# miniCycle - Comprehensive Technical Documentation

Version: 1.275
Last Updated: September 2025
Developer: sparkinCreations
License: Proprietary

## Table of Contents

## Overview

### Application Summary

miniCycle is a sophisticated web-based task management application that revolutionizes productivity through a unique "cycling" approach. Unlike traditional task managers, miniCycle automatically resets completed task lists, promoting habit formation and routine establishment.

Core Value Proposition: *"Turn Your Routine Into Progress"*

### Key Differentiators

- **Automatic Task Cycling**: Tasks reset when all are completed, promoting consistent habits
- **Multiple Cycle Types**: Auto-reset, manual reset, and to-do list modes
- **Advanced Recurring System**: Comprehensive scheduling with hourly to yearly frequencies

- **Gamification Elements**: Unlockable themes, milestone badges, and progress tracking
- **Dual Version Architecture**: Full-featured and lite versions for optimal device compatibility

### Target Users

- **Primary**: Habit-focused professionals, routine-oriented workers, neurodivergent individuals
- **Secondary**: Small team leaders, students, wellness enthusiasts
- **Enterprise**: Teams requiring structured, repeatable workflows

-----

## Architecture

### System Architecture Overview

```
┌─────────────────────────────────────────────────────────────
──┐
│                 miniCycle Application                   │
├─────────────────────────────────────────────────────────────
──┤
│  Presentation Layer                                     │
│  ├── HTML5 Structure                                    │
│  ├── CSS3 with Custom Properties                        │
│  ├── Progressive Web App (PWA) Features                 │
│  └── Responsive Design (Mobile-First)                   │
├─────────────────────────────────────────────────────────────
──┤
│  Application Layer                                      │
│  ├── Vanilla JavaScript (ES6+)                          │
│  ├── Event-Driven Architecture                          │
│  ├── Modular Component System                           │
│  └── State Management                                   │
├─────────────────────────────────────────────────────────────
──┤
│  Data Layer                                             │
│  ├── LocalStorage (Primary)                             │
│  ├── Schema Version Management (2.5)                    │
│  ├── Data Migration System                              │
│  └── Backup/Export System (.mcyc files)                 │
├─────────────────────────────────────────────────────────────
──┤
│  Compatibility Layer                                    │
│  ├── Modern Browser Support (Chrome, Firefox, Safari, Edge) │
│  ├── ES5 Fallback (Lite Version)                    │
│  ├── IE11+ Compatibility                            │
│  └── Progressive Enhancement                        │
└──────────────────────────────────────────────────────
___┘
```

### File Structure

```
miniCycle/
├── miniCycle.html              # Main application entry point
├── miniCycle-lite.html         # Lightweight version for older devices
├── miniCycle-scripts.js        # Core application logic
├── miniCycle-lite-scripts.js   # ES5-compatible scripts
├── miniCycle-styles.css        # Main stylesheet
├── miniCycle-lite-styles.css   # Optimized styles for lite version
├── user-manual.html            # Comprehensive user guide
├── user-manual-styles.css      # User manual styling
├── terms.html                  # Terms of service
├── privacy.html                # Privacy policy
├── product.html                # Marketing/product page
├── manifest.json               # PWA manifest (full version)
├── manifest-lite.json          # PWA manifest (lite version)
├── assets/
│   ├── images/
│   │   ├── logo/               # Application logos and icons
│   │   └── screenshots/        # App store screenshots
│   └── icons/                  # PWA icons in various sizes
└── utilities/
    └── testing-modal.js        # Development and testing utilities
```

## Core Features

### 1. Task Management System

#### Basic Task Operations

Task Creation

- Input validation with 50-character limit

- XSS prevention through input sanitization
- Duplicate prevention logic
- Auto-focus and keyboard shortcuts

Task Completion

- Visual checkbox interface
- Keyboard accessibility (Space, Enter)
- Completion animations and feedback
- Progress tracking integration

Task Modification

- In-line editing capabilities
- Priority marking system
- Due date assignment
- Drag-and-drop reordering

#### Advanced Task Features

Task Properties

```javascript
{
  id: "unique-task-identifier",
  text: "Task description",
  completed: false,
  priority: false,
  dueDate: "2025-09-16T10:00:00Z",
  remindersEnabled: false,
  recurring: false,
  recurringSettings: {},
  schemaVersion: 2,
  createdAt: "2025-09-16T08:00:00Z",
  completedAt: null
}
```

### 2. Cycle Management System

#### Cycle Types

Auto-Reset Mode

- Automatically resets all tasks when cycle completes
- Configurable delay before reset
- Completion celebration animations
- Milestone tracking and rewards

Manual Reset Mode

- User-controlled reset via “Complete” button
- Maintains completed state until manual action
- Progress preservation between sessions
- Batch completion operations

To-Do List Mode

- Traditional task deletion on completion
- No reset functionality
- Linear progress tracking
- Suitable for one-time project management

#### Multi-Cycle Management

Cycle Switching

- Seamless switching between different cycles
- Preview functionality before switching
- Recent cycle history
- Quick access to frequently used cycles

Cycle Operations

```javascript
// Create new cycle
createNewMiniCycle(name, type)

// Switch between cycles
switchToMiniCycle(cycleName)

// Import/export cycles
exportMiniCycle(cycleName)
importMiniCycle(fileData)

// Delete cycles
deleteMiniCycle(cycleName)
```

### 3. Recurring Tasks System

#### Frequency Options

Time-Based Frequencies

- **Hourly**: Every N hours
- **Daily**: Every N days
- **Weekly**: Specific days of the week
- **Biweekly**: Every two weeks
- **Monthly**: Specific dates or weekdays
- **Yearly**: Specific months and days

Advanced Scheduling

```javascript
const recurringSettings = {
  frequency: "weekly",
  indefinitely: true,
  count: null,
  time: {
    hour: 9,
    minute: 0,
    meridiem: "AM",
    military: "09:00"
  },

  /**
   * Fetch API polyfill for older browsers
   */
  polyfillFetch() {
    if (typeof fetch === 'undefined') {
      window.fetch = function(url, options = {}) {
        return new Promise((resolve, reject) => {
          const xhr = new XMLHttpRequest();

          xhr.open(options.method || 'GET', url);

          // Set headers
          if (options.headers) {
            Object.entries(options.headers).forEach(([key, value]) => {
              xhr.setRequestHeader(key, value);
            });
          }

          xhr.onload = () => {
            const response = {
              ok: xhr.status >= 200 && xhr.status < 300,
              status: xhr.status,
              statusText: xhr.statusText,
              text: () => Promise.resolve(xhr.responseText),
              json: () => Promise.resolve(JSON.parse(xhr.responseText))
            };
            resolve(response);
          };

          xhr.onerror = () => reject(new Error('Network error'));
          xhr.ontimeout = () => reject(new Error('Request timeout'));

          xhr.send(options.body);
        });
      };
    }
  },

  /**
   * Array methods polyfill
   */
  polyfillArrayMethods() {
    // Array.from polyfill
    if (!Array.from) {
      Array.from = function(arrayLike, mapFn, thisArg) {
        const result = [];
        for (let i = 0; i < arrayLike.length; i++) {
          const value = mapFn ? mapFn.call(thisArg, arrayLike[i], i) : arrayLike[i];
          result.push(value);
        }
        return result;
      };
    }

    // Array.includes polyfill
    if (!Array.prototype.includes) {
      Array.prototype.includes = function(searchElement, fromIndex) {
        const o = Object(this);
        const len = parseInt(o.length) || 0;
        if (len === 0) return false;
```

```
        const n = parseInt(fromIndex) || 0;
        let k = n >= 0 ? n : Math.max(len + n, 0);

        while (k < len) {
          if (o[k] === searchElement) return true;
          k++;
        }
        return false;
      };
    }
  }
};
```

## Development Guide

### Setup and Installation

#### Local Development Environment

```bash
# Clone or download the project
git clone [repository-url] minicycle
cd minicycle

# No build process required - pure HTML/CSS/JS
# Simply serve files with a local server

# Using Python 3
python -m http.server 8000

# Using Node.js http-server
npx http-server -p 8000

# Using PHP
php -S localhost:8000

# Access application at http://localhost:8000
```

#### Development Server Configuration

```javascript
// Optional: Development server with auto-reload
// server.js
const express = require('express');
const path = require('path');
const app = express();

// Serve static files
app.use(express.static('.'));

// Development middleware
if (process.env.NODE_ENV === 'development') {
  // Disable caching
  app.use((req, res, next) => {
    res.setHeader('Cache-Control', 'no-cache, no-store, must-revalidate');
    res.setHeader('Pragma', 'no-cache');
    res.setHeader('Expires', '0');
    next();
  });

  // Add development headers
  app.use((req, res, next) => {
    res.setHeader('X-Development-Mode', 'true');
    next();
  });
}

// Handle PWA routes
app.get('/miniCycle.html', (req, res) => {
  res.sendFile(path.join(__dirname, 'miniCycle.html'));
});

app.get('/miniCycle-lite.html', (req, res) => {
  res.sendFile(path.join(__dirname, 'miniCycle-lite.html'));
});

const port = process.env.PORT || 8000;
app.listen(port, () => {
  console.log(`🚀 miniCycle development server running on port ${port}`);
});
```

### Code Organization

#### File Naming Conventions

```
Components:
- miniCycle.html         # Main application
- miniCycle-lite.html    # Lightweight version
- miniCycle-scripts.js   # Core logic
- miniCycle-styles.css   # Main styles

Features:
- feature-name.js        # Feature-specific logic
- feature-name.css       # Feature-specific styles

Utilities:
- utilities/             # Helper functions and tools
- assets/                # Static assets (images, icons)

Documentation:
- user-manual.html       # User documentation
- terms.html             # Legal documents
- privacy.html           # Privacy policy
```

#### Code Style Guidelines

```javascript
// Use descriptive function names
function createTaskElement(taskData) { }       // ✅ Good
function createElement(data) { }               // ❌ Too generic

// Use consistent naming conventions
const taskElement = document.createElement('li'); // ✅ camelCase for variables
const TASK_LIMIT = 50;                          // ✅ UPPER_CASE for constants
const TaskManager = { };                        // ✅ PascalCase for objects/classes

// Use meaningful comments
/**
 * Validates task input and sanitizes for XSS prevention
 * @param {string} input - Raw user input
 * @returns {object} Validation result with sanitized input
 */
function validateTaskInput(input) {
  // Implementation...
}
```

```
// Use consistent error handling
function risky_operation() {
  try {
    // Main logic
    return { success: true, data: result };
  } catch (error) {
    console.error('Operation failed:', error);
    return { success: false, error: error.message };
  }
}

// Use defensive programming
function updateTaskElement(taskElement, taskData) {
  if (!taskElement || !taskData) {
    console.warn('Invalid parameters for updateTaskElement');
    return false;
  }

  // Implementation...
  return true;
}
```

### Contribution Guidelines

#### Code Review Process

```markdown
## Pull Request Requirements

### Before Submitting
- [ ] Code follows style guidelines
- [ ] All tests pass
- [ ] Documentation updated
- [ ] Browser compatibility tested
- [ ] Accessibility compliance verified

### Code Quality Standards
- Functions should be under 50 lines
- No global variables (except designated app globals)
- Error handling for all user inputs
- Comments for complex logic
- Consistent naming conventions
```

### Testing Requirements
- Unit tests for new functions
- Integration tests for user workflows
- Manual testing on 3+ browsers
- Mobile responsiveness verified
- Accessibility testing completed

#### Git Workflow

```bash
# Feature development workflow
git checkout -b feature/task-tags
git add .
git commit -m "feat: Add task tagging system

- Implement tag creation and assignment
- Add tag filtering interface
- Update storage schema for tags
- Add tests for tag functionality

Closes #123"

git push origin feature/task-tags
# Create pull request
```

