# FitnessWise - Back Workout Tracker Requirements Document

## Project Overview

**Project Name:** FitnessWise
**Type:** Progressive Web App (PWA)
**Primary User:** Eddie - Advanced fitness enthusiast with UX design expertise
**Core Problem:** Eliminating workout progress loss due to accidental page refreshes
**Target Platform:** Mobile-first web application with PWA capabilities

## Critical Context & User Background

### The Real User Profile

- **Usage Pattern:** 8:30 AM workouts, every other day schedule
- **Environment:** Home gym setting with touch devices, potentially sweaty hands
- **Experience Level:** Advanced lifter who understands progression and periodization
- **Technical Sophistication:** Can articulate complex UX requirements and product management concepts
- **Primary Pain Point:** Previously lost workout progress due to accidental browser refreshes during exercise sessions

### Authentic Usage Requirements

- Must work reliably during physical exertion when mental focus is on exercise execution
- Touch-friendly interface for gym environment conditions
- Reliable state persistence across all types of interruptions
- Professional-grade user experience that rivals commercial fitness apps

## Core Architecture Requirements

### Three-Banner Progressive Disclosure System

*Inspired by Amazon's mobile interface patterns*

#### 1. App Banner (Fixed Header)

- **Always visible:** Global navigation, authentication status, streak counter
- **Position:** Fixed at top, never scrolls away
- **Content:**

    - App title/logo
    - Day streak counter for motivation
    - Authentication status (login/logout)
    - Navigation trigger (hamburger menu)

**2. Day Banner (Progressive Disclosure)**

- **Full State:** Timer controls, workout context, exercise guidance
- **Minimized State:** Essential timer only (Amazon-style scroll minimization)
- **Critical Requirement:** Timer must remain visible during exercise execution
- **Content:**
    - Workout timer with Start/Pause/Complete controls
    - Current workout day indicator (A, B, C)
    - Workout progress overview

**3. Workout Banner (Exercise Level)**

- **Function:** Individual exercise tracking and progression
- **Content:**
    - Current exercise details
    - Predefined sets with guided progression
    - Rep tracking with touch-optimized controls
    - Video integration for form reference
    - Navigation between exercises (click/swipe)

### State Persistence Architecture (CRITICAL)

- **30-second auto-save** during active workouts
- **Session recovery** across browser refreshes and interruptions
- **Persistent timer state** (survives phone calls, app switching)
- **Visual feedback** showing save status at all times
- **Dual persistence:** localStorage for immediate + Firebase for cloud backup

### Mobile-First Design Requirements

- **Minimum 44px touch targets** for all interactive elements
- **Horizontal timer button layout** (proven to work effectively)

- **Sweat/glove-friendly interaction patterns**
- **Performance optimized** for mobile devices during physical activity
- **Progressive Web App** capabilities for home screen installation

## Technical Architecture

### Frontend Technology Stack

- **Core:** Modern ES6 modules (avoid scope pollution issues)
- **Framework:** Vanilla JavaScript with component-based architecture
- **Styling:** CSS Grid and Flexbox for responsive design
- **Event Handling:** Event delegation patterns (NOT onclick handlers)
- **PWA:** Service worker, manifest.json, offline capabilities

### Backend & Authentication

- **Firebase Authentication:** User accounts with email/password
- **Firestore Database:** Workout data, user progress, settings
- **Real-time Sync:** Multi-device data synchronization
- **Anonymous Support:** Option for users who prefer not to create accounts

### Data Architecture

#### Workout Structure

```javascript
// Day A: Vertical Pull Focus (V-taper development)
// Day B: Horizontal Pull & Barbell (thickness)
// Day C: Power & Mixed Patterns (explosive strength)

const workoutData = {
  dayA: {
    title: "Vertical Pull Focus",
    focus: "V-taper development",
    exercises: [
      {
        name: "Pull-ups",
        type: "bodyweight",
        predefinedSets: [
          { targetReps: 8, completedReps: 0, completed: false },
          { targetReps: 6, completedReps: 0, completed: false },
          { targetReps: 4, completedReps: 0, completed: false }
        ],
        videoUrl: "youtube_url_here",
        category: "vertical_pull"
      }
      // Additional exercises...
    ]
  }
  // Days B and C...
};
```

