# Concepts System Expansion and Integration Plan

## Overview

This document outlines a comprehensive expansion of the Concepts system, transforming it from a simple tagging mechanism into the central nervous system of the Setarcos app. The enhanced Concepts feature will serve as the primary connective tissue between all app features, creating a cohesive, exploratory philosophical experience while fostering community engagement and personalized learning.

## Core Design Principles

1. **Centrality**: Concepts serve as the primary organizing principle for all content and interactions
2. **Discoverability**: Users can explore philosophical ideas through multiple interconnected pathways
3. **Personalization**: The system adapts to user interests and learning patterns
4. **Community**: Shared exploration and contribution around philosophical concepts
5. **Depth**: Progressive layers of understanding from introductory to advanced

## System Architecture

### 1. Concepts Data Model

#### 1.1 Core Concept Entity

CREATE TABLE concepts (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

name VARCHAR(100) NOT NULL,

normalized\_name VARCHAR(100) NOT NULL,

short\_description TEXT NOT NULL,

long\_description TEXT NOT NULL,

etymology TEXT,

historical\_context TEXT,

difficulty\_level INTEGER NOT NULL CHECK (difficulty\_level BETWEEN 1 AND 5),

parent\_category\_id UUID REFERENCES concept\_categories(id),

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

is\_verified BOOLEAN DEFAULT FALSE,

verification\_date TIMESTAMP WITH TIME ZONE,

verified\_by UUID REFERENCES users(id),

UNIQUE(normalized\_name)

);

CREATE INDEX idx\_concepts\_normalized\_name ON concepts(normalized\_name);

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

CREATE INDEX idx\_concepts\_parent\_category\_id ON concepts(parent\_category\_id);

#### 1.2 Concept Relationships

CREATE TABLE concept\_relationships (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

source\_concept\_id UUID NOT NULL REFERENCES concepts(id),

target\_concept\_id UUID NOT NULL REFERENCES concepts(id),

relationship\_type VARCHAR(50) NOT NULL,

relationship\_strength FLOAT NOT NULL CHECK (relationship\_strength BETWEEN 0.0 AND 1.0),

description TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

is\_verified BOOLEAN DEFAULT FALSE,

verification\_date TIMESTAMP WITH TIME ZONE,

verified\_by UUID REFERENCES users(id),

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

CHECK (source\_concept\_id != target\_concept\_id)

);

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);

#### 1.3 Concept Categories

CREATE TABLE concept\_categories (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

name VARCHAR(100) NOT NULL,

description TEXT NOT NULL,

color\_hex VARCHAR(7) NOT NULL,

icon\_name VARCHAR(50) NOT NULL,

parent\_category\_id UUID REFERENCES concept\_categories(id),

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

UNIQUE(name)

);

CREATE INDEX idx\_concept\_categories\_parent ON concept\_categories(parent\_category\_id);

#### 1.4 User Concept Interactions

CREATE TABLE user\_concept\_interactions (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

user\_id UUID NOT NULL REFERENCES users(id),

concept\_id UUID NOT NULL REFERENCES concepts(id),

interaction\_type VARCHAR(50) NOT NULL,

interaction\_data JSONB,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

source\_feature VARCHAR(50) NOT NULL,

source\_item\_id UUID,

xp\_earned INTEGER DEFAULT 0,

UNIQUE(user\_id, concept\_id, interaction\_type, source\_feature, source\_item\_id)

);

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\_source ON user\_concept\_interactions(source\_feature);

#### 1.5 Concept Mastery

CREATE TABLE concept\_mastery (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

user\_id UUID NOT NULL REFERENCES users(id),

concept\_id UUID NOT NULL REFERENCES concepts(id),

mastery\_level INTEGER NOT NULL CHECK (mastery\_level BETWEEN 0 AND 100),

exposure\_count INTEGER NOT NULL DEFAULT 0,

interaction\_count INTEGER NOT NULL DEFAULT 0,

last\_interaction\_date TIMESTAMP WITH TIME ZONE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

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);

#### 1.6 Concept Resources

CREATE TABLE concept\_resources (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

concept\_id UUID NOT NULL REFERENCES concepts(id),

resource\_type VARCHAR(50) NOT NULL,

title VARCHAR(255) NOT NULL,

description TEXT NOT NULL,

url TEXT,

content TEXT,

author VARCHAR(255),

difficulty\_level INTEGER NOT NULL CHECK (difficulty\_level BETWEEN 1 AND 5),

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

is\_verified BOOLEAN DEFAULT FALSE,

verification\_date TIMESTAMP WITH TIME ZONE,

verified\_by UUID REFERENCES users(id)

);

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

CREATE INDEX idx\_concept\_resources\_type ON concept\_resources(resource\_type);

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

#### 1.7 Community Concept Contributions

CREATE TABLE concept\_contributions (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

user\_id UUID NOT NULL REFERENCES users(id),

concept\_id UUID NOT NULL REFERENCES concepts(id),

contribution\_type VARCHAR(50) NOT NULL,

content TEXT NOT NULL,

status VARCHAR(20) NOT NULL DEFAULT 'pending',

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

reviewed\_by UUID REFERENCES users(id),

review\_date TIMESTAMP WITH TIME ZONE,

review\_notes TEXT

);

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

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

CREATE INDEX idx\_concept\_contributions\_status ON concept\_contributions(status);

### 2. API Endpoints

#### 2.1 Core Concept Endpoints

GET /api/v1/concepts

GET /api/v1/concepts/{id}

GET /api/v1/concepts/search

GET /api/v1/concepts/categories

GET /api/v1/concepts/{id}/relationships

GET /api/v1/concepts/{id}/resources

#### 2.2 User Interaction Endpoints

GET /api/v1/concepts/user/interactions

GET /api/v1/concepts/user/mastery

POST /api/v1/concepts/{id}/interact

POST /api/v1/concepts/{id}/bookmark

POST /api/v1/concepts/{id}/share

#### 2.3 Community Endpoints

GET /api/v1/concepts/{id}/community

POST /api/v1/concepts/{id}/contribute

GET /api/v1/concepts/contributions

PUT /api/v1/concepts/contributions/{id}

#### 2.4 Integration Endpoints

GET /api/v1/concepts/{id}/ask-responses

GET /api/v1/concepts/{id}/journal-entries

GET /api/v1/concepts/{id}/quests

GET /api/v1/concepts/{id}/forum-posts

### 3. Feature Integration

#### 3.1 AI Router Integration

The AI Router will be enhanced to:

1. **Extract Concepts**: Automatically identify philosophical concepts in user questions and AI responses
2. **Rank Concept Relevance**: Determine primary and secondary concepts in content
3. **Suggest Related Concepts**: Recommend related concepts for further exploration
4. **Adapt to Mastery Level**: Tailor responses based on user's concept mastery

