
```

9.3 Database Configuration

yaml

```yaml
# k8s/postgres-deployment.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
  namespace: infinure
spec:
  serviceName: postgres
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
      - name: postgres
        image: postgres:15-alpine
        resources:
          requests:
            memory: "2Gi"
            cpu: "1000m"
          limits:
            memory: "4Gi"
            cpu: "2000m"
        env:
        - name: POSTGRES_DB
          value: "infinure"
        - name: POSTGRES_USER
          valueFrom:
            secretKeyRef:
              name: postgres-secret
              key: username
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: postgres-secret
              key: password
        ports:
        - containerPort: 5432
        volumeMounts:
        - name: postgres-storage
          mountPath: /var/lib/postgresql/data
  volumeClaimTemplates:
```

```
  - metadata:
      name: postgres-storage
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 100Gi
```

9.4 CI/CD Pipeline

yaml

```yaml
# .github/workflows/deploy.yml
name: Deploy Infinure
on:
  push:
    branches: [main, develop]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18'

      - name: Install dependencies
        run: |
          npm ci

      - name: Run tests
        run: |
          npm run test:unit
          npm run test:integration
          npm run test:e2e

      - name: Run security scan
        run: |
          npm run security:scan

      - name: Build application
        run: |
          npm run build

  deploy-staging:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/develop'
    steps:
      - uses: actions/checkout@v4

      - name: Build and push Docker images
        run: |
          docker build -t infinure/backend:staging .
          docker build -t infinure/frontend:staging ./frontend
```

```
          docker push infinure/backend:staging
          docker push infinure/frontend:staging

      - name: Deploy to staging
        run: |
          kubectl apply -f k8s/staging/
          kubectl rollout status deployment/infinure-backend -n infinure-staging
          kubectl rollout status deployment/infinure-frontend -n infinure-staging

      - name: Run smoke tests
        run: |
          npm run test:smoke -- --env=staging

  deploy-production:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    steps:
      - uses: actions/checkout@v4

      - name: Build and push Docker images
        run: |
          docker build -t infinure/backend:latest .
          docker build -t infinure/frontend:latest ./frontend
          docker push infinure/backend:latest
          docker push infinure/frontend:latest

      - name: Deploy to production
        run: |
          kubectl apply -f k8s/production/
          kubectl rollout status deployment/infinure-backend -n infinure
          kubectl rollout status deployment/infinure-frontend -n infinure

      - name: Run production health checks
        run: |
          npm run health:check -- --env=production
```

---

10. DEVELOPMENT PHASES

Phase 1: Core Infrastructure

Duration: 4-6 weeks

Backend Development:

- [x] Setup NestJS project structure

- [x] Implement authentication service (JWT + MFA)
- [x] Setup PostgreSQL multi-tenant database
- [x] Implement user management and RBAC
- [x] Basic API endpoints for auth and user management
- [x] Setup Redis for caching and sessions

Frontend Development:

- [x] Setup Next.js 14 project with TypeScript
- [x] Implement authentication pages (login/signup)
- [x] Create basic dashboard layout
- [x] Setup Tailwind CSS design system
- [x] Implement responsive navigation

Infrastructure:

- [x] Setup Kubernetes cluster
- [x] Configure PostgreSQL StatefulSet
- [x] Setup Redis cluster
- [x] Basic monitoring with Prometheus

Phase 2: Airbyte Integration

Duration: 3-4 weeks

Integration Service:

- [ ] Deploy Airbyte Open Source self-hosted
- [ ] Implement Airbyte management service
- [ ] Create organization workspace isolation
- [ ] Build connector registry and management
- [ ] Implement credential encryption and security

Frontend Integration:

- [ ] Build integration management UI
- [ ] Create connector selection interface
- [ ] Implement connection status monitoring
- [ ] Add sync scheduling interface

Testing:

- [ ] Test 5-10 priority connectors
- [ ] Validate multi-tenant data isolation
- [ ] Security testing for credential handling

Phase 3: Chat Interface

Duration: 4-5 weeks

Chat Service:

- [ ] Implement chat service with WebSocket support
- [ ] Create conversation and message management
- [ ] Build context service for role/industry awareness
- [ ] Integrate with ML service interface
- [ ] Implement response customization by role

Frontend Chat:

- [ ] Build real-time chat interface
- [ ] Implement typing indicators and message status
- [ ] Create role-specific UI customizations
- [ ] Add chart and visualization rendering
- [ ] Mobile responsive chat interface

ML Integration:

- [ ] Define ML service interface/contract
- [ ] Create mock ML service for testing
- [ ] Implement context passing to ML service
- [ ] Add response post-processing

Phase 4: Enterprise Features

Duration: 3-4 weeks

Security & Compliance:

- [ ] Implement audit logging
- [ ] Add data encryption at rest
- [ ] Setup compliance monitoring
- [ ] Implement DLP scanning
- [ ] Add security event alerting

Analytics & Monitoring:

- [ ] Implement usage analytics
- [ ] Setup performance monitoring
- [ ] Create admin analytics dashboard
- [ ] Add business metrics tracking

Advanced Features:

- [ ] Industry-specific configurations
- [ ] Advanced role permissions
- [ ] Bulk data operations
- [ ] Advanced chat features (file upload, etc.)

---

11. API DOCUMENTATION

11.1 Authentication Endpoints

typescript

