- Market Multiplier Technical Specifications
  - Database Schema (MongoDB)
    - 1. User Progress Collection
    - 2. Product Information
    - 3. Target Audience
    - 4. Competitor Analysis
    - 5. Content
  - API Endpoints
    - Walkthrough API
    - Content API
    - AI Integration API
  - Component Structure
    - 1. Layout Components
    - 2. Form Components
    - 3. Content Components
  - State Management
    - 1. Context Structure
    - 2. Local Storage Backup
  - AI Integration
    - 1. OpenAI Configuration
    - 2. Content Generation
  - Error Handling
    - 1. Error Types
    - 2. Error Boundaries
  - Loading States
    - 1. Loading Indicators

- Authentication (Future)
- Development Guidelines
  - File Naming Conventions
  - Database Operations
  - Environment Variables
  - MongoDB Best Practices
  - Component Development
  - Testing Guidelines
  - Documentation Requirements
  - Version Control
  - Deployment
  - Performance Considerations
  - Security Guidelines
  - Error Handling
  - AI Integration
  - Maintenance

Market Multiplier Technical Specifications

Database Schema (MongoDB)

1. User Progress Collection

```
Unset
interface UserProgress {
  _id: ObjectId;
  userId: string;
  currentStep: number;
  completedSteps: number[];
  lastUpdated: Date;
  data: {
    productInfo?: ProductInfo;
    targetAudience?: TargetAudience;
    competitors?: CompetitorInfo;
    messaging?: MessagingInfo;
    contentStrategy?: ContentStrategy;
    brandVoice?: BrandVoice;
  };
}
```

2. Product Information

```
Unset
interface ProductInfo {
  _id: ObjectId;
  userId: string;
  name: string;
  type: string;
  valueProposition: string;
  keyBenefits: string[];
  createdAt: Date;
  updatedAt: Date;
}
```

3. Target Audience

```
Unset
interface TargetAudience {
  _id: ObjectId;
  userId: string;
  personas: Array<{
    name: string;
    description: string;
    problems: string[];
  }>;
  createdAt: Date;
  updatedAt: Date;
}
```

4. Competitor Analysis

```
Unset
interface CompetitorInfo {
  _id: ObjectId;
  userId: string;
  competitors: Array<{
    name: string;
    website?: string;
    keyMessages: string[];
```

```
    differentiators: string[];
  }>;
  createdAt: Date;
  updatedAt: Date;
}
```

5. Content

```
Unset
interface Content {
  _id: ObjectId;
  userId: string;
  type: ContentType;
  title: string;
  content: string;
  originalFormat?: string;
  currentFormat?: string;
  metadata: {
    keywords: string[];
    tone: string;
    style: string;
  };
  versions: Array<{
    content: string;
    timestamp: Date;
    changes: string;
  }>;
  createdAt: Date;
  updatedAt: Date;
}
```

API Endpoints

Walkthrough API

```
Unset
// Progress Management
```

```
POST /api/walkthrough/progress
GET /api/walkthrough/progress/:userId
PUT /api/walkthrough/progress/:userId

// Step-specific Data
POST /api/walkthrough/product-info
GET /api/walkthrough/product-info/:userId
PUT /api/walkthrough/product-info/:userId

// Similar endpoints for each walkthrough step
```

Content API

Unset

```
// Content Management
POST /api/content
GET /api/content/:id
PUT /api/content/:id
DELETE /api/content/:id

// Content Operations
POST /api/content/:id/repurpose
POST /api/content/:id/enhance
GET /api/content/:id/versions
```

AI Integration API

Unset

```
// AI Operations
POST /api/ai/generate-content
POST /api/ai/enhance-content
POST /api/ai/analyze-competitors
POST /api/ai/suggest-keywords
```

Component Structure

1. Layout Components

```
Unset
interface LayoutProps {
  children: React.ReactNode;
  showNavigation?: boolean;
  showProgress?: boolean;
}

interface NavigationProps {
  currentStep: number;
  totalSteps: number;
  onNext: () => void;
  onPrev: () => void;
  onSave: () => void;
}
```

2. Form Components

```
Unset
interface FormStepProps {
  onSubmit: (data: any) => void;
  onBack: () => void;
  initialData?: any;
  isLoading?: boolean;
}

interface ValidationSchema {
  // Yup validation schema for each form
}
```

3. Content Components

```
Unset
interface ContentCreatorProps {
  initialContent?: string;
  contentType: ContentType;
  styleGuide: StyleGuide;
  onSave: (content: string) => void;
}

interface ContentRepurposerProps {
  content: string;
  targetFormat: ContentType;
  styleGuide: StyleGuide;
  onConvert: (newContent: string) => void;
}
```

State Management

1. Context Structure

```
Unset
interface WalkthroughContext {
  currentStep: number;
  progress: UserProgress;
  updateProgress: (data: Partial<UserProgress>) => void;
  saveProgress: () => Promise<void>;
}

interface ContentContext {
  contents: Content[];
  addContent: (content: Content) => void;
  updateContent: (id: string, content: Partial<Content>) => void;
  deleteContent: (id: string) => void;
}
```

