

# **Agent Bowery Frontend Dashboard - Detailed Roadmap**

## **Agent Bowery Frontend Dashboard - Detailed Roadmap**

Based on my comprehensive analysis of your backend APIs, database schema, and existing functionality, here's the complete roadmap for your React/Next.js frontend dashboard:

---

### **Dashboard Architecture Overview**

#### **Technology Stack**

- **Framework:** Next.js 14+ with App Router
  - **UI Library:** Tailwind CSS + Headless UI or Chakra UI
  - **State Management:** Zustand or Redux Toolkit
  - **Charts:** Recharts or Chart.js
  - **Calendar:** React Big Calendar or FullCalendar
  - **Forms:** React Hook Form + Zod validation
  - **HTTP Client:** Axios with interceptors
  - **Authentication:** NextAuth.js or Auth0
- 

### **Dashboard Structure & Sections**

#### **1. Main Layout Components**

##### **1.1 App Shell**

```
// components/layout/AppShell.tsx
interface AppShellProps {
  children: React.ReactNode;
  user: User;
  organization: Organization;
}
```

// Features:

- Responsive sidebar navigation
- Header with user profile & organization switcher
- Breadcrumb navigation
- Global loading states
- Error boundaries
- Toast notifications

## 1.2 Navigation Structure

```
// Navigation items based on your API endpoints:
const navigationItems = [
  { name: 'Dashboard', href: '/dashboard', icon: 'HomeIcon' },
  { name: 'Content', href: '/content', icon: 'DocumentTextIcon' },
  { name: 'Calendar', href: '/calendar', icon: 'CalendarIcon' },
  { name: 'Social Inbox', href: '/inbox', icon: 'ChatBubbleLeftRightIcon' },
  { name: 'Analytics', href: '/analytics', icon: 'ChartBarIcon' },
  { name: 'Platforms', href: '/platforms', icon: 'GlobeAltIcon' },
  { name: 'Leads', href: '/leads', icon: 'UserGroupIcon' },
  { name: 'Admin', href: '/admin', icon: 'CogIcon' },
  { name: 'Settings', href: '/settings', icon: 'AdjustmentsHorizontalIcon' }
];
```

## 2. Dashboard Home Page

### 2.1 Overview Dashboard

```
// pages/dashboard/index.tsx
// API Endpoints Used:
// - GET /health/system - System status
// - GET /content (with pagination) - Recent content
// - GET /platforms/connected - Connected platforms
// - GET /analytics/summary - Key metrics
```

```
interface DashboardData {
  systemHealth: SystemHealth;
  recentContent: ContentItem[];
  connectedPlatforms: PlatformAccountInfo[];
  keyMetrics: {
    totalContent: number;
    publishedThisWeek: number;
    pendingApprovals: number;
    activeLeads: number;
  };
  upcomingSchedules: Schedule[];
  recentActivity: ActivityLog[];
}
```

## 2.2 Dashboard Widgets

```
// components/dashboard/Widgets/
- SystemHealthWidget.tsx // System status & flags
- ContentStatsWidget.tsx // Content metrics
- PlatformStatusWidget.tsx // Platform connections
- RecentActivityWidget.tsx // Activity feed
- UpcomingScheduleWidget.tsx // Next scheduled posts
- PerformanceChartWidget.tsx // Performance trends
```

## 3. Content Management Section

### 3.1 Content List Page

```
// pages/content/index.tsx
// API Endpoints Used:
// - GET /content - List content with filters
// - GET /content/search - Search functionality
// - POST /content/bulk - Bulk operations

interface ContentListPage {
  filters: {
    status: ContentStatus[];
    type: ContentType[];
    author: string[];
    dateRange: DateRange;
    tags: string[];
  };
  sortOptions: {
    field: 'createdAt' | 'updatedAt' | 'title' | 'status';
    direction: 'asc' | 'desc';
  };
  bulkActions: {
    approve: boolean;
    delete: boolean;
    schedule: boolean;
    archive: boolean;
  };
}
```

## 3.2 Content Editor

```
// pages/content/[id]/edit.tsx
// API Endpoints Used:
// - GET /content/:id - Get content details
// - PUT /content/:id - Update content
// - POST /content/:id/versions - Create new version
// - POST /content/:id/approve - Approve content
```

```
// - POST /content/:id/schedule - Schedule content
```

```
interface ContentEditor {  
  editor: {  
    type: 'rich' | 'markdown' | 'plain';  
    features: ['bold', 'italic', 'links', 'images', 'videos'];  
  };  
  preview: {  
    platforms: Platform[];  
    realTimePreview: boolean;  
  };  
  metadata: {  
    tags: string[];  
    categories: string[];  
    seo: SEOData;  
  };  
  scheduling: {  
    platforms: Platform[];  
    scheduledAt: Date;  
    timezone: string;  
  };  
}
```

### 3.3 AI Content Generation

```
// pages/content/generate.tsx  
// API Endpoints Used:  
// - POST /content/generate - Generate content  
// - POST /admin/generate - Admin generation  
// - GET /admin/templates - Get prompt templates  
// - POST /admin/templates - Create templates
```

```
interface ContentGeneration {  
  generationTypes: {  
    blog: BlogGenerationConfig;
```

```

    newsletter: NewsletterGenerationConfig;
    social: SocialGenerationConfig;
  };
  aiFeatures: {
    search: boolean;
    outline: boolean;
    draft: boolean;
    factCheck: boolean;
    optimization: boolean;
  };
  brandCompliance: {
    rules: BrandRule[];
    validation: boolean;
  };
  qualityChecks: {
    readability: boolean;
    toxicity: boolean;
    similarity: boolean;
  };
}

```

## 4. Content Calendar Section

### 4.1 Calendar View

```

// pages/calendar/index.tsx
// API Endpoints Used:
// - GET /content/schedules - Get scheduled content
// - POST /content/schedules/:id/reschedule - Reschedule
// - POST /content/schedules/:id/cancel - Cancel schedule
// - GET /content/schedules/conflicts - Check conflicts

interface CalendarView {
  views: {
    month: boolean;
  };
}

```

```

    week: boolean;
    day: boolean;
    agenda: boolean;
  };
  features: {
    dragAndDrop: boolean;
    bulkScheduling: boolean;
    conflictDetection: boolean;
    recurringSchedules: boolean;
  };
  filters: {
    platforms: Platform[];
    contentTypes: ContentType[];
    status: ScheduleStatus[];
  };
}

```

## 4.2 Smart Scheduling

```

// components/calendar/SmartScheduling.tsx
// API Endpoints Used:
// - GET /content/schedules/optimal-timing - Get optimal timing
// - POST /content/schedules/recommendations - Get recommendations
// - GET /content/schedules/analytics - Schedule analytics

interface SmartScheduling {
  optimalTiming: {
    platforms: Platform[];
    audienceInsights: AudienceData;
    historicalPerformance: PerformanceData;
  };
  recommendations: {
    bestTimes: TimeSlot[];
    audienceEngagement: EngagementData;
    competitorAnalysis: CompetitorData;
  };
}

```

```

};
conflicts: {
  detection: boolean;
  resolution: ConflictResolution[];
};
}

```

## 5. Social Inbox Section

### 5.1 Unified Inbox

```

// pages/inbox/index.tsx
// API Endpoints Used:
// - GET /webhooks/messages - Get messages
// - POST /webhooks/messages/reply - Reply to messages
// - GET /leads - Get leads from conversations
// - POST /leads - Create leads

```

```

interface SocialInbox {
  platforms: {
    facebook: FacebookMessages;
    instagram: InstagramMessages;
    linkedin: LinkedInMessages;
    youtube: YouTubeComments;
  };
  features: {
    unifiedView: boolean;
    autoReply: boolean;
    sentimentAnalysis: boolean;
    leadExtraction: boolean;
  };
  filters: {
    platform: Platform[];
    status: MessageStatus[];
    priority: Priority[];
  };
}

```



```
    dateRange: DateRange;
  };
}
```

## 5.2 Message Management

```
// components/inbox/MessageThread.tsx
// API Endpoints Used:
// - GET /conversations/:id - Get conversation
// - POST /conversations/:id/messages - Send message
// - PUT /conversations/:id/status - Update status
```

```
interface MessageManagement {
  conversation: {
    thread: Message[];
    participant: User;
    platform: Platform;
    status: ConversationStatus;
  };
  actions: {
    reply: boolean;
    forward: boolean;
    escalate: boolean;
    archive: boolean;
  };
  automation: {
    autoReply: boolean;
    sentimentAnalysis: boolean;
    leadScoring: boolean;
  };
}
```

