Frontend-Backend Connectivity and Logic

1. System Architecture Overview

1.1 Architecture Pattern

The system follows a modern microservices architecture with clear separation of concerns:

1.1.1 Frontend Layer

- **Next.js Application**
- Server-side rendering for initial page load performance
- Client-side navigation for subsequent interactions
- API routes for backend proxy and authentication

- Static site generation for content-heavy pages

- **State Management**

- React Context API for local component state
- SWR for data fetching, caching, and revalidation
- Redux for global state (optional, based on complexity)
- Local storage for persistent user preferences

1.1.2 API Gateway Layer

- **Request Routing**
- Authentication and authorization checks
- Rate limiting and quota enforcement
- Request validation and sanitization
- Service discovery and routing

- Response caching and compression

- **Cross-Cutting Concerns**

- Logging and monitoring
- Error handling and standardization

- CORS configuration
- API versioning
- Documentation generation

1.1.3 Microservices Layer

- **Core Services**
- Listing Service
- User Service
- Order Service
- Wallet Service
- Metrics Service
- Content Service

- Reporting Service

- **Support Services**

- Notification Service
- Search Service
- File Storage Service
- Authentication Service
- Analytics Service

1.1.4 Data Layer

- **Primary Database**
- PostgreSQL for transactional data
- Schema management and migration system
- Connection pooling and optimization

- Replication and failover configuration

- **Specialized Storage**

- Elasticsearch for search functionality
- Redis for caching and real-time features
- MongoDB for unstructured content data
- S3-compatible storage for files and media

1.2 Communication Patterns

1.2.1 Synchronous Communication

- **REST API**
- Resource-based endpoint design
- HTTP method semantics (GET, POST, PUT, DELETE)
- Status code standardization
- Hypermedia controls for discoverability
- Pagination, filtering, and sorting standards
- **GraphQL API**
- Schema-first design approach
- Query complexity management
- Resolver optimization
- Batching and dataloader implementation
- Subscription support for real-time updates

1.2.2 Asynchronous Communication

- **Message Queue System**
- RabbitMQ or Apache Kafka implementation
- Topic and queue design
- Dead letter handling
- Message persistence configuration
- Consumer group management
- **Event Sourcing**
- Event store implementation
- Event schema versioning
- Replay capability
- Snapshot strategy
- Projection management

1.2.3 Real-time Communication

- **WebSocket Protocol**
- Connection management and authentication
- Channel and room organization
- Message format standardization

- Heartbeat and reconnection strategy
- Scalability considerations
- **Server-Sent Events**
- Event stream configuration
- Client reconnection handling
- Event filtering and authorization
- Backpressure management
- Monitoring and debugging

2. Authentication and Authorization Flow

2.1 Authentication System

2.1.1 User Authentication Flow

1. **Login Process**
2. User submits credentials to `/api/auth/login` endpoint
3. Backend validates credentials against User Service
4. On success, JWT token is generated with appropriate claims
5. Token is returned to client with refresh token
6. Client stores tokens securely (HTTP-only cookies or local storage)
7. **Token Validation**
8. Each API request includes JWT in Authorization header
9. API Gateway validates token signature and expiration
10. Claims are extracted and added to request context
11. Request proceeds to appropriate service if valid
12. 401 Unauthorized response if token is invalid
13. **Token Refresh**
14. Client detects expired token via 401 response or proactive check
15. Refresh token is sent to `/api/auth/refresh` endpoint
16. Backend validates refresh token and issues new JWT
17. Client updates stored tokens
18. Session is maintained without requiring re-login

19. **Logout Process**

20. Client sends request to `/api/auth/logout` endpoint
21. Backend invalidates refresh token in database
22. Client clears stored tokens
23. User is redirected to login page

2.1.2 Multi-factor Authentication

1. **MFA Enrollment**
2. User initiates MFA setup in account settings
3. Backend generates secret and QR code
4. User scans code with authenticator app
5. User verifies setup with first code

6. MFA flag is enabled in user profile

7. **MFA Verification**

8. After password validation, MFA challenge is issued
9. User submits verification code
10. Backend validates code against stored secret
11. On success, authentication flow continues
12. Failed attempts are logged and limited

2.2 Authorization Framework

2.2.1 Role-Based Access Control

1. **Role Assignment**
2. Admin assigns roles to users through User Management
3. Roles are stored in user profile
4. Role changes are logged in audit trail

