CalendarAI - Project Part C: Final Documentation

Table of Contents

---

1. Problem Description

1.1 Initial Problem

Modern professionals and individuals struggle with effective time management and maintaining balance across life priorities. Traditional calendar applications (Google Calendar, Outlook, Apple Calendar) excel at displaying scheduled events but fail to provide:

1. **Intelligent Scheduling Assistance**: No AI-powered recommendations based on user priorities
2. **Priority-Based Optimization**: Cannot allocate time based on what matters most to users
3. **Scenario Planning**: No “what-if” functionality to test schedule changes before committing
4. **Natural Language Interaction**: Users must manually create events through forms, lacking conversational interfaces

1.2 Proposed Solution: CalendarAI

CalendarAI is an intelligent calendar management system that combines: - **Priority Management**: Users define and rank life priorities (Work, Family, Health, etc.) - **AI-Powered Scheduling**: Natural language AI assistant that understands scheduling requests - **What-If Scenarios**: Test scheduling changes without committing to them - **Google Calendar Integration**: Seamless sync with existing calendar infrastructure - **Browser Extension**: Access AI assistant directly from Google Calendar

1.3 Key Features

- **Web Application**: Full-featured calendar interface with priority management
- **AI Chatbot**: Natural language event creation and management
- **Browser Extension**: Google Calendar integration with embedded AI assistant
- **What-If Mode**: Scenario planning without permanent changes
- **Priority-Based Recommendations**: AI suggests optimal scheduling based on user priorities

---

2. Database Design

2.1 Entity-Relationship Diagram

The database consists of 7 main tables with the following relationships:

```
Users (1) ──< (N) Priorities
Users (1) ──< (N) Events
Users (1) ──< (N) Messages
Users (1) ──< (N) Scenarios
Users (1) ──< (1) Preferences
Priorities (1) ──< (N) Events
Scenarios (1) ──< (N) ScenarioEvents
Events (1) ──< (N) ScenarioEvents
```

2.2 Database Schema

Users Table - `user_id` (PK, INT, AUTO_INCREMENT) - `username` (VARCHAR(50), UNIQUE, NOT NULL) - `email` (VARCHAR(255), UNIQUE, NOT NULL) - `password_hash` (VARCHAR(255), NOT NULL) - `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)

Priorities Table - `priority_id` (PK, INT, AUTO_INCREMENT) - `user_id` (FK → Users.user_id, NOT NULL) - `name` (VARCHAR(100), NOT NULL) - `rank` (INT, CHECK 1-10) - `hours_per_week` (DECIMAL(5,2)) - `color` (VARCHAR(7))

Events Table - `event_id` (PK, INT, AUTO_INCREMENT) - `user_id` (FK → Users.user_id, NOT NULL) - `priority_id` (FK → Priorities.priority_id, NULLABLE) - `title` (VARCHAR(255), NOT NULL) - `description` (TEXT) - `start_time` (DATETIME, NOT NULL) - `end_time` (DATETIME, NOT NULL) - `color_override` (VARCHAR(7)) - `is_whatif` (BOOLEAN, DEFAULT FALSE) - `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)

Messages Table - `message_id` (PK, INT, AUTO_INCREMENT) - `user_id` (FK → Users.user_id, NOT NULL) - `session_id` (VARCHAR(36), NOT NULL) - `sender_type` (ENUM('user', 'ai'), NOT NULL) - `content` (TEXT, NOT NULL) - `timestamp` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)

Scenarios Table - `scenario_id` (PK, INT, AUTO_INCREMENT) - `user_id` (FK → Users.user_id, NOT NULL) - `name` (VARCHAR(255), NOT NULL) - `description` (TEXT) - `is_applied` (BOOLEAN, DEFAULT FALSE) - `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)

ScenarioEvents Table - `scenario_event_id` (PK, INT, AUTO_INCREMENT) - `scenario_id` (FK → Scenarios.scenario_id, NOT NULL) - `event_id` (FK → Events.event_id, NOT NULL) - `modified_fields` (JSON)

Preferences Table - `preference_id` (PK, INT, AUTO_INCREMENT) - `user_id` (FK → Users.user_id, UNIQUE, NOT NULL) - `work_start_time` (TIME, DEFAULT '09:00:00') - `work_end_time` (TIME, DEFAULT '17:00:00') - `default_ai_mode` (ENUM('whatif', 'agent'), DEFAULT 'whatif') - `theme` (VARCHAR(20), DEFAULT 'light')