2. Local Storage Backup

```
Unset
interface LocalStorageSchema {
  currentProgress: UserProgress;
  lastSaved: Date;
  contentDrafts: Content[];
}
```

AI Integration

1. OpenAI Configuration

```
Unset
interface AIConfig {
  model: string;
  temperature: number;
  maxTokens: number;
  frequencyPenalty: number;
  presencePenalty: number;
}

interface PromptTemplate {
  template: string;
  variables: string[];
  generatePrompt: (data: any) => string;
}
```

2. Content Generation

```
Unset
interface ContentGeneration {
  type: ContentType;
  parameters: {
    tone: string;
    style: string;
    length: string;
```

```
    keywords: string[];
  };
  template: PromptTemplate;
}
```

Error Handling

1. Error Types

```
Unset
enum ErrorType {
  ValidationError = 'VALIDATION_ERROR',
  NetworkError = 'NETWORK_ERROR',
  AIError = 'AI_ERROR',
  DatabaseError = 'DATABASE_ERROR',
}

interface ErrorResponse {
  type: ErrorType;
  message: string;
  details?: any;
}
```

2. Error Boundaries

```
Unset
interface ErrorBoundaryProps {
  fallback: React.ReactNode;
  onError?: (error: Error) => void;
}
```

Loading States

1. Loading Indicators

```
Unset
interface LoadingState {
  isLoading: boolean;
  progress?: number;
  message?: string;
}

interface LoadingIndicatorProps {
  state: LoadingState;
  type: 'spinner' | 'progress' | 'skeleton';
}
```

Authentication (Future)

```
Unset
interface AuthConfig {
  providers: string[];
  callbacks: {
    onSuccess: (user: User) => void;
    onError: (error: Error) => void;
  };
}
```

Development Guidelines

File Naming Conventions

1. TypeScript Files:
    - Use camelCase for utils and services (`writingStyle.ts`)
    - Use kebab-case for pages (`test-writing-style.tsx`)
    - Use PascalCase for React components (`TestWritingStyle.tsx`)

Database Operations

1. MongoDB Client Usage:
    - Always use provided MongoDB client from `lib/mongodb.ts`
    - Include comprehensive error handling
    - Use TypeScript interfaces for data validation
    - Avoid direct database manipulation in components

Environment Variables

1. Management:

- Never commit `.env.local` to git
- Maintain `.env.example` for documentation
- Validate all environment variables before use

2. Required Variables:

```
Unset
MONGODB_URI=mongodb+srv://<username>:<password>@<cluster>.mongodb.net/?retryWrites=true&w=majority
```

MongoDB Best Practices

1. Connection:

    - Avoid special characters in passwords (particularly @, /, :)
    - Use connection pooling
    - Implement proper error handling
    - Cache database connections

2. Operations:

    - Use TypeScript interfaces for data validation
    - Implement proper error handling
    - Add logging for database operations
    - Use transactions where necessary

Component Development

1. Structure:

    - One component per file
    - Clear component interfaces
    - Proper prop typing
    - Error boundary implementation

2. Styling:

    - Use Tailwind utility classes
    - Avoid inline styles
    - Maintain consistent spacing
    - Follow responsive design principles

Testing Guidelines

1. Component Testing:

    - Test user interactions
    - Verify state changes
    - Check error handling
    - Validate loading states

2. API Testing:

    - Verify endpoint responses

- Test error conditions
- Validate data persistence
- Check authentication/authorization

Documentation Requirements

1. Code Documentation:

    - Clear function documentation
    - Interface/type documentation
    - Complex logic explanation
    - Example usage where needed

2. API Documentation:

    - Endpoint descriptions
    - Request/response formats
    - Error responses
    - Authentication requirements

Version Control

1. Git Guidelines:
    - Clear commit messages
    - Feature branch workflow
    - Regular commits
    - Pull request process

Deployment

1. Process:
    - Environment variable verification
    - Database migration checks
    - Build process validation
    - Deployment verification

Performance Considerations

1. Optimization:
    - Implement lazy loading
    - Use proper caching
    - Optimize database queries
    - Monitor API response times

Security Guidelines

1. Implementation:
    - Validate all inputs
    - Sanitize database queries
    - Secure API endpoints
    - Implement rate limiting

Error Handling

1. Guidelines:

- Use custom error types
- Implement error boundaries
- Log errors appropriately
- Provide user-friendly messages

AI Integration

1. Best Practices:
    - Implement retry logic
    - Cache common responses
    - Handle rate limits
    - Validate AI outputs

Maintenance

1. Regular Tasks:
    - Update dependencies
    - Review error logs
    - Monitor performance
    - Update documentation

Would you like me to:

1. Add more detail to any section?
2. Create example implementations?
3. Integrate this with our other documentation?
4. Add any missing guidelines?