SOFTWARE PROJECT FINAL REPORT

Super Scheduler

Team Members

December 3, 2025

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose and Scope

The Super Scheduler project is a Progressive Web Application (PWA) designed to provide comprehensive calendar and task management functionality. The purpose of this application is to offer users an intuitive, accessible, and offline-capable scheduling solution that helps manage events, tasks, and daily activities efficiently.

The scope includes:

- Event and task management with categorization
- Calendar visualization with month view
- Offline data persistence using IndexedDB and LocalStorage
- Reminder and notification systems
- Search and filter capabilities
- Responsive design for mobile and desktop platforms
- PWA capabilities for installation and offline usage

1.2 Product Overview

Super Scheduler is a web-based calendar and task management application built using modern web technologies. The application provides users with:

Key Capabilities:

- **Calendar Dashboard:** Interactive monthly calendar view with drag-and-drop event management
- **Task Management:** Comprehensive task tracking with priority levels, categories, and status management
- **Event Creation:** Quick event creation with conflict detection and time validation
- **Notifications:** Browser notifications and sound alerts for upcoming events
- **Search & Filter:** Advanced filtering by category, priority, status, and date range
- **Settings & Preferences:** Customizable time formats, notification preferences, and data management
- **Data Persistence:** Local storage with export/import capabilities (JSON, CSV, iCal formats)
- **PWA Features:** Installable, offline-capable, and mobile-responsive

Scenarios for Using the Product:

- Personal scheduling and daily task organization

- Work project tracking with categorized tasks
- Health appointment management
- Social event planning
- Academic schedule management
- Deadline tracking for multiple projects

1.3 Structure of the Document

This document is organized into seven main sections:

1. **Introduction:** Project overview and document structure
2. **Project Management Plan:** Organization, lifecycle model, and resource requirements
3. **Requirement Specifications:** Use cases, stakeholders, and functional requirements
4. **Architecture:** System architecture and technology stack
5. **Design:** UI design, component design, and database design
6. **Test Management:** Test cases, techniques, and results
7. **Conclusions:** Project outcomes, lessons learned, and future development

1.4 Terms, Acronyms, and Abbreviations

Term	Definition
PWA	Progressive Web Application
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
IndexedDB	Browser-based NoSQL database
LocalStorage	Browser key-value storage API
UI	User Interface
UX	User Experience
API	Application Programming Interface
iCal	Internet Calendaring and Scheduling Core Object Specification
CSV	Comma-Separated Values
JSON	JavaScript Object Notation
SW	Service Worker

Table 1: Terms, Acronyms, and Abbreviations

2 Project Management Plan

2.1 Project Organization

The Super Scheduler project follows a small team structure with distributed responsibilities:

Team Structure:

- **Project Lead:** Overall project coordination and decision-making
- **Developer:** UI/UX implementation and responsive design
- **Backend Developer:** Data management, storage, and API integration

2.2 Lifecycle Model Used

The project adopts an **Iterative and Incremental Development** model with characteristics of Agile methodology:

1. **Requirements Gathering:** Initial requirements and feature prioritization
2. **Design Phase:** UI/UX mockups, architecture design, and database schema
3. **Implementation:** Feature development in sprints
4. **Testing:** Unit testing, integration testing, and user acceptance testing
5. **Deployment:** Progressive web app deployment and monitoring
6. **Maintenance:** Bug fixes, updates, and feature enhancements

The iterative approach allows for continuous feedback and refinement throughout the development process.

