Phase II: Climate Anxiety Survey App - Supabase Integration

1. Phase II Objectives

1.1 Core Goal

Integrate Supabase backend to Phase I application, enabling survey data submission while maintaining all existing localStorage functionality as backup. Focus on seamless data persistence, analytics capability, and research data collection.

1.2 Phase II Scope

- ✅ Supabase database integration
- ✅ Survey response submission to cloud
- ✅ Hybrid storage: localStorage + Supabase
- ✅ Data analytics and aggregation capabilities
- ✅ Graceful offline/online handling
- ✅ Data export functionality for researchers
- ✅ Anonymous user identification and tracking

1.3 What Stays from Phase I

- All existing UI components and user experience
- localStorage as primary storage during survey
- Flow diagram and path management
- Question types and branching logic
- Reset/clear functionality

2. Technical Architecture

2.1 Enhanced Tech Stack

- **Frontend**: Vite + React + TypeScript (unchanged)
- **Hosting**: GitHub Pages (unchanged)
- **Primary Storage**: Browser localStorage (unchanged)
- **Backend**: Supabase PostgreSQL database
- **Data Sync**: localStorage → Supabase on completion
- **Authentication**: Anonymous sessions (no login required)

2.2 Data Flow Enhancement

```
Survey JSON → React State → localStorage → Flow Diagram
   ↑           ↓                  ↓
User Interaction ← UI Components ← State Updates
           ↓
        [Survey Complete]
           ↓
        Supabase Submit → Analytics Dashboard
```

3. Supabase Integration

3.1 Database Schema

```sql
-- Main survey responses table
CREATE TABLE survey_responses (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  session_id UUID NOT NULL,
  survey_id TEXT NOT NULL DEFAULT 'climate-anxiety-assessment-v1',

  -- Survey completion data
  response_data JSONB NOT NULL,
  paths_explored INTEGER DEFAULT 1,
  questions_answered INTEGER DEFAULT 0,
  completion_percentage DECIMAL(5,2) DEFAULT 0.0,
  -- Timing and behavior data
  started_at TIMESTAMP WITH TIME ZONE NOT NULL,
  completed_at TIMESTAMP WITH TIME ZONE NOT NULL,
  total_duration_minutes INTEGER,

  -- Technical metadata
  user_agent TEXT,
  screen_resolution TEXT,
  timezone TEXT,
  browser_language TEXT,

  -- Status tracking
  is_completed BOOLEAN DEFAULT true,
  submission_source TEXT DEFAULT 'web_app',

  -- Timestamps
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes for analytics queries
CREATE INDEX idx_survey_responses_survey_id ON survey_responses(survey_id);
CREATE INDEX idx_survey_responses_completed_at ON survey_responses(completed_at);
CREATE INDEX idx_survey_responses_session_id ON survey_responses(session_id);
CREATE INDEX idx_survey_responses_paths_explored ON survey_responses(paths_explored);

-- JSON indexes for response analysis
CREATE INDEX idx_survey_responses_q1_concern ON survey_responses
USING GIN ((response_data->'completedPaths'->0->'responses'->'Q1'->'selectedAnswers'));

-- Enable Row Level Security
ALTER TABLE survey_responses ENABLE ROW LEVEL SECURITY;

-- Allow anonymous inserts for survey submissions
CREATE POLICY "Allow anonymous survey submissions"
ON survey_responses FOR INSERT
TO anon
WITH CHECK (true);
```

```
-- Prevent updates and deletes for data integrity
CREATE POLICY "No updates or deletes"
ON survey_responses FOR UPDATE
TO anon
USING (false);

CREATE POLICY "No deletes"
ON survey_responses FOR DELETE
TO anon
USING (false);
```

