5.3 Explore Feature

Feature Goal: Provide users with an intuitive interface for discovering and exploring philosophical concepts through interactive visualizations, personalized recommendations, and seamless integration with other app features.

5.3.1 Definitions and Glossary

5.3.1.1 Concept

A fundamental philosophical idea, principle, or theory that forms the building blocks of philosophical understanding. Concepts are interconnected through various relationship types and organized within traditions, categories, and historical eras.

5.3.1.2 Concept Map

An interactive visualization that displays concepts and their relationships in a spatial arrangement, allowing users to explore connections between philosophical ideas visually.

5.3.1.3 Concept Mastery

A measurement of a user's familiarity and understanding of a specific philosophical concept, tracked through interactions, explorations, and engagement across the app.

5.3.1.4 Exploration Path

A sequence of concepts viewed by a user during a single exploration session, tracked to analyze learning patterns and provide personalized recommendations.

5.3.1.5 Constellation View

An alternative visualization mode that organizes concepts into thematic clusters, resembling star constellations, to highlight broader philosophical movements and schools of thought.

5.3.2 System Architecture

5.3.2.1 Frontend Components

ExploreScreen.tsx: Main screen for the Explore feature

- Implements filtering by category, tradition, era, and difficulty

- Renders concept cards with visual indicators for mastery and relevance

- Integrates with recommendation engine for personalized suggestions

- Provides access to different visualization modes

ConceptMap.tsx: Interactive visualization of concept relationships

- Implements WebGL-based rendering for smooth performance

- Supports touch and mouse-based zoom and pan interactions

- Renders nodes, connections, and visual state indicators

- Handles node selection and navigation events

ConceptDetail.tsx: Detailed view of individual concepts

- Renders concept metadata, description, and examples

- Displays related concepts with relationship types

- Shows user's mastery level and progress

- Provides navigation to connected concepts

ConceptSearch.tsx: Search interface for finding specific concepts

- Implements real-time search with typeahead suggestions

- Filters results by relevance and user history

- Highlights matching terms in search results

- Supports advanced filtering options

ConceptFilter.tsx: Filtering options for concept exploration

- Provides category, tradition, era, and difficulty filters

- Implements multi-select filter capabilities

- Shows filter counts and active filter indicators

- Supports saving filter preferences

PathwaySelector.tsx: Interface for selecting learning pathways

- Displays curated concept sequences for structured learning

- Shows progression and prerequisites for each pathway

- Indicates user progress within each pathway

- Provides difficulty and time estimates

ConstellationView.tsx: Alternative visualization of concept clusters

- Implements thematic grouping of related concepts

- Supports interactive exploration of concept constellations

- Provides zooming and focusing on specific clusters

- Animates transitions between clusters

RelatedConceptsList.tsx: List of related concepts with relevance indicators

- Displays concepts related to the current selection

- Shows relationship types and strengths

- Provides quick navigation to related concepts

- Supports filtering by relationship type

ConceptHistoryTracker.tsx: Component tracking user's exploration history

- Records sequence of concepts viewed

- Provides breadcrumb navigation to previous concepts

- Shows exploration statistics and patterns

- Enables returning to previous points in exploration

NotificationBanner.tsx: Component for displaying exploration notifications

- Shows milestone achievements and recommendations

- Displays mastery level increases and badges

- Provides contextual suggestions based on current exploration

- Supports dismissal and interaction with notifications

5.3.2.2 State Management

ExploreStore: Central state management for exploration-related data

- Maintains concept list, details, and visualization state

- Handles filtering and search operations

- Manages exploration history and pathways

- Coordinates with API service for data operations

MasteryStore: Tracks and manages user concept mastery

- Maintains mastery levels for all concepts

- Tracks exposure and interaction counts

- Manages mastery progression and milestones

- Coordinates with notification system for achievements

5.3.2.3 Frontend Services

ConceptApiService: Handles all concept-related API communications

- Fetches concept lists, details, and relationships

- Submits exploration history and interaction data

- Retrieves user mastery and recommendations

- Manages error handling and retry logic

VisualizationRenderer: Manages concept visualization

- Handles layout calculations and positioning

- Manages zoom and pan interactions

- Optimizes rendering performance

- Supports different visualization modes and device capabilities

5.3.2.4 Backend Components

concept\_service.py: Core service for concept data management

- Handles concept retrieval, filtering, and search

- Manages concept relationships and hierarchies

- Processes concept metadata and attributes

- Provides concept recommendation algorithms

concept\_relationship\_service.py: Service for managing concept relationships

- Implements relationship type definitions and rules

- Manages bidirectional and hierarchical relationships

- Calculates relationship strengths and relevance

- Provides graph traversal and path finding capabilities

exploration\_history\_service.py: Service for tracking user exploration patterns

- Records concept viewing sequences and durations

- Analyzes exploration patterns and preferences

- Provides historical navigation capabilities

- Generates insights for personalization

recommendation\_service.py: Service for generating personalized recommendations

- Implements collaborative and content-based filtering

- Generates personalized concept suggestions

- Identifies knowledge gaps and learning opportunities

- Adapts recommendations based on user behavior

visualization\_service.py: Service for generating concept visualizations

- Creates optimized graph layouts for concept maps

- Generates constellation clusters and arrangements

- Calculates positioning for visual balance

- Provides different visualization modes and algorithms

notification\_service.py: Service for managing exploration notifications

- Generates milestone and achievement notifications

- Creates personalized recommendations and suggestions

- Manages notification delivery and timing

- Tracks notification interactions and effectiveness

5.3.2.5 Data Access

ConceptRepository: Data access layer for concept-related operations

- Implements CRUD operations for concept entities

- Provides optimized queries for concept filtering and search

- Manages caching for frequently accessed concept data

- Handles transaction management for concept operations

UserConceptRepository: Data access for user-concept relationships

- Tracks user mastery and interaction history

- Manages user exploration paths and preferences

- Provides analytics queries for user engagement

- Implements efficient mastery tracking algorithms

5.3.3 Database Models

5.3.3.1 concepts: Concept definitions and metadata

CREATE TABLE concepts (

concept\_id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

description TEXT NOT NULL,

short\_description VARCHAR(255),

category VARCHAR(50),

tradition VARCHAR(50),

era VARCHAR(50),

difficulty\_level INTEGER CHECK (difficulty\_level BETWEEN 1 AND 5),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

is\_premium BOOLEAN DEFAULT FALSE,

image\_url VARCHAR(255),

color\_code VARCHAR(7),

UNIQUE(name)

);

CREATE INDEX idx\_concepts\_category ON concepts(category);

CREATE INDEX idx\_concepts\_tradition ON concepts(tradition);

CREATE INDEX idx\_concepts\_era ON concepts(era);

CREATE INDEX idx\_concepts\_difficulty\_level ON concepts(difficulty\_level);