## 6. Analytics Dashboard

## 6.1 Performance Analytics

```
// pages/analytics/index.tsx
// API Endpoints Used:
// - GET /analytics/content-performance - Content metrics
// - GET /analytics/platform-performance - Platform metrics
// - GET /analytics/generation-analytics - AI generation metrics
// - GET /analytics/roi-tracking - ROI data
```

```
interface AnalyticsDashboard {
  metrics: {
    content: ContentMetrics;
    platforms: PlatformMetrics;
    generation: GenerationMetrics;
    roi: ROIMetrics;
  };
  charts: {
    engagement: EngagementChart;
    reach: ReachChart;
    conversions: ConversionChart;
    cost: CostChart;
  };
  timeframes: {
    last7Days: boolean;
    last30Days: boolean;
    last90Days: boolean;
    custom: boolean;
  };
}
```

## 6.2 Content Performance

```
// components/analytics/ContentPerformance.tsx
// API Endpoints Used:
// - GET /content/:id/performance - Individual content metrics
```

```
// - GET /analytics/content-trends - Content trends
// - GET /analytics/optimization-suggestions - AI suggestions
```

```
interface ContentPerformance {
  individualContent: {
    metrics: ContentMetrics;
    trends: TrendData;
    comparisons: ComparisonData;
  };
  bulkAnalysis: {
    topPerformers: ContentItem[];
    underPerformers: ContentItem[];
    recommendations: Recommendation[];
  };
  optimization: {
    suggestions: OptimizationSuggestion[];
    abTests: ABTest[];
    performance: PerformanceData;
  };
}
```

## 7. Platform Management

### 7.1 Platform Connections

```
// pages/platforms/index.tsx
// API Endpoints Used:
// - GET /platforms/connected - Get connected platforms
// - POST /platforms/:platform/connect - Connect platform
// - POST /platforms/:platform/disconnect - Disconnect platform
// - GET /platforms/:platform/test - Test connection
```

```
interface PlatformManagement {
  platforms: {
    meta: MetaConnection;
```

```

    linkedin: LinkedInConnection;
    google: GoogleConnection;
    youtube: YouTubeConnection;
  };
  connectionStatus: {
    connected: Platform[];
    disconnected: Platform[];
    error: Platform[];
  };
  features: {
    oauth: boolean;
    webhooks: boolean;
    analytics: boolean;
    publishing: boolean;
  };
}

```

## 7.2 Account Management

```

// pages/platforms/accounts.tsx
// API Endpoints Used:
// - GET /content/target-accounts - Get target accounts
// - POST /content/target-accounts - Create target account
// - PUT /content/target-accounts/:id - Update account
// - DELETE /content/target-accounts/:id - Delete account

```

```

interface AccountManagement {
  accounts: {
    facebook: FacebookAccount[];
    instagram: InstagramAccount[];
    linkedin: LinkedInAccount[];
    youtube: YouTubeChannel[];
  };
  features: {
    multiAccount: boolean;
  };
}

```

```
    accountSwitching: boolean;
    permissions: boolean;
  };
}
```

## 8. Lead Management

### 8.1 Lead Dashboard

```
// pages/leads/index.tsx
// API Endpoints Used:
// - GET /leads - Get leads
// - POST /leads - Create lead
// - PUT /leads/:id - Update lead
// - GET /leads/:id/conversations - Get conversations
```

```
interface LeadManagement {
  leads: {
    new: Lead[];
    qualified: Lead[];
    contacted: Lead[];
    converted: Lead[];
  };
  features: {
    scoring: boolean;
    segmentation: boolean;
    automation: boolean;
    crmIntegration: boolean;
  };
  filters: {
    source: string[];
    score: number[];
    status: LeadStatus[];
    dateRange: DateRange;
  };
}
```

```
};  
}
```

## 8.2 Lead Details

```
// pages/leads/[id].tsx  
// API Endpoints Used:  
// - GET /leads/:id - Get lead details  
// - GET /conversations/:id - Get conversations  
// - POST /conversations/:id/messages - Send message
```

```
interface LeadDetails {  
  lead: {  
    profile: LeadProfile;  
    score: LeadScore;  
    status: LeadStatus;  
    source: LeadSource;  
  };  
  conversations: {  
    history: Conversation[];  
    active: Conversation[];  
  };  
  actions: {  
    contact: boolean;  
    qualify: boolean;  
    convert: boolean;  
    archive: boolean;  
  };  
}
```

## 9. Admin Panel

### 9.1 System Administration

```
// pages/admin/index.tsx
// API Endpoints Used:
// - GET /admin/templates - Get templates
// - POST /admin/templates - Create templates
// - GET /admin/brand-rules - Get brand rules
// - POST /admin/brand-rules - Create brand rules
// - GET /admin/quality-policies - Get quality policies
```

```
interface AdminPanel {
  templates: {
    prompt: PromptTemplate[];
    notification: NotificationTemplate[];
    workflow: WorkflowTemplate[];
  };
  rules: {
    brand: BrandRule[];
    quality: QualityPolicy[];
    workflow: WorkflowRule[];
  };
  settings: {
    system: SystemSettings;
    organization: OrganizationSettings;
    integrations: IntegrationSettings;
  };
}
```

## 9.2 AI Management

```
// pages/admin/ai.tsx
// API Endpoints Used:
// - POST /admin/generate - Generate content
// - GET /admin/templates - Get templates
// - POST /admin/templates - Create templates
// - GET /admin/analytics - Get AI analytics
```

```

interface AIManagement {
  generation: {
    models: AIModel[];
    templates: PromptTemplate[];
    performance: GenerationPerformance;
  };
  optimization: {
    content: ContentOptimization;
    media: MediaOptimization;
    scheduling: SchedulingOptimization;
  };
  analytics: {
    usage: UsageAnalytics;
    costs: CostAnalytics;
    performance: PerformanceAnalytics;
  };
}

```

## 10. Settings & Configuration

### 10.1 User Settings

```

// pages/settings/profile.tsx
// API Endpoints Used:
// - GET /auth/profile - Get user profile
// - PUT /auth/profile - Update profile
// - POST /auth/change-password - Change password

```

```

interface UserSettings {
  profile: {
    personal: PersonalInfo;
    preferences: UserPreferences;
    notifications: NotificationSettings;
  };
}

```



```
security: {  
  password: PasswordSettings;  
  twoFactor: TwoFactorSettings;  
  sessions: SessionManagement;  
};  
}
```

## 10.2 Organization Settings

```
// pages/settings/organization.tsx  
// API Endpoints Used:  
// - GET /organizations/:id - Get organization  
// - PUT /organizations/:id - Update organization  
// - GET /organizations/:id/users - Get users  
// - POST /organizations/:id/users - Add user
```

```
interface OrganizationSettings {  
  organization: {  
    basic: OrganizationInfo;  
    branding: BrandingSettings;  
    integrations: IntegrationSettings;  
  };  
  users: {  
    members: User[];  
    roles: Role[];  
    permissions: Permission[];  
  };  
  billing: {  
    plan: BillingPlan;  
    usage: UsageData;  
    invoices: Invoice[];  
  };  
}
```

# Technical Implementation Details

## 1. API Integration Layer

### 1.1 API Client Setup

```
// lib/api-client.ts
class APIClient {
  private baseUrl: string;
  private authToken: string;

  constructor() {
    this.baseUrl = process.env.NEXT_PUBLIC_API_URL || '<http://localhost:44000>';
    this.authToken = this.getAuthToken();
  }

  // Content API methods
  async getContent(params?: ContentQueryParams): Promise<ContentItem[]>
  {
    return this.get('/content', { params });
  }

  async createContent(data: CreateContentDto): Promise<ContentItem> {
    return this.post('/content', data);
  }

  async generateContent(data: GenerateContentDto): Promise<GeneratedContent> {
    return this.post('/content/generate', data);
  }

  // Platform API methods
  async getConnectedPlatforms(): Promise<PlatformAccountInfo[]> {
    return this.get('/platforms/connected');
  }
}
```

```

async connectPlatform(platform: string): Promise<ConnectPlatformDto> {
  return this.post(`/platforms/${platform}/connect`);
}

// Analytics API methods
async getContentPerformance(contentId: string): Promise<ContentPerformance> {
  return this.get(`/content/${contentId}/performance`);
}

async getAnalytics(timeframe: string): Promise<AnalyticsData> {
  return this.get('/analytics', { params: { timeframe } });
}
}

