AI Native Dev

Week 2

Adrián Catalan

adriancatalan@galileo.edu

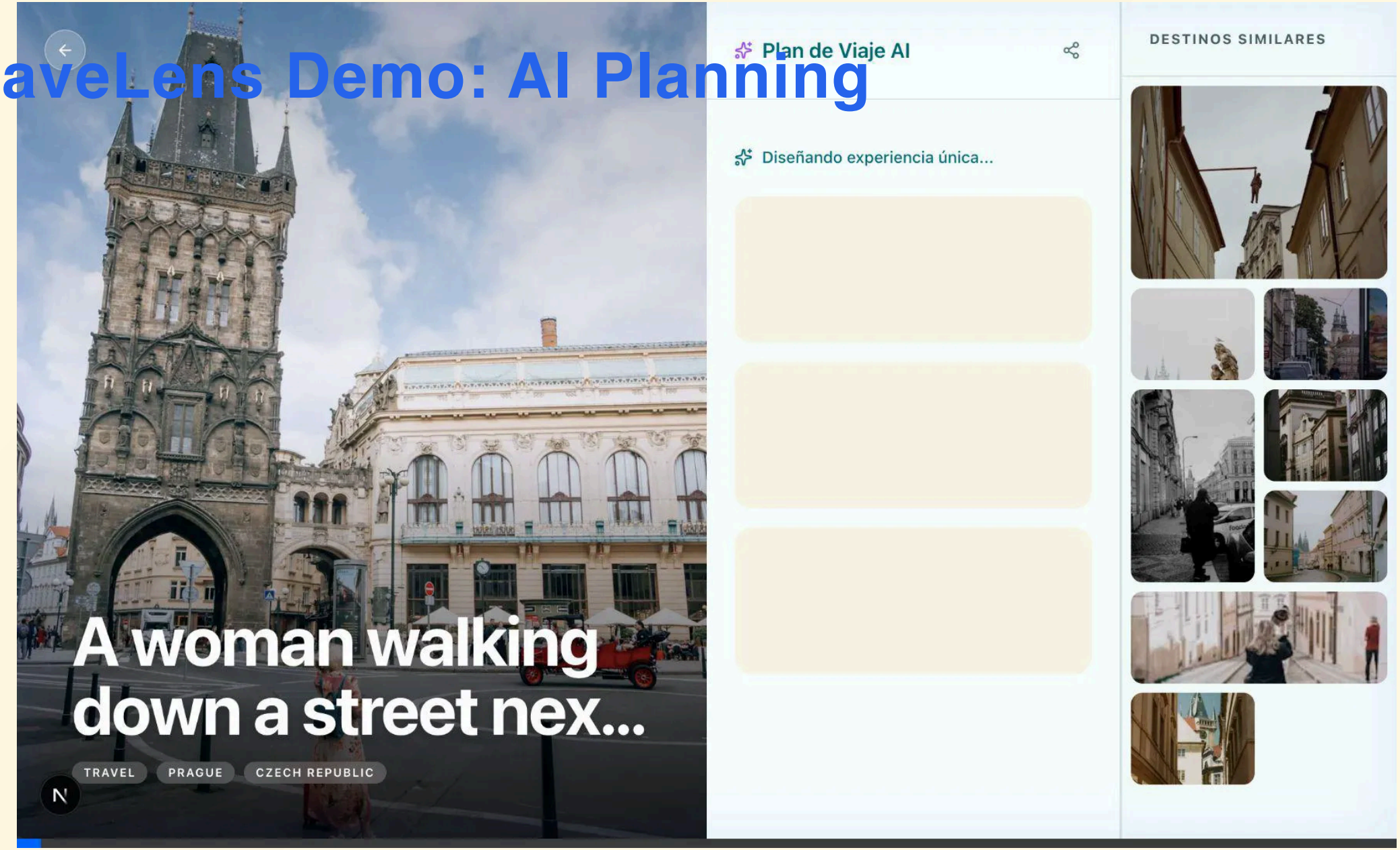
TraveLens

TraveLens is an **AI-powered travel planning application** built with Next.js 16. It combines dynamic imagery from Unsplash with intelligent itineraries generated by Google Gemini to create a premium travel planning experience.