2.3 Database Design Rationale

- **Normalization**: Tables are normalized to 3NF to minimize redundancy
- **Foreign Keys**: Proper referential integrity with CASCADE deletes
- **Indexes**: Strategic indexes on frequently queried columns (user_id, start_time, session_id)
- **Constraints**: CHECK constraints ensure data validity (rank 1-10, end_time > start_time)
- **JSON Support**: ScenarioEvents uses JSON for flexible field modifications

Source Files: - `packages/backend/database/schema.sql` - Complete database schema - `packages/backend/database/seed.sql` - Sample data for testing

---

3. System Design

3.1 Architecture Overview

CalendarAI follows a **three-tier architecture**:

1. **Presentation Layer**: React web application + Chrome extension
2. **Application Layer**: Node.js/Express API server with processing modules
3. **Data Layer**: MySQL database

3.2 System Components

3.2.1 DataStore Module

- **Purpose**: Manages all database operations
- **Implementation**: MySQL database with connection pooling
- **Location**: `packages/backend/src/config/database.js`

3.2.2 Data Processing Module

Consists of 7 main components:

1. **AuthManager** (`packages/backend/src/routes/auth.js`)
    - User registration, login, JWT token management
    - Password hashing with bcrypt
2. **EventManager** (`packages/backend/src/routes/events.js`)
    - CRUD operations for calendar events
    - Event validation and conflict detection
3. **PriorityManager** (`packages/backend/src/routes/priorities.js`)
    - Priority CRUD operations
    - Time allocation calculations
4. **AIAgent** (`packages/backend/src/routes/chat.js`, `agent.js`)
    - Natural language processing via Anthropic Claude API
    - Schedule recommendations based on priorities
    - Action extraction and execution
5. **ScenarioManager** (`packages/backend/src/routes/scenarios.js`)
    - What-if scenario creation and management
    - Scenario comparison and application
6. **GoogleCalendarSync** (`packages/backend/src/routes/google.js`)
    - OAuth2 authentication with Google
    - Bidirectional event synchronization
    - Direct event creation/deletion in Google Calendar
7. **ChatProcessor** (`packages/backend/src/routes/chat.js`)
    - Processes AI chat messages
    - Maintains conversation context
    - Extracts actionable commands

3.2.3 User Interface Module

Web Application Pages (`packages/web/src/pages/`): - `HomePage.tsx` - Landing page - `LoginPage.tsx` - User authentication - `RegisterPage.tsx` - User registration - `DashboardPage.tsx` - Main hub with statistics - `CalendarPage.tsx` - Calendar view with event management - `PrioritiesPage.tsx` - Priority management interface - `WhatIfPage.tsx` - Scenario listing and management - `ScenarioEditorPage.tsx` - Scenario editing interface - `AIAgentPage.tsx` - AI schedule optimization - `AskAgentPage.tsx` - Chat interface for AI assistance

Browser Extension (`packages/extension/`): - `popup.html/js` - Extension popup for authentication - `content.js` - Injects chatbot into Google Calendar - `chatbot.html/js/css` - AI chatbot interface - `background.js` - Service worker for extension lifecycle

3.3 Data Flow

```
User Action (UI/Extension)
    ↓
API Request (HTTP/REST)
    ↓
Processing Module (Business Logic)
    ↓
DataStore (MySQL Database)
    ↓
Response (JSON)
    ↓
UI Update
```

3.4 System Integration

The system integrates three main components: 1. **Web Application**: Standalone React app for full calendar management 2. **Browser Extension**: Google Calendar integration with embedded AI 3. **Backend API**: Unified REST API serving both interfaces

Source Files: - `packages/backend/src/server.js` - Main server configuration - `packages/web/src/App.tsx` - React router configuration - `packages/extension/manifest.json` - Extension configuration

---

4. Implementation

4.1 Technology Stack

Backend: - Node.js 16+ - Express.js 4.18.2 - MySQL2 3.6.5 - JWT (jsonwebtoken 9.0.2) - Anthropic Claude API (@anthropic-ai/sdk 0.71.0) - Google APIs (googleapis 166.0.0)

Frontend (Web): - React 18.3.1 - TypeScript 4.9.5 - Styled Components 6.1.19 - React Router 6.30.1