5. Role information is included in JWT claims

6. **Permission Checking**

7. Each API endpoint defines required permissions
8. API Gateway checks user roles against required permissions
9. Middleware enforces access control at service level
10. Frontend components conditionally render based on permissions
11. Unauthorized actions return 403 Forbidden response

2.2.2 Resource-Level Permissions

1. **Resource Ownership**
2. Resources store owner or creator information
3. Ownership checks are performed for sensitive operations
4. Delegation and sharing capabilities are supported
5. Hierarchical access is enforced (e.g., project access grants access to contained items)
6. **Permission Inheritance**
7. Permissions cascade through organizational hierarchy
8. Child resources inherit parent permissions by default
9. Override rules can be configured for exceptions
10. Effective permissions are calculated and cached

3. Data Flow Patterns

3.1 Listing Management Flow

3.1.1 Listing Creation Flow

1. **Frontend Initiation**
2. Admin completes listing form in admin panel
3. Client-side validation ensures data integrity
4. Form data is serialized and sent to `/api/listings` endpoint
5. Loading state is displayed during submission
6. **Backend Processing**
7. API Gateway authenticates request and validates permissions
8. Request is routed to Listing Service
9. Data validation and business rule enforcement
10. Domain information is verified for authenticity
11. Initial SEO metrics are requested from Metrics Service
12. **Metrics Enrichment**
13. Metrics Service queries configured API providers
14. Data is normalized and validated
15. Metrics are stored in database with timestamp

16. Results are returned to Listing Service

17. Quality score is calculated based on metrics

18. **Persistence and Indexing**

19. Listing is saved to primary database
20. Event is published for search indexing
21. Search Service indexes listing for discovery
22. Notification is triggered for review if required

23. Response is returned to client with listing ID

24. **Frontend Completion**

25. Client receives successful response
26. UI updates to show success message
27. Listing appears in admin listing table
28. Metrics visualization is displayed if available
29. Admin can proceed to additional configuration

3.1.2 Listing Update Flow

1. **Change Detection**
2. Admin modifies listing in edit form
3. Changed fields are tracked for audit purposes
4. Validation ensures data integrity

5. Update request is sent to `/api/listings/{id}` endpoint

6. **Validation and Processing**

7. Backend validates changes against business rules
8. Previous version is archived for history
9. Changes are applied to current version
10. If metrics-related fields change, re-evaluation is triggered

11. Search index is updated with new information

12. **Notification and Logging**

13. Change event is published to message queue
14. Interested services consume the event
15. Audit log entry is created
16. Notifications are sent to relevant users if needed

17. Analytics event is recorded for reporting

3.2 SEO Metrics Update Flow

3.2.1 Scheduled Update Process

1. **Scheduler Initiation**
2. Cron job triggers update process at configured intervals
3. Listings are prioritized based on tier and last update time
4. Batch size is determined based on API quotas

5. Update job is submitted to queue

6. **Metrics Service Processing**

7. Worker picks up update job from queue
8. Domains are grouped by API provider for efficiency
9. API requests are made with appropriate rate limiting
10. Responses are parsed and normalized

11. Error handling manages failed requests

12. **Data Processing and Storage**

13. New metrics are validated against historical data
14. Anomaly detection identifies suspicious changes
15. Valid metrics are stored with timestamp
16. Historical data is maintained for trend analysis

17. Derived metrics are calculated and stored

18. **Notification and Indexing**

19. Significant changes trigger notifications
20. Search index is updated with new metrics
21. Listing quality scores are recalculated
22. Dashboard widgets are refreshed via WebSocket
23. Update status is logged for monitoring

3.2.2 Manual Update Process

1. **Admin Initiation**
2. Admin requests update for specific listing
3. Request is sent to `/api/metrics/refresh/{domain}` endpoint

4. Frontend shows loading indicator

5. **Priority Processing**

6. Request is flagged as high priority
7. Metrics Service processes request immediately
8. API calls are made to all configured providers
9. Results are processed in real-time

10. Response includes updated metrics

11. **Frontend Update**

12. Client receives updated metrics
13. UI refreshes to show new values
14. Historical charts are updated
15. Change indicators highlight differences
16. Status message confirms completion