CREATE INDEX idx\_concepts\_is\_premium ON concepts(is\_premium);

5.3.3.2 concept\_relationships: Relationships between concepts

CREATE TABLE concept\_relationships (

relationship\_id SERIAL PRIMARY KEY,

source\_concept\_id INTEGER REFERENCES concepts(concept\_id) ON DELETE CASCADE,

target\_concept\_id INTEGER REFERENCES concepts(concept\_id) ON DELETE CASCADE,

relationship\_type VARCHAR(50) NOT NULL,

relationship\_strength FLOAT CHECK (relationship\_strength BETWEEN 0.0 AND 1.0),

description TEXT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

UNIQUE(source\_concept\_id, target\_concept\_id, relationship\_type)

);

CREATE INDEX idx\_concept\_relationships\_source ON concept\_relationships(source\_concept\_id);

CREATE INDEX idx\_concept\_relationships\_target ON concept\_relationships(target\_concept\_id);

CREATE INDEX idx\_concept\_relationships\_type ON concept\_relationships(relationship\_type);

CREATE INDEX idx\_concept\_relationships\_strength ON concept\_relationships(relationship\_strength);

5.3.3.3 user\_concept\_interactions: User interactions with concepts

CREATE TABLE user\_concept\_interactions (

interaction\_id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(user\_id) ON DELETE CASCADE,

concept\_id INTEGER REFERENCES concepts(concept\_id) ON DELETE CASCADE,

interaction\_type VARCHAR(50) NOT NULL,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

duration\_seconds INTEGER,

source\_feature VARCHAR(50),

context\_data JSONB,

UNIQUE(user\_id, concept\_id, interaction\_type, timestamp)

);

CREATE INDEX idx\_user\_concept\_interactions\_user ON user\_concept\_interactions(user\_id);

CREATE INDEX idx\_user\_concept\_interactions\_concept ON user\_concept\_interactions(concept\_id);

CREATE INDEX idx\_user\_concept\_interactions\_type ON user\_concept\_interactions(interaction\_type);

CREATE INDEX idx\_user\_concept\_interactions\_timestamp ON user\_concept\_interactions(timestamp);

CREATE INDEX idx\_user\_concept\_interactions\_source ON user\_concept\_interactions(source\_feature);

5.3.3.4 user\_exploration\_history: User exploration sessions and paths

CREATE TABLE user\_exploration\_history (

history\_id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(user\_id) ON DELETE CASCADE,

session\_id UUID NOT NULL,

exploration\_path JSONB NOT NULL,

start\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

end\_time TIMESTAMP,

total\_concepts\_viewed INTEGER DEFAULT 0,

total\_time\_seconds INTEGER,

entry\_point VARCHAR(50),

exit\_point VARCHAR(50)

);

CREATE INDEX idx\_user\_exploration\_history\_user ON user\_exploration\_history(user\_id);

CREATE INDEX idx\_user\_exploration\_history\_session ON user\_exploration\_history(session\_id);

CREATE INDEX idx\_user\_exploration\_history\_start ON user\_exploration\_history(start\_time);

CREATE INDEX idx\_user\_exploration\_history\_entry ON user\_exploration\_history(entry\_point);

CREATE INDEX idx\_user\_exploration\_history\_exit ON user\_exploration\_history(exit\_point);

5.3.3.5 concept\_mastery: User mastery levels for concepts

CREATE TABLE concept\_mastery (

mastery\_id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(user\_id) ON DELETE CASCADE,

concept\_id INTEGER REFERENCES concepts(concept\_id) ON DELETE CASCADE,

mastery\_level FLOAT CHECK (mastery\_level BETWEEN 0.0 AND 1.0),

exposure\_count INTEGER DEFAULT 0,

last\_interaction TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

notes TEXT,

UNIQUE(user\_id, concept\_id)

);

CREATE INDEX idx\_concept\_mastery\_user ON concept\_mastery(user\_id);

CREATE INDEX idx\_concept\_mastery\_concept ON concept\_mastery(concept\_id);

CREATE INDEX idx\_concept\_mastery\_level ON concept\_mastery(mastery\_level);

CREATE INDEX idx\_concept\_mastery\_exposure ON concept\_mastery(exposure\_count);

CREATE INDEX idx\_concept\_mastery\_last ON concept\_mastery(last\_interaction);

5.3.3.6 exploration\_notifications: Notifications related to exploration

CREATE TABLE exploration\_notifications (

notification\_id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(user\_id) ON DELETE CASCADE,

notification\_type VARCHAR(50) NOT NULL,

content TEXT NOT NULL,

related\_concept\_id INTEGER REFERENCES concepts(concept\_id) ON DELETE SET NULL,

is\_read BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

expiry\_at TIMESTAMP

);

CREATE INDEX idx\_exploration\_notifications\_user ON exploration\_notifications(user\_id);

CREATE INDEX idx\_exploration\_notifications\_type ON exploration\_notifications(notification\_type);

CREATE INDEX idx\_exploration\_notifications\_read ON exploration\_notifications(is\_read);

CREATE INDEX idx\_exploration\_notifications\_created ON exploration\_notifications(created\_at);

CREATE INDEX idx\_exploration\_notifications\_concept ON exploration\_notifications(related\_concept\_id);

5.3.3.7 concept\_learning\_pathways: Curated learning paths

CREATE TABLE concept\_learning\_pathways (

pathway\_id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

description TEXT NOT NULL,

difficulty\_level INTEGER CHECK (difficulty\_level BETWEEN 1 AND 5),

estimated\_duration\_minutes INTEGER,

concepts\_sequence JSONB NOT NULL,

prerequisites JSONB,

is\_premium BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE INDEX idx\_concept\_learning\_pathways\_difficulty ON concept\_learning\_pathways(difficulty\_level);

CREATE INDEX idx\_concept\_learning\_pathways\_premium ON concept\_learning\_pathways(is\_premium);

CREATE INDEX idx\_concept\_learning\_pathways\_duration ON concept\_learning\_pathways(estimated\_duration\_minutes);

5.3.3.8 user\_pathway\_progress: User progress on learning pathways

CREATE TABLE user\_pathway\_progress (

progress\_id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(user\_id) ON DELETE CASCADE,

pathway\_id INTEGER REFERENCES concept\_learning\_pathways(pathway\_id) ON DELETE CASCADE,

current\_position INTEGER DEFAULT 0,

completed\_concepts JSONB DEFAULT '[]',

started\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

completed\_at TIMESTAMP,

UNIQUE(user\_id, pathway\_id)

);

CREATE INDEX idx\_user\_pathway\_progress\_user ON user\_pathway\_progress(user\_id);

CREATE INDEX idx\_user\_pathway\_progress\_pathway ON user\_pathway\_progress(pathway\_id);

CREATE INDEX idx\_user\_pathway\_progress\_completed ON user\_pathway\_progress(completed\_at);

