# **TECHNICAL DESIGN DOCUMENT**

**PROJECT TITLE: FitLife**

**One-sentence pitch:** A modern, personalized fitness web application that provides tailored workout plans, nutrition guidance, mental health support, and progress tracking with offline capabilities.

## **1. OVERVIEW**

**Goal:**

* Solve the problem of generic, one-size-fits-all fitness apps by providing personalized workout and nutrition plans
* Address mental health aspects of fitness journeys with integrated wellness features and emotional support
* Enable users to maintain their fitness routine regardless of internet connectivity with robust offline mode

**Key features:**

* Personalized onboarding with goal-based customization (age, weight, height, activity level)
* Exercise library with YouTube integration, duration tracking, and illustrated descriptions
* Nutrition plans with recipes, meal planning, and healthy snack suggestions
* Mental health support including breathing exercises, mood playlists, and motivational quotes
* Vent journal for emotional expression and reflection
* Motivational popup system using user's personal reasons
* Offline mode with data synchronization
* Progress tracking dashboard with stats and achievements

**Target users & success criteria:**

* **Primary users:** Health-conscious individuals aged 18-45 seeking personalized fitness guidance
* **Secondary users:** Fitness beginners looking for structured, approachable workout plans
* **Success criteria:** 70% weekly retention, 60% workout completion rate, 4.5+ app store rating, 15-20 minute average session duration

## **2. TECH STACK (GOLDEN PATH)**

| **Component** | **Technology** | **Justification** |
| --- | --- | --- |
| **Runtime** | Node.js (Firebase Cloud Functions) | Serverless scaling, Firebase ecosystem integration |
| **Language** | TypeScript (strict) | Type safety, better developer experience |
| **Front-end** | React + Vite | Fast development, modern build tooling |
| **UI Kit** | shadcn/ui (Radix + Tailwind) | Accessible components, customizable design system |
| **Styling** | Tailwind CSS | Utility-first, consistent design tokens |
| **State Management** | TanStack Query + Zustand | Server state + client state separation |
| **Forms & Validation** | React Hook Form + Zod | Performance, type-safe validation |
| **Shared Validation** | Zod | Client & server schema consistency |
| **API Layer** | tRPC | End-to-end type safety |
| **Backend Services** | Firebase Auth · Firestore · Storage · Functions | Integrated ecosystem, real-time updates |
| **Package Manager** | PNPM workspaces | Efficient dependency management |
| **Build Orchestration** | Turborepo | Monorepo build optimization |
| **Component Workshop** | Storybook | UI component isolation and testing |
| **Unit/Component Tests** | Vitest + Testing Library | Fast, modern testing framework |
| **Visual/Interaction** | Storybook + @storybook/testing-library | Visual regression testing |
| **End-to-end Tests** | Playwright | Cross-browser e2e testing |
| **Linting** | ESLint + eslint-plugin-perfectionist | Code quality, consistent formatting |
| **Formatting** | Prettier | Code formatting consistency |
| **Type-safe Env Vars** | T3 Env (Zod-validated) | Runtime environment validation |
| **Versioning** | Changesets | Monorepo changelog management |
| **CI/CD** | GitHub Actions | Automated testing and deployment |
| **PWA/Offline** | Workbox | Service worker management |

