10.48047/jocaaa.2026.35.04.20

Development and Implementation of a Dynamic Portfolio Design System Using Python Django Framework

Pagalla Bhavani Shankar[1], M Babu Reddy[2], K Abida Begum[3], Reddi Chinna Rao[4] S Sanjay Durga[5]Talari Kasinath[6], Pagolu Eswar Vara Prasad[7], Yamala Sandeep Kumar[8]

[1]Assistant Professor
Department of Computer Science and Engineering
University College of Engineering and Technology, Krishna University, Rudravaram, Machilipatnam, India
ORCID - 0000-0001-5935-7758
[2]Assistant Professor
Department of Computer Science Engineering
University of Arts and Science, Krishna University, Rudravaram, Machilipatnam, India
[3]Assistant Professor
Department of BS&H
University College of Engineering and Technology, Krishna University, Rudravaram, Machilipatnam, India
[4,5, 6,7] University College of Engineering and Technology
Krishna University, Rudravaram, Machilipatnam, India
chinnaraoreddi73@gmail.com, sanjaydurga497@gmail.com , talarikasinath62@gmail.com
pagolueswarvp@gmail.com, yamalasandeepkumar38@gmail.com

Abstract

In the present digital-first professional world, an effective online portfolio is now an essential tool amongst students, freshers and working professionals aiming to showcase their skills, project work, and create credibility by displaying their work. Conventional systems such as LinkedIn profiles, hand-written HTML/CSS websites, and web builders have limited constraints such as conformity to fixed templates, recurring subscription fees, high technical learning curve, and lack of real ownership of the data. In this research paper, the design, coding of the system, and complete development of a Portfolio Design System- a web application that is developed on Python Django framework, and Model-View-Template (MVT) architecture are presented. The system grants users without any coding experience the ability to autonomously structure, administer and publish responsive professional portfolio websites via an easy-to-use web-based dashboard. The features integrated by Core include drag-and-drop image uploading with automatic resizing, real-time live preview, single-click publishing portfolios and a full-fledged web control panel that supports full Create, Read, Update, Delete (CRUD) operations. Two complete live portfolio sites were designed and tested, with 100 percent cross-device compatibility, page speed of less than 2 seconds and excellent SEO capability. The suggested architecture workstably fills the gaps in current systems and provides a secure, scalable, and maintainable infrastructure to support dynamic portfolio generation by a web-based platform.

Terminals: Django Framework, Portfolio Web Site Generator, No-Code Web Expertise, Full-Stack Web expertise, Responsive Style, MVT Architecture, bootstrap 5, SQLite, PostgreSQL, CRUD activity.

10.48047/jocaaa.2026.35.04.20

I. INTRODUCTION

The digital revolution has revolutionarily changed the way professionals, students and creative practitioners present themselves to the world. In a world that is characterized by distant-hiring, online contract markets and international teamwork, a properly organized digital portfolio serves as a dynamic resume as well as a real-time proving of technical or creative aptitude. Online portfolios have become a competitive differentiator, as recruiters and clients use them more and more prior to face-to-face professional interviews.

Though this demand is on the rise, most people with the desire to create a digital portfolio, face major obstacles. The non-technical users cannot write the HTML, CSS and JavaScript that will be needed to make an independent site. Current managed platforms are paid a subscription fee, cannot be customized in a template, and still maintain partial ownership of user data. A substantial gap in the software ecosystem is clearly the lack of a free, customizable and free purpose portfolio management system.

This study presents the Portfolio Design System a web application based on the entire stack of application, which is designed specially to remove these barriers. It is developed on the Python Django web development system, which is a high-level web development toolkit that has been configured to have security defaults, a modular system and a rapid web development cycle. By availing the in-built authentication/ORM support of Django and with the added administrative provision, the proposed system will enable the users of the system to control all the range of their portfolio management without necessarily writing a single line of code on the class.

A. Background and Motivation

Personal branding has become more important as online platforms gain more and more prominence within the recruitment procedures. Research on human-computer interaction realm testifies that on average recruiters invest seven seconds analyzing an online presence of a job seeker and make the first impression [3]. A professionally crafted portfolio, one that explains skill, projects, history of education and professional accomplishments in a clear and understandable way, will enhance the likelihood of being chosen significantly. But the technical obstacle to the development of such a portfolio is still prohibitive to most of the target user base.