### User Progress Data

javascript

```javascript
const userProgress = {
  userId: "firebase_user_id",
  currentStreak: 0,
  totalWorkouts: 0,
  preferences: {
    trainingFrequency: [1,0,1,0,1,0,0], // Weekly grid: work/rest pattern
    weightTracking: true,
    notifications: true
  },
  workoutHistory: [
    {
      date: "2025-01-15",
      workoutDay: "A",
      duration: 2400, // seconds
      exercises: [/* completed exercise data */],
      notes: "Great session, felt strong"
    }
  ]
};
```

## Core Features (Phase 1)

### Essential Functionality

1. **Three-Banner Navigation System**
   - Scroll-based progressive disclosure
   - Consistent context preservation
   - Mobile-optimized touch interactions
2. **Guided Workout Flow**
   - Predefined sets for each exercise
   - Automatic progression through workout
   - Clear visual feedback on completion status
3. **Timer Management**
   - Session timer with persistent state
   - Start/Pause/Complete controls
   - Horizontal button layout for mobile
4. **State Persistence**

    - 30-second auto-save during workouts
    - Recovery from accidental refreshes
    - Visual save status indicators
5. **User Authentication**
    - Firebase email/password authentication
    - User profile management
    - Data associated with user accounts

## Exercise Data (Day A Example)

javascript

```javascript
const dayAExercises = [
  {
    name: "Pull-ups",
    description: "Vertical pulling movement for lat development",
    sets: 3,
    targetReps: "8, 6, 4",
    videoUrl: "https://youtube.com/watch?v=...",
    equipment: "pull-up bar",
    muscleGroups: ["lats", "rhomboids", "middle traps"]
  },
  {
    name: "Lat Pulldowns",
    description: "Machine-based vertical pull",
    sets: 3,
    targetReps: "10, 8, 6",
    videoUrl: "https://youtube.com/watch?v=...",
    equipment: "cable machine",
    muscleGroups: ["lats", "rear delts"]
  },
  {
    name: "Cable Rows",
    description: "Horizontal pulling for thickness",
    sets: 3,
    targetReps: "12, 10, 8",
    videoUrl: "https://youtube.com/watch?v=...",
    equipment: "cable machine",
    muscleGroups: ["rhomboids", "middle traps", "rear delts"]
  }
];
```

## Advanced Features (Phase 2 Preparation)

### Weight Tracking System

- **Smart categorization:** ML-powered detection of weighted vs. bodyweight exercises
- **Quick entry interface:** Radio buttons for common plate weights (2.5, 5, 10, 25, 45 lbs)
- **Mid-workout adjustments:** Track weight changes during sessions
- **Progress analytics:** Weight progression tracking over time

### Training Frequency Configuration

- **Weekly grid interface:** Visual selection of workout/rest days
- **Flexible scheduling:** Support for advanced patterns (2 days on, 1 day off)
- **Goal-based recommendations:** Frequency suggestions based on experience level
- **Recovery tracking:** Integration with workout intensity and volume

### Journal Generation (AI-Powered Coaching)

- **Data synthesis:** Combine workout data + user goals + session feedback
- **Coach-style reflections:** AI-generated progress analysis and motivation
- **Progression suggestions:** Intelligent recommendations for sets/reps/weight increases
- **Pattern recognition:** Identify trends and plateaus in user performance

## User Experience Requirements

### Mobile Interface Standards

- **Touch targets:** Minimum 44px for all interactive elements
- **Responsive design:** Seamless experience across phone/tablet/desktop
- **Progressive disclosure:** Information hierarchy that minimizes cognitive load
- **Visual feedback:** Clear indication of all system states and user actions
- **Error prevention:** Confirmation dialogs for destructive actions

### Performance Requirements

- **Load time:** Initial page load under 2 seconds on 3G
- **Responsiveness:** UI updates within 100ms of user interaction
- **Offline capability:** Core workout tracking works without internet
- **Battery efficiency:** Minimal impact on device battery during workouts

## Technical Implementation Guidelines

### Event Handling Architecture

javascript

```javascript
// Use event delegation, NOT onclick handlers
document.addEventListener('click', (event) => {
  const action = event.target.dataset.action;
  const handler = actionHandlers[action];
  if (handler) {
    handler(event);
  }
});

const actionHandlers = {
  'start-timer': startTimer,
  'complete-set': completeSet,
  'navigate-exercise': navigateExercise
};
```