## **3. MONOREPO LAYOUT (PNPM)**

****fitlife/

├── apps/

│ └── web/ ← React front-end + PWA config

│ ├── src/

│ │ ├── components/ ← Reusable UI components

│ │ ├── pages/ ← Page components

│ │ ├── hooks/ ← Custom React hooks

│ │ ├── store/ ← Zustand stores

│ │ ├── utils/ ← Helper functions

│ │ └── types/ ← TypeScript definitions

│ ├── public/ ← Static assets, PWA manifest

│ └── .storybook/ ← Storybook configuration

├── functions/ ← Cloud Functions

│ ├── src/

│ │ ├── routers/ ← tRPC route handlers

│ │ ├── services/ ← Business logic

│ │ └── utils/ ← Server utilities

│ └── firestore.rules ← Security rules

├── packages/

│ ├── shared/ ← Shared utilities and schemas

│ │ ├── schemas/ ← Zod validation schemas

│ │ ├── types/ ← Shared TypeScript types

│ │ └── utils/ ← Common utilities

│ ├── seeding/ ← Data seeding for development

│ └── config/ ← Shared configuration

│ ├── eslint/ ← ESLint configurations

│ ├── tailwind/ ← Tailwind config

│ └── typescript/ ← TypeScript configurations

├── docs/ ← Project documentation

│ ├── technical-design-doc.md

│ ├── api-reference.md

│ └── deployment-guide.md

├── .github/ ← GitHub Actions workflows

│ └── workflows/

└── firebase.json ← Firebase configuration



## **4. ARCHITECTURE**

****┌─────────────────────────────────────────────────────────────┐

│ CLIENT LAYER │

├─────────────────────────────────────────────────────────────┤

│ React Components (Pages, UI Components) │

│ ↓ │

│ State Management (TanStack Query + Zustand) │

│ ↓ │

│ tRPC Client (Type-safe API calls) │

│ ↓ │

│ Service Worker (Offline capabilities) │

└─────────────────────────────────────────────────────────────┘

↕ HTTPS

┌─────────────────────────────────────────────────────────────┐

│ SERVER LAYER │

├─────────────────────────────────────────────────────────────┤

│ tRPC Routers (API endpoints) │

│ ↓ │

│ Business Logic Services │

│ ↓ │

│ Firebase Services (Auth, Firestore, Storage) │

└─────────────────────────────────────────────────────────────┘

**Data Flow:**

1. User interacts with React components
2. Components use TanStack Query for server state
3. Zustand manages client-side state (UI state, offline data)
4. tRPC provides type-safe API communication
5. Service Worker handles offline scenarios and caching
6. Server-side tRPC routers process requests
7. Firebase services handle authentication and data persistence

## **5. DATA MODEL**

| **Entity** | **Key Fields** | **Notes** |
| --- | --- | --- |
| **User** | uid, email, profile, createdAt, updatedAt | Auth via Firebase, profile includes onboarding data |
| **UserProfile** | age, height, weight, goal, activityLevel, reasons[] | Collected during onboarding |
| **Workout** | id, userId, exerciseId, duration, completedAt, caloriesBurned | Individual workout sessions |
| **Exercise** | id, name, description, type, difficulty, duration, youtubeUrl, imageUrl | Exercise library |
| **MealPlan** | id, userId, date, breakfast, lunch, dinner, snacks[] | Daily nutrition plans |
| **Recipe** | id, name, ingredients[], instructions[], prepTime, calories, difficulty | Recipe database |
| **JournalEntry** | id, userId, content, mood, type, createdAt | Vent journal entries |
| **ProgressStats** | userId, workoutsCompleted, currentStreak, totalMinutes, caloriesBurned | Dashboard metrics |
| **Quote** | id, content, author, category, isPersonal | Motivational quotes + user reasons |

**Security Rules:**

* Users can only access their own data
* Exercise and Recipe collections are read-only for users
* Quotes collection is read-only
* Admin role for content management

**Index Strategy:**

* Composite indexes on userId + createdAt for time-based queries
* Index on exercise.type and exercise.difficulty for filtering
* Index on recipe.difficulty and recipe.prepTime for meal planning