3.2 Supabase Client Setup

```typescript
// lib/supabase.ts
import { createClient } from '@supabase/supabase-js';

const supabaseUrl = import.meta.env.VITE_SUPABASE_URL;
const supabaseAnonKey = import.meta.env.VITE_SUPABASE_ANON_KEY;

export const supabase = createClient(supabaseUrl, supabaseAnonKey, {
  auth: {
    persistSession: false, // No user authentication needed
    autoRefreshToken: false,
  },
});

export type Database = {
  public: {
    Tables: {
      survey_responses: {
        Row: SurveyResponseRow;
        Insert: SurveyResponseInsert;
        Update: SurveyResponseUpdate;
      };
    };
  };
};


interface SurveyResponseRow {
  id: string;
  session_id: string;
  survey_id: string;
  response_data: LocalSurveyData; // Using existing type from Phase I
  paths_explored: number;
  questions_answered: number;
  completion_percentage: number;
  started_at: string;
  completed_at: string;
  total_duration_minutes: number;
  user_agent: string;
  screen_resolution: string;
  timezone: string;
  browser_language: string;
  is_completed: boolean;
  submission_source: string;
  created_at: string;
  updated_at: string;
}
```

4. Data Submission System

4.1 Survey Submission Service

typescript

```typescript
// services/SurveySubmissionService.ts
class SurveySubmissionService {

  static async submitSurvey(localData: LocalSurveyData): Promise<SubmissionResult> {
    try {
      const submissionData = this.prepareSurveyData(localData);

      const { data, error } = await supabase
        .from('survey_responses')
        .insert(submissionData)
        .select('id')
        .single();

      if (error) {
        throw new Error(`Submission failed: ${error.message}`);
      }

      return {
        success: true,
        submissionId: data.id,
        message: 'Survey submitted successfully'
      };

    } catch (error) {
      console.error('Survey submission error:', error);
      return {
        success: false,
        error: error.message,
        fallbackSaved: this.saveFallbackData(localData)
      };
    }
  }

  private static prepareSurveyData(localData: LocalSurveyData): SurveyResponseInsert {
    const startTime = new Date(localData.startedAt);
    const endTime = new Date();
    const durationMinutes = Math.round((endTime.getTime() - startTime.getTime()) / (1000 * 60));

    return {
      session_id: localData.sessionId,
      survey_id: localData.surveyId,
      response_data: localData,
      paths_explored: localData.currentPaths.length + localData.completedPaths.length,
      questions_answered: this.countQuestionsAnswered(localData),
      completion_percentage: this.calculateCompletionPercentage(localData),
      started_at: localData.startedAt,
      completed_at: endTime.toISOString(),
      total_duration_minutes: durationMinutes,
      user_agent: navigator.userAgent,

      screen_resolution: `${screen.width}x${screen.height}`,
      timezone: Intl.DateTimeFormat().resolvedOptions().timeZone,
      browser_language: navigator.language,
      is_completed: localData.isCompleted,
      submission_source: 'web_app'
    };
  }

  private static countQuestionsAnswered(localData: LocalSurveyData): number {
    const allPaths = [...localData.currentPaths, ...localData.completedPaths];
    const uniqueQuestions = new Set();

    allPaths.forEach(path => {
      Object.keys(path.responses).forEach(questionId => {
        uniqueQuestions.add(questionId);
      });
    });

    return uniqueQuestions.size;
  }

  private static calculateCompletionPercentage(localData: LocalSurveyData): number {
    // Calculate based on paths explored vs total possible
    const questionsAnswered = this.countQuestionsAnswered(localData);
    const estimatedTotalQuestions = 47; // Based on climate anxiety survey
    return Math.min((questionsAnswered / estimatedTotalQuestions) * 100, 100);
  }

  private static saveFallbackData(localData: LocalSurveyData): boolean {
    try {
      // Save to a separate localStorage key for failed submissions
      const fallbackKey = 'climate_survey_fallback_submissions';
      const existing = localStorage.getItem(fallbackKey);
      const fallbackList = existing ? JSON.parse(existing) : [];

      fallbackList.push({
        ...localData,
        failedSubmissionAt: new Date().toISOString()

      });

      localStorage.setItem(fallbackKey, JSON.stringify(fallbackList));
      return true;
    } catch (error) {
      console.error('Failed to save fallback data:', error);
      return false;
    }
  }
}

interface SubmissionResult {
  success: boolean;
  submissionId?: string;
  message?: string;
  error?: string;
  fallbackSaved?: boolean;
}
```