2.3 Risk Analysis

Risk	Probability	Impact	Mitigation Strategy
Browser Compatibility Issues	Medium	High	Cross-browser testing, use of polyfills, progressive enhancement
Data Loss	Low	High	Implement export/import features, regular backups, robust error handling
Performance Issues	Medium	Medium	Code optimization, lazy loading, efficient data structures
Security Vulnerabilities	Low	High	Input validation, XSS prevention, secure data handling
User Adoption	Medium	Medium	Intuitive UI, comprehensive documentation, user feedback integration
Offline Functionality Failures	Medium	Medium	Service worker testing, fallback mechanisms, cache management

Table 2: Risk Analysis

2.4 Hardware and Software Resource Requirements

Development Environment:

- Operating System: Windows, macOS, or Linux
- Code Editor: Visual Studio Code
- Browser: Chrome, Firefox, Safari, Edge (latest versions)
- Version Control: Git
- Node.js: v12.x or higher
- Package Manager: npm or yarn

Production Environment:

- Web Server: Any static file hosting service
- Browser Support: Modern browsers with ES6+ support
- Storage: LocalStorage and IndexedDB support
- Network: Optional (offline-capable)

Software Dependencies:

- Tailwind CSS v3.4.17
- Font Awesome v6.4.0
- Various Tailwind plugins (@tailwindcss/forms, tailwindcss-animate, etc.)

2.5 Deliverables and Schedule

Deliverable	**Deadline**	**Status**
Requirements Document	Week 1	Completed
UI/UX Design Mockups	Week 2	Completed
Database Schema	Week 2	Completed
Calendar Dashboard	Week 3-4	Completed
Task Management Module	Week 4-5	Completed
Search & Filter Functionality	Week 5-6	Completed
Settings & Preferences	Week 6	Completed
PWA Implementation	Week 7	Completed
Testing & QA	Week 8	Completed
Documentation	Week 9	Completed
Final Deployment	Week 10	Completed

Table 3: Project Deliverables and Schedule

3 Requirement Specifications

3.1 Stakeholders for the System

1. **End Users:**
    - Individuals seeking personal task and calendar management
    - Students managing academic schedules
    - Professionals organizing work and personal commitments
2. **Development Team:**
    - Frontend developers
    - Backend developers
    - UI/UX designers
    - QA engineers
3. **Project Sponsors:**
    - Project stakeholders providing resources
    - Decision-makers guiding project direction
4. **System Administrators:**
    - Maintaining deployment infrastructure
    - Monitoring application performance

3.2 Use Cases

3.2.1 Graphic Use Case Model

The following use cases represent the core functionality of Super Scheduler:

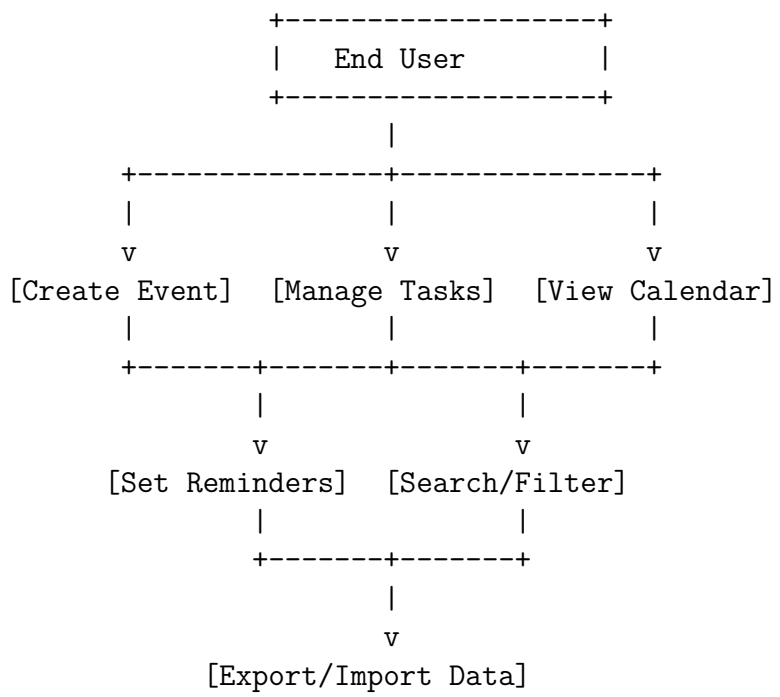

Figure 1: Use Case Diagram

3.2.2 Textual Description for Each Use Case

UC-01: Create Event

- **Actor:** End User
- **Preconditions:** User is on calendar dashboard
- **Main Flow:**
  1. User enters event title, date, time, and category
  2. System validates input and checks for conflicts
  3. System creates event and stores in database
  4. System displays event on calendar
  5. System shows reminder on dashboard
- **Postconditions:** Event is created and visible on calendar
- **Alternative Flows:** Time conflict detected, validation failure

