WorkSync - Technical Design Document

1. Architecture Overview

WorkSync is a full-stack web application with a React frontend and Node.js/Express backend.

```
graph TB
    subgraph Frontend["Frontend (React + Vite)"]
        App[App.tsx]
        Components[Components]
        Services[Services Layer]
    end

    subgraph Backend["Backend (Node.js + Express)"]
        Server[server.js]
        DB[(SQLite Database)]
    end

    Services -->|REST API| Server
    Server -->|SQL| DB
```

---

2. Technology Stack

Layer	Technology	Purpose
Frontend	React 18	UI Framework
	TypeScript	Type Safety
	Vite	Build Tool & Dev Server
	Tailwind CSS	Styling
	Recharts	Data Visualization
	Lucide React	Icon Library
Backend	Node.js	Runtime
	Express.js	Web Framework
	SQLite3	Database
	CORS	Cross-Origin Requests
AI	Google Gemini	AI Summaries

---

3. Project Structure

```
WorkSync-main/
├── App.tsx                 # Main application component
├── index.tsx               # React entry point
├── types.ts                # TypeScript type definitions
├── server.js               # Express backend server
├── database.js             # SQLite database setup
├── components/             # React components
│   ├── Homepage.tsx
│   ├── Dashboard.tsx
│   ├── Sidebar.tsx
│   ├── EngagementDetail.tsx
│   ├── TaskTracker.tsx
│   ├── Highlights.tsx
│   ├── InternalProjects.tsx
│   ├── IdeaBoard.tsx
│   ├── CalendarView.tsx
│   ├── UsefulLinks.tsx
│   └── ...modals
├── services/               # API & Storage services
│   ├── apiService.ts
│   ├── storageService.ts
│   └── geminiService.ts
└── vite.config.ts          # Vite configuration
```

4. Data Models

4.1 Engagement

```
interface Engagement {
  id: string;
  engagementNumber: string;
  orgId: string;
  accountName: string;
  name: string;
  status: 'Active' | 'On Hold' | 'Completed' | 'At Risk';
  timeline: TimelineEntry[];
  files: EngagementFile[];
  aiSummary: string | null;
  lastSummaryDate: string | null;
}
```

4.2 Task

```
interface Task {
  id: string;
  content: string;
  isCompleted: boolean;
  type: 'daily' | 'weekly';
  date: string;              // YYYY-MM-DD
  createdAt: string;
  isPriority?: boolean;
  subtasks?: Subtask[];
  engagementId?: string;
  projectId?: string;
}
```

4.3 Internal Project

```
interface InternalProject {
  id: string;
  name: string;
  description: string;
  status: 'Not Started' | 'In Progress' | 'Completed' | 'On Hold';
  startDate: string;
  dueDate: string;
  tasks: ProjectTask[];
  researchNotes: ResearchNote[];
  createdAt: string;
}
```

4.4 Idea

```
interface Idea {
  id: string;
  title: string;
  description: string;
  category: 'Team' | 'Product' | 'Process' | 'General';
  priority: 'Low' | 'Medium' | 'High';
  status: 'New' | 'Planned' | 'In Progress' | 'Implemented' | 'Discarded';
  createdAt: string;
}
```

4.5 Calendar Event

```
interface CalendarEvent {
  id: string;
  title: string;
  description?: string;
  date: string;              // YYYY-MM-DD
  startTime: string;         // HH:mm
  endTime: string;           // HH:mm
  type: 'meeting' | 'work' | 'personal';
  meetingNotes?: string;
  momSent?: boolean;
}
```

4.6 Highlight

```
interface Highlight {
  id: string;
  content: string;
  impact: string;
  date: string;
  needsFollowUp: boolean;
  followUpContext?: string;
  createdAt: string;
}
```

4.7 Useful Link

```
interface UsefulLink {
  id: string;
  title: string;
  url: string;
  category: string;
  description?: string;
  createdAt: string;
}
```

---

5. API Endpoints

5.1 Engagements

Method	Endpoint	Description
GET	`/api/engagements`	Fetch all engagements
POST	`/api/engagements`	Create/Update engagement
DELETE	`/api/engagements/:id`	Delete engagement

5.2 Tasks

Method	Endpoint	Description
GET	`/api/tasks`	Fetch all tasks
POST	`/api/tasks`	Create/Update task
DELETE	`/api/tasks/:id`	Delete task

5.3 Projects

Method	Endpoint	Description
GET	`/api/projects`	Fetch all projects with tasks and notes
POST	`/api/projects`	Create/Update project

DELETE	`/api/projects/:id`	Delete project (cascades)

5.4 Highlights

Method	Endpoint	Description
GET	`/api/highlights`	Fetch all highlights
POST	`/api/highlights`	Create/Update highlight
DELETE	`/api/highlights/:id`	Delete highlight

5.5 Ideas

Method	Endpoint	Description
GET	`/api/ideas`	Fetch all ideas
POST	`/api/ideas`	Create/Update idea
DELETE	`/api/ideas/:id`	Delete idea

5.6 Calendar

Method	Endpoint	Description
GET	`/api/calendar`	Fetch all calendar events
POST	`/api/calendar`	Create/Update event
DELETE	`/api/calendar/:id`	Delete event

