Product Requirements Document (PRD)

Student Task Board Application

Product Name: Student Task Board (STB)
Version: 1.0 - MVP
Date: February 17, 2026
Author: Product Team
Status: Approved for Development

---

1. Overview

Product Vision

Student Task Board is a lightweight, distraction-free task management application designed specifically for solo developers and students. It empowers users to organize academic projects, coding assignments, and personal development tasks efficiently using a simple, intuitive interface built with React and powered by localStorage for persistent, offline-first functionality.

Goals & Objectives

- Provide a simple, beginner-friendly task management tool for students learning to code
- Enable students to practice React development while building a useful application
- Create a portfolio-worthy project that demonstrates modern web development skills
- Minimize complexity through localStorage-based persistence (no backend required)
- Deliver a responsive, mobile-friendly interface for on-the-go task management

Target Market

- **Primary**: Computer Science/Engineering students learning React and web development
- **Secondary**: Solo developers, freelancers, indie hackers managing personal projects
- **Tertiary**: Anyone preferring simple, privacy-focused task management without cloud sync

Key Benefits

- **Offline-First**: All data stored locally; works without internet connection
- **Privacy-Focused**: No data collection, no tracking, no cloud dependency

- **Educational Value**: Clean codebase serves as learning reference for React patterns
- **Zero Setup**: No authentication, no backend, no deployment complexity
- **Customizable**: Extendable for personal learning and skill enhancement

---

2. User Personas

Persona 1: Alex Chen - Computer Science Student

- **Role:** Full-time Computer Science student, learning web development
- **Age/Background:** 20 years old, 2nd year CS major, beginner to intermediate React developer
- **Goals:**
  - Build a practical React project for portfolio
  - Stay organized with course assignments and personal coding projects
  - Learn component architecture and state management patterns
  - Complete projects on time and track progress
- **Pain Points:**
  - Generic task apps with too many features create decision fatigue
  - Complexity distracts from learning React fundamentals
  - Privacy concerns with apps collecting personal data
  - Wants proof-of-concept projects for resume
- **Needs:**
  - Simple, clean UI to focus on React implementation
  - Clear component structure to learn from
  - Ability to customize and extend the app
  - Fast, responsive experience on laptop and phone during study sessions

Persona 1: Alex Chen - Computer Science Student (revised)

- **Role:** Full-time Computer Science student, learning web development
- **Age/Background:** 20 years old, 2nd year CS major; building portfolio projects
- **Devices & Environment:** Primary: laptop (Windows 11, Chrome) used in dorm/library; Secondary: Android phone (Chrome mobile). Frequently studies with intermittent Wi-Fi.
- **Technical Skill:** Comfortable with React components and hooks; new to TypeScript and advanced state management patterns.
- **Usage Pattern:** Uses productivity tools 3–5× per week; typical session 10–20 minutes while working on assignments or reviewing project tasks.
- **Goals:**
  - Ship portfolio-ready React projects
  - Stay organized and submit assignments before deadlines

  - Learn component architecture, hooks, and basic TypeScript while building a real app
- **Pain Points:**
  - Decision fatigue from feature-bloated apps
  - Overly complex UIs distract from learning goals
  - Concerned about privacy and data collection
- **Constraints:**
  - Limited weekly time for side projects (~8–12 hours)
  - Uses devices with modest storage and occasional poor connectivity
- **Measurable Success Criteria:**
  - Completes 90% of weekly course-related tasks before due dates
  - Builds at least one portfolio-ready feature implementation per month
- **Quote:** "I need a minimal tool that helps me ship assignments on time and learn React without distractions."

Persona 2: Jamie Rodriguez - Solo Developer/Freelancer

- **Role:** Independent web developer freelancing while studying
- **Age/Background:** 23 years old, bootcamp graduate, intermediate React developer
- **Goals:**
  - Manage multiple project deliverables efficiently
  - Track personal development tasks alongside client work
  - Maintain code quality through task organization
  - Build tools that showcase technical skills
- **Pain Points:**
  - Heavy project management tools are overkill for one person
  - Subscription costs eat into already-tight freelance budgets
  - Privacy concerns about sharing personal work data
  - Wants minimalist tools that don't require vendor lock-in
- **Needs:**
  - Lightweight tool focused on solo developer workflow
  - Filter and categorize tasks by project or client
  - Quick task entry and status updates
  - Reliability without complex backup/recovery processes

