Code at the Speed of Thought: Quick Reference

Workshop: Google Stockholm, January 28, 2026 **Cheatsheet Version:** 1.0

---

1. Quick Start (5 minutes) — Most Essential

Basic Gemini Prompt

```
import { GoogleGenerativeAI } from "@google/generative-ai";

const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
const model = genAI.getGenerativeModel({ model: "gemini-2.0-
        flash" });

const result = await model.generateContent("Your prompt here");
console.log(result.response.text());
```

Covered in: Module 01 (AI Studio Exploration)

API Key Setup

- **Get free key:** aistudio.google.com → "Get API Key"
- **Environment variable:** `GEMINI_API_KEY=your_key_here` in `.env` file
- **Browser usage:** Store in variable, never commit to git
- **Free tier:** 15 requests per minute (RPM), 1500 requests per day (RPD)

---

2. Common Tasks — Use Often During Exercises

JSON Structured Output

```
const schema = {
  type: "object",
  properties: {
    emotion: {
      type: "string",
      enum: ["happy", "sad", "surprised", "angry", "calm"],
      description: "Detected emotion from facial expression"
    },
    confidence: {
      type: "number",
      minimum: 0,
      maximum: 1,
      description: "Confidence score for the emotion detection"
    }
  },
  required: ["emotion", "confidence"]
};

const model = genAI.getGenerativeModel({
  model: "gemini-2.0-flash",
  generationConfig: {
    responseMimeType: "application/json",
    responseSchema: schema
  }
});

const result = await model.generateContent("Analyze this
        expression: smiling");
const data = JSON.parse(result.response.text());
```

Key insight: Description fields act as model instructions, not just documentation **Covered in:** Module 02 (Structured Output) **Apply in:** part2/face-reactive/ (emotion detection returns JSON)

Image Analysis (Multimodal)

```
// Option 1: Inline image data
const imagePart = {
  inlineData: {
    data: base64ImageString,  // Base64-encoded image
    mimeType: "image/png"
  }
};

// Option 2: Using File API (recommended for production)
const imagePart = {
  fileData: {
    fileUri: "https://generativelanguage.googleapis.com/v1/
        files/...",
    mimeType: "image/png"
  }
};

const result = await model.generateContent([
  "Describe this image in detail",
  imagePart
]);
```

Token cost: Images $\leq$ 384px = ~258 tokens (resize for cost efficiency) **Covered in:** Module 03 (Multimodal Input)

Context Engineering (Few-Shot Examples)

```
const prompt = `
<system>
You are an expert at analyzing facial expressions and mapping them
        to emotions.
Provide concise, accurate emotion classifications.
```

```
</system>

<examples>
Input: "Corners of mouth raised, cheeks lifted"
Output: {"emotion": "happy", "confidence": 0.95}

Input: "Eyebrows raised, eyes wide, mouth slightly open"
Output: {"emotion": "surprised", "confidence": 0.88}

Input: "Mouth corners down, eyebrows lowered"
Output: {"emotion": "sad", "confidence": 0.72}
</examples>

<task>
Input: "${userInput}"
Output:
</task>
`;
```

Optimal: 2-3 examples (diminishing returns beyond this) **Covered in:** Module 04 (Context Engineering)

---

3. MediaPipe & Canvas — Part 2 Essentials

MediaPipe Face Detection

```
import { FaceLandmarker, FilesetResolver } from
  "https://cdn.jsdelivr.net/npm/@mediapipe/tasks-vision@latest";

// Initialize MediaPipe vision tasks
const vision = await FilesetResolver.forVisionTasks(
  "https://cdn.jsdelivr.net/npm/@mediapipe/tasks-vision@latest/
        wasm"
);

// Create Face Landmarker
const faceLandmarker = await
        FaceLandmarker.createFromOptions(vision, {
  baseOptions: {
    modelAssetPath: "https://storage.googleapis.com/mediapipe-
        models/face_landmarker/face_landmarker/float16/1/
        face_landmarker.task",
    delegate: "GPU"  // Use GPU acceleration (60fps on modern
        hardware)
  },
  runningMode: "VIDEO",
  numFaces: 1,
  outputFaceBlendshapes: true  // Enable 52 ARKit blendshapes
});

// Process video frame
const result = faceLandmarker.detectForVideo(videoElement,
        performance.now());
const blendshapes = result.faceBlendshapes[0].categories;
```

```
// Get specific blendshape score
function getBlendshapeScore(blendshapes, name) {
  const shape = blendshapes.find(b => b.categoryName === name);
  return shape ? shape.score : 0;
}

// Example: Detect smile
const smileScore = getBlendshapeScore(blendshapes,
        'mouthSmileLeft');
if (smileScore > 0.5) {
  console.log("Happy expression detected!");
}
```

Performance: 30fps on CPU, 60fps on GPU **Apply in:** part2/face-reactive/ (emotion detection from blendshapes)

Canvas 2D Particle System

```
const canvas = document.getElementById('canvas');
const ctx = canvas.getContext('2d');

// Object pooling for performance (reuse particles, avoid GC
        pauses)
const particles = Array(150).fill(null).map(() => ({
  x: Math.random() * canvas.width,
  y: Math.random() * canvas.height,
  vx: (Math.random() - 0.5) * 2,
  vy: (Math.random() - 0.5) * 2,
  size: 4,
  color: '#FFD700',
  active: true
}));

// Animation loop
function animate() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  particles.forEach(p => {
    if (!p.active) return;

    // Update position
    p.x += p.vx;
    p.y += p.vy;

    // Edge wrapping
    if (p.x < 0) p.x = canvas.width;
    if (p.x > canvas.width) p.x = 0;
    if (p.y < 0) p.y = canvas.height;
    if (p.y > canvas.height) p.y = 0;

    // Render
    ctx.fillStyle = p.color;
    ctx.beginPath();
    ctx.arc(p.x, p.y, p.size, 0, Math.PI * 2);
    ctx.fill();
  });

  requestAnimationFrame(animate);
}

animate();
```

