Workout Planner Web Application

Technical Design, Implementation, and Deployment Report

Pranav Kumar Kaliaperumal
M.S. Computer Science, University of Colorado Denver

January 12, 2026

Project Overview

The Workout Planner application is a single-page React web application designed to allow users to define workouts, assign them to days of the week, and track lightweight Key Performance Indicators (KPIs) related to weekly training structure. The application is fully client-side, persists state using browser `localStorage`, and is deployed publicly via GitHub Pages.

The primary goals of the project were:

- Build a production-quality React application with clean state management
- Demonstrate component-driven UI design
- Implement KPI-style analytics inspired by Agile metrics
- Practice Agile/Scrum development and Git-based workflows
- Deploy reliably to GitHub Pages using modern tooling (Vite)

Technology Stack

- **Frontend Framework**: React 18
- **Build Tool**: Vite
- **Language**: JavaScript (ES6+)
- **Styling**: Custom CSS (Black / Gold / Silver theme)
- **State Persistence**: Browser localStorage
- **Version Control**: Git + GitHub
- **Deployment**: GitHub Pages via GitHub Actions

Repository Structure

```
Workout-Planner-App/
 .github/workflows/deploy.yml
 docs/
    report.tex
    kpi_example.png
 src/
    utils/
       kpi.js
    App.jsx
    main.jsx
    index.css
 index.html
 package.json
 package-lock.json
 vite.config.js
 README.md
```

File Responsibilities

index.html Defines the root HTML shell and the mounting point for the React application.

src/main.jsx Initializes React and mounts the `App` component to the DOM. Also imports global styles.

src/App.jsx The core application component. Handles:

- Workout creation and deletion
- Assignment of workouts to days
- KPI computation and display
- localStorage persistence and recovery

src/utils/kpi.js Encapsulates all KPI logic. This separation ensures that analytical logic is decoupled from UI rendering.

index.css Defines the global theme, layout, typography, and component styling using a black, gold, and silver aesthetic.

vite.config.js Configures the Vite build system, including the `base` path required for GitHub Pages deployment.

Application Architecture

The application follows a unidirectional data flow:

- React state is held in the top-level `App` component
- User actions update state via event handlers
- Derived data (KPIs) are computed using `useMemo`
- UI re-renders automatically when state changes

Workout assignments are stored as references (IDs) rather than duplicated objects, which prevents data inconsistency and simplifies deletion logic.

KPI Design and Rationale

The KPI system was inspired by Agile metrics such as throughput and work distribution. While the domain is fitness rather than software delivery, the same principles apply.

Implemented KPIs include:

- Total workouts defined (analogous to backlog size)
- Total assignments for the week (work committed)
- Average assignments per day (load distribution)

All KPI calculations are defensive and tolerate missing or malformed data to prevent runtime crashes.

Agile and Scrum Development Process

The project was developed iteratively using Agile principles:

Sprint Planning

Features were implemented incrementally:

1. Base UI and state model
2. Workout creation
3. Day assignment
4. KPI analytics
5. Styling and theming
6. Deployment and hardening

Branching Strategy

- `main`: Stable, deployable branch
- `testing`: Experimental UI and feature changes

This mirrors real-world Scrum workflows where features are validated before merging.

Deployment Pipeline

Deployment is handled via GitHub Actions:

- Code pushed to `main`
- Vite builds static assets
- Assets deployed to GitHub Pages

The `base` option in `vite.config.js` ensures assets resolve correctly when served from a subdirectory.

Major Challenges and Debugging

Blank Page on GitHub Pages

Initial deployments resulted in a blank page despite successful builds. Root causes included:

- Incorrect script paths in `index.html`
- Cached build artifacts

Resolution involved switching to relative module paths and forcing rebuilds.

localStorage Backward Compatibility Bug

A runtime crash occurred due to older stored data lacking expected fields. This manifested as:

```
TypeError: Cannot convert undefined or null to object
```

The fix involved:

- Normalizing loaded data
- Making KPI computations defensive
- Clearing corrupted storage during testing

This issue highlights the importance of data migration and backward compatibility even in client-only applications.

Lessons Learned

Key takeaways from the project include:

- Deployment success does not guarantee runtime correctness
- Browser caching can mask code changes
- Defensive programming prevents catastrophic UI failures
- Proper Git workflows save time during debugging
- Real-world debugging skills are as important as writing new code

Conclusion

The Workout Planner application demonstrates full-stack thinking within a frontend-focused project. It integrates UI design, state management, analytics, deployment, and Agile workflows into a cohesive, production-ready system. The challenges encountered and resolved during development closely mirror those faced in professional software engineering environments.