## Testing & Quality Assurance

### Automated Testing Framework

#### Unit Testing

```javascript
/**
 * Comprehensive test suite for miniCycle
 */
const TestSuite = {
  /**
   * Run all automated tests
   */
```

```
runAllTests() {
  console.log('🧪 Starting miniCycle Test Suite...');

  const results = {
    passed: 0,
    failed: 0,
    tests: []
  };

  // Core functionality tests
  this.testTaskManagement(results);
  this.testDataPersistence(results);
  this.testRecurringTasks(results);
  this.testUndoRedo(results);
  this.testInputValidation(results);
  this.testThemeSystem(results);
  this.testCycleManagement(results);
  this.testAccessibility(results);

  // Generate report
  this.generateTestReport(results);

  return results;
},

/**
 * Test task management operations
 */
testTaskManagement(results) {
  const tests = [
    {
      name: 'Add Task',
      test: () => {
        const initialCount = document.querySelectorAll('.task').length;
        addTask('Test Task', false, false);
        const newCount = document.querySelectorAll('.task').length;
        return newCount === initialCount + 1;
      }
    },
    {
      name: 'Complete Task',
      test: () => {
        const taskElement = document.querySelector('.task');
        if (!taskElement) return false;
      const checkbox = taskElement.querySelector('input[type="checkbox"]');
      const wasChecked = checkbox.checked;

      checkbox.click();

      return checkbox.checked !== wasChecked;
    }
  },
  {
    name: 'Delete Task',
    test: () => {
      const initialCount = document.querySelectorAll('.task').length;
      const firstTask = document.querySelector('.task');

      if (!firstTask) return initialCount === 0;

      const deleteBtn = firstTask.querySelector('.delete-btn');
      if (deleteBtn) deleteBtn.click();

      const newCount = document.querySelectorAll('.task').length;
      return newCount === initialCount - 1;
    }
  },
  {
    name: 'Task Priority Toggle',
    test: () => {
      addTask('Priority Test Task', false, false);
      const taskElement = document.querySelector('.task:last-child');
      const priorityBtn = taskElement.querySelector('.priority-btn');

      priorityBtn.click();

      return taskElement.classList.contains('high-priority');
    }
  },
  {
    name: 'Task Editing',
    test: () => {
      addTask('Edit Test', false, false);
      const taskElement = document.querySelector('.task:last-child');
      const editBtn = taskElement.querySelector('.edit-btn');

      editBtn.click();
```

```
        // Simulate editing
        const taskText = taskElement.querySelector('.task-text');
        const newText = 'Edited Task Text';
        taskText.textContent = newText;

        return taskText.textContent === newText;
      }
    }
  ];

  this.executeTests('Task Management', tests, results);
},

/**
 * Test data persistence
 */
testDataPersistence(results) {
  const tests = [
    {
      name: 'Save to localStorage',
      test: () => {
        const testData = { test: 'persistence' };
        localStorage.setItem('miniCycle_test', JSON.stringify(testData));

        const retrieved = JSON.parse(localStorage.getItem('miniCycle_test'));
        localStorage.removeItem('miniCycle_test');

        return retrieved && retrieved.test === 'persistence';
      }
    },
    {
      name: 'Auto-save Functionality',
      test: () => {
        const initialData = localStorage.getItem('miniCycleData');
        addTask('Auto-save Test', false, true);
        const newData = localStorage.getItem('miniCycleData');

        return initialData !== newData;
      }
    },
    {
      name: 'Data Migration',
      test: () => {
        // Test schema migration logic
        const oldSchema = {
          version: "1.0",
          tasks: [{ text: "Test", completed: false }]
        };

        // This would need the actual migration function
        // const migrated = migrateToCurrentSchema(oldSchema);
        // return migrated.version === "2.5" && migrated.data.cycles;

        return true; // Placeholder for actual migration test
      }
    },
    {
      name: 'Data Validation',
      test: () => {
        const validData = {
          version: "2.5",
          metadata: { created: new Date().toISOString() },
          data: { cycles: {} }
        };

        // This would use the actual validation function
        // return validateSchemaData(validData).valid;

        return true; // Placeholder
      }
    }
  ];

  this.executeTests('Data Persistence', tests, results);
},

/**
 * Test recurring tasks functionality
 */
testRecurringTasks(results) {
  const tests = [
    {
      name: 'Create Recurring Task',
      test: () => {
        // Test recurring task creation
        const recurringSettings = {
          frequency: 'daily',
          indefinitely: true,
          time: { hour: 9, minute: 0 }
        };

        // This would test actual recurring task creation
        return true; // Placeholder
      }
    },
    {
      name: 'Recurring Task Recreation',
      test: () => {
        // Test that recurring tasks are recreated at appropriate times
        return true; // Placeholder
      }
    },
    {
      name: 'Recurring Settings Validation',
      test: () => {
        const validSettings = {
          frequency: 'weekly',
          weekly: { days: ['Monday', 'Wednesday'] }
        };

        // Test settings validation
        return true; // Placeholder
      }
    }
  ];

  this.executeTests('Recurring Tasks', tests, results);
},

/**
 * Test undo/redo functionality
 */
testUndoRedo(results) {
  const tests = [
    {
      name: 'Undo Task Addition',
      test: () => {
        const initialCount = document.querySelectorAll('.task').length;
        addTask('Undo Test', false, true);

        if (window.performUndo) {
          window.performUndo();
          const finalCount = document.querySelectorAll('.task').length;
          return finalCount === initialCount;
        }

        return false;
      }
    },
    {
      name: 'Redo Task Addition',
      test: () => {
        const initialCount = document.querySelectorAll('.task').length;
        addTask('Redo Test', false, true);

        if (window.performUndo && window.performRedo) {
          window.performUndo();
          window.performRedo();
          const finalCount = document.querySelectorAll('.task').length;
          return finalCount === initialCount + 1;
        }

        return false;
      }
    },
    {
      name: 'Undo Stack Limit',
      test: () => {
        // Test that undo stack respects limit
        const undoLimit = 4;

        for (let i = 0; i < undoLimit + 2; i++) {
          addTask(`Limit Test ${i}`, false, true);
        }

        // Check that undo stack doesn't exceed limit
        return window.undoStack ? window.undoStack.length <= undoLimit : true;
      }
    }
  ];

  this.executeTests('Undo/Redo', tests, results);
},

/**
```

```
 * Test input validation and security
 */
testInputValidation(results) {
  const tests = [
    {
      name: 'XSS Prevention',
      test: () => {
        const maliciousInput = '<script>alert("xss")</script>';
        const sanitized = sanitizeInput(maliciousInput);

        return !sanitized.includes('<script>');
      }
    },
    {
      name: 'Task Length Limit',
      test: () => {
        const longInput = 'A'.repeat(100);
        const sanitized = sanitizeInput(longInput);

        return sanitized.length <= 50;
      }
    },
    {
      name: 'Empty Input Handling',
      test: () => {
        const emptyInputs = ['', '   ', '\n\t', null, undefined];

        return emptyInputs.every(input => {
          const sanitized = sanitizeInput(input);
          return sanitized === '';
        });
      }
    },
    {
      name: 'HTML Tag Removal',
      test: () => {
        const htmlInput = '<div>Clean <b>this</b> up</div>';
        const sanitized = sanitizeInput(htmlInput);

        return sanitized === 'Clean this up' && !sanitized.includes('<');
      }
    }
  ];
  this.executeTests('Input Validation', tests, results);
},

/**
 * Test theme system
 */
testThemeSystem(results) {
  const tests = [
    {
      name: 'Dark Mode Toggle',
      test: () => {
        const body = document.body;
        const initialTheme = body.className;

        // Toggle dark mode
        if (window.toggleDarkMode) {
          window.toggleDarkMode();
          const afterToggle = body.className;

          window.toggleDarkMode(); // Reset

          return initialTheme !== afterToggle;
        }

        return false;
      }
    },
    {
      name: 'Theme Persistence',
      test: () => {
        // Test that theme preference is saved
        const themeKey = 'miniCycle_darkMode';
        localStorage.setItem(themeKey, 'true');

        const stored = localStorage.getItem(themeKey);
        localStorage.removeItem(themeKey);

        return stored === 'true';
      }
    },
    {
      name: 'Theme Unlock Tracking',
      test: () => {
        // Test milestone-based theme unlocking
        return true; // Placeholder for actual theme unlock logic
      }
    }
  ];

  this.executeTests('Theme System', tests, results);
},

/**
 * Test cycle management
 */
testCycleManagement(results) {
  const tests = [
    {
      name: 'Create New Cycle',
      test: () => {
        // Test new cycle creation
        if (window.createNewMiniCycle) {
          const result = window.createNewMiniCycle('Test Cycle');
          return result === true;
        }
        return false;
      }
    },
    {
      name: 'Switch Between Cycles',
      test: () => {
        // Test cycle switching
        if (window.switchToMiniCycle) {
          const result = window.switchToMiniCycle('Default');
          return result === true;
        }
        return false;
      }
    },
    {
      name: 'Cycle Export',
      test: () => {
        // Test cycle export functionality
        if (window.exportMiniCycle) {
          const exportData = window.exportMiniCycle('Default');
          return exportData !== null && typeof exportData === 'object';
        }
        return false;
      }
    },
    {
      name: 'Auto-Reset Functionality',
      test: () => {
        // Test auto-reset when all tasks completed
        return true; // Placeholder for auto-reset test
      }
    }
  ];

  this.executeTests('Cycle Management', tests, results);
},

/**
 * Test accessibility features
 */
testAccessibility(results) {
  const tests = [
    {
      name: 'ARIA Labels Present',
      test: () => {
        const buttons = document.querySelectorAll('button');
        const hasAriaLabels = Array.from(buttons).every(button =>
          button.hasAttribute('aria-label') ||
          button.textContent.trim() !== '' ||
          button.hasAttribute('aria-labelledby')
        );

        return hasAriaLabels;
      }
    },
    {
      name: 'Keyboard Navigation',
      test: () => {
        // Test that focusable elements can be navigated with keyboard
        const focusableElements = document.querySelectorAll(
          'button, input, select, textarea, [tabindex]:not([tabindex="-1"])'
        );

        return focusableElements.length > 0;
      }
    },
    {
```

```
      name: 'Screen Reader Support',
      test: () => {
        // Test for live regions and proper semantic markup
        const liveRegion = document.getElementById('live-region');
        return liveRegion && liveRegion.hasAttribute('aria-live');
      }
    },
    {
      name: 'Color Contrast',
      test: () => {
        // Basic color contrast test (would need more sophisticated checking)
        const style = getComputedStyle(document.body);
        const bgColor = style.backgroundColor;
        const textColor = style.color;

        return bgColor !== textColor; // Basic check
      }
    }
  ];

  this.executeTests('Accessibility', tests, results);
},

/**
 * Execute a set of tests
 */
executeTests(category, tests, results) {
  console.log(`\n📋 Testing ${category}:`);

  tests.forEach(test => {
    try {
      const passed = test.test();
      const result = {
        category,
        name: test.name,
        passed,
        error: null,
        timestamp: new Date().toISOString()
      };

      results.tests.push(result);

      if (passed) {
        results.passed++;
        console.log(`  ✅ ${test.name}`);
      } else {
        results.failed++;
        console.log(`  ❌ ${test.name}`);
      }
    } catch (error) {
      results.failed++;
      const result = {
        category,
        name: test.name,
        passed: false,
        error: error.message,
        timestamp: new Date().toISOString()
      };

      results.tests.push(result);
      console.log(`  💥 ${test.name}: ${error.message}`);
    }
  });
},

/**
 * Generate comprehensive test report
 */
generateTestReport(results) {
  const total = results.passed + results.failed;
  const passRate = total > 0 ? (results.passed / total * 100).toFixed(1) : 0;

  console.log('\n📊 Test Results Summary:');
  console.log(`Total Tests: ${total}`);
  console.log(`Passed: ${results.passed}`);
  console.log(`Failed: ${results.failed}`);
  console.log(`Pass Rate: ${passRate}%`);

  // Group results by category
  const categories = {};
  results.tests.forEach(test => {
    if (!categories[test.category]) {
      categories[test.category] = { passed: 0, failed: 0, tests: [] };
    }
    categories[test.category].tests.push(test);
    if (test.passed) {
      categories[test.category].passed++;
    } else {
      categories[test.category].failed++;
    }
  });

  console.log('\n📈 Results by Category:');
  Object.entries(categories).forEach(([category, data]) => {
    const categoryTotal = data.passed + data.failed;
    const categoryRate = (data.passed / categoryTotal * 100).toFixed(1);
    console.log(`${category}: ${data.passed}/${categoryTotal} (${categoryRate}%)`);
  });

  if (results.failed > 0) {
    console.log('\n❌ Failed Tests:');
    results.tests
      .filter(test => !test.passed)
      .forEach(test => {
        console.log(`  - ${test.category}: ${test.name}`);
        if (test.error) {
          console.log(`    Error: ${test.error}`);
        }
      });
  }

  // Generate downloadable report
  this.generateTestReportFile(results);

  return results;
},

/**
 * Generate downloadable test report
 */
generateTestReportFile(results) {
  const report = {
    summary: {
      timestamp: new Date().toISOString(),
      total: results.passed + results.failed,
      passed: results.passed,
      failed: results.failed,
      passRate: ((results.passed / (results.passed + results.failed)) * 100).toFixed(1) + '%'
    },
    environment: {
      userAgent: navigator.userAgent,
      url: window.location.href,
      viewport: `${window.innerWidth}x${window.innerHeight}`,
      appVersion: '1.275'
    },
    tests: results.tests
  };

  const blob = new Blob([JSON.stringify(report, null, 2)], {
    type: 'application/json'
  });

  const url = URL.createObjectURL(blob);
  const a = document.createElement('a');
  a.href = url;
  a.download = `minicycle-test-report-${Date.now()}.json`;
  a.style.display = 'none';

  document.body.appendChild(a);
  console.log('📄 Test report available for download');

  // Auto-download in development mode
  if (window.location.search.includes('autoDownload=true')) {
    a.click();
  }

  document.body.removeChild(a);
  URL.revokeObjectURL(url);
},

/**
 * Performance benchmarking
 */
benchmarkPerformance() {
  console.log('⚡ Running Performance Benchmarks...');

  const benchmarks = {
    taskCreation: this.benchmarkTaskCreation(),
    dataLoad: this.benchmarkDataLoad(),
    rendering: this.benchmarkRendering(),
    storage: this.benchmarkStorage(),
    memoryUsage: this.benchmarkMemoryUsage()
  };

  console.log('📊 Benchmark Results:', benchmarks);
  return benchmarks;
```

```
},

benchmarkTaskCreation() {
  const iterations = 100;
  const start = performance.now();

  for (let i = 0; i < iterations; i++) {
    const taskData = {
      id: `benchmark_${i}`,
      text: `Benchmark Task ${i}`,
      completed: false,
      createdAt: new Date().toISOString()
    };

    // This would use the actual task creation function
    if (window.createTaskElement) {
      window.createTaskElement(taskData);
    }
  }

  const end = performance.now();
  return {
    totalTime: end - start,
    averageTime: (end - start) / iterations,
    iterations,
    tasksPerSecond: iterations / ((end - start) / 1000)
  };
},

benchmarkDataLoad() {
  const start = performance.now();

  // Benchmark data loading operations
  for (let i = 0; i < 10; i++) {
    const testData = {
      version: "2.5",
      data: {
        cycles: {
          [`test_${i}`]: {
            tasks: Array.from({length: 50}, (_, j) => ({
              id: `task_${i}_${j}`,
              text: `Task ${j}`,
              completed: Math.random() > 0.5
            }))
          }
        }
      }
    };

    localStorage.setItem(`benchmark_${i}`, JSON.stringify(testData));
    JSON.parse(localStorage.getItem(`benchmark_${i}`));
    localStorage.removeItem(`benchmark_${i}`);
  }

  const end = performance.now();
  return {
    totalTime: end - start,
    averageTime: (end - start) / 10,
    operations: 10
  };
},

benchmarkRendering() {
  const taskList = document.getElementById('taskList');
  if (!taskList) return { error: 'Task list not found' };

  const start = performance.now();

  // Create many task elements
  const fragment = document.createDocumentFragment();
  for (let i = 0; i < 100; i++) {
    const li = document.createElement('li');
    li.className = 'task';
    li.innerHTML = `
      <input type="checkbox" id="bench_${i}">
      <label for="bench_${i}">Benchmark Task ${i}</label>
    `;
    fragment.appendChild(li);
  }

  taskList.appendChild(fragment);

  const end = performance.now();

  // Clean up
  Array.from(taskList.children)
    .filter(child => child.querySelector('input[id^="bench_"]'))
    .forEach(child => child.remove());
    return {
      totalTime: end - start,
      elementsCreated: 100,
      elementsPerSecond: 100 / ((end - start) / 1000)
    };
  },

  benchmarkStorage() {
    const iterations = 50;
    const testData = { test: 'benchmark', timestamp:
```

