CLST Digital Closet - Developer Documentation

Table of Contents

Project Overview

CLST is a modern fashion/closet management platform designed to help users digitally organize their wardrobe, create outfits, participate in style challenges, and engage with fashion trends. The platform emphasizes accessibility, performance, and user experience with a minimalist black and white design aesthetic.

Core Value Proposition

- **Digital Closet Management**: Complete wardrobe digitization and organization
- **Outfit Creation**: AI-powered outfit suggestions with weather integration
- **Fashion Marketplace**: Buy, sell, and trade fashion items
- **Style Challenges**: Community-driven fashion challenges and competitions
- **Trend Tracking**: Articles, trending styles, and color analysis
- **Social Features**: Share outfits, follow users, and engage with the fashion community

Target Platforms & Devices

CLST is built as a **progressive web application (PWA)** to ensure maximum reach and consistent experience across all platforms:

Primary Platforms

- **Mobile Web** (Primary focus - 70% of users)
    - iOS Safari 14+
    - Chrome for Android 90+
    - Samsung Internet 14+
- **Desktop Web** (30% of users)
    - Chrome 90+
    - Safari 14+
    - Firefox 88+
    - Edge 90+

Mobile App Distribution

- **iOS**: PWA via Safari "Add to Home Screen" (iOS 14.5+)
- **Android**: PWA via Chrome/Play Store TWA (Trusted Web Activity)
- **Desktop**: PWA installation supported on all major browsers

Device Requirements

- **Minimum Screen Size**: 320px width (iPhone SE)
- **Recommended**: 375px+ width for optimal experience
- **Tablet Support**: Responsive layouts up to 1024px
- **Desktop**: Optimized for 1280px - 1920px viewports

Total Addressable Market (TAM)

Market Size

- **Global Fashion E-commerce Market**: $871.2 billion (2023)
- **Digital Wardrobe Management Segment**: $2.4 billion (2023)
- **Expected CAGR**: 14.8% (2023-2030)

Target Demographics

1. **Primary Market** (Ages 18-35)
    - Fashion-conscious millennials and Gen Z
    - 65% female, 35% male/non-binary
    - Urban and suburban locations
    - Household income: $35,000+
    - **Estimated Users**: 180 million globally
2. **Secondary Market** (Ages 36-50)
    - Professional wardrobe management
    - Sustainable fashion advocates
    - Higher disposable income
    - **Estimated Users**: 90 million globally

Geographic Distribution

- **North America**: 35% (94.5M users)
- **Europe**: 30% (81M users)
- **Asia-Pacific**: 25% (67.5M users)
- **Rest of World**: 10% (27M users)

Market Penetration Strategy

- **Year 1**: 0.1% market penetration (270,000 users)
- **Year 3**: 1% market penetration (2.7M users)
- **Year 5**: 5% market penetration (13.5M users)

Revenue Potential

- **Average Revenue Per User (ARPU)**: $24/year
- **Premium Subscription Rate**: 15% of users

- **Marketplace Transaction Fee**: 8% per sale
- **Year 5 Revenue Target**: $324M annually

Technical Stack

Frontend

- **Framework**: React 18+ with TypeScript
- **Build Tool**: Vite
- **State Management**: Zustand
- **Server State**: React Query (TanStack Query)
- **Routing**: React Router v6
- **Styling**: Tailwind CSS (black and white only for WCAG AAA compliance)
- **Testing**: Vitest (unit/integration), Playwright (E2E)
- **Internationalization**: i18next

Backend Services

- **API**: RESTful API with TypeScript/Node.js
- **Database**: PostgreSQL with Redis caching
- **File Storage**: Cloud storage for images (S3-compatible)
- **Authentication**: JWT-based with refresh tokens
- **Real-time**: WebSocket for notifications

Infrastructure

- **Container**: Docker
- **CI/CD**: GitHub Actions
- **Monitoring**: Sentry for error tracking, custom performance monitoring
- **Analytics**: Mixpanel integration
- **CDN**: Cloudflare for static assets

Architecture & Design Principles

Core Principles

1. **Zero Technical Debt Approach**: Clean, maintainable code from the start
2. **Mobile-First Responsive Design**: Optimized for mobile devices
3. **Offline-First Architecture**: Advanced caching with service workers
4. **Accessibility First**: WCAG AAA compliance with black/white design
5. **Performance Obsessed**: Sub-second load times, 60fps interactions

Architectural Patterns

- **Feature-Based Folder Structure**: Organized by business domains
- **Dependency Injection**: For testability and flexibility
- **Repository Pattern**: For data access abstraction
- **Optimistic UI Updates**: For perceived performance
- **Progressive Enhancement**: Core functionality without JavaScript