# AI Router Concept Integration

def process\_response(response\_text, user\_id):

# Extract concepts from response

extracted\_concepts = concept\_extractor.extract(response\_text)

# Get user's mastery levels for these concepts

mastery\_levels = get\_user\_concept\_mastery(user\_id, [c['id'] for c in extracted\_concepts])

# Record concept exposures

record\_concept\_exposures(user\_id, extracted\_concepts, 'ask', response\_id)

# Enhance response with concept badges

enhanced\_response = concept\_badge\_enhancer.enhance(

response\_text,

extracted\_concepts,

mastery\_levels

)

# Generate related concepts suggestions

related\_concepts = get\_related\_concepts(extracted\_concepts, mastery\_levels)

return {

'enhanced\_response': enhanced\_response,

'concepts': extracted\_concepts,

'related\_concepts': related\_concepts

}

#### 3.2 Ask Feature Integration

The Ask feature will be enhanced to:

1. **Concept Badges**: Interactive concept badges in AI responses
2. **Concept Tooltips**: Hover tooltips with definitions and mastery level
3. **Concept Exploration**: Click-through to concept detail from badges
4. **Concept History**: Track concepts discussed in conversation history
5. **Concept-Based Suggestions**: Suggest follow-up questions based on concepts

// Concept Badge Component

const ConceptBadge = ({ concept, userMastery }) => {

const masteryColor = getMasteryColor(userMastery);

return (

<Badge

onClick={() => openConceptDetail(concept.id)}

onHover={() => showConceptTooltip(concept)}

style={{ backgroundColor: concept.category.color, borderColor: masteryColor }}

>

{concept.name}

{userMastery > 0 && <MasteryIndicator level={userMastery} />}

</Badge>

);

};

// Response Enhancement

const enhanceResponseWithConcepts = (responseText, concepts, userMastery) => {

// Replace concept mentions with interactive badges

return responseText.replace(conceptRegex, (match) => {

const concept = concepts.find(c => c.name === match);

if (!concept) return match;

return renderToString(

<ConceptBadge

concept={concept}

userMastery={userMastery[concept.id] || 0}

/>

);

});

};

#### 3.3 Seek Clarity Integration

The Seek Clarity feature will be enhanced to:

1. **Concept-Specific Pathways**: Tailor pathways based on concepts in the response
2. **Concept Mastery Challenges**: Create challenges focused on specific concepts
3. **Concept Relationship Exploration**: Explore connections between concepts
4. **Concept-Based Examples**: Provide real-world examples illustrating concepts
5. **Concept Learning Paths**: Generate learning paths for deeper concept understanding

# Seek Clarity Concept Integration

def generate\_concept\_based\_pathways(response\_id, concepts):

response = get\_response(response\_id)

pathways = []

# Generate practical examples pathway

if any(c for c in concepts if c['difficulty\_level'] >= 3):

pathways.append({

'type': 'practical\_examples',

'title': 'See It In Action',

'description': f"Explore real-world examples of {concepts[0]['name']}",

'concepts': concepts

})

# Generate journal challenge pathway

primary\_concept = concepts[0]

challenge = generate\_concept\_challenge(primary\_concept)

pathways.append({

'type': 'journal\_challenge',

'title': f"Reflect on {primary\_concept['name']}",

'description': challenge['short\_description'],

'challenge': challenge

})

# Generate learning path pathway

if len(concepts) >= 2:

learning\_path = generate\_concept\_learning\_path(concepts)

pathways.append({

'type': 'learning\_path',

'title': 'Explore Deeper',

'description': f"Discover how {concepts[0]['name']} connects to {concepts[1]['name']}",

'learning\_path': learning\_path

})

return pathways

#### 3.4 Journal Integration

The Journal feature will be enhanced to:

1. **Concept Tagging**: Automatic and manual tagging of journal entries
2. **Concept Reflections**: Guided reflection templates for specific concepts
3. **Concept Mastery Tracking**: Track concept understanding through journal entries
4. **Concept Collections**: Group journal entries by concept
5. **Concept Challenges**: Concept-specific writing prompts and challenges

// Journal Concept Integration

const JournalConceptIntegration = {

// Analyze journal entry and suggest concepts

analyzeEntry: async (entryText) => {

const response = await api.post('/api/v1/concepts/extract', { text: entryText });

return response.data.concepts;

},

// Get concept-based reflection prompts

getConceptPrompts: async (conceptIds) => {

const response = await api.get('/api/v1/concepts/reflection-prompts', {

params: { conceptIds: conceptIds.join(',') }

});

return response.data.prompts;

},

// Record concept interaction from journal

recordJournalConceptInteraction: async (journalId, conceptId) => {

return await api.post('/api/v1/concepts/interact', {

conceptId,

interactionType: 'journal\_reflection',

sourceFeature: 'journal',

sourceItemId: journalId

});

},

// Get concept-based challenges

getConceptChallenges: async (userMasteryLevels) => {

return await api.get('/api/v1/concepts/challenges', {

params: { masteryLevels: JSON.stringify(userMasteryLevels) }

});

}

};

#### 3.5 Quest Integration

The Quest feature will be enhanced to:

1. **Concept-Based Quests**: Quests organized around specific concepts
2. **Concept Progression**: Sequential quests that build concept mastery
3. **Concept Relationships**: Quests that explore connections between concepts
4. **Concept Challenges**: Multi-step challenges to deepen concept understanding
5. **Concept Mastery Rewards**: XP and badges for concept mastery through quests

# Quest Concept Integration

class ConceptQuestGenerator:

def generate\_concept\_quest(self, concept\_id, user\_id):

# Get concept details

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

# Get user's mastery level

mastery = self.mastery\_service.get\_user\_mastery(user\_id, concept\_id)

# Generate appropriate quest based on mastery level

if mastery < 20:

return self.\_generate\_introductory\_quest(concept)

elif mastery < 50:

return self.\_generate\_intermediate\_quest(concept)

else:

return self.\_generate\_advanced\_quest(concept)

def \_generate\_introductory\_quest(self, concept):

# Create a quest for beginners to this concept

steps = [

{

'type': 'learn',

'title': f"Understand {concept['name']}",

'description': f"Read about the basic definition of {concept['name']}",

'content': concept['short\_description'],

'xp': 10

},

{

'type': 'reflect',

'title': f"Reflect on {concept['name']}",

'description': f"Write a short reflection on what {concept['name']} means to you",