### ES5 Compatibility (Lite Version)

#### IE11+ Support Implementation

```javascript
// ES5-compatible task management for lite version
var MiniCycleLite = {
  // Core properties
  tasks: [],
  undoStack: [],
  redoStack: [],
  maxUndoLevels: 4,

  /**
   * Initialize the lite application
   */
  init: function() {
    this.bindEvents();
    this.loadTasks();
    this.updateUI();

    console.log('✅ miniCycle Lite initialized');
  },

  /**
   * Bind event listeners using ES5 syntax
   */
  bindEvents: function() {
    var self = this;

    // Add task button
```

```
var addButton = document.getElementById('addTask');
if (addButton) {
  addButton.addEventListener('click', function() {
    self.handleAddTask();
  });
}

// Task input enter key
var taskInput = document.getElementById('taskInput');
if (taskInput) {
  taskInput.addEventListener('keydown', function(e) {
    if (e.keyCode === 13) { // Enter key
      self.handleAddTask();
    }
  });
}

// Undo/Redo buttons
var undoBtn = document.getElementById('undo-btn');
var redoBtn = document.getElementById('redo-btn');

if (undoBtn) {
  undoBtn.addEventListener('click', function() {
    self.performUndo();
  });
}

if (redoBtn) {
  redoBtn.addEventListener('click', function() {
    self.performRedo();
  });
}

// Global keyboard shortcuts
document.addEventListener('keydown', function(e) {
  // Ctrl+Z for undo (keyCode 90)
  if ((e.ctrlKey || e.metaKey) && e.keyCode === 90 && !e.shiftKey) {
    e.preventDefault();
    self.performUndo();
  }

  // Ctrl+Y for redo (keyCode 89)
  if ((e.ctrlKey || e.metaKey) && e.keyCode === 89) {
    e.preventDefault();
```

```
      self.performRedo();
    }
  });
},

/**
 * Handle adding new task (ES5 compatible)
 */
handleAddTask: function() {
  var taskInput = document.getElementById('taskInput');
  if (!taskInput) return;

  var taskText = taskInput.value.trim();
  if (!taskText) return;

  // Validate input
  var validation = this.validateInput(taskText);
  if (!validation.valid) {
    this.showNotification(validation.error, 'warning');
    return;
  }

  // Save state for undo
  this.saveUndoState('add_task');

  // Create task object
  var task = {
    id: 'task_' + Date.now() + '_' + Math.random().toString(36).substr(2, 9),
    text: validation.sanitized,
    completed: false,
    createdAt: new Date().toISOString()
  };

  // Add to tasks array
  this.tasks.push(task);

  // Update UI
  this.renderTasks();
  this.updateProgress();

  // Clear input
  taskInput.value = '';

  // Save to storage
  this.saveTasks();

  this.showNotification('Task added successfully', 'success');
},

/**
 * Validate and sanitize input (ES5 compatible)
 */
validateInput: function(input) {
  if (typeof input !== 'string') {
    return { valid: false, error: 'Invalid input type' };
  }

  // Basic sanitization
  var sanitized = input.replace(/<[^>]*>/g, '').trim(); // Remove HTML tags

  if (sanitized.length === 0) {
    return { valid: false, error: 'Task cannot be empty' };
  }

  if (sanitized.length > 50) {
    return { valid: false, error: 'Task too long (max 50 characters)' };
  }

  return { valid: true, sanitized: sanitized };
},

/**
 * Render tasks in the UI (ES5 compatible)
 */
renderTasks: function() {
  var taskList = document.getElementById('taskList');
  if (!taskList) return;

  // Clear existing tasks
  taskList.innerHTML = '';

  // Render each task
  for (var i = 0; i < this.tasks.length; i++) {
    var task = this.tasks[i];
    var taskElement = this.createTaskElement(task);
    taskList.appendChild(taskElement);
  }
},
/**
 * Create task DOM element (ES5 compatible)
 */
createTaskElement: function(task) {
  var self = this;
  var li = document.createElement('li');
  li.className = 'task';
  li.setAttribute('data-task-id', task.id);

  // Create checkbox
  var checkbox = document.createElement('input');
  checkbox.type = 'checkbox';
  checkbox.id = 'checkbox-' + task.id;
  checkbox.checked = task.completed;

  // Create label
  var label = document.createElement('label');
  label.setAttribute('for', checkbox.id);
  label.className = 'task-text';
  label.textContent = task.text;

  // Create delete button
  var deleteBtn = document.createElement('button');
  deleteBtn.className = 'task-btn delete-btn';
  deleteBtn.textContent = '🗑️';
  deleteBtn.setAttribute('aria-label', 'Delete task');

  // Bind events
  checkbox.addEventListener('change', function() {
    self.toggleTask(task.id);
  });

  deleteBtn.addEventListener('click', function() {
    self.deleteTask(task.id);
  });

  // Assemble task element
  var taskContent = document.createElement('div');
  taskContent.className = 'task-content';
  taskContent.appendChild(checkbox);
  taskContent.appendChild(label);

  var taskOptions = document.createElement('div');
  taskOptions.className = 'task-options';
  taskOptions.appendChild(deleteBtn);

  li.appendChild(taskContent);
  li.appendChild(taskOptions);

  if (task.completed) {
    li.classList.add('completed');
  }

  return li;
},

/**
 * Toggle task completion (ES5 compatible)
 */
toggleTask: function(taskId) {
  // Find task in array
  for (var i = 0; i < this.tasks.length; i++) {
    if (this.tasks[i].id === taskId) {
      // Save state for undo
      this.saveUndoState('toggle_task');

      // Toggle completion
      this.tasks[i].completed = !this.tasks[i].completed;
      this.tasks[i].completedAt = this.tasks[i].completed ?
        new Date().toISOString() : null;

      // Update UI
      this.renderTasks();
      this.updateProgress();
      this.saveTasks();

      // Check if all tasks completed
      this.checkCycleCompletion();

      break;
    }
  }
},

/**
 * Delete task (ES5 compatible)
 */
```

```
deleteTask: function(taskId) {
  // Save state for undo
  this.saveUndoState('delete_task');

  // Remove from array
  for (var i = 0; i < this.tasks.length; i++) {
    if (this.tasks[i].id === taskId) {
      this.tasks.splice(i, 1);
      break;
    }
  }

  // Update UI
  this.renderTasks();
  this.updateProgress();
  this.saveTasks();

  this.showNotification('Task deleted', 'info');
},

/**
 * Save current state for undo (ES5 compatible)
 */
saveUndoState: function(action) {
  // Clone current tasks array
  var stateCopy = [];
  for (var i = 0; i < this.tasks.length; i++) {
    var task = this.tasks[i];
    stateCopy.push({
      id: task.id,
      text: task.text,
      completed: task.completed,
      createdAt: task.createdAt,
      completedAt: task.completedAt
    });
  }

  var state = {
    action: action,
    timestamp: Date.now(),
    tasks: stateCopy
  };

  // Add to undo stack
  this.undoStack.push(state);

  // Limit stack size
  if (this.undoStack.length > this.maxUndoLevels) {
    this.undoStack.shift();
  }

  // Clear redo stack
  this.redoStack = [];

  // Update undo/redo buttons
  this.updateUndoRedoButtons();
},

/**
 * Load tasks from storage (ES5 compatible)
 */
loadTasks: function() {
  try {
    var storedTasks = localStorage.getItem('miniCycleLite_tasks');
    if (storedTasks) {
      this.tasks = JSON.parse(storedTasks);
    }
  } catch (error) {
    console.error('Failed to load tasks:', error);
    this.tasks = [];
  }
},

/**
 * Save tasks to storage (ES5 compatible)
 */
saveTasks: function() {
  try {
    localStorage.setItem('miniCycleLite_tasks', JSON.stringify(this.tasks));
  } catch (error) {
    console.error('Failed to save tasks:', error);
    this.showNotification('Failed to save tasks', 'error');
  }
},

/**
 * Simple notification system (ES5 compatible)
 */
```

```
  showNotification: function(message, type) {
    var notification = document.createElement('div');
    notification.className = 'lite-notification lite-notification-' + (type || 'info');
    notification.textContent = message;

    // Simple styling
    notification.style.position = 'fixed';
    notification.style.top = '20px';
    notification.style.left = '50%';
    notification.style.transform = 'translateX(-50%)';
    notification.style.background = type === 'error' ? '#e74c3c' :
                                    type === 'warning' ? '#f1c40f' :
                                    type === 'success' ? '#2ecc71' : '#3498db';
    notification.style.color = 'white';
    notification.style.padding = '10px 20px';
    notification.style.borderRadius = '5px';
    notification.style.zIndex = '1000';
    notification.style.fontSize = '14px';

    document.body.appendChild(notification);

    // Auto remove
    setTimeout(function() {
      if (notification.parentNode) {
        notification.parentNode.removeChild(notification);
      }
    }, 3000);
  }
};

// Initialize when DOM is ready
if (document.readyState === 'loading') {
  document.addEventListener('DOMContentLoaded', function() {
    MiniCycleLite.init();
  });
} else {
  MiniCycleLite.init();
}
```

