AI Native Dev

Week 1

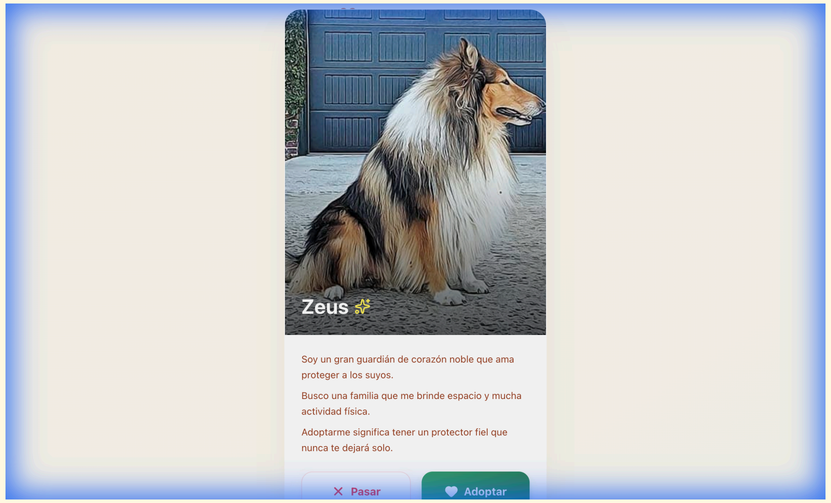
Adrián Catalan

adriancatalan@galileo.edu

Introduction: PawsMatch

PawsMatch is a **pet adoption application** designed to connect loving families with pets in need.

- **Objective**: Tinder-like interface for adopting pets.
- **Tech Stack**: React, Vite, Tailwind CSS, Framer Motion.
- **UX**: Premium, mobile-first, glassmorphism design.



Soy un gran guardián de corazón noble que ama proteger a los suyos.

Busco una familia que me brinde espacio y mucha actividad física.

Adoptarme significa tener un protector fiel que nunca te dejará solo.

Demo

Zero-latency swiping implementation with pre-fetch buffer.

What does "AI Native" mean?

It represents a **paradigm shift** in how we build software.

- **Old Way**: You write code -> You Google errors/use Stack Overflow -> You write tests.
- **AI Native**: You **orchestrate** models to write code, debug, and test *for* you.

You cease to be just a "coder" and become an "Apply-er" of Intelligence.

Why Now?

We are in the **Intelligence Era**.

1. **Speed**: Agents can scaffold entire apps in minutes.
2. **Complexity**: Models can reason through architecture better than tired humans.
3. **Future-Proofing**: The developer who ignores AI will be replaced by the developer who masters it.

The goal of this course is to make you that master.

Topic 1: Terminology

Under the Hood of LLMs

1. Large Language Model (LLM)

An **LLM** is a probabilistic system trained to predict the next token in a sequence. It does not "know" facts; it understands statistical correlations between words.

> "It's not a search engine, it's a reasoning engine."

- **Architecture**: Transformer (Attention Is All You Need, 2017).
- **Scale**: Billions of parameters (weights) learned from massive datasets.

LLM Flow Diagram

How an LLM processes your input:

```
+------------+       +-----------+       +--------------+
| Input Text | ----> | Token IDs | ----> | Model Layers |
+------------+       +-----------+       +-------+------+
(Tokenization)       (Processing)                |
                                                 | (Probabilities)
                                                 v
                     +--------------+      +----------+
                     | Output Token | <--- |  Logits  |
                     +--------------+      +----------+
                        (Decoding)
```

2. Transformers: The Engine

Transformers allow models to process entire sequences of data in parallel, rather than sequentially.

Key Innovation: Self-Attention
The model assigns a "relevance score" to every word's relationship with every other word.

Example: In "The animal didn't cross the street because **it** was too tired", the model knows "**it**" refers to "animal", not "street".