- **Visual Discovery**: Masonry grid layout.
- **AI Travel Plans**: 3-day itineraries generated by Gemini 3 Flash.
- **Modern UI/UX**: Glassmorphism and "Premium" aesthetic.

TraveLens Demo: Visual Discovery

TraveLens Demo: AI Planning



Topic 1: AI-Driven SDLC

Software Development Life Cycle

It is the structured framework that software organizations use to design, develop, and test high-quality software.

Why do we need it?

1. **Cost Efficiency**: Detecting errors early (Plan/Design) is 100x cheaper than in Production.
2. **Quality Assurance**: Ensures the final product meets user expectations.
3. **Predictability**: Provides a roadmap for timelines and resource allocation.

It acts as the "Blueprint" for the entire project life.

Core Phases of SDLC

1. **Plan**: Requirements & Feasibility.
    - *Ex: Product Manager defines "Travel Plan" feature.*
2. **Design**: Architecture & UI/UX.
    - *Ex: System Architect defines Next.js; Designer draws Cards.*
3. **Develop**: Coding the solution.
    - *Ex: Frontend builds Grid; Backend connects Gemini.*
4. **Test**: Verification & Validation.
    - *Ex: QA checks if "Save" button triggers animation.*
5. **Deploy**: Release & Maintenance.
    - *Ex: Deploy to Vercel; Monitor logs.*

The Implementation Gap

In Week 1, we saw that AI can write code. But **Context Management** is the real challenge.

- **The Issue**: LLMs have a limited "Context Window".
- **The Symptom**: As files grow, the AI "forgets" previous instructions.
- **The Solution**: We don't ask for the whole app at once. We break it down.

What is an Agent?

An autonomous entity that perceives its environment, reasons about it, and acts to achieve a goal.

Key Strength: Autonomy
Unlike a script (A -> B), an Agent can decide "Plan A failed, try Plan B".

```
graph LR
    Env[Environment] --Perception--> Agent
    subgraph Agent Loop
    Agent --Reasoning--> Plan[Plan]
    Plan --Action--> Tool[Tool Use]
    end
    Tool --Effect--> Env
```

What is a Coding Agent?

An AI specialized in the domain of Software Engineering.

Capabilities:

1. **Perception**: Reads File System, Terminal Stdout, Git Diff.
2. **Action**: Writes Files, Runs Commands, Commits Code.
3. **Strength**: It understands the *AST* (Abstract Syntax Tree), not just text.

Basics on SDLC and Roles

The Shift

In an AI-Native team, you are not just a contributor; you are the **Manager**. You define the roles that the AI agents will play.

We identified **8 Distinct Roles** to build TraveLens. Breaking the monolith prevents "Context Drift" (when the AI forgets instructions because the prompt is too long).

Role 1: System Architect

Standard Industry Responsibility:
The Architect makes high-level design choices and dictates technical standards. They don't write the UI; they decide *which UI library* to use.

AI Agent Responsibility:

- Sets up the `package.json`.
- Defines the directory structure (`src/services`, `src/components`).
- Establishes patterns (BFF, Repository Pattern).

SDLC Role 1: System Architect

> **System Architect Prompt**
>
> - Use **context7** to fetch the latest documentation for **Next.js v16+**.
> - Initialize the project and install `shadcn/ui`.
> - **Architectural Requirements:**
>   - **BFF Pattern:** Implement `src/app/api/` for secure routes.
>   - **Services Layer:** Create `src/services/` for core logic.
>   - **Component Hierarchy:** Separate `ui` from `features`.

Role 2: UI/UX Frontend Developer

Standard Industry Responsibility:
Translates Figma designs into code. Focuses on accessibility, responsiveness, and visual fidelity.

AI Agent Responsibility:

- Builds components using the *Architect's* specified library (`shadcn`).
- Implements layouts (Grids, Flexbox).
- **Constraint**: Must NOT implement backend logic. Mock data only.