3.3 Order Processing Flow

3.3.1 Order Creation Flow

1. **User Submission**
2. User selects listing and submits order form
3. Order details are validated client-side
4. Request is sent to `/api/orders` endpoint

5. Wallet balance is checked for sufficiency

6. **Order Service Processing**

7. Order data is validated against business rules
8. Listing availability is confirmed
9. Price is calculated with any applicable discounts
10. Wallet Service is called to reserve funds

11. Order record is created with "pending" status

12. **Publisher Notification**

13. Event is published for new order
14. Notification Service sends alert to publisher
15. Publisher dashboard is updated via WebSocket
16. Acceptance deadline is set based on SLA

17. Order appears in publisher's queue

18. **Status Tracking**

19. Order status is updated as it progresses
20. Each status change publishes an event
21. Timeline is maintained for tracking
22. SLA monitoring tracks time in each status
23. Alerts are triggered for approaching deadlines

3.3.2 Order Fulfillment Flow

1. **Content Submission**
2. Publisher submits content for order
3. Content is validated against requirements
4. Files are uploaded to storage service
5. Content is associated with order record

6. Status changes to "review_pending"

7. **Review Process**

8. Customer is notified of content submission
9. Content is displayed in review interface
10. Customer can approve or request revisions
11. Comments are tracked with the order

12. Status updates based on customer action

13. **Link Verification**

14. After content approval, link placement is verified
15. Automated checks confirm link existence and attributes
16. Screenshot is captured for record
17. Link details are stored with order

18. Status changes to "completed" upon verification

19. **Financial Settlement**

20. Completion triggers financial transaction
21. Reserved funds are transferred from escrow
22. Publisher payment is scheduled according to terms
23. Invoice is generated for customer
24. Transaction records are created in Wallet Service

3.4 Wallet and Payment Flow

3.4.1 Wallet Funding Flow

1. **Payment Initiation**
2. User selects amount and payment method
3. Request is sent to `/api/wallet/topup` endpoint
4. Payment gateway integration is selected based on method
5. **Payment Processing**
6. User is redirected to payment gateway or form
7. Payment details are submitted securely
8. Gateway processes payment and returns result
9. Webhook receives confirmation from gateway
10. Wallet Service verifies payment authenticity
11. **Balance Update**
12. On successful payment, wallet balance is increased
13. Transaction record is created with reference
14. User is notified of successful funding
15. Wallet history is updated in real-time
16. Analytics event is recorded for reporting

3.4.2 Publisher Payout Flow

1. **Payout Eligibility**
2. System identifies publishers eligible for payment
3. Minimum threshold and holding period are checked
4. Payment amount is calculated for each publisher
5. Batch is prepared for processing
6. **Approval Workflow**
7. Finance admin reviews pending payouts
8. Batch or individual approval is performed
9. Payment records are created with "pending" status
10. Payout job is submitted to queue
11. **Payment Execution**

12. Payment Service processes payout job
13. Appropriate payment method is selected per publisher
14. API calls are made to payment providers
15. Responses are processed and recorded
16. Status is updated based on provider response
17. **Reconciliation and Reporting**
18. Successful payments update publisher balance
19. Failed payments are flagged for review
20. Transaction records are updated with provider references
21. Financial reports are updated with new data
22. Tax information is recorded for compliance

3.5 Content Management Flow

3.5.1 Blog Post Creation Flow

1. **Content Authoring**
2. Admin creates post in rich text editor
3. Images are uploaded to storage service
4. Draft is automatically saved periodically
5. SEO recommendations guide optimization
6. Preview shows how post will appear
7. **Review and Approval**
8. Draft is submitted for review
9. Assigned reviewers receive notification
10. Comments and suggestions are tracked
11. Revisions are made based on feedback
12. Final approval changes status to "approved"
13. **Publishing Process**
14. Approved post is scheduled or published immediately
15. Content is stored in database
16. Search index is updated
17. Static pages are regenerated if needed
18. Social media notifications are triggered if configured

19. **Analytics Integration**

20. Published post is tracked for performance
21. View counts and engagement are recorded
22. Conversion events are associated with content
23. Performance dashboard is updated
24. A/B test variants are monitored if active

3.5.2 Page Management Flow

1. **Page Creation**
2. Admin uses page builder interface
3. Components are added and configured
4. Content is entered for each section
5. SEO settings are configured