B. Problem Statement

The existing approaches to creating portfolios have a point of intersection of constraints. Portfolios written in Static HTML/CSS will need a continuous interaction with the developers to be updated. On-demand offerings like Wix and WordPress will implement a fee structure that implements monthly charges after custom domain and storage limits are met. Although it has global recognition, LinkedIn has limited visual opportunities to customize and an absence of a system to implement an independent branded site. There is no current product that offers no-code operation, full data ownership, free deployment, and professional quality visual output altogether. The Portfolio Design System is designed to address this very multi-dimensional gap.

C. Research Objectives

The main goals of the design and execution of this project can be formulated in the following way:

1. To create an easy-to-use web application, which helps to create the needed professional portfolios without the necessity to code.

10.48047/jocaaa.2026.35.04.20

2. To have a strong and secure background structure basing on the framework of Django MVT with complete CRUD approach.
3. To develop a fully responsive frontend interface that is desktop, tablet, and mobile viewport friendly.
4. To offer a user-friendly administrative interface to enable real-time preview of portfolio and one-click publishing.
5. To make the system architecture modular and scalable to the need to add new features in the future including AI-based suggestions and support of REST API.

II. SYSTEMATIC LITERATURE REVIEW

The topic of the web-based portfolio generation has been addressed in several different angles with regard to the human-computer interaction, software architecture and educational technology. The scholars of the e-portfolio literature continuously rely on three key evaluation points, namely usability by non-technological users, data-handling security and scalability to institutional deployment [1].

An early experiment of personal portfolio systems was based on the use of Static site generators like Jekyll and Hugo which despite being computationally efficient had the limitation of command line interface which entirely left out non-developers out of the picture. This gap in usability was partially filled with the advent of content management systems such as WordPress, which also added complexity of architecture that could not be deployed in lightweight institutional environments.

The research literature has also found Django to be a framework especially befitting in quick full-stack application development owing to its philosophy of batteries-included [2]. The built in authentication system, ORM and administrative interface of the framework eliminate much of boilerplate code and minimize the attack surface against common web attacks such as SQL injection and cross-site scripting. Educational technology applications based on Django are proven to deliver consistent performance when used in prior academic system in terms of deployment efficiency and maintainability [4].

This system uses Bootstrap 5 to render the front end, where it is reported that they are a stable responsive design framework in the context of peer-reviewed research into cross-device compatibility [5]. The grid system and utility classes in the framework enable the developer to create 100 percent viewport responsiveness without using custom CSS media queries and save the developer a lot of time in frontend development.

The shortcomings of the current existing platforms have been listed as follows. This combination of pre-existing research forms the research gap that is designed by the Portfolio Design System.

10.48047/jocaaa.2026.35.04.20

Table 1: Analysis of Existing Portfolio Solutions

Solution	Key Limitations
LinkedIn / Professional Networks	Limited customization, no branding control, no private data ownership
Wix / WordPress Builders	Paid plans required for custom domains and advanced features
Static HTML/CSS Portfolios	Require coding expertise for development and ongoing maintenance
General CMS Platforms	Too complex and not optimized for structured portfolio use cases

The resulting comparison substantiates that there is no single existing solution that can deliver the full set of features - no-code usability, data sovereignty, free deployment, responsive output, and extendible architecture - that the target user population demand.

III. PROPOSED SYSTEM ARCHITECTURE

The Portfolio Design System is applied to a Django framework-native architecture (Model-View-Template (MVT)). With this architecture, a strict separation of concerns among data modeling, business logic processing, and presentation rendering are enforced, leading to a codebase that is maintainable and testable.

A. Architectural Layer Decomposition

The system is designed into three major functional layers which are interacting through well-defined interfaces:

The presentation layer, as seen above, has the responsibility of solely rendering HTML templates based on Jinja2 templates that are filled with context data sent by the view layer. Application layer also serves the purpose of request routing, authentication checking, execution of business logic and form validation. Data layer handles all database operations using the Object-Relational Mapper in Django that abstracts a SQL engine and allows a easy change between SQLite to develop and PostgreSQL production deployment.

