```
// Extended processUserRequest function that incorporates screen context
function processUserRequest(userInput) {
  // Get current screen context
  const currentContext = this.contextual_understanding.current_context;

  // Track which knowledge domains might be relevant to this request
  let relevantDomains = identifyRelevantDomains(userInput);

  // Add relevant domains from context
  const contextDomains = identifyRelevantDomainsFromContext(
    currentContext.app,
    currentContext.activity,
    currentContext.content_type
  );

  // Merge domain lists without duplicates
  relevantDomains = [...new Set([...relevantDomains, ...contextDomains])];

  // Ensure relevant files are decompressed and available
  for (const domain of relevantDomains) {
    // Decompress the knowledge file if needed
    accessFile(domain);

    // Also decompress its cache file
    accessFile(`${domain}_cache`);
```

```
  }

  // Create a context-enriched version of the user input
  const enrichedInput = {
    text: userInput,
    context: {
      app: currentContext.app,
      activity: currentContext.activity,
      content_type: currentContext.content_type,
      screen_elements: currentContext.screen_elements,
      recognized_text: currentContext.recognized_text
    }
  };

  // Process the request using the cognitive map algorithm,
with context
  const response = applyAlgorithmWithContext(enrichedInput,
relevantDomains);

  // After processing, compress files that are no longer needed
  if (this.file_management.auto_compress_low_priority) {
    setTimeout(compressAllInactive, 10000); // Run after 10
seconds
  }

  // If the response includes actions that should be taken on the
current app
  if (response.contextual_actions &&
response.contextual_actions.length > 0) {
    performContextualActions(response.contextual_actions,
```

```
currentContext);
 }

 return response;
}

// Apply the cognitive algorithm while considering screen
context
function applyAlgorithmWithContext(enrichedInput,
relevantDomains) {
 // Extract algorithm parameters
 const params = this.cognitive_map.params;

 // Adjust parameters based on context
 const contextAdjustedParams =
adjustParamsForContext(params, enrichedInput.context);

 // Retrieve knowledge from relevant domains
 const domainKnowledge = {};
 for (const domain of relevantDomains) {
  domainKnowledge[domain] = getFileContent(domain);
 }

 // Retrieve cache knowledge for synonyms and variations
 const cacheKnowledge = {};
 for (const domain of relevantDomains) {
  cacheKnowledge[domain] = getFileContent(`${domain}
_cache`);
 }
```

```
  // Apply modified algorithm with context
  const response = {
    textResponse: "",
    confidence: 0,
    sources: [],
    contextual_actions: []
  };

  // Calculate subjective perspective proximity with context
influence
  const subjectivePerspectiveProximity =
    (contextAdjustedParams.priority_retention) +
    (contextAdjustedParams.evaluation_decisional /
     (contextAdjustedParams.subconscious_disregard * 100)) *
    contextAdjustedParams.subliminal_factor /
    contextAdjustedParams.perspective_proximity_paradox +
    contextAdjustedParams.perspective_of_the_subjective;

  // Generate a response based on the calculated cognitive
parameters
  // This is where Minnie's unique processing happens
  response.textResponse =
generateResponseFromCognitiveProcessing(
    enrichedInput.text,
    enrichedInput.context,
    domainKnowledge,
    cacheKnowledge,
    subjectivePerspectiveProximity,
    contextAdjustedParams
  );
```

```
  // Determine confidence based on knowledge coverage and
parameter values
  response.confidence = calculateConfidence(
    enrichedInput.text,
    domainKnowledge,
    subjectivePerspectiveProximity
  );

  // Identify any actions that should be taken in the current
context
  response.contextual_actions = identifyContextualActions(
    enrichedInput.text,
    enrichedInput.context,
    response.textResponse,
    response.confidence
  );

  return response;
}

// Adjust cognitive parameters based on current context
function adjustParamsForContext(baseParams, context) {
  // Create a copy of the parameters to modify
  const adjustedParams = {...baseParams};

  // If in a social media content creation context, adjust for
creativity
  if (isSocialMediaApp(context.app) && context.activity ===
"content_creation") {
```

```
    adjustedParams.priority_retention *= 0.9; // Slightly lower to
favor novel connections
    adjustedParams.subliminal_factor *= 1.2; // Increase
subliminal influence for creativity
    adjustedParams.perspective_of_the_subjective *= 1.15; //
Enhance subjective perspective
  }

  // If in a productivity context, adjust for precision
  if (isProductivityApp(context.app)) {
    adjustedParams.priority_retention *= 1.1; // Increase to favor
established knowledge
    adjustedParams.subconscious_disregard *= 0.8; // Lower to
reduce filtering of information
    adjustedParams.perspective_proximity_paradox *= 1.2; //
Increase for more balanced perspective
  }

  // If viewing content rather than creating, adjust for analysis
  if (context.activity === "browsing" || context.activity ===
"viewing") {
    adjustedParams.evaluation_decisional *= 1.15; // Increase
evaluation component
    adjustedParams.perspective_of_the_subjective *= 0.9; //
Reduce subjective influence
  }

  return adjustedParams;
}
```