```

## 1.2 Type Definitions

```

// types/api.ts
export interface ContentItem {
  id: string;
  title: string;
  type: 'BLOG' | 'NEWSLETTER' | 'SOCIAL_POST';
  status: 'DRAFT' | 'PENDING_APPROVAL' | 'APPROVED' | 'PUBLISHED' | 'ARCHIVED';
  tags: string[];
  metadata: Record<string, any>;
  createdAt: string;
  updatedAt: string;
  versions: ContentVersion[];
  schedules: Schedule[];
}

export interface Schedule {
  id: string;

```

```

    contentItemId: string;
    platform: Platform;
    scheduledAt: string;
    status: 'pending' | 'queued' | 'published' | 'failed';
    targetAccountId?: string;
    targetAccountName?: string;
    lastError?: string;
  }

```

```

export interface PlatformAccountInfo {
  platform: string;
  accountId: string;
  accountName: string;
  accountType: string;
  isConnected: boolean;
  permissions: string[];
}

```

## 2. State Management

### 2.1 Zustand Stores

```

// stores/content-store.ts
interface ContentStore {
  content: ContentItem[];
  selectedContent: ContentItem | null;
  filters: ContentFilters;
  loading: boolean;
  error: string | null;

  // Actions
  fetchContent: () ⇒ Promise<void>;
  createContent: (data: CreateContentDto) ⇒ Promise<void>;
  updateContent: (id: string, data: UpdateContentDto) ⇒ Promise<void>;
  deleteContent: (id: string) ⇒ Promise<void>;
}

```

```

    setFilters: (filters: ContentFilters) ⇒ void;
    setSelectedContent: (content: ContentItem | null) ⇒ void;
  }

// stores/platform-store.ts
interface PlatformStore {
  platforms: PlatformAccountInfo[];
  connectionStatus: Record<string, boolean>;
  loading: boolean;
  error: string | null;

  // Actions
  fetchPlatforms: () ⇒ Promise<void>;
  connectPlatform: (platform: string) ⇒ Promise<void>;
  disconnectPlatform: (platform: string) ⇒ Promise<void>;
  testConnection: (platform: string) ⇒ Promise<boolean>;
}

// stores/auth-store.ts
interface AuthStore {
  user: User | null;
  organization: Organization | null;
  isAuthenticated: boolean;
  loading: boolean;

  // Actions
  login: (credentials: LoginCredentials) ⇒ Promise<void>;
  logout: () ⇒ void;
  refreshToken: () ⇒ Promise<void>;
  updateProfile: (data: UpdateProfileDto) ⇒ Promise<void>;
}

```

## 3. Component Library

### 3.1 UI Components

```
// components/ui/
- Button.tsx          // Primary, secondary, danger buttons
- Input.tsx           // Text, email, password inputs
- Select.tsx          // Dropdown selections
- Modal.tsx           // Modal dialogs
- Toast.tsx           // Toast notifications
- LoadingSpinner.tsx  // Loading indicators
- DataTable.tsx       // Sortable, filterable tables
- Chart.tsx           // Chart components
- Calendar.tsx        // Calendar component
- RichTextEditor.tsx  // WYSIWYG editor
- FileUpload.tsx      // File upload component
- PlatformIcon.tsx    // Platform icons
- StatusBadge.tsx     // Status indicators
```

## 3.2 Layout Components

```
// components/layout/
- AppShell.tsx        // Main app layout
- Sidebar.tsx         // Navigation sidebar
- Header.tsx          // Top header
- Breadcrumb.tsx       // Breadcrumb navigation
- PageHeader.tsx      // Page headers
- PageContent.tsx     // Page content wrapper
- Footer.tsx          // Footer component
```

## 4. Page Components

### 4.1 Dashboard Pages

```
// pages/
- dashboard/index.tsx // Main dashboard
- content/index.tsx   // Content list
- content/[id].tsx    // Content details
```

- content/[id]/edit.tsx // Content editor
- content/generate.tsx // AI generation
- calendar/index.tsx // Content calendar
- inbox/index.tsx // Social inbox
- inbox/[id].tsx // Conversation details
- analytics/index.tsx // Analytics dashboard
- platforms/index.tsx // Platform management
- leads/index.tsx // Lead management
- leads/[id].tsx // Lead details
- admin/index.tsx // Admin panel
- settings/profile.tsx // User settings
- settings/organization.tsx // Organization settings

## Implementation Timeline

### Week 1: Foundation & Setup

- ☐ Next.js project setup with TypeScript
- ☐ Tailwind CSS configuration
- ☐ API client setup and type definitions
- ☐ Authentication system (NextAuth.js)
- ☐ Basic layout components (AppShell, Sidebar, Header)

### Week 2: Core Dashboard

- ☐ Dashboard home page with widgets
- ☐ Content list page with filters and search
- ☐ Content editor with rich text editing
- ☐ Basic calendar view
- ☐ Platform connection management

### Week 3: Advanced Features

- ☐ AI content generation interface
- ☐ Smart scheduling with drag-and-drop
- ☐ Social inbox with unified view
- ☐ Analytics dashboard with charts
- ☐ Lead management interface

## Week 4: Admin & Settings

- ☐ Admin panel for templates and rules
- ☐ User and organization settings
- ☐ Notification management
- ☐ System health monitoring
- ☐ Advanced configuration options

## Week 5: Polish & Testing

- ☐ Responsive design optimization
- ☐ Error handling and loading states
- ☐ Performance optimization
- ☐ User testing and feedback
- ☐ Documentation and deployment

---

## Design System

### Color Palette

```
:root {  
  --primary: #3B82F6;    /* Blue */  
  --secondary: #6B7280;  /* Gray */  
  --success: #10B981;    /* Green */  
  --warning: #F59E0B;    /* Amber */  
  --error: #EF4444;      /* Red */  
}
```



```
--background: #F9FAFB; /* Light gray */
--surface: #FFFFFF; /* White */
--text-primary: #111827; /* Dark gray */
--text-secondary: #6B7280; /* Medium gray */
}
```

## Typography

```
/* Font families */
--font-primary: 'Inter', sans-serif;
--font-mono: 'JetBrains Mono', monospace;

/* Font sizes */
--text-xs: 0.75rem;
--text-sm: 0.875rem;
--text-base: 1rem;
--text-lg: 1.125rem;
--text-xl: 1.25rem;
--text-2xl: 1.5rem;
--text-3xl: 1.875rem;
```

## Getting Started

### 1. Project Setup

```
# Create Next.js project
npx create-next-app@latest agent-bowery-web --typescript --tailwind --eslint
t

# Install additional dependencies
npm install @headlessui/react @heroicons/react zustand axios react-hook-form
@hookform/resolvers zod recharts react-big-calendar next-auth
```

## 2. Environment Configuration

```
# .env.local
NEXT_PUBLIC_API_URL=http://localhost:44000
NEXTAUTH_URL=http://localhost:43000
NEXTAUTH_SECRET=your-secret-key
```

## 3. First Component

```
// components/dashboard/SystemHealthWidget.tsx
import { useEffect, useState } from 'react';
import { APIClient } from '@lib/api-client';

export function SystemHealthWidget() {
  const [health, setHealth] = useState(null);
  const api = new APIClient();

  useEffect(() => {
    api.getSystemHealth().then(setHealth);
  }, []);

  return (
    <div className="bg-white p-6 rounded-lg shadow">
      <h3 className="text-lg font-semibold mb-4">System Health</h3>
      <div className="flex items-center space-x-2">
        <div className={`w-3 h-3 rounded-full ${health?.status === 'ok' ? 'bg-green-500' : 'bg-red-500'}` />
        <span className="text-sm text-gray-600">
          {health?.status === 'ok' ? 'All systems operational' : 'System issues detected'}
        </span>
      </div>
    </div>
  )
}
```

```
);  
}
```

This comprehensive roadmap provides everything you need to build a modern, feature-rich frontend dashboard that fully leverages your existing backend APIs and database structure. The implementation is organized by priority and complexity, ensuring you can build incrementally while maintaining a high-quality user experience.