SDLC Role 2: Frontend Developer

> **UI/UX Frontend Developer Prompt**
>
> - Build the interface in **Spanish** using **shadcn/ui**.
> - **Homepage:** Responsive Masonry Grid using Tailwind `columns`.
> - **Detail View:**
>   - **Left Panel:** High-res photo with clean overlay.
>   - **Far-right Sidebar:** "Similar Destinations" as a **Bento Grid**.
> - **Constraints:** Use `urls.small` for performance.

Role 3: Backend Developer

Standard Industry Responsibility:
Handles data processing, database interactions, and API integrations.

AI Agent Responsibility:

- Implements the Interface defined by the Frontend (e.g., `Destination` type).
- Connects to 3rd party APIs (Unsplash).
- Ensures data security (BFF Pattern).

SDLC Role 3: Backend Developer

> **Backend Developer Prompt**
>
> - Implement **Unsplash API** in `src/services/unsplash.ts`.
> - **Logic:**
>   - `fetchPopular()`: Randomized selection of top cities.
>   - `fetchRelated(location, tags)`: Search logic with fallbacks.
> - **Data Mapping:**
>   - Ensure `title` never contains IDs.
>   - Map `urls.thumb`, `small`, and `regular`.

Role 4: Data Specialist

Standard Industry Responsibility:
Often a Data Scientist or AI Engineer. Manages model selection, prompting strategies, and data schema validation.

AI Agent Responsibility:

- Manages the Gemini API connection.
- Writes the *System Prompt* that the App sends to Gemini.
- Enforces JSON Schema validation.

SDLC Role 4: Data Specialist

> **Data Specialist (AI Integration) Prompt**
>
> - **Service:** Implement `src/services/gemini.ts`.
> - **Model:** Use `gemini-3-flash-preview` for speed.
> - **Function:** `generateTravelPlan()` returning **Structured JSON**.
> - **Frontend:** Create `TravelPlanPanel.tsx` to render the JSON output.

Role 5: QA & Code Reviewer

Standard Industry Responsibility:
Ensures quality before release. Looks for bugs, accessibility issues, and performance regressions.

AI Agent Responsibility:

- Scans generated code for "AI Hallucinations" (e.g., importing non-existent libs).
- Checks for Hydration Errors (Next.js specific).
- Adds educational comments.

SDLC Role 5: QA & Code Reviewer

> **QA & Code Reviewer Prompt**
>
> - Audit the code from previous roles.
> - **Requirements:** Check for hydration errors, accessibility (ARIA), and API error handling.
> - **Pedagogical Note:** Add comments titled 'REVIEWER_NOTE' to explain complex sections.

Role 6: API Explanation

Standard Industry Responsibility:
Senior Devs document the "Why" and "How" for junior devs.

AI Agent Responsibility:

- Reads the final code.
- Synthesizes a "How it works" summary.
- Explains complex logic (e.g., the Unsplash <-> Gemini sync).

SDLC Role 6: API Explanation

> **API Connection Explanation Prompt**
>
> - Analyze `src/services/unsplash.ts` and `src/services/gemini.ts`.
> - Provide a detailed explanation of:
>   - How the connection with Unsplash API and Gemini API is established.
>   - The logic used to synchronize external images with Gemini-generated plans.

SDLC Roles 7 & 8: Documentation & Media

Standard Industry Responsibility:

- **Role 7 (Media)**: Marketing or Design team creates "glamour shots".
- **Role 8 (Docs)**: Technical Writers or Junior devs write docs (often outdated).

AI Agent Responsibility:

- **Role 7**: Runs *Visual Regression Tests*. The "screenshots" are proof of UI stability.
- **Role 8**: Generates *Living Documentation*. Parses the actual AST to write accurate docs.

SDLC Roles 7 & 8: Documentation & Media

> **Automated Media Capture (Role 7)**
>
> - **Task**: Launch server, screenshot initial state, record video of interactions.
> - **Output**: `assets/home_demo.webp`, `assets/detail_demo.webp`.

> **Technical Documentation (Role 8)**
>
> - **Task**: Generate `README.md`.
> - **Content**: Architecture overview, API logic explanation, Setup guide.
> - **Visuals**: Embed assets from Role 7.

