ImageFlow - Custom Components Development Plan

Author: Sinem Eissler

Date: July 18, 2025

---

1. Overview

This document details all custom components that will be developed from scratch for ImageFlow, demonstrating comprehensive software development skills without relying on third-party services. Each component represents significant coding effort and technical complexity.

---

2. Frontend Components (React)

2.1 Canvas Editor Engine

CanvasEditor Component (~2000 lines)

```javascript
// Core canvas management with custom rendering pipeline
class CanvasEditor {
  - Layer management system
  - Custom rendering loop with requestAnimationFrame
  - Zoom/pan implementation with transform matrices
  - Undo/redo system using Command pattern
  - Selection tools with marching ants animation
  - Viewport culling for performance
}
```

Layer System (~1500 lines)

```javascript
// Custom layer implementation with blending modes
class LayerManager {
  - Layer stack with z-index management
  - 15+ blend mode implementations (multiply, screen, overlay, etc.)
  - Layer effects (drop shadow, glow, stroke)
  - Smart layer caching with dirty rectangle optimization
  - Layer groups and clipping masks
}
```

Drawing Tools (~1200 lines)

```javascript
// Custom brush engine with pressure sensitivity
class BrushEngine {
  - Bezier curve smoothing algorithm
  - Pressure interpolation for tablet support
  - Custom brush textures and patterns
  - Stabilizer for smooth lines
  - Vector and raster brush modes
}

class SelectionTools {
  - Magic wand with flood fill algorithm
  - Lasso tool with polygon detection
  - Quick selection with edge detection
  - Selection transformations and modifications
}
```

Transform Tools (~800 lines)

```javascript
// Custom transformation matrix implementation
class TransformManager {
  - Free transform with corner/edge handles
  - Perspective transform with 4-point distortion
  - Warp tool with mesh deformation
  - Content-aware scaling algorithm
}
```

2.2 Image Processing Library

Filter Engine (~1500 lines)

```javascript
```

```
// WebGL-accelerated custom filters
class FilterEngine {
  - Convolution kernel processor
  - Color adjustment algorithms (HSL, curves, levels)
  - Blur implementations (Gaussian, motion, radial)
  - Sharpen and edge detection filters
  - Custom shader compiler for effects
}
```

Color Analysis (~600 lines)

```
javascript

// Color science implementations
class ColorAnalyzer {
  - K-means clustering for palette extraction
  - Histogram calculation and equalization
  - Color harmony detection
  - Dominant color algorithm with perceptual weighting
}
```

2.3 Social Networking Components

Activity Feed (~1000 lines)

```
javascript

// Custom infinite scroll with virtualization
class ActivityFeed {
  - Virtual scroll implementation for performance
  - Real-time updates with WebSocket integration
  - Optimistic UI updates with rollback
  - Smart batching for network efficiency
  - Intersection Observer for lazy loading
}
```

Follow System (~700 lines)

```
javascript
```

```
// Social graph visualization and management
class SocialGraph {
  - Follow/unfollow with optimistic updates
  - Mutual follow detection
  - Follower/following lists with search
  - Follow suggestions algorithm
  - Privacy settings management
}
```

2.4 Real-time Collaboration

Collaboration Manager (~1200 lines)

javascript

```
// WebSocket-based collaborative editing
class CollaborationEngine {
  - Operational Transformation (OT) implementation
  - Cursor tracking and rendering
  - Presence awareness system
  - Conflict resolution algorithms
  - Permission-based action filtering
}
```

Live Cursors (~500 lines)

javascript

```
// Multi-user cursor system
class CursorManager {
  - Smooth cursor interpolation
  - Name labels with collision detection
  - Tool preview for other users
  - Cursor trails and animations
  - Efficient position broadcasting
}
```

2.5 Search and Discovery

Search Interface (~800 lines)

javascript

```
// Advanced search with facets
class SearchEngine {
  - Autocomplete with fuzzy matching
  - Search query parser and builder
  - Faceted search UI with filters
  - Search history and suggestions
  - Visual search by color/composition
}
```

2.6 Version Control UI

Version Tree Visualizer (~900 lines)

```
javascript

// Git-like branch visualization
class VersionTree {
  - DAG rendering with D3.js
  - Interactive branch navigation
  - Diff visualization between versions
  - Merge conflict resolution UI
  - Version comparison slider
}
```

---

3. Backend Components (Node.js/Express)

3.1 Authentication System

JWT Authentication (~800 lines)

```
javascript

// Custom JWT implementation
class AuthenticationService {
  - RS256 token signing and verification
  - Refresh token rotation
  - Device fingerprinting
  - Brute force protection
  - Session management with Redis
}
```

Authorization Middleware (~600 lines)

```javascript
// Role and permission based access control
class AuthorizationEngine {
  - Dynamic permission calculation
  - Resource-based access control
  - Hierarchical role system
  - API rate limiting per user tier
  - Audit logging for security events
}
```

