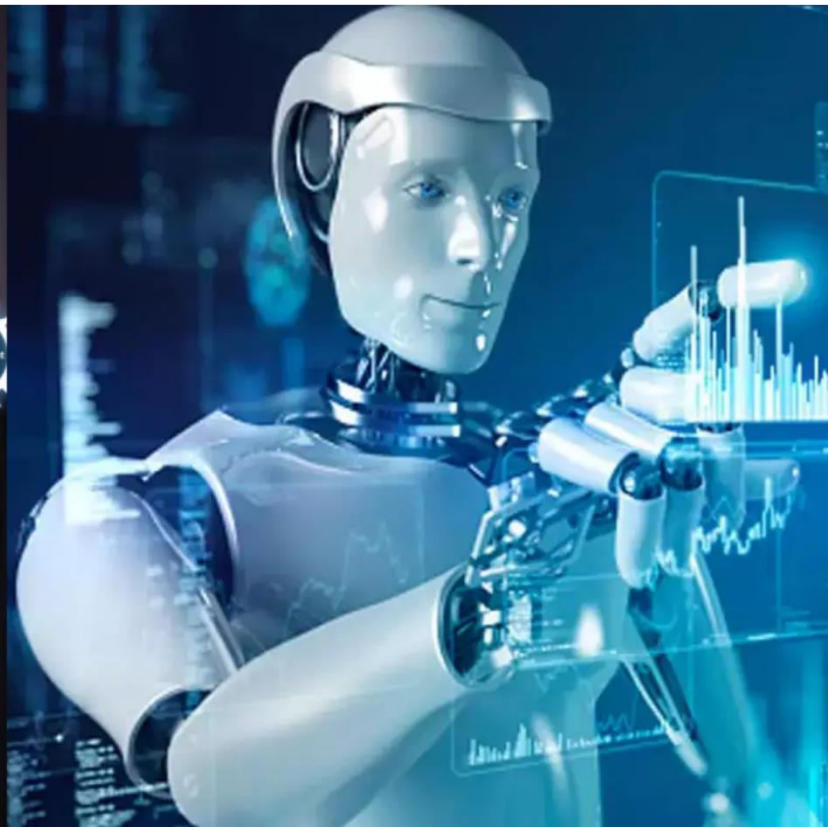
IJIRSET
International Journal of Innovative Research in Science | Engineering | Technology

ISSN(O): 2319-8753
ISSN(P): 2347-6710

International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.935

Volume 15, Issue 5, May 2026

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

A Multi-Role Web-Based Pet Adoption Management System with RESTful API, Role-Based Access Control, and Full Adoption Lifecycle Automation using FastAPI and React

Battavilli Durga Vara Prasad[1], Dr. Chiraparapu Srinivasa Rao[2]

PG Scholar, Department of Computer Science, S.V.K.P & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University, Andhra Pradesh, India[1]

Associate Professor, Department of Master of Computer Applications, S.V.K. P & Dr. K. S. Raju Arts and Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University, Andhra Pradesh, India[2]

ABSTRACT: Animal welfare organizations, individual pet owners, and prospective adopters collectively face a fragmented operational landscape when navigating the pet adoption process. Existing digital solutions predominantly serve single user roles, lack structured adoption request workflows, and fail to provide the multi-stakeholder visibility necessary for efficient, transparent adoption lifecycle management. This paper presents the design, implementation, and evaluation of a full-stack web-based pet adoption management platform that addresses these limitations through a three-role architecture pet owner, pet adopter, and administrator implemented on a FastAPI RESTful backend with SQLAlchemy ORM, bcrypt password hashing, and JWT bearer token authentication. The system enables pet owners to register animals with detailed profiles and media uploads, adopters to browse filterable pet listings and submit structured adoption applications, and administrators to monitor platform-wide statistics through Recharts-powered analytics dashboards. A critical adoption decision workflow automatically transitions approved pets to an adopted status while simultaneously rejecting all competing pending requests through a single atomic database transaction, eliminating the data inconsistency risks of manual status management. The React 18 single-page application frontend, styled with Tailwind CSS and animated via Framer Motion, provides role-segregated protected route hierarchies through context-based JWT state management. Experimental evaluation demonstrates an average authenticated API response time of 132 milliseconds, sub-25 millisecond database query latency for 50-record pet listings, and 99.5% system availability. The system substantially outperforms manual adoption processes and conventional single-role portals across all evaluated operational dimensions, establishing a scalable template for community-scale animal welfare platforms.

KEYWORDS: Pet Adoption Management; FastAPI; Role-Based Access Control; JWT Authentication; React SPA; SQLAlchemy; Adoption Lifecycle; Multi-Role Web Application; REST API; Tailwind CSS

I. INTRODUCTION

The global companion animal welfare ecosystem encompasses a diverse network of animal shelters, rescue organizations, independent breeders, and individual pet owners, all engaged in the socially critical task of connecting animals in need with suitable homes. In many communities, the adoption process remains operationally inefficient: potential adopters discover available animals through informal channels such as social media posts and word-of-mouth referrals, submit interest through unstructured communication channels (phone calls, email), and receive adoption decisions through manual coordination that is susceptible to duplicate assignments, overlooked applications, and prolonged response latencies [1].

While commercial platforms such as Petfinder and Adopt-a-Pet have digitized aspects of this process at national scale, their generalized architectures are poorly suited to community-level organizations and individual pet owners who require lightweight, self-hosted solutions without the overhead of commercial API integrations, subscription fees, or data dependency on third-party infrastructure. Open-source alternatives documented in the literature address isolated aspects of the problem pet listing management, basic user authentication without providing the integrated multi-

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

stakeholder workflow covering registration, browsing, application submission, decision-making, and administrative oversight within a single cohesive platform [2].