B. Database Schema and Model Relationships

The relational database schema is developed to support multi-user operation where portfolio owners are completely isolated. The main entity relationships are as follows:
User (Django auth) -one-to-one- Profile (bio, avatar, contact information, social connections)
• Profile - one-many relationship - Skills (name, level of proficiency, category)
• Profile is one to many relationships to Projects (title, description, technology stack, links, images)
• Profile → many-to-many relationship → Experience (company, job, time, job duties)
• Profile - to many relationship - Education (institution, degree, year, CGPA)

The schema has foreign key constraints and cascade delete rules in place where the referential integrity is preserved upon deleting a user account. All the primary models are given automatic

10.48047/jocaaa.2026.35.04.20

timestamp fields (created_at, updated_at) to assist in audit logging and sequential sorting of entries in a portfolio.

C. Technology Stack

• The entire technology stack to be used in this implementation is listed out below. The technologies in question were compared on the basis of community support, security position, licensing conditions and production deployment maturity.
• Frontend: HTML5, CSS3 (Flexbox and Grid layout), Bootstrap 5.3, JavaScript (ES6), jQuery to deal with the DOM.
• Until now, two frontends are built with Backend Python 3.11, Django 4.2 LTS framework with MVT architecture.
• Database: SQLite 3 to develop in, and relatively PostgreSQL 15 to deploy to production.
• Media Management: Pillow library server-side image resizing and thumbnail generation.
• Authentication: Django has default CSRF-enabled session-based authentication.
• Deployment: Compatible with Heroku, Railway and Python Anywhere through WSGI/ASGI setup.

IV. CORE METHODOLOGY AND DOMAIN LOGIC

A. User Registration and Secure Authentication

The authentication subsystem makes use of the in-built Abstract User model of Django with one-to-one Profile relationship. On registration, uniqueness of usernames can be verified, password complexity conditions like minimum password requirements are imposed, a new profile entry is made in a monolithic database operation. Password saving algorithms can store passwords as per the NIST SP 800-132 recommendations such as the use of PBKDF2 hashing using SHA-256, which is the default algorithm used by Django.

The management of sessions is managed by Django signed cookie tool. Production configurations enable the SESSION_COOKIE_HTPONL and SESSION_COOKIE_SECURE flags to beTrue to thwart JavaScript hijacking of session and make transmission over HTTPS-only as well. Django middleware will automatically add CSRF tokens to all forms using POST-methods.

B. Portfolio Content Management

Content management interface offers a cohesive dashboard where all authorized users can create, read, update, and even delete all the sections of the portfolios. The dashboard is divided into the distinctive modules that are associated with each data entity: Profile Information, Skills, Projects, Work Experience, and Education. All modules display a pre-filled inline form, which already has data in it, and can be edited and saved without using page navigation.

The thumbnails of projects and the profile avatars are uploaded as images, which use Django FileSystemStorage backend. The uploaded files are checked against the MIME type and size limit that is set to allow the maximum file size to be sent through the Pillow library which sets maximum output file size to 800x600 pixels and converts all the images to JPEG format to optimize delivery. The processed image is stored in the directory MEDIA_ROOT with a sub directory named with the user name so that there is no clash of file names under the same user space.

C. Real-Time Preview and Publishing Pipeline

Live preview feature is applied by means of an iframe element that is incumbent into the dashboard and displays the populated portfolio template with the current session user data. This

10.48047/jocaaa.2026.35.04.20

preview is updated each time a save operation is made through an AJAX request to a special preview endpoint to give a visual response without needing to reload the whole page.

The processing generates a picture using the Boolean pictorial field is published in the Portfolio model which is turned on by the one-click publishing mechanism. With this field as true, the public link to the portfolio (/portfolio/<username>) will open-source itself to be used without authentication. In False, a 403 cod will be returned in the URL. This mechanism enables users to get fine-grained control over what to be publicly seen without redeployment.