6. Page is saved as draft

7. **Page Structure**

8. Position in site hierarchy is defined
9. Navigation menus are updated
10. URL structure is configured
11. Related pages are linked

12. Breadcrumb structure is updated

13. **Publishing and Deployment**

14. Page is reviewed and approved
15. Publishing triggers static generation
16. CDN cache is invalidated
17. Sitemap is updated
18. Redirect rules are applied if needed

4. Real-time Update Patterns

4.1 WebSocket Communication

4.1.1 Connection Establishment

1. **Client Connection**
2. Frontend initiates WebSocket connection after authentication
3. Authentication token is included in connection request

4. Connection is established to `/ws` endpoint
5. Server validates token and authorizes connection

6. Client receives connection acknowledgment

7. **Channel Subscription**

8. Client subscribes to relevant channels based on role and context
9. User-specific channel: `user/{userId}`
10. Role-based channels: `role/{roleId}`
11. Entity-specific channels: `entity/{type}/{id}`
12. Subscription confirmations are sent to client

4.1.2 Event Broadcasting

1. **Server-side Events**
2. Backend services publish events to message broker
3. WebSocket service subscribes to relevant topics
4. Events are filtered based on channel subscriptions
5. Messages are formatted according to protocol

6. Events are pushed to appropriate clients

7. **Client-side Handling**

8. Client receives event on subscribed channel
9. Event type determines handling logic
10. UI is updated based on event data
11. Local cache is invalidated or updated
12. Notifications are displayed if relevant

4.2 Real-time Dashboard Updates

4.2.1 Metrics Widget Updates

1. **Data Change Detection**
2. Backend services detect relevant data changes
3. Change events are published to message broker
4. Events include entity ID and changed metrics
5. Aggregation service processes events for dashboards

6. Updated aggregates are prepared for broadcast

7. **Push Notification**

8. WebSocket service pushes updates to subscribed clients
9. Client receives update on dashboard channel
10. Widget components re-render with new data
11. Animations highlight changed values
12. Historical charts incorporate new data points

4.2.2 Activity Feed Updates

1. **Activity Recording**
2. User actions generate activity events
3. Events are stored in activity database
4. Events are categorized and prioritized
5. Relevance is determined for different users
6. **Feed Distribution**
7. New activities are pushed to relevant users
8. Client displays new activities in feed
9. Unread indicators show new items
10. Infinite scroll loads historical activities
11. Filters allow customization of feed content

5. Search and Filter System Logic

5.1 Search Implementation

5.1.1 Indexing Process

1. **Document Preparation**
2. Entity changes trigger indexing events
3. Search Service consumes events from queue
4. Documents are normalized and enriched
5. Field mappings are applied based on entity type
6. Documents are submitted to Elasticsearch
7. **Search Optimization**
8. Analyzers are configured for different languages
9. Synonyms and custom dictionaries are applied
10. Boosting rules prioritize important fields
11. Stemming and tokenization improve matching

12. Index settings are optimized for performance

5.1.2 Query Processing

1. **Query Construction**
2. User input is sanitized and tokenized
3. Query is expanded with synonyms
4. Fuzzy matching is applied for typo tolerance
5. Field-specific boosts are applied
6. Filters are added based on user context
7. **Result Processing**
8. Raw search results are retrieved from Elasticsearch
9. Results are augmented with additional data
10. Permissions are checked for each result
11. Highlighting is applied to matching terms
12. Results are formatted for presentation

5.2 Filter System

5.2.1 Filter Definition

1. **Filter Configuration**
2. Admin defines available filters in system settings
3. Filter types include range, select, multi-select, toggle
4. Dependencies between filters are configured
5. Default values and sorting are defined
6. Display options are customized
7. **Dynamic Filter Generation**
8. Available filters are determined by user role
9. Filter options are generated from current data
10. Option counts show result distribution
11. Unavailable options are disabled or hidden
12. Recently used filters are prioritized

5.2.2 Filter Application

1. **Client-side Processing**
2. User selects filter values in UI

3. URL parameters are updated to reflect selection
4. Filter state is stored in application state
5. Query is constructed with selected filters
6. Request is sent to appropriate API endpoint
7. **Server-side Processing**
8. Filter parameters are extracted from request
9. Parameters are validated against allowed values
10. Database query is constructed with filter conditions
11. Results are filtered for permissions
12. Pagination is applied to filtered results