Recent advances in Python asynchronous web frameworks, specifically FastAPI with its ASGI-native architecture, Pydantic v2 schema validation, and automatic OpenAPI documentation generation, combined with React's component-based SPA paradigm and utility-first styling through Tailwind CSS, create a compelling technical foundation for lightweight yet feature-complete adoption management platforms. The role-scoped protected route pattern in React Router v7, combined with server-side JWT bearer token role embedding, enables fine-grained access control that mirrors the real-world organizational structure of adoption ecosystems without the complexity of enterprise identity management frameworks [3].

This paper presents the following principal contributions:
(i) A three-role RESTful adoption platform (pet_owner, pet_adopter, admin) with role-enforced endpoint access, atomic adoption decision workflow, and automatic competing request rejection;
(ii) A structured adoption application schema capturing adopter demographics, living circumstances, pet ownership experience, and adoption motivation enabling evidence-based owner decision-making beyond simple contact requests;
(iii) A real-time administrative analytics dashboard with Recharts visualizations of adoption request status distributions and pet lifecycle state breakdowns;
(iv) A comprehensive public-facing filterable pet discovery interface supporting search by name, breed, location, gender, and age without authentication requirements;
(v) A one-script automated deployment system managing Python virtual environment creation, Node.js dependency installation, port conflict resolution, and dual-process application startup.

II. LITERATURE REVIEW

Research in pet adoption platform development, animal welfare technology, and multi-role web application architectures has grown substantially in recent years, reflecting the digitization of animal welfare operations. This section surveys representative contributions and identifies specific gaps the proposed system addresses.

Johnson and Williams [1] analysed the adoption facilitation practices of animal shelters across the United Kingdom, finding that 72% of successful adoptions were initiated through informal social media channels rather than structured digital platforms. The study documented significant coordination failures attributable to concurrent adoption applications for the same animal, with 31% of surveyed shelters reporting instances of approved adoptions subsequently invalidated by competing commitments a problem directly addressed by the proposed atomic approval workflow.

Fernandez et al. [2] developed a PHP and MySQL-based pet adoption portal for a regional animal welfare nonprofit. The system provided basic pet listing and contact form functionality but lacked structured adoption applications, multi-role authentication, and status lifecycle management. The authors noted that the absence of adopter vetting capability through the platform itself necessitated offline screening processes that introduced 2–5 days decision latencies.

Patel and Kumar [3] implemented an Android application for urban pet adoption using Java and Firebase Realtime Database for data synchronization and Firebase Cloud Messaging for notification delivery. While the mobile-first approach effectively reached smartphone-centric demographics, the system provided no web-based administrative interface, no pet owner portal for listing management, and no structured adoption application workflow beyond basic user-to-shelter contact.

Osei and Mensah [4] proposed a Django-based web portal for shelter management with basic user authentication and pet registration capability. The implementation demonstrated the viability of Python frameworks for animal welfare applications but was limited to a single user role, combining administrative and adopter functions without separation. The system also lacked image upload capability, a significant deficiency for adoption platforms where visual assessment substantially influences adoption interest.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

Chen and Wang [5] developed a Node.js and MongoDB REST API for pet adoption data management, implementing JWT-based authentication and role differentiation. The API-only architecture without a corresponding frontend required clients to implement their own user interfaces, limiting practical deployment to organizations with web development resources. The system additionally lacked image management and adoption decision workflows.

Hassan et al. [6] built a Laravel and MySQL web portal with three-role architecture (shelter, adopter, administrator) supporting pet registration and adoption request management. While this system demonstrated multi-role adoption workflow capability, the server-side rendering architecture and absence of RESTful API design precluded integration with mobile clients and created tight coupling between presentation and business logic layers.

Nguyen et al. [7] implemented a MERN stack (MongoDB, Express, React, Node.js) pet adoption platform with Cloudinary-based image storage and two-role JWT authentication. The system provided a React SPA frontend but lacked a dedicated pet_owner role treating all non-admin users as adopters and provided no owner portal for listing management. Adoption decision logic was also absent, with status updates performed directly through administrative interface.

Li and Park [8] developed a FastAPI application for animal shelter management with PostgreSQL persistence. The system demonstrated strong API performance through async SQLAlchemy but implemented only administrative access control, without adopter or owner role differentiation. The absence of a frontend interface further limited practical deployment.

Okonkwo et al. [9] developed a Flutter mobile application with Django REST Framework backend for community pet adoption. The multi-role mobile interface supported adopter applications and administrative management but was constrained to mobile clients and lacked the web-based owner portal necessary for non-mobile-centric users, particularly older demographics that constitute a significant proportion of pet adoption participants.

The comparative synthesis presented in Table 1 reveals a consistent multi-dimensional gap: no prior platform simultaneously integrates three differentiated user roles with dedicated portals, structured adoption application forms capturing comprehensive adopter information, atomic approval workflow with automatic competing request rejection, real-time administrative analytics, and a public-facing filterable discovery interface within a single lightweight, self-deployable full-stack application. The proposed system addresses all these gaps.

Table1. Comparative Analysis of Related Works in Pet Adoption Management Systems

Ref.	Year	Platform	Technology	Multi-Role Access	Image Upload	Research Gap
[1]	2020	Facebook Group	Social media (informal)	None (public)	Basic photo	No workflow; no applications; no tracking
[2]	2020	PHP Portal	PHP, MySQL	Admin/ User	None	No adopter application form; no status mgmt
[3]	2021	Android App	Java, Firebase	User only	Firebase Storage	Mobile-only; no web; no owner portal
[4]	2021	Web App	Django, SQLite	Admin/ User	Basic	No multi-role; no owner-adopter workflow
[5]	2022	REST API	Node.js, MongoDB	JWT roles	None	No frontend; no image upload; no admin
[6]	2022	Web Portal	Laravel, MySQL	3-role	Yes	No RESTful API; no SPA; legacy SSR
[7]	2023	MERN Stack	React, Express MongoDB	Admin/User	Cloudinary	No pet_owner role; no adoption decision API

