Technical Specification - Camtivate Admin Dashboard

Dashboard Technical Architecture

This document outlines the detailed technical specifications for building the Camtivate Admin Dashboard, a comprehensive real-time business intelligence platform.

1. System Architecture Overview

Frontend Architecture

- **Framework**: React 18 with TypeScript
- **State Management**: Redux Toolkit with RTK Query
- **UI Library**: Material-UI v5 or Ant Design
- **Charting**: Recharts or D3.js for data visualization
- **Real-time Updates**: WebSocket connections with Socket.io
- **Mobile Responsive**: CSS Grid/Flexbox with breakpoint system

Backend Architecture

- **API Gateway**: Node.js with Express.js or Python FastAPI
- **Database**: PostgreSQL for relational data, Redis for caching
- **Message Queue**: Redis Pub/Sub or Apache Kafka
- **Search Engine**: Elasticsearch for dashboard search functionality
- **File Storage**: AWS S3 or Google Cloud Storage

Infrastructure

- **Cloud Platform**: AWS or Google Cloud Platform

- **Container Orchestration**: Docker with Kubernetes
- **CDN**: CloudFlare or AWS CloudFront
- **Monitoring**: New Relic or Datadog
- **CI/CD**: GitHub Actions or GitLab CI

2. Dashboard Card Specifications

Card 1: Task Overview

Visual Components:

- Progress rings showing completion percentages
- Color-coded task status indicators
- Interactive filter chips for client tags
- Real-time task counter animations

Technical Implementation:

```
interface TaskOverviewCard {
  openTasks: number;
  overdueTasks: number;
  totalTasks: number;
  clientTags: ClientTag[];
  filters: TaskFilter[];
}

interface ClientTag {
  id: string;
  name: string;
  color: string;
  taskCount: number;
}
```

API Endpoints:

```
GET /api/dashboard/tasks/overview
GET /api/dashboard/tasks/filter?tags=client1,client2
WS /dashboard/tasks/live-updates
```

Data Refresh: Real-time via WebSocket, fallback polling every 30 seconds

Card 2: Workload Distribution

Visual Components:

- Interactive pie chart with hover effects
- Drill-down capability to task details
- Status legend with task counts
- Tag-based filtering sidebar

Technical Implementation:

```
interface WorkloadDistribution {
  activeTasks: number;
  completed: number;
  inProgress: number;
  dependentTasks: number;
  notStarted: number;
  chartData: ChartDataPoint[];
}

interface ChartDataPoint {
  label: string;
  value: number;
  color: string;
  percentage: number;
}
```

Chart Library Configuration:

```
// Using Recharts
<PieChart width={400} height={300}>
  <Pie
    dataKey="value"
    data={workloadData}
    cx={200}
    cy={150}
    outerRadius={80}
    label={({percentage}) => `${percentage}%`}
  />
  <Tooltip />
</PieChart>
```

Card 3: Calendar & Meetings

Visual Components:

- Weekly grid calendar view
- Color-coded meeting types
- Availability indicator
- Multi-platform integration badges

Technical Implementation:

```
interface CalendarCard {
  weekView: WeekViewData;
  upcomingMeetings: Meeting[];
  availability: AvailabilitySlot[];
  integrations: CalendarIntegration[];
}

interface Meeting {
  id: string;
  title: string;
  startTime: Date;
```

```
  endTime: Date;
  platform: 'google' | 'outlook' | 'teams' | 'zoom';
  attendees: string[];
  status: 'confirmed' | 'tentative' | 'cancelled';
}
```

Integration APIs:

```
// Google Calendar
GET /api/integrations/google/calendar/events
// Microsoft Outlook
GET /api/integrations/microsoft/calendar/events
// Zoom
GET /api/integrations/zoom/meetings
```

Card 4: Client Engagement

Visual Components:

- Client health score meters
- Communication timeline
- Response time indicators
- Relationship trend graphs

Technical Implementation:

```
interface ClientEngagement {
  clients: ClientMetrics[];
  recentCommunications: Communication[];
  pendingResponses: PendingResponse[];
  healthScores: HealthScore[];
}

interface ClientMetrics {
  clientId: string;
  clientName: string;
```

```
  healthScore: number; // 0-100
  lastContactDate: Date;
  responseTime: number; // hours
  communicationFrequency: number;
  projectStatus: 'active' | 'on_hold' | 'completed';
}
```

AI Integration:

```
# Client Health Scoring Algorithm
def calculate_health_score(client_data):
    factors = {
        'payment_history': 0.3,
        'communication_frequency': 0.25,
        'project_progress': 0.25,
        'response_time': 0.2
    }
    return weighted_score(client_data, factors)
```