Topic 2: Agent Workflows

The Hand-off Mechanism

The Traditional Handoff

(Friction Points)

- **Design -> Dev**: "Pixel Perfect" myth. Figma files drift from implemented code. Redlines are ignored.
- **Dev -> QA**: "Works on my machine." Context of environment variables is lost.
- **Back -> Front**: "Is the API ready?" Swagger docs are outdated.

Result: High friction, slow velocity.

The Agentic Handoff

(Context as Code)

In an Agentic SDLC, the "Output" of one agent becomes the **Immutable Context** for the next.

1. **Strict**: Agent B cannot proceed until Agent A's output (file) exists.
2. **Explicit**: No "Slack messages." Instructions are prompt-engineered.
3. **Verifiable**: A file is either present or absent.

Comparison: Context Management

Feature	Traditional	Agentic
Medium	Meetings / Syncs	Files / Prompts
Latency	Hours/Days	Milliseconds
Risk	"Context Drift" (Forgetting)	"Hallucination" (Inventing)
Fix	Documentation	Structuring Context (RAG)

The Agentic Loop

Using Agents turns the SDLC into a series of **Feedback Loops**.

1. **Draft**: Agent generates v1.
2. **Review**: Human (or another Agent) critiques.
3. **Refine**: Agent generates v2 based on critique.

This cycle repeats until the "Definition of Done" is met.

Automation vs Agentic

Feature	Traditional Automation	Agentic Workflow
Trigger	Script run manually/CI	Autonomous decision
Logic	`if/else` rigid paths	LLM Reasoning
Failure	Crashes on error	Self-corrects / Retries
Role	Tool	Collaborator

The Hand-off Mechanism

The critical part of an Agentic Workflow is not the *Agent*, but the **Hand-off**. How does information flow from Agent A to Agent B without loss?

Context Passing Strategy:

1. **File Based**: Agent A writes to `src/`. Agent B reads `src/`.
2. **Interface Based**: Architect defines `interfaces`. Frontend implements `UI`. Backend implements `Data`. Both adhere to the Architect's Interface.

Workflow: Architect -> Frontend

The Foundation Hand-off

- **Input**: Empty Folder.
- **Agent**: Architect.
- **Output**: `package.json`, `tsconfig.json`, Folder Structure.

Why is this hand-off critical?

If the Architect fails to install `shadcn/ui`, the Frontend agent will fail when asked to "Use shadcn Input". The Frontend *depends* on the Architect's output as its "Truth".

Workflow: Frontend -> Backend

The "Mock" Hand-off

- **Input**: UI Components with hardcoded/mock data.
- **Agent**: Backend.
- **Output**: Real API Services replacing mocks.

The Danger Zone:
The Backend agent must not *change* the UI structure. It should only *wire up* the data. We explicitly prompt: "Implement the service to match the existing UI props."

Workflow: Backend -> Data Specialist

The "Intelligence" Hand-off

- **Input**: A working data layer (Unsplash).
- **Agent**: Data Specialist.
- **Output**: AI Generation Service (`gemini.ts`).

Why separate?
Prompting Gemini for a travel plan is a *content* task. Fetching images is a *data* task. By separating them, we verify the Image API works *before* debugging why the AI is generating bad text.

Topic 3: UI Improvements

Theory & Principles

What makes a UI "Premium"?

1. **Visual Hierarchy**: The eye should be guided. Large images first, bold titles second, metadata last.
2. **Gestalt Principles**:
    - *Proximity*: Related items (Title + City) stay close.
    - *Common Region*: The Card container groups content.
3. **Feedback Loops**: Every interaction (Hover, Click) must have an immediate visual response (Scale, Opacity change).

The Default: Generic by Design

(What happens without prompting)

If you ask an LLM: *"Make a travel page"*, it will minimize risk.

- **Result**: Standard Bootstrap/Tailwind card.
- **Why**: It's the statistically most probable token sequence.
- **Problem**: It looks like a template.

Simple Implementation

(Before Refinement)

Standard Implementation without refinement:

- **Layout**: Simple CSS Grid (Rows/Columns).
- **Images**: standard `<img>` tags (slow loading, layout shift).
- **Interaction**: Browser default (blue outline, instant page changes).
- **Result**: Functional, but feels "Cheap".

