DraftGenie UI/UX Design Specification

Frontend Development Guide & API Integration Reference

Version: 1.0
Last Updated: 2025-10-16
Status: Production Ready
Target Audience: UI/UX Developers, Frontend Engineers, Product Designers

Table of Contents

1. Executive Summary

1.1 Product Overview

DraftGenie is a Speaker-centric AI-powered platform that improves medical draft quality by learning from speaker-specific patterns and applying personalized corrections using Retrieval-Augmented Generation (RAG) with Google Gemini.

1.2 Design Goals

- **Speaker-First Experience:** All workflows begin with speaker context
- **Clarity & Simplicity:** Complex AI processes presented in intuitive interfaces
- **Data Transparency:** Clear visualization of quality metrics and improvements
- **Efficiency:** Streamlined workflows for common tasks (SSA, BSA, DFN generation)
- **Trust & Confidence:** Clear feedback on AI-generated content and quality scores

1.3 Key User Journeys

1. **Speaker Onboarding** (SSA/BSA) - Add speakers to the system
2. **Draft Management** - View and manage historical drafts
3. **AI-Powered Generation** - Generate improved final notes (DFN)
4. **Quality Evaluation** - Review metrics and bucket assignments
5. **Dashboard Analytics** - Monitor system-wide performance

1.4 Technical Foundation

- **Backend:** Azure-hosted microservices (Node.js + Python)
- **API Gateway:** `https://api-gateway.gentleforest-322351b3.southindia.azurecontainerapps.io`
- **Authentication:** JWT-based with 24-hour token lifetime
- **API Documentation:** Swagger UI at `/api/docs`

---

2. Design Philosophy & Principles

2.1 Core Design Principles

2.1.1 Speaker-Centric Design

- **Every screen should answer:** "Which speaker am I working with?"
- Speaker context should be persistent and visible throughout workflows
- Quick speaker switching should be available from any screen

2.1.2 Progressive Disclosure

- Show essential information first, details on demand
- Use expandable sections for complex data (metrics, evaluations)
- Provide drill-down capabilities from summary to detail views

2.1.3 Feedback & Transparency

- Clear loading states for AI operations (can take 10-30 seconds)
- Progress indicators for multi-step workflows
- Explicit success/error messages with actionable guidance

2.1.4 Data Visualization

- Use color-coded buckets (NO-TOUCH → VERY-HIGH-TOUCH)
- Visual comparison of DFN vs IFN quality
- Trend charts for speaker performance over time

2.2 Visual Design Language

2.2.1 Color Palette

Bucket Colors (Quality Indicators):

- 🟢 **NO-TOUCH:** `#4CAF50` (Green)
- 🔵 **LOW-TOUCH:** `#2196F3` (Blue)
- 🟡 **MEDIUM-TOUCH:** `#FFC107` (Amber)
- 🟠 **HIGH-TOUCH:** `#FF9800` (Orange)
- 🔴 **VERY-HIGH-TOUCH:** `#F44336` (Red)

System Colors:

- **Primary:** `#1976D2` (Professional Blue)
- **Secondary:** `#424242` (Dark Gray)
- **Success:** `#4CAF50` (Green)
- **Warning:** `#FF9800` (Orange)
- **Error:** `#F44336` (Red)
- **Info:** `#2196F3` (Blue)

2.2.2 Typography

- **Headings:** Sans-serif, bold (e.g., Roboto, Inter, Open Sans)
- **Body:** Sans-serif, regular (14-16px for readability)
- **Code/IDs:** Monospace (e.g., Roboto Mono, Fira Code)
- **Medical Content:** Serif or high-readability sans-serif

2.2.3 Spacing & Layout

- **Grid System:** 8px base unit (8, 16, 24, 32, 48, 64px)
- **Card Padding:** 24px
- **Section Spacing:** 32px between major sections
- **Responsive Breakpoints:** Mobile (320px), Tablet (768px), Desktop (1024px+)

2.3 Interaction Patterns

2.3.1 Loading States

- **Skeleton screens** for list views
- **Spinner with message** for AI operations ("Generating DFN with AI...")
- **Progress bars** for multi-step workflows (1/3, 2/3, 3/3)

2.3.2 Error Handling

- **Inline validation** for form fields
- **Toast notifications** for success/error messages
- **Error boundaries** with retry options
- **Graceful degradation** when services are unavailable

2.3.3 Navigation

- **Persistent sidebar** with main sections
- **Breadcrumbs** for deep navigation
- **Quick actions** in header (Create Speaker, Generate DFN)
- **Context menu** for item-specific actions

---

3. User Personas & Roles

3.1 Primary Personas

3.1.1 DraftGenie Administrator

Role: System administrator managing speakers and quality **Goals:**

- Onboard new speakers efficiently (SSA/BSA)
- Monitor overall system quality metrics
- Reassign speaker buckets based on performance
- Troubleshoot quality issues

Key Screens:

- Dashboard (overview)
- Speaker Management
- Bulk Operations
- System Settings

3.1.2 Quality Analyst

Role: Reviews and evaluates draft quality **Goals:**

- Compare DFN vs IFN quality
- Analyze quality metrics (SER, WER)
- Identify patterns in corrections
- Provide feedback for system improvement

Key Screens:

- Evaluation Dashboard
- Draft Comparison View
- Metrics & Analytics
- Speaker Performance Reports