## Development Guide

### Setup and Installation

#### Local Development Environment

```bash
# Clone or download the project
git clone [repository-url] minicycle
cd minicycle

# No build process required - pure HTML/CSS/JS
# Simply serve files with a local server

# Using Python 3
python -m http.server 8000

# Using Node.js http-server
npx http-server -p 8000

# Using PHP
php -S localhost:8000

# Access application at http://localhost:8000
```

#### Development Server Configuration

```javascript
// Optional: Development server with auto-reload
// server.js
const express = require('express');
const path = require('path');
const app = express();

// Serve static files
app.use(express.static('.'));

// Development middleware
if (process.env.NODE_ENV === 'development') {
  // Disable caching
  app.use((req, res, next) => {
    res.setHeader('Cache-Control', 'no-cache, no-store, must-revalidate');
    res.setHeader('Pragma', 'no-cache');
    res.setHeader('Expires', '0');
    next();
  });

  // Add development headers
  app.use((req, res, next) => {
    res.setHeader('X-Development-Mode', 'true');
    next();
  });
}

// Handle PWA routes
app.get('/miniCycle.html', (req, res) => {
  res.sendFile(path.join(__dirname, 'miniCycle.html'));
});

app.get('/miniCycle-lite.html', (req, res) => {
  res.sendFile(path.join(__dirname, 'miniCycle-lite.html'));
});

const port = process.env.PORT || 8000;
app.listen(port, () => {
  console.log(`🚀 miniCycle development server running on port ${port}`);
});
```

### Code Organization

#### File Naming Conventions

```
Components:
- miniCycle.html          # Main application
- miniCycle-lite.html     # Lightweight version
- miniCycle-scripts.js    # Core logic
- miniCycle-styles.css    # Main styles

Features:
- feature-name.js         # Feature-specific logic
- feature-name.css        # Feature-specific styles

Utilities:
- utilities/              # Helper functions and tools
- assets/                 # Static assets (images, icons)

Documentation:
```

```
- user-manual.html       # User documentation
- terms.html            # Legal documents
- privacy.html          # Privacy policy
```

#### Code Style Guidelines

```javascript
// Use descriptive function names
function createTaskElement(taskData) { }        // ✅ Good
function createElement(data) { }                // ❌ Too generic

// Use consistent naming conventions
const taskElement = document.createElement('li'); // ✅ camelCase for variables
const TASK_LIMIT = 50;                          // ✅ UPPER_CASE for constants
const TaskManager = { };                        // ✅ PascalCase for objects/classes

// Use meaningful comments
/**
 * Validates task input and sanitizes for XSS prevention
 * @param {string} input - Raw user input
 * @returns {object} Validation result with sanitized input
 */
function validateTaskInput(input) {
  // Implementation...
}

// Use consistent error handling
function risky_operation() {
  try {
    // Main logic
    return { success: true, data: result };
  } catch (error) {
    console.error('Operation failed:', error);
    return { success: false, error: error.message };
  }
}

// Use defensive programming
function updateTaskElement(taskElement, taskData) {
  if (!taskElement || !taskData) {
    console.warn('Invalid parameters for updateTaskElement');
    return false;
  }
  // Implementation...
  return true;
}
```

### Testing and Quality Assurance

#### Built-in Testing Utilities

```javascript
const TestingUtils = {
  /**
   * Comprehensive application testing suite
   */
  runAllTests() {
    console.log('🧪 Starting miniCycle Test Suite...');

    const results = {
      passed: 0,
      failed: 0,
      tests: []
    };

    // Core functionality tests
    this.testTaskManagement(results);
    this.testDataPersistence(results);
    this.testRecurringTasks(results);
    this.testUndoRedo(results);
    this.testInputValidation(results);
    this.testThemeSystem(results);

    // Generate report
    this.generateTestReport(results);

    return results;
  },

  /**
   * Test task management operations
   */
  testTaskManagement(results) {
    const tests = [
      {
```

```
      name: 'Add Task',
      test: () => {
        const initialCount = document.querySelectorAll('.task').length;
        addTask('Test Task', false, false);
        const newCount = document.querySelectorAll('.task').length;
        return newCount === initialCount + 1;
      }
    },
    {
      name: 'Complete Task',
      test: () => {
        const taskElement = document.querySelector('.task');
        if (!taskElement) return false;

        const checkbox = taskElement.querySelector('input[type="checkbox"]');
        const wasChecked = checkbox.checked;

        checkbox.click();

        return checkbox.checked !== wasChecked;
      }
    },
    {
      name: 'Delete Task',
      test: () => {
        const initialCount = document.querySelectorAll('.task').length;
        const firstTask = document.querySelector('.task');

        if (!firstTask) return initialCount === 0;

        const deleteBtn = firstTask.querySelector('.delete-btn');
        if (deleteBtn) deleteBtn.click();

        const newCount = document.querySelectorAll('.task').length;
        return newCount === initialCount - 1;
      }
    }
  ];

  this.executeTests('Task Management', tests, results);
},

/**
 * Test data persistence
 */
testDataPersistence(results) {
  const tests = [
    {
      name: 'Save to localStorage',
      test: () => {
        const testData = { test: 'persistence' };
        localStorage.setItem('miniCycle_test', JSON.stringify(testData));

        const retrieved = JSON.parse(localStorage.getItem('miniCycle_test'));
        localStorage.removeItem('miniCycle_test');

        return retrieved && retrieved.test === 'persistence';
      }
    },
    {
      name: 'Schema Migration',
      test: () => {
        // Test schema migration logic
        const oldSchema = {
          version: "1.0",
          tasks: [{ text: "Test", completed: false }]
        };

        const migrated = migrateToCurrentSchema(oldSchema);
        return migrated.version === "2.5" && migrated.data.cycles;
      }
    }
  ];

  this.executeTests('Data Persistence', tests, results);
},

/**
 * Execute a set of tests
 */
executeTests(category, tests, results) {
  console.log(`\n📋 Testing ${category}:`);

  tests.forEach(test => {
    try {
      const passed = test.test();
      const result = {
        category,
        name: test.name,
        passed,
        error: null
      };

      results.tests.push(result);

      if (passed) {
        results.passed++;
        console.log(`  ✅ ${test.name}`);
      } else {
        results.failed++;
        console.log(`  ❌ ${test.name}`);
      }
    } catch (error) {
      results.failed++;
      const result = {
        category,
        name: test.name,
        passed: false,
        error: error.message
      };

      results.tests.push(result);
      console.log(`  💥 ${test.name}: ${error.message}`);
    }
  });
},

/**
 * Generate test report
 */
generateTestReport(results) {
  const total = results.passed + results.failed;
  const passRate = total > 0 ? (results.passed / total * 100).toFixed(1) : 0;

  console.log('\n📊 Test Results Summary:');
  console.log(`Total Tests: ${total}`);
  console.log(`Passed: ${results.passed}`);
  console.log(`Failed: ${results.failed}`);
  console.log(`Pass Rate: ${passRate}%`);

  if (results.failed > 0) {
    console.log('\n❌ Failed Tests:');
```

```
      results.tests
        .filter(test => !test.passed)
        .forEach(test => {
          console.log(`  - ${test.category}: ${test.name}`);
          if (test.error) {
            console.log(`    Error: ${test.error}`);
          }
        });
    }

    return results;
  },

  /**
   * Performance benchmarking
   */
  benchmarkPerformance() {
    console.log('⚡ Running Performance Benchmarks...');

    const benchmarks = {
      taskCreation: this.benchmarkTaskCreation(),
      dataLoad: this.benchmarkDataLoad(),
      rendering: this.benchmarkRendering()
    };

    console.log('Benchmark Results:', benchmarks);
    return benchmarks;
  },

  benchmarkTaskCreation() {
    const iterations = 100;
    const start = performance.now();

    for (let i = 0; i < iterations; i++) {
      createTaskElement({
        id: `benchmark_${i}`,
        text: `Benchmark Task ${i}`,
        completed: false
      });
    }

    const end = performance.now();
    return {
      totalTime: end - start,
      averageTime: (end - start) / iterations,
      iterations
    };
  }
};

// Expose testing utilities globally for manual testing
window.TestingUtils = TestingUtils;
```

#### Manual Testing Checklist

```markdown
## Manual Testing Checklist

### Core Functionality
- [ ] Add new task
- [ ] Edit existing task
- [ ] Mark task as complete
- [ ] Delete task
- [ ] Drag and drop reordering
- [ ] Undo/redo operations

### Cycle Management
- [ ] Create new cycle
- [ ] Switch between cycles
- [ ] Auto-reset functionality
- [ ] Manual reset
- [ ] Export cycle
- [ ] Import cycle

### Recurring Tasks
- [ ] Set up daily recurring task
- [ ] Set up weekly recurring task
- [ ] Set up monthly recurring task
- [ ] Modify recurring settings
- [ ] Delete recurring task

### UI/UX
- [ ] Responsive design on mobile
- [ ] Responsive design on tablet
- [ ] Responsive design on desktop
- [ ] Dark mode toggle
- [ ] Theme unlocking
```

- [ ] Modal interactions

### Data & Storage
- [ ] Data persistence across sessions
- [ ] Export functionality
- [ ] Import functionality
- [ ] Data migration
- [ ] Error handling for storage failures

### Accessibility
- [ ] Keyboard navigation
- [ ] Screen reader compatibility
- [ ] Focus management
- [ ] ARIA labels
- [ ] Color contrast

### Performance
- [ ] Load time under 3 seconds
- [ ] Smooth animations
- [ ] No memory leaks
- [ ] Efficient storage usage
~~~

Deployment

Production Build Process

File Optimization

```bash
#!/bin/bash
# build.sh - Production build script

echo "🏗️  Building miniCycle for production..."