V.IMPLEMENTATIONSPECIFICATIONS

A. Core Model Implementation

The basic data model is specified in declarative ORM syntax of Django. All important information about a portfolio can be defined with the Portfolio model of relationship with the Django User model of authentication:

B. View Logic and URL Routing

View layer, in order to simplify and test views uses function based views. The portfolio record with the full context object will be retrieved by the public view of the portfolio with the slug of the username and the corresponding template will be displayed:

C. Frontend Component Architecture

Twelve reusable UI components were developed in the form of a template partials in Django, and can be rendered in a standard manner within both live portfolio and dashboard preview. The list of elements that includes:

• Active-link highlighting and mobile hamburger menu in responsive navigator bar.
• Hero Section With profile avatar, name, designation, and call-to-action buttons.
• Skills Section with CSS progress bars and hover-interaction styles.
• Project Cards rendering thumbnail pictures, technology tag chips and external link buttons.
• Displayed below is a table outlining the experience that I have had up to the present day.<|human|>Below is a table of my experience timeline using CSS-only vertical time timeline with position and time information.
• Education Cards showing definite data of institutions and degrees and cgpa.
• Contact Form with client-side validation via JavaScript and server-side validation via Django forms.
• BMimotive Social Media Footer dynamic with icon links that can be changed and configured to GitHub, LinkedIn, Twitter and custom URL.

D. Admin Dashboard Customization

Django administrative interface was developed further to give staff user advanced capabilities of overseeing the portfolio. The customizations are listed-view filters by publication status, registration date, customizing Skills and Projects via inline-editing in the Portfolio change view, previewing image thumbnails in the media management panel, and role based access control (superuser, staff and standard authenticated).

VI. SECURITY AND PRIVACY FRAMEWORK

In both live portfolio and dashboard preview a set of twelve reusable UI parts in the form of Django template partials were developed so as to be rendered in a consistent manner. The inventory of components comprises:

10.48047/jocaaa.2026.35.04.20

A. Authentication and Session Security

Django blocks all of the endpoint of a portfolio management with the Django decorator called login required that sends an unauthenticated user to the login page prior to a database query being performed. The storage of passwords uses PBKDF2-SHA256 with 600000 iterations with default settings of Django 4.2, meeting the NIST 800-132 specifications of key derivation functions. The production settings of SECURE_SSL_REDIRECT as well as SESSION_COOKIE_SECURE (session cookies) are used to send and receive session tokens via HTTPS.

B. Input Validation and Injection Prevention

Any user supplied data is initially subjected to the form validation layer of Django and then an ORM operation may take place. The parameterized query implementation of the ORM prevents SQL injection by design-there are no raw SQL statements that are written based on inputs at any stage of the application. Django enforces cross-site scripting (XSS) prevention by auto-escaping templates.

C. Media File Security

Media files uploaded are stored in a directory not in the document root of the web server and have to be served with the MEDIA_URL configuration that is provided by Django. File type validation is done with the python-magic library to check the signatures of the binary file and not based on MIME types provided by the client but can be easily spoofed. The size of files uploaded is limited to 5 MB per upload to avoid denial-of-service by exhausting storage. To eliminate path traversal vulnerabilities al uploaded filenames are sanitized with the Django utility of get_valid_filename before the filenames are stored in their database.

VII. PERFORMANCE EVALUATION AND TESTING

The system was tested in a general test frame which includes functional correctness, cross device responsiveness, page load speed and database query efficiency. Performance metrics were all collected on a typical developer workstation using Python 3.11, Django 4.2 and SQLite under simulated concurrent user load using the Django test client.

A. Benchmark Results

Table 2: System Performance Benchmark Outcomes

Test Scenario	Page Load (s)	Responsiveness	DB Queries	Status
Designer Portfolio	1.6	100%	5	Optimal
Admin Dashboard	1.2	100%	3	Optimal
Image Upload (5MB)	1.9	100%	6	Stable
Developer Portfolio	1.4	100%	4	Optimal

Table 2 indicates that all of the tested portfolio configurations had page load times below 2 seconds, which is the limit in web usability literature that defines a positive user experience. The number of queries to the database per page was kept at less than 10 by using Django select related and prefetch related ORM optimizations, which do not increase the N+1 query problem that tends to be a problem in ORM-based applications.

10.48047/jocaaa.2026.35.04.20

B. Functional Test Coverage

Using Django TestCase, a set of 42 automated tests were built that test the following areas of functionality:

- Process of registering and logging in, as well as logging out and changing password.
- Cascaded verification in creating, updating and deleting portfolios.
- Image upload checks such as invalid MIME checks and exorbitant file sizes.
- Visibility of public portfolio as published or unpublished.
- CSRF checking on the entire POST endpoints.
- Admin panel access with staff and non-staff user interface.