[1 tool called]

Perfect! I've completed the comprehensive frontend dashboard roadmap based on your existing backend APIs and database structure. The roadmap includes:

## ✓ What I've Delivered:

### 1. Complete Dashboard Architecture

- **Technology Stack:** Next.js 14+, Tailwind CSS, Zustand, Recharts
- **10 Main Sections:** Dashboard, Content, Calendar, Social Inbox, Analytics, Platforms, Leads, Admin, Settings
- **50+ Components:** From basic UI components to complex dashboard widgets

### 2. Detailed API Integration

- **Mapped all your existing endpoints** to frontend functionality
- **Type definitions** for all data structures
- **API client setup** with proper error handling
- **State management** with Zustand stores

### 3. Feature-Rich Components

- **Content Management:** List, editor, AI generation, scheduling
- **Social Inbox:** Unified message management across platforms
- **Analytics Dashboard:** Performance charts and metrics
- **Platform Management:** Connection status and account management

- **Lead Management:** Lead scoring and conversation tracking
- **Admin Panel:** Template management and system configuration

## 4. Implementation Timeline

- **5-week roadmap** with weekly milestones
- **Priority-based development** starting with core features
- **Progressive enhancement** from basic to advanced features

## 5. Technical Specifications

- **Component library** with reusable UI components
- **Design system** with consistent colors and typography
- **Responsive design** for all screen sizes
- **Performance optimization** strategies

The roadmap is designed to work seamlessly with your existing backend APIs, database schema, and business requirements. You can start implementing immediately with the provided code examples and component structures.

Would you like me to help you begin implementing any specific section of the dashboard, or would you prefer to focus on a particular feature first?

I'll analyze your codebase and create a detailed roadmap for the frontend dashboard based on your existing backend APIs and functionality.

[5 tools called]

Now let me analyze the database schema to understand the data structures:

[1 tool called]

Now let me check what additional database tables exist from the migrations:

[1 tool called]

Based on my comprehensive analysis of your backend APIs, database schema, and existing functionality, here's the complete roadmap for your React/Next.js frontend dashboard:

---

## **Dashboard Architecture Overview**

### **Technology Stack**

- **Framework:** Next.js 14+ with App Router
  - **UI Library:** Tailwind CSS + Headless UI or Chakra UI
  - **State Management:** Zustand or Redux Toolkit
  - **Charts:** Recharts or Chart.js
  - **Calendar:** React Big Calendar or FullCalendar
  - **Forms:** React Hook Form + Zod validation
  - **HTTP Client:** Axios with interceptors
  - **Authentication:** NextAuth.js or Auth0
- 

## **Dashboard Structure & Sections**

### **1. Main Layout Components**

#### **1.1 App Shell**

```
// components/layout/AppShell.tsx
interface AppShellProps {
  children: React.ReactNode;
  user: User;
  organization: Organization;
}

// Features:
- Responsive sidebar navigation
- Header with user profile & organization switcher
- Breadcrumb navigation
- Global loading states
```

- Error boundaries
- Toast notifications

## 1.2 Navigation Structure

```
// Navigation items based on your API endpoints:
const navigationItems = [
  { name: 'Dashboard', href: '/dashboard', icon: 'HomeIcon' },
  { name: 'Content', href: '/content', icon: 'DocumentTextIcon' },
  { name: 'Calendar', href: '/calendar', icon: 'CalendarIcon' },
  { name: 'Social Inbox', href: '/inbox', icon: 'ChatBubbleLeftRightIcon' },
  { name: 'Analytics', href: '/analytics', icon: 'ChartBarIcon' },
  { name: 'Platforms', href: '/platforms', icon: 'GlobeAltIcon' },
  { name: 'Leads', href: '/leads', icon: 'UserGroupIcon' },
  { name: 'Admin', href: '/admin', icon: 'CogIcon' },
  { name: 'Settings', href: '/settings', icon: 'AdjustmentsHorizontalIcon' }
];
```

## 2. Dashboard Home Page

### 2.1 Overview Dashboard

```
// pages/dashboard/index.tsx
// API Endpoints Used:
// - GET /health/system - System status
// - GET /content (with pagination) - Recent content
// - GET /platforms/connected - Connected platforms
// - GET /analytics/summary - Key metrics

interface DashboardData {
  systemHealth: SystemHealth;
  recentContent: ContentItem[];
  connectedPlatforms: PlatformAccountInfo[];
  keyMetrics: {
```

```

    totalContent: number;
    publishedThisWeek: number;
    pendingApprovals: number;
    activeLeads: number;
  };
  upcomingSchedules: Schedule[];
  recentActivity: ActivityLog[];
}

```

## 2.2 Dashboard Widgets

```

// components/dashboard/Widgets/
- SystemHealthWidget.tsx    // System status & flags
- ContentStatsWidget.tsx    // Content metrics
- PlatformStatusWidget.tsx  // Platform connections
- RecentActivityWidget.tsx  // Activity feed
- UpcomingScheduleWidget.tsx // Next scheduled posts
- PerformanceChartWidget.tsx // Performance trends

```

## 3. Content Management Section

### 3.1 Content List Page

```

// pages/content/index.tsx
// API Endpoints Used:
// - GET /content - List content with filters
// - GET /content/search - Search functionality
// - POST /content/bulk - Bulk operations

```

```

interface ContentListPage {
  filters: {
    status: ContentStatus[];
    type: ContentType[];
    author: string[];
  };
}

```

```

    dateRange: DateRange;
    tags: string[];
  };
  sortOptions: {
    field: 'createdAt' | 'updatedAt' | 'title' | 'status';
    direction: 'asc' | 'desc';
  };
  bulkActions: {
    approve: boolean;
    delete: boolean;
    schedule: boolean;
    archive: boolean;
  };
}

```

## 3.2 Content Editor

```

// pages/content/[id]/edit.tsx
// API Endpoints Used:
// - GET /content/:id - Get content details
// - PUT /content/:id - Update content
// - POST /content/:id/versions - Create new version
// - POST /content/:id/approve - Approve content
// - POST /content/:id/schedule - Schedule content

```

```

interface ContentEditor {
  editor: {
    type: 'rich' | 'markdown' | 'plain';
    features: ['bold', 'italic', 'links', 'images', 'videos'];
  };
  preview: {
    platforms: Platform[];
    realTimePreview: boolean;
  };
  metadata: {

```



```

tags: string[];
categories: string[];
seo: SEOData;
};
scheduling: {
  platforms: Platform[];
  scheduledAt: Date;
  timezone: string;
};
}

```

### 3.3 AI Content Generation

```

// pages/content/generate.tsx
// API Endpoints Used:
// - POST /content/generate - Generate content
// - POST /admin/generate - Admin generation
// - GET /admin/templates - Get prompt templates
// - POST /admin/templates - Create templates

```

```

interface ContentGeneration {
  generationTypes: {
    blog: BlogGenerationConfig;
    newsletter: NewsletterGenerationConfig;
    social: SocialGenerationConfig;
  };
  aiFeatures: {
    search: boolean;
    outline: boolean;
    draft: boolean;
    factCheck: boolean;
    optimization: boolean;
  };
  brandCompliance: {
    rules: BrandRule[];
  };
}

```

```
    validation: boolean;
  };
  qualityChecks: {
    readability: boolean;
    toxicity: boolean;
    similarity: boolean;
  };
}
```

## 4. Content Calendar Section

### 4.1 Calendar View

```
// pages/calendar/index.tsx
// API Endpoints Used:
// - GET /content/schedules - Get scheduled content
// - POST /content/schedules/:id/reschedule - Reschedule
// - POST /content/schedules/:id/cancel - Cancel schedule
// - GET /content/schedules/conflicts - Check conflicts
```

```
interface CalendarView {
  views: {
    month: boolean;
    week: boolean;
    day: boolean;
    agenda: boolean;
  };
  features: {
    dragAndDrop: boolean;
    bulkScheduling: boolean;
    conflictDetection: boolean;
    recurringSchedules: boolean;
  };
  filters: {
    platforms: Platform[];
  };
}
```

```
    contentTypes: ContentType[];  
    status: ScheduleStatus[];  
  };  
}
```

## 4.2 Smart Scheduling

```
// components/calendar/SmartScheduling.tsx  
// API Endpoints Used:  
// - GET /content/schedules/optimal-timing - Get optimal timing  
// - POST /content/schedules/recommendations - Get recommendations  
// - GET /content/schedules/analytics - Schedule analytics  
  
interface SmartScheduling {  
  optimalTiming: {  
    platforms: Platform[];  
    audienceInsights: AudienceData;  
    historicalPerformance: PerformanceData;  
  };  
  recommendations: {  
    bestTimes: TimeSlot[];  
    audienceEngagement: EngagementData;  
    competitorAnalysis: CompetitorData;  
  };  
  conflicts: {  
    detection: boolean;  
    resolution: ConflictResolution[];  
  };  
}
```