5.3.4 API Endpoints

5.3.4.1 GET /api/v1/concepts

Description: Retrieve a paginated list of concepts with optional filtering

Authentication: Required

Parameters:

- page: Pagination page number (default: 1)

- limit: Items per page (default: 20, max: 50)

- category: Filter by category

- tradition: Filter by philosophical tradition

- era: Filter by historical era

- difficulty: Filter by difficulty level (1-5)

- search: Search term for concept name or description

- sort: Sort field (name, tradition, era, difficulty, popularity)

- order: Sort order (asc, desc)

Response:

```json

{

"total": 309,

"page": 1,

"pages": 16,

"limit": 20,

"concepts": [

{

"concept\_id": 1,

"name": "Categorical Imperative",

"short\_description": "Kant's ethical principle for determining moral actions",

"category": "Ethics",

"tradition": "German Idealism",

"era": "Enlightenment",

"difficulty\_level": 4,

"image\_url": "/assets/concepts/categorical\_imperative.jpg",

"color\_code": "#3A86FF",

"mastery\_level": 0.65

},

{

"concept\_id": 2,

"name": "Eudaimonia",

"short\_description": "Aristotle's concept of human flourishing and well-being",

"category": "Ethics",

"tradition": "Ancient Greek",

"era": "Classical",

"difficulty\_level": 3,

"image\_url": "/assets/concepts/eudaimonia.jpg",

"color\_code": "#4CC9F0",

"mastery\_level": 0.42

}

// Additional concepts...

]

}

```

5.3.4.2 GET /api/v1/concepts/{concept\_id}

Description: Retrieve detailed information about a specific concept

Authentication: Required

Parameters: None

Response:

```json

{

"concept\_id": 1,

"name": "Categorical Imperative",

"description": "The categorical imperative is the central philosophical concept in the deontological moral philosophy of Immanuel Kant. Introduced in Kant's Groundwork of the Metaphysics of Morals, it is a way of evaluating motivations for action. It commands that actions must be universalizable to be moral.",

"short\_description": "Kant's ethical principle for determining moral actions",

"category": "Ethics",

"tradition": "German Idealism",

"era": "Enlightenment",

"difficulty\_level": 4,

"image\_url": "/assets/concepts/categorical\_imperative.jpg",

"color\_code": "#3A86FF",

"related\_concepts": [

{

"concept\_id": 5,

"name": "Deontological Ethics",

"relationship\_type": "parent\_concept",

"relationship\_strength": 0.95,

"description": "The Categorical Imperative is a foundational principle within Deontological Ethics"

},

{

"concept\_id": 12,

"name": "Universalizability",

"relationship\_type": "component",

"relationship\_strength": 0.90,

"description": "Universalizability is a key test within the Categorical Imperative framework"

}

// Additional related concepts...

],

"mastery": {

"level": 0.65,

"exposure\_count": 12,

"last\_interaction": "2025-06-01T14:32:10Z"

},

"is\_premium": false

}

```

5.3.4.3 GET /api/v1/concepts/{concept\_id}/relationships

Description: Retrieve relationships for a specific concept

Authentication: Required

Parameters:

- relationship\_type: Filter by relationship type

- min\_strength: Minimum relationship strength (0.0-1.0)

- limit: Maximum number of relationships to return

- direction: Relationship direction (incoming, outgoing, both)

Response:

```json

{

"concept\_id": 1,

"name": "Categorical Imperative",

"relationships": [

{

"relationship\_id": 42,

"target\_concept\_id": 5,

"target\_name": "Deontological Ethics",

"relationship\_type": "parent\_concept",

"relationship\_strength": 0.95,

"description": "The Categorical Imperative is a foundational principle within Deontological Ethics",

"direction": "outgoing"

},

{

"relationship\_id": 43,

"target\_concept\_id": 12,

"target\_name": "Universalizability",

"relationship\_type": "component",

"relationship\_strength": 0.90,

"description": "Universalizability is a key test within the Categorical Imperative framework",

"direction": "incoming"

}

// Additional relationships...

]

}

```

5.3.4.4 POST /api/v1/explore/history/start

Description: Start a new exploration session

Authentication: Required

Request Body:

```json

{

"entry\_point": "search",

"initial\_concept\_id": 42

}

```

Response:

```json

{

"session\_id": "550e8400-e29b-41d4-a716-446655440000",

"start\_time": "2025-06-06T13:30:00Z",

"entry\_point": "search",

"initial\_concept\_id": 42,

"xp\_earned": 5

}

```

5.3.4.5 PUT /api/v1/explore/history/{session\_id}/update

Description: Update an exploration session with a new concept view

Authentication: Required

Request Body:

```json

{

"concept\_id": 67,

"timestamp": "2025-06-06T13:32:15Z",

"source\_action": "related\_concept\_click",

"duration\_seconds": 45

}

```

Response:

```json

{

"session\_id": "550e8400-e29b-41d4-a716-446655440000",

"concepts\_viewed": 3,

"current\_path": [42, 55, 67],

"last\_updated": "2025-06-06T13:32:15Z",

"xp\_earned": 2,

"total\_xp": 9

}

```

5.3.4.6 POST /api/v1/explore/history/{session\_id}/end

Description: End an exploration session

Authentication: Required

Request Body:

```json

{

"exit\_point": "navigation\_to\_journal",

"final\_concept\_id": 67

}

```

Response:

```json

{

"session\_id": "550e8400-e29b-41d4-a716-446655440000",

"start\_time": "2025-06-06T13:30:00Z",

"end\_time": "2025-06-06T13:35:22Z",

"total\_concepts\_viewed": 5,

"total\_time\_seconds": 322,

"entry\_point": "search",

"exit\_point": "navigation\_to\_journal",

"xp\_earned": 15,

"mastery\_increases": [

{

"concept\_id": 42,

"name": "Existentialism",

"previous\_level": 0.35,

"new\_level": 0.38

},

{

"concept\_id": 67,

"name": "Phenomenology",

"previous\_level": 0.20,

"new\_level": 0.25

}

]

}

```

5.3.4.7 GET /api/v1/explore/visualize/map

Description: Generate a concept map visualization

Authentication: Required

Parameters:

- central\_concept\_id: ID of the central concept

- depth: Relationship depth to include (default: 2)

- min\_strength: Minimum relationship strength (0.0-1.0)

- layout: Visualization layout type (radial, force, hierarchical)

- max\_nodes: Maximum number of nodes to include (default: 50)

Response:

```json

{

"nodes": [

{

"id": 42,

"name": "Existentialism",

"category": "Philosophy",

"size": 1.0,

"color": "#FF5733",

"mastery\_level": 0.38

},

{

"id": 67,

"name": "Phenomenology",

"category": "Philosophy",

"size": 0.8,

"color": "#C70039",

"mastery\_level": 0.25

}

// Additional nodes...

],

"links": [

{

"source": 42,

"target": 67,

"type": "influenced",

"strength": 0.8,

"description": "Existentialism was heavily influenced by Phenomenology"

}

// Additional links...

],

"layout\_parameters": {

"type": "force",

"center\_x": 0.5,

"center\_y": 0.5,

"gravity": 0.1

}

}