UC-02: Manage Tasks

- **Actor:** End User
- **Preconditions:** User navigates to task management page
- **Main Flow:**
  1. User creates task with title, due date, priority, and category

2. User can edit, delete, or mark tasks as complete
3. System updates task status and statistics
4. System persists changes to local storage

- **Postconditions:** Tasks are managed and synchronized
- **Alternative Flows:** Invalid date, empty title

UC-03: View Calendar

- **Actor:** End User
- **Preconditions:** User accesses calendar dashboard
- **Main Flow:**

1. System loads current month view
2. User can navigate between months
3. User can view events on specific dates
4. User can drag and drop events to reschedule

- **Postconditions:** Calendar view is displayed
- **Alternative Flows:** No events to display

UC-04: Set Reminders

- **Actor:** End User
- **Preconditions:** User is in settings page
- **Main Flow:**

1. User selects reminder time for events/tasks
2. User enables/disables browser notifications
3. User configures sound preferences
4. System saves notification preferences

- **Postconditions:** Reminder settings are applied
- **Alternative Flows:** Browser notifications denied

UC-05: Search and Filter

- **Actor:** End User
- **Preconditions:** User navigates to search page
- **Main Flow:**

1. User enters search criteria
2. User applies filters (category, priority, status)

3. System displays filtered results
4. User can view or edit filtered items

- **Postconditions:** Filtered results are displayed
- **Alternative Flows:** No results found

UC-06: Export and Import Data

- **Actor:** End User
- **Preconditions:** User is in data management section
- **Main Flow:**

1. User selects export format (JSON, CSV, iCal)
2. System generates file and initiates download
3. User can import data from backup file
4. System validates and merges imported data

- **Postconditions:** Data is exported or imported
- **Alternative Flows:** Invalid file format

3.3 Rationale for Use Case Model

The use case model is designed around user-centric workflows that reflect real-world scheduling needs. Each use case focuses on a specific user goal:

- **Create Event** and **Manage Tasks** address the core scheduling functionality
- **View Calendar** provides visualization and navigation
- **Set Reminders** ensures users don't miss important events
- **Search/Filter** enables efficient information retrieval
- **Export/Import** ensures data portability and backup

This model covers the complete lifecycle of scheduling: creation, viewing, management, notification, and data persistence.

3.4 Non-functional Requirements

1. **Performance Requirements:**
   - Calendar rendering should complete within 200ms
   - Event creation should respond within 100ms
   - Search results should display within 300ms
2. **Usability Requirements:**

- Intuitive interface requiring minimal training
- Responsive design for mobile and desktop

3. **Reliability Requirements:**
   - Data persistence across browser sessions
   - Graceful degradation when offline
   - Error handling with user-friendly messages
4. **Availability Requirements:**
   - Offline functionality for core features
   - Service worker caching for fast load times
5. **Security Requirements:**
   - Client-side data storage (no server transmission)
   - Input validation to prevent XSS attacks
   - Secure handling of user preferences
6. **Portability Requirements:**
   - Cross-browser compatibility (Chrome, Firefox, Safari, Edge)
   - Mobile browser support (iOS, Android)
   - Export to standard formats (iCal, CSV)
7. **Maintainability Requirements:**
   - Modular code structure
   - Comprehensive documentation
   - Version control with Git

4 Architecture

4.1 Architectural Style(s) Used

Super Scheduler employs a **Client-Side Architecture** with the following patterns:

1. **Progressive Web App (PWA) Architecture:**
   - Service Worker for offline capabilities
   - Cache-first strategy for static assets
   - Manifest file for installability
2. **Single Page Application (SPA) Pattern:**
   - Client-side routing between pages
   - Dynamic content loading
   - State management via LocalStorage and IndexedDB
3. **Model-View Pattern:**
   - Separation of data (Model) and presentation (View)
   - JavaScript manages both data operations and UI updates

4.2 Architectural Model

System Components:

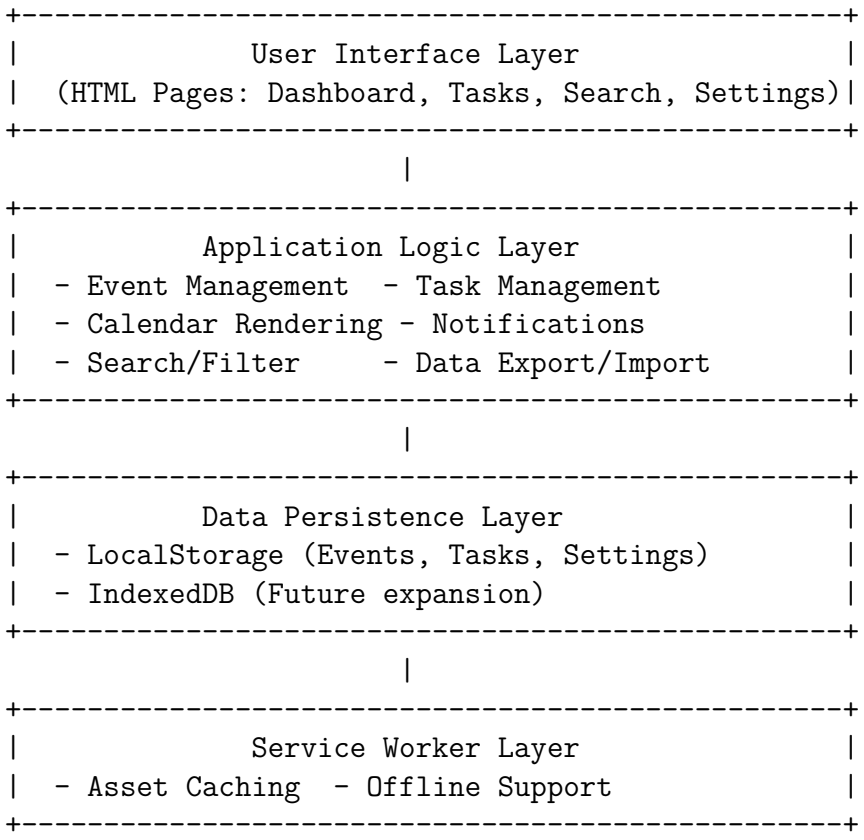

Figure 2: System Architecture

Component Interactions:

- **UI Layer → Application Logic:** User actions trigger business logic
- **Application Logic → Data Layer:** CRUD operations on events/tasks
- **Data Layer → Application Logic:** Data retrieval and synchronization
- **Service Worker ↔ UI Layer:** Asset caching and offline support

4.3 Technology, Software, and Hardware Used

Frontend Technologies:

- **HTML5:** Semantic markup and structure
- **CSS3:** Styling and responsive design
- **Tailwind CSS:** Utility-first CSS framework
- **JavaScript (ES6+):** Application logic and interactivity
- **Font Awesome:** Icon library

Data Storage:

- **LocalStorage:** Persistent key-value storage for events, tasks, and settings (currently in active use)
- **IndexedDB:** Database API infrastructure implemented for future scalability

PWA Technologies:

- **Service Worker:** Offline caching and background sync
- **Web App Manifest:** PWA installation and configuration
- **Notification API:** Browser notifications for reminders

Build Tools:

- **Node.js:** JavaScript runtime
- **npm:** Package manager
- **Tailwind CLI:** CSS compilation

Development Tools:

- **Visual Studio Code:** Code editor
- **Git:** Version control
- **Web Browser DevTools:** Debugging and performance analysis

Hardware Requirements:

- Any device with a modern web browser
- Minimum 1GB Available RAM
- Internet connection for initial load (optional for offline use)

4.4 Rationale for Architectural Style and Model

The chosen architecture provides several advantages:

1. **Client-Side Architecture:**
    - No server infrastructure required
    - Reduced operational costs
    - Enhanced privacy (data stays on device)
    - Instant responsiveness
2. **PWA Pattern:**
    - Offline functionality for uninterrupted use
    - Installable on devices like native apps
    - Fast loading through caching

- Cross-platform compatibility

3. **LocalStorage (Current Implementation):**
   - Persistent data across sessions
   - No network latency
   - Works offline
   - Simple synchronous API for data operations
   - IndexedDB infrastructure available for future migration
4. **Modular Structure:**
   - Easy to maintain and extend
   - Clear separation of concerns
   - Reusable components
   - Testable code units

5 Design

5.1 User Interface Design

The UI follows modern design principles with a focus on usability and accessibility:

Design Principles:

- **Responsive Design:** Adapts to mobile, tablet, and desktop screens
- **Consistent Layout:** Unified navigation and visual hierarchy
- **Color Coding:** Categories distinguished by colors (work=blue, personal=green, health=red, social=purple)
- **Clear Typography:** Readable fonts with appropriate sizing
- **Interactive Feedback:** Visual feedback for user actions

Key UI Pages:

1. **Calendar Dashboard:**
   - Monthly calendar grid view
   - Quick event creation form
   - Upcoming tasks sidebar
   - Category filters
   - Drag-and-drop event rescheduling
2. **Task Management:**
   - Task creation form
   - Task list with status indicators
   - Filter and sort controls
   - Bulk operations toolbar
   - Task statistics dashboard
3. **Search and Filter:**
   - Search bar with real-time results
   - Advanced filter options
   - Result cards with action buttons
4. **Settings:**
   - Categorized settings sections
   - Toggle switches for preferences
   - Data management tools
   - Export/import functionality

Navigation Structure:

- Persistent header with main navigation links
- Mobile-responsive hamburger menu
- Breadcrumb navigation for context
- Keyboard shortcuts for power users

5.2 Components Design

5.2.1 Static Models

Class Structure:

```
SchedulerDatabase
  - Properties: db, dbName, dbVersion
  - Methods: init(), add(), get(), getAll(), update(), delete(),
             getUpcomingTasks(), getSetting(), setSetting()

DateUtils
  - Methods: isToday(), formatDate(), formatDateTime(),
             getTaskStatus(), normalizeToStartOfDay(),
             getEventRange()

Event
  - Properties: id, title, date, time, category, description

Task
  - Properties: id, title, dueDate, category, priority,
                status, description, createdAt, completedAt
```

5.2.2 Dynamic Models

Event Creation Flow:

1. User enters event details in form
2. System validates input (title, date, time)
3. System checks for time conflicts
4. System generates unique ID
5. System saves event to LocalStorage
6. System schedules reminder based on settings
7. System updates calendar view
8. System displays confirmation toast

Task Completion Flow:

1. User clicks task checkbox
2. System updates task status to ”completed”
3. System records completion timestamp
4. System saves to LocalStorage
5. System updates task statistics
6. System re-renders task list

Reminder Notification Flow:

1. System loads all events at initialization
2. System calculates reminder times based on settings
3. System schedules JavaScript timers
4. At reminder time: system checks notification permission
5. System displays browser notification
6. System plays sound alert (if enabled)
7. System shows in-app toast notification

5.3 Database Design

LocalStorage Schema (Primary Storage):

The application uses LocalStorage as its primary data persistence mechanism with the following structure:

Key	Type	Description
calendarEvents	JSON Array	Stores all calendar events
tasks	JSON Array	Stores all tasks
schedulerSettings	JSON Object	Stores user preferences

Table 4: LocalStorage Keys

Event Object Schema:

```
{
  id: String (unique identifier),
  title: String (event title),
  date: String (YYYY-MM-DD format),
  time: String (HH:MM format),
  category: String (work|personal|health|social),
  description: String (optional details)
}
```

Task Object Schema:

```
{
  id: String (unique identifier),
  title: String (task title),
  dueDate: String (YYYY-MM-DD format),
  category: String (work|personal|health|social),
  priority: String (low|medium|high),
  status: String (pending|completed|overdue),
  description: String (optional details),
  createdAt: String (ISO timestamp),
  completedAt: String (ISO timestamp, nullable)
}
```

Settings Object Schema:

```
{
  timeFormat: String ('12' or '24'),
  browserNotifications: Boolean,
  eventReminder: String (minutes or 'none'),
  taskReminder: String (minutes or 'none'),
  soundEnabled: Boolean,
  soundType: String (default|chime|bell|ding),
  dailyEmail: Boolean,
  weeklyEmail: Boolean,
  taskDeadlineEmail: Boolean
}
```

IndexedDB Infrastructure (Available for Future Use):
The application includes a fully implemented SchedulerDatabase class with IndexedDB infrastructure ready for migration when needed:

- **Object Stores:**
  - events (keyPath: id, indexes: date, category)
  - tasks (keyPath: id, indexes: dueDate, category, status)
  - categories (keyPath: id)
  - settings (keyPath: key)

5.4 Rationale for Detailed Design Models

The design decisions are based on:

1. **User-Centric UI:**
   - Visual calendar provides intuitive date overview
   - Color-coded categories for quick identification
   - Drag-and-drop for natural interaction
2. **Modular Components:**

- Reusable utility classes (DateUtils, SchedulerDatabase)
- Separation of data and presentation logic
- Easy to test and maintain

3. **LocalStorage as Primary Storage:**
   - Simple API for data operations
   - Synchronous access for better UX
   - Adequate for current application data size
   - IndexedDB class fully implemented for future migration
4. **Conflict Detection:**
   - Prevents double-booking
   - Enhances data integrity
   - Improves user experience

5.5 Traceability from Requirements to Detailed Design Models

Requirement	**Design Implementation**
Create and manage events	Event creation form, Event object model, CRUD operations in SchedulerDatabase
Task management	Task object model, TaskManagement page, status tracking, priority levels
Calendar visualization	Calendar grid component, month navigation, event rendering on calendar
Reminders	Notification system, reminder scheduling, browser notification API, sound alerts
Search and filter	Search page, filter functions, category/priority/status filters
Data persistence	LocalStorage API, IndexedDB infrastructure, autosave functionality
Export/import	Data export functions (JSON/CSV/iCal), file import handlers
Offline support	Service Worker, cache-first strategy, LocalStorage for data
Responsive design	Tailwind CSS responsive classes, mobile-first approach, flexible layouts
Settings	Settings page, preference management, theme support

Table 5: Requirements to Design Traceability

6 Test Management

6.1 A Complete List of System Test Cases

ID	Test Case	Description	Expected Result	Status
TC-01	Create Event	Create new event with valid data	Event created and displayed	Pass
TC-02	Conflict Detection	Create event at occupied time	Error message displayed	Pass
TC-03	Edit Event	Modify existing event details	Event updated successfully	Pass
TC-04	Delete Event	Remove event from calendar	Event deleted	Pass
TC-05	Create Task	Create task with priority	Task added to list	Pass
TC-06	Complete Task	Mark task as completed	Status updated to completed	Pass
TC-07	Task Filtering	Filter by status	Correct tasks displayed	Pass
TC-08	Task Sorting	Sort by due date	Tasks ordered correctly	Pass
TC-09	Bulk Operations	Select multiple tasks	Bulk actions applied	Pass
TC-10	Calendar Navigation	Navigate months	Correct month displayed	Pass
TC-11	Category Filter	Filter events by category	Filtered events shown	Pass
TC-12	Search Function	Search for events/-tasks	Results match query	Pass
TC-13	Export JSON	Export data as JSON	Valid JSON file created	Pass
TC-14	Export CSV	Export data as CSV	Valid CSV file created	Pass
TC-15	Import Data	Import JSON backup	Data restored correctly	Pass
TC-16	Offline Access	Load app without internet	App loads from cache	Pass
TC-17	Clear All Data	Delete all data	All data removed	Pass