'prompt': f"What does {concept['name']} mean in your own words?",

'xp': 15

},

{

'type': 'apply',

'title': f"Apply {concept['name']}",

'description': f"Identify an example of {concept['name']} in your daily life",

'prompt': f"Describe a situation where you've encountered {concept['name']}",

'xp': 20

}

]

return {

'title': f"Discovering {concept['name']}",

'description': f"Begin your journey understanding {concept['name']}",

'concept\_id': concept['id'],

'difficulty': 1,

'steps': steps,

'total\_xp': 45,

'estimated\_time': '15 minutes'

}

#### 3.6 Explore Feature Integration

The Explore feature will be completely reimagined as a central hub for concept exploration:

1. **Concept Map**: Interactive visual map of philosophical concepts and relationships
2. **Concept Journey**: Personalized exploration paths based on interests and mastery
3. **Concept Collections**: Curated collections of related concepts
4. **Community Insights**: Community contributions and discussions around concepts
5. **Personal Progress**: Visual representation of concept mastery and exploration history

// Explore Feature Concept Map

class ConceptMapVisualization {

constructor(container, userId) {

this.container = container;

this.userId = userId;

this.graph = new ForceDirectedGraph(container);

this.userMastery = {};

this.conceptData = {};

this.relationshipData = {};

}

async initialize() {

// Load user's concept mastery data

const masteryResponse = await api.get('/api/v1/concepts/user/mastery');

this.userMastery = masteryResponse.data.reduce((acc, item) => {

acc[item.conceptId] = item.masteryLevel;

return acc;

}, {});

// Load initial concepts (based on user's interests and mastery)

const conceptsResponse = await api.get('/api/v1/concepts/explore/map', {

params: { userId: this.userId }

});

this.conceptData = conceptsResponse.data.concepts;

this.relationshipData = conceptsResponse.data.relationships;

// Initialize visualization

this.\_buildGraph();

this.\_setupInteractions();

}

\_buildGraph() {

// Add nodes for each concept

this.conceptData.forEach(concept => {

const mastery = this.userMastery[concept.id] || 0;

const nodeSize = this.\_calculateNodeSize(concept, mastery);

const nodeColor = this.\_calculateNodeColor(concept, mastery);

this.graph.addNode({

id: concept.id,

label: concept.name,

size: nodeSize,

color: nodeColor,

data: concept

});

});

// Add edges for relationships

this.relationshipData.forEach(rel => {

this.graph.addEdge({

source: rel.sourceConceptId,

target: rel.targetConceptId,

label: rel.relationshipType,

width: rel.relationshipStrength \* 5,

data: rel

});

});

// Apply layout and render

this.graph.applyForceLayout();

this.graph.render();

}

\_setupInteractions() {

// Node click - open concept detail

this.graph.onNodeClick(node => {

this.\_openConceptDetail(node.id);

});

// Node hover - show preview

this.graph.onNodeHover(node => {

this.\_showConceptPreview(node.id);

});

// Edge hover - show relationship

this.graph.onEdgeHover(edge => {

this.\_showRelationshipDetail(edge.data);

});

// Add exploration controls

this.\_setupExplorationControls();

}

async expandNode(conceptId) {

// Load related concepts

const relatedResponse = await api.get(`/api/v1/concepts/${conceptId}/relationships`);

const newConcepts = relatedResponse.data.concepts.filter(

c => !this.conceptData.some(existing => existing.id === c.id)

);

// Add new concepts to the graph

newConcepts.forEach(concept => {

const mastery = this.userMastery[concept.id] || 0;

this.conceptData.push(concept);

this.graph.addNode({

id: concept.id,

label: concept.name,

size: this.\_calculateNodeSize(concept, mastery),

color: this.\_calculateNodeColor(concept, mastery),

data: concept

});

});

// Add new relationships

relatedResponse.data.relationships.forEach(rel => {

if (!this.relationshipData.some(

existing => existing.sourceConceptId === rel.sourceConceptId &&

existing.targetConceptId === rel.targetConceptId

)) {

this.relationshipData.push(rel);

this.graph.addEdge({

source: rel.sourceConceptId,

target: rel.targetConceptId,

label: rel.relationshipType,

width: rel.relationshipStrength \* 5,

data: rel

});

}

});

// Update layout and render

this.graph.applyForceLayout();

this.graph.render();

}

}

#### 3.7 Forum Integration

The Forum feature will be enhanced to:

1. **Concept Forums**: Dedicated discussion spaces for specific concepts
2. **Concept Questions**: Concept-specific questions and answers
3. **Concept Debates**: Structured debates around philosophical concepts
4. **Concept Resources**: Community-contributed resources for concepts
5. **Concept Experts**: Recognition for users with high concept mastery

# Forum Concept Integration

class ConceptForumService:

def get\_concept\_forum(self, concept\_id):

# Get concept details

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

# Get forum posts for this concept

posts = self.forum\_service.get\_posts\_by\_concept(concept\_id)

# Get related concepts

related\_concepts = self.concept\_service.get\_related\_concepts(concept\_id)

# Get concept experts (users with high mastery)

experts = self.mastery\_service.get\_concept\_experts(concept\_id)

# Get concept resources

resources = self.resource\_service.get\_concept\_resources(concept\_id)

return {

'concept': concept,

'posts': posts,

'related\_concepts': related\_concepts,

'experts': experts,

'resources': resources

}

def create\_concept\_post(self, user\_id, concept\_id, post\_data):

# Create forum post

post\_id = self.forum\_service.create\_post(user\_id, post\_data)

# Link post to concept

self.forum\_service.link\_post\_to\_concept(post\_id, concept\_id)

# Record concept interaction

self.interaction\_service.record\_interaction(

user\_id,

concept\_id,

'forum\_post',

'forum',

post\_id

)

# Update user's concept mastery

self.mastery\_service.update\_mastery(user\_id, concept\_id, 'forum\_post')

return post\_id

### 4. User Experience

#### 4.1 Concept Detail View

The Concept Detail view will serve as a comprehensive hub for each philosophical concept:

1. **Overview Section**:
   * Name, short description, etymology
   * Visual category indicator
   * Difficulty level
   * User's mastery level and progress
2. **Understanding Section**:
   * Long description with rich formatting
   * Historical context
   * Key philosophers associated with the concept
   * Common misconceptions
3. **Connections Section**:
   * Related concepts with relationship explanations
   * Visual relationship map
   * Concept hierarchy (broader/narrower concepts)
4. **Resources Section**:
   * Curated learning resources (articles, videos, books)
   * Difficulty-appropriate content
   * Community-contributed resources
5. **Community Section**:
   * Recent forum discussions
   * User contributions
   * Expert insights
   * Shared journal entries (if permitted)
6. **Personal Section**:
   * User's interaction history with the concept
   * Journal entries tagged with this concept
   * Completed quests related to this concept
   * Ask conversations mentioning this concept
7. **Actions Section**:
   * Start a concept-specific quest
   * Join the concept forum
   * Create a journal entry about this concept
   * Share concept with others