4.2 Enhanced Survey Completion Flow

```typescript
// components/SurveyCompletion.tsx
const SurveyCompletion: React.FC = () => {
  const { localSurveyData, clearSurvey } = useSurveyContext();
  const [submissionStatus, setSubmissionStatus] = useState<'idle' | 'submitting' | 'success' | 'error'>('idle');
  const [submissionResult, setSubmissionResult] = useState<SubmissionResult | null>(null);

  const handleSubmitSurvey = async () => {
    if (!localSurveyData) return;

    setSubmissionStatus('submitting');

    const result = await SurveySubmissionService.submitSurvey(localSurveyData);
    setSubmissionResult(result);
    setSubmissionStatus(result.success ? 'success' : 'error');
    if (result.success) {
      // Mark as submitted in localStorage
      LocalStorageManager.markAsSubmitted();
    }
  };

  return (
    <div className="max-w-2xl mx-auto p-6 bg-white rounded-lg shadow-sm">
      <div className="text-center space-y-6">
        <div className="text-green-600">
          <CheckCircleIcon className="w-16 h-16 mx-auto mb-4" />
          <h2 className="text-2xl font-bold text-slate-900">Survey Complete!</h2>
          <p className="text-slate-600 mt-2">
            Thank you for participating in the Climate Anxiety Assessment.
          </p>
        </div>

        <div className="bg-slate-50 rounded-lg p-4 text-left">
          <h3 className="font-semibold text-slate-900 mb-2">Your Survey Summary:</h3>
          <ul className="text-sm text-slate-600 space-y-1">
            <li>• Paths explored: {localSurveyData?.currentPaths.length + localSurveyData?.completedPaths.length}</li
            <li>• Questions answered: {countAnsweredQuestions()}</li>
            <li>• Time taken: {formatDuration()}</li>
          </ul>
        </div>

        {submissionStatus === 'idle' && (
          <div className="space-y-4">
            <p className="text-sm text-slate-600">
              Would you like to submit your responses to contribute to climate anxiety research?
              Your data will be completely anonymous.
            </p>
            <div className="space-x-3">
              <button
                onClick={handleSubmitSurvey}
                className="px-6 py-3 bg-teal-600 text-white rounded-lg hover:bg-teal-700 font-medium"
              >
                Submit Responses
              </button>

              <button
                onClick={clearSurvey}
                className="px-6 py-3 bg-slate-200 text-slate-700 rounded-lg hover:bg-slate-300"
              >
                No Thanks, Clear Data
              </button>
            </div>
          </div>
        )}

        {submissionStatus === 'submitting' && (
          <div className="space-y-3">
            <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-teal-600 mx-auto"></div>
            <p className="text-slate-600">Submitting your responses...</p>
          </div>
        )}

        {submissionStatus === 'success' && (
          <div className="space-y-4">
            <div className="text-green-600">
              <p className="font-medium">✓ Successfully submitted!</p>
              <p className="text-sm text-slate-600 mt-1">
                Submission ID: {submissionResult?.submissionId}
              </p>
            </div>
            <button
              onClick={clearSurvey}
              className="px-6 py-3 bg-teal-600 text-white rounded-lg hover:bg-teal-700"
            >
              Start New Survey
            </button>
          </div>
        )}

        {submissionStatus === 'error' && (
          <div className="space-y-4">
```

```
          <div className="text-amber-600">
            <p className="font-medium">⚠ Submission failed</p>

            <p className="text-sm text-slate-600 mt-1">
              {submissionResult?.error}
            </p>
            {submissionResult?.fallbackSaved && (
              <p className="text-xs text-slate-500 mt-1">
                Your responses have been saved locally for later retry.
              </p>
            )}
          </div>
          <div className="space-x-3">
            <button
              onClick={handleSubmitSurvey}
              className="px-4 py-2 bg-teal-600 text-white rounded-lg hover:bg-teal-700"
            >
              Try Again
            </button>
            <button
              onClick={clearSurvey}
              className="px-4 py-2 bg-slate-200 text-slate-700 rounded-lg hover:bg-slate-300"
            >
              Clear Data
            </button>
          </div>
        </div>
      )}
    </div>
  </div>
 );
};
```