# Create build directory
mkdir -p dist

# Copy HTML files
cp miniCycle.html dist/
cp miniCycle-lite.html dist/
cp user-manual.html dist/
~~~

```
cp terms.html dist/
cp privacy.html dist/
cp product.html dist/

# Minify CSS (requires csso or similar)
echo "📦 Minifying CSS..."
csso miniCycle-styles.css > dist/miniCycle-styles.min.css
csso miniCycle-lite-styles.css > dist/miniCycle-lite-styles.min.css
csso user-manual-styles.css > dist/user-manual-styles.min.css

# Minify JavaScript (requires terser or similar)
echo "📦 Minifying JavaScript..."
terser miniCycle-scripts.js -o dist/miniCycle-scripts.min.js --compress --mangle
terser miniCycle-lite-scripts.js -o dist/miniCycle-lite-scripts.min.js --compress --mangle

# Copy assets
cp -r assets dist/
cp manifest.json dist/
cp manifest-lite.json dist/

# Update file references in HTML
sed -i 's/miniCycle-styles.css/miniCycle-styles.min.css/g' dist/miniCycle.html
sed -i 's/miniCycle-scripts.js/miniCycle-scripts.min.js/g' dist/miniCycle.html
sed -i 's/miniCycle-lite-styles.css/miniCycle-lite-styles.min.css/g' dist/miniCycle-lite.html
sed -i 's/miniCycle-lite-scripts.js/miniCycle-lite-scripts.min.js/g' dist/miniCycle-lite.html

# Generate service worker
echo "⚙️  Generating service worker..."
node generate-sw.js

echo "✅ Build completed! Files are in dist/ directory"
```

#### Service Worker Implementation

```javascript
// sw.js - Service Worker for offline functionality
const CACHE_NAME = 'minicycle-v1.275';
const ASSETS_TO_CACHE = [
  '/',
  '/miniCycle.html',
  '/miniCycle-lite.html',
  '/miniCycle-styles.min.css',
  '/miniCycle-lite-styles.min.css',
  '/miniCycle-scripts.min.js',
  '/miniCycle-lite-scripts.min.js',
  '/user-manual.html',
  '/assets/images/logo/minicycle_logo_icon.png',
  '/manifest.json'
];

// Install event - cache assets
self.addEventListener('install', (event) => {
  console.log('🔧 Service Worker installing...');

  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => {
        console.log('📦 Caching app assets');
        return cache.addAll(ASSETS_TO_CACHE);
      })
      .then(() => {
        console.log('✅ Service Worker installed');
        return self.skipWaiting();
      })
  );
});

// Activate event - clean old caches
self.addEventListener('activate', (event) => {
  console.log('🚀 Service Worker activating...');

  event.waitUntil(
    caches.keys()
      .then((cacheNames) => {
        return Promise.all(
          cacheNames.map((cacheName) => {
            if (cacheName !== CACHE_NAME) {
              console.log('🗑️ Deleting old cache:', cacheName);
              return caches.delete(cacheName);
            }
          })
        );
      })
      .then(() => {
        console.log('✅ Service Worker activated');
        return self.clients.claim();
      })
  );
});

// Fetch event - serve from cache, fallback to network
self.addEventListener('fetch', (event) => {
  // Skip non-GET requests
  if (event.request.method !== 'GET') return;

  // Skip cross-origin requests
  if (!event.request.url.startsWith(self.location.origin)) return;

  event.respondWith(
    caches.match(event.request)
      .then((cachedResponse) => {
        // Return cached version if available
        if (cachedResponse) {
          return cachedResponse;
        }

        // Fetch from network
        return fetch(event.request)
          .then((networkResponse) => {
            // Cache successful responses
            if (networkResponse.status === 200) {
              const responseClone = networkResponse.clone();

              caches.open(CACHE_NAME)
                .then((cache) => {
                  cache.put(event.request, responseClone);
                });
            }

            return networkResponse;
          })
          .catch(() => {
            // Return offline page for navigation requests
            if (event.request.mode === 'navigate') {
              return caches.match('/miniCycle-lite.html');
            }

            // Return placeholder for other requests
            return new Response('Offline', {
              status: 503,
              statusText: 'Service Unavailable'
          });
        });
      })
  );
});

// Background sync for data backup
self.addEventListener('sync', (event) => {
  console.log('🔄 Background sync triggered:', event.tag);

  if (event.tag === 'backup-data') {
    event.waitUntil(
      // Attempt to backup data when connection is restored
      performDataBackup()
    );
  }
});

async function performDataBackup() {
  try {
    // Get data from IndexedDB or localStorage
    const data = await getAllStoredData();

    // Attempt to send to backup service
    const response = await fetch('/api/backup', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
  weekly: {
    days: ["Monday", "Wednesday", "Friday"]
  },
  monthly: {
    days: [1, 15, 30]
  },
  yearly: {
    months: [1, 6, 12],
    daysByMonth: {
      1: [1], // January 1st
      6: [15], // June 15th
      12: [25] // December 25th
    }
  },
  specificDates: {
```

    enabled: false,
    dates: []
  }
};

#### Recurring Task Lifecycle

Template Management

- Recurring tasks stored as templates
- Separation from active task instances
- Template modification affects future instances
- Historical instance preservation

Recreation Logic

```javascript
function shouldTaskRecurNow(settings, currentTime) {
  // Complex logic evaluating:
  // - Current time vs last completion
  // - Frequency requirements
  // - Schedule constraints
  // - Time zone considerations
  return boolean;
}
```

### 4. Undo/Redo System

#### State Management

Snapshot Architecture

- Deep cloning of task and template state
- 4-level undo history limit
- Memory-efficient storage
- Automatic cleanup of old snapshots

Operations Tracking

```javascript
const undoSnapshot = {
  tasks: structuredClone(currentTasks),
```

  recurringTemplates: structuredClone(templates),
  title: currentCycleTitle,
  timestamp: Date.now(),
  action: "add_task" // or "delete_task", "edit_task", etc.
};

Keyboard Shortcuts

- `Ctrl+Z` / `Cmd+Z`: Undo last action
- `Ctrl+Y` / `Cmd+Y`: Redo action
- `Ctrl+Shift+Z` / `Cmd+Shift+Z`: Alternative redo

### 5. Theme & Customization System

#### Theme Architecture

Base Themes

- **Default**: Blue gradient with modern styling
- **Dark Mode**: High-contrast dark theme
- **Dark Ocean**: Unlockable deep blue theme (5 cycles)
- **Golden Glow**: Unlockable warm theme (50 cycles)

Theme Implementation

```css
:root {
  --primary-color: #4c79ff;
  --secondary-color: #74c0fc;
  --text-color: #ffffff;
  --background-gradient: linear-gradient(135deg, #4c79ff, #74c0fc);
}

body.theme-dark-ocean {
  --primary-color: #0e1d2f;
  --secondary-color: #152b3c;
  --accent-color: #4ea1ff;
  --text-color: #ffffff;
}
```

#### Gamification Elements

Milestone System

- Cycle completion tracking
- Achievement badges (5, 25, 50, 75, 100+ cycles)
- Progress visualization
- Reward unlocking mechanism

Progress Tracking

```javascript
const progressMetrics = {
  totalTasks: number,
  completedTasks: number,
  completionRate: percentage,
  cyclesCompleted: number,
  currentStreak: number,
  longestStreak: number,
  themeUnlocks: array
};
```

## Technical Implementation

### JavaScript Architecture

#### Core Application Structure

Initialization Sequence

```javascript
document.addEventListener('DOMContentLoaded', (event) => {
  // 1. Device detection and version selection
  detectAndLoadAppropriateVersion();

  // 2. Core system initialization
  initialSetup();
  loadRemindersSettings();
  setupReminderToggle();

  // 3. UI component setup
  setupMainMenu();
  setupSettingsMenu();
  setupRecurringPanel();
  setupThemeSystem();

  // 4. Event listener binding
  bindEventListeners();

  // 5. Data loading and migration
  migrateAndLoadData();

  // 6. Final UI updates
  updateProgressBar();
  checkCompleteAllButton();

  // 7. Background services
  startRecurringTaskWatcher();

  window.AppReady = true;
});
```

#### Event Management System

Safe Event Listener Pattern

```javascript
function safeAddEventListener(element, event, handler) {
  if (!element) return;

  // Remove existing listener to prevent duplicates
  element.removeEventListener(event, handler);

  // Add fresh listener
  element.addEventListener(event, handler);
}

function safeAddEventListenerById(id, event, handler) {
  const element = document.getElementById(id);
  if (element) {
    safeAddEventListener(element, event, handler);
  } else {
    console.warn(`⚠ Cannot attach event listener: #${id} not found.`);
  }
}
```

#### Error Handling Strategy

Global Error Capture

```javascript
window.addEventListener('error', function(e) {
  console.error('💥 JavaScript Error:', e.error);
  showNotification('An error occurred. Please refresh the page.', 'error');

  // Optional: Send error to analytics
  logErrorToAnalytics(e.error);
});

window.addEventListener('unhandledrejection', function(e) {
  console.error('💥 Unhandled Promise Rejection:', e.reason);
  showNotification('An error occurred. Please try again.', 'error');
});
```

### Data Management

#### Schema Version System

Current Schema: 2.5

```javascript
const SCHEMA_VERSION = 2.5;

const schemaStructure = {
  version: "2.5",
  metadata: {
    created: "2025-09-16T10:00:00Z",
    lastModified: "2025-09-16T10:00:00Z",
    appVersion: "1.275"
  },
  data: {
    activeCycle: "Default Cycle",
    cycles: {
      "Cycle Name": {
        title: "Display Title",
        tasks: [],
        recurringTemplates: {},
        settings: {
          autoReset: true,
          deleteCheckedTasks: false,
          threeDotsEnabled: true,
          moveArrowsEnabled: true
        },
        statistics: {
          cycleCount: 0,
          totalCompletions: 0,
          createdAt: "2025-09-16T10:00:00Z"
        }
      }
    }
  }
};
```

Migration System

```javascript
function migrateToSchema25() {
  const oldData = localStorage.getItem('miniCycle_savedMiniCycles');
  if (!oldData) return;

  try {
    const parsed = JSON.parse(oldData);
    const newSchema = {
      version: "2.5",
      metadata: {
        created: new Date().toISOString(),
        lastModified: new Date().toISOString(),
        appVersion: "1.275"
      },
      data: {
        activeCycle: localStorage.getItem('miniCycle_lastUsedMiniCycle') || 'Default',
        cycles: migrateCycleData(parsed)
      }
    };

    localStorage.setItem('miniCycleData', JSON.stringify(newSchema));
    console.log('✅ Migration to Schema 2.5 completed');
  } catch (error) {
    console.error('❌ Migration failed:', error);
  }
}
```

#### Storage Operations

Auto-Save System

```javascript
function autoSave(overrideTaskList = null) {
  try {
    const schemaData = loadMiniCycleData();
    if (!schemaData) return;

    const { cycles, activeCycle } = schemaData;
    const currentCycle = cycles[activeCycle];

    if (!currentCycle) return;

    // Update task data
    const taskElements = overrideTaskList || document.querySelectorAll('#taskList .task');
    currentCycle.tasks = Array.from(taskElements).map(extractTaskData);

    // Update metadata
    schemaData.metadata.lastModified = new Date().toISOString();

    // Save to localStorage
    localStorage.setItem('miniCycleData', JSON.stringify(schemaData));

    console.log('💾 Auto-save completed');
  } catch (error) {
    console.error('❌ Auto-save failed:', error);
    showNotification('Save failed. Please try again.', 'error');
  }
}
```

### Performance Optimization

#### Memory Management

Event Listener Cleanup

```javascript
function cleanupTaskEventListeners(taskElement) {
  const buttons = taskElement.querySelectorAll('button');
  buttons.forEach(button => {
    // Clone node to remove all event listeners
    const newButton = button.cloneNode(true);
    button.parentNode.replaceChild(newButton, button);
  });
}

function removeTask(taskElement) {
  // Clean up event listeners
  cleanupTaskEventListeners(taskElement);

  // Remove from DOM
  taskElement.remove();

  // Update storage
  autoSave();

  // Update UI
  updateProgressBar();
  checkCompleteAllButton();
}
```

Throttling and Debouncing

```javascript
function throttle(func, limit) {
  let inThrottle;
  return function() {
    const args = arguments;
    const context = this;
    if (!inThrottle) {
      func.apply(context, args);
      inThrottle = true;
      setTimeout(() => inThrottle = false, limit);
    }
  };
}

// Usage for drag operations
const throttledDragHandler = throttle(handleDragMove, 50);
```

#### Efficient DOM Operations

Batch DOM Updates

```javascript
function updateTaskList(tasks) {
  const fragment = document.createDocumentFragment();

  tasks.forEach(task => {
    const taskElement = createTaskElement(task);
    fragment.appendChild(taskElement);
  });

  // Single DOM operation
  const taskList = document.getElementById('taskList');
  taskList.innerHTML = '';
  taskList.appendChild(fragment);
}
```

## API Documentation

### Core Functions

#### Task Management

addTask()