## **6. API DESIGN (tRPC)**

| **Router** | **Procedure** | **Input (Zod Schema)** | **Output** | **Description** |
| --- | --- | --- | --- | --- |
| **auth** | getProfile | { uid: string } | UserProfile | Get user profile data |
| **auth** | updateProfile | UpdateProfileSchema | UserProfile | Update user profile |
| **onboarding** | completeOnboarding | OnboardingSchema | UserProfile | Save onboarding data |
| **exercises** | getAll | ExerciseFiltersSchema | Exercise[] | Get filtered exercises |
| **exercises** | getById | { id: string } | Exercise | Get single exercise |
| **exercises** | getRecommended | { userId: string } | Exercise[] | Get personalized recommendations |
| **workouts** | log | WorkoutLogSchema | Workout | Log completed workout |
| **workouts** | getHistory | { userId: string, limit?: number } | Workout[] | Get workout history |
| **nutrition** | getMealPlan | { userId: string, date: string } | MealPlan | Get daily meal plan |
| **nutrition** | getRecipes | RecipeFiltersSchema | Recipe[] | Get filtered recipes |
| **nutrition** | generateMealPlan | { userId: string, preferences: object } | MealPlan | Generate personalized meal plan |
| **journal** | createEntry | JournalEntrySchema | JournalEntry | Create journal entry |
| **journal** | getEntries | { userId: string, limit?: number } | JournalEntry[] | Get journal entries |
| **stats** | getProgress | { userId: string } | ProgressStats | Get user progress stats |
| **stats** | updateProgress | UpdateProgressSchema | ProgressStats | Update progress metrics |
| **quotes** | getRandom | { category?: string } | Quote | Get random motivational quote |
| **quotes** | getPersonal | { userId: string } | Quote[] | Get user's personal reasons as quotes |

**Error Handling Conventions:**

* **Authentication errors:** UNAUTHORIZED (401) for missing/invalid tokens
* **Validation errors:** BAD\_REQUEST (400) with Zod error details
* **Not found errors:** NOT\_FOUND (404) for non-existent resources
* **Rate limiting:** TOO\_MANY\_REQUESTS (429) for API abuse
* **Server errors:** INTERNAL\_SERVER\_ERROR (500) with sanitized messages

## **7. TESTING STRATEGY**

| **Level/Focus** | **Toolset** | **Scope** | **Coverage Target** |
| --- | --- | --- | --- |
| **Unit** | Vitest | Pure functions, utilities, hooks | 90% |
| **Component** | Vitest + Testing Library | React components, user interactions | 85% |
| **Visual/Interaction** | Storybook + @storybook/testing-library | UI snapshots, component states | 80% |
| **Integration** | Vitest + MSW | API integration, data flow | 75% |
| **End-to-end** | Playwright | User journeys, critical paths | Key flows |

**Test Scenarios:**

* **Unit Tests:** Validation schemas, utility functions, custom hooks
* **Component Tests:** Form validation, user interactions, state changes
* **Visual Tests:** Component variants, responsive design, accessibility
* **Integration Tests:** API calls, data persistence, offline scenarios
* **E2E Tests:** Onboarding flow, workout completion, journal entry

**Fixtures & Seeding:**

* pnpm seed runs scripts in packages/seeding against Firebase emulator
* Mock data for exercises, recipes, and quotes
* Test user profiles with various configurations

## **8. CI/CD PIPELINE (GITHUB ACTIONS)**

### **Development Workflow**

****name: Development

on: [push, pull\_request]

jobs:

test:

steps:

- Setup PNPM and restore Turbo remote cache

- pnpm exec turbo run lint typecheck

- pnpm exec turbo run test

- pnpm exec turbo run build-storybook

- pnpm exec turbo run e2e

- Deploy preview (Firebase Hosting channel)

### **Production Deployment**

****name: Production

on:

push:

branches: [main]

jobs:

deploy:

steps:

- Run full test suite

- Build production artifacts

- Deploy to Firebase Hosting

- Deploy Cloud Functions

- Update Firestore security rules

- Run post-deployment health checks

**Pipeline Features:**

* Turbo-aware caching for faster builds
* Parallel test execution
* Automatic preview deployments for PRs
* Changesets for version management
* Slack notifications for deployment status

## **9. ENVIRONMENTS & SECRETS**

| **Environment** | **URL/Target** | **Firebase Project** | **Notes** |
| --- | --- | --- | --- |
| **local** | localhost:5173 | fitlife-dev | .env + Firebase emulators |
| **staging** | https://fitlife-staging.web.app | fitlife-staging | Auto-deploy from develop branch |
| **preview-**\*\* | Firebase Hosting channel | fitlife-staging | Auto-created per PR |
| **production** | https://fitlife.app | fitlife-prod | Manual promotion from staging |