3.1.3 Medical Professional (Speaker)

Role: End user whose drafts are being improved **Goals:**

- View their own performance metrics
- See improvement over time
- Understand correction patterns
- Access final notes (DFN)

Key Screens:

- Personal Dashboard
- My Drafts
- My Performance
- Final Notes Library

3.2 User Permissions

Feature	Admin	Quality Analyst	Speaker
View Dashboard	✅	✅	✅ (Limited)

Feature	Admin	Quality Analyst	Speaker
Create Speakers	✅	❌	❌
Update Buckets	✅	✅	❌
Generate DFN	✅	✅	❌
View All Speakers	✅	✅	❌
View Own Data	✅	✅	✅
System Settings	✅	❌	❌

4. Core User Flows

4.1 Speaker Onboarding Flow (SSA - Single Speaker Addition)

4.1.1 User Journey

```
1. Admin clicks "Add Speaker" button
2. Form appears with required fields
3. Admin enters speaker information
4. System validates and creates speaker
5. Background process ingests historical drafts
6. Success confirmation with speaker profile link
```

4.1.2 UI Requirements

Screen: Speaker Creation Form

Required Fields:

- Name (text, min 2 chars)
- Email (email format, optional)
- Initial Bucket (dropdown: NO-TOUCH, LOW-TOUCH, MEDIUM-TOUCH, HIGH-TOUCH, VERY-HIGH-TOUCH)
- External ID (text, from InstaNote)

Optional Fields:

- Notes (textarea, for admin comments)
- Metadata (key-value pairs: specialty, hospital, etc.)

Validation:

- Real-time validation on blur
- Duplicate check for External ID
- Clear error messages below fields

Actions:

- Primary: "Create Speaker" (blue button)
- Secondary: "Cancel" (gray button)

Success State:

- Toast notification: "Speaker created successfully"
- Redirect to speaker profile page
- Show background ingestion status

4.1.3 API Integration

Endpoint: `POST /api/v1/speakers`

Request:

```json
{
  "name": "Dr. John Smith",
  "email": "john.smith@hospital.com",
  "bucket": "LOW-TOUCH",
  "externalId": "instanote-speaker-12345",
  "notes": "Cardiologist with 15 years of experience",
  "metadata": {
    "specialty": "Cardiology",
    "hospital": "General Hospital"
  }
}
```

Response (201 Created):

```json
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "name": "Dr. John Smith",
  "bucket": "LOW-TOUCH",
  "status": "ACTIVE",
  "createdAt": "2025-10-16T10:30:00Z"
}
```

Error Handling:

- 409 Conflict: "Speaker with this External ID already exists"
- 400 Bad Request: Show validation errors inline

4.2 Batch Speaker Addition Flow (BSA)

4.2.1 User Journey

```
1. Admin clicks "Bulk Import" button
2. Upload CSV/Excel file or paste data
3. System validates all entries
4. Preview table shows validation results
5. Admin reviews and confirms
6. System processes speakers sequentially
7. Progress indicator shows completion status
8. Summary report with success/failure counts
```

4.2.2 UI Requirements

Screen: Bulk Import Wizard

Step 1: Upload

- File upload (CSV, XLSX)
- Or paste data (tab-separated)
- Template download link

Step 2: Validation

- Table preview with validation status
- Color-coded rows (green=valid, red=error)
- Error messages per row
- Option to fix errors inline

Step 3: Confirmation

- Summary: X valid, Y errors
- Option to proceed with valid only
- Estimated processing time

Step 4: Processing

- Progress bar (X of Y completed)
- Real-time status updates
- Cancel option

Step 5: Results

- Success count
- Error count with details
- Download error report
- Link to speaker list

4.2.3 CSV Template Format

```
name,email,bucket,externalId,notes,specialty,hospital
Dr. John Smith,john@hospital.com,GOOD,inst-
```

```
001,Cardiologist,Cardiology,General Hospital
Dr. Jane Doe,jane@hospital.com,EXCELLENT,inst-
002,Neurologist,Neurology,City Hospital
```

4.3 Draft Management Flow

4.3.1 User Journey - View Speaker Drafts

```
1. User navigates to speaker profile
2. Drafts tab shows list of historical drafts
3. Filter by type (AD, LD, IFN) and status
4. Click draft to view details
5. See original vs corrected text comparison
6. View correction patterns identified
```

4.3.2 UI Requirements

Screen: Speaker Drafts List

Layout:

- Table view with columns:
  - Draft ID (truncated, tooltip shows full)
  - Type (badge: AD/LD/IFN)
  - Word Count
  - Corrections
  - Date
  - Status (badge: Processed/Pending)
  - Actions (View, Compare)

Filters:

- Draft Type (multi-select)
- Date Range (date picker)
- Processing Status (toggle)

Pagination:

- 20 items per page
- Page numbers + Next/Previous
- Jump to page input

Draft Detail View:

- Side-by-side comparison (Original | Corrected)
- Highlighted differences
- Correction categories (spelling, grammar, terminology)

- Metadata panel (word count, date, source)

4.3.3 API Integration

List Drafts: `GET /api/v1/drafts/speaker/{speakerId}?skip=0&limit=20`

Get Draft Details: `GET /api/v1/drafts/{draftId}`

4.4 AI-Powered DFN Generation Flow

4.4.1 User Journey

```
1. User selects speaker from list
2. Clicks "Generate DFN" button
3. Modal appears with generation options
4. User enters prompt (optional) and confirms
5. Loading state shows AI processing (10-30 seconds)
6. Success: DFN displayed with quality metrics
7. User can review, edit, or regenerate
```