```javascript
/**
 * Adds a new task to the current cycle
 * @param {string} taskText - The task description
 * @param {boolean} completed - Initial completion state
 * @param {boolean} shouldSave - Whether to auto-save after adding
 * @param {string|null} dueDate - ISO date string for due date
 * @param {boolean} highPriority - Priority flag
 * @param {boolean} isLoading - Whether this is a loading operation
 * @param {boolean} remindersEnabled - Reminder state
 * @param {boolean} recurring - Whether task is recurring
 * @param {string|null} taskId - Specific task ID (optional)
 * @param {object} recurringSettings - Recurring configuration
 * @returns {HTMLElement|null} Created task element
 */
```

```
function addTask(taskText, completed = false, shouldSave = true,
                dueDate = null, highPriority = null, isLoading = false,
                remindersEnabled = false, recurring = false,
                taskId = null, recurringSettings = {}) {
  // Implementation details...
}
```

editTask()

```javascript
/**
 * Edits an existing task
 * @param {HTMLElement} taskElement - Task DOM element
 * @param {string} newText - New task text
 * @returns {boolean} Success status
 */
function editTask(taskElement, newText) {
  if (!taskElement || !newText.trim()) return false;

  const sanitizedText = sanitizeInput(newText);
  const taskTextElement = taskElement.querySelector('.task-text');

  if (taskTextElement) {
    taskTextElement.textContent = sanitizedText;
    autoSave();
    return true;
  }

  return false;
}
```

#### Cycle Management

createNewMiniCycle()

```javascript
/**
 * Creates a new mini cycle
 * @param {string} cycleName - Name for the new cycle
 * @param {object} options - Configuration options
 * @returns {boolean} Success status
 */
```

```
function createNewMiniCycle(cycleName, options = {}) {
  const defaults = {
    autoReset: true,
    deleteCheckedTasks: false,
    title: cycleName
  };

  const config = { ...defaults, ...options };

  // Implementation details...
}
```

switchToMiniCycle()

```javascript
/**
 * Switches to a different mini cycle
 * @param {string} cycleName - Target cycle name
 * @returns {boolean} Success status
 */
function switchToMiniCycle(cycleName) {
  const schemaData = loadMiniCycleData();
  if (!schemaData || !schemaData.data.cycles[cycleName]) {
    return false;
  }

  // Save current state
  autoSave();

  // Update active cycle
  schemaData.data.activeCycle = cycleName;
  localStorage.setItem('miniCycleData', JSON.stringify(schemaData));

  // Reload UI
  loadMiniCycle();

  return true;
}
```

#### Recurring Tasks

setupRecurringTask()

```javascript
/**
 * Configures a task for recurring behavior
 * @param {HTMLElement} taskElement - Task to make recurring
 * @param {object} settings - Recurring configuration
 * @returns {boolean} Success status
 */
function setupRecurringTask(taskElement, settings) {
  const taskData = extractTaskData(taskElement);

  // Validate recurring settings
  const normalizedSettings = normalizeRecurringSettings(settings);
  if (!normalizedSettings) return false;

  // Update task properties
  taskData.recurring = true;
  taskData.recurringSettings = normalizedSettings;

  // Save as template
  saveRecurringTemplate(taskData);

  // Update UI
  updateTaskElement(taskElement, taskData);

  return true;
}
```

#### Data Management

exportMiniCycle()

```javascript
/**
 * Exports a cycle to .mcyc file format
 * @param {string} cycleName - Cycle to export
 * @returns {object|null} Export data or null if failed
 */
function exportMiniCycle(cycleName) {
  const schemaData = loadMiniCycleData();
  if (!schemaData || !schemaData.data.cycles[cycleName]) {
    return null;
  }
  const cycleData = schemaData.data.cycles[cycleName];
  const exportData = {
    version: "1.275",
    exportDate: new Date().toISOString(),
    cycleName: cycleName,
    data: structuredClone(cycleData)
  };

  return exportData;
}
```

### Utility Functions

#### Input Validation

sanitizeInput()

```javascript
/**
 * Sanitizes user input to prevent XSS attacks
 * @param {string} input - Raw user input
 * @returns {string} Sanitized input
 */
function sanitizeInput(input) {
  if (typeof input !== "string") return "";

  // Create temporary element for text content extraction
  const temp = document.createElement("div");
  temp.textContent = input;

  // Return cleaned and length-limited text
  return temp.textContent.trim().substring(0, 50);
}
```

validateTaskData()

```javascript
/**
 * Validates task data structure
 * @param {object} taskData - Task data to validate
 * @returns {boolean} Validation result
 */
function validateTaskData(taskData) {
  const required = ['id', 'text', 'completed'];
  const hasRequired = required.every(field => taskData.hasOwnProperty(field));

  if (!hasRequired) return false;

  // Additional validation
  if (typeof taskData.text !== 'string' || taskData.text.length === 0) return false;
  if (typeof taskData.completed !== 'boolean') return false;

  return true;
}
```

#### Device Detection

detectDeviceType()

```javascript
/**
 * Detects device capabilities and type
 * @returns {object} Device information
 */
function detectDeviceType() {
  const hasTouch = "ontouchstart" in window;
  const touchPoints = navigator.maxTouchPoints || navigator.msMaxTouchPoints || 0;
  const finePointer = window.matchMedia("(pointer: fine)").matches;
  const userAgent = navigator.userAgent.toLowerCase();

  return {
    isMobile: hasTouch && touchPoints > 0 && !finePointer,
    isTablet: hasTouch && touchPoints > 1 && window.innerWidth >= 768,
    isDesktop: finePointer && !hasTouch,
    touchCapable: hasTouch,
    maxTouchPoints: touchPoints,
    isOldBrowser: isOldBrowser(),
    shouldUseLite: shouldUseLiteVersion()
  };
}
```

## User Interface

### Component Architecture

#### Modal System

Base Modal Structure

```html
<div class="modal-overlay" id="modal-overlay">
  <div class="modal-panel" role="dialog" aria-labelledby="modal-title">
    <header class="modal-header">
      <h2 id="modal-title">Modal Title</h2>
      <button class="close-modal" aria-label="Close modal">×</button>
    </header>
    <main class="modal-content">
      <!-- Modal-specific content -->
    </main>
    <footer class="modal-footer">
      <!-- Action buttons -->
    </footer>
  </div>
</div>
```

Modal Management

```javascript
const modalManager = {
  open(modalId) {
    const modal = document.getElementById(modalId);
    if (!modal) return;

    modal.style.display = 'flex';
    modal.setAttribute('aria-hidden', 'false');

    // Focus management
    const firstFocusable = modal.querySelector('button, input, select, textarea');
    if (firstFocusable) firstFocusable.focus();

    // Prevent body scroll
    document.body.style.overflow = 'hidden';
  },
  close(modalId) {
    const modal = document.getElementById(modalId);
    if (!modal) return;

    modal.style.display = 'none';
    modal.setAttribute('aria-hidden', 'true');

    // Restore body scroll
    document.body.style.overflow = '';

    // Return focus to trigger element
    const returnFocus = document.querySelector('[data-modal-trigger="' + modalId + '"]');
    if (returnFocus) returnFocus.focus();
  }
};
```

#### Task Component

Task Element Creation

```javascript
function createTaskElement(taskData) {
  const taskElement = document.createElement('li');
  taskElement.className = 'task';
  taskElement.setAttribute('data-task-id', taskData.id);
  taskElement.setAttribute('draggable', 'true');

  // Priority styling
  if (taskData.priority) {
    taskElement.classList.add('high-priority');
  }

  // Due date styling
  if (taskData.dueDate && new Date(taskData.dueDate) < new Date()) {
    taskElement.classList.add('overdue');
  }

  taskElement.innerHTML = `
    <div class="task-options">
      <button class="task-btn move-up" aria-label="Move task up">▲</button>
      <button class="task-btn move-down" aria-label="Move task down">▼</button>
      <button class="task-btn recurring-btn" aria-label="Set recurring">🔁</button>
      <button class="task-btn set-due-date" aria-label="Set due date">📅</button>
      <button class="task-btn enable-task-reminders" aria-label="Enable reminders">🔔</button>
      <button class="task-btn priority-btn" aria-label="Toggle priority">⚠️</button>
      <button class="task-btn edit-btn" aria-label="Edit task">✏️</button>
      <button class="task-btn delete-btn" aria-label="Delete task">🗑️</button>
    </div>

    <div class="task-content">
      <input type="checkbox" id="checkbox-${taskData.id}"
             ${taskData.completed ? 'checked' : ''}>
      <label for="checkbox-${taskData.id}" class="task-text">${taskData.text}</label>
    </div>

    ${taskData.dueDate ? `
      <div class="due-date-display">
        Due: ${formatDate(taskData.dueDate)}
      </div>
    ` : ''}
  `;

  // Bind event listeners
  bindTaskEventListeners(taskElement);

  return taskElement;
}
```

### Responsive Design

#### Breakpoint System

```css
/* Mobile First Approach */
:root {
  --mobile-max: 767px;
  --tablet-min: 768px;
  --tablet-max: 1023px;
  --desktop-min: 1024px;
}

/* Base styles (Mobile) */
.task-view {
  width: 95%;
  max-width: 400px;
  padding: 10px;
}

/* Tablet */
@media (min-width: 768px) {
  .task-view {
    width: 80%;
    max-width: 500px;
    padding: 15px;
  }

  .task-options {
    opacity: 0;
    transition: opacity 0.2s ease;
  }

  .task:hover .task-options {
    opacity: 1;
  }
}

/* Desktop */
@media (min-width: 1024px) {
  .task-view {
    width: 70%;
    max-width: 600px;
    padding: 20px;
  }

  .drag-handle {
    display: block;
  }
}
```

#### Safe Area Support

```css
body {
  padding-top: env(safe-area-inset-top);
  padding-bottom: env(safe-area-inset-bottom);
  padding-left: env(safe-area-inset-left);
  padding-right: env(safe-area-inset-right);
}
```

```
.header {
  padding-top: calc(20px + env(safe-area-inset-top));
}

.footer {
  padding-bottom: calc(20px + env(safe-area-inset-bottom));
}
```

### Accessibility Features

#### ARIA Implementation

```html
<!-- Screen reader announcements -->
<div id="live-region" aria-live="polite" class="sr-only"></div>

<!-- Proper labeling -->
<button aria-label="Add new task" aria-describedby="task-input-help">
  Add Task
</button>
<div id="task-input-help" class="sr-only">
  Enter a task description and press Add or Enter key
</div>

<!-- Modal accessibility -->
<div role="dialog" aria-labelledby="modal-title" aria-modal="true">
  <h2 id="modal-title">Settings</h2>
</div>

<!-- Progress indication -->
<div role="progressbar" aria-valuenow="75" aria-valuemin="0"
     aria-valuemax="100" aria-label="Task completion progress">
  75% Complete
</div>
```

#### Keyboard Navigation

```javascript
function setupKeyboardNavigation() {
  // Global shortcuts
  document.addEventListener('keydown', (e) => {
    // Escape key closes modals
    if (e.key === 'Escape') {
      closeActiveModal();
    }

    // Ctrl/Cmd + Enter adds task
    if ((e.ctrlKey || e.metaKey) && e.key === 'Enter') {
      const taskInput = document.getElementById('taskInput');
      if (taskInput.value.trim()) {
        handleAddTask();
      }
    }

    // Undo/Redo shortcuts
    if ((e.ctrlKey || e.metaKey) && e.key === 'z' && !e.shiftKey) {
      e.preventDefault();
      performUndo();
    }

    if ((e.ctrlKey || e.metaKey) && (e.key === 'y' || (e.key === 'z' && e.shiftKey))) {
      e.preventDefault();
      performRedo();
    }
  });

  // Task navigation
  setupTaskKeyboardNavigation();
}

function setupTaskKeyboardNavigation() {
  document.addEventListener('keydown', (e) => {
    const focusedTask = document.querySelector('.task:focus-within');
    if (!focusedTask) return;

    switch(e.key) {
      case 'Space':
        e.preventDefault();
        toggleTaskCompletion(focusedTask);
        break;
      case 'Delete':
        e.preventDefault();
        deleteTask(focusedTask);
        break;
      case 'ArrowUp':
        e.preventDefault();
        focusPreviousTask(focusedTask);
        break;
      case 'ArrowDown':
        e.preventDefault();
        focusNextTask(focusedTask);
        break;
    }
  });
}
```

#### Screen Reader Support