Frontend (Extension): - Vanilla JavaScript - Chrome Extension Manifest V3

Database: - MySQL 8.0+

4.2 Key Implementation Details

4.2.1 Authentication System

File: `packages/backend/src/routes/auth.js` - JWT-based authentication - Password hashing with bcrypt (10 rounds) - Token expiration: 24 hours - Protected routes via middleware

4.2.2 AI Integration

File: `packages/backend/src/routes/chat.js` - Anthropic Claude Sonnet 4 API - System prompts from `agent.yaml` configuration - Action extraction from natural language - Context-aware responses using user priorities and events

4.2.3 Google Calendar Integration

File: `packages/backend/src/routes/google.js` - OAuth2 flow for Google authentication - Event creation with recurrence support - Color mapping from hex to Google Calendar color IDs - Timezone-aware event handling

4.2.4 Database Connection

File: `packages/backend/src/config/database.js` - Connection pooling for efficiency - Environment-based configuration - Error handling and reconnection logic

4.3 Source Code Organization

```
packages/
├── backend/
│   ├── src/
│   │   ├── config/          # Database configuration
│   │   ├── middleware/      # Authentication middleware
│   │   ├── routes/          # API route handlers
│   │   └── server.js        # Main server file
│   ├── database/
│   │   ├── schema.sql       # Database schema
│   │   └── seed.sql         # Sample data
│   └── agent.yaml           # AI agent configuration
├── web/
│   └── src/
│       ├── pages/           # React page components
│       ├── components/      # Reusable components
│       └── context/         # React context providers
└── extension/
    ├── manifest.json        # Extension manifest
    ├── popup.html/js        # Extension popup
    ├── content.js           # Content script
    └── chatbot.html/js/css  # Chatbot interface
```

Complete source code is available in the repository: - All backend routes: `packages/backend/src/routes/` - All frontend pages: `packages/web/src/pages/` - Extension files: `packages/extension/` - Database schema: `packages/backend/database/schema.sql`

---

5. Usability Design

5.1 User Interface Principles

1. **Consistency**: Uniform design language across web app and extension
2. **Accessibility**: Keyboard navigation, screen reader support, color contrast
3. **Responsiveness**: Works on desktop and mobile devices
4. **Feedback**: Clear success/error messages for all user actions
5. **Progressive Disclosure**: Advanced features (scenarios, AI agent) available but not overwhelming

5.2 Web Application Design

Home Page: - Clear value proposition - Feature highlights with icons - Prominent login/register buttons - Dark/light theme toggle

Dashboard: - Overview statistics (events, priorities, weekly progress) - Quick action cards for common tasks - Visual priority distribution - Chat sidebar for quick AI access

Calendar Page: - Month/week/day view options - Color-coded events by priority - Drag-and-drop event creation - Click to view/edit event details

Priorities Page: - Drag-and-drop ranking interface - Visual time allocation charts - Color picker for priority customization - Target hours per week tracking

5.3 Browser Extension Design

Popup Interface: - Simple authentication form - Token storage indicator - Google Calendar sync toggle - Status messages

Chatbot Interface: - Floating sidebar in Google Calendar - Natural language input - Color picker for event customization - Message history with markdown support - Loading indicators and error handling

5.4 User Experience Flow

1. **Registration/Login**: Simple form with validation
2. **Onboarding**: Dashboard shows quick actions to get started
3. **Priority Setup**: Guided interface to create and rank priorities
4. **Event Creation**: Multiple methods (manual, AI chat, import)
5. **AI Interaction**: Natural language, no technical knowledge required
6. **Scenario Planning**: Visual comparison of what-if scenarios

Source Files: - `packages/web/src/pages/` - All UI pages - `packages/web/src/components/` - Reusable UI components - `packages/extension/chatbot.css` - Extension styling

---

6. Testing

6.1 Testing Methodology

Testing was performed using: - **Manual Testing**: Direct API calls via Postman/curl - **Integration Testing**: Full user flows through web application - **Extension Testing**: Chrome extension in Google Calendar environment

6.2 Retrieval Tasks (3 Different Tasks)

Task 1: Retrieve All Events for a User

Purpose: Test basic SELECT query with JOIN and filtering

API Endpoint: `GET /api/events` **Authentication**: Required (JWT token)

Test Steps: 1. Login to get JWT token 2. Send GET request to `/api/events` with Authorization header 3. Verify response contains array of events 4. Verify events are filtered by user_id 5. Verify events include priority information (JOIN with priorities table)