Transformer Architecture

```
      ENCODER                          DECODER
+------------------+        +-------------------+
| Input Embeddings |        | Output Embeddings |
+--------+---------+        +---------+---------+
         |                            |
         v                            v
+--------+---------+        +---------+---------+
|  Self-Attention  |        |  Masked Attention |
+--------+---------+        +---------+---------+
         |                            |
         v                            v
+--------+---------+        +---------+---------+
|   Feed Forward   |-------->|  Cross-Attention  |
+------------------+        +---------+---------+
                                      |
                                      v
                            +---------+---------+
                            |    Feed Forward   |
                            +-------------------+
```

3. Tokens: The Atoms of Language

LLMs don't read words/letters; they read **tokens**.
A token can be a word, part of a word, or a space.

- **English**: ~0.75 words per token.
- **Code**: High efficiency (keywords usually single tokens).

Example:

```
"Ingeniería" -> [Ingen, iería] (2 tokens)
"import" -> [import] (1 token)
```

4. Context Window (Working Memory)

The **Context Window** is the maximum amount of text (input + output) the model can process at once.

If it falls out of the window, the model forgets it. The industry is racing towards "infinite context".

Model	Context Window
Gemini 3 Pro	10M Tokens
Claude Opus 4.5	5M Tokens
GPT-5.2	2M Tokens

Topic 2: Tech Stack

The AI Native Toolchain

Primary Tools

1. Claude Code (Anthropic)

- **Role**: The "Deep Thinker".
- **Strengths**: Complex reasoning, large context window (200k+), architectural planning.
- **Usage**: Terminology understanding, initial planning, refactoring logic (ReAct loops).

2. Antigravity (Google DeepMind)

- **Role**: The "Agentic Engineer".
- **Strengths**: Tool usage, file manipulation, deep context integration, creating artifacts.
- **Usage**: Execution, file creation, systematic modifications.

Other Tools

3. Cursor

- **Role**: The "IDE".
- **Strengths**: Inline editing (Cmd+K), codebase chat (Cmd+L), context-aware auto-complete.
- **Usage**: Quick edits, reviewing AI changes.

4. OpenCode

- **Role**: Technical Utility.
- **Strengths**: Direct API interaction, quick prototyping without IDE overhead.
- **Usage**: Testing generated assets, lightweight scripting.

Choosing the Right Tool

The "AI Native" developer is not defined by a single tool, but by their ability to **orchestrate** them.

- **Deep Reasoning**: Use **Claude Code** when you need architectural planning or complex refactoring.
- **Heavy Lifting**: Use **Antigravity** for file creation, systematic changes, and shell interactions.
- **Speed & Polish**: Use **Cursor** for visual fixes, and UI verification.

The workflow is flexible—adapt it to your immediate needs.

The Workflow

Flexible and non-linear. Use the best tool for the current task.

```
   +-----------------+
   |  Idea / Prompt  |
   +--------+--------+
            |
+-----------v-----------+
|    Reasoning Phase    | <---(Claude Code)
+-----------+-----------+
            |
+-----------v-----------+
|    Execution Phase    | <---(Antigravity / Cursor)
+-----------+-----------+
            |
+-----------v-----------+
|    Final Codebase     |
+-----------------------+
```

Topic 3: Basic Prompting

Strategies & Prompts

Context Engineering

Prompt Engineering: Writing the best letter to get a specific job done (Micro).

Context Engineering: Designing the entire office environment, tools, and filing system so the worker succeeds every time (Macro).

It involves curating the "System Prompt", identifying necessary files, and managing the "State" of the conversation.

Tool Calling

Tool Calling is the mechanism that allows an LLM to "act" in the real world.

1. **Intent**: The model recognizes it needs external info (not in its weights).
2. **Output**: Instead of text, it outputs a structured command (e.g., JSON).
3. **Execution**: The system runs the command (e.g., `read_file`, `search_web`).
4. **Feedback**: The result is fed back to the model as context.

Technique 1: Zero-Shot

Theory: Asking the model to perform a task without any prior examples.

- **Pros**: Fastest to write, lowest token cost.
- **Cons**: Unpredictable output format. The model guesses the structure you want.
- **Best for**: Simple questions, creative writing, or when you rely on the model's default training.