```javascript
function announceToScreenReader(message) {
  const liveRegion = document.getElementById('live-region');
  if (liveRegion) {
    liveRegion.textContent = message;

    // Clear after announcement
    setTimeout(() => {
      liveRegion.textContent = '';
    }, 1000);
  }
}

// Usage examples
function addTask(taskText) {
  // ... task creation logic ...
  announceToScreenReader(`Task added: ${taskText}`);
}

function completeTask(taskElement) {
  // ... completion logic ...
  const taskText = taskElement.querySelector('.task-text').textContent;
  announceToScreenReader(`Task completed: ${taskText}`);
}
```

## Data Management

### Storage Architecture

#### LocalStorage Structure

Primary Storage Key: `miniCycleData`

```javascript
{
  "version": "2.5",
  "metadata": {
    "created": "2025-09-16T10:00:00Z",
    "lastModified": "2025-09-16T12:30:00Z",
    "appVersion": "1.275",
    "deviceInfo": {
      "userAgent": "Mozilla/5.0...",
      "viewport": "390x844",
      "touchCapable": true
    }
  },
  "data": {
    "activeCycle": "Work Tasks",
    "cycles": {
      "Work Tasks": {
        "title": "Daily Work Routine",
        "tasks": [
          {
            "id": "task_1694865600000_abc123",
            "text": "Check emails",
            "completed": false,
            "priority": false,
            "dueDate": null,
            "remindersEnabled": false,
            "recurring": true,
            "recurringSettings": {
              "frequency": "daily",
              "indefinitely": true,
              "time": {"hour": 9, "minute": 0, "meridiem": "AM"}
            },
            "schemaVersion": 2,
            "createdAt": "2025-09-16T10:00:00Z",
            "completedAt": null
          }
        ],
        "recurringTemplates": {
          "template_check_emails": {
```

```
          "text": "Check emails",
          "settings": {
            "frequency": "daily",
            "time": {"hour": 9, "minute": 0}
          },
          "createdAt": "2025-09-16T10:00:00Z"
        }
      },
      "settings": {
        "autoReset": true,
        "deleteCheckedTasks": false,
        "threeDotsEnabled": true,
        "moveArrowsEnabled": true,
        "darkMode": false,
        "currentTheme": "default"
      },
      "statistics": {
        "cycleCount": 15,
        "totalCompletions": 156,
        "createdAt": "2025-09-01T10:00:00Z",
        "lastCompletedAt": "2025-09-15T18:30:00Z",
        "averageCompletionTime": 45.6,
        "completionHistory": []
      }
    }
  }
},
"preferences": {
  "reminders": {
    "enabled": true,
    "frequency": "daily",
    "time": {"hour": 20, "minute": 0}
  },
  "notifications": {
    "position": {"x": 50, "y": 20},
    "duration": 3000
  },
  "ui": {
    "animationsEnabled": true,
    "soundEnabled": false,
    "compactMode": false
  }
}
}
```

#### Migration System

Schema Version History

```javascript
const SCHEMA_MIGRATIONS = {
  "1.0": {
    description: "Initial schema with basic task storage",
    migrate: migrateFromV1
  },
  "2.0": {
    description: "Added recurring tasks and cycle management",
    migrate: migrateFromV2
  },
  "2.5": {
    description: "Unified storage structure with metadata",
    migrate: migrateFromV2_5
  }
};

function performMigration(currentVersion, targetVersion) {
  console.log(`🔄 Migrating from ${currentVersion} to ${targetVersion}`);

  let data = loadCurrentData();
  const versions = Object.keys(SCHEMA_MIGRATIONS);
  const startIndex = versions.indexOf(currentVersion);
  const endIndex = versions.indexOf(targetVersion);

  for (let i = startIndex + 1; i <= endIndex; i++) {
    const version = versions[i];
    const migration = SCHEMA_MIGRATIONS[version];

    console.log(`📦 Applying migration: ${migration.description}`);
    data = migration.migrate(data);
  }

  return data;
}
```

Data Validation

```javascript
function validateSchemaData(data) {
  const errors = [];

  // Version validation
  if (!data.version || typeof data.version !== 'string') {
    errors.push('Missing or invalid version');
  }

  // Metadata validation
  if (!data.metadata || typeof data.metadata !== 'object') {
    errors.push('Missing metadata');
  }

  // Data structure validation
  if (!data.data || !data.data.cycles) {
    errors.push('Missing cycles data');
  }

  // Cycle validation
  Object.entries(data.data.cycles).forEach(([cycleName, cycle]) => {
    if (!Array.isArray(cycle.tasks)) {
      errors.push(`Invalid tasks array in cycle: ${cycleName}`);
    }

    cycle.tasks.forEach((task, index) => {
      if (!validateTaskData(task)) {
        errors.push(`Invalid task at index ${index} in cycle: ${cycleName}`);
      }
    });
  });

  return {
    valid: errors.length === 0,
    errors: errors
  };
}
```

### Backup and Export System

#### Export Functionality

.mcyc File Format

```javascript
function generateMCYCFile(cycleName) {
  const schemaData = loadMiniCycleData();
  if (!schemaData || !schemaData.data.cycles[cycleName]) {
    throw new Error(`Cycle '${cycleName}' not found`);
  }

  const cycleData = schemaData.data.cycles[cycleName];
  const exportData = {
    // File metadata
    fileVersion: "1.0",
    exportedBy: "miniCycle v1.275",
    exportDate: new Date().toISOString(),

    // Cycle information
    cycleName: cycleName,
    cycleTitle: cycleData.title,

    // Core data
    tasks: cycleData.tasks.map(sanitizeTaskForExport),
    recurringTemplates: cycleData.recurringTemplates,
    settings: cycleData.settings,
    statistics: {
      cycleCount: cycleData.statistics.cycleCount,
      totalCompletions: cycleData.statistics.totalCompletions,
      createdAt: cycleData.statistics.createdAt
    },

    // Checksum for integrity
    checksum: generateChecksum(cycleData)
  };

  return JSON.stringify(exportData, null, 2);
}

function sanitizeTaskForExport(task) {
  return {
    text: task.text,
    priority: task.priority || false,
    dueDate: task.dueDate,
    remindersEnabled: task.remindersEnabled || false,
    recurring: task.recurring || false,
    recurringSettings: task.recurringSettings || {},
    createdAt: task.createdAt
  };
}
```

#### Import Functionality

```javascript
function importMCYCFile(fileContent, options = {}) {
  try {
    const importData = JSON.parse(fileContent);

    // Validate file format
    if (!validateMCYCFile(importData)) {
      throw new Error('Invalid .mcyc file format');
    }

    // Check for conflicts
    const cycleName = options.newName || importData.cycleName;
    if (cycleExists(cycleName) && !options.overwrite) {
      throw new Error(`Cycle '${cycleName}' already exists`);
    }

    // Create cycle structure
    const newCycle = {
      title: importData.cycleTitle,
      tasks: importData.tasks.map(task => ({
        ...task,
        id: generateTaskId(),
        completed: false, // Reset completion status
        schemaVersion: 2
      })),
      recurringTemplates: importData.recurringTemplates || {},
      settings: {
        ...defaultCycleSettings,
        ...importData.settings
      },
      statistics: {
        cycleCount: 0, // Reset statistics
        totalCompletions: 0,
        createdAt: new Date().toISOString(),
        importedFrom: {
          originalName: importData.cycleName,
          exportDate: importData.exportDate,
          originalStats: importData.statistics
        }
      }
    };

    // Save to storage
    const schemaData = loadMiniCycleData();
    schemaData.data.cycles[cycleName] = newCycle;
    schemaData.metadata.lastModified = new Date().toISOString();

    localStorage.setItem('miniCycleData', JSON.stringify(schemaData));

    return {
      success: true,
      cycleName: cycleName,
      tasksImported: newCycle.tasks.length,
      templatesImported: Object.keys(newCycle.recurringTemplates).length
    };

  } catch (error) {
    return {
      success: false,
      error: error.message
    };
  }
}
```

## Performance & Optimization

### Memory Management

#### Event Listener Optimization

```javascript
class EventManager {
  constructor() {
    this.listeners = new Map();
  }

  add(element, event, handler, options = {}) {
    const key = `${element.id || 'anonymous'}_${event}`;
    // Remove existing listener if present
    this.remove(element, event);

    // Add new listener
    element.addEventListener(event, handler, options);

    // Store reference for cleanup
    this.listeners.set(key, { element, event, handler });
  }

  remove(element, event) {
    const key = `${element.id || 'anonymous'}_${event}`;
    const listener = this.listeners.get(key);

    if (listener) {
      listener.element.removeEventListener(listener.event, listener.handler);
      this.listeners.delete(key);
    }
  }

  cleanup() {
    this.listeners.forEach(listener => {
      listener.element.removeEventListener(listener.event, listener.handler);
    });
    this.listeners.clear();
  }
}

// Global event manager instance
const eventManager = new EventManager();
```

#### DOM Optimization

```javascript
class DOMOptimizer {
  constructor() {
    this.pendingUpdates = new Set();
    this.rafId = null;
  }

  scheduleUpdate(updateFunction) {
    this.pendingUpdates.add(updateFunction);
    if (!this.rafId) {
      this.rafId = requestAnimationFrame(() => {
        this.processPendingUpdates();
      });
    }
  }

  processPendingUpdates() {
    // Batch DOM reads first
    const reads = [];
    const writes = [];

    this.pendingUpdates.forEach(update => {
      if (update.type === 'read') {
        reads.push(update);
      } else {
        writes.push(update);
      }
    });

    // Execute all reads first to avoid layout thrashing
    reads.forEach(read => read.execute());

    // Then execute all writes
    writes.forEach(write => write.execute());

    // Clear pending updates
    this.pendingUpdates.clear();
    this.rafId = null;
  }
}

// Usage example
const domOptimizer = new DOMOptimizer();

function updateTaskProgress() {
  domOptimizer.scheduleUpdate({
    type: 'write',
    execute: () => {
      const progressBar = document.getElementById('progressBar');
      progressBar.style.width = calculateProgress() + '%';
    }
  });
```

```
}
```

### Rendering Optimization

#### Virtual Scrolling for Large Task Lists

```javascript
class VirtualTaskList {
  constructor(container, itemHeight = 60) {
    this.container = container;
    this.itemHeight = itemHeight;
    this.visibleItems = Math.ceil(container.clientHeight / itemHeight) + 2;
    this.tasks = [];
    this.scrollTop = 0;

    this.setupScrollListener();
  }

  setTasks(tasks) {
    this.tasks = tasks;
    this.render();
  }

  setupScrollListener() {
    this.container.addEventListener('scroll',
      throttle(() => {
        this.scrollTop = this.container.scrollTop;
        this.render();
      }, 16) // ~60fps
    );
  }

  render() {
    const startIndex = Math.floor(this.scrollTop / this.itemHeight);
    const endIndex = Math.min(startIndex + this.visibleItems, this.tasks.length);

    // Clear container
    this.container.innerHTML = '';

    // Create spacer for items above viewport
    if (startIndex > 0) {
      const topSpacer = document.createElement('div');
      topSpacer.style.height = (startIndex * this.itemHeight) + 'px';
      this.container.appendChild(topSpacer);
    }

    // Render visible items
    for (let i = startIndex; i < endIndex; i++) {
      const taskElement = createTaskElement(this.tasks[i]);
      this.container.appendChild(taskElement);
    }

    // Create spacer for items below viewport
    const remainingItems = this.tasks.length - endIndex;
    if (remainingItems > 0) {
      const bottomSpacer = document.createElement('div');
      bottomSpacer.style.height = (remainingItems * this.itemHeight) + 'px';
      this.container.appendChild(bottomSpacer);
    }
  }
}
```

### Caching Strategy

#### Intelligent Data Caching

```javascript
class DataCache {
  constructor(maxSize = 50) {
    this.cache = new Map();
    this.maxSize = maxSize;
    this.accessOrder = [];
  }

  get(key) {
    if (this.cache.has(key)) {
      // Update access order (LRU)
      this.updateAccessOrder(key);
      return this.cache.get(key);
    }
    return null;
  }

  set(key, value) {
    // Remove oldest if at capacity
    if (this.cache.size >= this.maxSize && !this.cache.has(key)) {
      const oldest = this.accessOrder.shift();
      this.cache.delete(oldest);
    }

    this.cache.set(key, {
      data: value,
      timestamp: Date.now(),
      accessCount: 1
    });

    this.updateAccessOrder(key);
  }

  updateAccessOrder(key) {
    const index = this.accessOrder.indexOf(key);
    if (index > -1) {
      this.accessOrder.splice(index, 1);
    }
    this.accessOrder.push(key);