6. Error Handling and Recovery

6.1 Frontend Error Management

6.1.1 API Error Handling

1. **Request Error Detection**
2. Axios interceptors or fetch wrappers catch errors
3. HTTP status codes determine error type
4. Error responses are parsed for details
5. Generic errors are transformed to user-friendly messages
6. Specific error codes trigger special handling
7. **User Feedback**
8. Error messages are displayed in context
9. Toast notifications for transient errors
10. Modal dialogs for blocking errors
11. Form field errors highlight specific inputs
12. Retry options are provided when appropriate

6.1.2 UI Error Boundary

1. **Component Error Isolation**
2. React Error Boundaries catch rendering errors
3. Failed components are replaced with fallbacks
4. Error details are logged to monitoring service
5. User is provided with recovery options

6. Application state remains usable

7. **Recovery Mechanisms**

8. Refresh button reloads problematic data
9. Alternative views are offered when possible
10. Graceful degradation preserves core functionality
11. Local storage backup recovers unsaved work
12. Support contact is provided for unresolvable issues

6.2 Backend Error Management

6.2.1 Exception Handling

1. **Structured Error Response**
2. Exceptions are caught at service boundaries
3. Error codes are standardized across services
4. Detailed messages for developers
5. User-friendly messages for clients

6. Context information for troubleshooting

7. **Error Logging**

8. Errors are logged with context and stack traces
9. Log levels reflect error severity
10. Correlation IDs track errors across services
11. Aggregation identifies patterns and trends
12. Alerts trigger for critical errors

6.2.2 Resilience Patterns

1. **Circuit Breaker**
2. Failed external service calls are monitored
3. Breaker opens after threshold of failures
4. Fallback mechanisms provide degraded service
5. Breaker attempts reset after cooling period

6. Status is exposed for monitoring

7. **Retry Mechanism**

8. Transient errors trigger automatic retries
9. Exponential backoff prevents flooding
10. Retry limits prevent infinite loops

11. Successful retries are logged for monitoring
12. Permanent failures are handled gracefully

7. Caching Strategy

7.1 Multi-level Caching

7.1.1 Browser Caching

1. **Asset Caching**
2. Static assets use long-term cache headers
3. Versioned file names enable cache busting
4. Service worker caches critical resources
5. Cache-Control headers optimize browser behavior
6. Preloading improves initial load performance
7. **Data Caching**
8. SWR library manages client-side cache
9. Stale-while-revalidate pattern keeps UI responsive
10. Local storage persists non-sensitive data
11. Cache invalidation on relevant mutations
12. TTL settings prevent stale data

7.1.2 API Gateway Caching

1. **Response Caching**
2. GET requests are cached by URL and parameters
3. Vary headers respect authentication context
4. Cache-Control headers define TTL
5. Purge endpoints clear related caches
6. Cache statistics monitor hit rates
7. **Shared Cache**
8. Redis stores cached responses
9. Distributed cache enables scaling
10. Cache keys include version identifiers
11. Compressed storage optimizes memory usage
12. Background refresh prevents cache stampede

7.2 Database Caching

7.2.1 Query Result Caching

1. **ORM-level Cache**
2. Common queries are cached automatically
3. Entity-level cache stores individual records
4. Collection cache stores query results
5. Write-through updates maintain consistency
6. Cache regions isolate different entity types
7. **Computed Data Cache**
8. Expensive calculations are cached
9. Aggregate values use materialized views
10. Scheduled jobs refresh cached data
11. Invalidation triggers on relevant changes
12. Version tags track data freshness

8. Mobile-First Approach

8.1 Responsive Implementation

8.1.1 Fluid Grid System

1. **Layout Structure**
2. Container widths adapt to viewport
3. Column counts reduce on smaller screens
4. Breakpoints define layout transitions
5. Percentage-based widths enable flexibility
6. CSS Grid and Flexbox provide advanced layouts
7. **Component Adaptation**
8. Tables transform to cards on mobile
9. Multi-column forms stack vertically
10. Navigation collapses to hamburger menu
11. Touch targets increase in size
12. Font sizes adjust for readability