Example: One Shot + Tool Calling

(Phase 1 Prompt)

> “We are building a web app called PawsMatch... Use uv to initialize a new Python project inside a folder named asset-generation/. Install google-genai and create a script named main.py. This script must use gemini-3-flash-preview to generate a pets.json file containing 50 dog profiles... Create a README.md... Verify the existence of both files using terminal tools and run the script.”

Note the tool calling instruction: "Verify the existence..."

Technique 2: Few-Shot (K-Shot)

Theory: Providing K examples (shots) of "Input -> Output" pairs to guide the model.

- **Pros**: Drastically improves adherence to specific formats or styles (e.g., JSON structure, Coding Style).
- **Cons**: Consumes more context window.
- **Best for**: API payload generation, enforcing coding standards.

Example: K-Shot

(Phase 2 Prompt)

> “I want to define the coding standards for PawsMatch. Here are 3 examples of how I structure services and types in my projects:
>
> **Example 1 (Types)**: `export interface User { id: string... }`
> **Example 2 (Service)**: `export const fetchUsers = async ()...`
> **Example 3 (Hook)**: `export const useAuth = ()...`
>
> Based on these examples and the @pets.json, generate pet.ts file...”

Technique 3: Context-Augmented

Theory: Providing external knowledge or documentation before asking for generation.

- **Pros**: Reduces hallucinations, ensures use of correct/latest API versions.
- **Cons**: Requires retrieving the context first.
- **Best for**: Using new libraries (breaking changes) or private documentation.

Example: One-Shot + Context

(Phase 3 Prompt)

> “Using context7, fetch the latest stable documentation for Vite, React, and Tailwind CSS. Initialize a project named PawsMatch inside a new folder called "app/" using these versions.
>
> Follow this project structure: `src/components`, `src/hooks`…
>
> Make sure the entry point displays a clean, centered layout…”

Technique 4: Chain of Thought (CoT)

Theory: Instructing the model to "think step-by-step" or produce high-level reasoning before outputting the final answer.

- **Pros**: Increases accuracy on complex logic/math problems.
- **Cons**: Slower, more verbose.
- **Best for**: Architecture analysis, algorithm design, debugging.

Example: CoT + Context

(Phase 4 Prompt)

> "Search the web for the official documentation for The Dog API... Before coding, perform an analysis to:
>
> 1. Read the local pets.json...
> 2. Identify the exact JSON path...
> 3. Detail how to fetch...
> 4. Explain the TypeScript interface...
>
> Do not write the code yet; provide the analysis first and wait for my confirmation."

Technique 5: Tree of Thoughts (ToT)

Theory: Exploring multiple possible solution paths (branches), evaluating them, and selecting the best one.

- **Pros**: Avoids "tunnel vision" on the first probable token.
- **Cons**: Very high token cost (requires multiple generations).
- **Best for**: Critical architectural decisions, trade-off analysis.

Example: ToT

(Phase 5 Prompt)

> “Explore three technical approaches for the card stack behavior:
>
> - **Simple state**: Only the current pet is in memory.
> - **Pre-fetch stack**: Keep a buffer of 3 upcoming pets...
> - **Virtualized list**: For high-performance rendering...
>
> Compare them regarding network usage and UX fluidity. Wait for my decision before implementing.”

Technique 6: ReAct (Reasoning + Acting)

Theory: A loop of Reasoning -> Acting (using a tool) -> Observing Result -> Reasoning.

- **Pros**: Can solve tasks that require multiple steps and information gathering.
- **Cons**: Can get stuck in loops if not managed.
- **Best for**: Autonomous agents, complex refactoring across multiple files.

Example: ReAct

(Phase 6 Prompt)

> “Analyze the entire codebase. Focus on src/services/petProvider.ts… Your task is to refactor the UI into a modern, mobile-first application.
>
> 1. Review README…
> 2. Install lucide-react…
> 3. Create a PetCard.tsx…
> 4. Use useRef guards…
> 5. Ensure the UI feels like a premium native app. Act on all necessary files…
>
> ”

Technique 7: Feedback Loop

Theory: Using the output (often an error) of a previous run as input for the next correction.

- **Pros**: mimics how humans debug.
- **Best for**: Fixing compile errors, runtime bugs.