Expected Result:

```
[
  {
    "event_id": 1,
    "user_id": 1,
    "title": "Team Meeting",
    "start_time": "2025-12-10T14:00:00.000Z",
    "end_time": "2025-12-10T15:00:00.000Z",
    "priority_name": "Work",
    "priority_color": "#3b82f6"
  },
  ...
]
```

Source Code: `packages/backend/src/routes/events.js` (lines 7-22)

Task 2: Retrieve User Priorities with Rankings

Purpose: Test SELECT with ORDER BY and user filtering

API Endpoint: `GET /api/priorities` **Authentication**: Required (JWT token)

Test Steps: 1. Login to get JWT token 2. Send GET request to `/api/priorities` with Authorization header 3. Verify response contains array of priorities 4. Verify priorities are ordered by rank (ascending) 5. Verify priorities are filtered by user_id

Expected Result:

```
[
  {
```

```
    "priority_id": 1,
    "user_id": 1,
    "name": "Work",
    "rank": 1,
    "hours_per_week": 40.00,
    "color": "#3b82f6"
  },
  {
    "priority_id": 2,
    "user_id": 1,
    "name": "Family",
    "rank": 2,
    "hours_per_week": 20.00,
    "color": "#ec4899"
  },
  ...
]
```

Source Code: `packages/backend/src/routes/priorities.js` (lines 7-18)

Task 3: Retrieve Chat History for a Session

Purpose: Test SELECT with session filtering and timestamp ordering

API Endpoint: `GET /api/chat/history/:sessionId` **Authentication**: Required (JWT token)

Test Steps: 1. Login to get JWT token 2. Create a chat session by sending a message to `/api/chat/message` 3. Note the session_id from the response 4. Send GET request to `/api/chat/history/{sessionId}` with Authorization header 5. Verify response contains array of messages 6. Verify messages are filtered by session_id and user_id 7. Verify messages are ordered by timestamp (ascending)

Expected Result:

```
[
  {
    "role": "user",
    "content": "What's on my calendar today?",
    "timestamp": "2025-12-09T10:00:00.000Z"
  },
  {
    "role": "assistant",
    "content": "You have 3 events scheduled for today...",
    "timestamp": "2025-12-09T10:00:05.000Z"
  },
  ...
]
```

Source Code: `packages/backend/src/routes/chat.js` (lines 327-340)

6.3 Update Tasks (3 Different Operations)

Task 1: Insert a New Event (INSERT Operation)

Purpose: Test INSERT operation with foreign key relationships

API Endpoint: `POST /api/events` **Authentication**: Required (JWT token)

Test Steps: 1. Login to get JWT token 2. Send POST request to `/api/events` with event data:

```
{
  "title": "Test Meeting",
  "description": "Testing event creation",
  "start_time": "2025-12-15T14:00:00",
  "end_time": "2025-12-15T15:00:00",
  "priority_id": 1
}
```

3. Verify response contains new event_id
4. Verify event was inserted with correct user_id (from token)
5. Query database to confirm event exists

Expected Result:

```
{
  "event_id": 123,
  "message": "Event created successfully"
}
```

Database Verification:

```
SELECT * FROM events WHERE event_id = 123;
-- Should return the newly created event
```

Source Code: `packages/backend/src/routes/events.js` (lines 41-57)

Task 2: Update an Existing Priority (UPDATE Operation)

Purpose: Test UPDATE operation with WHERE clause filtering

API Endpoint: `PUT /api/priorities/:id` **Authentication**: Required (JWT token)

Test Steps: 1. Login to get JWT token 2. Get existing priority_id (e.g., 1) 3. Send PUT request to `/api/priorities/1` with updated data:

```
{
  "name": "Work - Updated",
  "rank": 1,
  "hours_per_week": 45.00,
  "color": "#2563eb"
}
```

4. Verify response indicates success
5. Query database to confirm priority was updated
6. Verify only the specified priority was updated (not others)

Expected Result:

```
{
  "message": "Priority updated successfully"
}
```

Database Verification:

```
SELECT * FROM priorities WHERE priority_id = 1;
-- Should show updated values: name='Work - Updated', hours_per_week=45.00
```

Source Code: `packages/backend/src/routes/priorities.js` (lines 39-50)

Task 3: Delete an Event (DELETE Operation)

Purpose: Test DELETE operation with user verification

API Endpoint: `DELETE /api/events/:id` **Authentication**: Required (JWT token)