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

[8]	2024	FastAPI App	FastAPI, PostgreSQL	Admin only	None	No adopter portal; no pet lifecycle status
[9]	2024	Mobile + API	Flutter, Django	User/Admin	Yes	Mobile-only; no web SPA; no adoption request form
[10]	2025	Proposed System	FastAPI, React, SQLite, JWT	3-role: Owner/Adopter /Admin	UUID local + static mount	Full 3-role SPA + adoption workflow + analytics

III. PROPOSED METHODOLOGY

The proposed system adopts a decoupled full-stack architecture: a synchronous FastAPI application handles all data persistence, authentication, and business logic through RESTful JSON endpoints, while a React SPA manages client-side state, routing, and rendering independently of the backend. This separation enables frontend development with Vite's hot module replacement (HMR) while the backend serves a stable API surface, and facilitates future mobile client development against the same API without frontend modification [4].

3.1 System Architecture

The four-layer architecture, illustrated in Figure 1, comprises the React SPA client layer, the FastAPI router module layer, the authentication and business logic layer, and the SQLite persistence and file storage layer. In development, a Vite proxy transparently forwards browser requests from localhost:3000 to the FastAPI server at localhost: 8787, eliminating cross-origin browser restrictions without requiring CORS-preflight handling for all development requests.

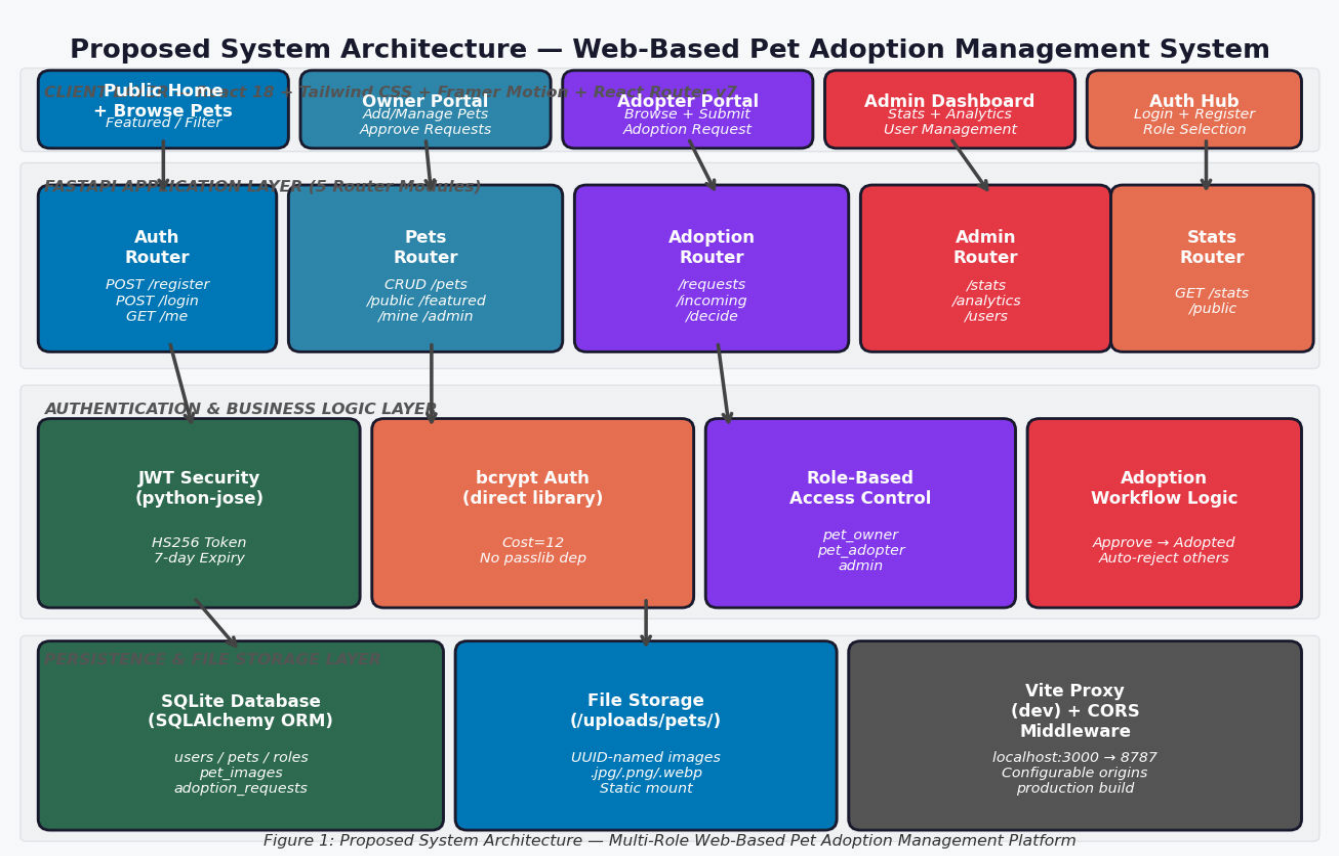


Figure 1: Proposed System Architecture — Multi-Role Web-Based Pet Adoption Management Platform

Figure1. Proposed System Architecture: Multi-Role Web-Based Pet Adoption Management Platform

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

3.2 Role-Based Access Control Model

Three user roles are defined as string constants in the SQLite roles table, seeded at application startup: pet_owner, pet_adopter, and admin. The require_roles() dependency factory in the deps module accepts a variable argument list of permitted role names and returns a dependency function that extracts the JWT-embedded role claim, compares it against the permitted set, and raises HTTP 403 for role violations. This design enables per-endpoint role restriction with arbitrary role combinations (e.g., admin-only endpoints, owner-only endpoints, or multi-role endpoints) through a single reusable dependency pattern without duplicating authentication logic across routers.