```
/**
 * @swagger
 * /auth/login:
 *   post:
 *     summary: User login
 *     tags: [Authentication]
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             properties:
 *               email:
 *                 type: string
 *               password:
 *                 type: string
 *             required:
 *               - email
 *               - password
 *     responses:
 *       200:
 *         description: Login successful
 *         content:
 *           application/json:
 *             schema:
 *               type: object
 *               properties:
 *                 accessToken:
 *                   type: string
 *                 refreshToken:
 *                   type: string
 *                 user:
 *                   $ref: '#/components/schemas/User'
 *       401:
 *         description: Invalid credentials
 *       429:
 *         description: Too many login attempts
 */

/**
 * @swagger
 * /auth/refresh:
 *   post:
 *     summary: Refresh access token
 *     tags: [Authentication]
```

```
 *     security:
 *       - refreshToken: []
 *     responses:
 *       200:
 *         description: Token refreshed successfully
 *       401:
 *         description: Invalid or expired refresh token
 */
```

11.2 Integration Endpoints

typescript

```
/**
 * @swagger
 * /integrations/sources:
 *   post:
 *     summary: Create new data source
 *     tags: [Integrations]
 *     security:
 *       - bearerAuth: []
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             properties:
 *               name:
 *                 type: string
 *               type:
 *                 type: string
 *                 enum: [postgres, mysql, mongodb, salesforce, stripe]
 *               credentials:
 *                 type: object
 *               syncFrequency:
 *                 type: string
 *                 enum: [hourly, daily, weekly, monthly]
 *             required:
 *               - name
 *               - type
 *               - credentials
 *     responses:
 *       201:
 *         description: Data source created successfully
 *       400:
 *         description: Invalid request data
 *       403:
 *         description: Insufficient permissions
 */

/**
 * @swagger
 * /integrations/sources/{sourceId}/sync:
 *   post:
 *     summary: Trigger manual sync
 *     tags: [Integrations]
 *     security:
 *       - bearerAuth: []
```

```
 *     parameters:
 *       - in: path
 *         name: sourceId
 *         required: true
 *         schema:
 *           type: string
 *     responses:
 *       200:
 *         description: Sync triggered successfully
 *       404:
 *         description: Data source not found
 *       403:
 *         description: Insufficient permissions
 */
```

11.3 Chat Endpoints

typescript

```
/**
 * @swagger
 * /chat/conversations:
 *   post:
 *     summary: Create new conversation
 *     tags: [Chat]
 *     security:
 *       - bearerAuth: []
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             properties:
 *               title:
 *                 type: string
 *     responses:
 *       201:
 *         description: Conversation created
 */

/**
 * @swagger
 * /chat/conversations/{conversationId}/messages:
 *   post:
 *     summary: Send message in conversation
 *     tags: [Chat]
 *     security:
 *       - bearerAuth: []
 *     parameters:
 *       - in: path
 *         name: conversationId
 *         required: true
 *         schema:
 *           type: string
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             properties:
 *               content:
 *                 type: string
 *             required:
```

```
 *               - content
 *     responses:
 *       201:
 *         description: Message sent successfully
 *       404:
 *         description: Conversation not found
 */
```

---

12. TESTING STRATEGY

12.1 Test Structure

```
tests/
├── unit/                    # Unit tests
│   ├── services/
│   ├── controllers/
│   └── utils/
├── integration/             # Integration tests
│   ├── auth/
│   ├── chat/
│   └── integrations/
├── e2e/                     # End-to-end tests
│   ├── auth.spec.ts
│   ├── chat.spec.ts
│   └── integrations.spec.ts
├── security/                # Security tests
│   ├── auth-security.spec.ts
│   └── data-security.spec.ts
└── performance/             # Performance tests
    ├── load-tests/
    └── stress-tests/
```

12.2 Test Examples

typescript

```typescript
// tests/integration/auth/auth.spec.ts
describe('Authentication Integration', () => {
  it('should authenticate user with valid credentials', async () => {
    const response = await request(app)
      .post('/auth/login')
      .send({
        email: 'test@example.com',
        password: 'securePassword123!'
      })
      .expect(200);

    expect(response.body).toHaveProperty('accessToken');
    expect(response.body).toHaveProperty('refreshToken');
    expect(response.body.user).toHaveProperty('role');
  });

  it('should require MFA for organizations with MFA enabled', async () => {
    const response = await request(app)
      .post('/auth/login')
      .send({
        email: 'mfa-user@example.com',
        password: 'securePassword123!'
      })
      .expect(200);

    expect(response.body.requiresMFA).toBe(true);
    expect(response.body).toHaveProperty('mfaToken');
  });
});

// tests/e2e/chat.spec.ts
describe('Chat E2E', () => {
  test('should allow user to chat with data', async ({ page }) => {
    await page.goto('/login');
    await page.fill('#email', 'analyst@testorg.com');
    await page.fill('#password', 'password123');
    await page.click('button[type="submit"]');

    await page.waitForURL('/dashboard');
    await page.click('[data-testid="chat-link"]');

    const chatInput = page.locator('[data-testid="chat-input"]');
    await chatInput.fill('Show me revenue trends for this quarter');
    await chatInput.press('Enter');

    const responseMessage = page.locator('[data-testid="assistant-message"]').last();
```

```
    await expect(responseMessage).toBeVisible();
    await expect(responseMessage).toContainText('revenue');
  });
});
```

This comprehensive technical specification provides all the necessary details for a freelancer to develop the Infinure platform. The document covers architecture, database design, security, integrations, frontend/backend implementation, deployment, and testing strategies.

The specification emphasizes:

- **Enterprise-grade security** with multi-tenant isolation
- **Scalable microservices architecture**
- **Self-hosted Airbyte** for data integrations
- **Role-based contextual responses**
- **Industry-specific configurations**
- **Comprehensive monitoring and compliance**

All code examples are production-ready and include proper error handling, security measures, and scalability considerations.