8.1.2 Media Optimization

1. **Responsive Images**
2. srcset provides resolution options
3. sizes attribute guides browser selection
4. Art direction uses picture element
5. Lazy loading defers offscreen images
6. WebP format with fallbacks for compatibility
7. **Video Optimization**
8. Adaptive bitrate streaming
9. Thumbnail placeholders before play
10. Reduced autoplay on mobile
11. Orientation handling for fullscreen
12. Bandwidth detection adjusts quality

8.2 Touch Interaction

8.2.1 Gesture Support

1. **Touch Events**
2. Touch feedback for interactive elements
3. Swipe gestures for common actions
4. Pull-to-refresh for content updates
5. Pinch-to-zoom for detailed views
6. Long-press for contextual menus
7. **Input Enhancement**
8. Mobile-optimized form controls
9. Appropriate keyboard types for fields
10. Autocomplete integration
11. Simplified validation feedback
12. Progressive disclosure of complex forms

9. Integration Patterns

9.1 Third-party API Integration

9.1.1 SEO Data Providers

1. **Ahrefs Integration**
2. API client with rate limiting
3. Credential management
4. Endpoint mapping for DR and backlinks
5. Response parsing and normalization
6. Error handling and fallback
7. **Moz Integration**
8. Authentication flow
9. DA retrieval endpoint
10. Batch processing optimization
11. Data transformation
12. Cache strategy
13. **SEMrush Integration**
14. API key rotation for quota management
15. Traffic and keyword data retrieval
16. Competitor analysis endpoints
17. Response validation
18. Historical data storage
19. **Majestic Integration**
20. Authentication mechanism
21. Trust Flow and Citation Flow retrieval
22. Backlink profile analysis
23. Data normalization
24. Quota monitoring

9.1.2 Payment Gateways

1. **Stripe Integration**
2. Secure API key storage

3. Payment intent creation
4. Webhook handling for events
5. Subscription management
6. Refund processing
7. **PayPal Integration**
8. OAuth authentication
9. Payment creation flow
10. IPN message handling
11. Payout API for publishers
12. Transaction reconciliation

9.2 External Service Integration

9.2.1 Email Service

1. **Transactional Email**
2. Template management
3. Variable substitution
4. Attachment handling
5. Delivery tracking
6. Bounce processing
7. **Bulk Email**
8. List management
9. Campaign creation
10. Scheduling and throttling
11. Analytics integration
12. Unsubscribe handling

9.2.2 File Storage

1. **S3 Integration**
2. Credential management
3. Upload preprocessing
4. Signed URL generation
5. Folder structure management
6. Lifecycle policies
7. **CDN Integration**

8. Origin configuration
9. Cache invalidation
10. Custom domain setup
11. Security settings
12. Performance monitoring

10. Deployment and DevOps Integration

10.1 CI/CD Pipeline

10.1.1 Build Process

1. **Frontend Build**
2. Next.js build optimization
3. Asset optimization
4. Bundle analysis
5. Environment configuration
6. Version tagging
7. **Backend Build**
8. Java compilation
9. Dependency resolution
10. Test execution
11. Docker image creation
12. Version management

10.1.2 Deployment Process

1. **Staging Deployment**
2. Environment configuration
3. Database migration
4. Smoke testing
5. Performance validation
6. Manual approval gate
7. **Production Deployment**
8. Blue-green deployment
9. Canary release option
10. Automated rollback triggers

11. Cache warming
12. Health check verification

10.2 Monitoring Integration

10.2.1 Application Monitoring

1. **Performance Tracking**
2. Response time measurement
3. Throughput monitoring
4. Error rate tracking
5. Resource utilization
6. Bottleneck identification
7. **User Experience Monitoring**
8. Page load time
9. Time to interactive
10. First contentful paint
11. Cumulative layout shift
12. User journey tracking

10.2.2 Alert System

1. **Alert Configuration**
2. Threshold definition
3. Escalation paths
4. Notification channels
5. On-call rotation
6. Alert grouping and correlation
7. **Incident Management**
8. Incident creation from alerts
9. Status page integration
10. Resolution workflow
11. Post-mortem process
12. Knowledge base updates

This detailed specification of frontend-backend connectivity and logic provides a comprehensive blueprint for implementing the technical architecture of the admin

panel, ensuring all components work together seamlessly to support the required functionality.