3.2 Image Processing Engine

Server-side Processing (~2000 lines)

```javascript
// High-performance image manipulation
class ImageProcessor {
  - Multi-threaded processing with worker pools
  - Custom thumbnail generation with smart cropping
  - Progressive JPEG encoding
  - WebP conversion with quality optimization
  - EXIF data extraction and manipulation
}
```

Processing Pipeline (~1200 lines)

```javascript
// Async job queue implementation
class ProcessingPipeline {
  - Custom job queue with priorities
  - Retry logic with exponential backoff
  - Progress tracking and reporting
  - Resource pooling for efficiency
  - Dead letter queue for failed jobs
}
```

3.3 Search Engine

Full-text Search (~1500 lines)

```javascript
// Custom search implementation
class SearchService {
  - Inverted index construction
  - TF-IDF ranking algorithm
  - Fuzzy matching with Levenshtein distance
  - Synonym expansion
  - Query optimization and caching
}
```

Recommendation Engine (~1000 lines)

```javascript
// ML-based recommendations
class RecommendationService {
  - Collaborative filtering implementation
  - Content-based filtering
  - Hybrid recommendation algorithm
  - User preference learning
  - Trending content detection
}
```

3.4 Real-time Infrastructure

WebSocket Server (~1200 lines)

```javascript
// Custom WebSocket implementation
class RealtimeServer {
  - Room-based broadcasting
  - Presence management
  - Message queuing and delivery
  - Connection state recovery
  - Horizontal scaling with Redis pub/sub
}
```

Operational Transformation (~1500 lines)

```javascript
```

```
// OT algorithm for concurrent editing
class OTEngine {
  - Transform function implementation
  - Operation composition and inversion
  - State vector for causality
  - Conflict resolution strategies
  - Undo/redo in collaborative context
}
```

3.5 Version Control System

Version Management (~1800 lines)

```javascript
// Git-inspired version control
class VersionControl {
  - Delta compression algorithm
  - Three-way merge implementation
  - Branch management
  - Conflict detection and resolution
  - Garbage collection for old versions
}
```

3.6 Social Graph Engine

Graph Operations (~1200 lines)

```javascript
// Efficient social graph queries
class SocialGraphService {
  - Friend-of-friend calculations
  - Shortest path between users
  - Community detection algorithm
  - Influence scoring
  - Activity propagation in network
}
```

3.7 Notification System

Notification Engine (~900 lines)

```javascript
// Real-time notification delivery
class NotificationService {
  - Priority queue implementation
  - Batching and deduplication
  - Template engine for messages
  - Delivery tracking
  - User preference filtering
}
```

---

4. Database Layer

4.1 Query Builder

Custom ORM (~2000 lines)

```javascript
// Lightweight ORM for PostgreSQL
class QueryBuilder {
  - Fluent interface for query construction
  - Prepared statement management
  - Connection pooling
  - Transaction support
  - Migration system
}
```

4.2 Caching Layer

Cache Manager (~800 lines)

```javascript
// Multi-tier caching system
class CacheService {
  - LRU cache implementation
  - Cache invalidation strategies
  - Distributed caching with Redis
  - Cache warming algorithms
  - Hit rate monitoring
}
```

---

5. Infrastructure Components

5.1 File Storage

Storage Abstraction (~1000 lines)

```javascript
// Custom file storage system
class StorageService {
  - Chunked upload handling
  - File deduplication with hashing
  - CDN integration
  - Cleanup job scheduler
  - Storage quota management
}
```

5.2 Monitoring and Logging

Custom Logger (~600 lines)

```javascript
// Structured logging system
class LoggingService {
  - Log level management
  - Structured JSON logging
  - Request ID tracking
  - Performance metrics collection
  - Error aggregation and alerting
}
```

Monitoring System (~800 lines)

```javascript
// Application metrics collection
class MonitoringService {
  - Custom metrics collector
  - Request timing and profiling
  - Resource usage tracking
  - Health check endpoints
  - Dashboard data aggregation
}
```

5.3 Testing Framework

Test Utilities (~1000 lines)

```javascript
// Custom testing helpers
class TestFramework {
  - Mock data generators
  - Database test fixtures
  - API integration test runner
  - Performance benchmarking
  - Visual regression testing for canvas
}
```

---

6. Algorithm Implementations

6.1 Image Processing Algorithms

Smart Crop Algorithm (~400 lines)

```javascript
// Content-aware image cropping
function smartCrop(image, targetAspectRatio) {
  - Edge detection using Sobel operator
  - Saliency map generation
  - Face detection integration
  - Rule of thirds optimization
  - Entropy-based region scoring
}
```

Perceptual Hash (~300 lines)

```javascript
```

```
// Image similarity detection
function perceptualHash(image) {
  - DCT-based hash generation
  - Hamming distance calculation
  - Rotation-invariant features
  - Scale normalization
  - Color histogram comparison
}
```