Test Steps: 1. Login to get JWT token 2. Create a test event first (using POST /api/events) 3. Note the event_id (e.g., 123) 4. Send DELETE request to `/api/events/123` with Authorization header 5. Verify response indicates success 6. Query database to confirm event was deleted 7. Verify event cannot be retrieved after deletion

Expected Result:

```
{
  "message": "Event deleted successfully"
}
```

Database Verification:

```
SELECT * FROM events WHERE event_id = 123;
-- Should return empty result set
```

Source Code: `packages/backend/src/routes/events.js` (lines 76-83)

6.4 Additional Testing

Integration Testing: - Full user registration → login → create priority → create event flow - AI chat → event creation → Google Calendar sync flow - What-if scenario creation → modification → application flow

Error Handling Testing: - Invalid authentication tokens - Missing required fields - Foreign key constraint violations - Duplicate username/email registration

Source Files for Testing: - All route handlers in `packages/backend/src/routes/` - Test API page: `packages/web/src/pages/TestAPIPage.tsx` - Extension testing guide: `packages/extension/TESTING.md` - Detailed testing guide: `TESTING_GUIDE.md` (included in project)

Testing Documentation: A comprehensive testing guide with step-by-step instructions, SQL queries, and verification steps is available in `TESTING_GUIDE.md` in the project root.

---

7. Deployment

7.1 Deployment Information

Deployment Status: ✅ System is deployed and accessible

Access Information: - **Web Application URL**: `http://localhost:3000` (development) - **API Base URL**: `http://localhost:3001/api` (development) - **Extension**: Load unpacked from `packages/extension/` directory in Chrome/Edge

Login Credentials for Testing: - **Username**: `testuser` - **Email**: `test@calendarai.com` - **Password**: `testpassword123`

Note: These are test credentials. For production deployment, these should be changed. The system supports user registration for creating new accounts.

Alternative: You can register a new account at `http://localhost:3000/register`

7.2 Deployment Instructions

Local Development Setup

1. **Prerequisites**:

   ```
   - Node.js >= 16.0.0
   - MySQL >= 8.0
   - npm >= 8.0.0
   ```

2. **Database Setup**:

   ```
   cd packages/backend/database
   mysql -u root -p < schema.sql
   mysql -u root -p < seed.sql
   ```

3. **Environment Configuration**:

   ```
   cd packages/backend
   cp .env.example .env
   # Edit .env with your database credentials and API keys
   ```

4. **Install Dependencies**:

```
npm install
```

5. **Start Backend**:

```
npm run dev:backend
# Server runs on http://localhost:3001
```

6. **Start Web Application**:

```
npm run dev:web
# App runs on http://localhost:3000
```

7. **Load Extension**:
    - Open Chrome/Edge
    - Go to `chrome://extensions/`
    - Enable “Developer mode”
    - Click “Load unpacked”
    - Select `packages/extension/` directory

Production Deployment

For production deployment: 1. Set up MySQL database on production server 2. Configure environment variables 3. Build React app: `npm run build:web` 4. Deploy backend to Node.js hosting (Heroku, AWS, etc.) 5. Serve built React app via static hosting or CDN 6. Update extension manifest with production API URL

7.3 System Accessibility

Web Application: - Accessible via web browser (Chrome, Firefox, Safari, Edge) - Responsive design works on desktop and mobile - No special software required

Browser Extension: - Chrome/Edge compatible (Manifest V3) - Installs via Chrome Web Store or developer mode - Works directly within Google Calendar interface

API Endpoints: - RESTful API accessible via HTTP/HTTPS - CORS configured for web app and extension origins - Authentication required for protected endpoints

Source Files: - `SETUP.md` - Detailed setup instructions - `README.md` - Quick start guide - `packages/backend/README.md` - Backend-specific setup

---

8. Contributions

8.1 Project Development

This project was developed as a comprehensive database application for CSCI 4333 by a team of four developers. The development process involved collaborative work across multiple phases:

Phase 1: Database Design (Led by Viviana Ayala) - Team designed normalized database schema through collaborative sessions - Created ER diagrams and relationship documentation - Implemented schema with proper constraints and indexes - Team reviewed and refined database design