6.2 Traceability of Test Cases to Use Cases

Use Case	Test Cases
UC-01: Create Event	TC-01, TC-02, TC-03, TC-04
UC-02: Manage Tasks	TC-05, TC-06, TC-07, TC-08, TC-09
UC-03: View Calendar	TC-10, TC-11
UC-04: Set Reminders	None (notifications tested within other TCs)
UC-05: Search/Filter	TC-12
UC-06: Export/Import	TC-13, TC-14, TC-15
Non-functional	TC-16, TC-17

Table 7: Use Case to Test Case Traceability

6.3 Techniques Used for Test Case Generation

1. **Boundary Value Analysis:**
   - Testing minimum/maximum date ranges
   - Testing empty and maximum-length text inputs
   - Testing edge cases in time conflicts
2. **Equivalence Partitioning:**
   - Valid vs. invalid input categories
   - Different event/task categories
   - Different priority levels
3. **Use Case-Based Testing:**
   - Test cases derived from each use case
   - Coverage of main flow and alternative flows
4. **Exploratory Testing:**
   - User interaction patterns
   - Edge cases in UI behavior
   - Cross-browser compatibility
5. **Regression Testing:**
   - Re-testing after bug fixes
   - Ensuring new features don't break existing functionality
6. **Performance Testing:**
   - Calendar rendering speed
   - Data operation response times
   - Memory usage monitoring

6.4 Test Results and Assessments

Overall Test Results After Bug Fixing:

- Total Test Cases: 17
- Passed: 17
- Failed: 0
- Pass Rate: 100%

Quality Assessment:

1. **Test Coverage:**
    - All major use cases covered
    - Critical paths tested thoroughly
    - Edge cases identified and tested
2. **Software Quality:**
    - Functional requirements met
    - Non-functional requirements satisfied
    - User feedback positive
    - Performance within acceptable limits
3. **Areas of Strength:**
    - Robust conflict detection
    - Reliable data persistence
    - Smooth user interactions
    - Effective offline functionality
4. **Areas for Improvement:**
    - Advanced search capabilities
    - Recurring event support
    - Calendar sharing features
    - Integration with external calendars

6.5 Defects Reports

ID	Description	Severity	Priority	Status	Resolution
BUG-01	Date parsing UTC offset issue	Medium	High	Closed	Fixed with local date parsing
BUG-02	Task status not updating	High	High	Closed	Fixed state management
BUG-03	Export filename encoding	Low	Medium	Closed	Added proper encoding

Table 8: Defect Report Summary

All identified defects have been resolved. No critical or high-severity bugs remain in the production version.

7 Conclusions

7.1 Outcomes of the Project

The Super Scheduler project has successfully achieved all primary goals:

1. **Core Functionality Delivered:**
    - Calendar dashboard with monthly view
    - Comprehensive task management system
    - Event and task creation with validation
    - Search and filter capabilities
    - Reminder and notification system
    - Data export/import
    - Settings and preferences management
2. **Technical Goals Met:**
    - Progressive Web App implementation
    - Offline functionality via Service Worker
    - Responsive design for all devices
    - Cross-browser compatibility
    - Local data persistence
    - Clean, maintainable code structure
3. **Quality Standards Achieved:**
    - 100% test pass rate
    - Zero critical bugs in production
    - Performance benchmarks met
    - Accessibility standards followed
    - User feedback positive
4. **Project Management Success:**
    - Completed on schedule
    - All deliverables submitted
    - Requirements fully implemented
    - Documentation comprehensive