4.4.2 UI Requirements

Screen: DFN Generation Modal

Input Section:

- Speaker name (read-only, for context)
- Prompt (textarea, optional, placeholder: "Generate a professional medical note")
- Advanced options (collapsible):
  - Use LangGraph (toggle, default: ON)
  - Max Tokens (slider, 500-4000, default: 2000)

Loading State:

- Animated spinner
- Message: "Generating DFN with AI..."
- Sub-message: "This may take 10-30 seconds"
- Progress steps:
  1. Validating speaker ✓
  2. Retrieving context ✓
  3. Generating with AI... ⏳

Success State:

- Generated DFN text (formatted, scrollable)
- Quality metrics panel:
  - Confidence Score (0-100%)
  - Context Vectors Used (number)

    - Generation Time (seconds)
- Actions:
    - "Accept & Save" (primary)
    - "Regenerate" (secondary)
    - "Edit" (secondary)
    - "Cancel" (tertiary)

Error State:

- Error icon + message
- Possible causes
- "Retry" button
- "Contact Support" link

4.4.3 API Integration

Endpoint: `POST /api/v1/workflow/generate-dfn`

Request:

```json
{
  "speakerId": "550e8400-e29b-41d4-a716-446655440000",
  "prompt": "Generate a professional medical note",
  "options": {
    "useLangGraph": true,
    "maxTokens": 2000
  }
}
```

Response (201 Created):

```json
{
  "workflow": {
    "status": "completed",
    "steps": [...]
  },
  "generation": {
    "dfn_text": "Generated medical note...",
    "confidence": 0.95
  },
  "dfn": {
    "draft_id": "dfn_789012",
    "created_at": "2025-10-16T10:35:00Z"
  }
}
```

4.5 Quality Evaluation & Metrics Flow

4.5.1 User Journey

```
1. User navigates to Evaluations section
2. Dashboard shows aggregate metrics
3. Filter by speaker, date range, bucket
4. Click evaluation to see detailed comparison
5. View DFN vs IFN side-by-side
6. See quality scores (SER, WER, Similarity)
7. Review bucket recommendation
8. Approve or override bucket change
```

4.5.2 UI Requirements

Screen: Evaluation Dashboard

Metrics Cards (Top Row):

- Total Evaluations (number + trend)
- Average Quality Score (percentage + trend)
- Bucket Distribution (donut chart)
- Recent Activity (timeline)

Evaluation List:

- Table with columns:
  - Speaker Name
  - Evaluation Date
  - Quality Score (color-coded)
  - Current Bucket → Recommended Bucket
  - Status (Pending/Approved/Rejected)
  - Actions (View Details, Approve)

Evaluation Detail View:

- Header: Speaker info + evaluation metadata
- Comparison Panel:
  - Left: IFN (InstaNote Final)
  - Right: DFN (DraftGenie Final)
  - Diff highlighting
- Metrics Panel:
  - SER (Sentence Edit Rate): X%
  - WER (Word Error Rate): Y%
  - Similarity Score: Z%
  - Overall Quality: A/100
- Bucket Recommendation:
  - Current: LOW-TOUCH
  - Recommended: EXCELLENT
  - Reason: "Quality score improved by 15%"

    - Actions: Approve / Reject / Defer

4.5.3 API Integration

List Evaluations: `GET /api/v1/evaluations?skip=0&limit=20`

Get Evaluation Details: `GET /api/v1/evaluations/{evaluationId}`

Approve Bucket Change: `PUT /api/v1/speakers/{speakerId}/bucket`

4.6 Dashboard & Analytics Flow

4.6.1 User Journey

```
1. User logs in → lands on dashboard
2. Overview shows key metrics at a glance
3. Charts visualize trends over time
4. Quick actions for common tasks
5. Recent activity feed
6. Drill down into specific areas
```

4.6.2 UI Requirements

Screen: Main Dashboard

Layout: 3-column grid (responsive)

Top Section - KPIs:

- Total Speakers (number + change from last month)
- Total Drafts Processed (number + trend)
- Average Quality Score (percentage + trend)
- Active Evaluations (number + pending count)

Middle Section - Visualizations:

- Speaker Distribution by Bucket (bar chart)
- Quality Trends Over Time (line chart)
- Draft Processing Volume (area chart)
- Top Performing Speakers (leaderboard)

Bottom Section - Activity:

- Recent Speakers Added (list with avatars)
- Recent DFNs Generated (list with status)
- Pending Evaluations (list with actions)
- System Health (service status indicators)

Quick Actions (Floating Action Button or Header):

- Add Speaker (SSA)
- Bulk Import (BSA)
- Generate DFN
- View Reports

4.6.3 API Integration

Dashboard Metrics: `GET /api/v1/dashboard/metrics`

Response:

```
{
  "speakers": {
    "total": 500,
    "byBucket": {
      "NO_TOUCH": 50,
      "LOW_TOUCH": 200,
      "MEDIUM_TOUCH": 150,
      "HIGH_TOUCH": 75,
      "VERY_HIGH_TOUCH": 25
    }
  },
  "drafts": {
    "total": 5000,
    "byType": {...}
  },
  "summary": {
    "totalSpeakers": 500,
    "totalDrafts": 5000,
    "healthPercentage": 100
  }
}
```