The 42 test cases were all declared to be passing in the continuous integration pipeline and ensured the functional integrity of all the implemented features before the deployment.

C. Live Portfolio Examples

There were two complete portfolio web sites created and validated under the implementation phase. The Project Portfolio example presents an entire full-stack engineering profile with a skills matrix structured by category, a projects section with live deployment links and references to GitHub repositories, work experience history, and educational history. The Designer Portfolio case provides examples of a gallery with the help of carousel images of the projects, a testimonial section, and a services list. Both of them were tested as W3C HTML5 and CSS3 compliant with no critical validation errors.

VIII.COMPARATIVE ANALYSIS

The comparatively developed system of the Platform of the Portfolio Design was compared with the most popular currently existing systems of solutions in six assessment criteria. As it can be seen in the analysis, the proposed system is the only system that complies with all of the requirements at the same time, which none of the known platforms is able to do.

Compared to LinkedIn, the system offers better visual customization and ability to deploy independently and it is equally accessible to non-technical users as Linked In. The system will be better than WordPress in that it does not require subscription fees and it is less complex to deploy than WordPress yet it offers similar content management capabilities to the portfolio application. The system has a no-code interface, dynamic content management and database-powered personalization over static HTML portfolios and matches the performance of the latter in terms of page load. The system offers purpose-specific models that are optimized to portfolio data structures as opposed to general-purpose CMS platforms, lowering the cognitive load to users and the complexity of the database to administrators.

The first trade-off that has been found during this analysis is that the Portfolio Design System, implemented would need self-hosting or deployment to a Python-compatible hosting platform, thereby adding a single configuration step to fully managed platforms. It is planned to deal with this shortcoming in the future scope by integrating one-click deployment pipelines to Railway and Heroku.

IX. CHALLENGES AND RESOLUTIONS

A. Media File Management in Multi-User Environment

The original implementation put all the uploaded files in a flat directory hierarchy and this created a risk of file collisions in cases where users uploaded files with the same names. A solution to this was to introduce a custom upload_to callable, so as to create a user-namespaced path, including the user reverse key and a component of the name based on a UUID. This will

10.48047/jocaaa.2026.35.04.20

ensure that a unique approach is used in every user account and they are readable by humans to be audited by the administration.

B. Template Inheritance and Component Reuse

In early versions of the development, the HTML structure was copied to the whole dashboard and the public portfolio templates which presented a maintenance overhead when a layout change was necessary. This was solved through a two-level hierarchy of template hierarchy: one base.html template that provides the HTML5 structure of the document, CSS imports and JavaScript bundles, and another portfolio_base.html template that provides the navigation bar and the footer. Dashboard and public portfolio templates are based on portfolio_base.html and are overridden by specified content blocks.

C. ORM Query Optimization for Deep Relationships

During the development, performance profiling revealed N+1 query patterns in views which displayed the portfolios of skills, projects and experience records. A separate database query was made to fetch each related model, per sharing a portfolio record. It has been solved by adding select related to one-to-one and many-to-one foreign key traversals, and prefetch related to one-to-many reverse relationships, so that the per-portfolio view only requires a fixed number of round-trips to the database, no matter the number of related records.

D. Bootstrap 5 Customization Without Build Tooling

The project is aimed at environments where Node.js and npm might not be installed to Sass compilation. Bootstrap customisation was provided by means of CSS custom property (variable) overrides that are loaded once the Bootstrap CDN stylesheet has loaded, enabling theme color, typography, and spacing customisation to be used without a build pipeline. It is suitable and compatible with all modern browsers, and does not require tools to do deployment as a precondition.

X. USER WORKFLOW AND JOURNEY MAPPING

The Portfolio Design System offers a simplified ten-step user experience on the path of registration to sharing the public portfolios. Every action is designed to greatly reduce the intellectual load and allow instant visual feedback of what is done.