```

5.3.4.8 GET /api/v1/explore/visualize/constellation

Description: Generate a constellation view visualization

Authentication: Required

Parameters:

- focus\_tradition: Tradition to focus on (optional)

- focus\_era: Era to focus on (optional)

- focus\_category: Category to focus on (optional)

- include\_mastered: Whether to include mastered concepts (default: true)

Response:

```json

{

"constellations": [

{

"name": "Existentialism",

"center": {"x": 0.3, "y": 0.4},

"radius": 0.15,

"color": "#FF5733",

"concepts": [42, 67, 89, 103],

"connections": [

{"source": 42, "target": 67},

{"source": 42, "target": 89},

{"source": 67, "target": 103}

]

},

{

"name": "Stoicism",

"center": {"x": 0.7, "y": 0.6},

"radius": 0.12,

"color": "#3A86FF",

"concepts": [12, 24, 36],

"connections": [

{"source": 12, "target": 24},

{"source": 24, "target": 36}

]

}

// Additional constellations...

],

"user\_position": {

"x": 0.4,

"y": 0.5,

"current\_constellation": "Existentialism"

},

"navigation\_history": [

{"constellation": "Stoicism", "timestamp": "2025-06-05T10:15:30Z"},

{"constellation": "Existentialism", "timestamp": "2025-06-06T13:30:00Z"}

]

}

```

5.3.4.9 GET /api/v1/explore/recommendations

Description: Retrieve personalized concept recommendations

Authentication: Required

Parameters:

- limit: Maximum number of recommendations (default: 5)

- context: Context for recommendations (recent, mastery, interests, trending)

- exclude\_viewed: Whether to exclude recently viewed concepts (default: true)

Response:

```json

{

"recommendations": [

{

"concept\_id": 89,

"name": "Phenomenology",

"short\_description": "Philosophical method focusing on the structure of first-person experience",

"recommendation\_reason": "Based on your interest in Existentialism",

"recommendation\_strength": 0.85,

"difficulty\_level": 4

},

{

"concept\_id": 103,

"name": "Dasein",

"short\_description": "Heidegger's term for the distinctive mode of human existence",

"recommendation\_reason": "Complements your recent exploration of Existentialism",

"recommendation\_strength": 0.78,

"difficulty\_level": 5

}

// Additional recommendations...

],

"exploration\_insights": {

"dominant\_traditions": ["Continental", "Existentialism"],

"dominant\_eras": ["20th Century", "19th Century"],

"knowledge\_gaps": ["Ancient Philosophy", "Eastern Traditions"],

"mastery\_distribution": {

"beginner": 45,

"intermediate": 32,

"advanced": 18,

"expert": 5

}

}

}

```

5.3.4.10 GET /api/v1/explore/pathways

Description: Retrieve available learning pathways

Authentication: Required

Parameters:

- difficulty: Filter by difficulty level

- tradition: Filter by philosophical tradition

- limit: Maximum number of pathways to return

- include\_completed: Whether to include completed pathways (default: false)

Response:

```json

{

"pathways": [

{

"pathway\_id": 1,

"name": "Introduction to Existentialism",

"description": "A journey through the core concepts of Existentialist philosophy",

"difficulty\_level": 3,

"estimated\_duration\_minutes": 60,

"concept\_count": 8,

"is\_premium": false,

"user\_progress": {

"status": "in\_progress",

"completed\_concepts": 3,

"percentage": 37.5

}

},

{

"pathway\_id": 2,

"name": "Ancient Greek Ethics",

"description": "Explore the ethical frameworks of Ancient Greek philosophers",

"difficulty\_level": 2,

"estimated\_duration\_minutes": 45,

"concept\_count": 6,

"is\_premium": false,

"user\_progress": {

"status": "not\_started",

"completed\_concepts": 0,

"percentage": 0

}

}

// Additional pathways...

],

"total": 12,

"recommended\_pathway": 2,

"recommendation\_reason": "Complements your interest in ethical frameworks"

}

```

5.3.4.11 GET /api/v1/explore/pathways/{pathway\_id}

Description: Retrieve detailed information about a specific learning pathway

Authentication: Required

Parameters: None

Response:

```json

{

"pathway\_id": 1,

"name": "Introduction to Existentialism",

"description": "A journey through the core concepts of Existentialist philosophy",

"difficulty\_level": 3,

"estimated\_duration\_minutes": 60,

"is\_premium": false,

"concepts": [

{

"position": 1,

"concept\_id": 42,

"name": "Existentialism",

"short\_description": "Philosophical movement emphasizing individual existence, freedom and choice",

"mastery\_level": 0.38,

"completed": true

},

{

"position": 2,

"concept\_id": 67,

"name": "Phenomenology",

"short\_description": "Philosophical method focusing on the structure of first-person experience",

"mastery\_level": 0.25,

"completed": true

},

{

"position": 3,

"concept\_id": 103,

"name": "Dasein",

"short\_description": "Heidegger's term for the distinctive mode of human existence",

"mastery\_level": 0.15,

"completed": true

},

{

"position": 4,

"concept\_id": 128,

"name": "Authenticity",

"short\_description": "Being true to one's own personality, spirit, or character",

"mastery\_level": 0.10,

"completed": false

}

// Additional concepts...

],

"prerequisites": [

{

"concept\_id": 24,

"name": "Enlightenment",

"mastery\_level": 0.45,

"met": true

}

],

"user\_progress": {

"status": "in\_progress",

"started\_at": "2025-06-05T10:15:30Z",

"last\_activity": "2025-06-06T13:35:22Z",

"completed\_concepts": 3,

"total\_concepts": 8,

"percentage": 37.5

},

"xp\_reward": 100,

"xp\_earned": 37

}

```

5.3.4.12 POST /api/v1/explore/pathways/{pathway\_id}/start

Description: Start a learning pathway

Authentication: Required

Request Body: None

Response:

```json

{

"success": true,

"pathway\_id": 2,

"name": "Ancient Greek Ethics",

"status": "in\_progress",

"started\_at": "2025-06-06T14:00:00Z",

"first\_concept": {

"concept\_id": 12,

"name": "Virtue Ethics",

"short\_description": "Ethical theory that emphasizes the virtues, or moral character"

},

"xp\_earned": 10,

"total\_xp": 1285

}

```

5.3.4.13 POST /api/v1/explore/pathways/{pathway\_id}/concepts/{concept\_id}/complete

Description: Mark a concept as completed in a learning pathway

Authentication: Required

Request Body:

```json

{

"time\_spent\_seconds": 180,

"notes": "I found this concept particularly interesting because..."

}