---

5. UI Components & Screens

5.1 Component Library

5.1.1 Core Components

SpeakerCard

- Purpose: Display speaker summary in lists
- Props: speaker (object), onClick (function), showActions (boolean)
- Layout:
  - Avatar/Initials (left)
  - Name + Email (center)
  - Bucket badge (right)
  - Metadata (bottom, optional)

- States: Default, Hover, Selected, Disabled

BucketBadge

- Purpose: Visual indicator of speaker quality bucket
- Props: bucket (enum), size (sm/md/lg)
- Variants: Filled, Outlined, Minimal
- Colors: Mapped to bucket colors (see 2.2.1)

DraftComparison

- Purpose: Side-by-side text comparison with diff highlighting
- Props: original (string), corrected (string), showLineNumbers (boolean)
- Features:
  - Syntax highlighting for changes
  - Line-by-line diff
  - Word-level highlighting
  - Expand/collapse sections

MetricsPanel

- Purpose: Display quality metrics in consistent format
- Props: metrics (object), layout (horizontal/vertical)
- Metrics:
  - SER (progress bar + percentage)
  - WER (progress bar + percentage)
  - Similarity (circular progress)
  - Quality Score (large number + trend)

LoadingState

- Purpose: Consistent loading indicators
- Variants:
  - Skeleton (for lists/cards)
  - Spinner (for actions)
  - Progress (for multi-step workflows)
- Props: type (variant), message (string), progress (0-100)

5.1.2 Form Components

SpeakerForm

- Fields: Name, Email, Bucket, External ID, Notes, Metadata
- Validation: Real-time with error messages
- Layout: 2-column on desktop, 1-column on mobile
- Actions: Submit, Cancel, Reset

BulkImportWizard

- Steps: Upload → Validate → Confirm → Process → Results
- Progress indicator at top

- Step navigation (Next, Previous, Cancel)
- Persistent data across steps

DFNGenerationForm

- Fields: Speaker (read-only), Prompt, Options
- Collapsible advanced options
- Preview of speaker context
- Submit triggers modal with loading state

5.2 Screen Specifications

5.2.1 Speaker List Screen

URL: `/speakers`

Layout:

- Header: Title + Search + Filters + "Add Speaker" button
- Filters: Bucket (multi-select), Status (multi-select), Search (text)
- Table/Grid view toggle
- Pagination at bottom

Table Columns:

- Name (sortable, clickable)
- Email
- Bucket (badge, filterable)
- Status (badge)
- Drafts Count
- Last Updated (sortable)
- Actions (View, Edit, Delete)

Grid View:

- SpeakerCard components in responsive grid
- 4 columns on desktop, 2 on tablet, 1 on mobile

Empty State:

- Illustration + message: "No speakers yet"
- "Add Your First Speaker" button

5.2.2 Speaker Profile Screen

URL: `/speakers/{id}`

Layout: Tabbed interface

Header:

- Speaker name + avatar

- Bucket badge (editable by admin)
- Status indicator
- Actions: Edit, Delete, Generate DFN

Tabs:

1. **Overview**
    - Summary metrics
    - Recent activity
    - Quick stats
2. **Drafts**
    - List of historical drafts
    - Filters and search
    - Pagination
3. **Evaluations**
    - List of evaluations
    - Quality trends chart
    - Bucket history
4. **Settings**
    - Edit speaker details
    - Manage metadata
    - Audit log

5.2.3 DFN Generation Screen

URL: `/dfn/generate` or Modal

Layout:

- Speaker selection (if not pre-selected)
- Generation form
- Preview panel (after generation)
- Actions panel

Workflow:

1. Select speaker
2. Enter prompt (optional)
3. Configure options
4. Generate (shows loading)
5. Review result
6. Accept/Regenerate/Edit

5.2.4 Evaluation Dashboard Screen

URL: `/evaluations`

Layout:

- Metrics cards at top
- Filters: Speaker, Date Range, Status
- Evaluation list (table)
- Pagination

Detail View (Modal or Side Panel):

- Evaluation metadata
- DFN vs IFN comparison
- Quality metrics
- Bucket recommendation
- Approval actions

---

6. API Integration Guide

6.1 Authentication Flow

6.1.1 Initial Setup

Base URL:

```
const API_BASE_URL = 'https://api-gateway.gentleforest-
322351b3.southindia.azurecontainerapps.io/api/v1';
```

API Client Configuration:

```
import axios from 'axios';

const apiClient = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
  timeout: 30000, // 30 seconds for AI operations
});
```

6.1.2 Token Management

Store Tokens:

```
// After successful login
localStorage.setItem('accessToken', response.data.accessToken);
localStorage.setItem('refreshToken', response.data.refreshToken);
```

Add Token to Requests:

```
apiClient.interceptors.request.use((config) => {
  const token = localStorage.getItem('accessToken');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
```

Handle Token Refresh:

```
apiClient.interceptors.response.use(
  (response) => response,
  async (error) => {
    if (error.response?.status === 401) {
      const refreshToken = localStorage.getItem('refreshToken');
      if (refreshToken) {
        try {
          const { data } = await
axios.post(`${API_BASE_URL}/auth/refresh`, {
            refreshToken,
          });
          localStorage.setItem('accessToken', data.accessToken);
          localStorage.setItem('refreshToken', data.refreshToken);

          // Retry original request
          error.config.headers.Authorization = `Bearer
${data.accessToken}`;
          return axios(error.config);
        } catch (refreshError) {
          // Redirect to login
          window.location.href = '/login';
        }
      }
    }
    return Promise.reject(error);
  }
);
```