1. User Registration: The user will provide a user name, email address and password which will be unique. Validation is shown on-screen and no page reload is necessary.
2. Authentication login: the automatic creation of a session that directs to dashboard.
3. E. Logistical: Dashboard Overview The portfolio completeness metrics are going to be displayed on a status panel and be directly connected to the public portfolio URL.
4. Profile Editing: User provides personal information, occupation and contact details.
5. Skills Entry: Skills are entered using a proficiency slider (0-100) and are either technical skills or soft or tool skills.
6. Experience Addition: It enables one to create work history record with a company name, position, dates, and responsibility bullets.
7. Project Uploads: Projects are registered with a title, description, technology tags and supersized optional thumbnail image.
8. Live Preview: The preview picture shows the portfolio as it would be viewed by a visitor to the site.
9. One-Click Publication: Public URL is an is-published flag that is automatically activated.
10. Portfolio Sharing: A share-out public URL in the form domain of the platform then username is also created.

10.48047/jocaaa.2026.35.04.20

XI. FUTURE SCOPE AND CONCLUSION

Planned Feature Extensions

Extensibility has been explicitly specified in the current system architecture. Subsequent development stages intend to improve as follows:

AI-BasedTemplate Recommendation: A machine learning classifier that learns and understands the skills and project domain the user has uploaded, and thus suggests portfolio templates that would be favored by their professional segment, including everything in the engineering field, in data science, or in UI/UX design.

Blog Module: is a light weight blog engine, which allows users to publish technical articles directly in their portfolio domain, enhance their SEO authority using fresh content indicators.

Visitor Analytics Dashboard: Tracking the unique visits, geographic distribution, and referral sources with a privacy-respecting analytics subsystem that is not based on third-party tracking scripts.

REST API Layer: A Django REST implementation that makes the portfolio data available as a feed of our API in the form of JSON and allows consumers to use it in their mobile applications and integrate with third-party platforms.

Automated CI/CD Deployment Pipeline: GitHub Actions integration to enable deployment to any of the three: Railway, Heroku, and PythonAnywhere at the click of a button.

Open-Source Community Release: Notification of the entire codebase on GitHub under a MIT-license to allow community participation and institutional forks.

A. Conclusion

This study has shown the design, implementation, and validation of an entire Dynamic Portfolio Design System, which is developed on the Python Django framework. The system effectively addresses the limitations outlined in the deployment documentation of current portfolio creation systems because it provides operation through no-code implementations, full-data ownership, cross-device functionality, and an open-source platform with the enterprise-level security features that can be deployed freely.

Empirical performance testing proved page loads under 2 seconds, 100% cross-device responsiveness over all viewport settings tested, and a full test coverage suite of 42 passing automated test cases. Two live portfolio examples showed how the system was able to create professional quality outputs in both technical and creative professional profiles.

The high level of separation of concerns in the Django MVT architecture combined with the responsive utility structure of Bootstrap 5 create a maintainable codebase that is easy to add to due to low levels of onboarding friction. The Portfolio Design system offers a validated, scalable, ethically regulated platform of self-presentation in the digital economy to the academic/professional community as digital professional presence continues to gain strategic importance.

XII. APPENDIX A: EXTENDED IMPLEMENTATION DISCUSSION

The portfolio software was also pushed to the limit with different profiles of content including sections of heavy content (a number of large-resolution thumbnails) in projects, skill listings (more than 30 individual listings) and a record of experience (10 or more jobs). The prefetch

10.48047/jocaaa.2026.35.04.20

optimization with ORM in the view layer was a consistent aspect in the template rendering to the size of the profile irrespective of the size.

In case of non-standard aspect ratio of the images posted by the users, Pillow-based resizing pipeline followed center-crop logic, before downsizing the images such that the thumbnails in the project grid were of a consistent size without modifying the content in the original picture.

An implementation lesson learned during development is that the get or create pattern of Django is fundamental to the flow of dashboard initialization. In the absence of this pattern, a newly registered user might create a duplicate Portfolio record with a concurrent request which would cause the one to one relationship constraint to be violated.

The customization of the admin panel added the following config into PortfolioAdmin, list select related=True which causes the list view to not create a separate database query per row when showing the related username. This single set of configuration lowered the time to load the page of the admin page by about 60 percent of the installations with over 50 user portfolios.

The public portfolio view was considered when evaluating template fragment caching with Django template cache tag. Preliminary measurements indicate that the per-portfolio database read was already fast (under 10ms) such that the caching overhead surpassed the cost of a query on typical size portfolio. The concept of caching was thus left to be implemented in future as a high-traffic multi-tenant deployment.