Persona 3: Morgan Taylor - Bootcamp Graduate

- **Role:** Recent bootcamp graduate preparing for first tech job
- **Age/Background:** 28 years old, career changer, looking to impress in interviews
- **Goals:**
  - Build impressive portfolio projects
  - Demonstrate understanding of React and modern web patterns
  - Show ability to build complete, functional applications
  - Learn about accessibility and responsive design

- **Pain Points:**
  - Existing task apps don't showcase development skills
  - Wants projects that demonstrate best practices
  - Limited time due to job search and other projects
  - Needs something polished enough for GitHub portfolio
- **Needs:**
  - Well-structured, professionally organized codebase
  - Clear separation of concerns and component architecture
  - Optional styling for presentation
  - Documentation for portfolio explanation

---

3. Use Cases

Use Case 1: Create and Organize Academic Assignment

- **Actor:** Computer Science Student (Alex Chen)
- **Preconditions:**
  - User has opened Student Task Board application
  - Application has loaded successfully
  - localStorage is available and not full
- **Main Flow:**
  1. User clicks "Add New Task" button
  2. Task creation form appears with input fields
  3. User enters task title: "Build React Todo App - CS401"
  4. User selects category: "Academic Assignments"
  5. User sets priority: "High"
  6. User sets due date: next Friday
  7. User adds description: "Build functional component with hooks, add/delete/edit todos"
  8. User clicks "Save Task"
  9. Task appears in task list with visual priority indicator
  10. Task data is persisted to localStorage automatically
- **Alternate Flow:**
  - User wants to add multiple related subtasks (scope dependent)
  - User wants to quickly add task with minimal details
- **Postconditions:**
  - Task is visible in task list
  - Task is stored in localStorage
  - Task can be edited or deleted
  - User receives visual confirmation of save

Use Case 2: Track Project Progress Through Task Completion

- **Actor:** Solo Developer (Jamie Rodriguez)
- **Preconditions:**

  - User has existing tasks in system
  - User has multiple projects tracked
  - Application has loaded successfully
- **Main Flow:**
  1. User views task dashboard/list view
  2. User filters tasks by project: "Client Website Redesign"
  3. Filtered tasks appear showing 8 total, 3 completed
  4. User reviews incomplete tasks
  5. User clicks "Start Work" on "Design mobile navigation menu"
  6. Task status changes to "In Progress" (optional visual change)
  7. User completes work and clicks "Mark Complete"
  8. Task moves to completed section
  9. Dashboard updates to show 4/8 tasks completed (50%)
  10. Progress is saved to localStorage
- **Alternate Flow:**
  - User wants to undo completion
  - User wants to reassign task to different project
  - User wants to add time tracking notes
- **Postconditions:**
  - Task status updated to completed
  - Completion percentage recalculated
  - Historical record maintained in localStorage
  - User can view progress analytics

Use Case 3: Search and Filter Tasks by Priority and Due Date

- **Actor:** Bootcamp Graduate (Morgan Taylor)
- **Preconditions:**
  - User has 15+ tasks in various states
  - User is preparing for job interview and needs to focus
  - Application is fully loaded
- **Main Flow:**
  1. User views task list with all tasks visible
  2. User clicks filter icon
  3. Filter panel expands showing options: Priority, Due Date, Status, Category
  4. User selects "High Priority" checkbox
  5. User selects "Due This Week" from date dropdown
  6. Task list instantly filters to show 3 high-priority tasks due this week
  7. User reviews tasks and identifies urgent items
  8. User clicks on first task to see details
  9. User edits task to add interview preparation notes
  10. Filter remains applied as user navigates between tasks
- **Alternate Flow:**
  - User wants to save filter as favorite
  - User wants to clear filters and see all tasks

  - User wants to export filtered task list
- **Postconditions:**
  - Filtered task view is displayed
  - User can edit filtered tasks
  - Filter state may be saved to localStorage for next session

---

4. Functional Requirements

Feature 1: Task Management Core (Priorities)