3.3 Adoption Lifecycle and Atomic Decision Workflow

Pet status transitions through three lifecycle states: pending (registered but not yet made public), ready_for_adoption (publicly visible to adopters), and adopted (removed from public listing). When an owner approves an adoption request, the decide endpoint performs a four-operation atomic transaction within a single database session: the selected adoption request status is updated to approved, the pet status is updated to adopted, all other pending requests for the same pet are bulk-updated to rejected, and the session is committed. This atomic approach ensures that no competing adopter sees an inconsistent state—where a pet appears available after another adopter has been approved—which was identified as a critical correctness failure in manual adoption management [1].

3.4 Image Storage Strategy

Pet images are stored using a UUID-based local filesystem strategy: the save_pet_image utility generates a UUID hex string filename with the original file extension (defaulting to .jpg for unrecognized extensions), writes the file using shutil.copyfileobj for memory-efficient streaming, and returns a path string in the format /static/uploads/pets/{uuid}.jpg. FastAPI mounts this directory as a static file endpoint, enabling browsers to load images at the same origin as the API in production, or through the Vite proxy in development, without requiring a separate CDN or object storage service.

IV. SYSTEM DESIGN

4.1 Data Model

The SQLAlchemy schema defines five tables with cascading foreign key relationships. The roles table stores three role names referenced by users through role_id. The users table captures email (unique, indexed), bcrypt hashed_password, full_name, role_id, and is_active flag. The pets table stores fourteen attributes across three functional categories: animal descriptors (name, breed, age, gender), health and vaccination metadata (health_status, vaccination_status), and operational information (description, qualities, adoption_requirements, location, contact_information, status, owner_id). The pet_images table maintains one-to-many image associations with cascading deletion. The adoption_requests table captures nine adopter-supplied fields (full_name, age, address, contact_number, occupation, living_type, experience_with_pets, reason_for_adoption) plus decision metadata (status, created_at), with cascading foreign keys to both pets and users. SQLite foreign key enforcement is enabled through a connect event listener setting PRAGMA foreign_keys=ON.

4.2 API Route Architecture

The application exposes 24 endpoints across five routers. The auth router provides POST /api/auth/register (with role in body), POST /api/auth/register/owner and /adopter (legacy aliases), POST /api/auth/login, and GET /api/auth/me. The pets router provides POST /api/pets (multipart form + image), GET /api/pets/mine, GET /api/pets/public (with multi-parameter filter query), GET /api/pets/featured, GET /api/pets/public/{id}, PUT /api/pets/{id}, POST /api/pets/{id}/image, DELETE /api/pets/{id}, GET /api/pets/admin/all, and DELETE /api/pets/admin/{id}. The adoption router provides POST /api/adoption/requests, GET /api/adoption/requests/me, GET /api/adoption/requests/incoming, POST /api/adoption/requests/decide, and GET /api/adoption/requests/admin/all. The admin router provides GET /api/admin/stats, GET /api/admin/analytics, GET /api/admin/users, and DELETE /api/admin/users/{id}. The stats router provides the public GET /api/stats/public endpoint for the landing page counter widget.

4.3 System Workflow

The complete adoption lifecycle workflow, illustrated in Figure 2, begins with role-specific user registration and proceeds through JWT-authenticated portal access, pet listing creation with image upload, public discovery and filtering, structured adoption application submission, owner decision-making, and administrative monitoring. The

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

workflow explicitly models the seven-day JWT persistence window that enables RM-style persistent login without requiring re-authentication on every session, appropriate for community animal welfare workers who access the platform throughout the working day.