### Component Organization

javascript

```javascript
// Clean module separation
import { FirebaseService } from './services/firebase.js';
import { TimerComponent } from './components/timer.js';
import { WorkoutTracker } from './components/workout.js';
import { NavigationManager } from './components/navigation.js';

class FitnessApp {
  constructor() {
    this.firebase = new FirebaseService();
    this.timer = new TimerComponent();
    this.workout = new WorkoutTracker();
    this.navigation = new NavigationManager();
  }
}
```

### State Management Pattern

javascript

```javascript
class AppState {
  constructor() {
    this.currentWorkout = null;
    this.timerState = { running: false, elapsed: 0 };
    this.userSession = { authenticated: false, user: null };
    this.saveInterval = 30000; // 30 seconds
  }

  persistState() {
    localStorage.setItem('fitnesswise-state', JSON.stringify(this.getState()));
    this.syncToFirebase();
  }

  restoreState() {
    const saved = localStorage.getItem('fitnesswise-state');
    if (saved) {
      this.setState(JSON.parse(saved));
    }
  }
}
```

## Critical Implementation Notes

### Scope Pollution Prevention

- **NEVER use onclick handlers** - they create scope pollution with ES6 modules
- **Use event delegation** for all user interactions
- **Organize code in ES6 modules** with clean import/export patterns
- **Avoid global function bridges** - they create maintenance nightmares

### Firebase Integration Best Practices

- **Separate configuration** from application logic
- **Handle all error states** with user-friendly messages
- **Implement offline-first** approach with local storage fallback
- **Use Firestore security rules** to protect user data

### Mobile Testing Requirements

- **Test on actual devices** - browser responsive mode is insufficient

- **Verify touch interactions** work with various finger sizes
- **Test during simulated exercise** (timer accuracy, state persistence)
- **Validate across iOS Safari and Android Chrome**

## Success Metrics

### Technical Success Indicators

- **Zero data loss** due to accidental refreshes
- **Sub-100ms response** to user interactions
- **Successful state restoration** after browser crashes
- **Reliable authentication** flow without errors

### User Experience Success Indicators

- **Eddie completes workouts** without technical friction
- **8:30 AM routine flows smoothly** without interruptions
- **Progress tracking provides** motivational value
- **App feels like professional** fitness platform

## Development Workflow

### Phase 1 Implementation Order

1. **Basic HTML structure** with three-banner layout
2. **Event delegation system** for user interactions
3. **Timer component** with persistent state
4. **Workout tracking** with predefined sets
5. **Firebase authentication** integration
6. **State persistence** with auto-save
7. **Mobile optimization** and PWA features

### Testing Strategy

- **Unit tests** for core functionality
- **Integration tests** for Firebase operations
- **Manual testing** on mobile devices during actual workout simulation
- **User acceptance testing** with Eddie's real workout routine

## Known Issues to Avoid

### Architectural Debt Prevention

- **Scope pollution** between ES6 modules and HTML event handlers
- **Mixed interaction patterns** (onclick vs event delegation)
- **Global function bridges** that create maintenance complexity
- **Inconsistent state management** across components

### Common Mobile Pitfalls

- **Vertical button stacking** that gets cut off on small screens
- **Touch targets too small** for reliable interaction during exercise
- **Timer state loss** during phone calls or app switching
- **Poor performance** during intensive JavaScript operations

## File Structure

```
fitnesswise/
├── public/
│   ├── index.html
│   ├── manifest.json
│   ├── service-worker.js
│   └── assets/
│       ├── icons/
│       └── styles/
├── src/
│   ├── components/
│   │   ├── app.js
│   │   ├── timer.js
│   │   ├── workout.js
│   │   └── navigation.js
│   ├── services/
│   │   ├── firebase.js
│   │   ├── storage.js
│   │   └── analytics.js
│   ├── data/
│   │   └── workouts.js
│   └── utils/
│       └── helpers.js
├── README.md
└── package.json
```

## Conclusion

This requirements document captures the lessons learned from extensive development sessions and real-world usage feedback. The architecture must prioritize reliability and user experience over feature complexity, ensuring that Eddie can maintain his daily workout routine without technical friction while providing a foundation for advanced features in future phases.

The key insight is that fitness applications must work flawlessly during physical exertion when users cannot tolerate technical problems. Every design decision should support this core requirement while creating a professional-grade experience that demonstrates the value of user-centered development practices.