Project Structure

```
src/
├── app/                    # Application shell and routing
|   ├── router/            # Route definitions
|   ├── providers/         # Global providers (auth, theme, etc.)
|   └── layouts/           # Layout components
├── features/              # Feature modules
|   ├── closet/           # Digital closet management
|   ├── outfits/          # Outfit creation and planning
|   ├── marketplace/      # Fashion marketplace
|   ├── challenges/       # Style challenges
|   ├── trends/           # Trend tracking
|   ├── auth/             # Authentication
|   └── profile/          # User profiles
├── shared/               # Shared resources
|   ├── components/       # Reusable UI components
|   ├── hooks/           # Custom React hooks
|   ├── utils/           # Utility functions
|   ├── services/        # API services
|   ├── types/           # TypeScript types
|   └── constants/       # App constants
├── assets/              # Static assets
└── tests/               # Test utilities and fixtures
```

Core Features

1. Digital Closet Module

- **Item Management**: Add, edit, delete clothing items
- **Smart Categorization**: AI-powered auto-tagging
- **Image Upload**: Multi-image support with optimization
- **Advanced Filtering**: By category, color, season, occasion
- **Virtual Scrolling**: Handle thousands of items efficiently
- **Batch Operations**: Select and manage multiple items

2. Outfit Creation & Planning

- **Outfit Builder**: Drag-and-drop interface
- **Weather Integration**: Location-based weather recommendations
- **Weekly Planning**: Plan outfits for the week ahead
- **AI Suggestions**: ML-powered outfit recommendations
- **Outfit History**: Track worn outfits
- **Share & Export**: Social sharing and calendar export

3. Fashion Marketplace

- **Listing Management**: Create and manage listings
- **Search & Discovery**: Advanced search with filters
- **Secure Transactions**: Escrow-based payments
- **Ratings & Reviews**: Buyer/seller feedback system
- **Shipping Integration**: Label generation and tracking
- **Price Suggestions**: AI-based pricing recommendations

4. Style Challenges

- **Challenge Types**: Daily, weekly, seasonal challenges
- **Progress Tracking**: Visual progress indicators
- **Leaderboards**: Global and friend rankings
- **Rewards System**: Points, badges, achievements
- **Community Voting**: Peer-based judging
- **Challenge Creation**: User-generated challenges

5. Trend Tracking

- **Trend Articles**: Curated fashion content
- **Color Trends**: Seasonal color analysis
- **Style Analytics**: Personal style insights
- **Trend Predictions**: AI-powered trend forecasting
- **Influencer Integration**: Follow fashion influencers
- **Personalized Feed**: ML-driven content recommendations

Development Setup

Prerequisites

- Node.js 18.x or higher
- pnpm 8.x
- Git
- Docker (for backend services)

Installation Steps

```bash
# Clone the repository
git clone https://github.com/your-org/clst.git
cd clst

# Install dependencies
pnpm install

# Set up environment variables
cp .env.example .env.local

# Start development server
pnpm dev

# Start backend services (in another terminal)
docker-compose up -d
```

Environment Variables

```
# API Configuration
VITE_API_URL=http://localhost:3000
VITE_API_VERSION=v1

# Authentication
VITE_AUTH_DOMAIN=auth.clst.local
VITE_AUTH_CLIENT_ID=your-client-id

# Third-party Services
VITE_SENTRY_DSN=your-sentry-dsn
VITE_MIXPANEL_TOKEN=your-mixpanel-token
VITE_WEATHER_API_KEY=your-weather-api-key
```

```
# Feature Flags
VITE_ENABLE_AI_FEATURES=true
VITE_ENABLE_MARKETPLACE=true
```

Code Standards

TypeScript Guidelines

- **Strict Mode**: Always use strict TypeScript settings
- **Type Coverage**: 100% type coverage required
- **No Any**: Avoid `any` type, use `unknown` when needed
- **Interface vs Type**: Use interfaces for objects, types for unions
- **Explicit Return Types**: Always declare function return types

React Best Practices

- **Functional Components**: Use hooks, no class components
- **Custom Hooks**: Extract business logic into custom hooks
- **Memo Wisely**: Use React.memo for expensive components
- **Error Boundaries**: Wrap features in error boundaries
- **Accessibility**: ARIA labels, keyboard navigation, screen reader support

Styling Guidelines

- **Tailwind Classes**: Use utility classes, avoid custom CSS
- **Color Palette**: Black (#000000) and white (#FFFFFF) only
- **Responsive**: Mobile-first approach with responsive utilities
- **Component Library**: Use shared UI components
- **Dark Mode**: Not applicable (black/white design)