Card 5: Financial Snapshot

Visual Components:

- Revenue trend line chart
- Expense breakdown donut chart
- Cash flow projection graph
- Outstanding invoices table

Technical Implementation:

```
interface FinancialSnapshot {
  currentRevenue: number;
  monthlyExpenses: number;
  projectedCashFlow: CashFlowProjection[];
  outstandingInvoices: Invoice[];
  comparisonData: PeriodComparison;
```

```
}

interface CashFlowProjection {
  date: Date;
  projected: number;
  actual?: number;
  variance?: number;
}
```

Integration with Financial Systems:

```
// Monarch Money API
GET /api/integrations/monarch/accounts/summary
GET /api/integrations/monarch/transactions
// Ghostfilio API
GET /api/integrations/ghostfilio/portfolio
```

Card 6: Marketing Performance

Visual Components:

- Social media metrics grid
- Campaign ROI charts
- Content performance heatmap
- Trend analysis graphs

Technical Implementation:

```
interface MarketingPerformance {
  socialMetrics: SocialMediaMetrics;
  campaignData: CampaignMetrics[];
  contentPerformance: ContentMetrics[];
  trendAnalysis: TrendData[];
}

interface SocialMediaMetrics {
```

```
  platform: string;
  followers: number;
  engagement: number;
  reach: number;
  scheduledPosts: number;
  contentGaps: string[];
}
```

Social Media API Integrations:

```
// Multi-platform social media data aggregation
const socialPlatforms = {
  facebook: '/api/integrations/facebook/insights',
  instagram: '/api/integrations/instagram/metrics',
  linkedin: '/api/integrations/linkedin/analytics',
  twitter: '/api/integrations/twitter/metrics',
  youtube: '/api/integrations/youtube/analytics',
  tiktok: '/api/integrations/tiktok/metrics'
};
```

Card 7: Software & Tool Utilization

Visual Components:

- Tool usage dashboard grid
- Subscription status indicators
- Cost per tool breakdown
- Integration health monitors

Technical Implementation:

```
interface ToolUtilization {
  tools: SoftwareTool[];
  totalMonthlyCost: number;
  usageStatistics: ToolUsage[];
  integrationStatus: IntegrationHealth[];
```

```
}

interface SoftwareTool {
  id: string;
  name: string;
  category: string;
  monthlyCost: number;
  lastUsed: Date;
  usageFrequency: number;
  subscriptionStatus: 'active' | 'expired' | 'trial';
  integrationStatus: 'connected' | 'disconnected' | 'error';
}
```

Card 8: AI Assistant Activity

Visual Components:

- Performance metrics dashboard
- Task automation statistics
- Time saved calculations
- Intervention requirement alerts

Technical Implementation:

```
interface AIAssistantActivity {
  tasksAutomated: number;
  timeSaved: number; // in hours
  accuracyRate: number;
  interventionsRequired: Intervention[];
  performanceMetrics: AIMetrics[];
}

interface AIMetrics {
  assistantType: 'openai' | 'gemini';
  requestsProcessed: number;
  averageResponseTime: number;
```

```
  successRate: number;
  costPerRequest: number;
}
```

3. Real-Time Data Architecture

WebSocket Implementation

```
// Client-side WebSocket connection
class DashboardWebSocket {
  constructor() {
    this.socket = io('/dashboard', {
      transports: ['websocket']
    });
    this.setupEventListeners();
  }

  setupEventListeners() {
    this.socket.on('task-update', this.handleTaskUpdate);
    this.socket.on('client-activity', this.handleClientActivity);
    this.socket.on('financial-update', this.handleFinancialUpdate);
    this.socket.on('ai-activity', this.handleAIActivity);
  }

  subscribeToCard(cardId) {
    this.socket.emit('subscribe', { cardId });
  }
}
```

Data Caching Strategy

```
// Redis caching implementation
const cacheConfig = {
  taskOverview: { ttl: 300 }, // 5 minutes
```

```
  clientData: { ttl: 900 }, // 15 minutes
  financialData: { ttl: 1800 }, // 30 minutes
  marketingMetrics: { ttl: 3600 }, // 1 hour
  toolUtilization: { ttl: 7200 } // 2 hours
};
```

4. Integration Specifications

Notion API Integration

```
// Notion database queries
class NotionIntegration {
  async getTaskOverview() {
    const response = await notion.databases.query({
      database_id: TASKS_DATABASE_ID,
      filter: {
        property: 'Status',
        select: { does_not_equal: 'Completed' }
      }
    });
    return this.processTaskData(response);
  }

  async getClientData() {
    return await notion.databases.query({
      database_id: CLIENTS_DATABASE_ID,
      sorts: [{
        property: 'Last Updated',
        direction: 'descending'
      }]
    });
  }
}
```