```

Response:

```json

{

"success": true,

"pathway\_id": 2,

"concept\_id": 12,

"status": "completed",

"completed\_at": "2025-06-06T14:03:00Z",

"mastery\_increase": {

"previous\_level": 0.30,

"new\_level": 0.38

},

"next\_concept": {

"concept\_id": 15,

"name": "Eudaimonia",

"short\_description": "Aristotle's concept of human flourishing and well-being"

},

"pathway\_progress": {

"completed\_concepts": 1,

"total\_concepts": 6,

"percentage": 16.7

},

"xp\_earned": 15,

"total\_xp": 1300

}

```

5.3.4.14 GET /api/v1/explore/notifications

Description: Retrieve exploration-related notifications for the user

Authentication: Required

Parameters:

- unread\_only: Only return unread notifications (default: false)

- limit: Maximum number of notifications (default: 10)

- types: Filter by notification types (comma-separated)

Response:

```json

{

"total": 3,

"unread": 2,

"notifications": [

{

"notification\_id": 123,

"notification\_type": "concept\_milestone",

"content": "You've explored 50 concepts! Keep going to unlock the 'Philosophy Explorer' badge.",

"related\_concept\_id": null,

"is\_read": false,

"created\_at": "2025-06-05T10:15:30Z"

},

{

"notification\_id": 124,

"notification\_type": "mastery\_increase",

"content": "Your mastery of Existentialism has reached Intermediate level!",

"related\_concept\_id": 42,

"is\_read": false,

"created\_at": "2025-06-06T13:35:22Z"

},

{

"notification\_id": 125,

"notification\_type": "recommendation",

"content": "Based on your exploration, you might enjoy learning about Phenomenology.",

"related\_concept\_id": 67,

"is\_read": true,

"created\_at": "2025-06-04T09:20:15Z"

}

]

}

```

5.3.4.15 PUT /api/v1/explore/notifications/{notification\_id}/read

Description: Mark a notification as read

Authentication: Required

Request Body: None

Response:

```json

{

"notification\_id": 123,

"is\_read": true,

"updated\_at": "2025-06-06T14:10:22Z"

}

```

5.3.5 User Experience Flow

5.3.5.1 Concept Discovery and Exploration

- User navigates to the Explore screen from the main navigation

- System displays concept map visualization with popular or recently viewed concepts

- Alternative visualization modes (Map, Constellation, List) are available via toggle

- User can search for specific concepts or apply filters (category, tradition, era, difficulty)

- Each concept is represented visually with name, category indicator, and mastery level

- User selects a concept to view details

5.3.5.2 Concept Detail Exploration

- System displays comprehensive concept information

- User sees description, examples, and historical context

- Related concepts are shown with relationship types and strengths

- User's current mastery level is displayed with progress indicator

- "Explore Further" suggestions are provided based on relationships

- User can navigate to related concepts by selecting them

- System tracks this navigation as part of the exploration path

5.3.5.3 Visualization Interaction

- User can zoom and pan within the concept map or constellation view

- Double-tapping a concept centers the visualization on that concept

- Pinch gestures control zoom level on touch devices

- Relationships between concepts are shown as connecting lines

- Line thickness indicates relationship strength

- Color coding indicates concept categories or traditions

- User can filter the visualization by relationship type or strength

5.3.5.4 Learning Pathway Navigation

- User can access learning pathways from the Explore screen

- System displays available pathways with difficulty and duration estimates

- Pathways are organized by philosophical tradition or theme

- User's progress is shown for previously started pathways

- User selects a pathway to view details

- System displays the sequence of concepts with completion status

- User can start a new pathway or continue an existing one

- Completing concepts in a pathway awards XP and increases mastery

5.3.5.5 Exploration History

- System tracks the sequence of concepts viewed during a session

- Breadcrumb navigation allows returning to previously viewed concepts

- "Recently Explored" section shows concepts from recent sessions

- Exploration statistics show patterns and trends in user's learning

- User can save exploration paths for future reference

- System uses exploration history to generate personalized recommendations

5.3.5.6 Mastery Progression

- Viewing concept details increases exposure count

- Longer engagement with concepts provides larger mastery increases

- Mastery levels are visualized consistently across the app

- Mastery milestones trigger notifications and achievements

- System recommends concepts to balance mastery across categories

- Mastery level affects recommendation algorithms and difficulty settings

5.3.5.7 Cross-Feature Integration

- Concepts viewed in Explore can be saved to Journal with one tap

- Questions about concepts can be asked directly via Ask feature integration

- Concepts can be shared to Forum for discussion

- Quests related to currently viewed concepts are recommended

- Mastery progression is synchronized across all features

- XP earned from exploration contributes to overall user level

5.3.6 XP Integration

5.3.6.1 XP Award Structure

- Starting exploration session: +5 XP

- Viewing new concepts: +2 XP per concept

- Completing learning pathways: +10-50 XP (based on difficulty and length)

- Beginner pathways: 10-25 XP

- Intermediate pathways: 25-50 XP

- Advanced pathways: 50-100 XP

- Reaching mastery milestones: +5-20 XP

- Beginner mastery (25%): +5 XP

- Intermediate mastery (50%): +10 XP

- Advanced mastery (75%): +15 XP

- Expert mastery (100%): +20 XP

- Extended exploration sessions (10+ concepts): +10 XP bonus

5.3.6.2 XP Visualization

- Standard XP bar shows progress toward next user level

- XP gains are animated to provide positive reinforcement

- Exploration-specific XP breakdown available in statistics view

- Pathway progress shows XP earned and remaining

- Mastery increases show associated XP rewards

- Total XP and XP to next level are displayed on profile and explore screens

5.3.6.3 Badge Integration

- Concept Explorer: View 10 different concepts

- Concept Enthusiast: View 50 different concepts

- Concept Master: View 100 different concepts

- Concept Sage: View 250 different concepts

- Pathway Pioneer: Complete first learning pathway

- Pathway Voyager: Complete 5 learning pathways

- Pathway Navigator: Complete 15 learning pathways

- Tradition Specialist: Reach advanced mastery in 10 concepts from same tradition

- Renaissance Thinker: Reach intermediate mastery in concepts from 5 different traditions

- Constellation Cartographer: Explore all constellations in the visualization

5.3.7 Notifications Integration

5.3.7.1 Notification Types

- Concept Milestones: Achievements related to concept exploration count

- Mastery Increases: Notifications when mastery levels reach new thresholds

- Pathway Progress: Updates on learning pathway completion percentage

- Recommendations: Suggested concepts based on exploration patterns

- Related Content: Notifications about new content related to explored concepts

- Exploration Insights: Interesting patterns or statistics from user's exploration