Git Workflow

- **Branch Naming**: `feature/`, `fix/`, `chore/` prefixes
- **Commit Messages**: Follow conventional commits
- **PR Process**: Code review required, all tests must pass
- **Squash Merges**: Keep main branch history clean

Testing Strategy

Unit Tests

```
// Example: Testing a custom hook
describe('useClosetItems', () => {
  it('filters items by category', async () => {
    const { result } = renderHook(() => useClosetItems({ category: 'tops' }));

    await waitFor(() => {
      expect(result.current.items).toHaveLength(5);
      expect(result.current.items.every(item => item.category === 'tops')).toBe(true);
    });
  });
});
```

Integration Tests

- Test feature workflows end-to-end
- Mock external services
- Test error scenarios
- Verify accessibility

E2E Tests

- Critical user journeys
- Cross-browser testing
- Mobile device testing
- Performance benchmarks

Coverage Requirements

- Minimum 80% code coverage
- 100% coverage for utilities
- Critical paths must have E2E tests

Performance Requirements

Core Web Vitals

- **LCP**: < 2.5s (Largest Contentful Paint)
- **FID**: < 100ms (First Input Delay)
- **CLS**: < 0.1 (Cumulative Layout Shift)
- **FCP**: < 1.8s (First Contentful Paint)

Application Metrics

- **Initial Load**: < 3s on 3G
- **Route Changes**: < 200ms
- **API Responses**: < 500ms (p95)
- **Frame Rate**: 60fps for animations
- **Bundle Size**: < 200KB initial JS

Optimization Techniques

- Code splitting by route
- Image lazy loading
- Virtual scrolling for lists
- Service worker caching
- CDN for static assets
- WebP image format
- Brotli compression

Security Implementation

Authentication & Authorization

- JWT tokens with refresh mechanism
- Role-based access control (RBAC)
- Session management
- Multi-factor authentication support

Data Protection

- HTTPS everywhere
- Input sanitization
- SQL injection prevention
- XSS protection
- CSRF tokens
- Rate limiting

Privacy & Compliance

- GDPR compliance
- Data encryption at rest
- PII handling guidelines
- Audit logging
- Cookie consent

Deployment & CI/CD

CI/CD Pipeline

```yaml
# GitHub Actions workflow
name: CLST CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  quality:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install dependencies
        run: pnpm install --frozen-lockfile
      - name: Lint
        run: pnpm lint
      - name: Type check
        run: pnpm type-check
      - name: Security audit
        run: pnpm audit --production

  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install dependencies
        run: pnpm install --frozen-lockfile
      - name: Run tests
        run: pnpm test:ci
      - name: Upload coverage
        uses: codecov/codecov-action@v3
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Build application
        run: pnpm build
      - name: Check bundle size
        run: pnpm size-limit
```

Deployment Environments

- **Development**: Auto-deploy from develop branch
- **Staging**: Manual deploy for QA testing
- **Production**: Manual deploy with approval

Monitoring & Alerts

- Error tracking with Sentry
- Performance monitoring
- Uptime monitoring
- Custom business metrics
- Slack notifications

API Documentation

Authentication Endpoints

```
POST   /api/v1/auth/login         # User login
POST   /api/v1/auth/logout        # User logout
POST   /api/v1/auth/refresh       # Refresh token
POST   /api/v1/auth/register      # User registration
POST   /api/v1/auth/forgot        # Password reset
```

Closet Endpoints

```
GET    /api/v1/closet/items       # List items (paginated)
POST   /api/v1/closet/items       # Create item
GET    /api/v1/closet/items/:id   # Get item details
PUT    /api/v1/closet/items/:id   # Update item
DELETE /api/v1/closet/items/:id   # Delete item
POST   /api/v1/closet/items/bulk  # Bulk operations
```

Response Format

```
// Success response
{
  "success": true,
  "data": { /* response data */ },
  "meta": {
    "page": 1,
    "limit": 20,
    "total": 100,
    "totalPages": 5
  }
}

// Error response
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data",
    "details": { /* field-specific errors */ }
  }
}
```

Contributing Guidelines

Getting Started

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Write/update tests
5. Update documentation
6. Submit a pull request

Code Review Checklist

- [ ] TypeScript types are correct
- [ ] Tests are passing
- [ ] Documentation is updated
- [ ] Accessibility is maintained
- [ ] Performance impact is acceptable
- [ ] Security best practices followed

Release Process

1. Version bump following semver
2. Update CHANGELOG.md
3. Create release notes
4. Tag release
5. Deploy to production

Support & Resources

Documentation

- API Reference
- Component Library
- Architecture Decision Records

Communication

- Slack: #clst-dev
- Email: dev@clst.app
- Issue Tracker: GitHub Issues

Additional Resources

- Figma Designs
- Brand Guidelines

- Security Policy

Last updated: August 2024