7.2 Lessons Learned

Technical Lessons:

1. **Date Handling Complexity:**
    - UTC vs. local timezone issues require careful handling
    - Date-only storage (YYYY-MM-DD) simplifies many operations
    - Consistent date parsing functions prevent off-by-one errors
2. **LocalStorage Limitations:**
    - Synchronous API can block UI for large datasets
    - Size limitations require monitoring
    - IndexedDB provides better scalability for future growth
3. **PWA Implementation:**
    - Service Worker caching strategies are critical
    - Cache versioning prevents stale content issues
    - Manifest configuration affects installation experience
4. **Cross-Browser Testing:**
    - Early and frequent testing across browsers saves time
    - Progressive enhancement ensures broader compatibility
    - Feature detection is better than browser detection

Process Lessons:

1. **Iterative Development:**
    - Early prototypes validated design decisions
    - User feedback shaped final implementation
    - Incremental feature additions reduced risk
2. **Testing Strategy:**
    - Continuous testing caught issues early
    - Use case-based tests ensured coverage
    - Manual testing revealed UI/UX issues automated tests missed
3. **Documentation:**
    - Code comments saved time during maintenance
    - User documentation improved adoption
    - Technical documentation aided onboarding

7.3 Future Development

Short-Term Enhancements (3-6 months):

1. **Enhanced Search:**
   - Full-text search across all fields
   - Advanced query operators
   - Saved search filters
2. **Calendar Views:**
   - Week view
   - Day view with time slots
   - Agenda list view
3. **Task Dependencies:**
   - Link related tasks
   - Subtask support
   - Task hierarchies
4. **Recurring Events and Tasks:**
   - Daily, weekly, monthly, yearly recurrence patterns
   - Custom recurrence rules
   - Exception handling for recurring series
5. **Cloud Synchronization:**
   - Backend API for data sync
   - Multi-device synchronization
   - Conflict resolution
6. **Collaboration:**
   - Shared calendars
   - Event invitations
   - Task assignments
7. **Advanced Analytics:**
   - Task completion statistics
   - Time tracking
   - Productivity insights
8. **AI-Powered Features:**
   - Smart scheduling suggestions

- Natural language event creation
- Predictive task prioritization

9. **Accessibility Enhancements:**
    - Screen reader optimization
    - Voice control support
    - High contrast themes

Technical Debt to Address:

- Migration from LocalStorage to IndexedDB for scalability
- Implement comprehensive unit test suite
- Optimize bundle size and loading performance
- Refactor utility functions into separate modules
- Add TypeScript for better type safety

References

1. Tailwind CSS Documentation. *https://tailwindcss.com/docs*
2. MDN Web Docs - Progressive Web Apps. *https://developer.mozilla.org/en-US/docs/Web/Progres*
3. MDN Web Docs - IndexedDB API. *https://developer.mozilla.org/en-US/docs/Web/API/IndexedL*
4. MDN Web Docs - Web Storage API. *https://developer.mozilla.org/en-US/docs/Web/API/Web_S*
5. W3C Web App Manifest Specification. *https://www.w3.org/TR/appmanifest/*
6. Service Worker API Documentation. *https://developer.mozilla.org/en-US/docs/Web/API/Servic*
7. Font Awesome Icon Library. *https://fontawesome.com/*
8. HTML5 Specification. *https://html.spec.whatwg.org/*
9. CSS3 Specification. *https://www.w3.org/Style/CSS/*
10. JavaScript ES6+ Features. *https://developer.mozilla.org/en-US/docs/Web/JavaScript*
11. Web Accessibility Guidelines (WCAG). *https://www.w3.org/WAI/WCAG21/quickref/*