- **FR-1.1:** Users can create new tasks with title, description, priority level, due date, and category **(Priority: High)**
- **FR-1.2:** Users can view all tasks in a list format with key information visible (title, priority, due date, status) **(Priority: High)**
- **FR-1.3:** Users can edit existing tasks to update any field (title, priority, due date, category, description) **(Priority: High)**
- **FR-1.4:** Users can delete tasks with optional confirmation dialog **(Priority: Medium)**
- **FR-1.5:** Users can mark tasks as complete/incomplete with single click **(Priority: High)**
- **FR-1.6:** All task operations (create, read, update, delete) are persisted to localStorage **(Priority: High)**

Feature 2: Task Organization & Filtering (Priorities)

- **FR-2.1:** Users can organize tasks by categories (Academic, Work, Personal, etc.) with customizable category creation **(Priority: Medium)**
- **FR-2.2:** Users can filter tasks by priority level (High, Medium, Low) **(Priority: High)**
- **FR-2.3:** Users can filter tasks by status (All, Active, Completed) **(Priority: Medium)**
- **FR-2.4:** Users can filter tasks by due date ranges (Today, This Week, Overdue, etc.) **(Priority: Medium)**
- **FR-2.5:** Users can search tasks by title or description keywords **(Priority: Medium)**
- **FR-2.6:** Multiple filters can be combined and applied simultaneously **(Priority: Medium)**

Feature 3: Task Prioritization & Progress (Priorities)

- **FR-3.1:** Users can assign priority levels to tasks (High, Medium, Low) with color-coded visual indicators **(Priority: High)**
- **FR-3.2:** Users can view overall progress statistics (total tasks, completed tasks, completion percentage) **(Priority: Medium)**

- **FR-3.3:** Users can view progress breakdown by category showing tasks completed per category **(Priority: Medium)**
- **FR-3.4:** Users can view upcoming deadlines summarized in dashboard view **(Priority: Medium)**
- **FR-3.5:** Task list can be sorted by priority, due date, or creation date (ascending/descending) **(Priority: Medium)**

Feature 4: User Interface & Experience (Priorities)

- **FR-4.1:** Application loads with responsive layout suitable for desktop (1920x1080), tablet (768x1024), and mobile (375x812) viewports **(Priority: High)**
- **FR-4.2:** Navigation is intuitive with clear sections: Task List, Dashboard/Analytics, Settings **(Priority: High)**
- **FR-4.3:** Task creation and editing use consistent form components with clear validation messages **(Priority: High)**
- **FR-4.4:** Empty states display helpful prompts when no tasks exist **(Priority: Low)**
- **FR-4.5:** Visual feedback provided for all user actions (create, update, delete, complete) **(Priority: High)**

Feature 5: Data Persistence & Settings (Priorities)

- **FR-5.1:** All task data is automatically saved to browser localStorage on any change **(Priority: High)**
- **FR-5.2:** Users can set application preferences: theme (light/dark), task sorting, category defaults **(Priority: Medium)**
- **FR-5.3:** Users can clear all data or export task list as JSON backup **(Priority: Medium)**
- **FR-5.4:** Preferences are saved to localStorage and apply to future sessions **(Priority: Medium)**
- **FR-5.5:** Application implements error handling for localStorage quota exceeded scenarios **(Priority: Medium)**

---

5. Non-Functional Requirements

Performance

- **NFR-P-1:** Task list view loads within 500ms on broadband connection
- **NFR-P-2:** Filtering and search results update in real-time with <100ms response time
- **NFR-P-3:** Application remains responsive with up to 500 tasks in localStorage
- **NFR-P-4:** localStorage operations complete in <50ms to avoid blocking UI

- **NFR-P-5:** Zero external API calls required; entirely client-side processing

Security & Privacy

- **NFR-S-1:** No user authentication required; all data stored locally in user's browser
- **NFR-S-2:** No personal data sent to external servers or analytics services
- **NFR-S-3:** No tracking, cookies, or telemetry collected
- **NFR-S-4:** Data never leaves user's device; 100% privacy-first architecture
- **NFR-S-5:** HTTPS recommended for deployment to protect against MITM attacks

Usability

- **NFR-U-1:** Application achieves WCAG 2.1 AA accessibility standards
- **NFR-U-2:** All text is readable with minimum 16px base font size
- **NFR-U-3:** Color contrast ratio meets WCAG AA standards (4.5:1 for text)
- **NFR-U-4:** Keyboard navigation supported throughout application
- **NFR-U-5:** Touch targets minimum 44x44px on mobile devices