5.7 Links

Method	Endpoint	Description
GET	`/api/links`	Fetch all useful links
POST	`/api/links`	Create/Update link
DELETE	`/api/links/:id`	Delete link

5.8 Settings

Method	Endpoint	Description
GET	`/api/settings/:key`	Get setting by key
POST	`/api/settings/:key`	Save setting

---

6. Database Schema

6.1 Tables

```sql
-- Engagements
CREATE TABLE engagements (
  id TEXT PRIMARY KEY,
  engagementNumber TEXT,
  orgId TEXT,
  accountName TEXT,
  name TEXT,
  status TEXT,
  aiSummary TEXT,
  lastSummaryDate TEXT
);

-- Timeline Entries
CREATE TABLE timeline_entries (
  id TEXT PRIMARY KEY,
  engagementId TEXT,
  date TEXT,
  content TEXT,
  type TEXT
);

-- Engagement Files
CREATE TABLE engagement_files (
  id TEXT PRIMARY KEY,
  engagementId TEXT,
  name TEXT,
  type TEXT,
  size INTEGER,
  data TEXT,
  uploadDate TEXT
);

-- Tasks
CREATE TABLE tasks (
  id TEXT PRIMARY KEY,
  content TEXT,
  isCompleted INTEGER,
  type TEXT,
  date TEXT,
  createdAt TEXT,
  isPriority INTEGER,
  subtasks TEXT,
  engagementId TEXT,
  engagementName TEXT,
  projectId TEXT,
  projectName TEXT
);

-- Projects
CREATE TABLE projects (
  id TEXT PRIMARY KEY,
  name TEXT,
  description TEXT,
  status TEXT,
  startDate TEXT,
  dueDate TEXT,
  createdAt TEXT
);

-- Project Tasks
CREATE TABLE project_tasks (
  id TEXT PRIMARY KEY,
  projectId TEXT,
  content TEXT,
  isCompleted INTEGER
);

-- Research Notes
CREATE TABLE research_notes (
  id TEXT PRIMARY KEY,
  projectId TEXT,
  content TEXT,
  date TEXT,
  createdAt TEXT
);

-- Highlights
CREATE TABLE highlights (
  id TEXT PRIMARY KEY,
  content TEXT,
  impact TEXT,
  date TEXT,
  needsFollowUp INTEGER,
  followUpContext TEXT,
  createdAt TEXT
);

-- Ideas
CREATE TABLE ideas (
  id TEXT PRIMARY KEY,
  title TEXT,
  description TEXT,
  category TEXT,
  priority TEXT,
  status TEXT,
  createdAt TEXT
);

-- Calendar Events
CREATE TABLE calendar_events (
  id TEXT PRIMARY KEY,
  title TEXT,
  description TEXT,
  date TEXT,
  startTime TEXT,
  endTime TEXT,
  type TEXT,
  meetingNotes TEXT,
  momSent INTEGER
);

-- Useful Links
CREATE TABLE useful_links (
  id TEXT PRIMARY KEY,
  title TEXT,
  url TEXT,
  category TEXT,
  description TEXT,
  createdAt TEXT
);

-- Settings
CREATE TABLE settings (
  key TEXT PRIMARY KEY,
  value TEXT
);
```

---

7. Frontend Components

7.1 Component Hierarchy

```
App.tsx
├── Sidebar
│   └── WorldClock
├── Homepage
├── Dashboard
│   └── (Engagement Table)
├── EngagementDetail
│   ├── Timeline
│   ├── AddEngagementModal
│   ├── EditEngagementModal
│   └── DeleteConfirmationModal
├── TaskTracker
├── Highlights
├── InternalProjects
├── IdeaBoard
├── CalendarView
│   └── EventDetailModal
└── UsefulLinks
```

7.2 State Management

- **Local State**: React useState for component-specific state
- **Props Drilling**: Data passed from App.tsx to child components

- **Callbacks**: onAdd, onUpdate, onDelete patterns for data mutations

---

8. Services Layer

8.1 API Service (`apiService.ts`)

Handles all HTTP communication with the backend:

- Fetch data from endpoints
- POST/PUT data updates
- DELETE operations
- Error handling

8.2 Storage Service (`storageService.ts`)

Manages data synchronization:

- Wraps API calls with caching logic
- Handles offline fallback (localStorage)
- Provides unified interface for data access

8.3 Gemini Service (`geminiService.ts`)

Integrates with Google Gemini API:

- Generate engagement summaries
- API key management via environment variables

---

9. Configuration

9.1 Environment Variables

```
VITE_GEMINI_API_KEY=<your-api-key>
```

9.2 Server Configuration

- **Frontend Port**: 3000 (Vite dev server)
- **Backend Port**: 3002 (Express server)
- **Database**: `database.db` (SQLite file)

9.3 Build Commands

```
# Development
npm run dev          # Start frontend (Vite)
node server.js       # Start backend (Express)

# Production
npm run build        # Build frontend
npm run preview      # Preview production build
```

---

10. Security Considerations

- CORS enabled for localhost development
- File uploads encoded as Base64 (size limit: 50MB)
- No authentication implemented (single-user application)
- SQLite database stored locally
- Gemini API key stored in environment variable