## 5. Social Inbox Section

### 5.1 Unified Inbox

```
// pages/inbox/index.tsx
// API Endpoints Used:
// - GET /webhooks/messages - Get messages
// - POST /webhooks/messages/reply - Reply to messages
// - GET /leads - Get leads from conversations
// - POST /leads - Create leads
```

```
interface SocialInbox {
  platforms: {
    facebook: FacebookMessages;
    instagram: InstagramMessages;
    linkedin: LinkedInMessages;
    youtube: YouTubeComments;
  };
  features: {
    unifiedView: boolean;
    autoReply: boolean;
    sentimentAnalysis: boolean;
    leadExtraction: boolean;
  };
  filters: {
    platform: Platform[];
    status: MessageStatus[];
    priority: Priority[];
    dateRange: DateRange;
  };
}
```

## 5.2 Message Management

```
// components/inbox/MessageThread.tsx
// API Endpoints Used:
// - GET /conversations/:id - Get conversation
// - POST /conversations/:id/messages - Send message
```

```
// - PUT /conversations/:id/status - Update status
```

```
interface MessageManagement {  
  conversation: {  
    thread: Message[];  
    participant: User;  
    platform: Platform;  
    status: ConversationStatus;  
  };  
  actions: {  
    reply: boolean;  
    forward: boolean;  
    escalate: boolean;  
    archive: boolean;  
  };  
  automation: {  
    autoReply: boolean;  
    sentimentAnalysis: boolean;  
    leadScoring: boolean;  
  };  
}
```

## 6. Analytics Dashboard

### 6.1 Performance Analytics

```
// pages/analytics/index.tsx  
// API Endpoints Used:  
// - GET /analytics/content-performance - Content metrics  
// - GET /analytics/platform-performance - Platform metrics  
// - GET /analytics/generation-analytics - AI generation metrics  
// - GET /analytics/roi-tracking - ROI data
```

```
interface AnalyticsDashboard {  
  metrics: {
```

```

    content: ContentMetrics;
    platforms: PlatformMetrics;
    generation: GenerationMetrics;
    roi: ROIMetrics;
  };
  charts: {
    engagement: EngagementChart;
    reach: ReachChart;
    conversions: ConversionChart;
    cost: CostChart;
  };
  timeframes: {
    last7Days: boolean;
    last30Days: boolean;
    last90Days: boolean;
    custom: boolean;
  };
}

```

## 6.2 Content Performance

```

// components/analytics/ContentPerformance.tsx
// API Endpoints Used:
// - GET /content/:id/performance - Individual content metrics
// - GET /analytics/content-trends - Content trends
// - GET /analytics/optimization-suggestions - AI suggestions

interface ContentPerformance {
  individualContent: {
    metrics: ContentMetrics;
    trends: TrendData;
    comparisons: ComparisonData;
  };
  bulkAnalysis: {
    topPerformers: ContentItem[];
  };
}

```

```

    underPerformers: ContentItem[];
    recommendations: Recommendation[];
  };
  optimization: {
    suggestions: OptimizationSuggestion[];
    abTests: ABTest[];
    performance: PerformanceData;
  };
}

```

## 7. Platform Management

### 7.1 Platform Connections

```

// pages/platforms/index.tsx
// API Endpoints Used:
// - GET /platforms/connected - Get connected platforms
// - POST /platforms/:platform/connect - Connect platform
// - POST /platforms/:platform/disconnect - Disconnect platform
// - GET /platforms/:platform/test - Test connection

```

```

interface PlatformManagement {
  platforms: {
    meta: MetaConnection;
    linkedin: LinkedInConnection;
    google: GoogleConnection;
    youtube: YouTubeConnection;
  };
  connectionStatus: {
    connected: Platform[];
    disconnected: Platform[];
    error: Platform[];
  };
  features: {
    oauth: boolean;
  };
}

```

```
webhooks: boolean;
analytics: boolean;
publishing: boolean;
};
}
```

## 7.2 Account Management

```
// pages/platforms/accounts.tsx
// API Endpoints Used:
// - GET /content/target-accounts - Get target accounts
// - POST /content/target-accounts - Create target account
// - PUT /content/target-accounts/:id - Update account
// - DELETE /content/target-accounts/:id - Delete account

interface AccountManagement {
  accounts: {
    facebook: FacebookAccount[];
    instagram: InstagramAccount[];
    linkedin: LinkedInAccount[];
    youtube: YouTubeChannel[];
  };
  features: {
    multiAccount: boolean;
    accountSwitching: boolean;
    permissions: boolean;
  };
}
```

## 8. Lead Management

### 8.1 Lead Dashboard



```
// pages/leads/index.tsx
// API Endpoints Used:
// - GET /leads - Get leads
// - POST /leads - Create lead
// - PUT /leads/:id - Update lead
// - GET /leads/:id/conversations - Get conversations
```

```
interface LeadManagement {
  leads: {
    new: Lead[];
    qualified: Lead[];
    contacted: Lead[];
    converted: Lead[];
  };
  features: {
    scoring: boolean;
    segmentation: boolean;
    automation: boolean;
    crmIntegration: boolean;
  };
  filters: {
    source: string[];
    score: number[];
    status: LeadStatus[];
    dateRange: DateRange;
  };
}
```

## 8.2 Lead Details

```
// pages/leads/[id].tsx
// API Endpoints Used:
// - GET /leads/:id - Get lead details
// - GET /conversations/:id - Get conversations
```

```
// - POST /conversations/:id/messages - Send message
```

```
interface LeadDetails {  
  lead: {  
    profile: LeadProfile;  
    score: LeadScore;  
    status: LeadStatus;  
    source: LeadSource;  
  };  
  conversations: {  
    history: Conversation[];  
    active: Conversation[];  
  };  
  actions: {  
    contact: boolean;  
    qualify: boolean;  
    convert: boolean;  
    archive: boolean;  
  };  
}
```

## 9. Admin Panel

### 9.1 System Administration

```
// pages/admin/index.tsx  
// API Endpoints Used:  
// - GET /admin/templates - Get templates  
// - POST /admin/templates - Create templates  
// - GET /admin/brand-rules - Get brand rules  
// - POST /admin/brand-rules - Create brand rules  
// - GET /admin/quality-policies - Get quality policies
```

```
interface AdminPanel {  
  templates: {
```

```

    prompt: PromptTemplate[];
    notification: NotificationTemplate[];
    workflow: WorkflowTemplate[];
  };
  rules: {
    brand: BrandRule[];
    quality: QualityPolicy[];
    workflow: WorkflowRule[];
  };
  settings: {
    system: SystemSettings;
    organization: OrganizationSettings;
    integrations: IntegrationSettings;
  };
}

```

## 9.2 AI Management

```

// pages/admin/ai.tsx
// API Endpoints Used:
// - POST /admin/generate - Generate content
// - GET /admin/templates - Get templates
// - POST /admin/templates - Create templates
// - GET /admin/analytics - Get AI analytics

interface AIManagement {
  generation: {
    models: AIModel[];
    templates: PromptTemplate[];
    performance: GenerationPerformance;
  };
  optimization: {
    content: ContentOptimization;
    media: MediaOptimization;
    scheduling: SchedulingOptimization;
  };
}

```

```
};  
analytics: {  
  usage: UsageAnalytics;  
  costs: CostAnalytics;  
  performance: PerformanceAnalytics;  
};  
}
```

## 10. Settings & Configuration

### 10.1 User Settings

```
// pages/settings/profile.tsx  
// API Endpoints Used:  
// - GET /auth/profile - Get user profile  
// - PUT /auth/profile - Update profile  
// - POST /auth/change-password - Change password
```

```
interface UserSettings {  
  profile: {  
    personal: PersonalInfo;  
    preferences: UserPreferences;  
    notifications: NotificationSettings;  
  };  
  security: {  
    password: PasswordSettings;  
    twoFactor: TwoFactorSettings;  
    sessions: SessionManagement;  
  };  
}
```

### 10.2 Organization Settings

```
// pages/settings/organization.tsx
// API Endpoints Used:
// - GET /organizations/:id - Get organization
// - PUT /organizations/:id - Update organization
// - GET /organizations/:id/users - Get users
// - POST /organizations/:id/users - Add user
```

```
interface OrganizationSettings {
  organization: {
    basic: OrganizationInfo;
    branding: BrandingSettings;
    integrations: IntegrationSettings;
  };
  users: {
    members: User[];
    roles: Role[];
    permissions: Permission[];
  };
  billing: {
    plan: BillingPlan;
    usage: UsageData;
    invoices: Invoice[];
  };
}
```