#### 4.2 Concept Mastery System

The Concept Mastery system will provide a structured approach to tracking and improving understanding:

1. **Mastery Levels**:
   * Level 1: Awareness (0-20%)
   * Level 2: Familiarity (21-40%)
   * Level 3: Understanding (41-60%)
   * Level 4: Application (61-80%)
   * Level 5: Expertise (81-100%)
2. **Mastery Indicators**:
   * Visual indicators on concept badges
   * Progress bars in concept detail view
   * Mastery-based recommendations
3. **Mastery Progression**:
   * Points earned through various interactions:
     + Reading about a concept: 1-2 points
     + Asking questions about a concept: 2-5 points
     + Completing concept quests: 5-20 points
     + Writing journal entries: 5-15 points
     + Participating in concept forums: 3-10 points
     + Teaching others (answers, contributions): 10-25 points
4. **Mastery Benefits**:
   * Unlock advanced resources
   * Access to expert discussions
   * Ability to contribute to concept knowledge base
   * Recognition as concept expert
   * Special badges and achievements

#### 4.3 Concept Map Experience

The Concept Map will be the central exploration tool:

1. **Visual Design**:
   * Force-directed graph visualization
   * Nodes representing concepts
   * Edges representing relationships
   * Color coding by category
   * Size variation by importance or mastery
2. **Interaction Modes**:
   * Explore Mode: Free exploration of the concept landscape
   * Focus Mode: Centered on a specific concept with relationships
   * Journey Mode: Guided path through related concepts
   * Mastery Mode: Visualization of user's concept mastery
3. **Personalization**:
   * Highlight concepts based on user interests
   * Emphasize concepts ready for learning (just beyond current mastery)
   * Show mastery progress visually
   * Custom concept collections and saved paths
4. **Discovery Features**:
   * Suggested exploration paths
   * Daily concept recommendations
   * "Concept of the day" highlights
   * Serendipitous discovery options

### 5. Community and Sharing

#### 5.1 Concept Contributions

Users can contribute to the concept knowledge base:

1. **Contribution Types**:
   * Concept definitions and descriptions
   * Relationship suggestions
   * Resource recommendations
   * Examples and applications
   * Clarifications and corrections
2. **Contribution Process**:
   * Submit contribution
   * Community voting/feedback
   * Expert review
   * Integration into knowledge base
   * Attribution and rewards
3. **Contribution Rewards**:
   * XP and badges
   * Recognition on concept pages
   * Progress toward expert status
   * Special privileges

#### 5.2 Concept Sharing

Users can share their concept interactions:

1. **Sharing Options**:
   * Share concept with friends
   * Share personal insights about a concept
   * Share concept collections
   * Share mastery achievements
   * Share exploration paths
2. **Privacy Controls**:
   * Public sharing
   * Friends-only sharing
   * Private collections
   * Anonymous contributions
3. **Sharing Formats**:
   * In-app notifications
   * Shareable links
   * Visual cards for social media
   * Embeddable concept maps

### 6. Technical Implementation

#### 6.1 Concept Extraction Service

class ConceptExtractionService:

def \_\_init\_\_(self, concept\_repository, ai\_router):

self.concept\_repository = concept\_repository

self.ai\_router = ai\_router

self.concept\_cache = {}

self.load\_concepts()

def load\_concepts(self):

# Load all concepts into memory for fast matching

concepts = self.concept\_repository.get\_all\_concepts()

for concept in concepts:

self.concept\_cache[concept['normalized\_name']] = concept

def extract\_concepts(self, text):

# First pass: Direct matching from our concept database

matched\_concepts = self.\_direct\_concept\_matching(text)

# Second pass: AI-based concept extraction for concepts not in our database

if len(matched\_concepts) < 3: # If we found fewer than 3 concepts

ai\_concepts = self.\_ai\_concept\_extraction(text)

# Merge results, prioritizing direct matches

all\_concepts = {\*\*{c['id']: c for c in ai\_concepts}, \*\*{c['id']: c for c in matched\_concepts}}

matched\_concepts = list(all\_concepts.values())

# Rank concepts by relevance to the text

ranked\_concepts = self.\_rank\_concepts\_by\_relevance(text, matched\_concepts)

return ranked\_concepts

def \_direct\_concept\_matching(self, text):

matched\_concepts = []

normalized\_text = self.\_normalize\_text(text)

# Check for each concept name in the normalized text

for norm\_name, concept in self.concept\_cache.items():

if norm\_name in normalized\_text:

matched\_concepts.append(concept)

return matched\_concepts

def \_ai\_concept\_extraction(self, text):

# Use AI Router to extract philosophical concepts

response = self.ai\_router.route\_request({

'type': 'concept\_extraction',

'text': text,

'max\_concepts': 5

})

extracted\_concepts = []

for concept\_name in response['concepts']:

# Check if concept exists in our database

concept = self.concept\_repository.find\_by\_name(concept\_name)

if concept:

extracted\_concepts.append(concept)

else:

# Log potential new concept for review

self.\_log\_potential\_new\_concept(concept\_name, text)

return extracted\_concepts

def \_rank\_concepts\_by\_relevance(self, text, concepts):

# Calculate relevance score for each concept

scored\_concepts = []

for concept in concepts:

score = self.\_calculate\_relevance\_score(text, concept)

scored\_concepts.append({\*\*concept, 'relevance\_score': score})

# Sort by relevance score (descending)

return sorted(scored\_concepts, key=lambda c: c['relevance\_score'], reverse=True)

def \_calculate\_relevance\_score(self, text, concept):

# Simple relevance calculation based on frequency and position

normalized\_text = self.\_normalize\_text(text)

normalized\_name = concept['normalized\_name']

# Count occurrences

count = normalized\_text.count(normalized\_name)

# Check position (concepts mentioned earlier are more important)

position = normalized\_text.find(normalized\_name)

position\_score = 1.0 if position == -1 else 1.0 - (position / len(normalized\_text))

# Calculate final score

return (count \* 0.7) + (position\_score \* 0.3)

def \_normalize\_text(self, text):

# Convert to lowercase, remove punctuation, etc.

return text.lower()

def \_log\_potential\_new\_concept(self, concept\_name, context):

# Log potential new concept for review and possible addition

self.concept\_repository.log\_potential\_concept(concept\_name, context)