Phase 2: Backend Development (Led by Viviana Ayala & Troy Rodriguez) - Implemented RESTful API with Express.js (collaborative development) - Created authentication system with JWT (Viviana Ayala) - Integrated Anthropic Claude API for AI functionality (Viviana Ayala) - Implemented Google Calendar OAuth and sync (Viviana Ayala) - Developed scenarios and messages routes Troy Rodriguez - Implemented error handling and middleware Troy Rodriguez

Phase 3: Frontend Development (Led by Team Carolina Espiritu, with Viviana Ayala) - Built React web application with TypeScript Carolina Espiritu - Created responsive UI with styled-components Carolina Espiritu - Implemented calendar visualization and event management (Viviana Ayala) - Developed priority management interface (Viviana Ayala) - Built What-If and AI Agent pages Carolina Espiritu - Developed reusable components Carolina Espiritu

Phase 4: Browser Extension Andres Montemayor - Created Chrome extension with Manifest V3 Andres Montemayor - Implemented content script injection into Google Calendar Andres Montemayor - Built AI chatbot interface for natural language interaction Andres Montemayor - Integrated color picker for event customization Andres Montemayor - Implemented extension-backend communication Andres Montemayor

Phase 5: Integration and Testing (All Team Members) - Integrated all components (collaborative effort) - Performed comprehensive testing - Documented all functionality (Viviana Ayala, with team input) - Prepared deployment (Viviana Ayala)

8.2 Individual Contributions

Viviana Ayala - Project Lead & Full-Stack Developer - Overall project architecture and design - Database schema design and implementation - Backend API development (authentication, events, priorities routes) - Frontend web application development (Calendar, Priorities, Dashboard pages) - AI integration and natural language processing (chat.js, agent.yaml) - Google Calendar integration (OAuth, sync functionality) - Project documentation (Part B and Part C) - Deployment setup and configuration

Troy Rodriguez - Backend Developer - Backend API development (scenarios, messages routes) - Database query optimization and indexing - API endpoint testing and validation - Error handling and middleware implementation - Testing documentation and test case development

Carolina Espiritu - Frontend Developer - Frontend web application development (What-If, AI Agent, Ask Agent pages) - React component development (ChatSidebar, GoogleCalendarSync) - UI/UX design and styling (styled-components, theme context) - User interface testing and responsive design - Frontend state management and routing

Andres Montemayor - Extension Developer - Browser extension development (manifest, popup, content scripts) - Chatbot interface development (chatbot.html, chatbot.js, chatbot.css) - Extension-backend communication (postMessage API) - Color picker implementation - Extension testing and Chrome Web Store preparation

Collaborative Work: - Code reviews and peer programming sessions - Database schema review and optimization - API endpoint design discussions - UI/UX design decisions - Testing strategy and implementation - Documentation review and refinement

Source Code Attribution: - All source code files in `packages/backend/src/`, `packages/web/src/`, and `packages/extension/` were collaboratively developed for this project - Third-party libraries are properly attributed in `package.json` files - AI agent configuration (`agent.yaml`) was collaboratively developed for CalendarAI

---

9. Conclusions

9.1 Project Summary

CalendarAI successfully implements a comprehensive calendar management system that addresses the identified problem of intelligent scheduling assistance. The system provides:

1. **Functional Database System**: 7 normalized tables with proper relationships, constraints, and indexes
2. **Complete CRUD Operations**: Full create, read, update, delete functionality for all entities
3. **AI-Powered Features**: Natural language event management via Anthropic Claude API
4. **Multiple Interfaces**: Web application and browser extension for different use cases
5. **Google Calendar Integration**: Seamless synchronization with existing calendar infrastructure

9.2 Achievements

✅ **Database Design**: Well-normalized schema with proper foreign keys and constraints ✅ **System Architecture**: Clean three-tier architecture with separation of concerns ✅ **User**

Interface: Intuitive and responsive design across web and extension ✅ **AI Integration**: Successful natural language processing for calendar management ✅ **Testing**: Comprehensive testing of retrieval and update operations ✅ **Documentation**: Complete documentation covering all aspects of the system

9.3 Challenges and Solutions

Challenge 1: Integrating AI responses with database operations - **Solution**: Implemented action extraction from AI responses, parsing JSON actions and executing them through appropriate managers

Challenge 2: Timezone handling for events - **Solution**: Implemented timezone-aware date calculations, storing times in user's local timezone and converting for Google Calendar API

Challenge 3: Browser extension communication - **Solution**: Used `postMessage` API for secure communication between extension iframe and Google Calendar page

