PROJECT SHOWCASE / ENGINEERING & PRODUCT PORTFOLIO

Petinha

Technical Case Studies

Prepared for professional review | November 4, 2026 | English edition

Evidence note: Based on prior project discussions, described architecture, selected development/debugging history and portfolio records available at preparation time. This is a portfolio case study, not a fresh source-code audit, external security review, market study or verified commercial due diligence. Planned capabilities and unverified outcomes are labeled accordingly.

Petinha | Technical Case Study

Virtual pet / interactive web game | Stage: Existing application under active engine migration and movement/animation debugging

Technical summary

Petinha is an interactive virtual-pet experience with state-based needs and player actions. Development discussions cover a Next.js application backed by MongoDB and authentication, while the game runtime is being migrated toward Phaser 3 so movement, animation and object interactions are owned by a coherent game loop.

Problem and engineering context

Interactive virtual pets must consistently coordinate persistent needs, character animations, inventory-like actions and scene interactions. Mixing DOM/CSS animations with game-engine transforms can introduce divergent state and unpredictable movement.

Technology stack

- Phaser 3 as the targeted runtime for scenes, sprites, animation, camera and movement.
- TypeScript and Next.js/React, with React intended primarily for authentication after migration.
- NextAuth / Google sign-in and MongoDB-backed pet/account state.
- Phaser scene modules including PetHomeScene and interaction helpers such as WorldInteraction.
- Spritesheets and assets organized by pet or avatar state, including separate gender-specific artwork.

System architecture and component boundaries

- Authentication shell: identify the account and supply an authorized session to the game layer.
- Game scene: Phaser owns input, entity positions, collision, animation playback and lifecycle events.
- Interaction subsystem: resolve hitboxes, proximity and object state transitions (e.g., fridge and microwave).

- State synchronization: translate authoritative pet state to scene visuals and persist actions without losing transitions.
- Asset pipeline: preload and validate spritesheet naming, frame ranges, origin, dimensions and animation maps.

Data model and domain contracts

- Pet state: hunger, happiness, energy, health, cleanliness/sleep-related state and progression.
- SceneView and spriteToken are an interface between application state and visual representation.
- Interaction states: carried item, selected target, animation state, activity completion and guarded transition.
- Potential event history: timestamps/actions for debugging or recovering from interrupted interactions; confirm whether persisted.

Primary application flows

- Sign in and load the user-specific pet.
- Explore the room and interact with world objects.
- Perform actions such as feeding, playing, cleaning, medicine or sleep; update states.
- Select the correct animation for gender, pet condition and action, then render consistently in Phaser.

Engineering challenges and tradeoffs

- Movement oscillation and apparent freezes after fridge/microwave interactions were reported.
- Character art briefly swaps between male/female or dirty/normal sprites; spriteToken mapping and cache keys need tests.
- Missing sprite paths causing HTTP 404; asset manifest and preload validation are essential.
- Dynamic hitbox typing and scene-state nullability triggered TypeScript build issues.
- React/DOM and Phaser ownership overlap can create double-rendered or contradictory motion.

Available evidence and limits

- Reported working pet interactions and existing NextAuth/MongoDB application architecture.
- Phaser migration files, animation debugging and movement bugs were explicitly discussed.
- A fully completed Phaser-only architecture, a stable build and public player adoption have not been verified.

Recommended technical validation / next steps

- Complete a single authoritative movement and interaction controller inside Phaser.
- Build deterministic reproduction tests for fridge-to-microwave-to-carried-item transitions.
- Add asset manifest validation and sprite-change debug telemetry.
- Define a finite-state-machine contract for pet actions, interruption and persistence.

- Profile mobile performance, memory use and animation lifecycle before publishing claims.

Recruiter interview prompts

- How is state ownership divided among Phaser, React and persistence?
- Explain how a finite-state machine would prevent the reported movement oscillation.
- How do you validate sprite assets and correct animation selection?

Disclosure: This report intentionally distinguishes observed discussions from proposed engineering improvements. Tests, benchmarks, revenue, user adoption and security compliance are not asserted without supporting evidence.