6.2 Compression Algorithms

Custom Image Compression (~600 lines)

```
javascript

// Lossy compression implementation
class CompressionEngine {
  - Discrete Cosine Transform (DCT)
  - Quantization matrix optimization
  - Huffman encoding
  - Progressive encoding
  - Quality vs. size optimization
}
```

6.3 Search Algorithms

Fuzzy String Matching (~400 lines)

```
javascript

// Advanced string matching
class FuzzyMatcher {
  - Levenshtein distance implementation
  - Damerau-Levenshtein for transpositions
  - Jaro-Winkler for short strings
  - N-gram based matching
  - Phonetic matching (Soundex/Metaphone)
}
```

6.4 Social Algorithms

Activity Feed Ranking (~500 lines)

```javascript
// EdgeRank-inspired algorithm
function rankActivities(activities, user) {
  - Affinity score calculation
  - Time decay function
  - Content weight scoring
  - Interaction prediction
  - Diversity injection
}
```

Follow Suggestions (~400 lines)

```javascript
// User recommendation algorithm
function suggestFollows(userId) {
  - Collaborative filtering
  - Common interest detection
  - Social proof scoring
  - Geographic proximity
  - Interaction frequency analysis
}
```

---

7. Development Timeline and Complexity

Phase 1: Core Infrastructure (Weeks 1-2)

- Authentication system: 1,400 lines
- Database layer: 2,800 lines
- Basic API structure: 1,000 lines
- **Total: ~5,200 lines**

Phase 2: Image Processing (Weeks 3-4)

- Canvas editor core: 3,500 lines
- Server processing: 3,200 lines
- Storage system: 1,000 lines
- **Total: ~7,700 lines**

Phase 3: Social Features (Weeks 5-6)

- Social components: 1,700 lines
- Activity system: 1,500 lines
- Search engine: 2,500 lines
- **Total: ~5,700 lines**

Phase 4: Collaboration (Weeks 7-8)

- Real-time infrastructure: 2,700 lines
- Version control: 1,800 lines
- Collaboration UI: 1,700 lines
- **Total: ~6,200 lines**

Phase 5: Polish and Optimization (Week 9)

- Performance optimization: 1,000 lines
- Testing suite: 1,000 lines
- Bug fixes and refactoring: 500 lines
- **Total: ~2,500 lines**

Total Custom Code: ~27,300 lines

---

8. Technical Challenges and Solutions

Challenge 1: Real-time Collaboration Consistency

Solution: Implement Operational Transformation with vector clocks to ensure all users see consistent state despite network latency.

Challenge 2: Large Image Processing Performance

Solution: Use Web Workers for client-side processing and worker threads for server-side, implementing progressive rendering for responsive UI.

Challenge 3: Scalable Social Feed

Solution: Implement push-on-write for active users and pull-on-read for inactive users, with smart caching strategies.

Challenge 4: Efficient Version Storage

Solution: Delta compression algorithm storing only differences between versions, with periodic snapshot consolidation.

Challenge 5: Search Performance

Solution: Inverted index with incremental updates, query result caching, and read replicas for search operations.

---

9. Code Quality Standards

Architecture Principles

- **Separation of Concerns:** Clear boundaries between layers
- **DRY (Don't Repeat Yourself):** Reusable utility functions
- **SOLID Principles:** Especially Single Responsibility
- **Event-Driven Architecture:** For loose coupling

Code Standards

- **ESLint Configuration:** Strict ruleset for consistency
- **TypeScript:** For type safety in critical components
- **JSDoc Comments:** Comprehensive documentation
- **Unit Test Coverage:** Minimum 80% coverage
- **Integration Tests:** For all API endpoints

Performance Standards

- **Canvas Operations:** 60 FPS minimum
- **API Response Time:** <200ms for 95th percentile
- **Image Processing:** <5s for standard images
- **Search Results:** <100ms response time
- **WebSocket Latency:** <50ms for collaboration

---

10. Demonstration of Skills

This project demonstrates proficiency in:

Frontend Development

- **Advanced Canvas API usage**

- **Complex state management**
- **Real-time synchronization**
- **Performance optimization**
- **Responsive design**

Backend Development

- **RESTful API design**
- **WebSocket implementation**
- **Database optimization**
- **Async processing**
- **Security best practices**

Algorithms & Data Structures

- **Graph algorithms**
- **Image processing**
- **Search algorithms**
- **Compression techniques**
- **Distributed systems concepts**

Software Engineering

- **Design patterns**
- **Testing strategies**
- **Performance profiling**
- **Code organization**
- **Documentation**

Conclusion

ImageFlow's custom component development plan demonstrates comprehensive full-stack development skills through ~27,000 lines of original code. Every feature is implemented from scratch, showcasing deep technical knowledge and problem-solving abilities across multiple domains including image processing, real-time collaboration, social networking, and distributed systems. This approach ensures the capstone project truly reflects software development expertise rather than service integration skills.