**Environment Variables:**

****// T3 Env validation

const env = createEnv({

server: {

FIREBASE\_ADMIN\_KEY: z.string(),

YOUTUBE\_API\_KEY: z.string(),

OPENAI\_API\_KEY: z.string().optional(),

},

client: {

VITE\_FIREBASE\_CONFIG: z.string(),

VITE\_APP\_VERSION: z.string(),

},

runtimeEnv: {

FIREBASE\_ADMIN\_KEY: process.env.FIREBASE\_ADMIN\_KEY,

YOUTUBE\_API\_KEY: process.env.YOUTUBE\_API\_KEY,

VITE\_FIREBASE\_CONFIG: process.env.VITE\_FIREBASE\_CONFIG,

VITE\_APP\_VERSION: process.env.VITE\_APP\_VERSION,

},

});

**Secrets Management:**

* GitHub repository secrets for CI/CD
* Firebase Functions config for server-side secrets
* Local development uses .env.local (gitignored)

## **10. PERFORMANCE & SCALABILITY**

### **Frontend Optimizations**

* **Code Splitting:** Route-based splitting with React.lazy()
* **Image Optimization:** WebP format with fallbacks, lazy loading
* **Bundle Analysis:** Webpack Bundle Analyzer integration
* **Caching Strategy:** TanStack Query with optimistic updates

### **Backend Optimizations**

* **Firestore Denormalization:** Duplicate data to avoid complex queries
* **Hot Document Prevention:** Distribute writes across multiple documents
* **Cloud Functions:** Cold start optimization with smaller bundle sizes
* **CDN:** Firebase Hosting CDN for static assets

### **Scalability Considerations**

* **Database:** Firestore scales automatically, monitoring query performance
* **Functions:** Auto-scaling Cloud Functions with concurrency limits
* **Storage:** Firebase Storage for exercise images and user uploads
* **Rate Limiting:** Implement per-user API rate limits

## **11. MONITORING & LOGGING**

| **Concern** | **Tool** | **Implementation** | **Alerts** |
| --- | --- | --- | --- |
| **Runtime Errors** | Firebase Crashlytics | Automatic crash reporting | Slack notifications |
| **Performance** | Firebase Performance | Web vitals monitoring | Performance degradation |
| **Server Logs** | Google Cloud Logging | Structured JSON logging | Error rate thresholds |
| **User Analytics** | Google Analytics 4 | Event tracking, funnels | Retention drops |
| **Uptime** | Firebase Hosting | Built-in monitoring | Service downtime |
| **API Metrics** | Cloud Functions | Request/response metrics | High error rates |

**Key Metrics to Track:**

* User engagement (session duration, page views)
* Feature adoption (onboarding completion, workout logs)
* Performance (page load times, API response times)
* Error rates (client/server errors, crash frequency)
* Business metrics (retention, workout completion rates)

## **12. ACCESSIBILITY & I18N**

### **Accessibility (WCAG 2.1 AA)**

* **Components:** shadcn/ui with Radix primitives (built-in ARIA)
* **Testing:** Storybook a11y addon for automated audits
* **Manual Testing:** Keyboard navigation, screen reader compatibility
* **Color Contrast:** 4.5:1 ratio for normal text, 3:1 for large text
* **Focus Management:** Visible focus indicators, logical tab order

### **Internationalization (Future)**

* **Framework:** react-i18next for translations
* **Localization:** Support for Spanish, French, German initially
* **Content:** Exercise descriptions, UI text, error messages
* **Formatting:** Date/time, numbers, currency based on locale

## **13. CODE QUALITY & FORMATTING**

### **Code Standards**

****{

"prettier": {

"semi": true,

"singleQuote": true,

"tabWidth": 2,

"trailingComma": "es5"

},

"eslint": {

"extends": [

"@typescript-eslint/recommended",

"plugin:react-hooks/recommended",

"plugin:perfectionist/recommended-natural"

]