## Technical Implementation Details

### 1. API Integration Layer

#### 1.1 API Client Setup

```
// lib/api-client.ts
class APIClient {
  private baseUrl: string;
```

```

private authToken: string;

constructor() {
  this.baseURL = process.env.NEXT_PUBLIC_API_URL || '<http://localhost:44000>';
  this.authToken = this.getAuthToken();
}

// Content API methods
async getContent(params?: ContentQueryParams): Promise<ContentItem[]>
{
  return this.get('/content', { params });
}

async createContent(data: CreateContentDto): Promise<ContentItem> {
  return this.post('/content', data);
}

async generateContent(data: GenerateContentDto): Promise<GeneratedContent> {
  return this.post('/content/generate', data);
}

// Platform API methods
async getConnectedPlatforms(): Promise<PlatformAccountInfo[]> {
  return this.get('/platforms/connected');
}

async connectPlatform(platform: string): Promise<ConnectPlatformDto> {
  return this.post(`/platforms/${platform}/connect`);
}

// Analytics API methods
async getContentPerformance(contentId: string): Promise<ContentPerformance> {
  return this.get(`/content/${contentId}/performance`);
}

```

```

    }

    async getAnalytics(timeframe: string): Promise<AnalyticsData> {
      return this.get('/analytics', { params: { timeframe } });
    }
  }
}

```

## 1.2 Type Definitions

```

// types/api.ts
export interface ContentItem {
  id: string;
  title: string;
  type: 'BLOG' | 'NEWSLETTER' | 'SOCIAL_POST';
  status: 'DRAFT' | 'PENDING_APPROVAL' | 'APPROVED' | 'PUBLISHED' | 'ARCHIVED';
  tags: string[];
  metadata: Record<string, any>;
  createdAt: string;
  updatedAt: string;
  versions: ContentVersion[];
  schedules: Schedule[];
}

export interface Schedule {
  id: string;
  contentItemId: string;
  platform: Platform;
  scheduledAt: string;
  status: 'pending' | 'queued' | 'published' | 'failed';
  targetAccountId?: string;
  targetAccountName?: string;
  lastError?: string;
}

```

```
export interface PlatformAccountInfo {
  platform: string;
  accountId: string;
  accountName: string;
  accountType: string;
  isConnected: boolean;
  permissions: string[];
}
```

## 2. State Management

### 2.1 Zustand Stores

```
// stores/content-store.ts
interface ContentStore {
  content: ContentItem[];
  selectedContent: ContentItem | null;
  filters: ContentFilters;
  loading: boolean;
  error: string | null;

  // Actions
  fetchContent: () ⇒ Promise<void>;
  createContent: (data: CreateContentDto) ⇒ Promise<void>;
  updateContent: (id: string, data: UpdateContentDto) ⇒ Promise<void>;
  deleteContent: (id: string) ⇒ Promise<void>;
  setFilters: (filters: ContentFilters) ⇒ void;
  setSelectedContent: (content: ContentItem | null) ⇒ void;
}

// stores/platform-store.ts
interface PlatformStore {
  platforms: PlatformAccountInfo[];
  connectionStatus: Record<string, boolean>;
  loading: boolean;
```



```

error: string | null;

// Actions
fetchPlatforms: () ⇒ Promise<void>;
connectPlatform: (platform: string) ⇒ Promise<void>;
disconnectPlatform: (platform: string) ⇒ Promise<void>;
testConnection: (platform: string) ⇒ Promise<boolean>;
}

// stores/auth-store.ts
interface AuthStore {
  user: User | null;
  organization: Organization | null;
  isAuthenticated: boolean;
  loading: boolean;

  // Actions
  login: (credentials: LoginCredentials) ⇒ Promise<void>;
  logout: () ⇒ void;
  refreshToken: () ⇒ Promise<void>;
  updateProfile: (data: UpdateProfileDto) ⇒ Promise<void>;
}

```

## 3. Component Library

### 3.1 UI Components

```

// components/ui/
- Button.tsx           // Primary, secondary, danger buttons
- Input.tsx            // Text, email, password inputs
- Select.tsx           // Dropdown selections
- Modal.tsx            // Modal dialogs
- Toast.tsx            // Toast notifications
- LoadingSpinner.tsx   // Loading indicators
- DataTable.tsx        // Sortable, filterable tables

```

```
- Chart.tsx          // Chart components
- Calendar.tsx       // Calendar component
- RichTextEditor.tsx // WYSIWYG editor
- FileUpload.tsx     // File upload component
- PlatformIcon.tsx   // Platform icons
- StatusBadge.tsx    // Status indicators
```

## 3.2 Layout Components

```
// components/layout/
- AppShell.tsx      // Main app layout
- Sidebar.tsx       // Navigation sidebar
- Header.tsx        // Top header
- Breadcrumb.tsx    // Breadcrumb navigation
- PageHeader.tsx    // Page headers
- PageContent.tsx   // Page content wrapper
- Footer.tsx        // Footer component
```

## 4. Page Components

### 4.1 Dashboard Pages

```
// pages/
- dashboard/index.tsx // Main dashboard
- content/index.tsx   // Content list
- content/[id].tsx    // Content details
- content/[id]/edit.tsx // Content editor
- content/generate.tsx // AI generation
- calendar/index.tsx  // Content calendar
- inbox/index.tsx     // Social inbox
- inbox/[id].tsx      // Conversation details
- analytics/index.tsx // Analytics dashboard
- platforms/index.tsx // Platform management
- leads/index.tsx     // Lead management
- leads/[id].tsx      // Lead details
```

- admin/index.tsx // Admin panel
- settings/profile.tsx // User settings
- settings/organization.tsx // Organization settings

## **Implementation Timeline**

### **Week 1: Foundation & Setup**

- ☐ Next.js project setup with TypeScript
- ☐ Tailwind CSS configuration
- ☐ API client setup and type definitions
- ☐ Authentication system (NextAuth.js)
- ☐ Basic layout components (AppShell, Sidebar, Header)

### **Week 2: Core Dashboard**

- ☐ Dashboard home page with widgets
- ☐ Content list page with filters and search
- ☐ Content editor with rich text editing
- ☐ Basic calendar view
- ☐ Platform connection management

### **Week 3: Advanced Features**

- ☐ AI content generation interface
- ☐ Smart scheduling with drag-and-drop
- ☐ Social inbox with unified view
- ☐ Analytics dashboard with charts
- ☐ Lead management interface

### **Week 4: Admin & Settings**

- ☐ Admin panel for templates and rules

- ☐ User and organization settings
- ☐ Notification management
- ☐ System health monitoring
- ☐ Advanced configuration options

## Week 5: Polish & Testing

- ☐ Responsive design optimization
- ☐ Error handling and loading states
- ☐ Performance optimization
- ☐ User testing and feedback
- ☐ Documentation and deployment

## Design System

### Color Palette

```
:root {  
  --primary: #3B82F6;    /* Blue */  
  --secondary: #6B7280;  /* Gray */  
  --success: #10B981;    /* Green */  
  --warning: #F59E0B;    /* Amber */  
  --error: #EF4444;      /* Red */  
  --background: #F9FAFB; /* Light gray */  
  --surface: #FFFFFF;     /* White */  
  --text-primary: #111827; /* Dark gray */  
  --text-secondary: #6B7280; /* Medium gray */  
}
```

### Typography

```
/* Font families */
--font-primary: 'Inter', sans-serif;
--font-mono: 'JetBrains Mono', monospace;

/* Font sizes */
--text-xs: 0.75rem;
--text-sm: 0.875rem;
--text-base: 1rem;
--text-lg: 1.125rem;
--text-xl: 1.25rem;
--text-2xl: 1.5rem;
--text-3xl: 1.875rem;
```

## Getting Started

### 1. Project Setup

```
# Create Next.js project
npx create-next-app@latest agent-bowery-web --typescript --tailwind --eslint

# Install additional dependencies
npm install @headlessui/react @heroicons/react zustand axios react-hook-form @hookform/resolvers zod recharts react-big-calendar next-auth
```

### 2. Environment Configuration

```
# .env.local
NEXT_PUBLIC_API_URL=http://localhost:44000
NEXTAUTH_URL=http://localhost:43000
NEXTAUTH_SECRET=your-secret-key
```

### 3. First Component

```
// components/dashboard/SystemHealthWidget.tsx
import { useEffect, useState } from 'react';
import { APIClient } from '@lib/api-client';

export function SystemHealthWidget() {
  const [health, setHealth] = useState(null);
  const api = new APIClient();

  useEffect(() => {
    api.getSystemHealth().then(setHealth);
  }, []);

  return (
    <div className="bg-white p-6 rounded-lg shadow">
      <h3 className="text-lg font-semibold mb-4">System Health</h3>
      <div className="flex items-center space-x-2">
        <div className={`w-3 h-3 rounded-full ${health?.status === 'ok' ? 'bg-green-500' : 'bg-red-500'} ` />
        <span className="text-sm text-gray-600">
          {health?.status === 'ok' ? 'All systems operational' : 'System issues detected'}
        </span>
      </div>
    </div>
  );
}
```

This comprehensive roadmap provides everything you need to build a modern, feature-rich frontend dashboard that fully leverages your existing backend APIs and database structure. The implementation is organized by priority and complexity, ensuring you can build incrementally while maintaining a high-quality user experience.