5. Offline/Online Handling

5.1 Network Status Detection

typescript

```typescript
// hooks/useOnlineStatus.ts
export const useOnlineStatus = () => {
  const [isOnline, setIsOnline] = useState(navigator.onLine);

  useEffect(() => {
    const handleOnline = () => setIsOnline(true);
    const handleOffline = () => setIsOnline(false);

    window.addEventListener('online', handleOnline);
    window.addEventListener('offline', handleOffline);

    return () => {
      window.removeEventListener('online', handleOnline);
      window.removeEventListener('offline', handleOffline);
    };
  }, []);

  return isOnline;
};
```

5.2 Automatic Retry Logic

typescript

```typescript
// services/RetryService.ts
class RetryService {

  static async retryFailedSubmissions(): Promise<void> {
    const fallbackKey = 'climate_survey_fallback_submissions';
    const fallbackData = localStorage.getItem(fallbackKey);

    if (!fallbackData) return;

    try {
      const submissions = JSON.parse(fallbackData);
      const successful: string[] = [];

      for (const submission of submissions) {
        const result = await SurveySubmissionService.submitSurvey(submission);
        if (result.success) {
          successful.push(submission.sessionId);
        }
      }

      // Remove successful submissions from fallback storage
      if (successful.length > 0) {
        const remaining = submissions.filter(s => !successful.includes(s.sessionId));
        if (remaining.length === 0) {
          localStorage.removeItem(fallbackKey);
        } else {
          localStorage.setItem(fallbackKey, JSON.stringify(remaining));
        }
      }

    } catch (error) {
      console.error('Retry failed submissions error:', error);
    }
  }
}

// Auto-retry when coming back online
export const useAutoRetry = () => {
  const isOnline = useOnlineStatus();

  useEffect(() => {
    if (isOnline) {
      RetryService.retryFailedSubmissions();
    }
  }, [isOnline]);
};
```

6. Analytics and Insights

6.1 Admin Dashboard Queries

6.1 Admin Dashboard Queries

sql

```sql
-- Top anxiety triggers
SELECT
  jsonb_array_elements_text(

    response_data->'completedPaths'->0->'responses'->'Q3'->'selectedAnswers'
  ) as anxiety_trigger,
  COUNT(*) as frequency
FROM survey_responses
WHERE response_data->'completedPaths'->0->'responses'->>'Q3' IS NOT NULL
GROUP BY anxiety_trigger
ORDER BY frequency DESC;

-- Average completion percentage by start time (hourly analysis)
SELECT
  EXTRACT(hour from started_at) as hour_of_day,
  AVG(completion_percentage) as avg_completion,
  COUNT(*) as response_count
FROM survey_responses
GROUP BY hour_of_day
ORDER BY hour_of_day;

-- Path exploration patterns
SELECT
  paths_explored,
  COUNT(*) as user_count,
  AVG(completion_percentage) as avg_completion,
  AVG(total_duration_minutes) as avg_duration
FROM survey_responses
GROUP BY paths_explored
ORDER BY paths_explored;

-- Climate concern levels distribution
SELECT
  response_data->'completedPaths'->0->'responses'->'Q1'->'selectedAnswers'->0 as concern_level,
  COUNT(*) as frequency,
  ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (), 2) as percentage
```

```
FROM survey_responses
WHERE response_data->'completedPaths'->0->'responses'->>'Q1' IS NOT NULL
GROUP BY concern_level
ORDER BY concern_level::int;
```