- Achievement Unlocks: Badge awards and XP milestones

5.3.7.2 Notification Delivery

- In-app notifications appear in the notification banner

- Important milestones trigger modal celebrations

- Subtle indicators show unread notifications

- Notification preferences allow customization of delivery

- Notifications are stored for later viewing

- Notification frequency is balanced to prevent overload

5.3.7.3 Notification Actions

- Concept notifications include direct links to the concept

- Recommendation notifications provide one-tap navigation

- Achievement notifications display badge details and progress

- Pathway notifications allow resuming the pathway

- Notifications can be marked as read individually or in bulk

- Notification settings accessible from the Explore screen

5.3.8 Accessibility Features

5.3.8.1 Visual Accessibility

- High-contrast mode enhances visibility of concept elements

- Color blindness considerations in visualization color schemes

- Adjustable text sizes for all concept content

- Visual indicators do not rely solely on color

- Animations can be reduced or disabled

- Sufficient contrast ratios (WCAG AA compliance)

5.3.8.2 Screen Reader Support

- ARIA labels for all interactive elements

- Semantic HTML structure for proper navigation

- Descriptive alt text for concept images and diagrams

- Announcement of XP gains and mastery updates

- Keyboard focus management for modal dialogs

- Screen reader optimization mode for visualizations

5.3.8.3 Input Accessibility

- Voice commands for navigation and interaction

- Alternative input methods for visualization navigation

- Touch targets sized appropriately (minimum 44x44 pixels)

- Keyboard navigation for all interactive elements

- Reduced motion option for animations

- Extended timeouts for user interactions

5.3.8.4 Content Accessibility

- Simplified language option for complex philosophical concepts

- Transcripts for any audio or video content

- Structured content with clear headings and sections

- Progress saving for users who need to take breaks

- Offline access to downloaded concept content

- Text-based alternative for visual concept maps

5.3.9 Implementation Considerations

5.3.9.1 Visualization Performance

- WebGL rendering for smooth performance on all devices

- Progressive loading for large concept maps

- Level-of-detail adjustments based on zoom level

- Efficient graph layout algorithms

- Caching of visualization data for frequently accessed concepts

- Fallback rendering for devices without WebGL support

5.3.9.2 Offline Support

- Concept data is cached after initial download

- Exploration history is stored locally and synced when connection is restored

- Offline mode indicator shows sync status

- Essential functionality works without internet connection

- Background sync when connection is reestablished

- Bandwidth-conscious loading for mobile users

5.3.9.3 Premium Content

- Premium concepts are clearly marked with visual indicators

- Free users can view basic information but not detailed content

- Upgrade prompts are non-intrusive but clear

- Some visualization modes may be premium-only

- Premium content offers higher XP rewards and exclusive badges

- Trial access to premium content through specific pathways

5.3.9.4 Performance Optimization

- Efficient database queries for concept filtering and relationships

- Caching strategy for frequently accessed concepts

- Lazy loading for concept details until needed

- Optimized graph traversal algorithms

- Background processing for complex visualizations

- Resource-conscious implementation for mobile devices

5.3.9.5 Mobile Optimization

- Responsive design adapts to different screen sizes

- Touch-friendly controls throughout the interface

- Gesture support for visualization navigation

- Offline capability for on-the-go learning

- Battery-efficient rendering and animations

- Reduced data usage option for metered connections

5.3.10 Celery Tasks

5.3.10.1 process\_concept\_relationships

```python

@celery\_app.task(name="explore.process\_concept\_relationships")

def process\_concept\_relationships(concept\_id):

"""

Processes and updates relationships for a specific concept.

This task analyzes existing concept data and updates relationship strengths

based on semantic similarity, co-occurrence patterns, and user interaction data.

Parameters:

- concept\_id: ID of the concept to process relationships for

Returns:

- Dictionary with processing results and statistics

"""

db = get\_db\_connection()

concept\_service = ConceptService(db)

relationship\_service = ConceptRelationshipService(db)

# Get the concept and its current relationships

concept = concept\_service.get\_concept(concept\_id)

current\_relationships = relationship\_service.get\_relationships(concept\_id)

# Analyze and update relationship strengths

updated\_relationships = relationship\_service.analyze\_relationship\_strengths(

concept\_id,

current\_relationships

)

# Identify potential new relationships

new\_relationships = relationship\_service.discover\_potential\_relationships(concept\_id)

# Update the database with new relationship data

for rel in updated\_relationships:

relationship\_service.update\_relationship(

rel["relationship\_id"],

rel["relationship\_strength"],

rel["description"]

)

# Create new relationships if they meet threshold criteria

created\_relationships = []

for rel in new\_relationships:

if rel["confidence"] >= 0.7:

rel\_id = relationship\_service.create\_relationship(

concept\_id,

rel["target\_concept\_id"],

rel["relationship\_type"],

rel["relationship\_strength"],

rel["description"]

)

created\_relationships.append(rel\_id)

return {

"concept\_id": concept\_id,

"relationships\_processed": len(current\_relationships),

"relationships\_updated": len(updated\_relationships),

"relationships\_created": len(created\_relationships),

"processing\_time\_ms": 1250

}

```

5.3.10.2 generate\_exploration\_recommendations

```python

@celery\_app.task(name="explore.generate\_exploration\_recommendations")

def generate\_exploration\_recommendations(user\_id, context=None):

"""

Generates personalized concept recommendations for a user.

This task analyzes user history, interests, and concept mastery to create

tailored recommendations for further exploration.

Parameters:

- user\_id: ID of the user to generate recommendations for

- context: Optional context for recommendation generation

Returns:

- Dictionary with recommendation results

"""

db = get\_db\_connection()

recommendation\_service = RecommendationService(db)

user\_service = UserService(db)

# Get user profile and preferences

user\_profile = user\_service.get\_user\_profile(user\_id)

# Get user's exploration history

exploration\_history = recommendation\_service.get\_user\_exploration\_history(

user\_id,

limit=50

)

# Get user's concept mastery levels

mastery\_levels = recommendation\_service.get\_user\_mastery\_levels(user\_id)

# Generate recommendations based on different factors

history\_based = recommendation\_service.generate\_history\_based\_recommendations(

user\_id,

exploration\_history

)

mastery\_based = recommendation\_service.generate\_mastery\_based\_recommendations(

user\_id,

mastery\_levels

)

interest\_based = recommendation\_service.generate\_interest\_based\_recommendations(

user\_id,

user\_profile["interests"]

)

# Combine and rank recommendations

combined\_recommendations = recommendation\_service.combine\_recommendations(

history\_based,

mastery\_based,

interest\_based,

weights={

"history": 0.4,

"mastery": 0.3,

"interests": 0.3

}

)

# Store recommendations for quick retrieval

recommendation\_service.store\_user\_recommendations(

user\_id,

combined\_recommendations,

context=context

)

return {

"user\_id": user\_id,

"recommendations\_generated": len(combined\_recommendations),

"recommendation\_factors": {

"history\_weight": 0.4,

"mastery\_weight": 0.3,

"interest\_weight": 0.3

},

"processing\_time\_ms": 850

}