Scalability

- **NFR-SC-1:** Architecture supports potential future expansion to 1000+ tasks
- **NFR-SC-2:** Code structure allows for easy addition of new features without refactoring core
- **NFR-SC-3:** Component design supports reusability and maintains performance

Reliability & Code Quality

- **NFR-R-1:** Application handles edge cases gracefully (corrupted localStorage, quota exceeded)
- **NFR-R-2:** Code maintainability: clear component structure, documented patterns, < 100 line components
- **NFR-R-3:** Codebase serves as learning reference with well-structured React patterns
- **NFR-R-4:** CSS-in-JS or modular CSS for component-scoped styling
- **NFR-R-5:** Code follows React best practices: hooks, functional components, proper dependencies

Compatibility

- **NFR-C-1:** Supports React 18.x

- **NFR-C-2:** Compatible with modern browsers (Chrome, Firefox, Safari, Edge - latest 2 versions)
- **NFR-C-3:** localStorage API available in all target browsers
- **NFR-C-4:** Responsive design works across viewport sizes 320px - 4K

---

6. Success Metrics

Key Performance Indicators (KPIs)

SMART Success Metrics (revised)

Key Performance Indicators (KPIs)

- **KPI-1 — Task Completion Rate:** Within 30 days of first use, average user completes 70% of tasks they created during that 30-day period (measured via opt-in telemetry or periodic user survey).
- **KPI-2 — Daily Active Users (DAU):** 100+ unique student DAUs within 4 months of public release (measured by opt-in usage telemetry or GitHub/fork/download activity if telemetry is disabled).
- **KPI-3 — Filtering Feature Adoption:** 80% of active users perform at least two filtering actions within their first 14 days.

Learning & Educational Metrics

- **LM-1 — GitHub Stars:** 50+ stars on the repository within 3 months of public release.
- **LM-2 — Code Quality:** Zero critical TypeScript/CI build errors on `main` branch for the v1.0 release (enforced by CI checks).
- **LM-3 — Documentation Completeness:** All exported components include docs or Storybook stories and a project setup guide completed before v1.0 release.
- **LM-4 — Community Feedback:** Average rating 4/5 from at least 25 survey responses on code clarity and learning value within 2 months.

Technical Metrics

- **TM-1 — Lighthouse Performance:** Production build achieves Lighthouse Performance score 90.
- **TM-2 — Bundle Size:** Production gzipped bundle <100KB (measured after build/minification/gzip).
- **TM-3 — localStorage Efficiency:** Typical single task <500 bytes; 100 tasks <50KB (validated with test data during QA).
- **TM-4 — Test Coverage:** 80% unit test coverage for core business logic (measured by CI coverage reports).

User Engagement Metrics

- **UM-1 — Task Creation Rate:** Average 10 tasks created per user within their first 30 days.
- **UM-2 — Feature Usage:** Within 30 days, 70% of active users use filtering at least once per week and 60% visit dashboard analytics at least twice.
- **UM-3 — Retention:** 50% of new users return at least once within 7 days of first use (measured by client-side active session markers or opt-in telemetry).
- **UM-4 — Portfolio Adoption:** 30% of surveyed early adopters report adding the project to their GitHub portfolio within 3 months (measured via follow-up survey).

---

7. Scope

In Scope (MVP - Version 1.0)

- Full CRUD operations for tasks
- Task organization by categories
- Priority levels with visual indicators
- Due date tracking and filtering
- Basic search and filter functionality
- Task completion status tracking
- localStorage persistence
- Responsive design (mobile-first approach)
- Dashboard with basic statistics
- Dark/light theme toggle
- Clean, educational React codebase
- TypeScript for type safety
- Professional documentation and README

Out of Scope (Future Phases)

- Backend API or cloud synchronization (versioning constraint: localStorage only for MVP)
- User authentication or multi-user sharing
- Recurring/recurring tasks
- Task time tracking or pomodoro timer
- Notifications or push reminders
- Mobile app (native iOS/Android)
- Advanced analytics or reporting
- Integration with calendar APIs (Google Calendar, Outlook)
- Collaboration features (comments, assignments, mentions)
- Database migration if database support added later