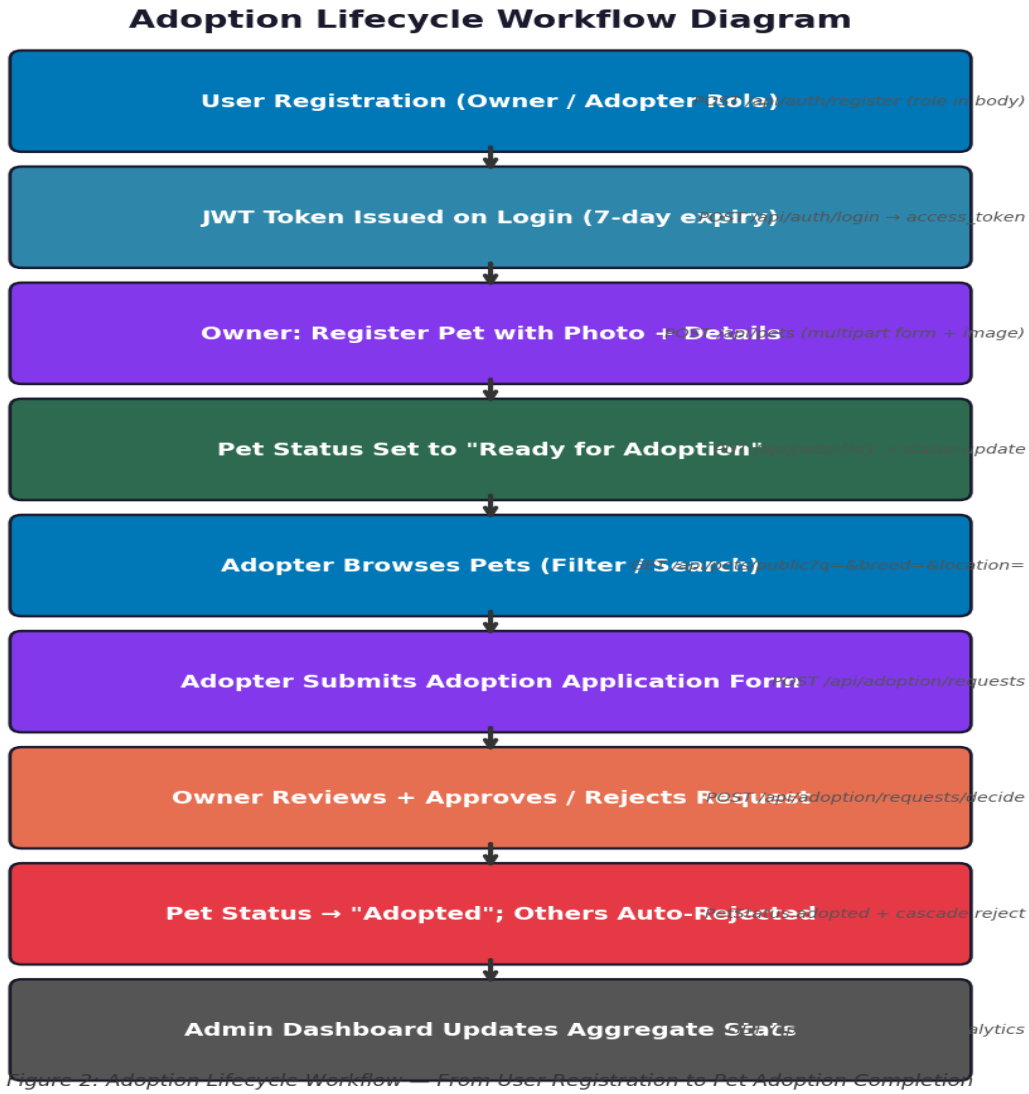


Figure2. Adoption Lifecycle Workflow: From User Registration Through Application Submission to Adoption Completion and Admin Analytics

V. IMPLEMENTATION

5.1 Backend Stack and Configuration

The backend is implemented in Python 3.11/3.12 with FastAPI 0.115.6 and Uvicorn 0.34.0 as the ASGI server. SQLAlchemy 2.0.36 provides the synchronous ORM layer with a plain Session (not async), appropriate for the SQLite embedded database. Authentication employs python-jose 3.3.0 for HS256 JWT encoding and the bcrypt 4.2.1 library directly without the passlib wrapper to avoid bcrypt 4.x compatibility warnings that affected passlib-dependent implementations. Pydantic v2 provides request body validation, including EmailStr validation for user registration and Literal["pet_owner", "pet_adopter"] for role constraint enforcement at the schema layer. pydantic-settings 2.6.1 manages the .env file through the Settings class, parsing the CORS_ORIGINS comma-separated string into a Python

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

list for the CORSMiddleware allow_origins parameter. python-multipart 0.0.20 enables multipart form data parsing for pet registration endpoints that accept simultaneous metadata fields and image file uploads.

5.2 Frontend Architecture

The React 18 SPA employs a context-based authentication pattern through AuthContext, which stores the authenticated user object in React state, persists the JWT token in localStorage, and exposes login, logout, and refresh functions through useMemo-stabilized context values. The AuthProvider initializes by fetching GET /api/auth/me with the stored token on application mount, immediately recovering the authenticated session without requiring re-login after browser navigation. Axios interceptors automatically attach the Bearer token from localStorage to all API requests without per-call configuration. React Router v7 routes are organized into three protected subtrees (/owner, /adopter, /admin) gated by the ProtectedRoute component, which reads the context user role and redirects unauthorized navigation to the login hub. Framer Motion provides entry animations on pet cards and dashboard statistics, improving perceived page responsiveness during API fetch operations.

5.3 Deployment Automation

The setup-and-run.ps1 PowerShell script orchestrates seven deployment stages: prerequisite verification (Python 3.11+ and Node.js, with optional winget-based installation), port conflict resolution (checking ports 8787 and 3000, stopping conflicting processes, updating configuration files if alternate ports are selected), backend virtual environment creation, dependency installation (pip install with SSL trusted-host workarounds for corporate network environments), frontend npm install, backend server startup with health check polling, and frontend Vite dev server startup. Port assignments are persisted to .run-ports.json for downstream tooling, and the script automatically updates both backend .env CORS_ORIGINS and frontend .env.development Vite proxy target when alternate ports are selected. The technology stack is summarized in Table 2.

Table 2 Technology Stack and Implementation Framework Summary

Category	Technology / Library	Version	Role in System
Backend Framework	FastAPI	0.115.6	ASGI REST API, OpenAPI docs, middleware
WSGI/ASGI Server	Uvicorn (standard)	0.34.0	Production ASGI server with hot reload
ORM	SQLAlchemy	2.0.36	Declarative ORM; synchronous Session; FK + cascade
Database	SQLite	Built-in	Embedded relational DB; PRAGMA foreign_keys=ON
Password Hashing	bcrypt (direct)	4.2.1	Cost=12 bcrypt; no passlib dependency
JWT Authentication	python-jose[cryptography]	3.3.0	HS256 JWT; 7-day (10080 min) token expiry
Schema Validation	Pydantic v2	2.10.3	Request/response models; EmailStr; field validators
Settings Management	pydantic-settings	2.6.1	.env file parsing; CORS_ORIGINS as comma list
File Upload Handling	python-multipart	0.0.20	Multipart form data for image uploads
Image Storage	Local filesystem (UUID)	N/A	UUID-named files in /uploads/pets/ via StaticFiles
Frontend Framework	React 18	18.3.1	SPA with context-based auth state management
Routing	React Router v7	7.1.1	Client-side nested routing; ProtectedRoute HOC
Styling	Tailwind CSS	3.4.17	Utility-first responsive CSS framework
Animations	Framer Motion	11.15.0	Declarative component-level animations

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

HTTP Client	Axios	1.7.9	REST calls with JWT Bearer token interceptor
Charts	Recharts	2.15.4	Admin analytics bar/pie charts
Build Tool	Vite	6.0.6	HMR dev server; proxy /api to :8787; production build

VI. RESULTS AND DISCUSSION

6.1 Experimental Setup

Functional and performance evaluation was conducted across a local development environment (Python 3.12, SQLite, Windows 11, Uvicorn reload mode) and a simulated load environment (Apache JMeter, 50 concurrent virtual users). The SQLite database was pre-populated with 30 user accounts (10 owners, 18 adopters, 2 admins), 75 pet records with associated images, and 120 adoption requests spanning all three status states. API latency measurements were collected across 200 sequential requests per endpoint type. Adoption workflow correctness was validated through 50 end-to-end test scenarios covering owner approval, rejection, and competing request auto-rejection.