Result of Detailed Prompting

(After Refinement)

Applying Role 2 Constraints:

- **Masonry Grid**: Tailwind `columns-3` for organic layout.
- **Bento Grid**: "Similar Destinations" in a structured, hierarchical grid.
- **Visuals**: High-res photos with *clean overlays* (Gradient text protection).
- **Performance**: `urls.small` used for speed, pre-fetched.

Prompt Specifics: The Masonry Grid

(UI Structure)

> "**Homepage:** Responsive Masonry Grid (Pinterest-style) using Tailwind `columns`... Use `urls.small` for performance."

- **Constraint**: We forced a specific layout engine (`columns`) instead of generic Flexbox.
- **Performance**: We explicitly requested the optimized image size.

Prompt Specifics: Visual Hierarchy

(Visual Fidelity)

> "**Left panel (Hero):** Selected high-res photo (`urls.regular`). Overlay must be **clean**: Display only **Title** and **Tags** (Remove location text and Compass/Map icons)."

- **Reduction**: We explicitly told the Agent what to *remove* to achieve a cleaner look.

Prompt Specifics: UX & Search

(Interactivity)

> "**Search Bar:** A centered shadcn `Input` with a search icon that updates the grid in real-time (Syncs with URL query params)."

- **Behavior**: We defined the *state mechanism* (URL Params), ensuring the search is shareable and bookmarks work (UX Best Practice).

Deep Dive: Code Spotlight

Why the AI wrote it this way

1. Performance First: CSS Masonry

The AI chose **CSS Columns** over JavaScript libraries to prevent Layout Shift.

```tsx
// MasonryGrid.tsx
export function MasonryGrid({ items }: MasonryGridProps) {
    return (
        <div className="columns-1 sm:columns-2 lg:columns-3 gap-4">
             {/* break-inside-avoid prevents items from being cut in half */}
            {items.map((item) => (
                <div className="break-inside-avoid mb-4">
                    <Card item={item} />
                </div>
            ))}
        </div>
    );
}
```

- **Decision**: `Masonry.js` causes "Hydration Mismatch" in Next.js 14+. CSS is instant.

2. Clean Architecture: `cn()` Utility

The AI standardizes class merging to avoid template literal hell.

```tsx
// lib/utils.ts
import { type ClassValue, clsx } from "clsx"
import { twMerge } from "tailwind-merge"

export function cn(...inputs: ClassValue[]) {
  // Merges Tailwind classes safely (e.g., p-4 vs p-8)
  return twMerge(clsx(inputs))
}
```

- **Decision**: This enables "Agentic Overrides". The Agent can pass `className="bg-red-500"` to a component and it *actually works*.

Weekly Project

Polish "TraveLens"

The Challenge: Polish to Perfection

Goal: The current app is functional. Your job is to make it delightful.

Definition of Done:

- [ ] **Favorites**: Implement "Save" button with `localStorage` persistence.
- [ ] **UI Animation**: Clicking "Save" triggers a Heart/Confetti scale effect.
- [ ] **UI Polish**: Grid items load with a "Staggered Fade-in" effect.
- [ ] **UX**: Masonry Grid supports Keyboard Navigation (Arrow Keys).
- [ ] **UX**: "Recent Searches" chips appear below the search bar.

Resources

Foundations & Strategy

- Software Engineering at Google - *Winters, Manshreck, Wright (O'Reilly).*
- The Bitter Lesson - *Rich Sutton.*
- Strategies for rigorous agentic workflows - *Anthropic Research.*

Architecture & Workflow

- Patterns for Building LLM-based Systems & Products - *Eugene Yan.*
- SWE-agent: Agent-Computer Interfaces - *Princeton University.*
- Building LLM Applications for Production - *Chip Huyen.*

Resources

Design Engineering

- Components as Data: The future of UI - *Guillermo Rauch.*
- Generative UI & Vercel AI SDK - *Concept deep dive.*
- Gestalt Principles in UI Design - *Theory.*
- Design Engineering at Vercel - *Rauno Freiberg.*
- Understanding React Server Components - *Josh W. Comeau.*