#### 6.2 Concept Mastery Service

class ConceptMasteryService:

def \_\_init\_\_(self, mastery\_repository):

self.mastery\_repository = mastery\_repository

# Define interaction weights for mastery calculation

self.interaction\_weights = {

'view': 1,

'ask\_question': 2,

'journal\_entry': 5,

'quest\_completion': 10,

'forum\_post': 3,

'forum\_reply': 5,

'contribution': 8,

'teach': 15

}

# Define mastery thresholds

self.mastery\_thresholds = [

{'level': 1, 'name': 'Awareness', 'threshold': 0},

{'level': 2, 'name': 'Familiarity', 'threshold': 20},

{'level': 3, 'name': 'Understanding', 'threshold': 40},

{'level': 4, 'name': 'Application', 'threshold': 60},

{'level': 5, 'name': 'Expertise', 'threshold': 80}

]

def get\_user\_mastery(self, user\_id, concept\_id):

# Get user's mastery record for this concept

mastery = self.mastery\_repository.get\_mastery(user\_id, concept\_id)

if not mastery:

# Initialize mastery record if it doesn't exist

mastery = {

'user\_id': user\_id,

'concept\_id': concept\_id,

'mastery\_level': 0,

'exposure\_count': 0,

'interaction\_count': 0

}

self.mastery\_repository.create\_mastery(mastery)

return mastery

def update\_mastery(self, user\_id, concept\_id, interaction\_type, interaction\_data=None):

# Get current mastery

mastery = self.get\_user\_mastery(user\_id, concept\_id)

# Calculate points for this interaction

points = self.\_calculate\_interaction\_points(interaction\_type, interaction\_data)

# Update mastery metrics

mastery['exposure\_count'] += 1

mastery['interaction\_count'] += 1

# Calculate new mastery level (0-100)

current\_points = mastery['mastery\_level']

new\_points = min(100, current\_points + points)

mastery['mastery\_level'] = new\_points

# Update last interaction date

mastery['last\_interaction\_date'] = datetime.now()

# Save updated mastery

self.mastery\_repository.update\_mastery(mastery)

# Check for level-up events

old\_level = self.\_get\_mastery\_level(current\_points)

new\_level = self.\_get\_mastery\_level(new\_points)

if new\_level > old\_level:

self.\_trigger\_mastery\_level\_up(user\_id, concept\_id, new\_level)

return {

'mastery': mastery,

'points\_earned': points,

'level\_up': new\_level > old\_level,

'new\_level': new\_level

}

def \_calculate\_interaction\_points(self, interaction\_type, interaction\_data):

# Get base points for this interaction type

base\_points = self.interaction\_weights.get(interaction\_type, 1)

# Apply modifiers based on interaction data

modifier = 1.0

if interaction\_data:

# Example modifiers:

if interaction\_type == 'journal\_entry' and interaction\_data.get('word\_count', 0) > 500:

modifier = 1.5

elif interaction\_type == 'quest\_completion' and interaction\_data.get('difficulty', 1) > 3:

modifier = 2.0

return base\_points \* modifier

def \_get\_mastery\_level(self, points):

# Convert points (0-100) to level (1-5)

for level in reversed(self.mastery\_thresholds):

if points >= level['threshold']:

return level['level']

return 1 # Default to level 1

def \_trigger\_mastery\_level\_up(self, user\_id, concept\_id, new\_level):

# Get concept details

concept = self.concept\_repository.get\_concept(concept\_id)

# Create notification

notification = {

'user\_id': user\_id,

'type': 'mastery\_level\_up',

'title': f"New mastery level in {concept['name']}!",

'message': f"You've reached {self.\_get\_level\_name(new\_level)} level in {concept['name']}.",

'data': {

'concept\_id': concept\_id,

'concept\_name': concept['name'],

'new\_level': new\_level

}

}

# Send notification

self.notification\_service.send\_notification(notification)

# Award XP

xp\_amount = new\_level \* 25 # 25 XP per level

self.xp\_service.award\_xp(user\_id, xp\_amount, 'concept\_mastery', concept\_id)

# Check for badges

self.\_check\_mastery\_badges(user\_id, new\_level)

def \_get\_level\_name(self, level):

for threshold in self.mastery\_thresholds:

if threshold['level'] == level:

return threshold['name']

return 'Unknown'

def \_check\_mastery\_badges(self, user\_id, new\_level):

# Check for concept mastery badges

if new\_level == 5:

# Count how many concepts the user has mastered

mastered\_count = self.mastery\_repository.count\_mastered\_concepts(user\_id)

# Award badges based on mastery count

if mastered\_count == 1:

self.badge\_service.award\_badge(user\_id, 'first\_concept\_mastered')

elif mastered\_count == 5:

self.badge\_service.award\_badge(user\_id, 'five\_concepts\_mastered')

elif mastered\_count == 10:

self.badge\_service.award\_badge(user\_id, 'ten\_concepts\_mastered')

elif mastered\_count == 25:

self.badge\_service.award\_badge(user\_id, 'concept\_master')

elif mastered\_count == 50:

self.badge\_service.award\_badge(user\_id, 'philosophical\_sage')

#### 6.3 Concept Relationship Service

class ConceptRelationshipService:

def \_\_init\_\_(self, relationship\_repository, concept\_repository):

self.relationship\_repository = relationship\_repository

self.concept\_repository = concept\_repository

def get\_concept\_relationships(self, concept\_id, relationship\_types=None, direction='both'):

# Get relationships for this concept

relationships = self.relationship\_repository.get\_relationships(

concept\_id,

relationship\_types,

direction

)

# Enhance with concept details

enhanced\_relationships = []

for rel in relationships:

source\_concept = self.concept\_repository.get\_concept(rel['source\_concept\_id'])

target\_concept = self.concept\_repository.get\_concept(rel['target\_concept\_id'])

enhanced\_relationships.append({

\*\*rel,

'source\_concept': {

'id': source\_concept['id'],

'name': source\_concept['name'],

'category': source\_concept['parent\_category\_id']

},

'target\_concept': {

'id': target\_concept['id'],

'name': target\_concept['name'],

'category': target\_concept['parent\_category\_id']

}

})

return enhanced\_relationships

def get\_related\_concepts(self, concept\_id, limit=10, min\_strength=0.3):

# Get relationships above minimum strength

relationships = self.relationship\_repository.get\_relationships\_by\_strength(

concept\_id,

min\_strength,

limit

)

# Extract related concept IDs

related\_ids = []

for rel in relationships:

if rel['source\_concept\_id'] == concept\_id:

related\_ids.append(rel['target\_concept\_id'])

else:

related\_ids.append(rel['source\_concept\_id'])

# Get concept details

related\_concepts = []

for concept\_id in related\_ids:

concept = self.concept\_repository.get\_concept(concept\_id)

related\_concepts.append(concept)

return related\_concepts

def suggest\_exploration\_path(self, start\_concept\_id, user\_id, path\_length=5):