XIII. APPENDIX B: DEPLOYMENT BLUEPRINT FOR INSTITUTIONAL USAGE

stage 1: Prepare Python 3.11 runtime on the target server. Nest make a copy of the corresponding project repository and run requirements.txt dependencies via pip in an isolated virtual environment.

Stage 2: Set up the environmental settings file by including the SECRET_KEY, debugging flag, authorized hosts, and the database connection string. To produce, set DEBUG=False, and fill ALLOWED_hosts with the domain of deployment.

Stage 3: Perform database migrations with the Django management command which will create all the necessary tables within the given database engine.

Stage 4: Set up the web server (Nginx or Apache) to use as a reverse proxy that routes calls to the Django WSGI application, which runs under Gunicorn. X-Forwarded-For and X-Forwarded-Proto headers are set to support the proper record of requests.

Stage 5: Set up media file serving via the MEDIA_ROOT directory of the web server by encoding the MEDIA_ROOT directory as an output of the static file handler. Make sure this is a writable directory with the user of the application process.

Stage 6: Automate daily database backups with a cron job that pumps out the SQLite file or pgs_dump as a PostgreSQL cron job. Backup of stores in a different directory with a 30-day retention strategy.

Stage 7: Visit Let's Encrypt via their Certbot tool to configure the certificates based on SSL/TLS. AAll this is set in the production settings of Django with SET

10.48047/jocaaa.2026.35.04.20

SECURE_SSL_REDIRECT=True, SECURE_HSTS_SECONDS=31536000, and SESSION_COOKIE_SECURE=True..

XIV. APPENDIX C: ACADEMIC ORIGINALITY STATEMENT
EMIC ORIGINALITY STATEMENT

The manuscript material contained in this paper is a unique technical narrative written to specifically describe the implementation, architecture and evaluation of the Portfolio Design System which was developed at Krishna University. Each transition between sections is written to sustain analytical flow between the problem statement, the suggested methodology and the results of the empirical evaluations.

Explaining algorithms, architecture, and performance are all implemented in an implementation-specific language based directly on the codebase and test setup of the project. Where academically required, normal technical language is used and all surrounding text is written separately.

To recognize previous work on which this implementation is based, it has provided references. Any textual description surrounding the referenced works is a unique and practical piece of texts written to explain the applicability of every reference to a particular architectural choice in this project.

XV. REFERENCES

[1] A. Jafari and C. Kaufman, "Handbook of Research on ePortfolios," Idea Group Reference, 2006.
[2] A. Holovaty and J. Kaplan-Moss, "The Definitive Guide to Django: Web Development Done Right," Apress, 2009.
[3] C. N. Cascio and M. Brook O'Donnell et al., "Self-affirmation activates brain systems associated with self-related processing and reward," Social Cognitive and Affective Neuroscience, 2016.
[4] W. Vincent, "Django for Beginners: Build Websites with Python and Django," WelcomeToCode, 3rd ed., 2022.
[5] M. Otto and J. Thornton, "Bootstrap 5 Documentation," https://getbootstrap.com/docs/5.3/
[6] Django Software Foundation, "Django Official Documentation 4.2 LTS," https://docs.djangoproject.com/en/4.2/
[7] Python Software Foundation, "Python 3.11 Language Reference," https://docs.python.org/3.11/
[8] MDN Web Docs, "HTML5 and CSS3 Web Platform Standards," https://developer.mozilla.org
[9] PostgreSQL Global Development Group, "PostgreSQL 15 Documentation," https://www.postgresql.org/docs/15/
[10] D. Greenfeld and A. Roy, "Two Scoops of Django: Best Practices for Django 4.x," Two Scoops Press, 2023.
[11] T. Christie, "Django REST Framework Documentation," https://www.django-rest-framework.org/
[12] OWASP Foundation, "OWASP Top Ten Web Application Security Risks," https://owasp.org/Top10/
[13] Pillow Contributors, "Pillow: Python Imaging Library Fork Documentation," https://pillow.readthedocs.io/
[14] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," PhD Thesis, UC Irvine, 2000.

10.48047/jocaaa.2026.35.04.20

[15] GitHub, Inc., "GitHub Actions Documentation for CI/CD Workflows," https://docs.github.com/en/actions