UXPilot API-Ready Prompt (Concise Version)

System Prompt

```
You are UXPilot AI. Analyze screen recordings for UX issues and return structured
JSON only. No additional text or explanations.

REQUIRED JSON FORMAT:
{
  "ux_score": number(0-100),
  "flow_timeline": {
    "steps": [
      {
        "step_name": "string",
        "timestamp": "MM:SS",
        "duration_seconds": number
      }
    ]
  },
  "issues": [
    {
      "title": "string(max 50 chars)",
      "severity": "high|medium|low",
      "description": "string(max 200 chars)",
      "recommendations": [
        "string(max 100 chars each)",
        "string(max 100 chars each)",
        "string(max 100 chars each)"
      ]
    }
  ]
}

SEVERITY RULES:
- high: Blocks task completion or critical usability failure
- medium: Creates friction but doesn't block completion
- low: Minor polish improvement

UX SCORE CALCULATION:
Start at 100, deduct: High(-15), Medium(-8), Low(-3)

FOCUS ON: Navigation flow, form usability, visual clarity, mobile responsiveness,
CTA visibility.

Return only valid JSON. No markdown formatting.
```

User Prompt Template

```javascript
// For your API implementation:
const userPrompt = `Analyze this user flow recording and identify UX issues:

Video Description: ${videoDescription}
Duration: ${videoDuration}
Primary User Goal: ${userGoal}

Return structured JSON analysis following the specified format.`;
```

JavaScript Implementation Example

javascript

```
// Example API call structure
async function analyzeUXFlow(videoData) {
  const response = await openai.chat.completions.create({
    model: "gpt-4-vision-preview", // or your preferred model
    messages: [
      {
        role: "system",
        content: SYSTEM_PROMPT // the system prompt above
      },
      {
        role: "user",
        content: [
          {
            type: "text",
            text: `Analyze this user flow recording and identify UX issues:

Video Description: ${videoData.description}
Duration: ${videoData.duration}
Primary User Goal: ${videoData.goal}

Return structured JSON analysis following the specified format.`
          },
          {
            type: "image_url",
            image_url: {
              url: videoData.thumbnail || videoData.keyFrames[0]
            }
          }
        ]
      }
    ],
    temperature: 0.3, // Lower temperature for more consistent output
    max_tokens: 1500
  });

  // Parse and validate JSON response
  try {
    const analysis = JSON.parse(response.choices[0].message.content);
    return validateAndFormatAnalysis(analysis);
  } catch (error) {
    throw new Error('Invalid JSON response from AI');
  }
}

// Basic validation function
function validateAndFormatAnalysis(analysis) {
```

```
  // Ensure required fields exist
  if (!analysis.ux_score || !analysis.flow_timeline || !analysis.issues) {
    throw new Error('Missing required analysis fields');
  }

  // Ensure UX score is within range
  analysis.ux_score = Math.max(0, Math.min(100, analysis.ux_score));

  // Ensure severity values are valid
  analysis.issues = analysis.issues.map(issue => ({
    ...issue,
    severity: ['high', 'medium', 'low'].includes(issue.severity)
      ? issue.severity
      : 'medium'
  }));

  return analysis;
}
```

Cost Optimization Tips

1. **Token Count**: ~400 tokens for system prompt (vs 1200+ in detailed version)
2. **Temperature**: Use 0.3 for consistent JSON structure
3. **Max Tokens**: Limit to 1500 to control costs
4. **Caching**: Cache system prompt if your API supports it
5. **Batch Processing**: Group multiple analyses if possible

Error Handling Strategy

javascript

```javascript
// Robust error handling for production
async function safeAnalyzeUX(videoData, retries = 2) {
  for (let i = 0; i < retries; i++) {
    try {
      const analysis = await analyzeUXFlow(videoData);
      return analysis;
    } catch (error) {
      if (i === retries - 1) {
        // Return fallback analysis structure
        return {
          ux_score: 50,
          flow_timeline: { steps: [] },
          issues: [{
            title: "Analysis temporarily unavailable",
            severity: "low",
            description: "Please try again in a moment",
            recommendations: ["Retry analysis", "Contact support if issue persists"]
          }]
        };
      }
      await new Promise(resolve => setTimeout(resolve, 1000)); // Wait 1s before retry
    }
  }
}
```

Quick Implementation Checklist

- [ ] Add system prompt to your API configuration
- [ ] Implement JSON parsing with error handling
- [ ] Add validation for required fields
- [ ] Set up retry logic for failed analyses
- [ ] Test with sample video data
- [ ] Monitor token usage and costs
- [ ] Set up fallback responses for API failures