# Get user's concept mastery levels

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

# Start with the initial concept

current\_concept\_id = start\_concept\_id

path = [self.concept\_repository.get\_concept(current\_concept\_id)]

# Build path of specified length

for i in range(path\_length - 1):

# Get related concepts

related = self.get\_related\_concepts(current\_concept\_id, limit=20)

# Filter out concepts already in path

related = [c for c in related if c['id'] not in [p['id'] for p in path]]

if not related:

break # No more related concepts

# Score concepts based on relationship strength and user mastery

scored\_concepts = []

for concept in related:

# Get relationship strength

rel = self.relationship\_repository.get\_relationship(

current\_concept\_id,

concept['id']

)

strength = rel['relationship\_strength'] if rel else 0.5

# Get user's mastery of this concept

mastery = user\_mastery.get(concept['id'], 0)

# Calculate score (prefer concepts just beyond current mastery)

mastery\_score = 1.0 - abs(mastery - 30) / 100 # Peak at 30% mastery

# Combined score

score = (strength \* 0.7) + (mastery\_score \* 0.3)

scored\_concepts.append({

\*\*concept,

'score': score

})

# Sort by score and pick the best one

scored\_concepts.sort(key=lambda c: c['score'], reverse=True)

next\_concept = scored\_concepts[0]

# Add to path

path.append(next\_concept)

current\_concept\_id = next\_concept['id']

return path

### 7. Integration with XP and Progression System

#### 7.1 Concept-Based XP Awards

# XP values for concept-related activities

CONCEPT\_XP\_VALUES = {

# Basic interactions

'concept\_view': 2,

'concept\_bookmark': 5,

# Learning interactions

'concept\_quest\_completion': {

'beginner': 15,

'intermediate': 25,

'advanced': 40

},

'concept\_resource\_completion': 10,

# Creation interactions

'concept\_journal\_entry': 15,

'concept\_forum\_post': 10,

'concept\_forum\_reply': 5,

# Mastery interactions

'concept\_mastery\_level\_up': {

'level\_2': 25,

'level\_3': 50,

'level\_4': 75,

'level\_5': 100

},

# Community interactions

'concept\_contribution\_approved': 30,

'concept\_relationship\_suggested': 15,

'concept\_resource\_contributed': 20,

# Special interactions

'concept\_collection\_created': 25,

'concept\_path\_completed': 50,

'daily\_concept\_streak': 10 # per day

}

#### 7.2 Concept-Based Badges

# Concept-related badges

CONCEPT\_BADGES = [

# Exploration badges

{

'id': 'concept\_explorer',

'name': 'Concept Explorer',

'description': 'Viewed 10 different philosophical concepts',

'requirement': {'type': 'concept\_view\_count', 'threshold': 10},

'icon': 'explorer.svg',

'xp\_value': 25

},

{

'id': 'concept\_voyager',

'name': 'Concept Voyager',

'description': 'Viewed 50 different philosophical concepts',

'requirement': {'type': 'concept\_view\_count', 'threshold': 50},

'icon': 'voyager.svg',

'xp\_value': 75

},

# Mastery badges

{

'id': 'first\_concept\_mastered',

'name': 'First Mastery',

'description': 'Reached expertise level in your first concept',

'requirement': {'type': 'concept\_mastery\_count', 'level': 5, 'threshold': 1},

'icon': 'first\_mastery.svg',

'xp\_value': 50

},

{

'id': 'five\_concepts\_mastered',

'name': 'Philosophical Adept',

'description': 'Reached expertise level in 5 concepts',

'requirement': {'type': 'concept\_mastery\_count', 'level': 5, 'threshold': 5},

'icon': 'adept.svg',

'xp\_value': 150

},

{

'id': 'ten\_concepts\_mastered',

'name': 'Philosophical Scholar',

'description': 'Reached expertise level in 10 concepts',

'requirement': {'type': 'concept\_mastery\_count', 'level': 5, 'threshold': 10},

'icon': 'scholar.svg',

'xp\_value': 300

},

# Category badges

{

'id': 'ethics\_explorer',

'name': 'Ethics Explorer',

'description': 'Reached understanding level in 5 ethics concepts',

'requirement': {'type': 'category\_mastery\_count', 'category': 'ethics', 'level': 3, 'threshold': 5},

'icon': 'ethics.svg',

'xp\_value': 100

},

{

'id': 'metaphysics\_explorer',

'name': 'Metaphysics Explorer',

'description': 'Reached understanding level in 5 metaphysics concepts',

'requirement': {'type': 'category\_mastery\_count', 'category': 'metaphysics', 'level': 3, 'threshold': 5},

'icon': 'metaphysics.svg',

'xp\_value': 100

},

# Contribution badges

{

'id': 'concept\_contributor',

'name': 'Concept Contributor',

'description': 'Had 3 concept contributions approved',

'requirement': {'type': 'approved\_contributions\_count', 'threshold': 3},

'icon': 'contributor.svg',

'xp\_value': 75

},

{

'id': 'concept\_curator',

'name': 'Concept Curator',

'description': 'Created 5 concept collections that were saved by other users',

'requirement': {'type': 'saved\_collections\_count', 'threshold': 5},

'icon': 'curator.svg',

'xp\_value': 125

},

# Special badges

{

'id': 'concept\_cartographer',

'name': 'Concept Cartographer',

'description': 'Suggested 10 approved concept relationships',

'requirement': {'type': 'approved\_relationships\_count', 'threshold': 10},

'icon': 'cartographer.svg',

'xp\_value': 150

},

{

'id': 'philosophical\_sage',

'name': 'Philosophical Sage',

'description': 'Reached expertise level in 25 concepts',

'requirement': {'type': 'concept\_mastery\_count', 'level': 5, 'threshold': 25},

'icon': 'sage.svg',

'xp\_value': 500

}

]

### 8. Analytics and Insights

#### 8.1 Concept Analytics Dashboard

class ConceptAnalyticsService:

def get\_concept\_analytics(self, concept\_id):

# Get basic concept info

concept = self.concept\_repository.get\_concept(concept\_id)

# Get interaction metrics

interactions = self.analytics\_repository.get\_concept\_interactions(concept\_id)

# Get user mastery distribution

mastery\_distribution = self.analytics\_repository.get\_mastery\_distribution(concept\_id)

# Get related content metrics

content\_metrics = {

'ask\_responses': self.analytics\_repository.count\_ask\_responses\_with\_concept(concept\_id),

'journal\_entries': self.analytics\_repository.count\_journal\_entries\_with\_concept(concept\_id),

'forum\_posts': self.analytics\_repository.count\_forum\_posts\_with\_concept(concept\_id),