Google Workspace Integration

```
// Google APIs integration
class GoogleIntegration {
  constructor() {
    this.calendar = google.calendar({ version: 'v3', auth: oauth2Client });
    this.gmail = google.gmail({ version: 'v1', auth: oauth2Client });
  }

  async getCalendarEvents() {
    const response = await this.calendar.events.list({
      calendarId: 'primary',
      timeMin: new Date().toISOString(),
      maxResults: 50,
      singleEvents: true,
      orderBy: 'startTime'
    });
    return response.data.items;
  }
}
```

Microsoft 365 Integration

```
// Microsoft Graph API integration
class MicrosoftIntegration {
  async getOutlookCalendar() {
    const response = await fetch('/api/microsoft/calendar/events', {
      headers: {
        'Authorization': `Bearer ${accessToken}`,
        'Content-Type': 'application/json'
      }
    });
    return response.json();
```

```
  }
}
```

5. Database Schema

Core Dashboard Tables

```
-- Dashboard configurations
CREATE TABLE dashboard_cards (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id),
    card_type VARCHAR(50) NOT NULL,
    position INTEGER NOT NULL,
    size VARCHAR(20) DEFAULT 'medium',
    config JSONB,
    is_visible BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Task management
CREATE TABLE tasks (
    id UUID PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    status VARCHAR(50) NOT NULL,
    priority VARCHAR(20) DEFAULT 'medium',
    client_id UUID REFERENCES clients(id),
    assigned_to UUID REFERENCES users(id),
    due_date DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Client management
```

```
CREATE TABLE clients (
    id UUID PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    company VARCHAR(255),
    email VARCHAR(255),
    phone VARCHAR(50),
    address TEXT,
    website VARCHAR(255),
    health_score DECIMAL(3,2),
    last_contact DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Financial tracking
CREATE TABLE financial_transactions (
    id UUID PRIMARY KEY,
    type VARCHAR(50) NOT NULL, -- 'income', 'expense'
    amount DECIMAL(10,2) NOT NULL,
    description VARCHAR(255),
    category VARCHAR(100),
    client_id UUID REFERENCES clients(id),
    transaction_date DATE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- AI activity tracking
CREATE TABLE ai_activity_logs (
    id UUID PRIMARY KEY,
    ai_service VARCHAR(50) NOT NULL, -- 'openai', 'gemini'
    task_type VARCHAR(100),
    input_tokens INTEGER,
    output_tokens INTEGER,
    cost DECIMAL(8,4),
    response_time_ms INTEGER,
    success BOOLEAN DEFAULT true,
```

```
    error_message TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Integration status
CREATE TABLE integration_status (
    id UUID PRIMARY KEY,
    service_name VARCHAR(100) NOT NULL,
    status VARCHAR(20) DEFAULT 'active', -- 'active', 'error', 'disabled'
    last_sync TIMESTAMP,
    error_count INTEGER DEFAULT 0,
    config JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

6. Performance Optimization

Frontend Performance

```
// React optimization strategies
const DashboardCard = React.memo(({ cardData, cardType }) => {
  // Memoized expensive calculations
  const processedData = useMemo(() => {
    return processCardData(cardData, cardType);
  }, [cardData, cardType]);

  // Debounced filter updates
  const debouncedFilter = useCallback(
    debounce((filters) => updateFilters(filters), 300),
    []
  );

  return (
    <Card>
```

```
      {/* Card content */}
    </Card>
  );
});

// Lazy loading for dashboard cards
const LazyTaskOverview = lazy(() => import('./cards/TaskOverviewCard'));
const LazyWorkloadDistribution = lazy(() => import('./cards/WorkloadCard'));
```

Backend Performance

```
// API response caching
const cacheMiddleware = (duration) => {
  return (req, res, next) => {
    const key = `${req.originalUrl}`;
    const cached = cache.get(key);

    if (cached) {
      return res.json(cached);
    }

    res.sendResponse = res.json;
    res.json = (body) => {
      cache.set(key, body, duration);
      res.sendResponse(body);
    };

    next();
  };
};

// Database query optimization
app.get('/api/dashboard/tasks/overview',
  cacheMiddleware(300), // 5 minute cache
  async (req, res) => {
```

```
    const result = await db.query(`
      SELECT
        status,
        COUNT(*) as count,
        client_tag
      FROM tasks
      WHERE deleted_at IS NULL
      GROUP BY status, client_tag
    `);

    res.json(processTaskOverview(result));
  }
);
```

7. Security Implementation

Authentication & Authorization