Example (Phase 7):

> "The application is currently throwing an error… Run it, capture the full stack trace… and use that information to fix the bug."

Technique 8: Explanation

Theory: Asking the model to explain its own code or logic.

- **Pros**: Great for generating documentation or validating the model's understanding.
- **Best for**: Writing docs, comments, or educational content.

Example (Phase 8):

> "Analyze the implementation in src/services/petProvider.ts. Explain in detail how the connection with 'The Dog API' works... Lifecycle... JSON path..."

Technique 9: Verification

Theory: Using the model to prove that the work was done correctly (Screenshots, Tests).

- **Pros**: establishing trust.
- **Best for**: Final sign-off.

Example (Phase 9):

> "Use your tools to launch the development server... Take a high-quality screenshot... Record a short video... Save these files in a 'assets/' folder..."

Deep Dive

Architecture & Pre-fetching

The Problem: Network Latency

In a "Tinder-like" app, waiting 1-2 seconds between swipes for the next image kills the experience.

Solution: The Pre-fetch Stack.

1. **Buffer**: Keep `N` pets in memory.
2. **Background Fetch**: When user swipes, immediately fetch `N+1`.
3. **Image Pre-loading**: Render hidden `<img>` tags to force browser caching.

Implementation: `usePetStack.ts`

```
const BUFFER_SIZE = 3;

export function usePetStack() {
    // ...
    const advance = useCallback(() => {
        setStack((prev) => {
            const [, ...rest] = prev; // Remove first (FIFO)
            return rest;
        });

        // Background fetch to replenish
        fetchRandomPetProfile().then((newPet) => {
            setStack((prev) => [...prev, newPet]);
        });
    }, []);
}
```

Service Layer: `petProvider.ts`

Merging local data (Bios) with Remote API (Images).

```
export const fetchRandomPetProfile = async (): Promise<PetProfile> => {
    // 1. Local Data
    const randomPet = pets[Math.floor(Math.random() * pets.length)];

    // 2. Remote Data (with Fallback)
    try {
        const response = await fetch(DOG_API_URL);
        // ... validation ...
        imageUrl = data.message;
    } catch (error) {
        console.error('Using fallback image');
    }

    // 3. Merge
    return { ...randomPet, imageUrl };
};
```

Weekly Project

Re-build "PawsMatch"

The Challenge

Objective: Re-build the PawsMatch demo app using a **different technology stack**.

Constraints:

1. **Functionality**: Must have 100% feature parity (Swiping, API integration, Adoption flow).
2. **Design**: Must look *exactly* the same (Glassmorphism, Animations).
3. **Tech**: You cannot use React.

Suggested Stacks:

- Vue 3 + Pinia + Tailwind
- Svelte 5 + Tailwind
- Vanilla JS + CSS Modules (Hard mode)

Definition of Done

- [ ] Project compiles/runs without errors.
- [ ] Swiping right "Likes", swiping left "Passes".
- [ ] Pets are fetched from Dog CEO API + merged with local JSON.
- [ ] **Pre-fetch buffer** is implemented (no loading between swipes).
- [ ] Responsive mobile-first design.
- [ ] README.md documents the new stack and how to run it.

Resources

Visual Guides to LLMs

- The Illustrated Transformer (Jay Alammar) - The classic visual guide to Attention.
- The Illustrated GPT-2 (Jay Alammar) - How GPT models actually generate text.
- Transformers Explained Visually (Towards Data Science)

Prompt Engineering & Agents

- Chain-of-Thought Prompting Elicits Reasoning (Paper) - The original Google Research paper.
- ReAct: Synergizing Reasoning and Acting (Paper) - The foundation of Agentic AI.
- Building LLM Applications for Production (Chip Huyen)

Resources

Deep Dive Articles

- The AI-Native Developer: A manifest - *Swyx.*
- What is Context Engineering? (Anthropic) - How to design better inputs.
- Tool Use & Function Calling Guide - Conceptual overview.
- The Rise of Agents (Sequoia) - Business and tech landscape of agents.

Tools

- Anthropic Claude
- React Documentation
- Vite Documentation