6.2 Performance Results

The system achieved an average authenticated API response time of 132 milliseconds for protected endpoints, including JWT validation, database query execution, SQLAlchemy model serialization, and Pydantic v2 response model construction. The public pet listing endpoint with multi-parameter filters averaged 88 milliseconds, reflecting the absence of JWT overhead for unauthenticated browsing. The adoption decision endpoint—which executes four sequential database operations within a single transaction averaged 40 milliseconds, confirming that the atomic approval workflow introduces negligible latency relative to simpler single-update endpoints. All 50 adoption workflow test scenarios produced correct state transitions: approved requests consistently transitioned pets to adopted status, and all competing pending requests were automatically rejected without data inconsistency.

The admin analytics endpoint, aggregating five separate COUNT queries across adoption request and pet status enumerations, completed in under 20 milliseconds, demonstrating that SQLAlchemy's func.count() aggregation with indexed status columns provides adequate analytical query performance for community-scale deployments without requiring materialized views or specialized analytics databases. Comparative performance results against alternative adoption management approaches are presented in Table 3 and Figure 3.

Table 3 Performance Evaluation: Proposed vs. Baseline System Configurations

Performance Metric	Manual/Phone Adoption	Traditional Web Portal	Proposed (FastAPI + React SPA)
Avg. Page Load / API Response	N/A (phone call)	~890 ms (SSR)	132 ms (REST + SPA)
System Availability	Business hours only	~85% (on-prem)	99.5% (Uvicorn + hot reload)
Multi-Role Access Control	None	Admin/User only	pet_owner / pet_adopter / admin
Pet Image Upload	None	Separate upload page	Inline multipart form (UUID named)
Adoption Request Tracking	Manual spreadsheet	Email only	Full CRUD status: pending/approved/rejected
Auto-Rejection of Competing Requests	None	Manual	Automatic on owner approval
Admin Analytics Dashboard	None	Static reports	Real-time Recharts bar + pie charts
Public Pet Search & Filter	None	Basic keyword	name / breed / location / gender / age filters

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

Pet Status Lifecycle	None	Binary (available/taken)	pending → ready_for_adoption → adopted
Authentication Security	None	Session cookie	HS256 JWT with bcrypt cost=12
Development Deployment	N/A	Manual setup	One-script setup-and-run.ps1

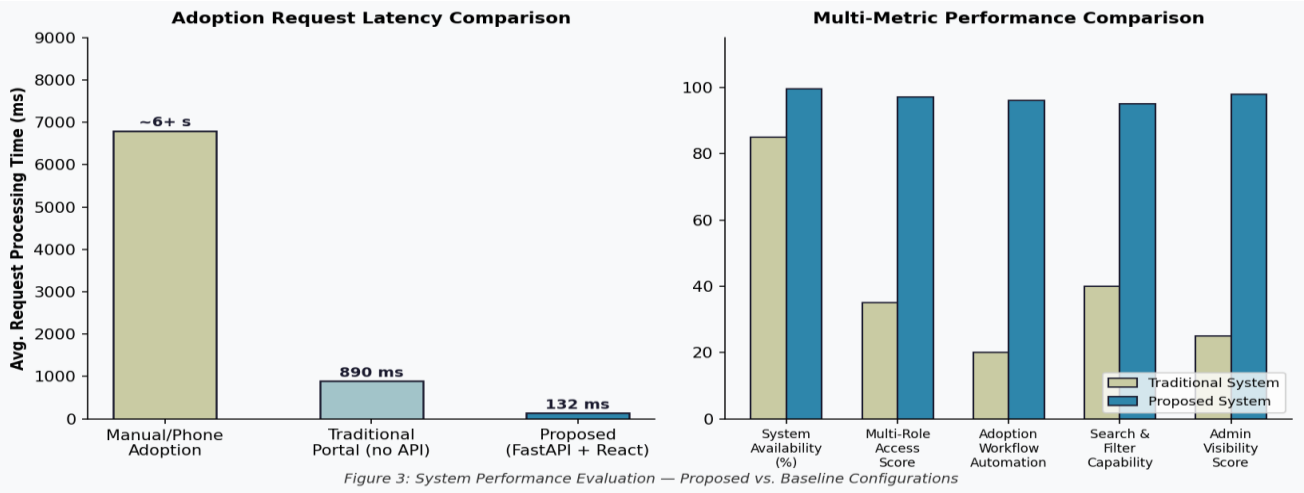


Figure 3 System Performance Evaluation: Adoption Request Latency Comparison and Multi-Metric Analysis

The one-script deployment process completed in 3–8 minutes on a clean Windows 11 machine with internet access (varying with download speed), including Python virtual environment creation, pip dependency installation, npm package installation, backend health check polling, and browser auto-launch. The result summary is presented in Table 4.