```
// JWT-based authentication
const authMiddleware = (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'No token provided' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(401).json({ error: 'Invalid token' });
  }
};
```

```
// Role-based access control
const requireRole = (roles) => {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ error: 'Insufficient permissions' });
    }
    next();
  };
};
```

Data Encryption

```
// Sensitive data encryption
const crypto = require('crypto');

class EncryptionService {
  constructor() {
    this.algorithm = 'aes-256-gcm';
    this.key = Buffer.from(process.env.ENCRYPTION_KEY, 'hex');
  }

  encrypt(text) {
    const iv = crypto.randomBytes(16);
    const cipher = crypto.createCipher(this.algorithm, this.key);
    cipher.setAAD(Buffer.from('dashboard'));

    let encrypted = cipher.update(text, 'utf8', 'hex');
    encrypted += cipher.final('hex');

    const authTag = cipher.getAuthTag();

    return {
      encrypted,
      iv: iv.toString('hex'),
      authTag: authTag.toString('hex')
```

```
    };
  }
}
```

8. Testing Strategy

Unit Testing

```
// Jest testing for dashboard components
describe('TaskOverviewCard', () => {
  test('displays correct task counts', () => {
    const mockData = {
      openTasks: 15,
      overdueTasks: 3,
      totalTasks: 45
    };

    render(<TaskOverviewCard data={mockData} />);

    expect(screen.getByText('15')).toBeInTheDocument();
    expect(screen.getByText('3')).toBeInTheDocument();
    expect(screen.getByText('45')).toBeInTheDocument();
  });

  test('filters tasks by client tag', () => {
    const mockData = { /* mock data */ };
    const mockFilter = jest.fn();

    render(<TaskOverviewCard data={mockData} onFilter={mockFilter} />);

    fireEvent.click(screen.getByText('CyberSecured AI'));
    expect(mockFilter).toHaveBeenCalledWith(['cybersecured-ai']);
  });
});
```

Integration Testing

```
// API endpoint testing
describe('Dashboard API', () => {
  test('GET /api/dashboard/tasks/overview returns task data', async () => {
    const response = await request(app)
      .get('/api/dashboard/tasks/overview')
      .set('Authorization', `Bearer ${validToken}`);

    expect(response.status).toBe(200);
    expect(response.body).toHaveProperty('openTasks');
    expect(response.body).toHaveProperty('overdueTasks');
    expect(response.body).toHaveProperty('totalTasks');
  });
});
```

9. Deployment Configuration

Docker Configuration

```
# Frontend Dockerfile
FROM node:18-alpine as builder
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Kubernetes Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dashboard-frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: dashboard-frontend
  template:
    metadata:
      labels:
        app: dashboard-frontend
    spec:
      containers:
      - name: dashboard-frontend
        image: camtivate/dashboard-frontend:latest
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "128Mi"
            cpu: "100m"
          limits:
            memory: "256Mi"
            cpu: "200m"
```

10. Monitoring & Analytics

Performance Monitoring

```
// Custom analytics for dashboard usage
class DashboardAnalytics {
```

```
  trackCardView(cardType, userId) {
    analytics.track('Dashboard Card Viewed', {
      cardType,
      userId,
      timestamp: new Date().toISOString()
    });
  }

  trackFilterUsage(cardType, filters, userId) {
    analytics.track('Dashboard Filter Applied', {
      cardType,
      filters,
      userId,
      timestamp: new Date().toISOString()
    });
  }

  trackPerformance(endpoint, responseTime) {
    if (responseTime > 1000) {
      analytics.track('Slow API Response', {
        endpoint,
        responseTime,
        timestamp: new Date().toISOString()
      });
    }
  }
}
```

11. Cost Estimation

Development Costs

- **Frontend Development**: $80K - $120K (3-4 months)
- **Backend API Development**: $60K - $90K (2-3 months)
- **Integration Development**: $40K - $60K (1-2 months)

- **Testing & QA**: $20K - $30K (1 month)
- **DevOps & Deployment**: $15K - $25K

Annual Operating Costs

- **Cloud Infrastructure**: $12K - $24K
- **Third-party API Costs**: $6K - $12K
- **Monitoring & Security Tools**: $3K - $6K
- **Database Hosting**: $3K - $6K

Total Development Cost: $215K - $325K

Annual Operating Cost: $24K - $48K

This technical specification provides a complete roadmap for building a comprehensive, real-time admin dashboard with robust integrations, AI-powered insights, and scalable architecture.