- Offline synchronization with backend

Phase-Based Rollout

- **Phase 1 (MVP v1.0):** Core task management, basic filtering, localStorage, responsive design
  - Target: March 2026
  - Deliverables: Functional app, GitHub repo, documentation
- **Phase 2 (v1.1):** Advanced features
  - Target: April 2026
  - Deliverables: Recurring tasks, time tracking, advanced analytics
- **Phase 3 (v2.0):** Potential cloud features (optional)
  - Target: June 2026
  - Deliverables: Optional backend API, cloud sync, multi-device support

---

8. Timeline

Milestone	Target Date	Status	Notes
Project Setup & Architecture	Feb 20, 2026	Planned	React project scaffold, folder structure
Core Features Development	Mar 10, 2026	Planned	Task CRUD, localStorage, filters
UI/UX Polish & Responsive Design	Mar 20, 2026	Planned	Styling, mobile optimization, accessibility
Testing & Documentation	Mar 25, 2026	Planned	Unit tests, README, deployment guide
v1.0 Release	Mar 31, 2026	Planned	Launch on GitHub, portfolio-ready
Gather Feedback & Plan v1.1	Apr 7, 2026	Planned	Community feedback, prioritize backlog

---

9. Assumptions & Dependencies

Assumptions

- Students have access to modern browsers with React 18 support
- Users are comfortable with localStorage limitations (5-10MB per domain)
- Target audience values simplicity over feature complexity
- Students want to learn from well-structured React code
- Privacy-first positioning resonates with target audience
- GitHub will be used as primary distribution channel

Dependencies

- **React 18.x library** - Core framework dependency
- **Vite build tool** - Build and dev server
- **TypeScript** - Type checking (optional but recommended)
- **Modern Browser APIs** - localStorage, responsive design, CSS Grid/Flexbox
- **GitHub** - For code hosting and distribution
- **Node.js & npm** - For development environment

No External Dependencies

- No backend server required
- No authentication service required
- No third-party task management API
- No payment processing
- No analytics or tracking service

---

10. Risks & Mitigation

Risk	Probability	Impact	Mitigation Strategy
localStorage quota exceeded	Medium	Medium	Implement warning when approaching quota; provide data export/cleanup guidance
Browser compatibility issues	Low	High	Target latest 2 browser versions; test on Chrome, Firefox, Safari, Edge

Risk	Probability	Impact	Mitigation Strategy
Performance degradation with 500+ tasks	Medium	Medium	Implement pagination or virtual scrolling; optimize render performance
React version conflicts	Low	Medium	Pin React 18.x in package.json; use official React 18 typing
Users lose data on browser clear	All	High	Add automatic backup export feature; educate users on browser storage risks
Scope creep (feature requests)	High	Medium	Strictly maintain MVP scope; defer features to phase 2/3; document rationale
Learning curve for beginner developers	Medium	Medium	Add inline code comments; create setup guide; provide component architecture docs

---

Peer Review Notes & Common Pitfalls

- **Generic Persona Pitfall Avoided:** Odak noktasını genel kullanıcı tanımlarından kaçınarak Alex Chen gibi spesifik persona'lara yönelttik; bu sayede tasarım kararları gerçekçi cihaz/bağlantı/kısıt koşullarına göre verildi.
- **Common Pitfalls:** Ensure metrics are measurable and timebound; avoid vague success criteria; keep epics independent and map them to PRD metrics.

11. Approval

Role	Name	Signature	Approved
Product Owner	[Student/Team Lead]	[ ]	[ ]
Technical Lead	[Senior Developer]	[ ]	[ ]
Project Manager	[Instructor/Supervisor]	[ ]	[ ]

Role	Name	Signature	Approved

---

12. Appendices

A. Glossary

- **localStorage**: Web storage API for client-side persistent data storage
- **MVP (Minimum Viable Product)**: Core features needed for MVP launch
- **React 18**: Latest major version of React with concurrent features
- **WCAG 2.1**: Web Content Accessibility Guidelines for inclusive design
- **Vite**: Modern frontend build tool with fast development server

B. References

- React 18 Documentation: https://react.dev
- localStorage MDN Guide: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
- WCAG 2.1 Accessibility: https://www.w3.org/WAI/WCAG21/quickref/
- Vite Documentation: https://vitejs.dev

C. Related Documents

- Epic: Task Management System (specs/epics/)
- User Stories: Individual feature implementations (specs/stories/)
- Architecture Diagram: [To be created in Phase 1]
- Component Structure: [To be documented in code]

PRD Version 1.0 - Approved for Development
Last Updated: February 17, 2026