```

5.3.10.3 update\_concept\_mastery

```python

@celery\_app.task(name="explore.update\_concept\_mastery")

def update\_concept\_mastery(user\_id, concept\_id, interaction\_data):

"""

Updates a user's mastery level for a specific concept.

This task processes new interaction data to adjust the user's

concept mastery level based on engagement patterns.

Parameters:

- user\_id: ID of the user

- concept\_id: ID of the concept

- interaction\_data: Dictionary containing interaction details

Returns:

- Dictionary with updated mastery information

"""

db = get\_db\_connection()

mastery\_service = ConceptMasteryService(db)

notification\_service = NotificationService(db)

xp\_service = XPService(db)

# Get current mastery level

current\_mastery = mastery\_service.get\_mastery\_level(user\_id, concept\_id)

# Calculate mastery increase based on interaction type and duration

interaction\_type = interaction\_data.get("interaction\_type", "view")

duration\_seconds = interaction\_data.get("duration\_seconds", 0)

source\_feature = interaction\_data.get("source\_feature", "explore")

# Different interaction types have different weights

interaction\_weights = {

"view": 0.01,

"detailed\_view": 0.02,

"pathway\_completion": 0.05,

"quiz\_correct": 0.03,

"reflection": 0.04

}

# Calculate base increase

base\_increase = interaction\_weights.get(interaction\_type, 0.01)

# Adjust for duration (diminishing returns after 2 minutes)

duration\_factor = min(duration\_seconds / 120, 1.0) \* 0.5 + 0.5

# Calculate final mastery increase

mastery\_increase = base\_increase \* duration\_factor

# Apply mastery increase with diminishing returns as mastery gets higher

diminishing\_factor = 1.0 - (current\_mastery["level"] \* 0.5)

final\_increase = mastery\_increase \* diminishing\_factor

# Calculate new mastery level (capped at 1.0)

new\_mastery\_level = min(current\_mastery["level"] + final\_increase, 1.0)

# Update mastery in database

mastery\_service.update\_mastery\_level(

user\_id,

concept\_id,

new\_mastery\_level,

interaction\_data

)

# Check for mastery level milestones

milestone\_reached = False

milestone\_thresholds = [0.25, 0.5, 0.75, 1.0]

for threshold in milestone\_thresholds:

if current\_mastery["level"] < threshold and new\_mastery\_level >= threshold:

milestone\_reached = True

milestone\_name = {

0.25: "Beginner",

0.5: "Intermediate",

0.75: "Advanced",

1.0: "Expert"

}[threshold]

# Create notification for milestone

notification\_service.create\_notification(

user\_id,

"mastery\_increase",

f"Your mastery of {current\_mastery['concept\_name']} has reached {milestone\_name} level!",

related\_concept\_id=concept\_id

)

# Award XP for milestone

xp\_amount = {

0.25: 5,

0.5: 10,

0.75: 15,

1.0: 20

}[threshold]

xp\_service.award\_xp(

user\_id=user\_id,

action\_type="mastery\_milestone",

context={"concept\_id": concept\_id, "level": milestone\_name},

amount=xp\_amount

)

return {

"user\_id": user\_id,

"concept\_id": concept\_id,

"previous\_mastery": current\_mastery["level"],

"new\_mastery": new\_mastery\_level,

"increase": final\_increase,

"milestone\_reached": milestone\_reached,

"factors": {

"interaction\_type": interaction\_type,

"duration\_factor": duration\_factor,

"diminishing\_factor": diminishing\_factor

}

}

```

5.3.10.4 generate\_exploration\_notifications

```python

@celery\_app.task(name="explore.generate\_exploration\_notifications")

def generate\_exploration\_notifications(user\_id):

"""

Generates exploration-related notifications for a user.

This task analyzes user activity and progress to create

relevant notifications about milestones, recommendations,

and learning opportunities.

Parameters:

- user\_id: ID of the user to generate notifications for

Returns:

- Dictionary with notification generation results

"""

db = get\_db\_connection()

notification\_service = NotificationService(db)

exploration\_service = ExplorationHistoryService(db)

recommendation\_service = RecommendationService(db)

# Get user's exploration statistics

exploration\_stats = exploration\_service.get\_user\_exploration\_statistics(user\_id)

notifications\_generated = []

# Check for concept count milestones

concept\_milestones = [10, 25, 50, 100, 250, 500]

concepts\_explored = exploration\_stats["total\_concepts\_explored"]

for milestone in concept\_milestones:

if concepts\_explored >= milestone and not notification\_service.milestone\_exists(

user\_id, "concept\_count", milestone

):

notification = notification\_service.create\_notification(

user\_id,

"concept\_milestone",

f"You've explored {milestone} concepts! Keep going to unlock the next badge.",

related\_concept\_id=None

)

notifications\_generated.append(notification)

# Record milestone to prevent duplicate notifications

notification\_service.record\_milestone(

user\_id, "concept\_count", milestone

)

# Generate recommendations if user hasn't explored in 3+ days

last\_exploration = exploration\_stats["last\_exploration\_time"]

days\_since\_exploration = (datetime.now() - last\_exploration).days

if days\_since\_exploration >= 3:

# Get personalized recommendations

recommendations = recommendation\_service.get\_top\_recommendations(

user\_id, limit=1

)

if recommendations:

top\_recommendation = recommendations[0]

notification = notification\_service.create\_notification(

user\_id,

"recommendation",

f"Continue your philosophical journey with {top\_recommendation['name']}.",

related\_concept\_id=top\_recommendation["concept\_id"]

)

notifications\_generated.append(notification)

# Check for new content related to user's interests

user\_interests = exploration\_service.get\_user\_interest\_areas(user\_id)

new\_content = exploration\_service.get\_new\_content\_for\_interests(

user\_interests, days=7

)

if new\_content:

notification = notification\_service.create\_notification(

user\_id,

"new\_content",

f"New content available in {new\_content[0]['name']} and {len(new\_content)-1} other areas you're interested in.",

related\_concept\_id=new\_content[0]["concept\_id"]

)

notifications\_generated.append(notification)

return {

"user\_id": user\_id,

"notifications\_generated": len(notifications\_generated),

"notification\_types": list(set(n["type"] for n in notifications\_generated)),

"processing\_time\_ms": 320

}