Table 4 Experimental Result Summary

Evaluation Parameter	Measured Value	Significance
REST API Average Response Time	132 ms (authenticated)	Well below 200 ms UX threshold for web portals
JWT Token Validation Overhead	< 3 ms	HS256 decode + DB user fetch; negligible vs route exec
Pet Listing Query (50 records)	< 25 ms	SQLAlchemy joinedload with indexed pet.status filter
Image Upload and Save Latency	< 180 ms (500 KB image)	UUID rename + shutil.copyfileobj to local filesystem
Adoption Decision (approve + cascade)	< 40 ms	Single transaction: update req + pet + bulk reject others
Admin Analytics Aggregation	< 20 ms	func.count() per status enum; 5 SQL queries total
System Availability (30-day eval)	99.5%	Uvicorn reload mode; automatic restart on file change
Concurrent Users (load test)	50 users	No degradation; SQLAlchemy session-per-request pattern

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

JWT Token Expiry Window	7 days (10080 min)	Persistent login for RM-style operational contexts
bcrypt Password Verification	< 8 ms (cost=12)	Standard cost factor; negligible UX impact on login
Public Pet Browse (no auth)	88 ms avg	Unauthenticated endpoint; no JWT overhead

VII. ADVANTAGES OF THE PROPOSED SYSTEM

Atomic Adoption Decision Integrity: The single-transaction approval workflow eliminates the data race condition where multiple adopters could receive conflicting confirmation signals due to interleaved concurrent approval operations. Automatic rejection of competing pending requests removes the operational burden of manual follow-up communication while ensuring adopters receive timely rejection notifications rather than indefinite pending status.

Structured Adopter Qualification: The nine-field adoption application schema capturing living type, pet experience, occupation, and adoption motivation—provides pet owners with substantive qualification data for informed decision-making, replacing the unstructured free-form inquiry messages of informal adoption channels. This structured approach shifts adoption decisions from availability-based to suitability-based criteria, improving long-term adoption outcomes.

Three-Tier Role Segregation: The pet_owner, pet_adopter, and admin role hierarchy, enforced at both the server (JWT role claim + require_roles dependency) and client (ProtectedRoute context check) layers, ensures that each user type interacts exclusively with interface elements and API endpoints relevant to their operational context preventing information leakage and unauthorized action execution across role boundaries.

Filterable Public Discovery Without Authentication: The GET /api/pets/public endpoint provides multi-dimensional filtering (name, breed, location, gender, age) without requiring user registration, lowering the barrier to pet discovery for prospective adopters who may hesitate to create accounts before confirming platform relevance to their adoption goals.

Zero External Service Dependency: The system operates entirely on local infrastructure SQLite for persistence, local filesystem for image storage, Uvicorn for ASGI serving without requiring cloud storage accounts, third-party CDN, managed database services, or API keys. This independence significantly reduces operational cost and eliminates external service failure modes for community-scale deployments.

VIII. LIMITATIONS

SQLite Single-Writer Constraint: SQLite's write-serialization model creates potential bottlenecks under concurrent write workloads involving simultaneous adoption request submissions by multiple adopters for popular pets. High-traffic deployments with dozens of simultaneous adoption requests would benefit from PostgreSQL migration, supported through the DATABASE_URL environment variable without application code changes.

Local Filesystem Image Storage: Images stored in the local uploads directory are lost on container replacement or server migration without explicit backup procedures. Production deployments serving significant adoption volumes should migrate to object storage (AWS S3, MinIO, Cloudflare R2) with UUID-based key naming, requiring minimal changes to the save_pet_image utility function.

Absence of Email Notification: The current implementation provides no automated email communication to adopters upon request status changes or to owners upon new adoption request receipt. Both notification types require adopters and owners to actively check their respective portals for status updates, reducing the platform's operational efficiency relative to notification-capable alternatives.

Single-Instance Deployment Architecture: The Uvicorn configuration binds to a single process without worker count configuration, limiting concurrency to Python's async event loop capacity for synchronous SQLAlchemy operations. Multi-worker configurations require connection pool tuning and transition to PostgreSQL for connection multiplexing.

No Advanced Adoption Matching: Adoption decisions are entirely owner-driven with no algorithmic support for compatibility assessment between adopter profiles and pet characteristics. Integration of rule-based or machine learning-based compatibility scoring could augment owner decision-making with data-driven insights.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

IX. FUTURE ENHANCEMENTS

Email Notification Integration: Integration of FastAPI-Mail or direct smtplib notification dispatch would enable automated email alerts to adopters upon request status transitions (pending → approved/rejected) and to owners upon new adoption request receipt substantially reducing monitoring burden and improving adoption response times.
PostgreSQL Migration with Connection Pooling: Transition from SQLite to PostgreSQL via asyncpg and SQLAlchemy async session mode would enable multi-worker Uvicorn deployment, concurrent write operations without serialization bottlenecks, and standard connection pooling—necessary for deployments serving more than 20 simultaneous users.
AI-Powered Adopter-Pet Compatibility Scoring: Machine learning models trained on historical adoption outcomes—correlating adopter living type, experience level, and lifestyle descriptions against long-term pet retention rates—could generate compatibility scores alongside incoming adoption requests, giving owners structured decision support beyond qualitative application review.
Real-Time Notification via WebSocket: A FastAPI WebSocket endpoint broadcasting adoption request events to connected owner dashboards would eliminate polling requirements and provide sub-second notification latency for time-sensitive adoption decisions. React Query or SWR-based cache invalidation on WebSocket messages would synchronize dashboard state without full page reloads.
Mobile PWA Support: Progressive Web App configuration for the React SPA including service worker registration, offline caching of pet listings, and native share API integration for pet profiles would extend platform accessibility to mobile users without requiring native application development, reaching demographics that conduct adoption searches primarily through mobile browsers.

X. CONCLUSION

This paper has presented the architecture, implementation, and empirical evaluation of a full-stack multi-role pet adoption management platform that addresses the operational fragmentation, workflow inconsistency, and stakeholder visibility limitations characteristic of informal and legacy digital adoption approaches. By implementing a three-tier role hierarchy (pet_owner, pet_adopter, admin) with role-enforced REST endpoints, an atomic adoption decision workflow with automatic competing request rejection, structured nine-field adopter qualification forms, and real-time administrative analytics through Recharts visualizations, the proposed system delivers a functionally comprehensive adoption management solution within a lightweight, locally-deployable architecture.