6.2 Analytics Dashboard Component

typescript

```typescript
// components/AnalyticsDashboard.tsx (for researchers)
const AnalyticsDashboard: React.FC = () => {
  const [analytics, setAnalytics] = useState<AnalyticsData | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    loadAnalytics();
  }, []);

  const loadAnalytics = async () => {
    try {
      const { data, error } = await supabase
        .from('survey_responses')
        .select('*')
        .order('created_at', { ascending: false });

      if (error) throw error;

      setAnalytics(processAnalyticsData(data));
    } catch (error) {
      console.error('Analytics loading error:', error);
    } finally {
      setLoading(false);
    }
  };

  const processAnalyticsData = (responses: SurveyResponseRow[]): AnalyticsData => {
    return {
      totalResponses: responses.length,
      avgCompletionTime: responses.reduce((sum, r) => sum + r.total_duration_minutes, 0) / responses.length,
      pathExplorationDistribution: calculatePathDistribution(responses),
      topAnxietyTriggers: extractAnxietyTriggers(responses),
      concernLevelDistribution: extractConcernLevels(responses),
      submissionsByDay: groupByDay(responses)
    };
  };

  if (loading) return <div>Loading analytics...</div>;

  return (
    <div className="p-6 space-y-6">
      <h1 className="text-2xl font-bold text-slate-900">Climate Anxiety Survey Analytics</h1>

      <div className="grid grid-cols-1 md:grid-cols-4 gap-4">
        <MetricCard
          title="Total Responses"
          value={analytics?.totalResponses.toLocaleString()}
        />
        <MetricCard
          title="Avg Completion Time"
          value={`${analytics?.avgCompletionTime.toFixed(1)} min`}
        />
        <MetricCard
          title="Avg Paths Explored"
          value={analytics?.avgPathsExplored.toFixed(1)}
        />
        <MetricCard
          title="Completion Rate"
          value={`${analytics?.completionRate.toFixed(1)}%`}
        />
      </div>

      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        <AnxietyTriggersChart data={analytics?.topAnxietyTriggers} />
        <ConcernLevelChart data={analytics?.concernLevelDistribution} />
        <PathExplorationChart data={analytics?.pathExplorationDistribution} />
        <SubmissionTimelineChart data={analytics?.submissionsByDay} />
      </div>
    </div>
  );
};
```

7. Data Export Functionality

7.1 Research Data Export

typescript

```
// services/DataExportService.ts
class DataExportService {

  static async exportToCSV(dateRange?: { start: string, end: string }): Promise<string> {
    let query = supabase
      .from('survey_responses')
      .select('*')
      .eq('is_completed', true);

    if (dateRange) {
      query = query
        .gte('completed_at', dateRange.start)
        .lte('completed_at', dateRange.end);
    }

    const { data, error } = await query;
    if (error) throw error;

    return this.convertToCSV(data);
  }

  static async exportAnonymizedResponses(): Promise<string> {
    const { data, error } = await supabase
      .from('survey_responses')
      .select(`
        session_id,
        survey_id,
        response_data,
        paths_explored,
        questions_answered,
        completion_percentage,
        total_duration_minutes,
        completed_at
      `)
      .eq('is_completed', true);


    if (error) throw error;

    // Remove any potentially identifying information
    const anonymized = data.map(response => ({
      ...response,
      session_id: this.hashSessionId(response.session_id),
      response_data: this.sanitizeResponseData(response.response_data)
    }));

    return this.convertToCSV(anonymized);
  }

  private static convertToCSV(data: any[]): string {
    if (data.length === 0) return '';

    const headers = Object.keys(data[0]);
    const csvContent = [
      headers.join(','),
      ...data.map(row =>
        headers.map(header => {
          const value = row[header];
          if (typeof value === 'object' && value !== null) {
            return `"${JSON.stringify(value).replace(/"/g, '""')}"`;
          }
          return `"${String(value).replace(/"/g, '""')}"`;
        }).join(',')
      )
    ].join('\n');

    return csvContent;
  }
  private static hashSessionId(sessionId: string): string {
    // Simple hash for anonymization
    let hash = 0;
    for (let i = 0; i < sessionId.length; i++) {
      const char = sessionId.charCodeAt(i);
      hash = ((hash << 5) - hash) + char;

      hash = hash & hash; // Convert to 32bit integer
    }
    return `anon_${Math.abs(hash)}`;
  }
}
```

8. Environment Configuration

8.1 Environment Variables

bash

```bash
# .env.local
VITE_SUPABASE_URL=https://your-project.supabase.co
VITE_SUPABASE_ANON_KEY=your-anon-key
VITE_ENABLE_ANALYTICS=true
VITE_ENABLE_DATA_EXPORT=false
```