    // Update access count
    const item = this.cache.get(key);
    if (item) {
      item.accessCount++;
    }
  }

  invalidate(pattern) {
    const keysToDelete = [];
    this.cache.forEach((value, key) => {
      if (key.match(pattern)) {
        keysToDelete.push(key);
      }
    });

    keysToDelete.forEach(key => {
      this.cache.delete(key);
      const index = this.accessOrder.indexOf(key);
      if (index > -1) {
        this.accessOrder.splice(index, 1);
      }
    });
  }
}
```

```javascript
// Global cache instance
const dataCache = new DataCache(100);

// Usage in recurring task calculation
function shouldTaskRecurNow(taskId, settings, currentTime) {
  const cacheKey = `recurring_${taskId}_${Math.floor(currentTime / 60000)}`; // 1-minute cache

  let result = dataCache.get(cacheKey);
  if (result !== null) {
    return result.data;
  }

  // Perform expensive calculation
  result = calculateRecurrenceState(settings, currentTime);

  // Cache result
  dataCache.set(cacheKey, result);

  return result;
}
```

## Security & Privacy

### Input Sanitization

#### XSS Prevention

```javascript
const SecurityUtils = {
  /**
   * Sanitizes HTML content to prevent XSS attacks
   * @param {string} input - Raw HTML input
   * @returns {string} Sanitized HTML
   */
  sanitizeHTML(input) {
    const div = document.createElement('div');
    div.textContent = input;
    return div.innerHTML;
  },
/**
 * Validates and sanitizes task input
 * @param {string} input - Task text input
 * @returns {object} Validation result
 */
validateTaskInput(input) {
  if (typeof input !== 'string') {
    return { valid: false, error: 'Input must be a string' };
  }

  // Remove potential script tags and harmful content
  const cleaned = this.sanitizeHTML(input.trim());

  // Length validation
  if (cleaned.length === 0) {
    return { valid: false, error: 'Task cannot be empty' };
  }

  if (cleaned.length > 50) {
    return { valid: false, error: 'Task too long (max 50 characters)' };
  }

  // Pattern validation (no script tags, no dangerous patterns)
  const dangerousPatterns = [
    /<script/i,
    /javascript:/i,
    /vbscript:/i,
    /onload/i,
    /onerror/i,
    /onclick/i
  ];

  for (const pattern of dangerousPatterns) {
    if (pattern.test(cleaned)) {
      return { valid: false, error: 'Invalid characters detected' };
    }
  }

  return { valid: true, sanitized: cleaned };
},

/**
 * Validates JSON data for import operations
 * @param {string} jsonString - JSON string to validate
```

```
   * @returns {object} Validation result
   */
  validateImportData(jsonString) {
    try {
      const data = JSON.parse(jsonString);

      // Check for dangerous properties
      const dangerousKeys = ['__proto__', 'constructor', 'prototype'];

      function checkObject(obj, path = '') {
        if (typeof obj !== 'object' || obj === null) return true;

        for (const key of Object.keys(obj)) {
          if (dangerousKeys.includes(key)) {
            throw new Error(`Dangerous property detected: ${path}.${key}`);
          }

          if (typeof obj[key] === 'object') {
            checkObject(obj[key], `${path}.${key}`);
          }
        }
      }

      checkObject(data);

      return { valid: true, data };
    } catch (error) {
      return { valid: false, error: error.message };
    }
  }
};
```

### Data Privacy

#### Local Storage Security

```javascript
const PrivacyManager = {
  /**
   * Encrypts sensitive data before storage
   * @param {object} data - Data to encrypt
   * @returns {string} Encrypted data
   */
```

```
encryptData(data) {
  // Simple encryption for client-side storage
  // Note: This is not cryptographically secure, just obfuscation
  const jsonString = JSON.stringify(data);
  const encoded = btoa(jsonString);

  // Add timestamp and checksum
  const timestamp = Date.now();
  const checksum = this.generateChecksum(encoded);

  return btoa(JSON.stringify({
    data: encoded,
    timestamp,
    checksum
  }));
},

/**
 * Decrypts data from storage
 * @param {string} encryptedData - Encrypted data string
 * @returns {object} Decrypted data
 */
decryptData(encryptedData) {
  try {
    const wrapper = JSON.parse(atob(encryptedData));

    // Verify checksum
    if (this.generateChecksum(wrapper.data) !== wrapper.checksum) {
      throw new Error('Data integrity check failed');
    }

    // Check age (optional expiration)
    const age = Date.now() - wrapper.timestamp;
    const maxAge = 365 * 24 * 60 * 60 * 1000; // 1 year

    if (age > maxAge) {
      console.warn('Stored data is very old, consider refreshing');
    }

    const jsonString = atob(wrapper.data);
    return JSON.parse(jsonString);

  } catch (error) {
    console.error('Failed to decrypt data:', error);
    return null;
  }
},

/**
 * Generates simple checksum for data integrity
 * @param {string} data - Data to checksum
 * @returns {string} Checksum hash
 */
generateChecksum(data) {
  let hash = 0;
  for (let i = 0; i < data.length; i++) {
    const char = data.charCodeAt(i);
    hash = ((hash << 5) - hash) + char;
    hash = hash & hash; // Convert to 32-bit integer
  }
  return hash.toString(36);
},

/**
 * Securely removes data from storage
 * @param {string} key - Storage key to remove
 */
secureRemove(key) {
  // Overwrite with random data before removal
  const randomData = Array.from({length: 1000}, () =>
    Math.random().toString(36).charAt(2)
  ).join('');

  localStorage.setItem(key, randomData);
  localStorage.removeItem(key);
},

/**
 * Gets user consent for data processing
 * @returns {Promise<boolean>} User consent status
 */
async getUserConsent() {
  return new Promise((resolve) => {
    const consentKey = 'miniCycle_dataConsent';
    const existingConsent = localStorage.getItem(consentKey);

    if (existingConsent) {
      resolve(existingConsent === 'granted');
```

```
        return;
      }

      // Show consent dialog
      this.showConsentDialog((granted) => {
        localStorage.setItem(consentKey, granted ? 'granted' : 'denied');
        resolve(granted);
      });
    });
  },

  showConsentDialog(callback) {
    const dialog = document.createElement('div');
    dialog.className = 'consent-dialog';
    dialog.innerHTML = `
      <div class="consent-content">
        <h3>Data Storage Consent</h3>
        <p>miniCycle stores your tasks locally on your device for functionality.
           No data is sent to external servers.</p>
        <div class="consent-actions">
          <button class="consent-accept">Accept</button>
          <button class="consent-decline">Decline</button>
        </div>
      </div>
    `;

    document.body.appendChild(dialog);

    dialog.querySelector('.consent-accept').onclick = () => {
      document.body.removeChild(dialog);
      callback(true);
    };

    dialog.querySelector('.consent-decline').onclick = () => {
      document.body.removeChild(dialog);
      callback(false);
    };
  }
};
```

## Browser Compatibility

### Progressive Enhancement Strategy

#### Feature Detection

```javascript
const FeatureDetector = {
  /**
   * Detects browser capabilities
   * @returns {object} Feature support object
   */
  detectCapabilities() {
    return {
      // Storage
      localStorage: typeof Storage !== 'undefined',
      sessionStorage: typeof sessionStorage !== 'undefined',

      // CSS Features
      customProperties: CSS.supports('color', 'var(--test)'),
      grid: CSS.supports('display', 'grid'),
      flexbox: CSS.supports('display', 'flex'),

      // JavaScript Features
      es6: typeof Symbol !== 'undefined',
      promises: typeof Promise !== 'undefined',
      fetch: typeof fetch !== 'undefined',

      // DOM Features
      querySelector: !!document.querySelector,
      addEventListener: !!document.addEventListener,

      // Device Features
      touch: 'ontouchstart' in window,
      geolocation: !!navigator.geolocation,
      vibration: !!navigator.vibrate,

      // PWA Features
      serviceWorker: 'serviceWorker' in navigator,
      webManifest: 'onbeforeinstallprompt' in window,

      // Performance Features
      requestAnimationFrame: !!window.requestAnimationFrame,
      webWorkers: typeof Worker !== 'undefined'
    };
  },

  /**
   * Determines if lite version should be used
   * @returns {boolean} Should use lite version
   */
  shouldUseLite() {
    const capabilities = this.detectCapabilities();
    const userAgent = navigator.userAgent.toLowerCase();

    // Force lite for very old browsers
    const isOldBrowser = (
      userAgent.includes('msie') ||
      userAgent.includes('trident') ||
      (userAgent.includes('safari') && !userAgent.includes('chrome') &&
       parseInt(userAgent.match(/version\/(\d+)/)?.[1] || '0') < 12)
    );

    if (isOldBrowser) return true;

    // Check essential features
    const essentialFeatures = [
      'localStorage',
      'querySelector',
      'addEventListener'
    ];

    const missingEssential = essentialFeatures.some(
      feature => !capabilities[feature]
    );

    if (missingEssential) return true;

    // Performance-based decision
    const isLowEnd = (
      navigator.hardwareConcurrency < 2 ||
      navigator.deviceMemory < 2 ||
      window.innerWidth < 400
    );

    return isLowEnd;
  }
};
```

#### Polyfills and Fallbacks

```javascript
const Polyfills = {
  /**
   * Loads necessary polyfills
   */
  init() {
    this.polyfillCustomProperties();
    this.polyfillPromises();
    this.polyfillFetch();
    this.polyfillArrayMethods();
  },

  /**
   * CSS Custom Properties polyfill for IE
   */
  polyfillCustomProperties() {
    if (!CSS.supports('color', 'var(--test)')) {
      // Simple variable replacement for critical properties
      const styleSheets = Array.from(document.styleSheets);

      styleSheets.forEach(sheet => {
        try {
          const rules = Array.from(sheet.cssRules);
          rules.forEach(rule => {
            if (rule.style) {
              // Replace common variables
              const variables = {
                '--primary-color': '#4c79ff',
                '--secondary-color': '#74c0fc',
                '--text-color': '#ffffff'
              };

              Object.entries(variables).forEach(([variable, value]) => {
                const regex = new RegExp(`var\\(${variable}\\)`, 'g');
                rule.style.cssText = rule.style.cssText.replace(regex, value);
              });
            }
          });
        } catch (e) {
          // Cross-origin stylesheets may throw errors
        }
    });
  }
},

/**
 * Promise polyfill for older browsers
 */
polyfillPromises() {
  if (typeof Promise === 'undefined') {
    // Simple Promise implementation
    window.Promise = function(executor) {
      const self = this;
      self.state = 'pending';
      self.value = undefined;
      self.handlers = [];

      function resolve(result) {
        if (self.state === 'pending') {
          self.state = 'fulfilled';
          self.value = result;
          self.handlers.forEach(handle);
          self.handlers = null;
        }
      }

      function reject(error) {
        if (self.state === 'pending') {
          self.state = 'rejected';
          self.value = error;
          self.handlers.forEach(handle);
          self.handlers = null;
        }
      }

      function handle(handler) {
        if (self.state === 'pending') {
          self.handlers.push(handler);
        } else {
          setTimeout(() => {
            const handlerCallback = self.state === 'fulfilled'
              ? handler.onFulfilled
              : handler.onRejected;

            if (handlerCallback) {
                try {
                  const result = handlerCallback(self.value);
                  handler.resolve(result);
                } catch (error) {
                  handler.reject(error);
                }
              } else {
                (self.state === 'fulfilled' ? handler.resolve : handler.reject)(self.value);
              }
            }, 0);
          }
        }

        self.then = function(onFulfilled, onRejected) {
          return new Promise((resolve, reject) => {
            handle({
              onFulfilled,
              onRejected,
              resolve,
              reject
            });
          });
        };

        try {
          executor(resolve, reject);
        } catch (error) {
          reject(error);
        }
      };
    }
  },
```