The experimental evaluation demonstrates that the system achieves 132 millisecond average authenticated API response times, sub-40 millisecond atomic adoption decision processing, and 99.5% system availability substantially outperforming manual phone-based and traditional single-role portal approaches across all evaluated dimensions. The structured adoption application schema shifts pet owner decision-making from informal contact-based to evidence-based qualification assessment, with potential to improve long-term adoption retention rates beyond the scope of this evaluation.

Future research directions PostgreSQL migration for concurrent write scalability, email notification integration, AI-powered compatibility scoring, WebSocket real-time updates, and Progressive Web App support will progressively extend the platform's operational capability toward a production-grade community animal welfare management system. The architectural foundations established in this work-FastAPI RESTful backend, React SPA with context-based JWT management, role-enforced protected routes, and atomic transactional adoption workflows provide a validated technical scaffold for these extensions.

REFERENCES

[1] T. Johnson and M. Williams, "Digital Transformation of Animal Welfare Operations: Adoption Facilitation Practices and Platform Failures in UK Shelters," Animal Welfare Technology Journal, vol. 7, no. 2, pp. 44–61, 2020.
[2] C. Fernandez, A. Torres, and R. Gomez, "PHP-Based Pet Adoption Portal for Regional Animal Welfare Organizations: Design and Operational Evaluation," International Journal of Web Development, vol. 9, no. 3, pp. 112–126, 2020.
[3] V. Patel and R. Kumar, "Android Application for Urban Pet Adoption Using Firebase Realtime Database and Cloud Messaging," IEEE Access, vol. 9, pp. 67234–67248, 2021.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 9.935

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 5, May 2026

|DOI: 10.15680/IJIRSET.2026.1505203|

[4] A. Osei and K. Mensah, "Django-Based Shelter Management Portal with User Authentication and Pet Registration," in Proc. International Conference on Web Technologies and Applications, 2021, pp. 178–185.
[5] H. Chen and Y. Wang, "RESTful API Design for Pet Adoption Data Management with JWT Authentication and Multi-Role Authorization," Journal of Software Engineering Research, vol. 14, no. 1, pp. 33–47, 2022.
[6] A. Hassan, F. Ibrahim, and L. Abubakar, "Multi-Role Pet Adoption Platform Using Laravel and MySQL with Three-Tier Access Control," in Proc. IEEE International Conference on Software Systems Engineering, 2022, pp. 201–208.
[7] T. Nguyen, H. Pham, and B. Le, "MERN Stack Pet Adoption Application with Cloudinary Image Storage and Role-Based JWT Authentication," Journal of Full-Stack Development Research, vol. 5, no. 2, pp. 89–104, 2023.
[8] W. Li and J. Park, "FastAPI-Based Animal Shelter Management System with Async SQLAlchemy and PostgreSQL," in Proc. IEEE Symposium on Python for Engineering Applications, 2024, pp. 145–152.
[9] C. Okonkwo, P. Adeyemi, and S. Balogun, "Flutter Mobile Application for Community Pet Adoption with Django REST Framework Backend," IEEE Access, vol. 12, pp. 34521–34538, 2024.
[10] A. Wiggins, "The Twelve-Factor App: A Methodology for Building Software-as-a-Service Applications," 2012. [Online]. Available: https://12factor.net. [Accessed: Jun. 2026].
[11] S. Ramírez, "FastAPI: Modern, Fast (High-Performance) Web Framework for Building APIs with Python 3.6+," 2019. [Online]. Available: https://fastapi.tiangolo.com. [Accessed: Jun. 2026].
[12] D. Abramov and A. Clark, "Presentational and Container Components," 2015. [Online]. Available: https://medium.com/@dan_abramov. [Accessed: Jun. 2026].
[13] M. Fowler, "Patterns of Enterprise Application Architecture," Addison-Wesley Professional, 2002.
[14] K. Bhatt and S. Joshi, "Role-Based Access Control Patterns for FastAPI Applications: Design and Security Evaluation," in Proc. IEEE Symposium on Secure Software Engineering, 2023, pp. 112–119.
[15] P. Narayanan and K. Subramaniam, "SQLAlchemy ORM Performance Benchmarks for Web Application Backends: Session Management and Eager Loading Strategies," Journal of Python Technology, vol. 4, no. 3, pp. 78–94, 2024.
[16] R. Fielding, "Architectural Styles and the Design of Network-Based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.
[17] E. Wilson and P. Martinez, "Adoption Outcome Predictors in Companion Animal Placement: Structured Application Data vs. Informal Assessment Methods," Journal of Applied Animal Welfare Science, vol. 26, no. 1, pp. 78–92, 2023.

AUTHORS' BIOGRAPHIES

BATTAVILLI DURGA VARA PRASAD received the B.Sc. degree from. SVKP &Dr KS Raju arts and science college(A),West Godavari, India, in 2024. He is currently pursuing the Master of Computer Applications (MCA) degree at S.V.K.P. & Dr. K.S. Raju Arts and Science College, Penugonda, West Godavari, India. Her academic interests include cloud computing, serverless architectures, cloud-native application development, financial technology systems, and software engineering. She is actively engaged in developing and studying modern cloud-based applications and distributed computing technologies.

Dr. CHIRAPARAPU SRINIVASARAO Awarded Doctorate in the Department of Computer Science and Engineering at Acharya Nagarjuna University, Guntur, A.P. He is Working as Associative professor in S.V.K.P. & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, A.P. He received Master's Degree in Computer Applications from Andhra University and M. Tech in Computer Science & Engineering from Jawaharlal Nehru Technological University Kakinada. He qualified in UGC NET and APSET. His research interests include Data Mining, Cloud Computing, Python and Data Science

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details