Performance tip: 150 particles = smooth on most devices, 500+ requires GPU
Apply in: part2/face-reactive/ (emotion-driven visualization)

---

4. Firebase Realtime Database — Multiplayer

Initialize Firebase (Local Emulator)

```
import { initializeApp } from 'firebase/app';
import { getDatabase, ref, set, onValue } from 'firebase/database';

const app = initializeApp({
  databaseURL: "http://localhost:9000?ns=demo-project"  // Local
        emulator
});

const db = getDatabase(app);
```

Workshop setup: Firebase Local Emulator (no internet, no production DB)

Write Data

```
const sessionRef = ref(db, `sessions/${sessionId}/players/$
        {playerId}`);

await set(sessionRef, {
  name: playerName,
  score: 0,
  timestamp: Date.now()
});
```

Pattern: Last-write-wins (simpler than transactions, teaches real-world tradeoffs)

Real-time Sync

```
const playersRef = ref(db, `sessions/${sessionId}/players`);

onValue(playersRef, (snapshot) => {
  const players = snapshot.val();

  // Update UI with player data
  Object.entries(players || {}).forEach(([id, player]) => {
    updateScoreboard(player.name, player.score);
  });
});
```

Apply in: part2/camera-game/ (multiplayer state sync)

5. Advanced Techniques — Optional

Grounding with Google Search

```
const model = genAI.getGenerativeModel({
  model: "gemini-2.0-flash",
  tools: [{
    googleSearch: {}  // Enable grounding with Google Search
  }]
});

const result = await model.generateContent(
  "What are the latest developments in AI from Google?"
);

// Access grounding metadata (sources, citations)
const metadata = result.response.groundingMetadata;
console.log("Sources:", metadata.searchEntryPoints);
```

AI Studio equivalent: Toggle "Grounding" on in Tools panel **Covered in:** Module 05 (Grounding with Search) **Use case:** Current events, recent facts, real-time information

System Instructions

```
const model = genAI.getGenerativeModel({
  model: "gemini-2.0-flash",
  systemInstruction: "You are a helpful code review assistant.
        Provide concise, actionable feedback on code quality,
        readability, and best practices."
});
```

Applies to all prompts in the same model instance **Covered in:** Module 04 (Context Engineering)

6. Troubleshooting — When Stuck

Common Errors

Error	Cause	Solution
`API_KEY_INVALID`	Missing or incorrect API key	Check `.env` file exists and has `GEMINI_API_KEY=your_key`
`Rate limit exceeded`	Too many requests	Free tier = 15 RPM. Wait 1 minute between bursts
`Image too large`	Image >20MB or wrong format	Resize to ≤ 384px width, use PNG/JPEG/WEBP
`JSON parse error`	Schema mismatch	Verify `responseSchema` matches actual model output

Error	Cause	Solution
`Camera permission denied`	HTTPS required or permission blocked	Use `localhost` or HTTPS. Check browser settings
`Firebase PERMISSION_DENIED`	Emulator rules misconfigured	Check `firebase.json` security rules allow read/write
`MediaPipe model 404`	CDN issue or network block	Check internet connection, verify CDN URLs

Performance Tips

- **Particle count:** 150 = smooth on most devices, 500+ requires dedicated GPU
- **MediaPipe delegate:** `"GPU"` for 60fps, `"CPU"` for 30fps fallback
- **Firebase writes:** Use `set()` for updates, not individual `push()` calls
- **Image optimization:** ≤ 384px width = ~258 tokens, reduces cost and latency
- **Blendshape thresholds:** 0.5 works well, but may need per-person calibration

Key Blendshapes for Emotions

Emotion	Primary Blendshapes	Threshold
Happy	`mouthSmileLeft`, `mouthSmileRight`	> 0.5
Sad	`mouthFrownLeft`, `mouthFrownRight`	> 0.5
Surprised	`browInnerUp`, `eyeWideLeft`, `eyeWideRight`	> 0.5
Angry	`browDownLeft`, `browDownRight`	> 0.5
Excited	Smile + `jawOpen`	> 0.4

Apply in: part2/face-reactive/src/emotionMapping.js

Resources

- **Gemini API Docs:** ai.google.dev/gemini-api/docs
- **MediaPipe Vision:** ai.google.dev/edge/mediapipe/solutions/vision
- **Firebase Realtime Database:** firebase.google.com/docs/database
- **AI Studio:** aistudio.google.com
- **Workshop Repo:** [GitHub link provided by workshop organizer]

Quick Links by Module: - Module 01: AI Studio basics, freeform prompts - Module 02: Structured output, JSON schemas - Module 03: Multimodal input, image analysis - Module 04: Context engineering, few-shot examples, system instructions - Module 05: Grounding with Google Search - Module 06: Logic engines, vibe coding patterns

Quick Links by Project: - Face-Reactive: MediaPipe + Canvas + emotion detection - Camera Game: QR scanning + Firebase + multiplayer sync - Custom Project: Architecture guide + helper modules