6.1.3 Login/Logout

Login:

```
async function login(email, password) {
  const { data } = await apiClient.post('/auth/login', { email, password
});
  localStorage.setItem('accessToken', data.accessToken);
  localStorage.setItem('refreshToken', data.refreshToken);
  return data.user;
}
```

Logout:

```
async function logout() {
  const refreshToken = localStorage.getItem('refreshToken');
  await apiClient.post('/auth/logout', { refreshToken });
  localStorage.removeItem('accessToken');
  localStorage.removeItem('refreshToken');
  window.location.href = '/login';
}
```

6.2 Speaker Management APIs

6.2.1 Create Speaker

Endpoint: `POST /api/v1/speakers`

UI Trigger: "Add Speaker" button click

Request:

```
async function createSpeaker(speakerData) {
  const { data } = await apiClient.post('/speakers', {
    name: speakerData.name,
    email: speakerData.email,
    bucket: speakerData.bucket,
    externalId: speakerData.externalId,
    notes: speakerData.notes,
    metadata: speakerData.metadata
  });
  return data;
}
```

Success Handling:

- Show toast: "Speaker created successfully"
- Redirect to speaker profile: `/speakers/${data.id}`
- Trigger background draft ingestion (automatic)

Error Handling:

- 409 Conflict: "Speaker with this External ID already exists"
- 400 Bad Request: Display validation errors inline
- 500 Server Error: "Unable to create speaker. Please try again."

6.2.2 List Speakers

Endpoint: `GET /api/v1/speakers`

UI Trigger: Navigate to speakers list

Request with Filters:

```
async function getSpeakers(filters) {
  const { data } = await apiClient.get('/speakers', {
    params: {
      page: filters.page || 1,
      limit: filters.limit || 20,
      bucket: filters.bucket, // Optional
      status: filters.status, // Optional
      search: filters.search, // Optional
      sortBy: filters.sortBy || 'createdAt',
      sortOrder: filters.sortOrder || 'desc'
    }
  });
  return data;
}
```

Response Structure:

```
{
  "data": [...], // Array of speaker objects
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 150,
    "totalPages": 8,
    "hasNext": true,
    "hasPrevious": false
  }
}
```

UI Updates:

- Populate table/grid with `data` array
- Update pagination controls with `pagination` object
- Show loading skeleton while fetching
- Show empty state if `data` is empty

6.2.3 Get Speaker Details

Endpoint: `GET /api/v1/speakers/{id}`

UI Trigger: Click speaker name or "View" button

Request:

```
async function getSpeaker(speakerId) {
  const { data } = await apiClient.get(`/speakers/${speakerId}`);
  return data;
}
```

UI Updates:

- Populate speaker profile header
- Display metadata in overview tab
- Enable tab navigation

6.2.4 Update Speaker

Endpoint: `PATCH /api/v1/speakers/{id}`

UI Trigger: "Save" button in edit form

Request:

```
async function updateSpeaker(speakerId, updates) {
  const { data } = await apiClient.patch(`/speakers/${speakerId}`,
updates);
  return data;
}
```

Optimistic Update:

- Update UI immediately
- Revert if API call fails
- Show success toast on confirmation

6.2.5 Update Speaker Bucket

Endpoint: `PUT /api/v1/speakers/{id}/bucket`

UI Trigger: Approve bucket recommendation

Request:

```
async function updateBucket(speakerId, newBucket, reason) {
  const { data } = await apiClient.put(`/speakers/${speakerId}/bucket`, {
    bucket: newBucket,
    reason: reason
  });
  return data;
}
```

UI Updates:

- Update bucket badge immediately
- Show success animation
- Refresh evaluation status

6.3 Draft Management APIs

6.3.1 Ingest Drafts

Endpoint: `POST /api/v1/drafts/ingest`

UI Trigger: "Ingest Drafts" button (automatic after speaker creation)

Request:

```
async function ingestDrafts(speakerId, limit = 10) {
  const { data } = await apiClient.post('/drafts/ingest', null, {
    params: { speaker_id: speakerId, limit }
  });
  return data;
}
```

UI Updates:

- Show progress indicator
- Display success message with count
- Refresh drafts list

6.3.2 Get Speaker Drafts

Endpoint: `GET /api/v1/drafts/speaker/{speakerId}`

UI Trigger: Navigate to speaker's Drafts tab

Request:

```
async function getSpeakerDrafts(speakerId, filters) {
  const { data } = await apiClient.get(`/drafts/speaker/${speakerId}`, {
    params: {
      skip: filters.skip || 0,
      limit: filters.limit || 100,
      draft_type: filters.draftType, // Optional: AD, LD, IFN
      is_processed: filters.isProcessed // Optional: true/false
    }
  });
  return data;
}
```

UI Updates:

- Populate drafts table
- Apply filters
- Show processing status badges

6.4 AI Workflow APIs

6.4.1 Generate DFN

Endpoint: `POST /api/v1/workflow/generate-dfn`

UI Trigger: "Generate DFN" button

Request:

```
async function generateDFN(speakerId, prompt, options) {
  const { data } = await apiClient.post('/workflow/generate-dfn', {
    speakerId,
    prompt: prompt || 'Generate a professional medical note',
    options: {
      useLangGraph: options?.useLangGraph ?? true,
      maxTokens: options?.maxTokens ?? 2000
    }
  });
  return data;
}
```