[1 tool called]

Perfect! I've completed the comprehensive frontend dashboard roadmap based on your existing backend APIs and database structure. The roadmap includes:

## **What I've Delivered:**

### **1. Complete Dashboard Architecture**

- **Technology Stack:** Next.js 14+, Tailwind CSS, Zustand, Recharts
- **10 Main Sections:** Dashboard, Content, Calendar, Social Inbox, Analytics, Platforms, Leads, Admin, Settings
- **50+ Components:** From basic UI components to complex dashboard widgets

### **2. Detailed API Integration**

- **Mapped all your existing endpoints** to frontend functionality
- **Type definitions** for all data structures
- **API client setup** with proper error handling
- **State management** with Zustand stores

### **3. Feature-Rich Components**

- **Content Management:** List, editor, AI generation, scheduling
- **Social Inbox:** Unified message management across platforms
- **Analytics Dashboard:** Performance charts and metrics
- **Platform Management:** Connection status and account management
- **Lead Management:** Lead scoring and conversation tracking
- **Admin Panel:** Template management and system configuration

### **4. Implementation Timeline**

- **5-week roadmap** with weekly milestones
- **Priority-based development** starting with core features
- **Progressive enhancement** from basic to advanced features

## 5. Technical Specifications

- **Component library** with reusable UI components
- **Design system** with consistent colors and typography
- **Responsive design** for all screen sizes
- **Performance optimization** strategies

The roadmap is designed to work seamlessly with your existing backend APIs, database schema, and business requirements. You can start implementing immediately with the provided code examples and component structures.

Would you like me to help you begin implementing any specific section of the dashboard, or would you prefer to focus on a particular feature first?

## Frontend Dashboard Implementation Plan

### Phase 5: Frontend Dashboard (Week 5-7)




---

#### Executive Summary



##### Project Overview

Transform the existing Node.js placeholder ( `apps/web/server.js` ) into a comprehensive React/Next.js dashboard that provides a complete user interface for the Agent Bowery content management system.






##### Current State

-  **Frontend:** Only basic Node.js server placeholder
-  **Backend:** Complete API with 50+ endpoints
-  **Database:** Full schema with 20+ tables



-  **Authentication:** JWT-based auth system
-  **Platforms:** Meta, LinkedIn, Google integrations

## Target State

-  **Modern React Dashboard:** Next.js 14+ with TypeScript
  -  **Complete UI:** All backend functionality accessible via frontend
  -  **Responsive Design:** Mobile-first, desktop-optimized
  -  **Real-time Updates:** Live data synchronization
  -  **User Experience:** Intuitive, professional interface
- 

## Success Metrics

### Technical Metrics

- **Performance:** < 3s initial load time, < 1s navigation
- **Accessibility:** WCAG 2.1 AA compliance
- **Mobile:** 100% responsive on all devices
- **Browser Support:** Chrome, Firefox, Safari, Edge (last 2 versions)

### Business Metrics

- **User Adoption:** 90% of backend features accessible via UI
  - **Efficiency:** 50% reduction in manual API calls
  - **Satisfaction:** Intuitive workflow for content creators
  - **Productivity:** Streamlined content creation and management
- 

## Detailed Implementation Timeline

### Week 1: Foundation & Core Setup (Days 1-7)

#### Day 1-2: Project Setup

#### # Tasks:

- [ ] Create Next.js 14 project with App Router
- [ ] Configure TypeScript with strict mode
- [ ] Set up Tailwind CSS with custom design system
- [ ] Install and configure essential dependencies
- [ ] Set up ESLint and Prettier
- [ ] Configure environment variables

#### # Deliverables:

- Working Next.js application
- TypeScript configuration
- Tailwind CSS setup
- Development environment ready

## Day 3-4: Authentication & API Integration

#### # Tasks:

- [ ] Set up NextAuth.js with JWT strategy
- [ ] Create API client with Axios
- [ ] Implement authentication guards
- [ ] Set up API interceptors for token refresh
- [ ] Create type definitions for all API responses
- [ ] Implement error handling middleware

#### # Deliverables:

- Authentication system working
- API client configured
- Type definitions complete
- Error handling implemented

## Day 5-7: Layout & Navigation

#### # Tasks:

- [ ] Create AppShell component with responsive sidebar

- [ ] Build Header component with user profile
- [ ] Implement navigation menu with role-based access
- [ ] Create breadcrumb navigation
- [ ] Set up page layouts and routing
- [ ] Add loading states and error boundaries

# Deliverables:

- Complete layout system
- Responsive navigation
- Role-based access control
- Error boundaries implemented

## **Week 2: Core Dashboard Features (Days 8-14)**

### **Day 8-9: Dashboard Home Page**

# Tasks:

- [ ] Create dashboard overview with key metrics
- [ ] Build system health widget
- [ ] Implement content statistics cards
- [ ] Add platform connection status
- [ ] Create recent activity feed
- [ ] Build upcoming schedules widget

# Deliverables:

- Dashboard home page complete
- Real-time system health monitoring
- Key metrics visualization
- Activity feed working

### **Day 10-11: Content Management Interface**

# Tasks:

- [ ] Build content list page with filters and search
- [ ] Create content editor with rich text editing

- [ ] Implement content preview for different platforms
- [ ] Add content versioning interface
- [ ] Build content approval workflow
- [ ] Create bulk operations interface

# Deliverables:

- Content management interface
- Rich text editor working
- Platform previews functional
- Approval workflow implemented

## Day 12-14: AI Content Generation

# Tasks:

- [ ] Create AI content generation interface
- [ ] Build prompt template management
- [ ] Implement brand rule validation UI
- [ ] Add quality check results display
- [ ] Create content optimization suggestions
- [ ] Build batch generation interface

# Deliverables:

- AI generation interface complete
- Template management working
- Brand validation UI
- Quality checks displayed

## Week 3: Advanced Features (Days 15-21)

### Day 15-16: Content Calendar

# Tasks:

- [ ] Implement calendar view with React Big Calendar
- [ ] Add drag-and-drop scheduling
- [ ] Create smart scheduling suggestions

- [ ] Implement conflict detection
- [ ] Build recurring schedule management
- [ ] Add calendar filters and views

# Deliverables:

- Interactive calendar interface
- Drag-and-drop scheduling
- Smart scheduling working
- Conflict detection functional

## Day 17-18: Social Inbox

# Tasks:

- [ ] Create unified message inbox
- [ ] Implement conversation threading
- [ ] Add auto-reply configuration
- [ ] Build lead extraction interface
- [ ] Create message templates
- [ ] Implement sentiment analysis display

# Deliverables:

- Unified social inbox
- Conversation management
- Lead extraction working
- Auto-reply configuration

## Day 19-21: Analytics Dashboard

# Tasks:

- [ ] Build analytics dashboard with charts
- [ ] Implement content performance metrics
- [ ] Create platform analytics views
- [ ] Add ROI tracking interface
- [ ] Build custom date range selectors
- [ ] Implement data export functionality

#### # Deliverables:

- Analytics dashboard complete
- Performance charts working
- ROI tracking functional
- Data export implemented

## **Week 4: Platform & Lead Management (Days 22-28)**

### **Day 22-23: Platform Management**

#### # Tasks:

- [ ] Create platform connection interface
- [ ] Build account management system
- [ ] Implement OAuth flow integration
- [ ] Add platform testing functionality
- [ ] Create webhook configuration
- [ ] Build platform analytics views