'quests': self.analytics\_repository.count\_quests\_with\_concept(concept\_id)

}

# Get trending metrics

trending = {

'view\_trend': self.analytics\_repository.get\_concept\_view\_trend(concept\_id),

'interaction\_trend': self.analytics\_repository.get\_concept\_interaction\_trend(concept\_id),

'mastery\_trend': self.analytics\_repository.get\_concept\_mastery\_trend(concept\_id)

}

return {

'concept': concept,

'interactions': interactions,

'mastery\_distribution': mastery\_distribution,

'content\_metrics': content\_metrics,

'trending': trending

}

def get\_user\_concept\_analytics(self, user\_id):

# Get user's concept interactions

interactions = self.analytics\_repository.get\_user\_concept\_interactions(user\_id)

# Get user's mastery levels

mastery\_levels = self.mastery\_repository.get\_user\_mastery\_levels(user\_id)

# Get user's concept journey

journey = self.analytics\_repository.get\_user\_concept\_journey(user\_id)

# Get recommendations based on current mastery

recommendations = self.recommendation\_service.get\_concept\_recommendations(user\_id)

return {

'interactions': interactions,

'mastery\_levels': mastery\_levels,

'journey': journey,

'recommendations': recommendations

}

def get\_global\_concept\_analytics(self):

# Get most viewed concepts

most\_viewed = self.analytics\_repository.get\_most\_viewed\_concepts()

# Get most mastered concepts

most\_mastered = self.analytics\_repository.get\_most\_mastered\_concepts()

# Get trending concepts

trending = self.analytics\_repository.get\_trending\_concepts()

# Get concept category distribution

category\_distribution = self.analytics\_repository.get\_concept\_category\_distribution()

return {

'most\_viewed': most\_viewed,

'most\_mastered': most\_mastered,

'trending': trending,

'category\_distribution': category\_distribution

}

#### 8.2 User Insights

class ConceptInsightsService:

def get\_user\_concept\_insights(self, user\_id):

# Get user's mastery data

mastery\_data = self.mastery\_repository.get\_user\_mastery\_data(user\_id)

# Calculate mastery statistics

mastery\_stats = self.\_calculate\_mastery\_statistics(mastery\_data)

# Identify strengths (categories with highest mastery)

strengths = self.\_identify\_strengths(mastery\_data)

# Identify gaps (popular concepts with low mastery)

gaps = self.\_identify\_gaps(user\_id, mastery\_data)

# Generate personalized recommendations

recommendations = self.\_generate\_recommendations(user\_id, mastery\_data, strengths, gaps)

# Get learning patterns

learning\_patterns = self.\_analyze\_learning\_patterns(user\_id)

return {

'mastery\_stats': mastery\_stats,

'strengths': strengths,

'gaps': gaps,

'recommendations': recommendations,

'learning\_patterns': learning\_patterns

}

def \_calculate\_mastery\_statistics(self, mastery\_data):

# Calculate various statistics about user's concept mastery

total\_concepts = len(mastery\_data)

mastery\_levels = [0, 0, 0, 0, 0, 0] # Levels 0-5

for concept in mastery\_data:

level = self.\_get\_mastery\_level(concept['mastery\_level'])

mastery\_levels[level] += 1

avg\_mastery = sum(c['mastery\_level'] for c in mastery\_data) / total\_concepts if total\_concepts > 0 else 0

return {

'total\_concepts': total\_concepts,

'mastery\_levels': mastery\_levels,

'average\_mastery': avg\_mastery,

'mastered\_count': mastery\_levels[5], # Level 5 count

'mastery\_percentage': (mastery\_levels[5] / total\_concepts \* 100) if total\_concepts > 0 else 0

}

def \_identify\_strengths(self, mastery\_data):

# Group concepts by category

categories = {}

for concept in mastery\_data:

category\_id = concept['category\_id']

if category\_id not in categories:

categories[category\_id] = {

'concepts': [],

'total\_mastery': 0,

'count': 0

}

categories[category\_id]['concepts'].append(concept)

categories[category\_id]['total\_mastery'] += concept['mastery\_level']

categories[category\_id]['count'] += 1

# Calculate average mastery per category

for category\_id, data in categories.items():

data['average\_mastery'] = data['total\_mastery'] / data['count'] if data['count'] > 0 else 0

# Get category details

category = self.category\_repository.get\_category(category\_id)

data['category'] = category

# Sort by average mastery (descending)

sorted\_categories = sorted(

categories.values(),

key=lambda c: c['average\_mastery'],

reverse=True

)

# Return top 3 categories

return sorted\_categories[:3]

def \_identify\_gaps(self, user\_id, mastery\_data):

# Get popular concepts (most viewed/discussed)

popular\_concepts = self.analytics\_repository.get\_popular\_concepts(limit=50)

# Create lookup for user's mastery

user\_mastery = {c['concept\_id']: c['mastery\_level'] for c in mastery\_data}

# Find popular concepts with low or no mastery

gaps = []

for concept in popular\_concepts:

mastery = user\_mastery.get(concept['id'], 0)

if mastery < 40: # Less than "Understanding" level

gaps.append({

'concept': concept,

'mastery': mastery,

'popularity\_rank': concept['popularity\_rank']

})

# Sort by popularity (most popular first)

gaps.sort(key=lambda g: g['popularity\_rank'])

# Return top 5 gaps

return gaps[:5]

def \_generate\_recommendations(self, user\_id, mastery\_data, strengths, gaps):

recommendations = []

# 1. Recommend concepts to fill gaps

for gap in gaps[:2]: # Top 2 gaps

recommendations.append({

'type': 'gap',

'concept': gap['concept'],

'reason': f"Popular concept with low mastery ({gap['mastery']}%)"

})

# 2. Recommend concepts related to strengths

if strengths:

top\_category = strengths[0]['category']

related\_concepts = self.concept\_repository.get\_concepts\_by\_category(

top\_category['id'],

limit=5

)

# Filter out concepts user already knows well

user\_mastery = {c['concept\_id']: c['mastery\_level'] for c in mastery\_data}

related\_concepts = [

c for c in related\_concepts

if user\_mastery.get(c['id'], 0) < 60 # Less than "Application" level

]

for concept in related\_concepts[:2]: # Top 2 related concepts

recommendations.append({

'type': 'strength',

'concept': concept,

'reason': f"Related to your strength in {top\_category['name']}"

})

# 3. Recommend concepts based on recent activity

recent\_concepts = self.interaction\_repository.get\_recent\_concept\_interactions(

user\_id,

limit=10

)

for interaction in recent\_concepts[:1]: # Top recent interaction

related = self.relationship\_service.get\_related\_concepts(

interaction['concept\_id'],

limit=3

)