Loading State (Important!):

- Show modal with loading spinner
- Display message: "Generating DFN with AI..."
- Show progress steps:
    1. Validating speaker ✓
    2. Retrieving context ✓
    3. Generating with AI... ⏳
- Timeout: 60 seconds (AI operations can be slow)

Success Handling:

- Display generated DFN text
- Show quality metrics (confidence, vectors used, time)
- Enable actions: Accept, Regenerate, Edit

Error Handling:

- Show error message with retry option
- Log error for debugging
- Provide "Contact Support" link

6.5 Evaluation APIs

6.5.1 List Evaluations

Endpoint: `GET /api/v1/evaluations`

UI Trigger: Navigate to evaluations dashboard

Request:

```
async function getEvaluations(filters) {
  const { data } = await apiClient.get('/evaluations', {
    params: {
      skip: filters.skip || 0,
      limit: filters.limit || 20
    }
  });
  return data;
}
```

6.5.2 Get Evaluation Details

Endpoint: `GET /api/v1/evaluations/{id}`

UI Trigger: Click evaluation row

Request:

```
async function getEvaluation(evaluationId) {
  const { data } = await apiClient.get(`/evaluations/${evaluationId}`);
  return data;
}
```

Response Structure:

```
{
  "id": "eval-123",
  "speaker_id": "speaker-456",
  "dfn_id": "dfn-789",
  "ifn_id": "ifn-012",
  "metrics": {
    "ser": 0.15,
    "wer": 0.08,
    "similarity": 0.92,
    "quality_score": 87.5
  },
  "recommended_bucket": "EXCELLENT",
  "status": "PENDING"
}
```

UI Updates:

- Display DFN vs IFN comparison
- Show metrics with visual indicators
- Display bucket recommendation
- Enable approval actions

6.6 Dashboard APIs

6.6.1 Get Dashboard Metrics

Endpoint: `GET /api/v1/dashboard/metrics`

UI Trigger: Load dashboard page

Request:

```
async function getDashboardMetrics() {
  const { data } = await apiClient.get('/dashboard/metrics');
  return data;
}
```

Response Structure:

```
{
  "speakers": {
    "total": 500,
    "byBucket": {
      "NO_TOUCH": 50,
      "LOW_TOUCH": 200,
      "MEDIUM_TOUCH": 150,
      "HIGH_TOUCH": 75,
      "VERY_HIGH_TOUCH": 25
    }
  },
  "drafts": {
    "total": 5000,
    "byType": {...}
  },
  "evaluations": {
    "total": 1200,
    "completed": 1000
  },
  "summary": {
    "healthPercentage": 100
  }
}
```

UI Updates:

- Populate KPI cards
- Render charts (bucket distribution, trends)
- Update activity feed
- Show system health status

---

7. Design-to-API Mapping

7.1 Screen-to-Endpoint Mapping

Screen	Primary Endpoints	Secondary Endpoints
Login	`POST /auth/login`	`POST /auth/refresh`
Dashboard	`GET /dashboard/metrics`	`GET /health/services`
Speaker List	`GET /speakers`	`GET /speakers/statistics`
Speaker Profile	`GET /speakers/{id}`	`GET /drafts/speaker/{id}`, `GET /evaluations`
Add Speaker	`POST /speakers`	-
Bulk Import	`POST /speakers` (loop)	-
Drafts Tab	`GET /drafts/speaker/{id}`	`GET /drafts/{id}`
Generate DFN	`POST /workflow/generate-dfn`	`GET /speakers/{id}`
Evaluations	`GET /evaluations`	`GET /evaluations/{id}`
Evaluation Detail	`GET /evaluations/{id}`	`PUT /speakers/{id}/bucket`

7.2 User Action-to-API Mapping

User Action	API Call	UI Update

User Action	API Call	UI Update
Click "Add Speaker"	`POST /speakers`	Show success toast, redirect to profile
Search speakers	`GET /speakers?search={query}`	Update table with filtered results
Filter by bucket	`GET /speakers?bucket={bucket}`	Update table with filtered results
Click speaker name	`GET /speakers/{id}`	Navigate to profile page
Edit speaker	`PATCH /speakers/{id}`	Update profile display
Generate DFN	`POST /workflow/generate-dfn`	Show loading modal, then result
Approve bucket change	`PUT /speakers/{id}/bucket`	Update bucket badge, show success
View draft details	`GET /drafts/{id}`	Open detail modal/panel
Ingest drafts	`POST /drafts/ingest`	Show progress, update count

7.3 Component-to-Data Mapping

Component	Data Source	API Endpoint
SpeakerCard	Speaker object	`GET /speakers`
BucketBadge	speaker.bucket	-
DraftComparison	draft.original_text, draft.corrected_text	`GET /drafts/{id}`
MetricsPanel	evaluation.metrics	`GET /evaluations/{id}`
DashboardKPIs	dashboard metrics	`GET /dashboard/metrics`
SpeakerForm	Form state	`POST /speakers` (on submit)
EvaluationList	Evaluations array	`GET /evaluations`