}

}

### **Git Workflow**

* **Conventional Commits:** Standard commit message format
* **Husky:** Pre-commit hooks for linting and formatting
* **lint-staged:** Only lint changed files
* **Changesets:** Automated changelog generation

## **14. OFFLINE CAPABILITIES**

### **Service Worker Strategy**

****// Workbox configuration

{

runtimeCaching: [

{

urlPattern: /^https:\/\/api\.fitlife\.app\//,

handler: 'NetworkFirst',

options: {

cacheName: 'api-cache',

expiration: { maxEntries: 50, maxAgeSeconds: 24 \* 60 \* 60 }

}

},

{

urlPattern: /\.(?:png|jpg|jpeg|svg|gif|webp)$/,

handler: 'CacheFirst',

options: { cacheName: 'image-cache' }

}

]

}

### **Offline Features**

* **Cached Workouts:** Download exercise routines for offline use
* **Progress Tracking:** Store completed workouts locally, sync when online
* **Journal Entries:** Write entries offline, sync when connection restored
* **Exercise Database:** Cache frequently accessed exercises
* **Background Sync:** Queue API calls for when connection returns

## **15. SECURITY CONSIDERATIONS**

### **Authentication & Authorization**

* **Firebase Auth:** Email/password, Google OAuth, Apple Sign-in
* **JWT Tokens:** Secure token-based authentication
* **Role-Based Access:** User, admin, content manager roles
* **Session Management:** Automatic token refresh, secure logout

### **Data Protection**

* **Firestore Security Rules:** Row-level security based on user ID
* **Input Validation:** Server-side validation with Zod schemas
* **XSS Protection:** Sanitize user inputs, CSP headers
* **CSRF Protection:** SameSite cookies, CSRF tokens

### **Privacy Compliance**

* **Data Minimization:** Only collect necessary user data
* **Consent Management:** Clear privacy policy, user consent
* **Data Retention:** Automatic cleanup of old data
* **Export/Delete:** User data export and deletion capabilities

## **16. OPEN QUESTIONS / RISKS**

| **Item** | **Risk Level** | **Owner** | **Resolution Date** | **Notes** |
| --- | --- | --- | --- | --- |
| YouTube API rate limits | Medium | Backend Team | 2024-01-15 | Need to implement caching strategy |
| Offline data sync conflicts | High | Frontend Team | 2024-01-20 | Define conflict resolution strategy |
| Firebase costs at scale | Medium | DevOps | 2024-02-01 | Monitor usage, implement cost controls |
| Exercise content licensing | High | Legal Team | 2024-01-25 | Ensure proper licensing for exercise images |
| GDPR compliance | Medium | Legal Team | 2024-02-15 | Implement data protection measures |

## **17. DEPLOYMENT STRATEGY**

### **Infrastructure as Code**

* **Firebase Projects:** Dev, staging, production environments
* **Hosting Configuration:** Custom domains, SSL certificates
* **Security Rules:** Version-controlled Firestore/Storage rules
* **Environment Variables:** Managed through Firebase Functions config

### **Release Process**

1. **Development:** Feature branches merged to develop
2. **Staging:** Automatic deployment from develop branch
3. **Testing:** QA testing on staging environment
4. **Production:** Manual promotion after approval
5. **Rollback:** Automatic rollback on health check failures

## **18. APPENDICES**

### **Setup Instructions**

****# Initial setup

pnpm install

pnpm exec turbo run setup

# Development

pnpm dev

# Testing

pnpm test

pnpm test:e2e

# Building

pnpm build

pnpm build-storybook

### **Useful Links**

* **Design System:** [Figma Design File](https://figma.com/fitlife-design)
* **API Documentation:** [Storybook API Docs](https://storybook.fitlife.app/)
* **Component Library:** [Design System Storybook](https://storybook.fitlife.app/)
* **Project Board:** [GitHub Project](https://github.com/orgs/fitlife/projects/1)

**Last Updated:** 2024-01-10 **Document Version:** 1.0.0 **Next Review:** 2024-02-10