#### # Deliverables:

- Platform management interface
- OAuth integration working
- Account management complete
- Webhook configuration functional

### **Day 24-25: Lead Management**

#### # Tasks:

- [ ] Build lead dashboard with scoring
- [ ] Create lead detail pages
- [ ] Implement conversation history
- [ ] Add lead segmentation
- [ ] Create lead automation rules
- [ ] Build CRM integration interface

#### # Deliverables:

- Lead management system
- Lead scoring working
- Conversation history complete
- Automation rules functional

## **Day 26-28: Admin Panel**

#### # Tasks:

- [ ] Create admin dashboard
- [ ] Build template management interface
- [ ] Implement brand rule configuration
- [ ] Add quality policy management
- [ ] Create user management system
- [ ] Build system configuration interface

#### # Deliverables:

- Admin panel complete
- Template management working
- Brand rules configurable
- User management functional

## **Week 5: Polish & Optimization (Days 29-35)**

### **Day 29-30: Responsive Design**

#### # Tasks:

- [ ] Optimize for mobile devices
- [ ] Test on tablets and different screen sizes
- [ ] Implement touch-friendly interactions
- [ ] Add mobile-specific navigation
- [ ] Optimize images and assets
- [ ] Test accessibility compliance

#### # Deliverables:

- Mobile-optimized interface
- Touch interactions working
- Accessibility compliant
- Cross-device tested

## **Day 31-32: Performance Optimization**

### **# Tasks:**

- [ ] Implement code splitting
- [ ] Add lazy loading for components
- [ ] Optimize bundle size
- [ ] Implement caching strategies
- [ ] Add performance monitoring
- [ ] Optimize API calls

### **# Deliverables:**

- Optimized performance
- Code splitting implemented
- Caching working
- Bundle size optimized

## **Day 33-35: Testing & Documentation**

### **# Tasks:**

- [ ] Write unit tests for components
- [ ] Add integration tests
- [ ] Create user documentation
- [ ] Build component library documentation
- [ ] Add deployment configuration
- [ ] Create user onboarding flow

### **# Deliverables:**

- Test suite complete
- Documentation written



- Deployment ready
- User onboarding complete

## Technical Implementation Details

### 1. Project Structure

```
apps/web/
├── src/
│   ├── app/                # Next.js App Router
│   │   ├── (auth)/         # Auth routes
│   │   ├── dashboard/      # Dashboard pages
│   │   ├── content/        # Content management
│   │   ├── calendar/       # Calendar views
│   │   ├── inbox/          # Social inbox
│   │   ├── analytics/      # Analytics dashboard
│   │   ├── platforms/      # Platform management
│   │   ├── leads/          # Lead management
│   │   ├── admin/          # Admin panel
│   │   └── settings/       # Settings pages
│   ├── components/         # Reusable components
│   │   ├── ui/             # Basic UI components
│   │   ├── layout/         # Layout components
│   │   ├── dashboard/      # Dashboard widgets
│   │   ├── content/        # Content components
│   │   ├── calendar/       # Calendar components
│   │   ├── inbox/          # Inbox components
│   │   ├── analytics/      # Analytics components
│   │   └── forms/          # Form components
│   ├── lib/                # Utilities and configurations
│   │   ├── api/            # API client
│   │   ├── auth/           # Authentication
│   │   ├── utils/          # Utility functions
│   │   └── validations/    # Form validations
```

```

|   |— stores/           # State management
|   |— types/           # TypeScript types
|   |— hooks/           # Custom React hooks
|— public/              # Static assets
|— package.json
|— tailwind.config.js
|— next.config.js
|— tsconfig.json

```

## 2. Dependencies

```

{
  "dependencies": {
    "next": "^14.0.0",
    "react": "^18.0.0",
    "react-dom": "^18.0.0",
    "@next/font": "^14.0.0",
    "next-auth": "^4.24.0",
    "axios": "^1.6.0",
    "zustand": "^4.4.0",
    "react-hook-form": "^7.47.0",
    "@hookform/resolvers": "^3.3.0",
    "zod": "^3.22.0",
    "recharts": "^2.8.0",
    "react-big-calendar": "^1.8.0",
    "@headlessui/react": "^1.7.0",
    "@heroicons/react": "^2.0.0",
    "clsx": "^2.0.0",
    "tailwind-merge": "^2.0.0",
    "date-fns": "^2.30.0",
    "react-hot-toast": "^2.4.0",
    "framer-motion": "^10.16.0"
  },
  "devDependencies": {
    "@types/node": "^20.0.0",

```

```
"@types/react": "^18.0.0",
"@types/react-dom": "^18.0.0",
"typescript": "^5.0.0",
"tailwindcss": "^3.3.0",
"autoprefixer": "^10.4.0",
"postcss": "^8.4.0",
"eslint": "^8.0.0",
"eslint-config-next": "^14.0.0",
"prettier": "^3.0.0",
"@testing-library/react": "^13.4.0",
"@testing-library/jest-dom": "^6.0.0",
"jest": "^29.0.0",
"jest-environment-jsdom": "^29.0.0"
}
}
```

### 3. Environment Configuration

```
# .env.local
NEXT_PUBLIC_API_URL=http://localhost:44000
NEXTAUTH_URL=http://localhost:43000
NEXTAUTH_SECRET=your-secret-key
NEXT_PUBLIC_APP_NAME=Agent Bowery
NEXT_PUBLIC_APP_VERSION=1.0.0
```

## Resource Requirements

### Development Team

- **1 Frontend Developer** (React/Next.js expert)
- **1 UI/UX Designer** (part-time, 2 weeks)
- **1 QA Tester** (part-time, 1 week)

### Infrastructure





- **Development Environment:** Local development with Docker
- **Staging Environment:** Cloud deployment for testing
- **Production Environment:** Cloud deployment with CI/CD

## Tools & Services




- **Design:** Figma for UI/UX design
  - **Version Control:** Git with GitHub
  - **CI/CD:** GitHub Actions
  - **Monitoring:** Sentry for error tracking
  - **Analytics:** Google Analytics for usage tracking
- 

## Key Deliverables




### Week 1 Deliverables

-  Next.js project setup
-  Authentication system
-  Basic layout components
-  API integration layer




### Week 2 Deliverables

-  Dashboard home page
-  Content management interface
-  AI content generation UI




### Week 3 Deliverables

-  Content calendar with scheduling
-  Social inbox interface
-  Analytics dashboard

## Week 4 Deliverables

-  Platform management
-  Lead management system
-  Admin panel

## Week 5 Deliverables

-  Responsive design
  -  Performance optimization
  -  Testing and documentation
- 

## Risk Mitigation

### Technical Risks

- **API Integration Issues:** Create comprehensive API client with error handling
- **Performance Problems:** Implement code splitting and lazy loading
- **Browser Compatibility:** Use progressive enhancement approach

### Timeline Risks

- **Scope Creep:** Stick to MVP features, defer nice-to-haves
- **Integration Delays:** Start API integration early
- **Testing Bottlenecks:** Implement testing throughout development

### Quality Risks

- **User Experience:** Regular user testing and feedback
  - **Accessibility:** WCAG compliance testing
  - **Security:** Security audit and penetration testing
- 

## Success Criteria

## Technical Success

- ☐ All backend APIs accessible via frontend
- ☐ < 3 second initial load time
- ☐ 100% responsive on all devices
- ☐ WCAG 2.1 AA accessibility compliance

## Business Success

- ☐ 90% of content operations manageable via UI
- ☐ 50% reduction in manual API usage
- ☐ Positive user feedback on usability
- ☐ Successful deployment to production

## User Experience Success

- ☐ Intuitive navigation and workflows
  - ☐ Consistent design language
  - ☐ Fast, responsive interactions
  - ☐ Clear error messages and feedback
- 

## Next Steps

### Immediate Actions (Next 24 Hours)

1. Set up development environment
2. Create Next.js project
3. Configure TypeScript and Tailwind
4. Set up Git repository

### Week 1 Priorities

1. Complete project setup

2. **Implement authentication**
3. **Create basic layout**
4. **Set up API integration**

## **Success Metrics Tracking**

- **Daily:** Development progress and blockers
- **Weekly:** Feature completion and quality metrics
- **End of Phase:** Full system integration and user testing

This comprehensive plan provides a clear roadmap for implementing the frontend dashboard with specific timelines, deliverables, and success criteria. The phased approach ensures steady progress while maintaining quality and meeting business requirements.