8.2 Feature Flags

typescript

```typescript
// config/features.ts
export const FEATURES = {
  SUPABASE_SUBMISSION: import.meta.env.VITE_SUPABASE_URL ? true : false,
  ANALYTICS_DASHBOARD: import.meta.env.VITE_ENABLE_ANALYTICS === 'true',
  DATA_EXPORT: import.meta.env.VITE_ENABLE_DATA_EXPORT === 'true',
  AUTO_RETRY: true,
  FALLBACK_STORAGE: true
} as const;
```

9. Migration from Phase I

9.1 Seamless Integration

typescript

```typescript
// Enhanced Survey Context for Phase II
const useSurveyContext = (): SurveyContextType => {
  // All Phase I functionality remains the same
  const phase1Context = usePhase1SurveyContext();

  // Add Phase II enhancements
  const [submissionStatus, setSubmissionStatus] = useState<SubmissionStatus>('idle');
  const isOnline = useOnlineStatus();

  const submitSurvey = async () => {
    if (!FEATURES.SUPABASE_SUBMISSION) {
      console.warn('Supabase submission disabled');
      return;
    }

    if (phase1Context.localSurveyData?.isCompleted) {
      setSubmissionStatus('submitting');
      const result = await SurveySubmissionService.submitSurvey(phase1Context.localSurveyData);
      setSubmissionStatus(result.success ? 'success' : 'error');
    }
  };

  return {
    ...phase1Context,
    submissionStatus,
    submitSurvey,
    isOnline,
    canSubmit: FEATURES.SUPABASE_SUBMISSION && isOnline
  };
};
```

10. Development Phases

Phase II.1: Basic Supabase Integration (Week 1)

- ✅ Supabase project setup and schema creation
- ✅ Basic submission service
- ✅ Enhanced completion screen with submit option

Phase II.2: Robust Data Handling (Week 2)

- ✅ Offline/online detection
- ✅ Fallback storage for failed submissions
- ✅ Automatic retry logic

Phase II.3: Analytics Foundation (Week 3)

- ✅ Basic analytics queries
- ✅ Admin dashboard skeleton
- ✅ Data export functionality

Phase II.4: Production Readiness (Week 4)

- ✅ Error handling and edge cases
- ✅ Performance optimization
- ✅ Security review and testing

11. Testing Strategy

11.1 Integration Tests

- Supabase connectivity and submission
- Offline/online scenarios
- Data consistency between localStorage and Supabase

- Error handling and fallback mechanisms

11.2 Data Integrity Tests

- Anonymous data submission validation
- Response data format consistency
- Analytics query accuracy
- Export functionality validation

12. Security Considerations

12.1 Data Privacy

- No personally identifiable information collected
- Anonymous session identification only
- Row-level security policies in Supabase
- Secure data transmission (HTTPS only)

12.2 Rate Limiting

sql

```sql
-- Implement basic rate limiting in Supabase
CREATE OR REPLACE FUNCTION check_submission_rate()
RETURNS TRIGGER AS $$
BEGIN
  -- Allow maximum 3 submissions per hour from same session
  IF (
    SELECT COUNT(*)
    FROM survey_responses
    WHERE created_at > NOW() - INTERVAL '1 hour'
    AND session_id = NEW.session_id
  ) >= 3 THEN
    RAISE EXCEPTION 'Rate limit exceeded';
  END IF;
  RETURN NEW;
```

```
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER submission_rate_limit
  BEFORE INSERT ON survey_responses
  FOR EACH ROW EXECUTE FUNCTION check_submission_rate();
```

13. Success Criteria for Phase II

- ✅ Seamless integration with Phase I functionality
- ✅ Reliable data submission to Supabase
- ✅ Graceful handling of network issues
- ✅ Zero data loss during submission failures
- ✅ Basic analytics capability for researchers
- ✅ Anonymous, privacy-compliant data collection
- ✅ Production-ready error handling and monitoring

14. Deployment Updates

14.1 Environment-Specific Builds

typescript

```typescript
// Enhanced vite.config.ts
export default defineConfig(({ mode }) => ({
  base: '/climate-anxiety-survey/',
  plugins: [react()],
  define: {
    __SUPABASE_ENABLED__: mode === 'production' ? 'true' : 'false'
  },
  build: {
    outDir: 'dist',
    sourcemap: mode !== 'production'
  }
}));
```