8. Technical Constraints & Requirements

8.1 Performance Requirements

8.1.1 Response Time Expectations

Operation	Expected Time	UI Handling
List Speakers	< 500ms	Skeleton loader
Get Speaker Details	< 300ms	Skeleton loader
Create Speaker	< 1s	Button loading state
Generate DFN	10-30s	Modal with progress

Operation	Expected Time	UI Handling
List Drafts	< 1s	Skeleton loader
Dashboard Load	< 2s	Progressive loading

8.1.2 Optimization Strategies

Pagination:

- Default: 20 items per page
- Max: 100 items per page
- Use `skip` and `limit` parameters

Caching:

- Cache speaker list for 5 minutes
- Cache dashboard metrics for 2 minutes
- Invalidate cache on create/update/delete

Lazy Loading:

- Load tabs on demand (not all at once)
- Infinite scroll for long lists (optional)
- Defer loading of charts until visible

Debouncing:

- Search input: 300ms debounce
- Filter changes: 200ms debounce

8.2 Browser Compatibility

Supported Browsers:

- Chrome 90+ (recommended)
- Firefox 88+
- Safari 14+
- Edge 90+

Not Supported:

- Internet Explorer (any version)

Polyfills Required:

- Fetch API (for older browsers)
- Promise (for older browsers)

8.3 Security Requirements

8.3.1 Token Storage

Recommended:

- Use `httpOnly` cookies for refresh tokens (if backend supports)
- Use `localStorage` for access tokens (with XSS protection)

Not Recommended:

- Storing tokens in `sessionStorage` (lost on tab close)
- Storing tokens in plain cookies (CSRF risk)

8.3.2 Input Validation

Client-Side:

- Validate all form inputs before submission
- Sanitize user input to prevent XSS
- Use parameterized queries (handled by API client)

Server-Side:

- Backend validates all inputs (don't rely on client-side only)
- Frontend should match backend validation rules

8.3.3 HTTPS Only

- All API calls must use HTTPS
- No mixed content (HTTP resources on HTTPS page)
- Enforce HTTPS in production

8.4 Error Handling Standards

8.4.1 HTTP Status Codes

Code	Meaning	UI Action
200	Success	Show success state
201	Created	Show success toast, redirect
204	No Content	Show success toast
400	Bad Request	Show validation errors inline
401	Unauthorized	Refresh token or redirect to login
403	Forbidden	Show "Access Denied" message
404	Not Found	Show "Not Found" message
409	Conflict	Show conflict message (e.g., duplicate)
429	Rate Limit	Show "Too many requests" message, retry after delay
500	Server Error	Show generic error, offer retry

Code	Meaning	UI Action
503	Service Unavailable	Show "Service temporarily unavailable"

8.4.2 Error Message Display

Inline Errors (Forms):

- Show below field
- Red text with error icon
- Clear on field change

Toast Notifications:

- Success: Green, auto-dismiss (3s)
- Error: Red, manual dismiss
- Warning: Orange, auto-dismiss (5s)
- Info: Blue, auto-dismiss (3s)

Modal Errors:

- For critical errors
- Require user acknowledgment
- Provide retry or cancel options

8.5 Accessibility Requirements

8.5.1 WCAG 2.1 Level AA Compliance

Keyboard Navigation:

- All interactive elements accessible via keyboard
- Logical tab order
- Visible focus indicators
- Escape key closes modals

Screen Reader Support:

- Semantic HTML (headings, lists, tables)
- ARIA labels for icons and buttons
- ARIA live regions for dynamic content
- Alt text for images

Color Contrast:

- Text: 4.5:1 minimum
- Large text (18pt+): 3:1 minimum
- UI components: 3:1 minimum

Responsive Text:

- Support text zoom up to 200%

- No horizontal scrolling at 200% zoom

8.5.2 Accessibility Testing

Tools:

- axe DevTools (browser extension)
- WAVE (web accessibility evaluation tool)
- Lighthouse (Chrome DevTools)

Manual Testing:

- Keyboard-only navigation
- Screen reader testing (NVDA, JAWS, VoiceOver)
- Color blindness simulation

---

9. Accessibility & Responsive Design

9.1 Responsive Breakpoints

Mobile First Approach:

```css
/* Mobile (default) */
.container {
  padding: 16px;
}

/* Tablet (768px+) */
@media (min-width: 768px) {
  .container {
    padding: 24px;
  }
}

/* Desktop (1024px+) */
@media (min-width: 1024px) {
  .container {
    padding: 32px;
  }
}

/* Large Desktop (1440px+) */
@media (min-width: 1440px) {
  .container {
    padding: 48px;
  }
}
```

9.2 Mobile Considerations

Navigation:

- Hamburger menu for mobile
- Bottom navigation for key actions
- Swipe gestures for tabs

Forms:

- Full-width inputs on mobile
- Large touch targets (44x44px minimum)
- Native input types (email, tel, date)

Tables:

- Horizontal scroll on mobile
- Or convert to card layout
- Sticky headers

Modals:

- Full-screen on mobile
- Slide-up animation
- Easy dismiss (swipe down or X button)

9.3 Touch Interactions

Minimum Touch Target Size:

- 44x44px (iOS guideline)
- 48x48px (Android guideline)
- Use 48x48px for consistency

Gestures:

- Swipe to delete (lists)
- Pull to refresh (lists)
- Pinch to zoom (images, charts)
- Long press for context menu