Challenge 4: Preventing accidental event creation - **Solution**: Refined AI prompts to only create events when explicitly requested, with clear examples of when NOT to add events

9.4 Future Enhancements

Potential improvements for future development: 1. **Mobile Applications**: Native iOS and Android apps 2. **Advanced Analytics**: Time allocation reports and insights 3. **Collaborative Features**: Shared calendars and group scheduling 4. **More AI Capabilities**: Proactive scheduling suggestions, conflict resolution 5. **Integration Expansion**: Support for Outlook, Apple Calendar, and other platforms 6. **Performance Optimization**: Caching, database query optimization 7. **Enhanced Testing**: Automated unit tests, integration tests, E2E tests

9.5 Lessons Learned

1. **Database Design First**: Starting with a well-designed schema made implementation smoother
2. **Modular Architecture**: Separating concerns (routes, managers, database) improved maintainability
3. **User Experience Matters**: Natural language interface significantly improves usability
4. **Testing is Critical**: Comprehensive testing caught many issues before deployment
5. **Documentation is Essential**: Good documentation helps with development and future maintenance

---

10. References

10.1 Technical Documentation

1. **Express.js Documentation**
    - URL: https://expressjs.com/
    - Used for: RESTful API framework and routing
2. **React Documentation**
    - URL: https://react.dev/
    - Used for: Frontend web application framework
3. **MySQL Documentation**
    - URL: https://dev.mysql.com/doc/
    - Used for: Database schema design and SQL queries
4. **Anthropic Claude API Documentation**
    - URL: https://docs.anthropic.com/
    - Used for: AI-powered natural language processing
5. **Google Calendar API Documentation**
    - URL: https://developers.google.com/calendar
    - Used for: Google Calendar integration and OAuth2
6. **Chrome Extension Documentation**
    - URL: https://developer.chrome.com/docs/extensions/
    - Used for: Browser extension development (Manifest V3)

10.2 Libraries and Frameworks

1. **Node.js**
    - Version: 16.0.0+
    - URL: https://nodejs.org/
    - License: MIT
2. **Express.js**
    - Version: 4.18.2
    - URL: https://expressjs.com/
    - License: MIT
3. **React**
    - Version: 18.3.1
    - URL: https://react.dev/
    - License: MIT
4. **MySQL2**
    - Version: 3.6.5
    - URL: https://github.com/sidorares/node-mysql2
    - License: MIT

5. **jsonwebtoken**
    - Version: 9.0.2
    - URL: https://github.com/auth0/node-jsonwebtoken
    - License: MIT
6. **@anthropic-ai/sdk**
    - Version: 0.71.0
    - URL: https://github.com/anthropics/anthropic-sdk-typescript
    - License: Apache-2.0
7. **googleapis**
    - Version: 166.0.0
    - URL: https://github.com/googleapis/google-api-nodejs-client
    - License: Apache-2.0
8. **styled-components**
    - Version: 6.1.19
    - URL: https://styled-components.com/
    - License: MIT

10.3 Design Inspiration

1. **Cursor IDE** - "Ask Mode" concept for what-if scenarios
2. **Notion AI** - Natural language interface design patterns
3. **Google Calendar** - Calendar visualization and event management UI

10.4 Academic Resources

1. **Database Design Principles** - Course materials from CSCI 4333
2. **Software Engineering Best Practices** - Three-tier architecture patterns
3. **RESTful API Design** - HTTP methods and status codes

Appendix: Source Code Locations

All source code is available in the repository with the following structure:

Backend Routes (API Endpoints): - Authentication: `packages/backend/src/routes/auth.js` - Events: `packages/backend/src/routes/events.js` - Priorities: `packages/backend/src/routes/priorities.js` - AI Chat: `packages/backend/src/routes/chat.js` - Scenarios: `packages/backend/src/routes/scenarios.js` - Google Calendar: `packages/backend/src/routes/google.js` - Messages: `packages/backend/src/routes/messages.js` - AI Agent: `packages/backend/src/routes/agent.js`

Frontend Pages: - All pages: `packages/web/src/pages/` - Components: `packages/web/src/components/`

Extension Files: - All extension code: `packages/extension/`

Database: - Schema: `packages/backend/database/schema.sql` - Seed data: `packages/backend/database/seed.sql`

Configuration: - AI Agent: `packages/backend/agent.yaml` - Server: `packages/backend/src/server.js` - Database config: `packages/backend/src/config/database.js`

End of Document