```

5.3.10.5 cache\_concept\_visualizations

```python

@celery\_app.task(name="explore.cache\_concept\_visualizations")

def cache\_concept\_visualizations(concept\_ids=None, force\_refresh=False):

"""

Pre-generates and caches concept visualization data.

This task creates visualization data for specified concepts

and stores it in the cache for faster retrieval.

Parameters:

- concept\_ids: List of concept IDs to process (None for all)

- force\_refresh: Whether to force refresh existing cache entries

Returns:

- Dictionary with caching results

"""

db = get\_db\_connection()

visualization\_service = VisualizationService(db)

concept\_service = ConceptService(db)

cache\_service = CacheService()

# If no specific concepts provided, get popular concepts

if not concept\_ids:

concept\_ids = concept\_service.get\_popular\_concept\_ids(limit=100)

cache\_hits = 0

cache\_misses = 0

visualizations\_generated = 0

for concept\_id in concept\_ids:

cache\_key = f"concept\_visualization:{concept\_id}"

# Check if visualization is already cached

if not force\_refresh and cache\_service.exists(cache\_key):

cache\_hits += 1

continue

cache\_misses += 1

# Generate visualization data for different depths and layouts

for depth in [1, 2, 3]:

for layout in ["radial", "force", "hierarchical"]:

visualization\_data = visualization\_service.generate\_concept\_map(

concept\_id,

depth=depth,

layout=layout,

min\_strength=0.3,

max\_nodes=50

)

# Cache the visualization data

cache\_key = f"concept\_visualization:{concept\_id}:{depth}:{layout}"

cache\_service.set(

cache\_key,

visualization\_data,

expiry\_seconds=86400 # 24 hours

)

visualizations\_generated += 1

# Also generate constellation view data

constellation\_data = visualization\_service.generate\_constellation\_view(

central\_concept\_id=concept\_id

)

cache\_key = f"concept\_constellation:{concept\_id}"

cache\_service.set(

cache\_key,

constellation\_data,

expiry\_seconds=86400 # 24 hours

)

visualizations\_generated += 1

return {

"concepts\_processed": len(concept\_ids),

"visualizations\_generated": visualizations\_generated,

"cache\_hits": cache\_hits,

"cache\_misses": cache\_misses,

"processing\_time\_ms": 4500

}

```

5.3.10.6 analyze\_exploration\_patterns

```python

@celery\_app.task(name="explore.analyze\_exploration\_patterns")

def analyze\_exploration\_patterns(user\_id=None, days=30):

"""

Analyzes user exploration patterns to identify trends and insights.

This task processes exploration history data to generate insights

about user learning patterns, preferences, and opportunities.

Parameters:

- user\_id: ID of the user to analyze (None for all users)

- days: Number of days of history to analyze

Returns:

- Dictionary with analysis results

"""

db = get\_db\_connection()

exploration\_service = ExplorationHistoryService(db)

if user\_id:

# Analyze individual user patterns

user\_sessions = exploration\_service.get\_user\_sessions(

user\_id,

days=days

)

# Extract exploration paths

paths = [session["exploration\_path"] for session in user\_sessions]

# Analyze common transitions between concepts

transitions = exploration\_service.analyze\_concept\_transitions(paths)

# Identify dominant traditions and categories

dominant\_areas = exploration\_service.identify\_dominant\_areas(

user\_id,

days=days

)

# Identify knowledge gaps

knowledge\_gaps = exploration\_service.identify\_knowledge\_gaps(

user\_id,

dominant\_areas

)

# Generate personalized insights

insights = exploration\_service.generate\_user\_insights(

user\_id,

transitions,

dominant\_areas,

knowledge\_gaps

)

# Store insights for later retrieval

exploration\_service.store\_user\_insights(user\_id, insights)

return {

"user\_id": user\_id,

"sessions\_analyzed": len(user\_sessions),

"insights\_generated": len(insights),

"dominant\_traditions": dominant\_areas["traditions"],

"dominant\_eras": dominant\_areas["eras"],

"knowledge\_gaps": knowledge\_gaps,

"processing\_time\_ms": 1200

}

else:

# Analyze patterns across all users

global\_patterns = exploration\_service.analyze\_global\_patterns(days=days)

# Identify trending concepts

trending\_concepts = exploration\_service.identify\_trending\_concepts(days=days)

# Update concept popularity scores

exploration\_service.update\_concept\_popularity\_scores(trending\_concepts)

return {

"users\_analyzed": global\_patterns["user\_count"],

"sessions\_analyzed": global\_patterns["session\_count"],

"trending\_concepts": len(trending\_concepts),

"popular\_pathways": global\_patterns["popular\_pathways"],

"processing\_time\_ms": 5500

}

}

```

5.3.10.7 sync\_offline\_exploration\_data

```python

@celery\_app.task(name="explore.sync\_offline\_exploration\_data")

def sync\_offline\_exploration\_data(user\_id, offline\_data):

"""

Synchronizes offline exploration data with the server database.

This task processes exploration data collected while the user

was offline and integrates it with their online history.

Parameters:

- user\_id: ID of the user

- offline\_data: Dictionary containing offline exploration data

Returns:

- Dictionary with synchronization results

"""

db = get\_db\_connection()

exploration\_service = ExplorationHistoryService(db)

mastery\_service = ConceptMasteryService(db)

# Extract offline sessions

offline\_sessions = offline\_data.get("sessions", [])

# Extract offline interactions

offline\_interactions = offline\_data.get("interactions", [])

# Process and store offline sessions

sessions\_synced = 0

for session in offline\_sessions:

# Check if session already exists

if not exploration\_service.session\_exists(session["session\_id"]):

exploration\_service.store\_exploration\_session(

user\_id,

session["session\_id"],

session["exploration\_path"],

session["start\_time"],

session["end\_time"],

session["entry\_point"],

session["exit\_point"]

)

sessions\_synced += 1

# Process and store offline interactions

interactions\_synced = 0

for interaction in offline\_interactions:

# Check if interaction already exists

if not exploration\_service.interaction\_exists(

user\_id,

interaction["concept\_id"],

interaction["interaction\_type"],

interaction["timestamp"]

):

exploration\_service.store\_concept\_interaction(

user\_id,

interaction["concept\_id"],

interaction["interaction\_type"],

interaction["timestamp"],

interaction.get("duration\_seconds"),

interaction.get("source\_feature"),

interaction.get("context\_data")

)

# Update mastery based on offline interaction

update\_concept\_mastery.delay(

user\_id,

interaction["concept\_id"],

{

"interaction\_type": interaction["interaction\_type"],

"duration\_seconds": interaction.get("duration\_seconds", 0),

"source\_feature": interaction.get("source\_feature", "explore"),

"offline": True

}

)

interactions\_synced += 1

# Generate notifications for significant offline activity

if sessions\_synced > 0 or interactions\_synced > 0:

generate\_exploration\_notifications.delay(user\_id)

return {

"user\_id": user\_id,

"sessions\_synced": sessions\_synced,

"interactions\_synced": interactions\_synced,

"total\_offline\_sessions": len(offline\_sessions),

"total\_offline\_interactions": len(offline\_interactions),

"processing\_time\_ms": 850

}

}

```