---

10. Appendices

10.1 Glossary

Term	Definition
AD	ASR Draft - Raw automatic speech recognition output
BSA	Batch Speaker Addition - Bulk onboarding of multiple speakers
DFN	DraftGenie Final Note - AI-improved final note
IFN	InstaNote Final Note - Human-edited final note from InstaNote

Term	Definition
LD	LLM Draft - LLM-generated draft from InstaNote
RAG	Retrieval-Augmented Generation - AI technique combining retrieval and generation
SER	Sentence Edit Rate - Metric for sentence-level changes
SSA	Single Speaker Addition - Manual onboarding of one speaker
WER	Word Error Rate - Metric for word-level accuracy

10.2 Bucket Definitions

Bucket	Quality Range	Description
NO_TOUCH	95-100%	Minimal corrections needed, high accuracy
LOW_TOUCH	85-94%	Few corrections needed, good accuracy
MEDIUM_TOUCH	70-84%	Moderate corrections needed, acceptable accuracy
HIGH_TOUCH	50-69%	Many corrections needed, below average accuracy
VERY_HIGH_TOUCH	0-49%	Extensive corrections needed, low accuracy

10.3 API Resources

Production API:

- Base URL: `https://api-gateway.gentleforest-322351b3.southindia.azurecontainerapps.io/api/v1`
- Swagger UI: `https://api-gateway.gentleforest-322351b3.southindia.azurecontainerapps.io/api/docs`
- Health Check: `https://api-gateway.gentleforest-322351b3.southindia.azurecontainerapps.io/api/v1/health`

Documentation:

- Complete API Docs: `docs/FRONTEND_API_DOCUMENTATION.md`
- Quick Reference: `docs/API_QUICK_REFERENCE.md`
- Postman Collection: `docs/DraftGenie_API.postman_collection.json`

10.4 Design Assets

Recommended Tools:

- Figma (design mockups)
- Storybook (component library)
- Chromatic (visual testing)

Icon Library:

- Material Icons (recommended)

- Font Awesome (alternative)
- Heroicons (alternative)

Chart Library:

- Chart.js (recommended)
- Recharts (React)
- ApexCharts (alternative)

10.5 Sample Workflows

Workflow 1: Complete Speaker Onboarding

```
1. Admin clicks "Add Speaker"
2. Fills form: Name, Email, Bucket, External ID
3. Clicks "Create Speaker"
4. API: POST /speakers → 201 Created
5. Success toast appears
6. Redirect to speaker profile
7. Background: POST /drafts/ingest (automatic)
8. Drafts tab shows "Ingesting..." status
9. After 10-30s, drafts appear
10. Admin can now generate DFN
```

Workflow 2: Generate and Evaluate DFN

```
1. Admin navigates to speaker profile
2. Clicks "Generate DFN" button
3. Modal appears with prompt field
4. Admin enters prompt (optional)
5. Clicks "Generate"
6. API: POST /workflow/generate-dfn
7. Loading modal shows progress (10-30s)
8. DFN appears with quality metrics
9. Admin reviews and clicks "Accept"
10. DFN saved to database
11. Evaluation triggered automatically
12. Evaluation appears in Evaluations tab
13. Admin reviews metrics
14. Approves bucket change if recommended
15. API: PUT /speakers/{id}/bucket
16. Bucket badge updates
```

10.6 Testing Checklist

Functional Testing:

- [ ] Login/logout works

- [ ] Create speaker (SSA)
- [ ] Bulk import (BSA)
- [ ] View speaker list with filters
- [ ] View speaker profile
- [ ] Edit speaker details
- [ ] Generate DFN
- [ ] View drafts
- [ ] View evaluations
- [ ] Approve bucket change
- [ ] Dashboard loads with metrics

UI/UX Testing:

- [ ] All buttons have hover states
- [ ] Loading states appear for async operations
- [ ] Error messages are clear and actionable
- [ ] Success messages appear and auto-dismiss
- [ ] Forms validate in real-time
- [ ] Pagination works correctly
- [ ] Filters apply correctly
- [ ] Search works as expected

Responsive Testing:

- [ ] Mobile (320px-767px)
- [ ] Tablet (768px-1023px)
- [ ] Desktop (1024px+)
- [ ] Touch interactions work on mobile
- [ ] Navigation adapts to screen size

Accessibility Testing:

- [ ] Keyboard navigation works
- [ ] Screen reader announces content
- [ ] Color contrast meets WCAG AA
- [ ] Focus indicators visible
- [ ] ARIA labels present

Performance Testing:

- [ ] Page load < 3s
- [ ] API calls < 2s (except DFN generation)
- [ ] No memory leaks
- [ ] Smooth animations (60fps)

10.7 Support & Resources

For Questions:

1. Check Swagger UI: `/api/docs`

2. Review API documentation: `docs/FRONTEND_API_DOCUMENTATION.md`
3. Test with Postman collection
4. Contact backend team

Useful Links:

- System Architecture: `docs/system_architecture_and_implementation_plan.md`
- Frontend Handoff: `docs/FRONTEND_TEAM_HANDOFF.md`
- Deployment Guide: `docs/deployment/azure-deployment-guide.md`

Document Version: 1.0 **Last Updated:** 2025-10-16 **Status:** ✅ Complete **Next Review:** 2025-11-16

Prepared by: DraftGenie Backend Team **For:** Frontend Development Team