# Filter out concepts user already knows well

user\_mastery = {c['concept\_id']: c['mastery\_level'] for c in mastery\_data}

related = [

c for c in related

if user\_mastery.get(c['id'], 0) < 60 # Less than "Application" level

]

if related:

concept = related[0]

source\_concept = self.concept\_repository.get\_concept(interaction['concept\_id'])

recommendations.append({

'type': 'recent',

'concept': concept,

'reason': f"Related to your recent exploration of {source\_concept['name']}"

})

return recommendations

def \_analyze\_learning\_patterns(self, user\_id):

# Get user's concept interaction history

interactions = self.interaction\_repository.get\_user\_concept\_interactions(

user\_id,

limit=100

)

# Analyze interaction patterns

feature\_distribution = {}

time\_distribution = {'morning': 0, 'afternoon': 0, 'evening': 0, 'night': 0}

session\_lengths = []

current\_session = {'start': None, 'end': None, 'concepts': set()}

for interaction in interactions:

# Feature distribution

feature = interaction['source\_feature']

if feature not in feature\_distribution:

feature\_distribution[feature] = 0

feature\_distribution[feature] += 1

# Time distribution

hour = interaction['created\_at'].hour

if 5 <= hour < 12:

time\_distribution['morning'] += 1

elif 12 <= hour < 17:

time\_distribution['afternoon'] += 1

elif 17 <= hour < 22:

time\_distribution['evening'] += 1

else:

time\_distribution['night'] += 1

# Session analysis

if current\_session['start'] is None:

# Start new session

current\_session['start'] = interaction['created\_at']

current\_session['concepts'].add(interaction['concept\_id'])

elif (interaction['created\_at'] - current\_session['end']).total\_seconds() > 1800:

# Gap of more than 30 minutes, end session and start new one

session\_lengths.append({

'duration': (current\_session['end'] - current\_session['start']).total\_seconds() / 60,

'concepts': len(current\_session['concepts'])

})

current\_session = {

'start': interaction['created\_at'],

'end': interaction['created\_at'],

'concepts': {interaction['concept\_id']}

}

else:

# Continue session

current\_session['end'] = interaction['created\_at']

current\_session['concepts'].add(interaction['concept\_id'])

# Calculate average session length and concepts per session

avg\_session\_length = sum(s['duration'] for s in session\_lengths) / len(session\_lengths) if session\_lengths else 0

avg\_concepts\_per\_session = sum(s['concepts'] for s in session\_lengths) / len(session\_lengths) if session\_lengths else 0

return {

'feature\_distribution': feature\_distribution,

'time\_distribution': time\_distribution,

'avg\_session\_length': avg\_session\_length,

'avg\_concepts\_per\_session': avg\_concepts\_per\_session

}

def \_get\_mastery\_level(self, points):

# Convert points (0-100) to level (0-5)

thresholds = [0, 20, 40, 60, 80]

for i, threshold in enumerate(thresholds):

if points < threshold:

return i

return 5 # Level 5 (80-100)

### 9. Implementation Plan

#### 9.1 Phase 1: Core Concept System (4 weeks)

1. **Week 1: Data Model and API**
   * Implement core concept data models
   * Create basic API endpoints
   * Set up concept repository
2. **Week 2: Concept Extraction and Mastery**
   * Implement concept extraction service
   * Develop mastery tracking system
   * Integrate with AI Router
3. **Week 3: Concept Detail View**
   * Create concept detail UI
   * Implement relationship visualization
   * Develop concept resources section
4. **Week 4: Basic Integration**
   * Integrate with Ask feature (concept badges)
   * Integrate with Journal (concept tagging)
   * Implement basic XP rewards

#### 9.2 Phase 2: Concept Map and Exploration (3 weeks)

1. **Week 1: Concept Map Visualization**
   * Develop force-directed graph visualization
   * Implement interactive controls
   * Create personalized view options
2. **Week 2: Exploration Features**
   * Implement concept journey paths
   * Develop discovery features
   * Create concept collections
3. **Week 3: User Experience Refinement**
   * Optimize performance for large concept maps
   * Implement progressive loading
   * Add animations and transitions

#### 9.3 Phase 3: Deep Feature Integration (4 weeks)

1. **Week 1: Quest Integration**
   * Implement concept-based quests
   * Develop quest generation system
   * Create concept progression paths
2. **Week 2: Forum Integration**
   * Create concept forums
   * Implement concept expert recognition
   * Develop concept resource sharing
3. **Week 3: Seek Clarity Integration**
   * Enhance pathways with concept awareness
   * Implement concept-based challenges
   * Create concept learning paths
4. **Week 4: Analytics and Insights**
   * Develop concept analytics dashboard
   * Implement user insights
   * Create recommendation engine

#### 9.4 Phase 4: Community and Sharing (3 weeks)

1. **Week 1: Contribution System**
   * Implement concept contribution workflow
   * Create review and approval process
   * Develop contribution rewards
2. **Week 2: Sharing Features**
   * Implement concept sharing options
   * Create shareable concept cards
   * Develop privacy controls
3. **Week 3: Community Features**
   * Implement concept experts system
   * Create community-curated collections
   * Develop concept leaderboards

### 10. Success Metrics

#### 10.1 Engagement Metrics

* **Concept Exploration Rate**: Average number of concepts explored per user per week
* **Concept Return Rate**: Percentage of users who return to previously viewed concepts
* **Concept Session Length**: Average time spent exploring concepts per session
* **Concept Map Usage**: Percentage of users who use the concept map feature
* **Cross-Feature Navigation**: Rate of navigation between features via concept links

#### 10.2 Learning Metrics

* **Concept Mastery Growth**: Average increase in concept mastery levels per month
* **Concept Coverage**: Percentage of total concepts with which users have interacted
* **Learning Path Completion**: Percentage of started concept learning paths that are completed
* **Quest Completion Rate**: Percentage of concept-based quests that are completed
* **Concept Challenge Completion**: Percentage of concept challenges completed

#### 10.3 Community Metrics

* **Contribution Rate**: Number of concept contributions per active user
* **Contribution Quality**: Approval rate of user contributions
* **Concept Sharing**: Average number of concepts shared per user
* **Forum Engagement**: Activity in concept-specific forum discussions
* **Collection Creation**: Number of user-created concept collections

#### 10.4 Technical Metrics

* **Concept Extraction Accuracy**: Percentage of correctly identified concepts in content
* **Concept Map Performance**: Load time and interaction responsiveness
* **API Response Time**: Average response time for concept-related API endpoints
* **System Scalability**: Performance with growing concept database
* **Cross-Feature Integration**: Successful integration rate across all app features